

Introdução à linguagem Python

Thiago Martins

Programação em Python

Um programa é uma *sequência de instruções*

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Programação em Python

Execução:

Inserir o conteúdo em um arquivo .py:

```
thiago@betelgeuse:~$ cat > programa.py
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
thiago@betelgeuse:~$
```

Programação em Python

Execução:

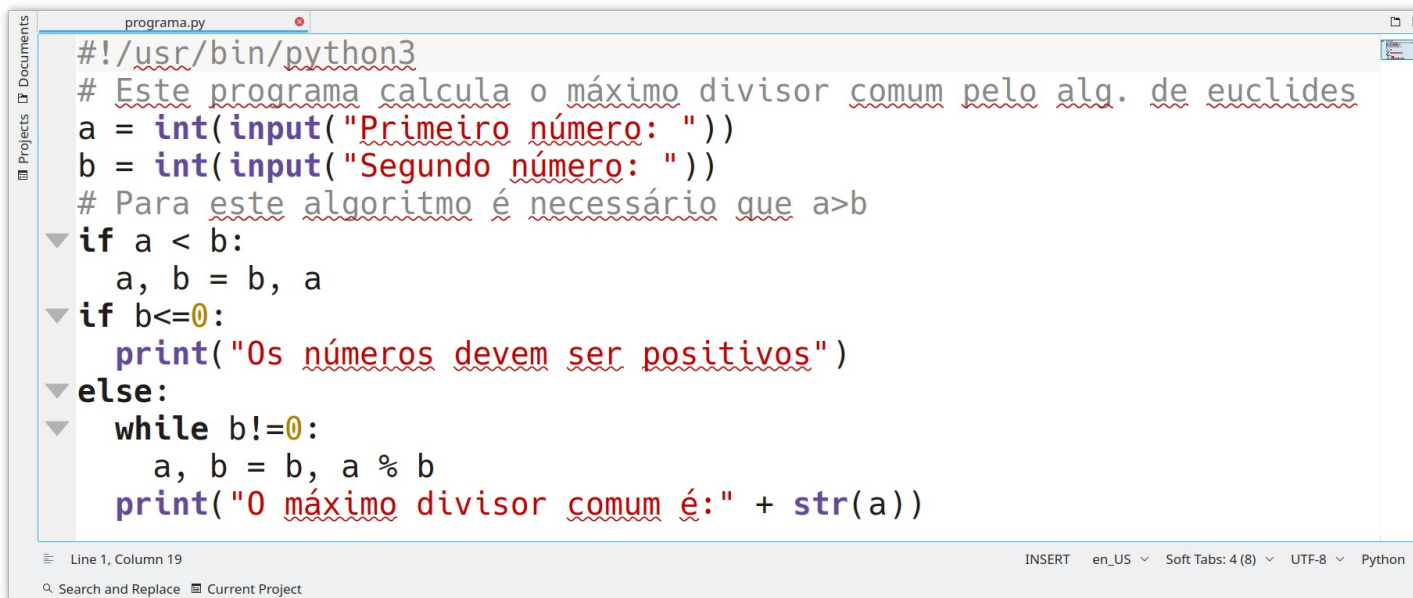
Invocar o interpretador:

```
thiago@betelgeuse:~$ python3 programa.py
Primeiro número: 125
Segundo número: 75
O máximo divisor comum é:25
thiago@betelgeuse:~$
```

Programação em Python

Execução:

Em ambientes -ux o próprio script pode ser executado diretamente!

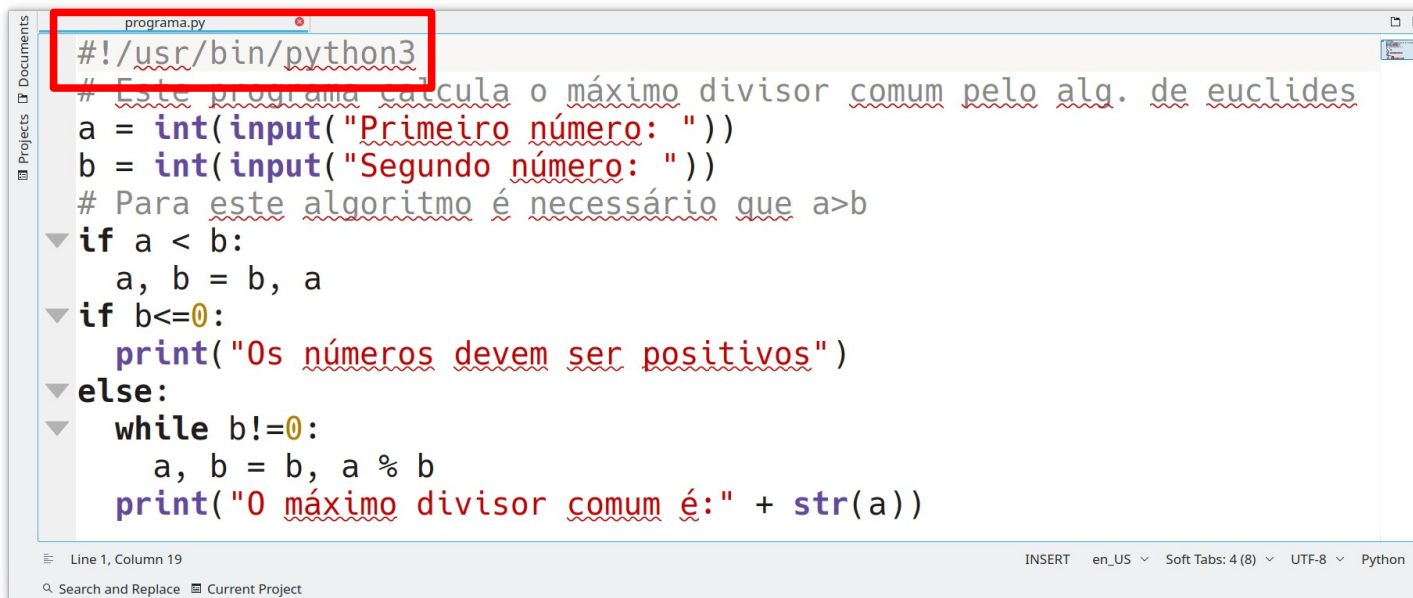
A screenshot of a code editor window titled 'programa.py'. The code is a Python script for calculating the Greatest Common Divisor (GCD) using the Euclidean algorithm. It includes comments in Portuguese and uses standard Python syntax for input, conditionals, and loops. The editor interface shows a sidebar with 'Projects' and 'Documents', a status bar at the bottom indicating 'Line 1, Column 19', and various settings like 'en_US', 'Soft Tabs: 4 (8)', 'UTF-8', and 'Python'.

```
#!/usr/bin/python3
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Programação em Python

Execução:

Em ambientes -ux o próprio script pode ser executado diretamente!



```
#!/usr/bin/python3
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Programação em Python

Execução:

