

# MAC 110 – Introdução à Ciência da Computação

## Aula 10

---

Nelson Lago

BMAC – 2024



**Previously on MAC110...**

# 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!**

# Exercícios

# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter “#”. Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```



# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))
```

# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))  
linha = 0  
while linha < m:
```

```
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))  
linha = 0  
while linha < m:  
    coluna = 0  
    while coluna < n:  
        coluna += 1  
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))  
linha = 0  
while linha < m:  
    coluna = 0  
    while coluna < n:  
        print("#", end="")  
        coluna += 1  
  
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))  
linha = 0  
while linha < m:  
    coluna = 0  
    while coluna < n:  
        print("#", end="")  
        coluna += 1  
    print()  
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter “#”. Por exemplo, para 4 e 5:

```
####  
#  #  
#  #  
#  #  
####
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:

        print("#", end="")

        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if linha == 0:
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if linha == 0 or linha == m :
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if linha == 0 or linha == m - 1 :
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if linha == 0 or linha == m - 1 or coluna == 0 :
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dados os números  $n$  e  $m$ , imprima o contorno de um retângulo  $n \times m$  com o caracter "#". Por exemplo, para 4 e 5:

```
n = int(input("Digite o número de colunas: "))
m = int(input("Digite o número de linhas: "))
linha = 0
while linha < m:
    coluna = 0
    while coluna < n:
        if linha == 0 or linha == m - 1 or coluna == 0 or coluna == n - 1:
            print("#", end="")
        else:
            print(" ", end="")
        coluna += 1
    print()
    linha += 1
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
divisor = 2
while n > 1:
    while n % divisor > 0:
        divisor += 1
    print(divisor, end = " ")
    n /= divisor
print()
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
divisor = 2
while n > 1:

    while n % divisor > 0:
        divisor += 1
    print(divisor, end = " ")

    n /= divisor
print()
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))

while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1
    print(divisor, end = " ")

    n /= divisor
print()
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))

while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1

    n /= divisor
    print()
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))

while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1

    print("fator {}, multiplicidade {}".format(divisor, multiplicidade))

    n /= divisor
print()
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))

while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1

    print("fator {}, multiplicidade {}".format(divisor, multiplicidade))

    n /= divisor
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
multiplicidade = 1
while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1

    print("fator {}, multiplicidade {}".format(divisor, multiplicidade))

    n /= divisor
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
multiplicidade = 1
while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1
    if n % divisor**2 > 0:
        print("fator {}, multiplicidade {}".format(divisor, multiplicidade))

    n /= divisor
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
multiplicidade = 1
while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1
    if n % divisor**2 > 0:
        print("fator {}, multiplicidade {}".format(divisor, multiplicidade))
        multiplicidade = 1

    n /= divisor
```

# Exercício

Dado um número inteiro  $n \geq 2$ , informe sua decomposição em fatores primos, incluindo a multiplicidade de cada fator. Por exemplo, para o número 600 ( $2 * 2 * 2 * 3 * 5 * 5$ ), a saída deve ser:

fator 2, multiplicidade 3  
fator 3, multiplicidade 1  
fator 5, multiplicidade 2

```
n = int(input("Digite um inteiro positivo: "))
multiplicidade = 1
while n > 1:
    divisor = 2
    while n % divisor > 0:
        divisor += 1
    if n % divisor**2 > 0:
        print("fator {}, multiplicidade {}".format(divisor, multiplicidade))
        multiplicidade = 1
    else:
        multiplicidade += 1
    n /= divisor
```

