Introdução a Python

IPL 2021

MIT BRAZIL

UNICAMP

Operadores extras de listas

x in y — Retorna **True** se *x* é um elemento da sequência *y*, **False** do contrário

x.index(y) — Retorna o índice da primeira ocorrência de *y* em *x*. Erro se não presente

x.pop(i) — Remove e retorna o elemento no índice *i* da lista *x*. Se passado sem *i*, remove último elemento da lista

x.insert(i, v) — Adiciona o objeto *v* na posição de índice *i* de *x*, de modo que *x[i]* retorna *v*

x.extend(y) — Adiciona cada elemento da sequência *y* ao final de *x*, na ordem original

x.count(y) — Retorna quantas vezes o elemento *y* aparece na sequência *x*

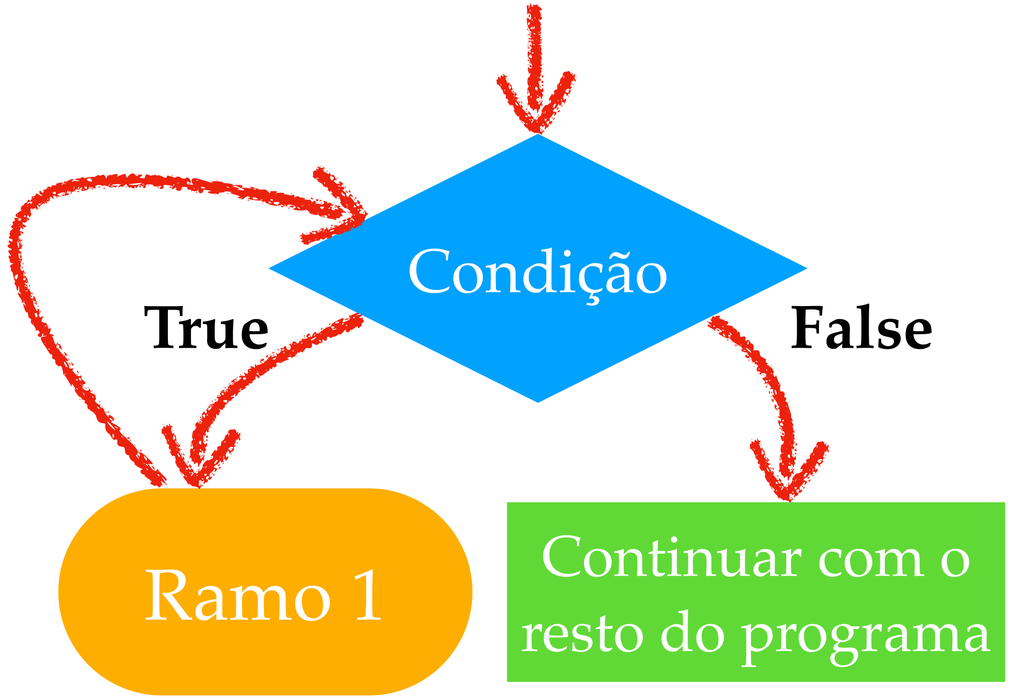
x.remove(y) — Remove o elemento *y* da lista *x* (por valor, não por índice)

x.sort() — Modifica a lista *x* para que os números estejam em ordem crescente

```python
x = [1, 2, 7, [4, 5, 6]]

print(1 in x)  # True: 1 é um elemento de x
print(4 in x)  # False: [4, 5, 6] é um
# elemento de x, mas 4 sozinho não é

print(x.index(2))  # 1 (segundo elemento)
print(x.pop())  # [4, 5, 6]
print(x)  # [1, 2, 7]

x.insert(2, 3)
print(x)  # [1, 2, 3, 7]

x.extend([7, 7, 7])
print(x)  # [1, 2, 3, 7, 7, 7, 7]
print(x.count(7))  # 4

print(x.pop(2))  # 3
print(x)  # [1, 2, 7, 7, 7, 7]

print(x.remove(2))  # None
print(x)  # [1, 7, 7, 7, 7]
```

While loops

- **for** loops implementam iteração: repetir um bloco de código uma certa quantidade de vezes
- **while** loops repetem um bloco de código *até que alguma condição seja satisfeita*
  - útil se não sabemos quantas vezes precisaremos repetir um pedaço de código
- Muito parecidos com a estrutura de uma condicional: checa uma condição para decidir se o corpo deve ser executado
  - diferença: condicional executa o corpo uma vez e segue em frente; **while** loops até a condição não ser mais verdadeira



