**Python Bootcamp**

**dir(object)**

* The dir() function returns all properties and methods of the specified object, without the values.
* This function will return all the properties and methods, even built-in properties which are default for all object.
* Applicable to all type of data structures, objects …etc

txt = "banana"

print(dir(txt))

# Output:

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

**COPY OBJECT**

|  |
| --- |
| **LIST** |
| **Common mistake:**  In this case, when = operator is used to copy a list, list\_2 is referencing to list\_1 object. Therefore, if list\_2 is modified, list\_1 will also be affected.  list\_1 = ['a','b','c','d']  list\_2 = list\_1  list­\_2[1] = 'g'  print(list\_1)  print(list\_2)  # Output:  #     ['a', 'g', 'c', 'd']  #     ['a', 'g', 'c', 'd']  **Recommended method:**  We can use copy() method or list slicing to make a new copy of an object. A new object and old object will be totally separate even though their content are the same. When the new object is modified, the old object is not affected.  **Examples:**  **List slicing**  list\_1 = ['a','b','c','d']  list\_2 = list\_1[:]  list\_2[1] = 'g'  print(list\_1)  print(list\_2)  # Output:  #     ['a', 'b', 'c', 'd']  #     ['a', 'g', 'c', 'd']  **copy() method**  copy() method returns a shallow copy of the object. Dictionary copy() is a bit more complicated than that. We would not dive in to the details here.  list\_1 = ['a','b','c','d']  list\_2 = list\_1.copy()  list\_2[1] = 'g'  print(list\_1)  print(list\_2)  # Output:  #     ['a', 'b', 'c', 'd']  #     ['a', 'g', 'c', 'd'] |
| **DICTIONARY** |
| **= operator:**    **Shallow copy method (Only works for immutable values)**  Shallow copy only works for dictionary with immutable values (i.e string, integer…) . If the value is a mutable object (List), shallow copy does not work.    Here is an example of Shallow copy on a dictionary with mutable values.    However, here is what actually happens inside the computer memory    Despite shallow copy of the original object, the **mutable** values of the original dictionary are stored as **separate variables** outside of the dictionary. When a new dictionary is created via copy() method, its values (lists) are also referring to the **variables** mentioned earlier. Therefore, any change made to the new dictionary values will also affect the first dictionary.  **Deep copy method (Works for mutable values)**      This is the logic behind deep copy |

**Type of data structure:**

|  |
| --- |
| **STRING**  **Definition:** A string is a series of characters / character, put inside a quotation mark. A string can be put inside double quotes “ “ or single quotes ‘ ‘.  **Example:** “Hello”   * String is immutable * String characters can be indexed and iterated. String index starts at 0 * String can be separated into a list and joined back into string format using a different variable   **STRING INDEXING:**  **Regular indexing:**  a\_string = 'Terry'  character = a\_string[0]  print(character)  # Output: T  character1 = a\_string[1]  print(character1)  # Output: e  **Reversed indexing:**  a\_string = 'Terry'  character = a\_string[-1]  print(character)  # Output: y  character1 = a\_string[-3]  print(character1)  # Output: r  **STRING METHODS:**  **string.upper()** Capitalise all character in a string  a\_string = 'terry'  print(a\_string.upper())  # Output: TERRY  **string.lower()** De-capitalise all character in a string  a\_string = 'tErRY'  print(a\_string.lower())  # Output: terry  **string.capitalize()** Capitalise the first character of a string  a\_string = 'tErRY'  print(a\_string.capitalize())  # Output: Terry  **string.find(value, start, end)** Return the first occurrence of the specified character inside a string.  a\_string = 'Terry Nguyen'  index = a\_string.find('g')  print(index)  # Output: 7  **Note:**   * The find() method returns -1 if the value is not found. * The find() method is almost the same as the [index()](https://www.w3schools.com/python/ref_string_index.asp) method, the only difference is that the index() method raises an exception (error) if the value is not found.   a\_string = 'Terry Nguyen'  position = a\_string.find('y', 5)  print(position)  # Output: 9  **string.strip()** The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)  txt = "    banana    "  x = txt.strip()  print(x)  # Output: ‘banana’  **Example 1: string.strip(character)**  txt = ",,,,,rrttgg.....banana....rrr"  x = txt.strip(",.grt")  print(x)  # Output: ‘banana’  **string.split()** The split() method splits a string into a list.  txt = "welcome to the jungle"  splitted\_txt = txt.split()  print(splitted\_txt)  # Output: ['welcome', 'to', 'the', 'jungle']  **Example 1 : string.split(separator, maxsplit)**  txt = "apple#banana#cherry#orange"  splitted\_txt = txt.split("#")  print(splitted\_txt)  # Output: ['apple', 'banana', 'cherry', 'orange']  **Example 2 : string.split(separator, maxsplit)**  txt = "apple#banana#cherry#orange"  splitted\_txt = txt.split("#", 2)  print(splitted\_txt)  # Output: ['apple', 'banana', 'cherry#orange']  More string methods can be found using **dir(object)** method and Python documentation or follow this link:  [https://www.w3schools.com/python/python\_ref\_string.asp](%20https:/www.w3schools.com/python/python_ref_string.asp) |

