

PYTHON



For...

While

professor



Lázaro
Santos



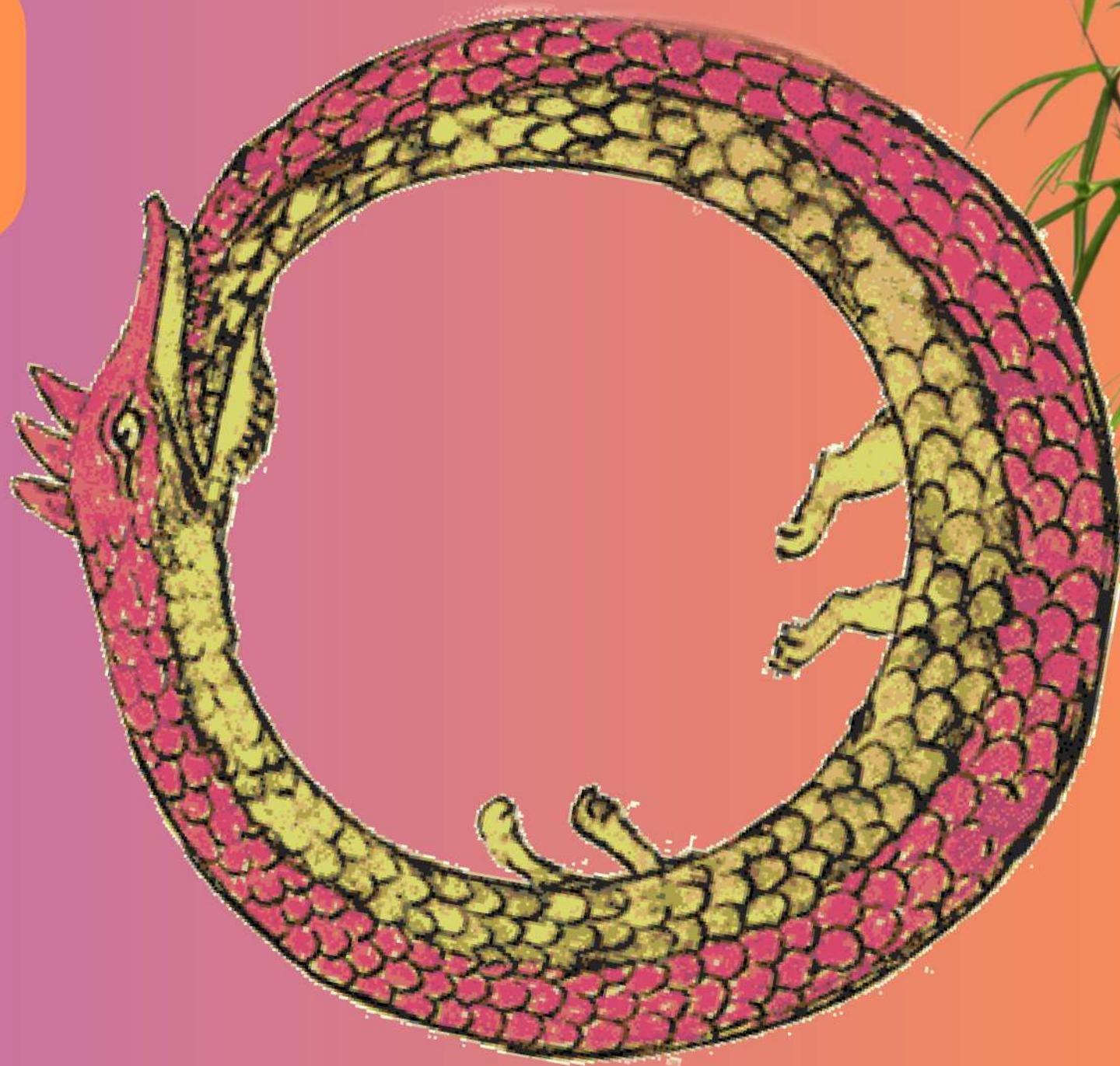
Unile



WHILE

Repetições representam a base de vários programas.

São utilizadas para executar a mesma parte de um programa várias vezes, normalmente dependendo de uma condição.





WHILE

```
x=1  
  
print(x)  
  
x=2  
  
print(x)  
  
x=3  
  
print(x)
```

imprimindo
números de 1 a 3

Início

$x = 1$

Enquanto
 $(x \leq 3)$

NÃO

Fim

Fluxograma de um laço de
repetição while

$x = x + 1$

Exibe x

x é o começo

3 é o fim

1 é o incremento



WHILE

imprimindo
números de 1 a 3
com while

x=1 ①

while x<=3: ②

print(x) ③

x = x + 1 ④



EXERCÍCIOS

Exercício 5.1 Modifique o programa para exibir os números de 1 a 100.

Exercício 5.2 Modifique o programa para exibir os números de 50 a 100.

Exercício 5.3 Faça um programa para escrever a contagem regressiva do lançamento de um foguete. O programa deve imprimir 10, 9, 8, ..., 1, 0 e Fogo! na tela.



