Funções

INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Reflexão

- Um problema quando dividido é muito mais fácil de ser resolvido;
- Um sistema pode ser subdividido em problemas menores, vamos chamar de tarefas ;
- Cada tarefa quando bem definida, resolve um problema específico;
- Em programação, cada tarefa pode ser implementada como um subprograma ;
- Ao implementar uma tarefa o programador deverá buscar (ao máximo) deixar essa tarefa independente do programa ou subprograma que a utiliza.

Montando estratégia

- Ao pensar em **cada tarefa**, devemos analisar ...

O que a minha tarefa deve fazer ?	Objetivo da função/subprograma
Quais informações preciso ter para resolver a tarefa e não tenho ?	Parâmetros
A tarefa é para realizar um processamento ou descobrir uma informação ?	Retorno

Subprograma

- Peça chave da programação estruturada;
- É um fragmento do programa com **começo**, **meio** e **fim**, que desempenha um papel **bem definido** dentro de um programa maior;
- Esses sub-programas podem ser implementados separadamente e por diversos programadores de uma equipe.

Subprograma em Python

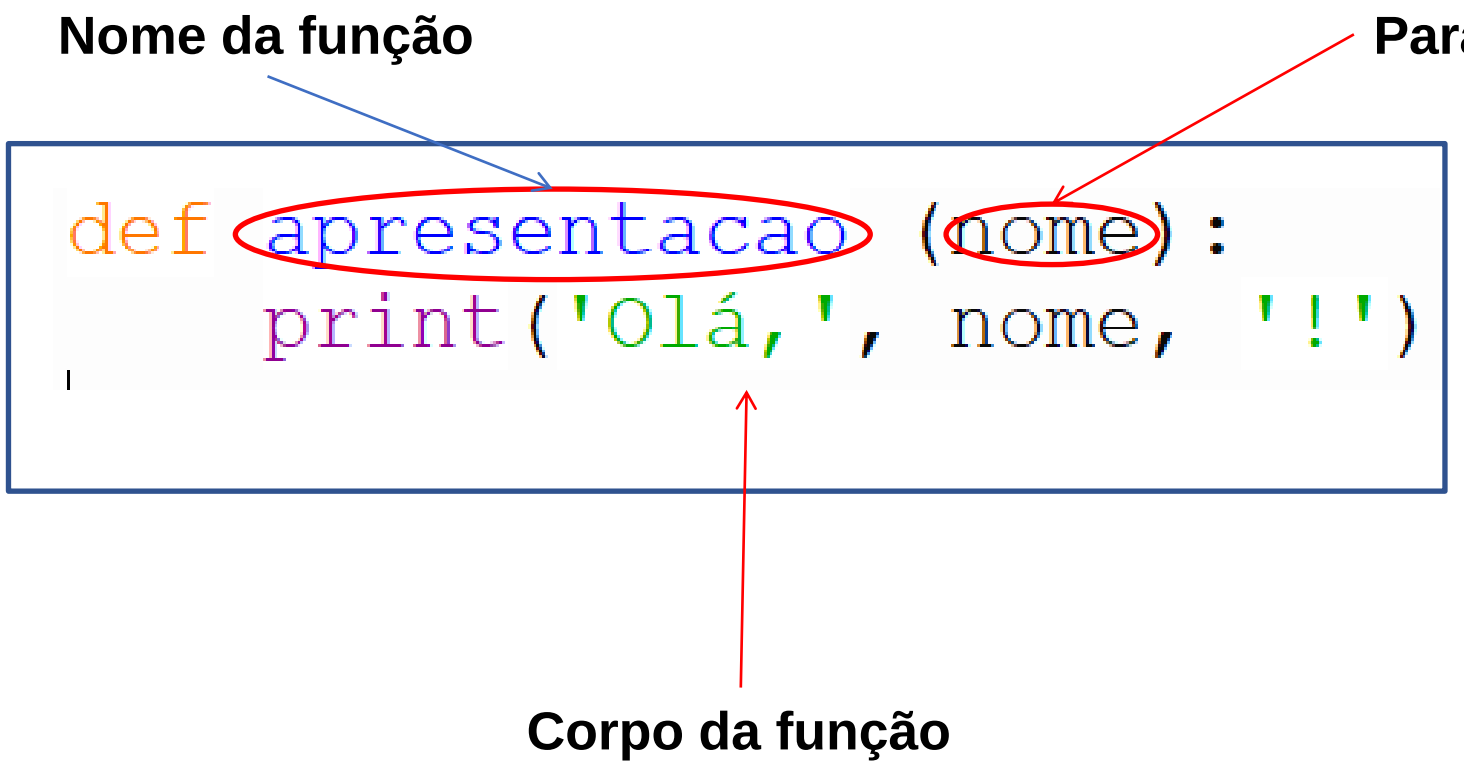
- Em Python, subprogramas têm o nome de funções
- **Formato geral:**

```
def nome (param_1, param_2, ... param_n):
    comando
    . . .
    comando
```

