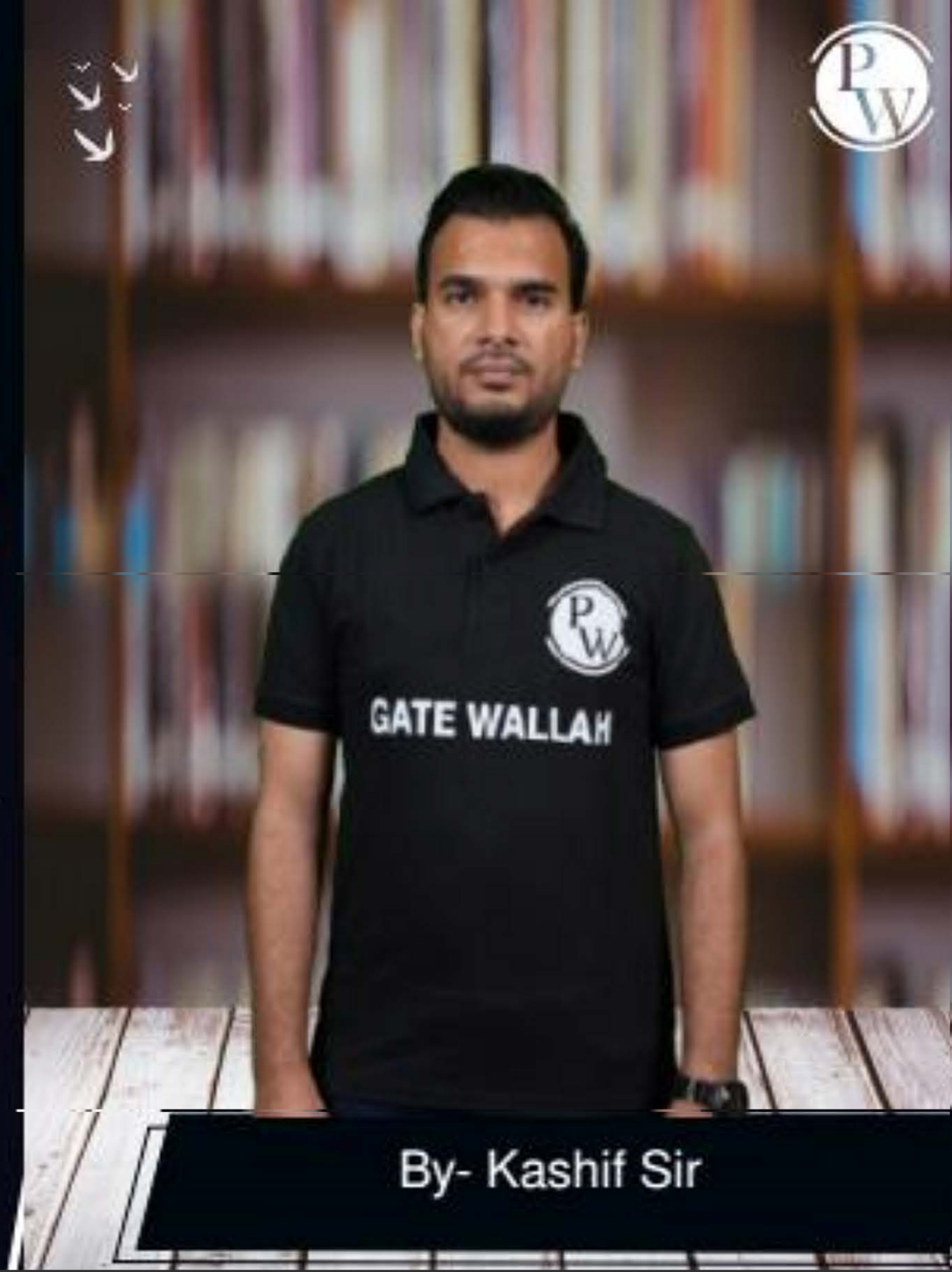
DS & AI

Python for Data Science

Functions and Functional Programming

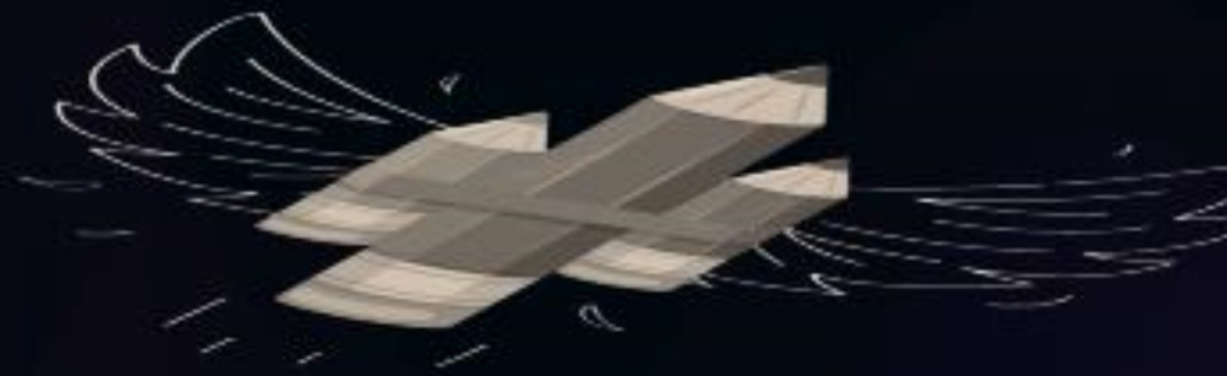


Lecture No. 02

By- Kashif Sir

RECAP

- → Function definition, syntax
- → Positional arguments, Keyword arguments
- → Default parameter
- → Any no of positional arguments & any no. of keyword arguments
- → positional only parameters
- → Keyword only parameters



Topics to be Covered

- **Functions and Functional Programming**

FUNCTIONS

Local variable

A local variable is a variable which is initialized inside a function and belong to that function. It cannot be accessed outside that function.

```
def fun1( ):
    a = 100
    print(a)
```

Local variable

a → 100

→ fun1( ) 100

→ print(a) → Error name is not defined

Global Variables are those which are defined outside any function and can be accessed throughout the program (i.e both inside & outside the function)

```
c = 100 ⟹ global variable
def fun2( ):
    print(c)

fun2( )        100
print(c)       100
```

```
a = 1000
def fun3( ):
    a = 10
    print(a)

fun3( )        10
print(a)       1000
```

```
d = 500
def func( ):
    d += 500
    print(d)
```

d = 10

```
print(d)
func( )
d += 500
print(d)
```

500
1000
1500

```
d = d+500
```

error

By defaut global variables can be accessed inside the function but cannot be modified inside the function.

Use of global Keyword

```
d = 500
def func( ):
    global d
    d += 500
    print(d)
```

global keyword is used inside a function to refer to a global variable

a = 100

global

def chng( ):

a = 50 → Local variable

chng( )

print(a) = 100

Accessing

Initializing

modifying

a = ~~100~~ 50

def chng( ):

global a

a = 50 → global variable

chng( )

print(a) => 50

def fun3( ):

global z

z = 100

fun3( )

print(z) = 100

=>

Nested Function

A Nested function is a function defined inside another function. The outer function can call the inner function. The inner function can access the variables and arguments of the outer function.

```
def fun1():
    a = 100
    b = 200
    def fun2():
        print(a,b)
    fun2()
```

Closure

Inner function remembers variables of outer function after it returns fun1()

100, 200

```
def outer():
    x = 5
    def inner():
        print(x)
    return inner

f = outer()
f()
```

5

```
def multiplier(factor):
    def multiply-by(x):
        return x * factor
    return multiply-by
```

