Curso Técnico Nível Médio Subsequente

Informática Para Internet

Informática

Thiago Medeiros Barros

Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte.

Natal-RN
2015

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação a Distância

Este Caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte e o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

GOVERNO DO BRASIL

Presidenta da República
Dilma Vana Rousseff

Ministro da Educação
Aloizio Mercadante

Diretor de Educação a Distância da CAPES
João Carlos Teatini

Reitor do IFRN
Belchior De Oliveira Rocha

Pró-Reitor de Pesquisa e Inovação
José Yvan Pereira Leite

Coordenador da Editora do IFRN
Paulo Pereira Da Silva

Diretor do *Campus* EaD/IFRN
Erivaldo Cabral

Diretor Acadêmico do *Campus* EaD/IFRN
Wagner de Oliveira

Coordenador Rede e-Tec Brasil no IFRN
Roberto Douglas da Costa

Coordenador do Curso Técnico Subsequente em Informática para Internet
Charles César Magno de Freitas

FUNDAMENTOS DE LÓGICA E ALGORITMOS

Professor-autor
Thiago Medeiros Barros
Bruno Emerson Gurgel Gomes

Projeto Gráfico
Eduardo Meneses e Fábio Brumana

Direção da Produção de Material Didático
Rosemary Pessoa Borges

Coordenação de Produção de Mídia Impressa
Leonardo dos Santos Feitoza

Coordenação de Revisão
Wagner Ramos Campos

Revisão ABNT
Francisco de Assis Noberto

Revisão Linguística
Maria Tânia Florentino de Sena Nascimento

Revisão Pedagógica
Kalina Alessandra Rodrigues de Paiva

Diagramação/ Ilustração
Cyntia Vale
Georgio Nascimento
Joaci de Paula
Rômulo França
Yann Valber

Ficha catalográfica

G978f Gomes, Bruno Emerson Gurgel.

Fundamentos de Lógica e Algoritmos / Bruno Emerson Gurgel Gomes, Thiago Medeiros Barros. – Natal : IFRN Editora, 2015.

210 f. : il. color.

ISBN 978-85-8333-142-1

Curso Técnico Nível Médio Subsequente Informática para Internet.

1. Fundamentos de Lógica e Algoritmos - EaD. 2. Argumento lógico. 3. Proposições. 4. Lógica clássica. 5. Controle de fluxo - Programação. 6. Linguagem de programação. 7. Estruturas de repetição. 8. Linguagem Python. I. Barros, Thiago Medeiros. II. Título.

RN/IFRN/EaD CDU 004.421

Ficha elaborada pela bibliotecária Edineide da Silva Marques, CRB 15/488

Apresentação e-Tec Brasil

Amigo(a) estudante!

O Ministério da Educação vem desenvolvendo Políticas e Programas para expansãoda Educação Básica e do Ensino Superior no País. Um dos caminhos encontradospara que essa expansão se efetive com maior rapidez e eficiência é a modalidade adistância. No mundo inteiro são milhões os estudantes que frequentam cursos a distância. Aqui no Brasil, são mais de 300 mil os matriculados em cursos regulares de Ensino Médio e Superior a distância, oferecidos por instituições públicas e privadas de ensino.

Em 2005, o MEC implantou o Sistema Universidade Aberta do Brasil (UAB), hoje, consolidado como o maior programa nacional de formação de professores, em nível superior.

Para expansão e melhoria da educação profissional e fortalecimento do Ensino Médio, o MEC está implementando o Programa Escola Técnica Aberta do Brasil (e-TecBrasil). Espera, assim, oferecer aos jovens das periferias dos grandes centros urbanose dos municípios do interior do País oportunidades para maior escolaridade, melhorescondições de inserção no mundo do trabalho e, dessa forma, com elevado potencialpara o desenvolvimento produtivo regional.

O e-Tec é resultado de uma parceria entre a Secretaria de Educação Profissionale Tecnológica (SETEC), a Secretaria de Educação a Distância (SED) do Ministério daEducação, as universidades e escolas técnicas estaduais e federais.

O Programa apóia a oferta de cursos técnicos de nível médio por parte das escolaspúblicas de educação profissional federais, estaduais, municipais e, por outro lado,a adequação da infra-estrutura de escolas públicas estaduais e municipais.

Do primeiro Edital do e-Tec Brasil participaram 430 proponentes de adequaçãode escolas e 74 instituições de ensino técnico, as quais propuseram 147 cursos técnicosde nível médio, abrangendo 14 áreas profissionais.

O resultado desse Edital contemplou193 escolas em 20 unidades federativas. A perspectiva do Programa é que sejam ofertadas10.000 vagas, em 250 polos, até 2010.

Assim, a modalidade de Educação a Distância oferece nova interface para amais expressiva expansão da rede federal de educação tecnológica dos últimos anos: aconstrução dos novos centros federais (CEFETs), a organização dos Institutos Federaisde Educação Tecnológica (IFETs) e de seus campi.

O Programa e-Tec Brasil vai sendo desenhado na construção coletiva e participaçãoativa nas ações de democratização e expansão da educação profissional no País,valendo-se dos pilares da educação a distância, sustentados pela formação continuadade professores e pela utilização dos recursos tecnológicos disponíveis.

A equipe que coordena o Programa e-Tec Brasil lhe deseja sucesso na sua formaçãoprofissional e na sua caminhada no curso a distância em que está matriculado(a).

Brasília, Ministério da Educação – setembro de 2008.

Sumário

Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 01

Argumento Lógico, Proposições Simples, Princípios Lógicos

Thiago Medeiros Barros

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno! Seja bem-vindo à disciplina de Fundamentos de Lógica e Algoritmos! Em nossa primeira aula, serão apresentados os princípios da Lógica Clássica. Você será apresentado a uma nova forma de raciocínio, a qual lhe dará ferramentas para compreender e elaborar pensamentos logicamente válidos, a partir da formalização de expressões do cotidiano. Esse conhecimento não será apenas importante para o seu curso; será também para a vida, uma vez que aprenderá a desenvolver um raciocínio pragmático para resolver problemas aplicados em diversas áreas do seu dia a dia.

Bons estudos!

Aula 1 - Argumento Lógico, Proposições Simples, Princípios Lógicos

Objetivos

Compreender o conceito de argumento lógico;

Compreender o conceito de proposições;

Entender o processo de formalização de expressões cotidianas;

Compreender e refletir sobre os princípios que embasam a Lógica Clássica.

Desenvolvendo o conteúdo

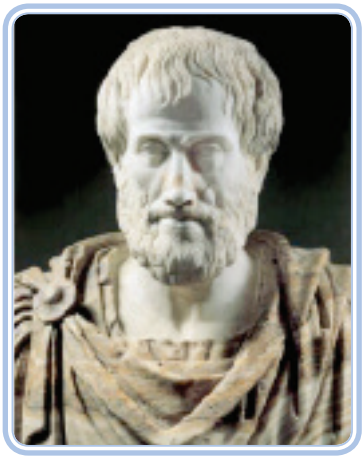
Veja a imagem a seguir:

Fonte: Valber e Vale (2015)

Figura 1.1: Um exemplo de lógica

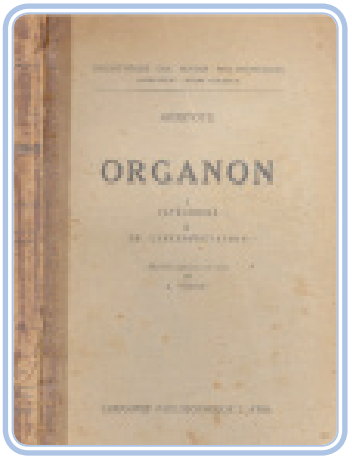
A lógica geralmente é associada quando utilizamos raciocínios pragmáticos para encontrar soluções. Podemos encontrar sua aplicação em diversos setores das nossas vidas: pesquisas científicas, jogos diversos, argumentaçãocom amigos sobre um tema polêmico. Antes de começar a compreender a Lógica Clássica, vamos ver um pouco de sua história.

A palavra Lógica deriva do Grego (logos), que significa: palavra, pensamento, ideia, argumento, relato, razão lógica ou princípio lógico.



Fonte: http://www.conocereisdeverdad.org/website/index.php?id=6371

Fig. 02: O grego Aristóteles



Fonte: https://sebodomessias.com.br/livro/filosofia/organon.aspx

Fig. 03: Organon

De forma histórica, podemos apontar algumas passagens importantes Costa e Raphael (2012):

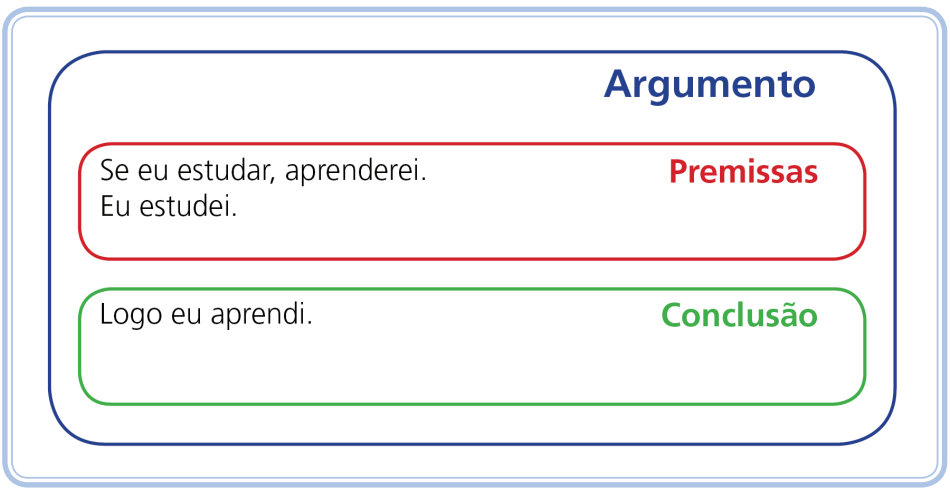
- Início na Grécia em 342 a.C.;
- Aristóteles sistematizou os conhecimentos existentes em Lógica, elevando-a à categoria de ciência;
- Em sua obra chamada Organom ("ferramenta para o correto pensar"), estabeleceu princípios tão gerais e tão sólidos que até hoje são considerados válidos;
- Aristóteles se preocupava com as formas de raciocínio que, a partir de conhecimentos considerados verdadeiros, permitiam obter novos conhecimentos; e
- A partir dos conhecimentos tidos como verdadeiros, caberia à Lógica

a formulação de leis gerais de encadeamentos lógicos que levariam à descoberta de novas verdades.

Que tal ver um pouco mais da história da Lógica em quadrinhos e treinar seu Inglês DOXIADIS, A.; PAPADIMITRIOU, C. H. **Logicomix:** An Epic Search for Truth. Nova York: Bloomsburry USA, 2009.

De acordo com a Enciclopedia Barsa (2010), temos que: Lógica é a Ciência que estuda as leis do raciocínio e as condiçõoes de verdade em vários domínios do conhecimento.

A Lógica se fundamenta no Argumento, o qual é o encadeamento de afirmações denominadas de Proposições. Em um argumento, essas proposições se dividem em dois grupos: as Premissas e a Conclusão. As Premissas são um conjunto de proposições consideradas provas evidentes, as quais, a partir de regras lógicas, deriva-se uma proposição verdadeira, também chamada de conclusão. O argumento pode ser ilustrado a partir da figura seguinte.



Fonte: Barros (2015)

Fig 04: Representação do Argumento

Há algumas dicas em tentar identificar as premissas e a conclusão de um argumento, tal como indicado por (NOLT; ROHATYN, 1991). Abaixo, segue a lista de alguns.

Tabela 1: Indicadores de Conclusão e Premissa

Indicadores de Conclusão	Indicadores de Premissa
Portanto	Pois
Por conseguinte	Desde que
Assim	Como
Dessa maneira	Porque
Neste caso	Assumindo que
Daí	Visto que
Logo	Admitindo que
De modo que	Isto é verdade porque
Então	A razão é que
Consequentemente	Em vista de
Assim sendo	Como consequência de
Segue-se que	Como mostrado pelo fato que
O (a) qual implica que	Dado que
O (a) qual acarreta que	Sabendo-se que
O (a) qual prova que	Supondo que
O (a) qual significa que	
Do (da) qual inferimos que	
Resulta que	
Podemos deduzir que	

Fonte: Nolt e Rohatyn (1991).

Exemplos dos indicadores da tabela 1.

- Supondo que não consiga chegar a tempo para a reunião, então é melhor avisar.
- Desde que não me faça promessas, neste caso não criarei expectativas.
- Ficarei triste com sua partida, pois estou acostumado com sua presença.
- O preço da energia aumentou, logo, evitaremos utilizar o ar-condicionado.

Durante o século XIX, diversos filósofos e matemáticos (George Boole, Alfred North, Augustos de Morgan, Bertrand Russel, Gottlod Frege) perceberam que a Lógica concebida por Aristóteles possuía problemas quanto ao rigor científico, uma vez que utlizava a linguagem natural (como a língua inglesa ou o português). A consequência é que tornava a lógica imprecisa e vulnerável a erros de dedução. A partir deste problema, surge então a Lógica Simbólica, constituída por uma linguagem escrita e universal, formada por símbolos específicos (Fajardo).

Alguns exemplos de como a ambiguidade da linguagem pode trazer problemas na lógica, conhecidos como paradoxos:

- Paradoxos de Zenão de Eléia - A flecha que voa nunca sai do lugar, pois, em cada instante de tempo ocupa uma só posição no espaço. Logo, ela está imóvel em todo o tempo.

- Eubulides de Mileto - Um homem diz que está mentindo. O que ele diz é verdade ou mentira?"

- Paradóxo da escola do Estoicismo - Se Você sabe que está morto, você está morto. Mas se você sabe que está morto, você não está morto. Portanto, você não sabe se está morto ou não.

Fonte: Valber (2015).

Figura 5: Ilustração de um paradoxo

Depois do debate em relação a ambiguidade do uso da linguagem natural, surge então a Lógica Proposicional, a qual todos os argumentos são baseados em proposições válidas. Para entender bem a importância da mudança desse paradigma, veja a metáfora de Frege (1879, p. 46):

> Creio que posso tornar mais clara a relação entre minha conceitografia e a linguagem comum comparando-a à que existe entre o microscópio e o olho. Este, pela extensão de sua aplicabilidade, pela agilidade com que é capaz de adaptar-se às diferentes circunstâncias, leva grande vantagem sobre o microscópio. Considerado como aparelho ótico, o olho exibe decerto muitas imperfeições que habitualmente permanecem despercebidas, em virtude da ligação íntima que tem com a vida mental. No entanto, tão logo os fins cinetíficos imponham exigências rigorosas quanto à exatidão das discriminações, o olho revelar-se-á insuciente. O microscópio, pelo contrário, conforma-se a esses fins de maneira mais perfeita, mas, precisamente por isso, é inutilizável para todos os demais

De acordo com Bedregal (2007, p. 39), temos que:

Uma proposição é uma sentença declarativa sobre objetos, que pode assumir valores verdade Verdadeiro (1) ou Falso (0), dependendo da interpretação.

Para ilustrar, podemos utilizar os seguintes exemplos:

1. Natal é a cidade mais bela do Nordeste.

2. Todo ser humano morre.

3. 2 é menor que 3.

As sentenças acima podem ser valoradas como verdadeiro ou falso, de acordo com a realidade. Entretanto, é importanter notar que a primeira expressão necessita definir formalmente o que é "bela".

É importante frisar que o valor verdade **não pode ser verdadeiro e falso ao mesmo tempo**. Outra característica comumente encontrada na sentença é possuir **Sujeito e Predicado**, mas não necessariamente é obrigatório, por exemplo uma expressão matemática " 2 + 3 = 5". Por ser uma sentença declarativa, uma proposição **não** deve ser **exclamativa, interrogativa, imperativa ou optativa.**

Alguns exemplos de setenças que **não** são proposições:

- O cachorro (falta predicado)
- Que excelente trabalho! (exclamativa)
- Onde você mora? (interrogativa)
- Pegue meu café! (imperativa – exprime ordem)
- Que Deus te abençoe (optativa – exprime desejo)
- Eu sou mentiroso (paradoxo)

Já exemplos de sentenças que são proposições:

- 1 + 1 = 2
- O planeta Terra é redondo.
- Thiago é professor de Lógica.

As sentenças também podem ser consideradas **Abertas**, as quais são aquelas que envolvem o uso de **variáveis**. Variáveis são termos que podem ser atribuídos diversos valores. Tais sentenças não são consideradas proposições.

Fonte: http://www.dds.com.br/blog/index.

Figura 6: omem pensando

Exemplos de Sentenças Abertas:

- X + 5 = 10 (depende do valor de X)
- Ela ganhou o Oscar de melhor ator em 2012 (não se sabe quem é Ela)

Outro importante detalhe é que as proposições são representadas geralmente com letras do alfabeto, exemplo:

- p: A sede do IFRN é localizado na Paraíba (proposição Falsa)
- q: Quem nasce no RN também é chamado de Potiguar (proposição Verdadeira)

Atividade de aprendizagem 1

Vamos testar um pouco o que aprendemos.

1. **(BB1/2007/Cespe)** Na lógica sentencial, denomina-se proposição uma frase que pode ser julgada como verdadeira (V) ou falsa (F), mas não como ambas. Assim, frases como "Como está o tempo hoje?" e "Esta frase é falsa" não são proposições porque a primeira é pergunta e a segunda não pode ser nem V nem F. As proposições são representadas simbolicamente por letras maiúsculas do alfabeto — A, B, C, etc. Uma proposição da forma "A ou B" é F se A e B forem F, caso contrário é V; e uma proposição da forma "Se A então B" é F se A for V e B for F, caso contrário é V.

Considerando as informações contidas no texto acima, julgue o item subsequente.

01. Na lista de frases apresentadas a seguir, há exatamente três proposições.

"A frase dentro destas aspas é uma mentira."

A expressão X + Y é positiva.

O valor de 4 + 3 = 7 .

Pelé marcou dez gols para a seleção brasileira.

O que é isto?

Resolução

"A frase dentro destas aspas é uma mentira.": É uma contradição, logo, não é uma proposição

A expressão X + Y é positiva: É uma sentença aberta

O valor de 4 + 3 = 7, Pelé marcou dez gols para a seleção brasileira: São proposições

O que é isto?: Sentença Interrogativa. Não é proposição

2. **(ICMS-SP/2006/FCC)** Das cinco frases abaixo, quatro delas têm uma mesma característica lógica em comum, enquanto uma delas não tem essa característica.

I. Que belo dia!

II. Um excelente livro de raciocínio lógico.

III. O jogo terminou empatado?

IV. Existe vida em outros planetas do universo.

V. Escreva uma poesia.

A frase que não possui essa característica comum é a

a) I.

b) II.

c) III.

d) IV.

e) V.

Resolução

As opções I, II, III, V são sentenças exclamativa, não possui predicado, interrogativa, imperativa respectivamente, logo, NÃO são proposições, sendo essa característica comum. IV é uma proposição.

3. **(BB2/2007/Cespe)** Uma proposição é uma afirmação que pode ser julgada como verdadeira (V) ou falsa (F), mas não como ambas. As proposições são usualmente simbolizadas por letras maiúsculas do alfabeto, como, por exemplo, P, Q, R, etc. Se a conexão de duas proposições é feita pela preposição "e", simbolizada usualmente por $\wedge$, então se obtém a forma $P \wedge Q$, lida como "P e Q" e avaliada como V se P e Q forem V, caso contrário, é F. Se a conexão for feita pela preposição "ou", simbolizada

usualmente por v, então se obtém a forma PvQ, lida como "P ou Q" e avaliada como F se P e Q forem F, caso contrário, é V. A negação de uma proposição é simbolizada por ¬P, e avaliada como V, se P for F, e como F, se P for V. A partir desses conceitos, julgue o próximo item. Há duas proposições no seguinte conjunto de sentenças:

(I) O BB foi criado em 1980.

(II) Faça seu trabalho corretamente.

(III) Manuela tem mais de 40 anos de idade.

Resolução

As opções I e III são proposições, já a II é uma sentença imperativa. A questão está correta.

4. **(SEBRAE 2010/CESPE-UnB)** Para os itens seguintes, serão consideradas como proposições apenas as sentenças declarativas, que mais facilmente são julgadas como verdadeiras — V — ou falsas — F —, deixando de lado as sentenças interrogativas, exclamativas, imperativas e outras. As proposições serão representadas por letras maiúsculas do alfabeto: A, B, C etc.

[...]

Sentenças como "x + 3 = 5", "Ele é um político", "x é jogador de futebol" são denominadas sentenças abertas; essas sentenças, como estão, não poderão ser julgadas como V ou F, pois os sujeitos, no caso, são variáveis. Essas expressões tornam-se proposições depois de substituída a variável por elemento determinado, permitindo o julgamento V ou F.

[...]

Tendo como referência as informações do texto, julgue os itens de 04 a 05.

04. Entre as frases apresentadas a seguir, identificadas por letras de A a E, apenas duas são proposições.

A: Pedro é marceneiro e Francisco, pedreiro.

B: Adriana, você vai para o exterior nessas férias?

C: Que jogador fenomenal!

D: Todos os presidentes foram homens honrados.

E: Não deixe de resolver a prova com a devida atenção.

Resolução

Frase A e D são proposições. B sentença interrogativa, C exclamativa, E imperativa

5. As frases "Transforme seus boletos de papel em boletos eletrônicos" e "O carro que você estaciona sem usar as mãos" são, ambas, proposições abertas.

Resolução

As duas frases não contém nenhuma variável.

Desenvolvendo o conteúdo

Uma vez compreendido o conceito de proposição, vamos compreender os três princípios fundamentais da Lógica Formal: Princípio da Identidade, Princípio da Não Contradição e o Princípio do Terceiro Excluído. Vejamos como podemos interpretar cada um deles.

1. Princípio da Identidade

- Se uma proposição qualquer é verdadeira, então ela é verdadeira. "Cada coisa é aquilo que é." (Gottfried Leibniz);

- Afirma A = A e não pode ser B, o que é, é;

- Este princípio afirma que proposições não podem ser mais verdadeiras que outras. Não há níveis de verdade. Todas as proposições estão em um mesmo nível de veracidade.

2. Princípio da Não Contradição

- Uma proposição não pode ser, simultaneamente, verdadeira e falsa;

- "Efetivamente, é impossível a quem quer que seja acreditar que uma mesma coisa seja e não seja." (Aristóteles);

- A = A e nunca pode ser não-A, o que é, é e não pode ser sua negação, ou seja, o ser é, o não ser não é;

- Este princípio afirma que a proposição não pode ter o valor de Verdadeiro e Falso ao mesmo tempo. Caso isso ocorra, temos a chamada contradição.

3. Princípio do Terceiro Excluído

- Toda proposição tem um dos dois valores lógicos: ou verdadeiro ou falso, excluindo-se qualquer outro;

- "Quem diz de uma coisa que é ou que não é ou dirá o verdadeiro ou dirá o falso. Mas se existisse um termo médio entre os dois contraditórios nem do ser nem do não ser poder-se-ia dizer que é o que não é." (Aristóteles);

- Afirma que ou A é x ou A é y, não existe uma terceira possibilidade;

- Este princípio afirma que só há dois valores lógicos possíveis: verdadeiro e falso.

Entendidos os três princípios fundamentais da lógica, é importante notar que uma proposição pode ser válida, entretanto, ser falsa ou verdadeira de acordo com a realidade. Por exemplo, seja p: "Natal foi fundada em 2000" é uma proposição válida, entretanto é falsa de acordo com a realidade, já a proposição q: "Natal se localiza no estado do Rio Grande do Norte" é uma proposição válida e verdadeira.

Atividade de aprendizagem 2

1. Em sua opinião, a expressão "Ser ou não ser" poderia representar qual princípio lógico?

2. Sendo que a expressão "O melhor queijo de manteiga é o de Caicó", é uma expressão válida e verdadeira?

Resumo

Na primeira aula do nosso curso, compreendemos os conceitos básicos da Lógica Clássica, como: argumento, premissas, conclusão, proposição. Começamos a entender o processo de formalização de expressões da língua natural em proposições, a fim de evitar textos dúbios. Por último, estudamos os princípios fundamentais da Lógica Clássica.

Leituras complementares

COSTA, P; RAPHAEL, W. **Uma breve história da lógica**. [2012]. Disponível em: <https://www.youtube.com/watch?v=ozMbmBp3onE>. Acesso em: 06 abr. 2015.

SOUSA, J. **Curso de raciocínio lógico:** aula 01. [2012]. Disponível em: <https://www.youtube.com/watch?v=uVBAOCFWSms>. Acesso em: 06 abr. 2015

Avaliando seus conhecimentos

1. **(TRT 17ª Região 2009/CESPE-UnB)** Proposições são frases que podem ser julgadas como verdadeiras — V — ou falsas — F —, mas não como V e F simultaneamente.

   [...]

A partir das informações do texto, julgue o item a seguir.

A sequência de frases a seguir contém exatamente duas proposições.

– A sede do TRT/ES localiza-se no município de Cariacica.

– Por que existem juízes substitutos?

– Ele é um advogado talentoso.

2. **(MRE 2008/CESPE-UnB)** Proposições são sentenças que podem ser julgadas como verdadeiras — V —, ou falsas — F —, mas não cabem a elas ambos os julgamentos.

   [...]

   Considerando as informações acima, julgue o item abaixo.

   Considere a seguinte lista de sentenças:

   I - Qual é o nome pelo qual é conhecido o Ministério das Relações Exteriores?

   II - O Palácio Itamaraty em Brasília é uma bela construção do século XIX.

   III - As quantidades de embaixadas e consulados gerais que o Itamaraty possui são, respectivamente, x e y.

   IV - O barão do Rio Branco foi um diplomata notável.

3. **(TCE-PB/2006/FCC)** Sabe-se que sentenças são orações com sujeito (o termo a respeito do qual se declara algo) e predicado (o que se declara sobre o sujeito). Na relação seguinte há expressões e sentenças:

   1. Três mais nove é igual a doze.

   2. Pelé é brasileiro.

   3. O jogador de futebol.

4. A idade de Maria.

5. A metade de um número.

6. O triplo de 15 é maior do que 10.

É correto afirmar que, na relação dada, são sentenças apenas os itens de números

a) 1,2 e 6.

b) 2,3 e 4.

c) 3,4 e 5.

d) 1,2,5 e 6.

e) 2,3,4 e 5.

4. Considere as seguintes expressões:

a) Ele foi um grande prefeito.

b) $\frac{x+5}{y} = 15$ é um número inteiro.

c) Luís da Câmara Cascudo foi um renomado historiador.

d) Não faça traquinagem!

Quais são consideradas proposições?

Referências

BEDREGAL, B. R. **Introdução à Lógica Clássica para a Ciência da Computação**. 2007. Disponível em: <https://www.dimap.ufrn.br/~jmarcos/books/BA_Jul07.pdf>. Acesso em: 05 fev. 2015.

COPI, I. M. **Introdução à lógica**. São Paulo: Ed. Mestre Jou, 1981.

COSTA, P.; RAPHAEL, W. **Uma breve história da Lógica**. [2012]. Disponível em: <https://www.youtube.com/watch?v=ozMbmBp3onE>. Acesso em: 09 abr. 2015.

ENCICLOPÉDIA Barsa Universal. 3. ed. São Paulo: Planeta do Brasil, 2010.

FAJARDO, R. A. **Introdução à Lógica**. [20--?]. Disponível em: <http://www.ime.usp.br/~fajardo/Logica.pdf>. Acesso em: 05 fev. 2015.

FREGE, G. Conceitografia, "Prefácio" (1879). In.:_____. **Lógica e Filosofia da Linguagem**. Tradução de Paulo Alcoforado. São Paulo: EDUSP, 2009. p. 43-50.

GERSTING, J. L. **Fundamentos matemáticos para ciência da computação**. São Paulo: LTC, 1999.

HUTH, M.; Ryan, M. **Lógica em ciência da computação**. São Paulo: LTC, 2008.

NOLT, J.; ROHATYN, D. **Lógica**. São Paulo: McGraw-Hill, 1991.

SANTOS, L. H. **O olho e o microscópio**. São Paulo: NAU, 2008.

SCALZITTI, A.; SILVA FILHO, J. I.; ABE, J. M. **Introdução à lógica para ciência da computação**. São Paulo: Arte e Ciência, 2001.

Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 02

Argumento Lógico, Proposições Simples, Princípios Lógicos

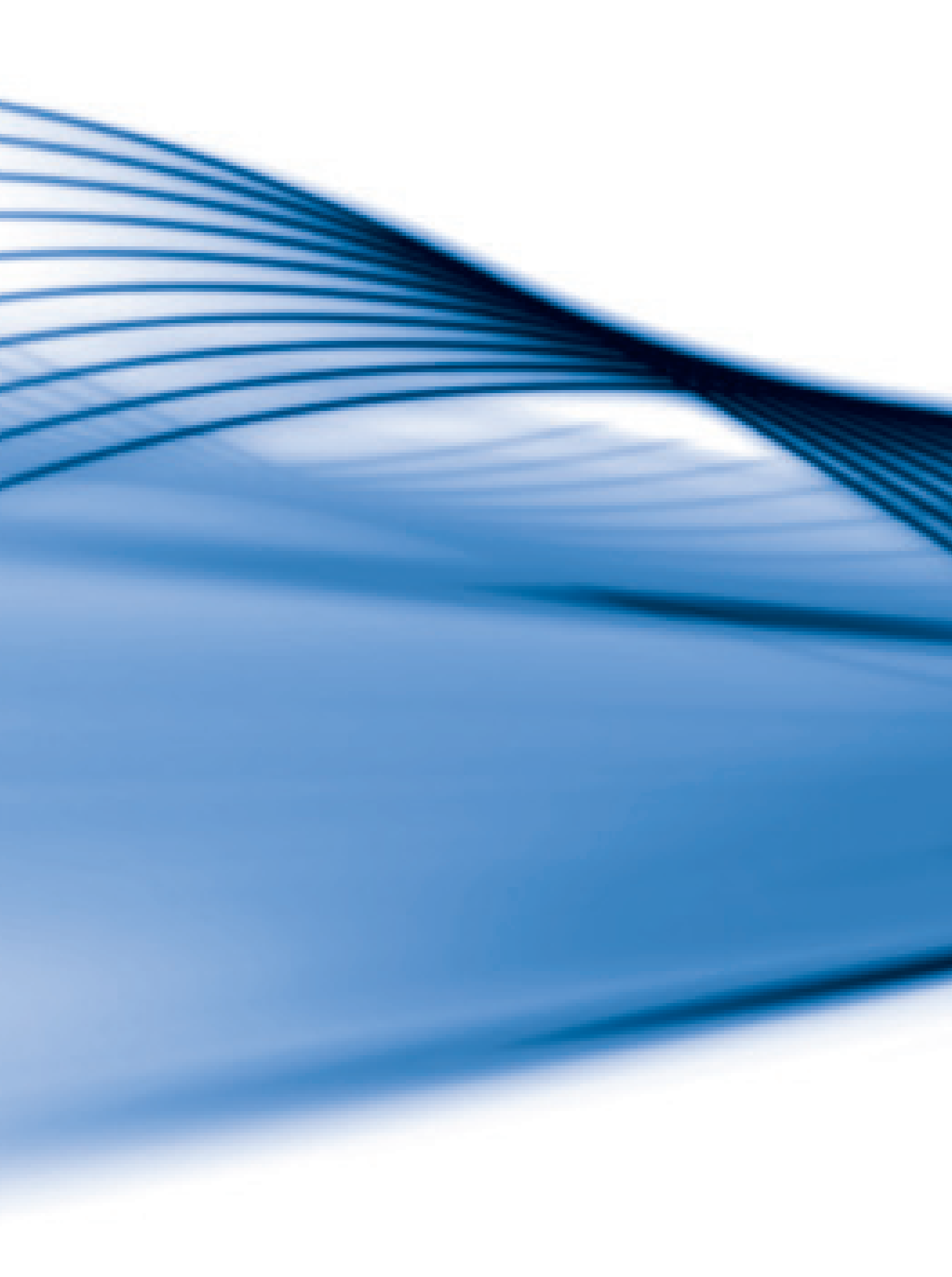
Thiago Medeiros Barros

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno! Espero que tenha gostado do início da nossa disciplina! Uma vez que compreendemos os conceitos iniciais, vamos estudar nesta aula Proposições Compostas, a fim de formalizar melhor expressões da nossa linguagem natural. Você também irá conhecer os famosos conectivos da Lógica Proposicional, como interpretá-los e utilizá-los.

Bons estudos!



Aula 02 - Argumento Lógico, Proposições Simples, Princípios Lógicos

Objetivos

Compreender como unir diferentes proposições em uma expressão complexa;

Utilizar e interpretar o significado dos conectivos: negação, conjunção, disjunção, implicação, bi-implicação;

Compreender a introdução do conceito de tabelas verdades.

Desenvolvendo o conteúdo

Veja a imagem a seguir:

Fonte: https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcSQla9DOoREAAeTOp53kDmklC8_Ni8qdNDX0ge4XTH8j2B8BJQo

Figura 01: exemplo de "brincadeiras" de raciocínio lógico

Na nossa última aula iniciamos a compreensão de alguns conceitos fundamentais na lógica, entre eles, o conceito da Proposição. Entretanto, estudamos apenas Proposições Simples, as quais conseguem representar pouca coisa do nosso mundo, por exemplo, como conseguir sistematizar em lógica proposicional o raciocínio da FIGURA 01. Surge então a necessidade de ligar proposições isoladas a partir dos **conectivos** com o intuíto de aumentar o poder de expressão. A essas novas estruturas se dá o nome de **Proposições Compostas**, que será base para a **Lógica Proposicional**. Com esse novo conhecimento, seremos capazes de transformar a linguagem natural em Lógica Formal, entendermos os significados dos conectivos e quando devemos aplicá-los, para, assim, conseguir chegamos a conclusões lógicas mais facilmente, sem cairmos nas falácias da linguagem natural. Além disso, vamos desenvolver um raciocínio mais sistemático, o qual nos ajudará na resolução de problemas matemáticos e lógicos do dia a dia.

As proposições compostas são combinações de proposições simples (também chamadas de átomos), através de unidades de ligação denominadas conectivos. A Lógica dispõe de cinco tipos de conectivos, sendo eles:

- Não (Negação), ¬ (também podendo usar o ' ~ ')
- E (Conjunção), Λ;
- Ou (Disjunção), V;
- Se – então (Condicional), → ; e
- Se e somente se (Bicondicional), ↔.

Exemplos:

- Não está chovendo;
- Está chovendo e está ventando;
- Está chovendo ou está nublado;
- Se choveu, então está molhado;
- Será aprovado se e somente se estudar.

Outro importante elemento da Lógica são os delimitadores representados pelos parênteses, os quais tem como objetivo evitar ambiguidades, exemplo:

p: Estudar; r: Fizer o trabalho;

q: Fazer a prova; s: Serei aprovado.

$((p \wedge q) \vee r) \rightarrow s$: "Se ((estudar e fazer a prova) ou fizer o trabalho), então será aprovado."

Escrever fórmulas corretas na Lógica Proposicional é fundamental para compreensão de outras pessoas, tal como ocorre na matemática, por exemplo a fórmula 1 + 1 = 2 está correta, mas a fórmula 1 + = 2 não. Na Lógica Proposicional também definimos regras para escrever fórmulas corretas, as quais são muito parecidas com a da matemática.