WHILE

imprimindo números pares

```
fim=int(input("Digite o último número a imprimir:"))

x = 0 ①

while x <= fim:

    if x % 2 == 0: ②

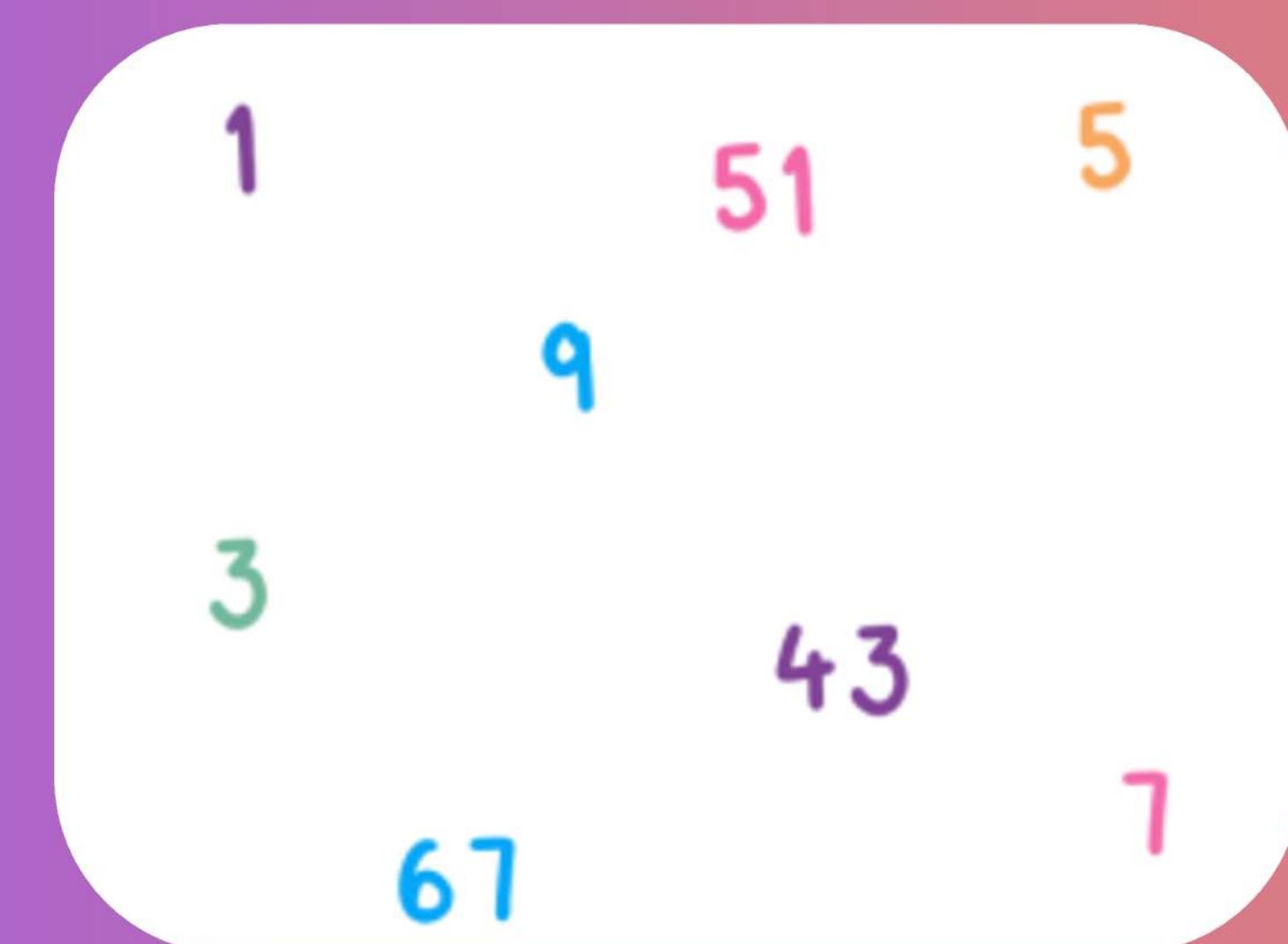
        print(x) ③

    x = x + 1
```



EXERCÍCIO

Exercício 5.4 Modifique o programa anterior para imprimir de 1 até o número digitado pelo usuário, mas, dessa vez, apenas os números ímpares.





WHILE

imprimindo uma tabuada de adição

```
n = int(input("Tabuada de:"))

x = 1

while x <= 10:

    print(n+x)

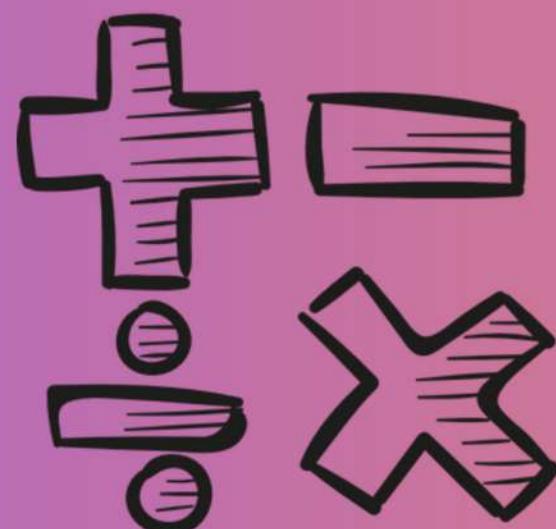
    x=x+1
```



EXERCÍCIOS

Exercício 5.6 Altere o programa anterior para exibir os resultados no mesmo formato de uma tabuada: $2 \times 1 = 2$, $2 \times 2 = 4$, ...

Exercício 5.7 Modifique o programa anterior de forma que o usuário também digite o início e o fim da tabuada, em vez de começar com 1 e 10.





WHILE... BREAK

A instrução **break** é utilizada para interromper a execução de while independentemente do valor atual de sua condição.





WHILE... BREAK

interrompendo a repetição

```
s=0  
while True: ❶  
    v=int(input("Digite um número a somar ou 0 para sair:"))  
    if v==0:  
        break ❷  
    s = s+v ❸  
    print(s) ❹
```



WHILE... BREAK

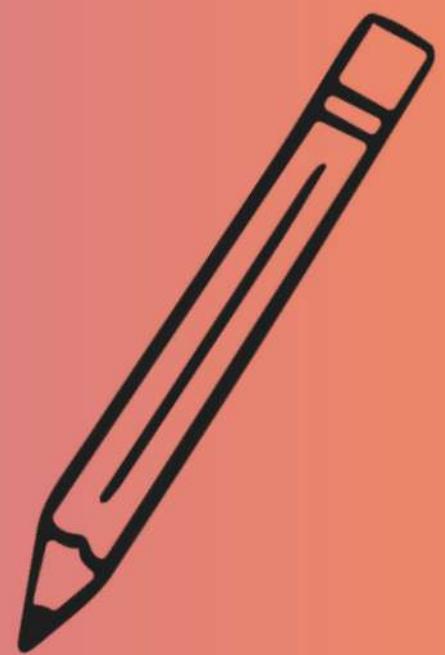
adicionando
números em
uma lista

```
L=[]  
while True:  
    n=int(input("Digite um número (0 sai):"))  
    if n == 0:  
        break  
    L.append(n)  
  
x=0  
while x < len(L):  
    print(L[x])  
    x=x+1
```



EXERCÍCIO

Exercício 5.14 Escreva um programa que leia números inteiros do teclado. O programa deve ler os números até que o usuário digite 0 (zero). No final da execução, exiba a quantidade de números digitados, assim como a soma e a média aritmética.





