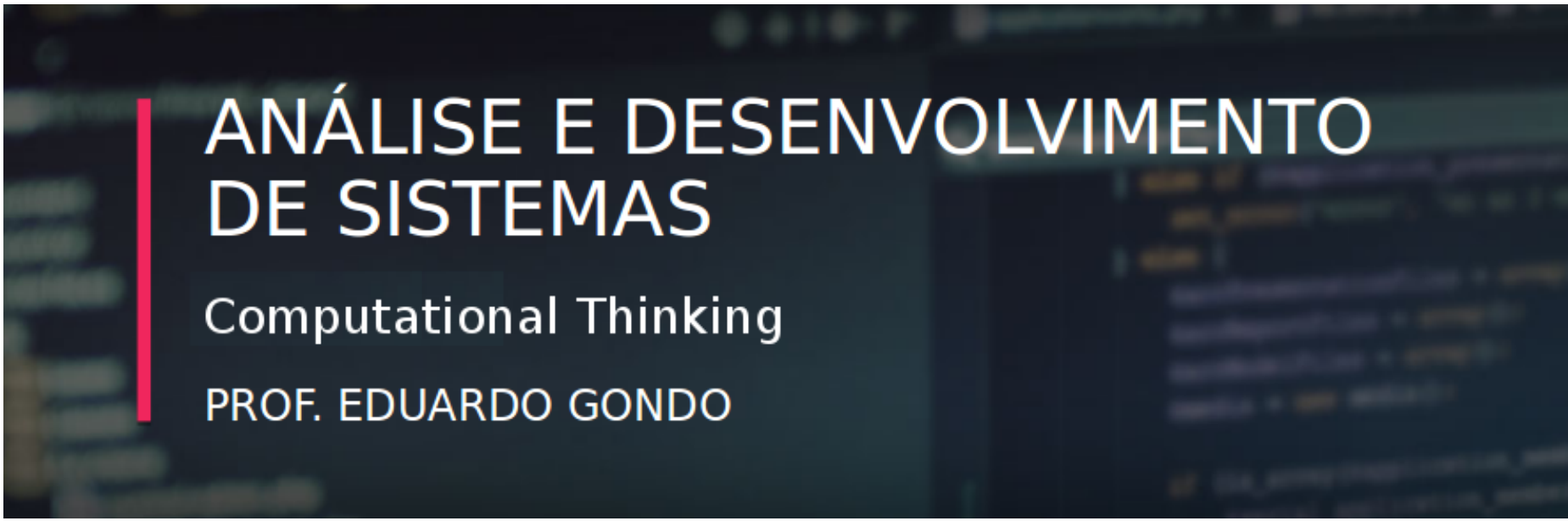
FIAP GRADUAÇÃO


ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

Agenda

- Revisão dos conceitos
- Comando de seleção ou decisão (if)
- Operadores condicionais
- Tipo de variável booleana
- Conectores lógicos

Revisando conceitos

- trabalhamos com variáveis de números inteiros e reais
- também vimos os operadores aritméticos

operador	Python
soma	+
subtração	-
multiplicação	*
divisão real	/
potência	**
resto da divisão	%
divisão inteira	//

- também trabalhamos com os comandos de entrada e saída:

```
1  <var> = input("texto")
2  print(<arg_1>, <arg_2>, <arg_3>,...)
```

- onde <var> é uma variável e <arg_n> são valores/variáveis a serem impressas numa única linha

Revisando conceitos

- vimos o comando = que serve para atribuirmos um valor a uma variável

```
1  produtoria = 1
2  soma = 0.0
3  total = produtoria + soma
```

- e uma das instruções mais importantes que é a do **acumulador**

```
1  soma = soma + num
```

- se você olhar da direita para a esquerda fica mais fácil a compreensão dessa instrução
- primeiro faço a adição (poderia ser qualquer operação aritmética) e depois atribuo o resultado na variável soma

Problema 3.1

PROBLEMA 3.1: Um número inteiro pode ser par ou ímpar. Escreva um algoritmo que recebe um números inteiro e imprime na tela a informação sobre sua paridade.

