**Texto base**

**4**

**Estrutura de Seleção - Operadores e Expressões Lógicas**

Gilberto Alves Pereira

***Resumo***

*Nem todos os comandos de um programa precisam ser sempre executados. Existem situações em que precisamos executar os comandos seletivamente e inclusive às vezes não executar. Vamos aprender nesse texto sobre a ferramenta que permite que isso seja feito - Estrutura de Seleção. Como base para a utilização dessa ferramenta iremos aprender um pouco mais sobre os operadores e as expressões lógicas. Discutimos também os conceitos de estrutura de seleção simples e composta. Tanto a estrutura de seleção simples quanto a estrutura de seleção composta utilizam o símbolo de decisão. Através dessa decisão definimos obrigatoriamente dois caminhos no nosso fluxograma. Abaixo são expostos os conceitos dessas estruturas assim como exemplos de utilização. Ao final é mostrado um exemplo usando fluxograma com sua conversão para Python.*

**1.1. 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**.

**1.2. Operadores relacionais**

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

**Tabela 4.1. Exemplos dos operadores relacionais.**

| **Operadores** | | **Exemplo** | **Resultado** |
| --- | --- | --- | --- |
| == | igual a | 10 == 10 | True |
| != | diferente de | 10 != 10 | False |
| < | menor que | 10 < 20 | True |
| > | maior que | 'Alan' > 'Brenda' | False |
| <= | menor ou igual a | 10 <= 10 | True |
| >= | maior ou igual a | 'A' >= 'D' | False |

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

Veja a tabela verdade dos operadores lógicos:

**Tabela 4.2. Exemplos de operadores lógicos.**

| **idade >18** | **sexo==”F”** | **idade >18 and sexo==”F”** | **idade >18 or sexo==”F”** | **not sexo==”F”** |
| --- | --- | --- | --- | --- |
| True | True | True | True | False |
| True | False | False | True | True |
| False | True | False | True |  |
| False | False | False | False |  |

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

(1 <= num <= 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 <= num <= 10) → (1 <= 15 <= 10) → (True <= 10) → ???

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 <= num) and (num <= 10)

ou

(num >= 1) and (num <= 10)

**1.4. Precedência de operadores (prioridade na avaliação)**

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

**Tabela 4.3. Exemplos da precedência de operadores.**

| **Operador** | **Associatividade** |
| --- | --- |
| < > <= >= != == | Esquerda para direita |
| not | Esquerda para direita |
| and | Esquerda para direita |
| or | Esquerda para direita |

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

**Tabela 4.4. Exemplos de precedência de operadores, com operadores aritméticos.**

| **Operador** | **Associatividade** |
| --- | --- |
| \*\* | Direita para esquerda |
| - (negação) | Esquerda para direita |
| \* / // % | Esquerda para direita |
| + – | Esquerda para direita |
| < > <= >= != == | Esquerda para direita |
| not | Esquerda para direita |
| and | Esquerda para direita |
| or | Esquerda para direita |

**1.5. 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.

**Tabela 4.5. Exemplos de expressões lógicas equivalentes.**

| **Expressão original** | **Equivalência** |
| --- | --- |
| x < y | not(x >= y) |
| x <= y | not(x > y) |
| x == y | not(x != y) |
| x != y | not(x == y) |
| not (x and y) | (not x) or (not y) |
| not (x or y) | (not x) and (not y) |
| not not x | x |
| not not not x | not x |

