Rumos

Reading Python Programming

Loops e Listas

6 de julho de 2024

Agenda da Sessão

01. Estruturas de Dados

02. Loops

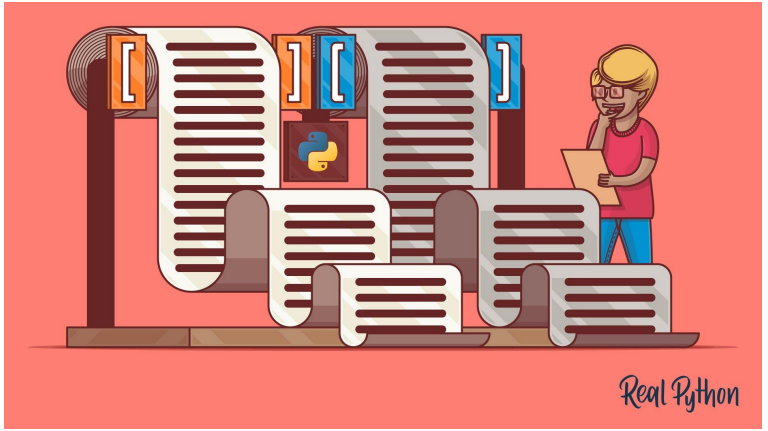
01 Estruturas de Dados

Estruturas de Dados

Contextualização

Geral

- Conjuntos de dados associados a apenas uma variável
- Permitem maior legibilidade ao código, nomeadamente quando utilizados com loops
- Utilização de métodos/funções disponíveis já pré-definidos
- Maior escalabilidade do código



Python

- Implementados como Listas, Tuplos, Dicionários e Conjuntos

Listas *Lists*

Implementação

Vantagens

- Flexibilidade e capacidade de alterar elementos conforme necessidade
- Métodos pré-definidos para ordenar - sort(), inverter - reverse(), entre outros

Sintaxe

- Declarar variável e usar parênteses retos para criar/atribuir listas, separando cada elemento por vírgula
- Pode-se atribuir/criar uma lista vazia e depois utilizar os métodos para adicionar elementos

```
shopping_list = [“apple”, “banana”]

another_shopping_list = []

another_shopping_list.append(“water”)
# = [“water”]
```

Listas *Lists*

Manipulação de Elementos

Adição

- append() Adiciona um elemento ao final da lista
- insert() Insere um elemento numa posição específica
- extend() Adiciona elementos de outra lista no final da lista atual

Remoção

- remove() Apaga a primeira ocorrência de um elemento específico
- pop() Remove e retorna o elemento numa posição específica
- del Apaga um elemento pelo índice

Acesso por índice

- Atribuir um novo valor a um elemento da lista
- Retornar valor usando sintaxe list[index] - *primeiro valor do index é sempre o 0*

```
fruits = [“manga”,”orange”,”banana”]

fruits.append(“grape”)
fruits.insert(1, “strawberry”)
fruits.extend([“lemon”, “apple”])

fruits.remove(“banana”)
element = fruits.pop(2)
del fruits[0]

fruits[1] = “pear”
fruit = fruits[1]
```

Tuplos *Tuples*

Implementação

Vantagens

- Conjunto de valores imutável, não permitindo qualquer alteração após criação
- Permite maior segurança de dados dada a imutabilidade
- São de mais rápido acesso e ocupam menos espaço em memória

```
coordinates = (41.123, -23.456)
```

Sintaxe

- Declarar variável e usar parênteses para criar/atribuir listas, separando cada elemento por vírgula

Tuplos *Tuples*

Manipulação de Elementos

Acesso a Elementos

- Identificando a posição requerida no nome da variável
- Desempacotar o tuplo fazendo uma “lista” de variáveis

Métodos Acessórios

- count() Saber o número de ocorrência de um valor
- index() Saber o index de determinado valor

```
pin = (2, 3, 4, 3)

pin_first_number = pin[0]

a, b, c, d = pin

three_repetition= pin.count(3)

four_index = pin.index(4)
```

Dicionários *Dictionaries*

Implementação

Vantagens

- Os dicionários são estruturas chave-valor (como um JSON)
- A grande vantagem dos dicionários é a capacidade de aceder aos valores através de chaves
- Organização eficiente de dados associativos

Implementação

- Declarar variável e usar chavetas para criar/atribuir dicionários, sendo que cada par {chave}:{valor} é separado por vírgulas.

```
country_capitals = {
  "United States": "Washington D.C.”,
  "Italy": "Rome",
  "England": "London"
}
```

Dicionários *Dictionaries*

Manipulação de Elementos

Adição

- Usar a sintaxe dictionary_name[key] = value (este método também pode ser usado para alterar pares já existentes)

Acesso a valor

- get(key) ou dictionary_name[key] Retorna valor associado a chave
- keys() Lista todas as chaves do dicionário
- values() Retorna todos os valores do dicionário
- items() Retorna os pares chaves/valor como tuplo

Remoção

- del Apaga um elemento pelo índice

```
person = {"name": "Alice", "age": 25}

person["city"] = "Porto"
name = person["name"]
name = person.get("name")

keys = person.keys()
values = person.values()
pairs = person.items()

del person["city"]
```

Conjuntos *Sets*

Implementação

Vantagens

- Flexibilidade e capacidade de alterar elementos conforme necessidade
- Não mantém ordem e todos os elementos são únicos
- Inúmeros métodos pré-definidos para comparação/interação com outros conjuntos

Sintaxe

- Declarar variável e usar parênteses curvos para criar conjuntos, separando cada elemento por vírgula
- Para fazer um set fazia tem que se usar o método set()

```
numbers_available = {3, 5}

empty_set = set()

empty_set.add(2)
```

Conjuntos *Sets*

Manipulação de Elementos

Adição

- add() Adiciona um elemento ao conjunto

Remoção

- remove() Apaga elemento, dando erro se não encontrado
- discard() Apaga elemento, sem erro se não encontrado
- pop() Remove e retorna o elemento numa posição específica

Interação com outros conjuntos

- union() Retornar a união de dois sets
- intersection() Retorna a interseção de dois sets

```
seats_one = {“A1”,”A2”,”A3”}

seats_one.add(“B1”)
seats_one.remove(“A3”)
seats_one.discard(“Z2”)

seats_two = {“B1”,”B2”}

seats_union =
seats_one.union(seats_two)
seats_intersection =
seats_one.intersection(seats_two)
```

Textual *Strings*

A exceção

Apesar de ser considerada um dado simples/básico, a string/texto é uma sequência de caracteres e por isso possui um conjunto alargado de manipulações. É também imutável o que significa que qualquer alteração obriga a guardar novo valor ou utilizá-lo imediatamente.

Por índice

- Retornar caracter usando sintaxe ***string[index]***
- Retornar pedaço de texto ***string[start:end]***

Manipulação

- replace() Cria nova string com a alteração requerida
- len() Retorna o tamanho da string
- upper()/lower()/capitalize() Retorna string na formatação esperada

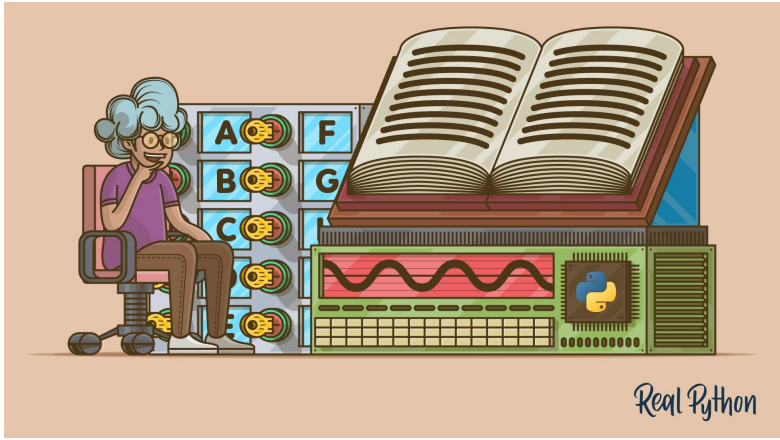
```
word = "Olá"

print(word[0]) # O
print(word[1:]) # lá

word = word.replace("O", "I") # Ilá

len(word) # 3

word.upper() # ILÁ
```

Listas, Tuplos, Dicionários e Conjuntos

Playground

Objetivo

Utilizar o ficheiro structure_data_playground.py para experimentar/criar exemplos de cada uma das estruturas de dados.



02 Loops

Loops

Contextualização

Geral

- Estruturas fundamentais na programação que nos permitem automatizar tarefas repetitivas de maneira eficaz
- Executa repetidamente código até que uma condição seja atendida ou até que todos os elementos de uma sequência sejam processados

Python

- for quando sabemos/possuímos uma quantidade exata de iterações a serem realizadas
- while quando precisamos repetir um bloco de código até que uma condição seja satisfeita

For

iterar listas/conjuntos

Sintaxe

- for {variável} in {lista}
- Em cada iteração é possível executar código com o nome da variável - que vai alterando de valor à medida que se percorre a lista

```
numbers = [1, 2, 3, 4, 5]

for number in numbers :
    print(number)
```

For

iterar dicionários

Sintaxe

- for {chave} in {dicionário}
- Em cada iteração é possível usar a chave para aceder ao valor no dicionário

```
person = {"name": "Paulo",
"age": 30, "city": "Porto"}

for key in person :
    value = person[key]
    print(key + ": " + value)
```

For

iterar com range()

Sintaxe

- for {chave} in range(início, fim, passo)

  início (opcional - default 0)O número a partir do qual a sequência inicia (inclusivo)

  fim O número no qual a sequência termina (exclusivo)

  passo (opcional - default 1): O intervalo entre os números na sequência

- Permite definir valores numéricos como limites da interação

```
total = 0

for number in range(1, 11):
    total += number

print("Sum for 1 to 10 is: ", total)
```

While

Sintaxe

- while (condition):
- O loop é mantido enquanto a condição seja verdadeira, podendo ser um estado (potencialmente infinito) ou funcionar como contador

```
lucky_number = 7
inputed = int(input("Lucky Number: "))

while inputed != lucky_number:
    inputed = int(input("Lucky Number: "))

print("You got lucky!")
```

```
counter = 0

while counter < 5:
    print("Count :", counter)
    counter += 1

print("Loop finished!")
```

Keywords

Loops

- return Para parar alguma iteração, retornando algum valor
- break Para parar alguma iteração, usualmente após uma condição cumprida
- continue Para prosseguir para a próxima iteração, usualmente após uma condição cumprida
- pass Utilizada quando ainda não existe qualquer implementação de código num loop ou até num método, e não provocar qualquer erro a correr o código

```
for number in range(1, 11):
    if number == 2 :
        continue
    print(number)


for number in range(1, 11):
    if number == 3 :
        break
    print(number)


for number in range(1, 11):
    pass
```

El Toro

É de estrela Michelin

Objetivo

Fazer a cozinha do El Toro preparar e servir um pedido de um cliente.

Como?

Preencher a zona para loops no ficheiro el_toro_restaurant.py

Rumos

Formação e Certificação Técnica que potenciam **Profissionais e Organizações**

Lisboa

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 217 824 100
Email info@training.rumos.pt

Porto

Edifício Mirage – Entrecampos
Rua Dr. Eduardo Neves, 3
1050-077 Lisboa

ver google maps ↗

Tel +351 222 006 551
Email info@training.rumos.pt

Siga-nos

Rumos