Tipos e formatos de dados (2)

Aula 3

Curso: BIG863 - Basic Python Programming for Ecologists

Professora: Dra. Cecilia F. Fiorini

Supervisor: Prof. Dr. Fernando A. O. Silveira

https://meet.google.com/zdi-ueoz-nsr, 29 de março de 2023

Roteiro*

1. Listas
2. Objetos Multidimensionais
3. Dicionários

*Conteúdo adaptado a partir de material desenvolvido pelo Prof. Zanoni Dias e disponível em https://ic.unicamp.br/ mc102.

ECMVS

Listas

- Podemos utilizar a estrutura de lista em Python para armazenar múltiplos dados.
- Listas podem ser criadas de forma implícita, listando os elementos entre colchetes.

```
1 frutas = ["Abacaxi", "Banana", "Caqui", "Damasco",
2 "Embaúba", "Figo", "Graviola"]
3 numeros = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
4 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
5 dados = ["Carlos", 19, True , "Pedro", "Ana", 1.78, 2001]
```


ECMVS

Listas

- Podemos também declarar uma lista de maneira explícita utilizando a função list.

```
1 a = list(range(10))
2 # a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1 a = list()
2 # a = []
```


ECMVS

Listas

- Podemos também declarar uma lista de maneira explícita utilizando a função list.

```
1 empresas = list(["Toyota", "Volkswagen", "Ford"])
2 # empresas = ['Toyota', 'Volkswagen', 'Ford']
```

```
1 UFMG = list("UFMG")
2 # UFMG = ['U', 'F', 'M', 'G']
3 # Strings são listas de caracteres
```


ECMVS

Selecionando um Elemento

- Podemos acessar o i-ésimo elemento da seguinte forma:

```
1 lista[i - 1]
```

- Essa operação retorna como resposta uma cópia do i-ésimo elemento da lista.
- O primeiro elemento de uma lista ocupa a posição 0.

ECMVS

Selecionando um Elemento

- Selecionando o primeiro elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[0])
3 # A
```

- Selecionando o segundo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[1])
3 # B
```


ECMVS

Selecionando um Elemento

- Selecionando o quinto elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[4])
3 # E
```


ECMVS

Selecionando um Elemento

- Podemos também acessar os elementos de uma lista, de trás para frente, da seguinte forma:

```
1 lista[-i]
```

- Como resposta, obtemos uma cópia do i-ésimo elemento da lista, de trás para frente.
- O último elemento de uma lista ocupa a posição -1.

Selecionando um Elemento

- Selecionando o último elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-1])
3 # H
```

- Selecionando o penúltimo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-2])
3 # G
```


ECMVS

Selecionando um Elemento

- Selecionando o antepenúltimo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-3])
3 # F
```


ECMVS

Selecionando um Elemento

- Caso seja informada uma posição inválida, será gerado um erro:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[8])
3 # IndexError: list index out of range
```

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-9])
3 # IndexError: list index out of range
```


ECMVS

Determinando o Tamanho de uma Lista

- A função len recebe como parâmetro uma lista e retorna o seu tamanho (número de elementos).

```
1  frutas = ["Abacaxi", "Banana", "Caqui", "Damasco",
2  "Embaúba", "Figo", "Graviola"]
3  numeros = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
4  letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
5  print(len(frutas))
6  # 7
7  print(len(numeros))
8  # 11
9  print(len(letras))
10 # 8
```

Selecionando Intervalos

- Podemos selecionar um trecho de uma lista:

```
1 lista[start:stop:step]
```

- O trecho inicia na posição start (inclusivo) e vai até a posição stop (exclusivo), selecionando de step em step os elementos da lista.
- Esta operação retorna uma nova lista, à qual normalmente nos referimos como uma sublista.
- Caso os parâmetros start, stop ou step não sejam especificados, Python automaticamente assume que seus valores são a posição do primeiro elemento (0), o tamanho da lista (len(lista)) e um (1), respectivamente.

