Princípios de Construção de Algoritmos

AULA 05 - Funções e Procedimentos

Prof. Gabriel Resende Machado

gabrielmachado@unifeso.edu.br

https://www.linkedin.com/in/machadogabriel

https://github.com/UNIFESO-Gabriel/pca

Última Aula

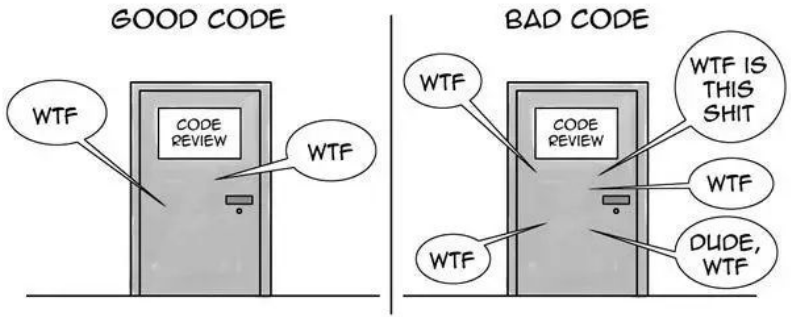
- Revisão sobre conceitos computacionais elementares: algoritmos, programas, sistemas;
- Fundamentos da Arquitetura de Von Neumann;
- Análise de uma linguagem de programação: léxica, sintática e semântica;
- Apresentação do Python, indentação, tipos primitivos e *type casting*;
- Operadores de atribuição, aritméticos e booleanos;
- Lógica booleana (AND, OR, XOR, NOT);
- Declaração de variáveis e constantes, nomenclatura de variáveis;
- Estruturas de Decisão: IF, ELSE, ELIF;
- Estruturas de Repetição: WHILE e FOR; Casos de uso.
- Comandos BREAK e CONTINUE.

Na Aula de Hoje

- Estruturação de Programas e o Conceito de Encapsulamento;
- Criando um primeiro projeto em Python;
- Modularização;
- Subrotinas, Procedimentos e Funções;
- Assinatura de Subrotinas;
- A palavra-chave *return*;
- Escopos locais e globais;
- Exercícios práticos.

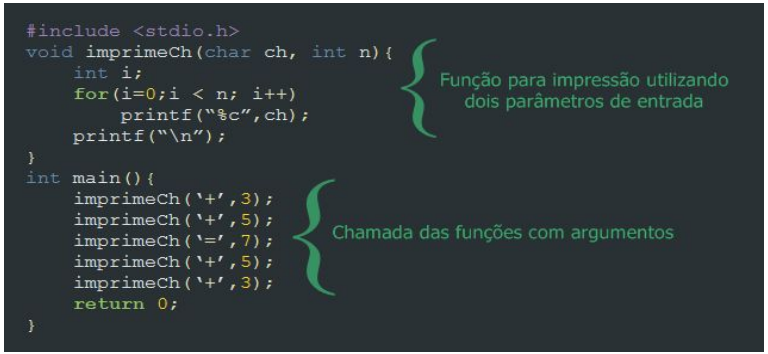
Python até o momento…

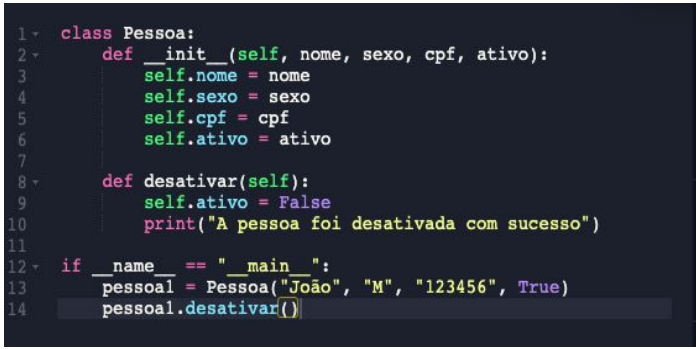
- Programas monolíticos, baseados em uma sequência de instruções *top-down*;
- Abordagem mais apropriada para didática e programas muito simples;
- Entretanto há problemas com essa abordagem em projetos maiores:
  - Torna o código bagunçado e difícil de manter;
  - Prejudica a produtividade entre os membros de uma equipe de desenvolvimento;
  - Mais difícil de rastrear erros e *bugs*.



