

**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



**MC102 - Aula 03**

**Comandos Condicionais**

Algoritmos e Programação de Computadores

Turmas

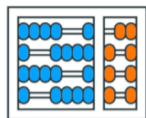
**OVXZ**

**Prof. Lise R. R. Navarrete**

[lrommel@ic.unicamp.br](mailto:lrommel@ic.unicamp.br)

Quinta-feira, 24 de março de 2022

19:00h - 21:00h (CB06)



**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



UNICAMP

**MC102 – Algoritmos e Programação de Computadores**

---

Turmas

**OVXZ**

<https://ic.unicamp.br/~mc102/>

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

# Conteúdo

- Expressões Relacionais
- Expressões Lógicas
- Comandos Condicionais

# Expressões Relacionais

- Já vimos que o tipo **bool** é utilizado para representar os valores booleanos verdadeiro (**True**) e falso (**False**).

```
1 a = True
2 b = False
```

- O uso mais comum dessas variáveis é na verificação de expressões relacionais e lógicas.

- Expressões relacionais são aquelas que realizam uma comparação entre duas expressões e retornam:
  - **True**, se o resultado for verdadeiro.
  - **False**, se o resultado for falso.
- Os operadores relacionais são:
  - == igualdade.
  - != diferente.
  - > maior que.
  - < menor que.
  - >= maior ou igual que.
  - <= menor ou igual que.
- Nos próximos exemplos, considere que foram feitas as seguintes atribuições:

```
1 a = 20
2 b = 21
```

- `<expressão> == <expressão>`: retorna verdadeiro quando as expressões forem iguais.

```
1 a == (10 * 2) # a = 20
2 # True
3 b == (10 * 2) # b = 21
4 # False
```

- `<expressão> != <expressão>`: retorna verdadeiro quando as expressões forem diferentes.

```
1 a != (10 * 2) # a = 20
2 # False
3 b != (10 * 2) # b = 21
4 # True
```

- `<expressão> > <expressão>`: retorna verdadeiro quando a expressão da esquerda tiver valor maior que a expressão da direita.

```
1 # a = 20 e b = 21
2 a > b
3 # False
```

- `<expressão> < <expressão>`: retorna verdadeiro quando a expressão da esquerda tiver valor menor que a expressão da direita.

```
1 # a = 20 e b = 21
2 a < b
3 # True
```



- $\langle \text{expressão} \rangle \geq \langle \text{expressão} \rangle$ : retorna verdadeiro quando a expressão da esquerda tiver valor maior ou igual que a expressão da direita.

```
1 # a = 20 e b = 21
2 a >= b
3 # False
```

- $\langle \text{expressão} \rangle \leq \langle \text{expressão} \rangle$ : retorna verdadeiro quando a expressão da esquerda tiver valor menor ou igual que a expressão da direita.

```
1 # a = 20 e b = 21
2 a <= b
3 # True
```

- Ordem considerada para os caracteres do alfabeto:

• *ABC...XYZabc...xyz*

menor

maior

```
1 "a" > "b"  
2 # False  
3 "a" == "a"  
4 # True  
5 "a" == "A"  
6 # False  
7 "Z" < "a"  
8 # True  
9 "z" < "a"  
10 # False  
11
```

- Ordem considerada para os caracteres do alfabeto:

• *ABC...XYZabc...xyz*

menor

maior

```
1 "azzzz" < "zaaaa"  
2 # True  
3 "azzzz" < "Zaaaa"  
4 # False  
5 "3" == 3  
6 # False  
7 3 > "4"  
8 # Traceback (most recent call last):  
9 #   File "<stdin>", line 1, in <module>  
10 # TypeError: '>' not supported between instances of 'int'  
    and 'str'
```

O que será impresso pelo código a seguir?

```
1 print((3 * 4) / 2 == (2 * 3))  
2 # True  
3 print((4 / 3) <= 1.33)  
4 # False
```

Operator	Description	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
is, is not	Identity	left-to-right
in, not in	Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
not	Logical NOT	right-to-left
and	Logical AND	left-to-right
or	Logical OR	left-to-right
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^=  =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	

# Expressões Lógicas

- Expressões lógicas são aquelas que realizam uma operação lógica e retornam verdadeiro ou falso (como as expressões relacionais).
- Os operadores lógicos são:
  - **and**: operador E.
  - **or**: operador OU.
  - **not**: operador NÃO.

