

# Linguagem de Programação



## 04: ESTRUTURAS CONDICIONAIS

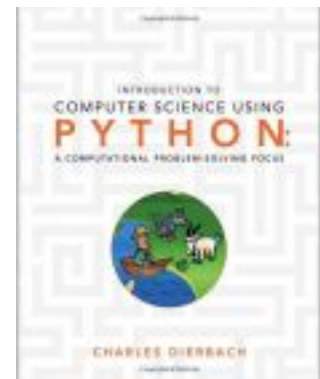
Nossos objetivos nesta aula são:

- Saber o que é uma estrutura de controle de fluxo.
- Conhecer os operadores relacionais e os operadores lógicos.
- Desenvolver e avaliar o resultado de expressões lógicas envolvendo operadores relacionais e lógicos.
- Construir algoritmos com desvio de fluxo.
- Utilizar os principais comandos condicionais: simples, composto, aninhado e encadeado.
- Ser capaz de ler, escrever e implementar programas que utilizem estruturas condicionais.



A referência para esta aula são as seções **3.1 (O que é uma estrutura de controle)**, **3.2 (Expressões lógicas)** e **3.3 (Controle de Seleção)** do **Capítulo 3 (Control Structures)** do livro:

DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus*. 1st Edition, New York: Wiley, 2012.



## MOTIVAÇÃO

Os primeiros computadores eletrônicos foram referidos como "cérebros eletrônicos". Deu-se a impressão enganosa de que os computadores poderiam "pensar". Embora possam ser complexos em seu design, computadores são máquinas que simplesmente fazem, passo a passo (instrução por instrução), o que lhes é informado.



Assim, não há mais inteligência em um computador do que aquilo que ele é instruído a fazer. O que os computadores podem fazer, no entanto, é executar uma série de instruções muito rapidamente e de forma confiável.

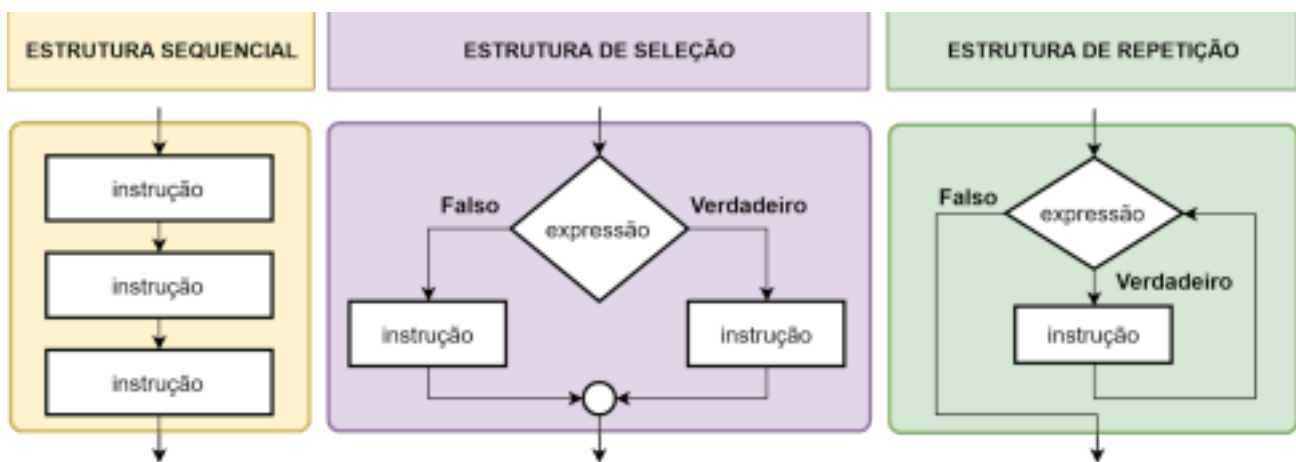
É a velocidade em que as instruções podem ser executadas que dá aos computadores o seu poder, uma vez que a execução de muitas instruções simples pode resultar em comportamentos complexos. Vamos a partir de agora controlar não só a ordem em que as instruções são executadas, mas também se de fato serão executadas.

## O QUE É UMA ESTRUTURA DE CONTROLE?

A ordem em que as instruções de um programa são executadas é chamada de **controle de fluxo**.

Existem três estruturas fundamentais para controlar o fluxo de um programa:

- I. Estrutura de controle sequencial;
- II. Estrutura de controle de seleção ou condicional; e
- III. Estrutura de controle de repetição, iterativo ou laço.



A estrutura de controle sequencial, caracterizada pelos problemas vistos até agora, é uma forma implícita de controle, pois as instruções são executadas na sequência em que são escritas.

A partir de agora, conheceremos o segundo tipo de controle de fluxo, a estrutura de seleção, também conhecida como estrutura condicional. Porém, é primordial entendermos o que é uma expressão lógica antes de usarmos estruturas de seleção.

## EXPRESSÕES LÓGICAS

Como vimos anteriormente, o resultado de expressões aritméticas são números inteiros ou reais. Em expressões lógicas o resultado gerado é um valor lógico (também conhecido como booleano), ou seja, **Verdadeiro** (True) ou **Falso** (False).

Novamente como analogia, em expressões aritméticas os operandos costumam ser números e os operadores são aritméticos (adição, subtração, multiplicação, divisão e etc.), já com expressões lógicas os operandos também podem ser números, porém há presença de **operadores relacionais** e/ou **operadores lógicos**.

