PYTHON PROGRAMS

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

In [2]:

```
def multiplication_table(multiplicand):
    print("\n{} Tables:\n".format(multiplicand))
    for multiplier in range(1, 13):
        print("{0} X {1} = {product}".format(multiplicand, multiplier,
product = multiplicand * multiplier))

number = int(input('Enter a number: '))
multiplication_table(number)
```

```
Enter a number: 45

45 Tables:

45 X 1 = 45
45 X 2 = 90
45 X 3 = 135
45 X 4 = 180
45 X 5 = 225
45 X 6 = 270
45 X 7 = 315
45 X 8 = 360
45 X 9 = 405
45 X 10 = 450
45 X 11 = 495
45 X 12 = 540
```

2. Write a program to print twin primes less than 1000.

If 2 consecutive odd numbers are both prime then they are known as twin primes.

```
In [3]: all_primes = []
        for num in range(2,1000):
            prime = True
            for divisor in range(2,num):
                if num % divisor == 0:
                    prime = False
                    break;
            if prime:
                all_primes.append(num)

        twin_primes = []; index = 0

        while index < len(all_primes) - 1:
            if all_primes[index] % 2 == 1 and all_primes[index + 1] % 2 == 1 an
        d all_primes[index + 1] - all_primes[index] == 2:
                twin_primes.append((all_primes[index], all_primes[index + 1]))
            index += 1

        print("Twin Primes less than 1000: ")
        for prime1, prime2 in twin_primes:
            print("{0} and {1}".format(prime1, prime2), end = '\t')
```

```
Twin Primes less than 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
```

3. Write a program to find out the prime factors of a number.

Example: prime factors of 56 - 2, 2, 2, 7

In [111]:

```python
def prime_factor(num):
    factor = 2

    while factor < num:
        if num % factor == 0:
            prime_factor(num // factor)
            break
        factor += 1

    print(factor, end = ' ')

num = int(input('Enter a number: '))
print("The prime factors of the number {} are: ".format(num), end = ' '
)
prime_factor(num)
```

```
Enter a number: 56
The prime factors of the number 56 are:  7 2 2 2
```

4. Write a program to implement these formula 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 [112]:

```
def fact(f):
    return 1 if f == 1 else f * fact(f - 1)

print('-' * 25 + 'Permutations & combinations' + '-' * 25)
n = int(input('Enter n: '))
r = int(input('Enter r: '))

p = fact(n) // fact(n - r)
c = p // fact(r)

print('Number of permutations of {n} objects taken {r} at a time: {p}'.
format(n = n, r = r, p = p))
print('Number of combinations of {n} objects taken {r} at a time: {c}'.
format(n = n,r = r, c = c))
```

```
-------------------------Permutations & combinations-------------------
------
Enter n: 4
Enter r: 3
Number of permutations of 4 objects taken 3 at a time: 24
Number of combinations of 4 objects taken 3 at a time: 4
```

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

In [101]:

```
def get_binary(decimal):
    return 0 if decimal == 0 else decimal % 2 + 10 * get_binary(decimal
 // 2)

num = int(input('Enter a decimal number: '))
print("The binary number is {}".format(get_binary(num)))
```

```
Enter a decimal number: 50
The binary number is 110010
```

6. Write a function cubesum() that accepts an integer and returns the sum of the cubes of individual digits of that number. Using this cubesum() function to make function printArmstrong and isArmstrong to print Armstrong numbers and to find whether is an Armstrong number.

In [113]:

```python
def cubesum(num):
    return 0 if num == 0 else pow(num % 10,3) + cubesum(num // 10)

def isArmstrong(num):
    return True if num == cubesum(num) else False

def printArmstrong():
    print("\nArmstrong numbers:")
    print(list(filter(isArmstrong,list(range(1000)))))

num = int(input("Enter a number: "))
if isArmstrong(num):
    print("{} is an Armstrong number".format(num))
else:
    print("{} is not an Armstrong number".format(num))
printArmstrong()
```

```
Enter a number: 153
153 is an Armstrong number

Armstrong numbers:
[0, 1, 153, 370, 371, 407]
```

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

number

In [114]:

```
def prodDigits(num):
    return 1 if num == 0 else num % 10 * prodDigits(num // 10)

number = int(input("Enter  a number: "))
print("Product of digits of the number {0} is {1}".format(number,prodDi
gits(number)))
```

```
Enter  a number: 47
Product of digits of the number 47 is 28
```

8. Using the function prodDigits(), write functions MDR() and MPersistence() that inputs a number and returns its multiplicative digital root and multiplicative persistence respectively.

