MAC2166 - Introdução à Computação

Prof. Dr. Helder Oliveira

Agenda

- **Funções**
- **Uso de Funções**
- **Exercícios**

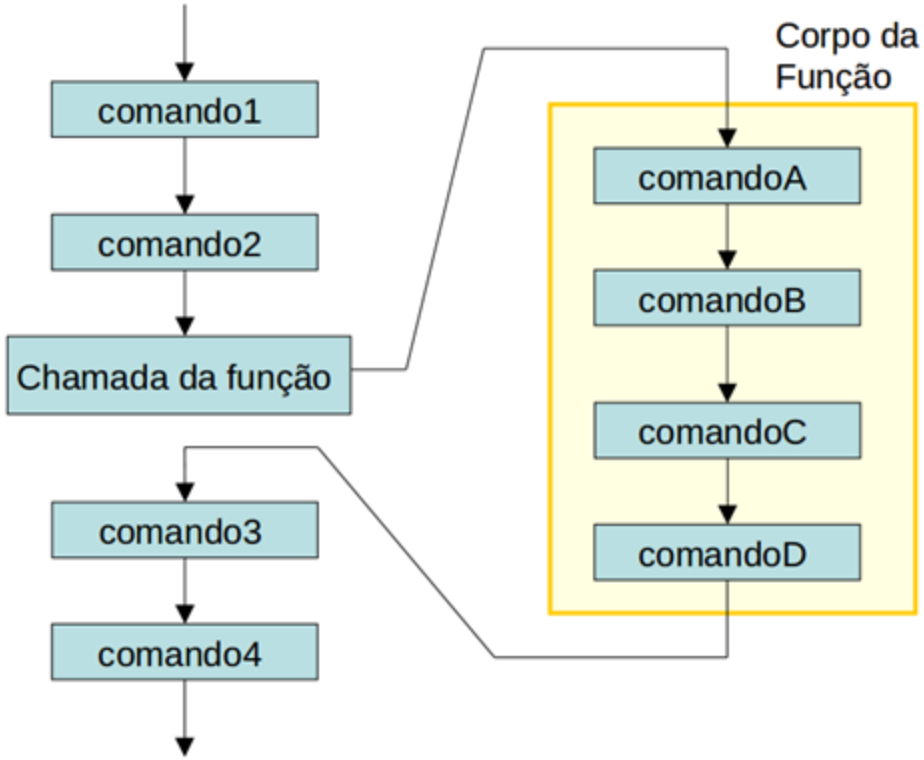
Funções

- 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.

Por que Utilizar Funções?

- Evitar que os blocos do programa fiquem 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.

Funções


comando1
comando2
Chamada da função
comando3
comando4
Corpo da Função
comandoA
comandoB
comandoC
comandoD

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

Definindo uma Função

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

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

Redefinindo uma Função

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

```
1  def imprime_mensagem():
2          print("Minha função")
3
4  def imprime_mensagem():
5          print("Minha função foi redefinida")
6
7  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.

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

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.

```
1 mensagem = "Variável global"
2 def imprime_mensagem():
3         print(mensagem)
4
5 imprime_mensagem() # Variável global
6 print(mensagem)
7 # Variável global
8
```

Escopo de uma Variável

- 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  print(a)
9   # 1
```

Escopo de uma variável

- 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
8 # before assignment
```

Escopo de uma Variável

- 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
```

Argumentos

- 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).

Argumentos

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

```
1  def imprime_soma(x, y):
2          print(x + y)
3
4  imprime_soma(2, 3)
5  # 5
6  imprime_soma("2", "3")
7  # 23
8  imprime_soma(2, "3")
9  # TypeError: unsupported operand type(s) for +:
10 # 'int' and     'str'
```

Argumentos

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

```
1  def imprime_subtração(x, y):
2          print(x - y)
3
4  imprime_subtração(1, 4)
5  # -3
6  imprime_subtração(1, y = 4)
7   # -3
8  imprime_subtração(y = 1, x = 4)
9  # 3
10 imprime_subtração(y = 1, 4)
   # SyntaxError: positional argument follows keyword argument
```

Argumentos

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

```
1  def imprime_soma(x, y):
2   print(x + y)
3
4  imprime_soma(1)
5  # TypeError: imprime_soma() missing 1 required positional
6  # argument: 'y'
7
8  imprime_soma(1, 2, 3)
9  # TypeError: imprime_soma() takes 2 positional arguments
10 # but 3 were given
```

Valor de Retorno

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

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

- Podemos usar tuplas para retornar múltiplos valores.

```
1 def soma_e_subtração(x, y):
2  return (x + y, x - y)
3
4 soma, subtração = soma_e_subtração(4, 1)
5 print(soma, subtração)
6 # 5 3
```

Valor de Retorno

- 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
```

A Função **main**

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

```
1 def main():
2  print("Execução da função main")
3
4 main()
5 # Execução da função main
```

A 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.

```
1  def main():
2   função1 () função2 ()
3
4  def função2 ():
5   print("Execução da função 2")
6
7  def função1 ():
8   print("Execução da função 1")
9
10 main()
11 # Execução da função 1
12 # Execução da função 2
13
```

Exercícios

- Crie uma função chamada soma que receba dois números como parâmetros e retorne a soma deles.

Exercícios

- Crie uma função chamada soma que receba dois números como parâmetros e retorne a soma deles.

```
1 def soma(a, b):
2     return a + b
3
4 # Teste
5 print(soma(5, 3))
6 # Saída: 8
```

Exercícios

- Crie uma função chamada eh_par que receba um número e retorne True se ele for par, e False caso contrário.

Exercícios

- Crie uma função chamada eh_par que receba um número e retorne True se ele for par, e False caso contrário.

```python
def eh_par(numero):
    return numero % 2 == 0

# Teste
print(eh_par(10))
# Saída: True
print(eh_par(7))
# Saída: False
```

Exercícios

- Crie uma função chamada saudacao que receba um nome e imprima uma saudação. Se nenhum nome for passado, ela deve dizer "Olá, visitante!".

Exercícios

- Crie uma função chamada saudacao que receba um nome e imprima uma saudação. Se nenhum nome for passado, ela deve dizer "Olá, visitante!".

```python
def saudacao(nome="visitante"):
    print(f"Olá, {nome}!")

# Testes
saudacao("Maria")
# Saída: Olá, Maria!
saudacao()
# Saída: Olá, visitante!
```

Exercícios

- Crie uma função chamada estatisticas que receba **três números inteiros** como argumentos. A função deve retornar:
  - A média entre os três números
  - O maior deles
  - O menor deles

Exercícios

- Crie uma função chamada estatisticas que receba **três números inteiros** como argumentos. A função deve retornar:
  - A média entre os três números
  - O maior deles
  - O menor deles

```python
def estatisticas(a, b, c):
    media = (a + b + c) / 3
    maior = max(a, b, c)
    menor = min(a, b, c)
    return media, maior, menor

# Teste
m, ma, mi = estatisticas(10, 20, 5)
print(f"Média: {m}, Maior: {ma}, Menor: {mi}")
# Saída: Média: 11.666..., Maior: 20, Menor: 5
```

Exercícios

- Escreva a **função** fatorial(n) que recebe um inteiro n e devolve o fatorial de n.

Exercícios

- Escreva a função fatorial(n) que recebe um inteiro n e devolve o fatorial de n.

```
1 def fatorial(n):
2     fat = 1
3     i = 2
4     while i <= n:
5         fat *= i
6         i += 1
7     return fat
```

Exercícios

- Escreva uma função que recebe dois inteiros n e k e calcula o coeficiente binomial, do número n, na classe k, isto é, o número de combinações de n termos, k a k, dado por n!/(k!(n-k)!). Escreva essa função usando a função fatorial(n) do item anterior.

Exercícios

- Escreva uma função que recebe dois inteiros n e k e calcula o coeficiente binomial, do número n, na classe k, isto é, o número de combinações de n termos, k a k, dado por n!/(k!(n-k)!). Escreva essa função usando a função fatorial(n) do item anterior.

```
1 def binomial(n,k):
2     return fatorial(n)//(fatorial(k)*fatorial(n-k))
```

Exercícios

- Faça um programa que lê um número inteiro n, com n > 0, e que imprime o Triângulo de Pascal com n linhas. Exemplo: para n = 5, o programa deve escrever:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Exercícios

- Faça um programa que lê um número inteiro n, com n > 0, e que imprime o Triângulo de Pascal com n linhas. Exemplo: para n = 5, o programa deve escrever:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

```python
def fatorial(n):
    fat = 1
    i = 2
    while i <= n:
        fat *= i
        i += 1
    return fat

def binomial(n,k):
    return fatorial(n)//(fatorial(k)*fatorial(n-k))

```

```python
def main():
    print("Triângulo de Pascal")
    n = int(input("Digite o número de linhas: "))
    i = 0
    while i < n:
        j = 0
        while j <= i:
            b = binomial(i,j)
            print("%3d "%b,end="")
            j+= 1
        print()
        i += 1

main()
```

Exercícios

- Faça um programa que lê um número inteiro n, com n > 0, e para cada número entre 1 e n indica se ele é soma de dois primos. Use funções!
- Por exemplo, para n = 8 o programa deve escrever

```
1 não é soma de primos
2 não é soma de primos
3 não é soma de primos
4 é soma dos primos 2 e 2
5 é soma dos primos 2 e 3
6 é soma dos primos 3 e 3
7 é soma dos primos 2 e 5
8 é soma dos primos 3 e 5
```

Exercícios

- Faça um programa que lê um número inteiro n, com n > 0, e para cada número entre 1 e n indica se ele é soma de dois primos.

```python
def primo(n):
    div = 2
    ehprimo = True
    while div < n:
        if n % div == 0:
            ehprimo = False
        div += 1
    if n <= 1:
        ehprimo = False
    return ehprimo

def main():
    n = int(input("Digite n: "))
    i = 1
    while i <= n:
        encontrou = False
        j = 2
        while j < i and not encontrou:
            if primo(j) and primo(i-j):
                encontrou = True
                p1 = j
                p2 = i-j
            j += 1
        if encontrou:
            print("%d é soma dos primos %d e %d"%(i,p1,p2))
        else:
            print("%d não é soma de primos"%i)
        i += 1

main()
```

Exercícios

- Em Python, existem várias funções matemáticas prontas do módulo math. Um exemplo de função desse módulo é a factorial, que calcula o fatorial de um número. Para usar uma função do módulo math, devemos primeiramente importar o módulo no código, usando o comando import math, e depois chamar a função.
- O exemplo abaixo mostra uma chamada à função factorial.

```
1  import math
2  n = int(input("Digite o valor de n: "))
3  fat = math.factorial(n)
4  print("Fatorial de",n,"é",fat)
```

Dúvidas