2

```
double = multiplier(2)
triple = multiplier(3)
```

```
print(double(5)) = 10
print(triple(5)) = 15
```

```
def multiply-by(x)
    x = x*2
    return x*2
```

```
def multiply-by(x)
    x = x*2
    return x*3
```

```
def outer():
    a = 1
    def inner():
        nonlocal a
        a += 1
        print(a)
    inner()
outer()
```

nonlocal keyword say that a is neither local nor global

nonlocal

```
a = a + 1
```

error

Recursion

When a function calls itself it is called Recursion, and the function is called Recursive function.

```
def fun(n):
    if n>0:
        print(n)
        fun(n-1)
```

```
def fun(n):
    print(n)
    fun(n-1)
```

```
fun(10)
  ↓
fun(9)
  ↓
fun(8)
  ↓
fun(7)
  ↓
  ⋮
  1
```

10
9
8
7
6
5
⋮

fun(10)
↓
fun(9)
↓
fun(8)
↓
⋮
fun(0)

10
9
8
7
6
5
4
3
2
1

fun(10)

A base condition is necessary in recursive functions.

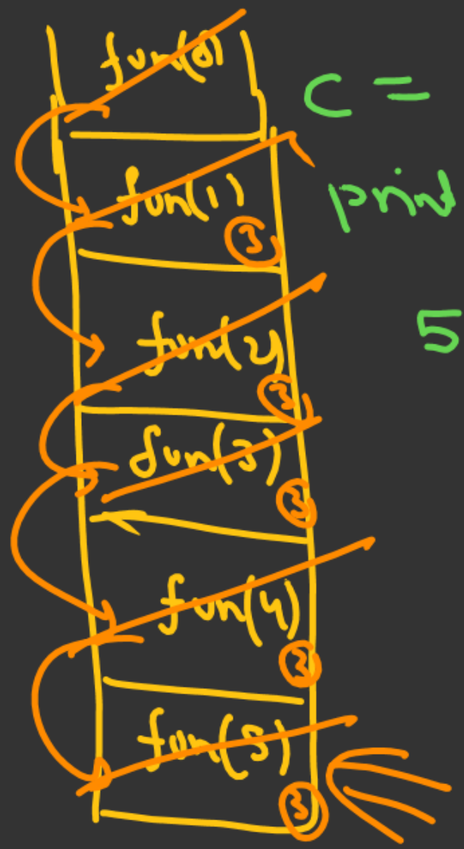
```
def fun(n):
    if n>0:
        print(n)
        fun(n-1)
        print(n)
```