As proposições simples, juntos aos conectivos e parênteses formam o alfabeto da Lógica Proposicional. Além do alfabeto, há regras para construção de expressões complexas, ou seja, não é qualquer sequência de proposições, conectivos e parênteses que seja válida para a Lógica Proposicional. A essas regras, se dá o nome de gramática, sendo elas (Fajardo):

- Toda proposição é uma fórmula;
- Se A é uma fórmula, $(\neg A)$ também é uma fórmula;
- Se A e B são fórmulas $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ e $(A \leftrightarrow B)$) também são fórmulas;
- Não há fórmulas além das obtidas pelo uso das regras 1 a 3.

Como exemplo de fórmulas válidas de acordo com o alfabeto e a gramática defina, temos:

- p
- $\neg(\neg p)$

- ((¬p) Λ p)
- q → ((¬p) Λ p)

Como exemplo de fórmulas erradas gramaticalmente, temos:

- p Λ
- → p (¬ q)
- q ↔ (Λ q)
- ¬ p V
- (p→q(
- (¬ p v q))

ATIVIDADE

Marque as proposições válidas gramaticalmente abaixo. Para as proposições erradas, indique o problema.

- ¬p
- p Λ q V
- p → (q V r)
- t ↔ ¬ (¬ r)
- (q V r) Λ p ¬ t
- p¬
- →p V r
- ((p→q) →r

- $r \rightarrow (s \wedge \neg t)$
- $p \wedge q \vee t$

LEMBRE-SE!

Uma expressão pode ser considerada válida de acordo com a gramática, entretanto, não fazer sentindo com base em nosso mundo, por exemplo:

p: Natal tem praia;

q: unicórnios brancos podem voar;

$p \rightarrow$ q: "Se Natal tem praia, então unicórnios brancos podem voar."

A sentença acima está correta gramaticalmente, mas não faz sentido no mundo real. Julgar se uma expressão é verdadeira, óbvia, possível, falsa é uma questão de Semântica.

Outro importante conceito das proposições compostas é o Grau de Complexidade da fórmula. Como visto em (Fajardo):

DEFINIÇÃO

Para cada fórmula da lógica proposicional determinamos um número natural conforme as seguintes regras:

1. Uma fórmula atômica tem grau de complexidade 0;

2. Se A tem grau de complexidade n, a fórmula (¬A) tem grau de complexidade n + 1;

3. Se A e B têm graus de complexidade n e m, respectivamente, então

(A ∧ B), (A V B), (A → B) e (A ↔ B) têm grau de complexidade

Max(n,m) + 1, onde max(n,m) é o maior valor entre n e m.

Outra fórmula de entender o cálculo da complexidade é criar uma função grau tal que, sendo A uma fórmula, N o conjunto dos números naturais, n a complexidade da fórmula A, m a complexidade da fórmula B, temos: grau: A → N, com as seguintes regras:

- $grau\ (proposição) = 0$
- $grau\ (\neg A) = n + 1$
- $grau(A \wedge B)$, $grau(A \vee B)$, $grau(A \rightarrow B)$, $grau(A \leftrightarrow B)$ é $max\ (n,m) + 1$

Vamos exemplificar como calcular o grau de complexidade.

- $grau(p) = 0$
- $grau\ (\neg q)=1$
- $grau(p \rightarrow (\neg q)) = max(grau(p), grau(\neg q))+1 = \max(0,1) + 1 = 2.$

ATIVIDADE

Calcule o grau de complexidade utilizando a função grau das seguintes proposições:

1. $\neg(\neg p)$

2. $(p \rightarrow q) \ \Lambda \ (r \leftrightarrow q)$

3. $\neg t \rightarrow ((p \rightarrow q) \ \Lambda \ (r \leftrightarrow q))$

Valoração

Como foi visto anteriormente, uma proposição é considerada verdadeira ou falsa não por causa da gramática, mas sim devido a semântica. Para isso, utilizamos a função de Valoração. De acordo com (Fajardo), temos:

DEFINIÇÃO

Seja L a linguagem da lógica proposicional (isto é, o conjunto de fórmulas). Uma valoração é uma função V de L em {0,1} (sendo que 0 significa falso e 1 significa verdadeiro) que satisfaz as seguintes condições:

- V ($\neg$A) = 1 se, e somente se, V (A) = 0.
- V (A Λ B) = 1 se, e somente se, V (A) = 1 e V (B) = 1.
- V (A V B) = 1 se, e somente se, V (A) = 1 ou V (B) = 1.

- V (A → B) = 1 se, e somente se, V (A) = 0 ou V (B) = 1.
- V (A ↔ B) = 1 se, e somente se, V (A) = V (B).

Dizemos que uma fórmula A é verdadeira para a valoração V se V (A) = 1.

Se V (A) = 0 dizemos que A é falsa para a valoração V .

Outra forma de visualizar a função de valoração é a partir das chamadas tabelas verdades. Cada conectivo possuí a sua, vejamos!

Negação ¬

Toda proposição submetida à operação de negação resulta na sua contraditória.

Exemplo:

- **p:** está fazendo sol; **¬p: não** está fazendo sol.
- **q:** Thiago é mais alto que Rafael; **¬q: Não é verdade** que Thiago é mais alto que Rafael.

Tabela 1: Verdade Negação (¬)

P	¬p
1	0
0	1

Fonte: Autoria própria.

Conjunção Λ

As fórmulas com conjunção expressam a **concomitância** dos fatos, ou seja, p Λ q expressa que p e q ocorre ambos ao mesmo tempo.

Exemplo:

p: está fazendo sol; q: está um clima seco; ***p* Λ *q***: está fazendo sol **e** o clima está seco.

Tabela 2: Verdade Conjunção Λ		
P	Q	p Λq
1	1	1
1	0	0
0	1	0
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão só é considerada verdadeira, caso ambos (p e q) sejam verdadeiros.

Como visto em (NOLT; ROHATYN, 1991), a conjunção pode ser expressa porpalavras como: 'mas', 'todavia', 'embora', 'contudo', 'no entanto', 'visto que', 'enquanto', 'além disso'

Disjunção Inclusiva V

As fórmulas de disjunção expressam **que pelo menos um dos dois** fatores deve ocorrer **ou ambos**, ou seja, $p \vee q$ representa **ou** p ocorre **ou** q ocorre **ou** ambos ocorrem.

Exemplo:

p: está nublado; **q:** está ventando; p v q: está nublado **ou** está ventando.

Tabela 3: Disjunção inclusiva		
P	Q	p Λq
1	1	1
1	0	1
0	1	1
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se ambas as proposições forem falsas.

Disjunção Exclusiva V

As fórmulas de disjunção expressam **que pelo menos um dos dois** fatores deve ocorrer, mas **não ambos**, ou seja, p V q representa **ou** p ocorre **ou** q ocorre, mas não ambos.

Exemplo:

p: está nublado; **q:** está ventando; p V q: **ou** está nublado **ou** está ventando, mas não ambos.

Tabela 4: Disjunção exclusiva		
P	Q	p Λq
1	1	0
1	0	1
0	1	1
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se ambas as proposições forem atribuídas com os mesmos valores.

O **ou** exclusivo também pode ser formalizado como (p V q) $\Lambda \neg(p \Lambda q)$.

Implicação →

As fórmulas da implicação expressam a noção de consequência, ou seja, p→q representa que o fato expresso por **p** garante a ocorrência do fato expresso por **q**. Também conhecido como o se então. Outra nomenclatura para a implicação comumente utilizada é condicional material.

Exemplo:

- p: chove; q: molhado; $p \rightarrow q$: se choveu então está molhado
- Hoje é um fim de semana se hoje for sábado.
- p: hoje é fim de semana;
- q: hoje é sábado; q→p.
- A expressão também pode ser lida como se hoje é sábado, então é um fim de semana.

Tabela 5: Verdade

P	Q	p → q
1	1	1
1	0	0
0	1	1
0	0	1

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se a premissa p for verdade, chegando a uma conclusão q falsa. Por exemplo, imagina a lenda de que se levantar a meia-noite (m) e dá três pulos (p), então uma mulher de branco vai aparecer no teto (b), logo temos: m Λ p→b. Hoje, a fim de testar a lenda, você acabou de fazer o procedimento descrito: levantar-se a meia-noite e dar três pulos. Eentretanto, nenhuma mulher de branco apareceu no teto, logo, a expressão na Lógica está falsa, pois as premissas m

Λ p foram verdadeiras (pois você realizou o procedimento) e a conclusão (b) foi falsa. Nos últimos dois casos da tabela verdade, as premissas já são falsas, logo, para a implicação materialista, qualquer sentença declarativa que você afirmar a partir de premissas falsas, fará a expressão ser considerada verdadeira.

Outro fato interessante causado pela Língua Portuguesa e a Implicação é descrita por Nolt e Rohatyn (1991), A sentença **"p somente se q"** significa que: **p [pode ocorrer] somente se q [ocorre]. Se q não ocorre então p não ocorre, i.e., Se ¬q então ¬p é equivalente a Se p então q ou p→q.**

Exemplo:

- p: 48 é divisível por 6 somente se q: 48 é divisível por 3. Essa expressão é equivalente a Se p: 48 é divisível por 6, então q: 48 é divisível por 3.

Entretanto é importante notar que **p somente se q** é DIFERENTE de **p se q**. **p se q** é equivalente a q → p. Ou seja, 'somente se' é outro modo de expressar uma condicional, o qual, num enunciado com 'somente se', o que segue o 'se' é o consequente e não o antecedente. Assim, o enunciado 'Existe fogo somente se existir oxigênio' significa 'Se existir fogo, então existe oxigênio'.

Bi-Implicação ↔

As fórmulas da bi-implicação expressam que os fatos expressos por p e q são interdependentes, ou seja, ou **os dois ocorrem juntos ou nenhum dos dois ocorrem**. Também conhecido pela expressão **se somento se**.

Exemplo:

- p: será aprovado; q: estudar; p↔q: será aprovado se somente se estudar.

- **p**: Thiago é de Natal; **q**: Thalita é de Natal; **p↔q**: ou Thiago e Thalita são de Natal, ou não são.

Tabela 6: Verdade		
P	Q	$p \rightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que para a expressão ser verdadeira, ambas as proposições devem ser verdadeiras ou falsas, ou seja, com o mesmo valor. Essa expressão também é conhecida como equivalência, representada pelo símbolo ≡.

Agora que aprendemos os principais conectivos e suas respectivas tabelas verdades, vamos exercitar a transformar da linguagem natural para Lógica Proposicional e vice-versa.

ATIVIDADE

Traduza para a linguagem natural as fórmulas abaixo, utilizando o seguinte esquema:

- P: O conhecimento é surpreendente.
- Q: O conhecimento é prazeroso.
- R: O conhecimento está nos livros.

a) ¬P

b) P ^ Q

c) P ^ ¬Q

d) ¬P ^ Q

e) ¬(P ^ Q)

f) (f) P → Q

g) P ↔ (¬Q V R)

Escreva as fórmulas para as sentenças abaixo utilizando os seguintes símbolos proposicionais:

- P: Paula vai Estudar Lógica.
- Q: Quincas vai Estudar Lógica.
- R: Ricardo vai Estudar Lógica.
- S: Sara vai Estudar Lógica.

a) Paula não vai.

b) Paula vai, mas Quincas não vai.

c) Se Paula for, então Quincas também irá.

d) Paula irá, se Quincas for.

e) Paula irá, somente se Quincas for.

f) Paula irá se e somente se Quincas for.

g) Nem Paula nem Quincas irão.

h) Paula e Quincas não irão.

i) Paula vai ou Quincas não vai.

j) Paula não irá, se Quincas for.

k) Ou Paula vai, ou Ricardo e Quincas vão.

l) Se Paula for, então Ricardo e Quincas irão.

m) Paula não irá, mas Ricardo e Quincas irão.

n) Se Ricardo for, então se Paula não for, Quincas irá.

o) Se nem Ricardo nem Quincas forem, então Paula irá.

p) Ricardo irá, somente se Paula e Quincas não forem.

q) Se Ricardo ou Quincas forem, então Paula irá e Sara não irá.

r) Ricardo e Quincas irão se e somente se Paula ou Sara for.

s) Se Sara for, então Ricardo ou Paula irá, e se Sara não for, então Paula e Quincas irão.

RESUMINDO

Na nossa segunda aula do curso, aprendemos a utilizar e interpretar os principais conectivos da lógica proposicional. Além disso, vimos o conceito de valoração e grau de complexidade. Finalizamos nosso estudo exercitando como transformar a linguagem natural para lógica proposicional e vice-versa.

LEITURAS COMPLEMENTARES

ALMEIDA, J. M. **Lógica aplicada à computação**. [20--?]. Disponível em: <https://sites.google.com/site/sequiturquodlibet/courses/laac>. Acesso em: 18 mar. 2015.

SMULLYAM, R. M. F**irst-order Logic.** Nova York: Dover Publications, 1995.

SMULLYAM, R. M. **O Enigma de Sherazade**. Rio de Janeiro: Jorge Zahar Ed, 1998.

SMULLYAM, R. M. **Alice no País dos Enigmas**. Rio de Janeiro: Jorge Zahar Ed, 2000.

SMULLYAM, R. M. **A Dama ou o Tigre?**. Rio de Janeiro: Jorge Zahar Ed, 2004.

AVALIANDO SEUS CONHECIMENTOS

A partir do conhecimento da tabela verdade Verdade e da definição de Valoração, defina as regras da Função de Valoração para os conectivos (¬, V, →, ↔) quando o resultado é "Falso", por exemplo:

V (A ∧ B) = 0 se, e somente se, V (A) = 0 ou V (B) = 0

Escreva as sentenças a seguir utilizando a linguagem da Lógica Clássica Proposicional.

a) Carlos virá ao cinema e Maria não gostará, ou Carlos não virá ao cinema e Maria gostará do cinema.

b) Thiago irá correr, a menos que não chova.

c) Se chover irei ver filme, caso contrário vou à praia.

d) Irei ao teatro somente se for uma peça de comédia.

e) Se minha namorada vier, irei ao teatro somente se for uma peça de comédia.

Crie interpretações para o português para as seguintes fórmulas:

- ¬(P V Q)
- ¬P ^ (R ^ Q)
- (R V Q) → (P ^ ¬S)
- R → (¬P ^ ¬Q)
- (S → (R V P)) ^ (¬S → (P ^ Q))

Por exemplo:

- P: Comprar um cachorro;
- Q: Eu gosto de animais;
- R: Adotar um gato.;

A fórmula Q → (P V R) pode ser interpretada como: "se eu gosto de animais, então vou comprar um cachorro ou adotar um gato."

EXERCÍCIOS COMPLEMENTARES

1. (Gestor Fazendário-MG/2005/Esaf) Considere a afirmação P:

P: "A ou B"

Onde A e B, por sua vez, são as seguintes afirmações:

A: "Carlos é dentista".

B: "Se Enio é economista, então Juca é arquiteto".

Ora, sabe-se que a afirmação P é falsa. Logo:

a) Carlos não é dentista; Enio não é economista; Juca não é arquiteto.

b) Carlos não é dentista; Enio é economista; Juca não é arquiteto.

c) Carlos não é dentista; Enio é economista; Juca é arquiteto.

d) Carlos é dentista; Enio não é economista; Juca não é arquiteto.

e) Carlos é dentista; Enio é economista; Juca não é arquiteto.

Resolução

Para falsificar o 'ou' a expressão A e B ambas devem ser falsas. Para A ser falso temos: Carlos não é dentista. Para B ser falso temos: Enio é economista e Juca não é arquiteto. Letra B

2. (ALESP 2010/FCC) Paloma fez as seguintes declarações:

– "Sou inteligente e não trabalho."

– "Se não tiro férias, então trabalho."

Supondo que as duas declarações sejam verdadeiras, é FALSO concluir que Paloma

(A) é inteligente.

(B) tira férias.

(C) trabalha.

(D) não trabalha e tira férias.

(E) trabalha ou é inteligente.

Resolução

Para uma conjunção ser verdadeiras, ambas as proposições que compõem devem ser verdadeiras, logo, é inteligente e não trabalho são verdades. Portanto, letra C é a expressão falsa.

3. (Petrobras/2007/Cespe) Julgue o item que se segue.

Considere as proposições abaixo:

p: 4 é um número par;

q: A Petrobras é a maior exportadora de café do Brasil.

Nesse caso, é possível concluir que a proposição p ∨ q é verdadeira.

Resolução

Uma vez que a disjunção é verdadeira caso uma das proposições for verdadeira, e a proposição p é verdadeira, logo está correto a afirmação.

4. (INSS 2008/CESPE-UnB) Proposições são sentenças que podem ser julgadas como verdadeiras — V — ou falsas — F —, mas não como ambas. Se P e Q são proposições, então a proposição "Se P então Q", denotada por P→Q, terá valor lógico F quando P for V e Q for F, e, nos demais casos, será V. Uma expressão da forma ¬P, a negação da proposição P, terá valores lógicos contrários aos de P. P∨ Q, lida como "P ou Q", terá valor lógico F quando P e Q forem, ambas, F; nos demais casos, será V.

Considere as proposições simples e compostas apresentadas abaixo, denotadas por A, B e C, que podem ou não estar de acordo com o artigo 5.º da Constituição Federal.

A: A prática do racismo é crime afiançável.

B: A defesa do consumidor deve ser promovida pelo Estado.

C: Todo cidadão estrangeiro que cometer crime político em território brasileiro será extraditado.

De acordo com as valorações V ou F atribuídas corretamente às proposições A, B e C, a partir da Constituição Federal, julgue os itens a seguir.

a) Para a simbolização apresentada acima e seus correspondentes valores lógicos, a proposição B→C é V.

b) De acordo com a notação apresentada acima, é correto afirmar que a proposição (¬A)∨ (¬C) tem valor lógico F.

Resolução

Artigo 5º da Constituição Federal.

XXXII – o Estado promoverá, na forma da lei, a defesa do consumidor;

XLII – a prática do racismo constitui crime inafiançável e imprescritível, sujeito à pena de reclusão, nos termos da lei;

LII – não será concedida extradição de estrangeiro por crime político ou de opinião.

Logo, temos: V(A) = F, V(B) = V, V(C) = F, então V(B→C) = F e V((¬A)v (¬C)) = V

5. (SADPE/2008/FGV) Considere as situações abaixo:

I. Em uma estrada com duas pistas, vê-se a placa:

Caminhões → Pista da Direita

Como você está dirigindo um automóvel, você conclui que deve trafegar pela pista da esquerda.

II. Você mora no Recife e telefona para sua mãe em Brasília. Entre outras coisas, você diz que "Se domingo próximo fizer sol, eu irei à praia". No final do domingo, sua mãe viu pela televisão que choveu no Recife todo o dia. Então, ela concluiu que você não foi à praia.

III. Imagine o seguinte diálogo entre dois políticos que discutem calorosamente certo assunto:

- A: Aqui na Câmara tá cheio de ladrão.

- B: Ocorre que eu não sou ladrão.

- A: Você é safado, tá me chamando de ladrão.

Em cada situação há, no final, uma conclusão. Examinando a lógica na argumentação:

a) são verdadeiras as conclusões das situações I e II, apenas.

b) são verdadeiras as conclusões das situações II e III, apenas.

c) são verdadeiras as conclusões das situações I e III, apenas.

d) as três conclusões são verdadeiras.

e) as três conclusões são falsas.

Resolução

Para falsificar uma implicação o antecedente deve ser verdadeiro e o consequente falso. Quando temos um antecedente falso, não podemos afirmar nada do consequente, pois a expressão será verdadeira independente do seu valor. Na sentença I e II ambos os antecedentes são falsos, logo, no primeiro caso, ele pode ir para esquerda ou direita, no segundo caso, ir ou não para a praia, que a expressão será verdadeira. Ou seja, não se pode fazer nenhuma conclusão. No item III nenhum político chamou o outro de ladrão. Alternativa E correta.

REFERÊNCIAS

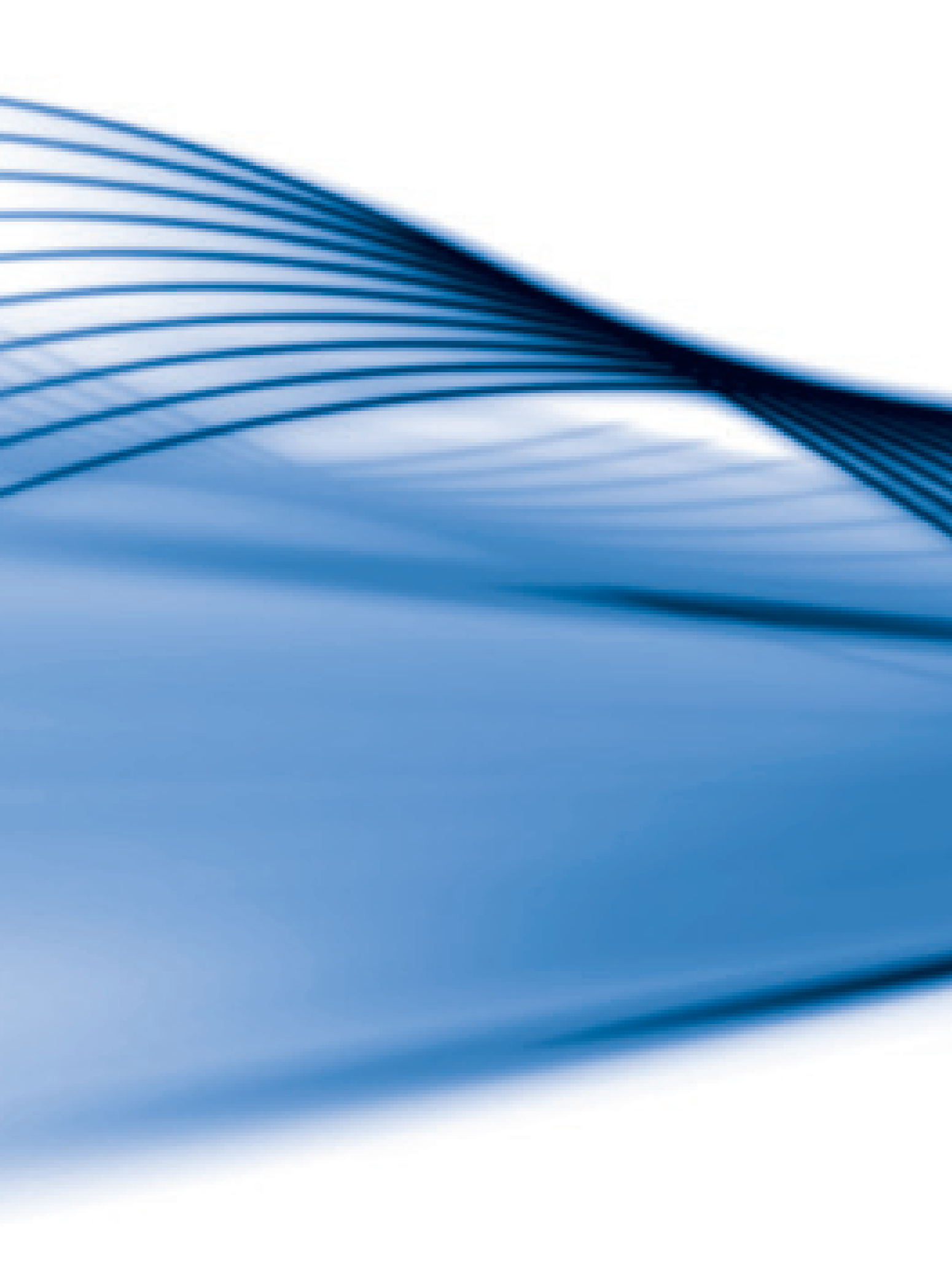
BEDREGAL, B. R. **Introdução à Lógica Clássica para a Ciência da Computação**. 2007. Disponível em: <https://www.dimap.ufrn.br/~jmarcos/books/BA_Jul07.pdf>. Acesso em: 05 fev. 2015.

ENCICLOPÉDIA Barsa Universal. 3. ed. São Paulo: Planeta do Brasil, 2010

FAJARDO, R. A. **Introdução à Lógica**. [20--?]. Disponível em: <http://www.ime.usp.br/~fajardo/Logica.pdf>. Acesso em: 05 fev. 2015.

NOLT, J.; ROHATYN, D. **Lógica**. São Paulo: McGraw-Hill, 1991.

SANTOS, L. H. **O olho e o microscópio.** São Paulo: NAU, 2008.



Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 02
Proposições Pompostas e Conectivos

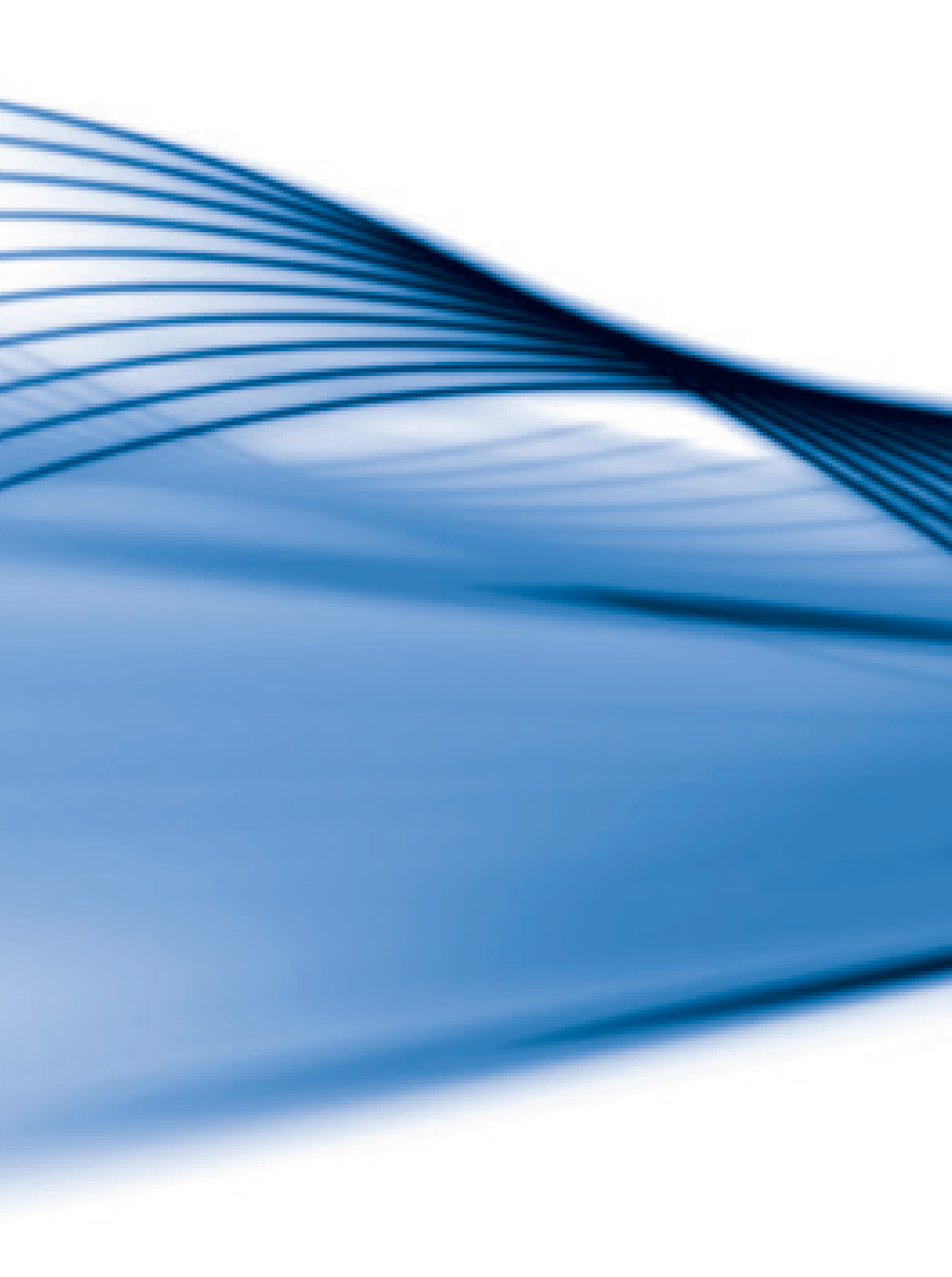
Thiago Medeiros Barros

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno! Espero que tenha gostado do início da nossa disciplina! Uma vez que compreendemos os conceitos iniciais, vamos estudar nesta aula Proposições Compostas, a fim de formalizar melhor expressões da nossa linguagem natural. Você também irá conhecer os famosos conectivos da Lógica Proposicional, como interpretá-los e utilizá-los.

Bons estudos!



Aula 02 - Argumento Lógico, Proposições Simples, Princípios Lógicos

Objetivos

Compreender como unir diferentes proposições em uma expressão complexa;

Utilizar e interpretar o significado dos conectivos: negação, conjunção, disjunção, implicação, bi-implicação;

Compreender a introdução do conceito de tabelas verdades.

Desenvolvendo o conteúdo

Veja a imagem a seguir:

Fonte: https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcSQla9DOoREAAeTOp53kDmklC8_Ni8qdNDX0ge4XTH8j2B8BJQo

Figura 01: exemplo de "brincadeiras" de raciocínio lógico

Na nossa última aula iniciamos a compreensão de alguns conceitos fundamentais na lógica, entre eles, o conceito da Proposição. Entretanto, estudamos apenas Proposições Simples, as quais conseguem representar pouca coisa do nosso mundo, por exemplo, como conseguir sistematizar em lógica proposicional o raciocínio da FIGURA 01. Surge então a necessidade de ligar proposições isoladas a partir dos **conectivos** com o intuíto de aumentar o poder de expressão. A essas novas estruturas se dá o nome de **Proposições Compostas**, que será base para a **Lógica Proposicional**. Com esse novo conhecimento, seremos capazes de transformar a linguagem natural em Lógica Formal, entendermos os significados dos conectivos e quando devemos aplicá-los, para, assim, conseguir chegamos a conclusões lógicas mais facilmente, sem cairmos nas falácias da linguagem natural. Além disso, vamos desenvolver um raciocínio mais sistemático, o qual nos ajudará na resolução de problemas matemáticos e lógicos do dia a dia.

As proposições compostas são combinações de proposições simples (também chamadas de átomos), através de unidades de ligação denominadas conectivos. A Lógica dispõe de cinco tipos de conectivos, sendo eles:

- Não (Negação), ¬ (também podendo usar o ' ~ ')
- E (Conjunção), Λ;
- Ou (Disjunção), V;
- Se – então (Condicional), → ; e
- Se e somente se (Bicondicional), ↔.

Exemplos:

- Não está chovendo;
- Está chovendo e está ventando;
- Está chovendo ou está nublado;
- Se choveu, então está molhado;
- Será aprovado se e somente se estudar.

Outro importante elemento da Lógica são os delimitadores representados pelos parênteses, os quais tem como objetivo evitar ambiguidades, exemplo:

p: Estudar; r: Fizer o trabalho;

q: Fazer a prova; s: Serei aprovado.

$((p \wedge q) \vee r) \rightarrow s$: "Se ((estudar e fazer a prova) ou fizer o trabalho), então será aprovado."

Escrever fórmulas corretas na Lógica Proposicional é fundamental para compreensão de outras pessoas, tal como ocorre na matemática, por exemplo a fórmula 1 + 1 = 2 está correta, mas a fórmula 1 + = 2 não. Na Lógica Proposicional também definimos regras para escrever fórmulas corretas, as quais são muito parecidas com a da matemática.

As proposições simples, juntos aos conectivos e parênteses formam o alfabeto da Lógica Proposicional. Além do alfabeto, há regras para construção de expressões complexas, ou seja, não é qualquer sequência de proposições, conectivos e parênteses que seja válida para a Lógica Proposicional. A essas regras, se dá o nome de gramática, sendo elas (Fajardo):

- Toda proposição é uma fórmula;
- Se A é uma fórmula, $(\neg A)$ também é uma fórmula;
- Se A e B são fórmulas $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ e $(A \leftrightarrow B)$) também são fórmulas;
- Não há fórmulas além das obtidas pelo uso das regras 1 a 3.

Como exemplo de fórmulas válidas de acordo com o alfabeto e a gramática defina, temos:

- p
- $\neg(\neg p)$

- ((¬p) Λ p)
- q → ((¬p) Λ p)

Como exemplo de fórmulas erradas gramaticalmente, temos:

- p Λ
- → p (¬ q)
- q ↔ (Λ q)
- ¬ p V
- (p→q(
- (¬ p v q))

ATIVIDADE

Marque as proposições válidas gramaticalmente abaixo. Para as proposições erradas, indique o problema.

- ¬p
- p Λ q V
- p → (q V r)
- t ↔ ¬ (¬ r)
- (q V r) Λ p ¬ t
- p¬
- →p V r
- ((p→q) →r

- $r \rightarrow (s \wedge \neg t)$
- $p \wedge q \vee t$

LEMBRE-SE!

Uma expressão pode ser considerada válida de acordo com a gramática, entretanto, não fazer sentindo com base em nosso mundo, por exemplo:

p: Natal tem praia;

q: unicórnios brancos podem voar;

$p \rightarrow q$: "Se Natal tem praia, então unicórnios brancos podem voar."

A sentença acima está correta gramaticalmente, mas não faz sentido no mundo real. Julgar se uma expressão é verdadeira, óbvia, possível, falsa é uma questão de Semântica.

Outro importante conceito das proposições compostas é o Grau de Complexidade da fórmula. Como visto em (Fajardo):

DEFINIÇÃO

Para cada fórmula da lógica proposicional determinamos um número natural conforme as seguintes regras:

1. Uma fórmula atômica tem grau de complexidade 0;

2. Se A tem grau de complexidade n, a fórmula (¬A) tem grau de complexidade n + 1;

3. Se A e B têm graus de complexidade n e m, respectivamente, então

(A ∧ B), (A V B), (A → B) e (A ↔ B) têm grau de complexidade

Max(n,m) + 1, onde max(n,m) é o maior valor entre n e m.

Outra fórmula de entender o cálculo da complexidade é criar uma função grau tal que, sendo A uma fórmula, N o conjunto dos números naturais, n a complexidade da fórmula A, m a complexidade da fórmula B, temos: grau: A → N, com as seguintes regras:

- $grau\ (proposição) = 0$
- $grau\ (\neg A) = n + 1$
- $grau(A \wedge B)$, $grau(A \vee B)$, $grau(A \rightarrow B)$, $grau(A \leftrightarrow B)$ é $max\ (n,m) + 1$

Vamos exemplificar como calcular o grau de complexidade.

- $grau(p) = 0$
- $grau\ (\neg q)=1$
- $grau(p \rightarrow (\neg q)) = max(grau(p), grau(\neg q))+1 = \max(0,1) + 1 = 2.$

ATIVIDADE

Calcule o grau de complexidade utilizando a função grau das seguintes proposições:

1. $\neg(\neg p)$

2. $(p \rightarrow q) \ \Lambda \ (r \leftrightarrow q)$

3. $\neg t \rightarrow ((p \rightarrow q) \ \Lambda \ (r \leftrightarrow q))$

Valoração

Como foi visto anteriormente, uma proposição é considerada verdadeira ou falsa não por causa da gramática, mas sim devido a semântica. Para isso, utilizamos a função de Valoração. De acordo com (Fajardo), temos:

DEFINIÇÃO

Seja L a linguagem da lógica proposicional (isto é, o conjunto de fórmulas). Uma valoração é uma função V de L em {0,1} (sendo que 0 significa falso e 1 significa verdadeiro) que satisfaz as seguintes condições:

- V ($\neg$A) = 1 se, e somente se, V (A) = 0.
- V (A Λ B) = 1 se, e somente se, V (A) = 1 e V (B) = 1.
- V (A V B) = 1 se, e somente se, V (A) = 1 ou V (B) = 1.

- V (A → B) = 1 se, e somente se, V (A) = 0 ou V (B) = 1.
- V (A ↔ B) = 1 se, e somente se, V (A) = V (B).

