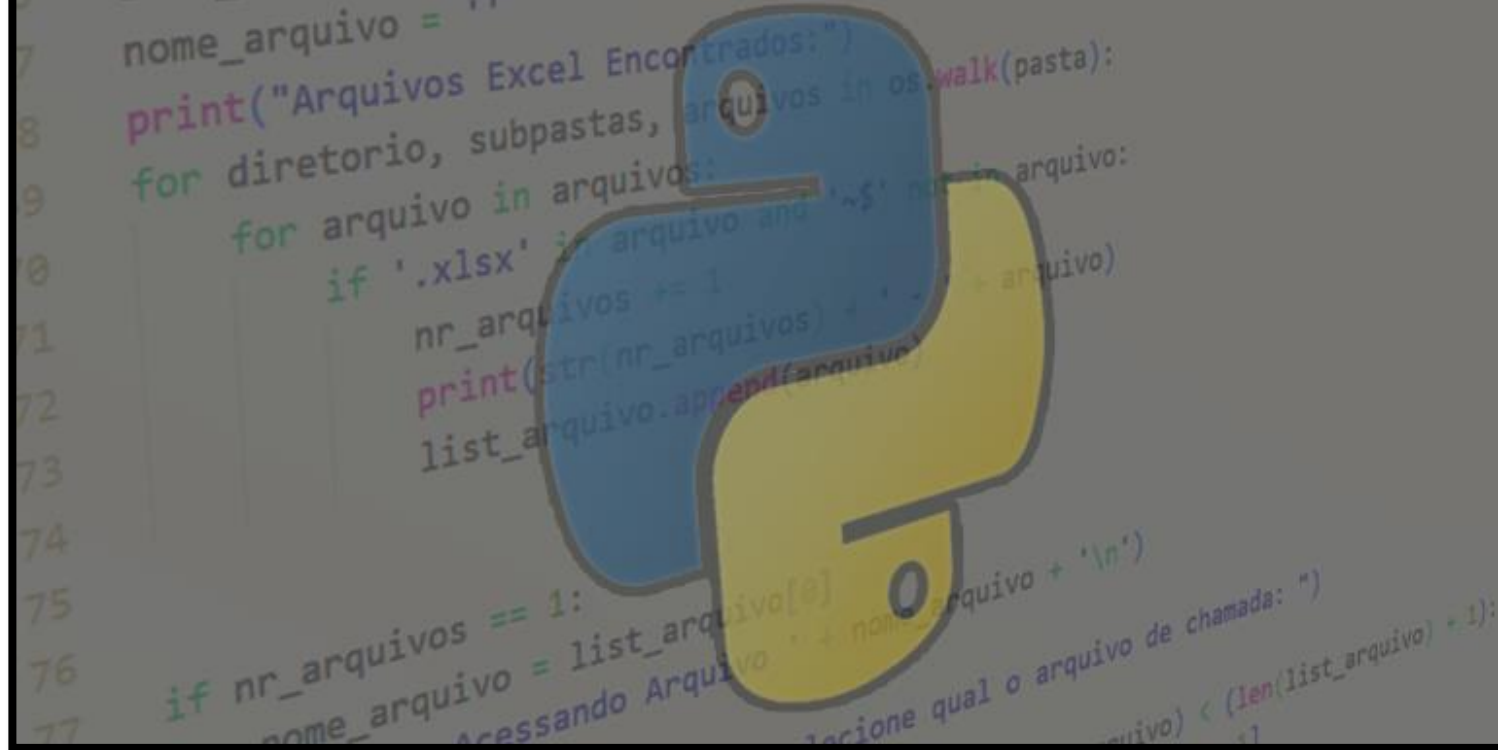


PROGRAMAÇÃO EM PYTHON



Prof. Me Jean Justino

Conjunto de dados

Quando precisamos armazenar múltiplos dados dentro de apenas uma variável usamos variáveis do tipo conjunto de dados. Em python temos 4 variáveis deste tipo:

- **List**
Variáveis do tipo **List** são ordenadas, mutáveis e permitem valores duplicados.
- **Tuple**
Variáveis do tipo **Tuple** são ordenadas, imutáveis e permitem valores duplicados.
- **Set**
Variáveis do tipo **Set** são desordenadas, imutáveis e não permitem valores duplicados.
- **Dictionary**
Variável do tipo **Dictionary** são ordenadas, mutáveis e não permitem valores duplicados. Esse tipo de variável armazena seus valores correlacionando-os com palavras-chave.

