

Assignment - 5

1. Explain the difference between formal and actual parameter with example.

A formal parameter, i.e. a parameter, is in the *function definition*. An actual parameter, i.e. an argument, is in a *function call*.

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

So n here:

```
def factorial(n):
```

Is a formal parameter.

And n - 1 (or rather, the *value it evaluates to*) here:

```
    return n * factorial(n-1)
```

Is an "actual parameter", i.e. an argument.

2. What is Interpreter? Explain How python interpreter works.

An **interpreter** is a kind of program that executes other programs. When you write **Python programs**, it converts source code written by the developer into **intermediate language** which is again translated into the native language / machine language that is executed.

3. What are the differences between python and C language?

C	Python
C is mainly used for hardware related applications.	Python is general purpose programming language.
Follows an imperative programming model.	Follows object-oriented programming language
Pointers available in C.	No pointers functionality available.
C is compiled.	Python is interpreted.
A limited number of built-in functions.	Large library of built-in functions.
Code execution is faster than python.	Slower compared to C as python has garbage collection.
Implementing data structures required its functions to be explicitly implemented.	Gives ease of implementing data structures with built-in insert, append functions.
It is compulsory to declare the variable type in C.	No need to declare a type of variable.
C program syntax is harder than python.	Python programs are easier to learn, write and read.

In line, an assignment is allowed.	In line, assignment gives an error. E.g. a=5 gives an error in python.
------------------------------------	--


4. Explain raw_input() function with example.

Python raw_input function is used to get the values from the user. We call this function to tell the program to stop and wait for the user to input the values. It is a built-in function. The input function is **used only in Python 2.x** version. The Python 2.x has two functions to take the value from the user. The first one is input function and another one is raw_input() function. The raw_input() function is similar to input() function in Python 3.x. Developers are recommended to use raw_input function in Python 2.x. Because there is a [vulnerability in input function in Python 2.x version.](#)

Python program to demonstrate
input() function in Python2.x

```
val1 = raw_input("Enter the name: ")  
print(type(val1))  
print(val1)
```

```
val2 = raw_input("Enter the number: ")  
print(type(val2))  
val2 = int(val2)  
print(type(val2))  
print(val2)
```

 C:\Python26\python.exe

```
>>>
>>> val1 = raw_input("Enter the name: ")
Enter the name: python3
>>>
>>> print(type(val1))
<type 'str'>
>>>
>>> print(val1)
python3
>>>
>>> val2 = raw_input("Enter the number: ")
Enter the number: 1997
>>>
>>> print(type(val2))
<type 'str'>
>>>
>>> val2 = int(val2)
>>>
>>> print(type(val2))
<type 'int'>
>>>
>>> print(val2)
1997
>>>
>>>
```

5. Write a python program to implement Fibonacci sequence for given input.

Program to display the Fibonacci sequence up to n-th term

```
nterms = int(input("How many terms? "))

# first two terms
n1, n2 = 0, 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
    print("Please enter a positive integer")
elif nterms == 1:
    print("Fibonacci sequence upto",nterms,":")
    print(n1)
else:
    print("Fibonacci sequence:")
    while count < nterms:
        print(n1)
        nth = n1 + n2
        # update values
        n1 = n2
        n2 = nth
        count += 1
```

6. Write a python program to implement isPalindrome() function to check given string is palindrome or not.

```
def isPalindrome(s):
    rev = ''.join(reversed(s))

    if (s == rev):
        return True
    return False

s = input()
ans = isPalindrome(s)

if (ans):
    print("Yes")
else:
    print("No")
```

