MAC 115 — Introdução à Ciência da Computação

Aula 12

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

Programando

1. **algoritmo vs implementação**
2. **entrada de dados → processamento → resultado**
   - Mostra o resultado para o usuário
   - **Utiliza o resultado como dado para fazer outra coisa**
3. **Existem *tipos de dados* diferentes em python (*int*, *float*, *string*, *bool*...)**
4. **Expressões são coisas que têm um *valor* (de algum *tipo*)**
   - E podem ser combinadas ou utilizadas como partes de outras expressões

`2 + 3 + 7` (int)

`2 > 3 and 5 > 4` (bool)

Nomes (variáveis)

- *Nomes* **permitem que pensemos mais sobre o problema a ser resolvido e menos sobre as idiossincrasias do computador**
- **Nomes em geral representam valores que** *variam* **(basicamente, alguma informação "real" que está em algum lugar na memória do computador)**
  - Como na matemática!
- **Por isso, chamamos esses nomes de "variáveis"**

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em python com “=”:

```
x = 5
x = x + 1
x = input("Digite seu nome: ")
```

Condicionais e 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)**

Condicionais e 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)**

Os casos são mutuamente excludentes **quando há `else`**!

Condicionais e 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)**

A ordem pode fazer diferença!

Condicionais e condições mutuamente excludentes

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()
```

Condicionais e condições mutuamente excludentes

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()
```

Condicionais e condições mutuamente excludentes

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)

Condicionais e condições mutuamente excludentes

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!

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!")
```

Indicadores de passagem

- Um **indicador de passagem** é uma variável usada para indicar se "alguma coisa" aconteceu durante o laço

Indicadores de passagem

- **Um indicador de passagem é uma variável usada 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 só é alterado uma vez

Indicadores de passagem

- **Um indicador de passagem é uma variável usada 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 só é alterado uma vez
- **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

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")
```

Truques com *strings* — escapes

- **O caracter "\" indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string***
  - Espaços em branco "especiais" (\n, \t ...)
  - Caracteres "problemáticos" (\", \', \\ ...)

```
print("Batatinha quando nasce\nEspalha a rama pelo chão")
print("Camões bebeu muitos copos d'água escrevendo \"Os Lusíadas\"")
print('Camões bebeu muitos copos d\'água escrevendo "Os Lusíadas"')
print('O caracter \'\\\' resolve "todos" os problemas do mundo')
print("O caracter '\\' resolve \"todos\" os problemas do mundo")
```

Truques com `print()` — `end` e `sep`

- **`print()` não termina mudando para a próxima linha, mas sim com o que é definido por `end`**
  - (se você não definir `end`, python utiliza `\n`)

Truques com `print()` — end e sep

- **`print()` não termina mudando para a próxima linha, mas sim com o que é definido por `end`**
  - (se você não definir end, python utiliza \n)
- **`print()` não separa os itens com um espaço, mas sim com o que é definido por `sep`**
  - (se você não definir sep, python utiliza um espaço)

Truques com `print()` — end e sep

```
print("super", "cali", "fragilistic",
      "expiali", "docious", sep="")
print("Batatinha quando nasce", end="\n.\n.\n.\n")
print("Espalha a rama pelo chão")
```

---

supercalifragilisticexpialidocious
Batatinha quando nasce
.
.
.
Espalha a rama pelo chão

Truques com `print()` — end e sep

```
print("Maria", 45678, "tem noção", sep="\t")
print("João", 123, "não tem noção", sep="\t")
print("Ana", 9, "tem noção (mas não muita)", sep="\t")
```

```
Maria   45678   tem noção
João    123     não tem noção
Ana     9       tem noção (mas não muita)
```

Truques com *strings* — formatação

```
import math
print("Você digitou {0} números ({1} pares e {2} ímpares)".format(12, 7, 5))
print("Você digitou {} números ({} pares e {} ímpares)".format(12, 7, 5))
print("Pi pode ser aproximado para {pi:.7f} ou {pi:.4f}".format(pi=math.pi))
print("x vale {:.7e}".format(1234.5678))
print("|{:9.3f}| -- |{:9.3f}|".format(13.22784, 1200.20004))
print("|{:9.3f}| -- |{:9.3f}|".format(13227.84, 37.6))
print("|{:9.3f}| -- |{:9.3f}|".format(0.0, 127))
```

- **Cada item (“{}”) tem duas partes opcionais separadas por “:”**
  - O identificador (0, 1 etc. ou um nome)
  - A definição de como o item deve ser apresentado (precisão, largura mínima, alinhamento etc.)

https://docs.python.org/3/library/string.html#formatspec

strings vs `print()`

- escapes são recursos das *strings*
- `sep` e `end` são recursos de `print()`
- comandos de formatação (`.format()`) são recursos das *strings*

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**

- **moqueca**
- **hambúrguer**

- pizza **ou** 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**
- **moqueca**
- **hambúrguer**

- **Sou guloso**
  - pizza **ou** 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))`