- **Onde**:
  - ***nome*** é o nome da função
  - ***param_1, param_2, ... param_n*** são parâmetros da função
    - Uma função pode ter 0, 1 ou mais parâmetros
  - ***comando*** refere-se a uma instrução a ser executadas quando a função é chamada

Criando funções em Python

- uma função é definida por três partes: **nome**, **parâmetros** e **corpo**

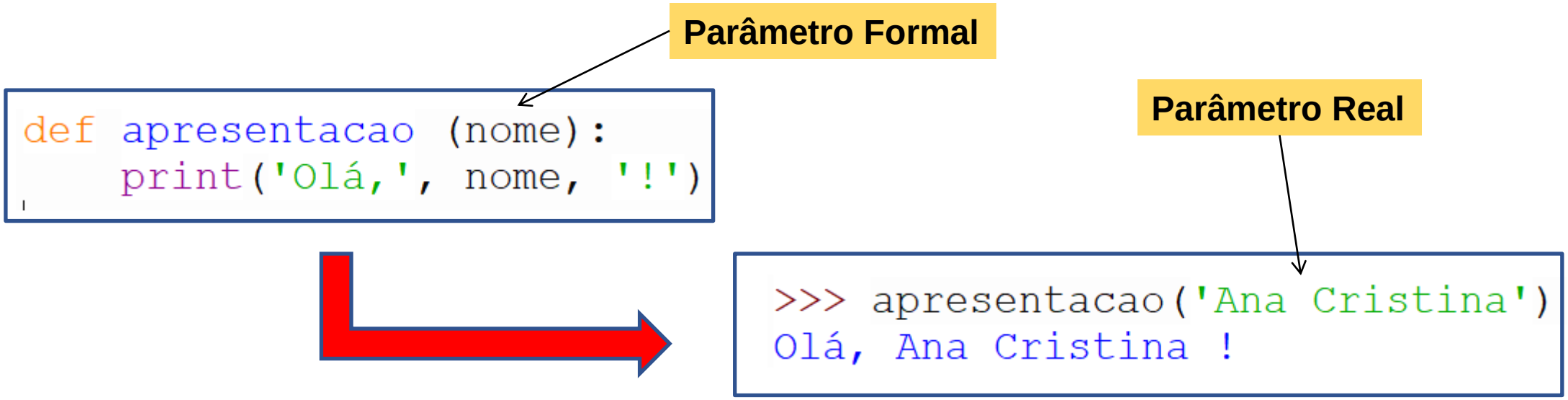


observação: os dois pontos (:) indicam que o código identado nas linhas abaixo faz parte da função

Parâmetros de Entrada

- Para executar a função, como em outras linguagens, devemos simplesmente **chamar seu nome** e **passar os parâmetros**, caso existam.