ECMVS

Selecionando Intervalos

- Selecionando do segundo até o quarto elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[1:4])
3 # ["B", "C", "D"]
```

- Selecionando os três primeiros elementos de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[:3])
3 # ["A", "B", "C"]
```


ECMVS

Selecionando Intervalos

- Selecionando os quatro últimos elementos de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-4:])
3 # ["E", "F", "G", "H"]
```

- Selecionando os elementos das posições pares de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[::2])
3 # ["A", "C", "E", "G"]
```


ECMVS

Selecionando Intervalos

- Selecionando os elementos das posições ímpares de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[1::2])
3 # ["B", "D", "F", "H"]
```

- Obtendo os elementos de uma lista, em ordem inversa:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[::-1])
3 # ["H", "G", "F", "E", "D", "C", "B", "A"]
```


ECMVS

Alterando um Elemento

- Podemos alterar o i-ésimo elemento de uma lista da seguinte forma:

```
1 lista[i - 1] = valor
```

```
1 empresas = ["Apple", "Samsung", "LG", "Facebook"]
2 empresas[2] = "Google"
3 print(empresas)
4 # ['Apple', 'Samsung', 'Google', 'Facebook']
```


ECMVS

Alterando um Trecho da Lista

- Podemos alterar um trecho de uma lista associando uma nova lista para esse trecho, inclusive uma lista vazia.

```
1 lista[start:stop] = [valor_1, ..., valor_n]
```

Alterando um Trecho da Lista

```
1  lista = [0, 1, 2, 3, 4, 5]
2  lista[2:4] = ["A", "B"]
3  print(lista)
4  # [0, 1, 'A', 'B', 4, 5]
5  lista[2:4] = [8, 8, 8]
6  print(lista)
7  # [0, 1, 8, 8, 8, 4, 5]
8  lista[4:6] = []
9  print(lista)
10 # [0, 1, 8, 8, 5]
```


ECMVS

Verificando a Inclusão de um Elemento

- Podemos verificar se um elemento está ou não em uma lista utilizando o operador de teste de inclusão in.

```
1 elemento in lista
```

- Esse operador retorna True ou False caso o elemento esteja ou não na lista, respectivamente.

Verificando a Inclusão de um Elemento

```
1 top5 = ["Black Mirror", "Breaking Bad", "Friends",
2 "Game of Thrones", "The Big Bang Theory"]
3 print("House MD" in top5)
4 # False
5 print("Game of Thrones" in top5)
6 # True
7 print("friends" in top5)
8 # False
```


ECMVS

Inserindo Elementos

- Podemos inserir novos elementos em uma lista utilizando o método append.
- O método append recebe como parâmetro um objeto que será inserido no final da lista.

```
1  animais = []
2  animais.append("Gato")
3  print(animais)
4  # ['Gato']
5  animais.append("Cachorro")
6  print(animais)
7  # ['Gato', 'Cachorro']
8  animais.append("Coelho")
9  print(animais)
10 # ['Gato', 'Cachorro', 'Coelho']
```


ECMVS

Inserindo Elementos

- Para inserir um novo elemento em uma posição específica de uma lista utilizamos o método insert.
- O método insert recebe como parâmetros uma posição e um objeto que será inserido na posição especificada.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição seguinte. Dessa forma, nenhum elemento é removido.

Inserindo Elementos

```
1 frutas = ["Abacaxi", "Banana", "Damasco"]
2 frutas.insert(2, "Caqui")
3 print(frutas)
4 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco']
5 frutas.insert(len(frutas), "Embaúba")
6 print(frutas)
7 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco', 'Embaúba']
```


ECMVS

Obtendo a Posição de um Elemento

- O método index é utilizado para obter a posição de um elemento em uma lista.
- Como parâmetro, o método index recebe um elemento a ser buscado na lista.
- A posição da primeira ocorrência do elemento especificado como parâmetro é retornada como resposta.
- Caso o elemento não esteja na lista, um erro será gerado.

Obtendo a Posição de um Elemento

```
1 cinema = ["Sony Pictures", "Walt Disney",
2 "Universal Pictures", "Warner"]
3 print(cinema.index("Warner"))
4 # 3
5 print(cinema.index("Disney"))
6 # ValueError: 'Disney' is not in list
```

Removendo Elementos

- Podemos remover um elemento de uma lista utilizando o método remove.
- O método remove recebe como parâmetro o elemento a ser removido da lista.
- A primeira ocorrência do elemento especificado como parâmetro é removida da lista.
- Caso o elemento não esteja na lista, um erro será gerado.

ECMVS

Removendo Elementos

```
1 paises = ["Argentina", "Argentina", "Brasil", "Canadá"]
2 paises.remove("Argentina")
3 print(paises)
4 # ['Argentina', 'Brasil', 'Canadá']
5 paises.remove("Dinamarca")
6 # ValueError: list.remove(x): x not in list
```


ECMVS

Removendo Elementos

- Outra opção para remover um elemento de uma lista é utilizando o método pop.
- O método pop recebe como parâmetro a posição do elemento a ser removido da lista. Caso o parâmetro seja omitido, o último elemento da lista será removido.
- Como resposta, o método retorna o elemento removido.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição anterior.

ECMVS

Removendo Elementos

```
1 paises = ["Argentina", "Dinamarca", "Brasil", "Canadá"]
2 print(paises.pop(1))
3 # Dinamarca
4 print(paises)
5 # ['Argentina', 'Brasil', 'Canadá']
6 print(paises.pop())
7 # Canadá
8 print(paises)
9 # ['Argentina', 'Brasil']
```


ECMVS

Contando Elementos

- O método count é utilizado para contar a quantidade de vezes que um elemento ocorre em uma lista.
- O método count recebe como parâmetro um elemento.
- Como resposta, o método retorna a quantidade de ocorrências do elemento na lista.

```
1 paises = ["Brasil", "brasil", "Brazil", "Brasil"]
2 print(paises.count("Brasil"))
3 # 2
4 print(paises.count("brasil"))
5 # 1
6 print(paises.count("Brazil"))
7 # 1
8 print(paises.count("brazil"))
9 # 0
```

Invertendo a Ordem dos Elementos

- O método reverse inverte a ordem dos elementos de uma lista.
- O método reverse não recebe nenhum parâmetro e modifica automaticamente a lista.

```
1 semana = ["Domingo", "Segunda", "Terça", "Quarta",
2 "Quinta", "Sexta", "Sábado"]
3 print(semana)
4 # ['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta',
5 # 'Sexta', 'Sábado']
6 semana.reverse()
7 print(semana)
8 # ['Sábado', 'Sexta', 'Quinta', 'Quarta', 'Terça',
9 # 'Segunda', 'Domingo']
```


ECMVS

Ordenando Listas

- Uma lista pode ser ordenada utilizando o método sort.
- O método sort possui o parâmetro opcional reverse, que indica se a lista deve ser ordenada de forma crescente (False) ou decrescente (True). Por padrão o valor desse parâmetro é False (ordenação crescente).

```
1  a = [5, 3, 1, 4, 2, 6]
2  a.sort()
3  print(a)
4  # [1, 2, 3, 4, 5, 6]
5  a.reverse()
6  print(a)
7  # [6, 5, 4, 3, 2, 1]
```


ECMVS

Ordenando Listas

- Podemos usar a função sorted para obter uma cópia ordenada de uma lista, sem alterar a lista original.

```
1  a = [5, 3, 1, 4, 2, 6]
2  print(sorted(a))
3  # [1, 2, 3, 4, 5, 6]
4  print(a)
5  # [5, 3, 1, 4, 2, 6]
6  print(sorted(a)[::-1])
7  # [6, 5, 4, 3, 2, 1]
```

Copiando Listas

- Podemos atribuir uma lista para diferentes variáveis, mas as variáveis estarão relacionadas a mesma lista (objeto).
- Isso implica que qualquer modificação feita em uma variável afetará todas as outras.

```
1  a = [1]
2  b = a
3  b.append(2)
4  c = b
5  c.append(3)
6  print(a)
7  # [1, 2, 3]
8  print(b)
9  # [1, 2, 3]
10 print(c)
11 # [1, 2, 3]
```