Problema 3.1

PROBLEMA 3.1: Um número inteiro pode ser par ou ímpar. Escreva um algoritmo que recebe um números inteiro e imprime na tela a informação sobre sua paridade.

```
1  entrada = input("Digite um numero inteiro")
2  num = int(entrada)
3  resto = num % 2
4  if resto == 0:
5      print(num, "e par")
6  else:
7      print(num, "e impar")
```

explicando algoritmo do problema 3.1

- o comando `if` seleciona o conjunto de instruções que serão executadas de acordo com uma *condição*
- uma *condição* é uma operação envolvendo operadores condicionais que resulta em verdadeiro (`True`) ou falso (`False`)
- no algoritmo anterior temos na linha 04 o comando `if`, se a condição `resto == 0` for verdadeira é executado a linha 05 do algoritmo
- se ela for falsa o algoritmo executa a instrução da linha 7
- podemos combinar a condição com a expressão aritmética:

```
1  if num % 2 == 0:
2      print(num, "e par")
3  else:
4      print(num, "e impar")
```

- neste caso a variável resto pode ser removida do algoritmo

Bloco de instruções

- vamos fazer uma pausa e explicar o que são blocos de instruções
- um bloco de instruções é um conjunto de instruções da linguagem de programação
- em muitas linguagens, um bloco de instruções é definido através dos caracteres { e }
- o { inicia um bloco de instruções e o } encerra ele
- no Python, é o alinhamento à esquerda que define um bloco de instruções
- chamamos de **indentação** esse alinhamento, portanto esse recuo é obrigatório no momento que queremos definir um bloco de instruções em Python

Importante!

A indentação é o que define um bloco de instruções na linguagem Python

Bloco de instruções: Exemplos

```python
#indentacao correta
texto = input("Idade:")
idade = int(texto)

if idade < 18:
  print("Voce e menor de
      idade")

print(f"Voce tem {idade}")
```

```python
#indentacao errada
texto = input("Idade:")
   idade = int(texto)

   if idade < 18:
       print("Voce e menor
           de idade")

print(f"Voce tem {idade}")
```

- no exemplo de código cuja indentação é incorreta, ocorre um erro no momento da execução do programa

Bloco de instruções e comando if

- até o momento, todos nossos programas tinham apenas um único bloco de instruções
- mas com o a inclusão dos comandos de decisão, nossos programas terão mais de um bloco
- vejamos algumas ilustrações comparando algumas possibilidades que teremos nos nossos algoritmos

```
1  #comando if
2  if <condicao>:
3    instrucao_1
4    instrucao_2
5    ...
6    instrucao_n
7
8  instrucao_fora_do_bloco
9  ...
```

as instrucões 1, 2 e n só serão executadas se a <condicao> retornar True caso contrário elas serão ignoradas e o fluxo do programa segue com `instrucao_fora_do_bloco`

O espaçamento à esquerda nas linhas 3 a 6 é o que define quais instruções estão dentro do bloco do `if`

Comando de seleção if

Vamos continuar com o exemplo anterior:

```
1  #comando if
2  if <condicao>:
3    instrucao_1
4    instrucao_2
5    ...
6    instrucao_n
7
8  instrucao_fora_do_bloco
9  ...
```

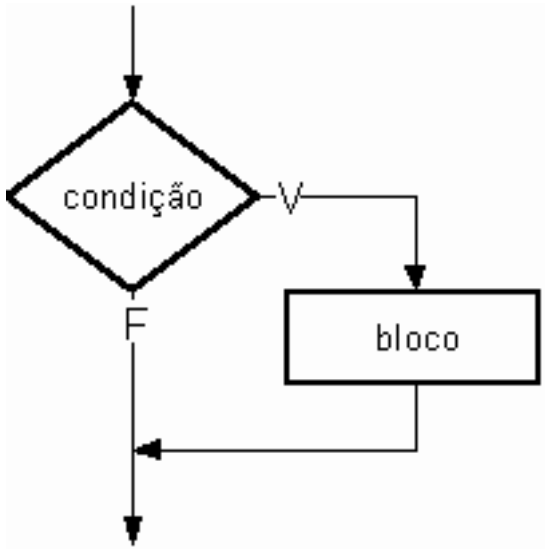


Figura: Fluxograma if

Comando if/else

- o comando if combinado com else permite que seja executado dois blocos distintos de instruções
- o bloco definido pelo `if` é executado quando a condição retorna `True`
- e o bloco do `else` é executado quando a condição retorna `False`

```
1  #comando if/else
2  if <condicao>:
3    instr_if_1
4    ...
5    instr_if_n
6  else:
7    instr_else_1
8    ...
9    instr_else_n
10
11 instr_fora_bloco
12 ...
```