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

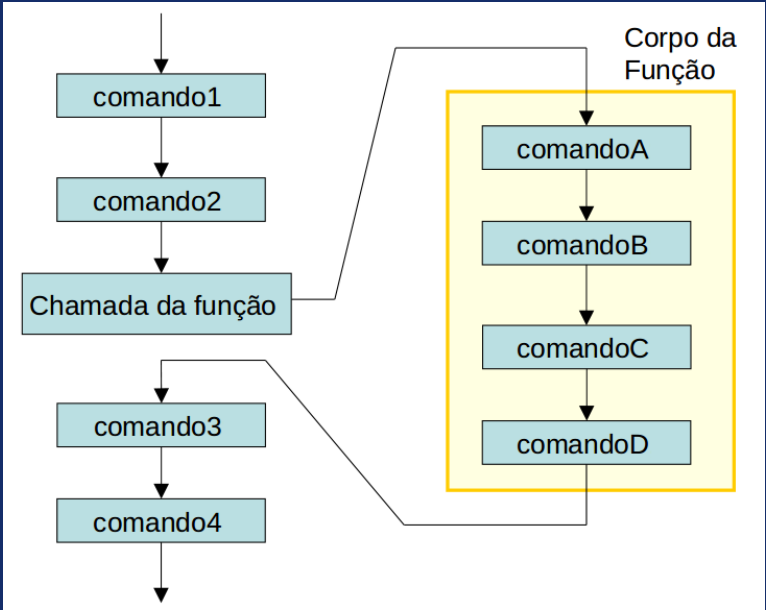
```
tabuada_do = 1
while tabuada_do <= 10:
    n = 1
    while n <= 10:
        print(tabuada_do * n, end="\t")
        n += 1
    tabuada_do += 1
    print()
```

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()
```

Funções

- **Vantagens das funções:**
  - Ajudam a dividir um programa em partes mais fáceis de compreender
    - » *O que permite a criação de programas mais complexos*
  - Facilitam o trabalho em equipe
  - Permitem que cada parte possa ser testada independentemente
  - Evitam que a mesma coisa seja criada várias vezes
    - » *E promovem a integração do código de vários programadores*


Corpo da
Função
comando1
comando2
Chamada da função
comando3
comando4
comandoA
comandoB
comandoC
comandoD

Funções

- **Funções são inspiradas nas funções matemáticas**
- **Em geral, recebem parâmetros e devolvem valores (resultados) que dependem desses parâmetros**
  - Mas nem sempre! `print()`, por exemplo, não devolve nenhum resultado
    - » *"Efeitos colaterais"*

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**
  - `print()` diz *o que fazer* com o valor de uma expressão

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**
  - `print()` diz *o que fazer* com o valor de uma expressão
  - O operador `=` *também* diz o que fazer com o valor de uma expressão

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**
  - `print()` diz *o que fazer* com o valor de uma expressão
  - O operador `=` *também* diz o que fazer com o valor de uma expressão
- **`int()` e `float()` são quase expressões (são *funções*)**

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**
  - `print()` diz *o que fazer* com o valor de uma expressão
  - O operador `=` *também* diz o que fazer com o valor de uma expressão
- **`int()` e `float()` são quase expressões (são *funções*)**
  - E, portanto, podem fazer parte de uma expressão

Nem tudo são expressões

Isto faz sentido?

```
x = print(2 + 3)
```

- **Nem tudo são expressões!**
- **`2 + 3` é uma expressão**
- **`print()` *não* é uma expressão**
  - `print()` diz *o que fazer* com o valor de uma expressão
  - O operador `=` *também* diz o que fazer com o valor de uma expressão
- **`int()` e `float()` são quase expressões (são *funções*)**
  - E, portanto, podem fazer parte de uma expressão

```
x = 2 + int(3.7)
print(2 + int(3.7))
```

Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):
    return (a + b) / 2

print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):
    print((a + b) / 2)


media(5, 7)
```

Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):
    return (a + b) / 2

print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):
    print((a + b) / 2)


media(5, 7)
```

A segunda versão nem precisa ter **return**

Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):
    return (a + b) / 2

print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):
    print((a + b) / 2)


media(5, 7)
```

A segunda versão nem precisa ter **`return`** (mas pode ter)

Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):
    return (a + b) / 2

print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):
    print((a + b) / 2)
    return (a + b) / 2

media(5, 7)
```

A segunda versão nem precisa ter **return** (mas pode ter)

Funções – return

Não precisa ter só um return:

```
def maximo(a, b):
    if a > b:
        return a
    else:
        return b

print(maximo(2, 7))
```

Funções

Dado um número inteiro $n \geq 2$, diga se ele é primo

```
def éPrimo(x):
    divisor = x -1
    primo = True
    while divisor >= 2:
        if x % divisor == 0:
            primo = False
        divisor -= 1
    return primo
```

Funções

Dado um número inteiro $n \geq 2$, diga se ele é primo

