Python_Practice_Questions

January 28, 2021

Importing Libraries

```
In [1]: from math import sqrt
        import pdb
```

1. Write a function that inputs a number and prints the multiplication table of that number.

```
In [2]: #creating list of integers from 1 to 10.
        num = [1,2,3,4,5,6,7,8,9,10]

        def Multiplication_table():
          """
          This function prints the multiplication table for an input number
          """
          _number = int(input("Enter the Number: "))
          table = list(map(lambda x : x * _number,num))
          print("Multiplication table for {0} is : ".format(_number),table)

        #calling a function
        if __name__ == '__main__':
          Multiplication_table()
```

```
Enter the Number: 15
Multiplication table for 15 is :  [15, 30, 45, 60, 75, 90, 105, 120, 135, 150]
```

2. Write a program to print twin primes less than 1000. If two consecutive odd numbers are both prime then they are known as twin primes.

```
In [3]: twin_primes=[]
        def get_Primes(_number):
          """
          This function gets all prime numbers between given range.
          """
          for i in range(2,_number):
            if (_number % i == 0):
              return False
              break
```

```
      else:
        return True

    #try to improve this code before sending.
    def get_TwinPrimes():

      """
      This funtion gets all twin prime numbers from given list of prime numbers.
      """
      prime_numbers = list(filter(get_Primes,range(2,1000)))
      for i in range(0,len(prime_numbers)-1):
          if(abs(prime_numbers[i] - prime_numbers[i+1]) == 2):
            twin_primes.append(tuple([prime_numbers[i],prime_numbers[i+1]]))

      print("Twin Primes are :",twin_primes)

    if __name__ == '__main__':
      get_TwinPrimes()
```

```
Twin Primes are : [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73),
```

3. Write a program to find out the prime factors of a number. Example: prime factors of 56 - 2, 2, 2, 7

```
In [4]: prime_factors = []
        def Find_primeFactors():

          """
          This function computes the prime factors of a given number.
          """
          _num = int(input("Enter the number: "))
          number = _num
          for i in range(2,_num):
            if(_num % i == 0):
              count = 0
              while(_num%i ==0):
                _num = _num/i
                count += 1
              prime_factors.extend([i]*count)
          print("Prime Factorization of {0} is :".format(number),prime_factors)

         #calling a function
        if __name__ == '__main__':
          Find_primeFactors()
```

```
Enter the number: 56
Prime Factorization of 56 is : [2, 2, 2, 7]
```

4. Write a program to implement these formulae of permutations and combinations. Number of permutations of n objects taken r at a time: p(n, r) = n! / (n-r)!. Number of combinations of n objects taken r at a time is: c(n, r) = n! / (r!*(n-r)!) = p(n,r) / r!

```
In [5]: def Fact(_num):
            """
            This function computes Factorial of a given number.
            """
            if(_num == 0 or _num == 1):
                return 1
            else:
                return (_num*Fact(_num-1))


        def Perm_Comb():
            """
            This function prints permutation and combination for a given number n and r.
            """
            n=(int(input("Input n objects : ")))
            r=(int(input("Input r objects : ")))

            if(n>r):

                permutation = Fact(n)/Fact(n-r)
                print("Permutation of n objects taken r at a time : ",permutation)
                combination = permutation / Fact(r)
                print("Combination of n objects taken r at a time : ",combination)
            else:
                print(" r objects should be less than or equal to n")

        if __name__ == '__main__':
            Perm_Comb()
```

```
Input n objects : 10
Input r objects : 5
Permutation of n objects taken r at a time :  30240.0
Combination of n objects taken r at a time :  252.0
```

5. Write a function that converts a decimal number to binary number

```
In [6]: binary_num = []
        def DectoBinary(_number):
          """
          This function converts a decimal number to binary number
          """
          while(_number > 0):
            remainder = _number % 2
            _number = _number//2
            binary_num.append(remainder)

          binary_num.reverse()

        if __name__ == '__main__':
            _num = int(input("Enter a Number : "))
            DectoBinary(_num)
            print("The Binary representation of {0} is :".format(_num),binary_num)
```

```
Enter a Number : 64
The Binary representation of 64 is : [1, 0, 0, 0, 0, 0, 0]
```

6.Write a function cubesum() that accepts an integer and returns the sum of the cubes of individual digits of that number. Use this function to make functions PrintArmstrong() and isArmstrong() to print Armstrong numbers and to find whether is an Armstrong number.

```
In [7]: def cubesum(_num):
          """

          This function returns the sum of the cubes of individual digits of the number.
          """
          _List = []

          while(_num):
            _List.append((_num%10)**3)
            _num=_num//10
          _List.reverse()
          _sum = sum(_List)
          return _sum

        def PrintArmstrong(flag):
          """
          This function prints Armstrong number.
          """
          print("Entered Number is ArmStrong") if flag==True else print("Entered Number is not

        def isArmstrong():
          """
          This function checks whether the number is Armstrong or not.
          """
          _num = int(input("Enter the Number : "))
          PrintArmstrong( True if(_num == cubesum(_num)) else False)

        if __name__ == '__main__':
          isArmstrong()
```

```
Enter the Number : 407
Entered Number is ArmStrong
```

