Bootcamp 134 | Python

Course 05 | Basic Python

Amir Hossein Chegouniyan

Head of the Technical Team at Dariche Tejarat

Lecturer of Python – Django at Maktab Sharif

Linkedin: Amirhossein-chegounian

Content

None Type

- The None keyword is used to define a null value, or no value at all.
- None is not the same as 0, False, or an empty string. None is a data type of its own (NoneType) and only None can be None.

```
x = None
if x:
        print("Do you think None is True?")
elif x is False:
        print ("Do you think None is False?")
elif not x:
        print("None is not True, or False, None is just None...")
```

Python Collections

- List: Is ordered, changeable, allow duplicate member
- Tuple: Is ordered, unchangeable, allow duplicate member
- Set: Is unordered, unchangeable, no duplicate member
- Dictionary: Is unordered, changeable, no duplicate member

Dictionary | Intro

- Dictionaries are used to store data values in key-values pairs.
- Are ordered, changeable, and do not allow duplicates
- Type: <class ‘dict’>

Dictionary | Create

<name_of_dict> = {'key_01': 'value_01', 'key_02': 'value_02'}

<name_of_dict> = dict({key_01='value_01', key_02='value_02'})

- *For example:*

  ali_info = {"name": "Ali", "grade": "A"}

  ali_info= dict(name= "Ali", grade= "A")

Dictionary | Access to keys

<name_of_dict>[key]

<name_of_dict>.get(key)

- *For example:*

  ali_info[“name”]

  ali_info.get(“name”)

Dictionary | In Loops

```
for i in <name_of_dict>:              # for waliking on ??????
for i in <name_of_dict>.keys():       # for waliking on KEYS
for i in <name_of_dict>.values():     # for waliking on VALUES
for i, j in <name_of_dict>.items():   # for waliking on KEYS and VALUES
```

- For example:

```
for i, j in ali_info.items():
    print(f"{i}: {j}")
```

Dictionary | In Conditions

if key in <name_of_dict>:

- *For example:*

if “name” in ali_info:

print(ali_info[“name”])

Dictionary | Update Dict Items

<name_of_dict>[“key”} = <new_value>

<name_of_dict>.update({“key”: <new_value>})

- *For example:*

  ali_info[“name”] = “Ali Alavi”

Dictionary | Add Dict Item

<name_of_dict>[“new_key”} = <new_value>

<name_of_dict>.update({“new_key”: <new_value>})

- *For example:*

  ali_info[“phone”] = “02112345678”

Dictionary | Delete Dict Item

```
<name_of_dict>.pop("key")        # delete item with same key
<name_of_dict>.popitem()         # delete last item
del <name_of_dict>["key"]        # delete item with same key
del <name_of_dict>               # delete dict
<name_of_dict>.clear()           # empties dict
```

- *For example:*

```
del ali_info["phone"] = "02112345678"
```

Dictionary | Copy a Dict

- Cannot copy a dict with *dict_2 = dict_1*, because *dict_2* will only be a refrence to *dict_1*

  <new_dict_name> = <name_of_dict>.copy()

  <new_dict_name> = dict(<name_of_dict>)

- *For example:*

  copy_of_ali_info = ali_info.copy()

Tuple | Intro

- Is ordered, unchangeable (**immutable**), allow duplicate members
- Access: is like to list
- Change: Convert it to list, edit or add.
- Loop: is like to list

Tuple | Create and Unpacking

color = (“Green”, “Red”, “Blue”)

(gr, re, bl) = color

Tuple | Work With It

- Join:
  - *tuple_1 + tuple_2*
  - *tuple_1 * 2*
- t*uple_1.index("value")* *# find index of first data that is equal to input*
- *Tuple_1.count("value")* *# retruns member o times a value appers in the tuple*

List | Intro

Comprehension

- *For list:*
  - *[output for x in <iterator > statement]*
- *For dictionary:*
  - *{key:value for x in <iterator> statement*
- *For tuple:*
  - *(output for x in <iterator> statement)*
- *For set:*
  - *{output for x in <iterator> statement}*
- *For Example:*
  - *new_list = [1, 2, 3]*
  - *new_dict = {str(i):i*i for i in new_list if i % 2 == 0}*

String Formmatting

- .fromat function

```
txt = "For only {price:.2f} dollars!"
print(txt.format(price = 49))
```

- *F-string*

```
price = 49
print(f“For only {price:.2f} dollars!”)
print(f“For only {price:.05d} dollars!”)
```

Copy

- *In Python, copy and deepcopy refer to different methods of creating copies of objects, particularly relevant for mutable objects like lists or dictionaries. Both are found in the copy module.*

Copy | Shallow Copy

```
import copy

original_list = [[1, 2], 3]
shallow_copy_list = copy.copy(original_list)
shallow_copy_list[0].append(4)  # Modifying the nested list
print(original_list)            # Output: [[1, 2, 4], 3]
print(shallow_copy_list)        # Output: [[1, 2, 4], 3]
shallow_copy_list[1] = 5        # Modifying an immutable element
print(original_list)            # Output: [[1, 2, 4], 3]
print(shallow_copy_list)        # Output: [[1, 2, 4], 5]
```

Copy | Deep Copy

```
import copy

original_list = [[1, 2], 3]
deep_copy_list = copy.deepcopy(original_list)
deep_copy_list[0].append(4)      # Modifying the nested list
print(original_list)             # Output: [[1, 2], 3]
print(deep_copy_list)            # Output: [[1, 2, 4], 3]
deep_copy_list[1] = 5            # Modifying an immutable element
print(original_list)             # Output: [[1, 2], 3]
print(deep_copy_list)            # Output: [[1, 2, 4], 5]
```

Any question?

Next course

- Introduction to Functional Programming
- Lambda Functions
- map, filter, and reduce
- Generators
- Decorators
- Recursion
- Sets
- Itertools Module