

December 11, 2021

1 Introdução a Python

1.1 Aula 3

2 Sumário

- Coleções de dados
- Exercícios

3 Coleções de dados

3.1 List

[]: Utilizam-se para representar uma lista

```
[1]: my_list = [1, 2, 3]

type(my_list)
```

```
[1]: list
```

list(): Construtor utilizado para criar uma lista

```
[2]: this_list = list("volvo")

type(this_list)
```

```
[2]: list
```

Propriedades de uma lista:

- Permite duplicados

```
[3]: duplicates_list = [2, 2]

duplicates_list
```

```
[3]: [2, 2]
```

- Permite vários tipos diferentes

```
[4]: diff_types_list = [1, "sand", 1.75, True]

diff_types_list
```

```
[4]: [1, 'sand', 1.75, True]
```

- Ordenada

Os seus elementos têm uma ordem definida e essa ordem não se irá alterar. Consequentemente, podem ser acedidos através de um índice.

- Mutável

É possível alterar, adicionar e remover itens numa lista, após esta ter sido criada.

3.1.1 Aceder a elementos

```
[5]: colors = ["blue", "red", "yellow"] #índices começam em 0

colors[2]
```

```
[5]: 'yellow'
```

```
[6]: transports = ["car", "plane", "bike"] #neste caso, o índice -1 refere-se ao
      ↪ último elemento

transports[-2]
```

```
[6]: 'plane'
```

```
[7]: vegetables = ["carrot", "broccoli", "mushroom", "spinach"]

vegetables[1:]
```

```
[7]: ['broccoli', 'mushroom', 'spinach']
```

```
[8]: animals = ["cat", "dog", "bird", "elephant", "zebra"]

animals[:2]
```

```
[8]: ['cat', 'dog']
```

```
[15]: food = ["spaghetti", "burger", "pizza"]
```

```
food[1:2]
```

```
[15]: ['burguer']
```

```
[10]: prog_languages = ["C", "C#", "C++", "Java", "R"]  
      prog_languages[-4:-1]
```

```
[10]: ['C#', 'C++', 'Java']
```

3.1.2 Alterar elementos

```
[11]: vowels = ["a", "e", "i", "o", "u"]  
      vowels[2] = "I"  
      vowels
```

```
[11]: ['a', 'e', 'I', 'o', 'u']
```

```
[12]: subjects = ["maths", "physics", "geometry", "chemistry"]  
      subjects[1:3] = ["english", "german"]  
      subjects
```

```
[12]: ['maths', 'english', 'german', 'chemistry']
```

```
[13]: countries = ["portugal", "denmark", "mozambique", "brazil"]  
      countries[1:2] = ["belgium", "switzerland"]  
      countries
```

```
[13]: ['portugal', 'belgium', 'switzerland', 'mozambique', 'brazil']
```

```
[14]: cities = ["viseu", "coimbra", "aveiro"]  
      cities[1:3] = ["braga"]  
      cities
```

```
[14]: ['viseu', 'braga']
```

3.1.3 Adicionar elementos

```
[15]: numbers = ["one", "two", "three"]

      numbers.append("four")

      numbers
```

```
[15]: ['one', 'two', 'three', 'four']
```

```
[16]: sports = ["football", "tennis"]

      sports.insert(1, "rugby")

      sports
```

```
[16]: ['football', 'rugby', 'tennis']
```

```
[17]: apparel = ["trousers", "skirts"]
      shoes = ["boots", "slippers"] #também poderia ser um tuplo, por exemplo

      apparel.extend(shoes)

      apparel
```

```
[17]: ['trousers', 'skirts', 'boots', 'slippers']
```

3.1.4 Remover elementos

```
[18]: planets = ["mercury", "venus", "earth", "pluto"]

      planets.remove("pluto")

      planets
```

```
[18]: ['mercury', 'venus', 'earth']
```

```
[19]: summer = ["sun", "beach"]

      summer.clear()

      summer
```

```
[19]: []
```

```
[20]: body_parts = ["arms", "legs", "head"]

body_parts.pop(1) #se o índice não for especificado, remove o último elemento

body_parts
```

```
[20]: ['arms', 'head']
```

```
[21]: body_parts = ["arms", "legs", "head"]

del body_parts[1]

body_parts
```

```
[21]: ['arms', 'head']
```