**and now for something completely different**

# Depuração

```
cartão = int(input("Digite o número do cartão: "))
dv = cartão % 10
falta = cartão // 10
par, soma = True, 0
while falta > 0:
    atual = falta % 10
    if par:
        val = 2 * atual
        val = val // 10 + val % 10
    else:
        val = atual
    soma += val
    falta //= 10
    par = not par
calculado = 10 - (soma % 10)
if dv == calculado:
    mastercard = ""
    if cartão // 10**14 >= 51 and cartão // 10**14 <= 55:
        mastercard = "(Mastercard)"
    print("cartão válido", mastercard)
else:
    print("cartão inválido:", end=" ")
    print("dígito verificador deveria ser", calculado)
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$



Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
```

```
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n

print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:

    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True

    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:

        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:
        if candidato % divisor == 0:
            primo = False
        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

- **Esse programa pode ser dividido em duas partes**
  - ① Processar uma lista de números e contar quantos deles são primos
  - ② Verificar se um número é primo

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:
        if candidato % divisor == 0:
            primo = False
        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:
        if candidato % divisor == 0:
            primo = False
        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

- Ao programar, preferimos pensar no problema a ser resolvido e não nas idiossincrasias do computador
- Linguagens de programação de alto nível procuram oferecer os recursos para isso
- Uma das coisas mais importantes para esse fim é utilizar *nomes*

- **Esse programa pode ser dividido em duas partes**
  - ▶ Processar uma lista de números e contar quantos deles são primos
  - ▶ Verificar se um número é primo

- **Esse programa pode ser dividido em duas partes**
  - Processar uma lista de números e contar quantos deles são primos
  - Verificar se um número é primo
- **Se há partes diferentes, podemos dar *nomes* para algumas dessas partes**

- **Funções são trechos de código com um nome**
  - ▶ `int()`, `float()`, `math.sqrt()`, `print()`...
- **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*

Exemplo:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

Exemplo:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

---

6.0

Exemplo:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

6.0

- Poderíamos usar apenas `print((5+7)/2)`, mas o nome torna a intenção do código mais clara

Exemplo:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

6.0

- Poderíamos usar apenas `print((5+7)/2)`, mas o nome torna a intenção do código mais clara
- Funções ajudam a dividir um programa em partes mais fáceis de compreender (além de simplificar o trabalho em equipe, entre outras vantagens)

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:
        if candidato % divisor == 0:
            primo = False
        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    primo = True
    divisor = candidato - 1
    while divisor >= 2:
        if candidato % divisor == 0:
            primo = False
        divisor -= 1
    if primo:
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:

    if éPrimo(candidato):
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

Dado um número inteiro  $n \geq 2$ , diga quantos primos existem entre 2 e  $n$

```
n = int(input("Digite um inteiro maior ou igual a 2: "))
encontrados = 0
candidato = n
while candidato >= 2:
    if éPrimo(candidato):
        encontrados += 1
    candidato -= 1
print("Há {} primos entre 2 e {}".format(encontrados, n))
```

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
```

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!**

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`!

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!**
  - ▶ **print()** e **input()** → comunicação com o “mundo”

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!**

- ▶ **print()** e **input()** → comunicação com o “mundo”
- ▶ **return** e os parâmetros das funções → comunicação entre partes diferentes do programa

- **No fundo, ainda temos repetições encaixadas**

- **No fundo, ainda temos repetições encaixadas**
- **Então, se podemos criar funções, por que falamos em repetições encaixadas antes?**

- **No fundo, ainda temos repetições encaixadas**
- **Então, se podemos criar funções, por que falamos em repetições encaixadas antes?**
  - ▶ Dependendo do contexto, pode ser mais claro usar as repetições encaixadas diretamente (por exemplo, ao processar uma tabela)

# Exercício

```
####  
####  
####  
####  
####
```

```
n = int(input("Digite o número de colunas: "))  
m = int(input("Digite o número de linhas: "))  
linha = 0  
while linha < m:  
    coluna = 0  
    while coluna < n:  
        print("#", end="")  
        coluna += 1  
    print()  
    linha += 1
```

- **Se é muito difícil dar um nome para uma função...**

- **Se é muito difícil dar um nome para uma função...**
  - ▶ provavelmente é porque o trecho de programa em que você está mexendo não corresponde bem à divisão das ideias da maneira que você imaginou

- **Se é muito difícil dar um nome para uma função...**
  - ▶ provavelmente é porque o trecho de programa em que você está mexendo não corresponde bem à divisão das ideias da maneira que você imaginou
    - » *Nesse caso, vale tentar encontrar outra maneira de pensar o problema*

- **Se é muito difícil dar um nome para uma função...**
  - ▶ provavelmente é porque o trecho de programa em que você está mexendo não corresponde bem à divisão das ideias da maneira que você imaginou
    - » *Nesse caso, vale tentar encontrar outra maneira de pensar o problema*
  - ▶ mas às vezes encontrar um bom nome é difícil mesmo!

# Nem tudo são expressões

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

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

# Nem tudo são expressões

x = `print`(2 + 3) → faz sentido?

