Tapalla,John Harven M.
BSCS3C

Dictionaries

● Creation of New Dictionary

A dictionary in Python is a collection of key-value pairs, where each key is unique. It is created using
curly braces `{}` and can contain various data types as keys and values.

Sample:

```
#Creation of New Dictionary
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
print(my_dict)
```

Output:

```
{'apple': 5, 'banana': 10, 'orange': 7}
```

● Accessing Items in the Dictionary

Items in a dictionary can be accessed by specifying the key within square brackets `[]`.

Sample:

```
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
print("Value of 'banana':", my_dict['banana'])
```

Output:

```
Value of 'banana': 10
```

● Change Values in the Dictionary

Values in a dictionary can be changed by assigning a new value to the corresponding key.

Sample:

```
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
my_dict['banana'] = 15
print("Value of 'banana':", my_dict['banana'])
```

Output:

```
Value of 'banana': 15
```

● Loop Through a Dictionary Values

You can loop through the values of a dictionary using a loop construct like `for`.

Sample:

```
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
for value in my_dict.values():
print(value)
```

Output:

```
5
10
7
```

● Check if Key Exists in the Dictionary

You can check if a key exists in a dictionary using the `in` keyword.

Sample:

```
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
```

Tapalla,John Harven M.
BSCS3C

key_to_check = 'apple'
if key_to_check in my_dict:
print(f"'{key_to_check}' exists in the dictionary.")
Output:
'apple' exists in the dictionary.

● **Checking for Dictionary Length**
You can determine the number of key-value pairs in a dictionary using the `len()` function.
Sample:
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
print("Length of the dictionary:", len(my_dict))
Output:
Length of the dictionary: 3

● **Adding Items in the Dictionary**
You can add new key-value pairs to a dictionary by assigning a value to a new key.
Sample:
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
my_dict['grapes'] = 12
print(my_dict)
Output:
{'apple': 5, 'banana': 10, 'orange': 7, 'grapes': 12}

● **Removing Items in the Dictionary**
You can remove items from a dictionary using various methods like `del` statement, `pop()` method,
or `popitem()` method.
Sample:
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
removed_value = my_dict.pop("orange")
print(my_dict)
Output:
{'apple': 5, 'banana': 10}

● **Remove an Item Using del Statement**
You can remove a specific item from a dictionary using the `del` statement followed by the key.
Sample:
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
del my_dict['orange']
print(my_dict)
Output:
{'apple': 5, 'banana': 10}

● The dict() Constructor

Tapalla,John Harven M.
BSCS3C

The `dict()` constructor is used to create a new dictionary. It can take various forms of inputs like sequences of key-value pairs or keyword arguments.

Sample:

```
new_dict = dict([('apple', 5), ('banana', 10), ('orange', 7)])
print(new_dict)
```

Output:

```
{'apple': 5, 'banana': 10, 'orange': 7}
```

● Dictionary Methods

Python dictionaries have various built-in methods for performing operations like retrieving keys, values, and items, adding or removing elements, etc.

Sample:

```
my_dict = {'apple': 5, 'banana': 10, 'orange': 7}
print("Keys in the dictionary:", my_dict.keys())
print("Items in the dictionary:", my_dict.items())
```

Output:

```
Keys in the dictionary: dict_keys(['apple', 'banana', 'orange'])
Items in the dictionary: dict_items([('apple', 5), ('banana', 10), ('orange', 7)])
```

Jupyter Notebook

● Adding Folder

Method 1:

To add a folder in Jupyter Notebook, you can use Python's `os` module to create a new directory.

Sample:

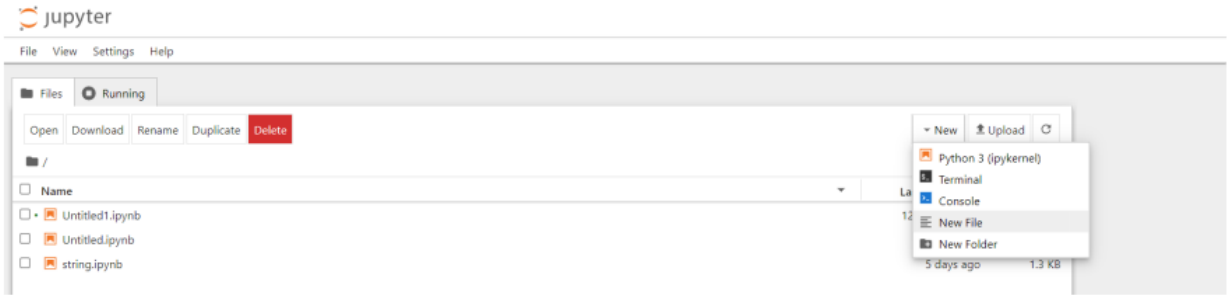
```
import os
# Define the folder name
folder_name = 'my_folder'
# Specify the directory path where you want to create the folder
directory_path = '/path/to/your/directory/'
# Create the folder
os.makedirs(os.path.join(directory_path, folder_name))
```

Method 2:

In the upper right-hand corner of the Jupyter Notebook home screen, click on the "New" drop-down
button and select "Folder" as shown in the image below. A new folder called "Untitled Folder" will
appear in the list of files on the Jupyter Notebook home screen.