Copiando Listas

- Se quisermos uma cópia independente de uma lista podemos utilizar o método copy.
- O método copy retorna uma cópia da lista.
- Esta cópia pode ser atribuída a uma variável.

```
1  a = [1]
2  b = a.copy()
3  b.append(2)
4  c = b.copy()
5  c.append(3)
6  print(a)
7  # [1]
8  print(b)
9  # [1, 2]
10 print(c)
11 # [1, 2, 3]
```

Clonando Listas

- Podemos clonar uma lista, para obter uma cópia independente.
- Uma lista pode ser clonada utilizando o operador de seleção de intervalos [:] ou com a função list().
- Este clone pode ser atribuído a uma variável.

```
1  a = [1]
2  b = a[:]
3  b.append(2)
4  c = list(b)
5  c.append(3)
6  print(a)
7  # [1]
8  print(b)
9  # [1, 2]
10 print(c)
11 # [1, 2, 3]
```

Concatenando Listas

- O operador + pode ser utilizado com listas com o objetivo de concatená-las.
- Como resultado, uma nova lista é obtida seguindo a ordem da concatenação realizada.

```
1  a = [1, 2]
2  b = [3, 4]
3  c = [5, 6]
4  print(a + b + c)
5  # [1, 2, 3, 4, 5, 6]
6  print(c + b + a)
7  # [5, 6, 3, 4, 1, 2]
8  print(b + c + a)
9  # [3, 4, 5, 6, 1, 2]
```


ECMVS

Funções Úteis para Listas Numéricas

- A função min retorna o menor valor em uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(min(numeros))
3 # 1.43
```

- A função max retorna o maior valor em uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(max(numeros))
3 # 5.32
```

Funções Úteis para Listas Numéricas

- A função sum retorna a soma de todos os elementos de uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(sum(numeros))
3 # 14.61
```


ECMVS

Matrizes e Objetos Multidimensionais

- Matrizes e objetos multidimensionais são generalizações de objetos simples vistos anteriormente (listas e tuplas).
- Esses tipos de dados nos permitem armazenar informações mais complexas em uma única variável.
- Exemplo de informações/operações que podem ser armazenadas/manipuladas utilizando matrizes e objetos multidimensionais:
  - Matemática: operações com matrizes.
  - Processamento de imagem: cor de cada pixel presente na imagem.
  - Mapas e geolocalização: informação sobre o relevo em cada ponto do mapa.
  - Jogos de tabuleiro: Xadrez, Damas, Go, Batalha Naval, etc.

Matrizes

- Uma lista pode conter elementos de tipos diferentes.
- Uma lista pode conter inclusive outras listas.
- Exemplo de declaração de uma matriz 2×2:

```
1  obj = [
2           [1, 2, 3, 4],
3           [5, 6],
4           [7, 8, 9]
5          ]
```


ECMVS

Declaração de Matrizes

- Uma matriz é um objeto bidimensional, formada por listas, todas do mesmo tamanho.
- Sua representação é dada na forma de uma lista de listas (a mesma ideia pode ser aplicada para tuplas).
- Exemplo de declaração de uma matriz 3×4:

```
1  matriz = [
2    [11, 12, 13, 14], # linha 1
3    [21, 22, 23, 24], # linha 2
4    [31, 32, 33, 34] # linha 3
5    ]
```


ECMVS

Acessando Elementos de uma Matriz

- Note que uma matriz é que uma lista de listas.
- Podemos acessar um elemento de uma matriz, localizado em uma determinada linha e coluna, da seguinte forma:

```
1 matriz = matriz[linha][coluna]
2 # Lembrete: linhas e colunas são numeradas
3 # a partir da posição zero
```


ECMVS

Acessando Elementos de uma Matriz

```
1 matriz = matriz[linha][coluna]
2 # Lembrete: linhas e colunas são numeradas
3 # a partir da posição zeromatriz = [[1, 2, 3], [4, 5, 6], [7, 8,
      9]]
4 print(matriz[0][2])
5 # 3
6 print(matriz[2][1])
7 # 8
```

Acessando Elementos de uma Matriz

- Similar ao que vimos em listas e tuplas, caso ocorra uma tentativa de acessar uma posição inexistente da matriz, um erro será gerado.

