Data Science

Introduction to Python

CS202 09.07.22

Introduction to Python

Python Data Types

- Text type : str
- Numeric types: int, float, complex
- Sequence types: list, tuple, range
- Mapping type: dict
- Set types: set, frozen set
- Boolean type: bool
- Binary types: bytes, bytearray, memoryview
- None type: NoneType

Mapping type

Dict

Dictionaries are used to store data values in key:value pairs.

Dictionaries are written with curly brackets, and have keys and values:

```
animal_dict = {
  "animal": "Dog",
  "breed": "Shiba",
  "age": 7
}
print(animal_dict)
```

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
print(animal_dict[“breed”]) #this should print “Shiba”
```

Mapping Type

Dict

You cannot have duplicates in a dictionary

```
animal_dict = {
    "animal": "Dog",
    "breed": "Shiba",
    "breed": “Husky",  #this will cause error
    "age": 7
  }
  print(animal_dict)
```

You can use the len( ) method to see the length of your dictionary.

The values of your dictionary can be of any data type:

```
animal_dict = {
    "animal": "Dog",
    "barks": True,
    "age": 7.
    “colors": ["red", "white", "sesame"]
  }
  print(animal_dict)
```

Dictionary

You can retrieve values in the following way:

```
animal_dict[“animal”]
```

or you can use the get( ) method to get the same result:

```
animal_dict.get(“animal”)
```

You can retrieve keys of a dictionary by using the keys( ) method:

```
animal_dict.keys()
```

This will return you a list of keys. If you add a new key to the dictionary this list of keys will update.

```
animal_dict[“breeds”] = “Shiba”
```

Dictionary

Similar to the keys( ) method, the values( ) method will return you a list of values.

```
animal_dict[“age”] = 8
```

This will update the age of the dictionary from 7 to 8, hence when you use the values( ) method you will get back an updated list of values.

The items( ) method will give you back each item in your dictionary as tuples in a list.

You can check if a key exists by using the ‘in’ keyword:

```
if "breed" in animal_dict:
  print("Yes, 'breed' is one of the keys")
```

Dictionary

Changing items

Suppose we have the following dictionary:

```
animal_dict = {
  "animal": "Dog",
  "breed": "Shiba",
  "age": 7
}
print(animal_dict)
```

and we want to change the “breed” to “husky”, i.e. we want to change a value of a particular key then we write the following:

```
animal_dict["breed"] = "Husky"
```

Alternatively, we can use the update( ) method:

```
animal_dict.update({"breed": "Husky"})
```

Dictionaries

- You can remove items using the pop( ) method, you just need to specify the key name.
- Unlike a list you would need to use popitem( ) to remove the last item on your dictionary
- The ‘del’ keyword can delete specific items on the list:
    - `del animal_dict[“age"]`
- You can use the ‘del’ keyword to delete the entire dictionary:
    - `del animal_dict`
- The clear( ) method empties the dictionary. Hence writing animal_dict.clear( ) and then printing animal_dict will just return you an empty dictionary { }
- Use the copy( ) or dict( ) method to copy a dictionary:
    - `my_dict = animal_dict.copy()`
    - `my_dict = dict(animal_dict)`

Dictionary

Nested dictionaries

We can have nested lists, like lists in lists, this is also true of dictionaries. Often times when you use API calls you will get a JSON object which is really just nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Alice",
    "year" : 2004
  },
  "child2" : {
    "name" : "Rachel",
    "year" : 2007
  },
  "child3" : {
    "name" : “Colette",
    "year" : 2011
  }
}
```

Dictionary

Nested dictionaries

You can make a nested dictionary from separate dictionaries:

```
child1 = {
  "name" : "Alice",
  "year" : 2004
}
child2 = {
  "name" : "Rachel",
  "year" : 2007
}
child3 = {
  "name" : “Colette”,
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Python

IF and ELSE statements

Now that we’ve covered most of the data types and how to use them we’re gonna start discussing all the statements we can write.

We begin by looking at ‘if and else’ statements.

The logic is as follows: if <insert condition> <insert return statement>

Else <insert condition><insert return statement>.

For example lets consider the following statement: say you are 5.5ft and the sign says “If you are taller than 5ft you can ride this rollercoaster else you cannot ride this rollercoaster.”

How would we write this in python?

```
a = 5.5
if (a > 5):
    print(“you can ride this rollercoaster!”)
else:
   print(“you cannot ride this rollercoaster!”)
```

If and Else

If you want to add more conditions we can use “elif” this is like saying “if your previous condition was not met then try this condition”

For example: Consider you are exactly 5ft tall and you want to ride this rollercoaster and the guy working at the theme park is willing to let you ride if you are at least 5 ft tall. How would this change your python code?

```
a = 5.0
if a > 5:
    print(“you can ride this rollercoaster!”)
elif a == 5:
    print(“you can ride this rollercoaster!”)
else:
   print(“you cannot ride this rollercoaster!”)
```

If and Else

We can use if and else with other statements: like examples we've seen before working through a dictionary or a list.

For example my_list = ["dog", "cat", "kitten", "raccoons"]

We can write something like:

```
if "kitten" in my_dict:
  my_dict.append(“mitten”)
```

Theres no logic or reasoning to why I would want that, but its something you can do which you will find helpful when writing functions.