# PYTHON LAB 2023 - PART B JOURNAL WRTIEUP

## Assign B1 - basic regular expression

- A **regular expression** is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- Python has a built-in module called re which provides support to regular expressions.
- To specify patterns in a regular expression, metacharacters are used.
- Regular expressions are used in input validations, search engines, text processing etc.

**Metacharacters**

➢ They are characters which are interpreted in a special way by regular expression.

| Character | Description | Example |
|---|---|---|
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "world$" |
| * | Zero or more occurrences | "aix*" |
| + | One or more occurrences | "aix+" |
| ? | Zero or one occurence | “aix?” |
| {} | Exactly the specified number of occurrences | "al{2}" |
| \| | Either or | "falls\|stays" |
| () | Capture and group | |

**Special Sequence**

- A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

| Character | Description |
|---|---|
| \b | Returns a match where the specified characters are at the beginning or at the end of a word |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word |
| \d | Returns a match where string contains digits (numbers from 0-9) |
| \D | Returns a match where the string DOES NOT contain digits |
| \s | Returns a match where the string contains a white space character |
| \S | Returns a match where the string DOES NOT contain a white space character |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character) |
| \W | Returns a match where the string DOES NOT contain any word characters |

## Assign B2 – Advanced regular expressions

**Types of Regular Expressions:**

These are the functions available in re module

1) The findall() method → It returns a list containing a list of all matches of a pattern within the string. It returns the patterns in the order they are found. If there are no matches, then an empty list is returned.
   example:
   findall(pattern, string)

2) The search() → It searches for first occurrence of pattern within string with optional flags and stops when it finds it.
   search(pattern, string, flags=0)

3) The sub() method → It replaces all occurrences of the pattern in string with replace, substituting all occurrences unless max provided.
   sub(pattern, repl, string, max=0)

4) The split() method → It returns a list where the string at each match.
   split(pattern, string, position)

5) The match() method → It attempts to match the pattern to string with optional flag. The match() returns match object on success and None on failure.
   match(pattern, string, flags=0)

## Assign B3 - Lists

- **List** is a collection (set of values) which is ordered and changeable and allows duplicate members.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

- Lists are constructed using square brackets [ ] wherein you can include a list of items separated by commas.
  - example: shopping_list = [“biscuits”, “bread”, “fruits”, “tea” ]

| Method | Description | Syntax |
|---|---|---|
| append() | Adds an element at the end of the list | list.append(item) |
| clear() | Removes all the elements from the list | *list*.clear() |
| copy() | Returns a copy of the list | x = list.copy() |
| count() | Returns the number of elements with the specified value | list.count(item) |
| extend() | Add the elements of a list 2, to the end of the current list | list.extend(list2) |
| index() | Returns the index of the first element with the specified value | list.index(item) |
| insert() | Adds an element at the specified position | list.insert(index, item) |
| pop() | Removes the element at the specified position | list.pop([index]) |
| remove() | Removes the first item with the specified value | list.remove(item) |
| reverse() | Reverses the order of the list | list.reverse() |
| sort() | Sorts the list | list.sort() |

## Assign B4 – Dictionaries

A dictionary is a collection of an ordered, mutable set of key : value pairs, wherein the keys are unique and hence don't allow duplicates.

Dictionaries are constructed using curly braces { }, wherein you include a list of key : value pairs separated by commas.

- Syntax to create dictionary:

```
dictionary_name = {  key_1 : value_1 , key_2:value_2   ,
                          key_3:value_3  , ……., key_n:value_n  }
```

Example:

```
fish = {"g": "goldfish",  "s":"shark",    "n": "needlefish", "b":"barramundi",
        "m":"mackerel“  }
print(fish)
```

| 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 |

## Assign B5 – SQLite database and operations

SQLite is a self-contained, file-based SQL database. SQLite comes bundled with Python and can be used in any of your Python applications without having to install any additional software.