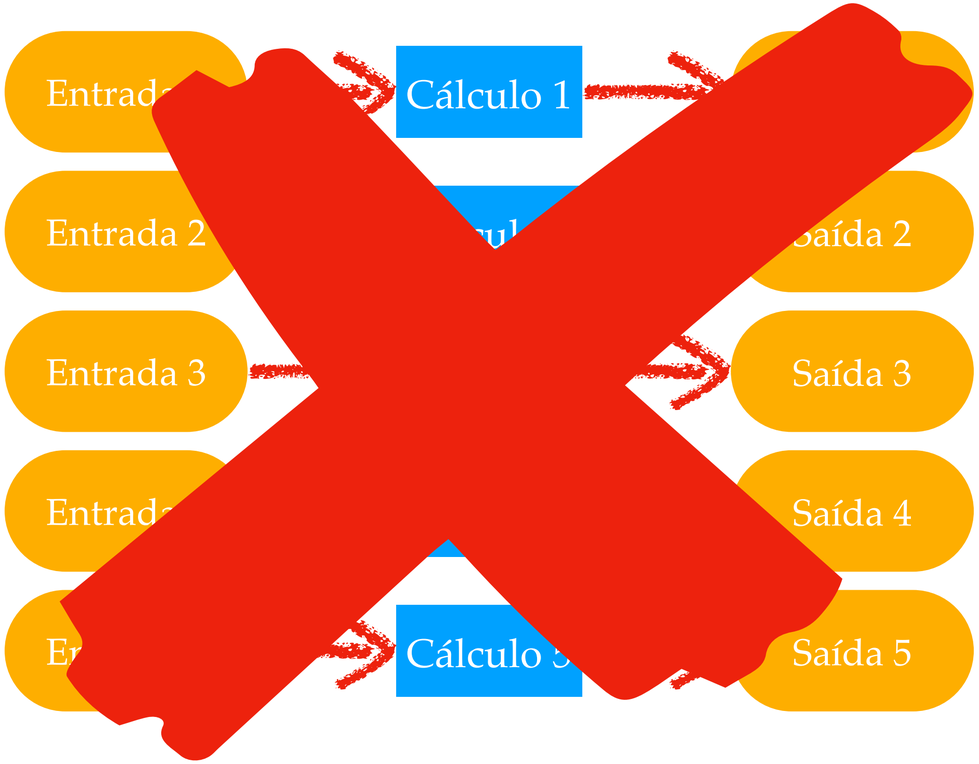

```
1    x = 10
2    while x > 2:
3        # ramo 1
4        print(x)
5        x -= 2
6    # imprime 10, 8, 6, 4
7
```


MIT BRAZIL

UNICAMP

Generalização

- Até agora nossos programas têm sido limitados: eles só funcionam para os valores específicos que definimos no começo
- Faz sentido imaginar um programa mais geral que resolve o mesmo problema para vários valores iniciais diferentes
- Copiar e colar não é uma boa solução:
  - precisaríamos mudar o nome das variáveis para cada seção copiada
  - se encontrarmos um erro, teríamos que voltar e corrigi-lo em cada uma das cópias
- Precisamos *generalizar a noção da computação*



Fácil de manter, corrigir, ou modificar!

MIT BRAZIL

UNICAMP

Funções e *built-ins*

- Objeto que representa uma *computação abstrata*
  - Basicamente um programa pequeno em si mesmo, que executa uma tarefa específica
- Uma sequência generalizada de instruções que Python pode avaliar para calcular um resultado a partir de entradas específicas passadas
- Python tem várias funções integradas

abs	all	any	**bool**	complex	**dict**	divmod
enumerate	eval	exec	filter	**float**	frozenset	globals
hash	id	input	**int**	isinstance	len	**list**
map	max	min	print	range	round	**set**
sorted	**str**	sum	**tuple**	type	zip	__import__

Todas as builtins: https://docs.python.org/pt-br/3/library/functions.html

all(x) Retorna **True** se todos os elementos da seq. *x* são **True**

any(x) Retorna **True** se pelo menos um elemento da seq. *x* é **True**

input(m) Mostra mensagem *m*, salva entrada do usuário como string

isinstance(x, t) Retorna **True** se *x* for do tipo *t*, **False** do contrário

round(x, n) Arredonda *x* para *n* casas decimais. Sem *n* passado, arredonda *x* para um inteiro

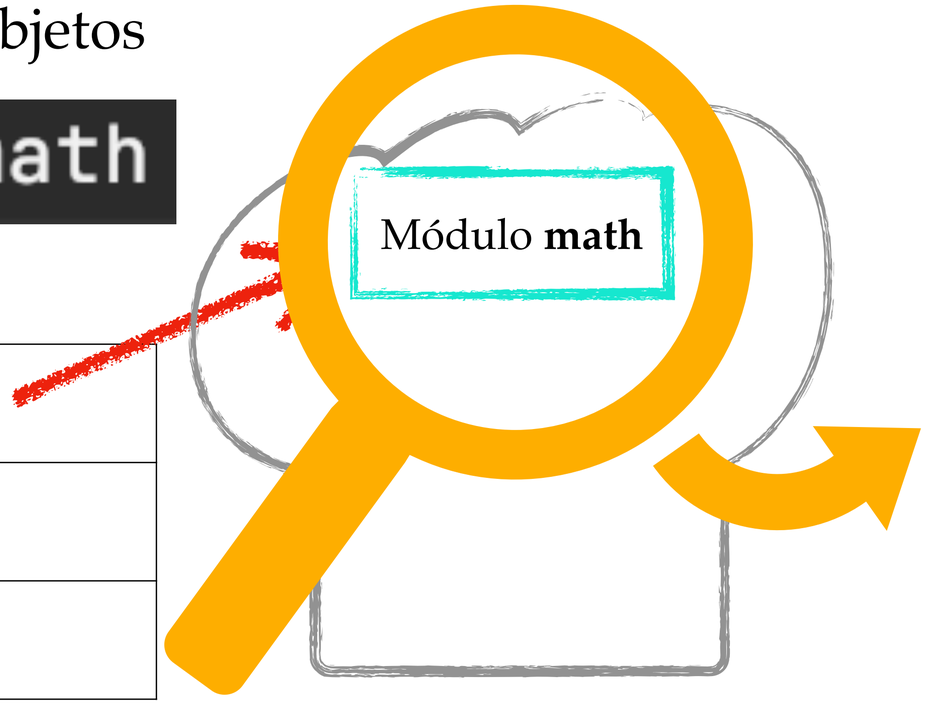
sorted(x) Retorna uma cópia da sequência *x* em ordem (e.g. crescente de números)

zip(x, y) Retorna objetos correspondentes de *x* e *y* como pares (e.g. para um for loop)

Importações / *dot notation*

- Uma grande vantagem de Python é a quantidade de bibliotecas de outras funções e constantes que podem ser adicionadas e usadas em código
- É necessário *importar módulos (bibliotecas)* para acessar esses objetos

```
import math
```



Nome

math	

Quadro global

Memória

Nome

e	
pi	
tau	
inf	
…	

Quadro do módulo *math*

math.pi

Python começa procurando **math** no quadro global e acha o módulo

Python então procura *dentro* desse módulo por **pi** e encontra o objeto float

math.e leva a este objeto exatamente!