Os operadores relacionais são usados em comparações entre valores. O resultado da comparação será um valor lógico, como exposto no quadro a seguir:

Operadores	Exemplo	Resultado
<code>==</code>	igual a 10	<code>10 == 10</code> True <code>10 != 10</code> False
<code>&lt;</code>	menor que 10	<code>10 &lt; 20</code> True
<code>&gt;</code>	maior que	<code>'Alan' &gt; 'Brenda'</code> False
<code>&lt;=</code>	menor ou igual a 10	<code>10 &lt;= 10</code> True
<code>&gt;=</code>	maior ou igual a	<code>'A' &gt;= 'D'</code> False

## PARA PRATICAR

Qual o resultado de cada expressão lógica a seguir?

```
>>> '2' < '9' >>> '12' < >>> 'Hello' < 'Zebra' >>>
>>> 10 == 20 >>> 10 != 20 '9' >>> '12' > '9' 'hello' < 'ZEBRA'
>>> 10 <= 20 >>> 'Hello' == 'Hello'
```

Considerando **a = 4, b = 10, c = 50, d = 1, e = 5**, qual será o resultado lógico (True/False) das seguintes expressões?

Expressão	Resultado
<code>a == c</code>	
<code>a &lt; b</code>	
<code>d &gt; b</code>	
<code>c != e</code>	
<code>a == b</code>	
<code>c &lt; d</code>	
<code>b &gt; a</code>	
<code>c &gt;= e</code>	
<code>c &lt;= c</code>	
<code>c &lt;= e</code>	

## OPERADORES LÓGICOS

Os operadores lógicos podem ser usados para construir expressões lógicas mais complexas,

permitindo a combinação de comparações. Consideraremos três operadores: E (`and`), OU (`or`) e NÃO (`not`).

- O operador lógico **and** resulta verdade somente se os dois operandos forem verdadeiros. •
- O operador lógico **or** resulta verdade se ao menos um dos dois operandos for verdadeiro. •
- O operador lógico **not** resulta verdade se o operando for falso e falso caso seja verdadeiro.

Veja a tabela verdade dos operadores lógicos:

x	y	x and y	x or y	not y
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	True

É preciso ser cauteloso aos usar operadores lógicos. Por exemplo, na matemática, para indicar que um valor está dentro de um determinado intervalo escrevemos:

$$(1 \leq \text{num} \leq 10)$$

No entanto, na maioria das linguagens de programação esta expressão não faz sentido. Para entender porque, vamos assumir que a variável `num` tem o valor 15. A conversão da expressão anterior para uma vasta gama de linguagens de programação seria assim:

$$(1 \leq \text{num} \leq 10) \rightarrow (1 \leq 15 \leq 10) \rightarrow (\text{True} \leq 10) \rightarrow ???$$

Em diversas linguagens de programação a expressão será avaliada de modo diferente ao tradicionalmente aceito na matemática. Inicialmente se avaliaria a primeira parte da expressão lógica (`1 <= num`) que geraria um resultado booleano **True**; na sequência seria avaliada a segunda parte da expressão (`True <= 10`). Porém, não faz sentido verificar se `True` é menor ou igual a 10. A forma comumente aceita para a expressão anterior usaria o operador **and**:

$$(1 \leq \text{num}) \text{ and } (\text{num} \leq 10)$$

ou

$$(\text{num} \geq 1) \text{ and } (\text{num} \leq 10)$$

## PARA PRATICAR

Qual o resultado de cada expressão lógica a seguir?

```

>>> True and False      a) 5 < 3 or 4 > 2 and 1 < 7
>>> True or False        >>> True and not False >>>
>>> not(True) and True   (10 < 0) and (10 > 2) >>>
>>> not True or True     (10 < 0) or (10 > 2) >>>
>>> not(True and False)  not(10 < 0) and (10 > 2)
>>> not(10 < 0 or 10 > 2)

```

E das expressões a seguir?

b) (not 5 < 3 or 3 < 2) and not 1 < 7

## PRECEDÊNCIA DE OPERADORES (PRIORIDADE NA AVALIAÇÃO)

A precedência dos operadores relacionais e lógicos, bem como suas associatividades, é definida conforme explicitado na seguinte tabela:

Prioridade	OPERADOR	ASSOCIATIVIDADE
5	< > <= >= != ==	Não há
6	not	→ Esquerda para direita
7	and	→ Esquerda para direita
8	or	→ Esquerda para direita

Expressões lógicas também podem conter operadores aritméticos. Agora podemos expandir nossa tabela de operadores conhecidos e suas respectivas precedências:

Prioridade	Operador	Associatividade
1	**	Direita para esquerda
2	-	(negação) → Esquerda para direita
3	* / // %	→ Esquerda para direita
4	+ -	→ Esquerda para direita
5	< > <= >= != ==	Não há
6	not	→ Esquerda para direita
7	and	→ Esquerda para direita
8	or	→ Esquerda para direita

## EXPRESSÕES LÓGICAS EQUIVALENTES

Assim como há expressões aritméticas equivalentes, como  $a*(b+c)$  é igual a  $a*b + a*c$ , também existem expressões lógicas equivalentes. É importante reconhecer as equivalências para

simplificar expressões e outras finalidades relacionadas.

Expressão original Equivalência

	<code>x &lt; y</code>	<code>not(x &gt;= y)</code>
	<code>x &lt;= y</code>	<code>not(x &gt; y)</code>
	<code>x == y</code>	<code>not(x != y)</code>
	<code>x != y</code>	<code>not(x == y)</code>
	<code>not (x and y)</code>	<code>(not x) or (not y)</code>
	<code>not (x or y)</code>	<code>(not x) and (not y)</code>
	<code>not not x</code>	<code>x</code>
	<code>not not not x</code>	<code>not x</code>

## PARA PRATICAR

Qual o resultado de cada expressão lógica a seguir? Indique uma expressão lógica equivalente.

```
>>> 10 < 20
```

```
>>> not (10 >= 20) >>> 10 !=
```

```
20
```

```
>>> not(10 == 20)
>>> not(10 < 20 and 10 < 30)
```

```
>>> (not 10 < 20) or (not 10 <
```

```
30) >>> not(10 < 20 or 10 <
```

```
30) >>> (not 10 < 20) and (not
```

```
10 < 30)
```

## ESTRUTURAS DE SELEÇÃO (ESTRUTURAS CONDICIONAIS)

A partir de agora vamos estudar as estruturas condicionais que permitem que programas executem sequências diferentes de instruções, dependendo da avaliação de uma expressão lógica.

