Funções - Parte 2

CIÊNCIA DA COMPUTAÇÃO

ATITUS EDUCAÇÃO

- Aula 14 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

ATITUS EDUCAÇÃO

Na aula de hoje...

01 Funções com Múltiplos retornos

02 Argumentos Nomeados

03 Passagem de Parâmetros

04 **Modularização**

Conceito

Definindo módulos personalizados

ALURA EDUCAÇÃO

Problema 4: Menu de Opções com retorno

Reescreva a função menu_de_opcoes, para que faça a leitura da opção informada. Se o valor digitado for válido, retorne o valor lido. Caso o valor seja inválido, retorne -1.

```
----------
   MENU
----------

  1 - Soma de dois valores reais
  2 - Divisores do número
  3 - Sequência de números pares
  4 - Verifica se o número é perfeito

Informe a opção desejada:
        ?
```

No programa principal, inclua a chamada desta função e escreva qual a opção foi escolhida (valor válido) ou uma mensagem de erro (valor inválido).

Problema 4: Menu de Opções com retorno

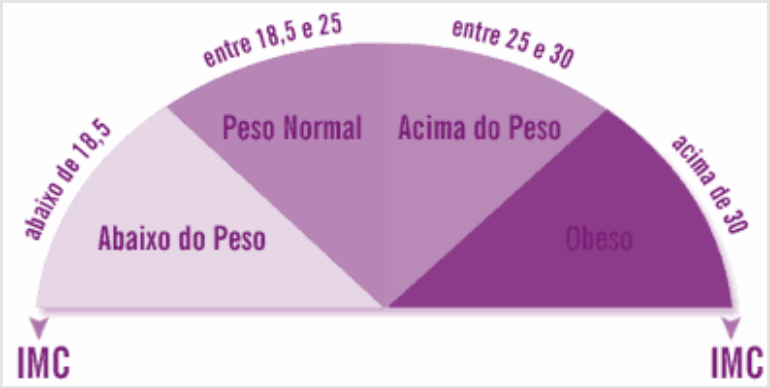
```python
def menu_de_opcoes():
    print("-" * 10)
    print("MENU".center(10))
    print("-" * 10)
    print()
    print("  1 - Soma de dois valores reais")
    print("  2 - Divisores do número")
    print("  3 - Sequência de números pares ")
    print("  4 - Verifica se o número é perfeito")

    print("\nInforme a opção desejada:")
    opcao = int(input("\t? "))
    if opcao >= 1 and opcao <= 4:
        return opcao
    else:
        return -1
```

```python
op = menu_de_opcoes()
if op != -1:
    print(f"Opção escolhida: {op}")
else:
    print("Opção Inválida!")
```

Problema 5: IMC

Faça um programa para calcular o IMC de uma pessoa, mostrando uma mensagem de acordo com a tabela abaixo.
Escreva uma função chamada `imc` que deve receber o peso e altura e que retorna o valor do IMC calculado.
Chame a função a partir do programa principal, após solicitar os dados do usuário e usando o resultado da função mostre a mensagem apropriada.

$$IMC = \frac{peso}{altura^2}$$



```python
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i

# Programa Principal
p = float(input("Digite seu peso em Kg: "))
a = float(input("Digite sua altura em m: "))

i = imc(p, a)

print(f"Seu IMC é {i:.2f} - Você está
if i < 18.5:
    print("abaixo do peso")
elif i <= 25:
    print("com peso normal")
elif i <= 30:
    print("acima do peso")
else:
    print("obeso")
```

A variável **i** tem seu valor atribuído no escopo da função e esse valor é retornado.

Na chamada, o retorno da função `imc` é atribuído para uma variável **i** definida no escopo do programa principal. Apesar do nome, **não é a mesma variável** do escopo da função. Falaremos de escopo de variáveis mais adiante.

```
Digite seu peso em Kg: 76
Digite sua altura em m: 1.8
Seu IMC é 23.46 - Você está com peso normal
```

Funções – Observações Gerais

Um argumento passado para uma função funciona da mesma forma que uma variável local à função

```
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i
```

Argumentos peso e altura agem como variáveis locais à função

Falaremos detalhadamente de escopo de variáveis na próxima aula!