|  |
| --- |
| **LIST**  **LIST METHODS** |

|  |
| --- |
| **DICTIONARY**  **DICTIONARY METHODS**  **dictionary.get()** The get() method returns the value of the item with the specified key. If the key does not exist, it will return None.  items = {      "computer" : 10,      "printer" : 8,      "mouse" : 15,  }  quantity = items.get("computer")  print(quantity)  # Output:  10  **Example 1 :** dictionary.get(keyname, value)  items = {      "computer" : 10,      "printer" : 8,      "mouse" : 15,  }  quantity = items.get("microphone")  print(quantity)  # Output:  None  **Example 2 :** dictionary.get(keyname, value)  items = {      "computer" : 10,      "printer" : 8,      "mouse" : 15,  }  quantity = items.get("microphone", 12)  new\_quantity = items.get("webcam", "The item does not exist")  print(quantity)  # Output:  12  print(new\_quantity)  # Output: The item does not exist  print(items)  # Output:  {'computer': 10, 'printer': 8, 'mouse': 15}  The original list is not subject to any modication by the get() method even if a key and value pair of a new item is provided.  **dictionary.setdefault()** The setdefault() method returns the value of the item with the specified key.  **Example 1 :** If the key provided exists, its value remains the same.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.setdefault("model", "Bronco")  print(x)  # Output:  Mustang  print(car)  # Output:  {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}  **Example 2 :** If the key provided does not exist, this becomes the key’s value. The original dictionary is also updated. Otherwise, default value None.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.setdefault("colour", "White")  print(x)  # Output:  White  print(car)  # Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'colour': 'White'}  **dictionary.keys()** returns a view object. The view object contains the keys of the dictionary, as a list. The view object cannot be modified nor iterated.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.keys()  print(x)  # Output:  dict\_keys(['brand', 'model', 'year'])  **Example 1 :** When an item is added in the dictionary, the view object also get updated  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.keys()  car["color"] = "white"  print(x)  # Output:  dict\_keys(['brand', 'model', 'year', 'color'])  The view object can be converted into a list by using list() method.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = list(car.keys())  print(x)  # Output:  ['brand', 'model', 'year']  **dictionary.fromkeys()** returns a dictionary with the specified keys and the specified value  x = ['key1', 'key2', 'key3']  thisdict = dict.fromkeys(x)  print(thisdict)  # {'key1': None, 'key2': None, 'key3': None}  **Example 1:** dict.fromkeys(keys, value)  x = ['key1', 'key2', 'key3']  thisdict = dict.fromkeys(x, "one")  print(thisdict)  # Output: {'key1': 'one', 'key2': 'one', 'key3': 'one'}  **dictionary.items()** The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.  The view object will reflect any changes done to the dictionary  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.items()  print(x)  # Output: dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])  **Example 1 :** When an item is added or changed in the dictionary, the view object also get updated  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.items()  car["year"] = 2018  print(x)  # Output: dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2018)])  **Example 2:** keys and values can be accessed using a for loop  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  for keys, values in car.items():    print(keys, values)    # Output:  # brand Ford  # model Mustang  # year 1964  **dictionary.updates()** The update() method inserts the specified items to the dictionary. The specified items can be a dictionary, or an iterable object with key value pairs.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  car.update({"color": "White"})  print(car)  # Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}  **dictionary.values()** The values() method returns a view object. The view object contains the values of the dictionary, as a list. The view object will reflect any changes done to the dictionary  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.values()  print(x)  # Output: dict\_values(['Ford', 'Mustang', 1964])  **Example 1:** When an item is added or changed in the dictionary, the view object also get updated  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.values()  car["year"] = 2018  print(x)  # Output: dict\_values(['Ford', 'Mustang', 2018])  **Example 2:**  The view object can be converted to a list using list() method  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = list(car.values())  print(x)  # Output: ['Ford', 'Mustang', 1964]  **dictionary.copy()** The copy() method returns a shallow copy of the specified dictionary.  **dictionary.pop()**  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  car.pop("model")  print(car)  # Output: {'brand': 'Ford', 'year': 1964}  **Example 1:** dictionary.pop(keyname, defaultvalue)  **defaultvalue**: Optional. A value to return if the specified key do not exist.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.pop("colour", "Item does not exist")  print(x)  # Output: Item does not exist  **Example 2:** If the key specified is not found and no defaultvalue is provided. Python will raise an error  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.pop("colour")  print(x)  # Output:  # Traceback (most recent call last):  #   File "d:\Local Disk E\PYTHON BOOTCAMP\draft.py", line 8, in <module>  #     x = car.pop("colour")  # KeyError: 'colour'  **Example 3:**  The value of the popped item can be stored inside a variable. Which could be used for other purposes.  car = {    "brand": "Ford",    "model": "Mustang",    "year": 1964  }  x = car.pop("brand")  print(x)  # Output: Ford  **dictionary.popitem()**  **dictionary.clear()** |