```
#necessary imports
import sys
import math
import pdb
import numpy as np
from functools import reduce
```

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

```
def mul_table(n):
  """
  It is assumed that a number will be passed and the multiplication table (upto 20 is printed)
  """
  for i in range(1,21):
    print("{} x {} = {}".format(i,n,(n*i)))

print("Enter n : ")
try:
  n = int(input())
  mul_table(n)
except:
  print('Got Exception : ', sys.exc_info()[0], '. So, enter a proper input.')
```

```
Enter n :
7
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
11 x 7 = 77
12 x 7 = 84
13 x 7 = 91
14 x 7 = 98
15 x 7 = 105
16 x 7 = 112
17 x 7 = 119
18 x 7 = 126
19 x 7 = 133
20 x 7 = 140
```

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

```
till = 1000

def isPrime(n):
```

```
    """
    returns true if a number is prime, after checking the divisibility till the sqrt(n)+1
    """
    end = math.ceil(math.sqrt(n))+1
    for j in range(2, end, 1):
      if n != j and n%j == 0 :
        return False
    return True

def printTwinPrimes():
    """
    Prints the twin prime numbers
    """
    oldPrime = 3
    for i in range(4, till+1):
      if isPrime(i):
        if oldPrime+2 == i:
          print("({},{})".format(oldPrime,i))
        oldPrime = i


printTwinPrimes()
```

```
(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

```
'''
After writing normal logic, following method has been found due to huge run time of bruteforce
Ref : https://www.geeksforgeeks.org/analysis-different-methods-find-prime-number-python
'''
def find_primes_Sieve_method(n):
  """
  Returns the list of prime numbers from 2 to n
  """
  seive = [True for i in range(n+1)]
  p = 2
  while (p*p <= n):
    if seive[p] == True: #prime number
      for i in range(p*p, n+1, p):
        seive[i] = False
    p = p+1

  seive_primes = [index for index, i in enumerate(seive) if i==True and index>1]
  return seive_primes

def find_factors(n, primes):
  """
  finds the factors based on the numbers present in the primes array
  """
  factors = []
  for i in range(len(primes)):
    while n%primes[i] == 0:
      factors.append(primes[i])
      n = n/primes[i]
  return factors

n = 56 #int(input())
primes = find_primes_Sieve_method(math.ceil(n/2)) #it is enough to find prime numbers from 2 to
print('Prime factors of ', n, ' are ', find_factors(n, primes))
```

```
Prime factors of  56  are  [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!

```
def nPr(n,r):
  """
  return nPr
  """
  #now it's the product of (r+1)(r+2) till n after doing number cancellations
  #rather than finding factorials
  fact = 1
```

```
  for i in range(r+1, n+1):
    fact = fact*i
  return fact

def nCr(n,r):
  """
  returns nCr
  """

  #nCr = nC(r-1). After cancellations, there will be only min(r,n-r) present in the numerator a
  r = min(r, n-r)
  num, den = 1,1
  for i in range(n, n-r, -1):
    num = num * i
  for i in range(1, r+1):
    den = den * i
  return num/den

print(nPr(10,6))
print(nPr(10,4))
print(nCr(10,6))
print(nCr(10,4))
```

```
5040
151200
210.0
210.0
```

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

```
def to_binary(n):
  """
  Input : n <br>
  Output : binary of n <br />
  Binary found by dividing the number by 2 and keep going till it reaches zero
  """
  bin = []
  while n > 0:
    bin.append(n%2)
    n = math.floor(n/2)
  return reduce(lambda x,y: str(x)+str(y), list(reversed(bin)))


print(to_binary(1))
print(to_binary(3))
print(to_binary(6))
print(to_binary(14))
print(to_binary(15))
```

```
1
11
110
```

```
1110
1111
```

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.

```
def cubesum(n):
  """
  Returns the sum of cube of digits of a number
  """
  sum = 0
  while n>0:
    sum += (n%10)**3
    n = math.floor(n/10)
  return sum

def isArmstrong(n):
  """
  Returns True : if a number n is equal to it's cube sum
  """
  return cubesum(n) == n

def PrintArmstrong(start, end):
  """
  Prints all the Armstrong numbers in the range [start, end]
  """
  for i in range(start, end+1):
    if(isArmstrong(i)):
      print(i)

PrintArmstrong(1, 1000)
```

```
1
153
370
371
407
```

---

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

```
def prodDigits(n):
  """
  returns product of each digits in a number 'n'
  """
  prod = 1
  while n > 0:
    prod = prod * (n%10)
    n = n//10
  return prod
```

```
print(prodDigits(100))
print(prodDigits(123))
print(prodDigits(12))
print(prodDigits(98))
print(prodDigits(1422))
```

```
0
6
2
72
16
```

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

```
def MDR(n):
  """
  Drill down method of prodDigits() method, till we get one digit result
  """
  prod = prodDigits(n)
  while prod>9:
    prod = prodDigits(prod)
  return prod

def MPersistence(n):
  """
  Find the level of drill down of prodDigits() method, till we get one digit result
  """
  prod = prodDigits(n)
  level = 1
  while prod>9:
    prod = prodDigits(prod)
    level += 1
  return level

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

```
6
2
3
2
```

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

```
def sumPdivisors(n, print_divisors=False):
  """
  Find the divisors of a number by dividing till n/2. Then sums it.

  Parameters
  ----------
  n : int - Input
  print_divisors : bool [Optional] - Used to determine if list of divisors have to be printed
  """
  pdiv = []
  sum = 0;
  for i in range(1, n//2+1):
    if n % i == 0:
      sum += i
      pdiv.append(i)
  if print_divisors:
    print('Proper Divisors of {} are {}'.format(n, pdiv))
  return sum

print(sumPdivisors(36, print_divisors=True))
```

```
Proper Divisors of 36 are [1, 2, 3, 4, 6, 9, 12, 18]
55
```

```
sumPdivisors
```

```
'\n  Find the divisors of a number by dividing till n/2. Then sums it.\n\n
Parameters\n  ----------\n  n : int - Input\n  print_divisors : bool - Use
d to determine if list of divisors have to be printed\n  '
```

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

```
start = 1
end = 100
for i in range(start, end+1):
  if sumPdivisors(i) == i:
    print(i)
```

```
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

```
start = 0
end = 300

result = [False for i in range(start, end+1)]
data = set()

for i in range(start, end+1):
  if result[i] == False:
    pdiv_sum = sumPdivisors(i)
    rev_sum = sumPdivisors(pdiv_sum)

    if (i == rev_sum and result[pdiv_sum] == False and i != pdiv_sum):
      result[i] = True
      result[pdiv_sum] = True
      data.add((pdiv_sum, rev_sum)) #answer returned as set of tuples

print(data)
```

```
{(284, 220)}
```

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

```
data = [i for i in range(0, 100)]
print(data)
filtered_data = list(filter(lambda x: x%2 != 0, data)) #i guessed that filter odd number means
print(filtered_data)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 4
```

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

```
data = [i for i in range(1,25, 2)]
print(data)
cubed_data = list(map(lambda x: x**3, data))
print(cubed_data)
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23]
[1, 27, 125, 343, 729, 1331, 2197, 3375, 4913, 6859, 9261, 12167]
```

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

```
data = [i for i in range(1,25,1)]
```

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

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000, 10648, 13824]
```

✓ 0s completed at 6:20 PM