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.

Example:

In [11]:
```
def factorial(n):
    fact=1

    for i in range(1,n+1):
        fact=fact*i

    return fact
```

In [20]:
```
def divide_by_2(num,count):
    if num>10:
        count = count+1
        return divide_by_2(num/2,count)
    else:
        return (num,count)

divide_by_2(100,0)
```

Out[20]: (6.25, 4)

In [24]:
```
def admin_portal():
    username = input("Enter Username: ")
    password = input("Enter Password: ")
    if username=="hemant" and password=="iota@123":
        print("Login Successful")
    else:
        print("Wrong details")
        admin_portal()
```

In [25]:
```
admin_portal()
```

```
Wrong details

Login Successful
```

In [17]:
```
#python program to print factorial of a number using recurion

def factorial(num):
    """
    This is a recursive function to find the factorial of a given number
    """
    if num == 1:
        return 1
    else:
        a = num * factorial(num-1)
        return a

num = 5

factorial(5)
```

Out[17]: 120

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.

Python program to display the fibonacci sequence up to n-th term using recursive function

In [ ]:
```
def fibonacci(num):
    return num if num <= 1 else fibonacci(num-1) + fibonacci(num-2)

nterms = 10
print("Fibonacci sequence")
for num in range(nterms):
    print(fibonacci(num))
```

```
Fibonacci sequence
0
1
1
2
3
5
8
13
21
34
```

That's Great!