



Estácio

ARA0066 - PARADIGMAS DE LINGUAGENS DE PROGRAMAÇÃO EM PYTHON

PROFESSOR: LUCAS SAMPAIO LEITE

Correções dos exercícios de Listas

https://github.com/lucassampaioleite/python-exercises/blob/main/006_listas.ipynb

Agenda

Funções

Funções

- Uma função é uma **sequência nomeada de instruções que executa uma determinada operação**, sendo criada a partir de um nome e de instruções que a compõe, possibilitando ser utilizada posteriormente.



Funções

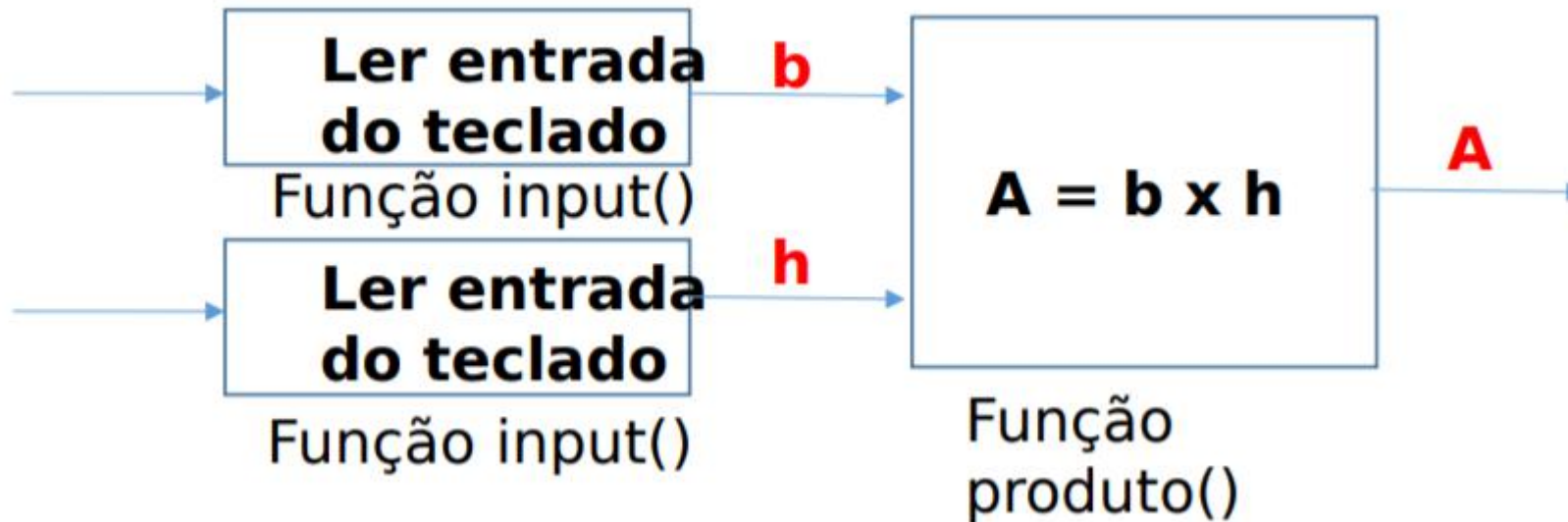
- ❑ Dividir para conquistar:
 - ❑ Abordagem comum para resolver um problema;
 - ❑ Dividir um problema em subproblemas menores;
 - ❑ Resolvemos cada subproblema;
 - ❑ Compomos as soluções de cada subproblema para chegar a uma solução do problema original.

Funções

- ❑ Na programação:
 - ❑ Dividimos a implementação de um algoritmo em funções;
 - ❑ Cada função resolve uma pequena parte do problema;
 - ❑ Uma mesma função pode ser usada para diversos outros problemas;
 - ❑ Ex: função multiplicação serve para calcular a área de um retângulo e de um triângulo;
 - ❑ Para resolver ambos os problemas precisamos calcular uma multiplicação.