Subrotinas

- **A contagem de linhas de código não é uma medida de produtividade!**
  - Funcionalidades muito grandes podem apontar para possíveis *bad smells;*
- **O conceito de *Single Responsibility Principle* precisa "estar na veia";**
- Para isso existem as chamadas **subrotinas**!
  - **Procedimentos** são subrotinas que não retornam nenhum valor;
  - **Funções** são subrotinas que retornam um ou mais valores;
  - **Métodos** são procedimentos ou funções presentes dentro de uma classe (será visto depois).



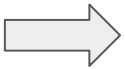


Diferenças: Monolítico *vs* Subrotinas

```python
def main():
    print("Bem-vindo à Calculadora")

    num1 = float(input("Digite o primeiro número: "))
    operador = input("Digite o operador (+, -, *, /): ")
    num2 = float(input("Digite o segundo número: "))

    if operador == '+':
        resultado = num1 + num2
    elif operador == '-':
        resultado = num1 - num2
    elif operador == '*':
        resultado = num1 * num2
    elif operador == '/':
        if num2 != 0:
            resultado = num1 / num2
        else:
            print("Erro: Divisão por zero")
            return
    else:
        print("Operador inválido")
        return

    print(f"O resultado é: {resultado}")

if __name__ == "__main__":
    main()
```

Diferenças: Monolítico *vs* Subrotinas

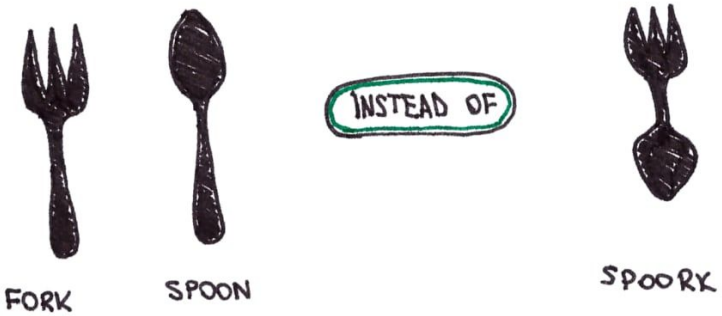
```python
def add(a: float, b: float) -> float:
    return a + b

def subtract(a: float, b: float) -> float:
    return a - b

def multiply(a: float, b: float) -> float:
    return a * b

def divide(a: float, b: float) -> float:
    if b == 0:
        raise ValueError("Erro: Divisão por zero")
    return a / b

def calculate(num1: float, operator: str, num2: float) -> float:
    operations = {
        '+': add,
        '-': subtract,
        '*': multiply,
        '/': divide
    }

    if operator in operations:
        return operations[operator](num1, num2)
    else:
        raise ValueError("Operador inválido")
```



```python
def main():
    print("Bem-vindo à Calculadora")

    try:
        num1 = float(input("Digite o primeiro número: "))
        operador = input("Digite o operador (+, -, *, /): ")
        num2 = float(input("Digite o segundo número: "))

        resultado = calculate(num1, operador, num2)
        print(f"O resultado é: {resultado}")

    except ValueError as e:
        print(e)

if __name__ == "__main__":
    main()
```

