MAC 115 — Introdução à Ciência da Computação

Aula 9

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

Álgebra booleana

Propriedades Comutativas	A and B = B and A	A or B = B or A
Propriedades Distributivas	A and (B or C) = (A and B) or (A and C)	A or (B and C) = (A or B) and (A or C)
Propriedades Associativas	(A or B) or C = A or (B or C)	(A and B) and C = A and (B and C)
Propriedades Idempotentes	A and A = A	A or A = A
Dupla Negação	not not A = A	
Elementos Absorventes	A or True = True	A and False = False
Elementos Neutros	A or False = A	A and True = A
Leis de De Morgan	not (A or B) = (not A) and (not B)	not (A and B) = (not A) or (not B)

Álgebra booleana

- pizza
- sushi
- moqueca
- hambúrguer

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

  - pizza **ou** hambúrguer

Álgebra booleana

- **pizza**
- **sushi**

  - pizza **ou** hambúrguer

- **moqueca**
- **hambúrguer**

  - **nem** sushi **nem** moqueca

Álgebra booleana

- **pizza**
- **sushi**

- pizza **ou** hambúrguer

- **moqueca**
- **hambúrguer**

- **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

Álgebra booleana

- **pizza**
- **sushi**

- **moqueca**
- **hambúrguer**

- pizza **ou** hambúrguer

- **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

- **Sou guloso**
  - pizza **ou** hambúrguer
  - **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **pizza**
- **sushi**

- **Sou guloso**
  - pizza **ou** hambúrguer

- **moqueca**
- **hambúrguer**

- **Sou alérgico a peixes**
  - **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `not sushi and not moqueca`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for sushi ou moqueca**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)
- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)
- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`
- **Topa lição de casa, jantar e depois cinema?**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)
- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`
- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - » `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles
    - » `(not jantar) or (not cinema)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles
    - `(not jantar) or (not cinema)`
    - `lição and ((not jantar) or (not cinema))`

Atalhos

- É muito comum fazer `x = x +1`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`
  - `x *= 2`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?

  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`
  - `x %= 2`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?

  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`
  - `x %= 2`
  - `x **= 2`

Exercícios

Dado um número n, diga seu equivalente em grosas, dúzias e unidades ("665 corresponde a 4 grosas, 7 dúzias e 5 unidades").

```
n = int(input("Digite um número natural: "))
grosas = n // 144
soburô = n % 144
dúzias = soburô // 12
unidades = soburô % 12
print("{} corresponde a {} grosas, {} dúzias e {} unidades"
      .format(n, grosas, dúzias, unidades))
```

Exercícios

Dado um número n, diga seu equivalente em grosas, dúzias e unidades ("665 corresponde a 4 grosas, 7 dúzias e 5 unidades").

```
n = int(input("Digite um número natural: "))
m = n
divisor = 12**2
print(n, "corresponde a", end=" ")
print(m // divisor, "grosas,", end=" ")
m %= divisor
divisor //= 12
print(m // divisor, "dúzias e", end=" ")
m %= divisor
divisor //= 12 # == 1
print(m // divisor, "unidades")
```

Exercícios

Dado um número n, diga seu equivalente em grosas, dúzias e unidades ("665 corresponde a 4 grosas, 7 dúzias e 5 unidades").

```
n = int(input("Digite um número natural: "))
m = n
divisor = 12**2
print(n, "corresponde a", end=" ")
print(m // divisor, "grosas,", end=" ")
m %= divisor
divisor //= 12
print(m // divisor, "dúzias e", end=" ")
m %= divisor
divisor //= 12 # == 1
print(m // divisor, "unidades")
```

Isso tem cara de laço!

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

Qual algoritmo vamos usar?