- `<expressão1> and <expressão2>`: retorna verdadeiro quando ambas as expressões são verdadeiras.
- Sua tabela verdade é:

<code>&lt;expressão1&gt;</code>	<code>&lt;expressão2&gt;</code>	resultado
True	True	True
True	False	False
False	True	False
False	False	False

```
1 a = 5
2 b = 10
3 print((a > 0) and (b == 0))
4 # False
5 print((a > 0) and (b != 0))
6 # True
```



- `<expressão1> or <expressão2>`: retorna verdadeiro quando pelo menos uma das expressões é verdadeira.
- Sua tabela verdade é:

<code>&lt;expressão1&gt;</code>	<code>&lt;expressão2&gt;</code>	resultado
True	True	True
True	False	True
False	True	True
False	False	False

```
1 a = 5
2 b = 10
3 print((a > 0) or (b == 0))
4 # True
5 print((a != 5) or (b == 0))
6 # False
```

- **not** <expressão>: retorna verdadeiro quando a expressão é falsa e vice-versa.
- Sua tabela verdade é:

<expressão>	resultado
True	False
False	True

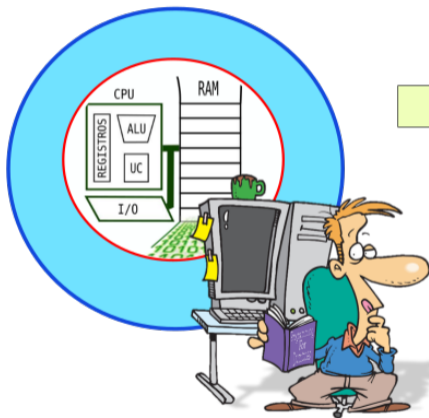
```
1 a = 5
2 b = 10
3 print(not(a < b))
4 # False
5 print(not(a == b))
6 # True
```

- $\text{not}(a == b)$  é equivalente a  $(a != b)$
- $\text{not}(a > b)$  é equivalente a  $(a <= b)$
- $\text{not}(a < b)$  é equivalente a  $(a >= b)$

O que será impresso pelo código a seguir?

```
1 a = True
2 b = False
3 print(not(a or b))
4 # False
5 print(not(a and b))
6 # True
7 print(not(a) and not(b))
8 # False
9 print(not(a) or not(b))
10 # True
```

# Comandos Condicionais



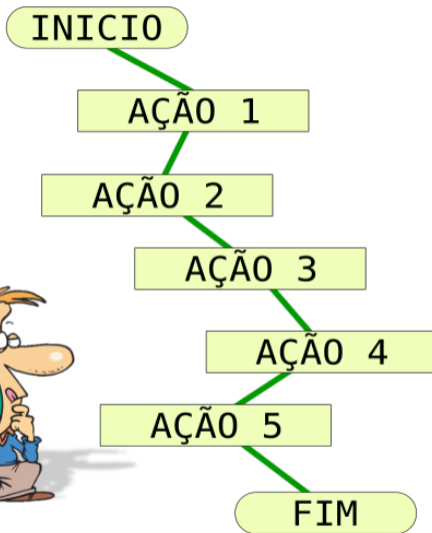
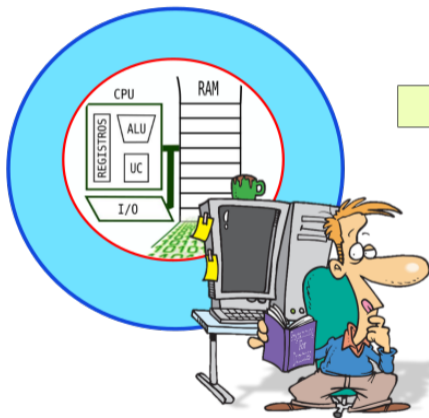
AÇÃO 1

