PROBLEM NO: 01

Python Basic Programs

PROGRAM TITLE:

Working with **python basic syntax, programs, loops, conditions, console input output** etc.

OBJECTIVES:

To know about **python basic programs**.

THEORY:

Python:

A simple **language** which is easier to learn. **Python** has a very simple and elegant syntax.

- Free and open-source.
- Portability.
- Extensible and Embeddable.
- A high-level, interpreted **language**.
- Large standard libraries to solve common tasks.
- Object-oriented.

1.1 print():

The **print()** function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

CODE:

```
1. print('Welcome all to python RE')
```

OUTPUT:

```
Welcome all to python RE
```

In this program, we have used the built-in **print()** function to print the string “Welcome all to Python RE” on our screen.

1.2 input():

The **input()** function reads a line entered on a console by an input device such as a keyboard and convert it into a string and returns it. One can use this input string in **python** code.

CODE:

```
1.  a = input('Enter whatever u want: ')
2.  print(a)
3.  print(type(a))
4.  aa = int(a)
5.  print(type(aa))
```

OUTPUT:

```
Enter whatever u want: 50

50

<class 'str'>

<class 'int'>
```

Whatever we enter as input, input function converting it into a string. if we enter an integer value still **input()** function convert it into a string. We need to explicitly convert it into an integer in our code using typecasting.

1.3 for loop:

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The **Python** for statement iterates over the members of a sequence in order, executing the block each time.

CODE:

```
1.  for i in range(0, 10, 1):
2.      if i % 2 == 0:
3.          print(i, end = ' ' )
```

OUTPUT:

```
0 2 4 6 8
```

The **range()** function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: **range(0,10,1)**, which means values from 0 to 10 (but not including 10).

PROBLEM NO: 02

NumPy Module in Python

PROGRAM TITLE:

Using and analyzing **NumPy** library in **python**.

OBJECTIVES:

To learn about **NumPy** library in **python**.

THEORY:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2.1 Array & Dimensions:

A **NumPy** array is a homogeneous block of data organized in a multidimensional finite grid. All elements of the array share the same data type, also called **dtype** (integer, floating-point number, and so on). The **dimensions** of an **array** can be accessed via the "**shape**" attribute that returns a tuple describing the length of each **dimension**.

CODE:

```
1.  import numpy as np
2.  #numpy
3.  mat1 = np.array( [ (1, 2, 3), (4, 5, 6), (7, 8, 9), (7, 4, 1) ] )
4.  print(mat1)
5.  dimension = mat1.shape
6.  print(dimension[0])
7.  print(dimension[1])
8.  print(dimension)
9.
10. mat_zeros = np.zeros((4, 3))
11. mat_ones = np.ones((3, 4))
12. print(mat_zeros)
13. print(mat_ones)
14.
15. print(type(mat_zeros))
16. print(type(mat_zeros[1][1]))
17.
18. for i in range(0, mat_zeros.shape[0]):
19.     for ii in range(0, mat_zeros.shape[1]):
20.         mat_zeros[i][ii] = i + ii
21.
22. print(mat_zeros)
```

OUTPUT:

```
[[1 2 3]
[4 5 6]
[7 8 9]
[7 4 1]]
4
3
(4, 3)
[[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]]
[[1. 1. 1. 1.]
[1. 1. 1. 1.]
[1. 1. 1. 1.]]
<class 'numpy.ndarray'>
<class 'numpy.float64'>
[[0. 1. 2.]
[1. 2. 3.]
[2. 3. 4.]
[3. 4. 5.]]
```

2.2 NumPy Matrix Multiplication:

numpy.dot(*a*, *b*, *out=None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If either *a* or *b* is 0-D (scalar), it is equivalent to multiply and using numpy.multiply(a, b) or a * b is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where M>=2), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])

CODE:

```
1. import numpy as np
2. #numpy
3. mat1 = np.array( [ (1, 2, 3), (4, 5, 6), (7, 8, 9), (7, 4, 1) ] )
4. print(mat1)
5. dimension1 = mat1.shape
6. print(dimension1)
7. mat2 = np.array( [ (1, 2, 3, 4), (4, 5, 7, 8), (9, 6, 3, 1) ] )
8. print(mat2)
9. dimension2 = mat2.shape
10. print(dimension2)
11. mat3 = np.dot(mat1, mat2)
12. print(mat3)
13. print(mat3.shape)
```

OUTPUT:

```
[[1 2 3]
[4 5 6]
[7 8 9]
[7 4 1]]
(4, 3)
[[1 2 3 4]
[4 5 7 8]
[9 6 3 1]]
(3, 4)
[[ 36 30 26 23]
[ 78 69 65 62]
[120 108 104 101]
[ 32 40 52 61]]
(4, 4)
```

PROBLEM NO: 03

RE Module in Python

PROGRAM TITLE:

Using and analyzing **RE module** in **python**.

OBJECTIVES:

To learn about **RE module** in **python**.

THEORY:

Regular expressions (RE) use the backslash character ('\') to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write '\\\\' as the pattern string, because the regular expression must be \\, and each backslash must be expressed as \\ inside a regular Python string literal. Also, please note that any invalid escape sequences in Python's usage of the backslash in string literals now generate a DeprecationWarning and in the future this will become a SyntaxError. This behaviour will happen even if it is a valid escape sequence for a regular expression.

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with 'r'. So r"\n" is a two-character string containing '\' and 'n', while "\n" is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

3.1 re.compile():

The **re.compile**(pattern, repl, string): We can combine a regular expression pattern into pattern objects, which can be used for pattern matching. It also helps to search a pattern again without rewriting it.

CODE:

```
1.  import re
2.
3.  data = '''''
4.  Hello everyone, I am MrA.In 11/Jan/2020 I met Messi and got his phone number +88017
    28366059.But his manager MrAugust
5.  gave me another number +8801987533108 and requested to enjoy a football match at 20
    /Mar/2021 and I accepted his offer.
6.  '''
7.
8.  date_pattern = re.compile(r'\d+/[A-Z][a-z]+/\d{4}')
9.
10. scraping_dates = date_pattern.findall(data)
11. print(scraping_dates)
12.
13. phone_number_pattern = re.compile(r'\+[8][8][0][1][3|5|6|7|8|9]\d{8}')
14. scraping_numbers = phone_number_pattern.findall(data)
15. print(scraping_numbers)
16.
17. name_pattern = re.compile(r' [A-Z][a-z][A-Z]*[a-z]*')
18. scraping_names = name_pattern.findall(data)
19. print(scraping_names)
```

OUTPUT:

```
['11/Jan/2020', '20/Mar/2021']
['+8801728366059', '+8801987533108']
[' MrA', ' Messi', ' MrAugust']
```

3.2 re.findall():

re.findall() module is used when you want to iterate over the lines of the file, it will return a list of all the matches in a single step.

CODE:

```
#re Personal Assesment
import numpy as np

mat1 = np.array( ['+8801738663624', '+88017205266h20', '+8801720526652', '+8801738663i245', '+8801776604777', '+880177o6047778' ] )
phone_number_pattern = re.compile(r'\+[8][8][0][1][3|5|6|7|8|9]\d{8}')


for i in range(0, mat1.shape[0]):
    if phone_number_pattern.findall(mat1[i]):
        print('Valid number: ' + mat1[i])
    else:
        print('invalid number: ' + mat1[i])
```

OUTPUT:

```
Valid number: +8801738663624

invalid number: +88017205266h20

Valid number: +8801720526652

invalid number: +8801738663i245

Valid number: +8801776604777

invalid number: +880177o6047778
```