J.N.N COLLEGE OF ENGINEERING, SHIVAMOGGA

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

INTRODUCTION TO PYTHON PROGRAMMING LAB BPLCL205B

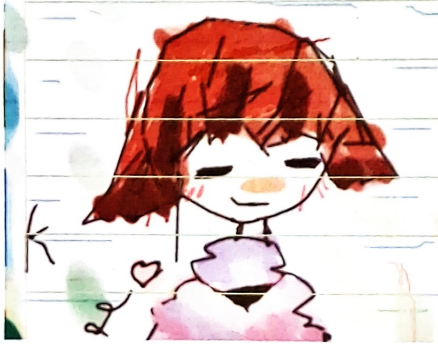
CODE EXPLANATION OF EACH PROGRAM

Prepared By

Aaliya Waseem

Assistant Professor

Department of AIML,JNNCE



For : 2nd sem

INTRODUCTION TO PYTHON PROGRAMMING LAB

BPLCK/205B → Lab Manual [2022-23]

1a] Program Code explanation :-

This program is to read student details [Name, USN, & marks for 3 subjects] and calculates the total marks & percentage.

Step 1 :- Input details of students
- ↳ Name
- ↳ USN

} Give the details such as name, usn

Step 2 :- Input marks for 3 subjects
- ↳ marks 1
- ↳ marks 2
- ↳ marks 3

} Here you read the marks of sub 1, 2 & 3

Step 3 :- Calculate total marks and percentage
- ↳ tot_marks = marks 1 + marks 2 + marks 3
- ↳ per = tot_marks / 3 → average.

Step 4 :- Display the results.
- ↳ Student details
- ↳ Name
- ↳ USN
- ↳ Total marks
- ↳ % → Display % upto 2 decimal places.

Output Example :-

```
Enter the student name : Aaliya
Enter the student USN  : 4JN22SCS01
Enter Marks 1          : 85
Enter Marks 2          : 90
Enter Marks 3          : 95
```

```
Student Details :
Name : Aaliya
USN  : 4JN22SCS01
Total Marks : 270
%    : 90.00 %
```

1.b] Program Code Explanation :-

This program determine whether a person is a senior citizen [aged 60 / older] based on their year of birth.

Step 1 :- Input the Person's Name & Year of birth.

name → stores the user's name as a string

year → stores the user's year of birth as integer

Step 2 :- Get the current year

datetime.date.today() : ↳ current date is fetched using this

cyear = current year

Step 3 :- Calculate the age.

The person's age is calculated as `age = cyear - year`

Step 4 :- Check if the person is a senior citizen or not

↳ If age > 60 ∴ Et Senior Citizen

↳ If age < 60 ∴ Not a Senior Citizen.

Output Example :-

1. Input

Name : Aaliya

Year of birth : 1960

Current year : 2025

2. Calculation :

Age = 2025 - 1960 = 65

3. Output

Aaliya is a senior citizen.

2a] Program Code Explanation :-

Fibonacci Sequence :- is a series of no's where each no [after the first two) is the sum of the two preceding ones.

Eg:- 1, 1, 2, 3, 5, 8, 13, 21, ...

2 3

1+1 = 2
1+2 = 3
2+3 = 5
3+5 = 8 ....

The goal of the program is to generate the Fibonacci sequence of length 'N', where 'N' is provided by the user.

Step 1 :- Input the value of N.

Step 2 : Initialize the first two terms.

↳ fib1 = 1
↳ fib2 = 1
} bcz 1st two terms of fibonacci sequence are always 1.

Step 3 : Print the first two terms.

↳ print(fib1)
↳ print(fib2)
} The first two terms of sequence are directly printed.

Step 4 : Generate the Remaining terms.

↳ fib3 = fib1 + fib2. } fib3 stores the sum of fib1 & fib2.

↳ later, the value of fib3 is printed

↳ fib1 = fib2
fib2 = fib3
} these are updated to move to the next terms in the sequence.

Output Eg :-

Enter the value of N 5

⇒ 1 1 2 3 5

1+1 = 2
1+2 = 3
2+3 = 5

2b Program Explanation :-

Factorial of a number 'n' [n!] is the product of all +ve integers from 1 to n.