**1.6. Estruturas de seleção (estruturas condicionais)**

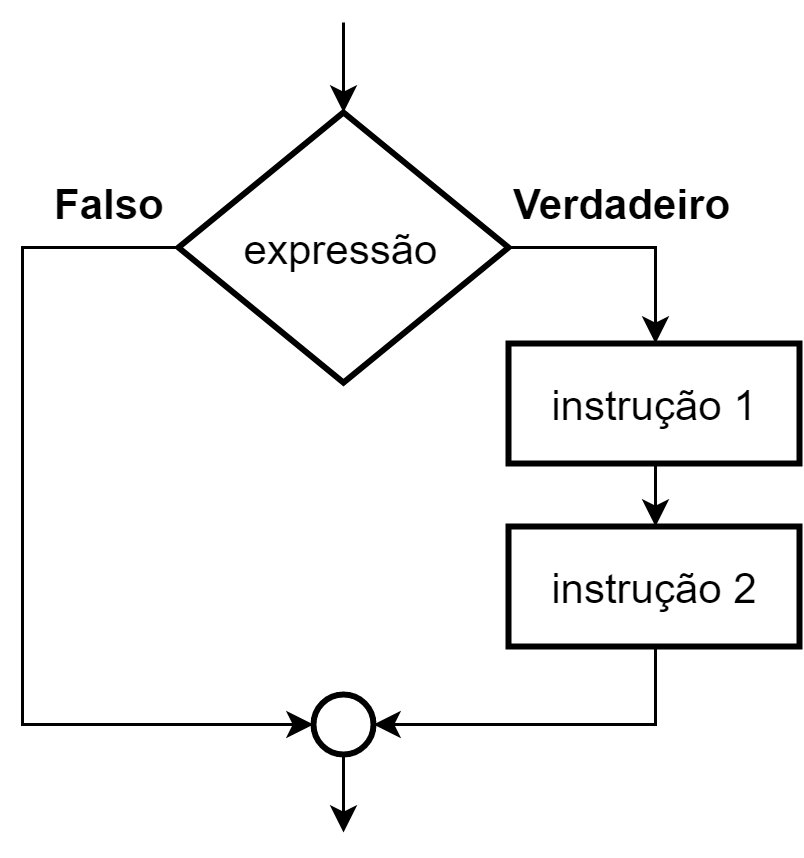
A partir de agora vamos estudar as estruturas condicionais que permitem que programas executem sequências diferentes de instruções, dependendo da avaliação de uma expressão lógica.

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**.

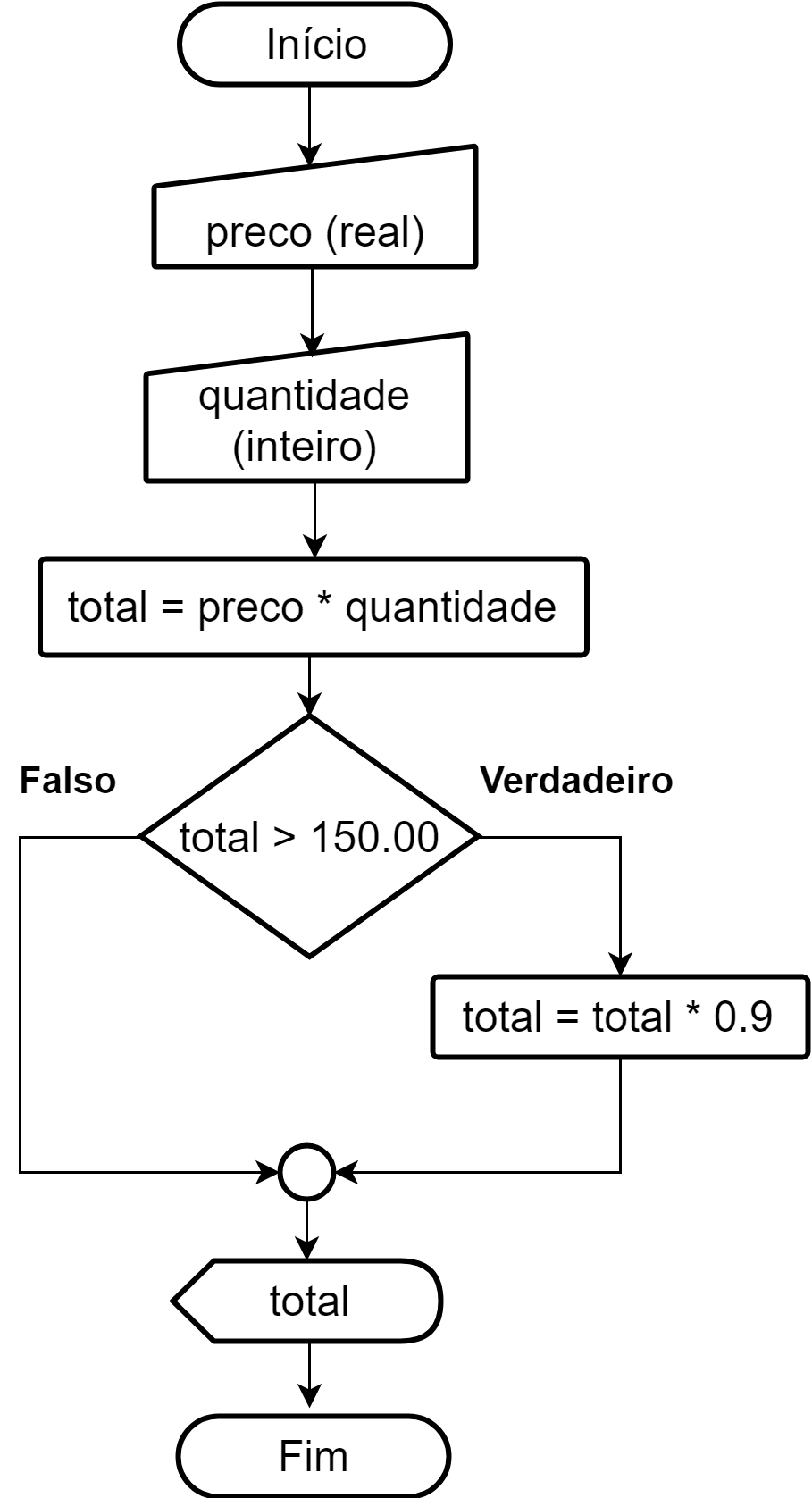
Existe outra estrutura, chamada de **estrutura de seleção composta,** em que ainda temos apenas uma condição a ser avaliada, porém **dois caminhos** que podem ser seguidos. Um dos caminhos leva à execução do bloco de instruções condicionado ao resultado **verdadeiro** e o outro à execução do bloco de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

**1.7. 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 em um valor lógico verdadeiro. Caso a condição resulte falsa, o bloco será ignorado e as instruções posteriores, externas à estrutura condicional, serão executadas.

**Figura 4.1. Fluxograma de estrutura de seleção simples. Fonte Autor.**

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 pode ser representado como a seguir:



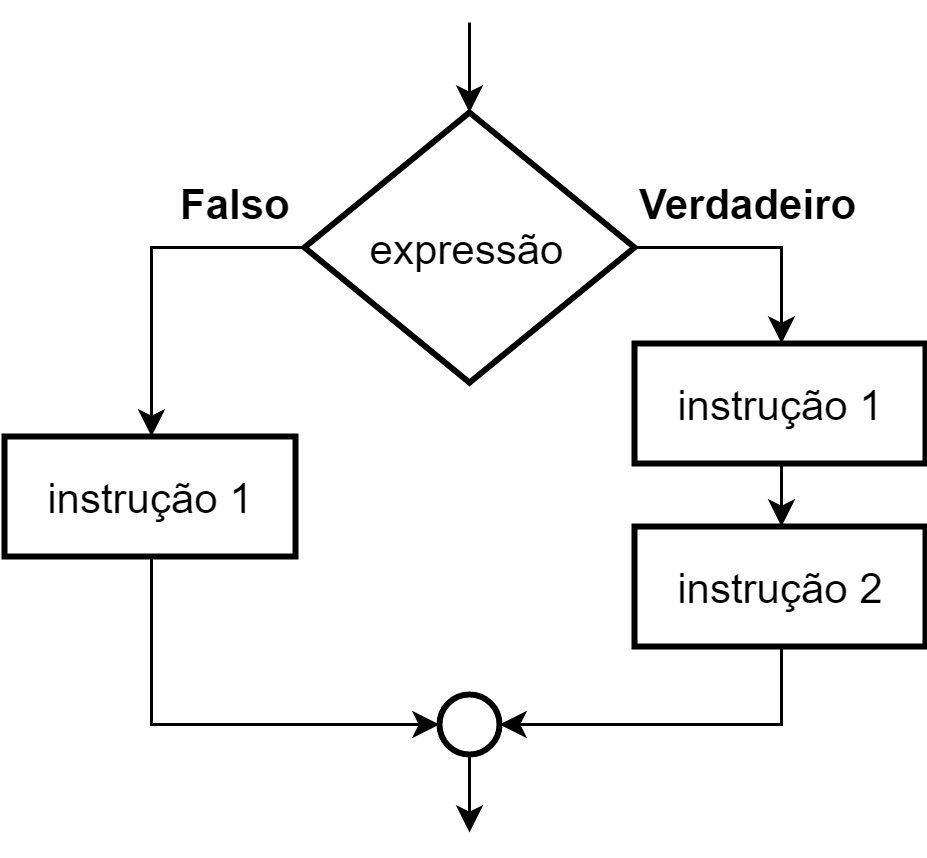
**Figura 4.2. Fluxograma referente à situação exemplo. Fonte Autor.**

