IOTA ACADEMY
Become Job Ready...

Recurison

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

every recursive function must have at least two cases :

- Recursive Case # The recursive case is the part where the function calls on itself

- Base Case (or the stoping case) - ALWAYS REQUIRED # for every recursion we have boundary case

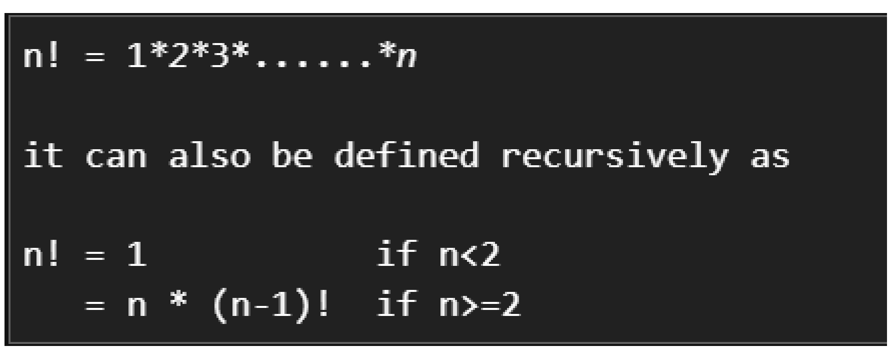
Example:

In [6]:
```python
# program to calculate factorial using recursion

def factorial(num):
    if num==1:  # Base case
        return 1
    else:
        return num*factorial(num-1)  #5*4*3*2*1 Recursive case


factorial(5)
```
Out[6]:
```
120
```

Computing Factorial Recursively



```
n! = 1*2*3*......*n

it can also be defined recursively as

n! = 1           if n<2
   = n * (n-1)!  if n>=2
```

In [7]:
```python
# program to calculate factorial using iterative code

num = int(input("Enter a number: "))
fact = 1

for i in range(1,num+1):
    fact =fact * i
print(fact)
```
```
120
```

In [9]:
```python
# Program to show the use of recursion in calculation of power e,g.,a^b

def power(a,b):

    if b==0:   # Base case:1
        return 1

    else:
        return a * power(a,b-1)


power(2,4)
```
Out[9]:
```
16
```

above base case condition will never be true for a negative value of b and if we put negative value of b infinite recursion will occur

In [10]:
```python
def power(a,b):

    if b<=0:   # Base case:2 # Now it takes care of negative values of b
        return 1

    else:
        return a * power(a,b-1)


power(2,0)
```
Out[10]:
```
1
```

In [11]:
```python
# program to calculate a^b using iterative code

def power(a,b):
    res = 1
    if b==0:

        return 1

    else:
        for i in range(b):
            res = res * a

    return res


power(4,3)
```
Out[11]:
```
64
```

In [12]:
```python
# recursion fuction to print the string backwards

def strback(sg,n):


    if n==0:
        return sg[n]

    return sg[n] + strback(sg,n-1)

strback("iota",len("iota")-1)
```
Out[12]:
```
'atoi'
```

In [13]:
```python
# code to calculate age if birth year is given.

usersayyes='y'
while usersayyes=='y':
    name = input("Enter your name")
    yob = int(input("Enter your birth year"))

    yearsafter = int(input("do you want to know your age after x years: Enter x:"))
    print(f"Hi {name} your age after {yearsafter} will be {2023-yob+yearsafter}")

    usersayyes=input("Do you want to try again? y/n")
```
```
Hi Rahul your age after 0 will be 26
```

Advantages

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Exercise:

1. Python program to display the fibonacci sequence up to n-th term using recursive function
2. Write a recursive function that compute the sum of number from 1 to n; get the value of last numbers(n) from user.
3. Make a login portal which allows maximum 3 wrong attempts.
   - Use pre-defined username and password to match inputs.
   - usr = 'admin' and password='12345'
4. Write a recursive code to find the multiplication of all elements of a list
5. Write a function that takes a number and tests if it is prime number using recursion.

Great Job!