7. Write a python program to implement Student class which has method to calculate CGPA. Assume suitable class variables.

```
class student:
    arg1 = None
    arg2 = None
    subData = None
    Scale = None
    credits = None
    initCourse = 0
    initgetCredit = 0
    totalCredits = 0
    temp = 0

    def getCourse(self):
        # "get the value of the no of course you registered"
        self.arg1 = input("No of course you have registered: " )
        pass

    def getSubject(self,value):
        # "get the subject value"
        self.arg2 = value
        pass

    def getScale(self):
        # "To get the scale value"
        self.Scale = input("Enter the Scale value(Either 5 or 10): " )
        pass

    def getSubjectData(self):
        # "get the subject Data in string"
```

```

        self.subData = raw_input("Enter the grade: " )
        pass
def getGradeData(self):
    # To calculate grade for two scale,one is for 5.0 and other one for 10.0
    if self.Scale == 10:

        grade1 = {'s':10,'a':9,'b':8,'c':7,'d':5,'e':3,'f':0}
        x=grade1[self.subData]

    else: #5.0 scale
        grade2 = {'a':5,'b':4,'c':3,'d':2,'e':1,'f':0}
        x=grade2[self.subData]
    return x
def getCredits(self):
    # "get credit value"
    self.credits = input("Enter the credits for a subject:" )
    pass

def gpa(self):
    print("Calculate GPA:")
    sem =input("Please Enter Semester: " )
    self.getScale() #input the scale value
    if self.Scale == 5 or self.Scale == 10:
        self.getCourse()
        if self.arg1 >= 2:
            self.calculateGpa()
        else:
            print("In order to calculate Gpa you should have atleast 2 subject minimum")
    else:
        print("you have not entered the scale correctly please try again")
    pass

def calculateGpa(self):
    # "Method to calculate Gpa "
    while self.initCourse!=self.arg1:
        self.initCourse=self.initCourse+1
        self.getCredits()
        self.initgetCredit = self.credits
        self.getSubjectData()
        #type(self.getSubjectData())
        self.temp = self.initgetCredit*self.getGradeData()+self.temp
        self.totalCredits=self.totalCredits+self.initgetCredit

    gpa = round((self.temp+.0)/(self.totalCredits+.0),2)
    print("you have registered for total credits:"+str(self.totalCredits)+" "+"and you have acquired
    GPA:"+str(gpa)+"")
    pass

```

```

def cgpa(self):
    print( "Calculate your cgpa: ")
    semesters = input("Enter how many semester cgpa has to be found of: " )
    counter = 0
    tempInit = 0
    tempTotalCredits = 0
    self.getScale() #input the scale value
    if self.Scale == 5 or self.Scale == 10:
        while counter != semesters:
            counter = counter+1
            print ("Please enter the details of the semester"+" "+str(counter))
            self.getCourse()
            self.calculateGpa()
            tempInit = self.temp+tempInit
            tempTotalCredits = tempTotalCredits + self.totalCredits
            # re-assigning
            self.arg1=0
            self.initCourse =0
            self.temp=0
            self.totalCredits=0
            print "\n"

        cgpa = round((tempInit+.0)/(tempTotalCredits+.0),2)

        print( "you have registered for total credits:"+" "+str(tempTotalCredits)+" "+"and you have acquired
CGPA:\\""+str(cgpa)+"\\" " " )
    else:
        print ("you have not entered the scale correctly please try again")
        pass

if __name__ == '__main__': # main method
    s1 = student() # Creating Instance
    # for calculation of Cgpa (cumulative grade point average)
    s1.cgpa()

```

8. Explain different types of comments in python.

A comment is text that doesn't affect the outcome of a code, it is just a **piece of text** to let someone know what you have done in a program or what is being done in a block of code. This is especially helpful when someone else has written a code and you are analysing it for bug fixing or making a change in logic, **by reading a comment you can understand the purpose of code much faster then by just going through the actual code.**

```
# This is just a text, it won't be executed.
```

There are two types of comments in Python.