Eg :- $4! = 4 \times 3 \times 2 \times 1 = 24$

$0! = 1$

Binomial Co-efficient : It denoted as $\binom{n}{k}$, where k items from 'n' items should be chosed w/o considering the order. It is calculated as:

$$\binom{n}{k} = \frac{n!}{(n-k)!\ k!}$$

Eg : $n = 5$, $k = 2$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$2! = 2 \times 1 = 2$

$(5-2)! = 3! = 3 \times 2 \times 1 = 6$

$$= \frac{5!}{3! \times 2!}$$

$$= \frac{120}{2 \times 6} = 10 //$$

Step 1 :- Factorial function

↳ Base case : If $n = 0/1$, it directly returns 1

↳ Recursive Case : For other values, the function calls itself with $(n-1)$ & multiplies 'n' by the result.

Eg : $n = 3$

↳ $3! = 3 \times 2 \times 1 = 6$

The function which calls itself again & again is recursive where it breaks down into smaller subproblem until it reaches a base case.

Step 2 :- Binomial Co-efficient

Step 3 :- Input / Output

3rd Program Code Explanation :-

This program calculates the mean, variance and standard deviation of a set of numbers.

1. Mean [average] :- The sum of no's divided by the count of no's

Eg: 5, 10, 15

$\Rightarrow 5+10+15 = 30$

$$\text{Mean} = \frac{\text{Sum}}{\text{Total no}} \Rightarrow \frac{30}{3} = 10 //$$

2. Variance :- A measure of how much the no's in the dataset differ from the mean.

Eg:- $\text{Mean} = \frac{5+10+15}{3} = 10$

For 5 : $(5-10)^2 = (-5)^2 = 25 //$

For 10 : $(10-10)^2 = (0)^2 = 0 //$

For 15 : $(15-10)^2 = (5)^2 = 25 //$

→ Sum up the squared d/fces

$25+0+25 = 50$

$$\text{Variance} = \frac{\text{Sum of squared d/fces}}{\text{Count of no's}} = \frac{50}{3} \approx 16.67 //$$

3. Standard Deviation (stdev) :- The square root of the variance, showing how spread out the no's are.

$$\text{Stdev} = \sqrt{\text{variance}} = \sqrt{16.67} \approx 4.08$$

Step 1 :- Input the count of numbers
↳ eg: n = 3

Step 2 :- Create a list
↳ l = list()
} This initializes an empty list to store the no's

Step 3 : Input the numbers

```
↳ for i in range(n):
      num = int(input())
      l.append(num)
```

A loop runs 'n' times, asking the user to input no's one by one, which are then stored in the list l.

Step 4 : Calculate the mean.

```
mean = sum(l) / len(l)
```

↳ sum(l) = adds all the no's in the list.
↳ len(l) = gives the total count of no's.
↳ Their division gives the mean.

Step 5 : Calculate the variance.

```
sm = 0
for i in range(n):
    diff = (l[i] - mean)**2
    sm = sm + diff
variance = sm / len(l)
```

→ calculates the mean then sq (if any -ves are there)
eg $(-2)^2 = 4$ //

Step 6 : Calculate the std deviation. → `stdev = sqrt(variance`

Step 7 : Output the result → `print(mean, variance, stdev)`

4th Program Code Explanation :-

The program explains

↳ Take a multi-digit no (as a string) from the user.

↳ Count how many times each digit (0-9) appears in the no.

↳ Display the fq of each digit.

Step 1 : Input

↳ input('Enter the number')

Step 2 : Initialize Frequency List

↳ digits = list() creates an empty list named digits that will hold the count of each digit (0-9)

Step 3 : Populate the list

The for i in range(10): ,

digits.append(0) , loop adds 10 zeros to the digits. Each position in this list will represent the fq of digits 0 to 9, withe the first position for '0', the second for '1' & so on.

Step 4 : Count digits

↳ The second loop for i in num : goes through each character (or digit) in the no that was i/p.

`digits[int(i)] = digits[int(i)] + 1` → converts the character i to an integer (int(i)) & ncs the count for that gt digit in the digits list.