- **Nem tudo são expressões!**

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

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

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

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

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- `2 + 3` é uma expressão
- `int()` *não* é uma expressão
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- **2 + 3 é uma expressão**
- **int() não é uma expressão**
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*
- **print(2 + 3) não é uma expressão**

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- **2 + 3 é uma expressão**
- **int() não é uma expressão**
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*
- **print(2 + 3) não é uma expressão**
  - ▶ Uma *chamada de função* é uma expressão apenas se a função devolve algum valor

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- **2 + 3 é uma expressão**
- **int() não é uma expressão**
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*
- **print(2 + 3) não é uma expressão**
  - ▶ Uma *chamada de função* é uma expressão apenas se a função devolve algum valor
- **int() e float() são funções que devolvem um valor**

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- **2 + 3 é uma expressão**
- **int() não é uma expressão**
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*
- **print(2 + 3) não é uma expressão**
  - ▶ Uma *chamada de função* é uma expressão apenas se a função devolve algum valor
- **int() e float() são funções que devolvem um valor**
  - ▶ E, portanto, podem fazer parte de uma expressão ao serem *chamadas*

# Nem tudo são expressões

`x = print(2 + 3)` → faz sentido?

- **Nem tudo são expressões!**
- **2 + 3 é uma expressão**
- **int() não é uma expressão**
  - ▶ Funções dizem *o que fazer* com o valor de uma expressão, mas não têm valor a menos que sejam *chamadas*
- **print(2 + 3) não é uma expressão**
  - ▶ Uma *chamada de função* é uma expressão apenas se a função devolve algum valor
- **int() e float() são funções que devolvem um valor**
  - ▶ E, portanto, podem fazer parte de uma expressão ao serem *chamadas*

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

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

## Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )



# Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )

```
a = int(input("Digite um inteiro positivo: "))
b = int(input("Digite outro inteiro positivo: "))

print("O máximo divisor comum é {}".format(mdc))
```

# Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )

```
a = int(input("Digite um inteiro positivo: "))
b = int(input("Digite outro inteiro positivo: "))
mdc = a

print("O máximo divisor comum é {}".format(mdc))
```

# Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )

```
a = int(input("Digite um inteiro positivo: "))
b = int(input("Digite outro inteiro positivo: "))
mdc = a
while

print("O máximo divisor comum é {}".format(mdc))
```

# Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )

```
a = int(input("Digite um inteiro positivo: "))
b = int(input("Digite outro inteiro positivo: "))
mdc = a
while
    mdc -= 1
print("O máximo divisor comum é {}".format(mdc))
```

# Exercício

Dados dois números inteiros positivos  $a$  e  $b$ , encontre seu máximo divisor comum ( $mdc$ )

```
a = int(input("Digite um inteiro positivo: "))
b = int(input("Digite outro inteiro positivo: "))
mdc = a
while a % mdc != 0 or b % mdc != 0:
    mdc -= 1
print("O máximo divisor comum é {}".format(mdc))
```

## Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

# Exercício

Dica:

Dado  $m = \text{mdc}(a, b)$ , podemos afirmar que:

$$a = m * f_1$$

$$b = m * f_2$$

$$\text{mdc}(f_1, f_2) = 1$$

Caso contrário, poderíamos fazer

$$a = m * x * \frac{f_1}{x}$$

$$b = m * x * \frac{f_2}{x}$$

E, portanto, o máximo divisor comum seria  $m * x$

Isso significa que qualquer divisor de  $a$  e  $b$  precisa ser também um divisor de  $m$ , pois ele não pode ser divisor de  $f_1$  e  $f_2$ , como visto acima. Portanto,  $\text{mdc}(a, b, c) = \text{mdc}(\text{mdc}(a, b), c)$

## Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)



## Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
```

```
if mdc > 0:  
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a

if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0:

    if mdc > 0:
        print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0:
    a = int(input("Digite um inteiro positivo (zero para sair): "))

if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0:
    a = int(input("Digite um inteiro positivo (zero para sair): "))
    divisor = mdc

if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0:
    a = int(input("Digite um inteiro positivo (zero para sair): "))
    divisor = mdc
    while mdc % divisor != 0 or n % divisor != 0:
        divisor -= 1
    mdc = divisor
if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0 and mdc > 1:
    a = int(input("Digite um inteiro positivo (zero para sair): "))
    divisor = mdc
    while mdc % divisor != 0 or a % divisor != 0:
        divisor -= 1
    mdc = divisor
if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

## Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)



# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0 and mdc > 1:
    a = int(input("Digite um inteiro positivo (zero para sair): "))

if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
a = int(input("Digite um inteiro positivo (zero para sair): "))
mdc = a
while a > 0 and mdc > 1:
    a = int(input("Digite um inteiro positivo (zero para sair): "))
    mdc = calcula_mdc(mdc, a)
if mdc > 0:
    print("O máximo divisor comum da sequência é {}".format(mdc))
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
def calcula_mdc(a, b):
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

```
def calcula_mdc(a, b):  
    mdc = a
```

