Pyhton Basics:

- Factorial of a given number as a procedure

```
* Factorial(5) -> 120

* Factorial (6) -> 720
```

In [2]:
```
def fact(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    print(fact)

n=int(input())
fact(n)

```

```
6
720
```

- Procedure to generate multiplication tables.

```
MT(3, 5, 7)->   3 X 5 = 15

                3 X 6 = 18

                3 X 7 = 21
```

In [11]:
```
def mulTable(n):
    for k in range(3,n+5):
        num=n*k
        print(n,"x",k,"=",num)

n=int(input())
mulTable(n)

```

```
3
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
```

In [ ]:
```

```

- Procedure to print the list of factors of a given number.

```
FactorList(6)  ->  1 2 3 6

FactorList (9) ->   1 3 9

FactorList (19) ->  1  19
```

In [16]:
```
def factorList(n):
    for i in range(1,n+1):
        if n%i==0:
            print(i,end=" ")

n=int(input())
factorList(n)
```

```
10
1 2 5 10
```

- Procedure to check if a given number is Prime and returns a Boolean value.

```
IsPrime(7) -> True

IsPrime(9) ->  False
```

In [31]:
```
def isPrime(n):
    if n>1:
        for i in range(2,n):
            if n%i==0:
                print("False")
            else:
                print("True")

n=int(input())
isPrime(n)
```

```
3
True
```

- Procedure to count the number of digits in a given number.

```
CountDigits(123456) ->  6

CountDigits (0) ->  1
```

In [42]:
```
def digitCount(n):
    c=0
    for i in range(1,n+1):
        c +=1
        print(c)


n=int(input())
digitCount(n)

```

```
4
1
2
3
4
```

```
1

```

- Procedure to check if a given number is a Perfect Number. (Perfect number is a number for which the sum of all it's divisors is equal to the number itself)

```
IsPerfect(3) -> False

IsPerfect (6) -> True
```

In [46]:
```
def isPerfect(n):
    sum1 = 0
    for i in range(1, n):
        if(n % i == 0):
            sum1 = sum1 + i
    if (sum1 == n):
        print("True")
    else:
        print("False")

n= int(input("Enter any number: "))
isPerfect(n)

```

```
Enter any number: 6
True
```

Type *Markdown* and LaTeX: α^2

- Procedure to generate the first N perfect numbers.

```
GeneratePerfect(2) ->  6   28

GeneratePerfect (4) ->  6   28   496    8128
```

In [56]:
```
def nPerfectNums(n):
    s=0
    for i in range(1,n):
        if n%i==0:
            s +=i
    if s==n:
        print("True")
    else:
        print("False")


n=int(input())
nPerfectNums(n)
```

```
6
True
```

Type *Markdown* and LaTeX: α^2

- Design a procedure calculate the maximum, minimum and average of N numbers
  - data(a[1,2,3,4,5]) -> Max = 5, Min = 1, Avg = 3

In [81]:
```
def nNums(n):
    print("Min = ",min(a))
    print("Max = ",max(a))
    s=0
    for i in a:
        s +=i
    print("Avg = ",s//i)
a=[1,2,3,4,5]
nNums(n)

```

```
Min =  1
Max =  5
Avg =  3
```

Type *Markdown* and LaTeX: α^2

- Design a procedure to determine if a given string is a Palindrome
  - Palindrome("racecar") -> True
  - Palindrome("raptor") -> False

In [85]:
```
def palindrome(s):
    for i in s:
        if s==s[::-1]:
            return True
        else:
            return False
s=input()
palindrome(s)
```

```
raptor
```

Out[85]:
```
False
```

Type *Markdown* and LaTeX: α^2

- Design a procedure to calculate the squareroot of a number "without using the math function sqrt".

```
* Squareroot(36) -> 6
```

In [101]:
```
def sqrtnum(n):
    m=(1/2)
    s=(n)**(m)
    print(s)

n=int(input())
sqrtnum(n)
```

```
36
6.0
```

Type *Markdown* and LaTeX: α^2

- Design a procedure to determine the frequency count of numbers in a given list

```
* Frequency( a[1,3,2,1] ) -> 1 : 2, 2 : 1, 3 : 1
```

In [103]:
```
def CountFrequency(my_list):

    # Creating an empty dictionary
    freq = {}
    for item in my_list:
        if (item in freq):
            freq[item] += 1
        else:
            freq[item] = 1

    for key, value in freq.items():
        print ("% d : % d"%(key, value))

my_list=[1,3,2,1]
CountFrequency(my_list)

```

```
1 :  2
3 :  1
2 :  1
```

Type *Markdown* and LaTeX: α^2

- Design a procedure to perform Linear search on list of N unsorted unique numbers. It take an array and the key element to be searched and returns the index of the element of key element if found. Else returns -1
  - LinearSearch(a[5,4,3,2,1], 4) -> 2
  - LinearSearch(a[90, 123, 324, 21, 56], 22) -> -1

In [112]:
```
def search(arr, x):

    for i in range(len(arr)):

        if arr[i] == x:
            return i+1

    return -1

arr=[5,4,3,2,1]
x=int(input())
#LinearSearch( a[90, 123, 324, 21, 56], 22) -> -1
search(arr,x)
```

```
4
```

Out[112]: 2

In [ ]:
```

```