Computação I - Python
Aula 2 - Teórica: Função

João C. P. da Silva

Carla A. D. M. Delgado

Ana Luisa Duboc

Dept. Ciência da Computação - UFRJ

Função

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

Função

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

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

Função

```
>>> def coroa(r1,r2):
        """Funcao que calcula a coroa circular formada pelos
    circulos de raio r1 e r2  (r1 > r2)"""
        return (3.14*r1**2) - (3.14*r2**2)

>>> coroa(3,2)
    15.700000000000001

>>> coroa(2,3)
    -15.700000000000001
```

Testar sua função é muito importante !

Função

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

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

Calcule a área de um círculo de raio R.

Função

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

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

Calcule a área de um círculo de raio R.

```
1 def areac(R):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*R**2
```

Função

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

```
1 def areac(R):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*R**2
```

Os parâmetros das funções são *locais*: um parâmetro só existe dentro de sua função. Isso significa que duas funções diferentes podem ter parâmetros com o mesmo nome.

Função

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

```
1 def areac(R):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*R**2
```

Os parâmetros das funções são *locais*: um parâmetro só existe dentro de sua função. Isso significa que duas funções diferentes podem ter parâmetros com o mesmo nome.

```
1 def areac(r1):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*r1**2
```

Função

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

```
1 def coroa(r1,r2):
2     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
3     return (3.14*r1**2) - (3.14*r2**2)
```

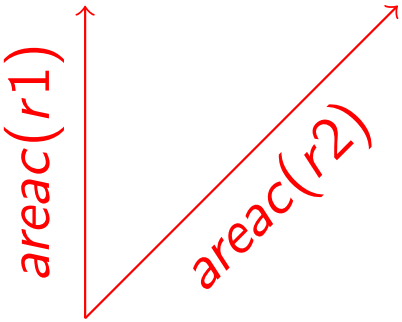
Calcule a área de um círculo de raio R.

```
1 def areac(R):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*R**2
```

O que estas duas funções têm em comum?

Função

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




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

Posso chamar uma função a partir de outra!

Função

```
1 def areac(R):
2     """Funcao que calcula a area de um circulo de raio R"""
3     return 3.14*R**2
4
5 def coroa(r1,r2):
6     """Funcao que calcula a coroa circular formada pelos circulos
      de raio r1 e r2 (r1 > r2)"""
7     return areac(r1) - areac(r2)
```

Posso chamar uma função a partir de outra!

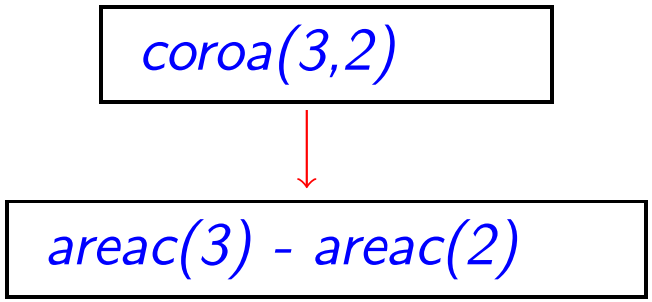
Função

Chamamos a função *coroa* com os parâmetros 3 e 2

```
coroa(3,2)
```

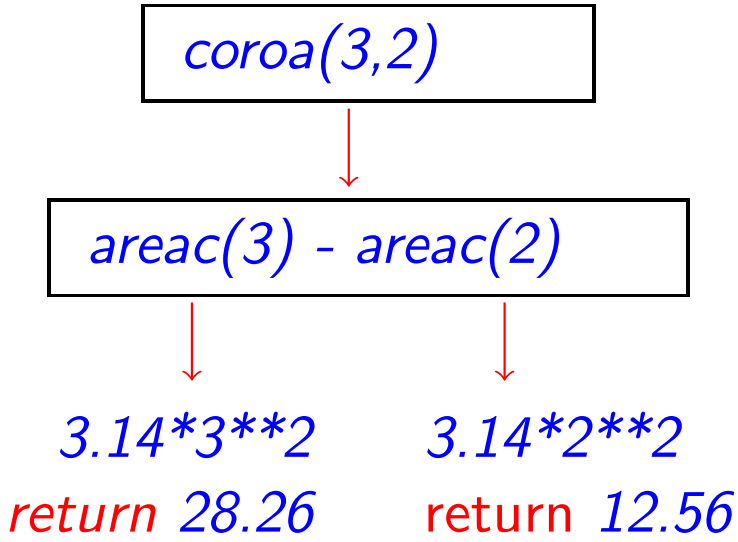
Função

Chamamos a função *coroa* com os parâmetros 3 e 2



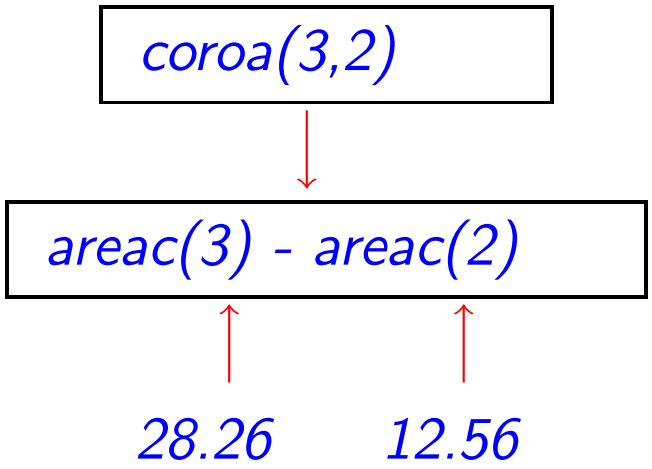
Que chama a função *areac* duas vezes: uma com o parâmetro 3 e outra com o parâmetro 2

Função



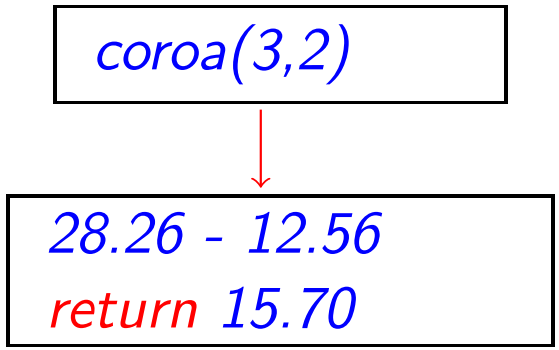
Cada chamada da função *areac* retorna o valor calculado para a função *coroa*

Função



Cada chamada da função *areac* retorna o valor calculado para a função *coroa*

Função

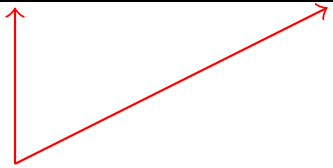


A função *coroa* usa os valores retornados pelas chamadas da função *areac* e calcula o valor da coroa.

Função

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

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



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

Função

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

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

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

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

Função

```
1  def quadrado(X):
2      """Funcao que retorna o quadrado de um numero"""
3      return X**2
4
5  def areac(R):
6      """Funcao que calcula a area de um circulo de raio R"""
7      return 3.14*quadrado(R)
8
9  def coroa(r1,r2):
10     """Funcao que calcula a coroa circular formada pelos circulos
       de raio r1 e r2 (r1 > r2)"""
11     return areac(r1) - areac(r2)
```

Função

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

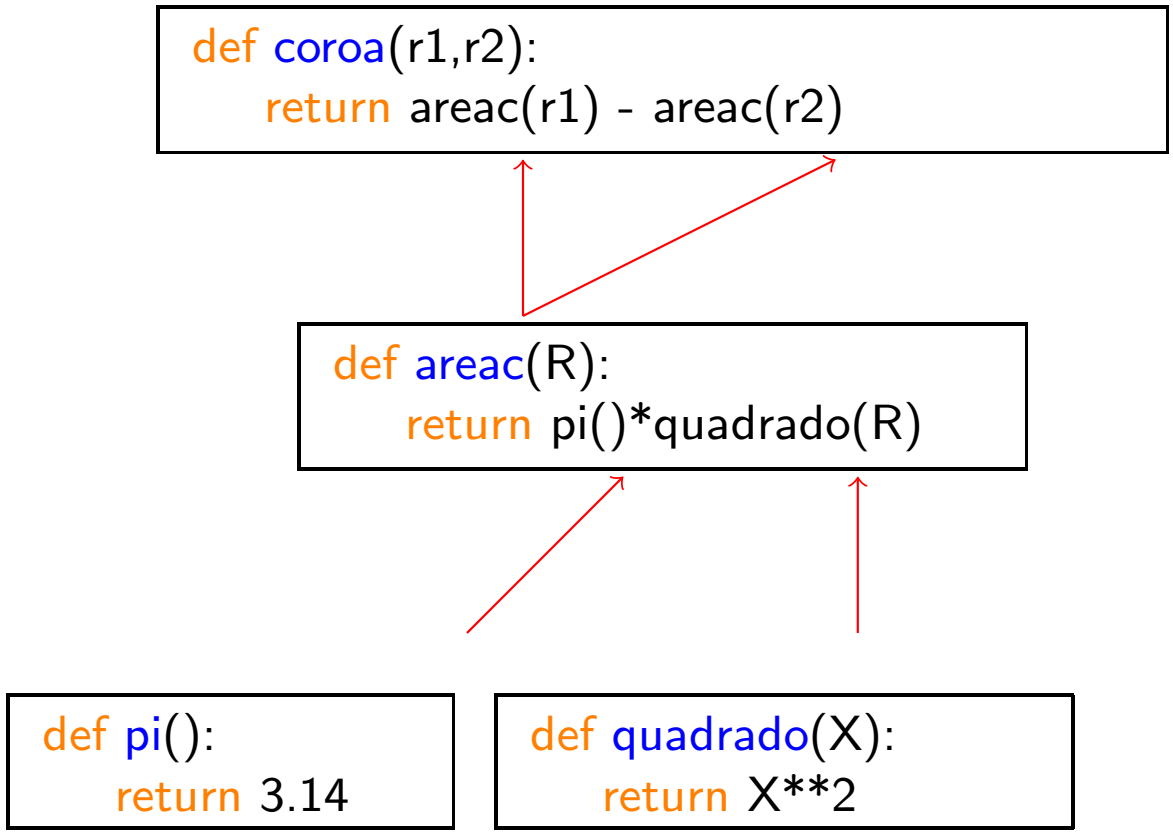
```
def coroa(r1,r2):
    return areac(r1) - areac(r2)
```

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

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

Função

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



Função

```
1  def pi():
2      """Funcao que define o valor de Pi como sendo 3.14"""
3      return 3.14
4
5  def quadrado(X):
6      """Funcao que retorna o quadrado de um numero"""
7      return X**2
8
9  def areac(R):
10     """Funcao que calcula a area de um circulo de raio R"""
11     return pi()*quadrado(R)
12
13 def coroa(r1,r2):
14     """Funcao que calcula a coroa circular formada pelos circulos
       de raio r1 e r2 (r1 > r2)"""
15     return areac(r1) - areac(r2)
```

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:

```
1 def quadrado(X):
2     """Funcao que retorna o quadrado de um numero"""
3     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:

```
1 def quadrado(X):
2     """Funcao que retorna o quadrado de um numero"""
3     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:

```
1 def quadrado(X):
2     """Funcao que retorna o quadrado de um numero"""
3     return X**2
```

Poderíamos facilmente definir a função *potencia*:

```
1 def potencia(X,Y):
2     """Funcao que dados os inteiros X e Y retorna X elevado a Y"""
3     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:

```
1 def quadrado(X):
2     """Funcao que retorna o quadrado de um numero"""
3     return X**2
```

Poderíamos facilmente definir a função *potencia*:

```
1 def potencia(X,Y):
2     """Funcao que dados os inteiros X e Y retorna X elevado a Y"""
3     return X**Y
```

Na verdade, podemos ficar só com esta função:

potencia(x,2)

```
1 def potencia(X,Y):
2     """Funcao que dados os inteiros X e Y retorna X elevado a Y"""
3     return X**Y
```

```
1 >>> potencia(3,2)
2     9
3
4 >>> potencia(2,3)
5     8
```

Função

Podemos definir a função potencia de outra forma:

```
1 def potencia(X,Y=2):
2     """Funcao que dados os inteiros X e Y retorna X elevado a Y.
      Caso o valor de Y n o seja passado, o numero X sera elevado ao
       quadrado."""
3     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.

```
1 >>> potencia(5)
2     25
3
4 >>> potencia(5,3)
5     125
```

Função

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

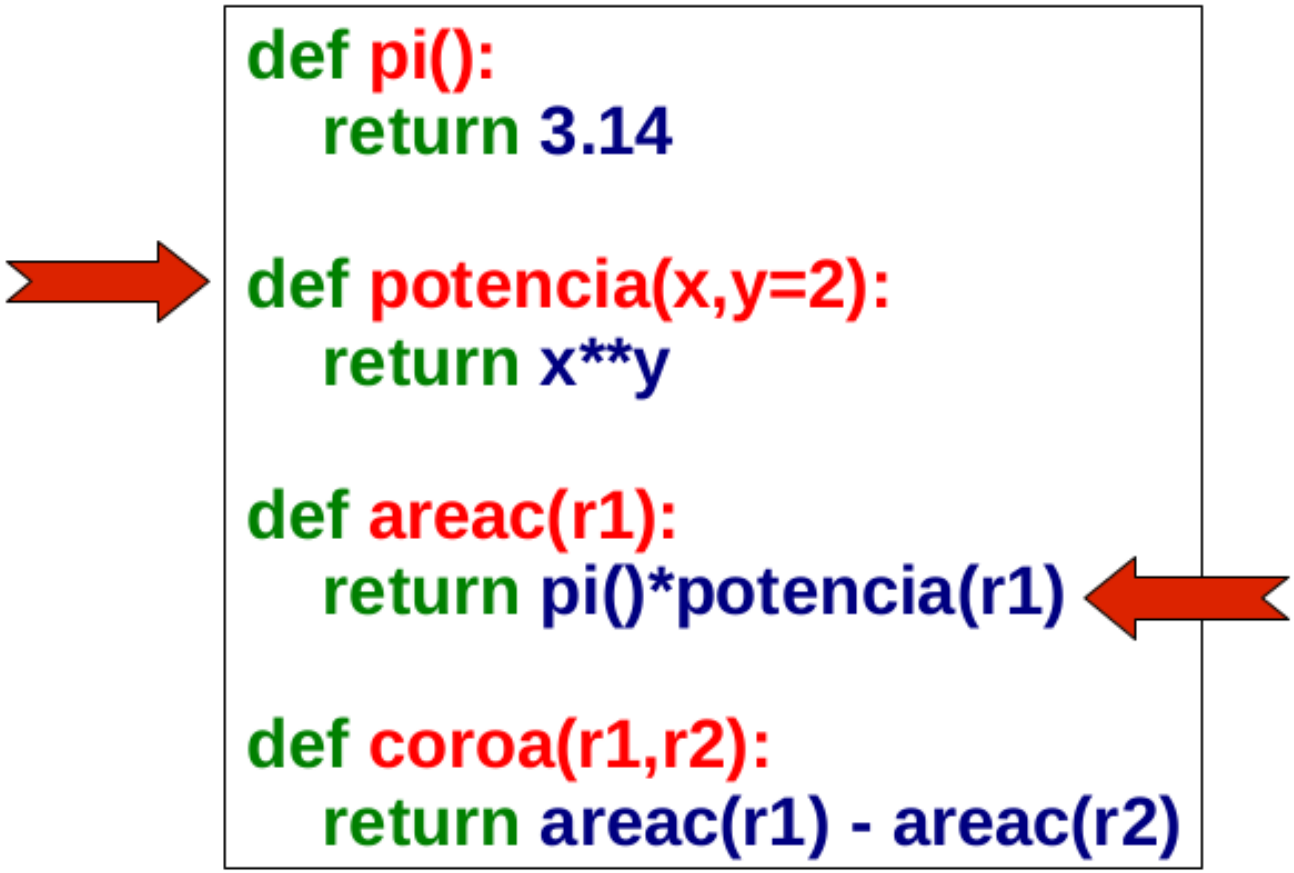
Formato

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

$$\cdots$$

- $arg_0, \cdots, arg_N$: *Argumentos sem valores default.*
- $arg_{N+1} = default_1, \cdots, arg_M = default_M$: *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)

Tipos Numéricos

Em computação, um tipo de dado é uma classificação dos dados. Essa classificação determina como os dados serão armazenados no computador e também permite a disponibilização de operações pré definidas na linguagem de programação.

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

Tipos Numéricos

- **Números Inteiros**: Int

  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 $2^{31}-1$)

- **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* (que deve ser um número do tipo inteiro) 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(123.00,5)	?
conta(-230.00)	?

Exercícios

3. Faça três funções que :
   a. **HoraemMinuto:** Dada uma quantidade em horas, a transforma para minutos.
   b. **MinutoemSegundo:** Dada uma quantidade de minutos, a transforma em segundos.
   c. **HoraemSegundo:** Dada uma quantidade de horas, a transforma em segundos usando os itens a e b.

4. Faça o chinês para a função definida no exercício 3 para as seguintes chamadas:

Chamada da Função	**Valor de Retorno**
HoraemMinuto(13)	?
MinutoemSegundo(45)	?
HoraemSegundo(10)	?