WHILE... BREAK

contador de cédulas

```
valor=int(input("Digite o valor a pagar:"))
cédulas=0
atual=50
apagar=valor
while True:
    if atual<=apagar:
        apagar-=atual
        cédulas+=1
```

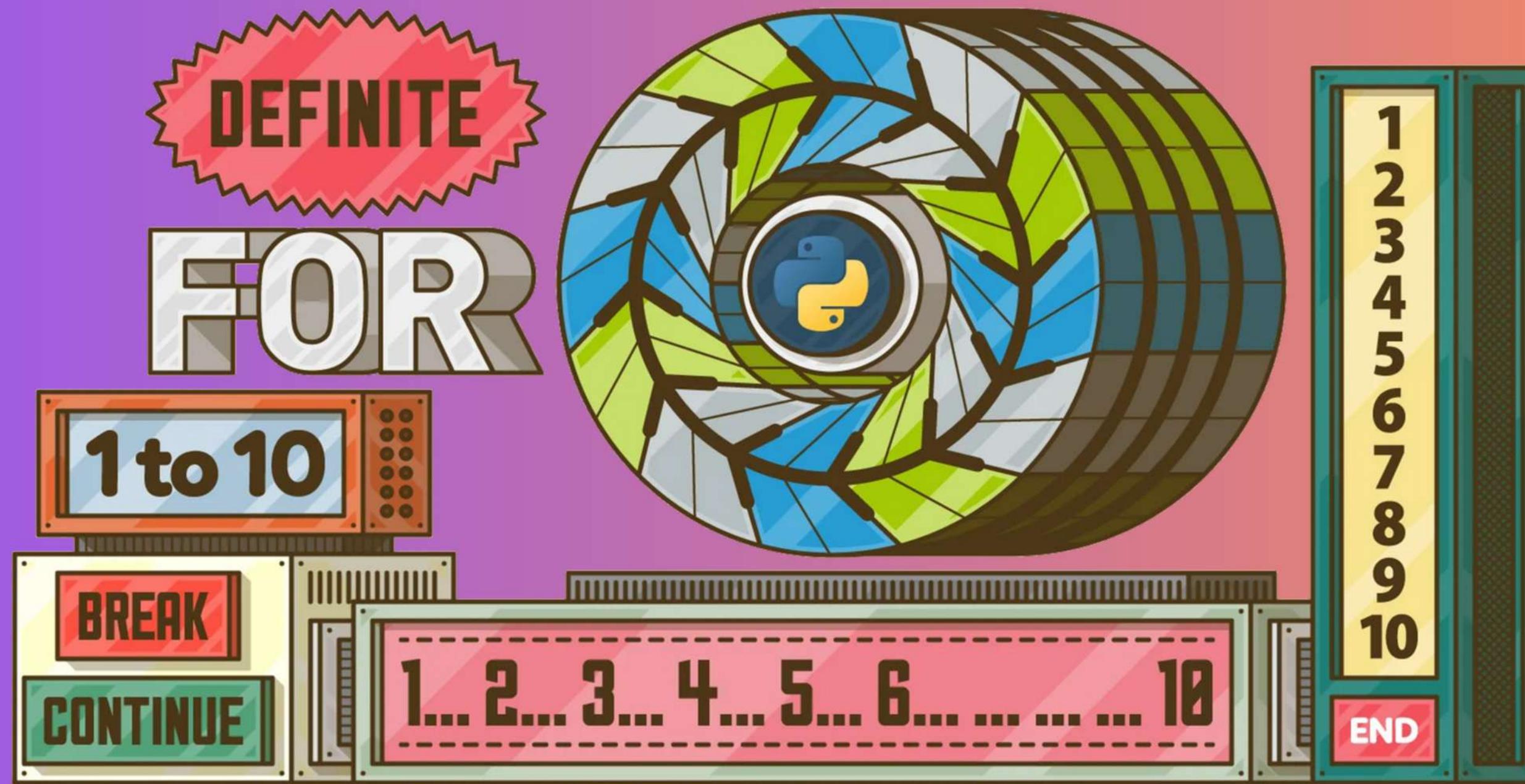
```
else:
    print("%d cédula(s) de R$%d" % (cédulas, atual))
    if apagar == 0:
        break
    if atual == 50:
        atual = 20
    elif atual == 20:
        atual = 10
    elif atual == 10:
        atual = 5
    elif atual == 5:
        atual = 1
    cédulas = 0
```

