Funções

Aula 6

Curso: BIG863 - Basic Python Programming for Ecologists

Professora: Dra. Cecilia F. Fiorini

Supervisor: Prof. Dr. Fernando A. O. Silveira

https://meet.google.com/zdi-ueoz-nsr, 19 de abril de 2023



Roteiro*

Roteiro

- **Funções**
- Exemplos de uso de funções
- Exercício

*Conteúdo adaptado a partir de material desenvolvido pelo Prof. Zanoni Dias e disponível em https://ic.unicamp.br/ mc102.



- Um aspecto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores.
- Sendo assim, ao criarmos um programa para resolver um determinado problema, uma tarefa importante é dividir o código em partes menores, fáceis de serem compreendidas e mantidas.
- As funções nos permitem agrupar um conjunto de comandos, que são executados quando a função é chamada.
- Nas aulas anteriores vimos diversos exemplos de uso de funções (range, sum, min, len, etc).
- Agora vamos nos aprofundar no uso de funções e aprender a criar nossas próprias funções.

- Evitar que os blocos do programa figuem grandes demais e, por consequência, difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de códigos, implementados por você ou por outros programadores.
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, evitando inconsistências e facilitando alterações.



Definindo uma Função

- Para criar uma nova função usamos o comando def.
- Para os nomes das funções valem as mesmas regras dos nomes de variáveis.

```
1 # Minha primeira função
2 def imprime_mensagem():
3     print("Minha primeira função")
4
5 imprime_mensagem()
```



Definindo uma Função

Funcões

• Precisamos sempre definir uma função antes de usá-la.

```
imprime_mensagem()
    NameError: name 'imprime_mensagem' is not defined
3
  def imprime_mensagem():
      print("Minha primeira função")
5
```



Redefinindo uma Função

Funcões

 Uma função pode ser redefinida, para isso basta declararmos outra função utilizando o mesmo nome, mas não necessariamente com o mesmo comportamento.

```
def imprime_mensagem():
      print("Minha função")
3
  def imprime_mensagem():
5
      print("Minha funcão foi redefinida")
6
  imprime_mensagem()
8
    Minha função foi redefinida
```



Escopo de uma Variável

- O escopo de uma variável é o local do programa onde ela é acessível.
- Quando criamos uma variável dentro de uma função, ela só é acessível nesta função. Essas variáveis são chamadas de locais.

```
def imprime_mensagem():
       mensagem = "Variável local"
2
       print(mensagem)
4
   imprime_mensagem()
  # Variável local
  print(mensagem)
8
    NameError: name 'mensagem' is not defined
```



Escopo de uma Variável

 Quando criamos uma variável fora de uma função, ela também pode ser acessada dentro da função. Essas variáveis são chamadas de globais.

```
mensagem = "Variável global"
def imprime_mensagem():
    print(mensagem)

imprime_mensagem()

wariável global
print(mensagem)

# Variável global

Variável global
```



• Uma variável local com o mesmo nome de uma global, "esconde" a variável global.

```
1  a = 1
2  def imprime():
3   a = 5
4    print(a)
5
6  imprime()
7  # 5
8
9  print(a)
10  # 1
```



• Uma variável local com o mesmo nome de uma global, "esconde" a variável global.

```
1  a = 1
2  def incrementa():
3     a = a + 1
4     print(a)
5
6  incrementa()
7  # UnboundLocalError: local variable 'a' referenced before assignment
```



• Uma variável local com o mesmo nome de uma global, "esconde" a variável global.

```
1  a = 1
2  def incrementa():
3     a = 12
4     a = a + 1
5     print(a)
6
7  incrementa()
8  # 13
9  print(a)
10  # 1
```



- Na medida do possível devemos evitar o uso de variáveis globais dentro de funções. que dificultam a compreensão, manutenção e reuso da função.
- Se uma informação externa for necessária, ela deve ser fornecida como argumento da função.



• Podemos definir argumentos que devem ser informados na chamada da função.

```
1 def imprime_mensagem(mensagem):
2    print(mensagem)
3
4 bomdia = "Bom dia"
5 imprime_mensagem(bomdia)
6 # Bom dia
```

 O escopo dos argumentos é o mesmo das variáveis criadas dentro da função (variáveis locais).