Decomposição por meio de Subrotinas

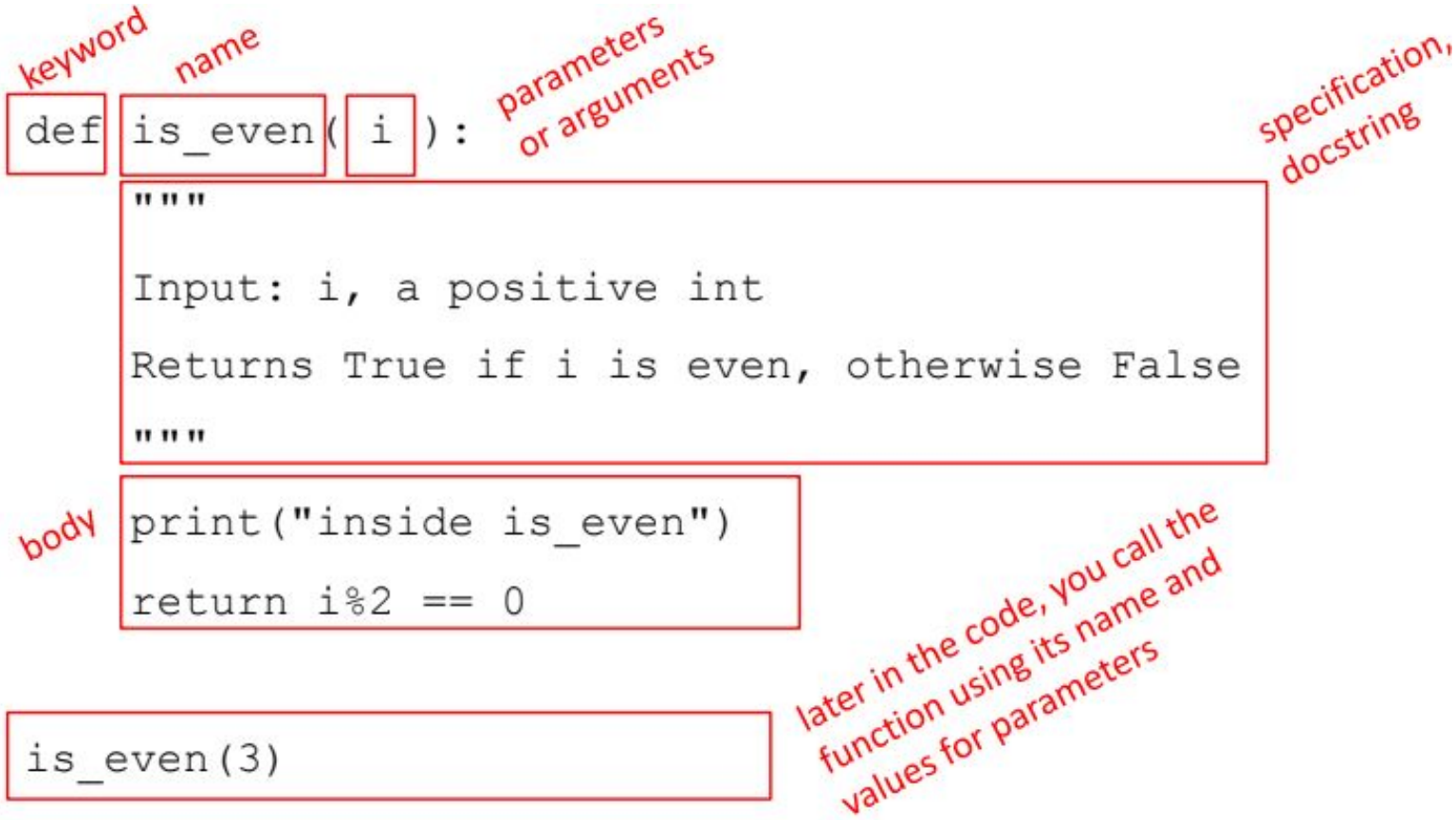
- No exemplo da calculadora visto anteriormente:
  - Funcionalidades repartidas em subrotinas que retornam um *float*: **funções**;
  - Cada funcionalidade faz apenas uma coisa: ***Single Responsibility Principle;***
  - As funções podem ser reutilizadas e testadas com maior facilidade;
  - Novas funcionalidades podem ser implementadas mais facilmente seguindo o exemplo;
  - As implementações não são visíveis nem diretamente acessíveis à **main**.
- Decomposição e separação de funcionalidades podem ser realizadas por meio de:
  - subrotinas;
  - classes;
  - módulos.



Destrinchando Subrotinas

- Subrotinas não são executadas nem alocadas em memórias até que sejam explicitamente **invocadas** ou **chamadas** em um programa;
- Características principais de subrotinas:
  - Têm um nome (respeitando as mesmas regras da declaração de nomes em variáveis);
  - Têm parâmetros (0 ou mais);
  - Têm uma documentação por meio de *docstrings* (recomendado, mas não é obrigatório);
  - Têm um corpo de implementação;
  - Retornam algo (None ou outro tipo qualquer).

