Practice Questions On Functions

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

In [1]:
```
import sys

# 1) With user input

def multiplicationTable(num):
    for i in range(1,11):
        print("{} x {} = {}".format(num,i,num*i))

# get a number from user input

try:
    num = int(input("Enter a number"))
    multiplicationTable(num)
except ValueError:
    print(sys.exc_info()[0])
    print("Not an Integer number")
```

```
Enter a number23
23 x 1 = 23
23 x 2 = 46
23 x 3 = 69
23 x 4 = 92
23 x 5 = 115
23 x 6 = 138
23 x 7 = 161
23 x 8 = 184
23 x 9 = 207
23 x 10 = 230
```

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 [2]:

```
import pdb
import math

def twinPrimeNumbers():
    lst = [3]

    for i in range(5,1000):
        isPrime = True
        for j in range(2,math.floor(math.sqrt(i))+1):
            if(i%j == 0):
                isPrime = False
                break

        if(isPrime):
            lst.append(i)

        if(len(lst) > 1):
            if((lst[1] - lst[0]) == 2):
                print(lst[0],lst[1])
            lst[0] = lst[1]
            lst.pop(1)

%time
twinPrimeNumbers()
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 11.2 µs
3 5
5 7
11 13
17 19
29 31
```

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

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

In [4]:
```
def primeFactorsOfNumber(num):
    if(not num in [0,1]):
        lst = []
```

```
        i = 2
        while(i <= num):
            if(num%i == 0):
                lst.append(i)
                num = num/i
                i = 2
            else:
                i += 1
        print(lst)
    else:
        lst = [num]
        print(lst)
primeFactorsOfNumber(56)
```

```
[2, 2, 2, 7]
```

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 [3]:
```
def factorial(n):
    if(n in [0,1]):
        return 1
    else:
        return n * factorial(n-1)

def p(n,r):
    if(n >= r):
        return factorial(n)/factorial(n-r)
    else:
        print("Not valid")

def c(n,r):
    if(n >= r):
        return factorial(n)/(factorial(r)*factorial(n-r))
    else:
        print("Not valid")
```

```
n = 6
r = 3
p = p(n,r)
c = c(n,r)

print(p)
print(c)

print("c({}, {}) = {}! / ({}!*({}-{})!) = p({},{}) / {}!".format(n,r,n,
r,n,r,n,r,r))
```

```
120.0
20.0
c(6, 3) = 6! / (3!*(6-3)!) = p(6,3) / 3!
```

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

In [2]:
```
# Code for Integer numbers

# for number being 7
# (num % 2) + 10 * getBinaryNumber(num//2)
# (1) + 10 * getBinaryNumber(3)
# (1) + 10 * ((3 % 2) + 10 * getBinaryNumber(3//2))
# 1 + 10 * (1 + 10 * (1))
# 1 + 10 * (1 + 10)
# 1 + 10 * 11
# 1 + 110
# 111


def getBinaryNumber(num):

    if(num == 1):
        return 1
    return (num%2)+ 10*getBinaryNumber(num//2)
```

```
n = getBinaryNumber(7)
print(n)
```

```
111
```

In [1]:

```
# Code for fractional numbers

# get the integral part binary
def getBinaryIntegral(num):

    if(num == 1):
        return 1
    return (num%2)+ 10*getBinaryIntegral(num//2)

# Logic to get the decimal fraction binary value is to multiply the fra
ctional part with 2 till the fractional part becomes 0

# Here is an example of such conversion using the fraction 0.375.
#  0.375 * 2 = 0 + 0.75
#  0.75 * 2 = 1 + 0.5
#  0.5 * 2 = 1 + 0

def getBinaryFractional(num):
    if(num == 0):
        return ""
    n = num * 2
    return str(int(n)) + str(getBinaryFractional(n - int(n)))


def binaryEquivalent(n):
    iLst = ""
    iNum = int(n)

    iLst = str(getBinaryIntegral(iNum))
    if(n-int(n) != 0):
        iLst = iLst + "."
        iFrac = str(getBinaryFractional(n-int(n)))
        iLst += iFrac
```

```
    print(iLst)
binaryEquivalent(2.375)
```

```
10.011
```

