Aula 9: Subprogramação

Luís Felipe

UFF

06 de Outubro de 2025

Luís Felipe

06/10/25

Funções em Programas

Um ponto chave na resolução de um problema complexo é conseguir "quebrá-lo" em subproblemas menores.

- Quebrar um problema em partes menores facilita o desenvolvimento.
- Cada parte pode ser implementada como uma função.
- Funções aumentam a legibilidade e reaproveitamento de código.
- Uma função agrupa comandos e pode retornar um valor.

Por que usar funções?

- Melhor organização do código.
- Evita repetições e facilita alterações.
- Permite reutilização em diferentes partes do programa.
- Facilita a depuração e manutenção do código.

Luís Felipe

06/10/25

Definindo uma função

```
1  def nome_funcao(param1, param2):
2      comandos...
3      return resultado
```

Os parâmetros são variáveis, que são inicializadas com valores indicados durante a invocação da função.
O comando return devolve para o invocador da função o resultado da execução desta.

- A execução da função começa ao ser chamada no programa.
- O valor retornado é passado para quem chamou a função.

Luís Felipe

06/10/25

Exemplo: soma de dois valores

A função que recebe como parâmetro dois valores inteiros, faz a soma destes valores, e devolve o resultado.

```
1  def soma(a, b):
2      c = a + b
3      return c
4
5  r = soma(12, 90)
6  print("r =", r)
7
8  r = soma(-9, 45)
9  print("r =", r)
```

Quando o comando return é executado, a função para de executar e retorna o valor indicado para quem fez a invocação (ou chamada) da função.

Luís Felipe

06/10/25

Mais exemplos de chamadas

```
1  def soma(a, b):
2      return a + b
3
4  def quadradoDaSoma(a, b):
5      return soma(a,b)**2
6
7  print("Soma:", soma(4, 2))
8  a = 2
9  c = 3
10 print("Quadrado da soma:", quadradoDaSoma(a, c))
```

Luís Felipe

06/10/25

Fluxo do programa

- Qualquer programa começa executando os comandos fora de qualquer função na ordem de sua ocorrência.
- Quando se encontra a chamada para uma função, o fluxo de execução passa para ela e se executa os comandos até que um return seja encontrado ou o fim da função seja alcançado.
- Depois disso, o fluxo de execução volta para o ponto onde a chamada da função ocorreu.

```
1  def soma(a, b):
2          c = a+b
3          return c
4
5  x1 = 4
6  x2 = -10
7
8  r = soma(5,6)
9  print(r)
10 r = soma(x1,x2)
11 print(r)
```

Luís Felipe

06/10/25

Definir funções antes do seu uso!

- Até o momento, definimos as funções antes do seu uso.
- O que aconteceria se declarássemos depois?

```
1  x1 = leNumero()
2  x2 = leNumero()
3  res = soma(x1, x2)
4  print("Soma é: ", res)
5
6  def soma(a,b):
7      c = a + b
8      return c
9
10 def leNumero():
11     c = int(input("Digite um número: "))
12     return c
```

- Ocorre um erro ao executarmos o programa

```
1  Traceback (most recent call last):
2    File "... .py", line 1, in <module>
3      x1 = leNumero()
4           ^^^^^^^^
5  NameError: name 'leNumero' is not defined
```

Luís Felipe

06/10/25

Função sem parâmetro

O que é e quando usar

- **Funções sem parâmetro** têm assinatura do tipo `def nome():` e não recebem valores de entrada.
- São úteis para encapsular tarefas que não dependem de dados variáveis na chamada (ex.: ler do teclado, mostrar um menu, inicializar algo).
- Podem **retornar um valor** (com return) ou apenas produzir um **efeito colateral** (ex.: imprimir na tela).
- Atenção: por não receberem dados, tendem a depender de **I/O** (input/output) ou variáveis globais; use com moderação para manter o código claro e testável.
- Se a tarefa precisar variar conforme a situação, transforme em função com parâmetros futuramente.

```
1  def leNumeroInt():
2      c = input("Digite um número inteiro: ")
3      return int(c)
4
5  r = leNumeroInt()
6  print("Número digitado:", r)
```

Luís Felipe

06/10/25

Funções que não retornam nada

- Faz sentido para uma função não retornar nada. Em particular, funções que apenas imprimem algo normalmente não precisam retornar nada.
- Há dois modos de criar funções que não retornam nada:
  - Não use o comando return na função.
  - Use o return None.
- None é um valor que representa o "nada".

```
1 def imprime(num):
2     print("Número: ", num)
```

Luís Felipe

06/10/25

Funções sem return ou com return None

```python
def imprimeCaixa(numero):
    tamanho=len(str(numero))
    for i in range(12+tamanho):
        print("+",end="")
    print()
    print("| Número:",numero,"|")
    for i in range(12+tamanho):
        print("+",end="")
    print()

imprimeCaixa(10)
imprimeCaixa(23456)

# Saída:
# +++++++++++++
# | Número: 10 |
# +++++++++++++
# +++++++++++++++++
# | Número: 23456 |
# +++++++++++++++++
```

```python
def imprimeCaixa(numero):
    tamanho=len(str(numero))
    for i in range(12+tamanho):
        print("+",end="")
    print()
    print("| Número:",numero,"|")
    for i in range(12+tamanho):
        print("+",end="")
    print()
    return None

imprimeCaixa(10)
imprimeCaixa(23456)

# Saída:
# +++++++++++++
# | Número: 10 |
# +++++++++++++
# +++++++++++++++++
# | Número: 23456 |
# +++++++++++++++++
```

Em ambos casos, a chamada da função é um comando por si só.