Destrinchando Subrotinas



Destrinchando Subrotinas

```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

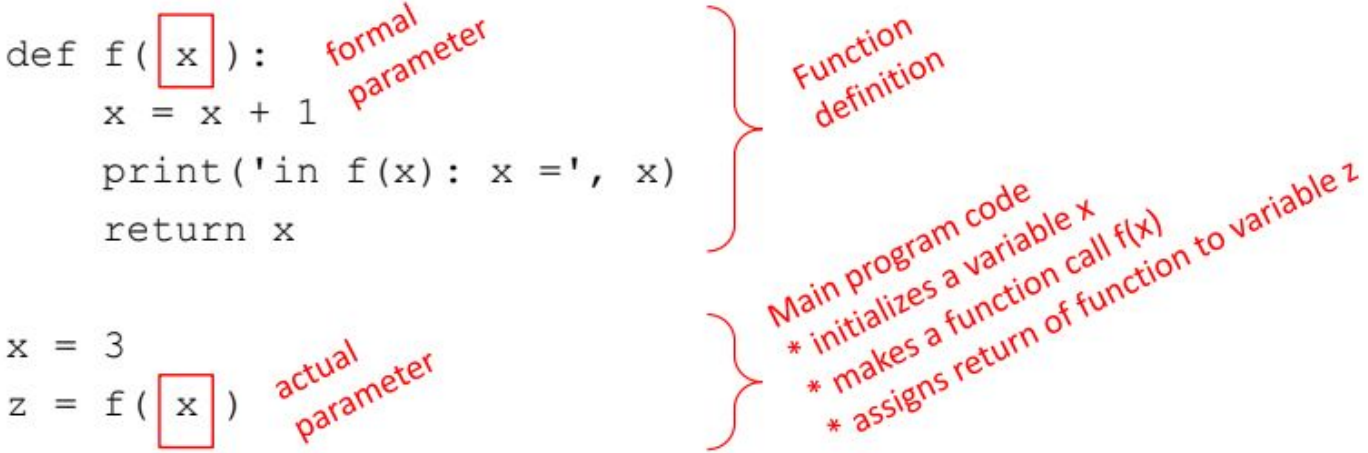
run some commands

keyword

expression to evaluate and return

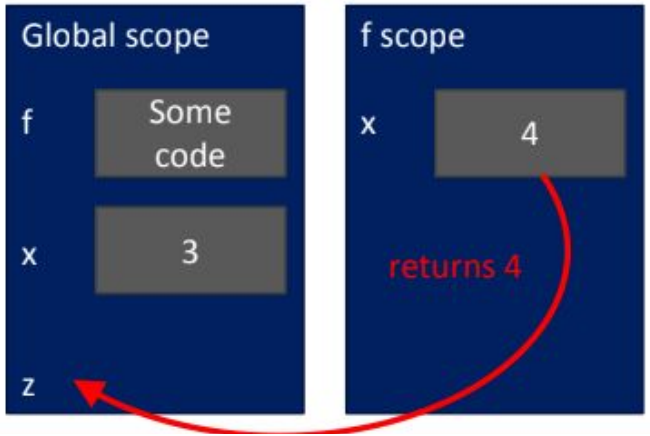
Escopo de Variáveis

- Um parâmetro formal (**parâmetro**) recebe o valor de um parâmetro real (**argumento**) sempre que uma subrotina é chamada;
- Um novo escopo é criado sempre que o fluxo de execução do programa adentra uma subrotina.

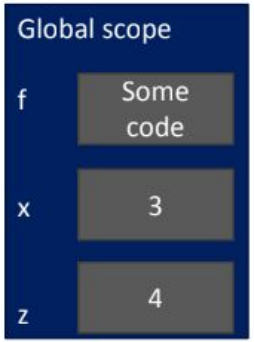


Retorno de Variáveis

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```

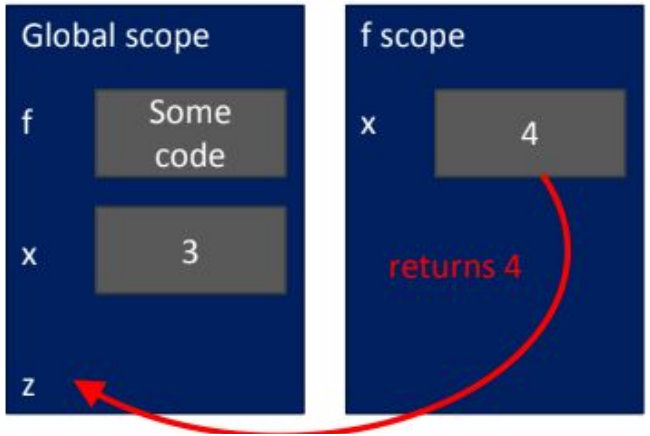