List

Armazena itens de forma ordenada: O primeiro item criado é endereçado no índice 0, o segundo no índice 1 e assim por diante:

```
lista_de_compras = ["feijão", "carne", "batata", "maça", "ovo"]  
  
print('Minha lista de compras:')  
print(lista_de_compras)  
  
print('\nMinha primeira compra: ' + lista_de_compras[0])
```

```
Minha lista de compras:  
['feijão', 'carne', 'batata', 'maça', 'ovo']  
  
Minha primeira compra: feijão
```

List



Os itens armazenados são mutáveis: É possível alterar o valor individual de cada item:

```
lista_de_compras = ["feijão", "carne", "batata", "maça", "ovo"]  
  
print('Minha lista de compras:')  
print(lista_de_compras)  
  
lista_de_compras[0] = 'feijão roxo'  
  
print(lista_de_compras)
```

```
Minha lista de compras:  
['feijão', 'carne', 'batata', 'maça', 'ovo']  
['feijão roxo', 'carne', 'batata', 'maça', 'ovo']
```

List

Permite valores duplicados:

```
lista_de_compras = ["feijão", "carne", "batata", "maça", "ovo", "batata"]  
  
print('Minha lista de compras:')  
print(lista_de_compras)
```

```
Minha lista de compras:  
['feijão', 'carne', 'batata', 'maça', 'ovo', 'batata']
```

List

Variáveis do tipo **List** permitem ter itens de tipos diversos dentro dela, inclusive uma lista dentro de uma lista:

```
lista_guloseimas = ["Chocolate", "Bis", "Bolacha"]  
  
lista_bebidas = ["Coca", "Suco", "Gatorade"]  
  
lista_de_compras = [lista_guloseimas, lista_bebidas, "Feijão"]  
  
print('Minha lista de compras:')  
print(lista_de_compras)
```

```
Minha lista de compras:  
[['Chocolate', 'Bis', 'Bolacha'], ['Coca', 'Suco', 'Gatorade'], 'Feijão']
```

List

É possível misturar tipos de dados diferentes dentro de uma lista:

```
lista_aleatoria = ["Daniel", 5, 0.75, True]  
print(lista_aleatoria)
```

```
['Daniel', 5, 0.75, True]
```

List



O acesso aos itens de uma lista é pelo índice:

```
lista_de_compras = ["Coca", "Suco", "Gatorade",  
                    "Feijão", "Chocolate", "Bis",  
                    "Bolacha"]
```

```
print('Minha lista de compras:')  
print(lista_de_compras)  
print()
```

```
print('Item 4: ' + lista_de_compras[3])  
print()
```

```
print('Ultimo item: ' + lista_de_compras[-1])  
print()
```

```
print('Penultimo item: ' + lista_de_compras[-2])  
print()
```

```
Minha lista de compras:  
['Coca', 'Suco', 'Gatorade', 'Feijão', 'Chocolate', 'Bis', 'Bolacha']  
  
Item 4: Feijão  
  
Ultimo item: Bolacha  
  
Penultimo item: Bis
```


List



É possível acessar um intervalo específico de itens da lista:

```
lista_de_compras = ["Coca", "Suco", "Gatorade",  
                    "Feijão", "Chocolate", "Bis",  
                    "Bolacha"]
```

```
print('Minha lista de compras:')  
print(lista_de_compras)  
print()
```

```
print('Itens 1~5: ',end='')  
print(lista_de_compras[1:6])  
print()
```

```
print('Itens 0~3: ',end='')  
print(lista_de_compras[:4])  
print()
```

```
print('Itens 4 em diante: ',end='')  
print(lista_de_compras[4:])  
print()
```

```
Minha lista de compras:  
['Coca', 'Suco', 'Gatorade', 'Feijão', 'Chocolate', 'Bis', 'Bolacha']  
  
Itens 1~5: ['Suco', 'Gatorade', 'Feijão', 'Chocolate', 'Bis']  
  
Itens 0~3: ['Coca', 'Suco', 'Gatorade', 'Feijão']  
  
Itens 4 em diante: ['Chocolate', 'Bis', 'Bolacha']
```

List

Podemos verificar se um item específico existe dentro de uma lista:

```
lista_de_compras = ["Coca", "Suco", "Gatorade",  
                    "Feijão", "Chocolate", "Bis",  
                    "Bolacha"]  
  
print('Minha lista de compras:')  
print(lista_de_compras)  
print()  
  
if "Chocolate" in lista_de_compras:  
    print('Tem chocolate na lista de compras')
```

```
Minha lista de compras:  
['Coca', 'Suco', 'Gatorade', 'Feijão', 'Chocolate', 'Bis', 'Bolacha']  
  
Tem chocolate na lista de compras
```

List

Podemos **desempacotar** uma lista podemos utilizar a seguinte lógica:

```
lista = ['joao', 'maria', 'jose']  
nome1, nome2, nome3 = lista  
print(nome3)
```

Podemos **desempacotar parcialmente** uma lista:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
nome1, nome2, *resto = lista  
print(nome2)  
print(resto)
```

List

É possível **iterar** sobre um list e manipular cada item:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
for nome in lista:  
    print('Olá ' + str(nome))
```

Para gerar um índice de cada item podemos utilizar uma destas maneiras:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
for indice in range(len(lista)):  
    print(str(indice+1) + ': Olá ' + str(lista[indice]))
```

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
for indice, nome in enumerate(lista, start=1):  
    print(str(indice) + ': Olá ' + str(nome))
```

List

Exemplo de aplicação: Lançamento de dado

```
import random # Importa Biblioteca

valor_dados = [0, 0, 0, 0, 0, 0] # Cria a lista de valores dos lançamentos dos dados

# Cria uma função de jogar os dados
def jogar_dados(num_vezes):
    for i in range(num_vezes):
        valor_dados[random.randint(0,5)] += 1

jogar_dados(10000) # Chama a função criada com o argumento de 10000

# Imprime o valor de cada item da lista valor_dados
for i in range(len(valor_dados)):
    print(str(i+1) + ': ' + str(valor_dados[i]))
```

```
1: 1672
2: 1694
3: 1601
4: 1675
5: 1618
6: 1740
```

List



Métodos de utilização de Lists:

Metódo	Descrição
append()	Adiciona um item no final da lista
clear()	Remove todos os itens da lista
copy()	Retorna a cópia de uma lista
count()	Retorna o número de vezes que um item aparece na lista
extend()	Adiciona os itens de uma lista no final de outra lista
index()	Retorna o índice da primeira vez que um item aparece em uma lista
insert()	Insere um item na posição especificada
pop()	Remove um item na posição especificada
remove()	Remove o primeiro item com o nome especificado
reverse()	Inverte a ordem da lista
sort()	Organiza a lista

List

Utilizando o método append:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
lista.append('eliana')  
print(lista)
```

Utilizando o método pop:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
lista.pop(2)  
print(lista)
```

Também é possível utilizar a keyword **del** para remover um item igual ao método pop:

```
lista = ['joao', 'maria', 'jose', 'antonio', 'carlos', 'julia']  
del lista[2]  
print(lista)
```

Tuples

Tuples são como listas, porém não é possível adicionar ou remover valores (a não ser que seja a adição de outra tuple)

```
1 familia_tuple = ('pai', 'mãe', 'irmão')
2
3 familia_tuple.append('tio')
```

Traceback (most recent call last):

File "/home/pi/Desktop/teste2.py", line 3, in <module>

 familia_tuple.append('tio')

AttributeError: 'tuple' object has no attribute 'append'

```
1 familia_tuple = ('pai', 'mãe', 'irmão')
2
3 familia_add = ('tio',)
4
5 familia_tuple += familia_add
6
7 print(familia_tuple)
```

('pai', 'mãe', 'irmão', 'tio')

Tuples



Métodos de utilização de Tuples:

Metódo	Descrição
count()	Retorna o número de vezes que um item aparece na lista
index()	Retorna o índice da primeira vez que um item aparece em uma lista

Sets

Sets são um conjunto de variáveis que não podem ser ordenados nem endereçados (ou seja, não tem índice para definir a posição de cada item) e os itens são imutáveis (mas podem ser deletados). Por não terem índice, não irá reconhecer itens duplicados no conjunto Set.

```
compras_set = {"arroz", "feijão", "ovo", "ameixa"}  
  
print(compras_set)
```

```
{'ameixa', 'ovo', 'feijão', 'arroz'}
```

```
{'feijão', 'arroz', 'ameixa', 'ovo'}
```

A cada execução do programa, a ordem do Set será exibida de maneira aleatória, já que os itens não são endereçados.