1. Single line comment
2. Multiple line comment

Single line comment

In python we use # special character to start the comment. Lets take few examples to understand the usage.

```
# This is just a comment. Anything written here is ignored by Python
```

Multi-line comment:

To have a multi-line comment in Python, we use triple single quotes at the beginning and at the end of the comment, as shown below.

```
'''
This is a
multi-line
comment
'''
```

9. Explain Indexing and Slicing operation for string manipulation with example in python.

Indexing and Slicing

Strings in python support indexing and slicing. To extract a single character from a string, follow the string with the index of the desired character surrounded by square brackets ([]), remembering that the first character of a string has index zero.

```
>>> what = 'This parrot is dead'
>>> what[3]
's'
>>> what[0]
'T'
```

If the subscript you provide between the brackets is less than zero, python counts from the end of the string, with a subscript of -1 representing the last character in the string.

```
>>> what[-1]
'd'
```

To extract a contiguous piece of a string (known as a slice), use a subscript consisting of the starting position followed by a colon (:, finally followed by one more than the ending position of the slice you want to extract. Notice that the slicing stops immediately before the second value:

```
>>> what[0:4]
'This'
>>> what[5:11]
'parrot'
```

One way to think about the indexes in a slice is that you give the starting position as the value before the colon, and the starting position plus the number of characters in the slice after the colon.

For the special case when a slice starts at the beginning of a string, or continues until the end, you can omit the first or second index, respectively. So to extract all but the first character of a string, you can use a subscript of 1: .

```
>>> what[1:]
'his parrot is dead'
```

To extract the first 3 characters of a string you can use :3 .

```
>>> what[:3]
```

```
'Thi'
```

If you use a value for a slice index which is larger than the length of the string, python does not raise an exception, but treats the index as if it was the length of the string.

10. Explain Lists along with methods associated with lists and explain mutability with respect to Lists.

A list is a Python object that represents an ordered sequence of other objects. If loops allow us to magnify the effect of our code a million times over, then lists are the containers we use to easily store the increased bounty from our programs, and to pass large quantities of data into other programs.

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Mutability with respect to lists:

Lists are the first Python type that we've seen that is "mutable." A mutable type means that the inner value can change, not just what the variable references. This can be a confusing concept so let's dive a bit into it:

When we assign a scalar value like `x = 7` we are essentially saying that `x` now has the value 7 until we say otherwise. If we then say `y = x` we are saying `y` is 7 too. If we change `x` later, `y` does not also change. Here's a demonstration:

```
>>> x = 7
>>> y = x
>>> print('x =', x, ' y =', y)
x = 7    y = 7
```

```
>>> x = 8
>>> print('x =', x, ' y =', y)
x = 8    y = 7
```

```
>>> y = 9
>>> print('x =', x, ' y =', y)
x = 8    y = 9
```

Lists offer a different option. If we set `x` to a list, we can also change the make-up of the list:

```
>>> x = [1, 2, 3]
>>> y = x
>>> print('x =', x, ' y =', y)
```



```
x = [1, 2, 3]    y = [1, 2, 3]

>>> x.append(4)
>>> print('x =', x, ' y =', y)
x = [1, 2, 3, 4]  y = [1, 2, 3, 4]
```

In our first example with the integers, any time we set $x =$ or $y =$ it didn't affect the other one. In our second example, when we called `x.append` we know that `y` changed too.

The common thread here is the `=` operator. When we set `x` to a new value, whether it is an int or list (or anything else)- we are creating a new value. When we call a method like `x.append(4)` though, we are not creating a new value but instead modifying (mutating) the underlying array we created when we said `x = [1, 2, 3]`.

It can take some getting used to but there's a separation between the concept of a value and a variable.

Whenever we define a variable by typing an int, str, float, or list directly (what we call a literal) we're doing the same thing, regardless of what type we're using:

- `x = [1, 2, 3]` creates a new list with the value `[1, 2, 3]` and points `x` at it.
- `y = "hello"` creates a new string with the value "hello" and points `y` at it.

When we assign from one variable to another the behavior varies slightly though:

- `x = y` when `y` is immutable means that `x` now has a copy of the value of `y`.
- `x = y` when `y` is mutable means that `x` is a reference to the same value as `y`.

