

# Estruturas de Repetição

Introdução a Programação

# Objetivo de Aprendizagem

- Conhecer e aplicar estruturas de repetição

# Agenda

- Listas
- `for`
- `while`

# Listas

# Lista

Trata-se de uma coleção de dados agrupados (listados) através de um único nome. Seria como armazenar diversos valores usando um único nome de variável.

# Listas

## Exemplos

- São definidas utilizando-se colchetes `[ ]` com elementos (itens) separados por vírgula

```
1 # exemplos de listas
2 notas = [9.0, 7.5, 5.6, 7.9, 0.0]
3 quant = [1, 6, 7, 8, 4, 6]
4 usuarios = ['@joaosilva', '@mpaula12', '@jfs']
5 info = ['Ricardo Lima', 37, 127.2]
```

# Listas

Tipo de dado

```
>>> l = []  
>>> type(l)  
<class 'list'>
```

# Acessando Itens

## Indexação

Cada item possui um índice para acesso individual

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja']
2 print(f'Primeira fruta: {frutas[0]}')
3 print(f'Segunda fruta: {frutas[1]}')
4 print(f'Terceira fruta: {frutas[2]}')
```



# Acessando Itens

Índices negativos

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja']
```

Índice

-4

-3

-2

-1

Valores

maçã

banana

limão

laranja

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja']  
2 print(f'Última fruta: {frutas[-1]}')  
3 print(f'Penúltima fruta: {frutas[-2]}')
```

# Erros típicos

- Acesso fora da faixa
  - `IndexError: list index out of range`
- Índices não inteiros
  - `TypeError: list indices must be integers or slices, not float`

# Fatiamento

## *Slicing*

- Extrair uma sublista ou fração de uma lista existente
- Sintaxe: `lista[inicio:fim:passo]`
  - `inicio` índice inicial do fatiamento
  - `fim` índice do último item do fatiamento mais um
  - `passo` quantidade de saltos de cada item (**opcional**)

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja']  
2 print(f'Algumas frutas: {frutas[0:3]}')  
3 print(f'Outras frutas: {frutas[0:5:2]}')
```

# Funções especiais

Adicionar, ordenar e apagar itens

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja']
2 frutas.append('abacaxi')
3 print(frutas)
4 frutas.append('kiwi')
5 print(frutas)
6 frutas.sort()
7 print(frutas)
8 frutas.remove('kiwi')
9 print(frutas)
```

for

# Estruturas de repetição

São estruturas que permitem a execução de blocos de código uma determinada quantidade de vezes ou até que uma condição seja atingida.

# for

*Loops* ou *laços*

- Permite percorrer itens de uma lista ou qualquer outro tipo iterável (*iterable*)

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja', 'abacaxi']
2 for f in frutas:
3     print(f)
4 print('Estas são as frutas disponíveis')
```

# for

*Loops* ou *laços*

- Permite percorrer itens de uma lista ou qualquer outro tipo iterável (*iterable*)

```
1 texto = 'algum conteúdo'
2 for c in texto:
3     print(c)
```



# for

## Exemplo 1

```
1  frutas = ['maçã', 'banana', 'limão', 'laranja', 'abacaxi']
2  cont = 0
3  frutas.sort()
4  for f in frutas:
5      cont += 1
6      print(f'{str(cont)} - {f.title()}')
7  print(f'Estas são as {cont} frutas em ordem')
```

# for

## Exemplo 2 continue

```
1 frutas = ['maçã', 'banana', 'limão', 'laranja', 'abacaxi']
2 for f in frutas:
3     if f == 'banana':
4         continue
5     else:
6         print(f)
7 print('Lista de frutas sem banana')
```

# for

## Exemplo 3 break

```
1  frutas = ['maçã', 'banana', 'limão', 'laranja', 'abacaxi']
2  for f in frutas:
3      if f == 'banana':
4          print(f)
5          break
6  print('Há banana na lista')
```

# range()

## Exemplo 4

- `range()` cria uma sequência de números
- É útil quando não há um iterável no código e deseja-se realizar um *loop*

```
1 for x in range(6):  
2     print(x)
```

# range()

Exemplo 5 (números pares entre 11 e 21)

```
1  for n in range(11, 22):  
2      if (n % 2 == 0):  
3          print(n)  
4      else:  
5          continue
```

# range()

Exemplo 6 (soma dos números ímpares entre 11 e 21)

```
1 soma = 0
2 for n in range(11, 22):
3     if n % 2:
4         soma += n
```

```
while
```

# while

- Executa os comandos do bloco enquanto uma condição é verdadeira

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
```



# Utilizando `while`

## Exemplo 7

```
1  frutas = ['maçã', 'banana', 'limão', 'laranja', 'abacaxi']
2  cont = 0
3  frutas.sort()
4  while cont < len(frutas):
5      cont += 1
6      print(f'{str(cont)} - {frutas[cont-1].title()}')
7  print(f'Estas são as {cont} frutas em ordem')
```

# Exemplo 8

Somar números até o usuário digitar 0

```
1 soma = 0
2
3 while True:
4     num = int(input("Digite um número (0 para sair): "))
5     if num == 0:
6         break
7     soma += num
8
9 print("A soma dos números digitados é:", soma)
```

# Laços infinitos

- `while` permite criar laços infinitos
- Laços que são executados indefinidamente, pois a condição pode não ser atendida nunca

# Laços infinitos

Exemplo 9 (controle de opções do usuário e saída de um programa)

```
1  while True:
2      print("=== Menu ===")
3      print("1. Sacar dinheiro\n2. Depositar dinheiro")
4      print("3. Consultar saldo\n0. Sair")
5      opcao = input("Escolha uma opção (1-3): ")
6      if opcao == '1':
7          print("Realizando saque")
8      elif opcao == '2':
9          print("Realizando depósito")
10     elif opcao == '3':
11         print("Seu saldo é .....")
12     else:
```

# while versus for

	for	while
Quando usar?	Quando o número de iterações é conhecido ou finito	Quando o número de iterações <b>é incerto</b>
Controle da repetição	Itera sobre uma <b>sequência</b> ou intervalo fixo	Repetição controlada por uma <b>condição lógica</b>
Exemplo típico	<pre>for i in range(5):</pre>	<pre>while x &lt; 10:</pre>
Interrupção manual	Pode usar <code>break</code> , mas geralmente não precisa	Normalmente usa <code>break</code> ou altera a condição
Loops infinitos	✅ Sim (se bem definido)	❌ Pode entrar em loop infinito

# Exercícios

1

Criar um *script* em Python que recebe um texto do usuário e conta a quantidade de palavras desse texto.

## 2

Modifique o *script* do exercício anterior para que seja feita a contagem das vogais do texto passado pelo usuário.



### 3

Escrever o jogo **Adivinhe o Número**. Um número aleatório entre 0 e 10 é iniciado em uma variável. Em seguida o usuário deve tentar adivinhar esse número através de uma quantidade ilimitada de tentativas. **Sugestão**: utilize a função `randint()`, conforme mostrado no exemplo para gerar o número aleatório (É necessário adicionar o módulo `random`).

```
1 from random import randint
2 # como usar o randint?
3 numero_aleatorio = randint(0, 10)
4 print(numero_aleatorio)
```

# Referências

- Python.org
- Python Academy

# José Roberto Bezerra

✉ [jbroberto@ifce.edu.br](mailto:jbroberto@ifce.edu.br)

🐙 [jbroberto76](#)

Powered by  Slidev

Cover image by [haikei](#)