Date
Page

UNIT - 3

→ ERRORS

1. What is error? Explain the classifications of error with an example.

— Error is a abnormal condition whenever it happens execution of the program is stopped. Computers can encounter either software or hardware errors.

→ Errors are mainly classified into following types

i) Syntax error
ii) Semantic error
iii) Run time error
iv) Logical error

i) Syntax error:

It occurs when rules of a programming language are misused. i e the grammatical rules of the language are violated.

Eg:
```
def main( ) :        → missing
    a = 3
    b = 5
    print ("sum is ", a + b )   missing ← error
```
Errors → colon missing, bracket missing.

ii) Semantic error:

It occurs when statements are not meaningful

Eg: 1) Rama plays guitar - both syntactically and semantica -lly correct - has meaning

2) Guitar plays Rama - syntactically correct but semantically wrong as it does not have proper meaning.

iii) Runtime error:

It occurs during the execution of program as it has some illegal operation taking place.

Eg: 1) If a program is trying to open a file but that file doesn't exist it results in a run time error.

2) An expression a number is dividing with zero it results in run time error.

iv) Logical error:

It causes a program to produce a undesired or incorrect

Eg:
```
x = 3
y = 5
average = x + y
print (average)
```

Here we need the average but we won't get the required answer we have not divided x+y by 2 to get the average value resulting in a logical error.

2. What is an exception?

- If a statement or expression is syntactically correct but it may cause an error while getting executing.
- Error detected during execution are 'exceptions'

→ Handling Exceptions:

→ try statement : (syntax)

try:
    # code that may cause exception
except:
    # code to run when exception occurs

→ Eg:
try:
    x = 10/0
except ZeroDivisionError:
    print("You can't divide by zero!")

- Try block contains code that might raise exceptions
- Except block handles specific exceptions.
- Multiple except blocks allow for different exception handling
- Unhandled exceptions propogate outward or terminate execution.
- Try block execution completes if no exceptions occur and except block is skipped if exception is handled

→ Except clause with no exceptions :-

- try block contains code that might raise exceptions
- except block without specifying exceptions will catch all exceptions.
- else block executes if no exception occurs in the try block.

syntax - Except block

```
try:
    # Code that might raise exception
except ExceptionType:
    # Code to handle the exception
```

Eg: For try and except

```
while True:
    try:
        x = int(input("Please enter a number"))
        break
    except ValueError:
        print("Oops! That was no valid number. Try again")
```

- Some common exceptions:

1. SyntaxError: Code syntax is incorrect
2. TypeError: Wrong data type used in operation
3. NameError: Variable or function not defined
4. IndexError: Index out of range.
5. KeyError: Dictionary key not found
6. ValueError: Invalid input or argument
7. AttributeError: Attribute or method doesn't exist.
8. IOError: File read / write error

9. ZeroDivisionError: Dividing by zero
10. ImportError: Module not found.

Eg:
```
def divide(a, b):
    try:
        res = a // b
        print("Answer: ", res)
    except ZerodivisionError:
        print("Error dividing it by zero")
divide(3, 2)
```

Output

1

→ raise statement

- You can raise a exception in the program code, when raised exception the current code breaks the execution and returns the exception back until it is handled.

Syntax:

raise [expression 1], [Expression 2]

Eg:
```
a = "hii"
if not type(a) is int:
    raise typeError("Only integers are allowed")
```

→ To handle multiple errors, what to do?

- Use multiple except blocks

Eg:
```
try:
    num = int("hello") # This will cause a ValueError
except ValueError:
    print("That's not a number!")
except TypeError:
    print("Wrong type of data")
```

- output: That's not a number

→ The Else clause:

- An else block can be added on the try-except block and it should always be present after all except blocks.
- If the try block successfully does not raise an exception then the code enters into the else block.

