Funções

Funções - Motivação

Como já visto, para desenharmos um quadrado no Python turtle utilizamos a seguinte sequência de commandos:

```
import turtle

tamQuadrado = 70

turtle.forward(tamQuadrado)
turtle.left(90)
turtle.forward(tamQuadrado)
turtle.left(90)
turtle.forward(tamQuadrado)
turtle.left(90)
turtle.forward(tamQuadrado)
```

Funções - Motivação

Uma forma mais simples de desenhar o quadrado é utilizando uma estrutura de repetição **for**

```
import turtle

tamQuadrado = 70

for i in range(4):
    turtle.forward(tamQuadrado)
    turtle.left(90)
```

Funções - Motivação

Suponha que queremos desenhar o quadrado em uma certa posição da tela:

```
import turtle

tamQuadrado = 70

turtle.penup()
turtle.setpos(40, 120)
turtle.pendown()
for i in range(4):
    turtle.forward(tamQuadrado)
    turtle.left(90)
```

Esse trecho de código significa: "Posicione a tartaruga na posição (40, 120) e desenhe um quadrado a partir dessa posição"

Funções - Motivação

Suponha que queremos desenhar o quadrado em uma certa posição da tela:

```
import turtle

tamQuadrado = 70

turtle.penup()
turtle.setpos(40, 120)
turtle.pendown()
for i in range(4):
    turtle.forward(tamQuadrado)
    turtle.left(90)
```

Esse trecho de código significa: "Posicione a tartaruga na posição (40, 120) e desenhe um quadrado a partir dessa posição"

Podemos tornar o código mais simples, intuitivo e modular utilizando funções!

Funções - Motivação

```
import turtle

def desenhaQuadrado(posx, posy, tamQuadrado):

    turtle.penup()
    turtle.setpos(posx, posy)
    turtle.pendown()
    for i in range(4):
        turtle.forward(tamQuadrado)
        turtle.left(90)
```

Função responsável por posicionar a tartaruga na posição (**posx**, **posy**) e desenhar um quadrado de tamanho **tamQuadrado**

Funções - Motivação

```
import turtle

def desenhaQuadrado(posx, posy, tamQuadrado):

    turtle.penup()
    turtle.setpos(posx, posy)
    turtle.pendown()
    for i in range(4):
        turtle.forward(tamQuadrado)
        turtle.left(90)


desenhaQuadrado(40, 120, 70)
```

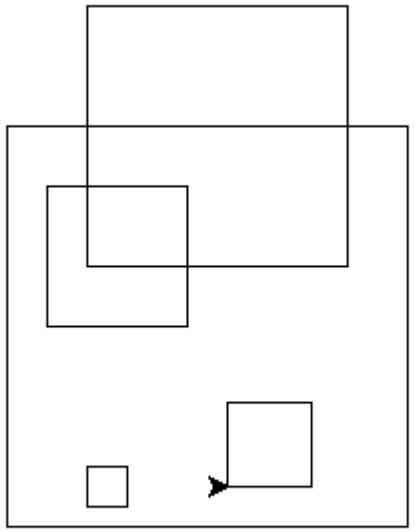
Função responsável por posicionar a tartaruga na posição (**posx**, **posy**) e desenhar um quadrado de tamanho **tamQuadrado**

Chama a função **desenhaQuadrado** com a posição e tamanho desejado

Funções - Motivação

Tendo definido a função **desenhaQuadrado**, podemos chamá-la quantas vezes quisermos no programa:

```
desenhaQuadrado(40, 120, 70)
desenhaQuadrado(20, 20, 200)
desenhaQuadrado(60, 30, 20)
desenhaQuadrado(60, 150, 130)
desenhaQuadrado(130, 40, 42)
```



Função - Sintaxe

Para definir funções utilizamos a seguinte sintaxe:

```
def nomeDaFuncao(parametro1, parametro2, ...):
    comando1
    comando2
    ...
    return valorRetornado
```

Chamamos a função da seguinte forma:

```
valor = nomeDaFuncao(argumento1, argumento2, ...)
```

Função - Sintaxe

Exemplo:

```
def somaValores(x, y):
    soma = x + y
    return soma


soma = somaValores(5, 6)
print(soma)
```

Exercício 1

Faça uma função que receba dois valores de entrada, e retorne o maior valor dentre os dois

Função - Sintaxe

```
def nomeDaFuncao(parametro1, parametro2, ...):
    comando1
    comando2
    ...
    return valorRetornado
```

Função - Sintaxe

```
def nomeDaFuncao(parametro1, parametro2, ...):
    comando1
    comando2
    ...
    return valorRetornado
```

Parâmetros da função

Argumentos da função

```
valor = nomeDaFuncao(argumento1, argumento2, ...)
```

Função - Sintaxe

- A ordem dos argumentos deve ser a mesma que a ordem dos parâmetros
- Qualquer tipo de variável pode ser parâmetro de função

