Tipos de dados: Coleções

Instrutor: Tarik Ponciano

Senac Fecomércio Sesc

Links da Disciplina

1. Discord: https://discord.gg/wt5CVZZWJs

2. Drive: https://drive.google.com/drive/folders/1hOl0DaPeAor7gnhKBUIlZ5n8lLRDNvUY?usp=sharing

3. Github: https://github.com/TarikPonciano/Programador-de-Sistema-SENAC

Tipos de Dados

Um tipo de dado nada mais que é algo do mundo real que pode ser representado computacionalmente. Por exemplo, os números que pertencem ao conjunto dos números inteiros, os números que pertencem ao conjunto dos números reais, letras, caracteres especiais, acentuação, pontuação, palavras, etc.

As linguagens de programação implementam formas de representar e manipular esses dados, que podem ser classificados em dois grandes grupos: os tipos de dados primitivos e os tipos de dados não primitivos.

Tipos de Dados

Os tipos de dados primitivos são os tipos básicos que devem ser implementados por todas as linguagens de programação, como os números reais, inteiros, booleanos, caracteres e strings.

Os tipos de dados não primitivos, normalmente são os vetores, matrizes, classes, enumerações, etc., que costumam ser estruturas de dados mais complexas do que os tipos de dados primitivos.

Tipos de Dados no Python

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

https://www.w3schools.com/python/python_datatypes.asp

Senac
Fecomércio
Sesc

Coleções

As coleções permitem armazenar múltiplos itens dentro de uma única unidade, que funciona como um container. Hoje veremos as quatro coleções utilizadas em Python, que são as listas, tuplas, dicionários e conjuntos(sets).

1. Lista(list): É uma coleção ordenada e mutável. Permite membros duplicados.

2. Tupla(tuple): É uma coleção ordenada e imutável. Permite membros duplicados.

3. Conjunto(set): É uma coleção não-ordenada, imutável* e não indexada. Não pode conter membros duplicados.

4. Dicionário(Dictionary): É uma coleção ordenada** e mutável. Não pode conter membros duplicados.

Lista

Lista é uma coleção de valores indexada, em que cada valor é identificado por um índice. O primeiro item na lista está no índice 0, o segundo no índice 1 e assim por diante.

Para criar uma lista com elementos deve-se usar colchetes e adicionar os itens entre eles separados por vírgula, como mostra o **Código:**

```
1 programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
2 print(type(programadores)) # type 'list'
3 print(len(programadores)) # 5
4 print(programadores[4]) # Luana
```

Lista

Outra característica das listas é que elas podem possuir diferentes tipos de elementos na sua composição. Isso quer dizer que podemos ter strings, booleanos, inteiros e outros tipos diferentes de objetos na mesma lista.

```
1 | aluno = ['Murilo', 19, 1.79] # Nome, idade e altura
2 |
3 | print(type(aluno)) # type 'list'
4 | print(aluno) # ['Murilo', 19, 1.79]
```

Lista

```
1 programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
2 print(type(programadores)) # type 'list'
3 print(len(programadores)) # 5
4 print(programadores[4]) # Luana
```

Na linha 1 criamos uma variável do tipo lista chamada programadores contendo cinco nomes como os seus itens. Como já visto antes a função **type()** (Linha 2) traz o tipo de variável e **len()** (Linha 3) o tamanho do objeto. Observe que na linha 4 imprimimos um item da lista acessando o índice 4.

Lista

Outra característica das listas no Python é que elas são mutáveis, podendo ser alteradas depois de terem sido criadas. Em outras palavras, podemos adicionar, remover e até mesmo alterar os itens de uma lista.

```
1  programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
2  print(programadores) # ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
3
4  programadores[1] = 'Carolina'
5  print(programadores) # ['Victor', 'Carolina', 'Samuel', 'Caio', 'Luana']
```

Primeiro, criamos uma lista contendo algumas strings e depois imprimimos o seu valor na linha 2. Após isso, acessamos um dos elementos dela e alteramos o valor dele para “Carolina” na linha 4.

Lista

Além de alterar elementos em listas, também é possível adicionar itens nelas, pois já vêm com uma coleção de métodos predefinidos que podem ser usados para manipular os objetos que ela contém. No caso de adicionar elementos, podemos usar o método append(), como veremos no **Código:**

```
1  programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
2  print(programadores) # ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana
3
4  programadores.append('Renato')
5  print(programadores) # ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana', 'Renato']
```

Na linha 1 criamos a lista e na linha 2 a imprimimos. Na linha 3 usamos o método append(), que adiciona elementos no final de uma lista. Quando imprimimos a lista na linha 4 vemos que ela exibirá o item adicionado na última posição.