**Steps to connect to SQLite database:**

1) open and create a database connection using sqlite3.connect() . This returns an Connection object con that represents the connection to the on-disk database.

```
import sqlite3
con = sqlite3.connect("tutorial.db")
```

2) Create a database cursor using con.cursor()

```
cur = con.cursor()
```

3) Now , execute SQL statements and fetch results from SQL queries using cur.execute()

```
cur.execute("CREATE TABLE movie(title, year, score)")
cur.execute("""  INSERT INTO movie VALUES
      ('Monty Python', 1975, 8.2), ('Somethin Different', 1971, 7.5)
        """)
```

4) Now, Call con.commit() on the connection object to commit ( save) the transaction:

```
con.commit()
```

5) we can use res.fetchall() to return all resulting rows. This returns a list of tuples.

```
res = cur.execute("SELECT * FROM movie")
res.fetchall()
```

**Placeholders ? in insert query:** placeholders are used to bind data to the query. Always use placeholders instead of string formatting to bind Python values to SQL statements

```
cur.executemany("INSERT INTO movie VALUES(?, ?, ?)", data)
```

## Assign B6 – GUI using Tkinter module

Tkinter is Python's standard GUI (Graphical User Interface) package. It is the most commonly used toolkit for GUI Programming in Python.

- To create a tkinter the following steps need to be followed:
  1. Importing the module – tkinter
  2. Create the main window (container) using Tk() and mainloop() methods
  3. Add any number of widgets to the main window - button, label, entry etc.
  4. Apply the event Trigger on the widgets.

- Using layout management provided by Tkinter , we can also organize the widgets in the parent windows. There are mainly three geometry manager classes namely pack(), grid() and place() .
- Tkinter provides a mechanism to let the programmer deal with events. For each widget, it's possible to bind Python functions and methods to an event.

  Syntax – widget.bind(event, handler)

  If the defined event occurs in the widget, the "handler" function is called with an event object describing the event.
- The Canvas widget supplies graphics facilities for Tkinter. Among these graphical objects are lines, circles, images, and even other widgets. Methods like create_line(), create_oval(), create_rectangle are used to draw these shapes.

Creating Widgets:

**Label**

- It refers to the display box where you can put any text or image which can be updated any time as per the code. The general syntax is:

  w=Label(master, option=value) master is used to represent the parent window.

  Example: w = Label(root, text='First Name')

**Button**

- To add a button in your application, this widget is used. The general syntax is:

  w=Button(master, option=value) master is used to represent the parent window.

  Example: b = Button(root, text="Submit")

**Entry**

- It is used to input the single line text entry from the user. For multi-line text input, Text widget is used. The general syntax is:

  w = Entry(master, option=value) master is used to represent parent window.

```
root = Tk()
Label(root, text='First Name').grid(row=0)
e1 = Entry(root)
```

## Assign B7 – Exception Handling

- **Runtime errors or exceptions** are produced by the runtime system if something goes wrong while the program is running.
- **An exception is an unwanted event that interrupts the normal flow of the program.**
- **Exception handling** is mechanism provided to handle the run-time errors or exceptions which often disrupt the normal flow of execution.
- Most common types of exceptions are: **TypeError, IndexError, NameError and ValueError.**

- There are 4 keywords used for **exception handling mechanism .** They are try, except, raise and finally.
  - **Try block:** If you feel that block of code is going to cause an exception than such code you need to embed in a try block.
  - **Except block** : Every try block should be followed by one or more except block(s). It is the place, where we catch the exception.
  - **Raise:** In some cases, we need to manually raise an exception, its done through raise keyword followed by the exception name.
  - **Finally:** The finally block will include house-keeping task or cleanup job. Its optional block, which always get executed.
  - **Try-else block:** if the exception has not occurred or raised then the control will be redirected to else block after try block.

## Assign B8 – Line chart and Bar chart

Data visualization is a field in data analysis that deals with visual representation of data. It graphically plots data, giving a visual summary and is an effective way to communicate inferences from data. Python provides various libraries like Matplotlib ,Seaborn, Bokeh,Plotly that come with different features for visualizing data.

