Objetos

Conceito

O conceito de objetos deve ser entendido como o formato disponibilizado na linguagem para **estruturar informações**. Ou seja, para estruturar o número do **pi** em um objeto (notação) python é necessário usar a estrutura (objeto) **float**. Para estruturar a **coleção** de probabilidades de ocorrências dos dígitos de 1 a 9 é possível usar a estrutura **dictionary**. Para estruturar um **nome**, recomenda-se o uso de uma **string**. O contexto da aplicação e a experiência do programador são fundamentais na **escolha** da estrutura python **adequada** para a representação de informações desestruturadas (pythonicamente).

Principais tipos de objetos existentes nativamente

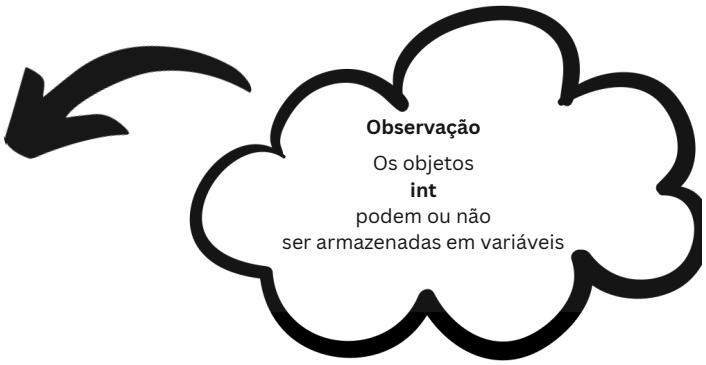
- int representação de números **inteiros** negativos e positivos
- float representação de números **reais** negativos e positivos
- string representação de uma **coleção de caracteres** (não-mutável)
- list representação de uma **coleção de elementos** (mutável)
- tuple representação de uma **coleção de elementos** (não-mutável)
- dict representação de uma **coleção de elementos pareados** (mutável)
- bool representação de uma informação **binária**

Sintaxe Python (int)

EXAMPLE 2

EXAMPLE 10

EXAMPLE 321



Objetos

Sintaxe Python (float)

Observe o **ponto**

EXAMPLE 1.6180

EXAMPLE 2.7172

EXAMPLE 3.1415

Observação
Os objetos **float** podem ou não ser armazenadas em variáveis

Sintaxe Python (string)

Observe as **aspas**

EXAMPLE “qualquer texto”

EXAMPLE “prandiano”

EXAMPLE “3.1415” → string

Observação
Os objetos **string** podem ou não ser armazenadas em variáveis

As strings são coleções de caracteres. Sendo uma coleção, esse objeto possui a **sintaxe de fatiamento** (slicing) como segue:

	0	1	2	3	4	5	6	7	8	
nome_escola = “	p	r	a	n	d	i	a	n	o”	**indices positivos**
	-9	-8	-7	-6	-5	-4	-3	-2	-1	**indices negativos**

Fatiamento 1 / 2

leitura humana	sintaxe python	retorna
retorna o **primeiro** elemento da coleção	nome_escola[**0**]	“p”
retorna o **quarto** elemento da coleção	nome_escola[**3**]	“n”
retorna os elementos de indice **3 até** o índice **5**	nome_escola[**3:6**]	“ndi”
retorna o **último** elemento da coleção	nome_escola[**-1**]	“o”
retorna os elementos de indice **-4 até** o índice **-2**	nome_escola[**-4:-1**]	“ian”

Slicing 1 / 2

Curso de Python 1

Objetos

Fatiamento 2 / 2

Slicing 2 / 2

índice inicial (inclusive)

índice final (exclusive)

passo e direção

leitura humana	sintaxe python	retorna
retorna os elementos de indice **0 até** o índice **2** **um elemento por vez**	nome_escola[**0:3:1**]	“pra”
retorna os elementos de indice **1 até** o índice **7** **dois elementos por vez**	nome_escola[**1:8:2**]	“rnin”
retorna os elementos de indice **-8 até** o índice **-1** **dois elementos por vez**	nome_escola[**-8:-1:2**]	“rnin”
retorna os elementos de indice **-1 até** o índice **-8** **dois elementos por vez na ordem inversa**	nome_escola[**-1:-8:-2**]	“ninr”
retorna os elementos do comeco ao fim **um elemento por vez na ordem inversa**	nome_escola[**::-1**]	“onaidnarp”

