Data Visualizatio n

Lecture 1

Introduction to Data Analysis and Python

&

Environment Preparation

Google ClassRoom Code

- For section materials, projects and updates

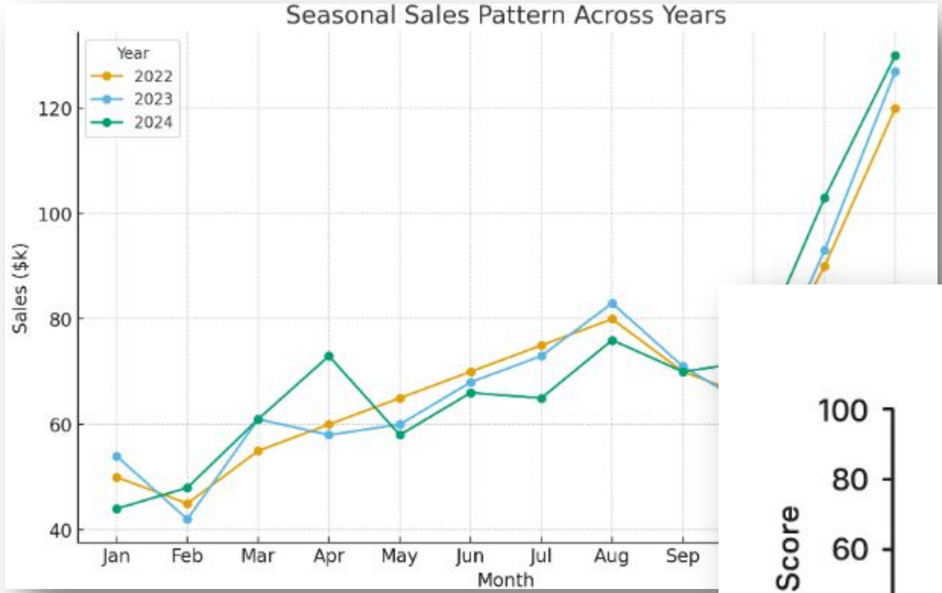
ljhz6roe

What is Data Visualization?

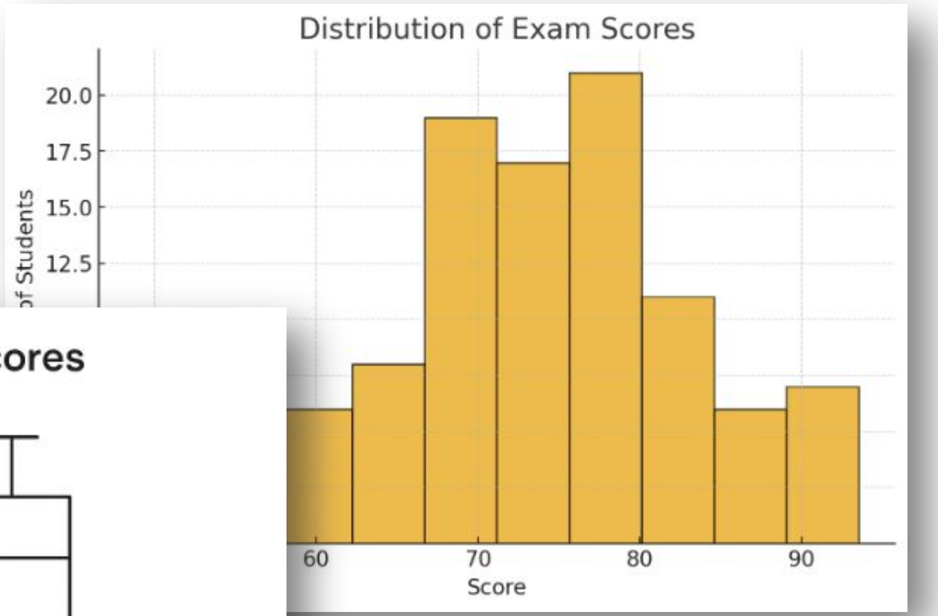
- It is the graphical representation of information and data. By using visual elements like charts, graphs, and maps.
- Data visualization tools provide an accessible way to see and **understand trends**, **outliers**, and **patterns** in data.
- It is essential to analyze massive amounts of information and make data-driven decisions.
- How to start the process?

Data Visualization

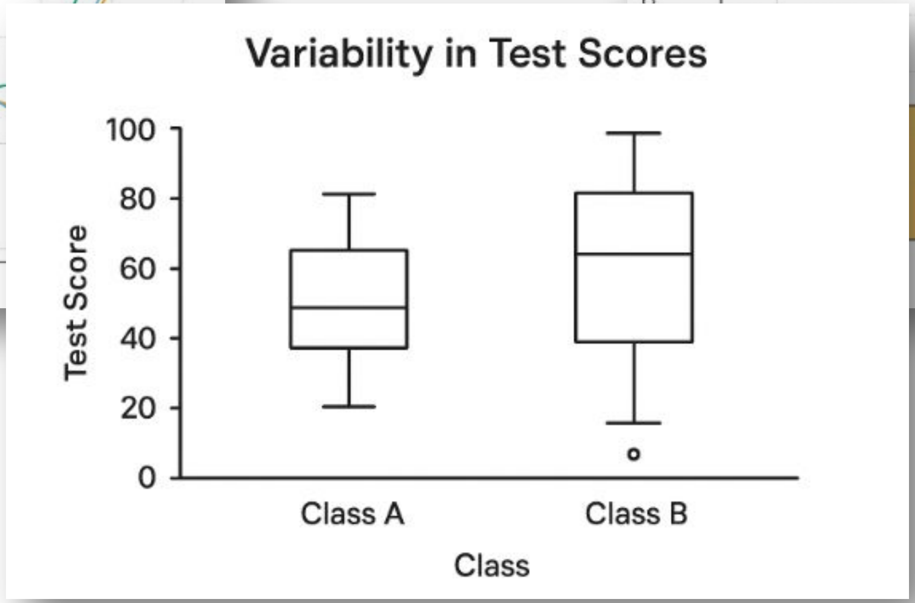
Patterns



Distribution



Variability



Data Analysis Definition

- Data analysis is the process of **cleaning, changing**, and **processing** raw data and extracting actionable, relevant information that helps businesses make informed decisions.

Data Analysis Steps

- It consists of five steps:

Defining the Problem
- What is the problem I'm trying to solve?

Collecting the Data
- Data from your organization, data from other companies and data from third party organizations

Cleaning the Data
- Removing errors, duplicates and outliers
- Making your data structured
- Filling in major gaps

Analyzing the Data
- Univariate or bivariate analysis
- Time-series analysis
- Regression analysis

Sharing the Results
- Using reports, dashboards and interactive visualizations to support your findings

Data Analysis types

- Descriptive Analysis:
  - Identifies what has already happened
- Diagnostic Analysis:
  - Understanding why something has happened
- Predictive Analysis:
  - Identify future trends based on historical data
- Prescriptive Analysis:
  - Make recommendations for the future
- Text Analysis:
  - Discover patterns residing in large datasets
  - Transforms raw data into useful business information

Data Analysis Methods

1. Qualitative Data Analysis:

The qualitative data analysis method derives data via words, symbols, pictures, and observations. This method doesn’t use statistics. The most common qualitative methods include:

- Content Analysis, for analyzing behavioral and verbal data.
- Narrative Analysis, for working with data culled from interviews, diaries, surveys.
- Grounded Theory, for developing causal explanations of a given event by studying and extrapolating from one or more past cases.

