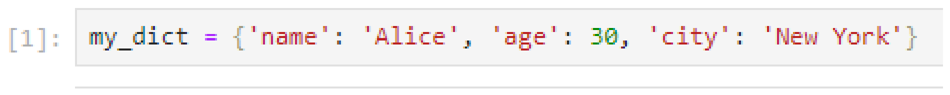
Malaluan, Nervin C. BSCS - 3C Group 3

Using Jupyter Notebook:

Dictionaries

Creation of New Dictionary: You can create a dictionary using curly braces {} and specifying key-value pairs separated by colons. For example:

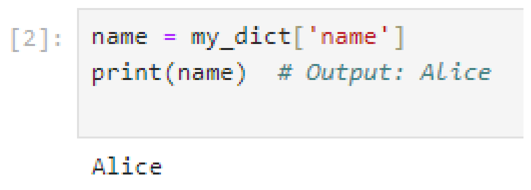
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}



```
[1]: my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

Accessing Items in the Dictionary: Use the key within square brackets [] to access the corresponding value.

name = my_dict['name']
print(name) # Output: Alice

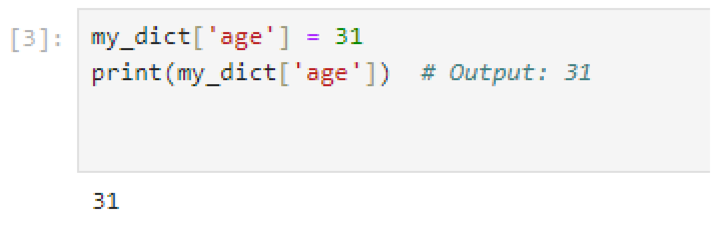


```
[2]: name = my_dict['name']
     print(name)  # Output: Alice

     Alice
```

Change Values in the Dictionary: Assign a new value to the key within square brackets.

my_dict['age'] = 31
print(my_dict['age']) # Output: 31

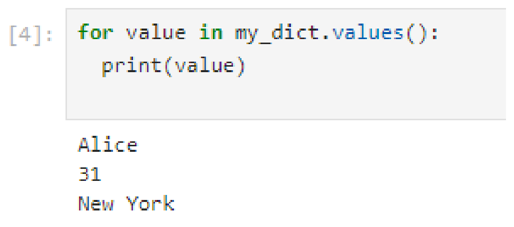


```
[3]: my_dict['age'] = 31
     print(my_dict['age'])  # Output: 31

     31
```

Loop Through Dictionary Values: Use a for loop to iterate over the values in the dictionary.

for value in my_dict.values():
  print(value)

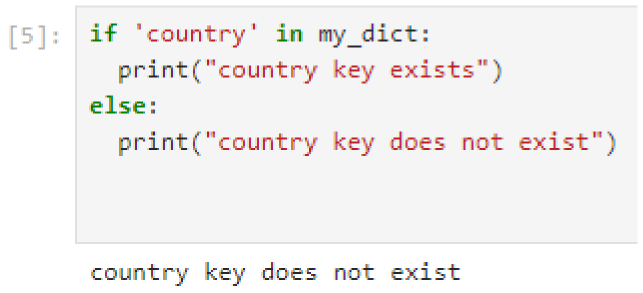


```
[4]: for value in my_dict.values():
       print(value)

     Alice
     31
     New York
```

Check if Key Exists in the Dictionary: Use the in operator to check if a key exists.

```
if 'country' in my_dict:
  print("country key exists")
else:
  print("country key does not exist")
```

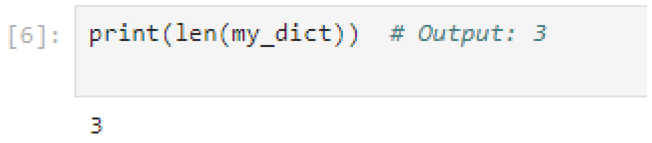


```
[5]: if 'country' in my_dict:
       print("country key exists")
     else:
       print("country key does not exist")

     country key does not exist
```

Checking for Dictionary Length: Use the len() function to get the number of key-value pairs.

```
print(len(my_dict))  # Output: 3
```

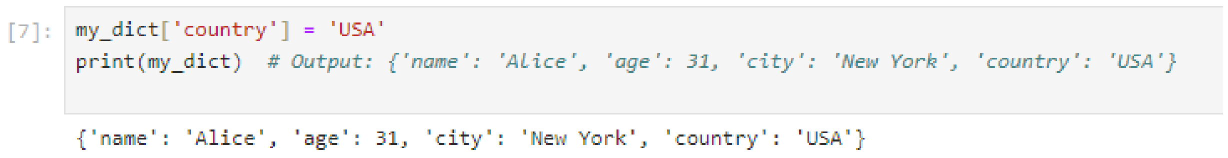


```
[6]: print(len(my_dict))  # Output: 3

     3
```

Adding Items in the Dictionary: You can add new key-value pairs using the assignment operator with the key in square brackets.

```
my_dict['country'] = 'USA'
print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}
```

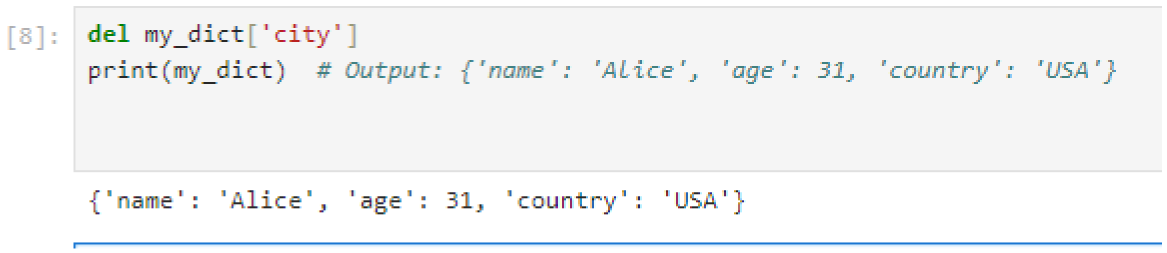


```
[7]: my_dict['country'] = 'USA'
     print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}

     {'name': 'Alice', 'age': 31, 'city': 'New York', 'country': 'USA'}
```

Removing Items in the Dictionary: Use the del keyword with the key in square brackets to remove a key-value pair.

```
del my_dict['city']
print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'country': 'USA'}
```

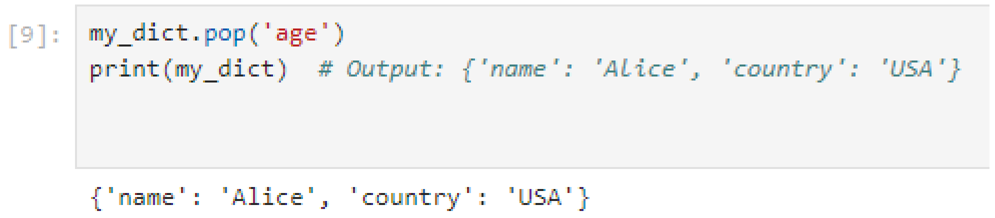


```
[8]: del my_dict['city']
     print(my_dict)  # Output: {'name': 'Alice', 'age': 31, 'country': 'USA'}

     {'name': 'Alice', 'age': 31, 'country': 'USA'}
```

Remove an Item Using del Statement: Alternatively, use the pop() method to remove a key-value pair and return the value.

my_dict.pop('age')
print(my_dict) # Output: {'name': 'Alice', 'country': 'USA'}

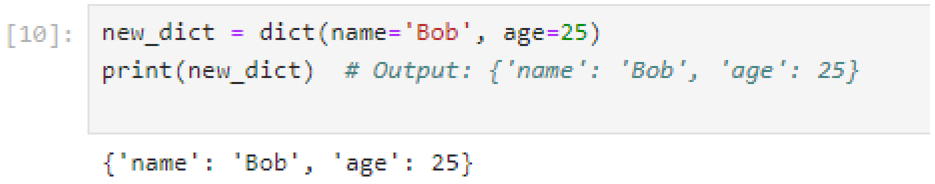


```
[9]: my_dict.pop('age')
     print(my_dict)  # Output: {'name': 'Alice', 'country': 'USA'}

     {'name': 'Alice', 'country': 'USA'}
```

The dict() Constructor: You can also create dictionaries using the dict() constructor and passing key-value pairs as arguments.

new_dict = dict(name='Bob', age=25)
print(new_dict) # Output: {'name': 'Bob', 'age': 25}

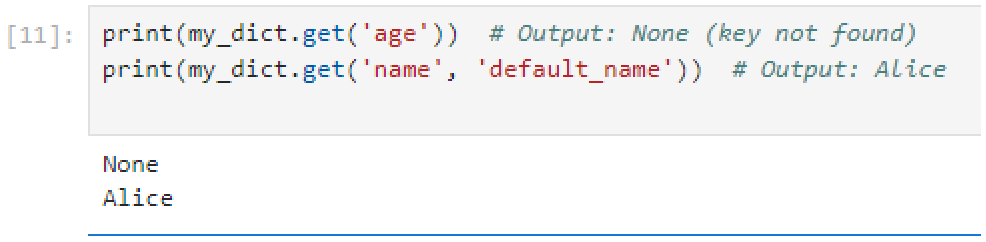


```
[10]: new_dict = dict(name='Bob', age=25)
      print(new_dict)  # Output: {'name': 'Bob', 'age': 25}

      {'name': 'Bob', 'age': 25}
```

Dictionary Methods: Dictionaries have built-in methods for various operations. For example, .get(key, default) returns the value for the key or a default value if the key doesn't exist.

print(my_dict.get('age')) # Output: None (key not found)
print(my_dict.get('name', 'default_name')) # Output: Alice



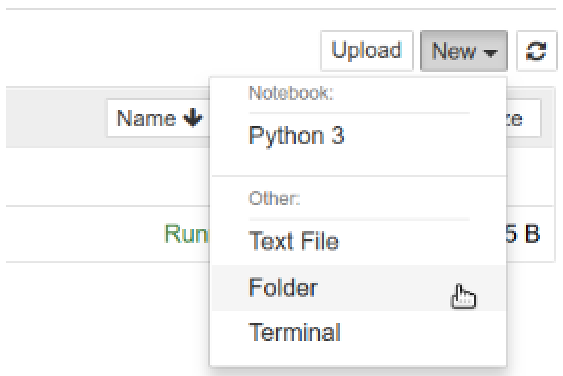
```
[11]: print(my_dict.get('age'))  # Output: None (key not found)
      print(my_dict.get('name', 'default_name'))  # Output: Alice

      None
      Alice
```

Jupyter Notebook

Adding Folders

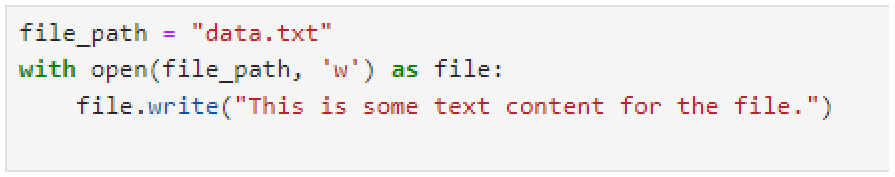
In the upper right-hand corner of the Jupyter Notebook home screen, click on the "New" drop-down button and select "Folder". A new folder called "Untitled Folder" will appear in the list of files on the Jupyter Notebook home screen.



Creating Text Files: You can use the open() function to create and write to text files. This function takes the file path and access mode ('w' for writing) as arguments.

Example:

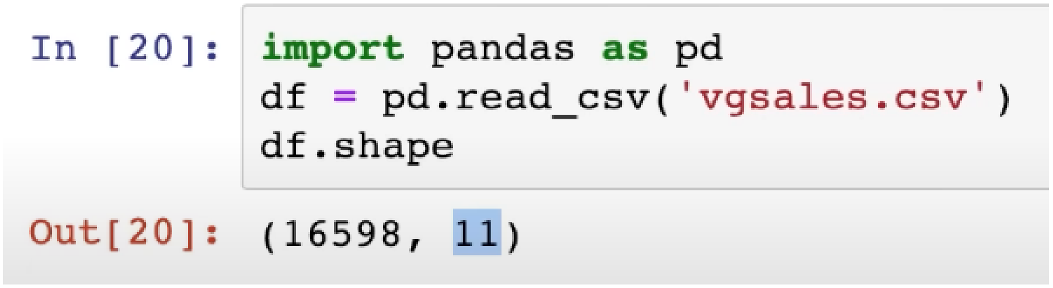
# Create a new text file named "data.txt"
file_path = "data.txt"
with open(file_path, 'w') as file:
  file.write("This is some text content for the file.")



```
file_path = "data.txt"
with open(file_path, 'w') as file:
    file.write("This is some text content for the file.")
```

CSV file for data analysis and visualization

CSV (Comma-Separated Values) files are a popular format for storing tabular data in a way that's easily readable by both humans and computers. They are ideal for data analysis and visualization in Jupyter Notebooks because of their simplicity and widespread compatibility.



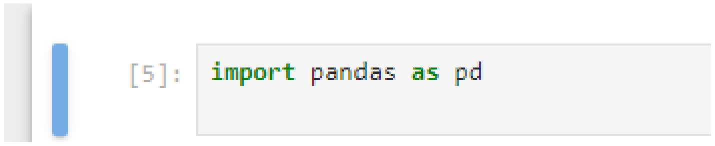
```
In [20]: import pandas as pd
         df = pd.read_csv('vgsales.csv')
         df.shape

Out[20]: (16598, 11)
```

Importing libraries: Python has a rich ecosystem of libraries for various tasks. In a Jupyter Notebook cell, you can use the import statement to import libraries like pandas for data analysis, numpy for numerical computing, or matplotlib for creating visualizations.

Example:

import pandas as pd

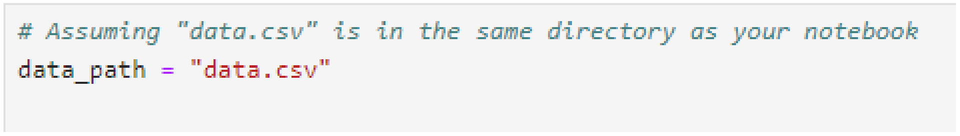


```
[5]: import pandas as pd
```

Finding data: Jupyter Notebook doesn't directly search for data, but you can use Python code within the notebook to specify the location of your data file (e.g., on your computer or cloud storage). For instance, you might use the os library to navigate directories or specify a URL to download data from the web.

Example:

Assuming "data.csv" is in the same directory as your notebook
data_path = "data.csv"



```
# Assuming "data.csv" is in the same directory as your notebook
data_path = "data.csv"
```

Importing data: Once you've identified your data source, you can use libraries like pandas to read the data. pandas offers functions like pd.read_csv() to read data from CSV files, pd.read_excel() for Excel files, and others depending on the data format.

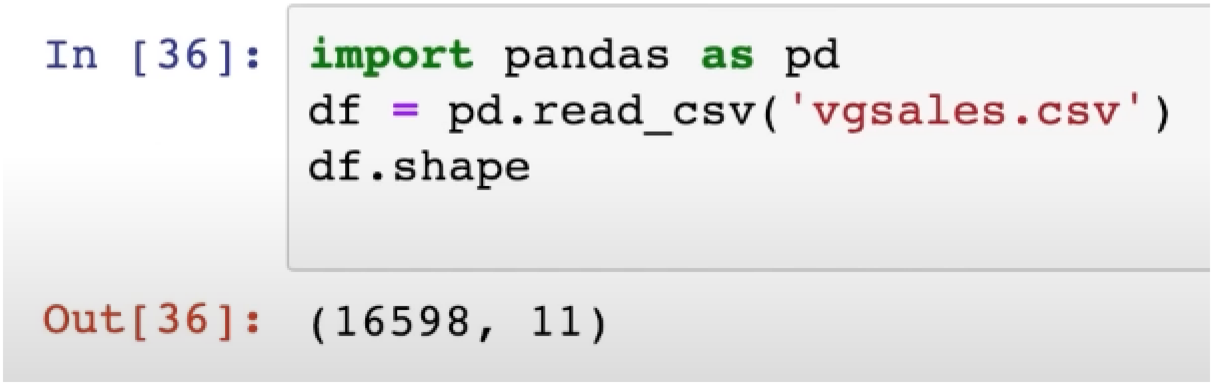
data = pd.read_csv(data_path)

```
[7]: data = pd.read_csv(data_path)
```

Data attributes: After importing the data, you can explore its attributes using the data object. You can check the number of rows and columns using data.shape, get column names using data.columns, or see a glimpse of the data using methods like data.head() (shows the first few rows). These attributes and methods help you understand the structure and content of your data.

Examples:

```
print(df.shape)  # Output: (number of rows, number of columns)
print(df.columns)  # List of column names
print(df.head())  # Show the first few rows
```



```
In [36]: import pandas as pd
         df = pd.read_csv('vgsales.csv')
         df.shape

Out[36]: (16598, 11)
```