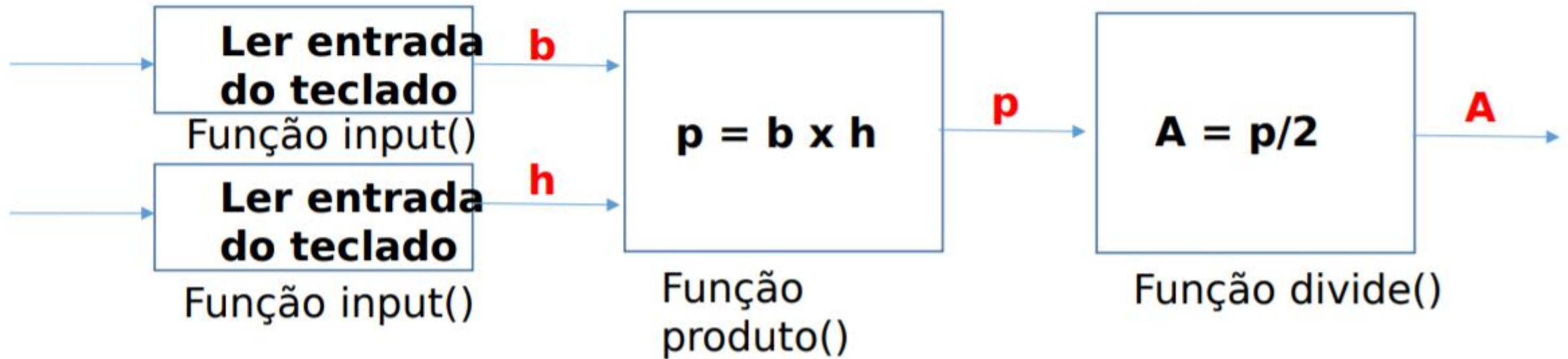
Funções

❑ Exemplo: cálculo área retângulo.



Funções

- Exemplo: cálculo área triângulo.



Funções

```
main.py > ...  
1  def identificador(parametros):  
2      #corpo da funcao  
3      #o corpo da funcao pode ter várias linhas  
4      #a indentação irá definir o corpo da funcao  
5      return #Uma função pode ter ou não retorno  
6  #este trecho está fora da função
```

- ❑ Def: define uma nova função.
- ❑ Identificador:
 - ❑ Serve para nomear a função;
 - ❑ É por meio do identificador da função que podemos invocá-la;
 - ❑ Ou seja, solicitar que ela seja executada para resolver parte de um problema.
- ❑ Parâmetros/argumentos (entre parênteses):
 - ❑ São valores de entrada usados na função;
 - ❑ Uma função pode ter 0 ou mais parâmetros.

