Dictionaries

- The dictionary is another Python data structure. It's not a sequence type (but can be easily adapted to sequence processing) and it is mutable.
- The list of pairs is surrounded by curly braces, while the pairs themselves are separated by commas, and the keys and values by colons.

```
dictionary = {“Pakistan": “Islamabad", “Bangladesh": “Dhaka“ }
phone_numbers = {'Ali':344122222, 'Naveed':3000222222}
empty_dictionary = {}
print(dictionary)
print(phone_numbers)
print(empty_dictionary)
print(dictionary[‘Pakistan'])
print(phone_numbers[‘Naveed'])
```

- **if the key is a string, you have to specify it as a string;**
- **keys are case-sensitive: ’Ali' is something different from ’ali**

Dictionaries

- The word you look for is named a **key**. The word you get from the dictionary is called a **value**.
- Each key must be **unique** – it's not possible to have more than one key of the same value;
- A key may be any **immutable** type of object: it can be a number (integer or float), or even a string, but not a list;
- A dictionary is **not a list** – a list contains a set of numbered values, while a **dictionary holds pairs of values**;
- the **len()** function works for dictionaries, too – it returns the number of key-value elements in the dictionary;
- A dictionary is a **one-way tool** – if you have an English-Urdu dictionary, you can look for urdu equivalents of English terms, but not vice versa.

Dictionaries

- When you write a big or lengthy expression, it may be a good idea to keep it vertically aligned. This is how you can make your code more readable and more programmer-friendly, e.g.:

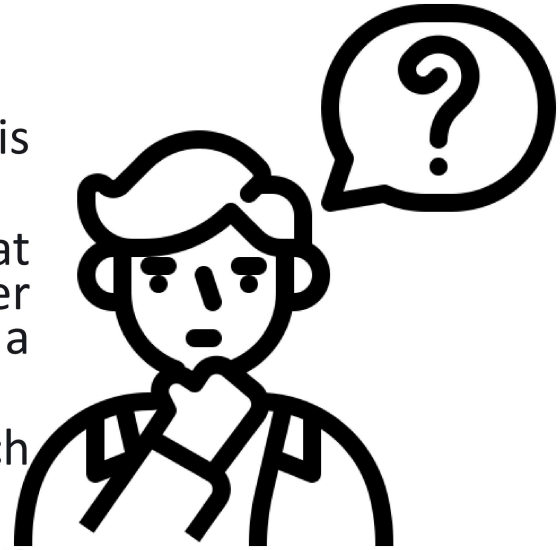
```
dictionary = {
             “Pakistan": “Islamabad",
             “Bangladesh": “Dhaka“
}
phone_numbers = {
             'Ali':344122222,
             'Naveed':3000222222
}
print(dictionary)
print(phone_numbers)
```

This kind of formatting is called a **hanging indent**

Dictionaries

Can dictionaries be browsed using the for loop, like lists or tuples?

No, because a dictionary is **not a sequence type** – the for loop is useless with it.



- Yes, because there are simple and very effective tools that can **adapt any dictionary to the for loop requirements** (in other words, building an intermediate link between the dictionary and a temporary sequence entity).
- The first of them is a method named keys(), possessed by each dictionary.
- The method **returns an iterable object consisting of all the keys gathered within the dictionary**.

```
dictionary = {"cat": “bali", "dog": “kuta", "horse": “ghura"}

for key in dictionary.keys():
    print(key, "->", dictionary[key])
```

Dictionaries

```
Dictionary = {"cat": "bali", "dog": "kuta", "horse": "ghura"}
for english, urdu in dictionary.items():
    print(english, "->", urdu)
```

Dictionaries are fully **mutable**, there are no obstacles to modifying them.

```
dictionary = {“Pakistan": “Islamabad", “Bangladesh": “Dhaka“ }
phone_numbers = {'Ali':344122222, 'Naveed':3000222222}
print(dictionary)
dictionary['Pakistan'] = 'Karachi'
print(dictionary)
```