This is why we have the ability to modify our list in the above examples, but when we assign a new value to `x` the other values do not update.

11. Explain lambda function along with map and filter functions.

Ans : the lambda keyword is used to create anonymous functions. It has the following syntax:

lambda arguments: expression

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

The `filter()` function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True.

Example:

```
# Python code to illustrate
# filter() with lambda()
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 != 0) , li))
print(final_list)
```

output: [5, 7, 97, 77, 23, 73, 61]

The map() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

Example:

```
# Python code to illustrate
# map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(map(lambda x: x*2 , li))
print(final_list)
```

Output: [10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

12. Write a python program to read line by line from a given files file1 & file2 and write into file3.

```
data = data2 = ""
```

```
# Reading data from file1
with open('file1.txt') as fp:
    data = fp.read()
```

```
# Reading data from file2
with open('file2.txt') as fp:
    data2 = fp.read()
```

```
# Merging 2 files
# To add the data of file2
# from next line
data += "\n"
data += data2
```

```
with open ('file3.txt', 'w') as fp:
    fp.write(data)
```

13. Explain Tuples, Lists and Dictionaries with example and give comment on mutability for each of them.

Tuples in Python

A tuple is a sequence of **immutable** Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists.

Example:

```
tup = (1, "a", "string", 1+2)
print tup
print tup[1]
```

Lists in Python

Lists are one of the most powerful tools in python. They are just like the arrays declared in other languages. But the most powerful thing is that list need not be always homogenous. A single list can contain strings,

integers, as well as objects. Lists can also be used for implementing stacks and queues. **Lists are mutable**, i.e., they can be altered once declared.

Example:

```
# Declaring a list
L = [1, "a" , "string" , 1+2]
print L
L.append(6)
print L
L.pop()
print L
print L[1]
```

Dictionary in python

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized. Dictionaries are **mutable** as we can update value of particular key.

Example:

```
Dict = { 1: 'Geeks', 'name': 'For', 3: 'Geeks'}
```

```
# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])
```

```
# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])
```

14. Give difference between black-box and glass-box testing.

#	Black Box Testing	White Box Testing
1	Black box testing is the <u>Software testing method</u> which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.

4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.

15. Explain usage of try-except and assert keywords.

try() is used in Error and Exception Handling

There are two kinds of errors :

Syntax Error : Also known as Parsing Errors, most basic. Arise when the Python parser is unable to understand a line of code.

Exception : Errors which are detected during execution. eg – ZeroDivisionError.

List of Exception Errors :

IOError : if file can't be opened

KeyboardInterrupt : when an unrequired key is pressed by the user

ValueError : when built-in function receives a wrong argument

EOFError : if End-Of-File is hit without reading any data

ImportError : if it is unable to find the module

Basic Syntax :

```
try:
    // Code
except:
    // Code
```

How try() works?

- First try clause is executed i.e. the code between try and except clause.
- If there is no exception, then only try clause will run, except clause is finished.
- If any exception occurred, try clause will be skipped and except clause will run.
- If any exception occurs, but the except clause within the code doesn't handle it, it is passed on to the outer try statements. If the exception left unhandled, then the execution stops.
- A try statement can have more than one except clause

Assert: In python assert keyword helps in achieving assumptions that a programmer knows always wants to be true and hence puts them in code so that failure of them doesn't allow the code to execute further.. This statement simply takes input a boolean condition, which when returns true doesn't return anything, but if it is computed to be false, then it raises an AssertionError along with the optional message provided.

*Syntax : **assert condition, error_message(optional)***

Parameters :

condition : The boolean condition returning true or false.

error_message : The optional argument to be printed in console in case of AssertionError

Returns :

Returns AssertionError, in case the condition evaluates to false along with the error message which when provided.