- Parâmetros são classificados como: **Formais** e **Reais**
  - **Formal** – quando usado na definição da função
  - **Real** – quando usado na chamada da função



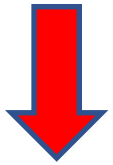
Funções com mais de um parâmetro ou sem parâmetros

- É possível definir funções com **mais de um parâmetro** ou **sem parâmetros** .

```
def apresentacao (nome, idade):
    print('Olá,', nome, '!')
    print('Eu tenho',idade,'anos')
```

```
>>> apresentacao('Ana Cristina',22)
Olá, Ana Cristina !
Eu tenho 22 anos
```

```
def bom_trabalho ():
    print('Olá,programador!')
    print('Bom trabalho')
```



```
>>> bom_trabalho()
Olá,programador!
Bom trabalho
```

Funções podem retornar valor

- Assim como podemos definir **parâmetros de entrada**, as funções também podem produzir **valores de saída**.

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

```
>>> soma(2,3)
5
```

Escopo de Variáveis (variável Local e variável global)

- O Escopo de uma variável está relacionada a definição do seu contexto em um programa ou subprograma
  - Esse contexto de definição por ser **local** ou **global**
- Parâmetros/argumentos e variáveis definidas em funções são **locais**,
  - só existem e só podem ser usadas no lugar onde foram definidas
  - Ao retornar ao ponto de chamada da função, as variáveis locais são descartadas
- Variáveis definidas fora das funções são conhecidas como variáveis **globais**
- É possível no código de uma função ler o conteúdo de uma variável global
  - Entretanto, para alterar uma variável global, ela precisa ser declarada no corpo da função usando o comando **global**

Escopo de Variáveis - Exemplificando

```
def soma(num1, num2):
    ''' adiciona dois números '''
    n1=100
    resultado=num1+num2+n1
    return resultado

n1=int(input('Digite um numero: '))
n2=int(input('Digite um numero: '))
print(soma(n1,n2))
print('n1=',n1)
```

n1 é uma variável local

```
Digite um numero: 2
Digite um numero: 3
105
n1= 2
```

Escopo de Variáveis - Exemplificando

```
def soma(num1, num2):
    ''' adiciona dois números '''
    global n1
    n1=100
    resultado=num1+num2+n1
    return resultado

n1=int(input('Digite um numero: '))
n2=int(input('Digite um numero: '))
print(soma(n1,n2))
print('n1=',n1)
```

Declarando com global

```
Digite um numero: 2
Digite um numero: 3
105
n1= 100
```

Fluxo de Execução

- Refere-se a ordem de execução dos comandos em um programa
- Chamadas de funções são como desvios do fluxo de execução.
- Compreender o fluxo de execução é importante quando trabalhamos com funções
  - Sabemos que, funções podem chamar outras funções
- Após a chamada de uma função e consequentemente a execução dos seus comandos, o fluxo de execução retorna ao local do programa onde a função foi chamada.
- Acompanhar um programa significa seguir o seu fluxo de execução.
  - É importante perceber as definições das funções, mas só considerá-las a partir de sua chamada.

Considerações sobre o desenvolvimento modularizado

- **Economia de código** (um trecho de código muito usado pode ser escrito uma única vez como um subprograma e usado diversas vezes);
- Desenvolvimento modularizado: (**projeto funcional**)
- Facilidade de **manutenção**;
- **Generalidade** de código com o uso de parâmetros
  - Os parâmetros servem como ponto de comunicação entre as funções (subprogramas).
  - Com o uso de parâmetros é possível definir funções genéricas (que não dependam das variáveis globais de um programa principal)

Funções em Python - considerações

- Se uma função **chega a seu fim sem nenhum valor de retorno** ter sido especificado, o **valor de retorno é None**
- Parâmetros em função podem ter **valor default**
  - def nome (param1=default1, ..., paramN=defaultN)