Observações

1. Fatiamento de **string retorna uma string**!!!
2. Todos os parâmetros do fatiamento (indice inicial, indice final e passo) são **opcionais**, ou seja, se não forem fornecidos, possuem um valor **padrão**. Se o parâmetro inicial não for fornecido, o fatiamento ocorre **desde o primeiro elemento**. Se o parametro indice final não for fornecido, o fatiamento ocorre **até o último elemento** (incluído). E, finalmente, se o passo não for fornecido, o fatiamento ocorre **um elemento por vez na direção normal**.

Sintaxe Python (list)

Observe os **colchetes**

EXAMPLE [“qualquer objeto”] — **um** elemento dentro da **lista**

EXAMPLE [1, 2, 3, 4, 5, 6] — **seis** elementos dentro da **lista**

EXAMPLE [1, 1.61, 2, 3] — **quatro** elementos dentro da **lista**

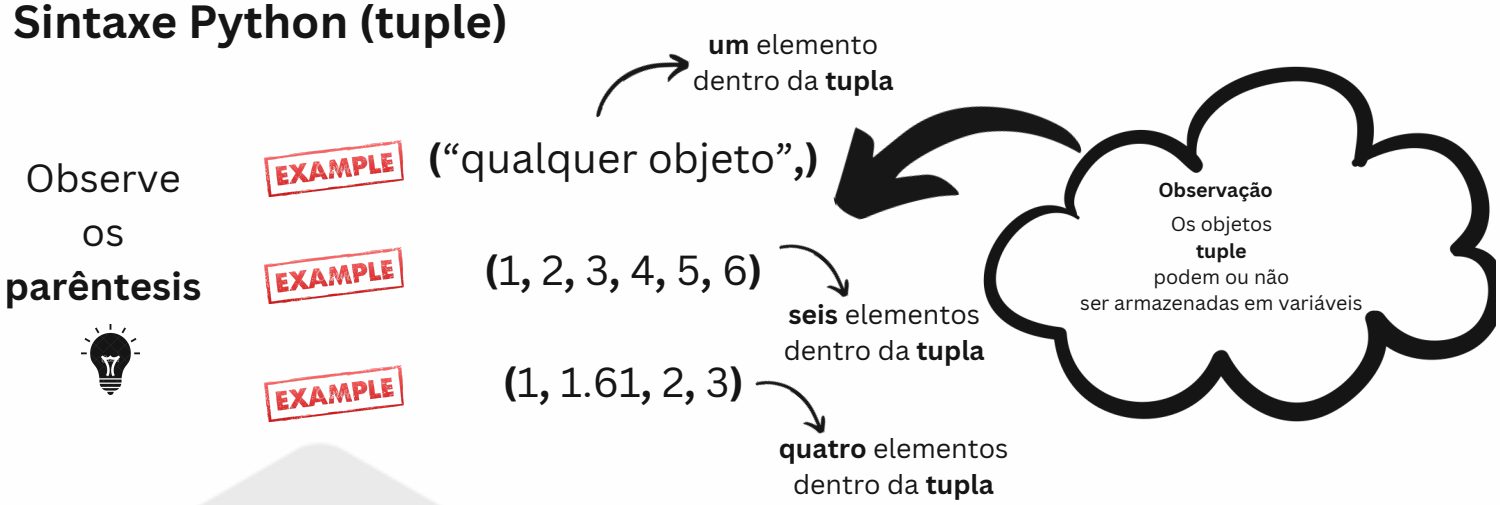
Observação
Os objetos **list** podem ou não ser armazenadas em variáveis

Resumo

As listas são coleções de qualquer objeto. Sendo uma coleção, esse objeto possui a **sintaxe de fatiamento** (slicing). Todas as regras de fatiamento das strings servem igualmente às listas. Os elementos devem ser separados por vírgula. Porém, o fatiamento de uma lista retorna uma lista!!!