Example:

```
# using assert to check for 0
print ("The value of a / b is : ")
assert b != 0, "Divide by 0 error"
print (a / b)
```

Output:

The value of a/b is :

Runtime Exception :

Traceback (most recent call last):

File "/home/40545678b342ce3b70beb1224bed345f.py", line 10, in

assert b != 0, "Divide by 0 error"

AssertionError: Divide by 0 error

16. Write a python program to implement linear search and binary search.

```
def LinearSearch(arr, x):
```

```
    for i in range(len(arr)):
```

```
        if arr[i] == x:
            return i
```

```
    return -1
```

```
def binary_search(alist, key):
    sort(alist)
    """Search key in alist[start... end - 1]."""
    start = 0
    end = len(alist)
    while start < end:
        mid = (start + end)//2
        if alist[mid] > key:
            end = mid
        elif alist[mid] < key:
            start = mid + 1
        else:
            return mid
    return -1

alist = input('Enter the list of numbers: ')
alist = alist.split()
alist = [int(x) for x in alist]
key = int(input('The number to search for: '))

index = binary_search(alist, key)
index = LinearSearch(alist, key)
```

17. Explain basic idea behind hash table with suitable example.

Hash tables are used to implement map and set data structures in most common programming languages. In C++ and Java they are part of the standard libraries, while Python and Go have builtin dictionaries and maps. A hash table is an unordered collection of key-value pairs, where each key is unique.

Hash tables offer a combination of efficient lookup, insert and delete operations. Neither arrays nor linked lists can achieve this:

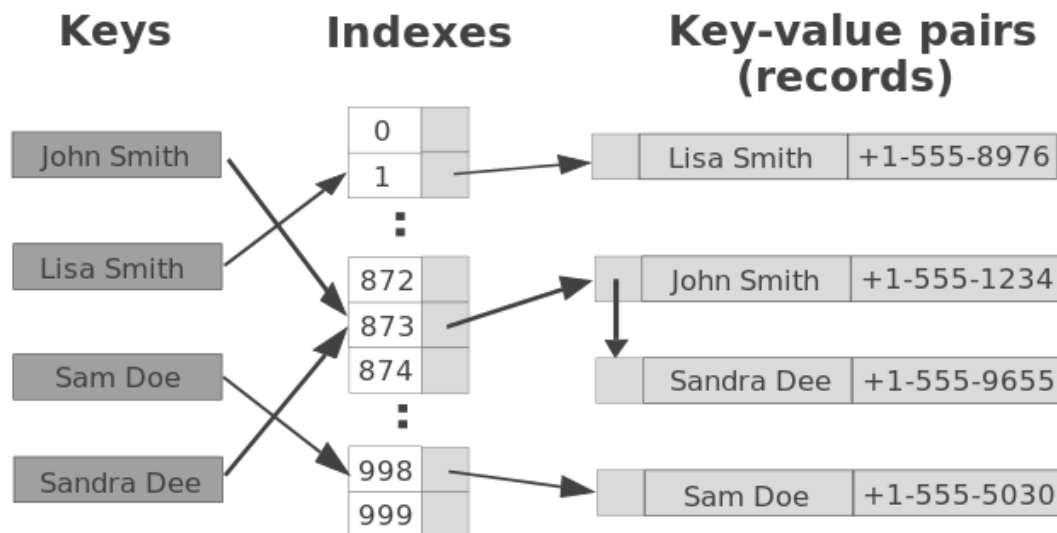
- a lookup in an unsorted array takes linear worst-case time;
- in a sorted array, a lookup using binary search is very fast, but insertions become inefficient;
- in a linked list an insertion can be efficient, but lookups take linear time.

Hashing with chaining (simplified example)

The most common hash table implementation uses chaining with linked lists to resolve collisions. This combines the best properties of arrays and linked lists.

Hash table operations are performed in two steps:

- A key is converted into an integer index by using a hash function.
- This index decides the linked list where the key-value pair record belongs.



This hash table consists of an array with 1000 entries, each of which refers to a linked list of key-value pairs.