3.1.5 Listas em compreensão

`newlist = [expression for item in iterable if condition == True]`: Sintaxe de uma lista em compreensão (a condição é opcional)

```
[22]: plants = ["rose", "cactus", "bamboo", "daisy"]

plants_with_s = []

for plant in plants:
    if "s" in plant:
        plants_with_s.append(plant)

plants_with_s
```

```
[22]: ['rose', 'cactus', 'daisy']
```

```
[14]: plants = ["rose", "cactus", "bamboo", "daisy"]

plants_with_s = [plant for plant in plants if "s" in plant]

plants_with_s
```

```
[14]: ['rose', 'cactus', 'daisy']
```

```
[24]: birds = ["chicken", "blackbird", "flamingo"]

birds_fixed = [bird if bird!="blackbird" else "common blackbird" for bird in
    ↪birds]
```

```
birds_fixed
```

```
[24]: ['chicken', 'common blackbird', 'flamingo']
```

3.1.6 Ordenar listas

```
[25]: weekdays = ["monday", "Tuesday", "wednesday", "thursday", "friday"]

weekdays.sort() #também funciona para números

weekdays
```

```
[25]: ['Tuesday', 'friday', 'monday', 'thursday', 'wednesday']
```

```
[26]: makeup = ["blush", "eyeshadow", "lipgloss"]

makeup.sort(reverse=True)

makeup
```

```
[26]: ['lipgloss', 'eyeshadow', 'blush']
```

```
[27]: def my_sort_function(x):
        return abs(x)

x = [-17, -1, 3, -2, 4]

x.sort(key=my_sort_function)

x
```

```
[27]: [-1, -2, 3, 4, -17]
```

```
[28]: weekend = ["saturday", "sunday"]

weekend.reverse()

weekend
```

```
[28]: ['sunday', 'saturday']
```

3.1.7 Copiar listas

```
[29]: months = ["january", "february", "march"]  
  
      new_list = months.copy()  
  
      new_list
```

```
[29]: ['january', 'february', 'march']
```

```
[30]: months = ["january", "february", "march"]  
  
      my_new_list = list(months)  
  
      my_new_list
```

```
[30]: ['january', 'february', 'march']
```

3.1.8 Outros métodos relevantes

- count()

```
[31]: money = ["euro", "pound", "dollar", "euro"]  
  
      money.count("euro")
```

```
[31]: 2
```

- index()

```
[32]: teams = ["slbenfica", "fcporto", "scbraga"]  
  
      teams.index("fcporto")
```

```
[32]: 1
```

3.2 Tuple

(): Utilizam-se para representar um tuplo

```
[33]: my_tuple = (1,2,3) #para criar um tuplo com um único elemento, é necessário  
      ↪ colocar uma vírgula no fim  
  
      type(my_tuple)
```

```
[33]: tuple
```

`tuple()` - Construtor utilizado para criar um tuplo

```
[34]: this_tuple = tuple((4,8))  
  
type(this_tuple)
```

```
[34]: tuple
```

Propriedades de um tuplo:

- Permite duplicados

```
[35]: duplicates_tuple = (2,2)  
  
duplicates_tuple
```

```
[35]: (2, 2)
```

- Permite vários tipos diferentes

```
[36]: diff_types_tuple = (2, "ana", 3.45, False)  
  
diff_types_tuple
```

```
[36]: (2, 'ana', 3.45, False)
```

- Ordenado

Os seus elementos têm uma ordem definida e essa ordem não se irá alterar. Consequentemente, podem ser acedidos através de um índice.

- Imutável

Não é possível alterar, adicionar e remover itens num tuplo, após este ter sido criado.