7.Write a function prodDigits() that inputs a number and returns the product of digits of that number.

```
In [8]: def prodDigits(_num):

            """

            This function returns the product of digits of the number
            """

            prod=1
            while(_num>0):
              prod = prod * (_num%10)
              _num = _num//10

            return prod

        if __name__ == '__main__':
            _num = int(input("Input the Number : "))
            prod=prodDigits(_num)
            print("Product of digits of the number is :",prod)
```

```
Input the Number : 48
Product of digits of the number is : 32
```

8.If all digits of a number n are multiplied by each other repeating with the product, the one digit number obtained at last is called the multiplicative digital root of n. The number of times digits need to be multiplied to reach one digit is called the multiplicative persistance of n. Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3) 341 -> 12->2 (MDR 2, MPersistence 2) Using the function prodDigits() of previous exercise write functions MDR() and MPersistence() that input a number and return its multiplicative digital root and multiplicative persistence respectively

```
In [9]: def MDR(_num):
            """

            This function computes the multiplicative digital root of a number.
            """

            while(_num > 10):
              _num=prodDigits(_num)

            print("Multiplicative digital root of a number is :",_num)

        def MPersistance(_num):
            """
            This function computes the multiplicative persistance of a given number.
            """
            i=0
```

```
          while(_num > 10):
            i+=1
            _num=prodDigits(_num)
          print("Multiplicative persistance a number is :",i)

        if __name__ == '__main__':
          _num = int(input("Enter the Number : "))
          MDR(_num)
          MPersistance(_num)
```

```
Enter the Number : 86
Multiplicative digital root of a number is : 6
Multiplicative persistance a number is : 3
```

9.Write a function sumPdivisors() that finds the sum of proper divisors of a number. Proper divisors of a number are those numbers by which the number is divisible, except the number itself. For example proper divisors of 36 are 1, 2, 3, 4, 6, 9, 18

```
In [10]: def sumPdivisors(_num):
           """

           This function finds the sum of proper divisor of a number.
           """
           sum = 0
           for i in range(1,int((_num)/2)+1):
             if(_num%i ==0):
               sum=sum+i
               #print(i)
           print("The Sum of proper divisor of a number is : ",sum)

         if __name__ == '__main__':
           sumPdivisors(int(input("Enter the Number : ")))
```

```
Enter the Number : 36
The Sum of proper divisor of a number is :  55
```

10.A number is called perfect if the sum of proper divisors of that number is equal to the number. For example 28 is perfect number, since 1+2+4+7+14=28. Write a program to print all the perfect numbers in a given range

```
In [11]: def perfectNumber(_num):
           """

           This function print all the perfect numbers in a given range
           """
           sum = 0
           for i in range(1,int((_num)/2)+1):
                if(_num%i ==0):
                  sum=sum+i
              #print(sum)

              return True if(_num==sum) else False

          if __name__ == '__main__':
              perfectno_list = list(filter(perfectNumber,range(1,10000)))
              print(perfectno_list)
```

```
[6, 28, 496, 8128]
```

11.Two different numbers are called amicable numbers if the sum of the proper divisors of each is equal to the other number. For example 220 and 284 are amicable numbers.

Sum of proper divisors of 220 = 1+2+4+5+10+11+20+22+44+55+110 = 284 Sum of proper divisors of 284 = 1+2+4+71+142 = 220 Write a function to print pairs of amicable numbers in a range

```
In [12]: def Sumofproper_div(_number):
             sum=0
             for i in range(1,int((_number)/2)+1):
                 if(_number%i ==0):
                     sum=sum+i
             return sum

         amicable_pairs =[]
         def amicableNum():
             _num = int(input("Enter the Range : "))
             for i in range(1,_num+1):
                 #pdb.set_trace()
                 _num1 = Sumofproper_div(i)
                 #print(_num1)
                 if (_num1 != i):
                     _num2 = Sumofproper_div(_num1)
                     #print(_num1,_num2)
                     if(i == _num2):
                         amicable_pairs.append(tuple(sorted([_num1,_num2])))

             print("Pairs of Amicable numbers in given range : ",list(set(amicable_pairs)))



         if __name__ == '__main__':
             amicableNum()
```

```
Enter the Range : 10000
Pairs of Amicable numbers in given range :  [(1184, 1210), (220, 284), (5020, 5564), (6232, 63
```

12.Write a program which can filter odd numbers in a list by using filter function

```
In [13]: def filter_Oddnumbers(_num):
           """

           This function filters odd numbers from a list.
           """
           return True if(_num %2==1) else False


         if __name__ == '__main__':
           odd_num = list(filter(filter_Oddnumbers,range(1,20)))
           print("Odd Numbers are :",odd_num)
```

```
Odd Numbers are : [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

13.Write a program which can map() to make a list whose elements are cube of elements in a given list

```
In [14]: Cube_ofNum = list(map(lambda x: x*x*x,range(1,11)))
         print("Cube of Elements in Given list :",Cube_ofNum)
```

```
Cube of Elements in Given list : [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

14.Write a program which can map() and filter() to make a list whose elements are cube of even number in a given list

```
In [15]: Cube_ofNum = list(map(lambda x: x*x*x,range(1,11)))
         Even_numcube = list(filter(lambda x:x%2==0,Cube_ofNum))
         print("Cubes of Numbers in a list :",Cube_ofNum)
         print("Cubes of Even Numbers in a list :",Even_numcube)
```

```
Cubes of Numbers in a list : [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
Cubes of Even Numbers in a list : [8, 64, 216, 512, 1000]
```