Practice Questions on Functions

In [1]:

```
from typing import List
import math
```

Q1. Write a function that inputs a number and prints the multiplication table of that number.

In [2]:

```
def multiplication_table(number: int, n: int) -> None:
    if n > 0:
        for i in range(1, n):
            print("{0} * {1} = {2}".format(number, i, number*i))


number = int(input("Enter the number: "))
multiples_size = int(input("Enter the number of multiples to be shown: "))

multiplication_table(number, multiples_size+1)
```

```
Enter the number: 12
Enter the number of multiples to be shown: 10
12 * 1 = 12
12 * 2 = 24
12 * 3 = 36
12 * 4 = 48
12 * 5 = 60
12 * 6 = 72
12 * 7 = 84
12 * 8 = 96
12 * 9 = 108
12 * 10 = 120
```

Q2. 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 get_primes_array(n: int) -> List:
    primes = [1]*(n+1)
    primes[0] = 0
    primes[1] = 0
    for i in range(0, int(math.sqrt(n))):
        if primes[i] == 1:
            j = 2
            while i*j <= n:
                primes[i*j] = 0
                j += 1
    return primes

def get_primes(n: int) -> int:
    primes_array = get_primes_array(n)
    primes = []
    for i in range(n):
        if primes_array[i] == 1:
            primes.append(i)
    return primes

def twin_primes(n: int) -> List:
    primes = get_primes(n)
    for i in range(len(primes)-1):
        if primes[i+1]-primes[i] == 2:
            print('({0}, {1})'.format(primes[i], primes[i+1]))


twin_primes(1000)
```

```
(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)
```

Ref: To computer primes using Sieve of Eratosthenes (https://www.youtube.com/watch?v=eKp56OLhoQs&list=PL2_aWCzGMAwLL-mEB4ef20f3iqWMGWa25&index=4 (https://www.youtube.com/watch?v=eKp56OLhoQs&list=PL2_aWCzGMAwLL-mEB4ef20f3iqWMGWa25&index=4)) in O(nlog(log n)) time.

Q3. Write a program to find out the prime factors of a number. Example: prime factors of 56 - 2, 2, 2, 7.

In [4]:

```
def prime_factors(n: int) -> str:
    if n < 2:
        return ''

    factors = []
    for i in range(2, n):
        if n%i == 0:
            while n%i == 0:
                n = n/i
                factors.append(i)

    return ','.join(str(i) for i in factors)

print(prime_factors(56))
```

```
2,2,2,7
```

Ref: Finding prime factors of a number (https://www.youtube.com/watch?v=6PDtgHhpCHo&list=PL2_aWCzGMAwLL-mEB4ef20f3iqWMGWa25&index=6 (https://www.youtube.com/watch?v=6PDtgHhpCHo&list=PL2_aWCzGMAwLL-mEB4ef20f3iqWMGWa25&index=6))

Q4. 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]:

```
factorial_map = {}

def factorial(n: int) -> int:
    if n ==0 or n == 1:
        return 1

    if n in factorial_map:
        return factorial_map.get(n)

    factorial_map[n] = n * factorial(n-1)
    return factorial_map.get(n)

def permutations(n: int, r: int) -> int:
    return factorial(n)//factorial(n-r)

def combinations(n:int, r: int) -> int:
    return factorial(n)//(factorial(r)*factorial(n-r))

print(factorial(5))
print(permutations(5, 3))
print(combinations(5, 3))
```

```
120
60
10
```

Q5. Write a function that converts a decimal number to binary number.

In [6]:

```
def decimal_to_binary(n: int, base: int) -> List:
    result = []
    while n > 0:
        n, remainder = divmod(n, base)
        result.append(remainder)
    return result[::-1]

print(decimal_to_binary(125, 2))
```

```
[1, 1, 1, 1, 1, 0, 1]
```

Q6. 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 cube_sum(n: int) -> int:
    if n == 0 or n == 1:
        return n

    total = 0
    while n != 0:
        n, digit = divmod(n, 10)
        total += digit**3
    return total

def is_armstrong(n: int) -> int:
    return n == cube_sum(n)

def print_armstrong(n: int) -> None:
    for i in range(1, n):
        if is_armstrong(i):
            print(i)


print_armstrong(1000)
```

```
1
153
370
371
407
```

Q7. Write a function prodDigits() that inputs a number and returns the product of digits of that number.

In [8]:

```
def prod_digits(n: int) -> int:
    if n == 0 or n == 1:
        return n

    product = 1
    while n != 0:
        n, digit = divmod(n, 10)
        if digit == 0:
            return 0
        product *= digit
    return product

print(prod_digits(10))
print(prod_digits(0))
print(prod_digits(7))
print(prod_digits(547))
```

```
0
0
7
140
```

Q8. 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 multiplicative_digit_root(n: int) -> int:
            while True:
                if len(str(n)) == 1:
                    return n

                n = prod_digits(n)

        def multiplicative_persistence(n: int) -> int:
            count = 0
            while True:
                if len(str(n)) == 1:
                    return count

                n = prod_digits(n)
                count += 1

        print(multiplicative_digit_root(86))
        print(multiplicative_persistence(86))
        print(multiplicative_digit_root(341))
        print(multiplicative_persistence(341))
```

```
6
3
2
2
```

Q9. 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, 12, 18.

In [10]:

```
def prime_divisiors_sum(n: int) -> int:
    divisiors_sum = 0
    if n > 0:
        for i in range(2, int(math.sqrt(n))+1):
            if n%i == 0:
                if i == n/i:
                    divisiors_sum += i
                else:
                    divisiors_sum += ((n//i) + i)
        divisiors_sum += 1
    return divisiors_sum

print(prime_divisiors_sum(36))
```

```
55
```

Q10. 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 perfect_numbers(n: int) -> List:
    perfect_numbers = []
    for i in range(1, n+1):
        if i == prime_divisiors_sum(i):
            perfect_numbers.append(i)
    return perfect_numbers

print(perfect_numbers(30))
```

```
[1, 6, 28]
```

Q11. 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 is_amicable(n1: int, n2: int) -> bool:
    return prime_divisiors_sum(n1) == n2 and n1 == prime_divisiors_sum(n2)

def amicable_number(n: int) -> List:
    amicable_numbers = []
    for i in range(n+1):
        for j in range(i, n+1):
            if i != j:
                if is_amicable(i, j):
                    amicable_numbers.append((i, j))
    return amicable_numbers

print(amicable_number(300))
```

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

Q12. Write a program which can filter odd numbers in a list by using filter function.

In [13]:

```
# Passing a function to the filter
def is_odd(n: int) -> bool:
    return n%2 == 1

def odd_numbers(nums: List) -> List:
    return list(filter(is_odd, nums))

print(odd_numbers([1, 4, 3, 6, 6, 9, 10]))

# Using lamda
print(list(filter(lambda x: x%2 == 1, [1, 4, 3, 6, 6, 9, 10])))
```

```
[1, 3, 9]
[1, 3, 9]
```

Q13. Write a program which can map() to make a list whose elements are cube of elements in a given list.

In [14]:

```
def cube(n: int) -> int:
    return n**3

print(list(map(cube, [1, 2, 3, 4, 5])))
```

```
[1, 8, 27, 64, 125]
```

Q14. 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]:

```
print(list(map(cube, list(filter(lambda x: x%2 == 0, [1, 2, 3, 4, 5, 6])))))
```

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

In [ ]: