

Computação 1 - Python

Aula 2 - Teórica: Função

Exercício: Calcule a área da coroa circular (anel) formada por dois círculos de raios r_1 e r_2 ($r_1 > r_2$ e $Pi = 3.14$).

Função

Exercício: Calcule a área da coroa circular (anel) formada por dois círculos de raios r_1 e r_2 ($r_1 > r_2$ e $Pi = 3.14$).

```
# Função que calcula a coroa circular formada  
# pelos círculos de raio r1 e r2  
def coroa(r1,r2):  
    return (3.14*r1**2) - (3.14*r2**2)
```

Python Tutor – Passo a Passo da Execução (Chinês)

Python 2.7

```
1 #Função Coroa Circular
2
→ 3 def coroa(r1,r2):
4     return (3.14*r1**2) - (3.14*r2**2)
5
6 #Calculando a coroa circular formada
7 #pelos círculos de raios 3 e 2
8
9 coroa(3,2)
```

[Edit code](#)

Frames

Objects

Python Tutor – Passo a Passo da Execução (Chinês)

Python 2.7

```
1 #Função Coroa Circular
2
→ 3 def coroa(r1,r2):
4     return (3.14*r1**2) - (3.14*r2**2)
5
6 #Calculando a coroa circular formada
7 #pelos círculos de raios 3 e 2
8
→ 9 coroa(3,2)
```

[Edit code](#)

Frames

Objects



Python Tutor – Passo a Passo da Execução (Chinês)

Python 2.7

```
1 #Função Coroa Circular
2
3 → def coroa(r1,r2):
4     return (3.14*r1**2) - (3.14*r2**2)
5
6 #Calculando a coroa circular formada
7 #pelos círculos de raios 3 e 2
8
9 → coroa(3,2)
```

[Edit code](#)

Frames

Objects

Global frame

coroa

function

coroa(r1, r2)

coroa

r1

3

r2

2

Python Tutor – Passo a Passo da Execução (Chinês)

Python 2.7

```
1 #Função Coroa Circular
2
3 def coroa(r1,r2):
4     return (3.14*r1**2) - (3.14*r2**2)
5
6 #Calculando a coroa circular formada
7 #pelos círculos de raios 3 e 2
8
9 coroa(3,2)
```

[Edit code](#)

Frames

Objects

Global frame

coroa

function

coroa(r1, r2)

coroa

r1

3

r2

2

Python Tutor – Passo a Passo da Execução (Chinês)

Python 2.7

```
1 #Função Coroa Circular
2
3 def coroa(r1,r2):
4     return (3.14*r1**2) - (3.14*r2**2)
5
6 #Calculando a coroa circular formada
7 #pelos círculos de raios 3 e 2
8
9 coroa(3,2)
```

[Edit code](#)

Frames

Objects

Global frame

coroa

function

coroa(r1, r2)

coroa

r1

3

r2

2

Return
value

15.7

Função

Exercício: Calcule a área da coroa circular (anel) formada por dois círculos de raios $r1$ e $r2$ ($r1 > r2$ e $Pi = 3.14$).

```
# Função que calcula a coroa circular formada  
# pelos círculos de raio r1 e r2  
def coroa(r1,r2):  
    return (3.14*r1**2) - (3.14*r2**2)
```

Exercício: Calcule a área de um círculo de raio $r1$.

Função

Exercício: Calcule a área da coroa circular (anel) formada por dois círculos de raios r_1 e r_2 ($r_1 > r_2$ e $Pi = 3.14$).

```
# Função que calcula a coroa circular formada  
# pelos círculos de raio r1 e r2  
def coroa(r1,r2):  
    return (3.14*r1**2) - (3.14*r2**2)
```

Exercício: Calcule a área de um círculo de raio r_1 .

```
# Função que calcula a área de  
# um círculo de raio r1  
def areac(r1):  
    return 3.14*r1**2
```

Função

Exercício: Calcule a área da coroa circular (anel) formada por dois círculos de raios r_1 e r_2 ($r_1 > r_2$ e $Pi = 3.14$).

```
# Função que calcula a coroa circular formada  
# pelos círculos de raio r1 e r2  
def coroa(r1,r2):  
    return (3.14*r1**2) - (3.14*r2**2)
```

Exercício: Calcule a área de um círculo de raio r_1 .

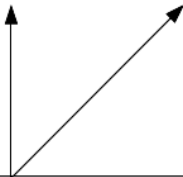
```
# Função que calcula a área de  
# um círculo de raio r1  
def areac(r1):  
    return 3.14*r1**2
```

O que estes programas têm em comum?

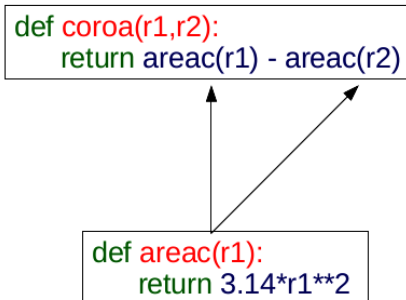
Função

```
def coroa(r1,r2):  
    return (3.14*r1**2) - (3.14*r2**2)
```

```
def areac(r1):  
    return 3.14*r1**2
```



Função



***Posso chamar uma
função a partir de
outra !***

Função

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

```
def areac(r1):  
    return 3.14*r1**2
```

coroa(3,2)

Função

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

```
def areac(r1):  
    return 3.14*r1**2
```

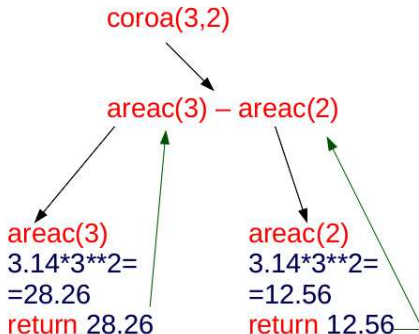
coroa(3,2)

areac(3) - areac(2)

Função

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

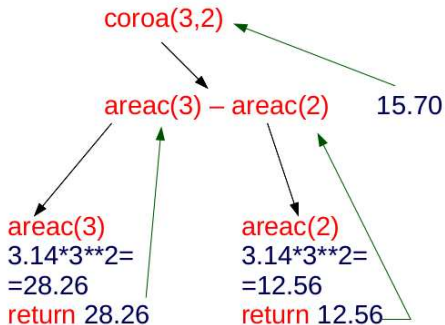
```
def areac(r1):  
    return 3.14*r1**2
```



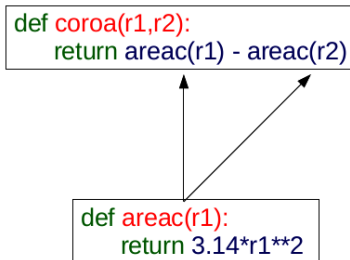
Função

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

```
def areac(r1):  
    return 3.14*r1**2
```

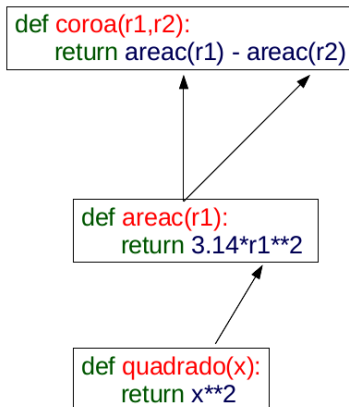


Função



**Podemos usar a
função quadrado
que definimos na
aula anterior**

Função



**Podemos usar a
função quadrado
que definimos na
aula anterior**

Função

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

```
def areac(r1):  
    return 3.14*quadrado(r1)
```

```
def quadrado(x):  
    return x**2
```

**Podemos usar a
função quadrado
que definimos na
aula anterior**

Função

Pi é bastante usado. Por que não definimos uma função para ele ?

```
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```

```
def areac(r1):  
    return 3.14*quadrado(r1)
```

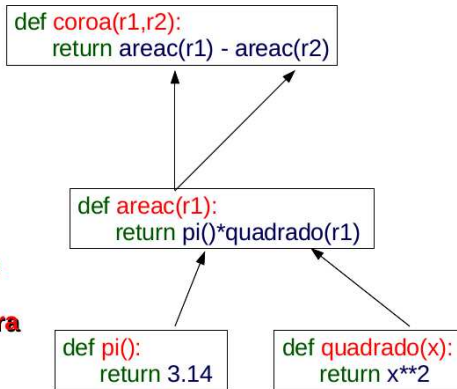
```
def quadrado(x):  
    return x**2
```



```
graph BT; quadrado --> areac; areac --> coroa;
```

Função

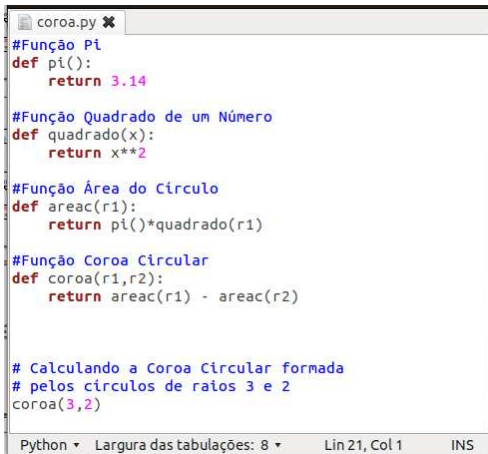
Pi é bastante usado. Por que não definimos uma função para ele ?



Pi é uma função constante.

Python Tutor

Use o Python Tutor para ver como estas funções funcionam



```
coroa.py X
#Função Pi
def pi():
    return 3.14

#Função Quadrado de um Número
def quadrado(x):
    return x**2

#Função Área do Círculo
def areac(r1):
    return pi()*quadrado(r1)

#Função Coroa Circular
def coroa(r1,r2):
    return areac(r1) - areac(r2)

# Calculando a Coroa Circular formada
# pelos círculos de raios 3 e 2
coroa(3,2)

Python ▾ Largura das tabulações: 8 ▾ Lin 21, Col 1 INS
```

Função

Exemplo: Defina uma função que dados dois inteiros x e y , retorna x^y .

Função

Exemplo: Defina uma função que dados dois inteiros x e y , retorna x^y .
Temos a função que eleva um número ao quadrado:

```
# Função que dado um inteiro x  
# retorna x elevado a 2  
def quadrado(x):  
    return x**2
```

Função

Exemplo: Defina uma função que dados dois inteiros x e y , retorna x^y .
Temos a função que eleva um número ao quadrado:

```
# Função que dado um inteiro x  
# retorna x elevado a 2  
def quadrado(x):  
    return x**2
```

Poderíamos facilmente definir a função *potencia*:

Função

Exemplo: Defina uma função que dados dois inteiros x e y , retorna x^y .
Temos a função que eleva um número ao quadrado:

```
# Função que dado um inteiro x  
# retorna x elevado a 2  
def quadrado(x):  
    return x**2
```

Poderíamos facilmente definir a função *potencia*:

```
# Função que dados os inteiros x e y  
# retorna x elevado a y  
def potencia(x,y):  
    return x**y
```

Função

Exemplo: Defina uma função que dados dois inteiros x e y , retorna x^y .
Temos a função que eleva um número ao quadrado:

```
# Função que dado um inteiro x  
# retorna x elevado a 2  
def quadrado(x):  
    return x**2
```

Poderíamos facilmente definir a função *potencia*:

```
# Função que dados os inteiros x e y  
# retorna x elevado a y  
def potencia(x,y):  
    return x**y
```

Na verdade, podemos ficar só com esta função:

potencia(x,2)

Python Tutor

```
1 # Dados dois valores x e y,  
2 # a função potência retorna o  
3 # valor de x elevado a y  
4  
→ 5 def potencia(x,y):  
6     return x**y  
7  
8 # teste 1  
9 potencia(3,2)  
10  
11 # teste 1  
12 potencia(2,3)
```

[Edit code](#)



<< First < Back Step 1 of 9 Forward > Last >>

→ line that has just executed

→ next line to execute

Função

Podemos definir a função potencia de outra forma:

```
# Função que dados dois inteiros x e y  
# retorna x elevado a y  
def potencia(x,y=2):  
    return x**y
```

O que fizemos foi definir um **argumento default**, ou seja, no exemplo, se o usuário não fornecer o segundo parâmetro, a função considera seu valor igual a 2.

```
>>> potencia(5)  
25  
>>> potencia(5,3)  
125
```

Função

Argumentos Default: Permitem que valores default sejam utilizados quando nenhum valor é especificado em um certo parâmetro.


Formato


def nome-funcao($arg_0, \dots, arg_N, arg_{N+1} = default_1, \dots, arg_M = default_M$)

...

- arg_0, \dots, arg_N : Argumentos sem valores default.
- $arg_{N+1} = default_1, \dots, arg_M = default_M$: Argumentos com valores default. Devem ser sempre os últimos argumentos.

Função



```
def pi():  
    return 3.14  
  
def potencia(x,y=2):  
    return x**y  
  
def areac(r1):  
    return pi()*potencia(r1)   
  