3.2.1 Aceder a elementos

Análogo à lista

3.2.2 Desconstruir tuplos

```
[37]: specialty = ("cardiology", "dermatology", "neurology")  
  
(heart, skin, brain) = specialty  
  
print(heart)  
print(skin)  
print(brain)
```



```
cardiology
dermatology
neurology
```

```
[38]: books = ("The Great Gatsby", "Don Quixote", "War and Peace", "Anna Karenina")

(fitzgerald, cervantes, *tolstoi) = books

print(fitzgerald)
print(cervantes)
print(tolstoi)
```

```
The Great Gatsby
Don Quixote
['War and Peace', 'Anna Karenina']
```

3.2.3 Métodos

Análogos aos que foram mencionados na última secção relativa à lista

3.3 Set

{}: Utilizam-se para representar um conjunto

```
[39]: my_set = {1,2,3} #caso não tenha elementos já não se trata de um conjunto, mas
      ↪ sim de um dicionário

type(my_set)
```

```
[39]: set
```

set() - Construtor utilizado para criar um conjunto

```
[40]: this_set = set(("banana", "apple", "orange"))

type(this_set)
```

```
[40]: set
```

Propriedades de um conjunto:

- Não permite duplicados

```
[41]: no_duplicates_set = {"tree", "tree"}

no_duplicates_set
```

```
[41]: {'tree'}
```

- Permite vários tipos diferentes

```
[42]: diff_types_set = {2, "cinema", 5.67, False}

diff_types_set
```

```
[42]: {2, 5.67, False, 'cinema'}
```

- Não ordenado

Os seus elementos não têm uma ordem definida e essa ordem pode alterar-se. Consequentemente, não podem ser acedidos através de um índice ou de uma chave.

- Mutável

É possível adicionar e remover itens num conjunto, após este ter sido criado.

3.3.1 Adicionar elementos

```
[43]: singers = {"Lady Gaga", "Shakira"}

singers.add("Beyoncé")

singers
```

```
[43]: {'Beyoncé', 'Lady Gaga', 'Shakira'}
```

```
[44]: bands = {"Maroon 5", "The Script"}
dead_bands = {"Nirvana", "Queen"} #também poderia ser uma lista, por exemplo

bands.update(dead_bands)

bands
```

```
[44]: {'Maroon 5', 'Nirvana', 'Queen', 'The Script'}
```

3.3.2 Remover elementos

```
[45]: jewelry = {"ring", "earring", "bracelet"}

jewelry.remove("ring")

jewelry
```

```
[45]: {'bracelet', 'earring'}
```

```
[46]: jewelry = {"ring", "earring", "bracelet"}

jewelry.discard("ring")

jewelry
```

```
[46]: {'bracelet', 'earring'}
```

```
[13]: weather = {"sunny", "rainy", "foggy"}

weather.pop()

weather
```

```
[13]: {'foggy', 'rainy'}
```

```
[48]: periodic_table = {"mg", "na", "li"}

periodic_table.clear()

periodic_table
```

```
[48]: set()
```

3.3.3 Juntar sets

```
[49]: living_painters = {"Cindy Sherman", "Liu Xiaodong"}
dead_painters = {"Vincent van Gogh", "Pablo Picasso"}

painters = living_painters.union(dead_painters)

painters
```

```
[49]: {'Cindy Sherman', 'Liu Xiaodong', 'Pablo Picasso', 'Vincent van Gogh'}
```

```
[50]: it_brands = {"microsoft", "apple", "google"}
fruits = {"apple", "cherry", "strawberry"}

it_brands.intersection_update(fruits)

it_brands
```

```
[50]: {'apple'}
```

```
[51]: x = {1,2,3}
y = {3,4,5}
```

```
z = x.intersection(y)

z
```

[51]: {3}

```
[52]: motorcycle_brands = {"ducati", "yamaha"}
      piano_brands = {"yamaha"}

      motorcycle_brands.symmetric_difference_update(piano_brands)

      motorcycle_brands
```