Funções - Observações Gerais

Os nomes dos argumentos utilizados na declaração de uma função são independentes dos nomes das variáveis usadas para chamar a função

```python
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i

# Programa Principal
p = float(input("Digite seu peso em Kg: "))
a = float(input("Digite sua altura em m: "))

i = imc(p, a)

print(f"Seu IMC é {i:.2f} - Você está ", end="")
if i < 18.5:
    print("abaixo do peso")
elif i <= 25:
    print("com peso normal")
elif i <= 30:
    print("acima do peso")
else:
    print("obeso")
```

p, a, peso e altura independentes!

Funções - Observações Gerais

A quantidade de argumentos usados para chamar uma função deve ser igual a declaração

```
def imc(peso, altura):
    i = peso / (altura ** 2)
    return i


...
i = imc(p, a)
...
```

Chamamos esses argumentos de **posicionais**. Veremos depois que também existem argumentos **nomeados**.

Inserir argumentos a mais ou a menos gera erros do tipo TypeError.

Por exemplo:

```
...
i = imc(p, a , x)
...
```

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

Funções – Observações Gerais

* Qualquer tipo de dado pode ser um argumento de função
  → Inclusive listas, dicionários, etc.

* Qualquer expressão lógica ou aritmética válida pode ser argumento para uma função

* Funções podem chamar outras funções
  → Ao final da execução de uma função o fluxo de execução retorna à função que chamou

* Uma função pode chamar a ela mesma (recursividade)

Problema 6: Fibonacci

Construa uma função que imprima todos os números da sequência de fibonacci, retornando o último valor calculado (da posição que foi pedida).

```
Informe a posição: 6
DENTRO FUNÇÃO: fib(0) = 0
DENTRO FUNÇÃO: fib(1) = 1
DENTRO FUNÇÃO: fib(2) = 1
DENTRO FUNÇÃO: fib(3) = 2
DENTRO FUNÇÃO: fib(4) = 3
DENTRO FUNÇÃO: fib(5) = 5
DENTRO FUNÇÃO: fib(6) = 8
PROGRAMA PRINCIPAL: fib(6) = 8
```

Problema 6: Fibonacci

```python
def fibonacci(posicao):
    t_1 = 1
    t_2 = 0
    for i in range(posicao + 1):
        match i:
            case 0:
                fib = 0
            case 1:
                fib = 1
            case _:
                fib = t_2 + t_1
                t_2 = t_1
                t_1 = fib
        print(f"DENTRO FUNÇÃO: fib({i}) = {fib}")
    return fib
```

```python
pos = int(input("Informe a posição: "))
f = fibonacci(pos)
print(f"PROGRAMA PRINCIPAL: fib({pos}) = {f}")
```

Funções com Múltiplos Retornos

* Servem para retornar mais de um valor como resultado de uma função, os quais podem ser:
  - → Valor de uma variável dentro da respectiva função
  - → Valor fixo
  - → Resultado de um cálculo

Funções com Múltiplos Retornos

```
def nome(arg1, arg2, …, argN):
    ...
    return valor1, valor2, …, valorN
```

Declaração da função

Aqui os valores estão sendo retornados a partir de variáveis definidas no escopo da função.

```
# Programa principal
...
r1, r2, …, rN = nome(a1, a2, …, aN)
...
```

Chamada da função

Cada valor retornado pela função está sendo **atribuído a uma variável** no programa principal.

Problema 1: Soma e Produto

Faça uma função que receba dois valores e retorne a soma e a multiplicação de um pelo outro. No programa principal, faça o pedido para que o usuário entre com os dados e chame a função. Após a chamada da função imprima os valores também no programa principal.