Em programação, o uso de condições para permitir a escolha de executar ou não um trecho de

código é muito utilizado, principalmente quando precisamos incluir no programa condições de controle, para evitar situações não permitidas que podem resultar em erros. Por exemplo, para evitar divisões por zero.

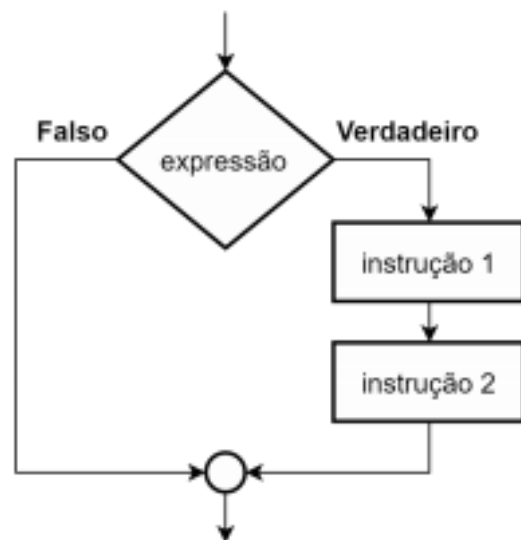
Quando há apenas **um bloco** de instruções cuja execução está condicionada ao resultado de uma expressão lógica, usamos uma **estrutura de seleção simples**.

Existe outra estrutura, chamada de **estrutura de seleção composta**, em que ainda temos apenas uma condição a ser avaliada, porém **dois caminhos** que podem ser seguidos. Um dos caminhos leva à execução do bloco de instruções condicionado ao resultado **verdade** e o outro à execução do bloco de instruções condicionado ao resultado **falso**.

### ESTRUTURA DE SELEÇÃO SIMPLES

Na estrutura de seleção simples ou estrutura condicional simples, um bloco de código só será executado caso a expressão da condição da estrutura resulte um valor lógico verdade. Caso a condição resulte falso, o bloco será ignorado e as instruções posteriores, externas à estrutura condicional, serão executadas.

Suponha a seguinte situação: dado o preço de um produto comercializado por uma loja e a quantidade comprada por um cliente, exiba o valor total da compra. Considere que se o total for maior do que R\$ 150,00 haverá desconto de 10%. Um fluxograma para esta situação, e o respectivo código em Python, pode ser representado como a seguir:



```
instrução fora da seleção
```

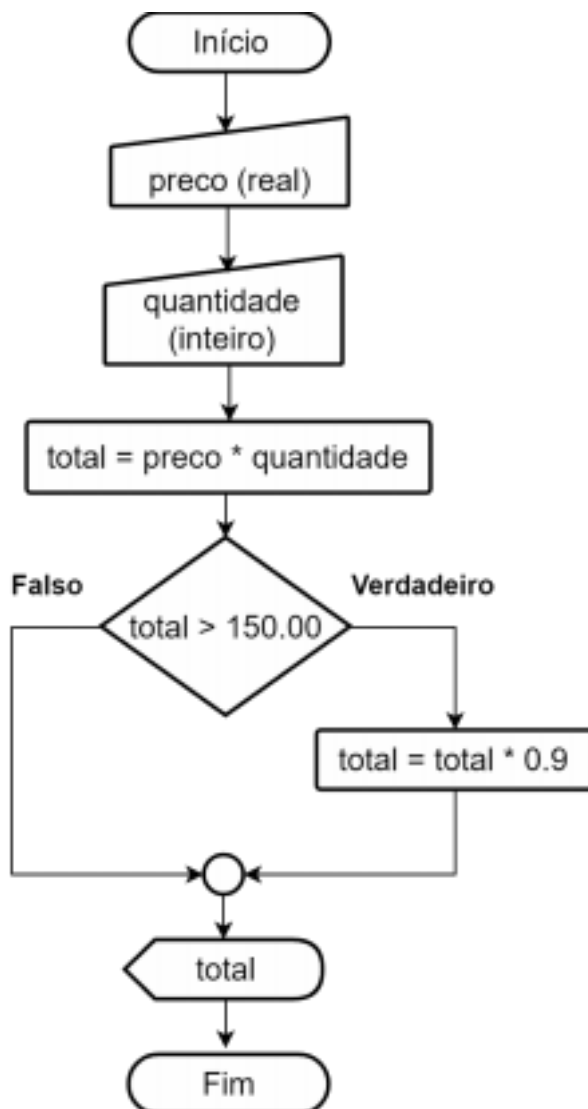
```
if expressão:
```

```
    1ª instrução dentro da seleção
```

```
    2ª instrução dentro da seleção
```

```
    3ª instrução dentro da seleção
```

```
instrução fora da seleção
```



```

preco = float(input())
quantidade = int(input())
total = preco * quantidade
if total > 150.00:
    total = total * 0.9
print(total)

```

Note que o valor da variável total sempre será exibido, independentemente se houver ou não desconto, porém se o valor de total for superior à 150.00 o trecho de código que o reduz em 10% será executado, pois a condição da seleção resultará verdade.

### PARA PRATICAR

Reescreva o código referente ao fluxograma ao



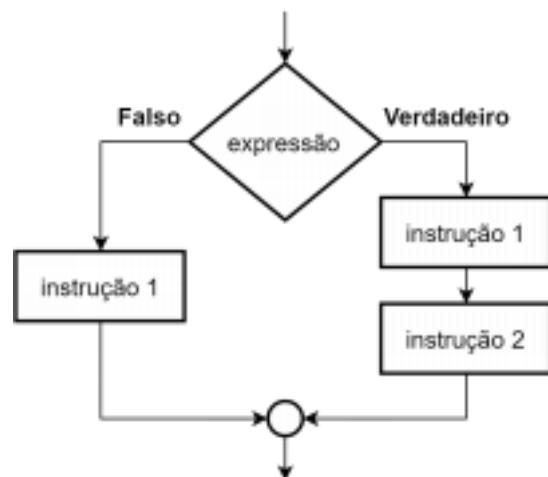
lado, porém supondo que o desconto é aplicado somente se o valor total da compra for superior a R\$ 200,00 e se forem compradas no mínimo 5 unidades do produto.