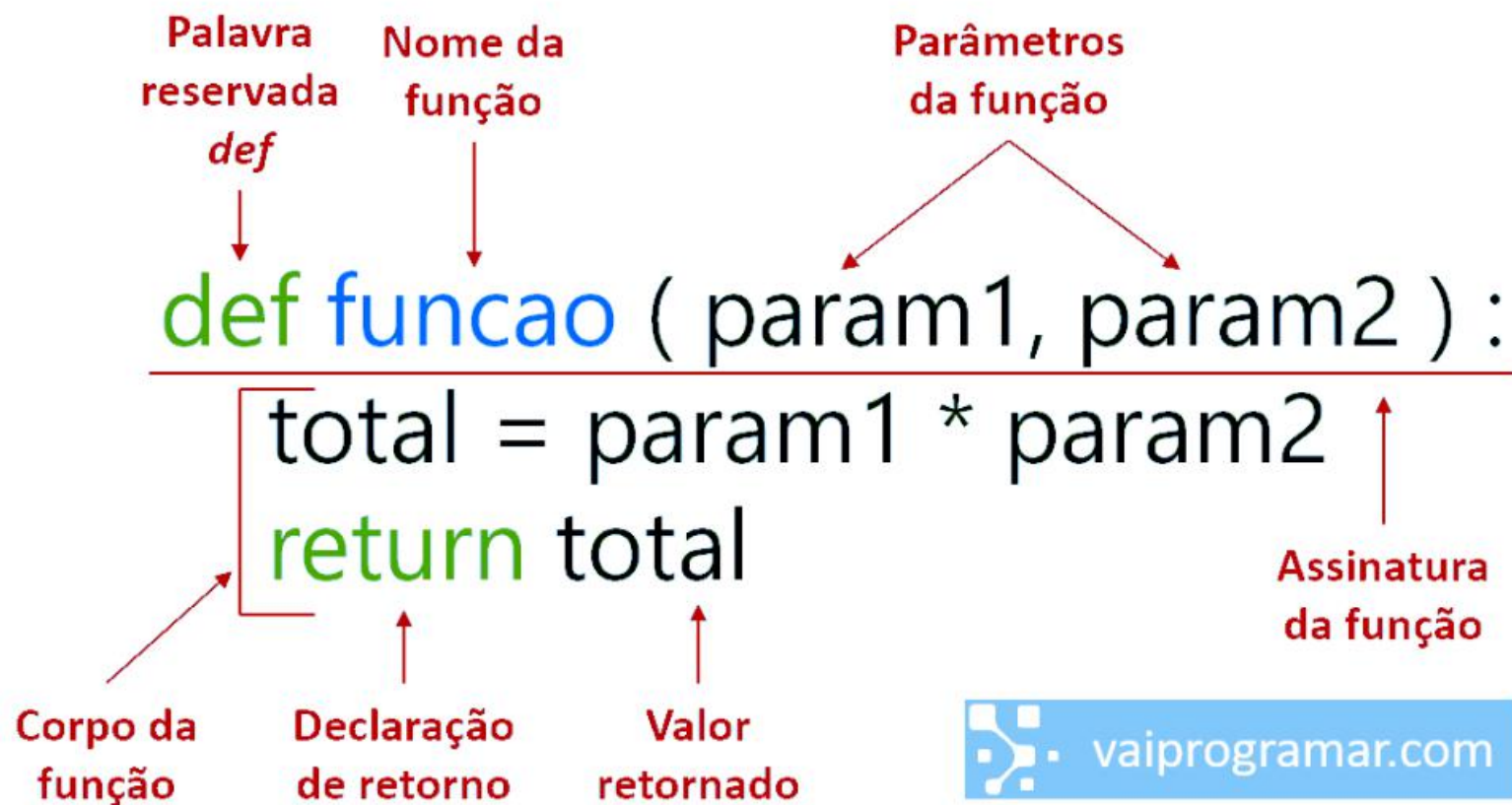
Funções

```
main.py > ...
1  def identificador(parametros):
2      #corpo da funcao
3      #o corpo da funcao pode ter várias linhas
4      #a indentação irá definir o corpo da funcao
5      return #Uma função pode ter ou não retorno
6      #este trecho está fora da função
```

- ❑ Def: define uma nova função.
- ❑ Corpo:
 - ❑ Espaço que pode conter uma ou mais linhas de código;
 - ❑ Dentro do corpo podemos ter um comando return.
- ❑ return:
 - ❑ Retorna o resultado da função;
 - ❑ Algumas funções não tem retorno ou não retornam nenhum valor;
 - ❑ return None;
 - ❑ return
 - ❑ Ou o return pode não aparecer