For



FOR

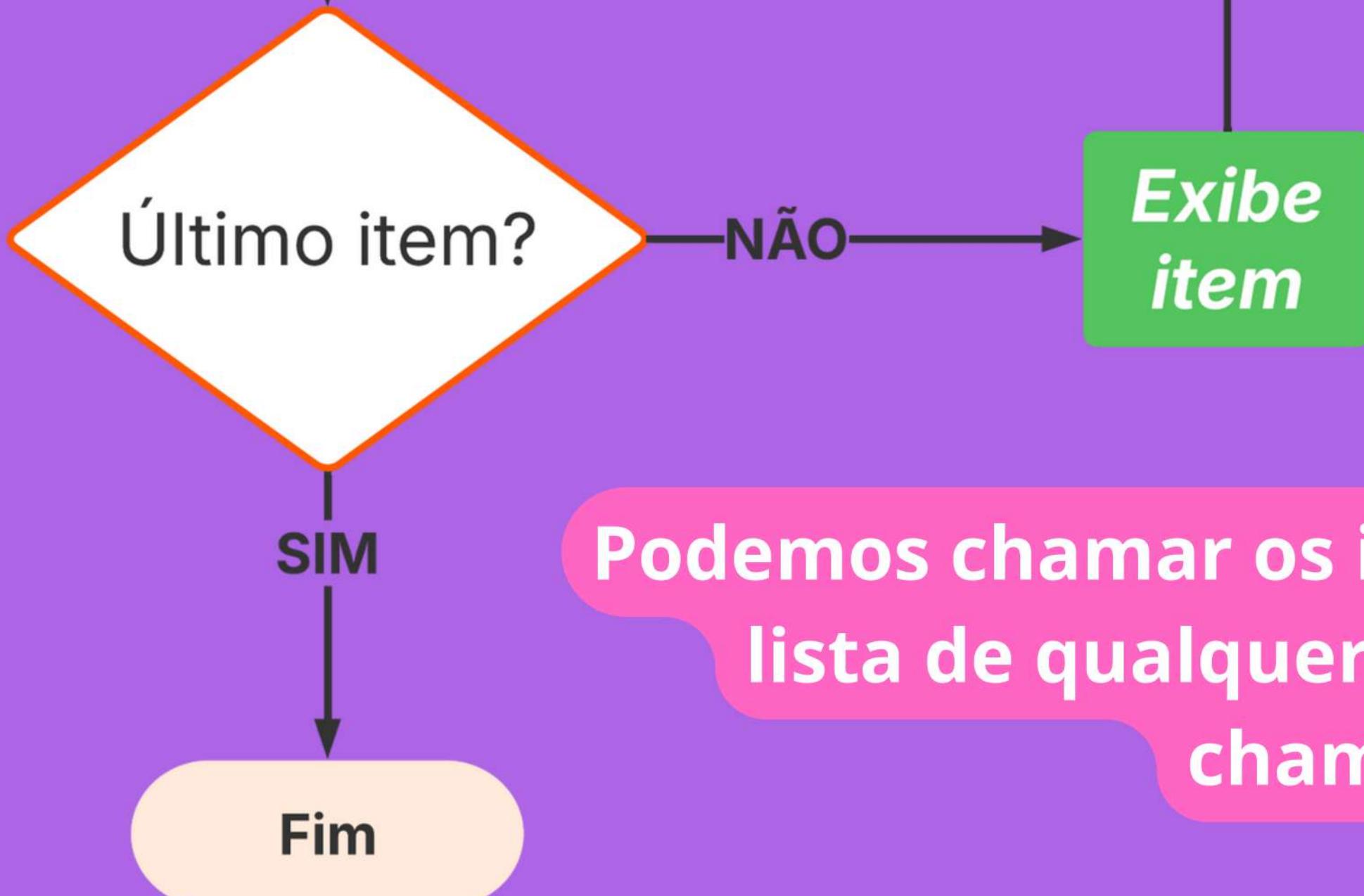
Python apresenta uma estrutura de repetição especialmente projetada para percorrer listas. A instrução for funciona de forma parecida a while, mas a cada repetição utiliza um elemento diferente da lista.



Início

$L = [8, 9, 15]$

Fluxograma de um laço de
repetição `for`



Podemos chamar os itens de uma
lista de qualquer coisa, então
chamaremos de *e*



FOR

imprimindo itens de
uma lista L

```
L=[8,9,15]
```

```
for e in L: ①
```

```
    print(e) ②
```

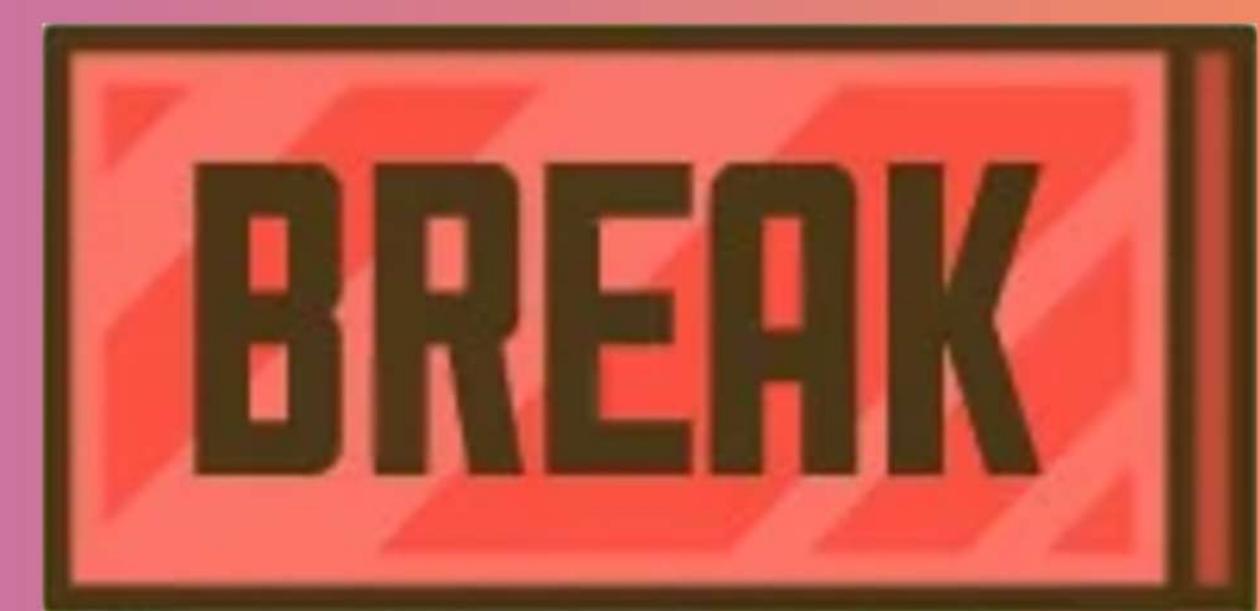


FOR

Embora a instrução for facilite nosso trabalho, ela não substitui completamente while.

while é indicado para repetições nas quais não sabemos ainda quantas vezes vamos repetir ou onde manipulamos os índices de forma não sequencial.

Vale lembrar que a instrução break também interrompe o for.