Data Analysis Methods

2. Quantitative Data Analysis:

Statistical data analysis methods collect raw data and process it into numerical data. Quantitative analysis methods include:

- Hypothesis Testing, for assessing the truth of a given hypothesis or theory for a data set or demographic.
- Mean, or average determines a subject’s overall trend by dividing the sum of a list of numbers by the number of items on the list.
- Sample Size Determination uses a small sample taken from a larger group of people and analyzed. The results gained are considered representative of the entire body.

Data Analysis Tools

1. Tableau Public
2. R Programming
3. Apache Spark
4. SAS
5. Excel
6. RapidMiner
7. Python ✔

Python for Data Analysis & Visualization

- It's a scripting language that is simple to understand, write, as well as maintain. Furthermore, it's a free open-source tool.

- It contains excellent machine learning packages such as Tensorflow, Theano, Scikitlearn, and Keras.

- Another useful characteristic of Python is that it can be built on any platform, such as a MongoDB, SQL, or JSON.

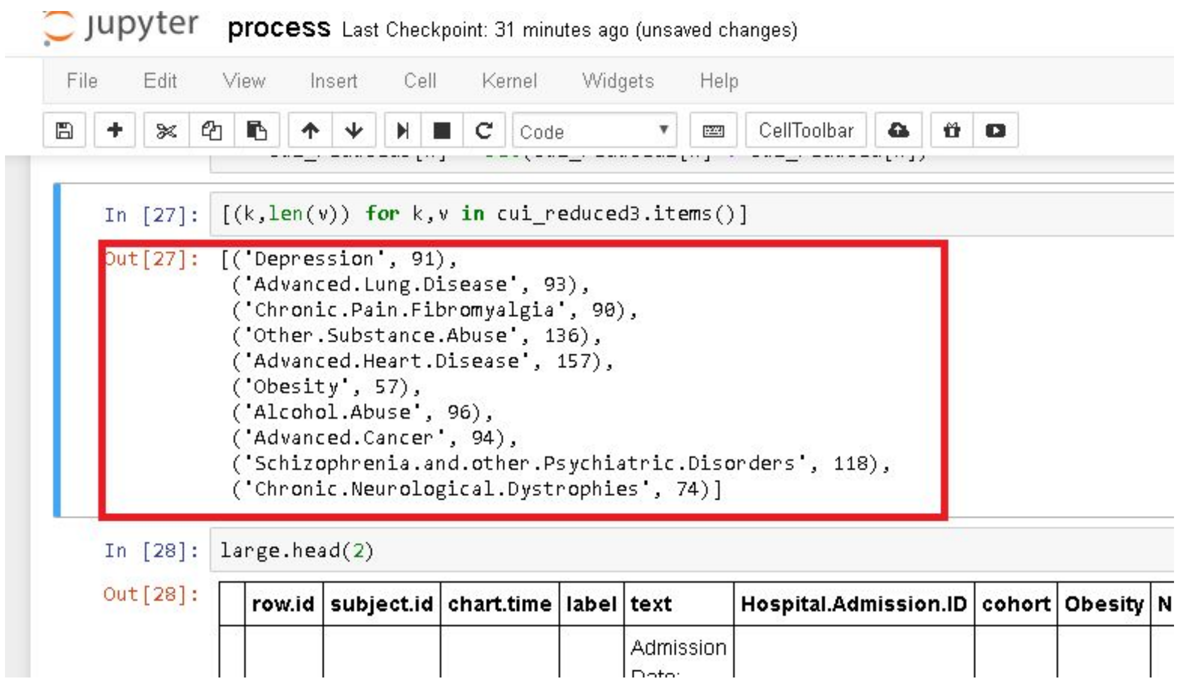
- It excels at handling text data.

Python using Jupyter Notebook

- What is Jupyter Notebook?
  - The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.
    - Language of choice
      - Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.
    - Share notebooks
      - Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.
    - Interactive output
      - Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.
    - Big data integration
      - Leverage big data tools, such as Apache Spark, from Python, R, and Scala. Explore that same data with pandas, scikit-learn, ggplot2, and TensorFlow.

Python using Jupyter Notebook

- Why use Jupyter?
  1. They're great for showcasing your work. You can see both the code and the results. The notebooks at Kaggle is a particularly great example of this.


jupyter
process Last Checkpoint: 31 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Code
CellToolbar

```
In [27]: [(k,len(v)) for k,v in cui_reduced3.items()]

Out[27]: [('Depression', 91),
          ('Advanced.Lung.Disease', 93),
          ('Chronic.Pain.Fibromyalgia', 90),
          ('Other.Substance.Abuse', 136),
          ('Advanced.Heart.Disease', 157),
          ('Obesity', 57),
          ('Alcohol.Abuse', 96),
          ('Advanced.Cancer', 94),
          ('Schizophrenia.and.other.Psychiatric.Disorders', 118),
          ('Chronic.Neurological.Dystrophies', 74)]

In [28]: large.head(2)

Out[28]:
```

	row.id	subject.id	chart.time	label	text	Hospital.Admission.ID	cohort	Obesity	N
					Admission				

Python using Jupyter Notebook

- Why use Jupyter?

2. It’s easy to use other people’s work as a starting point. You can run cell by cell to better get an understanding of what the code does.

3. Very easy to host server side, which is useful for security purposes. A lot of data is sensitive and should be protected, and one of the steps toward that is no data is stored on local machines. A server-side Jupyter Notebook setup gives you that for free.

Python using Jupyter Notebook

- How to install Jupyter Notebook?
  - There are many ways to use Jupyter notebook.
  - We will use Anaconda The world’s most popular open-source Python distribution platform
  - Please open installation guide for reference.

Graphs and Visualizations

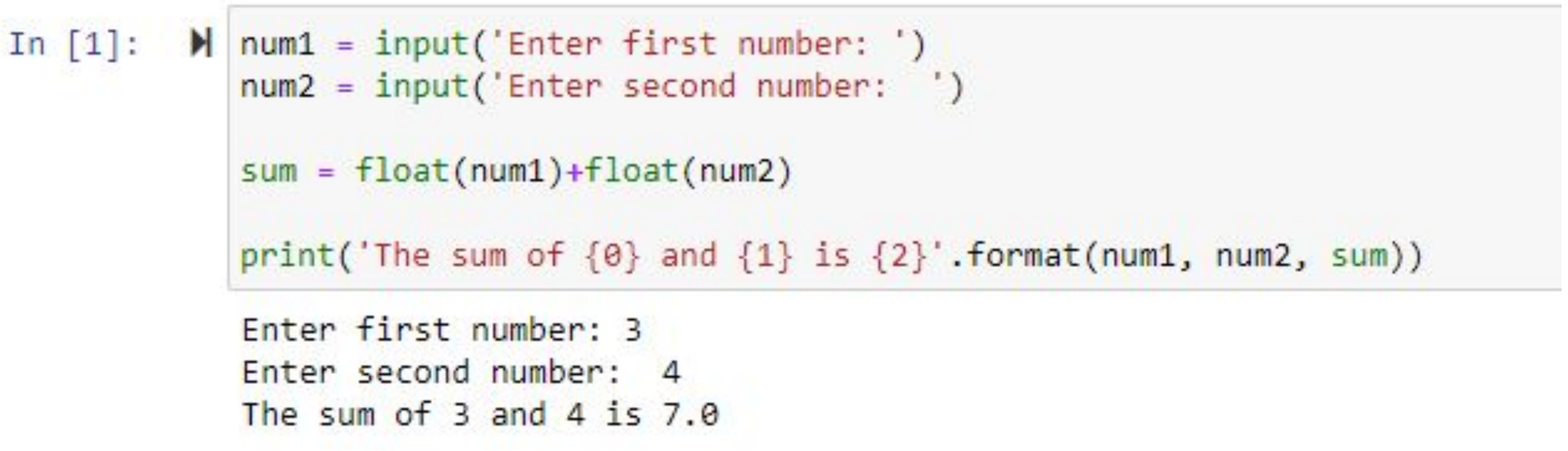
1. Charts
   - Area chart
   - Bubble chart
   - Column and bar charts
   - Funnel chart
   - Gantt chart
   - Line chart
   - Pie chart
   - Radar chart
   - Word cloud chart
   - Gauge
2. Frame Diagram

Graphs and Visualizations

3. Rectangular Tree Diagram
4. Scatter Plot
5. Maps:
   - Heat map
   - Flow map
   - Point map
   - Regional map

Python Fundamentals

- Input & Output:
  - Example:
    - Take two numbers from the user and show their sum.