Problema 1: Soma e Produto

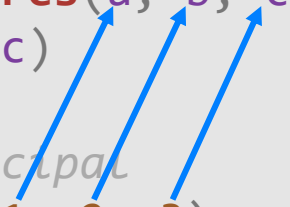
```python
def soma_mult(valor1, valor2):
    soma = valor1 + valor2
    produto = valor1 * valor2
    return soma, produto

## Programa Principal
v1 = float(input("Digite o primeiro valor: "))
v2 = float(input("Digite o segundo valor: "))

s, p = soma_mult(v1, v2)

print(f"Resultado da soma: {s:.2f}")
print(f"Resultado da produto: {p:.2f}")
```

Argumentos Nomeados

* Argumentos de funções vistos até agora são chamados de argumentos posicionais, onde cada valor é passado de acordo com sua posição na função

* Exemplo:

```
def imprime_valores(a, b, c):
    print(a, b, c)

## Programa Principal
imprime_valores(1, 2, 3)
```



```
1 2 3
```

Argumentos Nomeados

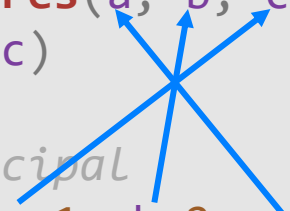
* Também podemos usar argumentos nomeados, onde cada valor é passado por uma associação de nome e valor

* Dessa forma, os argumentos podem ser passados em qualquer ordem para a função

* Utilizar o nome do argumento seguido de = e seu valor durante a chamada da função

* Exemplo:

```
def imprime_valores(a, b, c):
    print(a, b, c)

## Programa Principal
imprime_valores(c=1, b=2, a=3)
```



```
3 2 1
```

Argumentos Nomeados

* Também podemos misturar argumentos posicionais e nomeados, em uma mesma função
* Os argumentos posicionais devem vir primeiro que os argumentos nomeados
* Os argumentos nomeados se tornam “opcionais” porque têm um valor padrão atribuído na declaração
* Exemplo:

```
def imprime_valores(a, b, c, d=4):
    print(a, b, c, d)

## Programa Principal
imprime_valores(1, 2, 3)

imprime_valores(1, 2, 3, d=5)
```

```
1 2 3 4
1 2 3 5
```

Problema 2: Cadastro

Faça uma função que receba informações de uma pessoa: nome, idade e profissão e status (ativo ou inativo). A função deve apenas imprimir os dados na tela. No programa principal, solicite os dados ao usuário. Caso o status não seja informado a função deve ser chamada somente com os primeiros três argumentos.

Problema 2: Cadastro

```python
def cadastro(nome, idade, profissao, status = "ativo"):
    print(f"Nome     : {nome}")
    print(f"Idade    : {idade}")
    print(f"Profissão: {profissao}")
    print(f"Status   : {status}")

## Programa Principal
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))
prof = input("Digite sua profissão: ")
st = input("Digite o status (opcional): ")

if st: # testa se s é uma string não vazia
    cadastro(nome, idade, prof, st)
else:
    cadastro(nome, idade, prof)
```

Passagem de Parâmetros

* Considere os seguintes códigos:

```python
def funcao_valor(valor):
    print(f"F1: {valor}")
    valor = 2
    print(f"F2: {valor}")

# Programa Principal
v = 1
print(f"P1: {v}")
funcao_valor(v)
print(f"P2: {v}")
```

```
P1: 1
F1: 1
F2: 2
P2: 1
```

O valor de v é alterado fora da função apenas com o comando return.

Parâmetro imutável

```python
def funcao_lista(lista):
    print(f"F1: {lista}")
    lista[0] = 2
    print(f"F2: {lista}")

# Programa Principal
v = [1]
print(f"P1: {v}")
funcao_lista(v)
print(f"P2: {v}")
```

```
P1: [1]
F1: [1]
F2: [2]
P2: [2]
```

O valor de v é alterado fora da função mesmo sem o comando return.