FOR

imprimindo uma lista de strings, letra a letra

```
L= ["maçãs", "peras", "kiwis"]  
for s in L:  
    for letra in s:  
        print(letra)
```



FOR

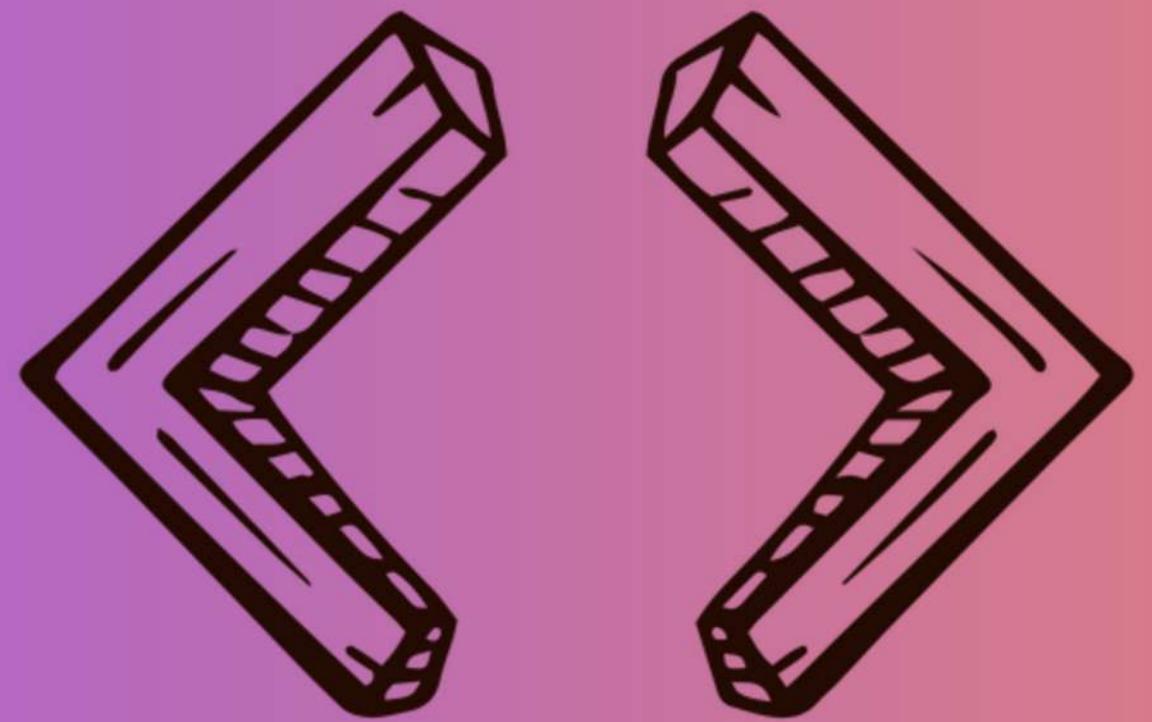
imprimindo o maior
elemento de uma lista

```
L=[1,7,2,4]  
máximo=L[0] ①  
for e in L:  
    if e > máximo:  
        máximo = e  
print(máximo)
```



EXERCÍCIO

Exercício 6.12 Altere o programa anterior de forma a imprimir o menor elemento da lista.