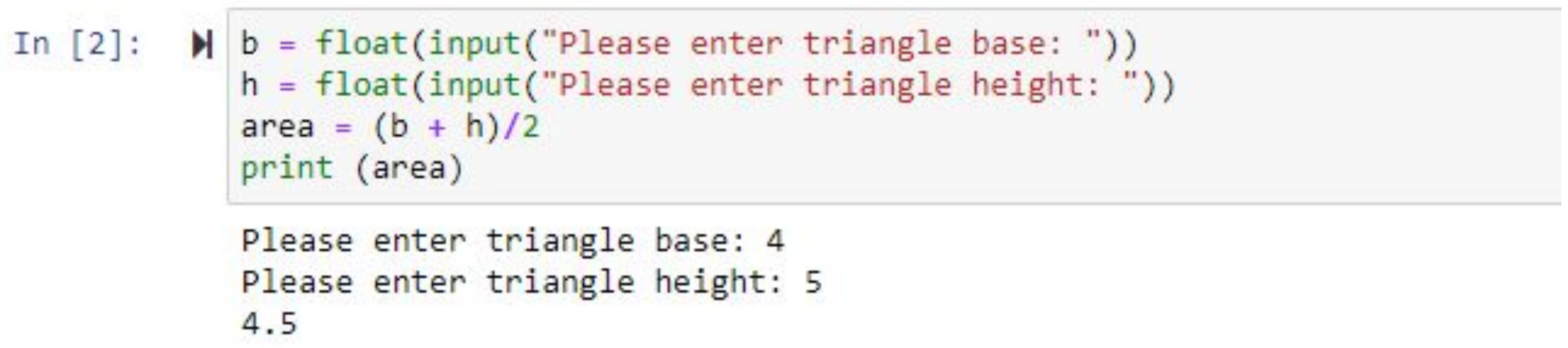


```
In [1]:
num1 = input('Enter first number: ')
num2 = input('Enter second number:  ')

sum = float(num1)+float(num2)

print('The sum of {0} and {1} is {2}'.format(num1, num2, sum))
```

```
Enter first number: 3
Enter second number:  4
The sum of 3 and 4 is 7.0
```

Python Fundamentals

- Input & Output:
  - Exercise:
    - Write a program that takes the base and height of a triangle and compute the area.

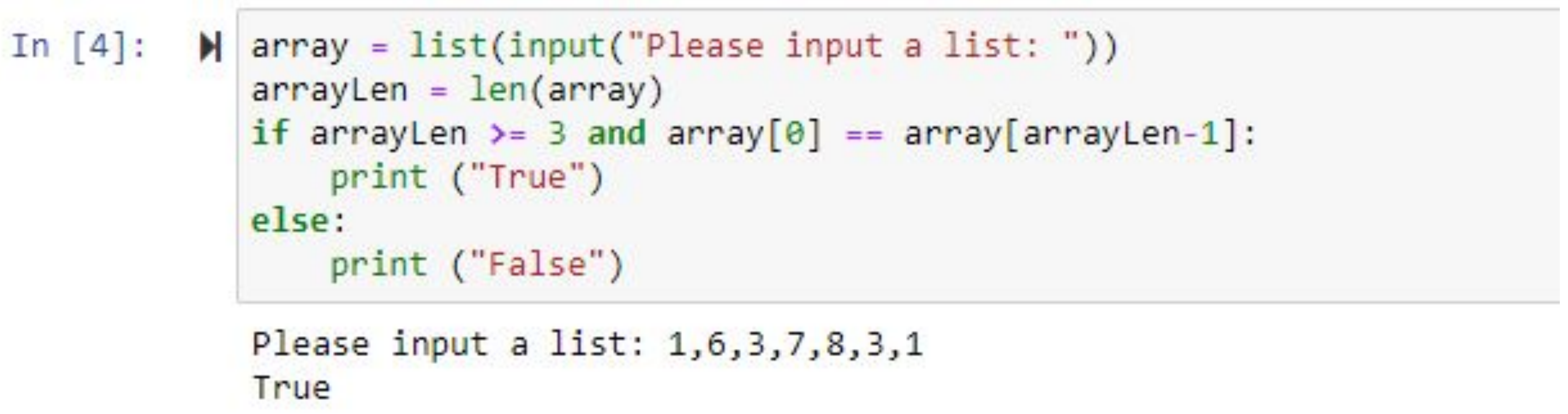


```
In [2]:
b = float(input("Please enter triangle base: "))
h = float(input("Please enter triangle height: "))
area = (b + h)/2
print (area)
```

```
Please enter triangle base: 4
Please enter triangle height: 5
4.5
```

Python Fundamentals

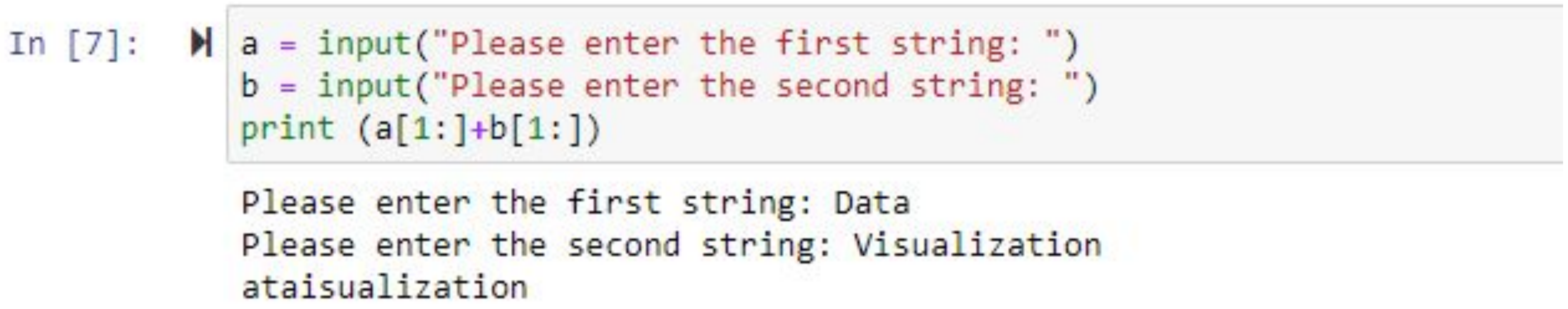
- If Condition:
  - Exercise:
    - Take from the user an array of integers, print True if the array length is 3 or more, and the first element and the last element are equal.



```
In [4]:  array = list(input("Please input a list: "))
         arrayLen = len(array)
         if arrayLen >= 3 and array[0] == array[arrayLen-1]:
             print ("True")
         else:
             print ("False")

         Please input a list: 1,6,3,7,8,3,1
         True
```