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

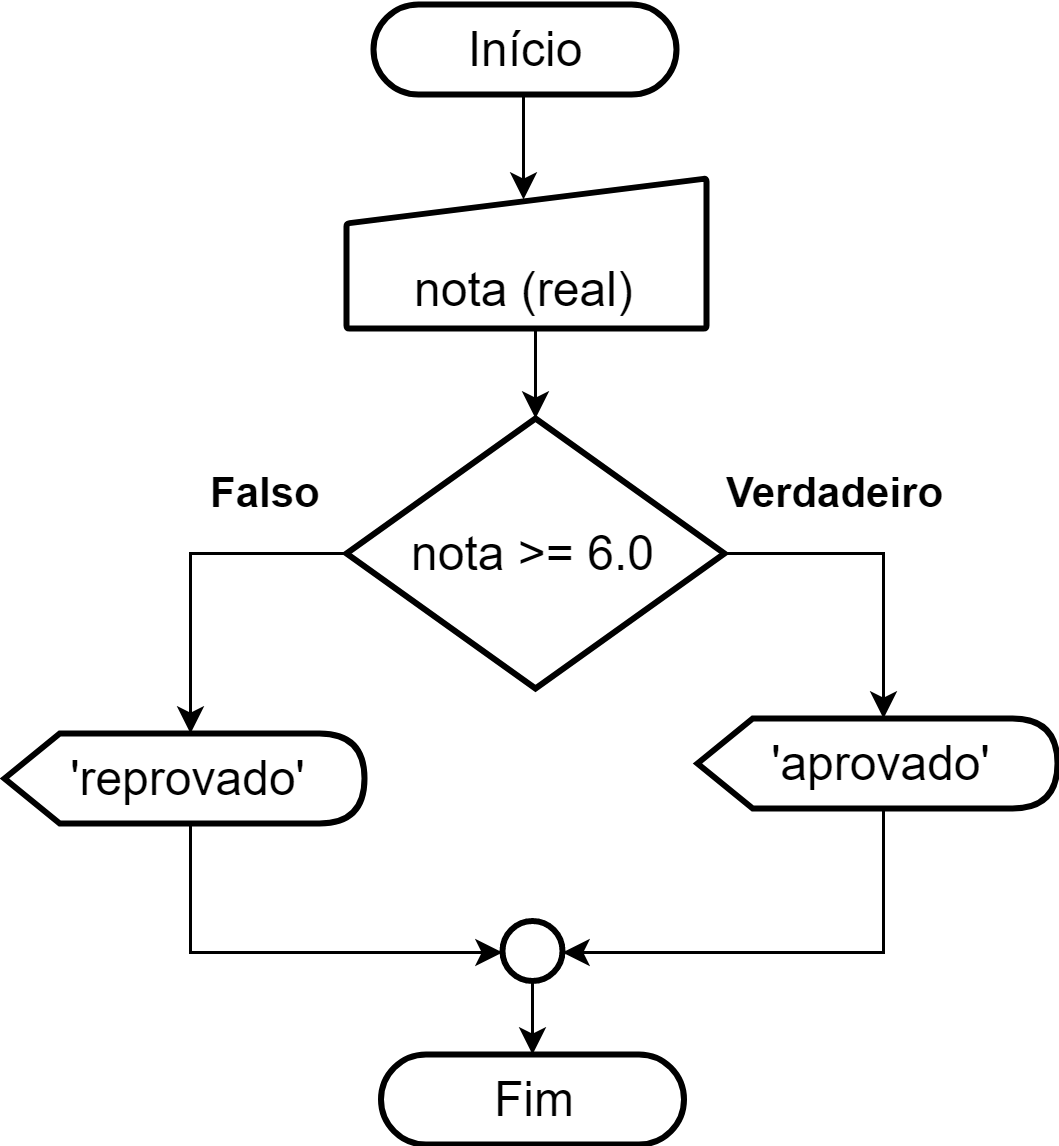
**1.8. Estrutura de seleção composta**

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 em um valor lógico ***verdadeiro.*** Caso a condição resulte ***falso***, outro bloco de códigos será executado.