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 [4]:

```
# to install numpy
import sys
!{sys.executable} -m pip install numpy
```

```
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/91/e7/6c780e612d2
45cca62bc3ba8e263038f7c144a96a54f877f3714a0e8427e/numpy-1.16.2-cp37-cp3
7m-manylinux1_x86_64.whl (17.3MB)
    100% |████████████████████████████████| 17.3MB 119kB/s eta 0:00:01�
�████      | 14.3MB 74.8MB/s eta 0:00:01
Installing collected packages: numpy
Successfully installed numpy-1.16.2
```

In [5]:

```
import numpy as np
from functools import reduce

def cubesum(num):
    lst = []
    checkNum = num
    while(checkNum > 0):
        lst.append(checkNum % 10)
        checkNum = checkNum // 10
    n = len(lst)
    nLst = np.array(list(map(lambda x: x ** n,lst)))
    y = np.sum(nLst)
    return y
```

```python
def isArmStrong(num):
    if(num == cubesum(num)):
        return True
    return False

# def PrintArmstrong(num):
#     if(isArmStrong(num)):
#         print(num," is an Armstrong number")
#     else:
#         print(num," is not an Armstrong number")

# num = int(input("Enter a number: "))

# PrintArmstrong(num)

def PrintArmstrong(numLst):
    for i in numLst:
        if(isArmStrong(i)):
            print(i)


PrintArmstrong(list(range(100,500)))
```

```
153
370
371
407
```

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

In [2]:

```python
def prodDigits(num):
    if(num < 10):
        return num
    return num%10 * prodDigits(num//10)
```

```
prod = prodDigits(3200110)
print(prod)
```

```
0
```

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 [6]:

```
def MDR(num):
    return prodDigits(num)

def MPersistence(num):
    count = 0
    while(num >= 10):
        count += 1
        mdr = MDR(num)
        num = mdr

    print("multiplicative digital root is",mdr)
    print("multiplicative persistence ",count)
MPersistence(86)
```

```
multiplicative digital root is 6
multiplicative persistence  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 [7]:

```
import numpy as np

def sumPDivisor(num):
    return np.sum([i for i in range(1,num) if(num % i == 0)])

divisorLst = sumPDivisor(36)
print(divisorLst)
```

```
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 [10]:

```
def isNumberPerfect(num):
    if(num == sumPDivisor(num)):
        print(num)


n = int(input("Enter the range: "))
print("**********")
print("List of perfect numbers:")
for i in range(1,n + 1):
    isNumberPerfect(i)
```

```
Enter the range: 100
**********
List of perfect numbers:
6
28
```

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 [24]:
```
def getAmicableNumbers(num):
    dic = {}
    for i in range(1,num+1):
        n = sumPDivisor(i)
        lstKeys = dic.keys()
        if(n in lstKeys):
            if(dic[n] == i):
                print(i,n)

        dic[i] = n

num = int(input("Enter a range: "))
getAmicableNumbers(num)
```

```
Enter a range: 10000
284 220
1210 1184
2924 2620
5564 5020
6368 6232
```

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

In [29]:
```
n = int(input("Enter a range to filter odd numbers: "))
oddLst = list(filter(lambda x: (x%2 != 0),list(range(n + 1))))
```

```
print(oddLst)
```

```
Enter a range to filter odd numbers: 40
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37,
39]
```

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

In [2]:

```
lst = list(range(11))

cubeLst = list(map(lambda x: (x ** 3),lst))
print(cubeLst)
```

```
[0, 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 [4]:

```
lst = list(range(10))

cubeEvenLst = list(map(lambda x: (x ** 3),filter(lambda x: (x % 2 == 0
),lst)))
print(cubeEvenLst)
```

```
[0, 8, 64, 216, 512]
```

In [ ]: