

PYTHON PARA JORNALISTAS DE DADOS

Lista, tupla, conjunto e *for loop*

Rodolfo Viana
MBA em Jornalismo de Dados
7 de agosto de 2021



Coleções de dados | Trabalhando com itens múltiplos

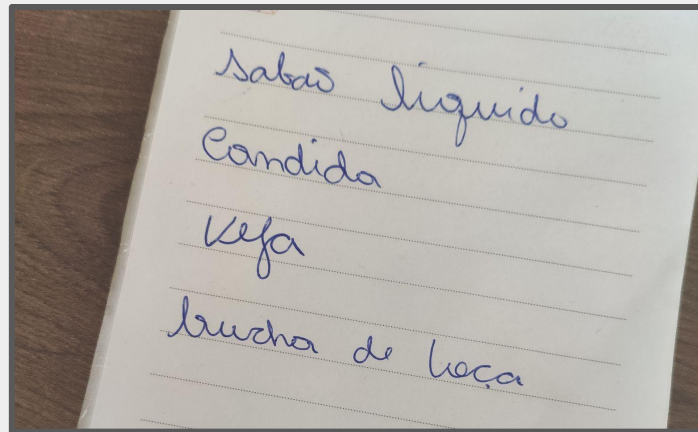
Vimos em aulas anteriores alguns tipos de dados, como `str`, `int`, `float` e `bool`.

Também vimos como armazenar os dados em variáveis.

Às vezes, porém, precisamos armazenar **mais de um item numa variável**. Por exemplo, uma lista de compras ou uma relação de alunos.

Para isso existem as **coleções de dados**. Hoje veremos três delas.

```
texto = "Terceira aula, guerreiros!" # str
inteiro = 8451 # int
ponto_flutuante = 16.97 # float
booleano = True # bool
```



Coleções de dados | Listas

Lista é a coleção mais simples de Python. Ela é construída de duas formas:

1. colocando itens entre colchetes (`[]`);
2. chamando a função `list()` quando quiser uma lista vazia ou transformar uma coleção de outro tipo em lista.

O acesso a cada item é feito com colchetes e a posição. Por exemplo, `lista[0]`, `lista[2]`, `lista[-1]`...

```
alunos = ["Ana", "Beatriz", "Lucas", "Cléber"]
# POSIÇÕES: 0      1      2      3

print(alunos)
# OUTPUT: ['Ana', 'Beatriz', 'Lucas', 'Cléber']

print(type(alunos))
# OUTPUT: <class 'list'>

print(len(alunos))
# OUTPUT: 4

print(alunos[0])
# OUTPUT: Ana

print(alunos[2])
# OUTPUT: Lucas

print(alunos[-1])
# OUTPUT: Cléber
```

Coleções de dados

Listas: características

→ aceita itens repetidos;

→ é mutável;

→ para modificar, usamos algumas funções como:

`.append(x)` para adicionar um item `x`

`.pop(i)` para tirar da lista um item de índice `i` e mostrar esse item

`.remove(x)` para tirar um item `x`

`.reverse()` para inverter a ordem

`.count(x)` para contar quantas vezes o item `x` aparece

`.sort()` para organizar os itens do menor ao maior (ou do maior ao menor, se usar `.sort(reverse=True)`)

```
# crio uma lista vazia
numeros = list()
```

```
# adiciono um valor
numeros.append(12)
print(numeros)
# OUTPUT: [12]
```

```
# adiciono outros valores
numeros.append(61)
numeros.append(37)
numeros.append(80)
numeros.append(37)
print(numeros)
# OUTPUT: [12, 61, 37, 80, 37]
```

```
# inverte a ordem
numeros.reverse()
print(numeros)
# OUTPUT: [37, 80, 37, 61, 12]
```

```
# ordeno do maior para o menor
numeros.sort(reverse=True)
print(numeros)
# OUTPUT: [80, 61, 37, 37, 12]
```

```
# conto quantas vezes aparece o número 37
print(numeros.count(37))
# OUTPUT: 2
```

Coleções de dados | Listas: exemplo

```
23 hoje = datetime.now().date().strftime("%Y-%m-%d")
24
25 ufs = [
26     "AC", "AL", "AM", "AP", "BA", "CE", "DF", "ES", "GO",
27     "MA", "MG", "MS", "MT", "PA", "PB", "PE", "PI", "PR",
28     "RJ", "RN", "RO", "RR", "RS", "SC", "SE", "SP", "TO"
29 ]
30
31 # Configurando colunas
32 cols_xlsx = [0, 1, 2, 3, 4, 5, 6, 11, 12, 13, 14, 17, 21, 22]
33 list_cols = [
34     "uf",
35     "doses_aplic_hoje_dose1",
36     "doses_aplic_hoje_dose2",
37     "vacinados_acumul_hoje_dose1",
38     "vacinados_acumul_ontem_dose1",
39     "vacinados_acumul_hoje_dose2",
40     "vacinados_acumul_ontem_dose2",
41     "perc_vacinados_poptotal_dose1",
42     "perc_vacinados_poptotal_dose2",
43     "perc_vacinados_popmaiores18_dose1",
44     "perc_vacinados_popmaiores18_dose2",
45     "doses_recebidas_acumul",
46     "poptotal",
47     "pop18anos",
48     "data",
49     "atualizacao"
50 ]
51 cols = {
```

Trecho do código de leitura da planilha de covid-19, do consórcio de veículos

Coleções de dados | Tuplas

Tupla é outra coleção comum de Python. Ela é construída de duas formas:

1. colocando itens entre parênteses (`()` e `)`);
2. chamando a função `tuple()` quando quiser uma tupla vazia ou transformar uma coleção de outro tipo em tupla.

Aqui também o acesso a cada item é feito com colchetes e a posição. Por exemplo, `tupla[0]`, `tupla[2]`, `tupla[-1]`...

```
alunos = ("Ana", "Beatriz", "Lucas", "Cléber")

print(alunos)
# OUTPUT: ('Ana', 'Beatriz', 'Lucas', 'Cléber')

print(type(alunos))
# OUTPUT: <class 'tuple'>

print(len(alunos))
# OUTPUT: 4

print(alunos[0])
# OUTPUT: Ana

print(alunos[2])
# OUTPUT: Lucas

print(alunos[-1])
# OUTPUT: Cléber
```

Coleções de dados | Tuplas: características

- aceita itens repetidos;
- é imutável.

```
# crio uma tupla vazia
numeros = tuple()

# adiciono um valor
numeros.append(12)
# OUTPUT: AttributeError: 'tuple' object has no attribute 'append'

# crio uma tupla com valores
nomes = ("José", "Pedro", "Paulo", "Ana")
print(nomes)
# OUTPUT: ('José', 'Pedro', 'Paulo', 'Ana')

# inverte a ordem
nomes.reverse()
# OUTPUT: AttributeError: 'tuple' object has no attribute 'reverse'
```

Se não dá para fazer nada com tuplas, por que raios alguém preferiria tuplas em vez de listas?

A resposta é simples: integridade de dados. Como não é possível adicionar, remover, alterar dados numa tupla, sabemos que os valores que estão ali não sofreram manipulações.

Então, quando quiser manipular dados, use listas; quando quiser se certificar que os dados não foram manipulados, use tuplas.

Coleções de dados | Conjuntos

Conjuntos é mais uma coleção de Python, mas menos usada do que listas e tuplas. Ela é construída de duas formas:

1. colocando itens entre chaves (`{ }` e `set()`);
2. chamando a função `set()` quando quiser um conjunto vazio ou transformar uma coleção de outro tipo em conjunto.

Aqui não é possível acessar cada item pela posição. Aceita somente **iteração** (falaremos disso a seguir).

```
alunos = {"Ana", "Beatriz", "Lucas", "Cléber"}

print(alunos)
# OUTPUT: {'Lucas', 'Beatriz', 'Cléber', 'Ana'}
# Repare que o retorno é aleatório

print(type(alunos))
# OUTPUT: <class 'set'>

print(len(alunos))
# OUTPUT: 4

print(alunos[0])
# OUTPUT: TypeError: 'set' object is not subscriptable

print(alunos[2])
# OUTPUT: TypeError: 'set' object is not subscriptable
```


Coleções de dados | Conjuntos: características

- aceita itens repetidos, mas retorna somente itens únicos;
- é imutável;
- itens não podem ser encontrados pela posição.

```
# crio um conjunto vazio
numeros = set()

# adiciono um valor
numeros.append(12)
# OUTPUT: AttributeError: 'set' object has no
attribute 'append'

# crio um conjunto com repetições
nomes = {"José", "Pedro", "José", "Ana", "Ana"}
print(nomes)
# OUTPUT: {'Ana', 'José', 'Pedro'}
```

Se não dá para fazer nada com conjuntos, por que raios alguém preferiria isso em vez de listas?

Porque conjuntos retornam valores sem repetição. Imagine encontrar dados repetidos numa lista com 1.974.456.987 valores! Isso fica simples quando você converte lista para conjunto (e, se quiser, depois converter de conjunto para lista novamente).

Lista

- função `list()` para lista vazia ou conversão;
- aceita itens repetidos;
- é mutável;
- aceita métodos para manipulação;
- itens acessados a partir da posição, sendo `0` a primeira posição.

Tupla

- função `tuple()` para tupla vazia ou conversão;
- aceita itens repetidos;
- é imutável;
- não aceita métodos para manipulação;
- itens acessados a partir da posição, sendo `0` a primeira posição.

Conjunto

- função `set()` para conjunto vazio ou conversão;
- aceita itens repetidos, mas retorna itens únicos;
- é imutável;
- não aceita métodos para manipulação;
- itens não podem ser acessados a partir da posição.