Dizemos que uma fórmula A é verdadeira para a valoração V se V (A) = 1.

Se V (A) = 0 dizemos que A é falsa para a valoração V .

Outra forma de visualizar a função de valoração é a partir das chamadas tabelas verdades. Cada conectivo possuí a sua, vejamos!

Negação ¬

Toda proposição submetida à operação de negação resulta na sua contraditória.

Exemplo:

- **p:** está fazendo sol; **¬p: não** está fazendo sol.
- **q:** Thiago é mais alto que Rafael; **¬q: Não é verdade** que Thiago é mais alto que Rafael.

Tabela 1: Verdade Negação (¬)

P	¬p
1	0
0	1

Fonte: Autoria própria.

Conjunção Λ

As fórmulas com conjunção expressam a **concomitância** dos fatos, ou seja, p Λ q expressa que p e q ocorre ambos ao mesmo tempo.

Exemplo:

p: está fazendo sol; q: está um clima seco; ***p* Λ *q***: está fazendo sol **e** o clima está seco.

Tabela 2: Verdade Conjunção Λ		
P	Q	p Λq
1	1	1
1	0	0
0	1	0
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão só é considerada verdadeira, caso ambos (p e q) sejam verdadeiros.

Como visto em (NOLT; ROHATYN, 1991), a conjunção pode ser expressa porpalavras como: 'mas', 'todavia', 'embora', 'contudo', 'no entanto', 'visto que', 'enquanto', 'além disso'

Disjunção Inclusiva V

As fórmulas de disjunção expressam **que pelo menos um dos dois** fatores deve ocorrer **ou ambos**, ou seja, $p \vee q$ representa **ou** p ocorre **ou** q ocorre **ou** ambos ocorrem.

Exemplo:

p: está nublado; **q:** está ventando; p v q: está nublado **ou** está ventando.

Tabela 3: Disjunção inclusiva		
P	Q	p Λq
1	1	1
1	0	1
0	1	1
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se ambas as proposições forem falsas.

Disjunção Exclusiva V

As fórmulas de disjunção expressam **que pelo menos um dos dois** fatores deve ocorrer, mas **não ambos**, ou seja, p V q representa **ou** p ocorre **ou** q ocorre, mas não ambos.

Exemplo:

p: está nublado; **q:** está ventando; p V q: **ou** está nublado **ou** está ventando, mas não ambos.

Tabela 4: Disjunção exclusiva		
P	Q	p Λq
1	1	0
1	0	1
0	1	1
0	0	0

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se ambas as proposições forem atribuídas com os mesmos valores.

O **ou** exclusivo também pode ser formalizado como (p V q) $\Lambda \neg(p \Lambda q)$.

Implicação →

As fórmulas da implicação expressam a noção de consequência, ou seja, p→q representa que o fato expresso por **p** garante a ocorrência do fato expresso por **q**. Também conhecido como o se então. Outra nomenclatura para a implicação comumente utilizada é condicional material.

Exemplo:

- p: chove; q: molhado; $p \rightarrow q$: se choveu então está molhado
- Hoje é um fim de semana se hoje for sábado.
- p: hoje é fim de semana;
- q: hoje é sábado; q→p.
- A expressão também pode ser lida como se hoje é sábado, então é um fim de semana.

Tabela 5: Verdade

P	Q	p → q
1	1	1
1	0	0
0	1	1
0	0	1

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que a expressão é falsa apenas se a premissa p for verdade, chegando a uma conclusão q falsa. Por exemplo, imagina a lenda de que se levantar a meia-noite (m) e dá três pulos (p), então uma mulher de branco vai aparecer no teto (b), logo temos: m Λ p→b. Hoje, a fim de testar a lenda, você acabou de fazer o procedimento descrito: levantar-se a meia-noite e dar três pulos. Eentretanto, nenhuma mulher de branco apareceu no teto, logo, a expressão na Lógica está falsa, pois as premissas m

Λ p foram verdadeiras (pois você realizou o procedimento) e a conclusão (b) foi falsa. Nos últimos dois casos da tabela verdade, as premissas já são falsas, logo, para a implicação materialista, qualquer sentença declarativa que você afirmar a partir de premissas falsas, fará a expressão ser considerada verdadeira.

Outro fato interessante causado pela Língua Portuguesa e a Implicação é descrita por Nolt e Rohatyn (1991), A sentença **"p somente se q"** significa que: **p [pode ocorrer] somente se q [ocorre]. Se q não ocorre então p não ocorre, i.e., Se ¬q então ¬p é equivalente a Se p então q ou p→q.**

Exemplo:

- p: 48 é divisível por 6 somente se q: 48 é divisível por 3. Essa expressão é equivalente a Se p: 48 é divisível por 6, então q: 48 é divisível por 3.

Entretanto é importante notar que **p somente se q** é DIFERENTE de **p se q**. **p se q** é equivalente a q → p. Ou seja, 'somente se' é outro modo de expressar uma condicional, o qual, num enunciado com 'somente se', o que segue o 'se' é o consequente e não o antecedente. Assim, o enunciado 'Existe fogo somente se existir oxigênio' significa 'Se existir fogo, então existe oxigênio'.

Bi-Implicação ↔

As fórmulas da bi-implicação expressam que os fatos expressos por p e q são interdependentes, ou seja, ou **os dois ocorrem juntos ou nenhum dos dois ocorrem**. Também conhecido pela expressão **se somento se**.

Exemplo:

- p: será aprovado; q: estudar; p↔q: será aprovado se somente se estudar.

- **p**: Thiago é de Natal; **q**: Thalita é de Natal; **p↔q**: ou Thiago e Thalita são de Natal, ou não são.

Tabela 6: Verdade		
P	Q	p → q
1	1	1
1	0	0
0	1	0
0	0	1

Fonte: Autoria própria.

A partir da tabela verdade, nota-se que para a expressão ser verdadeira, ambas as proposições devem ser verdadeiras ou falsas, ou seja, com o mesmo valor. Essa expressão também é conhecida como equivalência, representada pelo símbolo ≡.

Agora que aprendemos os principais conectivos e suas respectivas tabelas verdades, vamos exercitar a transformar da linguagem natural para Lógica Proposicional e vice-versa.

ATIVIDADE

Traduza para a linguagem natural as fórmulas abaixo, utilizando o seguinte esquema:

- P: O conhecimento é surpreendente.
- Q: O conhecimento é prazeroso.
- R: O conhecimento está nos livros.

a) ¬P

b) P ^ Q

c) P ^ ¬Q

d) ¬P ^ Q

e) ¬(P ^ Q)

f) (f) P → Q

g) P ↔ (¬Q V R)

Escreva as fórmulas para as sentenças abaixo utilizando os seguintes símbolos proposicionais:

- P: Paula vai Estudar Lógica.
- Q: Quincas vai Estudar Lógica.
- R: Ricardo vai Estudar Lógica.
- S: Sara vai Estudar Lógica.

a) Paula não vai.

b) Paula vai, mas Quincas não vai.

c) Se Paula for, então Quincas também irá.

d) Paula irá, se Quincas for.

e) Paula irá, somente se Quincas for.

f) Paula irá se e somente se Quincas for.

g) Nem Paula nem Quincas irão.

h) Paula e Quincas não irão.

i) Paula vai ou Quincas não vai.

j) Paula não irá, se Quincas for.

k) Ou Paula vai, ou Ricardo e Quincas vão.

l) Se Paula for, então Ricardo e Quincas irão.

m) Paula não irá, mas Ricardo e Quincas irão.

n) Se Ricardo for, então se Paula não for, Quincas irá.

o) Se nem Ricardo nem Quincas forem, então Paula irá.

p) Ricardo irá, somente se Paula e Quincas não forem.

q) Se Ricardo ou Quincas forem, então Paula irá e Sara não irá.

r) Ricardo e Quincas irão se e somente se Paula ou Sara for.

s) Se Sara for, então Ricardo ou Paula irá, e se Sara não for, então Paula e Quincas irão.

RESUMINDO

Na nossa segunda aula do curso, aprendemos a utilizar e interpretar os principais conectivos da lógica proposicional. Além disso, vimos o conceito de valoração e grau de complexidade. Finalizamos nosso estudo exercitando como transformar a linguagem natural para lógica proposicional e vice-versa.

LEITURAS COMPLEMENTARES

ALMEIDA, J. M. **Lógica aplicada à computação**. [20--?]. Disponível em: <https://sites.google.com/site/sequiturquodlibet/courses/laac>. Acesso em: 18 mar. 2015.

SMULLYAM, R. M. F**irst-order Logic.** Nova York: Dover Publications, 1995.

SMULLYAM, R. M. **O Enigma de Sherazade**. Rio de Janeiro: Jorge Zahar Ed, 1998.

SMULLYAM, R. M. **Alice no País dos Enigmas**. Rio de Janeiro: Jorge Zahar Ed, 2000.

SMULLYAM, R. M. **A Dama ou o Tigre?**. Rio de Janeiro: Jorge Zahar Ed, 2004.

AVALIANDO SEUS CONHECIMENTOS

A partir do conhecimento da tabela verdade Verdade e da definição de Valoração, defina as regras da Função de Valoração para os conectivos (¬, V, →, ↔) quando o resultado é "Falso", por exemplo:

V (A ∧ B) = 0 se, e somente se, V (A) = 0 ou V (B) = 0

Escreva as sentenças a seguir utilizando a linguagem da Lógica Clássica Proposicional.

a) Carlos virá ao cinema e Maria não gostará, ou Carlos não virá ao cinema e Maria gostará do cinema.

b) Thiago irá correr, a menos que não chova.

c) Se chover irei ver filme, caso contrário vou à praia.

d) Irei ao teatro somente se for uma peça de comédia.

e) Se minha namorada vier, irei ao teatro somente se for uma peça de comédia.

Crie interpretações para o português para as seguintes fórmulas:

- ¬(P V Q)
- ¬P ^ (R ^ Q)
- (R V Q) → (P ^ ¬S)
- R → (¬P ^ ¬Q)
- (S → (R V P)) ^ (¬S → (P ^ Q))

Por exemplo:

- P: Comprar um cachorro;
- Q: Eu gosto de animais;
- R: Adotar um gato.;

A fórmula Q → (P V R) pode ser interpretada como: "se eu gosto de animais, então vou comprar um cachorro ou adotar um gato."

EXERCÍCIOS COMPLEMENTARES

1. (Gestor Fazendário-MG/2005/Esaf) Considere a afirmação P:

P: "A ou B"

Onde A e B, por sua vez, são as seguintes afirmações:

A: "Carlos é dentista".

B: "Se Enio é economista, então Juca é arquiteto".

Ora, sabe-se que a afirmação P é falsa. Logo:

a) Carlos não é dentista; Enio não é economista; Juca não é arquiteto.

b) Carlos não é dentista; Enio é economista; Juca não é arquiteto.

c) Carlos não é dentista; Enio é economista; Juca é arquiteto.

d) Carlos é dentista; Enio não é economista; Juca não é arquiteto.

e) Carlos é dentista; Enio é economista; Juca não é arquiteto.

Resolução

Para falsificar o 'ou' a expressão A e B ambas devem ser falsas. Para A ser falso temos: Carlos não é dentista. Para B ser falso temos: Enio é economista e Juca não é arquiteto. Letra B

2. (ALESP 2010/FCC) Paloma fez as seguintes declarações:

– "Sou inteligente e não trabalho."

– "Se não tiro férias, então trabalho."

Supondo que as duas declarações sejam verdadeiras, é FALSO concluir que Paloma

(A) é inteligente.

(B) tira férias.

(C) trabalha.

(D) não trabalha e tira férias.

(E) trabalha ou é inteligente.

Resolução

Para uma conjunção ser verdadeiras, ambas as proposições que compõem devem ser verdadeiras, logo, é inteligente e não trabalho são verdades. Portanto, letra C é a expressão falsa.

3. (Petrobras/2007/Cespe) Julgue o item que se segue.

Considere as proposições abaixo:

p: 4 é um número par;

q: A Petrobras é a maior exportadora de café do Brasil.

Nesse caso, é possível concluir que a proposição p ∨ q é verdadeira.

Resolução

Uma vez que a disjunção é verdadeira caso uma das proposições for verdadeira, e a proposição p é verdadeira, logo está correto a afirmação.

4. (INSS 2008/CESPE-UnB) Proposições são sentenças que podem ser julgadas como verdadeiras — V — ou falsas — F —, mas não como ambas. Se P e Q são proposições, então a proposição "Se P então Q", denotada por P→Q, terá valor lógico F quando P for V e Q for F, e, nos demais casos, será V. Uma expressão da forma ¬P, a negação da proposição P, terá valores lógicos contrários aos de P. P∨ Q, lida como "P ou Q", terá valor lógico F quando P e Q forem, ambas, F; nos demais casos, será V.

Considere as proposições simples e compostas apresentadas abaixo, denotadas por A, B e C, que podem ou não estar de acordo com o artigo 5.º da Constituição Federal.

A: A prática do racismo é crime afiançável.

B: A defesa do consumidor deve ser promovida pelo Estado.

C: Todo cidadão estrangeiro que cometer crime político em território brasileiro será extraditado.

De acordo com as valorações V ou F atribuídas corretamente às proposições A, B e C, a partir da Constituição Federal, julgue os itens a seguir.

a) Para a simbolização apresentada acima e seus correspondentes valores lógicos, a proposição B→C é V.

b) De acordo com a notação apresentada acima, é correto afirmar que a proposição (¬A)∨ (¬C) tem valor lógico F.

Resolução

Artigo 5º da Constituição Federal.

XXXII – o Estado promoverá, na forma da lei, a defesa do consumidor;

XLII – a prática do racismo constitui crime inafiançável e imprescritível, sujeito à pena de reclusão, nos termos da lei;

LII – não será concedida extradição de estrangeiro por crime político ou de opinião.

Logo, temos: V(A) = F, V(B) = V, V(C) = F, então V(B→C) = F e V((¬A)v (¬C)) = V

5. (SADPE/2008/FGV) Considere as situações abaixo:

I. Em uma estrada com duas pistas, vê-se a placa:

Caminhões → Pista da Direita

Como você está dirigindo um automóvel, você conclui que deve trafegar pela pista da esquerda.

II. Você mora no Recife e telefona para sua mãe em Brasília. Entre outras coisas, você diz que "Se domingo próximo fizer sol, eu irei à praia". No final do domingo, sua mãe viu pela televisão que choveu no Recife todo o dia. Então, ela concluiu que você não foi à praia.

III. Imagine o seguinte diálogo entre dois políticos que discutem calorosamente certo assunto:

- A: Aqui na Câmara tá cheio de ladrão.

- B: Ocorre que eu não sou ladrão.

- A: Você é safado, tá me chamando de ladrão.

Em cada situação há, no final, uma conclusão. Examinando a lógica na argumentação:

a) são verdadeiras as conclusões das situações I e II, apenas.

b) são verdadeiras as conclusões das situações II e III, apenas.

c) são verdadeiras as conclusões das situações I e III, apenas.

d) as três conclusões são verdadeiras.

e) as três conclusões são falsas.

Resolução

Para falsificar uma implicação o antecedente deve ser verdadeiro e o consequente falso. Quando temos um antecedente falso, não podemos afirmar nada do consequente, pois a expressão será verdadeira independente do seu valor. Na sentença I e II ambos os antecedentes são falsos, logo, no primeiro caso, ele pode ir para esquerda ou direita, no segundo caso, ir ou não para a praia, que a expressão será verdadeira. Ou seja, não se pode fazer nenhuma conclusão. No item III nenhum político chamou o outro de ladrão. Alternativa E correta.

REFERÊNCIAS

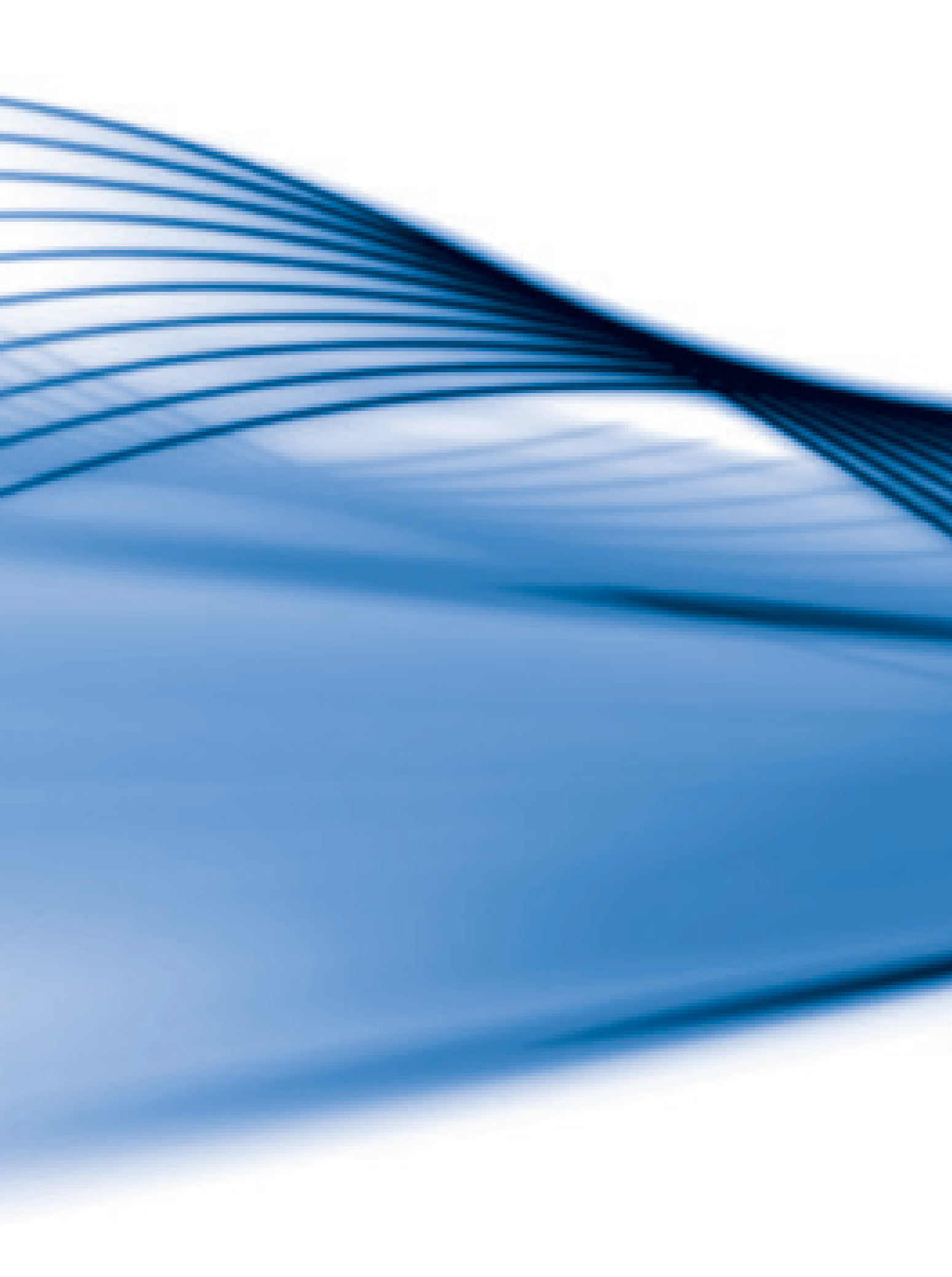
BEDREGAL, B. R. **Introdução à Lógica Clássica para a Ciência da Computação**. 2007. Disponível em: <https://www.dimap.ufrn.br/~jmarcos/books/BA_Jul07.pdf>. Acesso em: 05 fev. 2015.

ENCICLOPÉDIA Barsa Universal. 3. ed. São Paulo: Planeta do Brasil, 2010

FAJARDO, R. A. **Introdução à Lógica**. [20--?]. Disponível em: <http://www.ime.usp.br/~fajardo/Logica.pdf>. Acesso em: 05 fev. 2015.

NOLT, J.; ROHATYN, D. **Lógica**. São Paulo: McGraw-Hill, 1991.

SANTOS, L. H. **O olho e o microscópio.** São Paulo: NAU, 2008.



Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 03
Construção de Tabelas Verdades, Tautologia, Contradição e Contingência

Thiago Medeiros Barros

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno! Ultrapassamos a metade da nossa unidade e espero que esteja gostando! Após aprendermos a formalizar a linguagem natural em Proposições, vamos agora construir tabelas verdades a partir dessas proposições complexas e realizar conclusões lógicas. Vamos também entender o conceito de contingência, tautologia e contradição.

Bons estudos!

Aula 3 - Construção de Tabelas Verdades, Tautologia, Contradição e Contingência

Objetivos

Construir Tabelas Verdades;

Compreender o conceito de Tautologia, Contradição e Contingência.

Desenvolvendo o conteúdo

Fonte: http://goo.gl/C3tDlD

Figura 1: Quadrinho utilizando o conceito da disjunção

Na unidade anterior fomos introduzidos ao conceito de Tabela Verdade, a qual se caracteriza como uma ferramenta para auxiliar na valoração das proposições. Entretanto, apenas utilizamos essa técnica em proposições com um único conectivo. Nesta unidade vamos utilizar essa importante ferramenta para realizar a valoração de proposições compostas. Lembrem-se que as proposições compostas são formadas de proposições simples (p, q, r, s), as quais vamos chamá-las de ***átomos***, a fim de melhorar a compreensão.

O objetivo da tabela verdade é construir **todas as possibilidades** de valoração para uma proposição, dado que cada átomo pode assumir valores Verdadeiro (1) ou Falso (0). Uma vez que a tabela verdade representa todas

possibilidades de valoração de uma fórmula, a quantidade de linhas que compõe uma tabela verdade é igual a **2^n sendo *n* a quantidade de átomos diferentes** que compõe a expressão e **2 a quantidade de valores que uma proposição pode assumir** (Verdadeiro ou Falso). Vamos ver alguns exemplos.

- Dois átomos (a e b) em um total de 4 linhas (2^2)

Tabela 1: Dois átomos (a e b) em um total de 4 linhas (2^2)

A	B	¬(a → b) ∧ (b ∨ a)
1	1	0
1	0	1
0	1	0
0	0	0

Fonte: Autoria própria.

- 3 átomos (a, b e c) em um total de 8 linhas (2^3)

Tabela 02: três átomos (a, b e c) em um total de 8 linhas (2)

A	B	C	¬(a → b) ∧ (b → c)
1	1	1	0
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Fonte: Autoria própria.

Como visto no início da aula, o objetivo da Tabela Verdade é conseguir analisar todas as possibilidades de valoração para a proposição. Entretanto, a quantidade de linhas de uma Tabela Verdade cresce exponencialmente de acordo com a quantidade de átomos diferentes da proposição. Logo, é importante criar uma metodologia para construir a Tabela Verdade, a fim de não se confundir durante o processo de construção.

Construção de Tabela Verdade

Como visto em Nolt e Rohatyn (1991), utiliza-se um método de *dividir para conquistar*, a fim de realizar todas as combinações possíveis de valoração. De forma resumida, primeiro determina-se os valores-verdades dos átomos, logo após as subfórmulas (aquelas que, através dos conectivos ligam os

átomos), então as subformulas que conectam as subfórmulas anteriores, e assim por diante. Finalmente, calcula-se os valores da fórmula principal.

Fonte: http://goo.gl/xmBgPV

Figura 2: Todo conhecimento deve ser construído passo a passo

Talvez a explicação anterior tenha ficado um pouco abstrata. Por isso, logo abaixo segue uma descrição passo-a-passo para construção de uma tabela verdade complexa e um exemplo para melhor entendimento.

Os passos para compor uma Tabela Verdade podem ser resumidos abaixo:

1. Calcule a quantidade de linhas pela fórmula, sendo a quantidade de átomos distintos.

2. Coloque uma coluna para cada átomo.

3. Faça todas as combinações possíveis dos valores lógicos. **Dica**: Na primeira coluna preencha a primeira metade de linhas com valores verdadeiros, a segunda metade das linhas com valores falsos; já a próxima coluna, preencha alternando a cada ¼ das linhas com valores verdadeiros e depois ¼ das linhas com valores falsos até completar, na próxima coluna, faça o mesmo procedimento, mas dessa vez alternando a cada 1/8 das linhas e assim por diante. (Mais a frente haverá um exemplo para melhor ilustração do método). Faça esse procedimento até preencher todas as colunas das proposições simples.

4. Coloque uma coluna para cada átomo que esteja com o conectivo de negação.

5. Percorra a expressão da **esquerda para a direita**, separando as subfórmulas dos **parênteses mais internos aos mais externos**, na ordem em que aparecerem. Caso não haja parentes separando as subfórmulas, a ordem de precedência, será a seguinte:

    a) negação;

    b) conjunção e disjunção;

    c) condicionamento;

    d) bicondicionamento.

Para melhor entendimento dos passos acima descritos, veja o exemplo abaixo.

- A expressão $\neg((a \rightarrow \neg b) \wedge (b \vee a)) \rightarrow c$

    a) Átomos distintas: a, b, c, logo são = 8 linhas

    b) Colocar cada átomo em uma coluna

A	B	C

    c) Combinação dos valores lógicos utilizando a dica: primeira coluna ½ das linhas Verdadeiro (1) e ½ das linhas Falso (0), segunda coluna alternando entre ¼ Verdadeiro (8 linhas, logo, 8/4 = 2 linhas, ou seja, alternar a cada 2 linhas com verdadeiro e falso) e ¼ Falso, terceira coluna

alternando entre 1/8 Verdadeiro (8 linhas, logo, 8/8 = 1 linha, ou seja, alternar a cada 1 linha com verdadeiro e falso) e 1/8 Falso.

A	B	C
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

d) Insira as colunas dos átomos com o conectivo de negação.

$\neg((a \rightarrow \mathbf{\neg b}) \wedge (b \vee a)) \rightarrow c$

A	B	C	¬b
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

e) As subfórmulas de parênteses mais internos por ordem de precedência ¬((**a** → **¬b**) ∧ (**b** ∨ **a**)) → c:

- (a → ¬b), Operações por ordem:
  - ¬b (já existia uma coluna com essa subfórmula, logo, não precisava incluir uma nova coluna)
  - a → ¬b

A	B	C	¬b	a → ¬b
1	1	1		
1	1	0		
1	0	1		
1	0	0		
0	1	1		
0	1	0		
0	0	1		
0	0	0		

- (b ∨ a), Operações por ordem de precedência:

  ¬((**a** → **¬b**) ∧ (**b** ∨ **a**)) → c

  - b ∨ a

A	B	C	¬b	a → ¬b	b ∨ a
1	1	1			
1	1	0			
1	0	1			
1	0	0			
0	1	1			
0	1	0			
0	0	1			
0	0	0			

- ¬((a → ¬b) ∧ (b ∨ a)) → c, Operações por ordem de precedência:
    - (a → ¬b) ∧ (b ∨ a), obs.: cada parênteses interno já foi resolvido no passo anterior

A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)
1	1	1				
1	1	0				
1	0	1				
1	0	0				
0	1	1				
0	1	0				
0	0	1				
0	0	0				

- ¬((a → ¬b) ∧ (b ∨ a)) → c, Operações por ordem de precedência:
    - ¬((a → ¬b) ∧ (b ∨ a))
    - ¬((a → ¬b) ∧ (b ∨ a)) → c

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1						
2º	1	1	0						
3º	1	0	1						
4º	1	0	0						
5º	0	1	1						
6º	0	1	0						
7º	0	0	1						
8º	0	0	0						

Fonte: http://goo.gl/PYVIO9

Figura 3: Todo esforço será recompensado! Não desista!

Uma vez construída a Tabela Verdade, realize as operações contidas em cada uma das colunas da esquerda para direita se baseando nas tabelas verdades básicas dos conectivos vistos na aula anterior, por exemplo, vamos nos basear apenas na primeira linha.

LEMBRE-SE

Recapitule todas as tabelas verdades dos conectivos da aula passada.

Tabela 3: Resumo das tabelas verdades dos principais conectivos

		Conjunto (E)	Disjunção (Ou)	Implicação (Se...Então)	Bi-Implicação (se e somente se)	Negação
p	**q**	**p ∧q**	**p ∨ q**	**p→q**	**p↔q**	**¬p**
1	1	1	1	1	1	1
1	0	0	1	0	0	0
0	1	0	1	1	0	1
0	0	0	0	1	1	1

Fonte: Autoria própria.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1						

A coluna D deve negar o valor do átomo b (que se encontra na coluna B), ou seja, negar Verdadeiro (1) é igual a Falso (0).

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0					

A coluna E deve realizar a implicação da coluna A com a coluna D, ou seja, Verdadeiro implicando em Falso, pela Tabela Verdade da implicação vista na aula anterior, é Falso.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0				

A coluna F deve realizar a disjunção entre a coluna B e A, ou seja, Verdadeiro ou Verdadeiro é igual a Verdadeiro.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1			

A coluna G deve realizar a conjunção entre a coluna E e F, ou seja, Falso e Verdadeiro é Falso.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1	0		

A coluna H deve realizar a negação da coluna G, ou seja, **negação** de Falso é Verdadeiro.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1	0	1	

A coluna I deve realizar a implicação da coluna H na coluna C, ou seja, Verdadeiro implicando em Verdadeiro é Verdadeiro.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1	0	1	1

Repetindo as operações acima para todas as linhas, vamos ter a tabela abaixo completa.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1	0	1	1
2º	1	1	0	0	0	1	0	1	1
3º	1	0	1	1	1	1	1	0	1
4º	1	0	0	1	1	1	1	0	1
5º	0	1	1	0	1	1	1	0	1
6º	0	1	0	0	1	1	1	0	1
7º	0	0	1	1	1	0	0	1	1
8º	0	0	0	1	1	0	0	1	0

A última coluna é a fórmula final. As partes importantes da tabela verdade que devem ter destaque são: os valores de cada átomo e a coluna correspondente a fórmula final. Todas as colunas intermediárias são apenas para facilitar a construção.

	A	B	C	D	E	F	G	H	I
	A	B	C	¬b	a → ¬b	b ∨ a	(a→¬b)∧ (b ∨ a)	¬((a → ¬b) ∧ (b ∨ a))	¬((a → ¬b) ∧ (b ∨ a)) → c
1º	1	1	1	0	0	1	0	1	1
2º	1	1	0	0	0	1	0	1	1
3º	1	0	1	1	1	1	1	0	1
4º	1	0	0	1	1	1	1	0	1
5º	0	1	1	0	1	1	1	0	1
6º	0	1	0	0	1	1	1	0	1
7º	0	0	1	1	1	0	0	1	1
8º	0	0	0	1	1	0	0	1	0

ATIVIDADE

Construa as tabelas verdades das expressões abaixo.

- ¬p → q
- ¬ (¬p v q) ∧ q
- ((p v q) → r) ∧ (¬r v p)

A partir da construção de tabelas verdades, podemos classificar uma proposição de acordo com a valoração obtida, sendo elas: Tautologia, Contradição e Contingência.

Tautologia

Definição: é toda proposição composta cujo **valor lógico é sempre verdadeiro**, quaisquer que sejam os valores lógicos dos seus átomos que a compõe. As tautologias são também conhecidas como **proposições tautológicas** ou **proposições logicamente verdadeiras**.

Exemplo:

- *p* v ¬*p*

P	¬p	p v ¬p
1	0	1
0	1	1

- $p \rightarrow q \longleftrightarrow \neg p \vee q$

P	Q	¬p	p ⟷q	¬p v q	p → q⟷¬p v q
1	1	0	1	1	1
0	0	0	0	0	1
0	1	1	1	1	1
0	0	1	1	1	1

Abaixo, segue algumas tautologias famosas na lógica clássica.

- p → (q →p)
- (p → (q → r)) → ((p → q) → (p → r))
- (¬q → ¬p) → ((¬q → p) → q)
- a ∧ b → a
- a ∧ b → b
- a → ¬ ¬a
- ¬ ¬a → a
- a → a ∨ b
- a ∧ ¬a → b

Contradição

Definição: é toda proposição composta cujo o **valor lógico é sempre falso**, quais quer que sejam os valores lógicos dos seus átomos. As contradições são também denominadas **proposições contraválidas** ou **proposições logicamente falsas.**

Exemplo:

- $p \wedge \neg p$

P	¬p	p ∧ ¬p
1	0	0
0	1	0

- $p \rightarrow q \longleftrightarrow \neg(\neg p \vee q)$

P	Q	¬p	p →q	¬p v q	¬(¬p v q)	p → q⟷¬(¬p v q)
1	1	0	1	1	0	0
0	0	0	0	0	1	0
0	1	1	1	1	0	0
0	0	1	1	1	0	0

Contingência

Definição: é toda proposição composta que **não é tautológica e nem contradição**, ou seja, apresenta em sua coluna os valores lógicos Verdade e Falso. As contingências são também chamadas de **proposições contingentes** ou **proposições indeterminadas**.

Exemplo:

- $p \rightarrow \neg p$

P	¬p	p →¬p
1	0	0
0	1	1

- $p \rightarrow q \longleftrightarrow \neg p \wedge q$

P	Q	¬p	p →q	¬p ∨ q	p → q⟷¬p ∨ q
1	1	0	1	0	0
1	0	0	0	0	1
0	1	1	1	1	1
0	0	1	1	0	0

ATIVIDADE

Defina 6 proposições com suas respectivas tabelas verdades, sendo duas tautológias, duas contradições e duas contingências.

RESUMINDO

Nesta aula aprendemos como construir tabelas verdades para proposições compostas, além de compreender e classificar o que são tautologias, contradição e contingência.

LEITURAS COMPLEMENTARES

Lolita é um projeto desenvolvido no DIMAP/UFRN coordenado pelo prof. João Marcos de Almeida, o qual apresenta diversas ferramentas para trabalhar com lógica proposicional, inclusive, construção de tabelas verdades (Truth Table).

LOGICAMENTE BETA. Disponível em: <http://lolita.dimap.ufrn.br/logicamente/>. Acesso em: 06 abr. 2015.

Truth Table Generator é um construtor de tabela verdade.

TRUTH TABLE GENERATOR. Disponível em: <http://programming.dojo.net.nz/study/truth-table-generator/index>. Acesso em: 06 abr. 2015.

AVALIANDO SEUS CONHECIMENTOS

1. Demonstre que a proposição a seguir é uma tautologia.

$(\alpha \rightarrow \beta) \leftrightarrow ((\neg\beta) \rightarrow (\neg\alpha))$

2. Construa as tabelas verdades das proposições abaixo e defina se é uma Tautologia, Contradição ou Contingência.

a) $a. \neg p \vee \neg q \rightarrow (p \rightarrow q)$

b) b. (p ↔ ¬q) ↔ ¬p ∧ q

c) c. (p V (q → ¬ r))∧ (¬r V r ↔ ¬q)

3. Escreve as fórmulas das sentenças abaixo e construa suas tabelas verdades. Considere as seguintes proposições:

P: Thiago é um bom professor de Lógica.

Q: Eu aprendo tudo de Lógica na sala de aula.

R: Eu fiz todos os exercícios de Lógica.

S: Vou ficar em recuperação em Lógica.

T: Eu amo Lógica.

d) a. Se Thiago é um bom professor de Lógica e ou aprendo tudo na sala de aula ou faço todos os exercícios de Lógica, então não vou ficar em recuperação.

e) b. Eu amo Lógica e vou ficar em recuperação, somente se Thiago não for um bom professor de Lógica e eu não fizer todos os exercícios de Lógica.

EXERCÍCIOS COMPLEMENTARES

1. **(STF 2008/CESPE-UnB)** Filho meu, ouve minhas palavras e atenta para meu conselho. A resposta branda acalma o coração irado.

O orgulho e a vaidade são as portas de entrada da ruína do homem.

Se o filho é honesto, então o pai é exemplo de integridade.

Tendo como referência as quatro frases acima, julgue os itens seguintes.

a. A primeira frase é composta por duas proposições lógicas simples unidas pelo conectivo de conjunção.

b. A segunda frase é uma proposição lógica simples.

c. A terceira frase é uma proposição lógica composta.

d. A quarta frase é uma proposição lógica em que aparecem dois conectivos lógicos.

Resolução

A: sentença imperativa, b: correto, c: proposição simples com sujeito composto, d: aparece apenas a implicação.

2. (TCU/2004/Cespe) Considere que as letras P, Q e R representam proposições, e os símbolos ¬ , ∧ e são operadores lógicos que constroem novas proposições e significam "não", "e" e "então", respectivamente. Na lógica proposicional que trata da expressão do raciocínio por meio de proposições que são avaliadas (valoradas) como verdadeiras (V) ou falsas (F), mas nunca ambos, esses operadores estão definidos, para cada valoração atribuída às letras proposicionais, na tabela abaixo.

P	Q	¬P	P∧Q	P⟶Q
V	V	F	V	V
V	F	F	F	F
F	V	V	F	V
F	F	V	F	V

Suponha que P representa a proposição Hoje choveu, Q represente a proposição José foi à praia e R represente a proposição Maria foi ao comércio. Com base nessas informações e no texto, julgue os itens a seguir:

a. A sentença "Hoje não choveu então Maria não foi ao comércio e José não foi à praia" pode ser corretamente representada por ¬P → (¬R ∧ ¬Q)

b. A sentença "Hoje choveu e José não foi à praia" pode ser corretamente representada por P ∧ ¬Q

c. Se a proposição "Hoje não choveu" for valorada como F e a proposição José foi à praia for valorada como V, então a sentença representada por ¬P → Q é falsa.

d. O número de valorações possíveis para (Q ∧ ¬R) → P é inferior a 9.

Resolução

A:correto, b:correto, c: F →V = V, logo incorreto, d:número de valorações são 8 (2^3), inferior a 9, logo, correto.

3. (Gestor Fazendário-MG/2005/Esaf) Considere a afirmação P:

P: "A ou B"

Onde A e B, por sua vez, são as seguintes afirmações:

A: "Carlos é dentista".

B: "Se Enio é economista, então Juca é arquiteto".

Ora, sabe-se que a afirmação P é falsa. Logo,

a) Carlos não é dentista; Enio não é economista; Juca não é arquiteto.

b) Carlos não é dentista; Enio é economista; Juca não é arquiteto.

c) Carlos não é dentista; Enio é economista; Juca é arquiteto.

d) Carlos é dentista; Enio não é economista; Juca não é arquiteto.

e) Carlos é dentista; Enio é economista; Juca não é arquiteto.

Resolução

Para falsificar o 'ou' a expressão A e B ambas devem ser falsas. Para A ser falso temos: Carlos não é dentista. Para B ser falso temos: Enio é economista e Juca não é arquiteto. Letra B.

4. (ALESP 2010/FCC) Paloma fez as seguintes declarações:

– "Sou inteligente e não trabalho."

– "Se não tiro férias, então trabalho."

Supondo que as duas declarações sejam verdadeiras, é FALSO concluir que Paloma

(A) é inteligente.

(B) tira férias.

(C) trabalha.

(D) não trabalha e tira férias.

(E) trabalha ou é inteligente.

Resolução

Para uma conjunção ser verdadeiras, ambas as proposições que compõem devem ser verdadeiras, logo, é inteligente e não trabalho são verdades. Portanto, letra C é a expressão falsa.

5. (Petrobras/2007/Cespe) Julgue o item que se segue.

Considere as proposições abaixo:

p: 4 é um número par;

q: A Petrobras é a maior exportadora de café do Brasil.

Nesse caso, é possível concluir que a proposição $p \vee q$ é verdadeira

Resolução

Uma vez que a disjunção é verdadeira caso uma das proposições for verdadeira, e a proposição p é verdadeira, logo está correto a afirmação.

6. (INSS 2008/CESPE-UnB) Proposições são sentenças que podem ser julgadas como verdadeiras — V — ou falsas — F —, mas não como ambas. Se P e Q são proposições, então a proposição "Se P então Q", denotada por $P \rightarrow Q$, terá valor lógico F quando P for V e Q for F, e, nos demais casos, será V. Uma expressão da forma $\neg P$, a negação da proposição P, terá valores lógicos contrários aos de P. $P \rightarrow Q$, lida como "P ou Q", terá valor lógico F quando P e Q forem, ambas, F; nos demais casos, será V.

Considere as proposições simples e compostas apresentadas abaixo, denotadas por A, B e C, que podem ou não estar de acordo com o artigo 5.º da Constituição Federal.

A: A prática do racismo é crime afiançável.

B: A defesa do consumidor deve ser promovida pelo Estado.

C: Todo cidadão estrangeiro que cometer crime político em território brasileiro será extraditado.

De acordo com as valorações V ou F atribuídas corretamente às proposições A, B e C, a partir da Constituição Federal, julgue os itens a seguir.

a) Para a simbolização apresentada acima e seus correspondentes valores lógicos, a proposição $B \rightarrow C$ é V.

b) De acordo com a notação apresentada acima, é correto afirmar que a proposição $(\neg A) \vee (\neg C)$ tem valor lógico F.

Resolução

artigo 5º da Constituição Federal.

XXXII – o Estado promoverá, na forma da lei, a defesa do consumidor;

XLII – a prática do racismo constitui crime inafiançável e imprescritível, sujeito à pena de reclusão, nos termos da lei;

LII – não será concedida extradição de estrangeiro por crime político ou de opinião.

Logo temos: V(A) = F, V(B) = V, V(C) = F, então $V(B \rightarrow C) = F$ e $V((\neg A) \vee (\neg C)) = V$.

7. (SADPE/2008/FGV) Considere as situações abaixo:

I. Em uma estrada com duas pistas, vê-se a placa:

Caminhões → Pista da Direita

Como você está dirigindo um automóvel, você conclui que deve trafegar pela pista da esquerda.

II. Você mora no Recife e telefona para sua mãe em Brasília. Entre outras coisas, você diz que "Se domingo próximo fizer sol, eu irei à praia". No final do domingo, sua mãe viu pela televisão que choveu no Recife todo o dia. Então, ela concluiu que você não foi à praia.

III. Imagine o seguinte diálogo entre dois políticos que discutem calorosamente certo assunto:

- A: Aqui na Câmara tá cheio de ladrão.

- B: Ocorre que eu não sou ladrão.

- A: Você é safado, tá me chamando de ladrão.

Em cada situação há, no final, uma conclusão. Examinando a lógica na argumentação:

a) são verdadeiras as conclusões das situações I e II, apenas.

b) são verdadeiras as conclusões das situações II e III, apenas.

c) são verdadeiras as conclusões das situações I e III, apenas.

d) as três conclusões são verdadeiras.

e) as três conclusões são falsas.

Resolução

Para falsificar uma implicação o antecedente deve ser verdadeiro e o consequente falso. Quando temos um antecedente falso, não podemos afirmar nada do consequente, pois a expressão será verdadeira independente do seu valor. Na sentença I e II ambos os antecedentes são falsos, logo, no primeiro caso, ele pode ir para esquerda ou direita, no segundo caso, ir ou não para a praia, que a expressão será verdadeira. Ou seja, não se pode fazer nenhuma conclusão. No item III nenhum político chamou o outro de ladrão. Alternativa E correta.

17. (TRT-9ª Região/2004/FCC) Considere a seguinte proposição "Na eleição para a prefeitura, o candidato A será eleito ou não será eleito". Do ponto de vista lógico, a afirmação da proposição caracteriza

a) um silogismo.

b) uma tautologia.

c) uma equivalência.

d) uma contingência.

e) uma contradição.

Resolução

P: ser eleito. P V ¬P a Tabela Verdade dá uma tautologia.

18. (Fiscal do Trabalho 1998/Esaf) Chama-se tautologia a toda proposição que é sempre verdadeira, independentemente da verdade dos termos que a compõem. Um exemplo de tautologia é:

a) se João é alto, então João é alto ou Guilherme é gordo.

b) se João é alto, então João é alto e Guilherme é gordo.

c) se João é alto ou Guilherme é gordo, então Guilherme é gordo.

d) se João é alto ou Guilherme é gordo, então João é alto e Guilherme é gordo.

e) se João é alto ou não é alto, então Guilherme é gordo.

Resolução

a) $p \rightarrow (p \vee q)$, b) $p \rightarrow (p \wedge q)$, c) $(p \vee q) \rightarrow q$, d) $(p \vee q) \rightarrow (p \wedge q)$, e) $(p \vee \neg p) \rightarrow q$, sendo a opção a única como tautológica

19. Se A for considerada uma proposição F e B for considerada uma proposição V, então a proposição ¬B v A é F.

Resolução

A	b	¬b	¬b v a
V	V	F	V
V	F	V	V
F	V	F	F
F	F	V	V

Referências

BEDREGAL, B. R. **Introdução à Lógica Clássica para a Ciência da Computação**. 2007. Disponível em: <https://www.dimap.ufrn.br/~jmarcos/books/BA_Jul07.pdf>. Acesso em: 05 fev. 2015.

FAJARDO, R. A. **Introdução à Lógica.** [20--?]. Disponível em: <http://www.ime.usp.br/~fajardo/Logica.pdf>. Acesso em: 05 fev. 2015.

NOLT, J.; ROHATYN, D. **Lógica**. São Paulo: McGraw-Hill, 1991.

Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 04

Implicação e Equivalência Lógica, Técnica da Redução por Absurdos

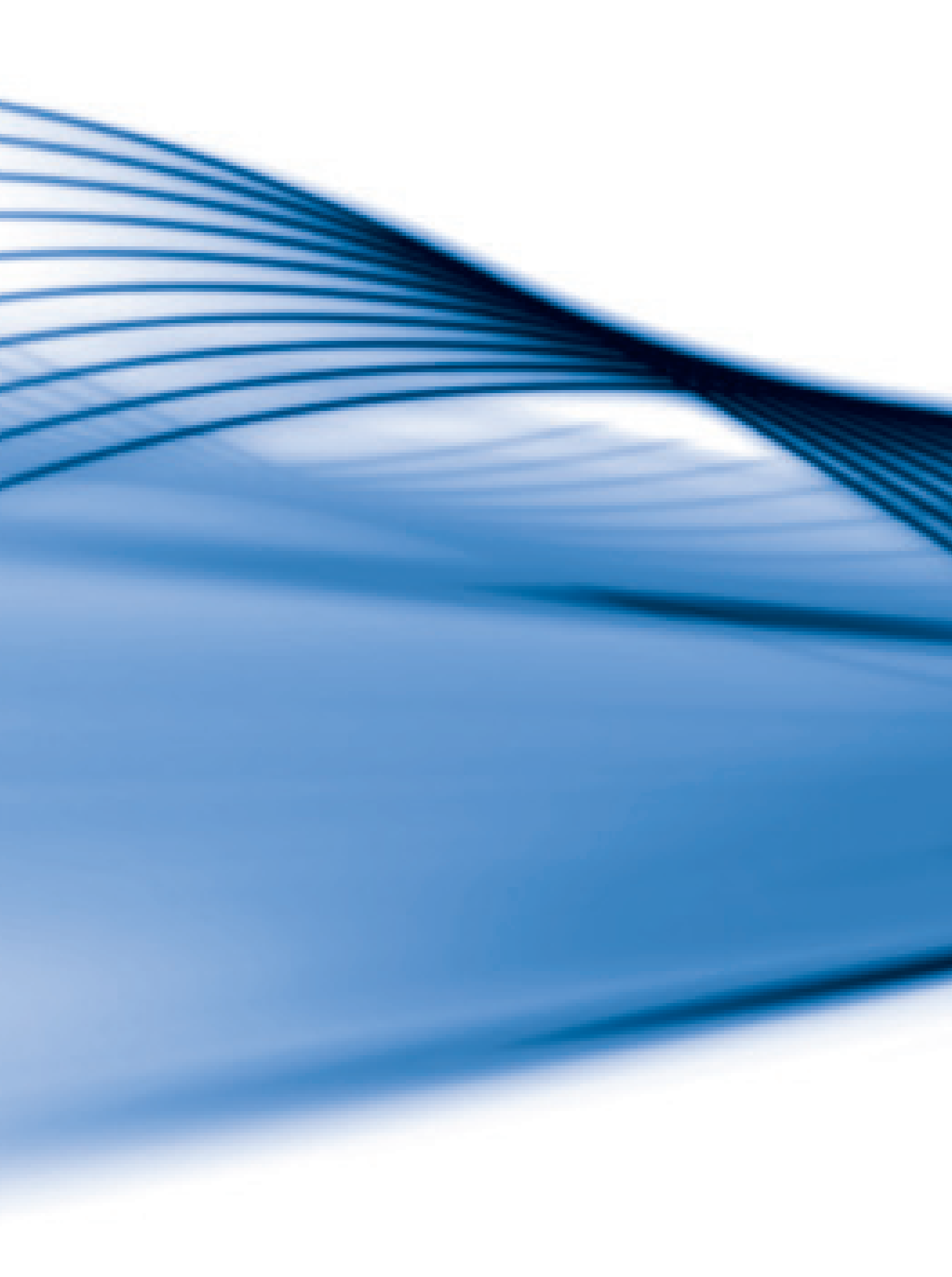
Thiago Medeiros Barros

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno! Estamos chegando ao final do nosso curso sobre Fundamentos de Lógica e espero você que esteja aprendendo bastante. Nesta última aula, vamos compreender como utilizamos a Lógica Clássica para concluir argumentos verdadeiros e verificar se uma expressão é uma falácia ou é de fato uma conclusão lógicamente correta, a partir dos conceitos de Implicação e Equivalência Lógica. Além disso, aprenderemos uma importante técnica chamada Redução por Absurdo, bastante utilizada em toda Lógica Clássica.

Bons estudos!



Aula 4 - Argumento Lógico, Proposições Simples, Princípios Lógicos

Objetivos

Compreender o conceito de Implicação e Equivalência Lógica;

Verificar se uma dada implicação ou equivalência é verdadeira;

Compreender e aplicar a técnica Redução por Absurdo.

Desenvolvendo o conteúdo

Figura 1: Será que o pensamento do Dinossauro está correto?

Olá, Aluno! Nas aulas anteriores aprendemos ferramentas para sistematizar a valoração da Lógica Proposicional através das tabelas verdades e transformar a Linguagem Natural em Lógica Proposicional. Nesta aula, vamos focar em como podemos utilizar esses conhecimentos e extrair conclusões Lógicas Válidas.

Para isso, vamos começar com a definição de Implicação Lógica.

Definição

A proposição A implica logicamente na proposição B, se B é verdadeiro todas as vezes que A é verdadeiro. A implicação lógica pode ser representada como $A \Longrightarrow B$.

Exemplo:

- $p \wedge q$, $p \vee q$ e $p \leftrightarrow q$

p	q	p∧q	p∨q	p↔q
1	1	1	1	1
1	0	0	1	0
0	1	0	1	0
0	0	0	0	1

Fonte: autoria própria.

A proposição $p \wedge q$ é verdadeira apenas na primeira linha. Nessa mesma linha, as proposições $p \vee q$ e $p \leftrightarrow q$ também são verdadeiras. Logo,

$p \wedge q \Longrightarrow p \vee q$ e $p \wedge q \Longrightarrow p \leftrightarrow q$

Uma forma mais simples de verificar se uma proposição implica logicamente em outra é dada pela definição: proposição A implica logicamente na proposição B se e somente se **$A \rightarrow B$ for uma tautologia**.

Exemplo:

- $p \rightarrow p \wedge q \Leftrightarrow p \rightarrow q$

Tabela 1: Tabela Verdade da expressão: $p \wedge q \Longrightarrow p \vee q$

p	q	p∧q	p∨q	p∧q→p∨q
V	V	V	V	V
V	F	F	V	V
F	V	F	V	V
F	F	F	F	V

Fonte: autoria própria.

Uma vez que a expressão $p \wedge q \rightarrow p \vee q$ é uma tautologia, logo podemos afirmar que $p \wedge q \Rightarrow p \vee q$. Ou seja, $p \wedge q$ implica logicamente em $p \vee q$.

O **indicador em português** da utilização da implicação lógica é **"concluir que"**,

Exemplo:

- "Se eu correr então vou ficar com sede, e hoje eu corri, **posso concluir** corretamente que estou com sede".

  » p: correr

  » q: ficar com sede

  » $(p \rightarrow q) \wedge p \Rightarrow q$

Exemplo: (SEFAZ-MG 2005/ESAF) O reino está sendo atormentado por um terrível dragão. O mago diz ao rei: "O dragão desaparecerá amanhã se e somente se Aladim beijou a princesa ontem". O rei, tentando compreender melhor as palavras do mago, faz as seguintes perguntas ao lógico da corte:

1. Se a afirmação do mago é falsa e se o dragão desaparecer amanhã, posso concluir corretamente que Aladim beijou a princesa ontem?

2. Se a afirmação do mago é verdadeira e se o dragão desaparecer amanhã, posso concluir corretamente que Aladim beijou a princesa ontem?

3. Se a afirmação do mago é falsa e se Aladim não beijou a princesa ontem, posso concluir corretamente que o dragão desaparecerá amanhã?

O lógico da corte, então, diz acertadamente que as respostas logicamente corretas para as três perguntas são, respectivamente,

a. Não, sim, não.

b. Não, não, sim.

c. Sim, sim, sim.

d. Não, sim, sim.

e. Sim, não, sim.

Resolução

Sendo d:dragão desaparecerá, a: Aladim beijou a princesa, temos: d↔a.

1) ¬(d↔a) ^d => a, não é tautologia

2) (d↔a)^d => a, é tautologia

3)¬(d↔a)^¬a => d, é tautologia

Então temos não, sim, sim.

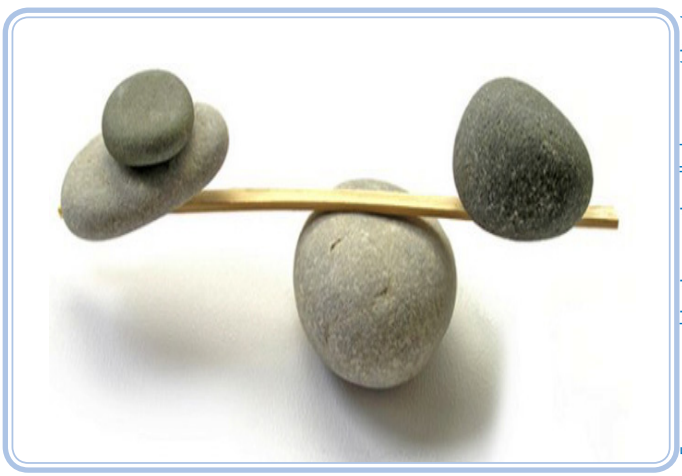
Atividade de aprendizagem 1

Verifique se as implicações lógicas abaixos estão corretas.

$(H \vee S) \wedge \neg H \Rightarrow S$

$(I \rightarrow C) \wedge \neg I \rightarrow D \Rightarrow C \vee D$

$(P \rightarrow L) \wedge (L \rightarrow N) \wedge N \Rightarrow P$



Fonte: www.cultivatingemotionalbalance.org/sites/default/files/rocks_on_balance_0.jpg

Fig. 02: A equivalência lógica pode ser interpretada como a igualdade matemática

A Equivalência Lógica pode ser comparada com a igualdade aritmética. Para entendê-la melhor, vamos à definição:

Definição

A proposição A é logicamente equivalente a proposição B, se as tabelas verdades dessas duas proposições forem identicas. A implicação lógica pode ser representado como $A \Leftrightarrow B$.

Exemplo:

Tabela 2: Tabela Verdade da expressão $p \rightarrow p \wedge q$, $p \rightarrow q$

P	Q	$p \wedge q$	$p \rightarrow p \wedge q$	$p \rightarrow q$
1	1	1	1	1
1	0	0	0	0
0	1	0	1	1
0	0	0	1	1

Fonte: autoria própria.

Uma forma mais simples de verificar se uma proposição é equivalente logicamente em outra é dada pela definição: A é equivalente logicamente à proposição B se e somente se $A \leftrightarrow B$ for uma tautologia.

Exemplo:

$$p \rightarrow p \wedge q \Leftrightarrow p \rightarrow q$$

Tabela 3: Verdade da expressão $p \rightarrow p \wedge q \Leftrightarrow p \rightarrow q$

P	Q	$p \rightarrow p \wedge q$	$p \rightarrow q$	$p \rightarrow p \wedge q \leftrightarrow p \rightarrow q$
V	V	V	V	V
V	F	F	F	V
F	V	V	V	V
F	F	V	V	V

Fonte: autoria própria.

Uma vez que a expressão $p \rightarrow p \wedge q \leftrightarrow p \rightarrow q$ é uma tautologia, logo podemos afirmar que $p \rightarrow p \wedge q \Leftrightarrow p \rightarrow q$. Ou seja, $p \rightarrow p \wedge q$ é equivalente a $p \rightarrow q$.

Fonte: VALBER (2015)

Fig. 03: Tirinha sobre verdade absoluta

Algumas equivalência famosas são:

- Leis Distributivas:

$$a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$$

$$a \wedge (b \vee c) \Leftrightarrow (a \wedge b) \vee (a \wedge c)$$

- Leis da Negação:

$$\neg(\neg a) \Leftrightarrow a$$

$$\neg(a \rightarrow b) \Leftrightarrow a \wedge \neg b$$

$$\neg(a \leftrightarrow b) \Leftrightarrow (a \wedge \neg b) \vee (\neg a \wedge b)$$

- Leis de Morgan:

$$a \vee b \Leftrightarrow b \vee a$$

$$a \wedge b \Leftrightarrow b \wedge a$$

$$a \vee (b \vee c) \Leftrightarrow (a \vee b) \vee c$$

$$a \wedge (b \wedge c) \Leftrightarrow (a \wedge b) \wedge c$$

$$\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$$

$$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$$

$$(a \vee b) \wedge b \Leftrightarrow b$$

$(a \wedge b) \vee b \Leftrightarrow b$

- Lei do Terceiro Excluído:

$a \vee \neg a$

- Lei da Contradição:

$\neg(a \wedge \neg a)$

- Lei da Contraposição:

$a \rightarrow b \Leftrightarrow \neg b \rightarrow \neg a$

- Lei da Exportação:

$(a \wedge b) \rightarrow c \Leftrightarrow a \rightarrow (b \rightarrow c)$

- Outras

$p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p \Leftrightarrow \neg p \vee q \Leftrightarrow \neg(p \wedge \neg q)$

$(p \leftrightarrow q) \Leftrightarrow [(p \rightarrow q) \wedge (q \rightarrow p)]$

$p \vee q \Leftrightarrow \neg p \rightarrow q$

Exemplos

(SGA/AC 2007/CESPE-UnB) As proposições $A \rightarrow B$ e $(\neg B) \rightarrow (\neg A)$ têm a mesma tabela verdade.

Resolução

Ao gerar as duas tabelas verdades é constatado que ambas são iguais, logo, são equivalentes.

(Agente Penitenciário SJDH-BA 2010/FCC) Uma afirmação equivalente à afirmação "Se bebo, então não dirijo" é

(A) Se não bebo, então não dirijo.

(B) Se não dirijo, então não bebo.

(C) Se não dirijo, então bebo.

(D) Se não bebo, então dirijo.

(E) Se dirijo, então não bebo.

Resolução

b: bebo, d:dirijo. Expressão b -> ¬d, a qual é equivalente: ¬b v ¬d e d -> ¬b (letra e) ao se fazer a tabela verdade.

Atividade de aprendizagem 2

Verifique as equivalências lógicas abaixo.

$$a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$$

$$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$$

$$\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$$

Redução por Absurdo

Segundo Medeiros (1995), a prova por redução por absurdo tem como ideia básica que uma proposição não pode ser verdadeira se dela deduzirmos uma contradição: em outras palavras, demonstrar por redução ao absurdo o argumento é válido **se e somente se** a conjunção das premissas com a negação da conclusão é uma expressão contraditória, sendo a contradição uma expressão que afirma e nega algo ao mesmo tempo.

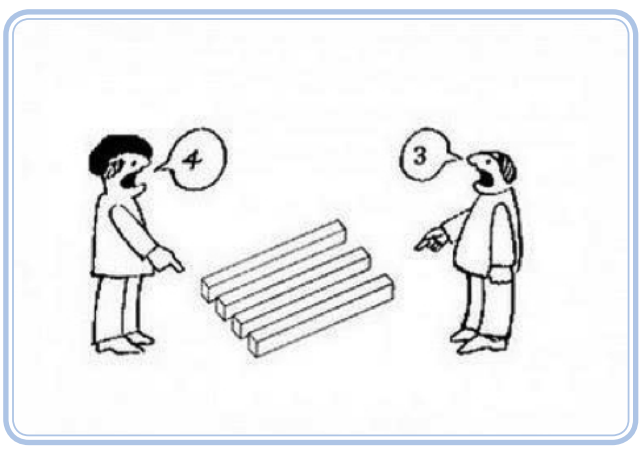
Em outras palavras, para verificar se uma **conclusão é verdadeira**, tentamos **provar que a mesma é falsa** através da redução por absurdo, a partir do seguinte pensamento: se ao tentarmos provar que a conclusão é falsa (ou seja, assumindo as premissas verdadeiras em conjunção com a negação da conclusão) **encontrarmos um absurdo**, falhamos em falsificar a expressão. Portanto, a **expressão é verdadeira**.

Exemplo: Se chover, então vou ficar molhado. Choveu, logo, fiquei molhado.

p: Chover

q: Ficar Molhado

$(p \rightarrow q) \ \Lambda \ p \Rightarrow p$



Fonte: http://www.barrett.com.au/blogs/SalesBlog/wp-content/uploads/2013/10/point-of-view.jpg

Fig. 04: Imagem ilustrando uma contradição ótica.

De acordo com a redução ao absurdo, se a **conjunção das premissas com a negação da conclusão possui alguma proposição que afirma e nega algo ao mesmo tempo temos um absurdo**, logo a expressão É VERDADEIRA.

Voltando ao nosso exemplo, temos:

- Premissas: $(p \rightarrow q) \ \Lambda \ p$
- Conclusão: p
- $(p \rightarrow q) \ \Lambda \ p \ \Lambda \ \neg p$

Como podemos notar, no passo 3 temos na mesma fórmula p e ¬p. Portanto, há uma proposição que afirma e nega ao mesmo tempo, logo, **encontramos um absurdo**, concluindo então que a **expressão é verdadeira**, pois **tentamos falsificar e não conseguimos**.

Podemos ter um olhar sobre a redução por absurdo utilizando a função de valoração. Tomemos o mesmo exemplo $(p \rightarrow q) \ \Lambda \ p \Rightarrow p$, o qual queremos comprovar se é uma conclusão logicamente válida. Pela redução por absurdo, como visto acima, temos:

1. $(p \rightarrow q) \ \Lambda \ p \ \Lambda \ \neg p$

E queremos comprovar se a expressão é válida, ou seja, utilizando a função de valoração temos:

1. $V((p \rightarrow q) \ \Lambda \ p \ \Lambda \ \neg p)=1$

Como visto na aula 2 do nosso curso, temos que para a conjunção ser verdadeira, todas as subfórmulas são verdadeiras:

2. $V(p \rightarrow q)=1$

3. $V(p)=1$

4. $V(\neg p)=1$

Com o passo 3 e 4 chegamos claramente em um absurdo, logo, não conseguimos falsificar a expressão. Portanto, a expressão é verdadeira.

Nem sempre o absurdo acontece entre as premissas. Há casos em que há a necessidade de um maior desdobramento na fórmula para encontrarmos o absurdo. Veja o exemplo abaixo.

Exemplo: Se chover então vai ficar nublado hoje. Se ficar nublado hoje não vou à praia. Portanto, se chover hoje eu não vou à praia.

Formalizando temos:

c: chover hoje

n: nublado hoje

p: ir a praia hoje

premissas: $(c \rightarrow n) \ \Lambda \ (n \rightarrow p)$

conclusão: $c \rightarrow p$

Pela redução ao absurdo temos:

$V((c \rightarrow n) \ \Lambda \ (n \rightarrow p) \ \Lambda \ \neg(c \rightarrow p))=1$

Pela regra da conjunção temos:

$V(c \rightarrow n)=1$

$V(n \rightarrow p)=1$

$V(\neg(c \rightarrow p))=1$

A partir de 4 e pela regra da negação, temos:

$V(c \rightarrow p)=0$

Para falsificar uma implicação (5), temos:

V(c)=1

V(p)=0

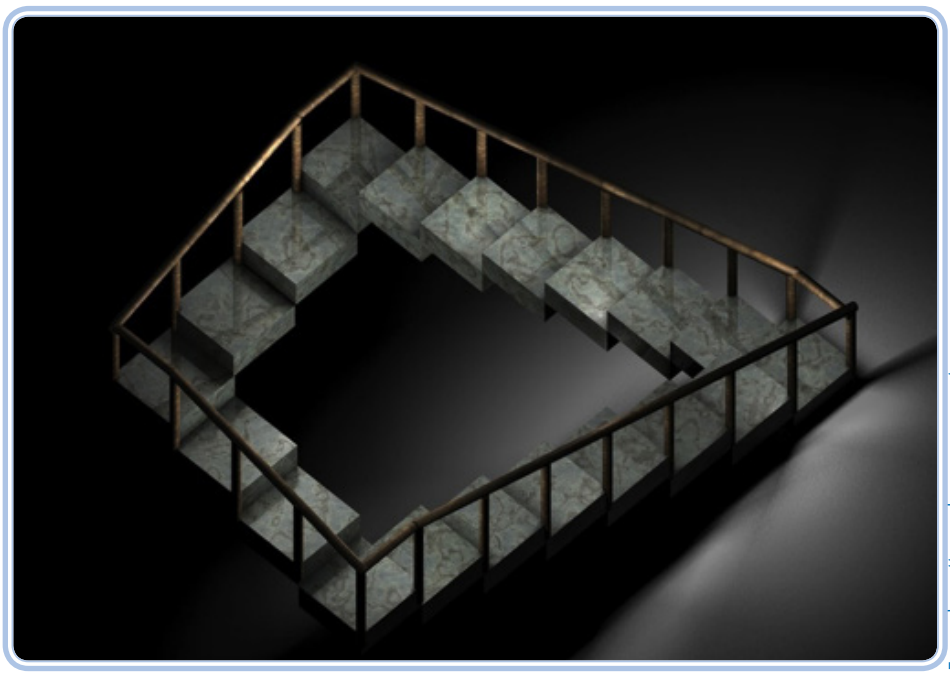
Do passo (7) e do passo (3) afirma-se que o valor de p é falso e a implicação n→p é verdadeira, para isso acontecer, necessariamente pela tabela verdade da implicação, o antecessor tem que ser falso, logo:

V(n)=0

Do passo (8) e do passo (2) e utilizando a Tabela Verdade da implicação, temos:

V(c)=0

Do passo (6) e do passo (9) encontramos um absurdo, ou seja, tentamos falsificar a expressão, não conseguimos, logo, a afirmação é verdadeira. Percebam que para encontrar o absurdo houve a necessidade de um maior desdobramento. Entretanto, não foi necessário a construção de toda a tabela verdade ($2^3 = 8$ linhas). Essa é a grande vantagem do uso da técnica de redução por absurdo.



Fonte: http://www.barrett.com.au/blogs/SalesBlog/wp-content/uploads/2013/10/point-of-view.jpg

Fig. 05: Imagem ilustrando uma contradição ótica

E, caso não encontrarmos o absurdo, o que deve ser feito? Nesse caso, nós encontramos o **contraexemplo**. Contraexemplo é um **conjunto de valorações para as proposições** que consegue **falsificar a expressão**, ou seja, é uma linha na Tabela Verdade a qual a conjunção das premissas com a negão da conclusão é verdadeiro. Quando encontramos o contraexemplo significa que conseguimos **falsificar a expressão**, logo, **a conclusão é falsa**. Veja o exemplo abaixo:

Se o sol aparecer então eu vou à praia. Eu fui à praia. Portanto, o sol apareceu.

s: sol aparecer

p: ir à praia

Premissas: $(s \rightarrow p) \ \Lambda \ p$

Conclusão: s

Utilizando a técnica de redução ao absurdo, temos:

1. $V((s \rightarrow p) \ \Lambda \ p \ \Lambda \ \neg s \ )=1$

Pela regra da conjunção, temos:

2. $V(s \rightarrow p)=1$

3. $V(p)=1$

4. $V(\neg s)=1$

Do passo (4) e pela regra da negação, temos:

5. $V(s)=0$

Percebam que se substituirmos os valores encontrados nos átomos (passo 5 e passo 3) em todas as fórmulas (por exemplo na implicação do passo 2) **não** encontramos **nenhum absurdo**. Portanto, **encontramos um contra-exemplo**, ou seja, um conjunto de valores para os átomos da expressão, o qual consegue falsificá-la. Uma vez que **conseguimos falsificar** a expressão, então a mesma **não é verdadeira**.

Nosso contraexemplo para o problema acima então é: V(s)=0 e V(p)=1

Algumas dicas para uso da redução ao absurdo:

- Inicie realizando as valorações dos átomos que estiverem soltos na fórmula e suas negações.

Exemplo: V($(p \rightarrow q) \ \Lambda \ p \ \Lambda \ \neg p$)=1, então valorar as proposições mais simples temos:

V(p)=1

V(¬p)=1

- Dê preferência em desenvolver as fórmulas que apresentam apenas uma única linha na Tabela Verdade.

Exemplo: V($(c \rightarrow n) \ \Lambda \ (n \rightarrow p) \ \Lambda \ \neg(c \rightarrow p)$)=1 , temos as seguintes fórmulas para analisar:

V(c→n)=1

V(n→p)=1

V(¬(c→p))=1

A fórmula acima, devido a negação, pode ser representada como:
V(c→p)=

Resgatando a Tabela Verdade da implicação, temos:

Tabela 4 – Tabela Verdade

p	q	p→q
1	1	1
1	0	0
0	1	1
0	0	1

Fonte: autoria própria.

Dessa forma, há três resultados possíveis para implicação ser verdadeira e apenas um único resultado para falsificar a implicação. Portanto, deve-se desenvolver o resultado a partir da falsificação da implicação. Como visto na aula 2, os outros casos em que há apenas uma única linha na tabela verdade são:

- Falsificar a disjunção (ou);
- A Conjunção (e) ser verdadeira.

RESUMINDO

Na nossa última aula do curso sobre Fundamentos de Lógica aprendemos sobre os conceitos de Implicação e Equivalência Lógica e como podemos utilizá-los para realizar interpretações de textos na linguagem natural. Também aprendemos uma importante técnica chamada Redução por Absurdo, a qual evita utilizar tabelas verdades a fim de verificar se uma conclusão lógica é válida ou não.

LEITURAS COMPLEMENTARES

Está empolgado sobre como verificar se conclusões são logicamente válidas ou não? Um texto bastante interessante diz respeito às histórias dos cavaleiros e cavilosos, o qual conta várias historinhas lógicas que vai enrolar sua cabeça. Entretanto, utilizando as técnicas que aprendemos nesta aula, você conseguirá resolver e brincar com seus amigos. O livro foi escrito por Raymond Smullyan e traduzido pelo prof. João Marcos de Almeida.

SMULLYAN, R. **A lógica de contar mentiras e verdades.**[200-?]. (Tradução não oficial de João Marcos de Almeida). Disponível em: <https://www.dimap.ufrn.br/~jmarcos/courses/LAaC/Trad-LCP/Smullyan_Cap3-7.pdf>. Acesso em: 13 abr. 2015.

