Functions Assignment

This assignment is on Python Functions as part of applied ai course. This has 14 questions.

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

   Steps :

   A. Accepts user input for multiplication table to be printed
   B. Invokes user-defined function multiplication
   C. Function accepts user input and prints multiplication table from 1 to 15

In [118]:

```
def multiplication(n):

  '''
     This function accepts user input and prints multiplication table till 15 for the i
nput number
  '''

  for i in range(1,16):
    print(n,'*',i,'=',n*i)


num = int(input('Print multiplication table of : '))
multiplication(num)
```

```
Print multiplication table of : 18
18 * 1 = 18
18 * 2 = 36
18 * 3 = 54
18 * 4 = 72
18 * 5 = 90
18 * 6 = 108
18 * 7 = 126
18 * 8 = 144
18 * 9 = 162
18 * 10 = 180
18 * 11 = 198
18 * 12 = 216
18 * 13 = 234
18 * 14 = 252
18 * 15 = 270
```

1. 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

   Steps :

   A. Define prime number user-defined function. A number which divides by 1 and itself is called prime.
   B. Define twinprime user-defined function. For each consecutive odd numbers from 3 till the number we want to print , check if both consecutive numbers are prime.
   C. if both are prime print the numbers, Otherwise skip.

In [119]:

```python
def prime(n):
  '''
     This function checks if the input number is prime or not.
     If Prime, returns True else it returns False
  '''

  for i in range(2,int(n ** 0.5)+2): # instead of going for entire n-1 numbers, we can do only till sqrt(n). We added 2 one for floor and other for indexing
    if n%i == 0 :
      return False
  return True

def twinprime(num):
  '''
     This function returns numbers if consecutive odd numbers are prime.
     Ex: 3 and 5 are consecutive odd numbers
  '''
  for i in range(3,num,2): # starts from 3 and increment by 2 as we need to find consecutive odd numbers
    nextnum = i + 2 # generate next num for comparison
    if (prime(i) and prime(nextnum)):
      print(i,'and',nextnum)

twinprime(1000)
```

```
3 and 5
5 and 7
11 and 13
17 and 19
29 and 31
41 and 43
59 and 61
71 and 73
101 and 103
107 and 109
137 and 139
149 and 151
179 and 181
191 and 193
197 and 199
227 and 229
239 and 241
269 and 271
281 and 283
311 and 313
347 and 349
419 and 421
431 and 433
461 and 463
521 and 523
569 and 571
599 and 601
617 and 619
641 and 643
659 and 661
809 and 811
821 and 823
827 and 829
857 and 859
881 and 883
```

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

   Steps :

   A. Check if number is divided by 2. If yes, now the new number will be half of original. Repeat this till the number is divisible by 2.
   B. if number is no longer divided by 2, we start from 3 till square-root of a number with increment of 2. Reason for chosing square root is the highest factor of a number can be square root of number. For example : prime factors of 49 is 7 which is square root of number.
   C. There are chances that step2 will not be reached for some numbers. If the number is prime, we add the numbers.

   Example : 56

   A. 56 is divided by 2. New Number is 28
   B. 28 is divided by 2. New Number is 14
   C. 14 is divided by 2. New Number is 7
   D. Sqrt of 7 is 2.64. Integer is 2. This doesnt go inside loop of step 2.
   E. Last if now adds the number 7 to factors.

Credit : https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/ (https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/) for step-2 logic on square-root

In [120]:

```
def primefactor(num):
  '''
     This function returns the primefactors of a given number in a list
  '''
  factors = []

  while num % 2 == 0 : # block to handle only 2
       factors.append(2)
       num /= 2

  for i in range(3, int(num ** 0.5)+1, 2): # for loop to handle from 3 and all other even numbers are ignored

       while num % i == 0:
           factors.append(i)
           num /= i

  if num > 2:
     factors.append(int(num))

  return factors

print(primefactor(56))
print(primefactor(1035))
```

```
[2, 2, 2, 7]
[3, 3, 5, 23]
```

1. Write a program to implement these formulae of permutations and combinations. Number of permutations of n objects taken r at 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!