Rango



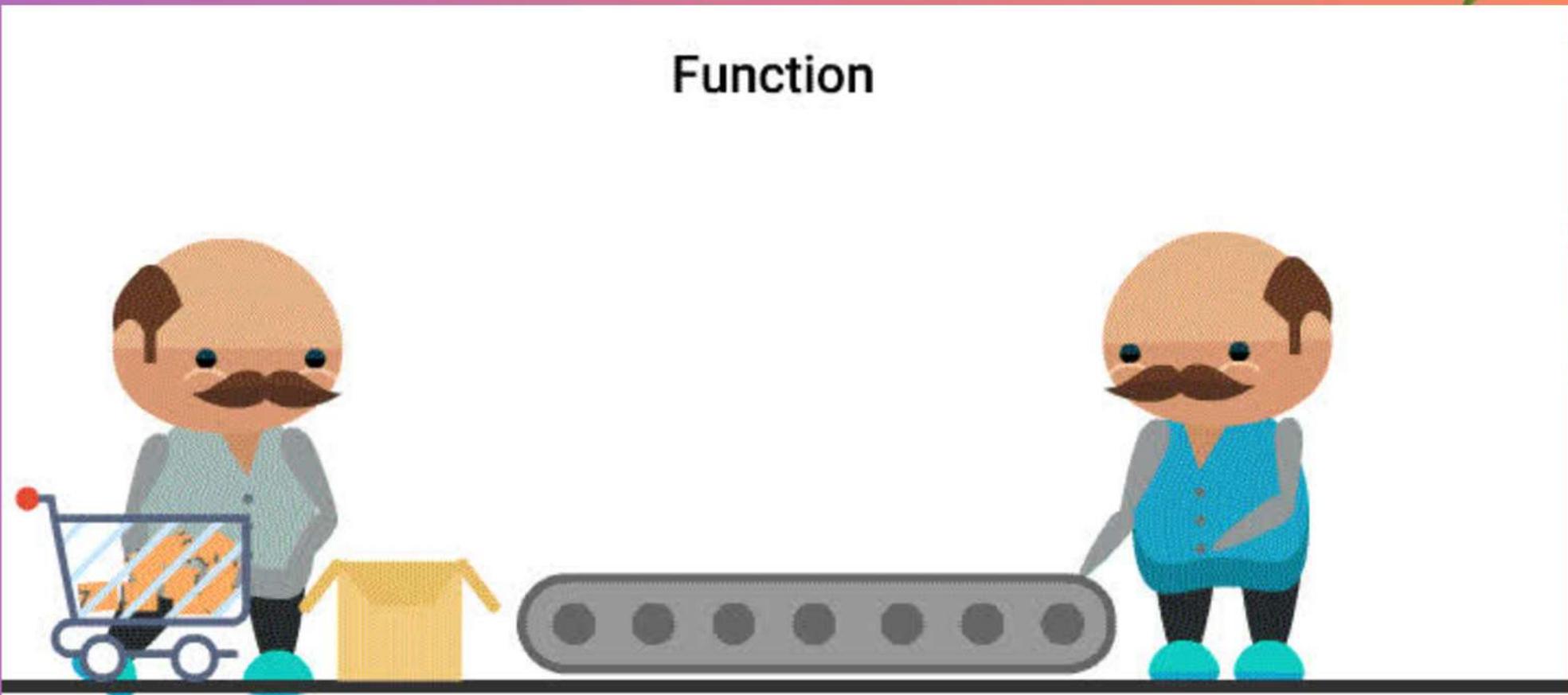


RANGE

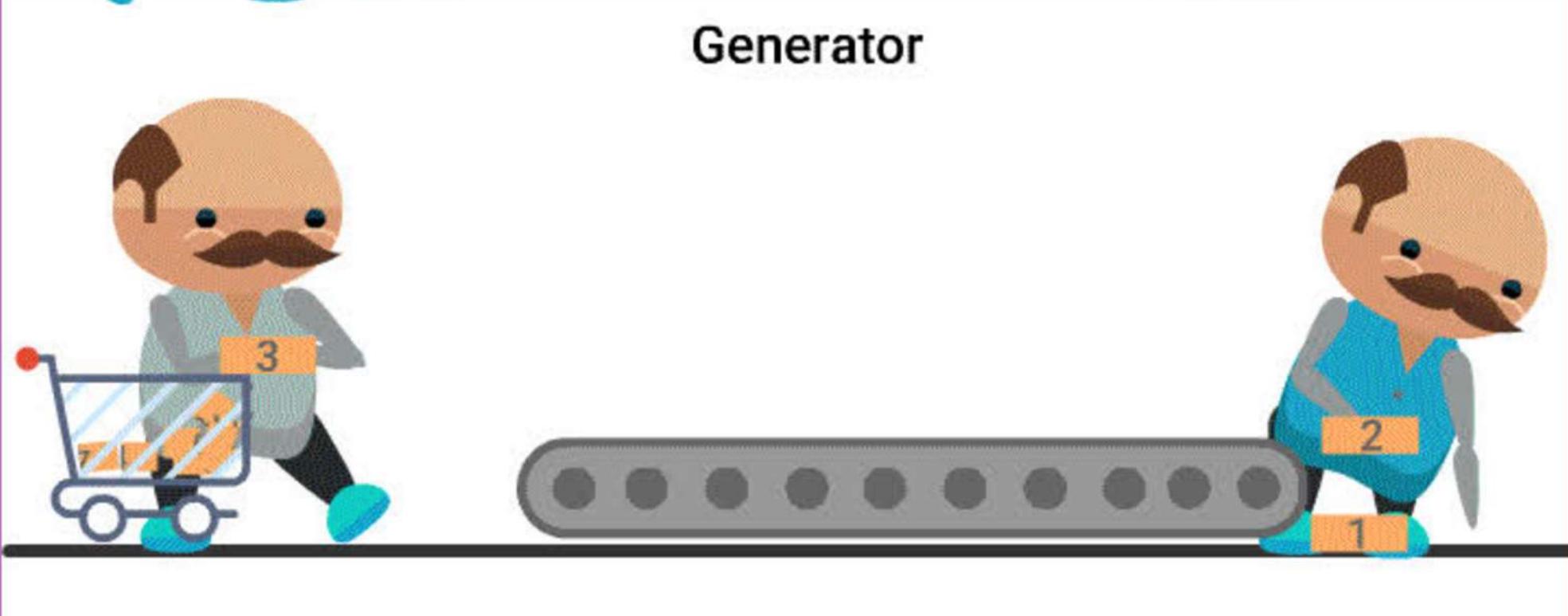
Podemos utilizar a função range para gerar listas simples.

A função range não retorna uma lista propriamente dita, mas um gerador ou *generator*.

Function



Generator





RANGE

imprimindo de 0 a 9 com a
função range()

```
for v in range(10):  
    print(v)
```



RANGE

A vantagem de utilizarmos a função range é gerar listas eficientemente, como mostrado no exemplo, sem precisar escrever muitos valores no programa. Com a mesma função range, podemos também indicar qual é o primeiro número a gerar. Para isso, utilizaremos dois parâmetros: início e fim.

```
for v in range(5, 8):  
    print(v)
```



RANGE

utilizando a função range() com intervalo e com saltos

```
for t in range(3,33,3):  
    print(t, end=" ")
```

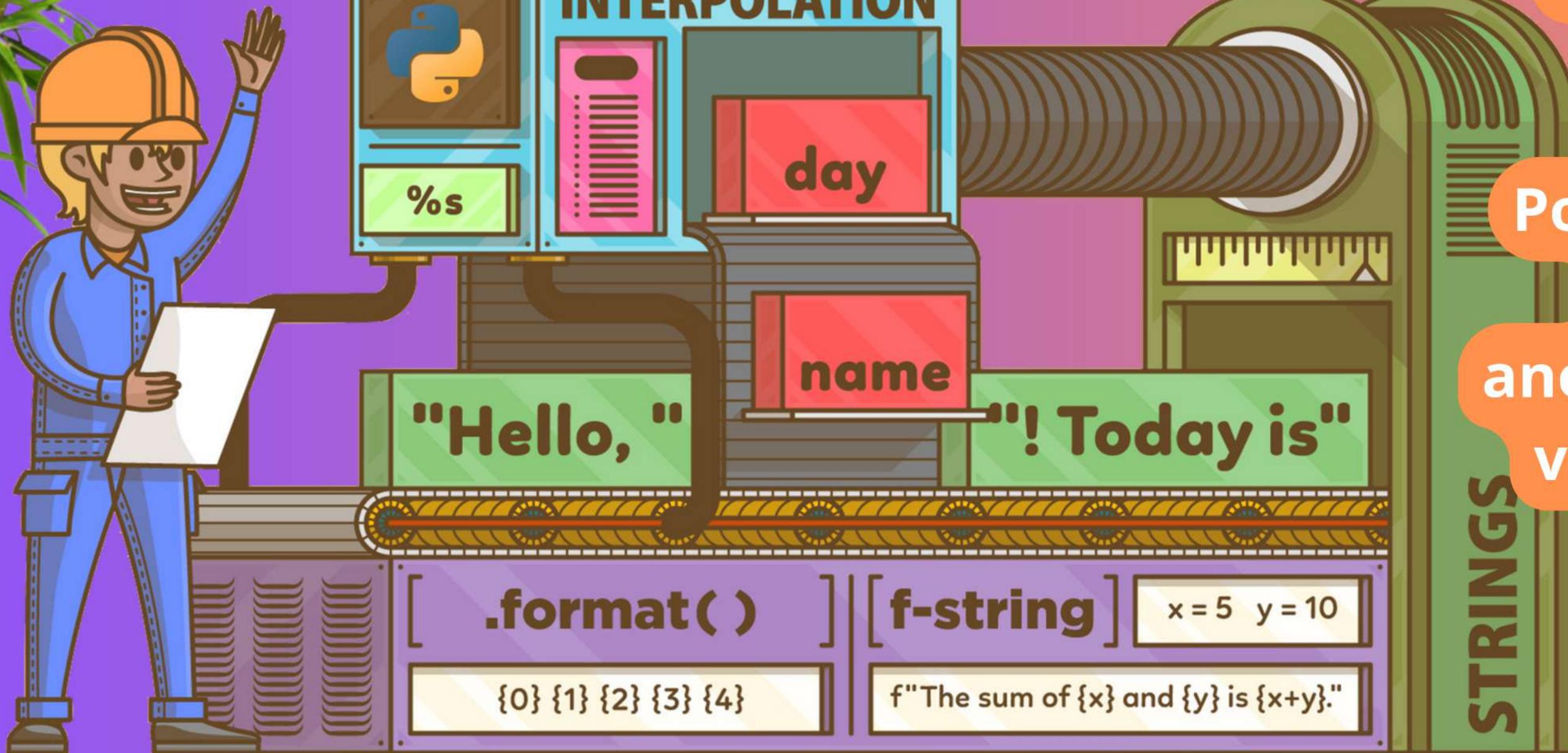
The background features a horizontal gradient from purple on the left to orange on the right, representing a sunset or sunrise. Overlaid on this gradient are various tropical elements: green palm fronds and leaves in the foreground and middle ground, a brown vine with small green leaves arching across the top, and a winding brown path or riverbed visible in the lower right quadrant.