```
1 matriz = matriz[linha][coluna]
2 # Lembrete: linhas e colunas são numeradas
3 # a partir da posição zero
```


ECMVS

Alterando Elementos de uma Matriz

- Podemos alterar um elemento de uma matriz, localizado em uma determinada linha e coluna, da seguinte forma:

```
matriz[linha][coluna] = valor
```

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matriz[0][0] = 0
matriz[2][2] = 10
print(matriz)
# [[0, 2, 3], [4, 5, 6], [7, 8, 10]]
```

Criando uma Cópia de uma Matriz

- Para criar uma cópia de uma matriz, precisamos criar uma nova matriz com as cópias de cada uma das linhas da matriz original.

```
1  A = [[1, 2], [3, 4]]
2  2
3  B = [linha.copy() for linha in A]
4  4
5  B[0][0] = 0
6  6
7  print(A)
8  # [[1, 2], [3, 4]]
9  print(B)
10 # [[0, 2], [3, 4]]
```

Objetos Multidimensionais

- Até agora criamos matrizes bidimensionais, mas podemos criar objetos com mais dimensões.
- Podemos criar objetos com d dimensões utilizando a mesma ideia de listas de listas.
- Exemplo de um objeto com dimensões $2 \times 2 \times 2$:

```
1  obj = [
2    [[1, 2], [3, 4]],
3    [[5, 6], [7, 8]]
4    ]
```

ECMVS

Objetos Multidimensionais

- Podemos acessar um elemento em um objeto com dimensões $d_1 \times d_2 \times \cdots \times d_n$ da seguinte forma:

```
objeto[index_1][index_2]...[index_n]
```

```
obj = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] # 2 x 2 x 2
print(obj[0][0][0])
# 1
print(obj[1][0][0])
# 5
```


ECMVS

Objetos Multidimensionais

- Podemos alterar um elemento em um objeto com dimensões $d1 \times d2 \times \cdots \times dn$ da seguinte forma:

```
objeto[index_1][index_2]...[index_n] = valor
```

```
obj = [[[0, 0], [0, 0]], [[0, 0], [0, 0]]] # 2 x 2 x 2
obj[1][0][1] = 5
obj[0][1][0] = 3
print(obj)
# [[[0, 0], [3, 0]], [[0, 5], [0, 0]]]
```


ECMVS

Dicionários

- Dicionários são estruturas de chave-valor, ou seja, os valores (dados) estão sempre associados a uma chave.

```
dicionario = {} # dicionario vazio.
print(type(dicionario))
# <class 'dict '>
```

```
localizacao = {
"Lat": -22.817087 ,
"Long": -47.069750
}
print(type(localizacao))
# <class 'dict '>
```


ECMVS

Dicionários

- Podemos também declarar um dicionário de maneira explícita utilizando a função dict.

```
1 dicionario = dict ({}) # dicionario vazio.
2 print(type(dicionario))
3 # <class 'dict '>
```

```
1 localizacao = dict ({
2 "Lat": -22.817087 ,
3 "Long": -47.069750
4 })
5 print(type(localizacao))
6 # <class 'dict '>
```


ECMVS

Dicionários

- Chaves e valores em um dicionário podem ser de diferentes tipos de dados (int, float, bool, entre outros).

```
1 dicionario = {
2 1.2: True ,
3 123: "Um Dois Três",
4 "cat": "dog",
5 ("X", "Y"): (2, 3, 5)
6 }
7 print(dicionario)
8 # {1.2: True , 123: 'Um Dois Três ', 'cat ': 'dog ',
9 # ('X', 'Y '): (2, 3, 5)}
```