8

## ESTRUTURA DE SELEÇÃO COMPOSTA

Na **estrutura de seleção composta** ou **estrutura condicional composta**, há dois caminhos possíveis, cada um com um conjunto de instruções, mas apenas um deles será executado.

Um bloco de código só será executado caso a condição da estrutura de seleção resulte um valor lógico **verdade**. Caso a condição resulte **falso**, outro bloco de códigos será executado.



Suponha a seguinte situação: dada a nota de um aluno, exibir 'aprovado' caso esta seja maior ou igual a 6.0, e 'reprovado' caso contrário. Um fluxograma para esta situação pode ser representado assim:

Note que neste caso somente uma mensagem será exibida. Se a condição da seleção resultar **verdade**, será exibido 'aprovado', caso contrário

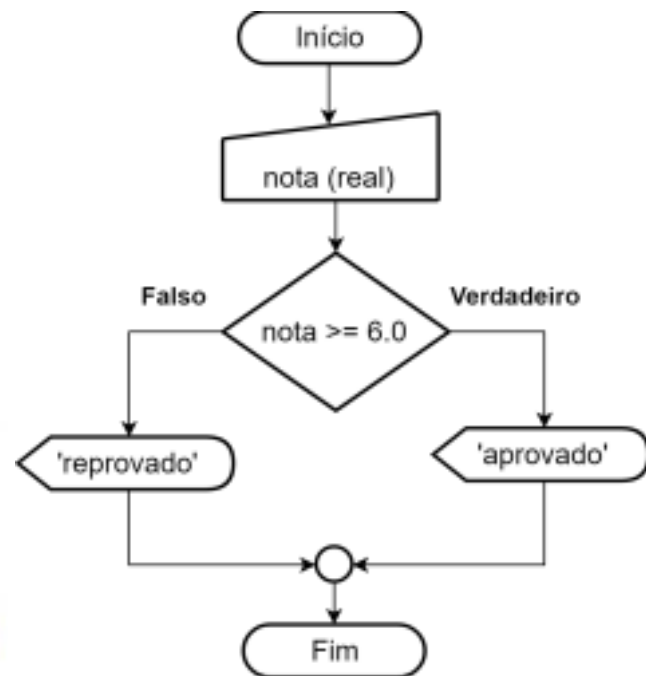
instrução fora da seleção

```
if expressão:
    1ª instrução dentro da seleção
    2ª instrução dentro da seleção
else:
    1ª instrução dentro da seleção
    2ª instrução dentro da seleção
```

será exibido 'reprovado'.

Costumeiramente, chamamos o lado que é executado quando a condição resulta falso de "**senão**".

```
nota = float(input())
if nota >= 6.0:
    print('aprovado')
else:
    print('reprovado')
```



9

#### PARA PRATICAR

- 1) Reescreva o exemplo anterior, porém solicitando também o número de faltas do aluno e exibindo 'aprovado' somente se a nota for maior do que 7.0 e o número de faltas inferior à 5. Caso contrário o programa deverá exibir 'reprovado'.
- 2) Escreva o programa apropriado para cada um dos seguintes itens considerando que a variável num receba um número real qualquer:
  - a) Exiba 'dentro do intervalo' se  $1 \leq \text{num} \leq 100$ .
  - b) Exiba 'dentro do intervalo' se  $30 < \text{num} < 70$ , caso contrário, exiba 'fora do intervalo'.
- 3) Resolva as expressões lógicas a seguir, calculando o valor final da variável s:
  - a)  $s = ((\text{not } (1 == 2)) \text{ and } (3 < 4))$
  - b)  $s = ((1 < 2) \text{ or } (3 > 4))$
  - c)  $s = ((2 == 2) \text{ and } (3 > 4)) \text{ or } (3 > 4)$
- 4) Construa um programa que lê um número inteiro e exibe uma mensagem indicando se é um número par ou ímpar.
- 5) Um comerciante comprou um produto e quer vendê-lo com um lucro de 45% se o valor da compra for menor do que R\$ 20,00, caso contrário, o lucro será de 30%. Construa um

programa que recebe o valor do produto e exibe o valor da venda.

- 6) As maçãs custam R\$ 1,30 a unidade, se for comprado no máximo uma dúzia, e R\$ 1,00 se for comprado mais de uma dúzia. Construa um programa que lê o número de maçãs compradas e exibe o valor total da compra.
- 7) Vamos simular o controle de um ventilador (liga/desliga) baseado na leitura de uma temperatura. Para esse exercício criar um fluxograma que lê a temperatura atual. Caso ela seja igual ou maior do que 25 graus ele deve ligar o ventilador. Caso contrário deve desligar o ventilador. OBS.: Vamos simular o comando de ligar/desligar o ventilador através de uma mensagem "LIGAR O VENTILADOR" ou "DESLIGAR O VENTILADOR".

10

## COMO TESTAR UM PROGRAMA?

Existem formas de testar a corretude de programas, ou seja, se o programa faz aquilo que ele deveria fazer, inclusive aqueles com estruturas de seleção. Uma das formas é o teste de **caixa-preta**, cujos passos resumidamente são:

- a) Escolhe-se uma entrada cuja saída correta correspondente seja conhecida, sem precisar usar o programa que será testado;
- b) Executa-se o programa passando a entrada escolhida;
- c) Compara-se a saída real do programa com aquela teórica inicialmente esperada; d) Caso haja diferença entre a saída teórica e a saída real, muito provavelmente o programa está incorreto e necessita de correções. As devidas modificações são feitas e retorna-se ao passo (b);
- e) Caso não existam divergências entre as saídas, opta-se por: (I) encerrar os testes ou (II) submeter o programa a outros casos de teste.