**Figura 4.3. Exemplo de fluxograma de estrutura de seleção composta. Fonte Autor.**

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:

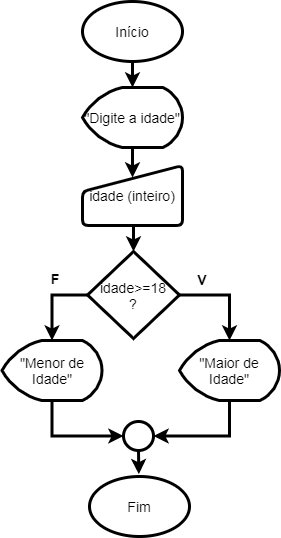


**Figura 4.4. Fluxograma referente à situação exemplo de nota e aprovação. Fonte Autor.**

Note que neste caso somente uma mensagem será exibida. Se a condição da seleção resultar **verdadeiro**, será exibido 'aprovado', caso contrário será exibido 'reprovado'. Costumeiramente, chamamos o lado que é executado quando a condição resulta falso de "**senão**".

Exemplo 1:

Elaborar um Fluxograma que recebe uma idade e exibe a mensagem ***Maior de Idade***, caso a idade seja maior ou igual a 18 e exiba a mensagem ***Menor de Idade,*** caso contrário.

****

**Figura 4.5. Fluxograma referente ao exemplo 1. Fonte Autor.**

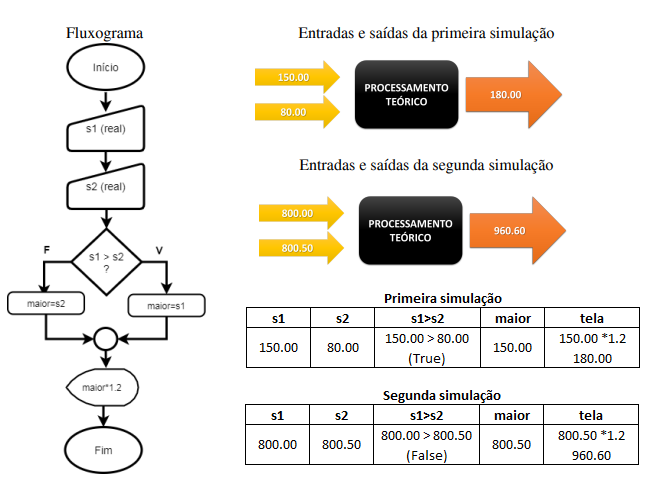
Saídas:

Nesse exemplo, veja que a saída são duas mensagens: Maior de Idade (caso a idade seja maior ou igual a 18) e menor de idade (caso a idade seja menor que 18).

Entradas:

Temos uma entrada - idade, que é um número inteiro.

Na criação do fluxograma, percebemos que primeiro fizemos a entrada de dados - idade e após isso utilizamos uma estrutura de seleção justamente para definir qual das duas mensagens será exibida dependendo do resultado do teste lógico envolvendo a idade (idade>=18). Caso o teste seja verdadeiro, significa que a idade digitada pelo usuário é maior ou igual a 18. Sendo assim, seguimos com o caminho da direita e será exibida a mensagem “Maior de idade”. Caso o resultado do teste seja falso, será exibida a mensagem “Menor de idade”. Perceba que quando usamos uma estrutura de seleção nunca serão usados os dois caminhos simultaneamente. Se usamos o caminho do verdadeiro, o caminho do falso não é usado e vice-versa.