```
def éPrimo(x):
    divisor = x -1
    primo = True
    while divisor >= 2:
        if x % divisor == 0:
            primo = False
        divisor -= 1
    return primo
```

- **Não confunda `print()` e `return`!**

Funções

Dado um número inteiro $n \geq 2$, diga se ele é primo

```
def éPrimo(x):
    divisor = x -1
    primo = True
    while divisor >= 2:
        if x % divisor == 0:
            primo = False
        divisor -= 1
    return primo
```

- **Não confunda `print()` e `return`!**

Funções – escopo

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
fatorial(4)
print(fat)
```

Funções – escopo

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial(4))
```

Funções – escopo

```
n = 4
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial())
```

Funções – escopo

```
n = 5
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
print(fatorial(4))
print(n)
```

Funções – escopo

```
def fatorial_enviesado(n):
    fat = 1
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1

    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    fat = 1
    print("Viés anterior:", viés)
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – escopo

```
viés = 3
def fatorial_enviesado(n):
    global viés
    fat = 1
    print("Viés anterior:", viés)
    viés = 5
    while n >= 2:
        fat *= n
        n -= 1
    return fat + viés
print(fatorial_enviesado(4))
print(viés)
```

Funções – `main()`

```
print(fatorial(4))
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat
```

Funções – `main()`

```
def main():
    x = int(input("Digite um inteiro positivo: "))
    print(fatorial(x))

def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat

main()
```

Funções – `main()`

```
def fatorial(n):
    fat = 1
    while n >= 2:
        fat *= n
        n -= 1
    return fat

def main():
    x = int(input("Digite um inteiro positivo: "))
    print(fatorial(x))

main()
```

Embora não seja obrigatório, em geral é uma boa ideia usar uma função `main()`

Exercício — sistema de *login*

Imagine um sistema de *login* com três usuários:

- **Alan Turing** — UID **`turing`**, senha **`tmachine`**
- **Ada Lovelace** — UID **`llace`**, senha **`anengine`**
- **Grace Hopper** — UID **`hopper`**, senha **`business`**

Crie um sistema que lê o UID (*login*) e a senha do usuário e, se os dados estiverem corretos, escreve “Bem-vindo, [nome]!”; caso contrário, o sistema escreve “Login ou senha incorreto”.

Exercício — sistema de *login*

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")









if user_ok:
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")
user_ok = False









if user_ok:
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")
user_ok = False
if uid == "turing" and senha == "tmachine":
    user_ok = True
    nome = "Alan Turing"









if user_ok:
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")
user_ok = False
if uid == "turing" and senha == "tmachine":
    user_ok = True
    nome = "Alan Turing"
elif uid == "llace" and senha == "anengine":
    user_ok = True
    nome = "Ada Lovelace"
elif uid == "hopper" and senha == "business":
    user_ok = True
    nome = "Grace Hopper"
if user_ok:
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")
nome = ""
if uid == "turing" and senha == "tmachine":

    nome = "Alan Turing"
elif uid == "llace" and senha == "anengine":

    nome = "Ada Lovelace"
elif uid == "hopper" and senha == "business":

    nome = "Grace Hopper"
if nome != "":
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
uid = input("username: ")
senha = input("senha: ")

nome = ""
if uid == "turing" and senha == "tmachine":
    nome = "Alan Turing"
elif uid == "llace" and senha == "anengine":
    nome = "Ada Lovelace"
elif uid == "hopper" and senha == "business":
    nome = "Grace Hopper"

if nome != "":
  print("Bem-vindo, {}!".format(nome))
else:
  print("Login ou senha incorreto")
```

Exercício — sistema de *login*

```
def main():
    uid = input("username: ")
    senha = input("senha: ")
    nome = checa_login(uid, senha)
    if nome == "":
        print("Login ou senha incorreto")
    else:
        print("Bem-vindo, {}!".format(nome))
```

Exercício — sistema de *login*

```
def main():
    uid = input("username: ")
    senha = input("senha: ")
    nome = checa_login(uid, senha)
    if nome == "":
        print("Login ou senha incorreto")
    else:
        print("Bem-vindo, {}!".format(nome))

def checa_login(uid, senha):
    nome = ""
    if uid == "turing" and senha == "tmachine":
        nome = "Alan Turing"
    elif uid == "llace" and senha == "anengine":
        nome = "Ada Lovelace"
    elif uid == "hopper" and senha == "business":
        nome = "Grace Hopper"
    return nome
```

Exercício — sistema de *login*

```
def main():
    uid = input("username: ")
    senha = input("senha: ")
    nome = checa_login(uid, senha)
    if nome == "":
        print("Login ou senha incorreto")
    else:
        print("Bem-vindo, {}!".format(nome))

def checa_login(login, pwd):
    name = ""
    if login == "turing" and pwd == "tmachine":
        name = "Alan Turing"
    elif login == "llace" and pwd == "anengine":
        name = "Ada Lovelace"
    elif login == "hopper" and pwd == "business":
        name = "Grace Hopper"
    return name
```