Sets



Métodos de utilização de Sets:

Metódo	Descrição
add()	Adiciona um item
clear()	Remove todos os itens
copy()	Retorna a cópia de um Set
difference()	Retorna um Set contendo todos os itens do primeiro Set que não estiverem em outros Sets especificados
difference_update()	Remove os itens deste Set que estiverem inseridos em outro Set
discard()	Remove um item especificado
intersection()	Retorna um Set contendo todos os itens do primeiro Set que estiverem em outros Sets especificados
intersection_update()	Remove os itens deste Set que não estiverem em outro Set
isdisjoint()	Retorna True se não tiver itens iguais em dois Sets

Sets



Metódo	Descrição
issubset()	Retorna True se todos os itens deste Set estiverem dentro de outro Set
issuperset()	Retorna True se todos os itens de outro Set estiverem dentro de deste Set
pop()	Remove um item aleatório de um Set, sendo possível salvar o item removido em uma variável
remove()	Remove um item especificado, assim como o método discard(), porém se o item não existir irá causar um erro
symmetric_difference()	Retorna um set com todos os itens diferentes entre dois Sets
symmetric_difference_update()	Salva no primeiro Set todos os itens diferentes entre dois Sets
union()	Retorna um set com a junção entre os Sets
update()	Salva no primeiro Set todos os itens entre dois Sets

Dictionaries

Dictionaries são um conjunto de variáveis que ao invés de ordenar por um índice numérico, trabalham com o conjunto key:value, ou seja, todo valor inserido precisa de uma palavra-chave para atribuir àquele valor.

Dictionaries são ordenados, mutáveis e não aceitam valores duplicados.

```
familia = {"mãe": "Maria", "pai": "João", "irmão": "Pedro"}  
  
print(familia)
```

```
{'mãe': 'Maria', 'pai': 'João', 'irmão': 'Pedro'}
```

Dictionaries

Para acessar os itens de um Dictionary, basta chamar pela palavra-chave:

```
familia = {"mãe": "Maria", "pai": "João", "irmão": "Pedro"}  
  
x = familia.get("pai")  
print(x)
```

João

Também podemos acessar as Keys já cadastradas:

```
familia = {"mãe": "Maria", "pai": "João", "irmão": "Pedro"}  
  
x = familia.keys()  
print(x)
```

```
dict_keys(['mãe', 'pai', 'irmão'])
```

Dictionaries

Podemos adicionar novos itens e alterar os antigos:

```
familia = {"mãe": "Maria", "pai": "João", "irmão": "Pedro"}  
print(familia)
```

```
familia["irmã"] = "Joana"  
familia["irmão"] = "José"  
print()  
print(familia)
```

```
{'mãe': 'Maria', 'pai': 'João', 'irmão': 'Pedro'}
```

```
{'mãe': 'Maria', 'pai': 'João', 'irmão': 'José', 'irmã': 'Joana'}
```

Dictionaries



Para fazer iteração de loops usando Dictionaries, podemos acessar os **values**, os **keys** ou ambos:

```
familia = {"mãe": "Maria", "pai": "João", "irmão": "Pedro"}

for x in familia.keys():
    print(x)

print()

for x in familia.values():
    print(x)

print()

for x,y in familia.items():
    print(x,y)
```

```
mãe
pai
irmão

Maria
João
Pedro

mãe Maria
pai João
irmão Pedro
```

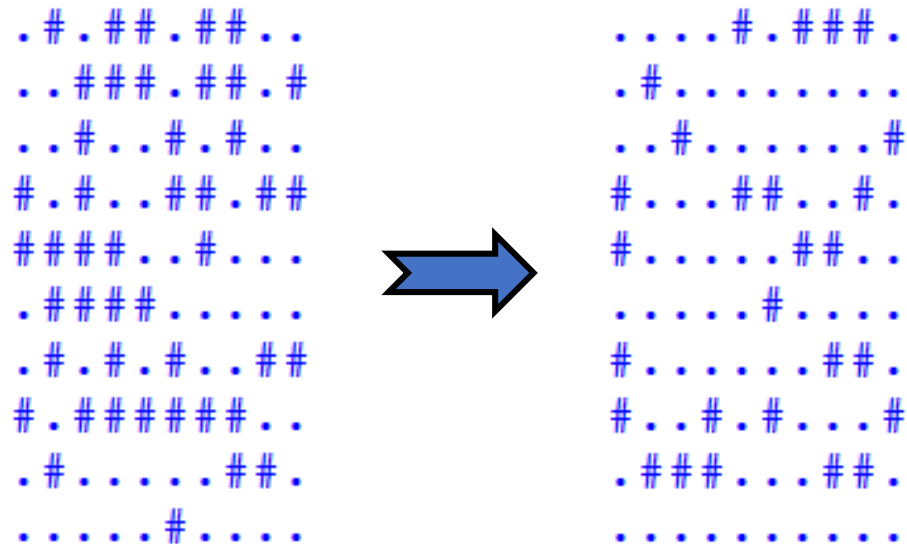

List

Exemplo de aplicação: Jogo da Vida de Conway

Este jogo foi criado em 1970 e é um exemplo programacional de autômato celular.