Python Fundamentals

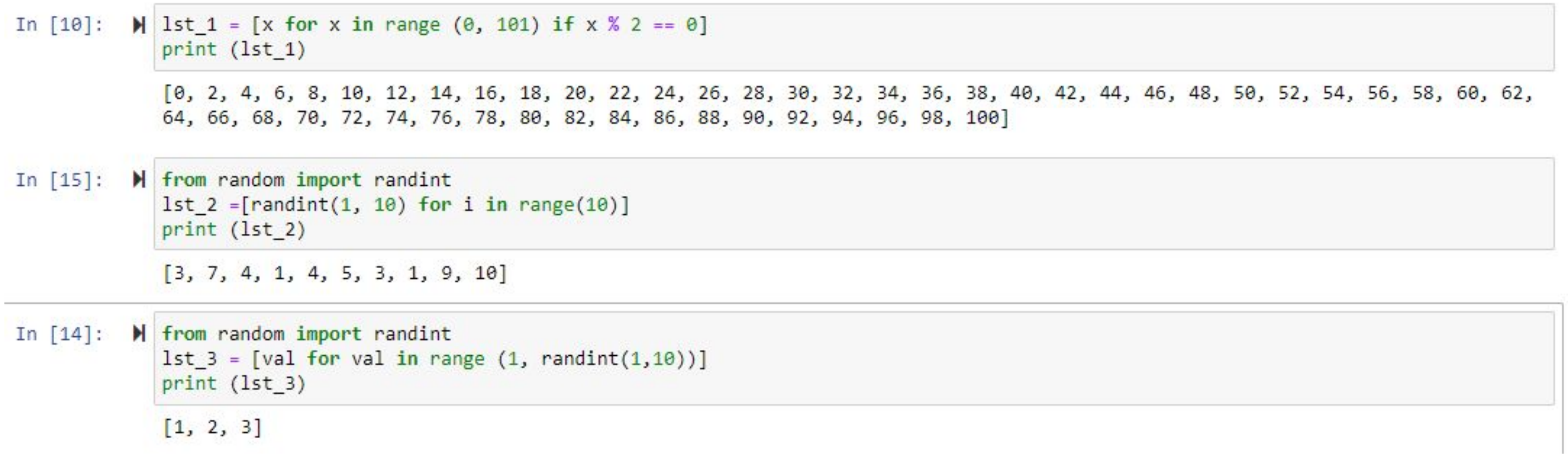
- List of Strings:
  - Example:
    - Given 2 strings, return their concatenation, except omit the first char of each. The strings will be at least length 1.



```
In [7]:  a = input("Please enter the first string: ")
         b = input("Please enter the second string: ")
         print (a[1:]+b[1:])

         Please enter the first string: Data
         Please enter the second string: Visualization
         ataisualization
```

Python Fundamentals

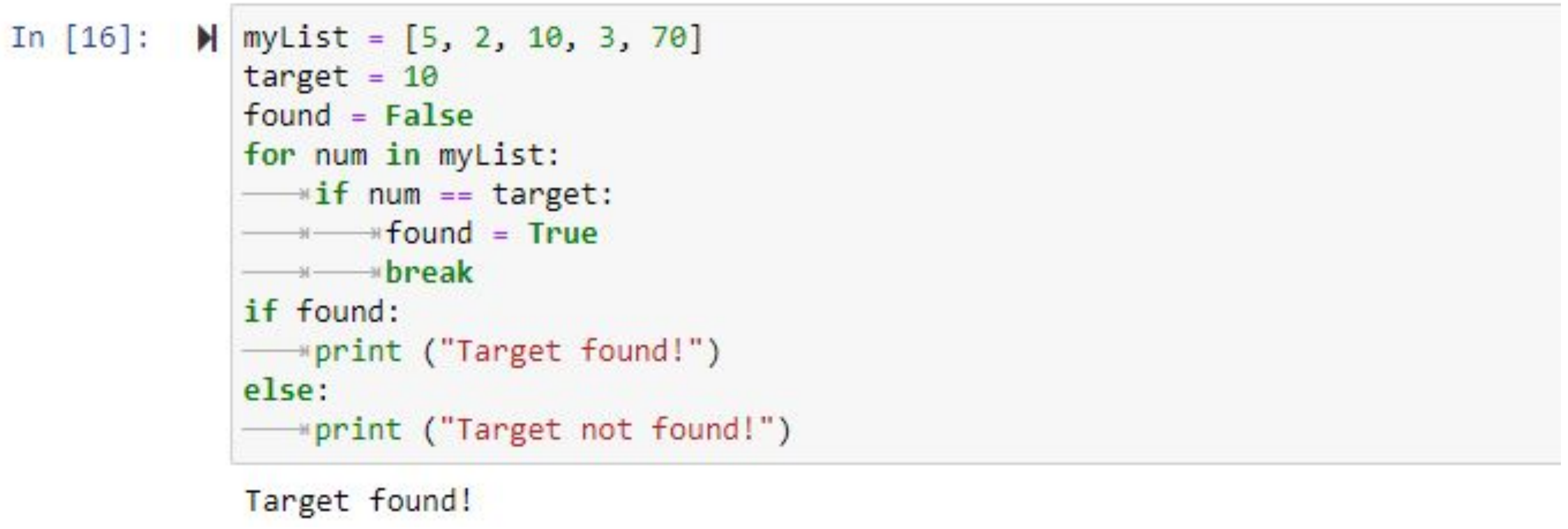
- List of Strings:
  - Exercise:
    - Write one line list comprehension expressions to define a list
      - All even numbers between 1 and 100
      - A list with 10 random numbers
      - A list whose size is a random number of elements (between 0 and 10)



```
In [10]:  lst_1 = [x for x in range (0, 101) if x % 2 == 0]
          print (lst_1)

          [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62,
          64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100]

In [15]:  from random import randint
          lst_2 =[randint(1, 10) for i in range(10)]
          print (lst_2)

          [3, 7, 4, 1, 4, 5, 3, 1, 9, 10]

In [14]:  from random import randint
          lst_3 = [val for val in range (1, randint(1,10))]
          print (lst_3)

          [1, 2, 3]
```

Python Fundamentals

- Loops:
  - Example:
    - Invent a loop that will search a list until it finds a certain number then print "found", otherwise it prints " Not found“.