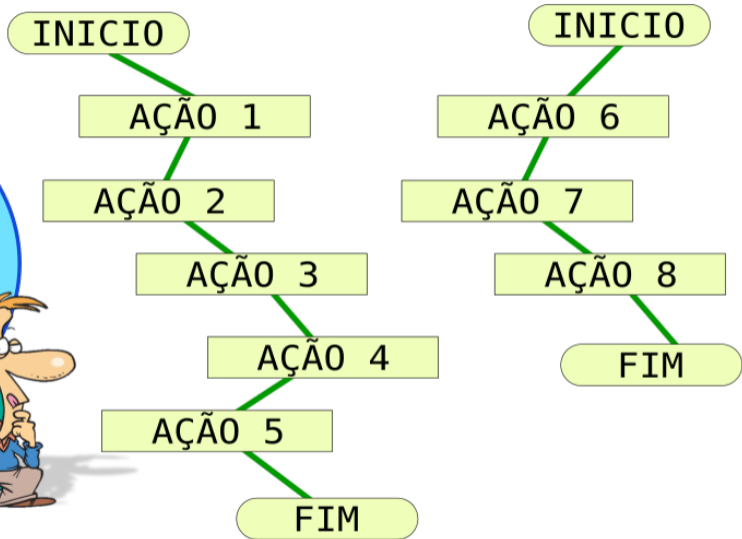
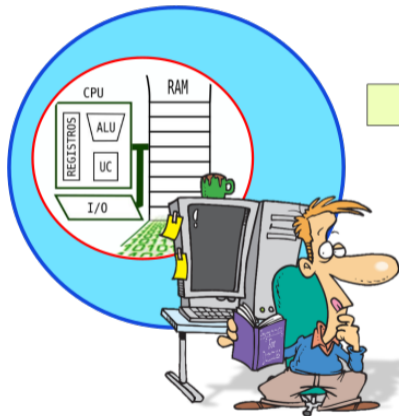
AÇÃO 2

AÇÃO 3

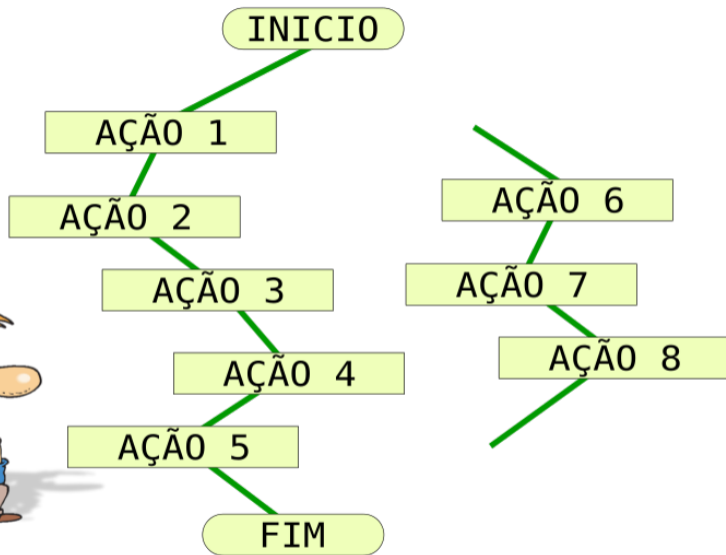
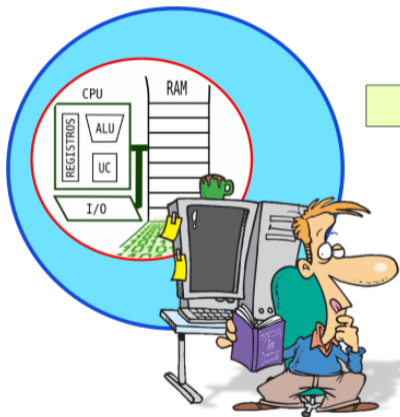
AÇÃO 4

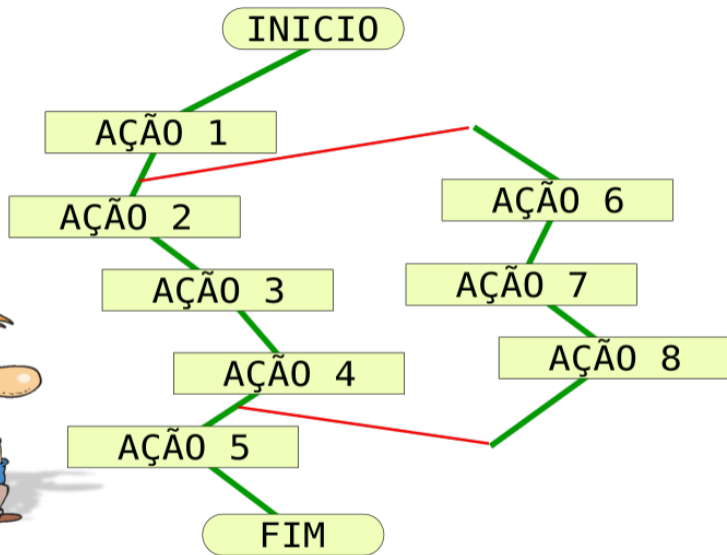
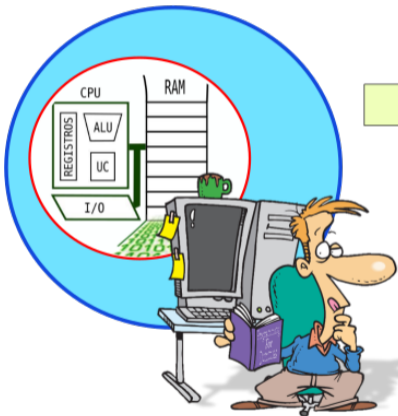
AÇÃO 5