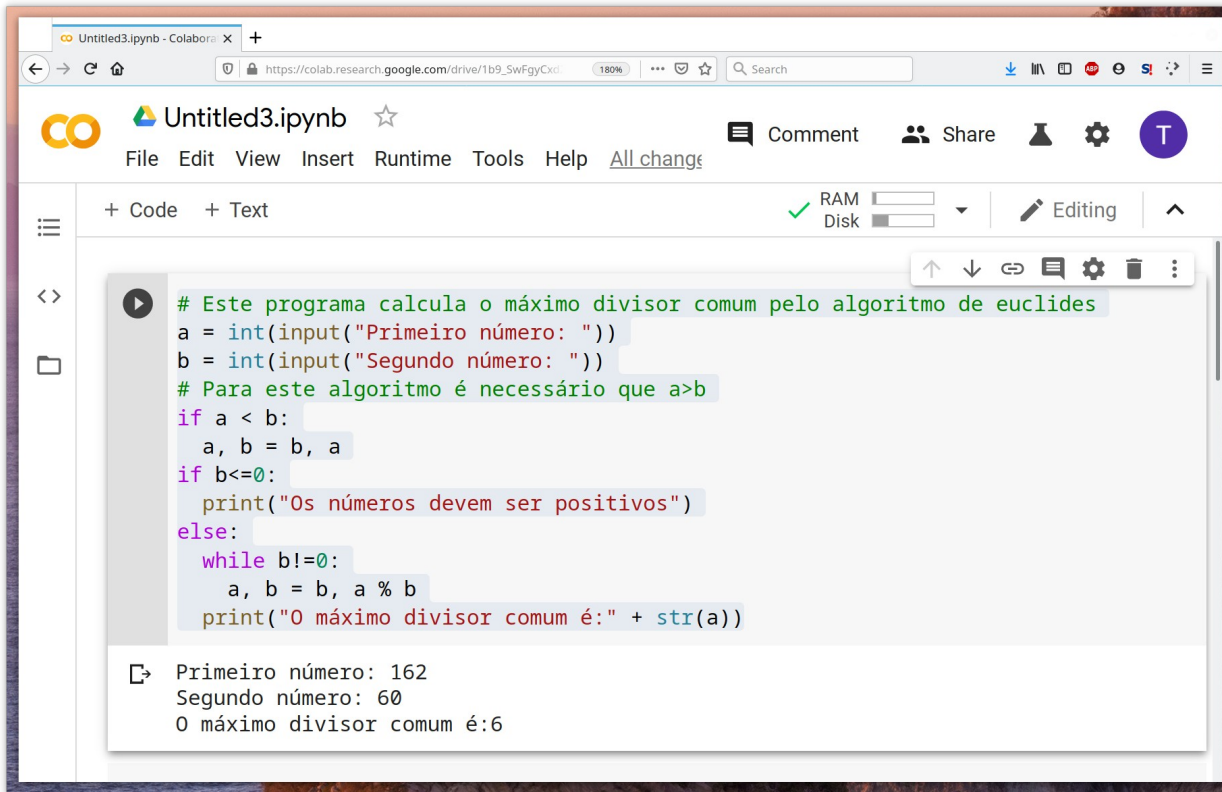
Em ambientes -ux o próprio script pode ser executado diretamente!

```
thiago@betelgeuse:~$ ./programa.py
Primeiro número: 563
Segundo número: 221
O máximo divisor comum é:1
thiago@betelgeuse:~$
```

Programação em Python

Execução:

É possível inserir programas em células do jupyter:



```
# Este programa calcula o máximo divisor comum pelo algoritmo de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Primeiro número: 162
Segundo número: 60
O máximo divisor comum é:6

Programação em Python

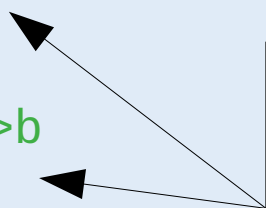
Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```



Comentários
(ignorados pelo
interpretador)

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Chamadas a funções
(subprogramas)

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Construtores de objetos



Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b <= 0:
    print("Os números devem ser positivos")
else:
    while b != 0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Controle de fluxo

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Blocos delimitados por
indentação

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```

Blocos delimitados por
indentação

Comentários

São marcados pelo caractere # e vão até o final da linha

```
# Inicializa o valor de a
```

```
a = 1
```

Podem se seguir a instruções

```
a = 1 # Inicializa o valor de a
```

Atenção! Docstrings não são comentários

```
""" Isso é uma docstring """
```

```
a = 1
```


Controle de fluxo

if... elif... else

```
if <condição 1>:  
    <bloco 1>  
elif <condição 2>:  
    <bloco 2>  
elif <condição 3>:  
    <bloco 3>  
else  
    <bloco alternativo>
```

(não é) Controle de fluxo

Expressão ternária em python, escolhe entre 2 valores dependendo de uma condição.

`<valor se verdadeiro> if <condição> else <valor se falso>`

Exemplo: Uma expressão que vale o *módulo* de x.

```
x if x >= 0 else -x
```

Controle de fluxo

while

```
while <condição>  
    <bloco>
```

Executa o bloco enquanto a condição for verdadeira.

Controle de fluxo

for

```
for <variável> in <coleção enumerável>  
    <bloco>
```

Executa o bloco atribuindo à variável valores sucessivos da coleção

ex:

```
a = 0
```

```
for i in [1, 2, 3]:
```

```
    a += i
```

```
print(a)
```

Controle de fluxo

for e range

A função range é particularmente apropriada para o laço for:

```
a = 0
```

```
for i in range(3)
```

```
    a += i
```

```
print(a)
```

Controle de fluxo

for e range

`range(n)` Enumera os inteiros de 0 a $n-1$

`range(n, m)` Enumera os inteiros de n a $m-1$

`range(n, m, s)`

Enumera os inteiros de n , $n+2s$, $n+3s$... até o maior valor de $n+ks < m$

Controle de fluxo

for e range

`range(n)` Enumera os inteiros de 0 a $n-1$

`range(n, m)` Enumera os inteiros de n a $m-1$

`range(n, m, s)`

Enumera os inteiros de n , $n+2s$, $n+3s$... até o maior valor de $n+ks < m$

Controle de fluxo

break, continue, else, pass

break força a saída *imediata* do bloco.

```
# Gera todos os primos menores que 100
primos = []
for i in range(2, 100):
    primo = True
    for div in primos:
        if (i % div) == 0:
            primo = False
            break
    if primo:
        primos.append(i)
```

continue força o encerramento imediato da execução atual do bloco

```
for x in range(n):
    """mostra números ímpares."""
    if x % 2 == 0:
        continue
    print(x)
```


Chamadas a funções/métodos

“Funções” em python são distintas do conceito matemático (verdade para toda linguagem imperativa estruturada)

```
a = iter([1, 2, 3, 4, 5])
```

```
next(a) → 1
```

```
next(a) → 2
```

```
next(a) → 3
```

O antigo termo “subprograma” evita esta confusão, mas não é mais usual.

Chamadas a funções/métodos

“Funções” em python são distintas do conceito matemático (verdade para toda linguagem imperativa estruturada)

```
a = iter([1, 2, 3, 4, 5])
```

```
next(a) → 1
```

```
next(a) → 2
```

```
next(a) → 3
```

O antigo termo “subprograma” evita esta confusão, mas não é mais usual.

Definindo uma Função

```
def mostra_fib(n):  
    """Mostra a sequência de Fibonacci até n."""  
    a, b = 0, 1  
    while a < n:  
        print(a, end=' ')  
        a, b = b, a+b  
    print()
```

```
def <nome da funcao> (<parâmetros>):  
    <bloco da função>
```

Exemplo adaptado da documentação de Python 3

Definindo uma Função

Funções podem retornar valores

```
def gcd(a, b):  
    """Calcula GCD(a,b) com a>b>0."""  
    while b != 0:  
        a, b = b, a % b  
    return a
```

return <expressão>

```
def procura(a, b):  
    """Encontra índice i tal que a[i]=b."""  
    for i, x in enumerate(a):  
        if x == b:  
            return i
```

return encerra *imediatamente* a execução da função.

Return é opcional!

Funções

Funções também são objetos.

```
gcd(125, 121) → 1
```

```
gcd → function at xxxx
```

```
a = gcd
```

```
a(30, 36) → 6
```

Como mencionado, a *docstring* não é um comentário:

```
help(gcd) → Calcula GCD(a, b) com a>b>0
```

Funções e escopo

Os parâmetros são variáveis definidas *dentro* do bloco da função.

Os nomes de parâmetros podem sobrepor-se a nomes já definidos!

```
a = 10
def mostra(a):
    print(a)

print(a)
mostra(20)
```

Isso se aplica a *toda* variável criada dentro do bloco da função

Variáveis criadas dentro de blocos de funções são ditas *locais*.

Tais variáveis são *invisíveis* fora do bloco de sua criação

```
def soma(a, b):
    s = a + b
    return s

print(soma(2,2))
print(s)
```

Funções e escopo

Os parâmetros são *cópias* de *referências*.

Uma função não pode alterar uma referência original.