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**

- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**

- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
  - Absurdamente complicado e ineficiente (e envolve coisas que ainda não vimos)
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
  - Absurdamente complicado e ineficiente (e envolve coisas que ainda não vimos)
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**
  - Medianamente complexo, mas bastante eficiente

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
  - Absurdamente complicado e ineficiente (e envolve coisas que ainda não vimos)
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**
  - Medianamente complexo, mas bastante eficiente

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
  - Absurdamente complicado e ineficiente (e envolve coisas que ainda não vimos)
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**
  - Medianamente complexo, mas bastante eficiente — estamos começando!

Exercícios

- **Vai testando todos os naturais até encontrar n números que satisfaçam o que foi pedido**
  - Simples, porém ineficiente
- **Encontra os n primeiros múltiplos de i e os n primeiros múltiplos de j; junta todos, coloca em ordem e pega apenas os n primeiros resultados**
  - Absurdamente complicado e ineficiente (e envolve coisas que ainda não vimos)
- **Vai testando os múltiplos de i e j "alternadamente" (na verdade, sempre o menor múltiplo ainda não testado) até encontrar n números**
  - Medianamente complexo, mas bastante eficiente — estamos começando!

"Premature optimization is the root of all evil" (Knuth?)

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
n = int(input("Digite n: "))
i = int(input("Digite i: "))
j = int(input("Digite j: "))
…
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:

    elif x % i == 0:
        print(x, end=" ")
        encontrados += 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:

    elif x % i == 0:
        print(x, end=" ")
        encontrados += 1
    elif x % j == 0:
        print(x, end=" ")
        encontrados += 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:

    elif x % i == 0:
        print(x, end=" ")
        encontrados += 1
    elif x % j == 0:
        print(x, end=" ")
        encontrados += 1
    else:
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:

    elif x % i == 0:
        print(x, end=" ")
        encontrados += 1
    elif x % j == 0:
        print(x, end=" ")
        encontrados += 1
    else:

    x += 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:

    elif  x % i == 0 or x % j == 0 :
        print(x, end=" ")
        encontrados += 1
    else:

    x += 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif  x % i == 0 or x % j == 0 :
        print(x, end=" ")
        encontrados += 1
    else:
        pass
    x += 1
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif  x % i == 0 or x % j == 0 :
        print(x, end=" ")
        encontrados += 1
    else:
        pass
    x += 1
print()
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif  x % i == 0 or x % j == 0 :
        print(x, end=" ")
        encontrados += 1


    x += 1
print()
```

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")
elif delta == 0:

    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **A indentação deixa mais claro que, na verdade, são casos do mesmo "nível" e mutuamente excludentes (um e apenas um dos casos é executado)**

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")
elif delta == 0:

    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- A indentação deixa mais claro que, na verdade, são casos do mesmo "nível" e mutuamente excludentes (**um** e **apenas um** dos casos é executado)

Mas a ordem pode fazer diferença!

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif  x % i == 0 or x % j == 0 :
        print(x, end=" ")
        encontrados += 1


    x += 1
print()
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif (x % i == 0 or x % j == 0) and not (x % i == 0 and x % j == 0):
        print(x, end=" ")
        encontrados += 1


    x += 1
print()
```

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif (x % i == 0 or x % j == 0) and not (x % i == 0 and x % j == 0):
        print(x, end=" ")
        encontrados += 1


    x += 1
print()
```

(implícito)

Exercícios

Dados os naturais positivos n, i e j, imprimir em ordem crescente os n primeiros naturais que sejam múltiplos de i ou de j, mas não de ambos.