Funções

Como Declarar uma Função em Python



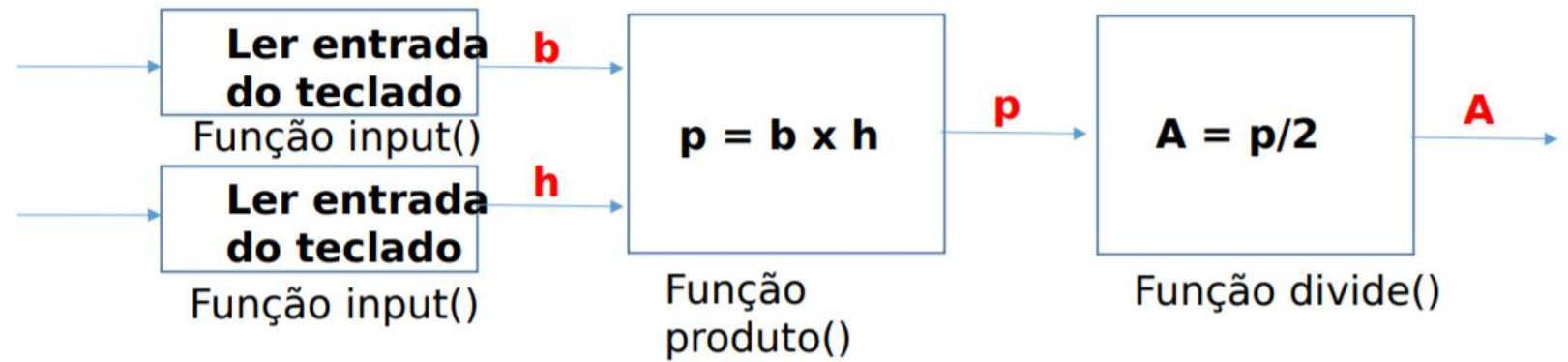
Funções

```
main.py > ...  
1  def nomeDaFuncao(parametro):  
2      """Esta é a forma de se declarar  
3      funções em Python."""  
4      print('Esta função recebeu ', parametro , ' como parâmetro. '  
5          '\nO tipo deste parâmetro , é: ', type(parametro))  
6      return '\nEstudamos funções no {}'.format(parametro)  
7  print(nomeDaFuncao('Curso de Férias'))
```



```
Esta função recebeu  Curso de Férias  como parâmetro.  
O tipo deste parâmetro , é:  <class 'str'>  
  
Estudamos funções no Curso de Férias.
```

Funções



```
main.py > ...
1  def produto (op1, op2):
2      res = op1*op2
3      return res
4
5  def divisao (op1, op2):
6      if op2 == 0:
7          return
8      res = op1/op2
9      return res
10
11  b=float(input('Digite o valor da base: '))
12  h=float(input('Digite o valor da altura: '))
13  p = produto(b,h)
14  area = divisao(p,2)
15  print ('Area do triângulo: ',area)
```


Funções

```
main.py > ...  
1  def calculoIMC(peso, altura):  
2      return peso/altura**2  
3  
4  print(calculoIMC(88,1.72))  
5
```



29.745808545159548

Funções

```
main.py > ...  
1  def calcularPagamento(qtd_horas, valor_hora):  
2      horas = float(qtd_horas)  
3      taxa = float(valor_hora)  
4      if horas <= 40:  
5          salario=horas*taxa  
6      else:  
7          hora_excedente = horas - 40  
8          salario = 40*taxa+(hora_excedente*(1.5*taxa))  
9      return salario  
10  
11 resultado = calcularPagamento(20,100)  
12 print(resultado)  
13 resultado = calcularPagamento(40,100)  
14 print(resultado)  
15 resultado = calcularPagamento(60,100)  
16 print(resultado)
```

Funções

- ❑ Escopo de Variável: Bloco de código em que uma variável pode ser acessada;
- ❑ Escopo local:
 - ❑ Uma variável criada dentro de uma função pertence ao escopo local dessa função;
 - ❑ Logo, só pode ser usada dentro dessa função.
- ❑ Escopo global:
 - ❑ Variável criada no corpo principal de um arquivo python;
 - ❑ Podem ser acessadas tanto no escopo global quanto local
 - ❑ Uma variável criada fora de uma função pode ser acessada dentro e fora de uma função.

Funções

❏ Exemplo de uso:


```
main.py > ...  
1   varGlobal = 'Sou uma variavel global \o/'  
2   def showMeTheCode():  
3       varLocal = 'Sou uma variável local o/'  
4       print (varLocal)  
5       print (varGlobal)  
6   showMeTheCode()  
7   print (varGlobal)  
8   print (varLocal)
```



```
Sou uma variável local o/  
Sou uma variavel global \o/  
Sou uma variavel global \o/  
Traceback (most recent call last):  
  File "c:\Users\Lucas Sampaio\Dropbox\IFPE\CursoFerias\project\main  
.py", line 8, in <module>  
    print (varLocal)  
NameError: name 'varLocal' is not defined. Did you mean: 'varGlobal'  
?
```