Let's start with a somewhat simplified example: a data structure that can store up to 1000 records with random integer keys.

To distribute the data evenly, we use several short lists. All records with keys that end with 000 belong to one list, those with keys that end with 001 belong to another one, and so on. There is a total of 1000 such lists. This structure can be represented as an array of lists:

var table = new LinkedList[1000]

where LinkedList denotes a linked list of key-value pairs.

Inserting a new record (key, value) is a two-step procedure:

- we extract the three last digits of the key, $\text{hash} = \text{key} \% 1000$,
- and then insert the key and its value into the list located at `table[hash]`.

hash = key % 1000

table[hash].AddFirst(key, value)

This is a constant time operation.

A lookup is implemented by

value = table[key%1000].Find(key)

Since the keys are random, there will be roughly the same number of records in each list. Since there are 1000 lists and at most 1000 records, there will likely be very few records in the list `table[key%1000]` and therefore the lookup operation will be fast.

The average time complexity of both the lookup and insert operations is $O(1)$. Using the same technique, deletion can also be implemented in constant average time.

18. Write a python program to implement multiple inheritance.

```
class Person:
    def __init__(self, personName):
        self.name = personName

    def showName(self):
        print(self.name)
```

```
class Student:
    def __init__(self, studentId):
        self.studentId = studentId

    def getId(self):
        return self.studentId

class Resident(Person, Student):
    def __init__(self, name, id):
        Person.__init__(self, name)
        Student.__init__(self, id)

resident_obj = Resident('Kirtan','17CE016')
resident_obj.showName()
print(resident_obj.getId())
```

19. Write a python program to implement merge sort.

```
def merge_sort(alist, start, end):
    if end - start > 1:
        mid = (start + end) // 2
        merge_sort(alist, start, mid)
        merge_sort(alist, mid, end)
        merge_list(alist, start, mid, end)

def merge_list(alist, start, mid, end):
    left = alist[start:mid]
    right = alist[mid:end]
    k = start
    i = 0
    j = 0
    while start + i < mid and mid + j < end:
        if left[i] <= right[j]:
            alist[k] = left[i]
            i = i + 1
        else:
            alist[k] = right[j]
            j = j + 1
        k = k + 1
    if start + i < mid:
        while k < end:
            alist[k] = left[i]
            i = i + 1
            k = k + 1
    else:
```



```

while k < end:
    alist[k] = right[j]
    j = j + 1
    k = k + 1

```

```

alist = input('Enter the list of numbers: ').split()
alist = [int(x) for x in alist]
merge_sort(alist, 0, len(alist))
print('Sorted list: ', end="")
print(alist)

```

20. Explain regular expression methods match and search with its parameters.

The match Function

This function attempts to match RE pattern to string with optional flags.

Here is the syntax for this function:

re.match(pattern, string, flags=0)

Here is the description of the parameters:

pattern: This is the regular expression to be matched.

string: This is the string, which would be searched to match the pattern at the beginning of string.

flags: You can specify different flags using bitwise OR (|).

The re.match() function returns a match object on success, None on failure.

The search Function

This function searches for first occurrence of RE pattern within string with optional flags.

Here is the syntax for this function:

re.search(pattern, string, flags=0)

Here is the description of the parameters:

pattern: This is the regular expression to be matched.

string: This is the string, which would be searched to match the pattern anywhere in the string.

flags: You can specify different flags using bitwise OR (|).

The re.search function returns a match object on success, none on failure.

Example:

```

import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)

```

```

if matchObj:
    print("matchObj.group() : ", matchObj.group())
    print("matchObj.group(1) : ", matchObj.group(1))
    print("matchObj.group(2) : ", matchObj.group(2))
else:
    print("No match!!")