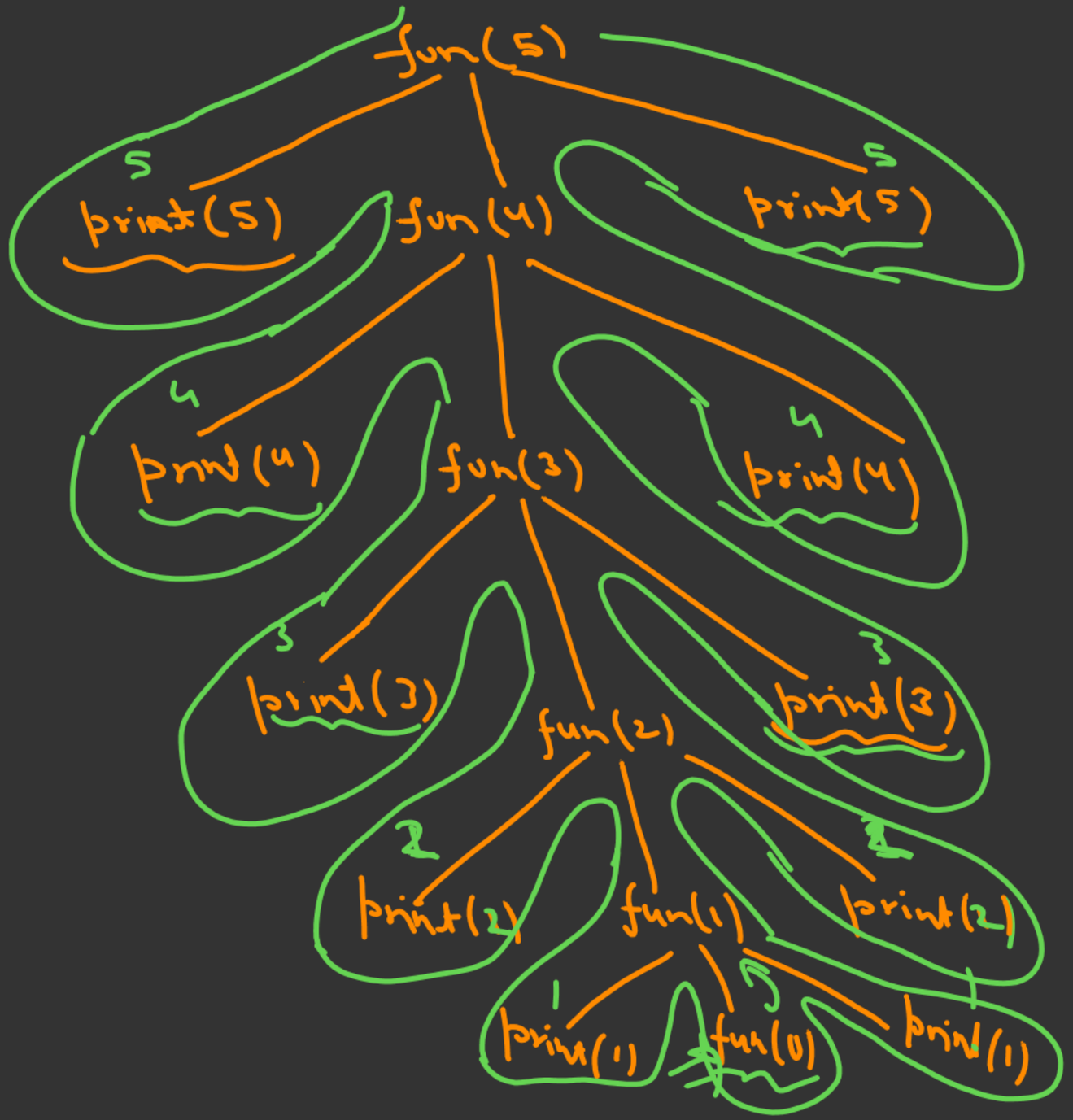
c = fun(5)

print(c)

5 4 3 2 1 1 2 3 4 5

5 4 3 2 1 1 2 3 4 5





```
def total(n):
    if n==1:
        return 1
    return n+total(n-1)
```

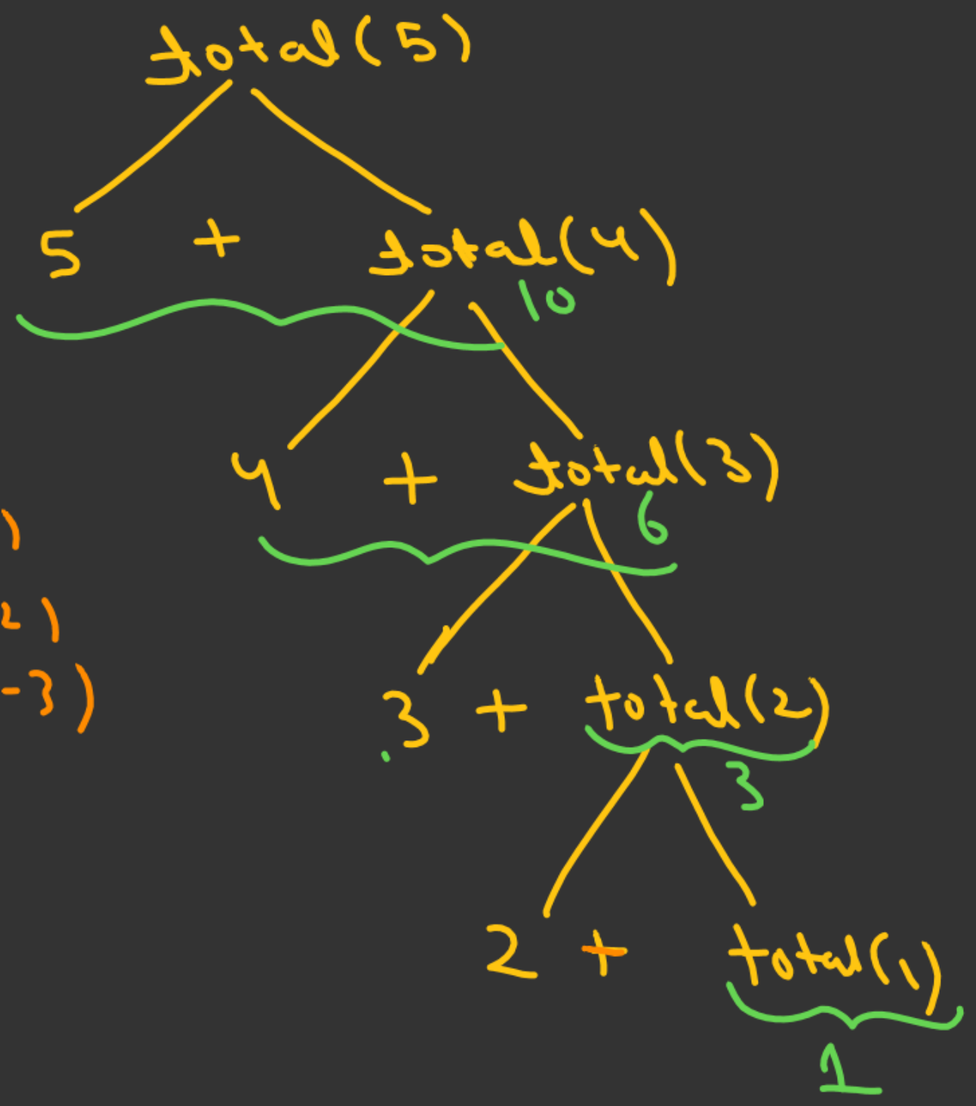
```
print(total(5))
```

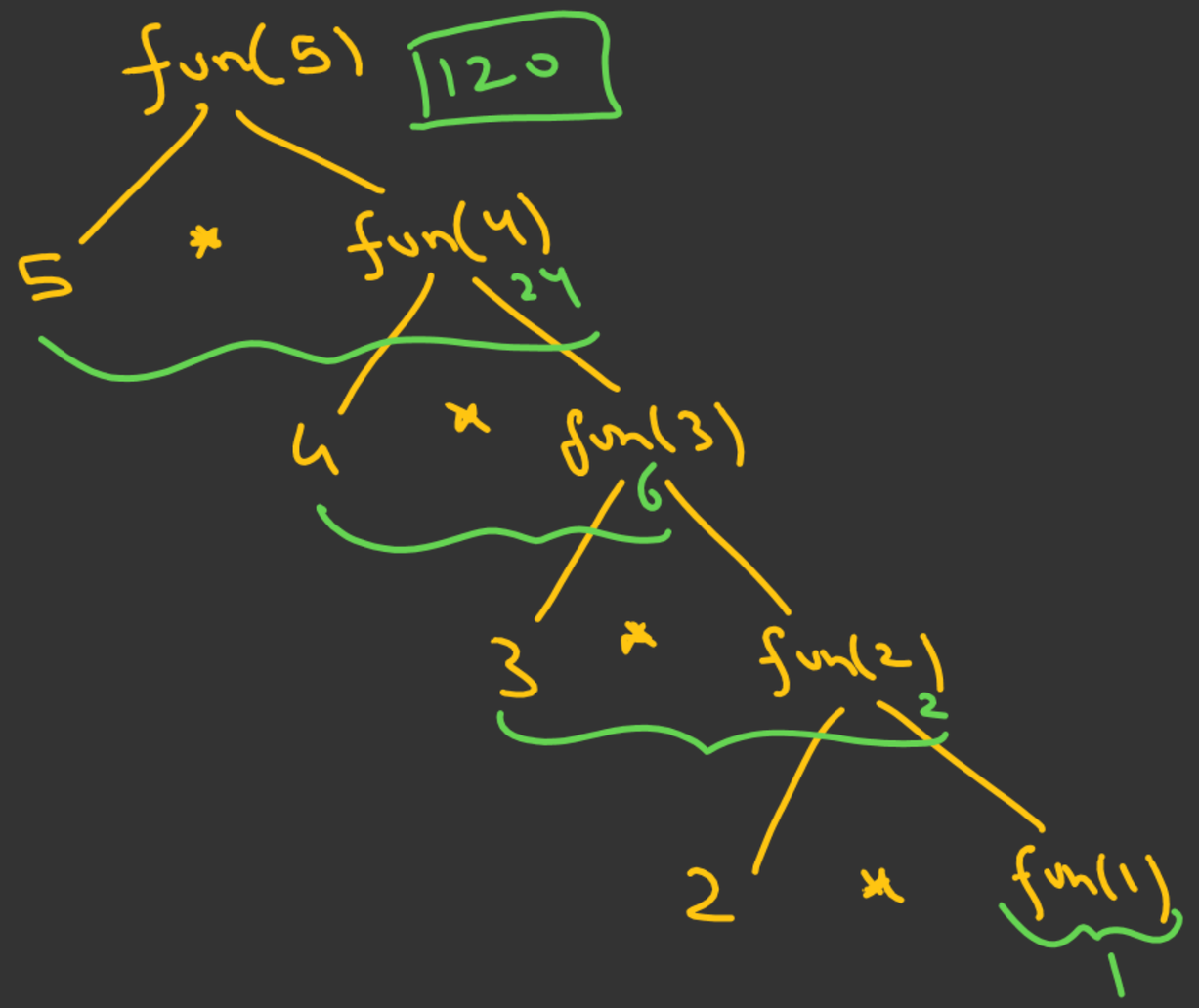
15

```
fun(n)
    fun(n-1)
    fun(n-2)
    fun(n-3)
```

```
def fun(n)
    if n==1:
        return 1
    return n*fun(n-1)
```

```
fun(5)
```




fun(5)
120
5
*
fun(4)
24
4
*
fun(3)
6
3
*
fun(2)
2
2
*
fun(1)
1

Types of Recursion :

- Finite Recursion vs Infinite recursion
- Head Recursion vs Tail recursion
- Direct Recursion vs Indirect recursion

```
def head(n)
    if n > 0
        head(n-1)
        print(n)
```

```
def tail(n)
    print(n)
    if n <= 0
        return
    tail(n-1)
```

recursion opⁿ will be the last opⁿ of that function

Direct Recursion vs Indirect Recursion

```
def fun(n):
    if n <= 1:
        return 1
    return fun(n-1) + fun(n-2)
```

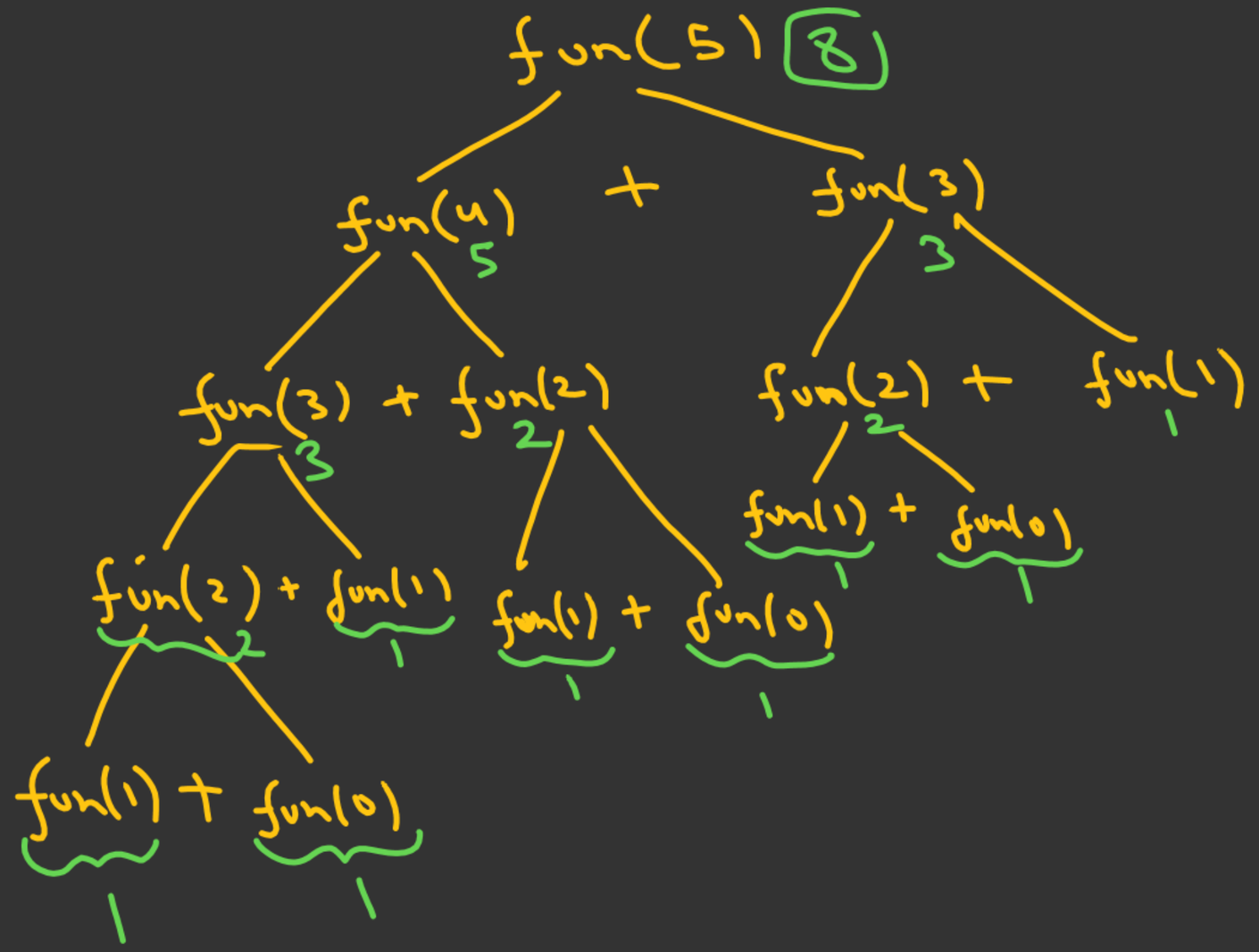
Direct Recursion

fun(5)

```
def A():
    B()

def B():
    A()
```

Indirect Recursion


fun(5) 8
fun(4) 5
+
fun(3) 3
fun(3) 3
+
fun(2) 2
fun(2) 2
+
fun(1) 1
fun(2)
+
fun(1)
1
fun(1)
+
fun(0)
1
1
fun(1)
+
fun(0)
1
1
fun(1)
+
fun(0)
1
1

THANK - YOU