Composição



COMPOSIÇÃO

Juntar várias strings para construir uma mensagem nem sempre é prático.



Por exemplo, exibir que "João tem X anos", onde X é uma variável numérica.



COMPOSIÇÃO

Usando a composição de strings do Python, podemos escrever de forma simples e clara:

```
"João tem %d anos" % X
```

o símbolo de % foi utilizado para indicar a composição da string anterior com o conteúdo da variável X.

O %d dentro da primeira string é o que chamamos de *placeholder* (marcador de posição). O *placeholder* indica que naquela posição estaremos colocando um valor inteiro, daí o %d.



COMPOSIÇÃO

Ou ainda, “Meu nome é A e minha idade é B, moro no endereço C, número D, bairro E [...]” onde A, B, C, D e E são variáveis...

Python suporta diversas operações com *placeholders*.

Marcador	Tipo
%d	Números inteiros
%s	Strings
%f	Números decimais



COMPOSIÇÃO COM INT

Observe que adicionamos 03 entre o % e o d. Se você precisar apenas que o número ocupe três posições, mas não desejar zeros à esquerda, basta retirar o zero e utilizar "%3d" % X.

```
>>> idade = 22  
>>> print("%d" % idade)  
[22]  
>>> print("%03d" % idade)  
[022]  
>>> print("%3d" % idade)  
[ 22]  
>>> print("%-3d" % idade)  
[22 ]
```



COMPOSIÇÃO COM FLOAT

```
>>> print("%5f" % 5)  
5.000000  
>>> print("%5.2f" % 5)  
5.00  
>>> print("%10.5f" % 5)  
5.00000
```

Podemos utilizar dois valores entre o símbolo de % e a letra f.

O primeiro indica o tamanho total em caracteres a reservar; e o segundo, o número de casas decimais.



COMPOSIÇÃO COM STRING

```
>>> nome = "João"
```

```
>>> idade = 22
```

```
>>> grana = 51.34
```

```
>>> print("%s tem %d anos e R$%f no bolso." % (nome, idade, grana))
```

```
João tem 22 anos e R$51.340000 no bolso.
```

```
>>> print("%12s tem %03d anos e R$%5.2f no bolso." % (nome, idade, grana))
```

```
João tem 022 anos e R$51.34 no bolso.
```

```
>>> print("%-12s tem %-3d anos e R$%-5.2f no bolso." % (nome, idade, grana))
```

```
João      tem 22  anos e R$51.34 no bolso.
```

Lembrando: também é possível utilizar posições em strings ao definir um número positivo ou negativo antes do *placeholder*.

The background features a tropical sunset with a gradient from purple to orange. Palm fronds and other tropical foliage frame the scene at the top and bottom.

Visionários



DICIONÁRIOS

Dicionários são estruturas de dados similares às listas, mas com propriedades de acesso diferentes. Um dicionário é composto por um conjunto de chaves e valores, e esses são relacionados





DICIONÁRIOS

criamos dicionários utilizando chaves {}

Cada elemento do dicionário é uma combinação de chave e valor.

```
tabela = { "Alface": 0.45,  
           "Batata": 1.20,  
           "Tomate": 2.30,  
           "Feijão": 1.50 }
```

Produto	Preço
Alface	R\$ 0,45
Batata	R\$ 1,20
Tomate	R\$ 2,30
Feijão	R\$ 1,50



DICIONÁRIOS

Um dicionário é acessado por suas **chaves**. Diferentemente de listas, onde o índice é um número, dicionários utilizam suas chaves como índice.

```
>>> tabela = { "Alface": 0.45,  
...  
    "Batata": 1.20,  
    "Tomate": 2.30,  
    "Feijão": 1.50 }  
  
>>> print(tabela["Tomate"]) ❶  
2.3
```

Produto	Preço
Alface	R\$ 0,45
Batata	R\$ 1,20
Tomate	R\$ 2,30
Feijão	R\$ 1,50



DICIONÁRIOS

Quando atribuímos um valor a uma chave,
duas coisas podem ocorrer:

1. Se a chave já existe: o valor associado é alterado para o novo valor.
2. Se a chave não existe: a nova chave será adicionada ao dicionário.

```
>>> tabela["Cebola"] = 1.20 ❸
```

```
>>> print(tabela)
```

```
{'Batata': 1.2, 'Alface': 0.45, 'Tomate': 2.5, 'Cebola': 1.2, 'Feijão': 1.5}
```



DICIONÁRIOS

para verificar a existência de uma chave, basta usar o operador “in”

```
>>> tabela = { "Alface": 0.45,  
...                 "Batata": 1.20,  
...                 "Tomate": 2.30,  
...                 "Feijão": 1.50 }  
  
>>> print("Manga" in tabela)  
False
```



EXERCÍCIO

Exercício 6.17 Altere o programa da listagem 6.53 de forma a solicitar ao usuário o produto e a quantidade vendida. Verifique se o nome do produto digitado existe no dicionário, e só então efetue a baixa em estoque.