[52]: {'ducati'}

```
[53]: chocolate_cake_ingredients = {"eggs", "sugar", "flour", "chocolate"}
      strawberry_cake_ingredients = {"eggs", "sugar", "flour", "strawberry"}

      flavours = chocolate_cake_ingredients.
        ↪symmetric_difference(strawberry_cake_ingredients)

      flavours
```

[53]: {'chocolate', 'strawberry'}

3.3.4 Outros métodos relevantes

- issuperset()

```
[54]: s1 = {"a", "b", "c", "d", "e"}
      s2 = {"a", "b"}

      s1.issuperset(s2)
```

[54]: True

- issubset()

```
[55]: set1 = {1,2}
      set2 = {1,2,3,4}

      set1.issubset(set2)
```

[55]: True

- isdisjoint()

```
[56]: first_set = {True, False}
      second_set = {False}

      first_set.isdisjoint(second_set)
```

[56]: False

- copy()

```
[57]: original_set = {"ball"}
      copied_set = original_set.copy()

      copied_set
```

[57]: {'ball'}

- difference()

```
[58]: a = {1.5, 3.45}
      b = {4.75, 1.5}

      c = a.difference(b)

      c
```

[58]: {3.45}

- difference_update()

```
[59]: d = {True, 3}
      e = {3, 4}

      d.difference_update(e)

      d
```

[59]: {True}

3.4 Dictionary

{}: Utilizam-se para representar um dicionário

```
[60]: my_dict = {
      "name": "John",
      "surname": "Doe",
      "job": "lawyer"
}
```

```
type(my_dict)
```

[60]: dict

dict() - Construtor utilizado para criar um dicionário

```
[8]: this_dict = dict({"ground": "bus", "air": "airplane", "water": "boat"})  
  
type(this_dict)
```

[8]: dict

Propriedades de um dicionário:

- Não permite chaves duplicadas

```
[61]: no_duplicates_dict = {  
    "university": "Minho",  
    "location": "Braga",  
    "campus": "Azurém",  
    "campus": "Gualtar"  
}  
  
no_duplicates_dict #o valor duplicado irá sobrescrever o valor existente
```

[61]: {'university': 'Minho', 'location': 'Braga', 'campus': 'Gualtar'}

- Permite vários tipos diferentes

```
[62]: computer = {  
    "brand": "Toshiba",  
    "touch": True,  
    "colors": ["grey", "black", "white"]  
}  
  
computer
```

[62]: {'brand': 'Toshiba', 'touch': True, 'colors': ['grey', 'black', 'white']}

- Ordenado (a partir da versão 3.7)

Os seus elementos têm uma ordem definida e essa ordem não se irá alterar. Consequentemente, podem ser acedidos através de uma chave.

- Mutável

É possível alterar, adicionar e remover itens num dicionário, após este ter sido criado.

3.4.1 Acceder a elementos

```
[3]: song = {  
    "name": "Grenade",  
    "artist": "Bruno Mars",  
    "year": 2010  
}  
  
song["year"]
```

[3]: 2010

```
[6]: song = {  
    "name": "Grenade",  
    "artist": "Bruno Mars",  
    "year": 2010  
}  
  
song.get("year")
```

[6]: 2010

```
[65]: show = {  
    "name": "The Ellen DeGeneres Show",  
    "TV host": "Ellen DeGeneres",  
    "country of origin": "USA"  
}  
  
show.keys()
```

[65]: dict_keys(['name', 'TV host', 'country of origin'])

```
[66]: cr7 = {  
    "team": "Manchester United",  
    "children": 4,  
    "nationality": "Portuguese"  
}  
  
cr7.values()
```

[66]: dict_values(['Manchester United', 4, 'Portuguese'])

```
[67]: car = {  
    "brand": "Volkswagen",  
    "color": "grey",  
    "year": 2016  
}
```

```
car.items()
```

```
[67]: dict_items([('brand', 'Volkswagen'), ('color', 'grey'), ('year', 2016)])
```