Função - Sintaxe

- Passando valores inteiros:

```
def soma(v1, v2, v3, v4, v5):
    soma = v1+v2+v3+v4+v5
    return soma

resultado = soma(1, 2, 3, 4, 5)
print(resultado)
```

Função - Sintaxe

- Passando valores inteiros:

```
def soma(v1, v2, v3, v4, v5):
    soma = v1+v2+v3+v4+v5
    return soma

resultado = soma(1, 2, 3, 4, 5)
print(resultado)
```

- Passando uma lista de valores:

```
def soma(listaDeValores):
    soma = 0
    for i in range(len(listaDeValores)):
        soma = soma + listaDeValores[i]

    return soma

resultado = soma([1,2,3,4,5])
print(resultado)
```

Função - Sintaxe

- Podemos também utilizar funções que não recebem nenhum parâmetro:

```
def sucesso():
    print("O programa foi executado com sucesso!")

val = float(input("Digite um valor: "))
...
sucesso()
```

- É importante lembrar que mesmo que a função não receba nenhum parâmetro, ainda precisamos utilizar () na definição e na chamada da função

Função

- A ordem dos argumentos deve ser a mesma que a ordem dos parâmetros
- Qualquer tipo de variável pode ser parâmetro de função
- Usualmente, funções são definidas **no início do código**

Função - Sintaxe

```
def verifica(par1):
    ...

def soma(v1, v2):
    ...

def adiciona(nome, lista):
    ...

valor = int(input("Digite um valor: "))
res = verifica(valor)
...
```

Definição das funções a serem utilizadas

Comandos do programa principal

Função

- A ordem dos argumentos deve ser a mesma que a ordem dos parâmetros
- Qualquer tipo de variável pode ser parâmetro de função
- Usualmente, funções são definidas **no início do código**
- Uma função pode chamar outra função

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

def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")

preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

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

def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")
```

```
preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

Recebe preços, cria uma lista com os preços e passa para a função **verificaSoma**

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

```
def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")
```

Chama função **soma**. Verifica se valor maior que R$50

```
preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

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

Calcula a soma dos valores de uma lista

```
def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")

preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

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

def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")

preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

Função

- A ordem dos argumentos deve ser a mesma que a ordem dos parâmetros
- Qualquer tipo de variável pode ser parâmetro de função
- Usualmente, funções são definidas **no início do código**
- Uma função pode chamar outra função
- É importante documentarmos o que cada função do código faz, incluindo quem são os argumentos. Isso porque a função provavelmente vai ser reutilizada no futuro

```
# Soma valores contidos em listaDeValores
def soma(listaDeValores):
    soma = 0
    for i in range(len(listaDeValores)):
        soma = soma + listaDeValores[i]
    return soma

# Verifica se a soma dos preços é maior que R$50
# precos é uma variável do tipo lista
def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")

preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

Exercício

Faça uma função que recebe um valor inteiro como entrada e retorna *True* se o número for par e *False* caso contrário.

Função

- Importante! Uma função não é executada na sua definição, apenas quando ela é chamada!

Função

- Importante! Uma função não é executada na sua definição, apenas quando ela é chamada!
- O código abaixo imprime corretamente o valor que o usuário digitou, já que a função **funcaoComErro** nunca é chamada

```
def funcaoComErro(valor):
    sdfa
    fgffsdgfgsd
    gsdgdsfg

    return dsafsf

valor = int(input("Digite um valor: "))
print("O valor é {}".format(valor))
```

Importante!

- A definição de uma função pode ser vista como um pedaço de código que será executado mais tarde pelo programa, quando necessário.

```
def somaValores(x, y):
    soma = x + y
    return soma
```

Define a função **somaValores**. Esse código não é executado nesse momento.

```
soma = somaValores(5, 6)
print(soma)
```

Executa o código definido acima utilizando os valores x=5 e y=6

Importante!!!

Colchetes possui um significado completamente diferente de parênteses

valores[i]

Acessa o elemento **i** da lista **valores**

valores(i)

Chama a função **valores** passando a variável **i** como argumento

Cuidado com nomes de funções

Variáveis não podem ter o mesmo nome que funções definidas no programa. Caso isso ocorra, a função será substituída pela variável

Cuidado com nomes de funções

Variáveis não podem ter o mesmo nome que funções definidas no programa. Caso isso ocorra, a função será substituída pela variável

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

maiorValor = maiorValor(3, 7)
print(maiorValor)
maiorValor = maiorValor(8, 13)
print(maiorValor)
```

Chama a função **maiorValor**, associa o resultado à variável **maiorValor** e imprime. Esse trecho executa corretamente

Cuidado com nomes de funções

Variáveis não podem ter o mesmo nome que funções definidas no programa. Caso isso ocorra, a função será substituída pela variável