Output Example :-

If the user 123211 , the output would look like

[0, 3, 2, 1, 0, 0, 0, 0, 0, 0]
0 1 2 3 4 5 6 7 8 9

The digit '1' appears 3 times.
The digit '2' appears 2 times
The digit '3' appears 1 time

Digits 0, 4, 5, 6, 7, 8 & 9 don't appear.

5th Program Code Explanation :-

This program is designed to find the 10 most frequently occuring words in the text file.

Step 1 : You need to create a text file "data.txt" & store the text in it

Eg' all the glitters is not gold
all is well
This is a well written story
all are fruited

Step 2 : Creating an Empty Dictionary

```
d= dict()
```

Step 3 : Reading the file Line by Line

```
for line in lines :
```

} The for loop iterates over each line in the file.

Step 4: Processing Each Line.

```
words = line.strip().split(" ")
```

line.strip() → removes any leading / trailing whitespaces / newline characters.

split(" ") → splits the line into individual words using space as a separator.

Step 5: Counting Word Frequencies

```
for word in words:
    d[word] = d.get(word, 0)+1
```

retrieves current count of the word from the dictionary (if the word doesn't exist, it returns 0)

+1, ↑ces the count by 1 for each occurance of the word.

Step 6: Sorting the dictionary by Frequency.

```
srtd = dict(sorted(d.item(), key: operator.items(), key =operato
 .itemgetter(1), reverse = True))
```

d.items() returns a list of tuples (word, fq)

operator.itemgetter(1) sorts based on the fq (the second element in each tuple)

reverse = True. sorts the dictionary in descending order (most frequent first)

Step 7: Displaying the Top 10 Most Frequent Words.

```
print(dict(list(srtd.item())[:10]))
```

srtd.items() → converts the sorted dictionary back into a list of tuples.

[:10] → slices the list to get the first 10 items (top most frequent words)

dict() → convert this list of tuples back into a dictionary

print() → displays the top 10 words & their frequencies.

6th Program Code Explanation :-

This program reads the contents of a text file, sorts the line alphabetically & writes the sorted lines into a new text file.

Step 1 :- Opening the Files :

`fd = open("alp.txt")` → This line open the file alp.txt in read mode. This is where we'll get the data.

`wfd = open("sorted.txt", "w")` → This line opens a new file sorted.txt in write mode. This is where we'll write the sorted lines.

Step 2 :- Reading the Contents :

`lines = fd.readlines()` → This reads all the lines from the alp.txt file & stores them in a list called lines.

Step 3 :- Printing the Original List :

print(lines) → This line prints the list of lines from the file so we can see what the data looks like before sorting.

Step 4 :- Sorting the Lines :

lines.sort() → this method sorts the lines list in alphabetical order. After this step, the lines will be rearranged alphabetically.

Step 5 :- Printing the Sorted List :

print(lines) → This prints the sorted list, so you can see the result of sorting.

Step 6 :- Writing the Sorted Lines to a New File :

for line in lines → This loop goes through each line in the sorted list

wfd.write(line) → For each line in the sorted list, this writes the line into sorted.txt file

Step 7 :- Closing the Files :

fd.close() → this closes the input file (alp.txt) to free up resources.

wfd.close() → This closes the o/p file (sorted.txt) to save the changes.

Summary :-

↳ Open & reads the content of a file alp.txt
↳ sorts the lines alphabetically.
↳ Write the sorted lines into another file sorted.txt

7th Program Code Explanation :-

This Python code snippet demonstrates how to create a zip file containing files & subfolders from a specific directory using the zipfile & os modules.

Step 1 :- Importing Modules :

↳ from zipfile import ZipFile
import os

- ZipFile → helps to create, read or extract zip files
- os → used for interacting with the file system (eg. navigating directories)

Step 2 :- Creating a Zip File :

```
with ZipFile ("backup.zip", "w") as zipob:
```

- Opens a new zip file named backup.zip in write mode ("w")
- Ensures the file is automatically closed after the block ends.

Step 3 :- Walking Through a Directory -

```
for folders, subfolders, files in os.walk ('programs'):
```

- folders → The current folder's path
- subfolders → A list of subfolder names inside the current folder
- files → A list of files names in the current folder
- os.walk ('programs') → Iterates through the programs directory

Step 4 :- Adding Files to the zip:

```
for file in files:
    file_path = os.path.join(folders, file)
    zipob.write(file_path)
```

os.path.join(folders, file) → combines the folder path & file name to get the full path of each file.

zipob.write(file_path) → Adds the file to the zip archieve.