Exercícios

1. Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.
2. Faça um programa, com uma função que necessite de um argumento. A função retorna o valor de caractere 'P', se seu argumento for positivo, e 'N', se seu argumento for zero ou negativo.
3. Faça um programa para imprimir de acordo com a imagem abaixo para um n informado pelo usuário. Use uma função que receba um valor n inteiro e imprima até a n-ésima linha.



```
1
2  2
3  3  3
...
n  n  n  n  n  n  ... n
```

Exercícios

4. Reverso do número. Faça uma função que retorne o reverso de um número inteiro informado. Por exemplo: 127 -> 721.
5. Faça uma função que retorne a quantidade de dígitos de um determinado número inteiro informado pelo usuário. Não realize conversão de tipos.
6. Faça uma função que computa a potência de a^b para valores de a e b informados pelo usuário. Assuma valores de a e b como inteiros e não utilize o operador `**` ou funções da biblioteca Math.
7. Utilizando funções, leia um número inteiro e retorne o seu equivalente em numeração romana.

Subprogramas (fundamentos)

Segundo Sebesta (2019, p. 415):

“Um cabeçalho de subprograma, isto é, a primeira parte da definição, serve a diversos propósitos.”

- 1 Especifica a unidade sintática subsequente como uma definição de subprograma de algum tipo em particular. O tipo do subprograma é especificado com uma palavra especial;
- 2 O cabeçalho fornece um nome para o subprograma;
- 3 Pode especificar uma lista de parâmetros.

```
1 procedure Adder(parameters)
```

Exemplo em Ada

```
1 def adder(parameters):
```

Exemplo em Python

Subprogramas (fundamentos)

Python

```
1 def somar_vetor(vetor):  
2     soma = 0  
3     for elemento in vetor:  
4         soma += elemento  
5     return soma  
6  
7 vetor = [1, 2, 3, 4, 5]  
8 resultado = somar_vetor(vetor)  
9 print("A soma dos elementos do vetor é:", resultado)
```


Subprogramas (fundamentos)

Java

```
public class SomaVetor {  
    public static int somarVetor(int[] vetor) {  
        int soma = 0;  
        for (int i = 0; i < vetor.length; i++) {  
            soma += vetor[i];  
        }  
        return soma;  
    }  
  
    public static void main(String[] args) {  
        int[] vetor = {1, 2, 3, 4, 5};  
        int resultado = somarVetor(vetor);  
        System.out.println("A soma dos elementos do vetor é: " + resultado);  
    }  
}
```

C

```
1  #include <stdio.h>  
2  
3  int somarVetor(int vetor[], int tamanho) {  
4      int soma = 0;  
5      for (int i = 0; i < tamanho; i++) {  
6          soma += vetor[i];  
7      }  
8      return soma;  
9  }  
10  
11 int main() {  
12     int vetor[] = {1, 2, 3, 4, 5};  
13     int tamanho = sizeof(vetor) / sizeof(vetor[0]);  
14     int resultado = somarVetor(vetor, tamanho);  
15     printf("A soma dos elementos do vetor é: %d\n", resultado);  
16     return 0;  
17 }
```

Subprogramas (definições)

```
1 def saudacao(nome):  
2     print("Olá,", nome)  
3  
4 # Parâmetro formal: nome  
5 # Parâmetro real: "João"  
6 saudacao("João")
```

- ❑ Parâmetro formal:
 - ❑ É uma variável que aparece no cabeçalho do subprograma e é usada no corpo do subprograma.
- ❑ Parâmetro real:
 - ❑ Representa um valor ou endereço usado na chamada do subprograma.
- ❑ O perfil de parâmetros (ou assinatura) de um subprograma contém o número, a ordem e os tipos de seus parâmetros formais;
- ❑ Protocolo é o perfil de parâmetros mais o tipo de retorno em caso de função, se for uma função, seu tipo de retorno.

Subprogramas (definições)

- ❑ Correspondência entre os parâmetros reais e formais:

- ❑ Posicional

- ❑ A vinculação dos parâmetros reais a parâmetros formais é por posição o primeiro real é vinculado ao primeiro formal e assim por diante;

- ❑ Seguro e efetivo.

- ❑ Palavra-chave

- ❑ O nome do parâmetro formal a que um parâmetro real deve ser vinculado é especificado com o parâmetro real.

