



# Algoritmos e Programação de Computadores

## Funções

# Agenda

— — —

- Funções
  - Definindo uma função
  - Chamando uma função
- Declarações tardias de funções
- Parâmetros com valor *default* (padrão)

# Como escrever código?

- Até agora nós cobrimos alguns mecanismos da linguagem.
- Sabemos como escrever diferentes trechos de código para cada computação.
- Cada código é uma sequência de instruções.
- Problemas com esta abordagem?

# Problemas?

- Problemas com esta abordagem?
  - Fácil para problemas em pequena escala.
  - Complicado para problemas maiores.
  - Difícil de acompanhar os detalhes.
  - Como você sabe que a informação certa é fornecida à parte correta do código?

Faça um programa que leia  $n$  notas, mostre as notas e a média.

```
# Lê e mostra as n notas  
n = int(input())  
notas = []  
for i in range(n):  
    dado = float(input())  
    notas.append(dado)  
print(notas)
```

Essa parte lê as  $n$  notas e mostra na tela.

```
# Calcula a média  
soma = 0  
for i in range(len(notas)):  
    soma = soma + notas[i]  
media = soma/n  
print(format(media, ".1f"))
```

Essa parte calcula a média e mostra na tela.

# Introdução a Funções

- Um ponto chave na resolução de um problema complexo é conseguir “quebrá-lo” em subproblemas menores.
- Ao criarmos um programa para resolver um problema, é crítico quebrar um código grande em partes menores, fáceis de serem entendidas e administradas.
- Isto é conhecido como **modularização**, e é empregado em qualquer projeto envolvendo a construção de um sistema complexo.

# Definindo Funções

- Funções são estruturas que **agrupam um conjunto de comandos**, que são executados quando a função é chamada.
- As funções podem retornar um valor ao final de sua execução.

```
def quadrado (x) :  
    return x * x
```

# Por que definir uma função?

- Evitar que os blocos do programa fiquem grandes demais e, por consequência, mais difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.



# Definindo Funções

- Uma função é definida da seguinte forma:

```
def nome(parâmetro1, parâmetro2, ..., parâmetroN):  
    comandos  
    return valor do retorno
```

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

# Definindo Funções

```
def nome(parâmetro1, parâmetro2, ..., parâmetroN):  
    comandos  
    return valor do retorno
```

```
def quadrado(x):  
    return x * x
```

**quadrado** é o nome da função

**x** é o parâmetro

O resultado da função (**return**) é dado por  $x * x$

# Definindo Funções

```
def nome(parâmetro1, parâmetro2, ..., parâmetroN):  
    comandos  
    return valor do retorno
```

```
def quadrado(x):  
    return x * x
```

```
def quadrado(x):  
    valor = x * x  
    return valor
```

# Exemplo de uma função

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

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

```
r = soma(3, 5)  
print("A soma é :", r)  
  
print("A soma é :", soma(3, 5))
```

# Exemplo de uma função

- A lista de parâmetros de uma função pode ser vazia.



```
def leNumeroInt():  
    numero = input("Digite um número inteiro: ")  
    return int(numero)
```

```
n = leNumeroInt()  
print("Número digitado:", n)
```

# Exemplo de uma função

- A lista de parâmetros de uma função pode ser vazia.

```
def leNumeroInt():  
    numero = input("Digite um número inteiro: ")  
    return print("Número digitado:", numero)
```

```
n = leNumeroInt()
```

# Exemplo de uma função

- A expressão contida dentro do comando **return** é chamado de valor de retorno (é a resposta da função). **Nada após ele será executado.**

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado  
    print("Bla bla bla!")
```

```
s = soma(3, 5)  
print("A soma é :", s)
```

```
s = 8
```

# Exemplo de uma função

```
def leNumeroInt():  
    numero = input("Digite um número inteiro: ")  
    return int(numero)  
  
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado  
  
n1 = leNumeroInt()  
n2 = leNumeroInt()  
res = soma(n1, n2)  
print("A soma é:", res)
```



# Invocando uma função

- Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

```
s = soma(3, 5)  
print("A soma é :", s)
```

- Na verdade, o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:

```
print("A soma é :", soma(3, 5))
```

# Invocando uma função

- Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

```
s = soma(3, 5)  
print("A soma é :", s)
```

- Na verdade, o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:

```
a = 3  
b = 5  
print("A soma é :", soma(a, b))
```

# Funções que não retornam nada

- Faz sentido ter funções que não retornam 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”.

# Funções que não retornam nada

- Há dois modos de criar funções que não retornam nada:
  - Não use o comando `return` na função.

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

- Use o `return None`.

```
def imprime(num) :  
    print("Número: ", num)  
    return None
```

# Funções que não retornam nada

```
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(123456)
```

# Funções que não retornam nada

```
+++++++  
| Número: 10 |  
+++++++  
+++++++  
| Número: 123456 |  
+++++++
```

# Definindo funções depois do seu uso

- Até o momento, aprendemos que devemos definir as funções antes do seu uso. O que ocorreria se declarassémos depois?