A ideia do jogo é reproduzir o comportamento de uma população através de regras simples, são elas:

- Se uma célula viva tiver 2 ou 3 vizinhos vivos, ela permanece viva no próximo ciclo → **Ciclo Sustentável**
- Se uma célula morta tiver exatamente 3 vizinhos vivos, ela volta a vida → **Reprodução**
- Se uma célula viva tiver mais de 3 vizinhos vivos, ela morre → **Super população**
- Se uma célula viva tiver menos que 2 vizinhos vivos, ela morre → **Isolamento**



List



```
1 # Jogo da Vida de Conway
2 import random, time, copy
3 WIDTH = 10
4 HEIGHT = 10
5
6 # Cria uma lista de listas de celulas:
7 nextCells = []
8 for x in range(WIDTH):
9     column = [] # Cria uma coluna em uma lista.
10    for y in range(HEIGHT):
11        if random.randint(0, 1) == 0:
12            column.append('#') # Adiciona uma célula viva.
13        else:
14            column.append('.') # Adiciona uma célula morta.
15    nextCells.append(column) # nextCells é uma lista de colunas.
16
```

List



```
17 while True:
18     print('\n\n\n\n\n')
19     currentCells = copy.deepcopy(nextCells) # Cria uma cópia da lista principal
20
21     # Imprime na tela:
22     for y in range(HEIGHT):
23         for x in range(WIDTH):
24             print(currentCells[x][y], end='') # Imprime cada célula.
25             print() # Imprime uma nova linha ao término de cada linha.
26
27     # Calcula o próximo ciclo com base nas células do ciclo atual:
28     for x in range(WIDTH):
29         for y in range(HEIGHT):
30             # Verifica as células vizinhas:
31             leftCoord = (x - 1)
32             rightCoord = (x + 1)
33             aboveCoord = (y - 1)
34             belowCoord = (y + 1)
35
```

List



```
35
36 # Conta o número de vizinhos vivos:
37 numNeighbors = 0
38 if aboveCoord >= 0:
39     if leftCoord >= 0:
40         if currentCells[leftCoord][aboveCoord] == '#':
41             numNeighbors += 1 # Celula Superior Esquerda está viva.
42     if currentCells[x][aboveCoord] == '#':
43         numNeighbors += 1 # Celula Superior está viva.
44     if rightCoord < 10:
45         if currentCells[rightCoord][aboveCoord] == '#':
46             numNeighbors += 1 # Celula Superior Direita está viva.
47
48 if leftCoord >= 0:
49     if currentCells[leftCoord][y] == '#':
50         numNeighbors += 1 # Celula Esquerda está viva.
51 if rightCoord < 10:
52     if currentCells[rightCoord][y] == '#':
53         numNeighbors += 1 # Celula Direita está viva.
54
55 if belowCoord < 10:
56     if leftCoord >= 0:
57         if currentCells[leftCoord][belowCoord] == '#':
58             numNeighbors += 1 # Celula Inferior Esquerda está viva.
59     if currentCells[x][belowCoord] == '#':
60         numNeighbors += 1 # Celula Inferior está viva.
61     if rightCoord < 10:
62         if currentCells[rightCoord][belowCoord] == '#':
63             numNeighbors += 1 # Celula Inferior Direita está viva.
64
```

List



```
65     # Anota como ficará cada célula de acordo com a regra do Jogo da Vida de Conway:
66     if currentCells[x][y] == '#' and (numNeighbors == 2 or numNeighbors == 3):
67         # Células Vivas com 2 ou 3 vizinhos vivos permanecem vivas:
68         nextCells[x][y] = '#'
69     elif currentCells[x][y] == '.' and numNeighbors == 3:
70         # Células Mortas com exatamente 3 vizinhos voltam a vida:
71         nextCells[x][y] = '#'
72     else:
73         # Qualquer outra condição, a célula morre ou permanece morta:
74         nextCells[x][y] = '.'
75     time.sleep(1)
```