Lista

Há outra forma de adicionarmos itens na lista, que é através do método insert(). Ele usa dois parâmetros: o primeiro para indicar a posição da lista em que o elemento será inserido e o segundo para informar o item a ser adicionado na lista.

```
1  programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
2  programadores.insert(1, 'Rafael')
3
4  print(programadores) # ['Victor', 'Rafael', 'Juliana', 'Samuel', 'Caio', 'Luana']
```

Lista

Assim como podemos adicionar itens em nossa lista, também podemos retirá-los. E para isso temos dois métodos: remove() para a remoção pelo valor informado no parâmetro, e pop() para remoção pelo índice do elemento na lista. Vejamos como isso funciona na lista de programadores que estamos usando como exemplo:

```python
programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
print(programadores) # ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']

programadores.remove('Victor')
print(programadores) # ['Juliana', 'Samuel', 'Caio', 'Luana']
```

```python
programadores = ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']
print(programadores) # ['Victor', 'Juliana', 'Samuel', 'Caio', 'Luana']

programadores.pop(0)
print(programadores) # ['Juliana', 'Samuel', 'Caio', 'Luana']
```

Tuplas

Tupla é uma estrutura de dados semelhante a lista. Porém, ela tem a característica de ser imutável, ou seja, após uma tupla ser criada, ela não pode ser alterada.

```
1  times_rj = ('Botafogo', 'Flamengo', 'Fluminense', 'Vasco')
2
3  print(type(times_rj)) # class='tuple'
4  print(times_rj) # ('Botafogo', 'Flamengo', 'Fluminense', 'Vasco')
```

Acima vemos o uso de uma tupla no Python delimitada por parênteses na sua sintaxe. Na linha 1, a variável times_rj recebe quatro objetos do tipo string. Na linha 3, imprimimos o tipo da variável, que é uma tupla. E na linha 4, imprimimos também o conteúdo de times_rj.

Tuplas

Assim como é feito nas listas, podemos acessar um determinado valor na tupla pelo seu índice:

```
1  times_rj = ('Botafogo', 'Flamengo', 'Fluminense', 'Vasco')
2
3  print(times_rj[2]) # Fluminense
```

Tuplas

Uma observação a ser feita no uso de uma tupla é que se ela tiver um único item, é necessário colocar uma vírgula depois dele, pois caso contrário, o objeto que vamos obter é uma string, porque o valor do item é do tipo string.

```
1  objeto_string = ('tesoura')
2  objeto_tupla = ('tesoura',)
3
4  print(type(objeto_string)) # class 'str'
5  print(type(objeto_tupla)) # class 'tuple'
```

Tuplas

O fato da tupla ser imutável faz com que os seus elementos não possam ser alterados depois dela já criada. Vamos usar a tupla vogais para mostrar um exemplo desse tipo.

```
1  vogais = ('a', 'e', 'i', 'o', 'u')
2
3  vogais[1] = 'E'
```

Tuplas

O fato da tupla ser imutável faz com que os seus elementos não possam ser alterados depois dela já criada. Vamos usar a tupla vogais para mostrar um exemplo desse tipo.

```
1  vogais = ('a', 'e', 'i', 'o', 'u')
2
3  vogais[1] = 'E'
```

O Código exibirá o erro TypeError: 'tuple' object does not support item assignment.

Veja que não é possível fazer alteração nas tuplas. Diferentemente do que acontece com as listas, não podemos trocar os elementos de um objeto do tipo tupla, pois se trata de uma sequência imutável.

Tuplas

As tuplas devem ser usadas em situações em que não haverá necessidade de adicionar, remover ou alterar elementos de um grupo de itens. Exemplos bons seriam os meses do ano, os dias da semana, as estações do ano etc. Nesses casos, não haverá mudança nesses itens (pelo menos isso é muito improvável).

Dicionários

Os dicionários representam coleções de dados que contém na sua estrutura um conjunto de pares chave/valor, nos quais cada chave individual tem um valor associado. Esse objeto representa a ideia de um mapa, que entendemos como uma coleção associativa desordenada. A associação nos dicionários é feita por meio de uma chave que faz referência a um valor.

```
1  dados_cliente = {
2      'Nome': 'Renan',
3      'Endereco': 'Rua Cruzeiro do Sul',
4      'Telefone': '982503645'
5  }
6
7  print(dados_cliente['Nome']) # Renan
```

Senac
Fecomércio
Sesc

Dicionários

```
1  dados_cliente = {
2      'Nome': 'Renan',
3      'Endereco': 'Rua Cruzeiro do Sul',
4      'Telefone': '982503645'
5  }
6
7  print(dados_cliente['Nome']) # Renan
```

A estrutura de um dicionário é delimitada por chaves, entre as quais ficam o conteúdo desse objeto. Veja que é criada a variável dados_cliente, à qual é atribuída uma coleção de dados que, nesse caso, trata-se de um dicionário. Na linha 7 do **Código**, imprimimos o conteúdo que é associado ao índice "Nome", trazendo o resultado Renan.

Nas listas e tuplas acessamos os dados por meio dos índices. Já nos dicionários, o acesso aos dados é feito por meio da chave associada a eles.