Note que **testes de caixa-preta não consideram a estrutura interna** do programa, ou seja, o algoritmo usado para construir o programa, assim como a linguagem de programação em que foi implementado, não importam.

Veja a simplificação do de teste de



ilustração do procedimento caixa-preta na

Figura 1.

**Figura 1** – Simplificação do funcionamento do teste de caixa-preta.

Existem sistemas que automatizam o procedimento de testes de caixa-preta para algoritmos implementados em linguagens de programação, como os juízes online The Huxley (<https://www.thehuxley.com/>), URI Online Judge (<https://www.urionlinejudge.com.br>) e SPOJ (<https://www.spoj.com/>) .

## 11

Lembre-se, **testes não garantem**, necessariamente, **que um programa está correto**. Um programa que seja aprovado em todos os casos de teste aos quais foi submetido não está obrigatoriamente correto, pois pode haver um caso de teste ausente que geraria uma falha.

Para garantir a corretude de um programa por meio de testes é necessário fazer um **teste exaustivo**. Testes exaustivos submetem o programa à todas as entradas esperadas possíveis, o que muitas vezes é impraticável. Imagine quantas possíveis entradas esperadas existem para um programa que soma dois números inteiros e exibe o resultado. Infinitas!

Uma forma de garantir a corretude de um programa sem precisar construir testes é por meio de **prova formal**, porém esse método exige maiores conhecimentos matemáticos, técnicas avançadas de análise de algoritmos e criatividade. Provas formais costumeiramente demandam mais recursos financeiros e tempo para serem satisfatoriamente concluídas em programas mais extensos.

Geralmente, para **sistemas não críticos** (aqueles que não gerenciam laboratórios com doenças altamente contagiosas; não administram usinas nucleares; não controlam produção de foguetes; não automatizam cirurgias, aviões, trens, etc.) são montadas sequências de testes com boa abrangência, testando principalmente as extremidades das possíveis entradas esperadas. Isso amplia o nível de confiança nas soluções propostas. Em nossas aulas optamos por essa abordagem

simplificada para o teste de programas.

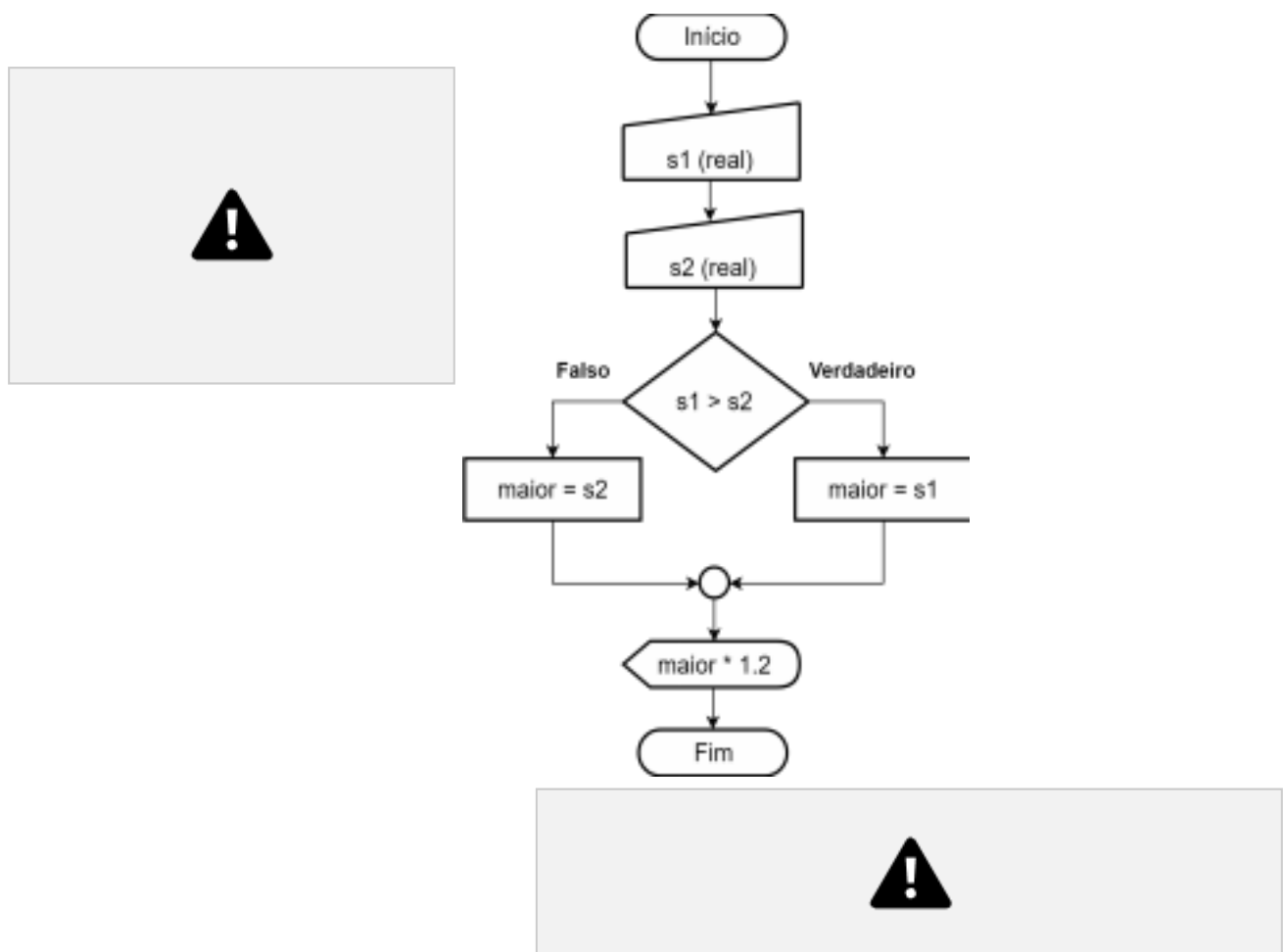
Também é possível usar **testes de caixa-branca**, em que a estrutura interna do programa é avaliada, ou seja, cada instrução do algoritmo implementado em uma linguagem de programação será analisada de acordo com a sequência de execução. Neste formato de teste é necessário "abrir a caixa e ver o que está dentro".

O uso de **testes de mesa** pode auxiliar na execução de testes de caixa-branca, pois permite que toda instrução que implique em mudança nas variáveis seja representada em uma linha-coluna da tabela. No caso de testes de mesa com estruturas condicionais, é facilitador acrescentar uma coluna com a expressão da condição de seleção, sendo útil para perceber erros na definição da condição (frequentes com iniciantes).

Bons testes de caixa-branca para programas com estruturas de seleção buscam passar por **todos os caminhos do algoritmo**, garantindo que sejam executadas e analisadas tanto as instruções do "bloco do caminho se verdadeiro" quanto do "bloco do caminho se falso", se houver.

Pense no seguinte **exemplo**: crie um fluxograma que leia como entrada dois salários distintos e exiba uma mensagem com o valor do maior deles, acrescido de 20% de bônus. Vamos construir um fluxograma e um programa, fazer dois testes de caixa-preta (comparando com as saídas do programa real) e dois testes de caixa-branca.





Como podemos perceber, os resultados produzidos pelo fluxograma e pelo programa, analisados com auxílio dos testes de mesa, são iguais àqueles previstos no teste de caixa-preta. Portanto, há maior confiança que o algoritmo faz o que deve fazer, ou seja, existem mais indícios de que esteja correto.

## 1º Teste de mesa

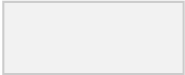
```
s1 s2 maior s1 > s2 Tela 10000.00 5000.00 10000.00 10000.00 >
5000.00
(True) 12000.00
```

## 2º Teste de mesa

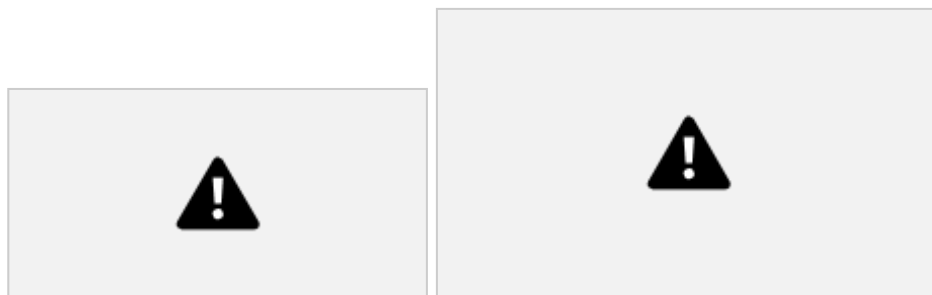
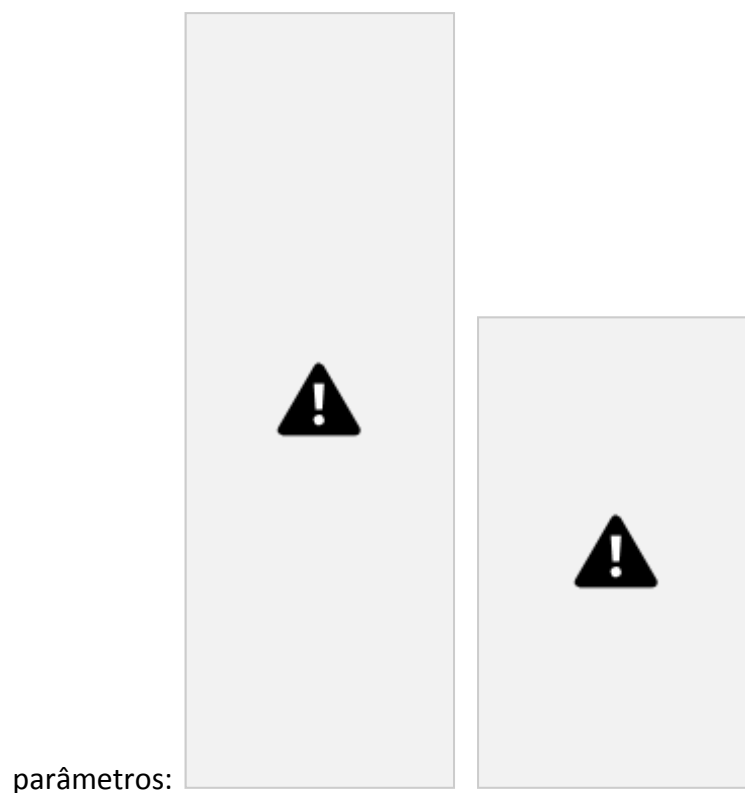
```
s1 s2 maior s1 > s2 Tela 2500.77 3500.25 3500.25 2500.77 > 3500.25
(False) 4200.30
```

13

## ROTINAS E FUNÇÕES - PASSAGEM DE PARÂMETROS

No capítulo anterior vimos a utilização de funções utilizando o símbolo . Neste capítulo vamos aprender a utilizar funções com parâmetros.

À esquerda está a função `soma()` do capítulo anterior e à direita a versão com



Percebemos duas diferenças:

- 1) Não existe no segundo fluxograma, e na respectiva função em Python, a entrada manual de dados;
- 2) Na assinatura da função aparecem elementos a mais `soma (n1, n2)` que são os dois parâmetros necessários para a função "fazer o que tem que fazer".

14

O nome da função é seguido por um par de parênteses e, entre eles, alguns identificadores chamados de **parâmetros formais** ou simplesmente parâmetros. No exemplo, os parâmetros são `n1` e `n2`.

