# Introdução a Python

# IPL 2021

MIT BRAZIL

UNICAMP

# Funções personalizadas

- Depois de definida, podemos chamar nossa função personalizada como faríamos com qualquer função integrada ou importada!
- Quando definimos uma função personalizada, Python precisa lembrar de três informações:
  1. Nomes dos parâmetros, em ordem
  2. Código do corpo da função
  3. Quadro em que a função foi *definida*
- Mas por que precisamos da referência 3? Para isso, precisamos entender o que realmente acontece quando chamamos uma função personalizada em Python

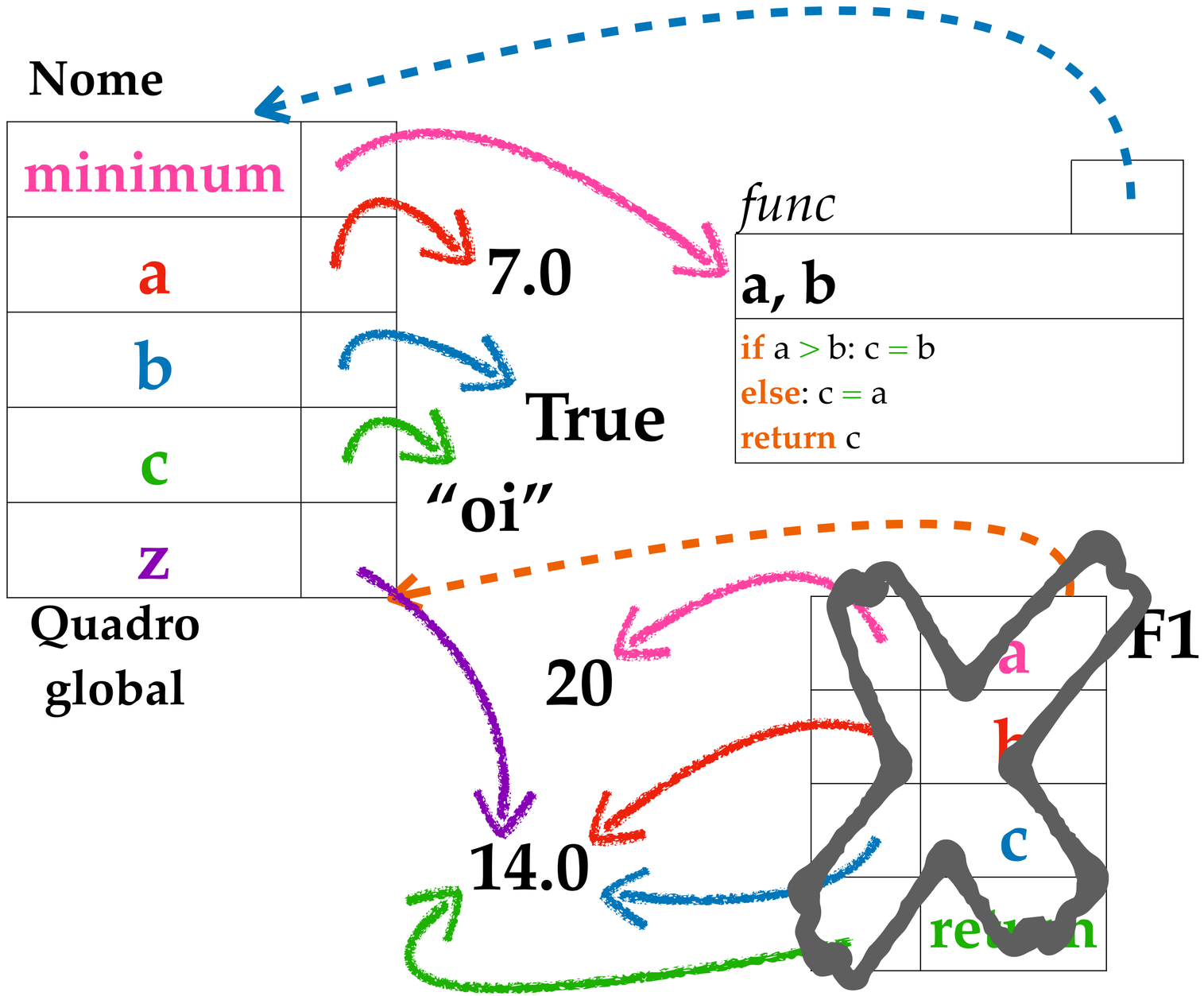
# Chamando funções personalizadas

- Quando *chamamos* uma função personalizada, o procedimento do Python segue os passos:

1. Avalia todos os argumentos da função no ambiente atual (em que a função é chamada)
2. Cria um novo ambiente cujo pai é o ambiente em que a função foi *definida* (chamado de **F1**)
   - O quadro é semelhante ao quadro global, mas as variáveis são **locais** (acessíveis apenas de dentro da função sendo chamada)
3. No novo ambiente, associa os parâmetros da função aos argumentos passados
4. Avalia o corpo da função dentro desse novo ambiente criado para execução
   - Python primeiro pesquisa uma variável no ambiente atual (**F1**). Se não encontrá-la, continuará procurando no quadro pai (quadro em que foi *definida*). Python continua procurando até atingir um quadro que não tem pai (geralmente quadro global) e desiste
   - Se o Python fizer novas atribuições, essas *também* serão feitas no quadro atual (aqui, **F1**)
   - **Variáveis dentro da função não afetam as variáveis fora da função!**
5. Determina o valor de retorno e o novo quadro é removido ao fim da execução

# Exemplo

```
1  a = 7.0
2  b = True
3  c = "oi"
4
5
6  def minimum(a, b):
7      if a > b:
8          c = b
9      else:
10         c = a
11     return c
12
13
14 z = minimum(12 + 2 ** 3, 3.0 + 80 // 7)
15 # 14.0
```



