Explaining recursions

```
def rec_sum(n):
    if n<=1:
        return 1
    else:
        return n+ rec_sum(n-1).

>> print(rec_sum(4)).
```

lets understand this simple recursion

>> n = 4.

function rec_sum(4) will be executed.

n is not <= 1

∴ it will go to else part.

there it will return

4+ call rec_sum(3).

>> Again, rec_sum(3) will be executed

n is not <= 1.

∴ it will go to else part

4 already in stack

it will return 4+3+ rec_sum(2).

>> Again, rec_sum(2) will be executed

n is not <= 1.

∴ it will go to else part

4 & 3 already in stack

it will return 4+3+2+ rec_sum(1).

>> Again, rec_sum(1) will be executed

n is <= 1

it will return 1.

⇒ finally, 4+3+2+1 = 10 will be returned.

PAGE NO.:
DATE: / /

② Factorial of a number :-

Suppose, we simply want factorial of
$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 \Rightarrow 6 * (6-1)!$
Now, we want factorial of 5
$5! = 5 \times 4 \times 3 \times 2 \times 1 \Rightarrow 5 \times (5-1)!$
Clearly, it shows.
factorial(n) = n * factorial(n-1).

```
def factorial(n):
    if n<=1:
        return 1
    else:
        return n * factorial(n-1).
print(factorial(5)).
```

>> function factorial(5) will be executed.
>> n is not <=1.
it will move to else part
it will return 5* factorial(4).
>> Again factorial(4) will be executed.
>> n is not <=1
it will move to else part
it will return 5* 4* factorial(3).
>> Again factorial(3) will be executed
n is not <=1
it will move to else part
it will return 5* 4* 3* factorial(2).
>> Again factorial(2) will be executed
n is not <=1
it will move to else part
it will return 5* 4* 3* 2* factorial(1).

PAGE NO.:
DATE: / /

>> Again factorial(1) will be executed
As n is <=1
it will return 5 * 4 * 3 * 2 * 1
⇒ 120.

Scanned with OKEN Scanner

>> Again factorial(1) will be executed
As n is <=1
it will return 5 * 4 * 3 * 2 * 1
⇒ 120.

③ fibonacci series

1 1 2 3 5 8 13 21 34 55 .....

Here, if we see the pattern

$$\boxed{f(n) = f(n-1) + f(n-2)}$$

Using this Concept we can write a recursive function.

```
def fibo(n):
    if n<=1:
        return 1.
    else:
        return (fibo(n-1) + fibo(n-2)).

n_terms = int(input(" Enter terms in Series: ")).
if n_terms < 0:
    print(" Invalid input").
else:
    print(" Your fibonacci series is: ").

for i in range(n_terms):
    print(fibo(i)).
```

PAGE NO.:
DATE: / /

Demonstration

Suppose n terms = 6
It will go to else part & print
the fibonacci series:

Then loop will run from 0, 5 as n terms = 6.

Cases

when i = 0;

fibo(0)
returns & prints 1.

when i = 1;
fibo(1)
returns & prints 1

when i = 2;
fibo(2)
As n is not <= 1
it will go to else part.

fibo(2)
fibo(2-1)
fibo(1).

again fibo(1) will be executed & it will return
1 at return (fibo(n-1) + fibo(n-2))
return (1 + fibo(n-2))

now, fibo(n-2)
fibo(2-2)
fibo(0) will ~~be executed~~ executed & it will also
return 1 at return (1 + fibo(n-2))
return (1 + 1) ⇒ 2 will be returned & printed

PAGE NO.:
DATE: / /

Now, for fibo(3) [fibo(n-2)]
>> fibo(1)
Now, 1 will be returned
at fibo(3) [fibo(n-2)] part.
Now, fibo(3) will return 2+1 => 3
to fibo(4) [fibo (n-1)] part.

Now, fibo(4) [fibo(n-2)] part.

fibo(4-2)
>> fibo(2).
>> else part will be executed
>> fibo(2-1)
=> fibo(1).
will return 1 at fibo(2) [fibo(n-1)].
Now, fibo(2) [fibo(n-2)] part
>> fibo(0) will return 1 at fibo(2)[fibo(n-2)].
∴ fibo(2) will return 2 to fibo(4) [fibo(n-2)]

Now, we have fibo(n-1) at 4 = 3
fib(n-2) at 4 = 2
=> 5 will be returned & printed.

PAGE NO.:
DATE: / /

>> when i = 3;
fibo(3)
>> n is not <= 1;
it will move to else part.
>> fibo(n-1)
>> fibo(3-1)
>> fibo(2).
Again fibo(2) will be executed.
n is not ~~equal~~ <= 1:
it will move to else part:
fibo(n-1)
>> fibo(2-1)
>> fibo(1). will be executed & will return 1
for fibo(2) [fib(n-1)].
Now going back for fibo(2) => fibo(n-2)
= fibo(0) will return
1 at fibo(n-2).
∴ for fibo(2) 1+1 = 2 will be
returned
& fibo(2) will return this 2
to fibo(3) [fibo(3-1)] part
Now, fibo(3) => fibo(3-2) will be
executed
=> fibo(1) will return 1.
∴ at return (fibo(n-1) + fibo(n-2).)
=> 2 + 1
=> 3 will be returned &
printed.

when $i = 4$;
fibo(4)
n is not <= 1
: it will move to
else
(i) fibo(n-1)
>> fibo(4-1)
>> fibo(3), again function
will be called
>> fibo(3)
n is not <= 1
else part will be executed
>> fibo(n-1)
>> fibo(3-1)
>> fibo(2), again fibo(n)
will be called
>> fibo (2-1)
>> fibo (1), Now when
function will be called
"if" will execute, it
will return 1, for
fibo (2) [fibo (n-1)] part.
>> Now, for fibo(n-2)
>> fibo (2-2)
fibo (0).

function will be called
"if" will execute,
1 will be returned
to fibo(2) [fo fib(n-2)] part.
fibo(2) will return 1+1 = 2
to fibo(3) [fibo(n-1) part].