AVALIANDO SEUS CONHECIMENTOS

Através da técnica de redução por absurdo, verifique se as conclusões estão logicamente corretas.

a) Se estudar bastante (E) e fizer todos os exercícios (F) então sou um bom aluno (B), se eu for um bom aluno então vou ter um resultado bom na prova (P). Eu estudei bastante e fiz todos os exercícios. Portanto, eu vou ter um bom resultado na prova.

b) Os meus deveres diários são estudar (E) e ou eu brinco com meus brinquedos (B) ou eu brinco com meus amigos (A). Portanto, todo dia ou estudo e brinco com meus brinquedos ou eu estudo e brinco com meus amigos.

c) Há três suspeitos de um crime: o cozinheiro, a governanta e o mordomo. Sabe-se que o crime foi efetivamente cometido por um ou por mais de um deles, já que podem ter agido individualmente ou não. Sabe-se, ainda que:

(i) se o cozinheiro é inocente, então a governanta é culpada;

(ii) ou o mordomo é culpado ou a governanta é culpada, mas não os dois;

(iii) o mordomo não é inocente.

Portanto, é possível concluir que o cozinheiro é culpado?

d) Se dermos arsênico a Rasputin ele ficará gravemente doente ou morrerá. Se ele ficar gravemente doente, não poderá influenciar o Czar; e se ele morrer, obviamente também não poderá influenciar o Czar. Logo, se lhe dermos arsênico, ele não poderá influenciar o Czar.

EXERCÍCIOS COMPLEMENTARES

1. (EBSERH|IBFC, 2013) Se o valor lógico de uma proposição p é verdadeiro e o valor lógico de uma preposição q é falso, então o valor lógico da proposição composta $[(p \rightarrow q) \vee \neg p] \wedge \neg q$ é

a) falso e Verdadeiro.

b) verdadeiro.

c) falso.

d) inconclusivo.

2. (EBSERH|IBFC, 2013) Seja uma proposição p: Maria é estagiária e a proposição q: Marcos é estudante. A negação da frase "Maria é estagiária ou Marcos é estudante" é equivalente a

a) Maria não é estagiária ou Marcos não é estudante.

b) Se Maria não é estagiária, então Marcos não é estudante.

c) Maria não é estagiária, se e somente se, Marcos não é estudante.

d) Maria não é estagiária e Marcos não é estudante.

3. (EBSERH|IBFC, 2013) Sejam as afirmações:

a) Se o valor lógico de uma proposição p é falso e o valor lógico de uma proposição q é verdadeiro, então o valor lógico da conjunção entre p e q é verdadeiro.

b) Se todo X é Y, então todo Y é X.

c) Se uma proposição p implica numa proposição q, então a proposição q implica na proposição p.

- Pode se afirmar que são verdadeiras:

a) Todas.

b) Somente duas delas.

c) Somente uma delas.

d) Nenhuma.

4. Se houver fraude no concurso da prefeitura (F) ou se os bugueiros deixarem de operar na cidade (B), então o turismo vai diminuir (D) e a cidade vai sofrer (S). Se o turismo diminuir, então a polícia ficará mais contente (P). A polícia nunca está contente. Portanto, haverá fraude no concurso da prefeitura.

5. Em um júri popular, o advogado de defesa do Sr. X argumenta o seguinte: Se meu cliente fosse culpado, a faca estaria na gaveta. Ou a faca não estava na gaveta ou Rodrigo viu a faca. Se a faca não estava lá no dia 10 de outubro, então Rodrigo não viu a faca. Além disso, se a faca estava lá no dia 10 de outubro, então a faca estava na gaveta e o martelo estava no celeiro. Mas todos sabemos que o martelo não estava no celeiro. Portanto, senhoras e senhores, meu cliente é inocente. Pergunta-se: Sr. X é inocente?

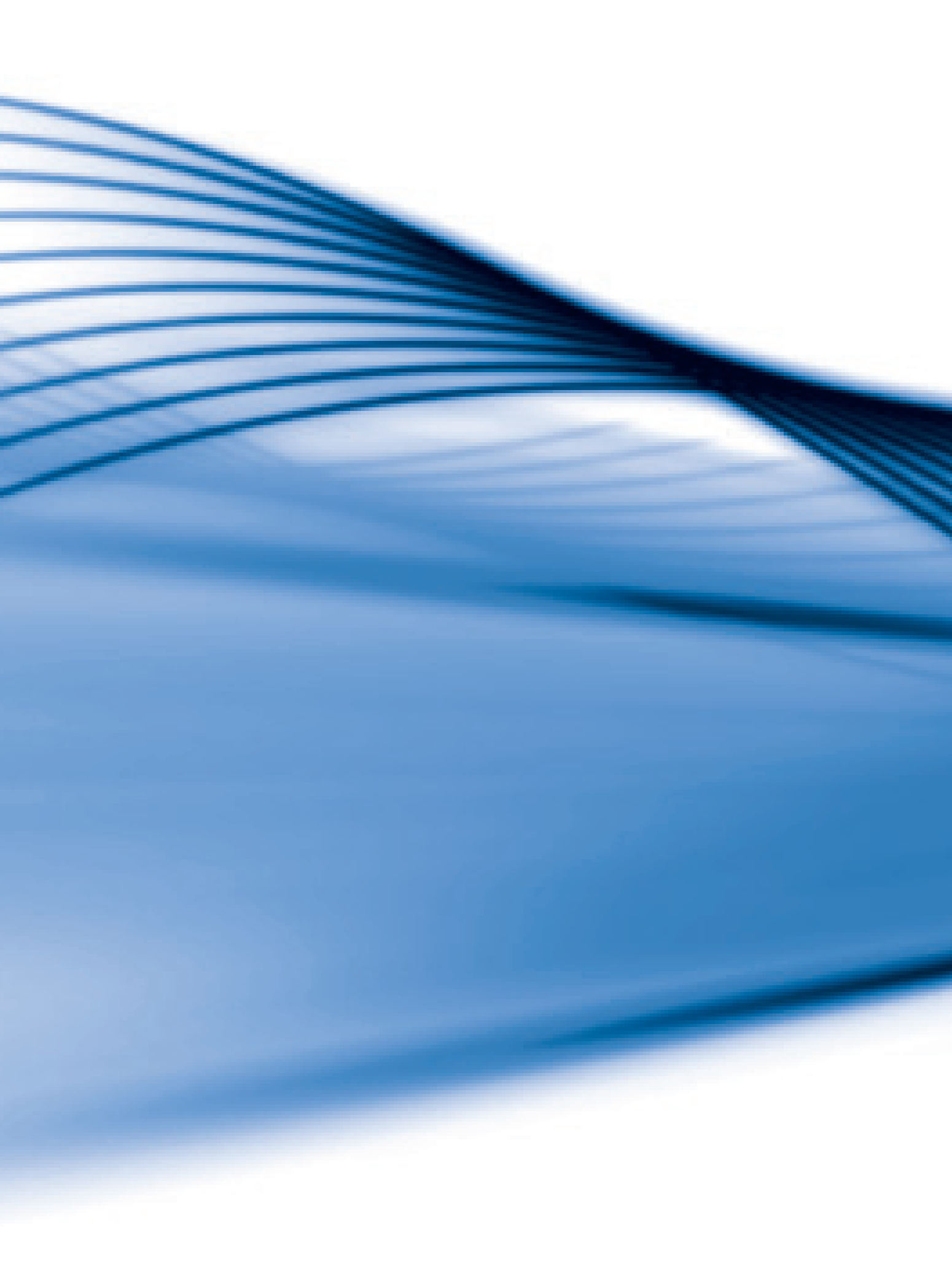
CONHECENDO AS REFERÊNCIAS

BEDREGAL, B. R. **Introdução à Lógica Clássica para a Ciência da Computação**. 2007. Disponível em: <https://www.dimap.ufrn.br/~jmarcos/books/BA_Jul07.pdf>. Acesso em: 05 fev. 2015.

FAJARDO, R. A. **Introdução à Lógica**. [20--?]. Disponível em: <http://www.ime.usp.br/~fajardo/Logica.pdf>. Acesso em: 05 fev. 2015.

MEDEIROS, M. P. N. Aprova por redução ao absurdo na lógica clássica. **Princípios**, Natal, v. 2, n. 01, p. 120-125, jun. 1995. Disponível em: <http://www.periodicos.ufrn.br/principios/article/view/740/682>. Acesso em: 13 abr. 2015.

NOLT, J.; ROHATYN, D. **Lógica**. São Paulo: McGrall-Hall, 1991.



Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 05
Conceitos Fundamentais de Algoritmos e
Introdução à Programação em Python

Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Prezado(a) aluno(a), na Unidade I você foi apresentado(a) aos fundamentos da lógica matemática. Lá, você aprendeu uma nova forma de organizar seu pensamento em um mundo de proposições, conectivos lógicos e regras bem estabelecidas. A partir deste momento, convido vocês a continuar nessa jornada, agora aplicando o raciocínio lógico para construir seus próprios programas de computador.

Imagine o quanto este aprendizado pode ser útil, não apenas no decorrer do seu curso, mas para a sua vida. Você já deve ter percebido o quanto os computadores estão presentes no nosso dia a dia, não é mesmo? Por exemplo, ao pagar uma conta em um restaurante, ao fazer pesquisas para um trabalho escolar em um *site* de busca e ao acessar nossas redes sociais e aplicativos favoritos usando o telefone celular.

Fonte: https://goo.gl/kOOYjl

Figura 1: Tirinha "O Grande Manual das Pequenas Ilusões"

Pois bem, caro(a) aluno(a). Em todo o nosso contato diário com os computadores, nós estamos na verdade lidando diretamente com programas, que também podem ser chamados de aplicativos ou software. Sendo assim, ao começar a compreender este universo a partir deste curso, você pode deixar de ser apenas um usuário leigo para, no mínimo, entender como funcionam os programas de computador. De fato, esperamos que este seja o pontapé inicial para que você possa desenvolver seus próprios aplicativos ou páginas de Internet. Tudo pronto, vamos começar?

Aula 5 - Conceitos Fundamentais de Algoritmos e Introdução à Programação em Python

Objetivos

Nesta aula são apresentados os conceitos básicos de algoritmos e de uma linguagem de programação. Ao término da aula, você deve criar e executar pequenos programas. Dessa forma, os objetivos dessa aula são:

Compreender as noções básicas de programação através de exemplos práticos do cotidiano;

Entender o conceito de algoritmos e a sua relação com programas de computador;

Criar e executar pequenos programas na linguagem de programação *Python*;

Aprender sobre o conceito de memória e como armazenar e modificar informações em um programa.

Desenvolvendo o conteúdo

Algoritmos e Programação de computadores

É importante destacar que construir um programa de computador real é uma tarefa que pode ser trabalhosa, a depender do fim a que se destina e da sua complexidade. Mas, assim como se constrói uma casa tijolo por tijolo, você vai aprender neste curso, de forma progressiva, as técnicas e ferramentas básicas para iniciar na criação de *software*. Com dedicação e o estudo de disciplinas complementares, você poderá em pouco tempo se tornar um programador.

Tipicamente, um programa é composto por centenas ou até milhares linhas de texto contendo a descrição passo-a-passo do problema que ele se propõe a resolver. Pois bem, a palavra "problema" aqui se aplica a uma necessidade de alguém que precisa ser atendida por meio de um programa de computador. Essa pessoa normalmente é um cliente que paga a uma outra pessoa (programador) ou empresa para adquirir o benefício que a resolução do problema irá trazer.

A essa altura, é possível que você pergunte "como eu posso escrever essas linhas de texto que formam um programa de computador?". Muito bem, caro(a) aluno(a), você deve escrever seus programas usando uma *linguagem de programação*. Essa linguagem deve ser de fácil compreensão pelo programador, mas deve ser estruturada de modo que possa ser traduzida em instruções que são entendidas e executadas pelo computador.

Assim sendo, você não deve escrever um texto em língua portuguesa e sim um texto que segue um conjunto bem definido de regras e instruções de uma determinada *linguagem de programação.* Observe que a escrita deve seguir esse padrão rigoroso, pois o computador só é capaz de entender sentenças bem formatadas, seguindo as regras da linguagem. Ele não tem a habilidade do ser humano de compreender e interpretar sentenças com ambiguidades ou erros de escrita.

Nesse contexto, é possível pensar em um programa de computador como sendo um conjunto de instruções descritas usando uma linguagem de programação. Não cabe aqui discorrer sobre os diversos tipos de linguagens. Na seção de leituras complementares, sugerimos referências para quem quiser se aprofundar no assunto. O importante aqui é entender que existem centenas delas disponíveis, cada uma criada pensando em resolver de forma adequada um determinado conjunto de problemas.

Nesse sentido, é possível dizer que existem linguagens que são melhores para a criação de *software* para computadores pessoais (*desktop*), outras para dispositivos móveis (celulares, tablets, etc.), e aquelas mais voltadas aos sistemas para Internet. No entanto, todas elas possuem diversos elementos em comum, como instruções para obter as informações que o programa necessita e um conjunto de estruturas para manipular essas informações e exibir um resultado para o usuário do programa.

Conceito de Algoritmos

Inicialmente, devemos observar que um algoritmo não está relacionado apenas à ciência da computação ou à programação de computadores. De forma geral, um algoritmo é uma forma de descrever passo-a-passo a solução de algum problema. Desse modo, você está fazendo um algoritmo ao explicar a solução de um problema de física a um colega ou quando anota os ingredientes e o procedimento para fazer uma receita da sua comida favorita.

O interessante, ao se fazer um algoritmo, é que ele pode ser reproduzido por qualquer outra pessoa que tenha interesse no problema que ele se propõe a solucionar. Se você fizer um algoritmo, e ele estiver correto, qualquer pessoa que segui-lo fielmente deve obter como resultado a solução do problema. Isso é semelhante a você realizar o preparo de uma comida a partir da sua receita. Pois bem, caro(a) aluno(a), vamos apresentar nosso primeiro algoritmo através de uma receita simples e saborosa. Espero que você esteja com fome, pois vamos aprender o algoritmo para preparar pipoca.

Algoritmo 1 - Preparo de pipoca

Ingredientes

- 1 (uma) xícara de chá de milho para pipoca
- 5 colheres de sopa de óleo
- Sal

Fonte: http://goo.gl/pAQovs

Figura 2: Pipoca

Modo de preparo

1. Despeje o conteúdo da xícara em uma panela fina e alta, cobrindo o fundo da panela, formando uma camada sem espaços;

2. Em seguida, acrescente o óleo de soja ou outro de sua preferência;

3. O milho deve ficar bem brilhante, mas não encoberto no óleo;

4. Em fogo alto, observe que as pipocas começam a estourar em até 2 minutos;

5. Assim que parar de fazer barulho (parar de estourar) a pipoca está pronta;

6. Retire com cuidado a pipoca da panela e despeje em uma bacia;

7. Acrescente um pouco de sal e prove;

8. Caso necessário, acrescente mais um pouco de sal até ficar ao seu gosto.

Fonte: adaptado de http://www.tudogostoso.com.br/receita/32450-pipoca-salgada.html]

No Algoritmo 1, duas partes estão em destaque, os ingredientes e o modo de preparo. Os ingredientes enfatizam o que é necessário para preparar nossa pipoca, ou seja, as "informações" que você precisa ter em mãos antes de começar a resolver o problema. O modo de preparo descreve, em um conjunto de passos, no caso 8 (oito), a solução do problema. Nesse caso, tem-se o algoritmo propriamente dito. Ao seguir esses passos na ordem em que são listados, usando os ingredientes nas quantidades recomendadas, é esperado que você obtenha ao final uma deliciosa pipoca.

O exemplo é bastante simples, porém útil para nos ajudar a entender diversos conceitos importantes. Primeiro, um algoritmo de fato descreve a solução de um problema, ou de parte dele, em um ou mais passos. Esses passos devem seguir uma ordem lógica, do primeiro passo até o último. O algoritmo pode ficar incorreto caso um passo seja adiantado ou suprimido. Por exemplo, se você esquecer de colocar o óleo (passo 3) boa parte das pipocas irá queimar.

Outra observação importante é que, após ser iniciado, o algoritmo deve sempre terminar. Ou seja, deve existir um passo final, que pode até mesmo ser uma mensagem indicando algum erro previsível no decorrer do processo. Essa parte de tratar e reportar um erro ao usuário será detalhada em aulas posteriores.

A partir do que vimos, é interessante definir o que é um algoritmo aplicado à solução de um problema computacional, como sendo:

Algoritmo

Um algoritmo é a descrição da solução de um problema em uma sequência ordenada e finita de passos.

Um programa de computador geralmente é composto por um ou vários algoritmos. É possível, e desejável, inicialmente descrever a ideia de forma geral por meio de um algoritmo e só depois traduzir esse algoritmo em um programa. Desse modo, você pode raciocinar melhor sobre o problema e a solução pode sair mais rápido ou ser de melhor qualidade. O algoritmo também é uma forma de comunicar a sua solução para outros programadores de modo independente de linguagem.

É importante destacar que, muitas vezes, existe mais de um caminho que leva à solução de um mesmo problema. Ou seja, é possível ter mais de um algoritmo que descreva o problema. Nesse caso, um algoritmo pode ser mais ou menos eficiente que outro em termos de tempo de conclusão da tarefa ou recursos consumidos pelo computador. No contexto desta disciplina, não importa se você fez um algoritmo melhor ou pior, com mais ou menos passos. O importante é chegar, de maneira correta, à solução proposta. Sugerimos leituras no material complementar para quem quiser se aprofundar no estudo dos algoritmos.

ATIVIDADE

1. Altere o algoritmo de preparação de pipoca, deixando-o mais detalhado. Nesse caso, insira ao menos mais 3 (três) passos no algoritmo.

2. Faça um algoritmo que descreva a troca do pneu de um carro. Se você nunca trocou um pneu, pesquise na internet os passos necessários.

a) a. Faça o algoritmo considerando que você tem certeza que o pneu já está furado e, a partir daí, precisa trocá-lo.

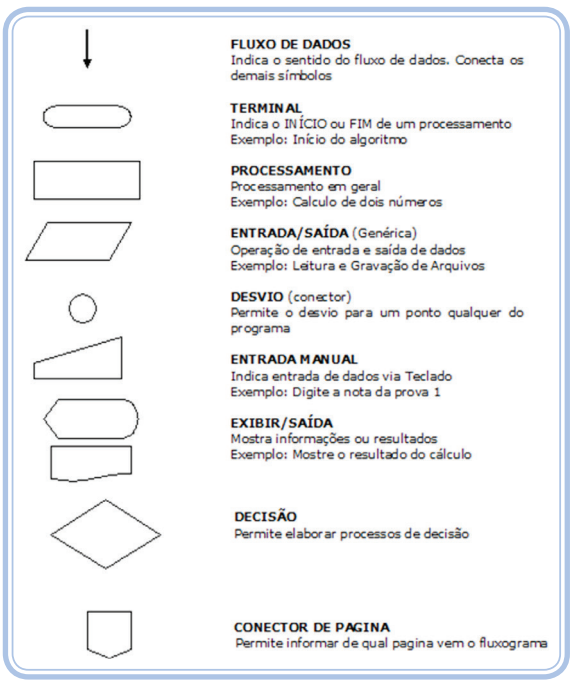
b) b. Considere incluir um ou mais passos para verificar se o pneu realmente está furado ou apenas precisando ser calibrado.

Formas de Representação de Algoritmos

Há diversas formas de representar um algoritmo. Inicialmente, no exemplo da preparação de pipoca (Código 1), usamos a narrativa em linguagem natural. Outras formas são a descrição por meio de figuras e fluxos de dados usando os diagramas de blocos e o uso do pseudocódigo ou português estruturado.

Cada forma tem seus prós e contras. No caso da linguagem natural, o lado positivo é que nós já conhecemos a linguagem que iremos trabalhar, que vem a ser a língua portuguesa escrita. A desvantagem é que as sentenças escritas em português podem levar a interpretações ambíguas ou deixar lacunas no entendimento. Por exemplo, no caso do algoritmo de preparação de pipoca, o que seria exatamente "acrescentar um pouco de sal"?. O pouco para uma pessoa pode significar muito para outra.

Os diagramas blocos usam figuras para representar processos comuns em algoritmos, como entrada de dados, fluxo e processamento de informações, tomadas de decisão e saída de dados (Figura 2). Essa forma de representação é interessante para algoritmos pequenos por facilitar a visualização do fluxo de informações. No entanto, para problemas mais complexos, pode se tornar difícil acompanhar o fluxo. Modificar o algoritmo também pode exigir certo esforço para reorganizar o diagrama.



Fonte: http://goo.gl/OzLYbR

Figura 3: Formas utilizadas em um diagrama de blocos

A forma mais comum de se representar algoritmos é por meio de pseudocódigo, também conhecido como *português estruturado* ou *portugol*. Ela estabelece uma linguagem bem estruturada, intermediária entre a linguagem natural e uma linguagem de programação. As instruções que são comumente encontradas nessas linguagens estão de certo modo presentes no pseudocódigo. Assim, ela se torna um modelo genérico que você programador pode traduzir no código final de uma linguagem de programação de sua preferência.

A desvantagem em usar o pseudocódigo é que não há um modelo específico que seja usado por todos. Embora, em geral, o código seja compreensível, cada um estabelece sua notação de pseudocódigo. Assim, é preferível usar essa forma de representação quando você deseja comunicar um algoritmo de forma independente de linguagem ou quando você quer rascunhar uma ideia antes de partir para a programação. Esse comportamento de esboçar uma ideia é bem interessante, pois permite a você pensar sobre os passos do algoritmo, a entender melhor o que você quer alcançar.

Como exemplo, temos um algoritmo que recebe dois números e diz se o primeiro número é maior que o segundo (Algoritmo 2).

Algoritmo 2 – Recebe dois números e diz se o primeiro número é maior que o segundo

```
1. início
2.     leia (n1, n2)
3.     se n1 > n2 então
4.         escreva (n1, " é maior que", n2)
5.     fimse
6. fim
```

Em um primeiro momento, peço a você que não se assuste com o código do Algoritmo 2. Vamos entendê-lo em todos os seus detalhes em poucas aulas. As linhas do algoritmo foram numeradas para facilitar a citação no texto, um procedimento que iremos seguir daqui em diante. Na *linha 1*, temos

a palavra "início", ela demarca o começo do algoritmo. A seguir, na *linha 2*, estamos colhendo os dados que serão usados no algoritmo, no caso, o primeiro número (n1) e o segundo número (n2). Na *linha 3*, é feito um teste para saber se *n1* é maior que *n2*. Caso esse teste seja verdadeiro, o texto da *linha 4* é impresso. A linha 6 demarca o fim do algoritmo.

Linguagem de Programação Python

Neste curso, nós decidimos por iniciar a atividade de programação diretamente em uma linguagem de programação. Uma motivação para isso é que você poderá visualizar, na prática, o resultado dos seus programas. O ambiente de programação que iremos usar destaca em cores as palavras que são parte da linguagem, permite executar o programa e aponta erros no código.

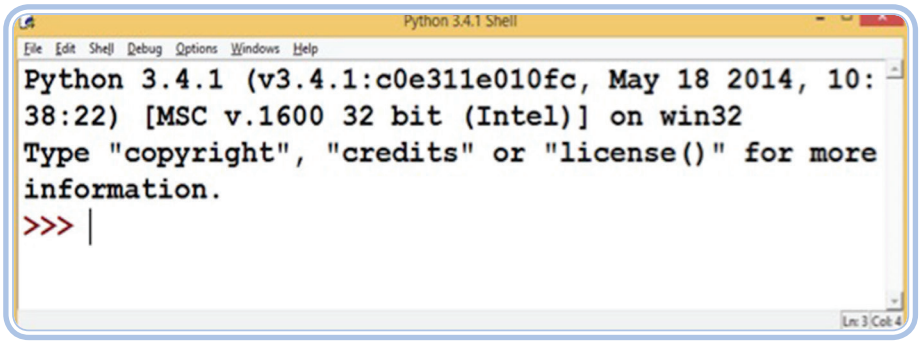
Por motivos didáticos, a linguagem escolhida para uso foi o *Python* (PYTHON, 2015), uma linguagem bastante simples, poderosa e popular. Ela pode ser aplicada tanto no desenvolvimento de programas para *desktop* quanto para a Internet. *Python* é excelente como uma primeira linguagem por possuir uma sintaxe simples, direta e de fácil aprendizado. Uma vez que as instruções nessa linguagem são formadas por palavras em Inglês, sempre que necessário essas instruções serão explicadas a partir dos termos equivalentes em Português.

Na seção de leituras complementares você encontra um tutorial que lhe ensina a instalar e usar o ambiente *Python* para começar a programar. Para os mais apressados, é possível seguir as instruções de download e instalação do ambiente no *site* oficial da linguagem, <http://www.python.org/downloads/>. É importante observar que este livro usa a versão 3 do *Python*. Esse detalhe merece destaque, pois essa versão apresenta algumas diferenças em relação a versão 2. Portanto, quando for fazer o *download* do ambiente, não esqueça de baixar a versão 3 ou uma posterior.

Introdução ao interpretador Python

O nosso primeiro contato com a linguagem *Python* será através de um programa chamado *idle*, que vem a ser um interpretador para a linguagem. Esse programa, como o próprio nome sugere, é responsável por interpretar cada instrução que você digitar e, caso ela esteja correta, realizar alguma ação no programa.

Inicialmente, vamos imaginar o ***Python*** como sendo uma calculadora. Após ter instalado o *Python*, abra o programa *idle*. Irá aparecer uma janela semelhante à Figura 3.



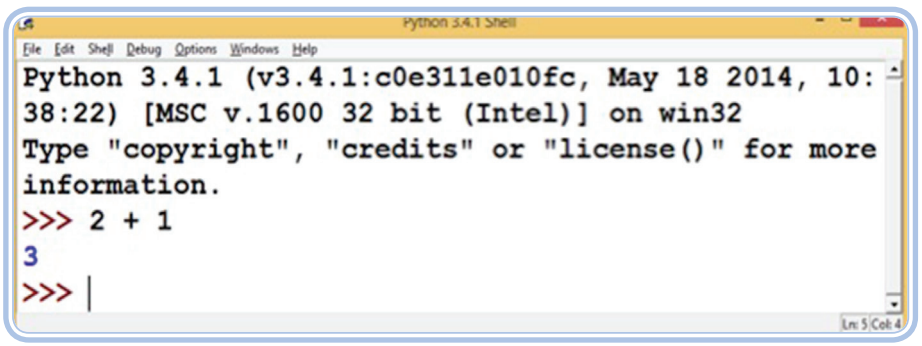
Fonte: Autoria própria.

Figura 4: Janela do interpretador *Python* (*idle*)

Na janela da Figura 3, o sinal ">>>" no interpretador Python indica que ele está aguardando que você digite um comando. Após digitá-lo, ao apertar o botão ENTER, o comando vai ser processado pelo interpretador e o resultado será exibido para você logo abaixo.

Observe que o símbolo (">>>") será utilizado no texto sempre que for necessário representar uma entrada de um valor no interpretador Python. Esse símbolo não é um comando da linguagem, apenas indica que você está no interpretador.

Tudo certo, agora vamos digitar um comando. No caso, vamos realizar uma soma entre dois números. No interpretador, digite 2 + 1 (O número 2, seguido de "+" e do número 1). Agora aperte o botão ENTER e veja o resultado, conforme a Figura 4. Nesse exemplo, 2 e 1 são valores constantes e o símbolo "+" é o operador para adição entre dois números. O interpretador lê o seu comando, calcula o resultado e exibe para você na linha abaixo (na cor azul). A partir daí ele fica pronto para receber outro comando (veja o símbolo ">>>").

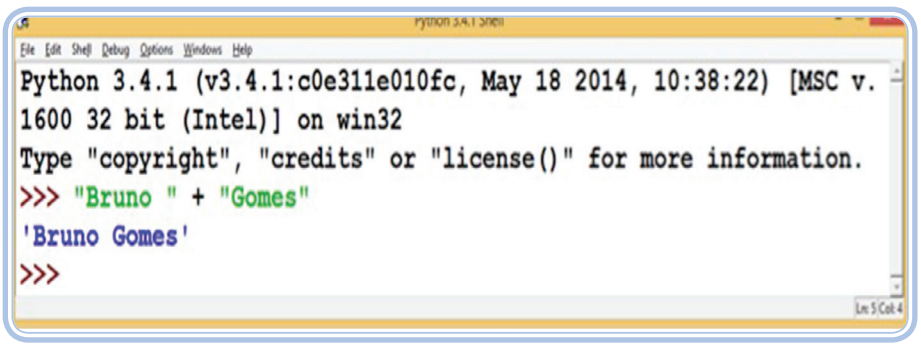


Fonte: Autoria própria.

Figura 5: Resultado da interpretação da expressão (2 + 1)

Você pode tentar uma expressão mais elaborada, como 2 * (10 / 3) + 1. As expressões são abordadas em detalhes na aula 2. Na linguagem, é possível também trabalhar com textos. No caso, se você digitar um texto qualquer, o interpretador irá gerar como resultado o próprio texto. A próxima seção descreve como salvar um valor, seja texto ou número para que possamos usar posteriormente no programa.

Perceba que, em *Python*, o operador "+" também pode ser usado com textos. Certo, mas como assim? Ele irá "somar" o texto? Não, ao ser aplicado a um ou mais textos, o símbolo de "+" tem o papel de juntá-los. Para ver como isso funciona, você pode tentar juntar o seu nome e sobrenome, como "Bruno " + "Gomes" (Figura 5). Observe que foi deixado, propositalmente, um espaço em branco no primeiro nome. Isso foi feito para que os nomes ficassem separados por um espaço.



Fonte: Autoria própria.

Figura 6: Operador "+" aplicado a dois textos

Variáveis e Atribuição de Valores

Na seção anterior vimos como inserir valores e instruções básicas no interpretador *Python*. Conforme explicamos, ao terminar de digitar no interpretador *Python* e clicar com o botão ENTER, o interpretador executa a instrução digitada e devolve o resultado. Ocorre que, nesse caso, o valor lido não fica "salvo" em nenhum lugar.

É importante destacar, caro(a) aluno(a), que na maioria das vezes, você está lendo um valor para usar posteriormente no programa. Esse valor pode ser, por exemplo um nome, uma data, ou um ou mais números para se usar em um cálculo matemático.

Pois bem, então como faremos para salvar um valor qualquer no nosso programa? Nesse caso, podemos salvar o valor em uma variável. Mas, o que

vem a ser uma variável? Inicialmente, vamos pensar em uma variável como uma gaveta, ou seja, um lugar onde guardamos algum objeto, como uma escova de cabelo. Sempre que for preciso pentear os cabelos, podemos abrir a nossa gaveta e pegar a escova. Caso a escova se quebre, é possível adquirir uma nova e substituir a anterior na gaveta.

No caso dos nossos programas de computador, as "gavetas" são representadas por espaços na memória do computador. Ou seja, ao criar uma variável, na verdade, estamos reservando uma gaveta na memória para guardar uma informação que o nosso programa irá precisar. Criar uma variável é bastante simples. Na verdade, basta apenas que você dê um nome a ela e armazene um valor.

Fonte: https://goo.gl/eKmyTj

Figura 7: Variável

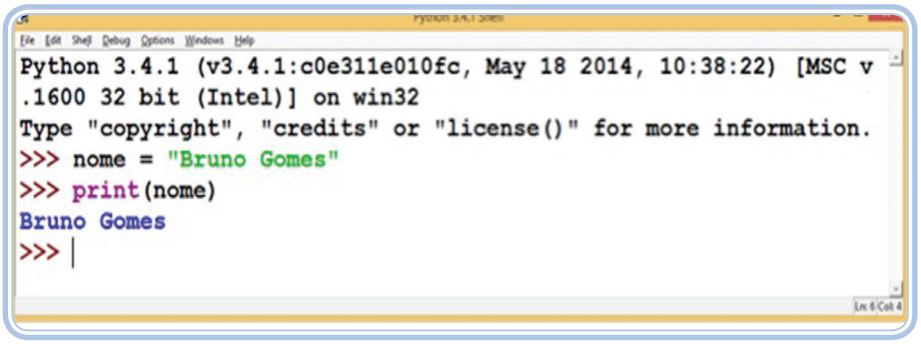
No nosso exemplo, podemos criar uma variável denominada "nome" para receber o um valor de texto com o nome de uma pessoa. Assim, o que for digitado no teclado ficará armazenado na variável *nome*, conforme pode ser visto no Código 1.

Código 1 – Salva um texto na variável nome

```
>>> nome = "Bruno Gomes"
```

Observe que o símbolo de "=" (igualdade), denominado de *atribuição*, tem uma função especial na linguagem. É com esse símbolo que dizemos que estamos querendo salvar (ou atribuir) o valor que está à direita dele, no caso o texto "Bruno Gomes", na variável que está à esquerda.

Se você der um *print* na variável nome, conforme a Figura 6, você tem como resultado o conteúdo que se encontra armazenado na variável. *Print* é um comando do **Python** utilizado para gerar uma saída na tela para o usuário do programa.



Fonte: Autoria própria.

Figura 8: Salvando um texto em uma variável e imprimindo o resultado

Observe que você pode fornecer qualquer nome a uma variável, desde que ele comece com uma letra ou *underscore* (_) e seja seguido por letra, número ou *underscore*. Um nome de variável não pode começar com um número, nem possuir caracteres especiais ou espaços em branco. Outra restrição é que não é permitido usar um nome já definido na linguagem, como *print* ou *input*.

Embora uma variável possa receber qualquer nome dentro desses requisitos, é uma boa prática que você dê um nome coerente com o que a variável irá armazenar. Por exemplo, se a variável armazena um nome, eu não vou nomeá-la de "casa", ou de "computador", e sim de "nome" ou "pessoa", a depender do contexto em que ela será utilizada.

Para facilitar a manipulação no programa, geralmente não colocamos acentos nos nomes de variáveis, embora seja permitido em *Python*. Uma outra observação é que nomes de variáveis são sensíveis à caixa. Ou seja, a variável "*pessoa*" é diferente das variáveis "*Pessoa*" e "*PESSOA*".

Sugiro que você mantenha um padrão de escrita de nomes de variáveis, possivelmente colocando todos os nomes em caixa baixa (letras minúsculas). No caso de uma variável possuir mais de um nome, você pode começar a letra da segunda palavra em diante em maiúscula ou separar as palavras com *underscore* como "*nomePessoa*" ou "*nome_pessoa*".

ATIVIDADE

1. Com a ajuda do interpretador Python, verifique qual desses nomes de variáveis é válido. Para testar, atribua um valor qualquer a cada nome de variável.

a. cidade

b. __Cidade

c. 2nome

d. idade pessoa

e. idadePessoa

f. CPF

g. len

h. &endereço

i. _rua_

j. x

Como o próprio nome sugere, e você já deve ter percebido, uma variável é um valor que pode mudar sempre que necessário. Desse modo, se você quiser trocar o valor armazenado na variável "*nome*", é só modificar o texto da atribuição. Por exemplo, se você digitar no interpretador a instrução do Código 2, o valor armazenado na variável "nome" será atualizado para o texto "Thiago Medeiros".

Código 2 – Atualizando o valor da variável "nome"

```
>>> nome = "Thiago Medeiros"
```

No caso, usamos um valor constante do tipo texto, que é representado por qualquer valor entre aspas (" ") ou apóstrofo (' '). Tanto faz você inserir "Thiago Medeiros" ou 'Thiago Medeiros', desde que você seja coerente. Se começou com aspas, termine com aspas. O mesmo vale para o apóstrofo.

LEMBRE-SE

O símbolo ">>>" (três sinais de maior seguidos) neste caso não faz parte do comando. Ele indica apenas que você está no interpretador *Python*.