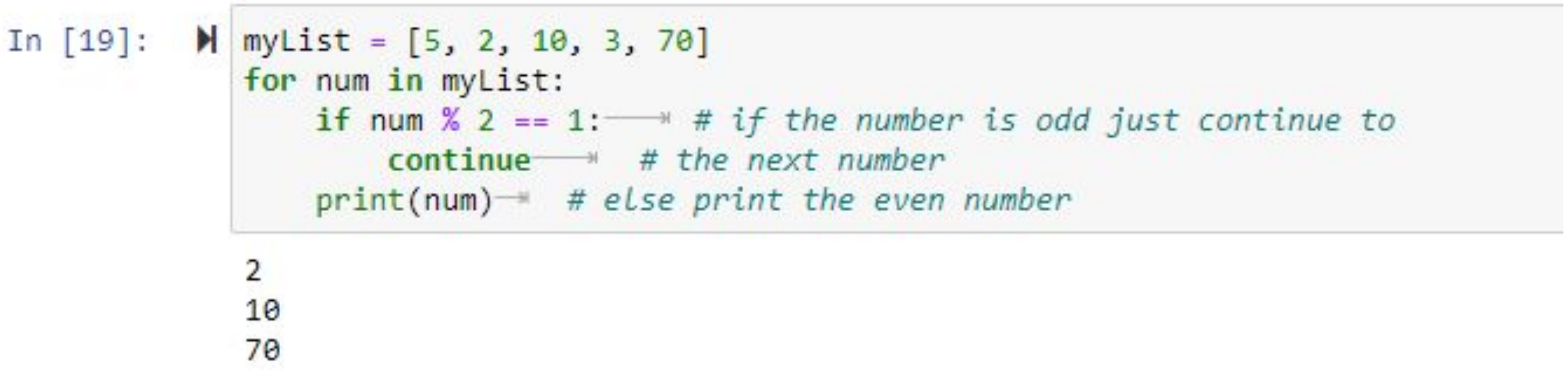


```
In [16]:
myList = [5, 2, 10, 3, 70]
target = 10
found = False
for num in myList:
    if num == target:
        found = True
        break
if found:
    print ("Target found!")
else:
    print ("Target not found!")
```

```
Target found!
```

Python Fundamentals

- Loops:
  - Exercise:
    - This loop will print only the even numbers in a list of integers.



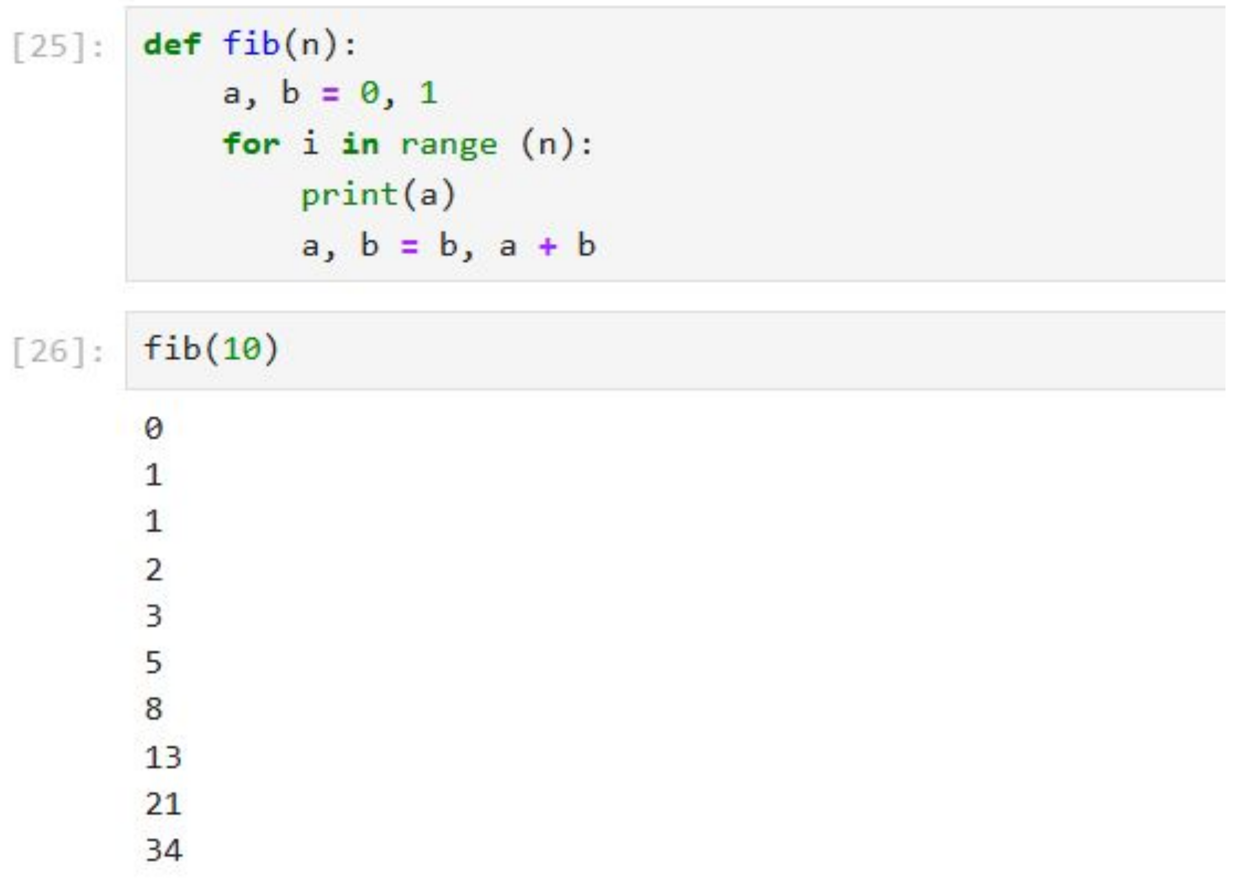
```
In [19]:  myList = [5, 2, 10, 3, 70]
          for num in myList:
              if num % 2 == 1:    # if the number is odd just continue to
                  continue        # the next number
              print(num)    # else print the even number

          2
          10
          70
```

Python Fundamentals

- Functions:
  - Exercise:
    - Define and test a function to calculate the first n Fibonacci numbers, where the first two numbers are 0 and 1 and each next number is the sum of the previous two. So, the first 10 Fibonacci numbers are: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

Python Fundamentals

- Functions:
  - Solution:



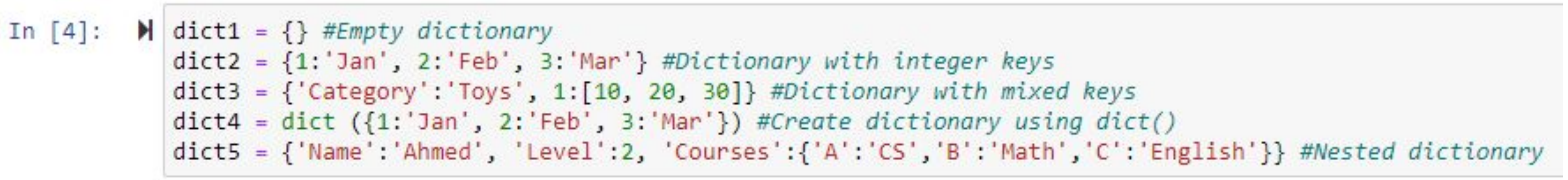
```
[25]: def fib(n):
          a, b = 0, 1
          for i in range (n):
              print(a)
              a, b = b, a + b
```

```
[26]: fib(10)
```

```
0
1
1
2
3
5
8
13
21
34
```

