Angelle D. Salvadora
CS3C

Dictionaries

Dictionaries are the Python programming language's way of implementing data structures. A Python dictionary consists of several key-value pairs; each pair maps the key to its associated value.

● **Creation of New Dictionary**

— To create a dictionary with items, you need to include key-value pairs inside the curly braces. Inside the curly braces you have a **key-value pair**. Keys are separated from their associated values with colon, **:**.

Syntax: dictionary_name = {key: value}

```
[1]: my_information = dict({'name': 'Angelle' ,'age': 21,'address': 'Muntinlupa'})

     print(my_information)
     print(type(my_information))
```

```
{'name': 'Angelle', 'age': 21, 'address': 'Muntinlupa'}
<class 'dict'>
```

● **Accessing Items in the Dictionary**

— We can access the item by using the dictionary name and square bracket notation, but this time in the square brackets you specify a key. Each key corresponds with a specific value, so you mention the key that is associated with the value you want to access.

Syntax: dictionary_name[key]

```
[1]: my_information = dict({'name': 'Angelle' ,'age': 21,'address': 'Muntinlupa'})

     print(my_information['name'])
```

```
Angelle
```

● **Change Values in the Dictionary**

— To update a dictionary, we can use the built-in **update()** method. This method is particularly helpful when you want to update more than one value inside a dictionary at the same time.

Syntax: my_dictionary.update()

```
[1]: my_information = dict({'name': 'Angelle' ,'age': 21,'address': 'Muntinlupa'})

     my_information.update(name= 'Angelle Salvadora', age = 22, address = "Muntinlupa")

     print(my_information)
```

```
{'name': 'Angelle Salvadora', 'age': 22, 'address': 'Muntinlupa'}
```

● Loop Through a Dictionary Values

— A **for loop** is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
[1]: my_information = {'name':'Angelle Salvadora','email':'angellesalvadora@yahoo.com'}
     for k,v in my_information.items():
         print(k, v)
```

```
name Angelle Salvadora
email angellesalvadora@yahoo.com
```

● Check if Key Exists in the Dictionary

— The **keys()** method returns a list that contains only the keys that are inside the dictionary.

```
[1]: year_of_creation = {'Python': 1993, 'JavaScript': 1995, 'HTML': 1993}

     print(year_of_creation.keys())
```

```
dict_keys(['Python', 'JavaScript', 'HTML'])
```

● Checking for Dictionary Length

— The **len()** function returns the total length of the object that is passed as an argument. When a dictionary is passed as an argument to the function, it returns the total number of key-value pairs enclosed in the dictionary.

```
[1]: my_information = {'name': 'Angelle', 'age': 21, 'address': 'Muntinlupa'}

     print(len(my_information))
```

```
3
```

● Adding Items in the Dictionary

— To add a key-value pair to a dictionary, use square bracket notation. First, specify the name of the dictionary. Then, in square brackets, create a key and assign it a value.

Syntax: dictionary_name[key] = value

```
[1]: my_information = dict({'name': 'Angelle' ,'age': 22,'address': 'Muntinlupa'})

     #add another key-value pairs
     my_information['zodiac sign'] = "Aries"

     #print dictionary
     print(my_information)
```

```
{'name': 'Angelle', 'age': 22, 'address': 'Muntinlupa', 'zodiac sign': 'Aries'}
```

● **Removing Items in the Dictionary**

— **popitem()** method takes no arguments, but removes and returns the last key-value pair from a dictionary.

```
[1]: my_information = dict({'name': 'Angelle', 'age': 22, 'address': 'Muntinlupa', 'zodiac sign': 'Aries'})

     popped_item = my_information.popitem()

     print(my_information)
     print(popped_item)
```

```
{'name': 'Angelle', 'age': 22, 'address': 'Muntinlupa'}
('zodiac sign', 'Aries')
```

— clear() method in python is used to erase all the elements present inside the dictionary. It does not take any parameters from the user and has to return value. It returns an empty dictionary with no elements.

```
[2]: my_information = dict({'name': 'Angelle', 'age': 22, 'address': 'Muntinlupa', 'zodiac sign': 'Aries'})

     my_information.clear()

     print(my_information)
```

```
{}
```

● **Remove an Item Using del Statement**

— **del()** statement is used to delete objects. In Python everything is an object, so the del keyword can also be used to delete variables, lists, or parts of a list etc.

```
[1]: my_information = dict({'name': 'Angelle', 'age': 22, 'address': 'Muntinlupa', 'zodiac sign': 'Aries'})

     del my_information['address']

     print(my_information)
```

```
{'name': 'Angelle', 'age': 22, 'zodiac sign': 'Aries'}
```

● **The dict() Constructor**

— The dict () method in Python is a constructor that is used to create dictionaries. Depending upon the parameters that are passed in the function, a dictionary of any length can be created using the constructor.