def coroa(r1,r2):  
    return areac(r1) - areac(r2)
```


Python Tutor

```
1 #Função Pi
2 def pi():
3     return 3.14
4
5 #Função Potência de um Número
6 def potencia(x,y=2):
7     return x**y
8
9 #Função Área do Círculo
10 def areac(r1):
11     return pi()*potencia(r1)
12
13 #Função Coroa Circular
14 def coroa(r1,r2):
15     return areac(r1) - areac(r2)
16
17 # Calculando a Coroa Circular formada
18 # pelos círculos de raios 3 e 2
19 coroa(3,2)
```

[Edit code](#)

Tipos Numéricos

- **Tipo inteiro (int)** : 10
- **Tipo inteiro longo (long)** : 10000L
- **Tipo ponto flutuante (float)**: 10.5 , -190.00005 , $15e - 5$
- **Tipo complexo (complex)** : $3 + 2j$, $20j$

Tipos Numéricos

■ Números Inteiros: **Int** / **Long**

Os inteiros (int) têm precisão fixa ocupando tipicamente uma palavra de memória

Em PC's são tipicamente representados com 32 bits (de -2^{31} a $2^{31} - 1$)

Os números inteiros de precisão arbitrária (long) são armazenados em tantas palavras quanto necessário.

Constantes do tipo long têm o sufixo L ou l.

Longs são manipulados bem mais lentamente que ints.

Quando necessário, cálculos usando ints são convertidos para longs.

Tipos Numéricos

- **Ponto Flutuante:** **Float**

Constantes têm que possuir um ponto decimal ou serem escritas em notação científica com a letra “e” (ou “E”) precedendo a potência de 10

10 int

10.0 float

- **Números Complexos:** **Complex**

Representados com dois números de ponto flutuante: um para a parte real e outro para a parte imaginária.

Constantes são escritas como uma soma sendo que a parte imaginária tem o sufixo j ou J

$2 + 3j$ $7j$ $5 + 0j$

Exercícios

1.
 - a. Defina as funções *base(r)*, *lateral(r,h)*, *total(r,h)* para calcular as áreas da base, da lateral e também a área total de um cilindro reto.
 - b. Faça o chinês para os seguintes casos:

Chamada da Função	Valor de Retorno
base(3)	?
lateral(3,4)	?
total(3,4)	?

Exercícios

2. a. Dado o valor de uma conta, faça a função *conta(valor,gorjeta)* que calcule o valor da conta com a gorjeta incluída. Considere que é possível que a gorjeta seja maior ou menor que 10%. Quando o parâmetro gorjeta não for informado, sua função deve assumir que a gorjeta é de 10%. Use uma função para calcular a gorjeta e outra para calcular o valor total da conta.
- b. Faça o chinês para os seguintes casos:

Chamada da Função	Valor de Retorno
conta(123,5)	?
conta(-230)	?

Módulo *math*

Módulo que permite que o programador realize certos cálculos matemáticos. Para usar uma função que está definida em um módulo, **primeiro** o programa deve importar o módulo usando o comando ***import***:

```
>>> import math
```

Após ter importado o módulo, o programa pode chamar as funções daquele módulo da seguinte forma:

NomeDoModulo.NomeDaFuncao(arg₀, ..., arg_n)

Exemplo

```
>>> math.sqrt(81)
9.0
```

- **Módulo:** math
- **Função:** sqrt
- **Parâmetro:** 81

Módulo *math*

Módulo que permite que o programador realize certos cálculos matemáticos. Para usar uma função que está definida em um módulo, **primeiro** o programa deve importar o módulo usando o comando *import*:

```
>>> import math
```

Podemos importar parte dos módulos:

- **from math import *** : importa todos os elementos do módulo *math*
- **from math import nome-função** : importa apenas a função nome-função.

Exemplos

```
>>> from math import *
```

```
>>> from math import sin
```


Módulo *math* - Exemplos

```
>>> from math import sin
```

```
>>> sin(30)
-0.988031624093
```

```
>>> sin(radians(30))
```

```
Traceback (most recent call last):
```

```
File "<pyshell#4>", line 1, in <module>
```

```
    sin(radians(30))
```

```
NameError: name 'radians' is not defined
```

```
>>> sin(math.radians(30))
```

```
Traceback (most recent call last):
```

```
File "<pyshell#5>", line 1, in <module>
```

```
    sin(math.radians(30))
```

```
NameError: name 'math' is not defined
```

```
>>> from math import *
```

```
>>> sin(radians(30))
0.49999999999999994
```

Módulo

- Para ter acesso aos módulos do python:

```
>>> help()
help> modules
```

- Para saber sobre um módulo específico, basta digitar o nome:

```
>>> import math
>>> help(math.cos)
```

Help on built-in function cos in module math:

```
cos(...)
cos(x)
```

Return the cosine of x (measured in radians).

Pressiona-se “q” para retornar ao interpretador.

Exercícios

- 1 Redefina a função que calcula a área do círculo usando o valor de π definido no módulo *math*.
- 2 Escreva uma função que determina o número de arranjos simples de n elementos agrupados k a k . Lembre:
$$A_{n,k} = \frac{n!}{(n-k)!}$$
- 3 Escreva uma função que determina o número de combinações simples de n elementos agrupados k a k . Use a função definida no exercício 2. Lembre: $C_{n,k} = \frac{n!}{k!(n-k)!}$

Computação 1 - Python

Aula 2 - Teórica: Função