```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```

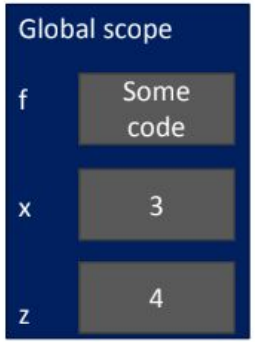


Retorno de Variáveis

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```




```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Se uma subrotina não tiver um cláusula *return*, o valor retornado é *None*.

None representa a ausência de valor.

Sobre *return vs print*

return	***print***
A cláusula return só tem sentido dentro de uma subrotina.	Utilizada em diversos contextos de um programa para exibir valores no console.
Somente um return é executado dentro dentro de uma subrotina.	É uma subrotina per se, e pode ser executada diversas vezes dentro de uma subrotina.
Código após um comando return sequencial não é executado.	O código após um comando print é executado normalmente.
Sempre tem um valor associado com ele, devolvido à subrotina chamadora.	Recebe um valor como parâmetro, mostrado no console. Se nenhum for passado, exibe uma linha em branco.

Subrotinas como argumentos

- Argumentos podem ser de qualquer tipo, inclusive outras subrotinas;

```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```

call `func_a`, takes no parameters

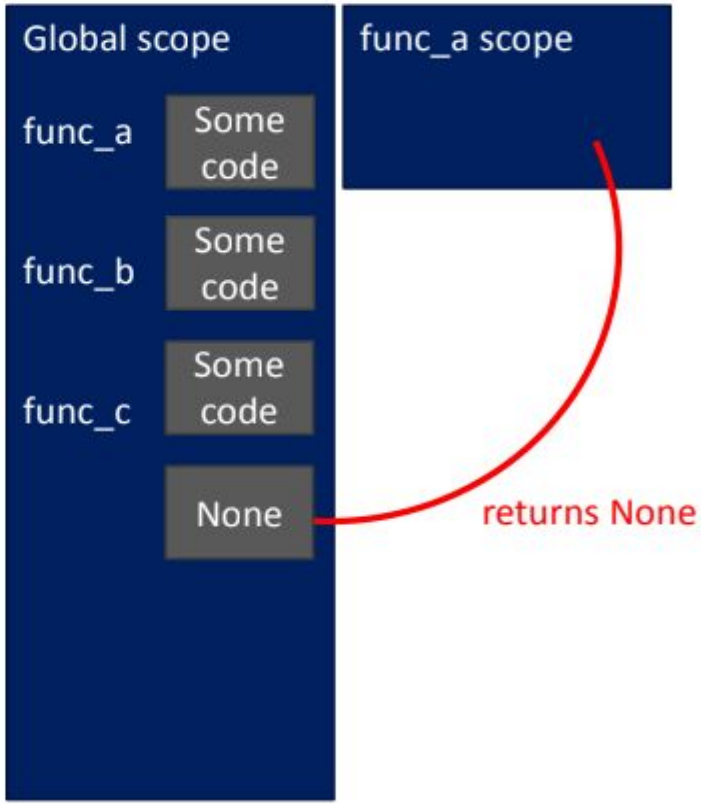
call `func_b`, takes one parameter

call `func_c`, takes one parameter, another function

Subrotinas como argumentos

- Argumentos podem ser de qualquer tipo, inclusive outras subrotinas;

