Raciocínio Computacional

UNIDADE 06

Funções

Esta unidade tem por objetivo apresentar o conceito de funções em Python. Funções são fragmentos de códigos que permitem organizar algoritmos e reduzir sua repetição. A partir do seu uso correto, um projeto fica mais fácil de manter, bem como torna-se mais legível, uma vez que funcionalidades específicas podem ser colocadas em poucas linhas.

Funções

Um código feito em Python pode muito bem ser feito de forma contínua, por meio do uso de comandos e estruturas básicas da linguagem em um único arquivo de código. Entretanto, esse tipo de abordagem não é recomendado, uma vez que torna o código mais difícil de ser corrigido quando alguma função não executa de forma adequada. Além disso, quando é preciso executar a mesma função várias vezes, ela deve ser reescrita, aumentando a quantidade de código produzida.

Para resolver as situações mencionadas, o Python, assim como outras linguagens de programação, permite a criação de funções. Uma função, como o nome sugere, modulariza uma funcionalidade específica, podendo receber parâmetros e retornar resultados. Seu uso deixa mais fácil a tarefa de identificar erros, bem como ela pode ser chamada várias vezes, evitando que o mesmo código seja escrito novamente.

Vimos todo esse processo na semana 2 brevemente; se não compreendeu o conteúdo lá, este é o momento de melhorar seu entendimento.

Os objetos de Python, como listas, tuplas e dicionários, usam funções para realizar suas tarefas. Por serem parte de objetos, essas funções recebem o nome de **métodos**, nomenclatura de programação orientada a objetos, como será visto na última unidade desta disciplina; por exemplo, o método ***clear()*** de listas apaga todo o seu conteúdo. É uma pequena funcionalidade, criada em uma pequena porção de código, chamada por um nome sugestivo.

De forma análoga, o Python permite aos programadores criar suas próprias funções para executar pequenas funcionalidades, assim como ocorre com os seus objetos internos. **A qualidade de um código é medida por sua capacidade de reaproveitamento e de modularização, características que são obtidas pela correta aplicação dos conceitos referentes a funções.**

Importante: função não é somente uma funcionalidade de Python, mas existe em toda linguagem moderna, devendo ser amplamente utilizada para melhorar a qualidade de seu código.

Funções básicas: parâmetros e retornos

Uma função deve executar uma funcionalidade específica, recebendo todas as informações necessárias para realizar uma tarefa e retornando, quando necessário, um resultado. A criação de uma função em Python é efetuada usando a palavra reservada ***def***. Segue a sintaxe geral de uma função:

```
def nomedafuncao(param1, param2, ...):
    return resultado
```

O nome de uma função deve ser sugestivo com relação à tarefa que realiza. Por exemplo, uma função que soma dois números e retorna o resultado da operação poderia ter o nome de **soma**, recebendo como parâmetros os dois números e retornando o resultado da operação. Essa função teria a seguinte estrutura:

```
def soma(num1, num2):
    s = num1 + num2
    return s
```

A mesma função seria chamada em um código Python da seguinte forma:

```
print(f“A soma de 2 e 3 = {soma(2, 3)}”)
```

Exemplo de aplicação 1: Elabore um programa que cadastre contatos de uma agenda telefônica. A função de cadastro deve ser realizada dentro de uma função chamada **inserir**, que recebe como parâmetros o nome e o telefone do contato, bem como a agenda de contatos.

```
1. agenda_telefonica = {}
2.
3. def inserir(nome, telefone, agenda):
4.     agenda[nome] = telefone
5.
6. while True:
7.     nome = input("Digite o nome do contato: ")
8.     telefone = input("Digite o telefone do contato: ")
9.     inserir(nome, telefone, agenda_telefonica)
10.    if input("Gostaria de adicionar um novo contato? (s/n)
   ") == "n":
11.        break
12.
13. print(agenda_telefonica)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic01.py
Digite o nome do contato: Maria
Digite o telefone do contato: (41) 98765-4321
Gostaria de adicionar um novo contato? (s/n) s
Digite o nome do contato: João
Digite o telefone do contato: (11) 98877-6655
Gostaria de adicionar um novo contato? (s/n) n
{'Maria', '(41) 98765-4321'), ('João', '(11) 98877-6655'}


Process finished with exit code 0
```

Como pode ser visto no exemplo, a função **inserir()** não retorna valores, uma vez que sua função não exige isso. Entretanto, se ela for expandida para verificar se um nome já existe na lista de contatos e perguntar se deseja modificar o telefone cadastrado, poderia retornar *true* ou *false*, indicando que foi feita uma inclusão ou não na agenda. Segue como esse código ficaria.

Exemplo de aplicação 2: Elabore um programa que cadastre contatos de uma agenda telefônica. A função de cadastro deve ser realizada dentro de uma função chamada **inserir**, que recebe como parâmetros o nome e o telefone do contato, bem como a agenda de contatos. A função deve verificar se o contato já existe e, em caso positivo, perguntar se o telefone deve ser modificado, retornando *true* ou *false*, de acordo com a inclusão/modificação executada ou não na agenda.

```
1. agenda_telefonica = {}
2.
3. def inserir(nome, telefone, agenda):
4.     if nome in agenda:
5.         if input("Contato já cadastrado. Deseja alterar o
   telefone? (s/n) ") == "n":
6.             return False
7.     agenda[nome] = telefone
8.     return True
9.
10. while True:
11.     nome = input("Digite o nome do contato: ")
12.     telefone = input("Digite o telefone do contato: ")
```

```
13.     if inserir(nome, telefone, agenda_telefonica):
14.         print("Contato adicionado ou atualizado com
    sucesso!")
15.     else:
16.         print("Falha ao tentar adicionar o contato!")
17.     if input("Gostaria de adicionar um novo contato? (s/n)
    ") == "n":
18.         break
19.
20. print(agenda_telefonica)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic02.py
Digite o nome do contato: Maria
Digite o telefone do contato: (41) 98765-4321
Contato adicionado ou atualizado com sucesso!
Gostaria de adicionar um novo contato? (s/n) s
Digite o nome do contato: Maria
Digite o telefone do contato: (11) 98877-6655
Contato já cadastrado. Deseja alterar o telefone? (s/n) n
Falha ao tentar adicionar o contato!
Gostaria de adicionar um novo contato? (s/n) n
{'Maria': '(41) 98765-4321'}


Process finished with exit code 0
```

Nos exemplos de aplicação 1 e 2, a agenda foi passada como parâmetro para a função. Em ambos os casos, a função teria acesso à variável **agenda_telefonica**, declarada na linha 01, não sendo necessário, da forma como o código está estruturado, passá-la como parâmetro. Entretanto, uma função só deve interagir com objetos e variáveis que se encontram em seu escopo, ou seja, os parâmetros passados ou variáveis criadas em seu corpo. No caso de essa função ser reaproveitada em outros locais, ela não deve ter dependências externas ao seu escopo.

Escopo de função

Escopo é um conceito fundamental das linguagens de programação que deve ser bem entendido, a fim de não causar confusões durante o desenvolvimento. Como explicado anteriormente, uma variável declarada fora da função pode ser vista por todas as funções. Isso ocorre porque essa variável tem escopo público, podendo ser acessada em qualquer local do arquivo ***.py*** no qual foi criada.

Uma função, por sua vez, embora possa interagir com variáveis públicas, não deve fazê-lo, uma vez que isso impede que seja, por exemplo, copiada e usada em outro arquivo. Dessa forma, uma função deve usar apenas os dados que recebe como parâmetros de fora do seu escopo, evitando que a variável externa seja acessada diretamente.

> E qual é o escopo de uma função? Como todo bloco em Python, seus limites começam a partir do símbolo de dois-pontos (:) e terminam quando o código deixa de estar indentado, voltando ao nível de arquivo.

Outra questão relacionada a escopo é o tempo de vida das variáveis criadas dentro da função. Uma variável criada em uma função tem seu tempo de vida limitado ao seu escopo. Além disso, ela tem precedência de acesso frente a uma variável externa ao escopo, mesmo tendo o mesmo nome. Segue um exemplo:

```
1. def minha_funcao():
2.     x = 10
3.     print(f"Valor de x dentro da função: {x}")
4.
5. x = 20
6. minha_funcao()
7. print(f"Valor de x fora da função: {x}")
```

Este exemplo gera a seguinte saída:

```
C:\


Valor de x dentro da função: 10
Valor de x fora da função: 20
```

Essa diferença ocorre pela precedência da variável criada com o mesmo nome dentro do escopo da função. No caso da remoção da linha 02, a variável pública **x** estaria visível dentro da função **minha_funcao()**, gerando o seguinte resultado:

```
C:\

Valor de x dentro da função: 20
Valor de x fora da função: 20
```

Parâmetros com valor-padrão

Funções em Python permitem que sejam passados parâmetros com valores-padrão, os quais podem ser omitidos no momento de chamar a função.

```
def nomedafuncao(param1 = valor_padrao, param2 = valor_padrao, ...):
```

Para exemplificar, a função **inserir()** dos exemplos anteriores poderia receber como valor-padrão para o telefone o valor “Sem telefone”.

```
3. def inserir(nome, agenda, telefone = “Sem telefone”):
```

Agora, imagine um sistema para uma pizzaria em que você efetua o pedido; na função que cadastra o pedido, é possível adicionar uma opção: pizza com borda ou sem borda. Caso o cliente não selecione a opção “com borda”, por padrão a pizza é cadastrada no sistema “sem borda”.

```
1. def fazerPedidoPizza(sabor, borda=”sem borda”):
```

É importante destacar que um ou mais parâmetros que possuam um valor-padrão devem ser deslocados para o final da lista de parâmetros na função, uma vez que, caso sejam omitidos, não haja confusão para identificar os demais parâmetros obrigatórios.