- ❑ Vantagem: Parâmetros podem aparecer em qualquer ordem, evitando erros de correspondência;

- ❑ Desvantagem: O usuário deve saber os nomes dos parâmetros formais.

Subprogramas (definições)

```
1 def somaComDefault(a, b=5):  
2     c = a + b  
3     return '\n' + str(a) + ' + ' + str(b) + ' = ' + str(c)  
4 print(somaComDefault(7))  
5 print(somaComDefault(7, 7))  
6 print(somaComDefault(b=10, a=12))
```



```
7 + 5 = 12  
7 + 7 = 14  
12 + 10 = 22
```

Subprogramas (procedimentos e funções)

❑ Existem duas categorias de subprogramas:

❑ Procedimento são coleções de instruções parametrizadas que definem uma determinada abstração;

```
1 def imprimir_saudacao(nome):  
2     print("Olá, " + nome + "! Bem-vindo(a).")  
3  
4 # Chamando o procedimento  
5 imprimir_saudacao("João")
```

❑ Funções parecem estruturalmente com os procedimentos mas são semanticamente modeladas como funções matemáticas.

```
1 def somar(a, b):  
2     return a + b  
3  
4 # Chamando a função e atribuindo o resultado a uma variável  
5 resultado = somar(3, 4)  
6 print(resultado) # Output: 7
```

Subprogramas (Ambientes de referenciamento local)

- ❑ Variáveis locais:

- ❑ São variáveis que são definidas dentro de um subprograma, e geralmente têm o mesmo escopo do subprograma;

```
1  # Variável global
2  nome = "João"
3
4  def saudacao():
5      # Variável local
6      idade = 30
7      print("Olá, " + nome) # Acessando a variável global dentro da função
8      print("Você tem", idade, "anos") # Acessando a variável local
9
10 saudacao()
11 print("Seu nome é", nome) # Acessando a variável global fora da função
12 print("Sua idade não é conhecida aqui") # Tentando acessar a variável local
13                                     # fora da função (irá resultar em um erro)
```

Subprogramas (Ambientes de referenciamento local)

- ❑ Variáveis locais:

- ❑ São variáveis que são definidas dentro de um subprograma, e geralmente têm o mesmo escopo do subprograma;

```
1 nome = "João" # Variável global
2
3 def alterar_nome(novo_nome):
4     global nome # Definindo a variável global dentro da função
5     nome = novo_nome # Atribuindo um novo valor à variável global
6
7 print("Nome original:", nome) # Imprime o nome original
8
9 alterar_nome("Maria") # Chama a função para alterar o nome
10
11 print("Novo nome:", nome) # Imprime o novo nome atribuído à variável global
```


Subprogramas (métodos de passagem de parâmetros)

```
1 def dobrar(numero):  
2     numero *= 2  
3     return numero  
4  
5 valor = 5  
6 valor_dobrado = dobrar(valor)  
7 print("Valor original:", valor)  
8 print("Valor dobrado:", valor_dobrado)
```

Passagem por valor

Passagem por valor e resultado

```
1 def dobrar_por_resultado(numero):  
2     return numero * 2  
3  
4 valor = 5  
5 valor = dobrar_por_resultado(valor)  
6 print("Valor dobrado:", valor)
```

Subprogramas (métodos de passagem de parâmetros)

```
1 def saudacao_por_nome():
2     print("Olá,", nome)
3
4 nome = "Maria"
5 saudacao_por_nome()
```

Passagem por nome

Passagem por referência

```
1 def adicionar_item(lista, item):
2     lista.append(item)
3
4 minha_lista = [1, 2, 3]
5 adicionar_item(minha_lista, 4)
6 print(minha_lista)
```

Passagem por valor e referência em C

```
#include <stdio.h>

void zera(float a){
    a = 0;
}

void main(){
    float f;
    f = 20.7;
    zera(f);
    printf("%.1f", f);
}
```

```
#include <stdio.h>

void zera(float *a){
    *a = 0;
}

void main(){
    float f;
    f = 20.7;
    zera(&f);
    printf("%.1f", f);
}
```

Dúvidas???



Fonte: <https://institutoseculoxxi.com.br/duvidas-entramos-em-contato-com-voce/>