Python

Experiment No. 5

Aim: To implement functions and recursion.

Problem Statements:

1. Write a program to implement simple calculator using function
2. Write a program to find out the prime factors of a number.
3. Given three integers between 1 and 11, if their sum is less than or equal to 21, return their sum. If their sum exceeds 21 and there's an eleven, reduce the total sum by 10. Finally, if the sum (even after adjustment) exceeds 21, return 'BUST'
4. Program to print Fibonacci series using recursion.
5. Write a function that takes in a list of integers and returns True if it contains 007 in order

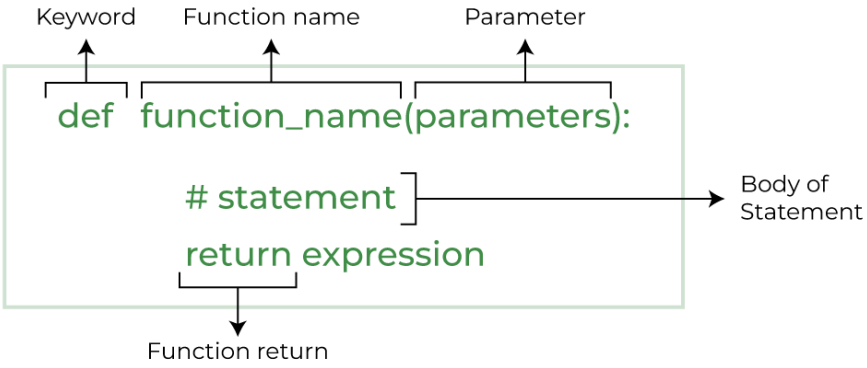
Theory :

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability
-

Python function declaration:



Types of Functions in Python

There are mainly two types of functions in Python.

- Built-in library function: These are Standard functions in Python that are available to use.
- User-defined function: We can create our own functions based on our requirements.

Creating a Function in Python

We can create a user-defined function in Python, using the def keyword. We can add any type of functionalities and properties to it as we require.

```
def fun():
  print("Welcome to GFG")
```

Calling a Python Function

After creating a function in Python we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

```
def fun():
    print("Welcome to GFG")
# Driver code to call a function
fun()
```

Python Function Arguments

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

```
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
# Driver code to call the function
evenOdd(2)
```

```
even0dd(3)
```

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

Default argument

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

Keyword arguments (named arguments)

You can also send arguments with the key = value syntax.

```
def my_function(child3, child2, child1):
 print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Positional arguments

We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places.

Arbitrary arguments (variable-length arguments *args and **kwargs)

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

*args in Python (Non-Keyword Arguments)

```
def myFun(*argv):

   for arg in argv:

      print(arg)

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

**kwargs in Python (Keyword Arguments)

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Docstring

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function:

Syntax: print(function_name.__doc__)

```
def evenOdd(x):
    """Function to check if the number is even or odd"""

    if (x % 2 == 0):
        print("even")
    else:
        print("odd")
  # Driver code to call the function
print(evenOdd.__doc__)
```

Python Function within Functions

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

```
def f1():

  s = 'I love GeeksforGeeks'

  def f2():

    print(s)

  f2()

# Driver's code

f1()
```

Return Statement in Python Function

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller. The syntax for the return statement is:

return [expression_list]

The return statement can consist of a variable, an expression, or a constant which is returned at the end of the function execution. If none of the above is present with the return statement a None object is returned.

```
def square_value(num):

  """This function returns the square

  value of the entered number"""

  return num**2

print(square_value(2))

print(square_value(-4))
```

Pass by Reference and Pass by Value

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

```
def myFun(x):
```

```
    x[0] = 20

# Driver Code (Note that lst is modified

# after function call.

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)
```

Example 2:

```
def myFun(x):

    x = [20, 30, 40]

 # Driver Code (Note that lst is not modified

# after function call.

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Advantages of using recursion

A complicated function can be split down into smaller sub-problems utilizing recursion.

Sequence creation is simpler through recursion than utilizing any nested iteration.

Recursive functions render the code look simple and effective.

Disadvantages of using recursion

A lot of memory and time is taken through recursive calls which makes it expensive for use.

Recursive functions are challenging to debug.

The reasoning behind recursion can sometimes be tough to think through.

Syntax:

```
def func(): <--
            |
            | (recursive call)
            |
    func() ---
```

Q.1 Write a program to implement simple calculator using function

Code:

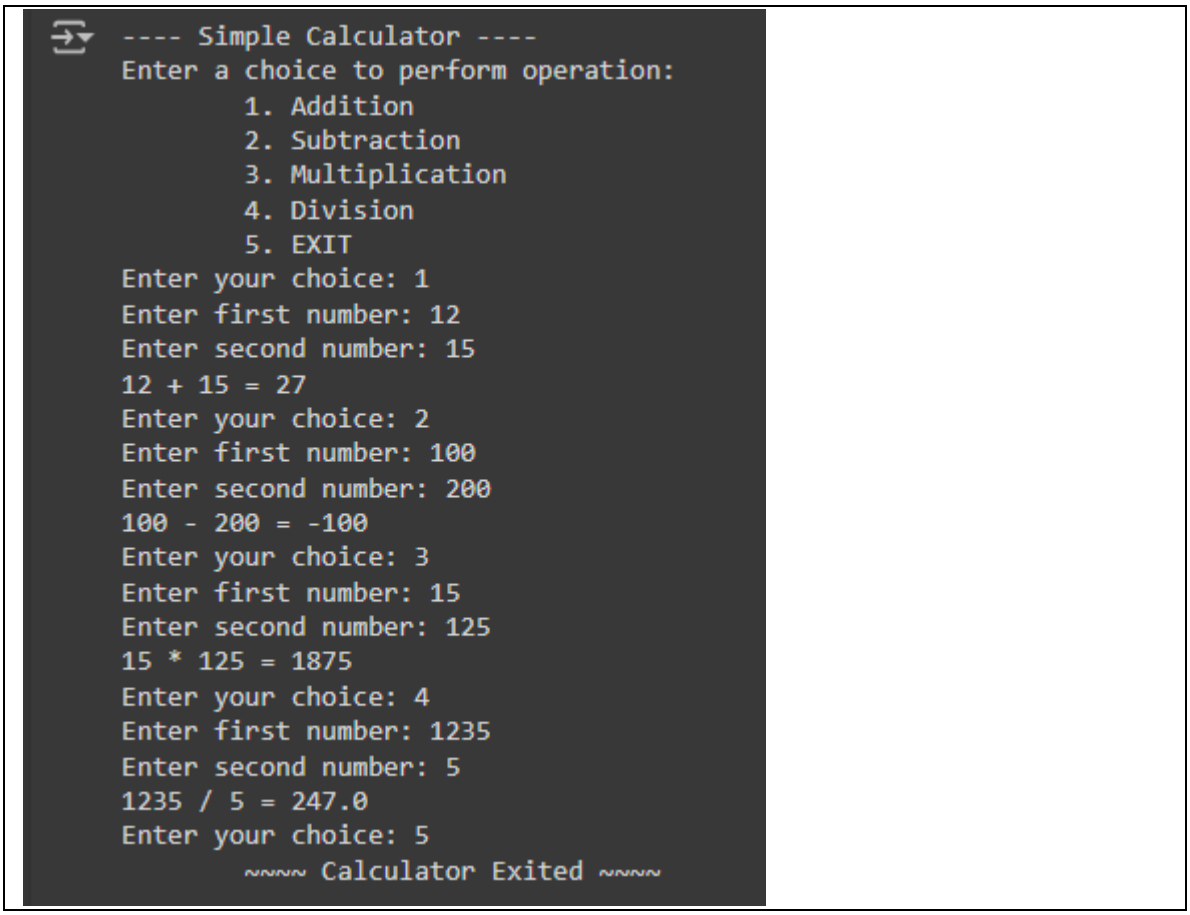
```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b

choice=0
print("---- Simple Calculator ----")
print("Enter a choice to perform operation:\n\t1.
Addition\n\t2. Subtraction\n\t3. Multiplication\n\t4.
Division\n\t5. EXIT")

while choice!=5:
    choice=int(input("Enter your choice: "))
    if choice in range(1,5):
        a=int(input("Enter first number: "))
        b=int(input("Enter second number: "))

        if choice==1:
            print(a,"+",b,"=",add(a,b))
        elif choice==2:
            print(a,"-",b,"=",sub(a,b))
        elif choice==3:
            print(a,"*",b,"=",mul(a,b))
        elif choice==4:
            print(a,"/",b,"=",div(a,b))
    elif choice==5:
        print("\t~~~~ Calculator Exited ~~~~\t")
    else:
        print("Invalid Choice")
```

Output:



```
---- Simple Calculator ----
Enter a choice to perform operation:
        1. Addition
        2. Subtraction
        3. Multiplication
        4. Division
        5. EXIT
Enter your choice: 1
Enter first number: 12
Enter second number: 15
12 + 15 = 27
Enter your choice: 2
Enter first number: 100
Enter second number: 200
100 - 200 = -100
Enter your choice: 3
Enter first number: 15
Enter second number: 125
15 * 125 = 1875
Enter your choice: 4
Enter first number: 1235
Enter second number: 5
1235 / 5 = 247.0
Enter your choice: 5
        ~~~~ Calculator Exited ~~~~
```

Q.2 Write a program to find out the prime factors of a number.

Code:

```
def prime_factors(num):
    factors = []
    for i in range(2, num+1):
        while num % i == 0:
            factors.append(i)
            num //= i
    return factors

num = int(input("Enter a number: "))
print("Prime factors of", num, "are:", prime_factors(num))
```

Output:

```
Enter a number: 50
Prime factors of 50 are: [2, 5, 5]
```

Q.3 Given three integers between 1 and 11, if their sum is less than or equal to 21, return their sum. If their sum exceeds 21 and there's an eleven, reduce the total sum by 10. Finally, if the sum (even after adjustment) exceeds 21, return 'BUST'

Code:

```
a=int(input("Enter a number : "))
b=int(input("Enter a number : "))
c=int(input("Enter a number : "))

def adjustment(a,b,c):
    if (a,b,c in range(0,12)):
        sum=a+b+c
        if sum<=21:
            return sum
        elif sum>21 and (a==11 or b==11 or c==11):
            sum-=10
            if sum>21:
                return "BUST"
            else:
                return sum
    return "Numbers not in range of 1 and 11."

print(adjustment(a,b,c))
```

Output:

```
Enter a number : 11
Enter a number : 11
Enter a number : 11
BUST
```

Q.4 Program to print Fibonacci series using recursion.

Code:

```
n=int(input("Enter a number: "))

def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)

print("Fibonacci series:")
for i in range(n):
    print(fibonacci(i),end=" ")
```

Output:

```
Enter a number: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
```

Q.5 Write a function that takes in a list of integers and returns True if it contains 007 in order

Code:

```
intList = input("Enter numbers in list (seperated by Space):
").split()

def check007(intList):
    for i in range(len(intList)):
        if intList[i:i+3] == ['0','0','7']:
            return True
    return False

print(check007(intList))
```

Output:

```
Enter numbers in list (seperated by Space): 1 2 3 0 0 7 8 9 10
True
```

Conclusion: Thus studied functions and recursions.