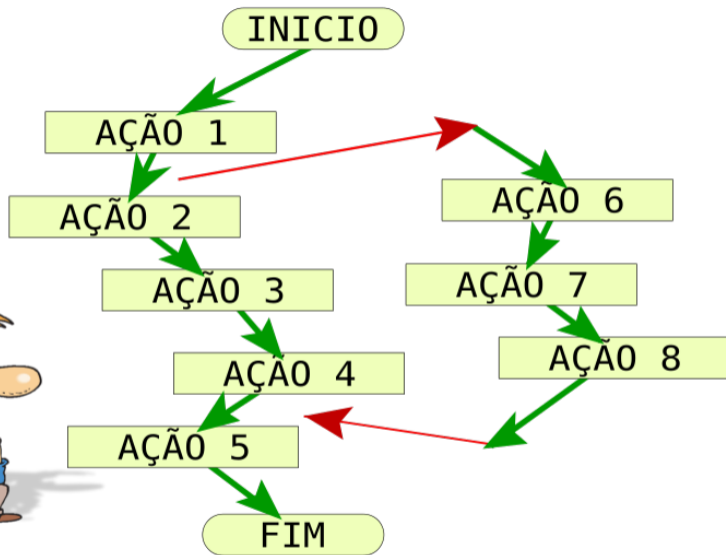
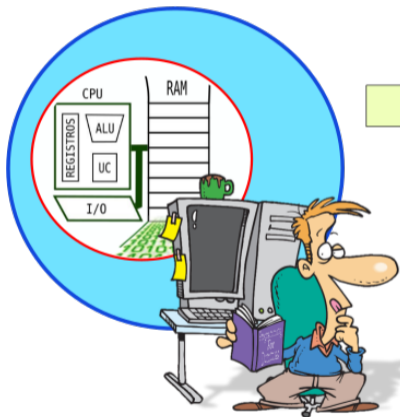


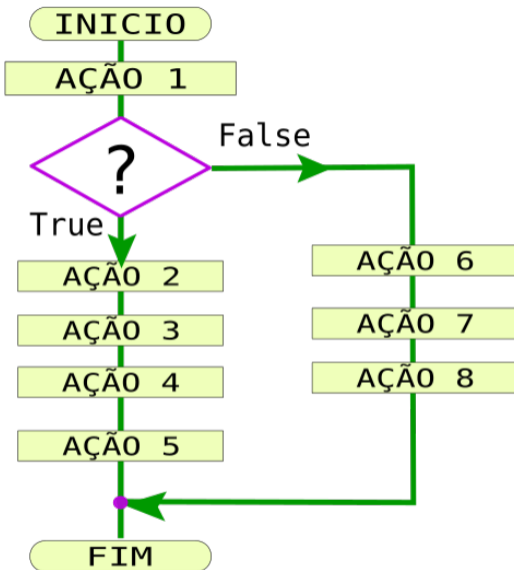
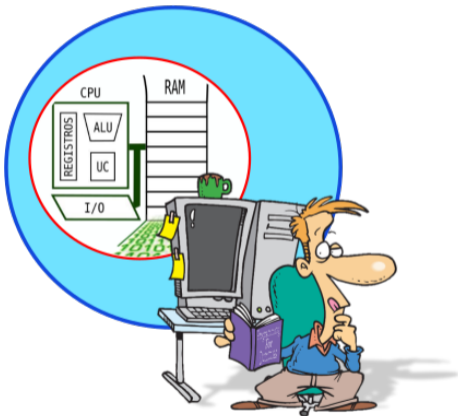














- Um bloco é um conjunto de comandos agrupados.
- Os programas Python são estruturados através de indentação, ou seja, os blocos são definidos pelo seu espaçamento (*tabs*) em relação ao início da linha.

- O principal comando condicional é o `if`:

```
1 if <condição>:  
2 # bloco a ser executado se a condição for verdadeira  
3   <comando1>  
4   <comando2>  
5   ...  
6   <comandoY>
```

- O bloco de comandos é executado somente se a condição (expressão relacional, expressão lógica ou variável booleana) for verdadeira.
- Na estrutura do comando `if` sempre há um “:” após a condição.