Luís Felipe

06/10/25

Função main()

- É comum criarmos uma função main() que executa os comandos iniciais do programa.
- O programa conterá várias funções (incluindo a main()) e um único comando no final do arquivo que é a chamada da função main().
- O programa será organizado da seguinte forma:

```
def main():
    Comandos Iniciais

def fun1(parâmetro):
    Comandos

def fun2(parâmetro):
    Comandos


...
...
main()
```

Exemplo:

```
def main():
    x1 = leNumero()
    x2 = leNumero()
    res = soma(x1, x2)
    print("Soma é: ", res)

def soma(a,b):
    c = a + b
    return c

def leNumero():
    c = int(input("Número: "))
    return c

main()
```

Luís Felipe

06/10/25

Funções com argumentos nomeados

- Até agora, na chamada de uma função era preciso colocar tantos argumentos quantos os parâmetros definidos para a função.
- Mas é possível definir uma função onde alguns parâmetros vão ter um valor default, e se não houver na invocação o argumento correspondente, este valor default é usado como valor do parâmetro.

```
1  def fx (a,b=9):
2  return a+b
3  >>> fx(3)
4  12
5  >>> fx(3,4)
6  7
```

- Os argumentos de uma função podem ser passados por nome em vez de por posição.
- Parâmetros com valor default costumam ser chamados por nome, isso não é obrigatório (outros parâmetros também podem ser nomeados).

```
1  def fx2(a,b=9,c=0):
2  return 100*a+10*b+c
3  >>> fx2(3)
4  390
5  >>> fx2(3,4,5)
6  345
7  >>> fx2(b=8,a=5,c=7)
8  587
```

Luís Felipe

06/10/25

A função `print`

- A função `print` tem dois parâmetros com valor default:
  - sep: define o separador entre os argumentos (default é `' '`).
  - end: define o que é impresso ao final (default é `'\n'`).
- Esses parâmetros devem ser passados nomeados.
- Exemplo:

```
1 print(3, 4, 5, end=' = ', sep=' + ')
```

- Saída: `3 + 4 + 5 =`
- O separador foi `' + '`, e o final foi `' = '`, sem quebra de linha.
- O `print` pode receber qualquer número de argumentos (não veremos isso neste curso).

Luís Felipe

06/10/25

Troca de elementos

- Faça um programa que receba uma lista de elementos e faça a troca de posições de um par arbitrário de elementos dessa lista.

```python
def trocar(lista, pos1, pos2):
    if 0 <= pos1 < len(lista) and 0 <= pos2 < len(lista):
        aux = lista[pos1]
        lista[pos1] = lista[pos2]
        lista[pos2] = aux
    return lista

def main():
    print("Digite as pessoas, separando por vírgulas: ")
    pessoas = input().split(", ")
    x = int(input("Diga qual a primeira posição: "))
    y = int(input("Diga qual a segunda posição: "))
    print("Lista de entrada era: ", pessoas)
    print(f"Lista trocando posições {x} com {y}: ", trocar(pessoas, x, y))
    return None

main()
```

```
Digite as pessoas, separando por vírgulas:
ana, lucio, joao, beatriz
Diga qual a primeira posição: 1
Diga qual a segunda posição: 2
Lista de entrada era:  ['ana', 'lucio', 'joao', 'beatriz']
Lista trocando posições 1 com 2 é:  ['ana', 'joao', 'lucio', 'beatriz']
```

Luís Felipe

06/10/25

Uniformização

- Uniformizar saída com nomes separados somente por vírgula:

```python
def trocar(lista, pos1, pos2):
    if 0 <= pos1 < len(lista) and 0 <= pos2 < len(lista):
        aux = lista[pos1]
        lista[pos1] = lista[pos2]
        lista[pos2] = aux
    return lista

def organizarLista(lista):
    for i in range(len(lista)-1):
        print(lista[i], end = ", ")
    print(lista[len(lista)-1])

def main():
    print("Digite as pessoas, separando por vírgulas: ")
    pessoas = input().split(", ")
    x = int(input("Diga qual a primeira posição: "))
    y = int(input("Diga qual a segunda posição: "))
    print("Lista de entrada era: ", end = "")
    organizarLista(pessoas)
    print(f"Lista trocando posições {x} com {y} é: ", end = "")
    organizarLista(trocar(pessoas, x, y))

main()
```

```
Digite as pessoas, separando por vírgulas:
ana, luis carlos, beatriz, maria, joao
Diga qual a primeira posição: 1
Diga qual a segunda posição: 2
Lista de entrada era: ana, luis carlos, beatriz, maria, joao
Lista trocando posições 1 com 2 é: ana, beatriz, luis carlos, maria, joao
```