MAC 115 — Introdução à Ciência da Computação

Aula 10

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

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)**

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

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!

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!

O que acontece se i e j são iguais?

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")
```

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()
```

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 "#"

Exercício

Dados os números n e m, imprima um retângulo $n \times m$ com o caracter "#"

```
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 "#"

```
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 "#"

```
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 "#"

```
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 "#"

```
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

Mostrar uma pirâmide semelhante a esta, sendo que o maior valor da pirâmide é definido pelo usuário. Ex: $n = 9$

```
9 8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
7 6 5 4 3 2 1
6 5 4 3 2 1
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

Intermezzo

VPL no eDisciplinas

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

and now for something completely different

Funções

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

Funções

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: "))
```

Funções

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))
```

Funções

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))
```

Funções

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))
```

Funções

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))
```

Funções

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))
```

Funções

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))
```

Funções

- **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

Funções

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))
```

Funções

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))
```

Nomes

- 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*

Funções

- **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

Funções

- **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

- **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*

Funções

Exemplo:

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

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

Funções

Exemplo:

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

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

6.0

Funções

Exemplo:

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

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

```
6.0
```

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

Funções

Exemplo:

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

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)**

Funções

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))
```

Funções

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))
```

Funções

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))
```

Funções

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))
```

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

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

Funções

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

Funções

- **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

Dados os números n e m, imprima um retângulo $n \times m$ com o caracter “#”

```
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
```

Funções

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

Funções

- **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

Funções

- **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*

Funções

- **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!

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

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




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

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

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




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

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

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

while


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

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

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

while
    mdc -= 1

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

Exercício

Dados dois números inteiros positivos n e m, encontre seu máximo divisor comum (mdc)

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

while n % mdc != 0 or m % mdc != 0:
    mdc -= 1

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

Exercício

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

Exercício

Dica:

Dado o mdc de um conjunto de números (ex: a e b), sabemos que:

$a = mdc * f_1$
$b = mdc * f_2$
$mdc(f_1, f_2) = 1$

Caso contrário, poderíamos fazer

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

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

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

Isso significa que qualquer divisor de a e b precisa ser também um divisor de mdc e que $mdc(a, b, c) = mdc(mdc(a, b), c)$

Exercício

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

Exercício

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

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









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

Exercício

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

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









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

Exercício

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

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

while n > 0:









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

Exercício

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

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

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




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

Exercício

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

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

while n > 0:
    n = 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

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

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

while n > 0:
    n = 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

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

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

while n > 0 and mdc > 1:
    n = 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

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

Exercício

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

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

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


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

Exercício

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

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

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

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

Exercício

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

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

Exercício

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

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

Exercício

Dados 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

Dados 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
```