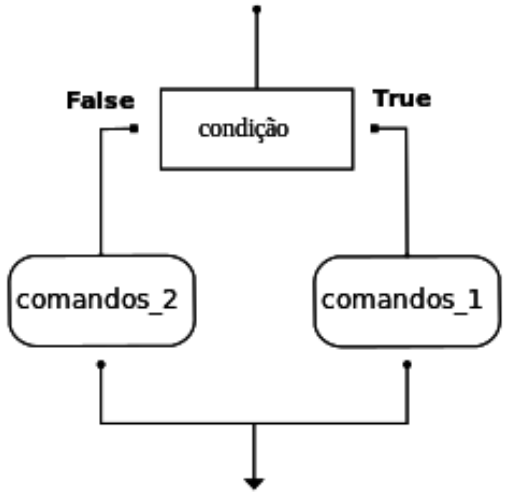


Figura: Fluxograma if/else

Operadores condicionais

- uma `<condicao>` é uma expressão booleana que pode retornar dois valores: verdadeiro ou falso
- para criar expressões booleanas usamos os seguintes operadores:

significado	Python
igual	==
menor	<
maior	>
diferente	!=
menor ou igual	<=
maior ou igual	>=

Tabela: Operadores condicionais

Expressões booleanas

Considere $a = 5$, $b = 2$ e $c = 7$; julgue as seguintes expressões em Python:

- $a < b$
- $a - c >= b$
- $3 * b < c$
- $c - b! = a$
- $c == a + b$
- $c \ \% \ b < a \ // \ b$
- $b + c >= 2 * a$
- $b - c == a$

Comandos de decisão encadeados

- algumas vezes nos deparamos com problemas em que nem o `if` e nem o `if/else` são suficientes
- quando isso acontece, devemos encadear os comandos de decisão
- vamos ver isso através de um exemplo:

EXEMPLO: Suponha que você está escrevendo um programa para verificar o vencedor de uma partida de futebol. A entrada do seu programa são 2 números inteiros não negativos representando a quantidade de gols do Time A e a quantidade de gols do Time B, respectivamente. Você deverá exibir na tela o time ganhador da partida ou se houve empate.

Solução

```
1  #entrada de dados
2  placarA = int(input("Gols do
       time A:")
3  placarB = int(input("Gols do
       time B:")
4
5  #decidindo resultado
6  if placarA == placarB:
7     print("Empate")
8  else:
9     if placarA > placarB:
10       print("Time A ")
11    else:
12       print("Vencedor Time B"
             )
13
14 print("Fim do programa")
```

- na entrada de dados, estamos combinando as instruções `input` e `int`
- agora no comando `if`, observe a indentação do código
- a indentação delimita as instruções que estão dentro dos blocos definidos pelo `if` e `else`
- contudo, nessa situação de encadeamento de if, podemos usar o comando `elif`

Solução com elif

```
1  #entrada de dados
2  placarA = int(input("Gols do time A:")
3  placarB = int(input("Gols do time B:")
4
5  #decidindo resultado
6  if placarA == placarB:
7     print("Empate")
8  elif placarA > placarB:
9     print("Time A ")
10 else:
11    print("Vencedor Time B")
12
13 print("Fim do programa")
```

- não há limite para o comando `elif`, mas ele só pode aparecer após uma instrução `if`
- o comando `else`, se necessário, só aparece no "fim"
- veja nos próximos eslaides alguns exemplos:

Problema 3.2 - Solução Python

Problema 3.2: Escreva um algoritmo que recebe um número e imprime na tela a informação que ou o número é positivo ou é negativo ou é igual a zero.

Problema 3.2 - Solução Python

PROBLEMA 3.2: Escreva um algoritmo que recebe um número e imprime na tela a informação que ou o número é positivo ou é negativo ou é igual a zero.

```
1  num = int(input("Digite um numero: "))
2  if num > 0:
3      print(f"{num} e positivo")
4  elif num < 0:
5      print(f"{num} e negativo")
6  else:
7      print(str(num) + " e zero")
```

- veja que podemos converter um número inteiro ou real em uma String através da função `str`

Problema 3.3

PROBLEMA 3.3: Escreva um algoritmo que recebe dois números e um caractere (representando uma das operações matemáticas(+,-,*,/)) e calcula o valor da operação matemática, ou seja, se a entrada for 5, $*$ e 6 então seu programa deverá mostrar 30.

