GATE

CRASH COURSE

Data Science & AI

Subject

Python-for data science

Functions, Recursion in Python

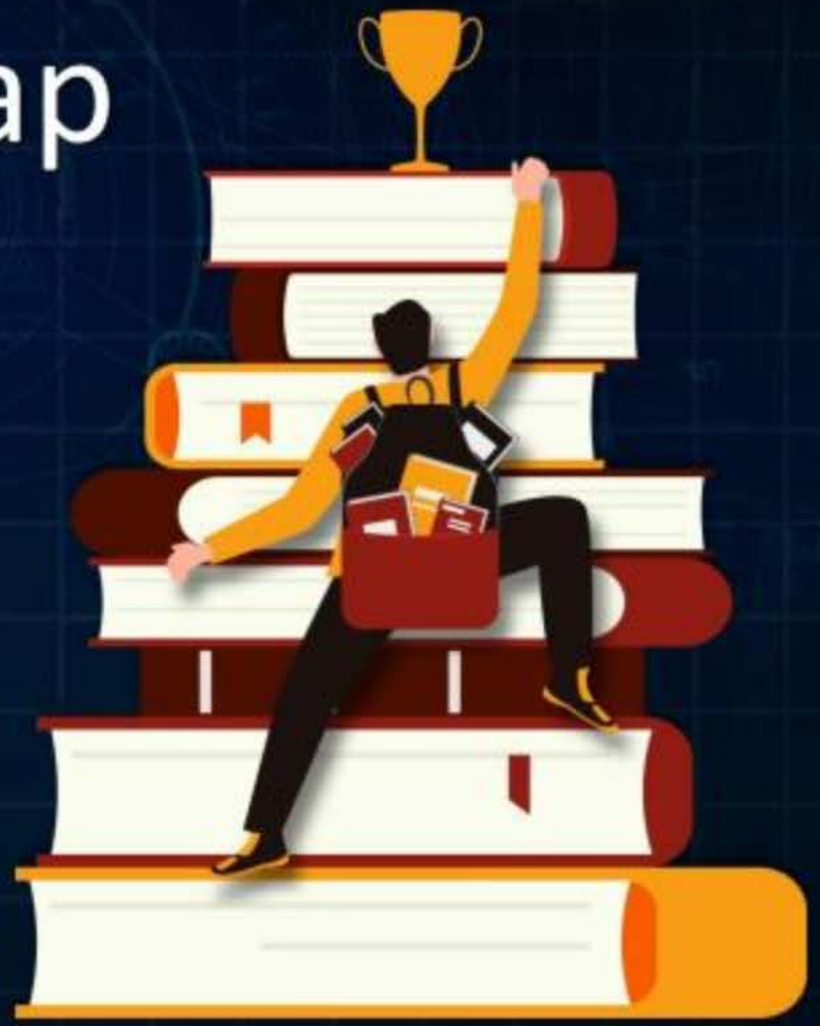
Lect No. 06

By – Satya Sir

Last Class

Quick Recap

1. Homework Questions Solution
2. Nested Lists
3. String Methods
4. Tuples
5. Sets and Examples



Topics to be covered

1. Homework Questions Solution
2. Set Methods and Operations
3. Dictionaries
4. Functions, Recursion
5. Examples

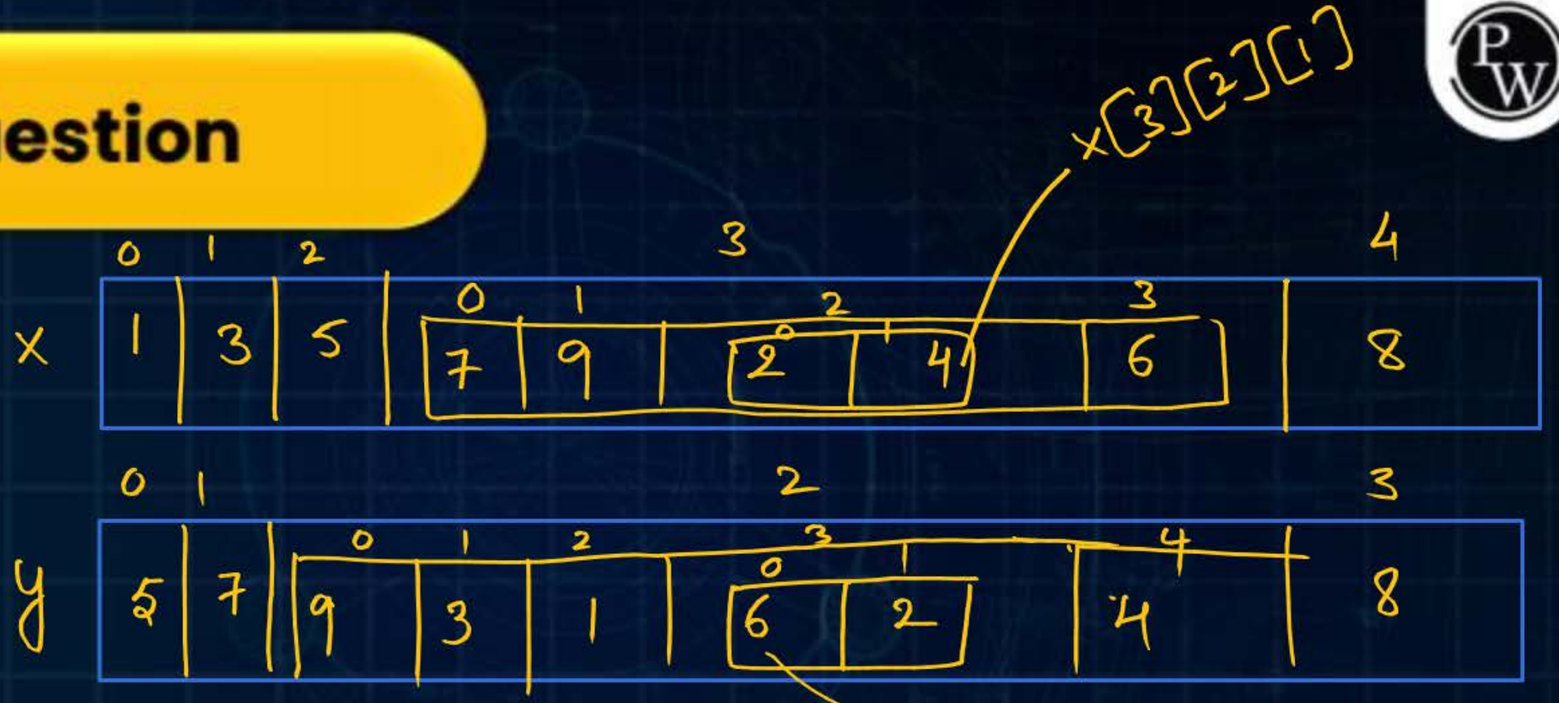
Homework Question

```
count=1
x=[1, 3, 5, [7, 9, [2, 4], 6], 8]
y=(5, 7,(9, 3, 1, (6, 2), 4), 8)
i=x[3][2][1]
j=y[2][3][0]
for a in range(i):
        for b in range(j):
                count=count+a+b
print(count)
```

i = 4

j = 6

ANS : 97



	b=0	b=1	b=2	b=3	b=4	b=5
a=0	count=1+0+0	1+0+1=2	2+0+2=4	4+0+3=7	7+0+4=11	11+5=16
a=1	16+1=17	17+2=19	19+1+2=22	22+4=26	26+5=31	31+6=37
a=2	37+2=39	39+3=42	42+4=46	46+5=51	51+6=57	57+7=64
a=3	64+3=67	67+4=71	71+5=76	76+6=82	82+7=89	89+8=97

Set Methods

Method	Shortcut	Description
add()		Adds an element to the set
clear()		Removes all the elements from the set
copy()		Returns a copy of the set
difference()	-	Returns a set containing the difference between two or more sets
difference_update()	-=	Removes the items in this set that are also included in another, specified set
discard()		Remove the specified item
intersection()	&	Returns a set, that is the intersection of two other sets
intersection_update()	&=	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()		Returns whether two sets have a intersection or not

Set Methods

Method	Shortcut	Description
issubset()	<=	Returns whether another set contains this set or not
	<	Returns whether all items in this set is present in other, specified set(s)
issuperset()	>=	Returns whether this set contains another set or not
	>	Returns whether all items in other, specified set(s) is present in this set
pop()		Removes an element from the set
remove()		Removes the specified element
symmetric_difference()	^	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	^=	Inserts the symmetric differences from this set and another
union()	|	Return a set containing the union of sets
update()	|=	Update the set with the union of this set and others

