In [1]:
```
!python --version
```

```
Python 3.4.3
```

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

In [2]:
```
def print_multiplication_table(number, count):
    for i in range(number, number * (count+1), number):
        print(i)

print_multiplication_table(5, 6)
```

```
5
10
15
20
25
30
```

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]:
```
def check_prime(number):
    for i in range(2, (number//2)+1, 1):
        if number % i == 0:
            return False
            break
    else:
        return True
```

In [4]:
```
# get odd series till 1000
odd_numbers_set = [(i, i+2) for i in range(3, 1000, 2) if i+2 < 1000 ]

odd_numbers_set[1][0]
# apply prime function and store information
twin_prime_set = [t for t in odd_numbers_set if check_prime(t[0]) and check_pr

print(twin_prime_set)
```

```
[(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 [10]:
```
import math
def list_prime_factors(number):
    factors = []

    while number % 2 == 0:
        factors.append(2)
        number = number / 2

    for i in range(3, int(math.sqrt(number) + 1), 2):

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

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

    return factors
```

In [13]:
```
print(list_prime_factors(56))
```

```
[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 [4]:
```
def factorial(n):
    acc = 1
    def fact(n, acc):
        if n <= 1:
            return acc
        return fact(n-1, acc * n)
    return fact(n, acc)
```

In [6]:
```
def permutation(n, r):
    return factorial(n)/factorial(n-r)

def combination(n, r):
    return permutation(n, r)/factorial(r)
```

In [7]:
```
print(permutation(5,5))
```

```
120.0
```

In [8]:
```
print(combination(5, 5))
```

```
1.0
```

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

In [13]:
```
def decimal_to_binary(n):
    acc = []
    for i in range(31, -1, -1):
        k = n >> i
        if (k & 1):
            acc.append(1)
        else:
            acc.append(0)
    return acc
```

In [15]:
```
print(decimal_to_binary(10))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 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 [30]:
```
def cubesum(n):
    acc = []
    while n >=10:
        acc.append(n%10)
        n = n//10
    acc.append(n)
    cube_sum = sum([i**3 for i in acc])
    return cube_sum
```

In [32]:
```
def printArmStrong(n):
    print(cubesum(n))

def isArmStrong(n):
    return True if cubesum(n)==n else False
```

In [33]:
```
printArmStrong(153)

print(isArmStrong(153))
```

```
153
True
```

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

number.

In [35]:
```
from functools import reduce

def prodDigits(n):
    acc = []
    while n >= 10:
        acc.append(n%10)
        n = n//10
    acc.append(n)
    return reduce((lambda x,y : x*y), acc)
```

In [37]:
```
print(prodDigits(145))
```

```
20
```

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 [42]:
```
def MDR(n):
    while n >=10:
        n = prodDigits(n)
    return n

def MPeristence(n):
    counter = 0
    while n >= 10:
        n = prodDigits(n)
        counter += 1
    return counter
```

In [43]:
```
print(MDR(86))
print(MPeristence(86))
```

```
6
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 [45]:
```
def get_proper_divisors(n):
    acc = []
    for i in range(1, n):
        if n%i == 0:
            acc.append(i)
    return acc

def sumPdivisors(n):
    return sum(get_proper_divisors(n))
```

In [48]:
```
print(sumPdivisors(36))
```

```
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 [51]:
```
def is_perfect_number(n):
    return True if sumPdivisors(n) == n else False
```

In [52]:
```
print(is_perfect_number(28))
```

```
True
```

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 [56]:
```
def print_amicable_series(n_range):
    acc = []
    for i in range(1, n_range):
        for j in range(1, n_range):
            if sumPdivisors(i) == j and sumPdivisors(j) == i and i != j:
                acc.append((i, j))
    print(acc)
```

In [57]:
```
print_amicable_series(1000)
```

```
[(220, 284), (284, 220)]
```

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

In [63]:
```
def filter_odd_numbers(numbers):
    return list(filter((lambda x: x%2 == 1), numbers))
```

In [64]:
```
print(filter_odd_numbers([2,3,67,34,13]))
```

```
[3, 67, 13]
```

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

a given list

In [66]:
```
def get_cube_list(numbers):
    return list(map((lambda x : x**3), numbers))
```

In [67]:
```
print(get_cube_list([2,4,6]))
```

```
[8, 64, 216]
```

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 [68]:
```
def get_cube_for_even_numbers(numbers):
    return list(map((lambda x: x**3), filter((lambda x: x %2 ==0), numbers)))
```

In [69]:
```
print(get_cube_for_even_numbers([1,2,3,4,5,6,7]))
```

```
[8, 64, 216]
```