Algoritmo Problema 3.3 - Python

```python
valor = input("Digite numero: ")
numA = float(valor)
op = input("Operador (+-*/): ")
valor = input("Digite numero: ")
numB = float(valor)

if op == "+":
    resultado = numA + numB

if op == "-":
    resultado = numA - numB

if op == "*":
    resultado = numA * numB

if op == "/":
    resultado = numA / numB

print(f"Resposta {resultado}")
```

Algoritmo Problema 3.3 - Python

Agora a solução usando `if`, `elif` e `else`:

```
1  valor = input("Digite numero: ")
2  numA = float(valor)
3  op = input("Operador (+-*/): ")
4  valor = input("Digite numero: ")
5  numB = float(valor)
6
7  if op == "+":
8      resultado = numA + numB
9  elif op == "-":
10     resultado = numA - numB
11 elif op == "*":
12     resultado = numA * numB
13 elif op == "/":
14     resultado = numA / numB
15
16 print(f"Resposta {resultado}")
```

Algumas considerações

- Nas linhas de 03 a 06 do algoritmo, executamos todas as comparações das operações matemáticas
- Desse modo, as comparações no comando `if` e `elif` se tornam mais legíveis
- Na linha 17 temos a instrução `print(f"Resp: {resultado})`
- Conhecido como **f-strings**
- o `{resultado}` é substituído pelo conteúdo da variável resultado
- dentro dessas chaves, é possível definir a formatação do valor da variável
- por exemplo, coloque `:.2f` entre as chaves e veja o que acontece

Conectores lógicos

- algumas vezes precisamos combinar expressões lógicas
- os conectores lógicos **and** e **or** são os responsáveis por essa combinação no Python
- note que o resultado da expressão lógica combinada continua sendo verdadeiro ou falso
- por exemplo, considere $a = 5$, $b = 7$, $c = 4$ e $d = 8$, julgue verdadeiro ou falso as seguintes expressões:

a) $(a \leq b)$ *and* $(b < d)$
b) $(a = b)$ *or* $(c \neq b)$
c) $(d > a)$ *and* $(c \geq b)$
d) $(a \leq b)$ *or* $(c \leq d)$
e) $((b > c)$ *or* $(c < a))$ *and* $(d \leq b)$

Conectores lógicos

- algumas vezes precisamos combinar expressões lógicas
- os conectores lógicos **and** e **or** são os responsáveis por essa combinação no Python
- note que o resultado da expressão lógica combinada continua sendo verdadeiro ou falso
- por exemplo, considere $a = 5$, $b = 7$, $c = 4$ e $d = 8$, julgue verdadeiro ou falso as seguintes expressões:

a) $(a \leq b)$ *and* $(b < d)$
b) $(a = b)$ *or* $(c \neq b)$
c) $(d > a)$ *and* $(c \geq b)$
d) $(a \leq b)$ *or* $(c \leq d)$
e) $((b > c)$ *or* $(c < a))$ *and* $(d \leq b)$

Gabarito: a) V b) V c) F d) V e) F

Conectores lógicos

Segue uma tabela ilustrando os conectores lógicos em Python e em Java:

Signifcado	Python	Java
não	not	!
e	and	&&
ou	or	||

Tabela: conectores lógicos

Tabela verdade

- sejam x e y duas expressões lógicas
- cada uma dessas expressões pode assumir o valor **verdadeiro** ou **falso**
- para os dois valores possíveis de x e y vamos construir uma tabela com x *and* y e x *or* y

x	y	x and y	x or y	not x
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Tabela: Tabela verdade

Problema 3.4

PROBLEMA 3.4: Escreva um algoritmo que lê o salário de um funcionário e mostra qual o percentual de desconto que será aplicado para sua contribuição ao INSS. Use a tabela abaixo para calcular o desconto:

salário contribuição	alíquota/valor
até R$ 1.693,72	8%
de R$ 1.693,73 até R$ 2.822,90	9%
de R$ 2.822,91 até R$ 5.645,80	11%
acima de R$ 5.645,80	11% sobre R$ 5.645,80

Tabela: Contribuição INSS 2019

Por exemplo, um trabalhador com salário de R$ 2.000,00 o percentual de desconto será de 9%. Quem ganha R$ 8.000,00 terá um desconto de 11% sobre o teto da aposentadoria.