Parâmetros arbitrários

Uma função que possa receber um número variável de parâmetros precisa declarar um parâmetro arbitrário. Segue a sintaxe de função com declaração de parâmetro arbitrário:

```
def nomedafuncao(*args):
    print(args[0])
    print(args[1])
    ...
```

Exemplo de aplicação 3: Elabore um programa que use uma função chamada **somar()**, que efetua a soma de uma quantidade aleatória de números informados, retornando o resultado da operação.

```
2. def somar(*numeros):
3.     soma = 0
4.     for i in range(len(numeros)):
5.         soma += numeros[i]
6.     return soma
7.
8. resultado = somar(2, 3, 4, 5, 8)
9. print(f"A soma dos números é {resultado}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic03.py
A soma dos números é 22


Process finished with exit code 0
```

Na mesma função do pedido de uma pizza, o cliente pode ter a opção de adicionar sabores, porém não sabemos quantos ingredientes serão incluídos, sendo possível adicionar um parâmetro arbitrário, que conseguirá inserir uma quantidade de ingredientes variável na pizza.

```
1. def fazerPedidoPizza(sabor, borda=”sem borda”,
   *adicionais):
```

Retornos múltiplos (tuplas)

Diferentemente de outras linguagens de programação, o Python permite que seja retornado mais de um valor a partir de uma função. Esse retorno pode ser capturado como uma tupla, sendo os valores acessados de forma individual a partir de cada índice.

Exemplo de aplicação 4: Elabore um aplicativo que faça uso de uma função que receba diversos valores numéricos e retorne o maior e o menor valor da lista.

```
1. def maior_menor(*numeros):
2.     maior = -1000000
3.     menor = 1000000
4.     for numero in numeros:
5.         if numero > maior:
6.             maior = numero
7.         if numero < menor:
8.             menor = numero
9.     return maior, menor
10.
11. resultado = maior_menor(7, 15, 3, 22, 1, 8)
12. print(f"O maior número é {resultado[0]} e o menor número é
    {resultado[1]}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic04.py
O maior número é 22 e o menor número é 1


Process finished with exit code 0
```

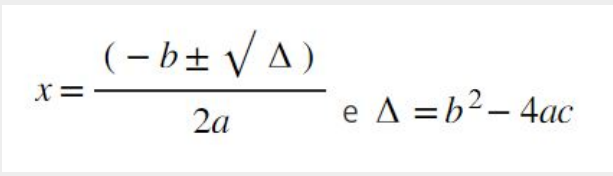
No exemplo do pedido de pizza, é possível retornar mais de um valor, como uma lista com todos os ingredientes da pizza e outro valor correspondente ao valor dela.

Encadeamento de funções

As funções podem ter chamadas encadeadas, de qualquer lugar, inclusive de dentro do corpo de outra função. Isso corrobora o conceito de modularizar os códigos o máximo possível, devendo uma função executar, normalmente, uma única tarefa.

Exemplo de aplicação 5: Elabore um programa que, aplicando a fórmula de Bhaskara por funções, encontre as raízes de um polinômio do segundo grau, a saber:

#ParaTodosVerem



$$x = \frac{(-b \pm \sqrt{\Delta})}{2a} \text{ e } \Delta = b^2 - 4ac$$

```
 1. def delta(a, b, c):
 2.     return b*b - 4*a*c
 3.
 4. def bhaskara(a, b, c):
 5.     d = delta(a, b, c)
 6.     if d < 0:
 7.         print("As raízes são imaginárias")
 8.         return 0, 0, False
 9.     else:
10.         x1 = (-b + d ** 0.5) / 2 * a
11.         x2 = (-b - d ** 0.5) / 2 * a
12.         return x1, x2, True
13.
14. result1 = bhaskara(1, 3, 1)
15. if result1[2]:
16.     print(f"As raízes são {result1[0]} e {result1[1]}")
17.
18. result2 = bhaskara(1, 2, 1)
19. if result2[2]:
20.     print(f"As raízes são {result2[0]} e {result2[1]}")
21.
22. result3 = bhaskara(1, 1, 1)
23. if result3[2]:
24.     print(f"As raízes são {result3[0]} e {result3[1]}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic05.py
As raízes são -0.3819660112501051 e -2.618033988749895
As raízes são -1.0 e -1.0
As raízes são imaginárias


Process finished with exit code 0
```

Na linha 01, é declarada a função **delta()**. Na linha 05, a função **bhaskara()** chama a função **delta()** para usar seu valor em seus cálculos internos. Dessa forma, outro local no código que precise usar o valor de delta pode chamar a mesma função, não precisando refazer esse cálculo.

No exemplo da pizzaria, podemos chamar uma função que calcula o valor da entrega dentro da função que cadastra o pedido.

Documentando funções

Documentar os códigos é uma boa prática de desenvolvimento. Especificamente no caso de funções, a documentação dentro de um padrão específico permite, com ferramental extra, a criação de documentação automática, gerada a partir das informações cadastradas. Da mesma forma, com o padrão correto, o PyCharm consegue ler internamente a descrição das funções, gerando uma ajuda que pode ser acessada internamente no projeto.

O desenvolvimento em Python é regido por alguns documentos de boas práticas, chamados *Python Enhancement Proposals* (PEPs). No caso da documentação, existe o padrão ***docstrings***, registrado pelo PEP 257 – *Docstring Conventions*.