```
[1]: x = dict(name = "Gelle", age = 21, birthday = "April 9")

     print(x)

     {'name': 'Gelle', 'age': 21, 'birthday': 'April 9'}
```

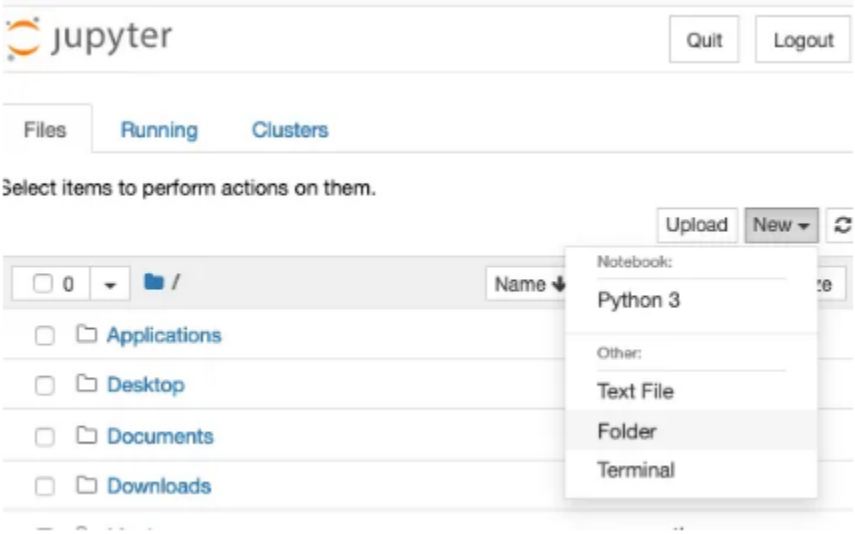
● **Dictionary Methods**

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Jupyter notebook

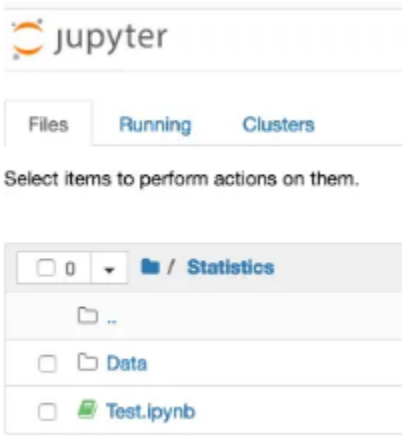
● Adding Folder

— Press Launch Jupyter Notebook. It will start a terminal and open a browser. To create a folder, click the New button on the top right. Then click Folder.



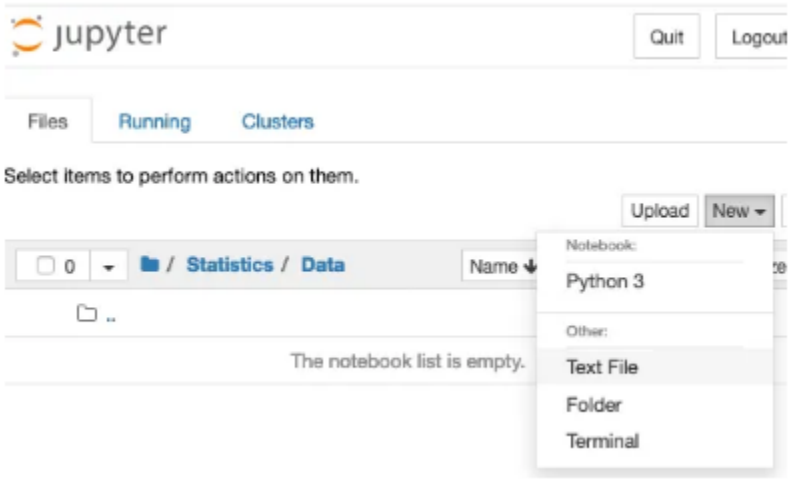
You need to rename the folder to Statistics.

Create a folder called Data and we are going to store all our data in this folder.



● Adding Text file

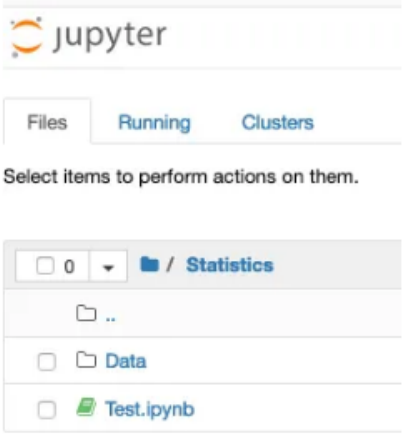
— In the Data folder, create a text file called mydata1.csv.



● CSV file for data analysis and visualization

— Open http://bit.ly/2M1yrW7 in a browser. Command(or ctrl in windows) + A to select all and Command + C to copy. Command + V to paste it to the mydata1.csv and Command + S to save the file.

In the Statistics folder, create a file called Test by New > Python 3.



That’s all for setup. Let’s start writing some codes.

● **Import libraries**

— Use ***import ... as ...*** to import a library. We use ***%matplotlib inline*** to show graphs inline. In our Test file please write/copy the following.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

After typing the above, **press SHIFT + RETURN**. This will run the code you wrote. If you have written correctly, nothing is going to happen. But if you have misspelled it then it will return an error. Please read the last line of error and fix the problem.

● **Finding data**

— There are many resources you can find online. This is a collection of data sources. CSV file is a comma-separated values file and is the easiest file to work on. Most of the data sources provided a CSV file. Find a couple of data you are interested in from the list. Download them to your hard drive.

● **Importing data**

— We are going to read the CSV file and store the data in a variable called ***df***. In the next cell please write the following.

df = pd.read_csv('data/mydata1.csv')

Please **do not** forget to press **SHIFT + RETURN**.

● **Data attributes**

— We are going to see the overview of our data using shape, ***info(), head(), tail(), describe().***

We use **shape** to print the dimension of our data. **info()** prints a concise summary of our data. Press SHIFT + RETURN to run the program.

df.shape

It prints out (569, 33).

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                       569 non-null int64
diagnosis                569 non-null object
radius_mean              569 non-null float64
texture_mean             569 non-null float64
perimeter_mean           569 non-null float64
area_mean                569 non-null float64
smoothness_mean          569 non-null float64
```