Python Fundamentals

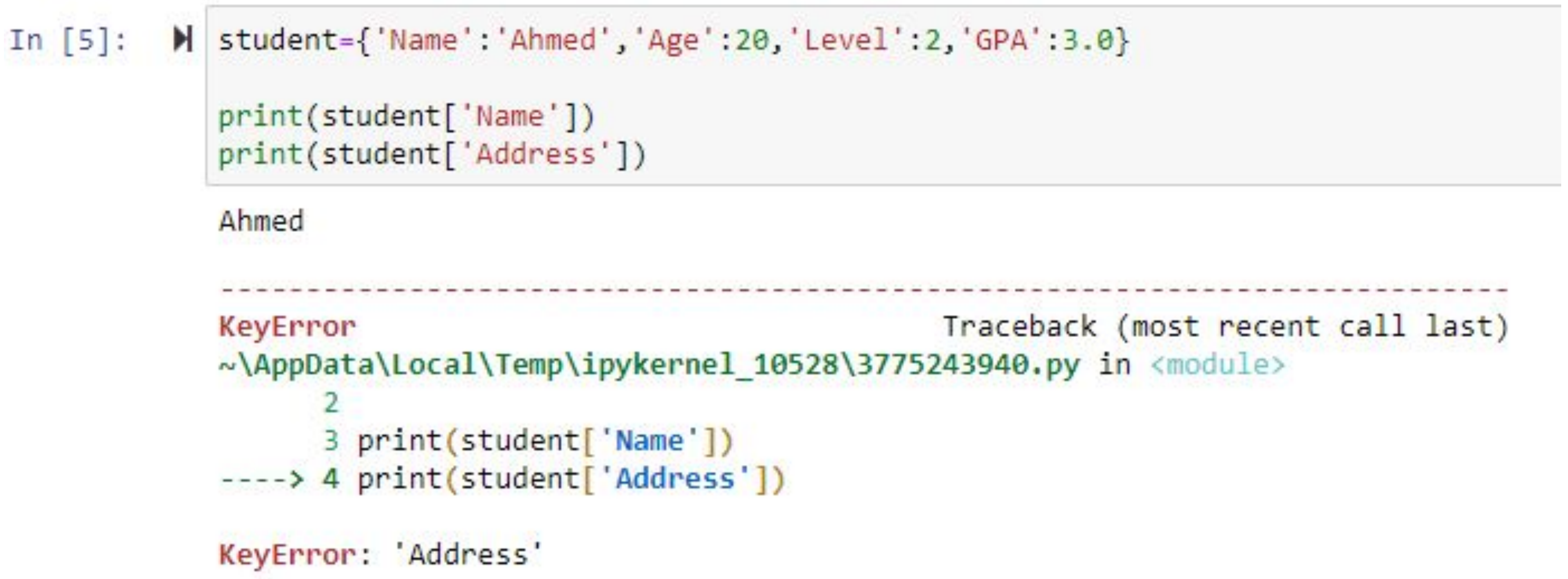
- Dictionaries:
    - Example □ Creating dictionaries:



```
In [4]:   dict1 = {} #Empty dictionary
          dict2 = {1:'Jan', 2:'Feb', 3:'Mar'} #Dictionary with integer keys
          dict3 = {'Category':'Toys', 1:[10, 20, 30]} #Dictionary with mixed keys
          dict4 = dict ({1:'Jan', 2:'Feb', 3:'Mar'}) #Create dictionary using dict()
          dict5 = {'Name':'Ahmed', 'Level':2, 'Courses':{'A':'CS','B':'Math','C':'English'}} #Nested dictionary
```

Python Fundamentals

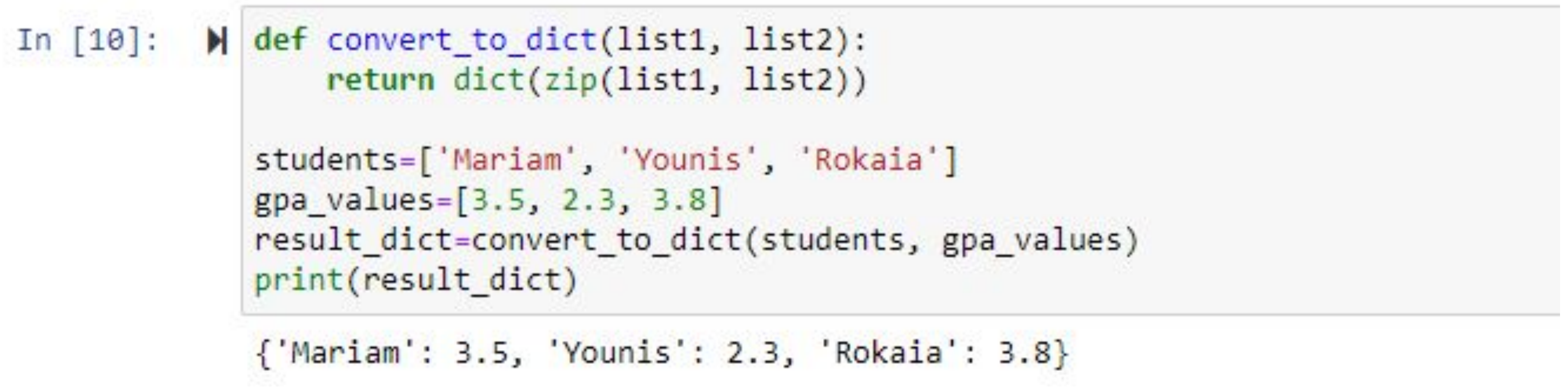
- Dictionaries:
  - Example Accessing dictionaries:



```
In [5]:  student={'Name':'Ahmed','Age':20,'Level':2,'GPA':3.0}

         print(student['Name'])
         print(student['Address'])

         Ahmed

         ---------------------------------------------------------------------------
         KeyError                                  Traceback (most recent call last)
         ~\AppData\Local\Temp\ipykernel_10528\3775243940.py in <module>
               2
               3 print(student['Name'])
         ----> 4 print(student['Address'])

         KeyError: 'Address'
```

Python Fundamentals

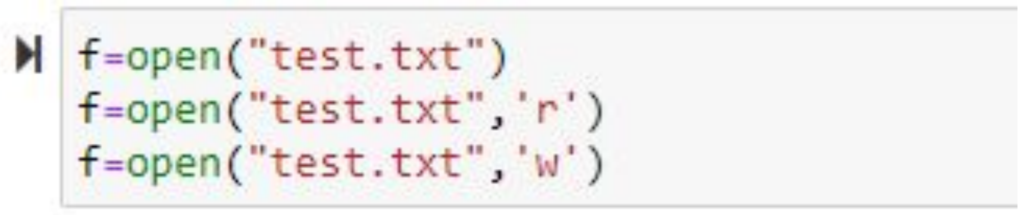
- Dictionaries:
  - Exercise:
    - Write a Python function that converts two given lists into a dictionary in a way that items from the first list are the keys and items from the second list are the values.



```
In [10]:  def convert_to_dict(list1, list2):
              return dict(zip(list1, list2))

          students=['Mariam', 'Younis', 'Rokaia']
          gpa_values=[3.5, 2.3, 3.8]
          result_dict=convert_to_dict(students, gpa_values)
          print(result_dict)

          {'Mariam': 3.5, 'Younis': 2.3, 'Rokaia': 3.8}
```

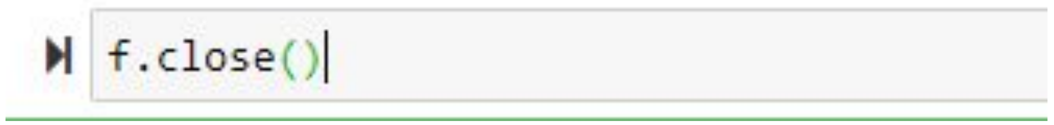
Python Fundamentals

- Files:
  - Opening a file:



```
f=open("test.txt")
f=open("test.txt",'r')
f=open("test.txt",'w')
```

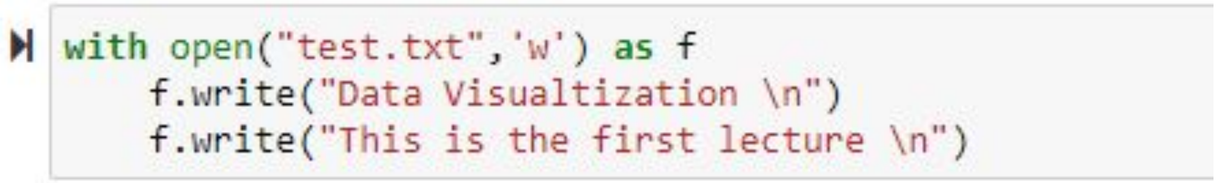
  - Closing a file:



```
f.close()
```