Cabe destacar que, para os usuários do PyCharm, ambiente de desenvolvimento integrado (IDE) da empresa JetBrains, esse padrão de documentação pode ser gerado automaticamente. Para tanto, na primeira linha da função após ela estar completa,

devem ser digitadas três aspas seguidas para comentário de várias linhas e pressionar *Enter*. Segue como exemplo o padrão criado automaticamente no PyCharm sobre a função **delta()** do exemplo anterior:

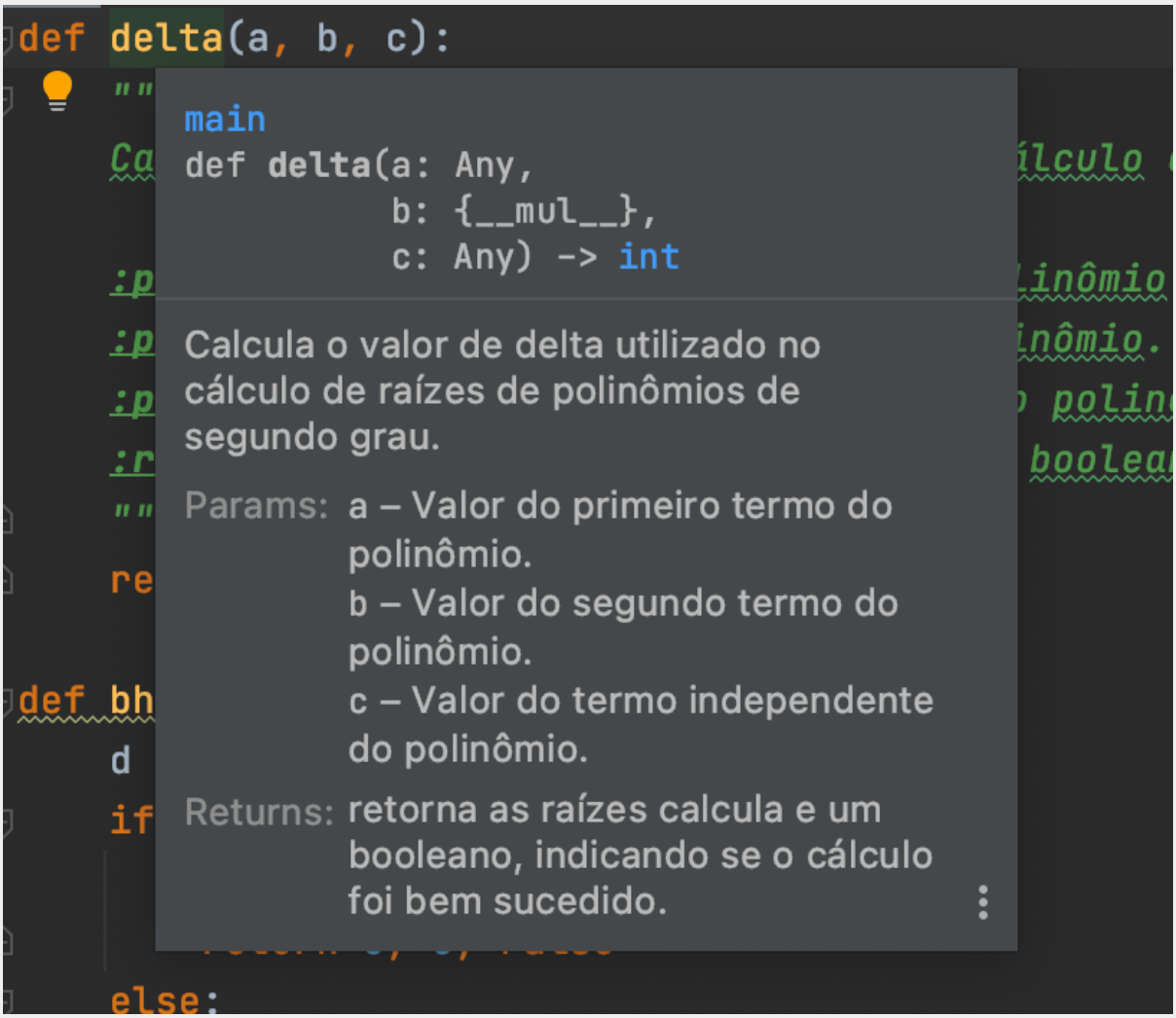
```
1. def delta(a, b, c):
2.     """
3.
4.     :param a:
5.     :param b:
6.     :param c:
7.     :return:
8.     """
9.     return b*b - 4*a*c
```

Considerando o padrão criado, a função pode ser documentada como segue:

```
1. def delta(a, b, c):
2.     """
3.     Calcula o valor de delta utilizado no cálculo de raízes
   de polinômios de segundo grau.
4.
5.     :param a: Valor do primeiro termo do polinômio.
6.     :param b: Valor do segundo termo do polinômio.
7.     :param c: Valor do termo independente do polinômio.
8.     :return: Retorna as raízes calculadas e um booleano,
   indicando se o cálculo foi bem-sucedido.
9.     """
10.    return b*b - 4*a*c
```

É importante destacar que a linha 04 está em branco propositalmente; essa separação se faz necessária para que a documentação exibida fique bem formatada. Posicionando o cursor sobre a função **delta** ou sua chamada ao longo do algoritmo e pressionando a tecla **F1**, será aberta a documentação criada.

#ParaTodosVerem

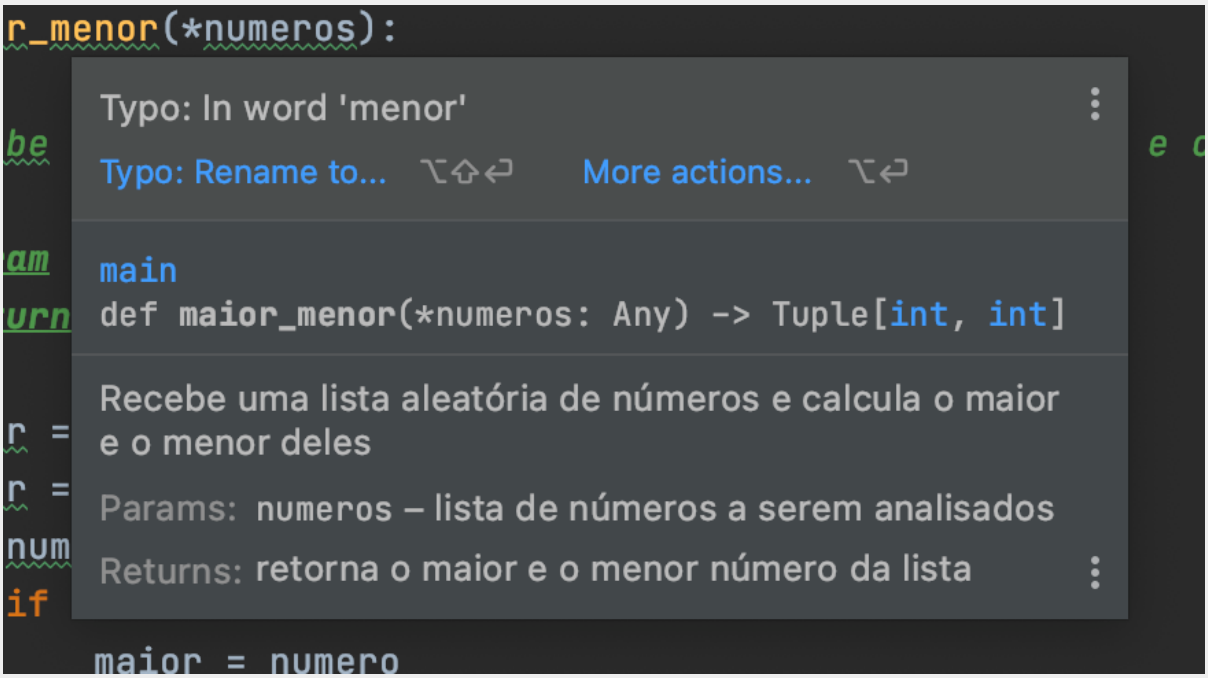


Exibição-padrão da documentação criada pelo padrão PEP 257 no PyCharm

Exemplo de aplicação 6: Tomando por base o exemplo de aplicação 4, elabore a documentação no padrão PEP 257 da função.

```
11.  def maior_menor(*numeros):
12.      """
13.       Recebe uma lista aleatória de números e calcula o
maior e o menor deles
14.
15.       :param numeros: lista de números a ser analisados
16.       :return: retorna o maior e o menor número da lista
17.       """
18.       maior = -1000000
19.       menor = 1000000
20.       for numero in numeros:
21.           if numero > maior:
22.               maior = numero
23.           if numero < menor:
24.               menor = numero
25.       return maior, menor
```