**Figura 4.6. Fluxograma e esquemas de entrada e saída referentes ao exemplo 1. Fonte Autor.**

Nesse exemplo temos 2 entradas (os dois salários) que foram armazenadas em duas variáveis, s1 e s2. Criamos uma terceira variável com o objetivo de armazenar o maior salário chamada maior.

Caso o primeiro salário seja maior que o segundo (s1>s2), a variável maior recebe o valor do primeiro salário (maior=s1). Receberá o segundo salário caso esse teste s1>s2 seja falso - (maior=s2). Ao final, após a estrutura de seleção, é apresentado o valor da variável maior multiplicado por 1.2 (aumento de 20%) na tela.

**Figura 4.7. Diagrama de entradas e saídas do exemplo 1. Fonte Autor.**

No diagrama acima estamos mostrando também dois testes e seus resultados teóricos.Veja nos teste abaixo temos como entradas 150.00 e 80.00 e veja que na saída é apresentado o valor 180.00 (150\*1.2).

Podemos seguir a simulação desse teste pelo quadro abaixo:

**Tabela 4.6. Simulação dos valores das variáveis do exemplo 1.**

| **s1** | **s2** | **s1 > s2** | **maior** | **tela** |
| --- | --- | --- | --- | --- |
| 150.00 | 80.00 | 150.00>80.00  (True) | 150.00 | 150.00 \*1.2  180.00 |

Nesse quadro notamos que são mostrados em colunas as variáveis s1,s2 e maior, a expressão lógica da decisão s1>s2 e o que é exibido na tela.

A simulação ocorre de cima para baixo. Então, temos:

1 - a variável s1 recebe 150.00 (digitado pelo usuário)

2 - a variável s2 recebe 80.00 (digitado pelo usuário)

3 - é feito um teste s1>s2 (150.00>80.00) resultando em Verdadeiro (o fluxo toma o caminho da direita

4 - a variável maior recebe 150,00 que é o valor da variável s1

5 - A estrutura de seleção finaliza e é exibo cálculo maior\*1.2 (150.00\*1.2) que apresenta na tela o valor 180.00

6- O Fluxograma finaliza

O segundo teste com os valores 800.00 e 800.50 ocorre de maneira semelhante.

Observe que neste exemplo o objeto de escolha da estrutura de seleção foram comandos de atribuição (cálculos), enquanto que no exemplo anterior foram comandos de saída.

Exemplo 2

Vamos elaborar um fluxograma de um programa que recebe um número inteiro e exibe uma mensagem indicando se ele é par ou ímpar (vamos usar o operador % para obter o resto de uma divisão inteira).

| Fluxograma | Entrada e saída da primeira simulação  Entrada e saída da segunda simulação  Primeira simulação   | **n** | **n % 2 == 0** | **Tela** | | --- | --- | --- | | 13 | 13 % 2 == 0  (False) | ímpar |   Segunda simulação   | **n** | **n % 2 == 0** | **Tela** | | --- | --- | --- | | 256 | 256 % 2 == 0  (True) | par | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

**Figura 4.8. Fluxograma e diagramas de entrada e saída referentes ao exemplo 2. Fonte Autor.**

Nesse exemplo, diferentemente aos anteriores, veja que na decisão foi colocado um cálculo (n%2==0). A expressão lógica é avaliada depois do cálculo (n%2). Esse é o resto de n que foi digitado pelo usuário por 2. Quando obtemos o resto da divisão por 2 igual a 0 significa que o número é par. Caso seja 1 é ímpar.

**1.9. Conversão para Python**

| Fluxograma | Python |
| --- | --- |
|  | n=int (input ( ) )  if n%2==0:  print (“PAR”)  else:  print (“IMPAR”) |

**Figura 4.9. Exemplo de fluxograma e sua conversão para código Python. Fonte Autor.**

Veja a equivalência dos comandos:

| Fluxograma | Python | Significado |
| --- | --- | --- |
|  | n=int (input ( ) ) | Entrada (número inteiro) |
|  | if n%2==0:  print (“PAR”) | Estrutura de Seleção Composta - Caminho verdadeiro |
|  | else:  print (“IMPAR”) | Estrutura de Seleção Composta - Caminho falso |

**Figura 4.10. Estruturas do fluxograma e suas respectivas conversões para código Python. Fonte Autor.**

**1.10. Referências**

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