Tapalla,John Harven M.
BSCS3C

● **Adding Text file**
To add a text fle in Jupyter Notebook, you can use the Jupyter Notebook interface directly:
1. Click on the "New" button.
2. Select "Text File" from the dropdown menu.
3. This will create a new text fle in the current directory where you can add your text content.



● **CSV file for data analysis and visualization**
Files using CSV (Comma Separated Values) are frequently used for data display and analysis. Python packages such as Pandas may be used to read CSV fles and manipulate the data they contain. Libraries such as Matplotlib and Seaborn can be used for visualization.

1. **Import necessary libraries**
We import the pandas library as `pd` and the matplotlib.pyplot module as `plt`. Pandas provides data
manipulation and analysis tools, allowing us to work with structured data efciently. Matplotlib is a powerful plotting library used for creating visualizations in Python, enabling us to generate various
types of plots to explore and visualize our data.

Sample:
import pandas as pd
import matplotlib.pyplot as plt

2. Load data from a CSV fle
Using the `pd.read_csv()` function, we read the data from a CSV fle named 'data.csv' and store it in a
pandas DataFrame called `data`. A DataFrame is a 2-dimensional labeled data structure with

columns that can be of different types, providing a convenient way to work with tabular data in Python.

Sample:
# Assuming you have a CSV fle named 'data.csv'
data = pd.read_csv('data.csv')

3. Explore the data
We use the `.head()` method to display the frst few rows of the DataFrame, allowing us to quickly
inspect the structure and contents of the loaded data. This initial exploration helps us understand
the format of the data and identify any potential issues or patterns.
Sample:
# Display the frst few rows of the DataFrame
data.head()
4. Perform data analysis
We conduct data analysis tasks such as generating descriptive statistics using the `.describe()` method, which summarizes the central tendency, dispersion, and shape of the numerical data in the
DataFrame. Additionally, we can manipulate the data by fltering rows based on specifc conditions
and performing aggregation operations on grouped data to gain insights and answer questions about the dataset.

Sample:
# Summary statistics
data.describe()

# Filter data
fltered_data = data[data['column_name'] > threshold]

# Group data
grouped_data = data.groupby('column_name').mean()

5. Data visualization
Utilizing matplotlib's plotting functions, we create visual representations of the data to gain further
insights and communicate fndings effectively. We can create various types of plots, such as scatter
plots and histograms, to visualize relationships between variables, distributions of values, and other
patterns present in the data

Tapalla,John Harven M.
BSCS3C

Sample:

```
# Plotting
data.plot(x='x_column', y='y_column', kind='scatter')
plt.title('Scatter Plot')
plt.xlabel('X Label')
plt.ylabel('Y Label')
plt.show()

# Histogram
data['column'].plot(kind='hist', bins=20)
plt.title('Histogram')
plt.xlabel('X Label')
plt.ylabel('Frequency')
plt.show()
```

6. Exporting modifed data

After performing data manipulation and analysis, we may need to save the modifed DataFrame to a new CSV fle for further use or sharing. We use the `.to_csv()` method to export the DataFrame to a CSV fle, specifying parameters such as the fle name and whether to include the index column in the exported fle.

Sample:

```
# Export the modifed DataFrame to a new CSV fle
fltered_data.to_csv('fltered_data.csv', index=False)
```

7. Further analysis and visualization

We can continue exploring and analyzing the data, iterating through steps 4 to 6 as needed to delve deeper into specifc aspects of the dataset or address additional questions. By refning our analysis and visualization techniques, we can extract meaningful insights and make informed decisions based on the data.

● Import libraries

To import libraries in Jupyter Notebook, you can use standard Python import statements.

Sample:

```
import pandas as pd
import matplotlib.pyplot as plt
```

● Finding data

You can fnd data from various sources such as online repositories, libraries like Seaborn, or by

Tapalla,John Harven M.
BSCS3C

creating your own dataset.
Sample:
```
import seaborn as sns
iris_df = sns.load_dataset('iris')
```

● **Importing data**
To import data into your Jupyter Notebook, you can use appropriate methods based on the fle format.

Sample:
```
import pandas as pd
df = pd.read_csv('your_data.csv')
```

● **Data attributes**
After importing the data, you can explore its attributes using methods like `head()`, `info()`, and `describe()`:

Sample:
```
# Display the frst few rows of the dataframe
print(df.head())

# Display concise summary of the dataframe
print(df.info())

# Generate descriptive statistics
print(df.describe())
```