# Tuplas

Joyce Teixeira

# Introdução

- Iniciaremos com o seguinte problema:
  - Em uma aplicação que trabalha com a localização de endereços a partir de coordenadas geográficas;
  - Como é possível armazenar estas coordenadas?
  - Uma variável para **latitude** e outra para **longitude**?
  - Mas se for necessário trabalhar com rotas será preciso outra coordenada, então criamos mais duas variáveis de latitude e longitude?
  - E a medida que o código for crescendo?

# Introdução

- Iniciaremos com o seguinte problema:
  - O ideal seria não deixar latitude e longitude separados, mas junto em uma variável só, realmente organizados como uma coordenada;
  - Para guardar mais de um valor em uma única variável logo pensamos em **listas**.

```
school_coordenadas = [-8.0596986, -34.8724827]
machu_picchu_coordenadas = [-13.163136, -72.5471516]
```

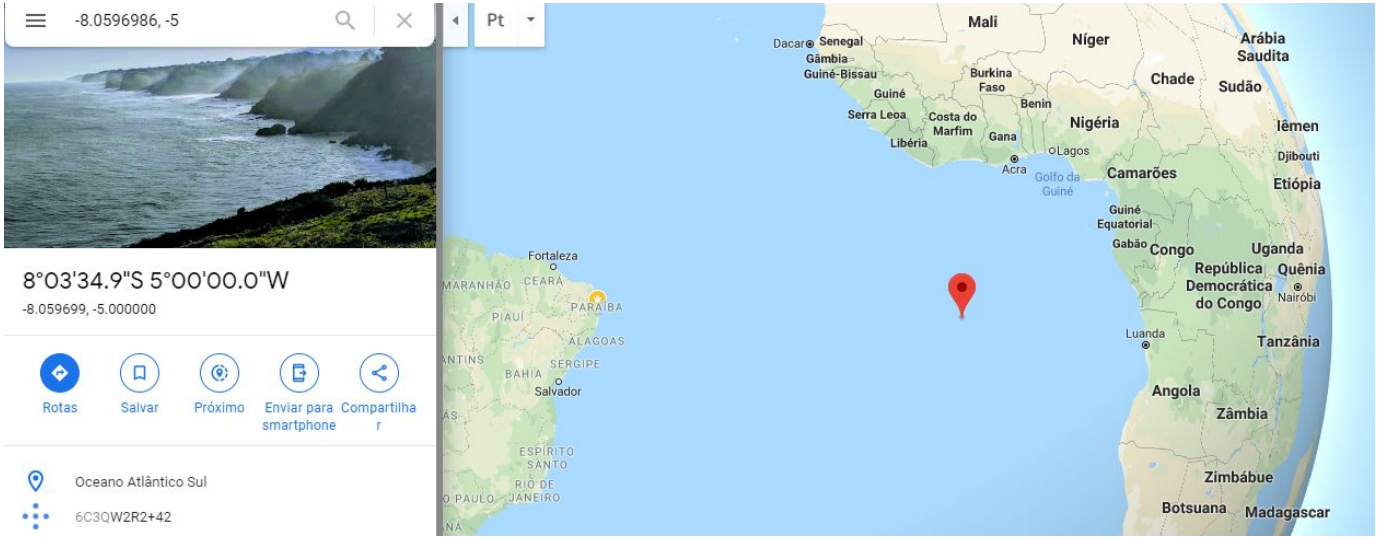
  - Melhorou! Mas é o ideal?

# Introdução

- O problema de usar listas nesse contexto é que elas são mutáveis, ou seja, os valores podem ser alterados;
- Adicionar, remover ou alterar um valor em school_coordenadas não faria sentido;
- Mudando a segunda posição da school_coordenadas para -5, iríamos da School para algum lugar do oceano atlântico, por exemplo!

# Introdução

```
school_coordenadas = [-8.0596986, -34.8724827]
school_coordenadas[1] = -5
```




C.E.S.A.R school

# Introdução

- Listas, segundo a própria documentação do Python, indica que os elementos são, geralmente, homogêneos, têm o mesmo tipo e significado;
- Com as nossas coordenadas, temos valores heterogêneos, pois representam duas coisas diferentes (para então formar uma só);
- Então o que poderia ser utilizado?
- Tuplas!