```

```

searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)

```

```

if searchObj:

```

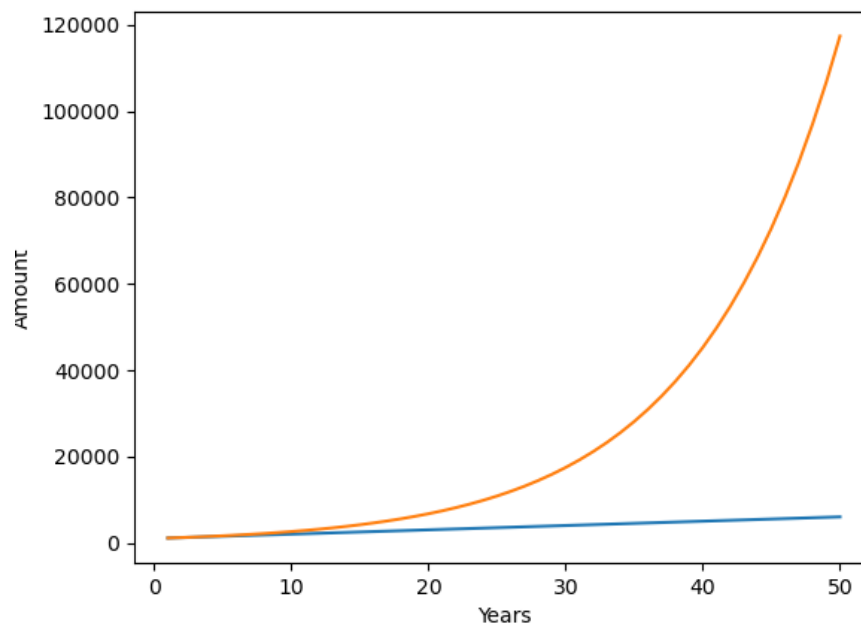
```
print("\nsearchObj.group() : ", searchObj.group())
print("searchObj.group(1) : ", searchObj.group(1))
print("searchObj.group(2) : ", searchObj.group(2))
else:
    print("Nothing found!!")
```

21. Write a python program to create a plotting of years of compounding v/s value of principal using pyplot library. Assume reasonable values for principal, interest rate and years

```
import numpy as np
import matplotlib.pyplot as plt

principal_amount=1000
no_of_years=50
interest_rate=10
simple_interest=list()
compound_interest=list()
for i in range(no_of_years):
    amount=0
    amount+=principal_amount+principal_amount*(i+1)*(interest_rate/100)
    simple_interest.append(amount)
amount=0
for i in range(no_of_years):
    amount+=principal_amount+principal_amount*(interest_rate/100)
    principal_amount=amount
    compound_interest.append(amount)
    amount=0
years=np.arange(1,no_of_years+1)
plt.plot(years,simple_interest)
plt.plot(years,compound_interest)
plt.xlabel('Years')
plt.ylabel('Amount')
plt.show()
```

Output:



22. Explain Caesar cipher and Write a python program to implement Caesar cipher

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials. Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,..., Z = 25. Encryption of a letter by a shift n can be described mathematically as:

$$E_n(x) = (x+n) \bmod 26$$

(Encryption Phase with shift n)

$$D_n(x) = (x-n) \bmod 26$$

(Decryption Phase with shift n)

Program:

```
L2I = dict(zip("ABCDEFGHIJKLMNOPQRSTUVWXYZ", range(26))) #letter to int
I2L = dict(zip(range(26), "ABCDEFGHIJKLMNOPQRSTUVWXYZ")) #int to letter

plaintext = input("\nEnter Plaintext: ")
key = int(input('Enter Key: '))

# encipher
ciphertext = ""
for c in plaintext.upper():
    if c.isalpha():
        ciphertext += I2L[(L2I[c] + key) % 26]
    else:
        ciphertext += c
```

```
# decipher
plaintext2 = ""
for c in ciphertext.upper():
    if c.isalpha():
        plaintext2 += I2L[(L2I[c] - key) % 26]
    else:
        plaintext2 += c

print('\nPlaintext:', plaintext)
print('Cipher Text:', ciphertext)
print('After Decipher:', plaintext2)
```