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

maiorValor = maiorValor(3, 7)
print(maiorValor)
maiorValor = maiorValor(8, 13)
print(maiorValor)
```

Trata a variável **maiorValor** como função, o que está incorreto. Essa linha dará erro.

Exercício

Faça uma função que receba uma lista de valores **listVal** e um número inteiro **n**. A função retorna uma nova lista, cada elemento dessa nova lista é dado pelo respectivo elemento da lista **listVal** multiplicado por **n**.

Função forward()
da biblioteca turtle:

```
def forward(dist):
    """Move the pen to the point end, thereby drawing a line
    if pen is down. All other methods for turtle movement depend
    on this one.
    """
    ## Version with undo-stuff
    go_modes = ( self._drawing,
                 self._pencolor,
                 self._pensize,
                 isinstance(self._fillpath, list))
    screen = self.screen
    undo_entry = ("go", self._position, end, go_modes,
                  (self.currentLineItem,
                  self.currentLine[:],
                  screen._pointlist(self.currentLineItem),
                  self.items[:])
                  )
    if self.undobuffer:
        self.undobuffer.push(undo_entry)
    start = self._position
    if self._speed and screen._tracing == 1:
        diff = (end-start)
        diffsq = (diff[0]*screen.xscale)**2 + (diff[1]*screen.yscale)**2
        nhops = 1+int((diffsq**0.5)/(3*(1.1**self._speed)*self._speed))
        delta = diff * (1.0/nhops)
        for n in range(1, nhops):
            if n == 1:
                top = True
            else:
                top = False
            self._position = start + delta * n
            if self._drawing:
                screen._drawline(self.drawingLineItem,
                                 (start, self._position),
                                 self._pencolor, self._pensize, top)
            self._update()
        if self._drawing:
            screen._drawline(self.drawingLineItem, ((0, 0), (0, 0)),
                                           fill="", width=self._pensize)
    # Turtle now at end,
    if self._drawing: # now update currentLine
        self.currentLine.append(end)
    if isinstance(self._fillpath, list):
        self._fillpath.append(end)

    self._position = end
    if self._creatingPoly:
        self._poly.append(end)
    if len(self.currentLine) > 42: # 42! answer to the ultimate question
                                   # of life, the universe and everything
        self._newLine()
    self._update() #count=True)
```

Escopo de variáveis

Escopo de variáveis

- O termo *escopo de variáveis* está relacionado com o local do código no qual uma determinada variável pode ser acessada.

- Em um programa, podemos ter dois tipos de escopo:
  - Local
  - Global

Escopo de variáveis

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

def verificaSoma(precos):
    valorTotal = soma(precos)
    if valorTotal>50:
        print("Está muito caro!")
    else:
        print("Pode comprar!")

preco1 = float(input("Preço do produto 1: "))
preco2 = float(input("Preço do produto 2: "))
preco3 = float(input("Preço do produto 3: "))

valores = [preco1, preco2, preco3]
verificaSoma(valores)
```

Escopo local.
Dentro de funções.

Escopo global.
Fora de qualquer função.

Escopo de variáveis

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(resultado)
```

Variáveis **valor1**, **valor2** e **maior** estão no escopo da função **maiorValor**. Portanto, não podem ser acessadas por comandos que estão fora da função.

Variáveis **x**, **y** e **resultado** estão no escopo global. Portanto elas são chamadas de variáveis globais, pois podem ser acessadas de qualquer lugar do programa, inclusive dentro de funções.

Escopo de variáveis

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(resultado)
```

Variáveis **valor1**, **valor2** e **maior** estão no escopo da função **maiorValor**. Portanto, não podem ser acessadas por comandos que estão fora da função.

Variáveis **x**, **y** e **resultado** estão no escopo global. Portanto elas são chamadas de variáveis globais, pois podem ser acessadas de qualquer lugar do programa, inclusive dentro de funções.

Variáveis globais, na grande maioria dos casos, não devem ser acessadas dentro de funções!

Escopo de variáveis - Exemplo

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(maior)
```

Esse código retorna um erro!

```
Traceback (most recent call last):
  File "D:/Dropbox/temp.py", line 12, in <module>
    print(maior)
NameError: name 'maior' is not defined
```

Escopo de variáveis - Exemplo

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(maior)
```

Esse código retorna um erro!

Variável **maior**, definida dentro da função, não pode ser acessada pelo programa principal

```
Traceback (most recent call last):
  File "D:/Dropbox/temp.py", line 12, in <module>
    print(maior)
NameError: name 'maior' is not defined
```

Escopo de variáveis - Exemplo

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    print(x)
    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
```

Esse código imprime na tela o valor da variável **x**. Pois a variável global **x** pode ser acessada pela função

Escopo de variáveis - Exemplo

É comum tentarmos fazer o seguinte código:

```
def maiorValor():
    if x>=y:
        maior = x
    else:
        maior = y

    return maior

x = 3
y = 7
resultado = maiorValor()
print(resultado)
```

Escopo de variáveis - Exemplo

É comum tentarmos fazer o seguinte código:

```
def maiorValor():
    if x>=y:
        maior = x
    else:
        maior = y

    return maior

x = 3
y = 7
resultado = maiorValor()
print(resultado)
```

- O código funciona nesse caso, mas utiliza as variáveis do programa principal nos cálculos, o que não é recomendado.
- Imaginem o que vai acontecer em um programa possuindo diversas funções, todas utilizando as mesmas variáveis do programa principal!!

Escopo de variáveis - Exemplo

Uso correto das variáveis na função:

```
def maiorValor(valor1, valor2):
    if valor1>=valor2:
        maior = valor1
    else:
        maior = valor2

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(resultado)
```

Escopo de variáveis - Exemplo

Notem que os nomes dos parâmetros definidos na função podem ser os mesmos das variáveis do programa principal:

```
def maiorValor(x, y):
    if x>=y:
        maior = x
    else:
        maior = y

    return maior

x = 3
y = 7
resultado = maiorValor(x, y)
print(resultado)
```

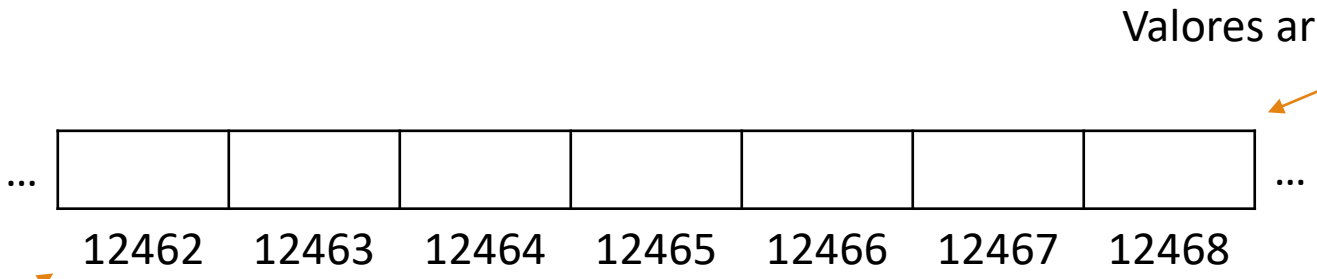
As variáveis **x** e **y** dentro da função **maiorValor** são **diferentes** das variáveis **x** e **y** definidas no programa principal!!

Endereçamento de variáveis

- Vamos dar uma olhada no funcionamento da memória do computador
- A ideia é entender porque podemos ter duas variáveis de mesmo nome mas que armazenam valores diferentes

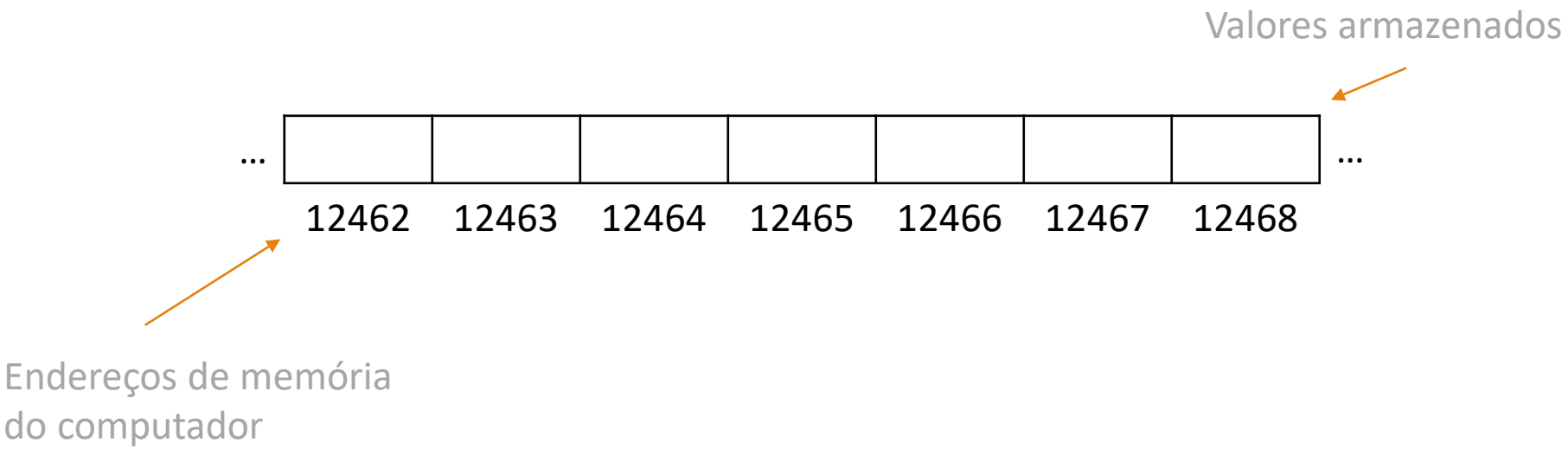
Endereçamento de variáveis