Parâmetro mutável

Passagem de Parâmetros

* Dependendo do tipo de parâmetro que passamos em Python, ele pode ou não ser alterado na função

* Parâmetros em Python podem ser imutáveis ou mutáveis

* Em parâmetros imutáveis, uma cópia é criada dentro da função, mantendo a instância original sem alterações

  Tipos imutáveis: `int`, `float`, `bool`, `str`, ...

* Em parâmetros mutáveis, a instância original é alterada dentro da função, refletindo também fora da função

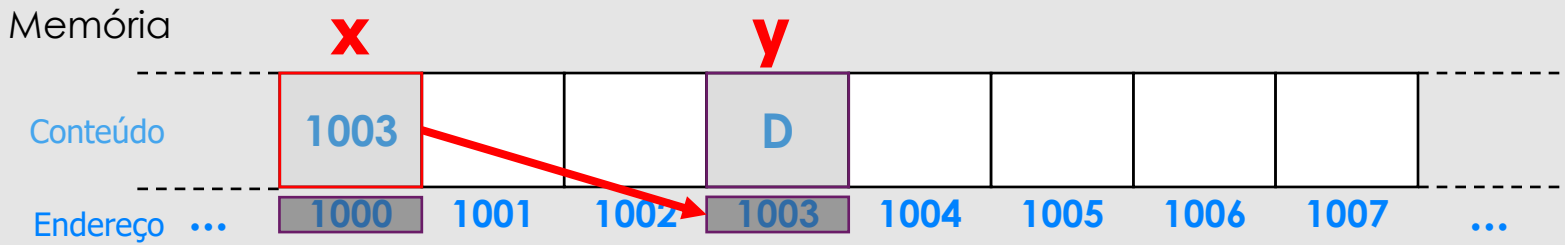
  Tipos mutáveis: `list`, `dict`, ...

Variáveis na memória

* Variáveis ocupam espaço na memória principal

* Em determinadas linguagens de programação (ex: linguagem C) é possível manipular diretamente endereços de memória utilizando ponteiros

* Ponteiros melhoram a performance do programa, mas perdem legibilidade



Variáveis na memória

* Em Python, na verdade, todos os tipos de dados são chamados de `PyObject`

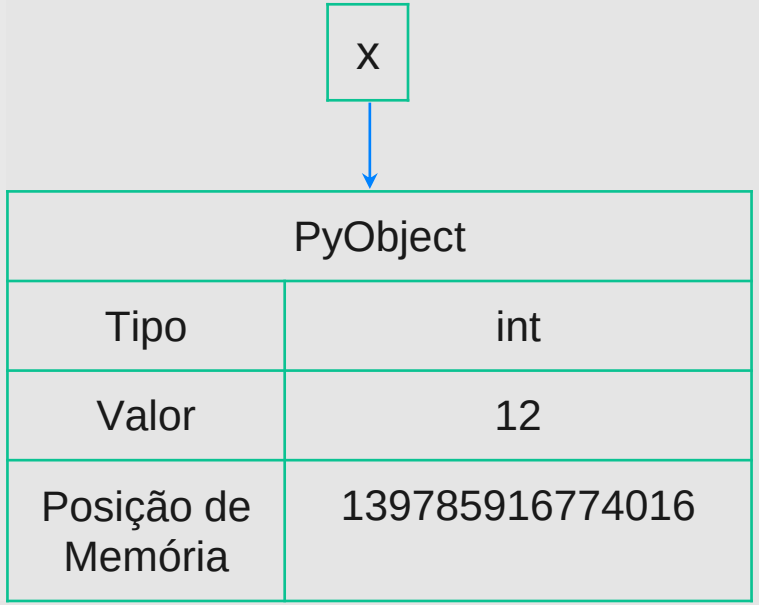
* Quando uma variável com um novo valor é criado, um novo `PyObject` é alocado para ela em determinada posição de memória