- O programa a seguir verifica se um número inteiro é ímpar.

```
1 a = int(input("Digite um número inteiro: "))
2 impar = ((a % 2) == 1)
3 if impar:
4     print("Número ímpar")
5
6 print("Fim do programa")
```

- O programa a seguir verifica se um número inteiro é ímpar.

```
1 a = int(input("Digite um número inteiro: "))
2
3 if (a % 2) == 1:
4     print("Número ímpar")
5
6 print("Fim do programa")
```



- O programa a seguir verifica se um número inteiro é par ou ímpar.

```
1 a = int(input("Digite um número inteiro: "))
2
3 if (a % 2) == 0:
4     print("Número par")
5 if (a % 2) == 1:
6     print("Número ímpar")
7
8 print("Fim do programa")
```

- Uma variação do comando `if` é o `if/else`:

```
1 if <condição>:  
2     # bloco a ser executado se a condição for verdadeira  
3     <comando>  
4     ...  
5     <comando>  
6 else:  
7     # bloco a ser executado se a condição for falsa  
8     <comando>  
9     ...  
10    <comando>
```

- O programa a seguir verifica se um número inteiro é par ou ímpar.

```
1 a = int(input("Digite um número inteiro: "))
2
3 if (a % 2) == 0:
4     print("Número par")
5 else:
6     print("Número ímpar")
7
8 print("Fim do programa")
```

- O programa a seguir determina o maior entre dois números.

```
1 a = float(input("Digite o primeiro número: "))
2 b = float(input("Digite o segundo número: "))
3
4 if a > b:
5     print("O maior número é", a)
6 else:
7     print("O maior número é", b)
```

- O programa a seguir compara dois números.

```
1 a = float(input("Digite o primeiro número: "))
2 b = float(input("Digite o segundo número: "))
3
4 if a == b:
5     print("Os dois números são iguais")
6 else:
7     if a > b:
8         print("O maior número é o primeiro")
9     else:
10        print("O maior número é o segundo")
```

- Podemos simplificar o código anterior utilizando `elif`.

```
1 a = float(input("Digite o primeiro número: "))
2 b = float(input("Digite o segundo número: "))
3
4 if a == b:
5     print("Os dois números são iguais")
6 elif a > b:
7     print("O maior número é o primeiro")
8 else:
9     print("O maior número é o segundo")
10
```

- O comando `elif` é utilizado quando queremos fazer o teste de várias alternativas.

```
1 ra = input("Entre com o RA de um aluno: ")
2 if ra == "155446":
3     print("Gabriel Siqueira")
4 elif ra == "192804":
5     print("Alexsandro Alexandrino")
6 elif ra == "209823":
7     print("Ana Paula Dantas")
8 elif ra == "188948":
9     print("Klairton Brito")
10 # ...
11 elif ra == "999999":
12     print("...")
13 else:
14     print("Aluno não encontrado")
```

```
1 a = int(input())
2
3 if a > 3:
4     if a < 7:
5         print("a")
6 else:
7     if a > -10:
8         print("b")
9     else:
10        print("c")
```

- No código acima, o que será impresso...
  - ... quando  $a = 5$ ? "a".
  - ... quando  $a = 10$ ? Nada.
  - ... quando  $a = -5$ ? "b".
  - ... quando  $a = -15$ ? "c".



```
1 a = int(input())
2
3 if a > 3:
4     if a < 7:
5         print("a")
6     else:
7         if a > -10:
8             print("b")
9         else:
10            print("c")
```

- No código acima, o que será impresso...
  - ... quando  $a = 5$ ? "a".
  - ... quando  $a = 10$ ? "b".
  - ... quando  $a = -5$ ? Nada.
  - ... quando  $a = -15$ ? Nada.

```
1 a = int(input())
2
3 if a > 3:
4     if a < 7:
5         print("a")
6     else:
7         if a > -10:
8             print("b")
9 else:
10    print("c")
```

- No código acima, o que será impresso...
  - ... quando  $a = 5$ ? "a".
  - ... quando  $a = 10$ ? "b".
  - ... quando  $a = -5$ ? "c".
  - ... quando  $a = -15$ ? "c".

1. Escreva um programa que, dados três números inteiros, imprima o menor deles.
2. Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.
3. Escreva um programa que, dadas duas datas, determine qual delas ocorreu cronologicamente primeiro. Para cada uma das duas datas, leia três números referentes ao dia, mês e ano, respectivamente.