Nesta seção, definimos como exemplo uma variável "nome" que recebe um texto. A próxima aula apresenta outros tipos de dados básicos que podem ser usados nos nossos programas. Na aula seguinte, também vamos detalhar as instruções de entrada e saída de dados, *input* e *print*, respectivamente, e aprender a criar e interpretar expressões mais complexas, formadas por variáveis e operadores de diversos tipos.

RESUMINDO

Nesta aula, vimos os conceitos básicos da programação de computadores. Apresentamos conceito de algoritmos e as suas formas de representação em linguagem natural, diagrama de blocos e pseudocódigo. Iniciamos o estudo da programação com a linguagem *Python* usando seu interpretador para o desenvolvimento e execução dos nossos primeiros códigos. Em um primeiro momento, aprendemos a usar o interpretador como se fosse uma calculadora. Posteriormente, vimos que através do uso de variáveis, é possível armazenar informações no nosso programa. As variáveis são nomes que podem receber valores e guardá-los para uso posterior no programa.

LEITURAS COMPLEMENTARES

Iniciação aos algoritmos e à programação

Code.org (https://code.org) – Esta página apresenta diversos materiais para de introdução a programação. Sugiro que você faça a "hora do código", um jogo bem interessante que exercita a lógica de programação.

Code Academy (http://www.codecademy.com) – Site que se propõe a ensinar programação em diversas linguagens, incluindo Python. O *site* fornece uma interface para você praticar online enquanto aprende cada lição.

Algoritmos (MANZANO; OLIVEIRA, 2014) – Este livro apresenta uma introdução interessante dos conceitos iniciais de algoritmos e linguagens de programação e sobre os diversos paradigmas de programação.

Linguagem de programação Python

Instalação do ambiente Python – Você pode obter ajuda para instalar o ambiente de programação Python de diversas fontes. No Youtube, o vídeo https://www.youtube.com/watch?v=wpqkZJ10Gmo ensina a instalação do Python 3 para o Windows 7. O *site Python Club* (http://pythonclub.com.br/instalacao-python-django-windows.html) explica a instalação de forma textual. Observe que ele demonstra a instalação do *Python 2.7*, mas você deve obter a versão 3, mais recente.

Site oficial da linguagem (https://www.python.org) – Apresenta a linguagem, sua documentação oficial e permite o download do ambiente de programação.

Python para Zumbis (http://pycursos.com/python-para-zumbis) – Curso online gratuito da linguagem Python através de vídeo-aulas.

AVALIANDO SEUS CONHECIMENTOS

1. O que é um programa de computador?

2. Qual(is) a diferença (as) entre um programa de computador e um algoritmo?

3. Resolva o seguinte problema de lógica por meio de um algoritmo em linguagem natural:

> Três jesuítas e três canibais precisam atravessar um rio. Para tanto, dispõem de um barco com capacidade para apenas duas pessoas. Por medidas de segurança, não se deve permitir que em alguma margem do rio a quantidade de jesuítas seja inferior a de canibais. Elabore um algoritmo indicando os passos necessários para que a travessia seja feita de forma segura.

4. Imagine que você acionou o interruptor da lâmpada do seu quarto e ela não ligou. Faça um algoritmo que descreva o procedimento para realizar a troca da lâmpada.

5. Verifique o que está errado nos códigos a seguir e faça as devidas correções.

a. >>> print 'Instituto Federal"

b. >>> nome$ = 10

6. Qual o valor da variável "x" ao final da execução do código a seguir?

```
>>> x = 37

>>> x = x + 5

>>> x  = x – 2
```

Referências

CODE.ORG. Disponível em: <https://code.org/>. Acesso em: 05 fev. 2015.

CODE ACADEMY. Disponível em <http://www.codecademy.com/>. Acesso em: 09 fev. 2015.

CORMEN, T. H. et al. **Algoritmos**: Teoria e Prática. 3. ed. Rio de Janeiro: Campus, 2012.

CORÔA, T. **Instalando e Configurando o Python e Django no Windows**. 2014. Disponível em: <http://pythonclub.com.br/instalacao-python-django-windows.html>. Acesso em: 29 abr. 2015.

FORBELLONE, A. L. V.; EBERSPACHER, F. H. **Lógica de Programação**: A construção de Algoritmos e Estruturas de Dados. 3. ed. São Paulo: Pearson, 2005.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Algoritmos**: Lógica para Desenvolvimento de Programação de Computadores. São Paulo: Érica, 2014.

MASANORI, F. **Python para Zumbis**. [2013]. Disponível em: <http://pycursos.com/python-para-zumbis/>. Acesso em: 05 fev. 2015.

MEDINA, M.; FERTIG, C. **Algoritmos e Programação**: Teoria e Prática. 2. ed. São Paulo: Novatec, 2006.

PYTHON. Disponível em: <https://www.python.org/>. Acesso em: 05 fev. 2015.

PYTHON BRASIL. Disponível em: <http://wiki.python.org.br/>. Acesso em: 05 fev. 2015.

REIS, F. **Lógica de programação**: Vídeo 03: Instalação do ambiente Python 3.3 e IDLE no Windows 7. [2013]. Disponível em: <https://www.youtube.com/watch?v=wpqkZJ10Gmo>. Acesso em: 29 abr. 2015.

SEVERANCE, C. **Python for Informatics**: Exploring Information. 2013. Disponível em: <http://www.pythonlearn.com/book.php>. Acesso em: 05 fev. 2015.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de Dados e Seus Algoritmos**. 3. ed. São Paulo: LTC, 2010.

Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 06
Entrada e Saída, Tipos de Dados Básicos e Expressões

Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno(a)! Na aula 5 aprendemos sobre os fundamentos da programação. Vimos que nossos programas são compostos por algoritmos, que são uma forma de descrever um problema usando um conjunto de instruções estruturadas que produzem um resultado. Convido você a continuar esta caminhada em busca de conhecimento na programação de computadores. Nesta aula, vamos aprender como pedir ao usuário do programa para inserir valores e como exibir um resultado para esse usuário. Cada valor possui um tipo de dados. Vamos conhecer quais os tipos de dados mais comuns da linguagem *Python*. Além disso, você irá aprender a criar expressões compostas de valores desses diversos tipos de dados.

Aula 6 - Entrada e Saída, Tipos de Dados Básicos e Expressões

Objetivos

Caro(a) aluno(a), nesta aula são apresentados comandos na linguagem *Python* que permitem inserir, modificar e exibir valores em seu programa. Posteriormente, são demonstrados os tipos de dados básicos utilizados na representação de informações. Por fim, vamos conhecer os operadores aritméticos, lógicos e relacionais e aprender a criar expressões que usem esses operadores. Ao término da aula, você dever estar apto(a) a:

- Ler do teclado e escrever valores na tela com o uso das instruções *input* e *print*;
- Utilizar os tipos básicos de dados utilizados na representação de informações em um programa;
- Utilizar operadores, variáveis e valores constantes para criar expressões.

Desenvolvendo o conteúdo

Instruções Básicas de Entrada e Saída de dados

Até o momento trabalhamos com valores constantes, por exemplo, números como 50.0 ou um texto como "Bom dia, Usuário!" na aula anterior, vimos também como salvar esses valores em variáveis para possibilitar o seu uso em um momento posterior no programa. Neste momento, vamos aprender como obter um valor que deve ser fornecido por quem está usando seu programa (usuário). Vamos ver também a operação contrária, ou seja, como exibir um valor ou texto de saída para o usuário.

É importante destacar que essas atividades de obter informações (entrada de dados) e exibir um resultado (saída) são uma das tarefas mais básicas em

programação. Em verdade, um *software* basicamente é formado por uma ou mais etapas de entrada de dados, processamento desses dados e saída (Figura 1).

Fonte: http://voufazersemreclamar.blogspot.com.br/p/6-ano.html

Figura 01: Entrada, processamento e saída

Observe que neste curso a entrada de dados será feita através do teclado do computador, em um ambiente de linha de comando. É possível também a entrada por uma interface gráfica composta por janelas, botões e ícones. Nesse caso, o usuário também pode usar o *mouse* como dispositivo de entrada. O processamento corresponde a usar esses dados recebidos para realizar a tarefa a qual o programa se propõe. A saída é a etapa final, onde o(s) resultado(os) do processamento é(são) exibido(s) para o usuário do programa.

Por exemplo, em um sistema de cadastro de um cliente em uma loja na Internet, a entrada seria o formulário que o cliente preenche com os seus dados, como nome, CPF, identidade, endereço etc. O processamento seria a etapa de salvar essas informações em um banco de dados no computador. Por fim, a saída corresponde à mensagem que é exibida após o processamento, indicando se a operação foi realizada com sucesso ou não.

Muito bem, você está pronto para aprimorar ainda mais seus conhecimentos na programação em *Python*? Pois vamos começar pela instrução (ou comando) *print*, que significa escrever ou imprimir. Abra o interpretador *Python* (*idle*) e digite a instrução abaixo.

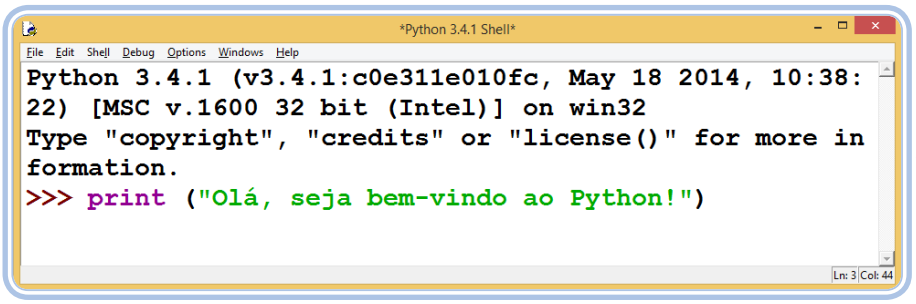
Código 1 - Imprime na tela uma mensagem para o usuário

```
>>> print ("Olá, seja bem-vindo ao Python!")
```

LEMBRE-SE

O sinal ">>>" no interpretador *Python* indica que ele está aguardando que você digite um comando.

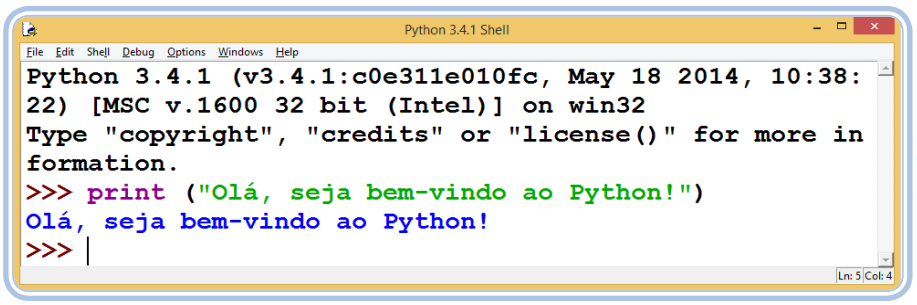
Para que você veja se está no caminho certo, a janela do interpretador com o código digitado deve se assemelhar a Figura 2. Após digitar o comando, ao apertar o botão ENTER, o comando vai ser processado pelo interpretador e, caso ele tenha algum resultado, esse resultado será exibido para você logo abaixo (Figura 3).



Fonte: Autoria própria.

Figura 02: Interpretador Python com instrução *print* digitada

O resultado que será exibido no comando *print* é o valor que está entre os parênteses. No nosso caso, o texto "*Olá, seja bem-vindo ao Python!*" nta verdade, vamos usar *print* sempre que for necessário fornecer uma mensagem ou resultado do programa para o usuário.



Fonte: Autoria própria.

Figura 03: Resultado do processamento da instrução *print* (texto em azul)

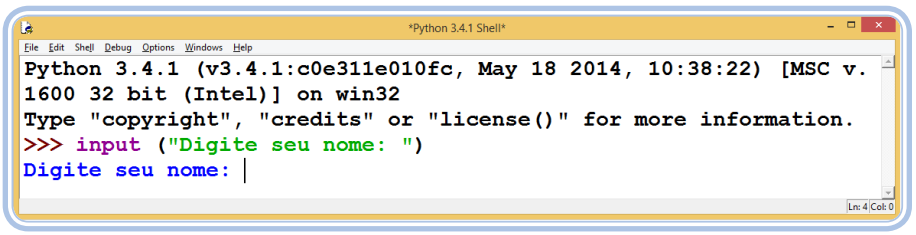
Para receber uma entrada do usuário pelo teclado do computador devemos usar o comando *input*, que tem a forma input ("*Mensagem*"). A mensagem que é recebida no comando deve ser qualquer texto que informe ao usuário do programa o que ele deve digitar.

Vamos fazer um programa para ler o nome de uma pessoa, que pode ser o seu. Nesse caso, devemos inserir uma instrução *input* solicitando que o nome seja digitado, como no código 2 a seguir.

Código 2 – Obtém o nome de uma pessoa usando a instrução input

```
>>> input ("Digite seu nome: ")
```

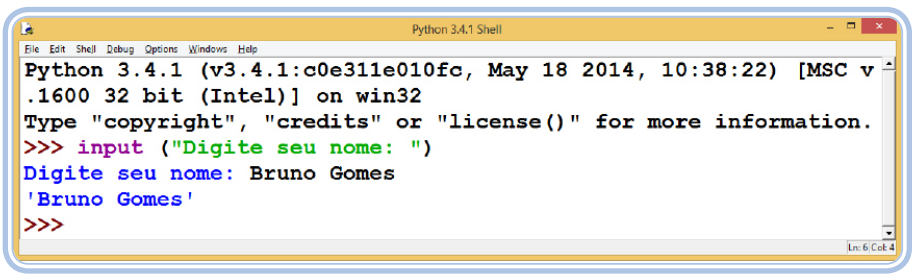
Inicialmente, o comando *input* insere na tela a mensagem pedindo a entrada de algum valor e fica aguardando o usuário digitá-lo. No caso do nosso exemplo, esse valor é o nome de uma pessoa (Figura 4).



Fonte: Autoria própria.

Figura 04: Entrada de dados com o comando *input*

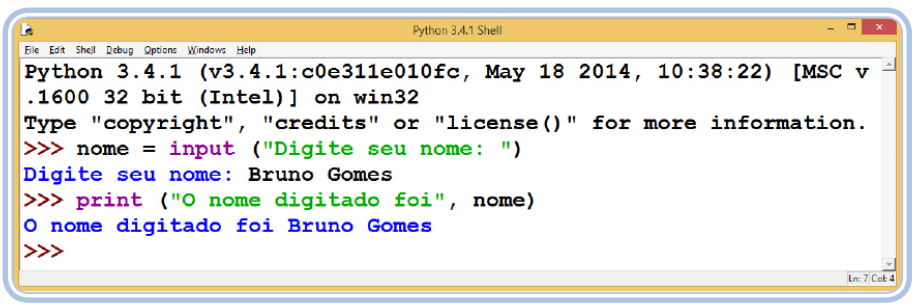
Após inserir o valor (o texto com o seu nome), você deve clicar no botão ENTER. Isso faz com que o valor seja enviado para o mesmo lugar do programa onde se encontra a instrução *input* (Figura 5).



Fonte: Autoria própria.

Figura 05: Resultado da leitura com input usando o interpretador Python

Caso você queira salvar esse valor em uma variável, é preciso apenas que você coloque o comando *input* à direita da atribuição (sinal de "="), conforme pode ser visto na Figura 6. Nesse caso, o nome digitado fica salvo na variável *nome*. A Figura 6 também exibe o resultado da impressão do valor armazenado usando a instrução *print* (*nome*).



Fonte: Autoria própria.

Figura 06: Salvando um nome lido com input e imprimindo com *print*

ATIVIDADE

1. Leia o primeiro nome de uma pessoa e a sua idade e salve nas variáveis nome e idade, respectivamente. Imprima a frase "*Olá, nome, você tem, idade, anos*", na qual os *nome* e *idade* representam os valores que estão armazenados nas variáveis.

OBS: Para imprimir mais de um valor em uma mesma frase, é preciso apenas que você separe os valores usando vírgula. E, sempre que for imprimir um texto, lembre-se de colocá-lo entre as aspas ou apóstrofo. Por exemplo: print ("Olá, ", nome, "hoje é dia", data), irá imprimir "Olá, Bruno, hoje é dia 22/02/2015, para os valores *nome = Bruno e data = 22/02/2015.*

2. Leia dois números, salve-os cada um em uma variável e imprima a soma desses números.

OBS: O que o usuário digita no teclado é recebido no comando *input* como um texto. Nesse programa, você deve converter a saída de *input* para um número com o comando *float* () envolvendo o *input*, como em:

```
n1 = float (input("Digite um número: "))
```

Na seção a seguir vamos detalhar os tipos de dados básicos que podem ser usados na linguagem *Python*. Os tipos presentes na linguagem não são restritos a esses. Porém, os tipos destacados abaixo são comumente usados na maioria dos programas.

Tipos de Dados Básicos

É importante você lembrar que criamos variáveis para guardar alguma informação que será útil ao programa. Essa informação, ou valor, deve ser de um tipo válido na linguagem. Desse modo, é possível fazer um cálculo, no caso de números, ou realizar alguma operação sobre um texto. A linguagem possui alguns tipos que são definidos por padrão e podem ser utilizados livremente nos seus programas.

Nesta seção, destacamos alguns tipos básicos que são usados com bastante frequência. Na seção de expressões, apresentamos como montar expressões

compostas com valores e operadores dos tipos detalhados a seguir.

Tipo inteiro (*int*)

Representa os números inteiros. Ou seja, qualquer valor numérico sem a parte fracionária. Exemplos: 0, -1, 234, -98.

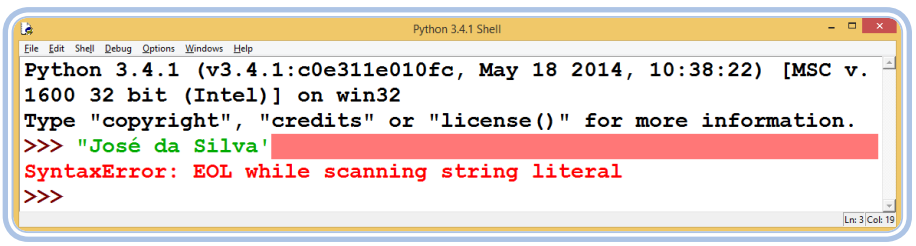
Tipo real (*float*)

Compreende os números reais, compostos por uma parte inteira e uma parte fracionária separadas por um ponto decimal. É importante reforçar que o separador decimal é um ponto, como na língua inglesa, e não uma vírgula, como no português. Exemplos: 0.0, 8.95, 2.1.

Tipo texto (*string*)

Informações compostas por zero ou mais caracteres alfanuméricos (letras, dígitos e símbolos especiais) colocados entre aspas ou apóstrofos. Na linguagem *Python*, o tipo texto é denominado de *string*. Exemplos: "IFRN", "José da Silva", 'Rua das flores, n. 44', ''a", '"".

É importante notar que se você começou o texto com aspas ('') você deve fechá-lo também com aspas. O mesmo vale para apóstrofo('). Caso você misture os dois formatos em um mesmo *string*, irá ocorrer um erro de sintaxe, como pode ser visto na Figura 7.



Fonte: Autoria própria.

Figura 07: Erro ao tentar iniciar um texto com aspas e fechar com apóstrofo

É interessante perceber que o espaço em branco também é um caractere significativo, podendo ser representado por (' ' ou " "). Se você juntar as aspas, sem dar nenhum espaço, você está definindo um texto vazio. Isso pode ser útil para criar uma nova variável do tipo *string* sem colocar nenhum valor significativo nela.

Tipo booleano (*bool*)

O tipo *bool* corresponde as informações que podem assumir os valores lógicos *verdadeiro* ou *falso*. Em *Python*, o valor verdadeiro é representado pela palavra *True* e *falso* pela palavra *False*. Essas palavras devem ser escritas da forma como foram apresentadas, com a primeira letra em maiúscula.

Por exemplo, suponha que no seu programa você esteja procurando um valor, como um número em uma lista de números. É possível criar uma variável denominada "encontrado" para representar se esse valor foi ou não encontrado. Na inicialização da variável é interessante inserir o valor falso, pois o valor ainda não foi encontrado (encontrado = *False*). Após o processamento do seu programa, caso o valor seja encontrado, você deve mudar o valor armazenado para verdadeiro (encontrado = *True*).

Consultando o tipo de um valor

Caso você queira consultar o tipo de um valor em *Python*, você pode usar o comando *type* (*valor*) como pode ser visto nos exemplos do Código 2.

Código 2 – Consultando o tipo de alguns valores em Python

```
>>> type (10)

<class 'int'>

>>> type ("João")

<class 'str'>

>>> x = 80.5

>>> type (x)

<class 'float'>
```

Você deve ter notado que não é necessário dizer de forma explícita qual o tipo de uma variável em *Python*. O tipo é estabelecido de acordo com o valor que a variável recebe. Caso ela receba um número inteiro, o seu tipo será inteiro, caso receba um texto, o tipo será *string* e assim por diante. Isso é o que chamamos de atribuição dinâmica de tipos.

Embora a mudança de tipo em uma variável seja permitida, não é aconselhável que você use este recurso, pois torna mais difícil a descoberta e a correção de erros em seu programa. Portanto, sempre pense em uma variável como sendo de apenas um tipo. Caso você precise guardar outro valor, é preferível criar uma nova variável.

Conversão entre tipos

É possível, e por vezes necessário, converter entre os tipos de dados. *Python* possui funções tais como *int*(expressão), *float*(expressão) e *str*(expressão) para forçar a conversão para inteiro (*int*), real (*float*) e texto (*string*), respectivamente. Por exemplo, caso você queira converter a constante 9.5 para o inteiro 9, basta fazer *int* (9.5).

A conversão é particularmente importante quando você quer ler um valor usando *input*, que sempre devolve um texto (*string*). Nesse caso, se você quer ler um número, é necessário converter a saída de *input* para *int* ou *float*, como no exemplo do Código 3.

Código 3 – Convertendo a saída de input para número

```
>>> x = input (“Digite a sua nota: “)

>>> nota = float (x)
```

Você também pode fazer a conversão direto na linha da leitura com *input*, como por ser visto no código 4.

Código 4 – Convertendo a saída de input para número direto na linha do input

```
>>> nota = float (input (“Digite a sua nota: “))
```

ATIVIDADE

1. Leia dois valores do tipo inteiro e armazene-os em duas variáveis, denominadas de *n1* e *n2*, respectivamente. Lembre-se de converter o a saída da função *input* para inteiro (*int*), nas duas leituras (*n1* e *n2*).

a) Realize e imprima o resultado a divisão de *n1* por *n2* (*n1*/*n2*).

b) Verifique, usando *type*, o tipo da divisão realizada no item 'a'. Exemplo: *type* (n1/n2) ou *type* (res), caso você tenha salvo o resultado em uma variável denominada de *res*.

Expressões

As expressões são úteis sempre que precisamos realizar um cálculo, fazer comparações ou realizar testes. Sendo assim, é de extrema importância saber como criar expressões e entender o valor que elas devolvem. Por exemplo, se digitarmos o número 10 no interpretador *Python*, nós temos uma expressão formada por um valor constante do tipo inteiro, o valor 10. Caso você digite 2 * 4.0, tem-se uma expressão do tipo real composta de duas constantes e do operador de multiplicação (*). Nesse caso, o resultado é o valor 8.0. Antes de seguir adiante, vamos definir o que é uma expressão.

[DEFINIÇÃO] Expressão

Uma expressão é uma variável, uma constante, ou qualquer combinação válida de variáveis, constantes e operadores que devolve um resultado após a sua avaliação.

Como pode ser visto pela definição, expressões usualmente são compostas por operadores. Há diversos operadores disponíveis, a depender do tipo da expressão. Nesta seção, você irá trabalhar com expressões aritméticas (inteiros e reais), lógicas (verdadeiro – *True* ou falso – *False*) e relacionais (resultados de comparações, verdadeiro ou falso). No caso de expressões aritméticas, são exemplos de operadores os símbolos + (adição), - (subtração), * (multiplicação) e / (divisão).

Observe que um mesmo operador pode aparecer em expressões de mais de um tipo, como o operador "+". Caso a expressão seja *string*, o operador "+" tem a função de unir duas ou mais *strings* na expressão. Por exemplo, "Olá" + "mundo!" irá gerar o texto "Olá mundo!".

Os operadores podem ser classificados como *unários*, quando são aplicados sobre apenas um operando (valor) ou *binários* quando devem ser aplicados a dois valores. Como exemplo, o operador "+" (soma) na expressão 5 + 6 é

binário, pois é aplicado à soma do valor 5 com o valor 6. Já o operador "–" (negação) é unário na expressão –6, representando a negação do número 6. Por outro lado, ele é binário e significa subtração na expressão 9 – 3.

Expressões aritméticas (numéricas)

São aquelas que operam sobre valores inteiros ou reais. Se os operandos em uma expressão são inteiros, o resultado da expressão será inteiro, exceto para a divisão, que na versão 3 do *Python* resulta em um real. Por exemplo, 4 / 2 (4 dividido por 2) irá resultar em 2.0. O resultado de uma expressão numérica será real sempre que ela for composta por valores reais ou quando há operandos inteiros e reais em uma mesma expressão.

A Tabela 1 a seguir apresenta os símbolos usados para os operadores aritméticos e o seu significado. Observe que devido ao teclado do computador não ser capaz de apresentar todos os símbolos, alguns não são exatamente como aprendemos na escola. Por exemplo, a multiplicação é o "*" (asterisco) e a exponenciação é representada por "**" (dois asteriscos juntos).

Tabela 01: Operadores Aritméticos

Operador	Significado	Exemplos
+	Adição	2 + 3, x = y + 2
–	Subtração	6 – 1, x – a – 2
*	Multiplicação	7 * 3, num * 2
/	Divisão	15 / 3, (4 + 4) / 2
//	Parte inteira da divisão (quociente)	15 // 2 irá resultar em 7
%	Resto da divisão	6 % 2 (resultado é 0)
**	Exponenciação	2 ** 3 (resultado é 8)

Fonte: Autoria própria.

Observe que as expressões são escritas na forma linear, ou seja, em uma linha de texto, usando os operadores descritos na tabela. Por exemplo, a expressão $5^{3/2}$ (5 vezes 3 dividido por 2) deve ser traduzida em *Python* na forma linear como *5 * 3 / 2.* No caso da expressão $a^2 + b^2 = c^2$ temos como equivalente em *Python* a forma *a**2 + b**2 = c**2*.

ATIVIDADE

Traduza as expressões matemáticas abaixo para a forma linear usando os operadores da linguagem Python.

a) $(4\pi r^3)/3$

b) $b^2 - 4ac$

c) $r = 1/2\ at^2 + v_o t + r_0$

Você deve ter observado pelos exemplos que é possível ter em uma mesma expressão diversos operadores. Nesse caso, saiba que a linguagem estabelece uma ordem para a avaliação de operadores, o que é chamado de precedência. Operadores com maior precedência são avaliados (calculados) primeiro que aqueles com menor precedência. Caso tenhamos operadores com a mesma precedência sendo avaliados, a ordem de avaliação é da esquerda para a direita.

Tabela 02: Ordem de execução (precedência) dos operadores aritméticos. Da maior precedência para a menor

Procedência	Operador(es)	Nome
1	**	Exponenciação
2	*, /, //, %	Multiplicação, divisão e resto da divisão
3	+, -	Adição e subtração

Fonte: Autoria própria.

A lista de precedência (ou prioridade) da Tabela 2 diz que, em uma expressão aritmética, primeiro devemos resolver a exponenciação, depois, na mesma prioridade, a multiplicação, divisão e o resto (nesse caso, da esquerda para a direita), e por último a adição e a subtração. Caso você queira forçar a avaliação de operadores com precedência menor, antes de um de prioridade maior, você deve colocar essa parte da expressão entre parênteses. Abaixo, vemos alguns exemplos de resolução de expressões passo-a-passo seguindo a ordem de precedência dos operadores. Caso você insira essas expressões no interpretador, você irá obter apenas o resultado final da resolução.

Exemplos:

5 + 9 + 7 + 8/4 5 + 9 + 7 + 2.0 21 + 2.0 23.0	**1 - 4 * 3//6 - 3**2** 1 - 4 * 3//6 - 9 1 - 12//6 – 9 1 - 2 – 9 -10
(5 + 9 + 7) + 8/4 21 + 8/4 21 + 2.0 23.0	**(1 - 4) * 3/6 - 3**2** -3 * 3/6 - 3**2 -3 * 3/6 - 9 -9/6 - 9 -1.5 - 9 -10.5

Expressões Lógicas

Uma expressão lógica é aquela formada por operadores lógicos ou de comparação (relacionas). O resultado da avaliação de uma expressão lógica é sempre do tipo lógico, ou seja, verdadeiro (*True*) ou falso (*False*). Você já deve estar familiarizado com estas expressões, pois de certo modo elas já foram tratadas no contexto da lógica proposicional, na unidade 1.

Tabela 03: Operadores lógicos

Operador	Significado	Exemplos
Or	Ou (disjunção)	x **or** y
And	E (conjunção)	(2 > x) **and** (y < z)
Not	Não (negação)	**not** (a == b)

Fonte: Autoria própria.

Embora já seja do seu conhecimento, pelo estudo da unidade 1, para relembrar exibimos a seguir a tabela verdade para cada operador. As tabelas verdade mostram o resultado de todas as possíveis combinações de valores entre operandos que estão conectados usando algum operador lógico.

Tabela 04: Tabela verdade do operador 'and' Fonte: autoria própria.

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Fonte: Autoria própria.

Tabela 05: Tabela verdade do operador 'or'

A	B	A or B
False	**False**	**False**
False	**True**	**True**
True	**False**	**True**
True	**True**	**True**

Fonte: Autoria própria.

Tabela 06: Tabela verdade do operador 'not'

A	not B
True	False
False	True

Fonte: Autoria própria.

A precedência entre os operadores lógicos é exibida na Tabela 7. Inicialmente, é feito a negação (*not*). Posteriormente, é tratado o e (*and*) e, por fim, o ou (*or*).

Tabela 07: Precedência entre os operadores lógicos (da maior para a menor)

Prededência	Operador	Nome
1.	Not	Não
2.	And	E
3.	Or	OU

Fonte: Autoria própria.

Exemplos:

2 < 5 and 15 > 15 True and FalsetTrue	not (5 >= 3) or True not (True) or True False or True True

Operadores relacionais

São utilizados para se fazer comparações entre expressões. O resultado dessas comparações é lógico, ou seja, verdadeiro (*True*) ou falso (*False*).

Tabela 08: Operadores Relacionais

Operador	Significado	Exemplos
==	Igual a	3 == 3, x == y
>	Maior que	5 > 4, (y + z) > 8
<	Menor que	3 < 6, num < 12
>=	Maior ou igual a	5 >= 56, (a * b) >= (8 * 2)
<=	Menor ou igual a	3 <= 9, (a - b) + c >= 1
!=	Diferente de	8 != 9, a != b
is	É o mesmo que	x is y (x faz referência ao mesmo objeto de y)
is not	Não é o mesmo que	x is not y

Fonte: Autoria própria.

Exemplos:

2 * 4 == 24 / 3 8 == 8 True	3 * 5 / 4 <= 3**2 /0.5 3 * 5 / 4 <= 9 / 0.5 15 / 4 <= 18 3.75 <= 18 True
15 % 4 < 19 % 6 3 < 1 False	2 + 8 % 7 >= 3 * 6 - 15 2 + 1 >= 18 - 15 3 >= 3 True

Os operadores relacionais possuem a mesma precedência entre si. A precedência entre todos os operadores vistos é exibida na Tabela 9. Lembre-se que para forçar a avaliação de uma expressão com precedência menor, você deve colocá-la entre parênteses.

Tabela 09: Precedência entre todos os operadores vistos

Precedência	Operador(es)
1.	**
2.	*, /, //, **
3.	+, -
4.	>, >=, <, <=, ==, !=
5.	is, is not
6.	Not
7.	And
8.	Or

Fonte: Autoria própria.

RESUMINDO

Você aprendeu nesta aula a realizar a entrada e a saída de dados usando *input* e *print*. Vimos que o *input* lê do teclado e devolve um texto com o valor lido. Nesse caso, se você quiser usar um número você deve converter a saída com input usando *int* e *float*. Os tipos de dados básicos *int, float, string e bool* foram detalhados e aprendemos a criar expressões com esses tipos. Uma expressão é composta por variáveis, constantes e operadores e, após ser avaliada, devolve um resultado para o usuário.

LEITURAS COMPLEMENTARES

PYTHON.ORG (https://docs.python.org/3/tutorial/inputoutput.html) – Esta

página faz parte da documentação oficial da linguagem *Python* e é interessante para quem quer se aprofundar mais em entrada e saída de dados. Nela, você terá exemplos de saída com formatação e de entrada e saída em arquivos, dentre outros tópicos.

IBM (http://www.ibm.com/developerworks/br/library/os-python1/) – Neste *site*, você tem um material complementar sobre tipos básicos em Python e as expressões que vimos nesta aula.

AVALIANDO SEUS CONHECIMENTOS

1. Verifique o que está errado nos códigos a seguir e faça as devidas correções.

a) >>> print 'Instituto Federal"

b) >>> nome$ = 10

c) >>> x = input ("Número 1: ", "Número 2: ")

2. Qual o valor da variável "x" ao final da execução do código a seguir supondo que o usuário digite o valor 10 na instrução *input*?

>>> x = input ("Digite um número")

>>> x = (x + 5) * 2 – 10

>>> print ("Resultado", x)

3. Leia dois textos usando a instrução *input* e salve-os nas variáveis *txt1* e *txt2*. Posteriormente, junte esses textos usando o operador "+".

4. Insira as expressões abaixo no interpretador *Python*. Posteriormente, verifique o tipo de cada uma delas usando *type.*

a) 5 + 10 – 6

b) 89 / 10 * (2 + 1)

c) 100 * “3”

d) 10 > 5 or 9 <= 30 and ‘a’ == ‘b’

e) 8.5 != 3 and 5 > 10

5. Dadas as variáveis numéricas “**x**”, “**y**” e “**z**”, contendo os valores 3, 5 e 7, respectivamente; a variável **tipo**, contendo o literal “TEXTO”; e a variável lógica **teste**, contendo o valor lógico falso(*False*), assinale abaixo qual a expressão lógica cujo resultado possui o valor lógico verdadeiro (*True*). Desenvolva a opção escolhida passo-a-passo.

( ) tipo == “TEXTO” **and** teste

( ) teste **or** x + y < z

( ) x – y > z **and** tipo == “NUMÉRICO”

( ) (x**3) – y > z **and** teste **or** tipo == “TEXTO”

6. Com o auxílio do interpretador *Python* calcule o valor das expressões abaixo.

a) 2 * 3 + 4

b) 2 * (3 + 4)

c) 2 * (3 + 4) + 3 * 5

d) 8 / 3 + 4 * 6 - 2 ** 5

CONHECENDO AS REFERÊNCIAS

MANZANO, J. A. N. G.; OLIVEIRA, J. F. **Algoritmos:** Lógica para Desenvolvimento de Programação de Computadores. São Paulo: Érica, 2014.

IBM. Descobrir Python, Parte 1: Tipos Numéricos Integrados da Python. [200-?] Disponível em: <http://www.ibm.com/developerworks/br/library/os-python1/>. Acesso em: 02 mar. 2015.

PYTHON BRASIL. Disponível em: <http://wiki.python.org.br/>. Acesso em: 05 fev. 2015.

PYTHON.ORG. Input and Output. [20--?]. Disponível em: <https://docs.python.org/3/tutorial/inputoutput.html>. Acesso em: 02 mar. 2015.