Eg:
```
try:
    num = int("100")
except ValueError:
    print("Invalid input!")
else:
    print("Valid number:", num)
```

output: Valid number: 100

syntax
```
try:
# some code to check errors
except:
# if an error in the try block then this block runs
else:
# executes this block if there is no exception in try block
```

→ Finally Keyword:

- We use finally keyword, which is executed always even if the exception is not handled and try-except block is terminated.
- It is executed after try-except block.
- Runs no matter what, useful for cleanup tasks.

Syntax:

```
try:
    # some code to check for errors
except:
    # if an error in try block then execute this block
else:
    # if no exception then execute this
finally:
    # this code always executed after try-except block
```

Eg:

```
try:
    f = open("test.txt", "r")
except FileNotFoundError:
    print("File not found!")
finally:
    print("Execution completed")
```

Output → If file found - Execution completed
→ If file is missing - File not found! then Execution completed

```
try:
    code to execute
except:
    code to execute in case of error
else:
    code to execute in case of no error
finally:
    code to execute in all cases
```

→ Pandas data Visualization using Matplotlib

1. What are pandas? or Write a note on Pandas.

- PANDAS IS A PYTHON LIBRARY - used for data manipulation and analysis.
- PROVIDES TWO MAIN DATA STRUCTURES - Series (1D), DataFrame (2D)
- DESIGNED FOR STRUCTURED DATA - like CSV, Excel, and SQL databases.
- SUPPORTS DATA CLEANING - handling missing values, filtering and aggregation.
- ALLOWS EASY DATA TRANSFORMATION - like sorting, merging, and grouping.
- WORKS WELL WITH NUMPY and MATPLOTLIB for efficient data analysis and Visualization.

2. Panda Series:

A panda series is one-dimensional labelled array.

To create a series:

- Use pd.Series() function
- Pass a list of values
- Auto indexing assigns default integer indexes.

Eg:

```
import pandas as pd
data = pd.Series([10, 20, 30, 40])
print(data)
```

Output

```
0  10
1  20
2  30
3  40
dtype: int64
```

3. Panda Data Frame:

A pandas DataFrame is a two dimensional labelled data structure. To create a data frame:

- Use pd.DataFrame() function
- Pass a dictionary with column names as keys and lists of values.
- Data is organized in a tabular format with automatic indexing.

Eg: import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)

Output

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

3. Reading and Previewing a CSV file:

- Use pd.read_csv() to load a CSV file into a DataFrame
- df.head() displays the first 5 rows of the DataFrame
- Helps in understanding the dataset, structure, columns and intial values.

Eg: import pandas as pd
df = pd.read_csv('data.csv')
print(df.head())

4. Data Cleaning and Preprocessing:

- Handling missing data: Replace missing values with suitable values or remove rows / columns with missing data.
- Filtering data: Select specific rows or columns based on conditions to focus on relavant data.

- Adding new columns: Introduce new variables or features to enhance data analysis and modeling.

Eg: 1) Handling missing values

```
df.fillna(0)
df.dropna()
```

2) Filtering data:

```
df[df['Age'] > 30]
```

3) Adding new columns

```
df['salary'] = [50000, 60000, 70000]
```

5. Write a note on Matplotlib?

- MATPLOTLIB IS A PYTHON LIBRARY used for creating visualizations like graphs and charts.
- SUPPORTS STATIC, ANIMATED, and INTERACTIVE PLOTS for data analysis.
- COMMONLY USED FOR PLOTS include line plot, barcharts, scatter plots, histograms, and pie charts.
- HIGHLY CUSTOMIZABLE - allows setting labels, titles, colors and legends.
- WORKS WELL WITH PANDAS and NUMPY for easy data visualization.
- CAN SAVE PLOTS AS IMAGES in formats like PNG, JPG and PDF.