Parâmetros servem para que não seja necessário utilizar a entrada de dados manual na função. Os parâmetros são dados externos à função, necessários para a execução de seu processamento e gerar a saída correspondente. Note, a função `soma` deve somar dois números e **exibir** o resultado, portanto, quais são os dados externos necessários? Apenas os dois números que devem ser somados!

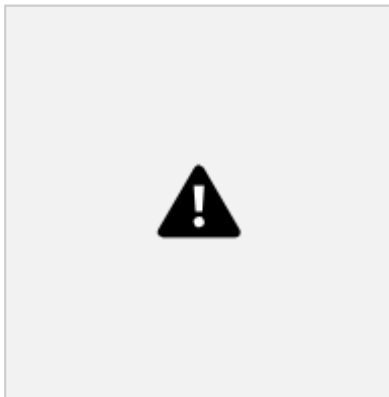


Reforçando: parâmetros servem, essencialmente, para **comunicar algum dado externo** à função. Por exemplo, para calcular a soma de dois números a função precisa saber quais são os dois números, por isso os valores são **passados como parâmetros** para ela.

Todo parâmetro deve ter um nome que, preferencialmente, faça referência ao dado que será armazenado, assim como variáveis comuns. As funções podem ser construídas com **qualquer número** de parâmetros, inclusive zero.

### EXEMPLO

Veja a figura à seguir. À esquerda temos um exemplo do fluxograma criado para realizar a mesma operação (receber dois números, somá-los e exibir o resultado) duas vezes, e à direita, temos a representação do mesmo problema resolvido com o uso de uma função.



Na primeira chamada da função, os valores a serem somados, que são passados para a função, foram digitados pelo usuário. O valor da soma exibido na saída dependerá dos valores digitados pelo usuário. Na segunda chamada foram passados pelo programador 5 e 8 diretamente. O valor da soma exibido na saída será 13.

Nem sempre quando utilizamos uma função queremos que os valores usados como entrada da função sejam digitados pelo usuário.

15

### EXERCÍCIOS PROPOSTOS

#### (EXPRESSÕES LÓGICAS - ESTRUTURA DE SELEÇÃO SIMPLES/COMPOSTA)

1) Determine os resultados obtidos na avaliação das expressões lógicas a seguir sabendo que:

`a = 2, b = 7, c = 3.5, m = True, n = False:`

a) `b == a * c and (m or n)`

b) `b > a or b == pow(a, a)`

c) `m and b // a >= c or not a <= c`

d) `not m or n and sqrt(a + b) >= c`  
e) `b/a == c or b/a != c`  
f) `m or pow(b,a) <= c * 10 + a * b`

- 2) Elabore um programa que leia dois números reais e mostre o resultado da diferença do maior valor pelo menor.
- 3) Elabore um programa que leia um número inteiro de três dígitos e imprima se o dígito da centena é par ou ímpar.
- 4) Elabore um programa que leia as notas de três avaliações de um aluno. A primeira avaliação tem peso 2, a segunda tem peso 3 e a terceira, peso 5. Calcule a média do aluno e exiba-a na tela. Se a média do aluno for maior ou igual a 6.0, também exiba 'aprovado'; caso contrário, exiba 'reprovado'.
- 5) Um pescador comprou um computador para controlar o rendimento diário de seu trabalho. Toda vez que ele traz um peso de peixes maior que o estabelecido pelo regulamento de pesca do Estado de São Paulo (50 kg), deve pagar uma multa de R\$ 4,00 por quilo excedente. Elabore um programa que leia o peso de peixes, e verifique se há excesso. Se houver excesso, exiba o peso excedente e o valor da multa, caso contrário, exiba 'dentro do regulamento'.
- 6) Faça um programa que coloque dois nomes em ordem alfabética.
- 7) Faça um programa que exiba uma mensagem indicando se o número que o usuário digitou é divisível por 3 e por 5 ao mesmo tempo.
- 8) Faça um programa que receba um ano (quatro dígitos) e informe se é um ano bissexto ou não. Pesquise quais as regras para um ano ser bissexto.
- 9) Faça um programa que leia do teclado o sexo de uma pessoa. Se o sexo digitado for 'M', 'm', 'F' ou 'f', escrever na tela 'sexo válido!'. Caso contrário, exibir 'sexo inválido!'.
- 10) Para transferências de veículos num determinado estado, o DETRAN cobra uma taxa de 2,5% sobre o valor de tabela do carro para carros fabricados antes de 2010, e uma taxa de 3,5% para os fabricados a partir de 2010. Faça um programa que lê o ano e o preço do carro e em seguida calcula e imprime a taxa a ser paga.