Coleções de dados | Conversão de coleção

para lista: `list()`

```
tupla = (1, 2, 3, 4)
print(type(tupla))
# OUTPUT: <class 'tuple'>

conjunto = {1, 2, 3, 4}
print(type(conjunto))
# OUTPUT: <class 'set'>

lista_1 = list(tupla)
print(lista_1)
# OUTPUT: [1, 2, 3, 4]

print(type(lista_1))
# OUTPUT: <class 'list'>

lista_2 = list(conjunto)
print(lista_2)
# OUTPUT: [1, 2, 3, 4]

print(type(lista_2))
# OUTPUT: <class 'list'>
```

para tupla: `tuple()`

```
lista = [1, 2, 3, 4]
print(type(lista))
# OUTPUT: <class 'list'>

conjunto = {1, 2, 3, 4}
print(type(conjunto))
# OUTPUT: <class 'set'>

tupla_1 = tuple(lista)
print(tupla_1)
# OUTPUT: (1, 2, 3, 4)

print(type(tupla_1))
# OUTPUT: <class 'tuple'>

tupla_2 = tuple(conjunto)
print(tupla_2)
# OUTPUT: (1, 2, 3, 4)

print(type(tupla_2))
# OUTPUT: <class 'tuple'>
```

para conjunto: `set()`

```
lista = [1, 2, 3, 4]
print(type(lista))
# OUTPUT: <class 'list'>

tupla = (1, 2, 3, 4)
print(type(tupla))
# OUTPUT: <class 'tuple'>

conj_1 = set(lista)
print(conj_1)
# OUTPUT: {1, 2, 3, 4}


print(type(conj_1))
# OUTPUT: <class 'set'>

conj_2 = set(tupla)
print(conj_2)
# OUTPUT: {1, 2, 3, 4}

print(type(conj_2))
# OUTPUT: <class 'set'>
```

For loop | Iteração

Estamos habituados a fazer operações com valor único...



...mas como realizar operações com uma coleção de dados? Como iterar (ou seja, repetir cálculos sobre) os valores de uma coleção?

A resposta: com **for loop**!

```
# Ver se o número é par e,  
# 1. se sim, calcular o valor multiplicado por 15.6 e imprimir  
# 2. se não, calcular o valor multiplicado por 13.1 e imprimir  
  
numero = 6574  
  
if numero % 2 == 0:  
    valor = numero * 15.6  
    print("{} é par, e a multiplicação dá {}".format(numero, valor))  
else:  
    valor = numero * 13.1  
    print("{} é ímpar, e a multiplicação dá {}".format(numero, valor))
```

For loop | Iteração

com valor único



```
# Ver se o número é par e,  
# 1. se sim, calcular o valor multiplicado por 15.6 e imprimir  
# 2. se não, calcular o valor multiplicado por 13.1 e imprimir
```

```
numero = 6574
```

```
if numero % 2 == 0:  
    valor = numero * 15.6  
    print("{} é par, e a multiplicação dá {}".format(numero, valor))  
else:  
    valor = numero * 13.1  
    print("{} é ímpar, e a multiplicação dá {}".format(numero, valor))
```

com valores em coleção



```
# Ver se cada número da lista é par e,  
# 1. se sim, calcular o valor multiplicado por 15.6 e imprimir  
# 2. se não, calcular o valor multiplicado por 13.1 e imprimir
```

```
numeros = [6574, 94651, 4921]
```

```
for num in numeros:  
    if num % 2 == 0:  
        valor = num * 15.6  
        print("{} é par, e a multiplicação dá {}".format(num, valor))  
    else:  
        valor = num * 13.1  
        print("{} é ímpar, e a multiplicação dá {}".format(num, valor))
```

Prática | Atividade 1

Em nossa primeira aula, tivemos uma tarefa: ordenar um código para calcular área e perímetro de um círculo de 12,7 metros de raio. [[link da atividade](#)]

Retome este script. Além de reordenar as linhas,

1. utilize uma lista com os valores 12,7, 29,1, 10,5 e 14,3
2. crie um fluxo que retorne os valores calculados para cada item da lista.

Tarefa | Temperaturas em cinco capitais

Na aula passada, fizemos uma atividade em sala que pedia ao usuário a informação de temperatura e retornava uma descrição (congelante, muito frio etc.). [[link para a apresentação da aula anterior](#); veja página 13]

Esta atividade é parecida, mas em vez de uma temperatura qualquer, peça ao usuário as temperaturas de São Paulo, Rio de Janeiro, Belo Horizonte, Brasília e Recife. Adicione cada temperatura numa lista e, ao fim, retorne uma frase com a descrição de cada temperatura.

Dica: para saber a qual capital pertence cada temperatura, use a **posição** do valor na lista.

Atividade	Entrega	Nota
código	17/08	0,3
Enviar arquivo py para eu@rodolfoviana.com.br		

Entenda mais | Material complementar

Para assistir

- *Estruturas de Dados (List, Dict, Tuple, Set)*, em Solyd Offensive Security [[link](#)]
- *Loop for*, em Bóson Treinamentos [[link](#)]
- *Percorrendo listas, tuplas, dicionários e conjuntos*, em Marcos Castro [[link](#)]

Para ler

- *Coleções no Python: Listas, Tuplas e Dicionários*, em DevMedia [[link](#)]
- *Principais Estruturas de Dados no Python*, em TreinaWeb [[link](#)]

Documentação

- Estruturas de dados [[link](#)]