2.71828…

3.14159…

6.28318…

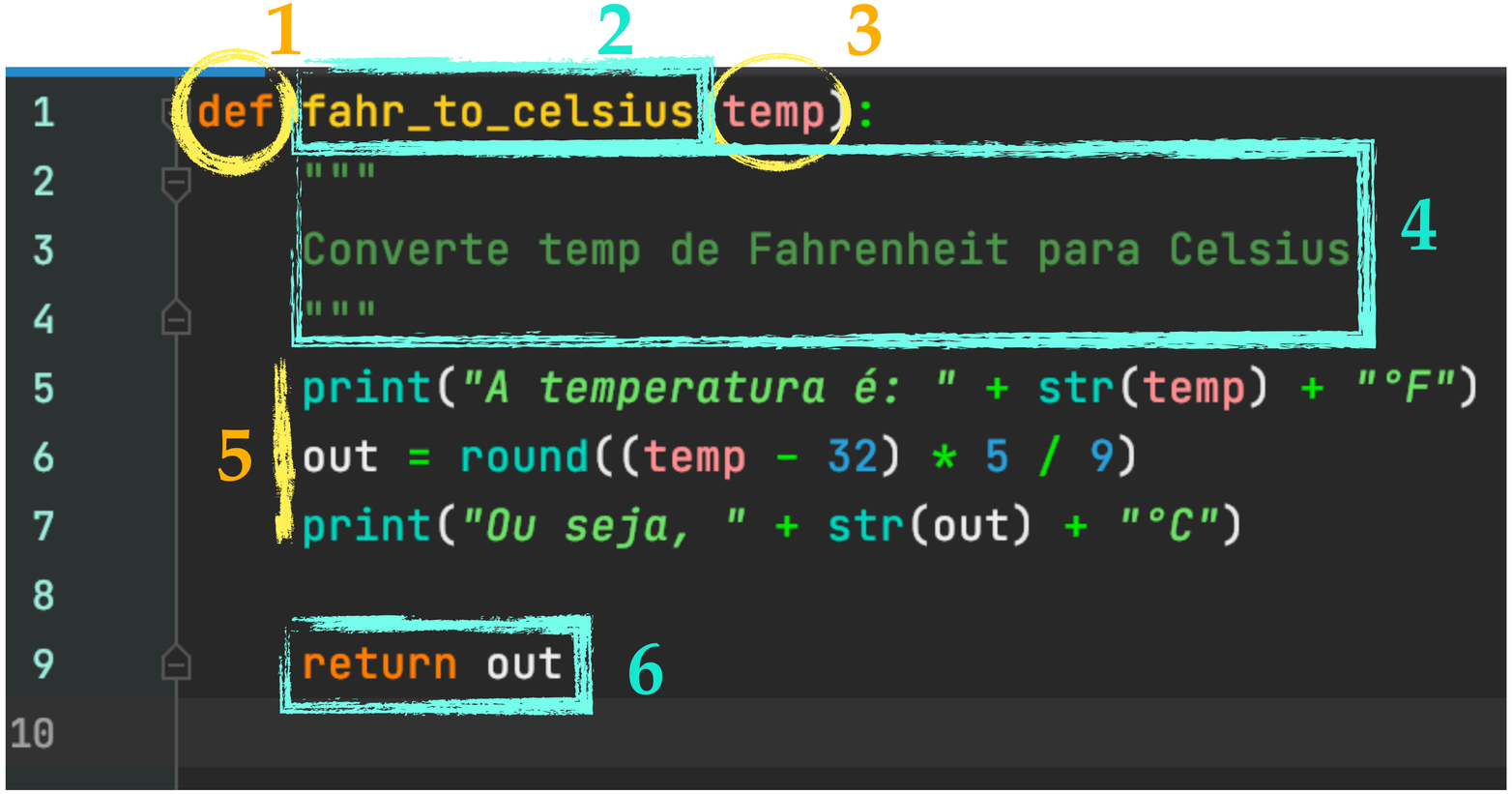
objeto inf

O módulo também contém várias funções úteis!

Documentação **math**: https://docs.python.org/pt-br/3/library/math.html

Funções personalizadas

- Usar funções integradas ou importadas é bom, mas *poder real* vem de definir nossas próprias funções
- Para definir funções, usamos a palavra-chave **def**




```python
def fahr_to_celsius(temp):
    """
    Converte temp de Fahrenheit para Celsius
    """
    print("A temperatura é: " + str(temp) + "°F")
    out = round((temp - 32) * 5 / 9)
    print("Ou seja, " + str(out) + "°C")

    return out

```

1. Palavra-chave **def**
2. *Nome* da função
3. *Argumentos* da função: repr. abstrata das entradas
4. *Docstring* (opcional): descreve o que a função faz
5. *Corpo* da função
6. *Return* statement: determina o valor de retorno

MIT BRAZIL

UNICAMP

Funções personalizadas

- Com essa estrutura, Python cria um objeto função na memória e associa o nome dado à função ao objeto no quadro atual
- A instrução apenas *define* a função!
  - O código do corpo não é executado ainda
- Funções precisam salvar três informações:
  1. Nomes dos parâmetros, em ordem
  2. Código do corpo da função
  3. Quadro em que a função foi definida
- Agora podemos chamar nossa função como faríamos com qualquer função integrada ou importada!