#ParaTodosVerem



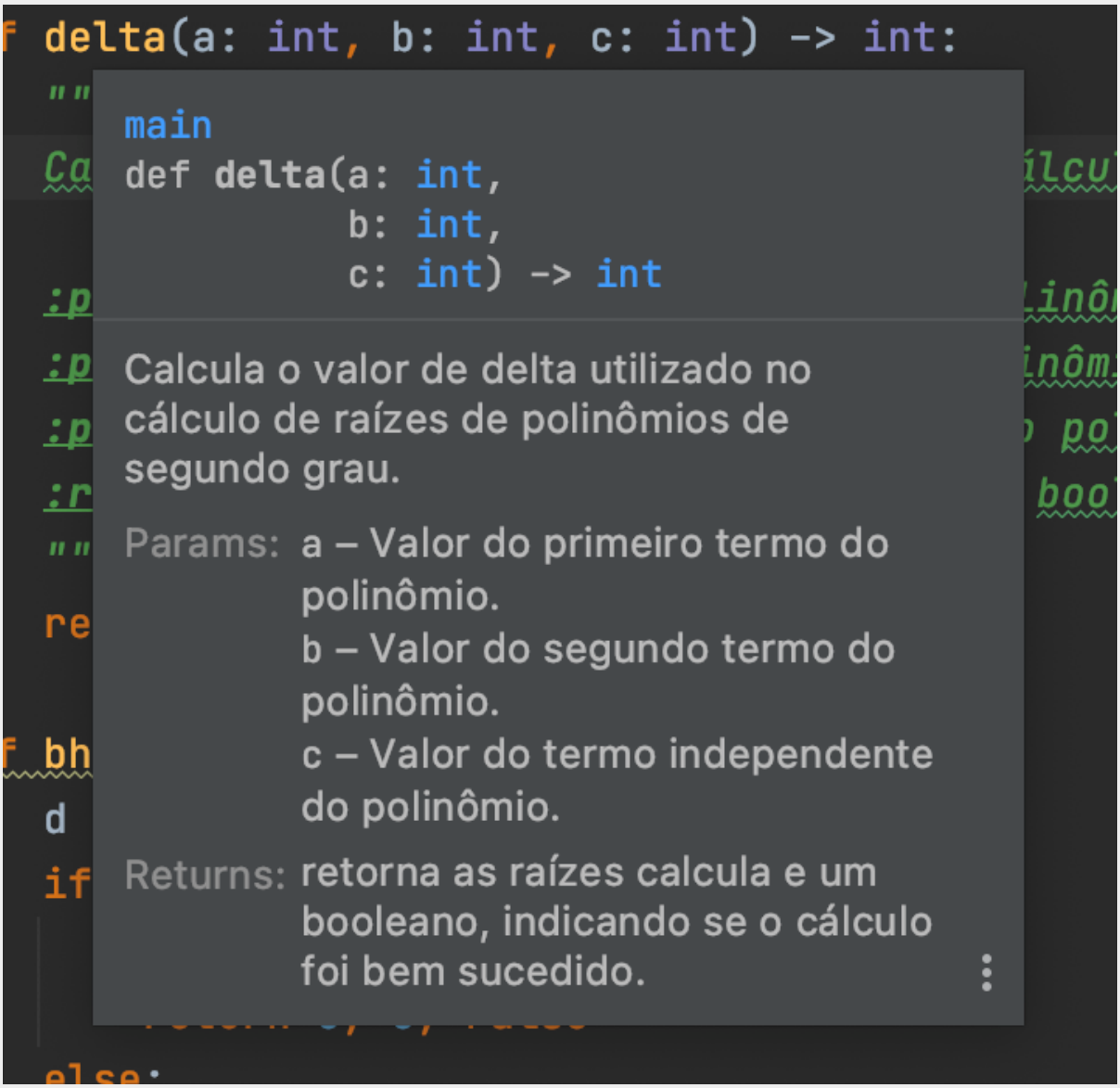
Exibição da documentação criada para a função **maior_menor()**

É possível perceber pelas documentações criadas em ambos os exemplos que o próprio Python identifica o tipo de dado de retorno e dos parâmetros, porém nem sempre ele fica correto para exibição. Assim, pode-se declarar explicitamente esses tipos de dado, para que na documentação fiquem corretos.

Voltando à função **delta()**, ela pode ser declarada como segue:

```
1. def delta(a: int, b: int, c: int) -> int:
2.     """
3.     Calcula o valor de delta utilizado no cálculo de raízes
   de polinômios de segundo grau.
4.
5.     :param a: Valor do primeiro termo do polinômio.
6.     :param b: Valor do segundo termo do polinômio.
7.     :param c: Valor do termo independente do polinômio.
8.     :return: Retorna as raízes calculadas e um booleano,
   indicando se o cálculo foi bem-sucedido.
9.     """
10.    return b*b - 4*a*c
```

#ParaTodosVerem



Exibição da documentação criada para a função **delta()** com declaração explícita de tipos

EXERCÍCIO

Funções

Exercício de fixação 1: Crie um programa que calcule, a partir de uma função, o fatorial de um número. Exemplo: Fatorial de 5 => 5! = 5.4.3.2.1. Observação: por propriedade, 0! = 1.

Resolução do exercício

Resolução exercício 1

```
def fatorial(numero):
    # segundo as propriedades de fatorial
    if numero == 0:
        return 1

    fat = 1
    for i in range(numero, 0, -1):
        fat *= i
    return fat

numero = int(input("Digite um número inteiro
para calcular seu fatorial: "))
fat = fatorial(numero)
print(f"O fatorial de {numero} é {fat}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix01.py
Digite um número inteiro para calcular seu
fatorial: 5
O fatorial de 5 é 120


Process finished with exit code 0
```

Exercício de fixação 2: Crie um programa que calcule o fatorial de um número, mas de forma recursiva, ou seja, chamando a função fatorial de dentro dela mesma.

 Resolução do exercício 

Resolução exercício 2

```
def fatorial(numero):
    # segundo as propriedades de fatorial
    if numero == 0:
        return 1

    return numero * fatorial(numero - 1)

```

```
numero = int(input("Digite um número inteiro para calcular seu fatorial: "))
fat = fatorial(numero)
print(f"O fatorial de {numero} é {fat}")
```

C:\

```
/Users/user/untitled1/bin/python /Users/user/projects/untitled1/ExFix02.py
Digite um número inteiro para calcular seu fatorial: 5
O fatorial de 5 é 120


Process finished with exit code 0
```

Exercício de fixação 3: Crie um programa que receba uma lista de números e retorne a média.