PYTHON SOFTWARE FOUNDATION. Disponível em: <https://www.python.org/>. Acesso em: 05 fev. 2015.

SEVERANCE, C. **Python for Informatics:** Exploring Information. 2013. Disponível em: <http://www.pythonlearn.com/book.php>. Acesso em: 05 fev. 2015.

Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 07

Estruturas de Decisão e Introdução às Estruturas de Repetição

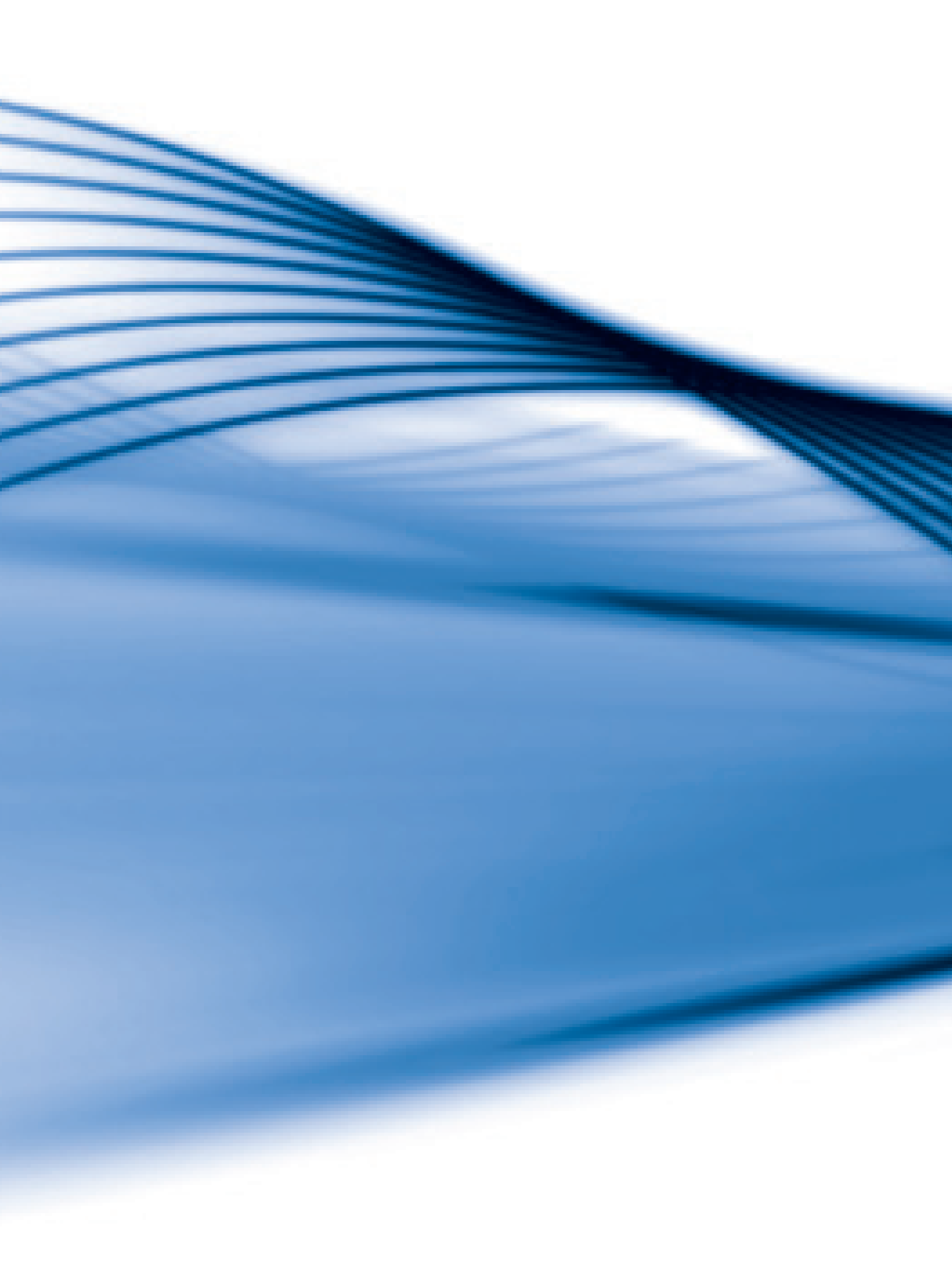
Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Olá, aluno(a), seja bem-vindo(a) a mais uma aula de programação em Python. Aprendemos até aqui a reconhecer e trabalhar com valores dos diversos tipos de dados e a armazenar esses valores em variáveis usando o sinal de atribuição (=). Vimos também como interagir com o usuário da sua aplicação através da leitura de valores com o teclado, usando o comando input e a imprimir mensagens que serão exibidas no monitor do computador através do comando print.

Nesta aula, vamos dar um passo adiante e aprender a trabalhar com estruturas que são extremamente importantes na programação de computadores. Iremos começar pelas estruturas de decisão, que como o nome sugere, permitem desviar o fluxo de execução do programa com base em uma decisão a ser tomada. Posteriormente, vamos começar a explorar a repetição de instruções. Juntas, essas duas estruturas permitem resolver uma vasta gama de problemas. Então, vamos nessa.



Aula 07 - Estruturas de Decisão e Introdução às Estruturas de Repetição.

Objetivos

Esta aula tem por objetivo apresentar as estruturas de decisão e realizar uma breve introdução às estruturas de repetição. Esse último tópico será explorado em maiores detalhes na aula seguinte. Ao final da aula, o aluno deve estar apto a:

- Compreender os princípios do desvio do fluxo de instruções de um programa através do uso das estruturas de decisão e repetição;
- Reconhecer e criar blocos de instruções;
- Resolver uma gama maior de problemas através do uso da estrutura de decisão IF (se) e suas variações.

Desenvolvendo o conteúdo

Controle do fluxo de execução do programa

Até o presente momento, nossos programas são formados por uma sequência de instruções simples que são interpretadas uma depois da outra, em um fluxo linear, sem nenhum desvio. Como ponto de partida, vamos resolver o seguinte problema: "Calcular o consumo médio de um veículo a partir da quantidade de combustível em litros que ele gasta para percorrer uma determinada distância em quilômetros (km)". Com o que sabemos até agora, é possível resolver esse problema apenas usando instruções em sequência. Primeiro, é preciso ler os dados de entrada do teclado, depois calcular a quantidade de combustível gasta e, por fim, gerar uma saída compreensível para o usuário do programa.

Pois bem, vamos resolvê-lo por partes. Primeiro, é preciso saber da distância percorrida pelo veículo e da quantidade de combustível gasta. Nesse caso, como essas informações podem mudar de acordo com cada usuário do programa, elas devem ser fornecidas por ele. Assim, devemos usar o comando *input* e guardar as informações em variáveis, conforme o Código 1. Do lado esquerdo do código, inserimos a numeração das linhas para facilitar a explicação do programa. Essa numeração não deve ser inserida no interpretador *Python*.

Código 1 – Consumo médio de combustível: leitura dos dados

1. >>> dist = float (input (“Distância percorrida: “))
2. Distância percorrida: 45.0
3. >>> comb = float (input (“Combustível gasto: “))
4. Combustível gasto: 3.2

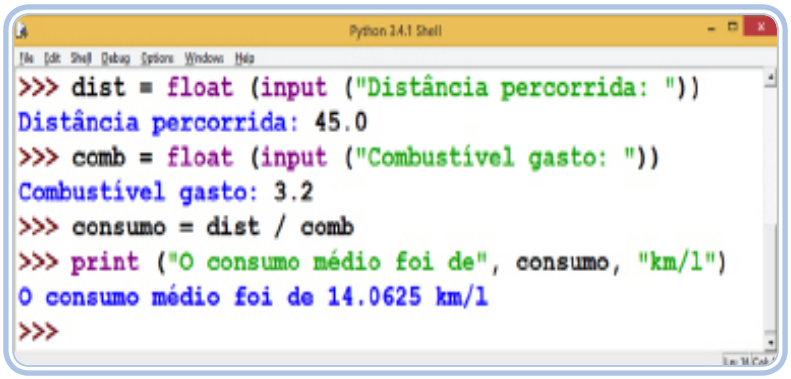
No código acima, na linha 1, a variável dist recebe a informação da distância percorrida pelo veículo. Ao pressionar o botão ENTER do teclado, o programa pede ao usuário que ele digite esse valor. Nesse caso, o valor inserido foi 45.0, representando 45.0 quilômetros. A quantidade de combustível gasta é armazenada na variável *comb*. Nesse caso, na linha 4, o valor lido foi 3.2, representando 3.2 litros de combustível.

Uma vez que já temos as informações que o programa necessita, estamos aptos a calcular a quantidade média de combustível gasta, que é o objetivo do programa. O Código 2 exibe esse cálculo e a informação do resultado do programa para o usuário.

Código 2 – Consumo médio de combustível: cálculo e impressão de resultados

1. >>> consumo = dist / comb
2. >>> print (“O consumo médio foi de”, consumo, “km/l”)
3. O consumo médio foi de 14.0625 km/l

O consumo médio do veículo é salvo na variável consumo (linha 1), sendo o resultado do cálculo da distância percorrida dividida pelo combustível consumido para percorrer essa distância (dist / comb). Após o cálculo, um comando print da linha 2 cria a mensagem que será exibida para o usuário do programa. O resultado dessa mensagem, incluído o consumo calculado é exibido na linha 3. Na Figura 1, é possível ver o código completo do programa conforme exibido no interpretador.



Fonte: Autoria própria.

Figura 01: Programa do consumo médio de combustível visto no interpretador Python

Esse exemplo ilustra um programa sequencial e linear. Esse programa precisa de apenas duas informações para realizar um cálculo simples e devolver um resultado ao usuário. Agora, prezado aluno(a), suponha que esse mesmo programa pudesse ter a possibilidade de escolher entre fornecer o resultado do cálculo em quilômetros por litro (km/l) ou em metros por litro (m/l). Nesse caso, precisaríamos tomar uma decisão para imprimir a mensagem ao usuário. Para desviar o fluxo do programa a partir de uma decisão, devemos usar uma estrutura da decisão.

Estruturas de decisão (ou seleção)

Olá, Certamente, você já deve ter tomado alguma decisão importante em sua vida, não é mesmo? Em verdade, tomamos pequenas decisões diariamente. Onde vamos almoçar hoje? Vamos sair para o cinema ou ficar em casa assistindo televisão? Hoje é dia de estudar programação ou matemática? Toda decisão tem por base um julgamento de valor.

Por exemplo, podemos decidir se vamos ao restaurante A ou B a partir da preferência pela comida de um deles. É possível decidir se vamos ao cinema com base na quantidade de dinheiro que nós temos. Esse dinheiro será sufi-

ciente para pagar o cinema e a pipoca? Ele não fará falta para pagar alguma conta?

Fonte: VALBER (2015)

Figura 02: Tirinha "Decisão"

Pois bem, em um programa de computador por diversas vezes temos que tomar uma decisão ou realizar um teste. Essa decisão ou teste irá guiar a execução de um trecho do código do programa a partir da análise do seu resultado. Por exemplo, é possível testar se o valor fornecido pelo usuário é um número antes de se realizar uma conta matemática. Ou podemos perguntar se um valor obedece a um certo critério, como a idade de uma pessoa ser maior ou igual a 18 anos.

Assim, estamos prontos para apresentar a forma básica da estrutura de decisão IF na linguagem Python. Essa estrutura também é conhecida como estrutura de seleção ou simplesmente "comando SE", sendo "se" o significado em português de *if*.

Estrutura de decisão IF ("se")

Permite executar um conjunto de comandos a partir da avaliação de uma condição. Se a condição for avaliada verdadeira, os comandos dentro do corpo do IF são executados. Caso contrário, os comandos dentro do corpo do IF não são executados. Em ambos os casos, a execução do programa prossegue após o comando IF.

A forma (sintaxe) do comando IF em Python é a seguinte:

```
if condição :

        comandos
```

Na instrução IF, a condição é uma expressão que deve devolver verdadeiro *(True)* ou falso *(False)* e *comandos* corresponde a zero ou mais comandos que devem ser executados caso a condição do IF seja avaliada verdadeira.

LEMBRE-SE!

Uma expressão é uma constante, uma variável ou uma combinação de variáveis, constantes e operadores que após ser avaliada devolve um valor para o usuário.

Antes de prosseguirmos com um exemplo, há algumas considerações a se fazer nesse momento. Primeiro, a palavra IF no comando deve sempre ser grafada em letra minúscula. Segundo, o sinal de dois pontos (":") após a condição marca o início de um bloco de comandos. Mas, o que vem a ser um bloco de comandos?

Bloco de comandos

Um bloco de comandos corresponde a um ou mais comandos que estão definidos dentro de uma determinada estrutura. Em Python, um bloco de comandos é delimitado por identação (ou endentação), ou seja, um recuo que você deve realizar através de tabulações ou espaços em branco. Essas tabulações ou espaços devem ser consistentes. Assim, um comando (ou instrução) que está no mesmo bloco do comando anterior deve estar na outra linha, mas alinhado com o seu antecessor através do mesmo número de tabulações ou espaços.

Pode parecer muita informação até o momento, mas vamos explicar em detalhes cada um desses conceitos através de um exemplo simples. O programa do Código 3 pede a leitura de dois números e informa qual deles é o maior. Esse programa foi adaptado de (MASANORI, 2015).

Código 3 – Exibe o maior entre dois números fornecidos pelo usuário

```
1. a = int ( input ("Primeiro número: ") )
2. b = int ( input ("Segundo número: ") )
3. if a > b :
4.     print ("O primeiro número é maior")
5. if b > a :
6.     print ("O segundo número é maior")
```

Nas linhas 1 e 2 do Código 3, temos a leituras dos dois números, armazenados nas variáveis "*a*" e "*b*", respectivamente. A linha 3 exibe uma decisão com base no valor das variáveis "a" e "b". Suponha que o usuário digitou 5 para o valor de "a" e 2 para o valor de "b".

Observe que, para esses valores fornecidos (a = 5 e b = 2), a expressão (a > b) na instrução IF da linha 3 devolve o valor verdade verdadeiro (*True*), pois 5 > 2 (5 é maior que 2). Assim, de acordo com a definição do IF, o programa irá para o bloco de comandos após o sinal de dois pontos (":"). Nesse caso, temos apenas um comando, a instrução *print* da linha 4.

Correto, mas como o interpretador *Python* sabe que a instrução faz parte do bloco de comandos do IF da linha 3? Ele sabe, pois a instrução está identada, ou se preferir, está recuada com uma tabulação (tecla TAB do seu teclado) ou com espaços em branco em relação a linha anterior, que corresponde ao início do comando IF. Assim sendo, a mensagem "*O primeiro número é maior*" é impressa e o programa prossegue analisando a instrução IF da linha 5. Veja que a condição de teste do IF da linha 5 é a expressão (b > a), que irá resultar no valor verdade falso (*False*). Nesse caso, como o teste falhou, a mensagem do corpo do segundo IF não é executada e o programa termina.

Teste com "senão"

No caso do exemplo do Código 3, nós temos duas condições que são complementares. Ou seja, se uma for satisfeita, a outra não precisa ser verificada. No caso, se "a" for maior que "b", então você não precisa testar com outro IF se "b" é maior que "a". Para esses casos, a estrutura IF pode ser usada com um senão (ELSE).

Estrutura de decisão IF ("se") com a parte ELSE ("senão")

A forma (sintaxe) do comando IF em com ELSE em Python é a seguinte:

```
if condição :
    comandos_verdadeiro
else :
    comandos_falso
```

Na instrução IF com a parte ELSE (IF-ELSE), a condição é uma expressão que deve devolver verdadeiro (*True*) ou falso (*False*) e *comandos_verdadeiro* corresponde a zero ou mais comandos que devem ser executados caso a condição do IF seja avaliada verdadeira. No caso da condição ser avaliada falsa (*False*), os comandos da parte ELSE são executados (*comandos_falso*).

O Código 4, a seguir, demonstra uma versão melhorada do código 3, agora inserindo o senão (ELSE) no comando IF. Nesse caso, é possível eliminar um teste (o segundo IF).

Código 4 – Maior entre dois números com IF-ELSE

```
1. a = int ( input ("Primeiro número: ") )
2. b = int ( input ("Segundo número: ") )
3. if a > b :
4.     print ("O primeiro número é maior")
5. else :
6.     print ("O segundo número é maior")
```

No caso do Código 4, se formos descrever em português o seu funcionamento a partir da linha 3, seria: Se "a" for maior que "b", então será impressa a mensagem "O primeiro número é maior", caso contrário (senão), será impressa a mensagem "O segundo número é maior".

ATIVIDADE

1. Leia a idade de uma pessoa e imprima uma mensagem dizendo se ela é maior ou menor de idade. Considere que uma pessoa maior de idade é aquela que tem 18 anos ou mais.

2. Tendo como entrada a altura em centímetros (p.ex. 1.70m = 170cm) e o sexo de uma pessoa ("m" para masculino ou "f" para feminino), calcule o seu peso ideal de acordo com a fórmula abaixo, conhecida como fórmula de Lorentz. Nessa fórmula, o valor "k" é igual a 4 para homens e 2 para mulheres.

$$peso = altura - 100 - ((altura - 150) / k)$$

É possível ter um teste dentro do outro, usando mais de um comando IF. Isso é possível, pois o bloco de comandos do IF (instruções identadas e contidas após o ":"), como qualquer outro comando de bloco na linguagem *Python*, aceita qualquer tipo de instruções válidas na linguagem, até mesmo outros IFs.

Para exemplificar, suponha que queremos resolver o problema de saber se um número é par ou ímpar. Nesse caso, há uma restrição a mais, pois o número tem que ser maior que zero. Então, antes de testar se o número é par ou ímpar, temos que testar (inserir um IF) para saber se o número é maior que zero. A solução para este problema pode ser vista no Código 5.

Código 5 – Maior entre dois números com IF-ELSE

```
1. x = int (input ("Digite um número inteiro positivo (>0): "))
2. if x > 0 :
3.     if x % 2 == 0 :
4.         print ("O número", x, "é par!")
5.     else :
6.         print ("O número", x, "é ímpar!")
7. else :
8.     print ("O número deve ser maior que 0!")
```

Observe que o IF mais interno (linha 3) está identado em relação ao IF mais externo (linha 2) que verifica se o número é maior que zero. Nesse caso, o IF da linha 3 é parte do corpo do IF da linha 2 e será executado apenas se a condição de teste do IF da linha 2 devolver o valor verdadeiro. Caso contrário, a mensagem "O número deve ser maior que 0!" será exibida para o usuário.

No caso do número ser maior que zero, o teste do IF da linha 3 é feito. Esse

teste verifica se o resto da divisão do número lido (x) por 2 é igual a zero (x % 2 == 0). Caso essa expressão seja verdade (*True*), então o número é par, pois o número é divisível por 2, e a mensagem da linha 4 é exibida para o usuário. Caso contrário, é exibida a mensagem da parte ELSE, na linha 6, dizendo que o número é ímpar.

Testes aninhados

Outra situação que pode ocorrer é quando temos vários casos de testes aplicados a uma situação e apenas um deles deve ser executado. Nesse caso, teríamos vários testes, um dentro do outro. Por exemplo, suponha que em uma competição de natação, os nadadores sejam classificados em diversas categorias, de acordo com a sua idade. Podemos criar um programa que receba a idade de um nadador e imprima a qual categoria ele pertence. Nesse caso, temos que testar cada caso, um por vez, até que se encontre a categoria na qual o nadador se enquadra. Assim, é necessário um IF para cada categoria, conforme pode ser visto no Código 6.

Código 6 – Imprime a categoria em uma competição de natação dada a idade do nadador

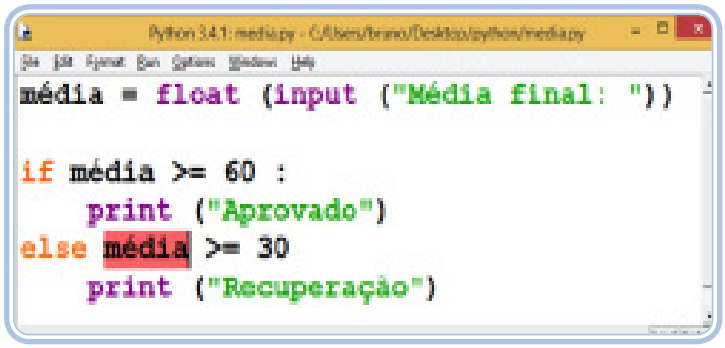
```
1. idade = int (input (“Idade: “))
2. if idade >= 5 and idade <= 7 :
3.     print (“Infantil A”)
4. else :
5.     if idade >= 8 and idade <= 10 :
6.         print (“Infantil B”)
7.     else :
8.         if idade >= 11 and idade <= 13 :
9.             print (“Juvenil A”)
```

```
10.         else :
11.             if idade >= 14 and idade <= 17 :
12.                 print (“Juvenil B”)
13.             else :
14.                 print (“Não existe categoria para a idade informada!”)
```

Prezado aluno, espero que você não tenha se assustado com o Código 6. Em verdade, apesar de um pouco extenso, ele é simples. Para cada intervalo de idade em cada categoria é feito um teste. Se a idade informada estiver dentro de uma das categorias, então a mensagem do IF correspondente é impressa. Caso contrário, a mensagem no corpo do último ELSE é impressa, informando ao usuário que não existe categoria para a idade informada.

Observe que no exemplo do Código 6 as expressões estão em pares, conectadas por um “and”, como no caso de “idade >= 8 and idade <= 10”. Isso é necessário, pois o operador de maior ou igual (>=), assim como os demais operadores de comparação, pode(m) receber apenas dois operandos, no caso a variável idade e uma constante inteira (p.ex.: idade >= 8). Não é possível fazer como na matemática, cuja expressão seria algo da forma (8 <= idade <= 10). Assim, para dizer que as duas condições têm que ser satisfeitas, nós usamos o operador “and”, que irá devolver o valor verdadeiro apenas se a idade estiver dentro do intervalo estabelecido.

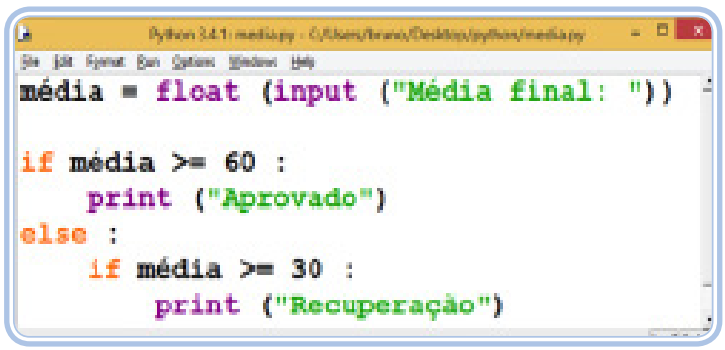
Outra situação comum que pode levar o usuário a erro, é tentar fazer um teste com a parte ELSE. É importante frisar que isso não é possível, conforme está destacado no programa da Figura 3, que diz se um aluno foi aprovado ou ficou em recuperação, dada a sua média. Isso leva a um erro de sintaxe na linguagem, pois esse tipo de construção não existe. Portanto, sempre que você precisar realizar um teste, ele deve ser feito em um IF. Por isso, no caso de um teste ser falso, na parte ELSE inserimos um outro IF para testar o próximo caso.



Fonte: Autoria própria.

Figura 03: ERRO: Tentando inserir um teste na parte ELSE

A versão correta do programa que exibe o resultado da média pode ser vista na Figura 4. Nesse caso, foi inserido mais um IF para realizar o teste que verifica se a média é maior ou igual que 30, situação em que o aluno fica em recuperação.



Fonte: Autoria própria.

Figura 04: Versão correta do programa da média, com um teste a mais inserido em um IF

Esses casos em que temos vários testes em sequência, relacionados entre si, sendo que apenas um deve ser executado, são chamados de testes aninhados. Para esse tipo de estrutura, a linguagem Python possui a palavra ELIF que abrevia a parte ELSE IF (senão-se), tornando o código menor e mais legível. Assim sendo, é possível obter no Código 7 a mesma funcionalidade apresentada no Código 6 usando ELIF com o ganho do código ficar mais enxuto e organizado para o programador.

Código 7 – Imprime as categorias em uma competição de natação (versão com ELIF)

1. idade = **int** (**input** (“Idade: “))

2. **if** idade >= 5 **and** idade <= 7 :

```
3.      print ("Infantil A")
4.  elif idade >= 8 and idade <= 10 :
5.      print ("Infantil B")
6.  elif idade >= 11 and idade <= 13 :
7.      print ("Juvenil A")
8.   elif idade >= 14 and idade <= 17 :
9.      print ("Juvenil B")
10. else :
11.     print ("Não existe categoria para a idade informada!")
```

Introdução à Repetição

Prezado aluno(a), com o repertório de técnicas e elementos da linguagem Python que vimos até agora já somos capazes de criar diversos programas que envolvam cálculos e tomada de decisões com base no valor de expressões.

Pois bem, a partir deste momento vamos iniciar no estudo de uma nova instrução que permite repetir trechos de código do nosso programa. Nesta aula, nós vamos apenas contextualizar o uso da instrução de repetição para que possamos detalhá-la na próxima aula.

Você pode estar se perguntando: mas como assim repetir código do programa? Como isso pode ser de fato aplicado? Isso não apenas pode ser aplicado como é uma ferramenta extremamente útil, que ajuda a evitar a repetição de código e possibilita resolver uma gama muito maior de problemas.

Para entender a repetição, suponha o seguinte problema: seu professor lhe pediu para que você calculasse a média aritmética das notas da sua turma na disciplina de programação. Se a turma for composta por 8 alunos, a média corresponde a soma das notas dividido por 8. Com o que vimos até

agora, como poderíamos ler todas as notas e efetuar o cálculo? Simples, nós teríamos que criar 8 variáveis, para armazenar cada nota antes de efetuar o cálculo. O código 3 demonstra a solução desse problema.

Código 8 – Média das notas de uma turma de 8 alunos

```
1.  nota1 = float (input ("Nota1: "))
2.  nota2 =  float (input ("Nota2: "))
3.  nota3 =  float (input ("Nota3: "))
4.  nota4 =  float (input ("Nota4: "))
5.  nota5 =  float (input ("Nota5: "))
6.  nota6 =  float (input ("Nota6: "))
7.  nota7 = float (input ("Nota7: "))
8.  nota8 = float (input ("Nota8: "))
9.  média=(nota1 + nota2 + nota2 + nota4 + nota5 + nota6 + nota7 +
    nota8) / 8
10. print ("Média da turma=", média)
```

Na solução do Código 9, a soma das notas foi feita logo após a leitura de cada nota. Na linha 1 desse programa, iniciamos a soma com 0. Isso foi feito devido ao papel da variável "soma" como acumulador de uma soma de vários valores, no caso as notas dos alunos. Assim, na leitura da primeira nota, a soma recebe o valor dessa nota, uma vez que a expressão fica soma = 0 + nota1, o que resulta em soma = nota1. Na linha 5, a soma recebe o valor que ela tinha antes (a nota 1), mais o valor lido para a segunda nota, e assim por diante.

Perceba que o processo de receber um valor e acumular esse valor na variável soma é bastante semelhante, da linha 2 até a linha 17. Na próxima aula, vamos ver como realizar esse mesmo programa com apenas uma variável para receber a nota dos alunos e repetir, automaticamente, esses trechos de

código para realizar o acúmulo da soma das notas na variável "soma". Para tanto, vamos ver em detalhes e como mais exemplos a estrutura de repetição while (enquanto).

RESUMINDO

Prezado aluno, nesta aula detalhamos a estrutura de decisão "se". Ela é utilizada sempre que precisamos fazer um teste sobre uma determinada expressão ou simplesmente, como o próprio nome sugere, tomar uma decisão sobre qual caminho o programa deve seguir. Em Python, a estrutura se corresponde ao comando IF. Exibimos a versão do IF com e sem a parte ELSE. No caso do IF com ELSE, os comandos que estão nessa última parte são feitos apenas se o teste da parte IF devolver o valor falso (false). Mostramos também como testar diversas condições complementares usando o IF com o ELIF, que significa ELSE-IF. Você deve lembrar também a importância de identar o código sempre que estiver em uma estrutura de bloco, tal como o IF ou ELSE. Identar o código corresponde a inserir uma tabulação (tecla TAB do teclado) ou uma certa quantidade de espaços em branco de modo que todos os comandos em um bloco fiquem na mesma identação. Isso faz com que o comando pertença aquele bloco. Por fim, fizemos uma breve introdução para motivar o uso de repetição de trechos de código, assunto esse que será detalhado na próxima aula.

LEITURAS COMPLEMENTARES

MELANDA, J. C. E. Blocos de código em Python. 2011 (http://programeempython.blog.br/blog/blocos-de-codigo-em-python/). Nesta página, você pode encontrar mais informações sobre a identação de código para estabelecer os blocos de código em Python.

D! TUTORIAIS (https://dtutoriais.wordpress.com/2010/07/20/aprendendo-python-4-estruturas-de-decisao/) – Nesta página, você pode ler mais a respeito das estruturas de decisão em Python e suas variações.

AVALIANDO SEUS CONHECIMENTOS

1. Suponha que um funcionário ganha R$ 800,00 por semana por 30 horas de trabalho. Esse mesmo funcionário recebe R$150,00 para cada hora extra de trabalho durante a semana, sendo que ele pode trabalhar no máximo até 40 horas. Faça um programa que receba a quantidade de horas trabalhas por um funcionário (horas >= 30 e horas <= 40) e informe o seu salário final naquela semana.

2. Faça um algoritmo para calcular a área de um círculo, dado o valor do seu raio, que deve ser positivo (maior que 0). A fórmula da área do círculo é:

área = π * r2 , onde π = 3.14 e "r" é o raio.

3. Faça um programa em Python para calcular a média parcial de um aluno em uma disciplina semestral de acordo com as instruções a seguir.

- O programa deve informar se o estudante foi aprovado, ficou em avaliação final ou foi reprovado;
- O aluno é considerado aprovado se obtiver média igual ou superior a 60 pontos;
- O aluno deve ir para a prova final caso a sua média seja maior ou igual a 30;
- O aluno está reprovado se a sua média for menor que 30;
- A fórmula para o cálculo da média está descrita abaixo. A variável n1 representa a nota do primeiro bimestre e n2 a nota do segundo bimestre.

$$Média = \frac{2n_1+3n_2}{5}$$

4. Faça um programa que calcule o que deve ser pago por um produto, considerando o preço normal de venda e a escolha da condição de pa-

gamento. Utilize os códigos da tabela abaixo para ler qual a condição de pagamento escolhida (1, 2 ou 3) e efetuar o cálculo do valor do preço final do produto.

5.

Código	Condição de Pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto.
2	À vista no cartão de crédito, 5% de desconto.
3	Em 2 vezes, preço normal de venda, sem juros.

6. Determinar o tipo de um triângulo dados os valores dos seus três lados. Você deve inserir testes de acordo com as expressões abaixo para verificar se os valores fornecidos para os lados formam um triângulo e, caso formem, escrever qual o tipo do triângulo. Nas expressões abaixo, "a", "b" e "c" representam os lados do triângulo.

- **Triângulo**: Figura geométrica de três lados, em que cada lado é menor que a soma dos outros dois.

  (a < b + c) **and** (b < a + c) **and** (c < a + b)

- **Triângulo equilátero**: Os três lados são iguais.

  (a == b) **and** (b == c)

- **Triângulo isósceles**: Apenas dois lados são iguais.

  (a == b) or (a == c) or (b == c)

- **Triângulo escaleno**: Todos os lados são diferentes.

  (a != b) **and** (b != c) **and** (c != a)

CONHECENDO AS REFERÊNCIAS

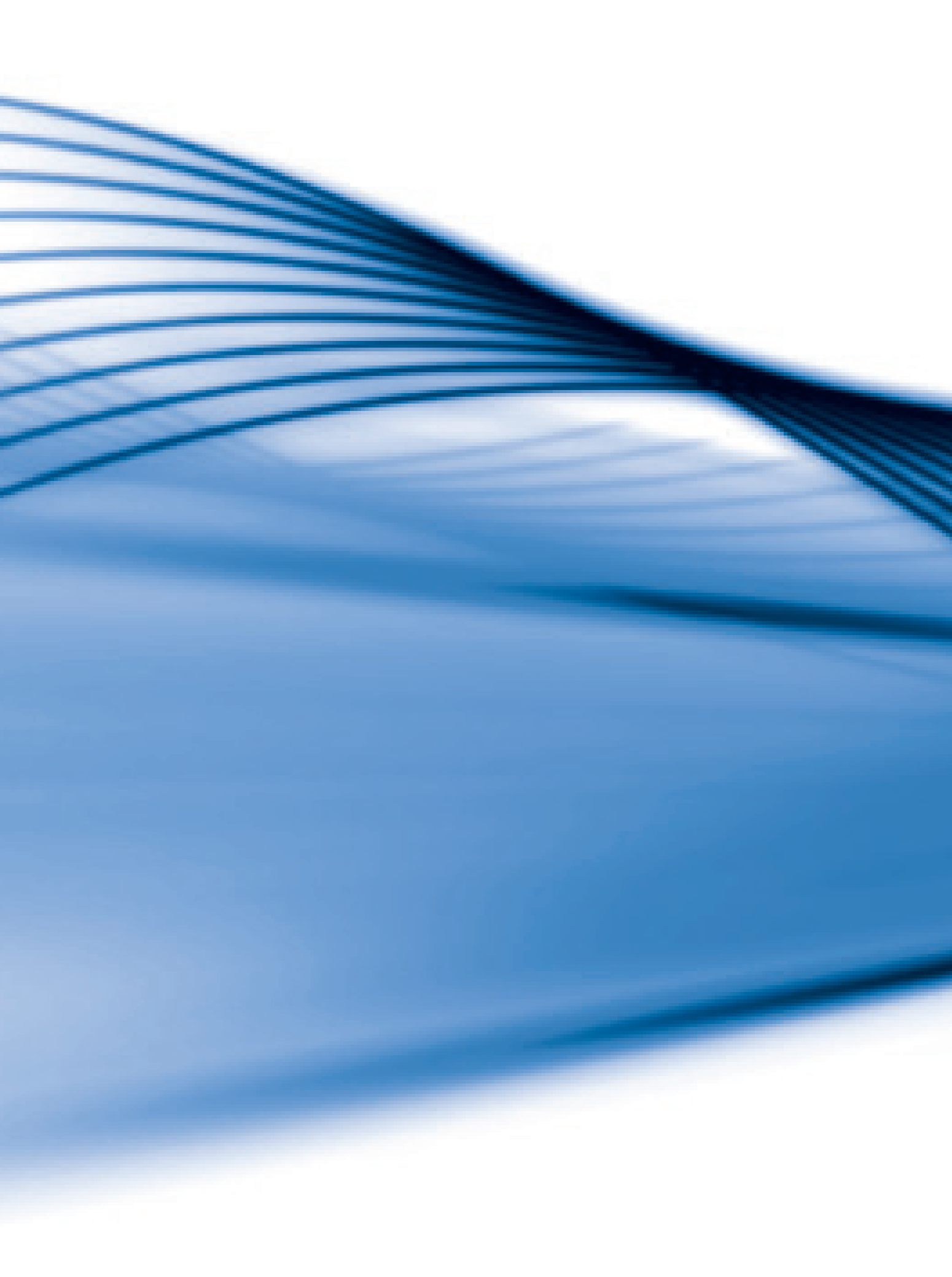
D! TUTORIAIS. **Aprendendo Python 4**: Estruturas de Decisão. Disponível em: <https://dtutoriais.wordpress.com/2010/07/20/aprendendo-python-4--estruturas-de-decisao/>. Acesso em: 12 mar. 2015

MASANORI, F. **Python para Zumbis**. Disponível em: <http://pycursos.com/python-para-zumbis/>. Acesso em: 10 mar. 2015.