# Tuplas

- Tuplas são sequências de valores, similares as listas;
- Mas, existem importantes diferenças:
  - Os valores de uma tupla são imutáveis;
  - Tuplas usam parênteses enquanto listas usam colchetes.

```
school_coordenadas_lista = [-8.0596986, -34.8724827]
school_coordenadas_tupla = (-8.0596986, -34.8724827)

print(type(school_coordenadas_lista))
print(type(school_coordenadas_tupla))
```

```
<class 'list'>
<class 'tuple'>
```

C.E.S.A.R school

# Tuplas

- Não podemos alterar um mesmo objeto tupla, ou seja, mudar uma de suas referências internas (seus valores), nem adicionar ou remover elemento algum:

```
school_coordenadas = (-8.0596986, -34.8724827)
school_coordenadas[1] = -5
```

```
    school_coordenadas[1] = -5
TypeError: 'tuple' object does not support item assignment
```

# Criando Tuplas

- Há algumas maneiras de criar uma tupla:
  - tupla = ()
  - tupla = tuple()

  Tupla vazia
  - tupla = (‘a’, ‘b’, ‘c’)

  Tuple iniciada
  - tupla = (‘a’,)

  Tupla de um único elemento

```
tupla = ('a')
print(type(tupla))
```

```
<class 'str'>
```

```
tupla = ('a',)
print(type(tupla))
```

```
<class 'tuple'>
```

C.E.S.A.R school

# Acessando Itens da Tupla

```
nomes = ("Joao", "Maria", "Joaquim", "Eduarda", "Ester")
print(nomes[1])
print(nomes[1:3])
```

```
Maria
('Maria', 'Joaquim')
```

# Concatenação e Multiplicação de Tuplas

- É possível concatenar tuplas por meio do operador de adição **+** e multiplicá-las por um **inteiro**, o que gerará várias cópias dos seus itens.

# Concatenação e Multiplicação de Tuplas

- 

```
nomes_atuais = ('valentina', 'enzo')
nomes_tradicionais = ('ana', 'joao', 'jose')

nomes = nomes_atuais + nomes_tradicionais
print(nomes)
```

```
('valentina', 'enzo', 'ana', 'joao', 'jose')
```

# Concatenação e Multiplicação de Tuplas

- 

```
numeros = (0,)
novo_numeros = numeros * 5
print(novo_numeros)
```

```
(0, 0, 0, 0, 0)
```

# Tamanho da Tupla

- O comprimento de uma tupla, ou o número de itens que a compõem, pode ser obtido a partir da função **len()**

```
nomes = ("Joao", "Maria", "Joaquim", "Eduarda", "Ester")
len(nomes)
```

# Mínimo, Máximo e Soma

- As funções **min()**, **max()** e **sum()**, encontram o menor valor da tupla, maior valor da tupla e realiza a soma de todos os elementos da tupla, respectivamente.

# Mínimo, Máximo e Soma

```
notas = (6.0, 8.5, 4.5, 10.0, 9.5)

print("A menor nota foi:", min(notas))
print("A maior nota foi:", max(notas))
print("A soma de todas as notas foi:", sum(notas))
print("A média da turma foi:", sum(notas)/len(notas))
```

```
A menor nota foi: 4.5
A maior nota foi: 10.0
A soma de todas as notas foi: 38.5
A média da turma foi: 7.7
```

# Número de Ocorrências na Tupla

- A função **count()** retorna o número de ocorrências de determinado objeto, passado como parâmetro, em uma tupla.

```
animais_domesticos = ('cachorro', 'peixe', 'gato', 'coelho', 'hamster', 'gato')
print("Há", animais_domesticos.count('gato'), "gatos")
print("Há", animais_domesticos.count('ovelhas'), "ovelhas")
```

```
Há 2 gatos
Há 0 ovelhas
```

# Verificando Item na Tupla

- in

```
nomes = ("Joao", "Maria", "Joaquim", "Eduarda", "Ester")
print("Maria" in nomes)
print("Enzo" in nomes)
```

```
True
False
```

# Procurando Elemento e Posição na Tupla

- A função index() procura um elemento na tupla e retorna seu index.

```
nomes = ("Joao", "Maria", "Joaquim", "Eduarda", "Ester")
print(nomes.index('Maria'))
```

```
1
```

# Copiando Tuplas

- Qual será o valor de animais_domesticos e animais?

```
animais_domesticos = ('cachorro', 'gato', 'coelho')

animais = animais_domesticos + ('dinossauro',)

print(animais_domesticos)
print(animais)
```

```
('cachorro', 'gato', 'coelho')
('cachorro', 'gato', 'coelho', 'dinossauro')
```