# Exercício

Dada uma sequência de números inteiros positivos, encontre seu máximo divisor comum (*mdc*)

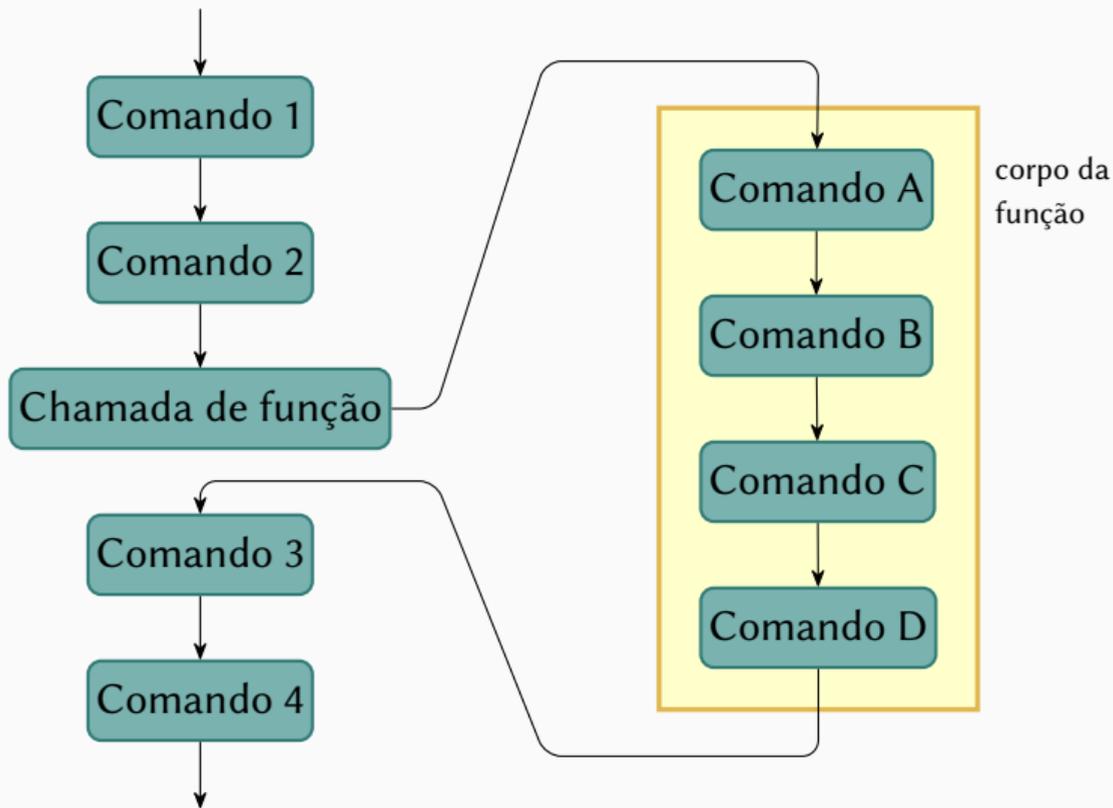
```
def calcula_mdc(a, b):  
    mdc = a  
    while a % mdc != 0 or b % mdc != 0:  
        mdc -= 1
```

# Exercício

Dada uma sequência de números inteiros positivos,  
encontre seu máximo divisor comum (*mdc*)

```
def calcula_mdc(a, b):  
    mdc = a  
    while a % mdc != 0 or b % mdc != 0:  
        mdc -= 1  
    return mdc
```

# Funções são “desvios”



- **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*

- 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”

# Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    print(m)  
  
media(5, 7)
```

# Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    print(m)  
  
media(5, 7)
```

A segunda versão nem precisa ter **return**

# Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    print(m)  
  
media(5, 7)
```

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

# Funções – Efeitos colaterais

Sem efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    return m  
  
print(media(5, 7))
```

Com efeito colateral:

```
def media(a, b):  
    m = (a + b) / 2  
    print(m)  
    return m  
  
media(5, 7)
```

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

Não precisa ter só um return:

```
def maximo(a, b):  
    if a > b:  
        return a  
    else:  
        return b  
  
print(maximo(2, 7))
```

Não precisa ter só um return (mas só um é executado de cada vez)

```
def maximo(a, b):  
    if a > b:  
        return a  
    else:  
        return b  
  
print(maximo(2, 7))
```

# 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()`**