Algoritmo Problema 3.4

```
1  salario = float(input("Digite o salario"))
2  if salario >= 0 and salario <= 1693.72:
3      contribuicao = salario * 0.08
4
5  if salario >= 1693.73 and salario <= 2822.90:
6      contribuicao = salario * 0.09
7
8  if salario >= 2822.91 and salario <= 5645.80:
9      contribuicao = salario * 0.11
10
11 if salario  > 5645.80:
12     contribuicao = 5645.80 * 0.11
13
14 print("O valor do INSS sera de R$ {%:2f}".format(
       contribuicao))
```

Problema 3.5

PROBLEMA 3.5 Escreva um algoritmo que recebe três números inteiros e imprime eles em ordem crescente.

```
1  numa = int(input("digite 1 num: "))
2  numb = int(input("digite 2 num: "))
3  numc = int(input("digite 3 num: "))
4  if (numa <= numb) and (numb <= numc):
5      print(numa, " ", numb, " ", numc)
6
7  if (numa <= numc) and (numc <= numb):
8      print(numa, " ", numc, " ", numb)
9
10 if (numb <= numa) and (numa <= numc):
11     print(numb, " ", numa, " ", numc)
12
13 if (numb <= numc) and (numc <= numa):
14     print(numb," ", numc , " ", numa)
15
16 if (numc <= numa) and (numa <= numb):
17     print(numc, " ", numa, " ", numb)
18
19 if (numc <= numb) and (numb <= numa):
20     print(numc, " ", numb, " ", numa)
```

Problema 3.5

Há um problema no algoritmo anterior. Quando há repetição de números, ele apresenta mais de uma vez a resposta. Nesse caso, a solução seria encadear os comandos de decisão:

```
1  numa = int(input("digite 1 num: "))
2  numb = int(input("digite 2 num: "))
3  numc = int(input("digite 3 num: "))
4  if (numa <= numb) and (numb <= numc):
5      print(numa, " ", numb, " ", numc)
6  elif (numa <= numc) and (numc <= numb):
7      print(numa, " ", numc, " ", numb)
8  elif (numb <= numa) and (numa <= numc):
9      print(numb, " ", numa, " ", numc)
10 elif (numb <= numc) and (numc <= numa):
11     print(numb," ", numc , " ", numa)
12 elif (numc <= numa) and (numa <= numb):
13     print(numc, " ", numa, " ", numb)
14 else:
15     print(numc, " ", numb, " ", numa)
```

Comando **match/case**

- disponível apenas a partir da versão 3.10 do Python
- o comportamento do comando **match/case** é parecido com o **if**
- porém este comando somente é adequado quando a comparação é uma igualdade e os dados são discretos: números inteiros, string
- no exemplo 3.3 verificamos se o operador é igual as strings: +, -, * e /
- quando há a igualdade é executado o código que está no bloco case do valor correspondente
- o _ identifica a situação *default, ou seja, quando não há igualdade com nenhum padrão*

Comando **match/case** — Exemplo

```
1       op = '+'
2       match op:
3           case '+':
4             print('operador +')
5           case '-':
6             print('operador -')
7           case '*':
8             print('operador *')
9           case '/':
10            print('operador /')
11          case _:
12         print('operador nao definido')
```

Comando **match/case** — conclusão

O comando **match/case** possui algumas limitações, por exemplo em uma situação que você precisa saber se um número está entre dois números devemos utilizar o comando **if**.

O comando **if** é um comando mais genérico, ou seja, tudo que é feito com o **match/case** pode ser feito com o **if** mas o inverso não é verdade, ou seja, não podemos escrever todos comandos **if** usando o **match/case**, por exemplo ele não funciona com intervalo de números como o caso da tabela de salários.

Exercícios

Faça os exercícios da lista de exercícios 3!

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft

Copyleft © 2025 Prof. Eduardo Gondo Todos direitos liberados.
Reprodução ou divulgação total ou parcial deste documento é liberada.