# Copiando Tuplas

- Qual será a saída?

```
animais_domesticos = ('cachorro', 'gato', 'coelho')

animais = animais_domesticos + ('dinossauro',)
animais.append('ovelha')
```

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

# Percorrendo Tuplas

```
notas = (6.0, 8.5, 4.5, 10.0, 9.5)

for i in notas:
    print(i)
```

```
notas = (6.0, 8.5, 4.5, 10.0, 9.5)

for i in range(len(notas)):
    print(notas[i])
```

```
6.0
8.5
4.5
10.0
9.5
```

# Tuplas com Objetos Mutáveis

- Tuplas podem conter objetos mutáveis, como listas.

```
pessoas = ([30, 25, 50], ['Carlos', 'Lucas', 'Maria'])
print(pessoas)
print(pessoas[0])
print(pessoas[1])
```

```
([30, 25, 50], ['Carlos', 'Lucas', 'Maria'])
[30, 25, 50]
['Carlos', 'Lucas', 'Maria']
```

# Tuplas com Objetos Mutáveis

- Tuplas podem conter objetos mutáveis, como listas.

```
pessoas = ([30, 25, 50], ['Carlos', 'Lucas', 'Maria'])

pessoas[0].append(2)
pessoas[1].append('Enzo')
print(pessoas)
```

```
([30, 25, 50, 2], ['Carlos', 'Lucas', 'Maria', 'Enzo'])
```

# Tuplas com Objetos Mutáveis

- Não alteramos nada na tupla! Modificamos a lista dentro dela, e assim isso foi permitido.

# Tuplas

- Exercício
    - Faça um programa que leia a idade e nome de 10 pessoas, e armazene-os apenas se a idade for maior ou igual a 18. Por fim, adicione-os em uma tupla onde o primeiro index será para os nomes e o segundo para as idades.

# Tuplas

- Exercício
  - Crie um programa que permita ao usuário inserir os nomes e as notas de três provas de quatro alunos.
  - O programa deve calcular a média de notas de cada aluno e armazená-las em uma lista de tuplas, onde cada tupla contém o nome do aluno e sua média de notas.
  - Por fim, exiba a lista de alunos com suas médias.

# Dicionários

# Dicionários

- Uma alternativa mais simples e fácil de manter conteúdos é o Dicionário;
- Possui chave (key) e valor (value);
- A chave funciona como um índice para acessar o conteúdo;
- O conteúdo pode ser qualquer coisa, inclusive outro dicionário;

# Dicionários

- Em uma agenda telefônica com nomes e números, qual seria a chave? E o valor?
- A chave seria o elemento por qual quero fazer o acesso, seria o nome do contato;
- O valor seria o elemento associado a chave, seria o(s) número(s) de telefone.

# Criando Dicionários

- Há algumas maneiras de criar uma dicionário:
  - dicionario = {}
  - dicionario = dict()

Dicionário vazio

  - dicionario = `{'Pedro': 92345678}`

Dicionario iniciado

  - dicionario = `{'Maria': [99887766, 99887755]}`

Dicionario iniciado
Com valor contendo uma lista

# Criando Dicionários com { }

- **Chave** é separada de seu **valor** por dois pontos (**:**);
  - ```
    agenda = {'Pedro': 92345678}
    ```
- Cada par **chave-valor** é separado por vírgula dos pares seguintes:
  - ```
    agenda = {"Maria": [99887766, 99887755],
              "Pedro": [92345678],
              "Joaquim": [99887711, 99665533]}
    ```

# Criando Dicionários com { }

- **Chave** pode ser do tipo String ou Int. **Valor** pode assumir qualquer tipo de variável do python.

```
pessoa = {1: 1.75, 2: 1.95, 3: 1.66}
pessoa_2 = {'Maria': 1.75, 'Joao': 1.95, 'Jose': 1.66}
print(pessoa)
print(pessoa_2)
```

```
{1: 1.75, 2: 1.95, 3: 1.66}
{'Maria': 1.75, 'Joao': 1.95, 'Jose': 1.66}
```

# Acessando Itens do Dicionário

- O acesso sempre é feito pela chave
  - ```
    agenda = {"Maria": [99887766, 99887755],
              "Pedro": [92345678],
              "Joaquim": [99887711, 99665533]}
    print(agenda["Maria"])
    ```
  - ```
    [99887766, 99887755]
    ```

# Acessando Itens do Dicionário

- E se o contato não existir?
  - ```
    agenda = {"Maria": [99887766, 99887755],
              "Pedro": [92345678],
              "Joaquim": [99887711, 99665533]}
    print(agenda["Ana"])
    ```
  - ```
    KeyError: 'Ana'
    ```
- Os dicionários possuem um método específico para busca de valores, o **get()**, no qual podemos passar como parâmetros a chave que queremos e um valor padrão para retornar caso essa chave não seja encontrada.