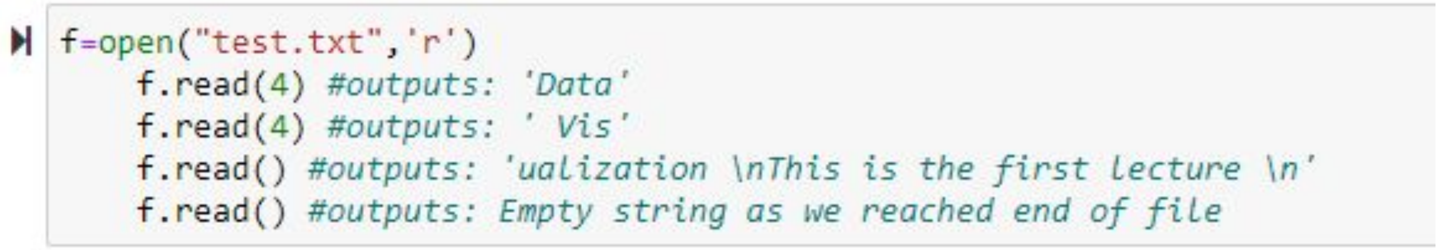
Python Fundamentals

- Files:
  - Writing to a file:



```
with open("test.txt",'w') as f
    f.write("Data Visualtization \n")
    f.write("This is the first lecture \n")
```

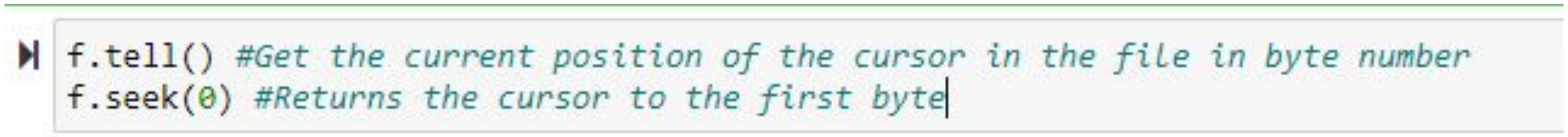
  - Reading from a file:



```
f=open("test.txt",'r')
    f.read(4) #outputs: 'Data'
    f.read(4) #outputs: ' Vis'
    f.read() #outputs: 'ualization \nThis is the first lecture \n'
    f.read() #outputs: Empty string as we reached end of file
```

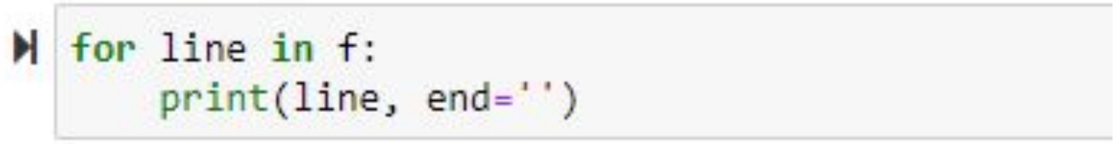
Python Fundamentals

- Files:
  - Cursor Position:



```
f.tell() #Get the current position of the cursor in the file in byte number
f.seek(0) #Returns the cursor to the first byte
```

  - Read line by line:



```
for line in f:
    print(line, end='')
```

Dealing with Datasets

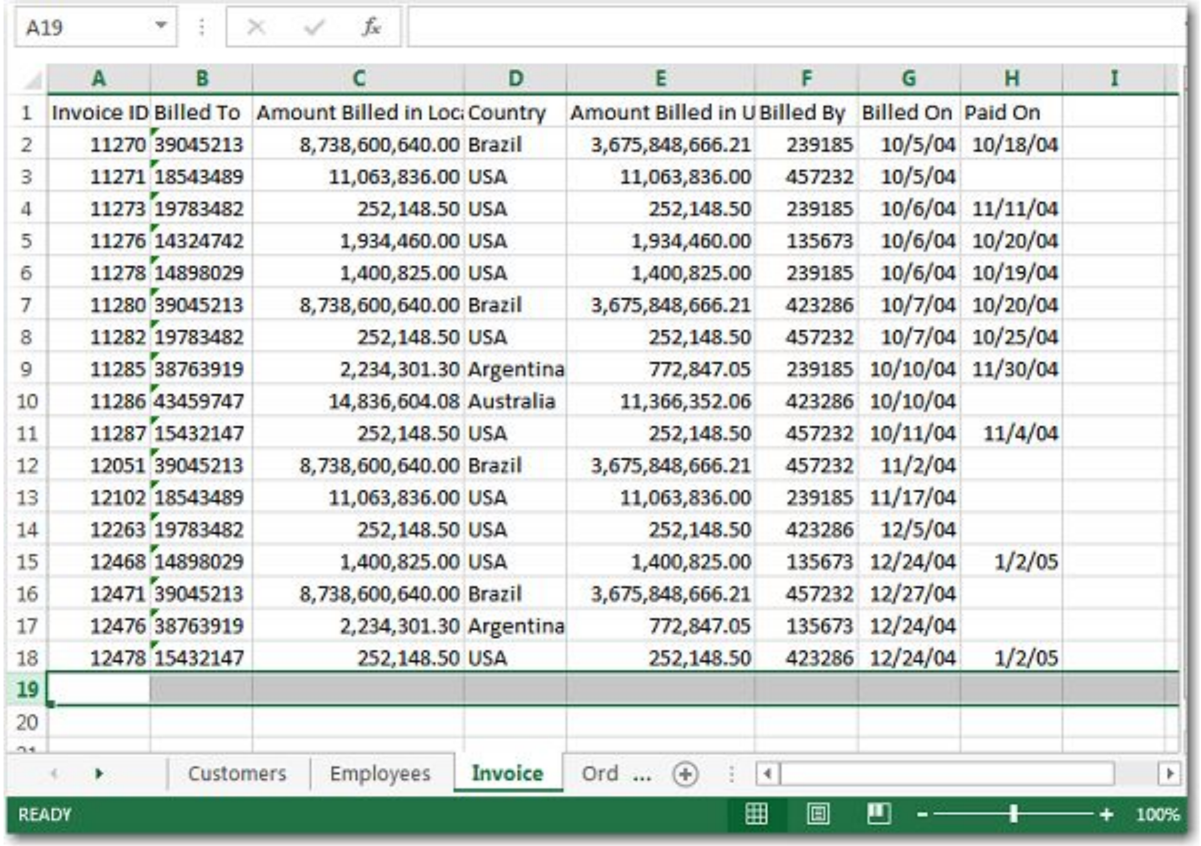
- Datasets may come in different files formats:
  - A file format is a standard way in which information is encoded for storage in a file.
    - First, the file format specifies whether the file is a binary or ASCII file.
    - Second, it shows how the information is organized.
- Based on the application you are building the file format is decided:
  - Example: In image processing you need image files as inputs and outputs, thus, the files needed will be jpeg, png or gif.

Dealing with Datasets

- Examples on different formats:
    - Comma-separated values
    - XLSX
    - ZIP
    - Plain Text (txt)
    - JSON
    - XML
    - HTML
    - Images
    - Hierarchical Data Format
    - PDF
    - DOCX
    - MP3
    - MP4
- How can we read those files in python?