- Escreva um programa que, dados três números inteiros, imprima o menor deles.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4
5 if (a <= b) and (a <= c):
6     print(a)
7 if (b <= a) and (b <= c):
8     print(b)
9 if (c <= a) and (c <= b):
10    print(c)
```

- Este programa tem um comportamento indesejado quando o menor número não é único. Como corrigi-lo?

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4
5 if (a <= b) and (a <= c):
6     print(a)
7 if (b <= a) and (b <= c):
8     print(b)
9 if (c <= a) and (c <= b):
10    print(c)
```

- Escreva um programa que, dados três números inteiros, imprima o menor deles.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4
5 if (a <= b) and (a <= c):
6     print(a)
7 elif (b <= c):
8     print(b)
9 else:
10    print(c)
```

- Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4 if (a <= b) and (b <= c):
5     print(a, b, c)
6 elif (a <= c) and (c <= b):
7     print(a, c, b)
8 elif (b <= a) and (a <= c):
9     print(b, a, c)
10 elif (b <= c) and (c <= a):
11     print(b, c, a)
12 elif (c <= a) and (a <= b):
13     print(c, a, b)
14 elif (c <= b) and (b <= a):
15     print(c, b, a)
```

- Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4 if (a <= b) and (b <= c):
5     print(a, b, c)
6 elif (a <= c) and (c <= b):
7     print(a, c, b)
8 elif (b <= a) and (a <= c):
9     print(b, a, c)
10 elif (b <= c) and (c <= a):
11     print(b, c, a)
12 elif (c <= a) and (a <= b):
13     print(c, a, b)
14 else:
15     print(c, b, a)
```



- Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4 if (a <= b <= c):
5     print(a, b, c)
6 elif (a <= c <= b):
7     print(a, c, b)
8 elif (b <= a <= c):
9     print(b, a, c)
10 elif (b <= c <= a):
11     print(b, c, a)
12 elif (c <= a <= b):
13     print(c, a, b)
14 else:
15     print(c, b, a)
```

- Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.

```
1 a = int(input("Digite o primeiro número: "))
2 b = int(input("Digite o segundo número: "))
3 c = int(input("Digite o terceiro número: "))
4
5 if (a <= b) and (a <= c): # 0 menor é o primeiro (a)
6     if (b <= c):
7         print(a, b, c)
8     else:
9         print(a, c, b)
10 elif (b <= c): # 0 menor é o segundo (b)
11     if (a <= c):
12         print(b, a, c)
13     else:
14         print(b, c, a)
15 # ...
```

- Escreva um programa que, dados três números inteiros, imprima os números em ordem crescente.

```
1 # ...
2 else:           # O menor é o terceiro (c)
3     if (a <= b):
4         print(c, a, b)
5     else:
6         print(c, b, a)
7
8
9
10
11
12
13
14
15
```

- Python possui as funções `min` (mínimo) e `max` (máximo).
- A função `min` retorna o menor valor dentre todos os valores passados como argumento.

```
1 a = 5
2 b = 10
3 print(min(100, a, 7, b))
4 # 5
```

- A função `max` retorna o maior valor dentre todos os valores passados como argumento.

```
1 print(max(100, a, 7, b))
2 # 100
```

- Refaça os dois exercícios anteriores sem utilizar comandos condicionais. Dica: use as funções `min` e `max`.

- Escreva um programa que, dadas duas datas, determine qual delas ocorreu cronologicamente primeiro. Para cada uma das duas datas, leia três números referentes ao dia, mês e ano, respectivamente.

```
1 dia1 = int(input("Digite o dia da primeira data: "))
2 mes1 = int(input("Digite o mês da primeira data: "))
3 ano1 = int(input("Digite o ano da primeira data: "))
4
5 dia2 = int(input("Digite o dia da segunda data: "))
6 mes2 = int(input("Digite o mês da segunda data: "))
7 ano2 = int(input("Digite o ano da segunda data: "))
8
9
10
11
12
13 # ...
```

- Escreva um programa que, dadas duas datas, determine qual delas ocorreu cronologicamente primeiro. Para cada um das duas datas, leia três números referentes ao dia, mês e ano, respectivamente.

```
1 # ...
2 if ano1 < ano2:
3     print(dia1, mes1, ano1, sep="/")
4 elif ano2 < ano1:
5     print(dia2, mes2, ano2, sep="/")
6 elif mes1 < mes2:
7     print(dia1, mes1, ano1, sep="/")
8 elif mes2 < mes1:
9     print(dia2, mes2, ano2, sep="/")
10 elif dia1 < dia2:
11     print(dia1, mes1, ano1, sep="/")
12 else:
13     print(dia2, mes2, ano2, sep="/")
```

4. Escreva um programa que calcule as raízes de uma equação de segundo grau. O seu programa deve receber três números  $a$ ,  $b$  e  $c$ , sendo que a equação é definida como  $ax^2 + bx + c = 0$ . O seu programa também deve tratar o caso em que  $a = 0$ .
5. Escreva um programa que simula o jogo conhecido como “Pedra, Papel e Tesoura” de um jogador A contra um jogador B. O programa deve ler a escolha do jogador A e a escolha do jogador B. Por fim, o programa deve indicar quem foi o vencedor.