Objetos

Sintaxe Python (tuple)



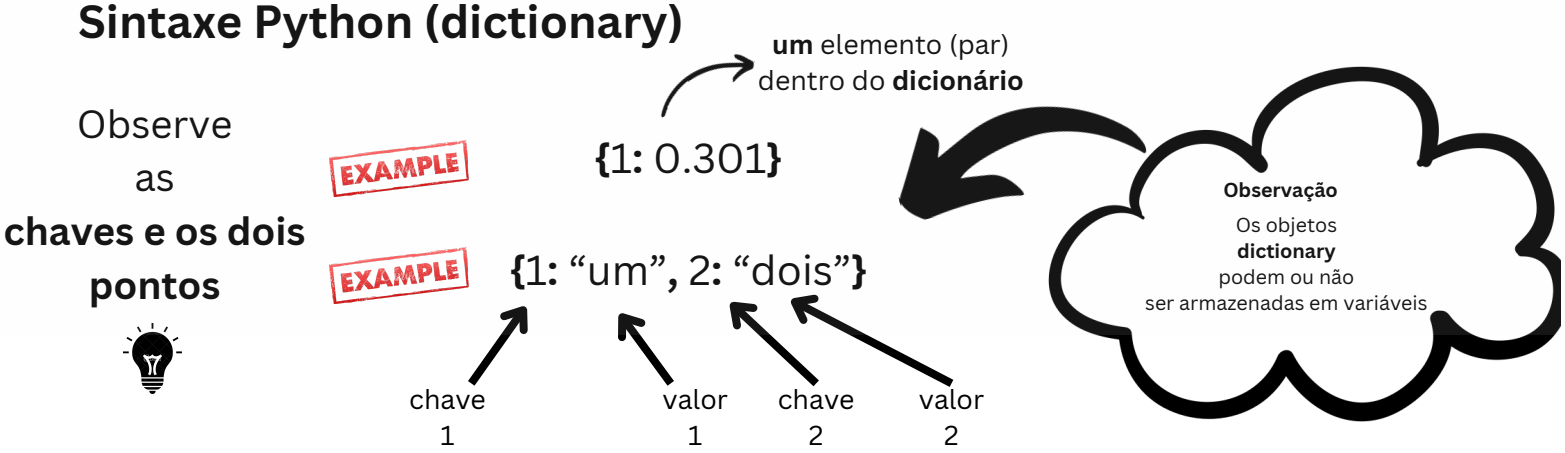
Resumo

As tuplas são coleções de qualquer objeto. Sendo uma coleção, esse objeto possui a **sintaxe de fatiamento** (slicing). Todas as regras de fatiamento das strings servem igualmente às tuplas. Os elementos devem ser separados por vírgula. Porém, o fatiamento de uma tupla retorna uma tupla!!!

Observações

1. O principal fator que diferencia as listas das tuplas, é que a lista é um objeto **mutável**, ou seja, sempre é possível **acrescentar, remover, modificar** elementos da lista. Porém, as tuplas **nunca** permitem essas operações; as tuplas são **imutáveis**;
2. Para se criar uma tupla com apenas um elemento (quando necessário for), é necessário incluir uma vírgula no final (observe o primeiro exemplo acima). A vírgula incluída explicita que se trada de uma coleção. Como a coleção é limitada por parêntesis, a coleção será do tipo tupla.

Sintaxe Python (dictionary)



Curso de Python 1

Objetos

Resumo

Os dicionários são coleções de pares chave:valor separados por dois pontos (:). Sendo uma coleção, esse objeto possui a **sintaxe de fatiamento** (slicing). As regras de fatiamento dos dicionários são específicas para este objeto. Os pares chave:valor devem ser separados por vírgula.

dicionario = **{**1**:** “int”**,** 1.0**:** “float”**,** “1”: “string”**}**

Fatiamento | **Slicing**

leitura humana	sintaxe python	retorna
retorna o **valor** associado com a chave **1**	nome_escola[**1**]	“int”
retorna o **valor** associado com a chave **1.0**	nome_escola[**1.0**]	“float”
retorna o **valor** associado com a chave **“1”**	nome_escola[**”1”**]	“string”