Question

S1 = {1, 3, 5, 3, 7, 1, 5} ⇒ S1 = {1, 3, 5, 7}

S2 = {1, 2, 3, 4, 5}

S3 = {3, 4, 5, 6, 7}

S4 = S1.union(S3) # S4 = {1, 3, 4, 5, 6, 7}

S5 = S2 & S3 # S5 = {3, 4, 5}

i = len(S4 - S5) # {1, 6, 7} ⇒ len() = 3 ⇒ i = 3

j = S3.difference(S4) # j = { } Empty set

≅ S3 - S4

Question

$S1 = \{'G', 'A', 'T', 'E'\}$

$S2 = \{'E', 'X', 'A', 'M', 'I', 'N', 'A', 'T', 'I', 'O', 'N'\} \Rightarrow S2 = \{'E', 'X', 'A', 'M', 'I', 'N', 'T', 'O'\}$

$S3 = \{'G', 'A', 'T', 'E', 'E', 'X', 'A', 'M'\} \Rightarrow S3 = \{'G', 'A', 'T', 'E', 'X', 'M'\}$

$S4 = \{'I', 'I', 'T', 'R', 'O', 'O', 'R', 'K', 'E', 'E'\} \quad S4 = \{'I', 'T', 'R', 'O', 'K', 'E'\}$

```
result = len(S3.intersection(S1))
for i in range(len(S4 - S2)):
    for j in range(len(S3 ∧ S1)):
        result = result + i + j
Print(result)
```

#8

$len(\{'G', 'A', 'T', 'E'\}) = 4$

$S4 - S2 = \{'R', 'K'\} \Rightarrow len() = 2$

$S3 \wedge S1 = \{'X', 'M'\} \Rightarrow len() = 2$

	j=0	j=1
i=0	4+0+0=4	4+0+1=5
i=1	5+1+0=6	6+1+1=8

Dictionaries

- Ordered Collection
- Mutable [values can be Mutable]
- dictionaries Support duplicates [duplicate values not keys]
- Dictionaries are Collection of {Key : Value} Pairs.
- Can be created as:

Object = { 'key1' : value1, 'key2' : value2 .... }

Ex: d = { 'a' : 1, 'b' : 2, 'c' : 3 }

Empty Dictionary:

object = { } Ex: d = { }

object = dict( )

Ex: d = dict( )

Dictionaries

Access Elements of dictionary

```
d = {'a': 3, 'b': 7, 'c': 5}

Print(d.keys())   # {'a', 'b', 'c'}

Print(d.values()) # {3  7  5}

Print(d.items())  # {'a': 3, 'b': 7, 'c': 5}
```

Ex:

```
for i in d.keys():
    for j in d.values():
        Print(i, j)
```

o/p:
```
a : 3
b : 7
c : 5
```

(OR)

```
for (i,j) in d.items():
    Print(i, j)
```

Dictionaries

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Question

```
d={'IITB':2022, 'IITM':2023, 'IISC':2024}

d['IITR']=2025      # d={'IITB':2022, 'IITM':2023, 'IISC':2024, 'IITR':2025}

d.update('IITM')=2020   #d={'IITB':2022, 'IITM':2020, 'IISC':2024, 'IITR':2025}

d.Pop('IITB') # d={'IITM':2020, 'IISC':2024, 'IITR':2025}

d.Popitem() # {'IITM':2020, 'IISC':2024}

del d['IITM'] # {'IISC':2024}

del d #removes Complete Dictionary
```

Question

Nested Dictionaries

```
Student = { 'Name' : { 'fname' : 'abc', 'lname' : 'xyz' }
            'Marks' : { 'I Year' : 70, 'II Year' : 90 }
            'Occupation' : { 'Mother' : 'Home maker', 'Father' : 'BUSINESS' } }

Print( Student['Name']['lname'] )   # xyz
Print ( Student ['Occupation'] ['Father'] ) # BUSINESS

Student['marks']['I Year'] = 80
```

Functions, Recursion

Function : A Single statement (or) multiple statements, that Performs Specific Task (or) Sub Task is called function.

Method Vs function	object.x() ⇒ x is method x(object) ⇒ x is function	- method is defined Inside class - function is defined outside class	- At a time through one object only, method is accessed. - function can have as many as needed objects as arguments.

Advantages : 1) Modularity 2) Reusability 3) Readability of Code

Definition Syntax : def Name(argument(s)):
Statement(s)

Invoke/call function : Name(arguments)

Ex:
```
def add(x,y):
    z = x+y
    Print(z)
add(4,5) # 9
```

Functions, Recursion

Recursion : The Process of Calling a function by itself.

- For recursive calls, to be finite, Base condition need to be included in function code.