If all digits of a number n ar multiplied by each other repeating with the product, thhe one digit number obtained at last is called the multiplicative digit root of n. The number of times digits need to be multiplied to reach one digit is the multiplicative persistence of n.

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

In [115]:

```
def MDR(num):
    return num if num >= 0 and num <= 9 else MDR(prodDigits(num))
def MPersistence(num):
    return 0 if num >=0 and num <= 9 else 1 + MPersistence(prodDigits(n
um))

number = int(input("Enter a number: "))
print("The Multiplicative Digital Root of {} is {}".format(number, MDR(
number)))
print("The Multiplicative Persistence of {} is {}".format(number, MPers
istence(number)))
```

```
Enter a number: 86
The Multiplicative Digital Root of 86 is 6
The Multiplicative Persistence of 86 is 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 [116]:

```
def sumPdivisors(num):
    return sum(list(filter(lambda pdivisors : num % pdivisors == 0, lis
t(range(1,num)))))

number = int(input("Enter a number: "))
print("The sum of proper divisors of the number {num} is {sum_pd}".form
at(num = number, sum_pd = sumPdivisors(number)))
```

```
Enter a number: 36
The sum of proper divisors of the number 36 is 55
```

10. Write a program to print all the perfect numbers in a given range.

A number is called perfect if the sum of proper divisors of that number is equal to that number. For example 28 is perfect number, since 1 + 2 + 4 + 7 + 14 = 28.

In [51]:

```
start = int(input("Enter the starting number: "))
end = int(input("Enter the ending number: "))
```

```
perfect_numbers = list(filter(lambda num : num == sumPdivisors(num), list(range(start, end + 1))))

if perfect_numbers:
    print("All the perfect numbers between {0} and {1} are {2}.".format(start, end, perfect_numbers))
else:
    print("There are no perfect numbers between the provided range {} and {}.".format(start, end))
```

```
Enter the starting number: 1
Enter the ending number: 1000
All the perfect numbers between 1 and 1000 are [6, 28, 496].
```

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

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 = 284. Sum of proper divisors of 284 = 220.

In [117]:

```
def sumPdivisors(num):
    return sum(list(filter(lambda pdivisors : num % pdivisors == 0, list(range(1, num)))))

def amicable_numbers(start, end):
    amicable = {}

    for num in range(start, end + 1):
        aliquot = sumPdivisors(num)
        if sumPdivisors(aliquot) == num and num != aliquot and aliquot not in amicable:
            amicable[num] = aliquot
    return amicable

start = int(input("Enter the range's starting number: "))
end = int(input("Enter the range's ending number: "))
amicable = amicable_numbers(start, end)

if amicable:
    print("\nPairs of Amicable numbers in the range {} - {} are: ".format(start, end))
    for number, aliquot in amicable.items():
        print(number, aliquot)
else:
    print("\nThere are no amicable numbers in the provided range {} - {}.".format(start, end))
```

```
Enter the range's starting number: 200
Enter the range's ending number: 3000

Pairs of Amicable numbers in the range 200 - 3000 are:
220 284
1184 1210
2620 2924
```

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

In [119]:

```
numbers_list = [int(num) for num in input("Enter a list of numbers seperated by space: ").split()]
filter_odds = list(filter(lambda x : x % 2 != 1, numbers_list))

print("\nList after filtering the odds: {}".format(filter_odds))
```

```
Enter a list of numbers seperated by space: 23 45 65 44 78 22

List after filtering the odds: [44, 78, 22]
```

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

In [120]:

```
numbers_list = [int(num) for num in input("Enter a list of numbers seperated by space: ").split()]
cubes_list = list(map(lambda num : pow(num, 3), numbers_list))

print("\nThe cubes of the numbers {} are {} respectively.".format(numbers_list, cubes_list))
```

```
Enter a list of numbers seperated by space: 6 7 8 9 11

The cubes of the numbers [6, 7, 8, 9, 11] are [216, 343, 512, 729, 1331] respectively.
```

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

In [121]:

```
numbers = [int(num) for num in input("Enter a list of numbers seperated by space: ").split()]

evens = list(filter(lambda num : num % 2 == 0, numbers))
even_cubes = list(map(lambda num : pow(num, 3), evens))

print("\nThe cubes of the numbers {} are {} respectively.".format(evens, even_cubes))
```

```
Enter a list of numbers seperated by space: 6 7 8 9 11 12 13 14

The cubes of the numbers [6, 8, 12, 14] are [216, 512, 1728, 2744] respectively.
```