Dictionaries-Adding new value

- Adding a new key-value pair to a dictionary is as simple as changing a value – you only have to assign a value to a new, **previously non-existent key**.

```
dictionary = {"Pakistan": "Islamabad", “Bangladesh": “Dhaka“ }
print(dictionary)
dictionary['India'] = 'Delhi'
print(dictionary)
```

- You can also insert an item to a dictionary by using the update() method

```
dictionary = {"Pakistan": "Islamabad", “Bangladesh": “Dhaka“ }
print(dictionary)
dictionary['India'] = dictionary.update({“India": “Delhi"})
print(dictionary)
```

Replace Function

- The `replace()` method replaces a specified phrase with another specified phrase

syntax *string*.replace(*oldvalue, newvalue, count*)

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three")

print(x)
```

```
three three was a race horse, two two was three too.
```

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three", 2)

print(x)
```

```
three three was a race horse, two two was one too.
```

Replace Function

```
original_string = "Remove spaces from this string."

new_string = original_string.replace(" ", "")

print("Original String:", original_string)
print("New String:", new_string)
```

```
Original String: Remove spaces from this string.
New String: Removespacesfromthisstring.
```

Write a Python program that allows the user to input a string, a word they want to replace in the string, and the new word they want to replace it with. The program should then perform the replacement and display the modified string.

strip() Method

- Removes leading (leftmost) and trailing (rightmost) whitespaces by default

Syntax *string*.strip(*characters*)

```
original_string = "   Hello, world!   "
stripped_string = original_string.strip()
```

Hello, world!

```
text = "\nHello, World!\n"
stripped_text = text.strip("\n")
print(stripped_text)
```

Hello, World!

rstrip() Method

- The **rstrip()** method removes any **trailing characters (characters at the end a string)**, space is the default trailing character to remove.

Syntax *string*.rstrip(*characters*)

```
original_string = "***Python is awesome!***"
stripped_string = original_string.rstrip("*")

print("Original String:", original_string)
print("Stripped String:", stripped_string)
```

```
Original String: ***Python is awesome!***
Stripped String: ***Python is awesome!
```

rstrip() Method

```
original_string = "###   Clean me up!   ###"
stripped_string = original_string.rstrip("#")

print("Original String:", original
_string)
print("Stripped String:", stripped_string)
```

```
Original String:          ###   Clean me up!   ###
Stripped String:          ###   Clean me up!
```

Create a basic Python program that receives a user input string containing extra whitespaces at the end. Implement the rstrip() method to remove trailing whitespaces and then display the cleaned string.

lstrip() Method

- In Python, the **lstrip()** method is used to remove **leading (leftmost) characters** or a specified **set of characters from a string**. It returns a new string with the leading characters removed.

Syntax *string*.lstrip(*characters*)

```
txt = ",,,,,ssaaww.....banana"

x = txt.lstrip(",.asw")

print(x)
```

```
banana
```

lstrip() Method

```
original_string = "___Hello, Python!"
stripped_string = original_string.lstrip("_")

print("Original String:", repr(original_string))
print("Stripped String:", repr(stripped_string))
```

```
Original String: '___Hello, Python!'
Stripped String: 'Hello, Python!'
```

Write a Python program that takes user input for a sentence containing underscores. Utilize the lstrip() method to remove these leading characters and display the cleaned sentence.

Split Function

- Python, the **split() function** is a built-in method used to split a string into a **list of substrings** based on a specified delimiter.

Syntax string.split(separator, maxsplit)

- **Separator:** The seperator based on which the string will be split. If not specified, whitespace characters (spaces, tabs, and newlines) are used by default.
- **maxsplit:** Specifies the maximum number of splits. **Default is -1**, meaning "**all occurrences.**

```
Splitting a string into words
sentence = "Hello world, how are you today?"
words = sentence.split()
print(words)
```

```
['Hello', 'world,', 'how', 'are', 'you', 'today?']
```