```
encontrados = 0
x = 1
while encontrados < n:
    if x % i == 0 and x % j == 0:
        pass
    elif (x % i == 0 or x % j == 0) and not (x % i == 0 and x % j == 0):
        print(x, end=" ")
        encontrados += 1


    x += 1
print()
```

(implícito)

A ordem faz diferença!

Lição de casa:

o que acontece se i e j são iguais?

and now for something slightly different

Partes mínimas de um laço

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

```
usuarioQuerJogar = True
while usuarioQuerJogar:
    # Joga uma partida...
    resposta = input("Você quer jogar novamente? ")
    if resposta != "S":
        usuarioQuerJogar = False
print("Cabô!")
```

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Mas podemos usar mais variáveis no laço além da variável de controle

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma

```
n = int(input("Digite um número (zero para sair): "))
soma = 0
while n != 0:
    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma)
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
n = int(input("Digite um número (zero para sair): "))
soma = 0
while n != 0:
    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma)
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
n = int(input("Digite um número (zero para sair): "))

soma = 0
while n != 0:




    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma)
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))

soma = 0
while n != 0:




    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma , end="")
if ordenado:
    print(" e eles estão em ordem", end="")
print()
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))

soma = 0
while n != 0:
    if n < anterior:
        ordenado = False

    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma , end="")
if ordenado:
    print(" e eles estão em ordem", end="")
print()
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))

soma = 0
while n != 0:
    if n < anterior:
        ordenado = False
    anterior = n
    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma , end="")
if ordenado:
    print(" e eles estão em ordem", end="")
print()
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma, informando se os números estavam em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))
anterior = n
soma = 0
while n != 0:
    if n < anterior:
        ordenado = False
    anterior = n
    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é", soma , end="")
if ordenado:
    print(" e eles estão em ordem", end="")
print()
```

Indicadores de passagem

- **Chamamos uma variável similar a `ordenado` do exemplo anterior de indicador de passagem**
- **O uso de indicadores de passagem é um padrão de programação bastante comum**
  - Padrões de programação são técnicas (ou truques!) úteis e, portanto, comuns, porém não óbvios
- **Um indicador de passagem é usado para indicar se "alguma coisa" aconteceu durante o laço**
  - Quando usamos um indicador de passagem, não estamos preocupados com **qual** elemento do laço causou o evento
  - Na verdade, pode haver um ou mais elementos responsáveis por essa mudança; o valor do indicador de passagem é alterado no máximo uma vez

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço
  - O indicador de passagem muda de valor **zero ou uma vezes** e a mudança pode acontecer em qualquer uma das iterações

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço
  - O indicador de passagem muda de valor **zero ou uma vezes** e a mudança pode acontecer em qualquer uma das iterações
    - » *(o comando que altera o valor do indicador de passagem pode ser executado mais de uma vez, mas o valor da variável só é efetivamente alterado na primeira)*

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço
  - O indicador de passagem muda de valor **zero ou uma vezes** e a mudança pode acontecer em qualquer uma das iterações
    - » *(o comando que altera o valor do indicador de passagem pode ser executado mais de uma vez, mas o valor da variável só é efetivamente alterado na primeira)*
- **MAS...**

Indicadores de passagem

- **O indicador de passagem é diferente da variável de controle do laço**
  - A variável de controle **sempre** muda de valor **no mínimo** uma vez (na última repetição) para indicar o fim do laço
  - O indicador de passagem muda de valor **zero ou uma vezes** e a mudança pode acontecer em qualquer uma das iterações
    - » *(o comando que altera o valor do indicador de passagem pode ser executado mais de uma vez, mas o valor da variável só é efetivamente alterado na primeira)*
- **MAS...**
- **Às vezes podemos usar o indicador de passagem para terminar o laço antecipadamente**

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e informe se os números estão em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))
anterior = n
while n != 0:
    if n < anterior:
        ordenado = False
    anterior = n
    n = int(input("Digite um número (zero para sair): "))
if ordenado:
    print("Os números estão em ordem")
else:
    print("Os números não estão em ordem")
```

Indicadores de passagem

Leia uma série de números terminada por zero fornecida pelo usuário e informe se os números estão em ordem crescente

```
ordenado = True
n = int(input("Digite um número (zero para sair): "))
anterior = n
while n != 0 and ordenado:
    if n < anterior:
        ordenado = False
    anterior = n
    n = int(input("Digite um número (zero para sair): "))
if ordenado:
    print("Os números estão em ordem")
else:
    print("Os números não estão em ordem")
```