- Base Case / Base Condition : The Point at which, Recursive calling will Stop.

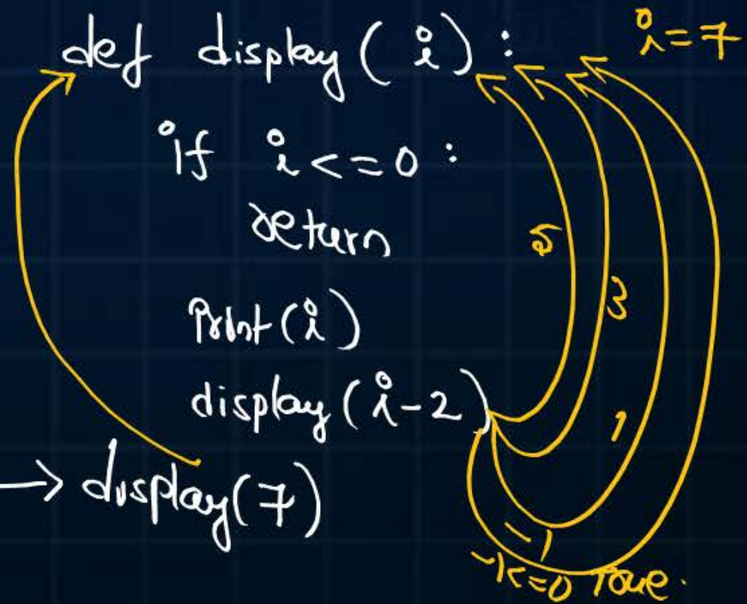
Syntax:

```
def fun(argument(s)):
    Base Condition :
        Statement(s)
    fun(argument)
fun(argument)
```

Ex:

```
def display(i):
    if i<=0:
        return
    Print(i)
    display(i-2)
```

→ display(7)



o/p:

7
5
3
1

Functions, Recursion

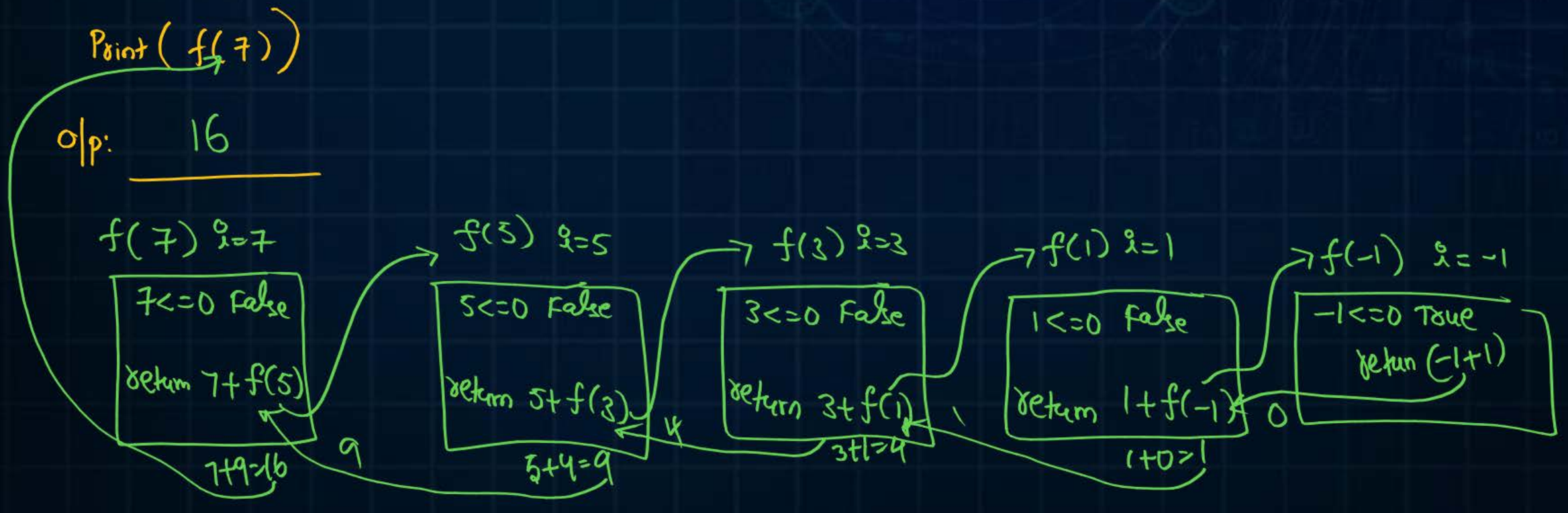
Question

```
def f(i):
    if i<=0:
        return i+1
    return i+f(i-2)

Print(f(7))
```