Dealing with Datasets

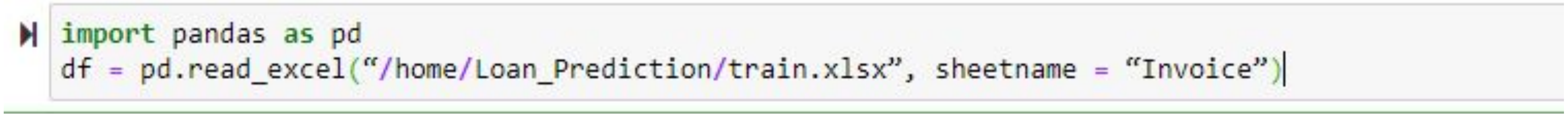
- Example 1:
  - XLSX file data is organized under the cells and columns in a sheet.



	A	B	C	D	E	F	G	H	I
1	Invoice ID	Billed To	Amount Billed in Loc	Country	Amount Billed in U	Billed By	Billed On	Paid On	
2	11270	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	239185	10/5/04	10/18/04	
3	11271	18543489	11,063,836.00	USA	11,063,836.00	457232	10/5/04		
4	11273	19783482	252,148.50	USA	252,148.50	239185	10/6/04	11/11/04	
5	11276	14324742	1,934,460.00	USA	1,934,460.00	135673	10/6/04	10/20/04	
6	11278	14898029	1,400,825.00	USA	1,400,825.00	239185	10/6/04	10/19/04	
7	11280	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	423286	10/7/04	10/20/04	
8	11282	19783482	252,148.50	USA	252,148.50	457232	10/7/04	10/25/04	
9	11285	38763919	2,234,301.30	Argentina	772,847.05	239185	10/10/04	11/30/04	
10	11286	43459747	14,836,604.08	Australia	11,366,352.06	423286	10/10/04		
11	11287	15432147	252,148.50	USA	252,148.50	457232	10/11/04	11/4/04	
12	12051	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	457232	11/2/04		
13	12102	18543489	11,063,836.00	USA	11,063,836.00	239185	11/17/04		
14	12263	19783482	252,148.50	USA	252,148.50	423286	12/5/04		
15	12468	14898029	1,400,825.00	USA	1,400,825.00	135673	12/24/04	1/2/05	
16	12471	39045213	8,738,600,640.00	Brazil	3,675,848,666.21	457232	12/27/04		
17	12476	38763919	2,234,301.30	Argentina	772,847.05	135673	12/24/04		
18	12478	15432147	252,148.50	USA	252,148.50	423286	12/24/04	1/2/05	
19									
20									

Dealing with Datasets

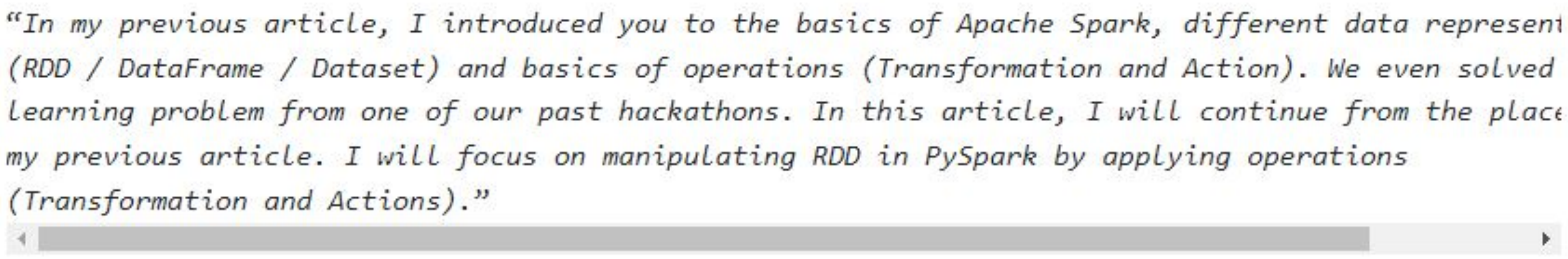
- Example 1:
  - Read file and load data Using Pandas library in python:



```
import pandas as pd
df = pd.read_excel("/home/Loan_Prediction/train.xlsx", sheetname = "Invoice")
```

Dealing with Datasets

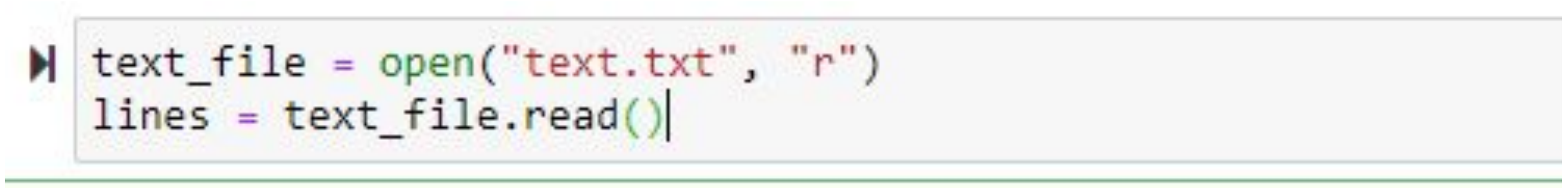
- Example 2:
  - Plain Text (txt) file everything is written in plain text (Usually, this text is in unstructured form and there is no meta-data associated with it).



“In my previous article, I introduced you to the basics of Apache Spark, different data represent
(RDD / DataFrame / Dataset) and basics of operations (Transformation and Action). We even solved
learning problem from one of our past hackathons. In this article, I will continue from the place
my previous article. I will focus on manipulating RDD in PySpark by applying operations
(Transformation and Actions).”

Dealing with Datasets

- Example 2:
  - Read file



```
text_file = open("text.txt", "r")
lines = text_file.read()
```

NumPy Library

- NumPy stands for Numerical Python.
- It is mainly a Python library used for working with arrays.

- Why use NumPy?
  - It provides an array object that is up to 50x faster than traditional Python lists.
  - The array object in NumPy is called ndarray.
  - Arrays are very frequently used in data science, where speed and resources are very important.

NumPy Library

- Installation of NumPy:
  - If you have Python and PIP already installed on system, then install it using this command:
    - C:\Users\your name> pip install numpy
  - Note: Anaconda python distribution has already NumPy installed.

- Example on importing and using NumPy:

```
import numpy OR import numpy as np

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```

NumPy Library

- Example on 3D arrays:
  - Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

  - What will we get from this line?

```
print(arr.ndim)
```

NumPy Library

- Array Indexing:

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

  - Access the element on the first row, second column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

NumPy Library

- Array Indexing:
  - Access the third element of the second array of the first array:

```
    import numpy as np
    arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], 
[10, 11, 12]]])
     print(arr[0, 1, 2])
```

- Negative Indexing:
  - Print the last element from the 2nd dim:

```
    import numpy as np
    arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
    print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Library

- Slicing Arrays:
  - Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

- Negative Slicing:
  - Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

NumPy Library

- Slicing Arrays:
  - Return every other element from the entire array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

  - From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

NumPy Library

- Copy VS View

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
------------------------------------------------------------------------
```

```
import numpy as np

arr2 = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

NumPy Library

- Array Shape:
  - Print the shape of a 2D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

  - Convert the following 1-D array with 12 elements into a 2-D array. The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

NumPy Library

- Array Search:
  - Find the indexes where the values are 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
```

  - Find the indexes where the values are even:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

NumPy Library

- Creating the filter array:
  - Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

Thanks ☺️

Any Questions?