# Acessando Itens do Dicionário

```
agenda = {"Maria": [99887766, 99887755],
          "Pedro": [92345678],
          "Joaquim": [99887711, 99665533]}


print(agenda.get("Ana", "Não encontrado"))
print(agenda.get("Pedro", "Não encontrado"))
```

```
Não encontrado
[92345678]
```

# Alterando Valor do Conteúdo em Dicionário

- Usando a chave

```
agenda = {"Maria": [99887766, 99887755],
          "Pedro": [92345678],
          "Joaquim": [99887711, 99665533]}


agenda["Pedro"] = [87654433]

print(agenda)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433], 'Joaquim': [99887711, 99665533]}
```

# Alterando Valor do Conteúdo em Dicionário

- Usando a chave

```
agenda = {"Maria": [99887799, 99887755],
          "Pedro": [92345678],
          "Joaquim": [99887711, 99665533]}

agenda["Pedro"].append(87654433)

print(agenda)
```

```
{'Maria': [99887799, 99887755], 'Pedro': [92345678, 87654433], 'Joaquim': [99887711, 99665533]}
```

# Alterando Valor do Conteúdo em Dicionário

- Usando a chave

```
agenda = {"Maria": [99887799, 99887755],
          "Pedro": 92345678,
          "Joaquim": [99887711, 99665533]}

agenda["Pedro"].append(87654433)

print(agenda)
```

```
    agenda["Pedro"].append(87654433)
AttributeError: 'int' object has no attribute 'append'
```

C.E.S.A.R school

# Acrescentando Novos Valores no Dicionário

- Basta usar uma chave não existente

```
agenda = {"Maria": [99887766, 99887755],
          "Pedro": [87654433],
          "Joaquim": [99887711, 99665533]}


agenda["Teresa"] = [65443322]
print(agenda)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433], 'Joaquim': [99887711, 99665533], 'Teresa': [65443322]}
```

# Removendo Valores do Dicionário

- Usando o statement **del**

```
agenda = {"Maria": [99887766, 99887755], "Pedro": [87654433],
          "Joaquim": [99887711, 99665533], "Teresa": [65443322]}

del agenda["Joaquim"]
print(agenda)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433], 'Teresa': [65443322]}
```

# Removendo Valores do Dicionário

- E se a chave não existir?

```
agenda = {"Maria": [99887766, 99887755], "Pedro": [87654433],
          "Teresa": [65443322]}

del agenda["Catarina"]
print(agenda)
```

```
KeyError: 'Catarina'
```

# Removendo Valores do Dicionário

- O método **pop()**, além de remover o elemento com a chave especificada do dicionário, nos retorna o valor desse elemento. Também podemos definir um valor padrão de retorno, para caso a chave não seja encontrada:

```
agenda = {"Maria": [99887766, 99887755], "Pedro": [87654433],
          "Teresa": [65443322]}

print(agenda.pop("Catarina", "Não encontrado"))
print(agenda.pop("Teresa", "Não encontrado"))
print(agenda)
```

```
Não encontrado
[65443322]
{'Maria': [99887766, 99887755], 'Pedro': [87654433]}
```

# Alterando o Nome da Chave em Dicionário

- Usando o método .pop()

```
agenda = {"Maria": [99887799, 99887755],
          "Pedro": [92345678],
          "Joaquim": [99887711, 99665533]}

agenda["Marya"] = agenda.pop("Maria")

print(agenda)
```

```
{'Pedro': [92345678], 'Joaquim': [99887711, 99665533], 'Marya': [99887799, 99887755]}
```

# Tamanho do Dicionário

- O número de itens (chave e valor) que compõem um dicionário pode ser obtido a partir da função **len()**

```
idades = {"Joao":10, "Maria":12, "Alice":4}
print(len(idades))
```

```
3
```

# Verificando Item no Dicionário

- in

```
agenda = {"Maria": [99887766, 99887755], "Pedro": [87654433]}

print("Thiago" in agenda)
print("Maria" in agenda)
print([87654433] in agenda)
print([87654433] in agenda.values())
```

```
False
True
Error
True
```