Step 5 :- Checking if the zip File Was Created.

```
if os.path.exists("backup.zip")
    print("zip file created")
else:
    print("Error in creation")
```

os.path.exists("backup.zip")

↳ Checks if the file backup.zip exists

↳ Prints a success message if the zip file is created otherwise, prints an error message.

Logical Flow →

Import the required modules

↓

Open/ Create a zip file named backup.zip in write mode

↓

Traverse the target directory (programs) & gather files / sub folders

↓

Add each file to the zip archieve

↓

Check if the zip file was succesfully created.

8th Program Code Explanation :-

This program calculates the division of two numbers a & b using a function called DivExp.

↳ The numerator a must be greater than 0. If not, the program will stop & show an error.

↳ The denominator (b) can't be zero becz dividing by zero is mathematically invalid. If b is zero, the program will show an error message.

Division Logic :-

If all checks pass, the division is performed using $c = a / b$ and the result is returned.

Case (I) :- Normal Division

Enter the numerator (a): 10
Enter the denominator (b): 2

10/2 = 5.0

Output: The result of division is : 5.0

Case (II) : Numerator is less than or equal to 0

Enter the numerator (a) : -5
Enter the denominator (b) : 3

Output : Error → the numerator must be > than 0

Case (III) : Denominator is zero :-

Enter the numerator (a) : 8
Enter the denominator (b) : 0

Output : Denominator can't be zero.

9th Program Code Explanation :-

This program defines a complex class for working with complex no's (numbers with real & imaginary parts)

Step 1 : Class Definition & Constructor (__init__ method):

```
class Complex:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag
```

this method is special method in python used to initialize new objects of this class

Step 2 : String Representation (__str__ method):

```
def __str__(self):
    return str(self.real) + "+i" + str(self.imag)
```

This method defines how a complex object will look when converted to a string (eg, for printing)

Example : If real = 3 & imag = 4, it will return "3+i4"

Step 3 : Addition of Complex Numbers (add function).

```
def add(c1, c2):          → complex objects c1 & c2
    return Complex(c1.real + c2.real, c1.imag
                   +c2.imag)
```

real part

imaginary part

Step 4 : Input and Logic

```
N = int(input("Enter N: "))
l = []  → empty list to store complex objects
          ↳ no of complex no's to i/p
for i in range(N):
    real = int(input('Enter real : '))
    imag = int(input('Enter imag : '))
    l.append(complex(real, imag))
```

Step 5 : Adding all Complex Numbers :

```
result = [0]
for i in range(1,N):
    result = add(result, l[i])
print(result)
```

↳ Finally prints the sum of all complex no's using the _str_ method

OUTPUT :-

```
Enter N : 2
Enter real : 3
Enter imag : 4
Enter real : 1
Enter imag : 2
```

Real part : 3+1 = 4

Imaginary part : 4+2 = 6

Output :- 4 + i6

Process :-

- Two complex no's are created
  - 3+i4
  - 1+i2
- They are added

10th Program Code Explanation :

This code defines a Student class to represent student information & their marks. However, the code has a few syntax. &

Step 1 : Class Definition

The class Student represents a student with

↳ name
↳ USN
↳ Marks

Step 2 : Constructor (_init_ method)

```
def _init_ (self, name, USN):
    self.name = name
    self.USN = USN
    self.marks = list()
```

* The constructor initializes the name, USN & marks as an empty list.
* Correct Syntax : self.name = name

(~~dtse~~-

Step 3 : getmarks Method

↳ This method takes input of 3 subject marks.
↳ Calculates the total marks.
↳ Calculates the %
↳ Adds total & % to the marks list.

Step 4 : display Method

```
def display(self):
    print("Marks List:", self.marks)
```

Displays the marks list, including the 3 subject marks total & %.

Step 5 : Object Creation & Method Calls

```
s1 = Student("Arman", "4JN22A1000")
s1.getmarks()
s1.display()
```

A student object s1 is created with name 'Arman' & USN '4JN22A1000'

collects 3 subject marks, calculates total & % & stores them in s1.marks

prints the marks