Resolução do exercício

Resolução exercício 3

#ParaTodosVerem

```
C:\

/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix03.py
O valor da média é 6.2


Process finished with exit code 0
```

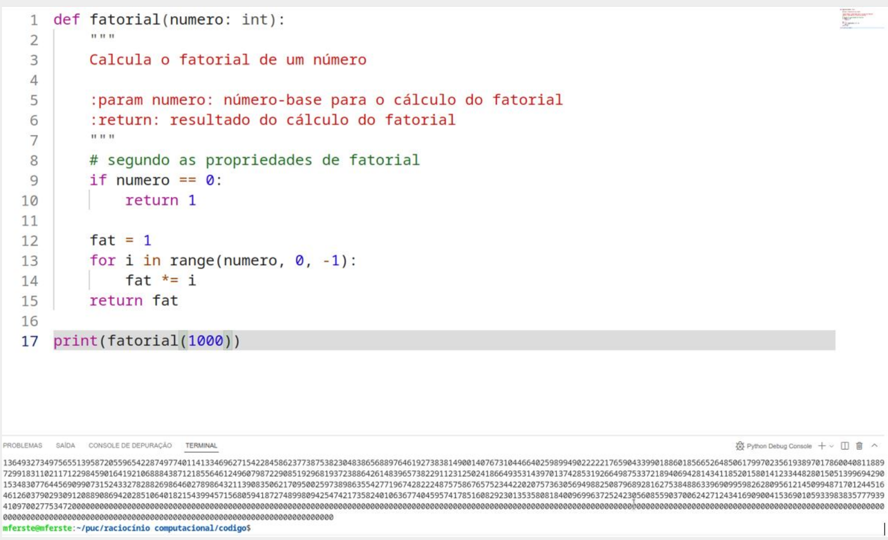
Documentação

Exercício de fixação 4: Documente a função **fatorial()** desenvolvida no exercício de fixação 1 desta unidade. Cuidado com o número para teste, pois pode travar sua máquina.

Resolução do exercício

Resolução exercício 4

#ParaTodosVerem



```python
def fatorial(numero: int):
    """
    Calcula o fatorial de um número

    :param numero: número-base para o cálculo do fatorial
    :return: resultado do cálculo do fatorial
    """
    # segundo as propriedades de fatorial
    if numero == 0:
        return 1

    fat = 1
    for i in range(numero, 0, -1):
        fat *= i
    return fat

print(fatorial(1000))
```

Exercício de fixação 5: Documente a função **media()** desenvolvida no exercício de fixação 3 desta unidade.

Resolução do exercício

Resolução exercício 5

```
def media(*numeros):
    """
    Calcula a média dos números passados para a função

    :param numeros: lista de números
    :return: valor da média calculada
    """
    soma = 0
    for numero in numeros:
        soma += numero
    return soma / len(numeros)
```

Desafio

Exercício de fixação 6: Crie um programa que, fazendo uso de funções, cadastre contatos em uma agenda telefônica, podendo exclui-los. Deve ser exibido um menu com as opções: inserir, remover e sair.

Resolução do exercício

Resolução exercício 6

```
def receber_dados_contato():
    nome = input("Digite o nome do contato: ")
    telefone = input("Digite o telefone do contato: ")
    return nome, telefone

def inserir(agenda):
```

```
    contato = receber_dados_contato()
    if contato[0] in agenda:
        if input("Contato já cadastrado.
Deseja alterar o telefone? (s/n) ") == "n":
            return False
    agenda[contato[0]] = contato[1]
    return True

def remover(nome, agenda):
    if nome in agenda:
        del agenda[nome]
        return True
    else:
        return False

def menu():
    print("*** Agenda Telefônica ***")
    print("1. Inserir contato")
    print("2. Remover contato")
    print("3. Sair")
    return int(input("Escolha uma das opções:
"))

agenda = {}
while True:
    op = menu()
    if op == 1:
        if inserir(agenda):
            print("Contato cadastrado com
sucesso")
            print(agenda)
        else:
            print("Operação não realizada")
    elif op == 2:
        nome = input("Digite o nome do
contato a ser removido: ")
        if remover(nome, agenda):
            print("Contato removido com
sucesso")
            print(agenda)
        else:
            print("Operação não realizada")
    else:
        break
```