- Equação do 2º grau:

$$ax^2 + bx + c = 0$$

- Fórmula de Bhaskara:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$\Delta = b^2 - 4ac$$



- Escreva um programa que calcule as raízes de uma equação de segundo grau. O seu programa deve receber três números  $a$ ,  $b$  e  $c$ , sendo que a equação é definida como  $ax^2 + bx + c = 0$ . O seu programa também deve tratar o caso em que  $a = 0$ .

```
1 a = float(input("Digite o coeficiente a: "))
2 b = float(input("Digite o coeficiente b: "))
3 c = float(input("Digite o coeficiente c: "))
4
5
6 if a == 0: # equação do primeiro grau
7     if b == 0:
8         print("Não existe raiz.")
9     else:
10        raiz = (-c / b)
11        print("A raiz é:", raiz)
12 # ...
```

- Escreva um programa que calcule as raízes de uma equação de segundo grau. O seu programa deve receber três números  $a$ ,  $b$  e  $c$ , sendo que a equação é definida como  $ax^2 + bx + c = 0$ . O seu programa também deve tratar o caso em que  $a = 0$ .

```
1 # ...
2 else: # equação do segundo grau
3     delta = (b ** 2) - (4 * a * c)
4     if delta < 0:
5         print("Não existem raízes reais.")
6     elif delta != 0:
7         raiz1 = (-b + delta ** (1 / 2)) / (2 * a)
8         raiz2 = (-b - delta ** (1 / 2)) / (2 * a)
9         print("As raízes são:", raiz1, "e", raiz2)
10    else:
11        raiz = -b / (2 * a)
12        print("A raiz é:", raiz)
```

- Escreva um programa que simula o jogo conhecido como “Pedra, Papel e Tesoura” de um jogador A contra um jogador B. O programa deve ler a escolha do jogador A e a escolha do jogador B. Por fim, o programa deve indicar quem foi o vencedor.

```
1 jogadorA = input("Digite a primeira escolha: ")
2 jogadorB = input("Digite a segunda escolha: ")
3
4 if jogadorA == "pedra":
5     if jogadorB == "pedra":
6         print("Empate")
7     elif jogadorB == "tesoura":
8         print("O jogador A ganhou")
9     else:
10        print("O jogador B ganhou")
11 # ...
```

- Escreva um programa que simula o jogo conhecido como “Pedra, Papel e Tesoura” de um jogador A contra um jogador B. O programa deve ler a escolha do jogador A e a escolha do jogador B. Por fim, o programa deve indicar quem foi o vencedor.

```
1 # ...
2 elif jogadorA == "tesoura":
3     if jogadorB == "pedra":
4         print("O jogador B ganhou")
5     elif jogadorB == "tesoura":
6         print("Empate")
7     else:
8         print("O jogador A ganhou")
9 # ...
10
11
```

- Escreva um programa que simula o jogo conhecido como “Pedra, Papel e Tesoura” de um jogador A contra um jogador B. O programa deve ler a escolha do jogador A e a escolha do jogador B. Por fim, o programa deve indicar quem foi o vencedor.

```
1 # ...
2 else: # jogadorA == "papel"
3     if jogadorB == "pedra":
4         print("O jogador A ganhou")
5     elif jogadorB == "tesoura":
6         print("O jogador B ganhou")
7     else:
8         print("Empate")
9
10
11
```

- Associar objetos a números é uma forma de abstração. No código a seguir usamos as seguintes associações:
  - `pedra = 0`
  - `papel = 1`
  - `tesoura = 2`
- O resultado da expressão `(jogadorA - jogadorB) % 3` indica, de forma única, o vencedor da partida.
- Complete o código analisando o resultado da expressão anterior.