• Uma função pode receber qualquer tipo de dado como argumento.

```
def imprime_soma(x, y):
      print(x + y)
2
3
   imprime_soma(2, 3)
  imprime_soma("2", "3")
  # 23
8
  imprime soma(2, "3")
    TypeError: unsupported operand type(s) for +: 'int' and 'str'
```



Funcões

Podemos escolher atribuir explicitamente os valores aos argumentos (argumento = valor), mas estas atribuições devem ser as últimas a serem feitas.

```
def imprime_subtração(x, y):
       print(x - y)
2
   imprime subtração (1, 4)
   # -3
   imprime_subtração(1, y = 4)
   # -3
   imprime_subtração(y = 1, x = 4)
   imprime_subtração(y = 1, 4)
   # SyntaxError: positional argument follows keyword argument
10
```



Funcões

• Quando não informamos o número correto de argumentos, obtemos um erro.

```
def imprime_soma(x, y):
    print(x + y)

imprime_soma(1)

# TypeError: imprime_soma() missing 1 required positional argument
    : 'y'

imprime_soma(1, 2, 3)

# TypeError: imprime_soma() takes 2 positional arguments but 3
    were given
```



• Podemos informar valores padrões para alguns dos argumentos.

```
1 def imprime_soma(x, y = 0):
2    print(x + y)
```



Argumentos com valores padrões não precisam ser explicitamente passados na chamada da função.

```
imprime_soma(1)
2
   imprime_soma(1, 0)
  imprime_soma(1, 2)
```



• Os argumentos funcionam como atribuições. Quando passamos variáveis associadas a tipos simples, qualquer alteração no argumento não altera a variável original.



 Assim como no caso de atribuições, quando os argumentos são estruturas mutáveis, como listas e dicionários, estamos apenas dando um novo nome para a mesma estrutura.

```
1  def duplica_ultimo(lista): # lista = numeros
2     lista.append(lista[-1])
3     print(lista)
4
5     numeros = [1, 2, 3, 4]
6     duplica_ultimo(numeros)
7  # [1, 2, 3, 4, 4]
8     print(numeros)
9  # [1, 2, 3, 4, 4]
```



Assim como no caso de atribuições, se não queremos que a estrutura original seja modificada, podemos criar uma cópia da estrutura usando o método copy.

```
def duplica ultimo(lista): # lista = [1, 2, 3, 4]
      lista.append(lista[-1])
2
      print(lista)
3
4
  numeros = [1, 2, 3, 4]
  duplica_ultimo(numeros.copy())
  # [1, 2, 3, 4, 4]
  print(numeros)
  # [1, 2, 3, 4]
```



Funcões

Uma função pode retornar um valor. Para determinar o valor retornado usamos o comando return.

```
def mensagem():
      return "Mais uma função"
2
3
  x = mensagem()
  print(x, len(x))
  # Mais uma função 15
```



• Podemos usar tuplas para retornar múltiplos valores.

```
def soma_e_subtração(x, y):
      return (x + y, x - y)
3
  soma, subtração = soma_e_subtração(4, 1)
  print(soma, subtração)
  # 5 3
```



 Quando não utilizamos o comando return ou não informamos nenhum valor para o return a função retorna o valor None.

```
1  def soma(x, y):
2     z = x + y
3  def subtração(x, y):
4     z = x - y
5     return
6
7  resposta1 = soma(2, 3)
8  resposta2 = subtração(2, 3)
9  print(resposta1, resposta2)
10  # None None
```



- O comando return interrompe a execução da função, ou seja, nenhum outro comando da função é executado.
- Uma função pode ter múltiplos comandos return, mas apenas um deles será executado.

```
def retorna_soma(x, y):
      z = x + y
      return z
      print("Esta mensagem não será impressa")
5
  print(retorna_soma(2, 3))
    5
```



Função main

- Para manter o código bem organizado, podemos separar todo o programa em funções.
- Neste caso, a ultima linha do código contém uma chamada para a função principal (por convenção chamada de main).



Função main

• Como a chamada da função main fica no final do código, não precisamos nos preocupar com a ordem em que as outras funções são definidas.

```
def main():
        função1()
        função2()
 3
 4
    def função2():
        print("Execução da função 2")
 6
    def função1():
        print("Execução da função 1")
10
    main()
      Execução da função 1
      Execução da função 2
28342 #
```

• Em aulas anteriores, vimos como testar se um número é primo:

```
n = int(input("Entre com um número inteiro positivo: "))
   primo = True
3
   for divisor in range(2, int(n**0.5)+1):
5
       if n % divisor == 0:
          primo = False
          break
8
   if primo:
       print("Primo")
10
   else:
11
       print("Composto")
12
```



• Vamos criar uma função que realiza este teste.

```
def testa_primo(n):
       primo = True
2
       for divisor in range(2, int(n**0.5)+1):
            if n \% divisor == 0:
5
                primo = False
6
                break
            return primo
8
   n = int(input("Entre com um número inteiro positivo: "))
10
   if testa_primo(n):
       print("Primo")
12
   else:
13
       print("Composto")
```