Split Function

Splitting a CSV (Comma-Separated Values) string with a specific separator and limiting the number of splits

```
csv_data = "John,Doe,30,New York,USA"
fields = csv_data.split(',', 2)
print(fields)
```

```
['John', 'Doe', '30,New York,USA']
```

```
sentence = "Python is an awesome programming language"
words = sentence.split(" ", 1)
print(words)
```

```
['Python', 'is an awesome programming language']
```

Replace Function

- The `replace()` method replaces a specified phrase with another specified phrase

syntax *string*.replace(*oldvalue, newvalue, count*)

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three")

print(x)
```

```
three three was a race horse, two two was three too.
```

```
txt = "one one was a race horse, two two was one too."

x = txt.replace("one", "three", 2)

print(x)
```

```
three three was a race horse, two two was one too.
```

Replace Function

```
original_string = "Remove spaces from this string."

new_string = original_string.replace(" ", "")

print("Original String:", original_string)
print("New String:", new_string)
```

```
Original String: Remove spaces from this string.
New String: Removespacesfromthisstring.
```

Write a Python program that allows the user to input a string, a word they want to replace in the string, and the new word they want to replace it with. The program should then perform the replacement and display the modified string.

strip() Method

- Removes leading (leftmost) and trailing (rightmost) whitespaces by default

Syntax *string*.strip(*characters*)

```
original_string = "   Hello, world!   "
stripped_string = original_string.strip()
```

Hello, world!

```
text = "\nHello, World!\n"
stripped_text = text.strip("\n")
print(stripped_text)
```

Hello, World!

rstrip() Method

- The **rstrip()** method removes any **trailing characters (characters at the end a string)**, space is the default trailing character to remove.

Syntax *string*.rstrip(*characters*)

```
original_string = "***Python is awesome!***"
stripped_string = original_string.rstrip("*")

print("Original String:", original_string)
print("Stripped String:", stripped_string)
```

```
Original String: ***Python is awesome!***
Stripped String: ***Python is awesome!
```

rstrip() Method

```
original_string = "###   Clean me up!   ###"
stripped_string = original_string.rstrip("#")

print("Original String:", original
_string)
print("Stripped String:", stripped_string)
```

```
Original String:          ###   Clean me up!   ###
Stripped String:          ###   Clean me up!
```

Create a basic Python program that receives a user input string containing extra whitespaces at the end. Implement the rstrip() method to remove trailing whitespaces and then display the cleaned string.

lstrip() Method

- In Python, the **lstrip()** method is used to remove **leading (leftmost) characters** or a specified **set of characters from a string**. It returns a new string with the leading characters removed.

Syntax *string*.lstrip(*characters*)

```
txt = ",,,,,ssaaww.....banana"

x = txt.lstrip(",.asw")

print(x)
```

```
banana
```

lstrip() Method

```
original_string = "___Hello, Python!"
stripped_string = original_string.lstrip("_")

print("Original String:", repr(original_string))
print("Stripped String:", repr(stripped_string))
```

```
Original String: '___Hello, Python!'
Stripped String: 'Hello, Python!'
```

Write a Python program that takes user input for a sentence containing underscores. Utilize the lstrip() method to remove these leading characters and display the cleaned sentence.

Split Function

- Python, the **split() function** is a built-in method used to split a string into a **list of substrings** based on a specified delimiter.

Syntax string.split(separator, maxsplit)

- **Separator:** The seperator based on which the string will be split. If not specified, whitespace characters (spaces, tabs, and newlines) are used by default.
- **maxsplit:** Specifies the maximum number of splits. **Default is -1**, meaning "**all occurrences.**

```
Splitting a string into words
sentence = "Hello world, how are you today?"
words = sentence.split()
print(words)
```

```
['Hello', 'world,', 'how', 'are', 'you', 'today?']
```

Split Function

```
Splitting a CSV (Comma-Separated Values) string with a specific separator and
limiting the number of splits
csv_data = "John,Doe,30,New York,USA"
fields = csv_data.split(',', 2)
print(fields)
```

```
['John', 'Doe', '30,New York,USA']
```

```
sentence = "Python is an awesome programming language"
words = sentence.split(" ", 1)
print(words)
```

```
['Python', 'is an awesome programming language']
```