# Copiando Dicionários

- O que é impresso?

```
agenda_joyce = {"Maria": [99887766, 99887755], "Pedro": [87654433]}
agenda_nova = agenda_joyce
agenda_joyce["Carlos"] = [97777777]

print(agenda_nova)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433], 'Carlos': [97777777]}
```

# Copiando Dicionários

- O que é impresso usando **copy()**?

```
agenda_joyce = {"Maria": [99887766, 99887755], "Pedro": [87654433]}
agenda_nova = agenda_joyce.copy()
agenda_joyce["Carlos"] = [97777777]

print(agenda_nova)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433]}
```

# Copiando Dicionários

- Mas **CUIDADO**. Alterando o conteúdo da lista, o que é impresso?

```
agenda_joyce = {"Maria": [99887799, 99887755], "Pedro": [87654433]}
agenda_nova = agenda_joyce.copy()
agenda_joyce["Carlos"] = [97777777]
agenda_joyce["Pedro"].append(99999999)

print(agenda_nova)
print(agenda_joyce)
```

```
{'Maria': [99887766, 99887755], 'Pedro': [87654433, 99999999]}
{'Maria': [99887766, 99887755], 'Pedro': [87654433, 99999999], 'Carlos': [97777777]}
```

# Copiando Dicionários

- Quando o copy() é usado para criar uma cópia de um dicionário, ele cria uma cópia superficial;
- Em agenda_nova, os valores das chaves já existentes (os objetos que as chaves apontam) serão os mesmos objetos que estão em agenda_joyce;
- Qualquer alteração dos valores das chaves já existentes vai espelhar nos dois dicionários;
- Para realizar uma cópia profunda, é necessário o uso de uma biblioteca externa, chamada “copy”.

```
import copy

agenda_nova = copy.deepcopy(agenda_joyce)
```

# Acessando Dicionários

- items()
  - Retorna uma lista com todos os itens (chave/valor) do dicionário no formato de tupla.

```
notas = { "Joao":[9.0,8.0], "Maria":[10.0] }
print(notas.items())
```

```
dict_items([('Joao', [9.0, 8.0]), ('Maria', [10.0])])
```

# Acessando Dicionários

- keys()
  - Retorna uma lista com todas as chaves do dicionário.

```
notas = { "Joao":[9.0,8.0], "Maria":[10.0] }
print(notas.keys())
```

```
dict_keys(['Joao', 'Maria'])
```

# Acessando Dicionários

- values()
  - Retorna uma lista com todos os valores do dicionário.

```
notas = { "Joao":[9.0,8.0], "Maria":[10.0] }
print(notas.values())
```

```
dict_values([[9.0, 8.0], [10.0]])
```

# Percorrendo Dicionários

- A iteração em elementos de um dicionário é feita a partir da chave.

```
notas = {'Joao': [9.0, 8.0], 'Maria': [10.0, 2.0], 'Pedro': [8.5, 5.0]}
for nome in notas:
    print(notas[nome])
```

```
[9.0, 8.0]
[10.0, 2.0]
[8.5, 5.0]
```

# Percorrendo Dicionários

- A iteração em elementos de um dicionário é feita a partir da chave.

```
notas = {"Joao":[9.0,8.0], "Maria":[10.0, 2.0], "Pedro": [8.5, 5.0]}
for nome in notas:
    media = sum(notas[nome])/len(notas[nome])
    print("A média de", nome, "é:", media)
```

```
A média de Joao é: 8.5
A média de Maria é: 6.0
A média de Pedro é: 6.75
```

# Dicionários

- Vamos praticar?
    - Escreva um programa que solicite o nome e a idade de três pessoas. Adicione essas informações em um dicionário, onde a chave é o nome e o valor a idade. Em seguida, solicite ao usuário que insira um nome e exiba a idade correspondente. Se o nome não estiver no dicionário, exiba uma mensagem informando que o nome não foi encontrado.

# Dicionários

- Vamos praticar?
    - Escreva um programa que conte quantas vezes cada palavra aparece em uma frase. Use um dicionário para armazenar a contagem de cada palavra. A frase pode conter letras maiúsculas e minúsculas, mas as palavras devem ser contadas independentemente da capitalização. Exiba o dicionário.

# Tuplas e Dicionários

- Lista de exercício 07 disponível.