MELANDA, J. C. M. **Blocos de Código em Python**. Disponível em: <http://programeempython.blog.br/blog/blocos-de-codigo-em-python/>. Acesso em: 12 mar. 2015.

PYTHON SOFTWARE FOUNDATION. Disponível em: <https://www.python.org/>. Acesso em: 10 mar. 2015.

SEVERANCE, C. **Python for Informatics**: Exploring Information. 2013. Disponível em: <http://www.pythonlearn.com/book.php>. Acesso em: 12 mar. 2015.



Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 08
Estruturas Repetição e Listas

Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte
Natal-RN
2015

Apresentação da disciplina

Prezado(a) aluno(a), na aula anterior vimos conceitos e instruções bastante importantes para a programação. Destaco inicialmente os blocos de comandos. Um bloco é composto por um ou mais comandos que fazem parte de alguma instrução e devem ser executados em sequência. Podemos ter blocos, por exemplo, na instrução IF (se). O início de um bloco é marcado por ":" (dois pontos).

É importante que você lembre que todas as instruções que fazem parte de um bloco de devem ser identadas. Para identar uma instrução, você deve usar a tecla TAB do teclado ou inserir uma quantidade de espaços em branco. Isso faz com que os comandos em um bloco fiquem recuados mais à direita. Observe que se há mais de uma instrução em um bloco, cada uma deve ser inserida em uma linha diferente e no mesmo alinhamento de texto (identação) que a instrução anterior.

Na aula anterior, exploramos a decisão com IF e suas variações com ELSE (senão) e ELSIF (senão-se). É importante lembrar que o IF deve ser usado sempre que precisamos tomar uma decisão, ou simplesmente fazer um teste. Nesta aula, vamos aprender a uma nova instrução que irá economizar muito esforço de programação e permitir resolver mais problemas. Trata-se da repetição, que pode ser feita por meio dos comandos WHILE (enquanto) e FOR (para). Além disso, vamos trabalhar com listas de informações dos mais diversos tipos.

Aula 8 - Estruturas Repetição e Listas

Objetivos

Objetivo desta aula é conhecer em detalhes as estruturas de repetição e trabalhar com listas. Dessa forma, ao final da aula, o aluno deve estar apto a:

Compreender os princípios e técnicas envolvidas na repetição de instruções;

Conhecer e aplicar as estruturas WHILE e FOR para realizar a repetição;

Aprender a criar listas e usar algumas funções da linguagem *Python* para obter informações sobre uma lista e facilitar a sua manipulação no programa.

Desenvolvendo o conteúdo

Algoritmos e Programação de computadores

Na aula anterior, foi realizada uma breve introdução para motivar o uso de estruturas de repetição. Apresentamos o programa do cálculo da média das notas de uma turma de 8 alunos (Código 1). Esse é um tipo de programa ideal para aplicarmos uma repetição, pois há um padrão que "se repete" em várias instruções.

Fonte: https://goo.gl/kOOYjJ

Figura 1: Repetição

Observe com atenção o Código 1. Nas linhas 1 a 17 é possível notar que há um padrão repetitivo. A cada linha, uma nova nota é lida e somada às notas lidas anteriormente. No caso, usamos a variável "soma" para fazer esse papel de acumulador (ou "somador") de notas. Na linha 1 ela é iniciada com o valor 0 (zero), indicando que nenhuma nota foi lida. Na linha 2 a primeira nota é lida e salva na variável "*nota1*". Esse valor é adicionado à variável soma na linha 3 (*soma* = 0 + *nota1*). De modo semelhante, na linha 4 é lida a segunda nota e na linha 5 é feita a inclusão dessa nota à variável soma. Nesse momento "soma" contém a soma da primeira e da segunda nota. Esse processo se repete até que a oitava nota seja adicionada. Sendo assim, é possível calcular a média da turma, que é a soma das notas lidas (o valor acumulado na variável "soma") dividido pela quantidade de notas (linha 18).

Código 1 – Média das notas de uma turma de 8 alunos

```
1.  soma = 0
2.  nota1 = float (input ("Nota1: "))
3.  soma = soma + nota1
4.  nota2 = float (input ("Nota2: "))
5.  soma = soma + nota2
6.  nota3 = float (input ("Nota3: "))
7.  soma = soma + nota3
8.  nota4 = float (input ("Nota4: "))
9.  soma = soma + nota4
10. nota5 = float (input ("Nota5: "))
11. soma = soma + nota5
12. nota6 = float (input ("Nota6: "))
```

```
13. soma = soma + nota6

14. nota7 = float (input ("Nota7: "))

15. soma = soma + nota7

16. nota8 = float (input ("Nota8: "))

17. soma = soma + nota8

18. média = soma / 8

19. print ("Média da turma=", média)
```

Vamos, neste momento, reescrever este programa usando uma estrutura de repetição que irá facilitar bastante o nosso trabalho. Trata-se da estrutura WHILE (ou, em português, "enquanto").

Estrutura de repetição WHILE ("enquanto")

Permite repetir um bloco de código um certo número de vezes determinado pela avaliação de uma expressão (condição).

A forma (sintaxe) do comando WHILE em Python é a seguinte:

```
While condição;

    comandos
```

Na instrução WHILE, a condição é uma expressão que deve devolver verdadeiro (*True*) ou falso (*False*). Os comandos (ou *instruções*) dentro do corpo da estrutura WHILE são executados (ou repetidos) ENQUANTO a condição for verdadeira (*True*). Quando a condição se torna falsa (*False*), os comandos não são executados e o programa continua depois da instrução WHILE.

LEMBRE-SE

Um bloco de código corresponde a uma ou mais instruções da linguagem que estão relacionadas dentro de uma mesma estrutura. O início do bloco é determinado por ":" (sinal de dois pontos) e o restante do bloco corresponde a instruções em uma mesma identação (alinhadas por uma mesma tabulação ou o mesmo número de espaços).

O Código 2 a seguir exibe a versão do programa da média das notas de 8 (oito) alunos usando a instrução *while*.

Código 2 – Média das notas de uma turma de 8 alunos com WHILE

```
1. soma = 0
2. quantidade = 1
3. while quantidade <= 8:
4. nota = float (input ("Nota " + str(quantidade) + ": "))
5. soma = soma + nota
6. quantidade = quantidade + 1
7. média = soma / 8
8. print ("Média da turma=", média)
```

Você deve ter percebido algo interessante. O Código 2, que usa repetição, é muito menor que o Código 1. Pois bem, essa é uma das vantagens de se usar a estrutura de repetição para esse tipo de problema. Todo o código que tinha um padrão para ser repetido, no caso a leitura de cada uma das oito notas e a soma delas, foi inserido dentro do bloco da estrutura WHILE (repita).

Vamos passear pelo código. As linhas 1 e 2 fazem a inicialização das variáveis *soma* e *quantidade*, respectivamente. *Soma* recebe o valor zero, pois ela será um acumulador de notas. Sendo assim, o valor zero representa que, naquele momento, nenhuma nota foi adicionada à variável. Por sua vez, a *quantidade* tem o papel de estabelecer a condição do WHILE, determinando assim quantas repetições serão feitas. Ela recebe na linha 2 o valor 1 e terá que ser atualizada até o valor 8, desse modo, permitindo repetir o bloco do WHILE oito vezes, que é o que precisamos para ler e somar as notas dos oito alunos.

Quando o programa chega na linha 3, é feito o teste da expressão do WHILE para determinar se as instruções em seu corpo serão executadas. No caso, a expressão (quantidade <= 8) retorna verdadeiro, pois ao substituir o valor que a variável quantidade tem no momento, o número 1, temos como resultado a expressão 1 <= 8 (1 é menor ou igual que 8). Desse modo, sendo a expressão verdadeira, as instruções das linhas 4 a 6 serão executadas.

A linha 4 do programa faz a leitura da nota. Se for a primeira vez que o programa estiver executando, será a primeira nota. Perceba que no corpo do *input* temos três textos que são unidos com o operador +. Esse operador, no caso de textos, serve exatamente para juntá-los, formando assim um só texto. No caso, o objetivo foi gerar a frase "*Nota 1*:", "*Nota 2*:" e assim por diante, dependendo do valor da variável quantidade.

A linha 5 acumula o valor de cada nota lida na variável *soma* e a linha 6 atualiza o valor da variável *quantidade*. Perceba que essa atualização é essencial, pois sem ela o seu programa iria ficar preso no WHILE para sempre, uma vez que a expressão de teste seria sempre verdadeira (1 <= 8). Você pode fazer esse teste rodando o seu programa sem a instrução da linha 6.

Nesse programa, atualizamos a variável quantidade de 1 em 1 (*quantidade* = *quantidade* + 1), até 8, pois queremos repetir oito vezes. Na primeira vez que essa instrução for executada, ela terá o seu valor atualizado para 2.

Nesse momento vemos a repetição acontecer. Após atualizar o valor da variável quantidade, que é o último comando do corpo do WHILE (perceba pela identação), o programa volta para a linha 3, para testar novamente a expressão *quantidade* <= 8. Como quantidade foi atualizada para 2 (dois), então a expressão resultante 2 <= 8 retorna verdadeiro e as instruções das linhas 4 a 6 são executadas novamente, dessa vez para ler a segunda nota.

Esse processo de verificar a condição do WHILE, executar as instruções dentro do WHILE, atualizar a variável quantidade e voltar para o teste do WHILE se repete até que o valor de quantidade seja 9. Nesse momento, a expressão 9 <= 8 devolve o valor falso (*False*) e o programa sai do WHILE e vai para a próxima instrução após ele, na linha 7, que irá calcular a média. Por fim, o programa termina na linha 8, com a impressão na tela do resultado da média.

Enfim, descrevemos o nosso programa completo usando a repetição WHILE. Há alguns pontos que você precisa se concentrar para aprender a usar o WHILE em diversas situações. Primeiro, você deve sempre criar uma expressão de teste que em algum momento termine. Não há necessariamente uma receita para isso, depende do problema que você quer resolver. No nosso caso, usamos a variável quantidade para fazer esse controle da expressão e é por isso que esse tipo de variável é chamado de variável de controle da repetição. Inicializamos com o valor 1 e criamos a expressão (quantidade <= 8) para irmos de 1 a 8, repetindo assim 8 vezes, uma vez que quantidade é atualizada em 1 a cada repetição.

Perceba que podemos muito bem inicializar a variável quantidade com o valor 0 (zero) e repetir até 7, alterando a expressão de teste para *quantidade* < 8 ou, ainda, *quantidade* <= 7. Isso irá gerar os valores 0, 1, 2, 3, 4, 5, 6, 7, fazendo com que ainda tenhamos 8 repetições. Nós tomamos a decisão de inicializar quantidade com 1 apenas para permitir a impressão dos textos "Nota 1:", "Nota 2:", no comando *input* que pede a entrada das notas.

Outro ponto que você deve prestar atenção é na atualização da variável de controle, pois é ela que faz com que em algum momento o WHILE termine, quando ela se tornar falsa (*False*). Geralmente, essa expressão é a última instrução do WHILE, mas isso não precisa ser seguido à risca. Caso você esqueça de atualizar a variável de controle, a repetição e, consequentemente, o seu programa, nunca irão terminar.

O funcionamento da estrutura WHILE pode ser resumido a partir do diagrama da Figura 2. Na primeira etapa, você deve fornecer o valor inicial da variável de controle. Isso ocorre apenas uma vez, antes de chegarmos a etapa de teste da expressão do WHILE. Nessa etapa, por sua vez, é feito o teste para saber se a repetição deve ser ou não realizada. Caso a expressão devolva o valor verdade verdadeiro (True), as instruções no corpo do WHILE são realizadas. Uma dessas instruções geralmente é a atualização da variável

de controle. Após essa atualização, o teste é feito novamente. Esse processo se repete até que a expressão de teste se torne falsa e o programa vai para a próxima instrução após o WHILE.

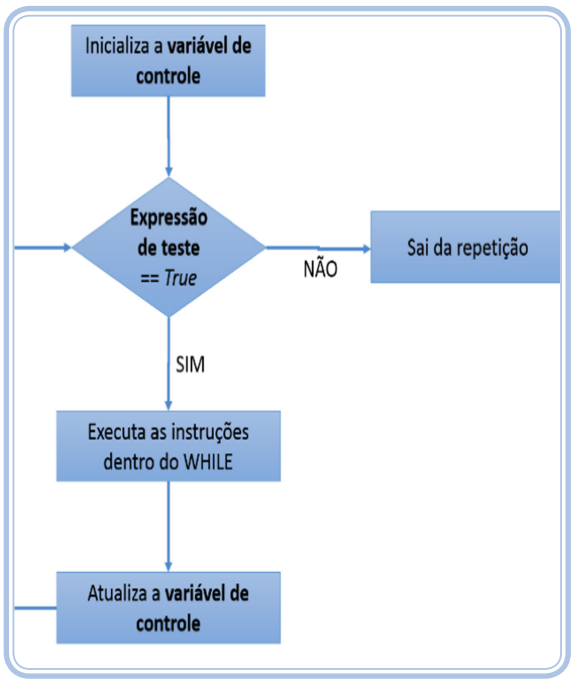


Figura 2: Funcionamento da estrutura WHILE

Antes de você tentar resolver seus próprios problemas com o WHILE, vamos praticar um pouco mais com alguns exemplos. Suponha que você queira imprimir todos os números pares de 2 até 100, incluindo 2 e 100. Pois bem, vamos criar juntos essa repetição. Primeiro, qual seria o valor inicial da repetição? Se queremos ir de 2 até 100, então o valor inicial é 2. Suponha que a variável de controle se chama "par". Nesse caso, ela deve receber o inteiro 2, o primeiro par do intervalo (*par* = 2). O valor final é 100. Sendo assim, nossa expressão de teste deve ir até 100 (*par* <= 100). Agora falta apenas trabalharmos com a atualização da variável "par", que deve crescer de 2 em 2, até chegar em 100 (*par* = *par* + 2).

Código 3 – Impressão de todos os pares de 2 a 100

```
1. par = 2

2. while par <= 100:

3. print (par)

4. par = par + 2
```

Vamos mudar um pouco o Código 3, de modo que o valor final do intervalo, que antes era 100, agora deve ser fornecido pelo usuário. Ou seja, você deve pedir a leitura desse valor usando *input*. É importante que você verifique se o valor lido está correto. Nesse caso, o valor superior deve ser maior que 2, que é o valor inicial. Para tanto, é necessário apenas acrescentar um teste antes do WHILE, conforme pode ser visto no Código 4.

Código 4 – Impressão de todos os pares de 2 a um valor superior (lim)

```
1. lim = int (input ("Limite superior: "))
2. par = 2
3. if lim > 2:
4. while par <= lim:
5. print (par)
6. par = par + 2
7. else:
8.     print ("O limite superior deve ser maior que 2.")
```

Até o momento, nós usamos os operadores de comparação na expressão de teste do WHILE. Na verdade, você pode usar qualquer expressão e operadores correspondentes que devolvam verdadeiro ou falso. Por exemplo, é possível testar a igualdade (==) ou a diferença (!=) de um determinado valor. É possível deixar a variável de controle verdadeira (*True*) e em algum momento torná-la falsa (*False*). Por exemplo, o Código 5 exibe um pequeno jogo que pede para o usuário adivinhar o número que foi gerado pelo programa (entre 0 e 20). Esse número é gerado com a função *randint*.

Código 5 – Adivinhe um número (entre 0 e 20)

```
1. from random import *
2. lido = int (input ("Digite um número entre 0 e 20: "))
```

```
3. gerado = randint (0, 21) #randint: gera um número inteiro entre 0 e 20

4. while gerado != lido:

5.     if lido > gerado:

6.         print ("O número é menor que", lido, ". Tente novamente!")

7.     else:

8.         print ("O número é maior que", lido, ". Tente novamente!")

9.

10.    lido = int (input ("Digite um número entre 0 e 20: "))

11. print ("Você acertou. Parabéns!")
```

Na primeira linha do programa de adivinhação de um número, temos a inclusão de uma biblioteca de funções, denominada *random*. Uma função nada mais é que um trecho de código que executa uma ação específica. Já trabalhamos com funções, como *print* e *input*. A linha 1 é necessária para acessarmos a função *randint*, que recebe dois valores, o primeiro é o valor inicial do intervalo no caso 0 e o segundo é o valor final menos 1. Ou seja, se queremos que a função gere e devolva um número qualquer entre 0 e 20, temos que colocar, como na linha 3, *randint*(*0, 21*).

Na linha 2 é feita a leitura do número digitado pelo usuário do programa. Esse valor será comparado com o valor gerado pela função *randint* na linha 3. Ainda na linha 3, o texto após o sinal de "#" é um comentário, ele tem a função apenas de documentar o programa, tornando mais claro o significado de algum trecho de código para quem está lendo o programa.

O WHILE da linha 4 irá repetir enquanto o valor digitado pelo usuário for diferente do valor gerado, ou seja, enquanto o usuário não adivinhar o número gerado. Para ajudar o jogador, é inserido um teste na linha 4. Ele vai dizer se o número digitado é maior ou menor que o número lido. A última instrução do WHILE pede novamente a leitura, até que o usuário digite o mesmo número que foi gerado. Quando isso acontecer, o teste do WHILE será falso e será impressa a mensagem da linha 11 indicando que o jogador acertou o número.

Na primeira linha do programa de adivinhação de um número, temos a inclusão de uma biblioteca de funções, denominada random. Uma função nada mais é que um trecho de código que executa uma ação específica. Já trabalhamos com funções, como print e input. A linha 1 é necessária para acessarmos a função randint, que recebe dois valores, o primeiro é o valor inicial do intervalo no caso 0 e o segundo é o valor final menos 1. Ou seja, se queremos que a função gere e devolva um número qualquer entre 0 e 20, temos que colocar, como na linha 3, randint(0, 21).

Na linha 2 é feita a leitura do número digitado pelo usuário do programa. Esse valor será comparado com o valor gerado pela função randint na linha 3. Ainda na linha 3, o texto após o sinal de "#" é um comentário, ele tem a função apenas de documentar o programa, tornando mais claro o significado de algum trecho de código para quem está lendo o programa.

O WHILE da linha 4 irá repetir enquanto o valor digitado pelo usuário for diferente do valor gerado, ou seja, enquanto o usuário não adivinhar o número gerado. Para ajudar o jogador, é inserido um teste na linha 4. Ele vai dizer se o número digitado é maior ou menor que o número lido. A última instrução do WHILE pede novamente a leitura, até que o usuário digite o mesmo número que foi gerado. Quando isso acontecer, o teste do WHILE será falso e será impressa a mensagem da linha 11 indicando que o jogador acertou o número.

ATIVIDADE

1. Faça um programa, usando repetição, que imprima os números de 1 até 30.

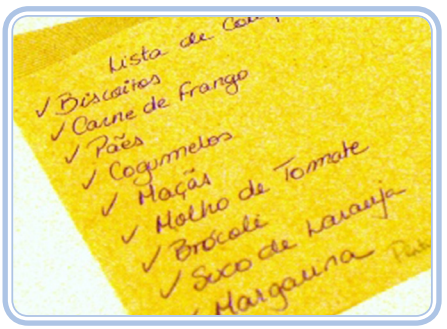
2. Modifique o programa da questão 1 e imprima os números de 1 até um valor maior que 1. Esse valor deve ser fornecido pelo usuário (use a instrução *input* para ler o valor).

3. Faça um programa que leia os pontos dos últimos 10 jogos de um time de futebol e imprima o total de pontos do time nesses 10 jogos. Ou seja, você deve imprimir a soma de todos os pontos lidos. Os pontos podem ser 3 (três), em caso de vitória, 1 (um) em caso de empate e 0 (zero) no caso de derrota.

Listas

Após apresentar a estrutura de repetição WHILE, vamos iniciar o estudo de uma nova estrutura de dados, a lista. Não se preocupe, que o estudo anterior também irá se aplicar aqui, pois precisamos das estruturas de repetição para percorrer as listas.

Trabalhar com listas nos permite resolver vários problemas. Você pode criar listas de cada um dos tipos básicos e até mesmo de outras listas. Mas, o quem vem a ser uma lista? Ora tenho certeza que você já criou uma lista alguma vez na vida. Podemos citar diversos exemplos, como uma lista de compras no supermercado (Figura 3), a lista dos seus filmes favoritos, a lista das matérias que você precisa estudar mais.



Fonte: http://goo.gl/1IFhp0

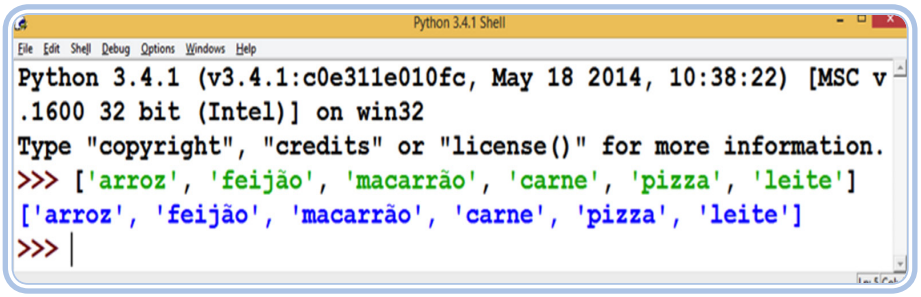
Figura 3: Lista de compras

Em *Python*, a lista é uma estrutura muito flexível e bastante simples de criar e usar.

Lista

Uma lista corresponde a zero ou mais valores (elementos) definidos entre colchetes []. Usa-se a vírgula (,) para separar os elementos no caso da lista ser composta por mais de um elemento. A lista é dita vazia caso ela não possua nenhum elemento.

Vamos começar com um exemplo simples, a lista de compras no mercado. Pois bem, vamos supor que a lista é composta pelos seguintes produtos: arroz, feijão, macarrão, carne, pizza e leite. Essa é uma lista de *strings* (textos) constante, pois já sabemos de antemão quais são os seus elementos. A Figura 3 exibe a lista criada no interpretador. Perceba que o interpretador devolve como resultado a própria lista. Nesse caso, como não salvamos a lista em nenhum lugar, não podemos fazer nada com ela.



Fonte: Autoria própria.

Figura 4: Lista de compras criada no interpretador Idle

Vamos agora salvar a lista em uma variável para manipulá-la usando algumas propriedades. No Código 6, salvamos a lista de compras na variável compras. Uma primeira propriedade sobre listas é que os seus elementos ocupam uma posição determinada. O primeiro elemento está na posição 0, o segundo na posição 1, e o último na posição *tamanho da lista* - 1. Assim, se você quiser obter o terceiro elemento da lista de compras, você deve fazer *compras*[2] (linha 2). Observe que você sempre deve seguir esse mesmo padrão (nome da lista, abre colchetes, posição, fecha colchetes) quando quiser obter um elemento da lista.

Código 6 – Lista de compras salva em uma variável

1. compras = [‘arroz’, ‘feijão’, ‘macarrão’, ‘carne’, ‘pizza’, ‘leite’]

2. **print** (compras[2]) #irá devolver o valor ‘macarrão’

3. tamanho = **len**(compras)

4. último = tamanho - 1

5. **print**(compras[último])

Caso você queira obter o tamanho da lista você deve usar a função “*len*”, passando para ela o nome da lista, como na linha 3 do Código 6 (*len*(*compras*)). Nós usamos essa informação para obter o último elemento da lista, que está na posição tamanho da lista – 1 (linhas 4 e 5).

Da mesma forma que podemos acessar os elementos de uma lista pela sua posição, é possível também modificar esses elementos. Se você quiser trocar o item “macarrão” por “iogurte”, você deve fazer *compras*[2] = ‘iogurte’. Agora, se você der um *print* na lista (*print*(*compras*)) verá que na posição 3 (compras[2]) o item “macarrão” foi substituído por “iogurte”.

Preste atenção ao fato de que você só pode acessar um elemento que esteja na lista, ou seja, entre a posição 0 (primeiro elemento) e a posição *tamanho -1* (último elemento). Se você tentar acessar um elemento fora dos limites da lista, o programa irá terminar com um erro.

Se você quiser adicionar um elemento em uma lista já criada anteriormente, você pode usar a palavra *append*. O *append* adiciona o novo elemento ao final da lista. Caso queira remover definitivamente um elemento de uma determinada posição, você deve usar a palavra *del*, na forma como é exemplificada no código 7.

Código 7 – Lista de compras – adicionando e removendo elementos

1. compras = [‘arroz’, ‘feijão’, ‘macarrão’, ‘carne’, ‘pizza’, ‘leite’]

2. compras.**append** (‘sabão’) #adiciona o elemento ‘sabão’

3. **print** (compras)

4. **del** (compras [1]) #remove o elemento feijão

5. **print** (compras)

No Código 7, ao adicionar o elemento “sabão” na linha 3 usando append a lista será composta da seguinte forma: [‘arroz’, ‘feijão’, ‘macarrão’, ‘carne’, ‘pizza’, ‘leite’, ‘sabão’]. Por sua vez, na linha 4 removemos o elemento da posição 1, ou seja, o “feijão”, tendo como resultado a lista: [‘arroz’, ‘macarrão’, ‘carne’, ‘pizza’, ‘leite’, ‘sabão’].

Com o uso do *append*, é possível construirmos uma lista a partir da lista vazia. Por exemplo, vamos gerar a lista dos números ímpares de 3 até 15. Para isso, vamos precisar também de uma estrutura de repetição, conforme pode ser visto no Código 8.

Código 8 – Gera uma lista de números ímpares de 3 a 15

1. ímpares = []

2. n = 3

3. **while** n <= 15:

4. ímpares.**append**(n)

5. n = n + 2

6. **print** (ímpares)

Observe que na linha 1 do Código 8 foi criada uma lista vazia, denominada de *ímpares*. A variável "n" da linha 2 recebe o primeiro número ímpar. Ela serve de variável de controle do WHILE, ao mesmo tempo que é usada para construir a lista. Na linha 3 temos o teste do WHILE, que irá parar apenas quando "n" atingir o valor 15. Na linha 4, o *append* é usado para inserir cada elemento, um por vez em cada repetição. A linha 5 atualiza o valor de "n" para o próximo número ímpar. Por fim, a linha 6, que está fora do WHILE, realiza a impressão da lista gerada.

ATIVIDADE

1. Crie uma lista com as notas de 5 alunos, como por exemplo: [80.0, 76.0, 56.8, 68.0, 92.5].

2. Usando *append*, adicione mais 3 notas à sua lista.

3. Usando *del*, remova a primeira e a terceira notas.

4. Usando repetição (WHILE), percorra toda a lista e aumente o valor de cada nota em 5.0 (cinco) pontos.

Como você pode perceber, o assunto de lista é bastante vasto. Ainda há diversos recursos, operadores e funções que podemos usar. Você pode se aprofundar mais no assunto usando a bibliografia sugerida ao final desta aula. Por hora, vamos ver mais alguns recursos interessantes. Um desses recursos são os operadores **in** e **not in** que, em português, seriam traduzidos por dentro (*in*) e fora, ou, literalmente, "não dentro" (*not in*). Ou seja, usamos "in" se queremos saber se algum elemento está na lista e "not in" se queremos perguntar se determinado elemento não pertence à lista.

Código 9 – Procura por um nome em uma lista de nomes

```
1. pessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']
2. nome = input ("Digite um nome: ")
3. x = 0
4. encontrado = False
5. while x <= len (pessoas) - 1:
6.     if pessoas [x] == nome:
7.         encontrado = True
8.     x = x + 1
9. if encontrado:
10.    print (nome, "está na lista")
11. else:
12.    print (nome, "não está na lista")
```

O Código 9 é bastante interessante. Ele realiza a busca de um nome em uma lista de nomes. O nome a ser buscado deve ser fornecido pelo usuário do programa na linha 2. A variável "x" na linha 3 é usada para representar as posições da lista, de modo que possamos percorrer cada uma das posições. Ela também é usada para controlar as repetições do WHILE, indo de 0 até o tamanho da lista menos 1. A variável "encontrado" da linha 4 representa se o nome foi ou não encontrado. Ela foi inicializada com falso (*False*) representando que o nome não foi encontrado. No corpo do WHILE temos um teste (linha 6) para verificar se o nome da posição atual (x) da lista é igual ao nome que estamos procurando. Caso, em algum momento, seja encontrado um nome igual, a variável "encontrado" recebe o valor verdade verdadeiro (*True*). Na linha 8, o valor de "x" é atualizado para que se possa obter o próximo nome na lista. Por fim, o IF da linha 9 irá imprimir se o nome foi ou não encontrado com base no valor da variável encontrado.

O programa do Código 9 pode ficar bem mais curto se usarmos o recurso do operador "in". Você pode ver como isso é possível no Código 10.

Código 10 – Procura por um nome em uma lista de nomes (versão com "in")

```
1. pessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']
2. nome = input ("Digite um nome: ")
3. if nome in pessoas:
4.     print (nome, "está na lista")
5. else:
6.     print (nome, "não está na lista")
```

Em uma primeira vista no Código 10 dá para perceber que o programa ficou bastante enxuto com relação ao código 9. Pois bem, isso foi possível pelo uso da palavra **in** na linha 3. Se traduzirmos a linha 3, seria algo como "se o nome está na lista de pessoas". Ou seja, o operador "in" automaticamente faz a busca para você e devolve verdadeiro se o valor foi encontrado e falso caso contrário.

Para finalizarmos esta aula e o estudo de *Python* nesse curso, vamos mostrar a estrutura de repetição FOR (para). Ela é mais flexível que o WHILE ao trabalharmos com listas, pois elimina a necessidade da expressão de teste. Basta que você use uma variável de controle e, com o operador "in" indique a lista que será percorrida. O que o FOR faz é, a cada repetição, atribuir o elemento atual da lista, começando com o primeiro elemento, à variável de controle até que a lista termine. O Código 9 como o uso do FOR ficaria da forma como apresentado no código 11.

Código 11 – Procura por um nome em uma lista de nomes (uso do FOR)

```
1. pessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']
2. nome = input ("Digite um nome: ")
```

```
3. for x in pessoas:
6.     if x == nome:
7.         encontrado = True
8. if encontrado:
9.     print (nome, "está na lista")
10. else:
11.    print (nome, "não está na lista")
```

No caso do Código 11, com o uso do FOR eliminamos a necessidade de inicializar e atualizar a variável "x". Isso é feito automaticamente pelo FOR, que atribui a "x" o elemento atual de "pessoas". Assim, ao trabalhar com listas e a depender do problema, você pode ficar livre para escolher se quer usar WHILE ou FOR.

RESUMINDO

Caro aluno, esta foi a nossa última aula neste curso. Nessa aula, exploramos o conceito de repetição de trechos de código em *Python*. A técnica de repetição pode ser aplicada à resolução de diversos problemas, diminuindo a necessidade de variáveis e a repetição de código com instruções semelhantes. Vimos as estruturas WHILE e FOR para realizar a repetição. A primeira, de uso mais geral, permite realizar a repetição mesmo quando não sabemos de antemão quantas repetições serão necessárias. A segunda é aplicada a listas, agilizando o processo de percorrer toda uma lista de valores. Por falar em listas, essa importante estrutura de dados foi assunto da segunda parte desta aula. As listas facilitam o trabalho de definição e manipulação de um conjunto de valores relacionados, facilitando assim a resolução de diversos problemas.

LEITURAS COMPLEMENTARES

(BOSON TREINAMENTOS, 2015a) e (BOSON TREINAMENTOS, 2015b) Vídeo aulas sobre WHILE, FOR e outros comandos – Nas duas páginas que estão nas referências você encontra informações sobre os conceitos básicos de estruturas de repetição em *Python*. Alguns recursos que não foram trabalhados nessa aula também são explicados, como a saída da repetição com *break* e a geração de um intervalo para o FOR com o a palavra *range*.

(MELANDA, 2015a), (MELANDA, 2015b) e (DEVLINUXBR, 2015) – Nesses *sites* você também encontra explicações sobre as estruturas vistas nesta aula.

(STUMM JR., 2015) – Neste *site* você encontra exemplos de diversas funções e operadores que podem ser aplicados a listas.

AVALIANDO SEUS CONHECIMENTOS

1. Faça um algoritmo que leia **n** valores e calcule a média aritmética desses valores.

2. Faça um algoritmo que leia uma quantidade de N números que serão digitados e, ao final do processamento do algoritmo, mostre qual foi o maior número digitado.

Exemplo:

Se o usuário digitou 5 como a quantidade de números (N), devem ser lidos 5 números. Ao final, o algoritmo deve exibir qual desses cinco números é o maior. Dica: *leia o primeiro número fora da repetição e diga que ele é o maior (crie uma variável chamada maior), uma vez que foi o primeiro. Dentro da repetição você deve comparar os demais números lidos com o valor que está na variável maior. Se o valor for maior, então maior recebe esse novo valor*.

5 90 123 -232 0 → maior número = 123

3. Em uma eleição presidencial existem 4 candidatos. Os votos são informados através de código. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

a) 1, 2, 3, 4 = Voto para os respectivos candidatos

1 para Cecília Meireles

2 para Ariano Suassuna

3 para Machado de Assis

4 para Graciliano Ramos

b) **5** para voto em branco

c) **1234** para encerrar a votação

d) Qualquer outro valor diferente dos anteriores para anular o voto

Faça um programa que calcule e escreva:

- O total de votos para cada candidato;
- O total de votos nulos;
- O total de votos em branco.

Para finalizar a entrada dos dados (a repetição) e exibir o resultado da votação, use o valor 1234 (mil duzentos e trinta e quatro).

4. Dada uma lista de inteiros, faça um programa que procure se um número está na lista. O número a ser buscado deve ser fornecido pelo usuário do programa.

5. A partir de um texto fornecido pelo usuário, conte quantos caracteres esse texto possui, sem contar os espaços em branco.

6. Leia um texto com 10 caracteres minúsculos e diga quantas vogais foram lidas.

Referências

BOSON TREINAMENTOS. **Python**: loop While (Estrutura de Repetição) e Instrução break. 2013. Disponível em: <http://www.bosontreinamentos.com.br/programacao-em-python/19-python-loop-while-estrutura-de-repeticao-e-instrucao-break/>. Acesso em: 01 abr. 2015.

_____. **Python**: loop FOR: Estruturas de Repetição: função range. 2013. Disponível em: <http://www.bosontreinamentos.com.br/programacao-em-python/20-python-loop-for-estruturas-de-repeticao-funcao-range/>. Acesso em: 01 abr. 2015.

DEVLINUXBR. **Vamos falar de Python?**: Laços de repetição. [201-?] Disponível em: <http://devlinuxbr.blogspot.com.br/2014/09/vamos-falar-de-python-lacos-de.html>. Acesso em: 01 abr. 2015.

MELANDA, J. C. E. **Estruturas de Repetição Parte 1**: While. [2014]. Disponível em: <http://programeempython.blog.br/blog/estruturas-de-repeticao-parte-1-while/>. Acesso em: 01 abr. 2015.

_____. **Estruturas de Repetição Parte 2**: For. [2014]. Disponível em: <http://programeempython.blog.br/blog/estruturas-de-repeticao-parte-2-for/>. Acesso em: 01 abr. 2015.

PYTHON SOFTWARE FOUNDATION. Disponível em: <https://www.python.org/>. Acesso em: 10 mar. 2015.

STUMM JÚNIOR, V. **Brincando com listas**. 2013. Disponível em: <https://pythonhelp.wordpress.com/2013/06/26/brincando-com-listas/>. Acesso em: 01 abr. 2015.