Disponível na pasta exercícios do
repositório Alunos_Fabrica_Programadores:

<https://github.com/evitarafadiga>



A lush tropical landscape at sunset, featuring dense foliage like palm fronds, monstera leaves, and various tropical plants. The sky is a warm gradient from purple to pink to orange, with a bright sun visible on the horizon.

Upas

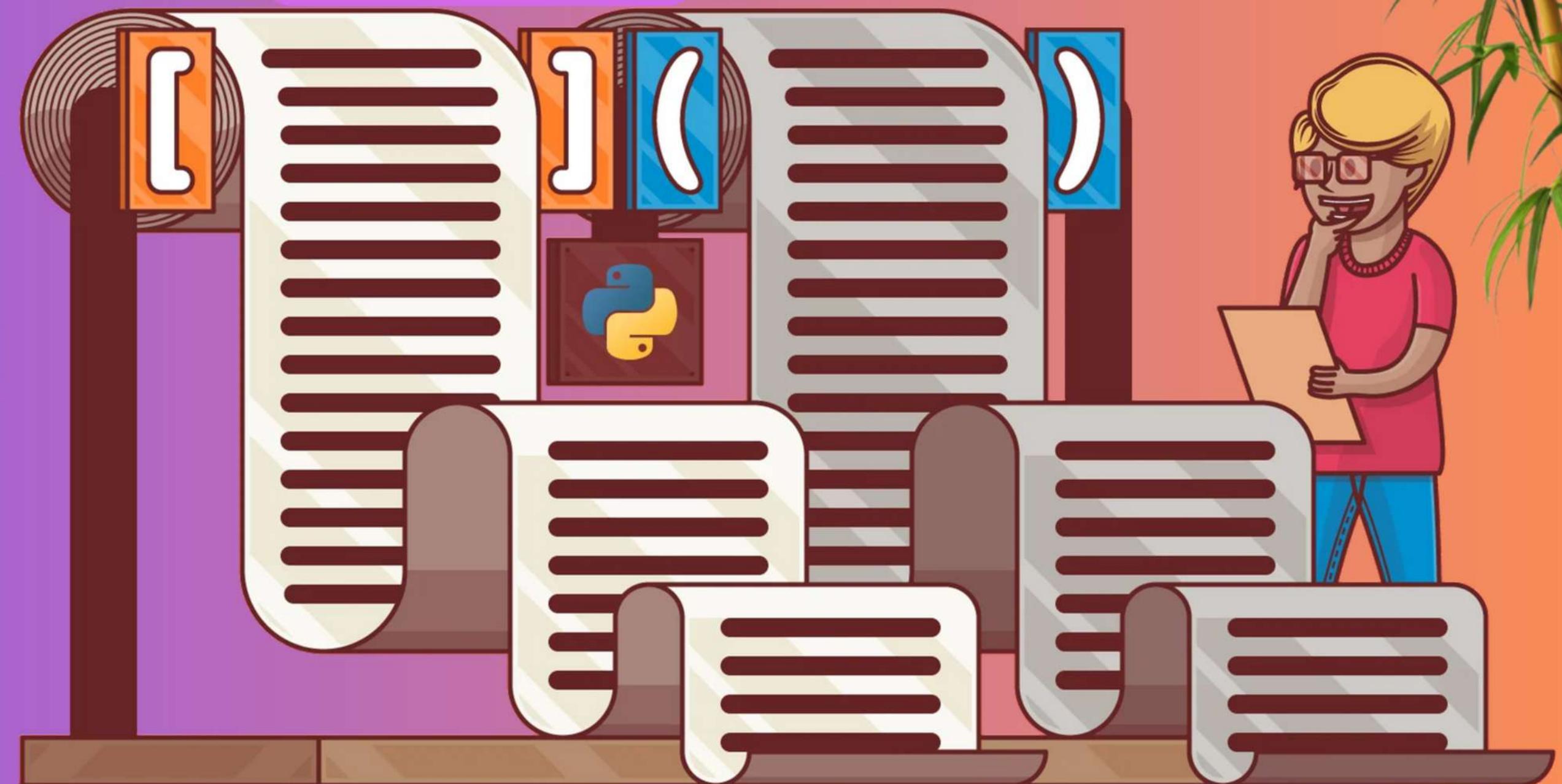


TUPLAS

Tuplas podem ser vistas como listas em Python, com a grande diferença de serem imutáveis.

são ideais para representar listas de valores constantes e também para realizar operações de empacotamento e desempacotamento de valores.

criamos tuplas utilizando parênteses ()





TUPLAS

```
>>> tupla = ("a", "b", "c")  
  
>>> tupla  
('a', 'b', 'c')
```

também é possível criar tuplas usando valores separados por vírgula,
independente de usarmos parênteses

```
>>> tupla = 100, 200, 300  
  
>>> tupla  
(100, 200, 300)
```

The background features a warm, colorful gradient from purple on the left to orange on the right, suggesting a sunset or sunrise. Overlaid on this are various tropical elements: large green palm fronds in the foreground, a winding brown vine with small green leaves across the top, and a dark brown path or riverbed curving through the center.

Enumerate



ENUMERATE

função enumerate() gera uma tupla em que o primeiro valor é o índice e o segundo é o elemento da lista sendo enumerada.

```
L=[5,9,13]  
for x, e in enumerate(L):  
    print("%d %d" % (x,e))
```

Ao utilizarmos **x, e** em **for**, indicamos que o primeiro valor da tupla deve ser colocado em **x**, e o segundo, em **e**. Assim, na primeira iteração teremos a tupla (0,5), onde **x=0** e **e=5**.

REFERÊNCIA BIBLIOGRÁFICA:

MENEZES, Nilo Ney Coutinho.;
Introdução à Programação com
Python, ed. 2. Novatec, 2014.