3.4.2 Alterar elementos

```
[68]: sports_club = {  
    "name": "FC Barcelona",  
    "stadium": "Camp Nou",  
    "coach": "Pep Guardiola"  
}  
  
sports_club["coach"] = "Ronald Koeman"  
  
sports_club
```

```
[68]: {'name': 'FC Barcelona', 'stadium': 'Camp Nou', 'coach': 'Ronald Koeman'}
```

```
[69]: sports_club = {  
    "name": "FC Barcelona",  
    "stadium": "Camp Nou",  
    "coach": "Pep Guardiola"  
}  
  
sports_club.update({"coach": "Ronald Koeman"})  
  
sports_club
```

```
[69]: {'name': 'FC Barcelona', 'stadium': 'Camp Nou', 'coach': 'Ronald Koeman'}
```

3.4.3 Adicionar elementos

Análogo à secção anterior

3.4.4 Remover elementos

```
[70]: singer = {  
    "name": "Amy Winehouse",  
    "sex": "female",  
    "age": 27,  
    2008: ["Album of the Year", "Record of the Year"]  
}
```



```
singer.pop("age")
```

```
singer
```

```
[70]: {'name': 'Amy Winehouse',  
      'sex': 'female',  
      2008: ['Album of the Year', 'Record of the Year']}
```

```
[71]: singer = {  
      "name": "Amy Winehouse",  
      "sex": "female",  
      "age": 27,  
      2008: ["Album of the Year", "Record of the Year"]  
}
```

```
del singer["age"]
```

```
singer
```

```
[71]: {'name': 'Amy Winehouse',  
      'sex': 'female',  
      2008: ['Album of the Year', 'Record of the Year']}
```

```
[72]: country = {  
      "name": "Portugal",  
      "money": "euro",  
      "language": "portuguese"  
}
```

```
country.popitem()
```

```
country
```

```
[72]: {'name': 'Portugal', 'money': 'euro'}
```

```
[73]: cat = {  
      "date of birth": "30th of september of 2020",  
      "colors": ["white", "black", "yellow"]  
}
```

```
cat.clear()
```

```
cat
```

```
[73]: {}
```

3.4.5 Copiar dicionários

Análogo à lista

3.4.6 Outros métodos relevantes

- `fromkeys()`

```
[74]: keys = ("monday", "tuesday", "wednesday")
      values = "weekday"

      new_dict = dict.fromkeys(keys, values)

      new_dict
```

```
[74]: {'monday': 'weekday', 'tuesday': 'weekday', 'wednesday': 'weekday'}
```

- `setdefault()`

```
[9]: game = {
      "name": "LoL",
      "mode": "multiplayer",
      "year": 2009
    }

      game.setdefault("developer", "Riot Games")

      game
```

```
[9]: {'name': 'LoL', 'mode': 'multiplayer', 'year': 2009, 'developer': 'Riot Games'}
```

4 Exercícios

- 1) Escreva uma função que recebe uma lista e retorna um par com o primeiro e o último elemento dessa lista.
- 2) Escreva uma função que retorna um conjunto com os elementos que existem apenas no primeiro conjunto e não no segundo conjunto.
- 3) Escreva uma função que retorna o maior elemento de uma lista.
- 4) Escreva uma função que retorna a distância euclidiana entre dois pontos.
- 5) Escreva uma função que retorna a soma dos números que são múltiplos de 3 ou 5 entre 1 e um limite, sendo que esse limite é um parâmetro.
- 6) Escreva uma função que filtra a altura e o peso de estudantes, características estas que, por sua vez, estão armazenadas num dicionário.

- 7) Escreva uma função que retorna todas as combinações de pares de 0 até um limite, com recurso a uma lista em compreensão.
- 8) Escreva uma função que retorna o resultado de somar os elementos de cada tuplo presente numa lista de tuplos.
- 9) Escreva uma função que retorna os n menores números de uma lista de números.
- 10) Escreva uma função que recebe uma lista de listas de números e retorna a sublista que resulta na maior soma.