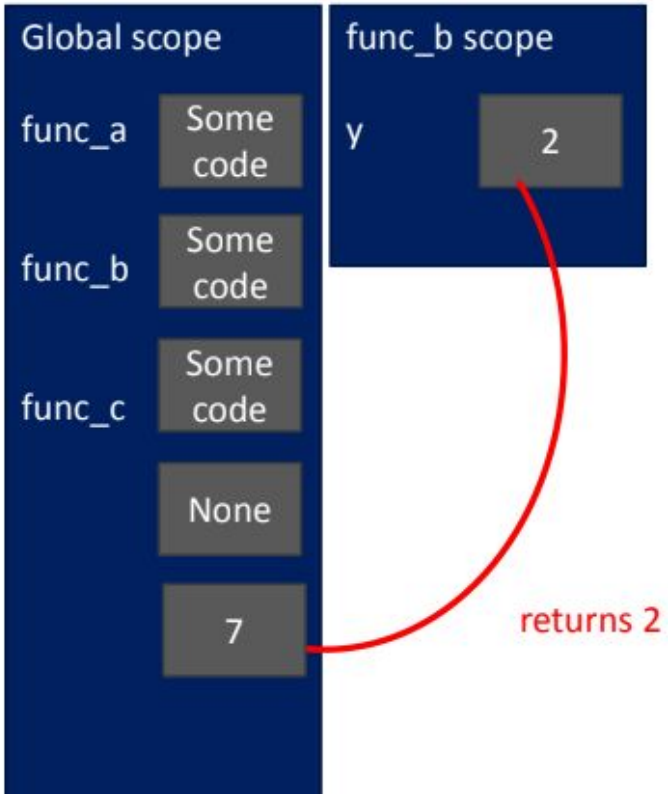
```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



Subrotinas como argumentos

- Argumentos podem ser de qualquer tipo, inclusive outras subrotinas;

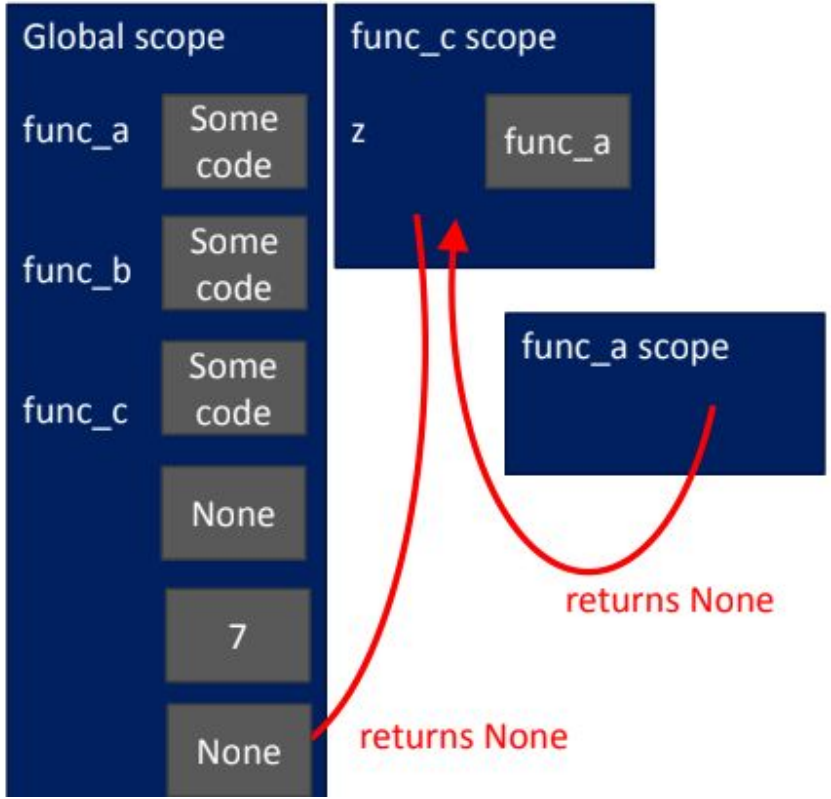
```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