Observações

1. Os dicionários são fatiados pelas suas chaves, que por sua vez, estão associadas com seus valores;
2. Não existe índices (posições) nos dicionários;
3. Os dicionários são objetos não ordenados;
4. As chaves dos dicionários devem ser bem escolhidas para que a estrutura dicionário sirva adequadamente uma aplicação. Por exemplo, no caso dos pares de benford digito:probabilidade, os dígitos devem estar nas posições das chaves do dicionário, pois, fornecendo o dígito (chave) para o fatiamento, o retorno obtido é a probabilidade (valor). Na estrutura proposta acima, não é possível obter uma chave tendo sido fornecido um valor;
5. Os dicionários são unidirecionais, ou seja, dada uma chave, é possível obter o valor associado. Nunca o contrário;
6. Essa estrutura de mapeamento chave:valor permite uma execução computacional de procura muito mais rápida do que nas demais coleções. Para objetos com muitos elementos (pares) essa performance melhorada dos dicionários é mais percebida.

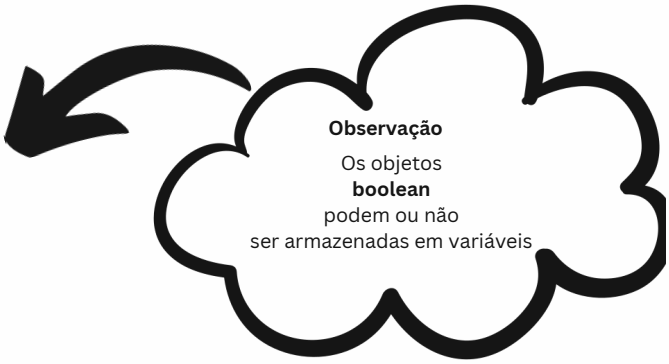
Objetos

Sintaxe Python (boolean)

Observe as **letras ‘T’ e ‘F’ Maiúsculas**

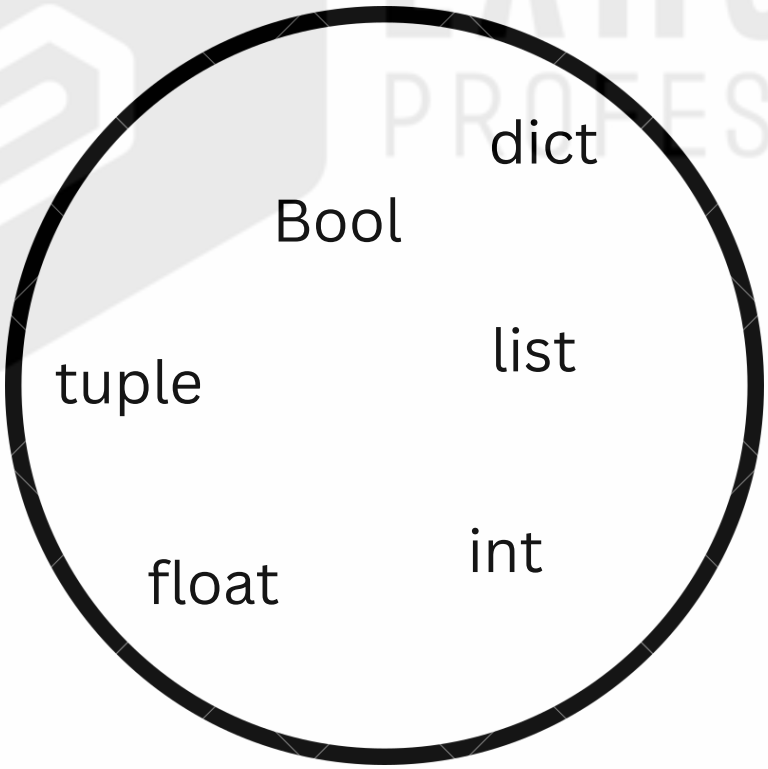
EXAMPLE **False**

EXAMPLE **True**



Resumo

Os booleanos não são coleções (assim como os objetos int ou float). Esses objetos permitem operações de álgebra de boole (and, or, not, xor). As informções que podem ser representadas por booleanos deve ser do tipo binárias (por exemplo, ligado-desligado, aceso-apagado, horizontal-vertical, etc.). Os booleanos podem se comportar como int: False = 0 | True = 1.



Objetos Nativos do tipo Built-in

Curso de Python 1