Curso de Python Fatec e Huawei – Dia 4

Capitulo 2– Classes e Objetos

2.3 Objeto de Classes simples

- Objetos simples: compreendem os tipos de dados mais básicos e são chamados de simples pois são indivisíveis. São aqueles cujo valor representa uma única informação;

Classe	Descrição
`int`	Contém números inteiros tanto positivos, como negativos (e também zero)
`float`	Contém números reais positivos e negativos. O ponto deve ser usado como caractere de separação entre as partes inteira e decimal.
`complex`	Contém números complexos constituídos de parte real e parte imaginária. O valor da parte imaginária é acompanhado do caractere "`j`". Na matemática usa-se o caractere "`i`", mas a equipe de Python adotou o "`j`" seguindo o padrão usado na engenharia.
`bool`	Contém valores booleanos, ou lógicos. As duas possibilidades são: **`False`** para falso e **`True`** para verdadeiro, que são escritos com as iniciais maiúsculas.
`None`	`None` representa a ausência de valor. Não é algo usado com muita frequência em programação, mas há situações em que a ausência de valor ocorre. Em Python o `None` (com letra maiúscula) é usado nas situações em que há sua ocorrência. Um caso típico ocorre quando desejamos verificar se um objeto contém valor ou não. Esse objeto poderá ser comparado com `None` para realizar essa verificação.

- Objetos compostos: são constituídos de elementos que podem ser acessados em conjunto ou individualmente (em outras linguagens também são conhecidos como "tipos estruturados").

Classe	Descrição
str	Exemplo: `S = 'Isto é um Texto!!'` **ou** `S = "Isto é um Texto!!"` A classe `str` é usada para conter texto, ou seja, uma cadeia de caracteres. Em um programa para definir uma cadeia de caracteres pode-se usar aspas simples ou duplas. Os caracteres são armazenados utilizando-se a codificação Unicode (ao invés de ASCII). Strings são construídos usando aspas. O Python aceita que sejam usadas tanto aspas simples ('), quanto aspas duplas ("), desde que o mesmo tipo de aspas seja usado no início e no final do string. Para saber mais acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#text-sequence-type-str
list	Exemplo: `L = [12, 17, 9, 25, 41, 36, 23]` A classe lista é uma sequência. Listas são usadas para armazenar objetos de qualquer classe. É comum que todos os elementos da lista sejam da mesma classe e quando isso ocorre dizemos que a lista é homogênea. Porém isso não é obrigatório, ou seja, uma lista pode conter elementos classes diferentes e aí dizemos que a lista é heterogênea. Cada elemento da lista pode ser diretamente acessado e alterado através do uso de um índice, sendo que o primeiro elemento terá índice 0 (zero). Listas são construídas com o uso de colchetes [ ]. As listas serão vistas em detalhes no capítulo 7 deste material. Para mais detalhes acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#lists

tuple	Exemplo: `T = (23, 18, 99, 48, 56)` Uma tupla se assemelha a uma lista, porém com a diferença de que ela é apenas para leitura. Ou seja, depois de criada não pode ser alterada. Se houver tentativa de alterar um elemento o interpretador Python gerará uma mensagem de erro. Assim como as listas, as tuplas também podem ser homogêneas ou heterogêneas. Cada elemento da lista pode ser diretamente acessado através do uso de um índice, sendo que o primeiro elemento terá índice 0 (zero). Tuplas são construídas com o uso de parênteses ( ). As tuplas serão vistas em detalhes no capítulo 7 deste material. Para mais detalhes acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#tuples
range	Range é uma classe usada para criar sequências de números inteiros seguindo as regras de uma progressão aritmética. Para produzir o resultado essa classe deve receber 1, 2 ou 3 parâmetros. Exemplos: `class range(stop)` Se usada deste modo, `range` produzirá uma sequência iniciando em 0, avançando de 1 em 1 e terminando em `stop-1`
	`class range(start, stop[, step])` Se usada deste modo, range produzirá uma sequência iniciando em `start`, avançando com o valor `step` e terminando em `stop-1` A classe `range` é muito usada em conjunto com o comando for para criação de laços de repetição que deve executar um determinado número de vezes previamente conhecido. Range será vista em detalhes no capítulo 7 deste material. Para mais detalhes acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#ranges

dict	Exemplo: `D = {'maçã': 9.75, 'laranja': 6.55, 'banana': 5.43}` A classe `dict` é conhecida como dicionário ou mapeamento. Trata-se de uma coleção onde cada elemento é definido como um "par chave-valor". A chave identifica o elemento e o valor é o conteúdo. No dicionário a chave é análoga ao índice de uma lista e deve ser usada para acessar e alterar seu valor. Dicionários são construídos com o uso de chaves { }. Os dicionários serão vistos em detalhes no capítulo 8 deste material. Para mais detalhes acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#mapping-types-dict
set	Exemplo: `C = {23, 35, 14, 28, 9, 37}` A classe `set` é usada para a criação de conjuntos de elementos obrigatoriamente distintos. A classe set permite inclusão de novos elementos e a exclusão de elementos existentes. Porém, não é possível a alteração de um elemento existente. A classe `set` suporta as operações definidas na Teoria dos Conjuntos da matemática. A ordem dos elementos dentro do conjunto é estabelecida pelo interpretador Python e o programador não tem controle sobre ela. Um conjunto não pode ser ordenado. É possível livremente converter listas e tuplas em conjuntos e vice-versa. Se uma lista contiver elementos repetidos, ao ser convertida em conjunto os repetidos serão eliminados. Conjuntos são construídos com o uso de chaves { }. (Opa, Cuidado!) Dicionários também são construídos usando chaves { }, por isso, é preciso que o programador tenha cuidado para não confundir os dois. Observando a natureza de seus elementos pode-se diferenciar conjuntos de dicionários. Para mais detalhes acesse: https://docs.python.org/pt-br/3/library/stdtypes.html#set-types-set-frozenset
frozenset	A classe `frozenset` é semelhante à classe set em tudo, com a exceção de que `frozenset` não permite inclusão ou exclusão de elementos. O link da documentação de `frozenset` é o mesmo do set (na linha acima deste quadro).
bytes **bytearray** **memoryview**	Estas três classes, `bytes`, `bytearray` e `memoryview`, existem para a realização de tarefas relacionadas à manipulação de dados binários, tais como reproduzir um som mp3, manipular as cores dos pixels de uma imagem, tratar os bytes que trafegam em uma rede, entre muitas outras possibilidades de exemplos. Estas classes não são vistas neste material, pois trata-se de conteúdo avançado e específico. Deixamos aqui o link da documentação para quem desejar conhecer essas classes https://docs.python.org/pt-br/3/library/stdtypes.html#binary-sequence-types-bytes-bytearray-memoryview

2.4 Comandos de atribuição

O comando de atribuição é um elemento de programação muito simples e necessário existente emtodas as linguagens. Na maioria delas é representado pelo caractere de igual: "=". Na linha a seguir o identificador "a" está recebendo o valor numérico inteiro 10.

```
a = 10
```

A forma geral do comando de atribuição é esta:

```
identificador = expressão
```

Do lado esquerdo deve-se colocar o identificador que é um nome para o objeto.

A expressão do lado direito pode ser uma das seguintes possibilidades:

- um literal (um valor fixo numérico ou texto); ex: a = 10 ou m = 'texto'
- um objeto; ex: b = a
- uma fórmula matemática; c = a + 10
- uma função; d = len('exemplo')
- ou ainda uma expressão com uma combinação dos itens acima.