Dicionários

Para adicionar elementos num dicionário basta associar uma nova chave ao objeto e dar um valor a ser associado a ela.

```
dados_cliente = {
    'Nome': 'Renan',
    'Endereco': 'Rua Cruzeiro do Sul',
    'Telefone': '982503645'
}

print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul',
   'Telefone': '982503645'}

dados_cliente['Idade'] = 40

print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul',
  'Telefone': '982503645', 'Idade': 40}
```

Dicionários

Para remover um item do dicionário, podemos usar o método pop()

```
1  dados_cliente = {
2      'Nome': 'Renan',
3      'Endereco': 'Rua Cruzeiro do Sul',
4      'Telefone': '982503645'
5  }
6
7  print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul',
8    'Telefone': '982503645'}
9
10 dados_cliente.pop('Telefone', None)
11
12 print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul'}
```

Dicionários

Na linha 9 temos o uso do método pop(), usado para remover o item 'Telefone' do dicionário dados_clientes. Temos na chamada do método o parâmetro None, que é passado depois da chave a ser removida. O None serve para que a mensagem de erro KeyError não apareça devido a remoção de uma chave inexistente.

```
1  dados_cliente = {
2      'Nome': 'Renan',
3      'Endereco': 'Rua Cruzeiro do Sul',
4      'Telefone': '982503645'
5  }
6
7  print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul',
8    'Telefone': '982503645'}
9
10 dados_cliente.pop('Telefone', None)
11
12 print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul'}
```

Senac Fecomércio Sesc

Dicionários

Também poderíamos usar a palavra-chave del, que remove uma chave e o valor associado a ela no dicionário. Isso se faz por meio da passagem no parâmetro

```
1  dados_cliente = {
2      'Nome': 'Renan',
3      'Endereco': 'Rua Cruzeiro do Sul',
4      'Telefone': '982503645'
5  }
6
7  print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul',
8    'Telefone': '982503645'}
9
10 del dados_cliente['Telefone']
11
12 print(dados_cliente) # {'Nome': 'Renan', 'Endereco': 'Rua Cruzeiro do Sul'}
```

Conjuntos

https://www.w3schools.com/python/python_sets.asp

Funções para coleções

min() e max()

Veremos a seguir, as funções min() e max()

```
1  numeros = [15, 5, 0, 20, 10]
2  nomes = ['Caio', 'Alex', 'Renata', 'Patrícia', 'Bruno']
3
4  print(min(numeros)) # 0
5  print(max(numeros)) # 20
6  print(min(nomes)) # Alex
7  print(max(nomes)) # Renata
```

Nesse código temos duas listas com os nomes numeros e nomes. A primeira lista trabalha com números, então a função min() retorna o menor valor dela, enquanto que a função max() retorna o maior valor. Já a segunda lista contém strings, o que faz com que as funções trabalhem com comparações alfabéticas. Portanto, nesse exemplo o menor valor é Alex e o maior Renata.

Funções para coleções

sum()

Para trabalhar com coleções na linguagem Python, temos também a função sum(), que é usada para retornar a soma de todos os elementos da coleção.

```
1 | numeros = [1, 3, 6]
2 |
3 | print(sum(numeros)) # 10
```

Como vemos no código acima, sum() retornou a soma dos itens da lista numeros. Essa função não trabalha com strings, pois não é um tipo suportado por ela. Caso fossem usadas strings, a mensagem de erro TypeError: unsupported operand type(s) for +: 'int' and 'str' seria exibida.

Funções para coleções

len()

A função len() é bastante usada em Python para retornar o tamanho de um objeto. Quando usada com coleções, retorna o total de itens que a coleção possui.

```
1  paises = ['Argentina', 'Brasil', 'Colômbia', 'Uruguai']
2
3  print(len(paises)) # 4
```

Essa função é de grande utilidade, pois pode ser usada em diversas situações, como nas estruturas condicionais e em laços de repetição por exemplo.

Funções para coleções

type()

Com a função type() podemos obter o tipo do objeto passado no parâmetro.

```
1  professores = ['Carla', 'Daniel', 'Ingrid', 'Roberto']
2  estacoes = ('Primavera', 'Verão', 'Outono', 'Inverno')
3  cliente = {
4      'Nome': 'Fábio Garcia',
5      'email' : 'fabio_garcia_9@outlook.com'
6  }
7
8  print(type(professores)) # list
9  print(type(estacoes)) # tuple
10 print(type(cliente)) # dict
```

Exercícios

https://www.w3schools.com/python/exercise.asp

Exercícios

Dado o nome de uma pessoa, retorne o número de letras do nome e a primeira letra do nome.

Dada uma palavra, retorna a palavra invertida

Dada uma palavra, retorna os caracteres nas posições ímpares

https://www.w3resource.com/python-exercises/python-data-types.php

Senac
Fecomércio
Sesc

Obrigado!!

Senac Fecomércio Sesc

Referências

1. https://www.devmedia.com.br/colecoes-no-python-listas-tuplas-e-dicionarios/40678
2. https://www.w3schools.com/python/
3. https://www.w3resource.com/python-exercises/

Senac
Fecomércio
Sesc