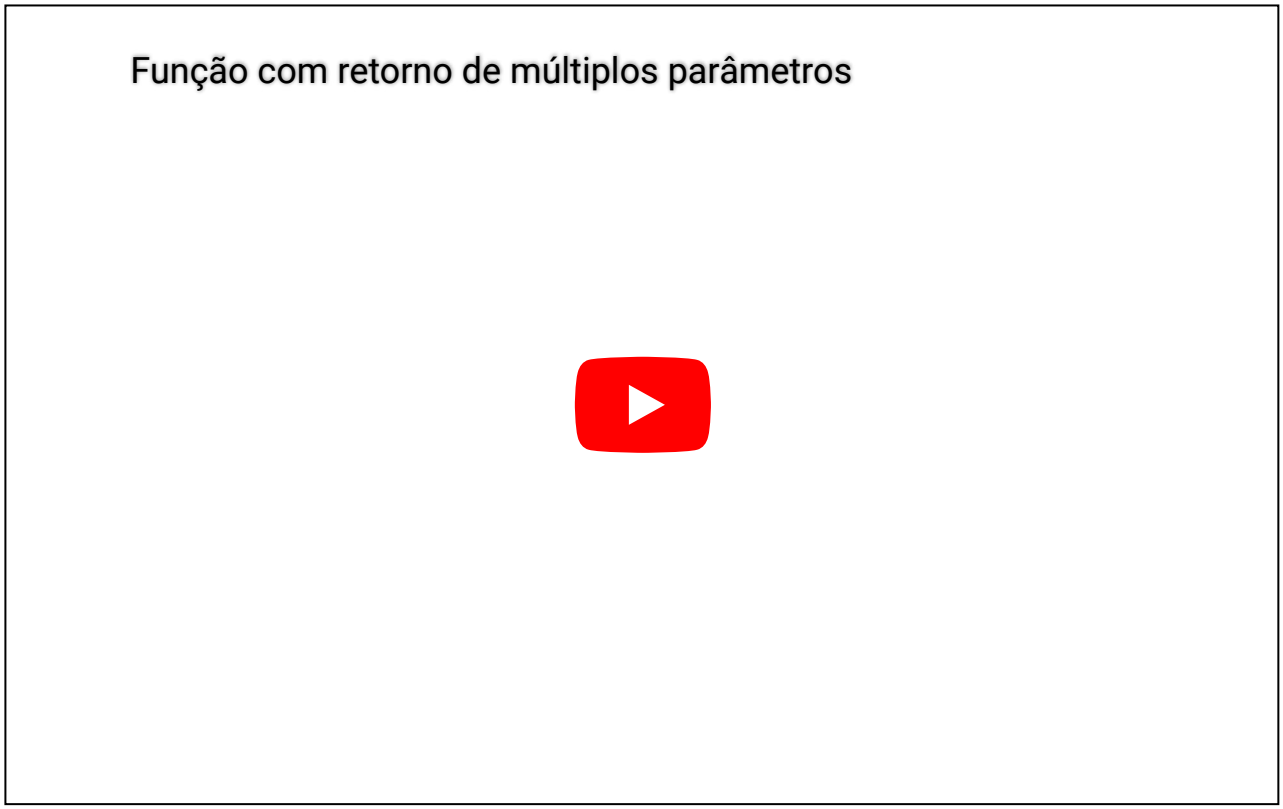
C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix03.py
*** Agenda Telefônica ***
1. Inserir contato
2. Remover contato
3. Sair
Escolha uma das opções: 1
Digite o nome do contato: Maria
Digite o telefone do contato: (41) 98765-
4321
Contato cadastrado com sucesso
{'Maria': '(41) 98765-4321'}
*** Agenda Telefônica ***
1. Inserir contato
2. Remover contato
3. Sair
Escolha uma das opções: 1
Digite o nome do contato: João
Digite o telefone do contato: (11) 98877-
6655
Contato cadastrado com sucesso
{'Maria': '(41) 98765-4321', 'João': '(11)
98877-6655'}
*** Agenda Telefônica ***
1. Inserir contato
2. Remover contato
3. Sair
Escolha uma das opções: 2
Digite o nome do contato a ser removido:
Maria
Contato removido com sucesso
{'João': '(11) 98877-6655'}
*** Agenda Telefônica ***
1. Inserir contato
2. Remover contato
3. Sair
Escolha uma das opções: 2
Digite o nome do contato a ser removido:
José
Operação não realizada
*** Agenda Telefônica ***
1. Inserir contato
2. Remover contato
3. Sair
Escolha uma das opções: 3


Process finished with exit code 0
```

Função com retorno de múltiplos parâmetros

Neste vídeo, veremos como criar funções em Python, com passagem de parâmetros de entrada para uma função, e como trabalhar com retorno de dados.



Conclusão

A respeito do uso de funções na programação em Python, podemos concluir que:

- As funções permitem modularizar os algoritmos.
- Além de organizar melhor os códigos, facilitam a sua manutenção.
- Funções permitem o reaproveitamento de código, reduzindo a quantidade de linhas escritas.
- As funções podem receber parâmetros e retornar valores.
- As funções possuem escopo próprio, que determina o tempo de vida de suas variáveis.
- Os parâmetros de uma função podem ter valores-padrão.
- Uma função pode receber uma lista de valores a partir de parâmetros arbitrários.
- Uma função pode retornar múltiplos valores no formato de uma tupla.

- A documentação de funções segue o padrão *docstrings*, orientação PEP 257.
- Documentar as funções permite o uso de ajuda rápida no PyCharm.
- As funções podem declarar explicitamente seus tipos de dado, para mostrar corretamente a ajuda rápida na documentação.

Referências

BANIN, S. L. **Python 3**: conceitos e aplicações uma abordagem didática. São Paulo: Érica, 2018.

PYTHON SOFTWARE FOUNDATION. **Documentação Python 3.9.0**. Disponível em: https://docs.python.org/pt-br/3/. Acesso em: 27 out. 2020.