4.1 REGULAR EXPRESSIONS:

4.1.1 Concept of regular expression:

Regular expressions can be used to explicitly describe patterns within strings of text. In addition to simply describing such patterns, regular expression engines can typically be used to iterate through matches, to parse strings into substrings using patterns as delimiters, or to replace or reformat text in an intelligent fashion. It consists of two types of characters. Literal are specific character in an expression. And Metacharacters are simple way to specify a pattern.

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module.

Metacharacters	Description
*	It specify 0 or more occurrences of character
+	It specify 1 or more occurrences of characters
?	It specify 0 or 1 occurrence of character
.	It matches any single character except a newline.
|	It is used for alternation, essentially to specify 'this OR that' within a pattern.
[]	It matches one character in a range of valid characters
{m}	The previous character can occur exact m times.
{m,n}	The previous character can occur from m to n times.
^	Depending on whether the MultiLine option is set, it matches the position before the first character in a line, or the first character in the string.
$	Depending on whether the MultiLine option is set, it matches the position after the last character in a line, or the last character in the string.
\w	It matches any alphanumeric character. **[a-zA-Z_0-9]**.
\W	The negation of \w, this equals to **[^a-zA-Z_0-9]**
\s	It matches any white-space character. [\t\n\r\f\v]
\S	It matches any non-white-space character. [^ \t\n\r\f\v]
\d	It matches any decimal digit. Equivalent to **[0-9]**
\D	It matches any non-decimal digit. Equivalent to **[^0-9]**

4.1.2 Various types of regular expressions:

Method	Purpose
match()	It determines if the RE matches at the beginning of the string.
search()	It scans the complete string, looking for any location where this RE matches.
findall()	It finds all substrings where the RE matches, and returns them as a list.
Split	It can be used to split a string into a list of substrings.
Sub	Using sub every match of the regular expression regex in the string subject will be replaced by the string replacement.
Compile	If you want to use the same regular expression more than once in a script, it might be a good idea to use a regular expression object.

Flags for regular expression methods:

syntax	long syntax	Meaning
re.I	re.IGNORECASE	It ignores case while matching.
re.M	re.MULTILINE	It makes begin/end {^, $} consider each line.
re.S	re.DOTALL	It makes dot(.) to match any character. For newline too.
re.U	re.UNICODE	It makes {\w, \W, \b, \B} to follow Unicode rules.
re.L	re.LOCALE	It makes {\w, \W, \b, \B}to follow current locale.
re.X	re.VERBOSE	It ignores whitespace and allows comment in regex.

Match object methods:

Method/Attribute	Purpose
group()	If it is called without argument then it will returns the substring, which had been matched by the complete regular expression.
start()	It returns the string index where the regular expression started matching in the string.
end()	It returns the string index where the regular expression ended matching in the string.
span()	It returns a tuple with the string index where the regular expression started matching in the string and ended matching.
groups()	It returns all matching subgroups in a tuple.

4.1.3 Using match function:

match() : The match() method only find matches if they occur at the start of the string being searched.

In Python a regular expression search is typically written as:

result = re.match(pattern, string[, flags])

Methods of Priority Queue:

q.put(priority, data): It put priority and data in a queue. The queue output shows priority number.

q.get(): It used to get the elements from the queue or it removes the element from the queue.

Example:

Create a priority queue and retrieve its priority.

```
import queue
q= queue.PriorityQueue() # define the queue q
q.put(1, "O grade") #put priority and data
q.put(10 ,"Fail")
q.put(8, "D grade")
q.put(5, "C grade")
q.put(4, "B+ grade")
# show the queue element
print(q.queue)
while not q.empty(): # output from queue using get method
    print("Priority =",q.get())
```

Output:

```
>>>
[1, 4, 8, 10, 5]
Priority = 1
Priority = 4
Priority = 5
Priority = 8
Priority = 10
```

4.4 MODULES:

To write a longer program, it is better to use a file. This file you can use it number of times as well as you can make corrections in it. This is known as creating a script. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use functions that are already present in your file without copying its definition into each program. In Python you can put definitions in a file and use them in a script or in an interactive instance of the interpreter. This file is known as module. Once the script file is ready you can import definitions from one into other modules. Modules provide reusability of code.

Modules refer to a file containing Python statements and definitions. A file containing Python code is called a module.

4.4.1 Importing module:

Python provides different ways to import modules. You have to use the import statement or the from statement, or the built-in __import__ function.