Subrotinas como argumentos

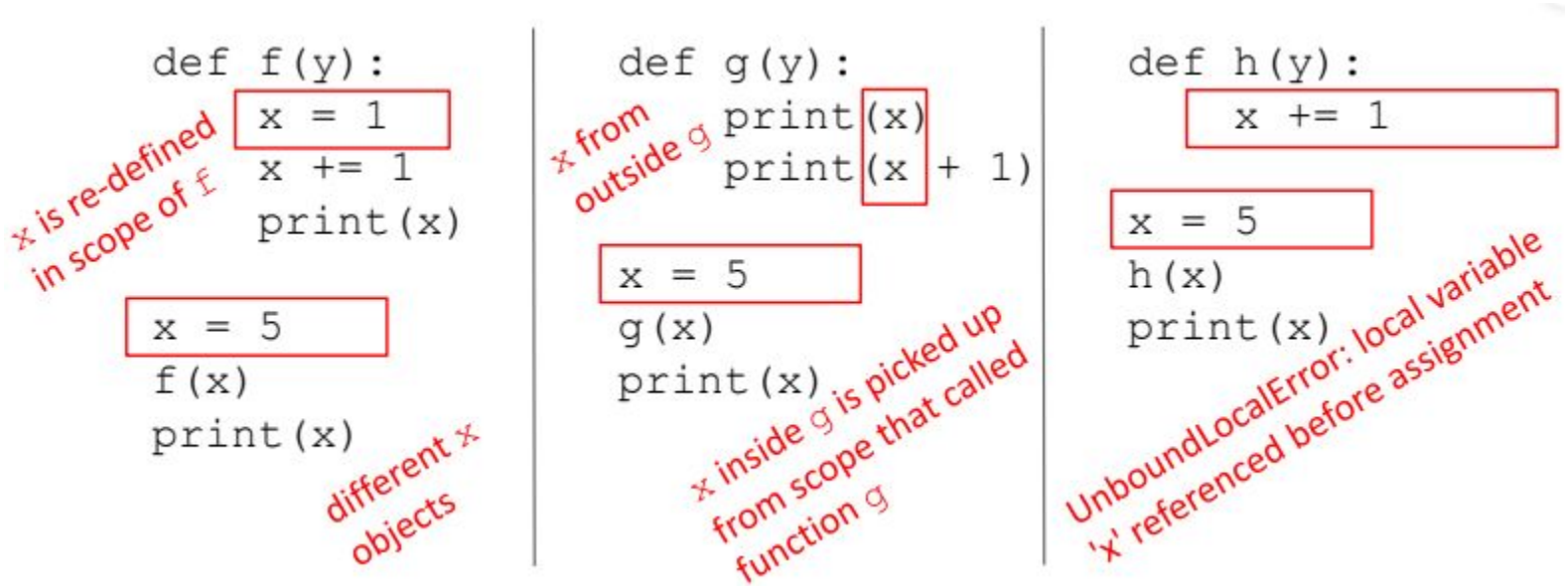
- Argumentos podem ser de qualquer tipo, inclusive outras subrotinas;

```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



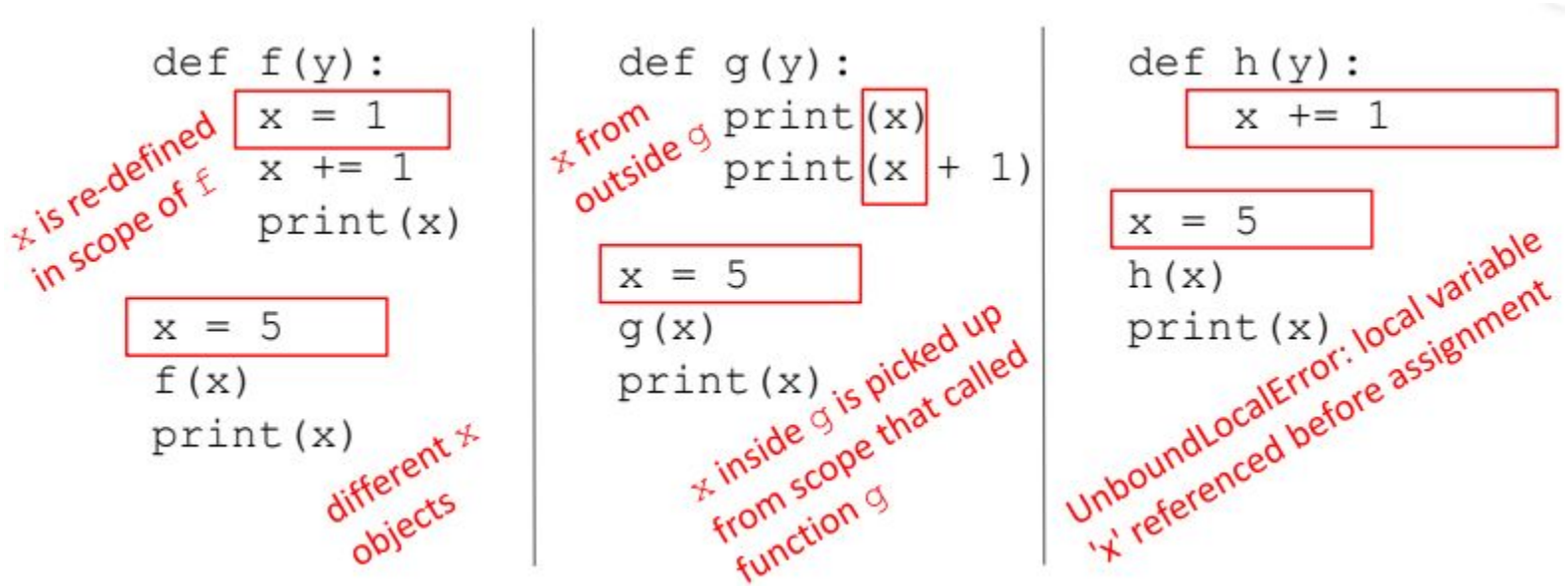
Exemplos de escopos

- Dentro de uma subrotina, uma variável definida fora do seu escopo pode ser acessada;
- Dentro de uma subrotina, não é possível modificar uma variável definida fora dela
  - É possível usando variáveis globais, mas isso não é recomendado.



Exemplos de escopos

- Dentro de uma subrotina, uma variável definida fora do seu escopo pode ser acessada;
- Dentro de uma subrotina, não é possível modificar uma variável definida fora dela
  - É possível usando variáveis globais, mas isso não é recomendado.



Closures

- Uma *closure* é uma função aninhada (uma função dentro de outra função) que tem acesso às variáveis do escopo externo, mesmo depois que a função externa termina de ser executada.
- Isso permite que a função aninhada "lembre" do valor das variáveis do escopo externo.

```python
def logger(msg):
    def log_message():
        print(f"LOG: {msg}")
    return log_message

log_hello = logger("Hello")
log_hello()  # Saída: LOG: Hello
```

```python
def uppercase(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        return result.upper()
    return wrapper

@uppercase
def say_hello(name):
    return f"hello, {name}"

print(say_hello("john"))  # Saída: HELLO, JOHN
```

Python - Mão na Massa!

1. Escreva um programa em que leia uma sequência de números e exiba a soma deles.
2. Escreva um programa que leia uma lista de números inteiros e exiba-os em ordem crescente;
3. Escreva um programa onde o computador escolhe um número aleatório entre 1 e 100, e o jogador tenta adivinhar. O programa deve dar dicas se o palpite é muito alto ou muito baixo e contar o número de tentativas.
4. Escreva um programa que valide senhas com base nos critérios: tenha pelo menos 8 caracteres, contendo letras maiúsculas, minúsculas, números e caracteres especiais.
5. Faça um programa que conte o número de vogais e consoantes em uma *string* fornecida pelo usuário;
6. Crie um programa que simule as operações básicas de um caixa eletrônico: verificar saldo, fazer depósitos e saques. O programa deve manter um saldo atual e registrar todas as transações.