ECMVS

Dicionários

- Geralmente as chaves são mantidas na ordem em que o dicionário é criado ou alterado.

```
1 dicionario = {
2 "Z": 1,
3 "A": 2,
4 "C": 3
5 }
6 print(dicionario)
7 # {'Z ': 1, 'A ': 2, 'C ': 3}
```

Dicionários

- Podemos acessar um valor do dicionário da seguinte forma:

```
1 <dicionário >[<chave >]
```


ECMVS

Dicionários

- Essa operação retorna o valor associado à chave informada.

```
1  dicionario = {
2  "Nome": "Ash Ketchum",
3  "Idade": 10,
4  "Profissão": "Treinador Pokémon"
5  }
6  print(dicionario["Nome"])
7  # Ash Ketchum
8  print(dicionario["Idade"])
9  # 10
10 print(dicionario["Profissão"])
11 # Treinador Pokémon
```


ECMVS

Dicionários

- E se informarmos uma chave que não está no dicionário? O que acontece?

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon"
5 }
6 print(dicionario["Cidade"])
7 # KeyError: 'Cidade'
```


ECMVS

Dicionários

- Similar ao que vimos em listas, podemos verificar se uma chave está presente ou não em um dicionário utilizando o operador de inclusão in.

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon"
5 }
6 print("Cidade" in dicionario)
7 # False
8 print("Nome" in dicionario)
9 # True
```


ECMVS

Dicionários

- O método get é outra forma para obter valores de um dicionário.
- Ele recebe como parâmetro a chave associada ao valor desejado.
- Caso a chave não seja encontrada no dicionário, será retornado None como resposta.

Dicionários

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon"
5 }
6 print(dicionario.get("Nome"))
7 # Ash Ketchum
8 print(dicionario.get("Cidade"))
9 # None
```


ECMVS

Dicionários

- O tamanho de um dicionário também pode ser verificado através da função len.
- Cada conjunto de chave e valor corresponde a um elemento.

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon"
5 }
6 print(len(dicionario))
7 # 3
```


ECMVS

Dicionários

- Novos valores podem ser adicionados dinamicamente a um dicionário informando o novo par chave-valor.

```
1 <dicionário >[<nova_chave >] = <novo_valor >
```


ECMVS

Dicionários

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon"
5 }
6 dicionario["Cidade"] = "Pallet"
7 print(dicionario)
8 # {'Nome ': 'Ash Ketchum ', 'Idade ': 10, 'Profissão ':
9 # 'Treinador Pokémon ', 'Cidade ': 'Pallet '}
```


ECMVS

Dicionários

- Para atualizar um valor já existente em um dicionário, basta atribuir à chave o valor atualizado.

```
1 <dicionário >[<chave >] = <novo_valor >
```


ECMVS

Removendo Valores

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 10,
4 "Profissão": "Treinador Pokémon",
5 "Cidade": "Pallet"
6 }
7 dicionario["Idade"] = 12
8 print(dicionario["Idade"])
9 # 12
```

Dicionários

- Para remover um valor (e chave associada) a um dicionário podemos utilizar o método pop.
- O método pop recebe como parâmetro a chave que está associada ao valor que deve ser removido.
- Como resposta o método retorna o valor que foi removido do dicionário.
- Caso a chave informada como parâmetro não esteja no dicionário, um erro será gerado.

Removendo Valores

- Removendo um valor de um dicionário utilizando o método pop:

```
1  dicionario = {
2  "Nome": "Ash Ketchum",
3  "Idade": 12,
4  "Profissão": "Treinador Pokémon",
5  "Cidade": "Pallet"
6  }
7  profissao = dicionario.pop("Profissão")
8  print(profissao)
9  # Treinador Pokémon
10 print(dicionario)
11 # {'Nome ': 'Ash Ketchum ', 'Idade ': 12, 'Cidade ': 'Pallet '}
```


ECMVS

Removendo Valores

- Outra forma de remover um valor (e chave associada) em um dicionário é utilizando a declaração del. A declaração del pode ser utilizada da seguinte forma:

```
1 del <dicionario >[<chave >]
```

- Caso a chave informada como parâmetro não esteja no dicionário, um erro será gerado.

ECMVS

Removendo Valores

- Removendo um valor de um dicionário utilizando a declaração del:

```
1 dicionario = {
2 "Nome": "Ash Ketchum",
3 "Idade": 12,
4 "Profissão": "Treinador Pokémon",
5 "Cidade": "Pallet"
6 }
7 del dicionario["Profissão"]
8 print(dicionario)
9 # {'Nome ': 'Ash Ketchum ', 'Idade ': 12, 'Cidade ': 'Pallet '}
```


ECMVS

Atualizando um Dicionário

- O método update pode ser utilizado para atualizar um dicionário.
- Esse método recebe como parâmetro um outro dicionário.
- O dicionário original é modificado com base no dicionário informado como parâmetro, de tal forma que os valores das chaves previamente existentes no primeiro são atualizados e novos valores são adicionados para as novas chaves.

```
1 dic_a = {"A": "Avião", "B": "Barco"}
2 dic_b = {"B": "Balão", "C": "Carro"}
3 dic_a.update(dic_b)
4 print(dic_a)
5 # {'A ': 'Avião ', 'B ': 'Balão ', 'C ': 'Carro '}
```


ECMVS

Chaves e Valores de um Dicionário

- O método keys retorna uma estrutura com as chaves do dicionário, que pode ser convertida para uma lista.

```
1 lugar = {"Lat": -22.817087 , "Long": -47.069750}
2 print(lugar.keys ())
3 # dict_keys (['Lat ', 'Long '])
4 print(list(lugar.keys ()))
5 # ['Lat ', 'Long ']
```


ECMVS

Chaves e Valores de um Dicionário

- O método values retorna uma estrutura com os valores do dicionário, que pode ser convertida para uma lista.

```
1 lugar = {"Lat": -22.817087 , "Long": -47.069750}
2 print(lugar.values ())
3 # dict_values ([ -22.817087 , -47.06975])
4 print(list(lugar.values ()))
5 # [ -22.817087 , -47.06975]
```


ECMVS

Copiando Dicionários

- Similar ao que vimos em listas, podemos atribuir um dicionário para diferentes variáveis, mas as variáveis estarão relacionadas ao mesmo dicionário (objeto).

```
1 dic_a = {"Nome": "João", "Idade": 18}
2 print(dic_a)
3 # {'Nome ': 'João ', 'Idade ': 18}
4 dic_b = dic_a
5 dic_b["Nome"] = "Maria"
6 print(dic_b)
7 # {'Nome ': 'Maria ', 'Idade ': 18}
8 print(dic_a)
9 # {'Nome ': 'Maria ', 'Idade ': 18}
```


ECMVS

Copiando Dicionários

- Similar ao que vimos em listas, se quisermos criar uma cópia independente de um dicionário devemos utilizar o método copy.

```
1 dic_a = {"Nome": "João", "Idade": 18}
2 print(dic_a)
3 # {'Nome ': 'João ', 'Idade ': 18}
4 dic_b = dic_a.copy ()
5 dic_b["Nome"] = "Maria"
6 print(dic_b)
7 # {'Nome ': 'Maria ', 'Idade ': 18}
8 print(dic_a)
9 # {'Nome ': 'João ', 'Idade ': 18}
```

Criando Dicionários com Listas de Chaves e Valores

- É possível criar um dicionário a partir de duas listas com o auxílio da função zip.
- A função zip recebe dois parâmetros, o primeiro é uma lista contendo as chaves desejadas para o dicionário, enquanto o segundo é uma lista contendo os respectivos valores.

```
1 pessoas = ["Alice", "Beatriz", "Carlos"]
2 telefones = ["99999 -0000", "99999 -1111", "99999 -2222"]
3 contatos = dict(zip(pessoas , telefones))
4 print(contatos)
5 # {'Alice ': '99999-0000' ,
6 # 'Beatriz ': '99999-1111' ,
7 # 'Carlos ': '99999 -2222 '}
```


ECMVS