```
def zera_variavel(a):
```

```
    a = 0
```

```
teste = 1
```

```
zera_variavel(teste)
```

```
print(teste)
```

Funções e escopo

Os parâmetros são *cópias* de *referências*.

Mas objetos mutáveis podem ser alterados.

```
def zera_posicao(a, i):
```

```
    a[i] = 0
```

```
teste = [1, 2]
```

```
zera_variavel(teste, 0)
```

```
print(teste)
```


Funções e escopo

Variáveis globais são visíveis dentro de funções.

```
var = "teste"
def func():
    print(var)
func()
var = "teste2"
func()
```

Funções e escopo

Variáveis globais são visíveis dentro de funções.

MAS atribuições criam novas variáveis!

```
var = "teste"
```

```
def func():
```

```
    var = "novo valor"
```

```
    print(var)
```

```
func()
```

```
print(var)
```

Funções e escopo

A palavra-chave “global” atrela um símbolo ao escopo global.

```
var = "teste"
def func():
    global var
    var = "novo valor"
func()
print(var)
```

Funções e escopo

Funções podem ser definidas dentro do escopo de outras funções.

```
def segundo_grau(vetor, a, b, c):  
    """ Aplica  $ax^2+bx+c$  em cada elemento de vetor """  
    def valor(x):  
        return a*x*x + b*x + c  
    for i, x in vetor:  
        vetor[i] = valor(x)
```

Este recurso é mais poderoso do que parece! Mais sobre isso em clausuras.

Funções e escopo

O escopo é *reentrante* (cada chamada gera seu próprio escopo). Mesmo quando a função é chamada dentro de si mesma. Isso permite *recursão*.

```
def gcd(a, b):  
    return a if b==0 else gcd(b, a%b)
```

$\text{gcd}(8,5) = \text{gcd}(5,3) = \text{gcd}(3,2) = \text{gcd}(2,1) = \text{gcd}(1,0) = 1$

Cada chamada recursiva tem seus valores próprios para a e b!

Funções – Mais sobre parâmetros

Valores padrão de parâmetros.

```
def fatorial(x, y=1):  
    if x==1:  
        return y  
    else:  
        return fatorial(x-1, x*y)
```

Funções – Mais sobre parâmetros

Valores padrão de parâmetros.

```
a = 1
```

```
def soma(x, y=a):  
    return x+y
```

```
a = 2
```

```
soma(2)
```

O valor de atributos padrão é avaliado somente na definição da função.

Funções – Mais sobre parâmetros

Parâmetros rotulados.

```
def segundo_grau(vetor, c=0, b=0, a=0):  
    """ Aplica  $ax^2+bx+c$  em cada elemento de vetor """  
    def valor(x):  
        return a*x*x + b*x + c  
    for i, x in enumerate(vetor):  
        vetor[i] = valor(x)  
  
segundo_grau(vetor, a=1)
```


Funções – Mais sobre parâmetros

Parâmetros rotulados.

```
def segundo_grau(vetor, c=0, b=0, a=0):
```

```
...
```

Parâmetros rotulados sempre aparecem *depois* de posicionais!.

```
segundo_grau([1, 2, 3], a=1) ok!
```

```
segundo_grau(a=1, [1, 2, 3]) errado!
```

Funções – Mais sobre parâmetros

Funções variádicas (tupla)

```
def soma_parametros(*x):  
    soma = 0  
    for i in x:  
        soma+=i  
    return soma
```

Com parâmetros rotulados (dicionário):

```
def mostra_parametros(*posicionais, **rotulados):  
    print(posicionais)  
    print(rotulados)
```

Programação em Python

Elementos do programa

```
# Este programa calcula o máximo divisor comum pelo alg. de euclides
a = int(input("Primeiro número: "))
b = int(input("Segundo número: "))
# Para este algoritmo é necessário que a>b
if a < b:
    a, b = b, a
if b<=0:
    print("Os números devem ser positivos")
else:
    while b!=0:
        a, b = b, a % b
    print("O máximo divisor comum é:" + str(a))
```