Funcionamento da memória do computador



Endereçamento de variáveis

Funcionamento da memória do computador

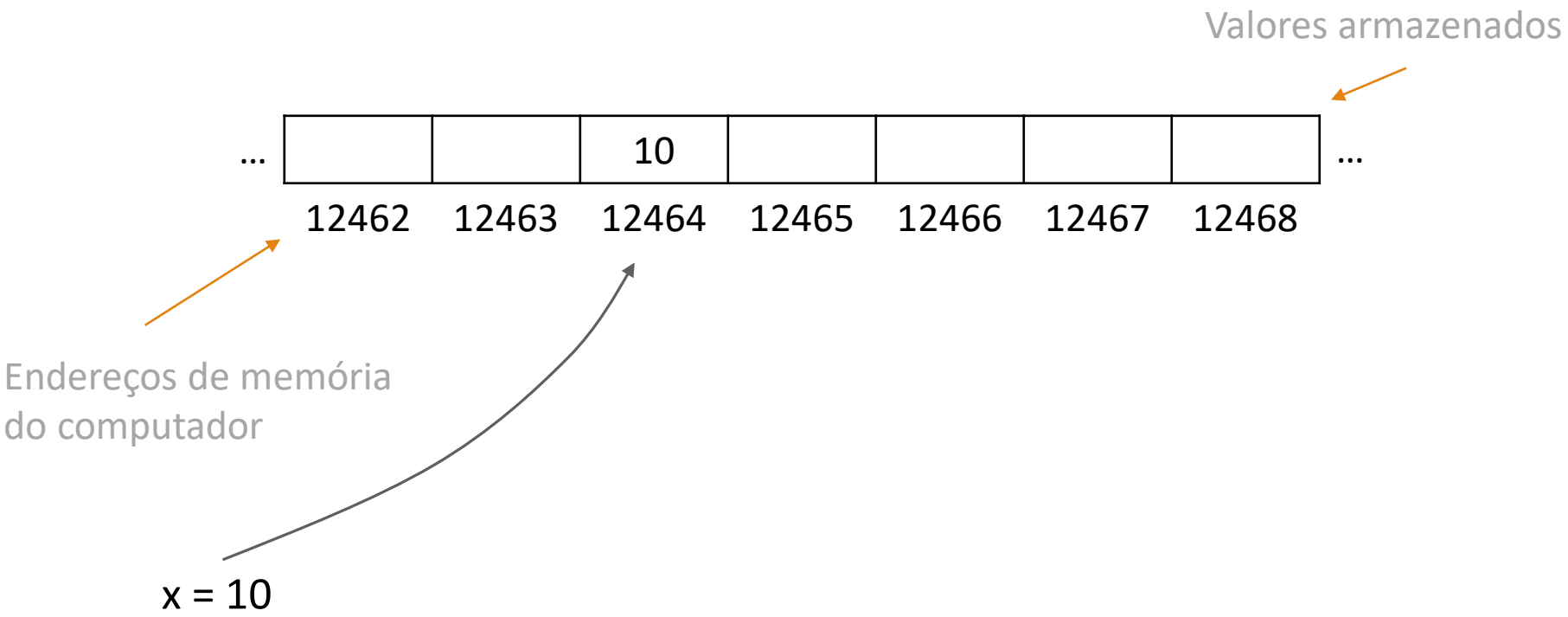


Vamos supor que o seguinte comando é utilizado no código:

x = 10

Endereçamento de variáveis

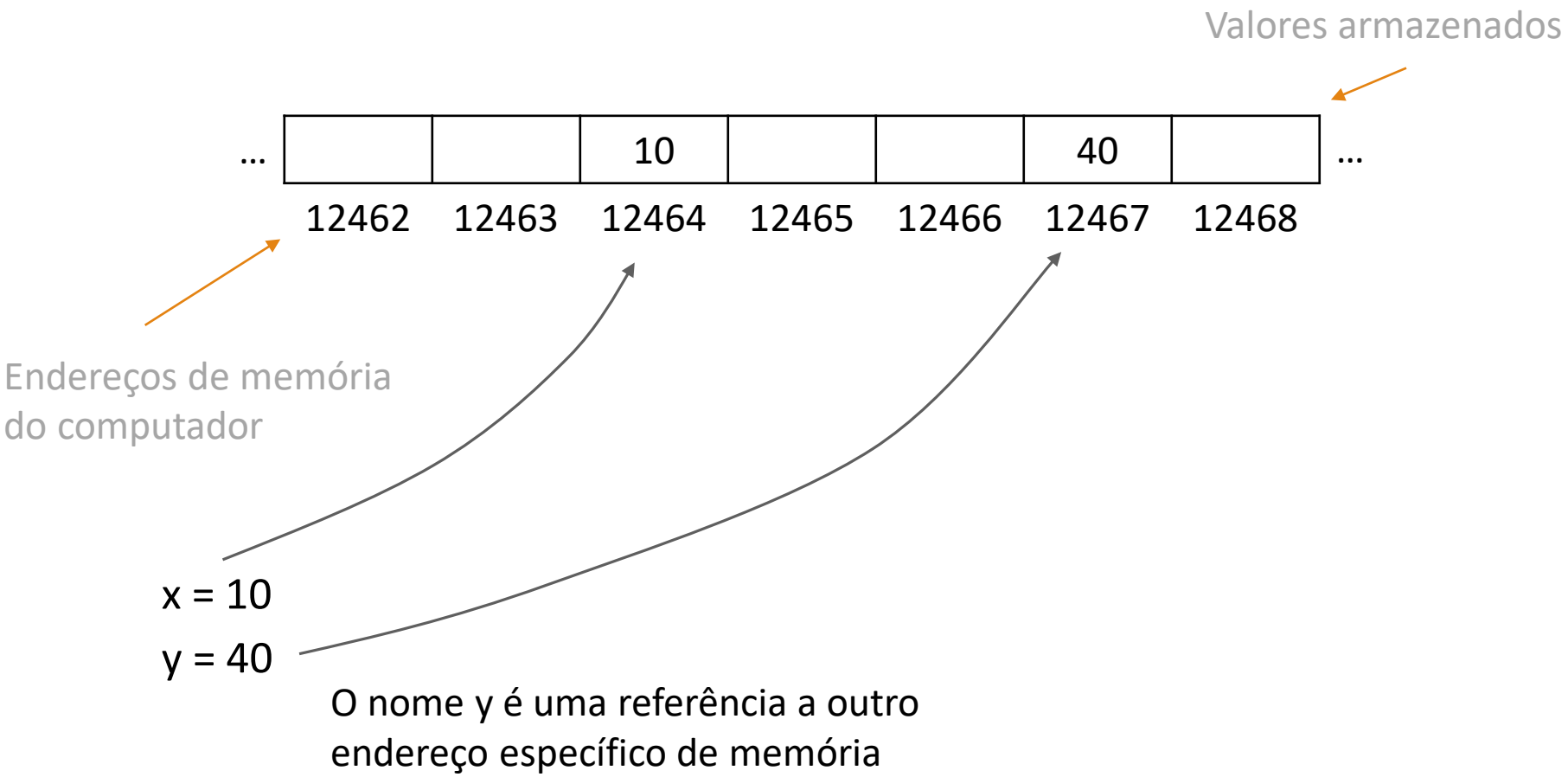
Funcionamento da memória do computador



O nome x é uma referência a um endereço específico de memória (selecionado de forma “aleatória”)

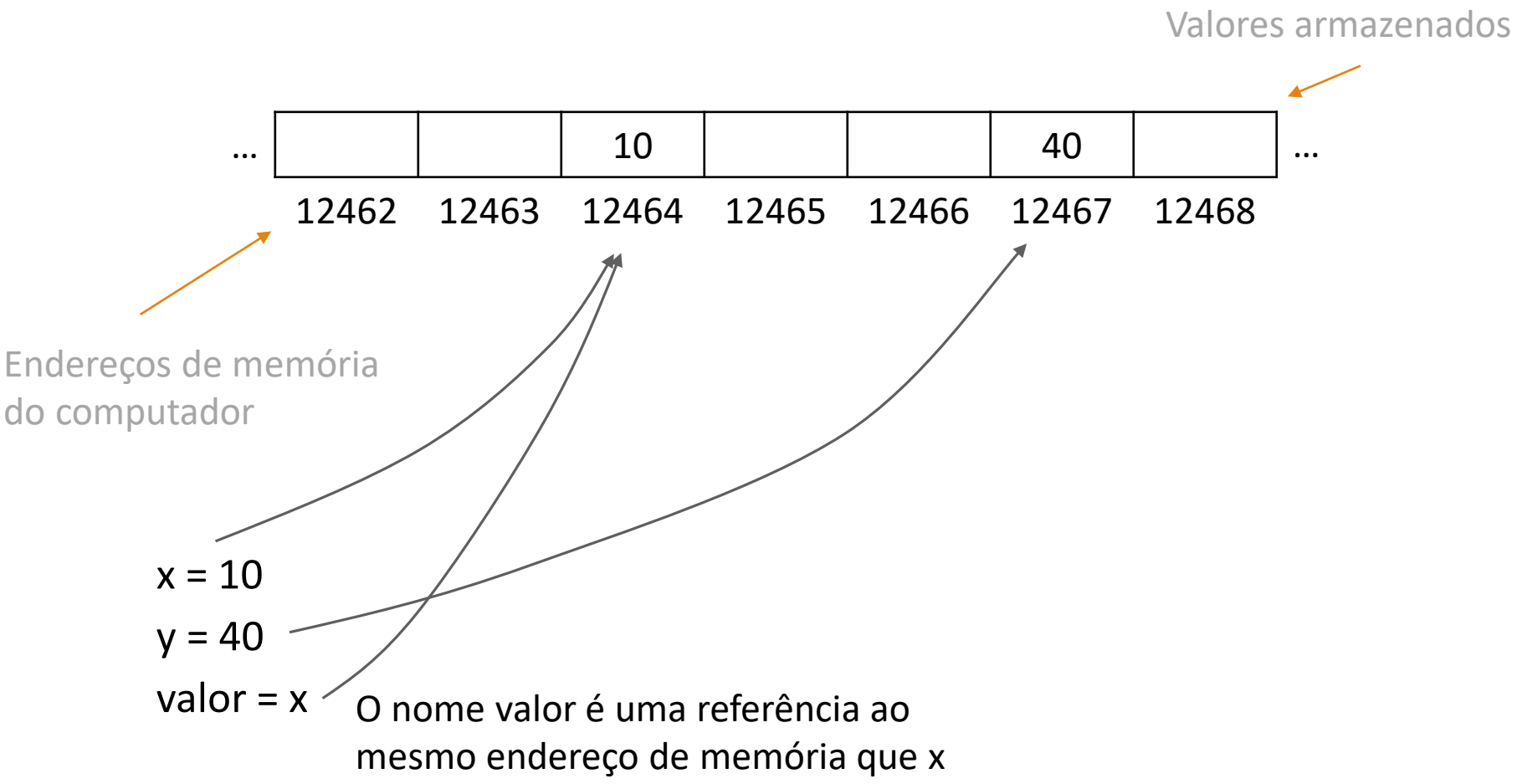
Endereçamento de variáveis

Funcionamento da memória do computador

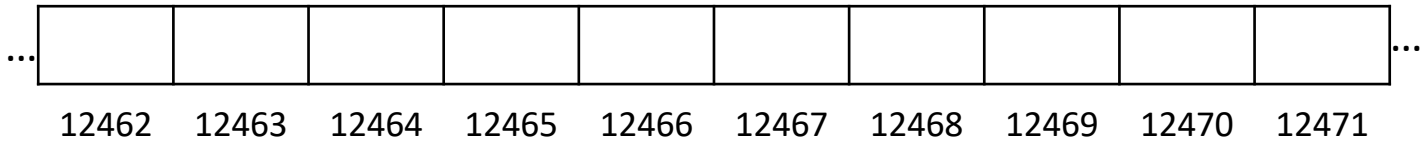


Endereçamento de variáveis

Funcionamento da memória do computador



Endereçamento de variáveis

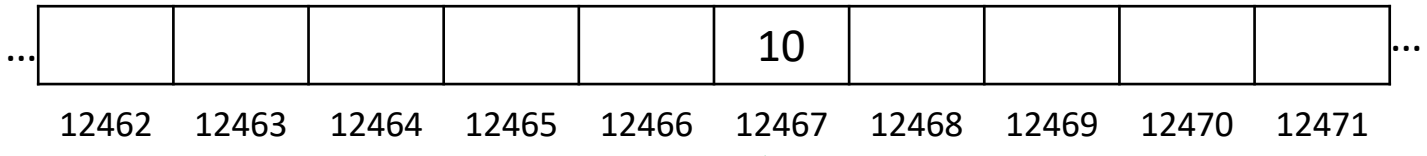