   Steps :

   A. Write a recursive function to calculate factorial.
   B. Write a function to calculate permutations using the formula
   C. Write a function to calculate combinations using the formula

In [121]:

```
def factorial(num):
  '''
  This function returns factorial of a number using recursive method
  '''
  return 1 if num<=1 else num * factorial(num-1) # using recursive. Boundary case : 0 and 1 if factorial is 1. else n * (n-1) !

def nbofpermutations(n,r):
  '''
  This function retuerns number of permulations using the factorial formula
  '''
  return (factorial(n)/ factorial(n-r))

def nbofcombinations(n,r):
  '''
  This function returns number of combinations using nb of permutaions and factorial formula
  '''
  return (nbofpermutations(n,r)/ factorial(r))

print(nbofpermutations(9,2))
print(nbofcombinations(9,4))
```

```
72.0
126.0
```

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

17 — 1
8 — 0
4 - 0
2 — 0
1

25 - 1
12 - 0
6 — 0
3 — 1
1

16 8 4 2 1
1 1 0 0 1
16 + 8 + 1
= 25

In [122]:

```
def dectobin(num):
  '''
  This function prints the binary equvialent of a given integer
  '''

  if num >= 1: # call recursive function till the result reaches 1
    dectobin(num//2)
  print(num%2,end='') # end parameter used to remove new line in output


dectobin(25)
```

```
011001
```

Credit : https://www.guru99.com/print-without-newline-python.html (https://www.guru99.com/print-without-newline-python.html) for converting the output to a single line.

1. Write a function cubesum() that accepts an integer and returns the sum of 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

   Steps :

   A. Create a function cubseum which calculates the sum by adding cube of each digits of a number.
   B. Create a function printArmStrong which prints Armstrong numbers from 100 as Armstrong numbers can be calculated using the cubesum function. It uses cubesum function to print ArmStrong
   C. Create a function isArmstrong which identifies given number is armstrong. It uses cubesum to identify it.

In [123]:

```
def cubesum(num):
  '''
  This function returns sum of cube of the digits of number passed
  '''

  sum =0

  for i in range(len(str(num))): # Loop based on the number of digits
    sum = sum + ((num%10)**3) # cube of each digit by taking remainder
    num = num // 10 # number set for next iteration

  return sum

def printArmStrong(num):
  '''
  This function prints Armstrong numbers for a range of numbers
  '''

  print('Armstrong Numbers are :')
  for i in range(100,num): # only 3 digit numbers are cosniderded
      if cubesum(i) == i :
         print(i)

def isArmStrong(num):
  '''
  This function checks if the number is armstrong or not.
  Prints message whether number is armstronmg or not
  '''

  if (cubesum(num)== num):
    print(num, 'is an armstrong number')
  else :
    print(num, 'is not an armstrong number')

printArmStrong(1000)
print('***************************************')
isArmStrong(153)
print('***************************************')
isArmStrong(408)
print('***************************************')
isArmStrong(12)
print('***************************************')
isArmStrong(370)
```

```
Armstrong Numbers are :
153
370
371
407
***************************************
153 is an armstrong number
***************************************
408 is not an armstrong number
***************************************
12 is not an armstrong number
***************************************
370 is an armstrong number
```

1. Write a function prodDigits() that inputs a number and return product of digits of that number

   Steps :

   A. convert input number to a list of numbers
   B. List comprhension can be used to convert from number to list
   C. use reduce function to calculate the product

In [124]:

```
from functools import reduce

def prodDigits(num):
    '''
    This function returns the product of digits of a number
    '''
    lnum = [int(x) for x in str(num)]
    prod = reduce(lambda x,y : x * y , lnum)
    return prod

prodDigits(12345)
```

Out[124]:

```
120
```

1. If all digits of a number n are multiplied by each other repeating with 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 multiplicative persistance of n.

   Example : 86 -> 48 -> 32 -> 6 (MDR 6, Mpersistence 3) Example : 341 -> 12 -> 2 (MDR 2, MPersistence 2)

   using the function prodDigits() of previous exercise write functions MDR()and MPersistence() that returns a number and return its multiplicative digital root and multiplicative persistence respectivley

   Steps :

   A. In MDR, call prodDidgits till the resultant number is single digit
   B. In Mpersistence , call prodDigits and count the nb of times. That gives Mpersistence

In [125]:

```
def MDR(num):
  '''
  This function return the multiplicative digital root of a number
  '''
  num = prodDigits(num) # calculate product
  res = MDR(num) if num > 9 else num # call same function recursivley till the result reaches single digit
  return res

def MPersistence(num):
  '''
  This function returns the multiplicative persistence of a number
  '''
  cnt = 0
  while (num >9): # loop till the result reaches single digit
      num = prodDigits(num)
      cnt += 1
  return cnt


print(MDR(341))
print(MPersistence(341))
print(MDR(86))
print(MPersistence(86))
```

```
2
2
6
3
```