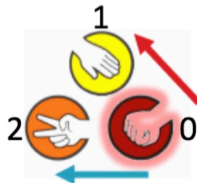
$$(2 - 1) \% 3 = 1$$

$$(2 - 0) \% 3 = 2$$



$$(1 - 0) \% 3 = 1$$

$$(1 - 2) \% 3 = 2$$



$$(0 - 2) \% 3 = 1$$

$$(0 - 1) \% 3 = 2$$

```
1 print("Pedra = 0")
2 print("Papel = 1")
3 print("Tesoura = 2")
4
5 jogadorA = int(input("Digite a primeira escolha: "))
6 jogadorB = int(input("Digite a segunda escolha: "))
7
8 resultado = (jogadorA - jogadorB) % 3
9
10 if resultado == 1:
11     print("O jogador A ganhou")
12 elif resultado == 2:
13     print("O jogador B ganhou")
14 else:
15     print("Empate")
```



- Podemos usar uma variável para armazenar um estado do programa.
- Por exemplo, podemos criar uma variável para indicar se um sistema está funcionando corretamente (ou se apresentou alguma falha).
- Normalmente inicializamos esta variável com um valor padrão (por exemplo, **True**) e atualizamos a variável caso uma mudança de estado ocorra (trocando o valor, por exemplo, para **False**).
- Este tipo de variável, que serve para sinalizar uma situação específica, é chamada de *flag*.
- Uma variável *flag* pode simplificar significativamente a escrita, manutenção e o entendimento de um programa.

```
1 ...
2
3
4 if <condição1>:
5     print("Falha do tipo 1")
6
7 if <condição2>:
8     print("Falha do tipo 2")
9
10 if <condição3>:
11     print("Falha do tipo 3")
12
13
14 ...
```

```
1 ...
2
3
4 if <condição4>:
5     print("Falha do tipo 4")
6
7 ...
8
9 if <condição100>:
10    print("Falha do tipo 100")
11
12
13 if not(<condição1>) and not(<condição2>) ... not(<condição100>):
14    print("Sistema funcionando normalmente")
```

```
1 OK = True
2
3
4 if <condição1>:
5     print("Falha do tipo 1")
6     OK = False
7 if <condição2>:
8     print("Falha do tipo 2")
9     OK = False
10 if <condição3>:
11     print("Falha do tipo 3")
12     OK = False
13
14 ...
```

```
1 ...
2
3
4 if <condição4>:
5     print("Falha do tipo 4")
6     OK = False
7 ...
8
9 if <condição100>:
10    print("Falha do tipo 100")
11    OK = False
12
13 if OK:
14    print("Sistema funcionando normalmente")
```

# Perguntas ....

# Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. <https://ic.unicamp.br/~mc102/>
  - Aula Introdutória [ [slides](#) ] [ [vídeo](#) ]
  - Primeira Aula de Laboratório [ [slides](#) ] [ [vídeo](#) ]
  - Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [ [slides](#) ] [ [vídeo](#) ]
  - Comandos Condicionais [ [slides](#) ] [ [vídeo](#) ]
  - Comandos de Repetição [ [slides](#) ] [ [vídeo](#) ]
  - Listas e Tuplas [ [slides](#) ] [ [vídeo](#) ]
  - Strings [ [slides](#) ] [ [vídeo](#) ]
  - Dicionários [ [slides](#) ] [ [vídeo](#) ]
  - Funções [ [slides](#) ] [ [vídeo](#) ]
  - Objetos Multidimensionais [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Busca [ [slides](#) ] [ [vídeo](#) ]
  - Recursão [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação Recursivos [ [slides](#) ] [ [vídeo](#) ]
  - Arquivos [ [slides](#) ] [ [vídeo](#) ]
  - Expressões Regulares [ [slides](#) ] [ [vídeo](#) ]
  - Execução de Testes no Google Cloud Shell [ [slides](#) ] [ [vídeo](#) ]
  - Numpy [ [slides](#) ] [ [vídeo](#) ]
  - Pandas [ [slides](#) ] [ [vídeo](#) ]
- Panda - Cursos de Computação em Python (IME -USP) <https://panda.ime.usp.br/>
  - Como Pensar Como um Cientista da Computação <https://panda.ime.usp.br/pensepy/static/pensepy/>
  - Aulas de Introdução à Computação em Python <https://panda.ime.usp.br/aulasPython/static/aulasPython/>
- Fabio Kon, Introdução à Ciência da Computação com Python <http://bit.ly/FabioKon/>
- Socratica, Python Programming Tutorials <http://bit.ly/SocraticaPython/>
- Google - online editor for cloud-native applications (Python programming) <https://shell.cloud.google.com/>
- w3schools - Python Tutorial <https://www.w3schools.com/python/>
- Outros, citados nos Slides.