```
def muda_valor(x):
    x = 30
    return x

x = 10
resultado = muda_valor(x)
```

Vamos ver como as variáveis do código ao lado são alocadas na memória

Endereçamento de variáveis

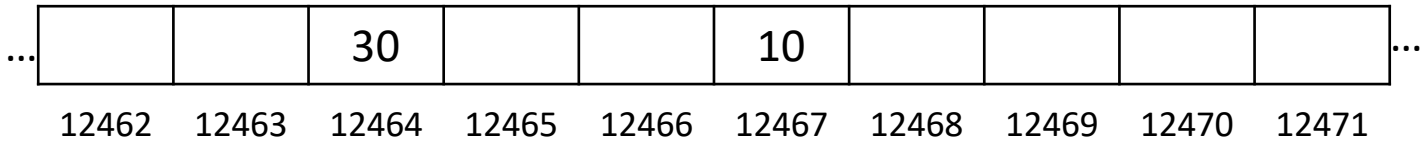



```
def muda_valor(x):
    x = 30
    return x

x = 10
resultado = muda_valor(x)
```

O nome x é associado a um endereço de memória contendo o valor 10

Endereçamento de variáveis

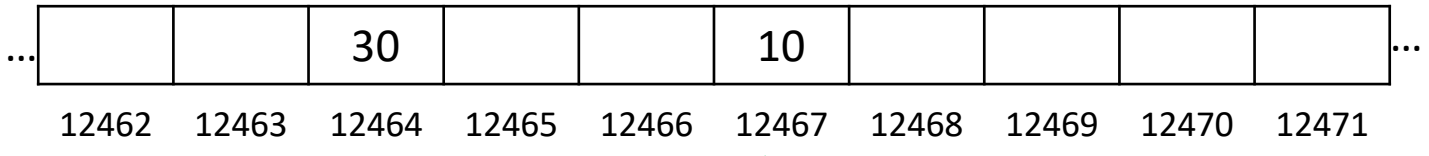



```
def muda_valor(x):
    x = 30
    return x

x = 10
resultado = muda_valor(x)
```

Quando a função é executada, a variável x dentro da função é associada a um novo endereço de memória

Endereçamento de variáveis




```
def muda_valor(x):
    x = 30
    return x

x = 10
resultado = muda_valor(x)
```

Após o término da função, o nome **x**, definido dentro da função, não existe mais. Mas o nome **resultado** armazena a referência para o valor

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    print("x dentro da função: {}".format(x))


x = 3
print("x fora da função: {}".format(x))
```

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    print("x dentro da função: {}".format(x))


x = 3
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x fora da função: 3
```

Isso ocorre porque a função minhaFuncao nunca foi chamada

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x dentro da função: 3
x fora da função: 3
```

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    x = 10
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    x = 10
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x dentro da função: 10
x fora da função: 3
```

Escopo de variáveis - Exemplos

```
def minhaFuncao():
    x = 10
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao()
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x dentro da função: 10
x fora da função: 3
```

Quando definimos um valor de **x** dentro da função, estamos criando uma nova variável que pertence somente à função

Portanto, o programa acima possui duas variáveis chamadas **x**: a variável **x** do programa principal e a variável **x** da função **minhaFuncao**.

Escopo de variáveis - Resumo

- Variáveis globais podem ser acessadas dentro de uma função
- Variáveis globais não podem ser redefinidas dentro de uma função
  - A redefinição de uma variável global dentro de uma função cria uma nova variável com o mesmo nome

Uso mais comum de parâmetros de funções

```
def minhaFuncao(x):
    print("x dentro da função: {}".format(x))


x = 3
minhaFuncao(x)
print("x fora da função: {}".format(x))
```

O programa imprime:

```
x dentro da função: 3
x fora da função: 3
```

Apenas lembrem-se que qualquer modificação da variável **x** dentro da função **minhaFuncao** não alterará o valor de **x** no programa principal.

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Define uma nova variável **x**, pertencente apenas ao escopo dessa função, e eleva ela ao quadrado

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Define uma nova variável **x**, pertencente apenas ao escopo dessa função, e eleva ela ao quadrado

Define novas variáveis **x** e **aux** pertencentes apenas ao escopo dessa função. Modifica o valor de **x** dentro da função e eleva **aux** ao quadrado

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Define uma nova variável **x**, pertencente apenas ao escopo dessa função, e eleva ela ao quadrado

Define novas variáveis **x** e **aux** pertencentes apenas ao escopo dessa função. Modifica o valor de **x** dentro da função e eleva **aux** ao quadrado

O valor impresso será 9 para todos os casos

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Define uma nova variável **x**, pertencente apenas ao escopo dessa função, e eleva ela ao quadrado

Define novas variáveis **x** e **aux** pertencentes apenas ao escopo dessa função. Modifica o valor de **x** dentro da função e eleva **aux** ao quadrado

O valor impresso será 9 para todos os casos

O valor impresso será 3

Exemplos de funções com diferentes tratamentos de escopo

```
def potencia2_v1():
    return x**2

def potencia2_v2(x):
    return x**2

def potencia2_v3(x):
    aux = x
    x = 0
    return aux**2

x = 3

r1 = potencia2_v1()
r2 = potencia2_v2(x)
r3 = potencia2_v3(x)

print(r1)
print(r2)
print(r3)
print(x)
# print(aux)
```

Eleva ao quadrado a variável **x** do programa principal

Define uma nova variável **x**, pertencente apenas ao escopo dessa função, e eleva ela ao quadrado

Define novas variáveis **x** e **aux** pertencentes apenas ao escopo dessa função. Modifica o valor de **x** e eleva **aux** ao quadrado

O valor impresso será 9 para todos os casos

O valor impresso será 3

Não podemos imprimir **aux**, pois não foi definido no programa principal

Exercício

Faça um código possuindo duas funções:

menor: recebe uma lista de valores e retorna o menor valor
maior: recebe uma lista de valores e retorna o maior valor

Gere uma lista qualquer e chame ambas as funções utilizando a lista gerada como argumento. Não utilizar as funções prontas do Python min() e max()!

Exercício – Operações em listas

Vamos fazer uma biblioteca de operações matemáticas com listas!

Nossa biblioteca deve conter as seguintes funções:

somaListas: Recebe duas listas e retorna a soma delas
multiplicaConstante: Recebe uma lista e um número, retorna uma nova lista contendo cada elemento da lista original multiplicado pelo número de entrada
somaConstante: Recebe uma lista e um número, retorna uma nova lista contendo cada elemento da lista original somado pelo número de entrada
soma: Recebe uma lista e retorna a soma dos valores da lista