* (In)felizmente Python não permite que trabalhemos com ponteiros

* Entretanto, é possível descobrir qual o endereço de memória do PyObject que determinada variável está associada pelo comando `id(variável)`

```
x = 12
end_x = id(x)
print(end_x)
```

```
139785916774016
```



Módulos

* Um módulo consiste em um conjunto de funções definidas, que podem ser usadas posteriormente em outros programas

* Módulos podem ser criados por programadores ou podem estar pré-definidos pela linguagem

Módulos

* Para criar um módulo basta codificar um programa que contenha apenas as declarações das funções desejadas e salvá-lo com a extensão .py (como se fosse um código normal)

* Para chamar um módulo basta utilizar o comando `import` seguido do nome do módulo no início do código (certifique-se de que o módulo se encontra na mesma pasta do código principal)

* Para chamar as funções do módulo utilize `nomeMódulo.nomeFunção`

Módulos

* Ao desenvolver módulos, é importante ter em mente duas principais características:

  → Alta coesão: funções de um módulo devem ser fortemente relacionadas entre si
    - Ex: um módulo matemático deve ter apenas funções matemáticas

  → Baixo acoplamento: módulos devem ser compreendidos por si só, sem depender de outros módulos
    - Ex: um módulo matemático não deve depender do módulo math já existente

Problema 6: Combinação de n elementos p a p

Faça um programa que leia dois valores n e p, e calcule a combinação de n elementos, p a p. Assuma que $n \geq p$.

A fórmula para calcular a combinação é:

$$C_p^n = \frac{n!}{p! * (n-p)!}$$

Escreva duas funções, uma para o cálculo do fatorial, e outra para o cálculo da combinação.

Problema 6: Combinação de n elementos p a p

PLANEJAMENTO:

1. Fatorial de um valor
    a. Recebe um valor
    b. Calcula seu fatorial
    c. Retorna fatorial
2. Combinações de n elementos, p a p
    a. Recebe n e p
    b. Calcula n!/(p! * (n-p)!), chamando a função fatorial
    c. Retorna o valor calculado para quem chamou
3. Programa Principal
    a. Lê os valores n e p
    b. Calcula combinação de n elementos, p a p, utilizando a função combinações

```python
def fatorial(numero):
    fat = 0
    if numero > 0:
        fat = 1
        for i in range(numero, 1, -1):
            fat *= i
    return fat

def combinacoes(n, p):
    return fatorial(n) / (fatorial(p) * fatorial(n - p))

n = int(input("Informe o valor de n (número de elementos): "))
p = int(input("Informe o valor de p (qtde elementos por grupo): "))

comb = combinacoes(n, p)
print(f"Combinações de {n} valores {p} a {p} = {comb}")
```

Problema 6: Combinação de n elementos p a p

Modularização:

Podemos separar nossas funções em um módulo diferente da função main do programa.

Para utilizar as funções na função principal, devemos incluir o módulo através da diretiva `import`.

Arquivo: matemática.py

```
def fatorial(numero):
    fat = 0
    if numero > 0:
        fat = 1
        for i in range(numero, 1, -1):
            fat *= i
    return fat

def combinacoes(n, p):
    return fatorial(n) / (fatorial(p) * fatorial(n - p))
```

Programa Principal

```
import matematica

n = int(input("Informe o valor de n (número de elementos): "))
p = int(input("Informe o valor de p (qtde elementos por grupo): "))

comb = matematica.combinacoes(n, p)
print(f"Combinações de {n} valores {p} a {p} = {comb}")
```

Problema 7: Cálculo de e^x

Calcule o valor de e^x usando a série abaixo:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \cdots$$

onde x é um valor real, lido do teclado. Os termos devem ser inseridos enquanto forem maiores do que 0.0001 (em valor absoluto)

Funções envolvidas:

`potencia (float x, int n)` (não pode usar a função pow)
`fatorial (int n)` (já feita)
`EnaX (float x)`