1. Write a function sumPdivisors() that finds the sum of proper divisions 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

   Steps:

   A. Divisors are come in pairs. For example 1,36 and 36,1 , 2,18 and 18,2 etc.
   B. To Optimize it, we will loop till the line is reached and after that reversal will happen.
   C. Square root of a number gives the line.
   D. We add both of the numbers to the list like 1 and 36, 2 and 18 etc
   E. Need to handle 2 boundary cases. if both the divisors are equal like 6 and 6, we consider only one number. Also, we should not consider the actual number
   F. List is populated now and we sort the list
   G. Last, we call reduce function to calculate the sum.

In [126]:

```
def sumPdivisors(num):
  '''
  This function returns sum of proper divisors of a number
  '''
  i = 1
  res =[]

  while i**2 <= num: # loop only till square root of number
    if (num % i ==0): # only if numbver is divided exactly, then add the number to list
      if ((i == num/i) or (num/i == num)): ## Boundary cases if both divisors are same or divisor is actual number add only one number to list
        res.append(i)
      else: ## All other cases add both numbers to list
        res.append(i)
        res.append(int(num/i))
    i = i +1 # loop counter
  res.sort() # In-place sort the list
  #print(res)
  #print('********************************************')
  return reduce(lambda x,y : x+y, res) # calculate sum of divisors

print(sumPdivisors(36))
print(sumPdivisors(28))
print(sumPdivisors(496))
```

```
55
28
496
```

1. A number is called perfect if the sum of the 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 range

   Steps :

   A. Loop through range of numbers
   B. for each number, check the number and sum of proper divisors are same
   C. If same print , otherwise dont print it

In [128]:

```
def perfectNumbers(num):
  '''
  This function prints all the perfect numbers in a range
  '''

  for i in range(1,num+1): # Loop through range of numbers
      if i == sumPdivisors(i): # check if number and sumof divisors are same
          print(i)

perfectNumbers(1000)
```

```
1
6
28
496
```

1. 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. Write a function to print pairs of amicable numbers in a range.

   Steps :

   A. Loop all the elements in a range
   B. Calculate the sumof divisors of a element
   C. calculate the sum of divisors of a above result
   D. If both are same, its amicable and add to a list as Tuples
   E. sort the tuples
   F. Finally convert to set to remove same pairs

In [129]:

```
def amicable(num):
  '''
  This function geneartes amicable numbers for a range of numbers
  '''
  result = []
  for i in range(1, num+1): # Loop through range of numbers
      sumdiv = sumPdivisors(i) # calculate sumof divisors of an element

      if sumPdivisors(sumdiv) == i and i != sumdiv : # check if element and sum of divi
sors of sum are same
          result.append(tuple(sorted([i, sumdiv])))
  return set(result) # remove duplicate pairs


amicable(10000)
```

Out[129]:

```
{(220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368)}
```

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

   Steps :

   A. Write lambda function which returns True for Odd Numbers and False for even numbers
   B. Filter function selects only true results
   C. convert the result to list and return

In [130]:

```
def filterodd(lst):
  '''
  This program returns od number if a input list
  '''

  res = list(filter(lambda x : True if x%2 != 0 else False, lst))
  return res

lstodd = filterodd(range(51))
print(lstodd)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 3
9, 41, 43, 45, 47, 49]
```

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

   Steps :
   A. Define a function which accepts list
   B. write a lambda function to calculate cube
   C. use map function and return the result
   D. convert the result to list and return

In [131]:

```python
def cubelist(lst):
    '''
    This function returns cube of elements in a list
    '''
    
    res = list(map(lambda x : x ** 3, lst))
    return res

lstcube = cubelist(range(11))
print(lstcube)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

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

   Steps :
   A. Define a function which accepts list
   B. Write a lambda and filter only even numbers
   C. Apply map to the above
   D. convert to list and return

In [132]:

```python
def cubeeven(lst):
    '''
    This function returns cube of even number in a list
    '''
    
    res = list(map(lambda x : x**3,filter(lambda x : True if x%2 == 0 else False, lst)))
    return res

lstcubeeven = cubeeven(range(1,11))
print(lstcubeeven)
```

```
[8, 64, 216, 512, 1000]
```