- **import module_name –** It imports the module with given name. It creates a reference to that module in the current namespace.

- **from module_name import *** - It imports the module with given name. It creates references in the current namespace to all public objects defined by that module
- **from module_name import x, y, z** - It imports the module with given name. It creates references in the current namespace to the given objects. You can now use x, y and z objects in your program.
- **module_name = __import__('module_name')**- It works as import module_name. First you have to pass the module name as a string and then explicitly we have to assign it to a variable in current namespace.

Example:

Write a program to display the area of circle.

```
import math
# from math import *
r=int(input("enter radius:"))
print("The area of circle=", math.pi*r*r)
```

Output:

```
>>>
enter radius:5
The area of circle= 78.53981633974483
```

Example:

Write a program to display the area of circle.

```
from math import pi
r=int(input("enter radius:"))
print("The area of circle=", pi*r*r)
```

Output:

```
>>>
enter radius:5
The area of circle= 78.53981633974483
```

Example:

Write a program to display 5 random numbers between 1 to 25 and pi value.

```
import random , math
for i in range(5):
    print(random.randint(1, 25))
print(math.pi)
```

Output:

```
>>>
19
14
12
19
4
3.141592653589793
```

Aliasing Modules: You can modify the names of modules and their functions with new names by using the as keyword. You can make alias because you have already used

the same name for something else in your program or you may want to shorten a longer name.

Syntax:

import module as another_name

Example:

Write a program to display an example of an alias.

```
import math as m
print(m.pi)
print(m.e)
```

Output:

```
>>>
3.141592653589793
2.718281828459045
```

4.4.2 Creating and exploring modules:

You can create your own module as a file which can contain Python definitions and statements. The file name is the module name with the extension .py

To create a module, create a file with Python definitions and statements and save it with any proper name. To include module in a file, use import statement.

Steps to create own module:

(1) First create a regular Python program with the extension as **.py**. This is your module file.

(2) In the same folder create a new Python program that contains the main function.

(3) Add the import module_name to the top of the file to get definitions and statements from the module file .

(4) Python automatically checks to directory for modules as well as the usual standard libraries.

Example:

Create a module to define two functions. One for finding whether the given number is palindrome or not. Other for finding whether the given number is Armstrong or not. And include this module in any other file to check the output.

Step 1: Start a new file and the following code:

```
def palin(x):
    rev=0
    x1=x
    while x>0:
        r=x%10
        x=x//10
        rev=rev*10+r
    if x1==rev:
        print("Palindrome")
    else:
        print("not a Palindrome")
```

```
def Am(x):
        sum=0
        x1=x
        while x>0:
                r=x%10
                x=x//10
                sum=sum+r*r*r
        if x1==sum:
                print("Armstrong")
        else:
                print("not a Armstrong")
```

Step 2: Save it with math_module.py name.

Step 3: Start a new file to include the above module. Add the following code and save it with name eg1.py

```
import math_module as m
def main():
        x=int(input("enter no to check it is palindrome or not="))
        m.palin(x)
        y=int(input("enter no to check it is armstrong or not="))
        m.Am(y)
main()
```

Step 4: Execute the eg1.py file

Output:

```
enter no to check it is palindrome or not=121
Palindrome
enter no to check it is armstrong or not=153
Armstrong
```

4.4.3 Math module:

The math module implements many mathematical functions. They are normally be found in the native platform C libraries for complex mathematical operations using floating point values, including logarithms and trigonometric operations.

Special Constants:

math module includes two Special Constants as π (pi) and e.

Example:

Display the value of Special Constants of math module.

```
import math
print('π=',  math.pi)
print('e=',  math.e)
```

Output:

```
>>>
π= 3.141592653589793
e= 2.718281828459045
```

List of inbuilt math functions: Refer unit 2

Example:

Give an example of math functions.

```
import math

# Absolute value
print("floor output=",math.floor(1000.5))
print("ceil output=",math.ceil(1000.5))

# fsum.
values = [0.9999999, 1, 2, 3]
r = math.fsum(values)
print("fsum output=",r)

# Truncate value
print("trunc output=",math.trunc(123.45))

# power method
print("power output=",math.pow(2, 3))

# Use sqrt method.
print("square root output=",math.sqrt(9))

# Use fabs method.
print("absolute output=",math.fabs(-123.5))

# Use exp method.
print("exp output=",math.exp(2))

# Use degrees method.
print("degrees output=",math.degrees(.50))

# Use radians method.
print("radians output=",math.radians(20))
```

Output:

```
>>>
floor output= 1000
ceil output= 1001
fsum output= 6.9999999
trunc output= 123
power output= 8.0
square root output= 3.0
absolute output= 123.5
exp output= 7.38905609893065
degrees output= 28.64788975654116
radians output= 0.3490658503988659
```

4.4.4 Random module:

Sometimes we want the computer to pick a random number in a given range, pick a random element from a list, pick a random card from a deck, flip a coin, etc. The random module provides access to functions that support these types of operations.

List of inbuilt random functions:

Function	Description
random	It generates a random number in the range (0.0, 1.0)
randint(a, b)	It generate a random integer between a and b inclusive.
randrange(start, stop)	It generates a random number in the range (start, stop).
randrange(stop)	It generates a random number in the range (0, stop).
seed(n)	It initializes the random number generator.
choice(seq)	It returns a random element from the non-empty sequence.
shuffle(seq)	It shuffles the sequence.
uniform(a, b)	It return a random floating point number between a and b inclusive.
sample(population, k)	It returns k length list of unique elements chosen from the population sequence.

Example:

Give an example of random functions.

```
import random

# Use random method.
print("random output=",random.random())

# Use randint method.
print("randint output=",random.randint(1,5))


# Use randrange(start, stop) method.
print("randrange output=",random.randrange(10,15))

# Use randrange(stop) method.
print("randrange output=",random.randrange(5))


 # Use seed method.
 random.seed(100)
 print("Random number with seed 100=", random.random())
```

```
# It will generate same random number
random.seed(100)
print("Random number with seed 100=", random.random())


# Use choice method.
print("choice output ", random.choice([5,10,15]))
print ("choice output ", random.choice('xyz'))

# Use shuffle method.
list1= [1,2,3,5,10]
random.shuffle(list1)
print("shuffle output ", list1)


# Use uniform method.
print("uniform output ", random.uniform(1, 10))


# Use sample method.
print("sample output ", random.sample([1, 2, 3, 4, 5],  2))
```

Output:

```
>>>
random output= 0.845447166702447
randint output= 1
randrange output= 10
randrange output= 3
Random number with seed 100= 0.1456692551041303
Random number with seed 100= 0.1456692551041303
choice output  10
choice output  x
shuffle output  [10, 1, 2, 3, 5]
uniform output  5.7961127317831425
sample output  [1, 4]
```

4.4.5 Time module:

This module provides various time related functions. Most of the functions defined in this module call platform C library functions with the same name.

List of inbuilt random functions:

Function	Description
time()	It returns the number of seconds from the start of the epoch as a floating point value.
ctime()	It returns the time with human readable representation.

clock()	It returns processor clock time with a floating point value.
sleep()	It is used to halt your program for a number of seconds.
gmttime ([secs])	It returns the current time in UTC. Also it converts a time expressed in seconds since the epoch to a struct_time in UTC. If secs is or None, then current time as returned by time() is used.
localtime([secs])	It returns the current time with the current time zone applied. Also it is same as gmtime() but it converts to local time. If secs is None, then current time as returned by time() is used.
mktime(t)	It takes an argument as struct_time and converts it to the floating point representation. It is an inverse function of localtime().
strptime(string[, format])	It parses a string representing a time according to a format and it returns value is a struct_time as returned by gmtime() or localtime().
strftime(format[, t])	It converts a tuple or struct_time representing a time as returned by gmtime() or localtime() to a string as specified by the format argument. If t is not provided, the current time as returned by localtime() is used.

Directives to format string:

Directive	Meaning
%a	It is used to represent Locale's abbreviated weekday name.
%A	It is used to represent Locale's full weekday name.
%b	It is used to represent Locale's abbreviated month name.
%B	It is used to represent Locale's full month name.
%c	It is used to represent Locale's appropriate date and time representation.
%d	It is used to represent Day of the month as a decimal number [01,31].
%H	It is used to represent Hour (24-hour clock) as a decimal number [00,23].
%I	It is used to represent Hour (12-hour clock) as a decimal number [01,12].
%j	It is used to represent Day of the year as a decimal number [001,366].
%m	It is used to represent Month as a decimal number [01,12].
%M	It is used to represent Minute as a decimal number [00,59].
%p	It is used to represent Locale's equivalent of either AM or PM.
%S	It is used to represent Second as a decimal number [00,61].
%U	It is used to represent Week number of the year (**Sunday** as the first day of the week) as a decimal number [00,53].
%w	It is used to represent Weekday as a decimal number [0(Sunday),6].
%W	It is used to represent Week number of the year (**Monday** as the first day of the week) as a decimal number [00,53].

Directive	Meaning
%x	It is used to represent Locale's appropriate date representation.
%X	It is used to represent Locale's appropriate time representation.
%y	It is used to represent Year without century as a decimal number [00,99].
%Y	It is used to represent Year with century as a decimal number.
%z	It is used to represent Time zone offset indicating a positive or negative time difference from UTC/GMT of the form +HHMM or -HHMM.
%Z	It is used to represent Time zone name.

Example:

Give an example of time functions.

```
import time

# Use time method.
print("The time output=",time.time())

# Use ctime method.
print("The ctime output=",time.ctime())

# Use clock method.
print("The clock output=",time.clock())

# Use seek method.
for i in range(1, 3):
    print("The sleep output=",time.ctime())
    time.sleep(i)

# Use gmttime method.
print("The gmttime output=",time.gmtime())
print()
# Use localtime method.
print("The localtime output=",time.localtime())
print()

# Use mktime method.
print("The mktime output=",time.mktime(time.localtime()))
print()

# Use strptime method.
print("The strptime output=",time.strptime("25 Nov 10", "%d %b %y"))
print()

# Use strftime method.
```

```
  print("The strftime output=",time.strftime("%a, %d %b %Y %H:%M:%S",
time.gmtime()))
```

Output:

```
>>>
The time output= 1498750777.964221
The ctime output= Thu Jun 29 21:09:38 2017
The clock output= 1.2800385035581871e-06
The sleep output= Thu Jun 29 21:09:38 2017
The sleep output= Thu Jun 29 21:09:39 2017
The gmttime output= time.struct_time(tm_year=2017, tm_mon=6, tm_mday=29,
tm_hour=15, tm_min=39, tm_sec=41, tm_wday=3, tm_yday=180, tm_isdst=0)

The localtime output= time.struct_time(tm_year=2017, tm_mon=6, tm_mday=29,
tm_hour=21, tm_min=9, tm_sec=41, tm_wday=3, tm_yday=180, tm_isdst=0)

The mktime output= 1498750781.0

The strptime output= time.struct_time(tm_year=2010, tm_mon=11, tm_mday=25,
tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=329, tm_isdst=-1)

The strftime output= Thu, 29 Jun 2017 15:39:41
```

Question Bank for Self-Practice

(1) Explain the Concept of regular expression.
(2) What are the Various types of regular expressions?
(3) Explain match function with example.
(4) Explain search function with example.
(5) Explain findall function with example.
(6) Explain split function with example.
(7) Explain sub function with example.
(8) Explain compile function with example.
(9) What is the difference between search and match?
(10) What are the different Meta characters?
(11) What are the features of Object Oriented Programming?
(12) What is class? How to define it?
(13) What is object? How to create it?
(14) What are the data members of a class?
(15) What is the difference between class variables and instance variable? Give an example.
(16) What is constructor? Give an example.
(17) What is destructor? Give an example.
(18) How to delete objects and attributes?
(19) Give an example of Instances as Arguments.
(20) Write a program to demonstrate Instances as return values.
(21) What are the different Built-in Class Attributes?
(22) What is inheritance? What are their types?

(23) Write a program to implement single inheritance.
(24) Write a program to implement multiple inheritance.
(25) Write a program to implement mutilevel inheritance.
(26) What is method overriding? Give an example.
(27) What is data encapsulation? Give an example.
(28) Write a short note on data hiding.
(29) Write a program to implement data hiding.
(30) Whether method overloading is supported in python?
(31) What is Multithreaded Programming?
(32) What are the advantages of Multithreaded Programming?
(33) What are different modules to implement a thread?
(34) How to create a thread using thread module?
(35) How to create a thread using threading module?
(36) How to Synchronizing thread?
(37) What is Synchronizing a thread?
(38) What the different methods are for thread synchronizing?
(39) What is priority queue?
(40) Write a program to implement priority queue.
(41) What are the different methods of threading module?
(42) What is module? What are its types?
(43) What area the different ways to import module?
(44) How to create our own module?
(45) What is math module? Explain its functions.
(46) What is random module? Explain its functions.
(47) What is time module? Explain its functions.
(48) Write a program to implement math module functions.
(49) Write a program to implement random module functions.
(50) Write a program to implement time module functions.
(51) What are the directives to format time string?