## TAREFA MÍNIMA (ESTRUTURAS DE SELEÇÃO SIMPLES E COMPOSTA)

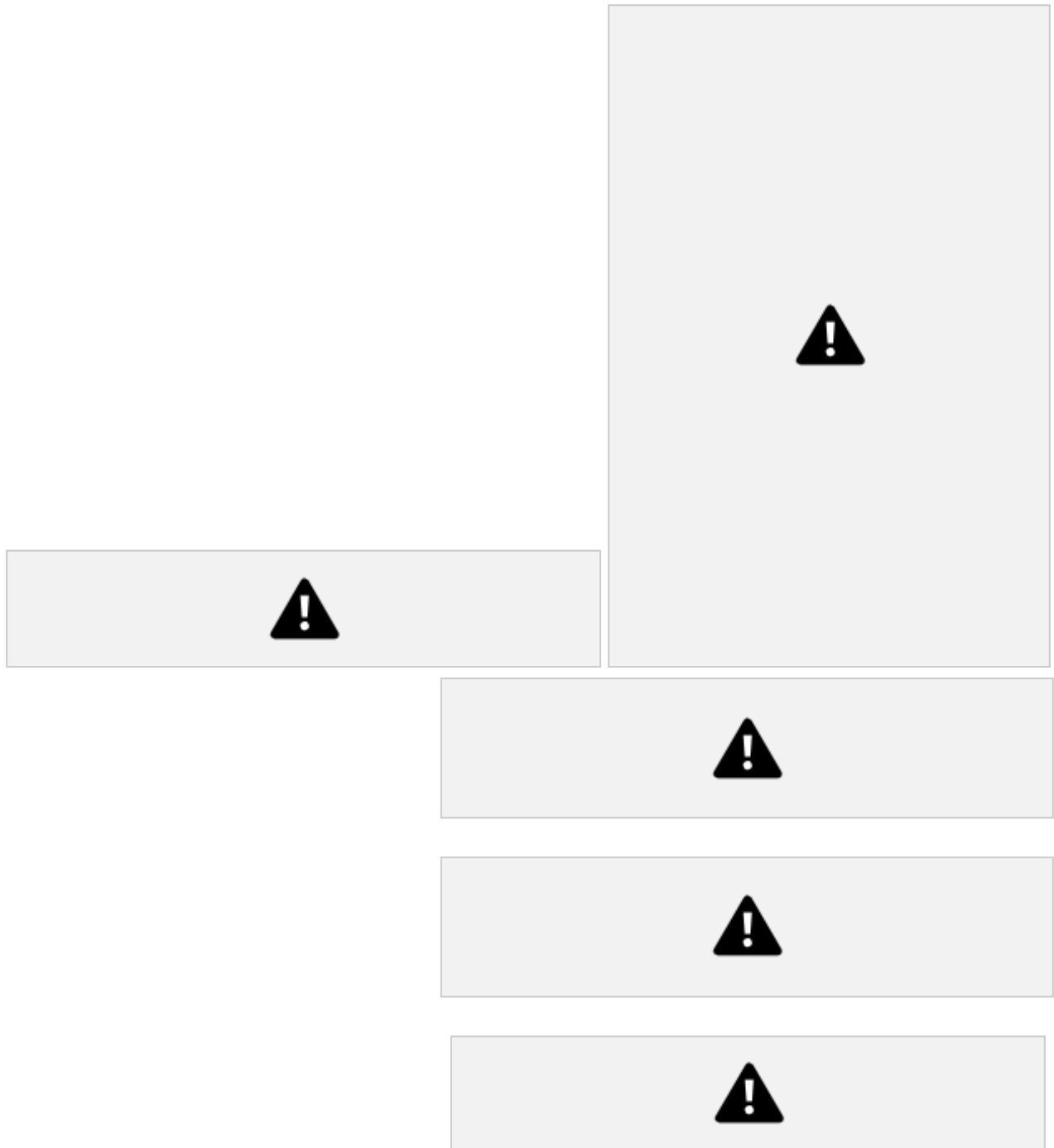
### Exercício 1 - Parcialmente resolvido

Refaça o programa do salário dado como exemplo, porém em versão função. A função solicitará dois salários distintos ao usuário e exibirá o maior entre eles acrescido de 20% de bonificação. Não se esqueça de fazer os testes de mesa. Em seguida, faça mais uma versão, agora com uma função

que recebe os dois salários como parâmetros, ao invés de recebê-los como entradas manuais.

Obs.: Nosso padrão para fluxograma de funções exige substituir a palavra "início" e "fim" nos símbolos de terminador por, respectivamente, nome\_da\_funcao() e return.

**RESOLUÇÃO** ✓



```

print(maior * 1.2)
def salario(s1, s2):
    if s1 > s2:
        maior = s1
    else:
        maior = s2
    print(maior * 1.2)

```

```

def salario():
    s1 = float(input())
    s2 = float(input())
    if s1 > s2:
        maior = s1
    else:
        maior = s2

```

17

## Exercício 2

Crie um programa que leia um número inteiro  $n$  e apresente a raiz quadrada de  $n$  caso seja positivo ou nulo, e o quadrado de  $n$  caso seja negativo. Em seguida faça outro programa para o mesmo problema, porém usando uma função com leitura manual de  $n$ . Não se esqueça de fazer testes de mesa. Por último, faça mais uma versão do programa agora com um função que receba  $n$  como parâmetro.

## Exercício 3 - Parcialmente resolvido

Faça um programa que recebe um número inteiro  $n$  e exibe uma mensagem indicando se ele é par ou ímpar (use o operador  $\%$  para obter o resto de uma divisão inteira). Não se esqueça de fazer testes de mesa. Por último, faça outra versão do programa, usando uma função que receba  $n$  como parâmetro.

**RESOLUÇÃO ✓**





#### Exercício 4

Faça um programa que recebe um número inteiro  $x$ , sendo  $x > 9$ , e exibe uma mensagem indicando se o segundo dígito da direita para a esquerda é par ou ímpar (use o operador % para obter o resto de uma divisão inteira). Em seguida faça outro programa do mesmo problema, usando uma função. Não se esqueça de fazer testes de mesa.

18

#### Exercício 5 - Resolvido

Considerando operadores relacionais e lógicos, responda cada item com o que se pede: 1) Defina o resultado de cada uma das expressões a seguir:

- a)  $10 \geq 8$
- b)  $8 \leq 10$
- c) `'Dave' < 'Dale'`
- d)  $10 \neq 8$
- e) `'8' < '10'`

2) Avalie as seguintes expressões lógicas seguindo as regras de precedência dos operadores:

- a)  $10 \geq 8$  and  $5 \neq 3$
- b)  $10 \geq 8$  and  $5 == 3$  or `'a' != 'A'`

#### RESOLUÇÃO ✓

1) a) True; b) True; c) False; d) True; e) False. 2)  
a) True; b) True.

#### Exercício 6

Considerando operadores relacionais e lógicos, responda cada item com o que se

pede: 1) Defina o resultado de cada uma das expressões a seguir:

- a) `'Dave' < 'Ed'`
- b) `'dave' < 'Ed'`
- c) `10 == 8`
- d) `10*3 == 120//4`

2) Qual das seguintes expressões lógicas não é logicamente equivalente às outras duas?

- a) `not(num < 0 or num > 10)`
- b) `num > 0 and num < 10`
- c) `num >= 0 and num <= 10`