Exercício

Dado um número $n > 0$, informe se esse número possui ao menos dois dígitos adjacentes que sejam iguais

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

Leia uma série de números terminada por zero fornecida pelo usuário e calcule sua soma

```
n = int(input("Digite um número (zero para sair): "))
soma = 0
while n != 0:
    soma = soma + n
    n = int(input("Digite um número (zero para sair): "))
print("A soma dos números é " + str(soma))
```

Tipos de repetição

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Tipos de repetição

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, 6437 → 7346

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Professores mentem!

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
  2. Repetições sobre os elementos de um conjunto

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
  2. Repetições sobre os elementos de um conjunto
- **Na verdade...**

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
  2. Repetições sobre os elementos de um conjunto
- **Na verdade...**
  - Tudo poderia ser feito com `while`

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
  2. Repetições sobre os elementos de um conjunto
- **Na verdade...**
  - Tudo poderia ser feito com `while`
  - Há casos “intermediários”

Mas, em muitos casos, cada estilo de repetição ajuda a deixar mais clara a intenção do laço

Repetições encaixadas

- **Repetições usam uma variável de controle**

Repetições encaixadas

- Repetições usam uma variável de controle
- Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto

Repetições encaixadas

- **Repetições usam uma variável de controle**
- **Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto**
  - De maneira unidimensional!

Repetições encaixadas

- **Repetições usam uma variável de controle**
- **Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto**
  - De maneira unidimensional!
  - Números naturais, nomes de pessoas, notas de alunos...

Repetições encaixadas

- **Repetições usam uma variável de controle**
- **Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto**
  - De maneira unidimensional!
  - Números naturais, nomes de pessoas, notas de alunos...
- **Como trabalhar com conjuntos de maneira bidimensional?**

Repetições encaixadas

- **Repetições usam uma variável de controle**
- **Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto**
  - De maneira unidimensional!
  - Números naturais, nomes de pessoas, notas de alunos...
- **Como trabalhar com conjuntos de maneira bidimensional?**
  - Pontos de um plano, tabelas...

Repetições encaixadas

- **Repetições usam uma variável de controle**
- **Quando essa variável corresponde a um conjunto, as repetições permitem trabalhar sobre os elementos desse conjunto**
  - De maneira unidimensional!
  - Números naturais, nomes de pessoas, notas de alunos...
- **Como trabalhar com conjuntos de maneira bidimensional?**
  - Pontos de um plano, tabelas...
- **Repetições encaixadas**

Repetições encaixadas

```
while condição1:
    while condição2:
        …
```

Repetições encaixadas

```
while condição1:
    while condição2:
        …
```

- A cada “rodada” do laço mais externo, o laço interno é executado várias vezes (até sua condição deixar de ser verdadeira)

Repetições encaixadas

```
while condição1:
    while condição2:
        …
```

- A cada "rodada" do laço mais externo, o laço interno é executado várias vezes (até sua condição deixar de ser verdadeira)
- Para isso funcionar, `condição2` deve voltar a ser verdadeira a cada nova rodada do laço mais externo

Exemplo — tabuada

Exemplo — tabuada

```
tabuada_do = 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:




    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:

    while n <= 10:


    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:


    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n)

    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n)
        n += 1
    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n , end=" ")
        n += 1
    tabuada_do += 1
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n , end=" ")
        n += 1
    tabuada_do += 1
    print()
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n, end="\t")
        n += 1
    tabuada_do += 1
    print()
```

Exemplo — tabuada

```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print("{:3}".format(tabuada_do * n), end=" ")
        n += 1
    tabuada_do += 1
    print()
```

Exercício

Leia uma sequência de números naturais positivos terminada por zero e, para cada número, imprima seu fatorial