```
n1 = leNumeroInt()
n2 = leNumeroInt()
res = soma(n1, n2)
print("A soma é:", res)

def leNumeroInt():
    numero = input("Digite um número inteiro: ")
    return int(numero)

def soma(numero1, numero2):
    resultado = numero1 + numero2
    return resultado
```

# Definindo funções depois do seu uso

- Até o momento, aprendemos que devemos definir as funções antes do seu uso. O que ocorreria se declarássemos depois?

```
n1 = leNumeroInt()
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-1-f562e295eb6d> in <module>()  
----> 1 n1 = leNumeroInt()  
      2 n2 = leNumeroInt()  
      3 res = soma(n1, n2)  
      4 print("A soma é:", res)  
      5  
  
NameError: name 'leNumeroInt' is not defined
```

```
return resultado
```



# Definindo funções depois do seu uso

- É comum criarmos um função **main()** que executa os comandos iniciais do programa.
- O seu programa conterà então várias funções (incluindo a **main()**) e um único comando no final do código que é a chamada da função **main()**.

# Definindo funções depois do seu uso

- O programa será organizado da seguinte forma:

```
def main() :  
    comandos  
  
def função1 (parâmetros) :  
    comandos  
  
def função2 (parâmetros) :  
    comandos  
...  
  
main()
```

# Definindo funções depois do seu uso

```
def main():  
    n1 = leNumeroInt()  
    n2 = leNumeroInt()  
    res = soma(n1, n2)  
    print("A soma é:", res)  
  
def leNumeroInt():  
    numero = input("Digite um número inteiro: ")  
    return int(numero)  
  
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado  
  
main()
```

Faça um programa que leia  $n$  notas, mostre as notas e a média.

```
# Lê e mostra as n notas
n = int(input())
notas = []
for i in range(n):
    dado = float(input())
    notas.append(dado)
print(notas)

# Calcula a média
soma = 0
for i in range(len(notas)):
    soma = soma + notas[i]
media = soma/n
print(format(media, ".1f"))
```

```
def main():
    n = int(input("Digite o número de notas: "))
    notas = leNota(n)
    print("As notas são:", notas)
    media = calculaMedia(notas)
    print("A média é:", format(media, ".1f"))

def leNota(num):
    notas = []
    for i in range(num):
        dado = float(input("Digite a nota: "))
        notas.append(dado)
    return notas

def calculaMedia(notas):
    soma = 0
    for i in range(len(notas)):
        soma = soma + notas[i]
    return (soma/len(notas))
```

```
main()
```

```
def main():  
    n = int(input("Digite o número de notas: "))  
    notas = leNota(n)  
    print("As notas são:", notas)  
    print("A média é:", format(calculaMedia(notas), ".1f"))
```

```
def leNota(num):  
    notas = []  
    for i in range(num):  
        dado = float(input("Digite a nota: "))  
        notas.append(dado)  
    return notas
```

```
def calculaMedia(notas):  
    soma = 0  
    for i in range(len(notas)):  
        soma = soma + notas[i]  
    return (soma/len(notas))
```

```
main()
```

# Definindo parâmetros com valor *default*

- 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* (padrão), e se não houver na invocação o argumento correspondente, este valor *default* é usado como valor do parâmetro.

```
def soma(numero1, numero2=5):  
    return numero1 + numero2
```

```
soma(3)  
soma(3, 10)
```

# Definindo parâmetros com valor *default*

- 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* (padrão), e se não houver na invocação o argumento correspondente, este valor *default* é usado como valor do parâmetro.

```
def soma(numero1, numero2=5):  
    return numero1 + numero2
```

```
soma(3)  
soma(3, 10)
```

```
8  
13
```



# Invocando funções com argumentos nomeados

- Os argumentos de uma função podem ser passados por nome em vez de por posição.

```
def soma(numero1, numero2=5):  
    return numero1 + numero2
```

```
soma(numero2=10, numero1=3)
```

# Funções com diferentes retornos

- Faça um programa, com uma função que necessite de um argumento. A função retorna 'P', se seu argumento for positivo, 'N', se seu argumento negativo, e 'Z' se seu argumento for zero.

```
def posOuNeg(numero):  
    if numero < 0:  
        return 'N'  
    elif numero > 0:  
        return 'P'  
    else:  
        return 'Z'
```

# Exercícios

# Exercícios

1. Faça uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -> 721.
2. Faça uma função que informe a quantidade de dígitos de um determinado número inteiro informado.
3. Faça uma função que computa a potência  $a^b$  para valores  $a$  e  $b$  (assuma números inteiros) passados por parâmetro (não use o operador `**`).

# Exercício 1: Reverso

Faça uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -> 721.

```
def reverso(n):  
    invert = str(n)  
    print(invert[::-1])
```

# Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 do Prof. Eduardo Xavier (IC/Unicamp)
- <https://wiki.python.org.br/ExerciciosFuncoes>
- <https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html>