Módulos

- **Módulos Python**: Funções que realizam tarefas comuns tais como cálculos matemáticos, manipulações de strings, manipulação de caracteres, programação Web, programação gráfica, etc.

- **Bibliotecas**: coleção de módulos.

Módulo *math*

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

```
1 >>> import math
```

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

$$NomeDoModulo.NomeDaFuncao(arg_0, \cdots, arg_n)$$

Exemplo

```
1 >>> math.sqrt(81)
2     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 a função 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

```
>>> import math
>>> sin(30)
    Traceback (most recent call last):
    File "<stdin>", line 1, in ?
    NameError: name 'sin' is not defined

>>> math.sin(30)
    -0.988031624093

>>> import math
>>> sin(radians(30))
    Traceback (most recent call last):
    File "<pyshell#4>", line 1, in <module>
    sin(radians(30))
    NameError: name 'sin' is not defined

>>> math.sin(radians(30))
    Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
    math.sin(radians(30))
    NameError: name 'radians' is not defined

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

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:

```
help> math
    Help on built-in module math:
    NAME
        math
    FILE
        (built-in)
    DESCRIPTION
        This module is always available. It provides access to the
        mathematical functions defined by the C standard.
    FUNCTIONS
    acos(...)
        acos(x)
        Return the arc cosine (measured in radians) of x.
```

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 *pi* 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: $C_{n,k} = \frac{n!}{k!(n-k)!}$

Autores

- João C. P. da Silva ▸ Lattes
- Carla Delgado ▸ Lattes
- Ana Luisa Duboc ▸ Lattes

Colaboradores

- Anamaria Martins Moreira ▸ Lattes
- Fabio Mascarenhas ▸ Lattes
- Leonardo de Oliveira Carvalho ▸ Lattes
- Charles Figueiredo de Barros ▸ Lattes
- Fabrício Firmino de Faria ▸ Lattes

Computação I - Python
Aula 2 - Teórica: Função

João C. P. da Silva

Carla A. D. M. Delgado

Ana Luisa Duboc

Dept. Ciência da Computação - UFRJ