• Vamos criar uma função que realiza este teste.

```
def testa_primo(n):
       for divisor in range(2, int(n**0.5)+1):
2
            if n % divisor == 0:
                return False
4
5
       return True
6
   n = int(input("Entre com um número inteiro positivo: "))
8
   if testa_primo(n):
       print("Primo")
10
   else:
11
       print("Composto")
12
```



 Usando esta função vamos escrever um programa que imprima os n primeiros números primos.

```
1 def testa_primo(n):
2  # ...
3
4 n = int(input("Numero de primos a serem calculados: "))
5 candidato = 2
6
7 while n > 0:
8   if testa_primo(candidato):
9     print(candidato)
10     n = n - 1
11 candidato = candidato + 1
```



- As funções aumentam a clareza do código.
- Também tornam mais simples as modificações no código.
- Exemplo: melhorar o teste de primalidade.
 - Testar se o candidato é um número par.
 - Se for impar, testar apenas divisores impares (3, 5, 7, etc).
- O uso de funções facilita a manutenção do código.



Neste caso, basta alterar a função testa_primo.

```
1  def testa_primo(n):
2     if n % 2 == 0:
3         return n == 2
4     for divisor in range(3, int(n**0.5)+1, 2):
5         if n % divisor == 0:
6         return False
7     return True
```



 Vamos criar uma função que recebe um valor em segundos e imprime este valor em horas, minutos e segundos.

```
def converte_tempo_segundos(segundos_totais):
    horas = segundos_totais // 3600
    resto = segundos_totais % 3600
    minutos = resto // 60
    segundos = resto % 60
    print("{:02d}:{:02d}:{:02d}".format(horas, minutos, segundos))

converte_tempo_segundos(65135)
    # 18:05:35
```



• Se quisermos receber o tempo em minutos podemos usar a função anterior.

```
def converte_tempo_segundos(segundos_totais):
    # ...

def converte_tempo_minutos(minutos_totais):
    converte_tempo_segundos(minutos_totais * 60)

converte_tempo_minutos(539)

# 08:59:00
```



• O mesmo vale para receber o tempo em horas.

```
def converte_tempo_segundos(segundos_totais):
2
      #
3
  def converte_tempo_horas(horas_totais):
       converte_tempo_segundos(horas_totais * 3600)
5
6
  converte_tempo_horas(5)
    05:00:00
```

Exemplos de uso de funções 000000000000



• Podemos criar uma única função que recebe a unidade como argumento.

```
def converte_tempo(total, unidade):
       if unidade == "segundos":
2
            converte_tempo_segundos(total)
4
       elif unidade == "minutos":
            converte tempo segundos(total * 60)
       elif unidade == "horas":
6
            converte_tempo_segundos(total * 3600)
       else:
           print("Unidade inválida")
10
   converte_tempo(35135, "segundos")
11
   # 09:45:35
12
   converte_tempo(539, "minutos")
13
   # 08:59:00
```

• Podemos criar uma única função que recebe a unidade como argumento.

```
def converte_tempo(total, unidade = "segundos"):
       if unidade == "segundos":
2
            converte_tempo_segundos(total)
       elif unidade == "minutos":
4
            converte tempo segundos(total * 60)
       elif unidade == "horas":
6
            converte_tempo_segundos(total * 3600)
       else:
           print("Unidade inválida")
10
   converte_tempo(35135)
11
   # 09:45:35
12
   converte_tempo(539, "minutos")
13
   # 08:59:00
```

Dias, Horas, Minutos e Segundos

 Se quisermos agora imprimir o tempo em dias, basta modificar a função converte_tempo_segundos.

```
def converte_tempo_segundos(segundos_totais):
       dias = segundos totais // (3600 * 24)
2
       segundos do dia = segundos totais % (3600 * 24)
       horas = segundos_do_dia // 3600
4
       resto = segundos_do_dia % 3600
       minutos = resto // 60
       segundos = resto % 60
8
       print("{} dias, {} horas, {} minutos e {} segundos".
           format(dias, horas, minutos, segundos))
   def converte_tempo(total, unidade = "segundos"):
10
       # . . .
11
   converte tempo (1000000)
12
     11 dias, 13 horas, 46 minutos e 40 segundos
```

Exercício

Implemente uma função que, dada uma lista, retorne uma outra lista, com os elementos da lista original, sem repetições.



Exercício

```
def remove_duplicates(x):
      for i in x:
           if i not in a:
5
               a.append(i)
6
      return a
  print remove_duplicates([1,2,2,3,3,4])
```