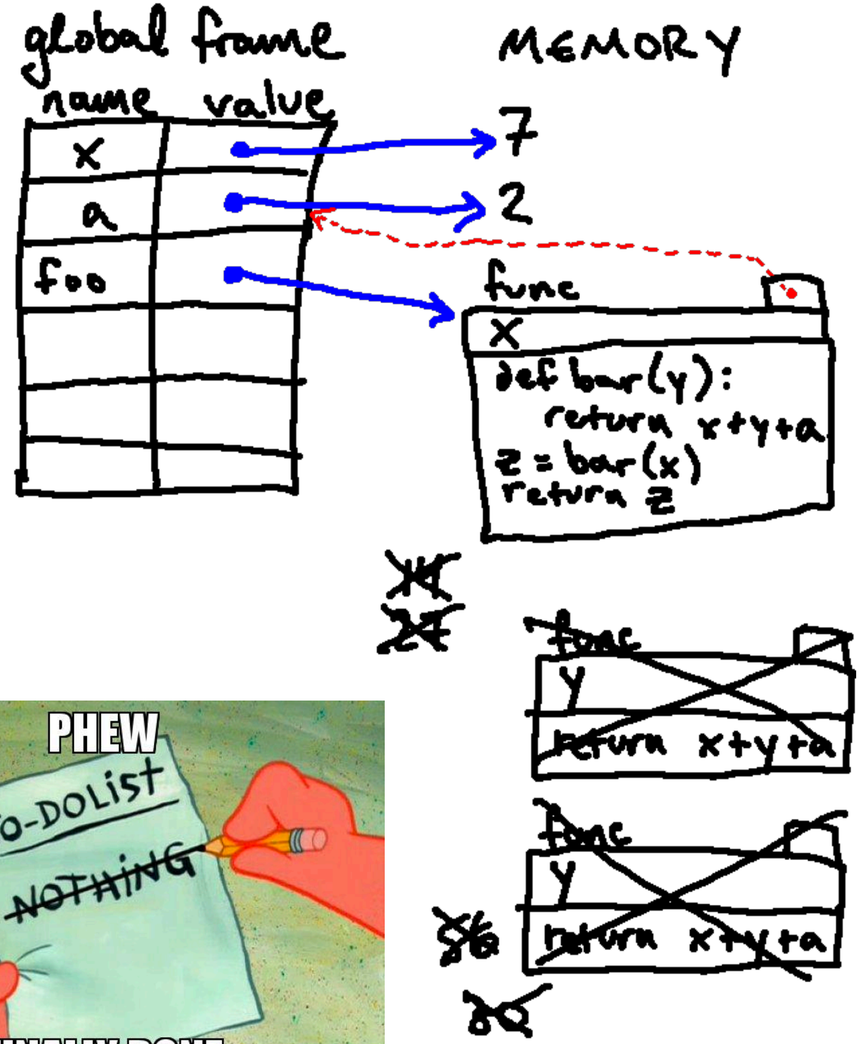



MIT BRAZIL

UNICAMP

# Definindo uma função dentro de outra

```
1   x = 7
2   a = 2
3
4
5   def foo(x):
6       def bar(y):
7           return x + y + a
8       z = bar(x)
9       return z
10
11
12  print(foo(14))
13  print(foo(27))
14
```









MIT BRAZIL

UNICAMP

# Poder de abstração

- A criação de um novo ambiente para execução parece complicada, mas nos fornece um recurso incrível: *as variáveis dentro do corpo da função não afetam as variáveis fora do corpo da função*
  - Podemos usar o mesmo nome de variável dentro e fora da função sem conflito (valor de fora não é alterado depois de chamar a função)
- Funções são então o método perfeito de **abstração**
  - Podemos usá-las como **caixa pretas**, conhecendo apenas seu comportamento *end-to-end*
  - Não precisamos nos preocupar com *como* o comportamento foi implementado no corpo da função, nomes de variáveis usados, …

```
1   x = 20
2
3
4   def foo(x):
5       y = "banana"
6       return str(x) * 5 + y
7
8
9   z = foo(x)
10
11  print(x)  # 20
12  # print(y)  daria um NameError
13  print(z)  # "2020202020banana"
14
```

# Funções são objetos

- Em Python, funções são objetos como qualquer outro, o que significa que podemos:
  - Atribuir funções a variáveis além do nome da definição
  - Conter funções em sequências (e.g. listas, tuplas, sets)
  - Funções podem ser definidas dentro de outras funções e podem ser retornadas por outras funções
  - Podem ser passadas como argumentos para outras funções: permite passar *comportamento* no programa (função influenciando outra função)

**map(f, x)** Aplica a função *f* a cada elemento da sequência *x*, substituindo *x* for *f(x)* na nova sequência

**filter(f, x)** Filtra a sequência *x*, mantendo apenas os elementos *i* que têm *f(i)* **True** na nova sequência

**lambda** Criando objetos função como outros objetos

```
def add(x, y):
    return x + y

add = lambda x, y: x + y
```

```python
def gritar(texto: str):
    return texto.upper() + "!"


def sussurrar(texto: str):
    return texto.lower() + "..."


berrar = gritar
x = [berrar, gritar, lambda x, y: x + y*10]
y = tuple(x)

def obter_func_falar(volume):
    if volume > 0.5:
        return gritar
    return sussurrar


gritaria = obter_func_falar(0.7)
sussurro = obter_func_falar(0.3)

print(gritaria("Oi, tudo bem"))
# OI, TUDO BEM!
print(sussurro("Oi, tudo bem"))
# oi, tudo bem...

```

definição de função "sem nome": **lambda**



Recurso: https://dbader.org/blog/python-first-class-functions