Matplotlib consists of inbuilt modules for plotting different graphs like line charts, bar graphs, etc. It mainly works with datasets, frames and arrays (mainly Numpy).

Pyplot is a submodule of MatPlotLib. Pyplot consists of various functions to plot graphs or creates figure, create plotting area in a figure, plot some lines in a plotting area, decorate the plot with labels, etc.

**Plot () Function for Line chart:** The plot() function is used to draw points (markers) in a diagram or it draws a line chart from point to point. This function accepts parameters as lists or arrays that helps us to set axes (x and y) scales and format the graphs.

```
x =[1, 2, ,3,4]  y= [10,20,30,40]
plt.plot(x, y,  color='green', marker='o', linestyle='dashed',
                linewidth=2, markersize=12)
plot.xlabel('X-axis')     #names x-axis
plot.ylabel('Y-axis')     #names y-axis
```

**Bar() Function for Bar chart :** The bar() function draws a bar chart and takes arguments that describes the layout of the bars. The categories and their values represented by the *first* and *second* argument as arrays or lists.

```
plt.bar(sem, per, color=colors)   # plot a vertical bar chart
plt.barh(sem, per, color=colors) # plot a horizontal bar chart
```

Formatting graphs:
Markers are used to emphasize each point with a specified marker . Example: 0, *, X, p , D, H, >, < etc
Markersize and markerendcolor (mec) can also be specified. Linestyle can be specified as dashed, dotted or dotteddash

## Assign B9 – Histogram and Pie chart

Plotting Histogram:

A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency.

The hist() function is used to compute and create a histogram.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument. The bins argument represents the interval or range of values.

Example:

```
plt.hist(num_observations, bins=[20,30, 40, 50, 60, 70, 80, 90, 99],color='green')
```

Plotting Pie Chart:

The pie(array or list) function to draw pie charts. It draws one piece (called as wedge) for each value in the array. By default the plotting of the first wedge starts from the x-axis and moves counterclockwise.

Labels can be added to pie chart using the label parameter and it must be an array with one label for each wedge.

To add a list of explanation for each wedge, we can use the legend() function.

Example:

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```

## Assign B10 – Arrays using Numpy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python.

The array object in NumPy is called ndarray. Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

We can create a NumPy ndarray object by using the array() function. We can pass a list , tuple or object to this array function as parameter.
Example:

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

We can access an array element by referring to its index number.

```
print( arr[0] )
print( arr[0,1])
```

**Builtin methods on arrays**

1) concatenate(arr1, arr2) → joins or puts the contents of two arrays into one
2) array_split( array, no of splits) → splits the array into specified number of parts
3) where(value) → search an array for a certain value, and return the indexes that get a match.
4) searchsorted()→ performs a binary search in the array, and returns the index where the specified value would be inserted.
5) sort(arr) → sorts the array and returns a sorted copy of the array.

## Assign B11 – Operations on Excel using Pandas

Pandas is a Python library used for working with data sets which are mainly relational or labelled. It has functions for analyzing, cleaning, exploring, and manipulating data.It is mainly used for data science/data analysis and machine learning tasks

Pandas is used to load the Excel file and perform data manipulation. Also we can export our results back from pandas to excel using OpenPyXl library.

Pandas is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

**DataFrame:**

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

For example, we can open an excel sheet using read_excel() method of pandas and store the data into the dataframe.

Syntax to create dataframe:

```
df = pd.read_excel('Example.xlsx')
print(df)
```

Pandas use the loc attribute to return one or more specified row(s)

example: print(df.loc[1])

Builtin Methods to work with dataframes

1) head() → display the first few rows of our DataFrame. If no argument is passed, it will display first five rows.
2) tail() → display the last few rows of our DataFrame. If no argument is passed, it will display last five rows.
3) shape() → displays the number of rows and columns in the data frame
4) sort_values() → it sorts any column that consists of numerical data
5) min () and max() → displays minimum and maximum values from the specified column in dataframe