6. Give a example where Pandas are used with Matplotlib.

- Pandas integrate with Matplotlib for seamless visualization. Its workflow is as follows:

1. Load data using Pandas
2. Perform analysis and transformations
3. Visualize using Matplotlib.

Eg:-

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

#Load data
data = pd.DataFrame({'Year': [2020, 2021, 2022], 'Sales':
                                        [200, 250, 300]})
#Visualize data
data.plot(x='Year', y='sales', kind='line', title=
                                        'Yearly sales')
plt.show()
```

7. How to create a basic line Plot in Matplotlib ?

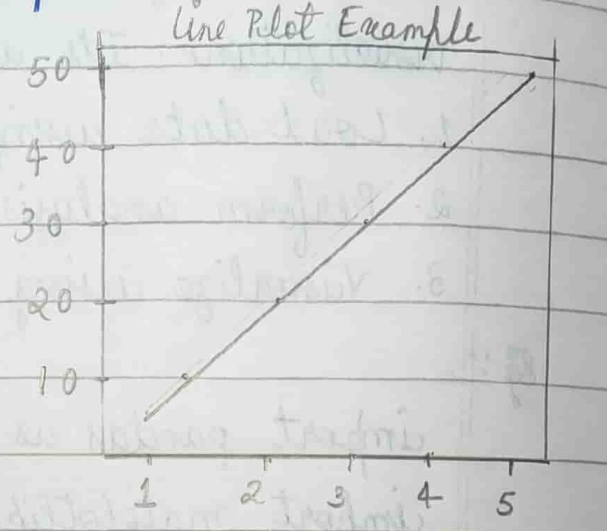
1. Creates a line plot using plt.plot(x, y) where x represents the x-axis values and y represents the Y-Axis values.
2. Adds label and a title using xlabel(), ylabel() and title() to describe the plot.

3. Displays the plot using plt.show()

Eg:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [10, 20, 30, 40, 50]
plt.plot(x, y)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.title('Line Plot Example')
plt.show()
```

output



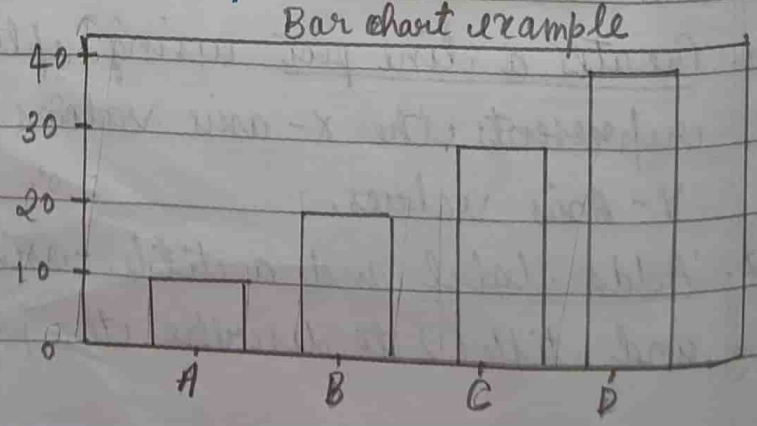
8. How to create a Bar Chart using Matplotlib?

1. Use plt.bar(categories values) to create a bar chart.
2. Customize appearance with options like color and title
3. Display the chart using plt.show()

Eg:

```
import matplotlib.pyplot as plt
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 30, 40]
plt.bar(categories, values, color = 'blue')
plt.title('Bar chart example')
plt.show()
```

→ Output



9. Write about the key benefits of Pandas and Matplotlib.

- PANDAS SIMPLIFIES DATA MANIPULATION by providing efficient tools for handling structured data.
- MATPLOTLIB HELPS VISUALIZE DATA through charts and graphs making patterns easier to understand.
- TOGETHER, THEY ENABLE EXPLORATORY DATA ANALYSIS by allowing users to clean, analyze and visualize data seamlessly.
- SUPPORTS VARIOUS DATA FORMATS like CSV, Excel, and SQL for easy integration.
- HELPS IN DATA-DRIVEN DECISION MAKING by presenting insights clearly through graphs
- WIDELY USED IN DATA SCIENCE AND ANALYTICS for tasks like trend analysis and pattern recognition.

— Shravya N Bhat
ISE-I