Base case

o/p: 16



Question

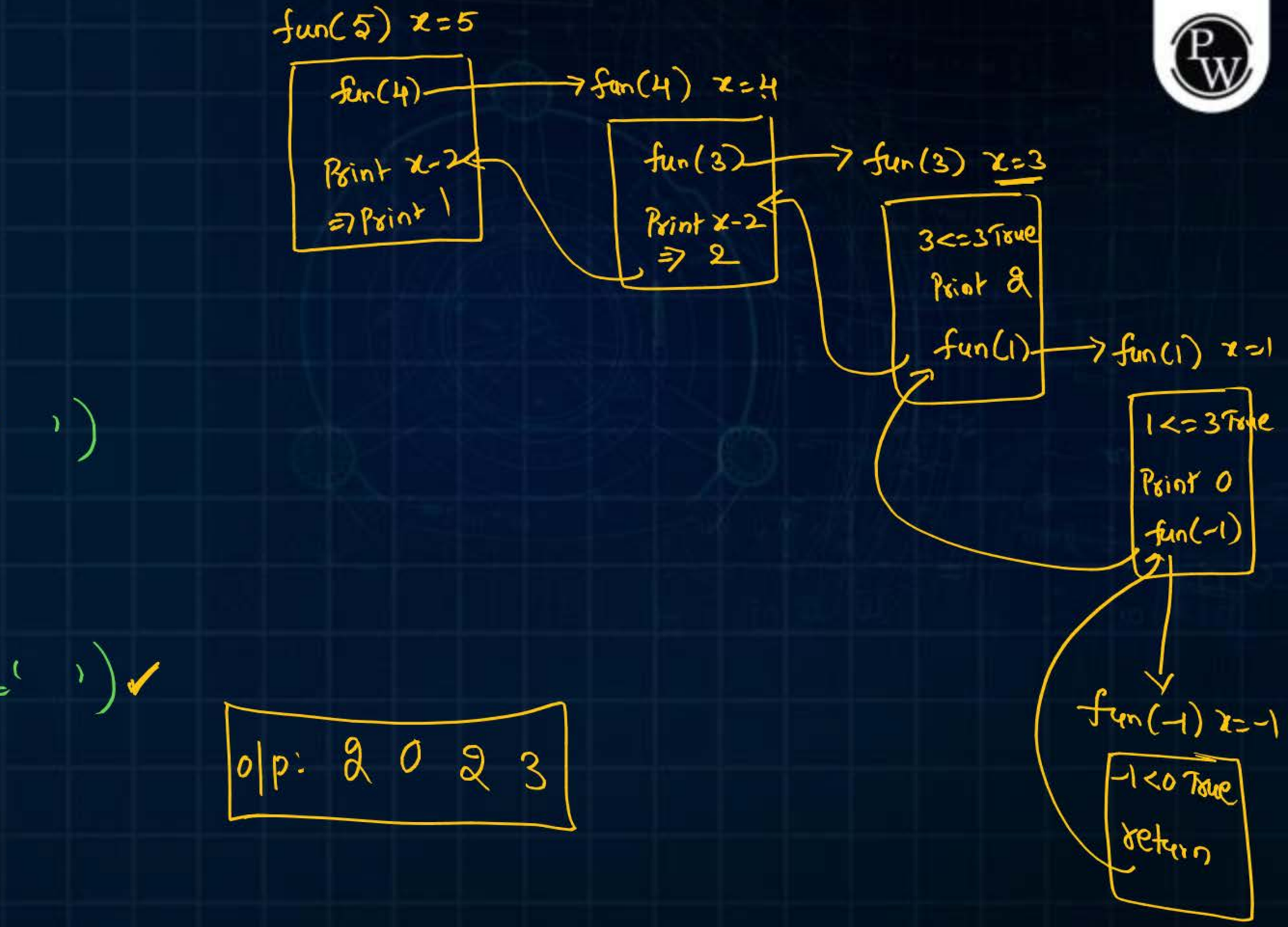
```
def fun(x):
    if x<0:
        return
    elif x<=3:
        print(x-1, end=' ')
        fun(x-2)
    else:
        fun(x-1)
        print(x-2, end=' ')
```

✓

fun(5)

o/p= _____

o/p: 2 0 2 3



Question

The Return value of f(7) is ____22____

```
def f(x):
    if x<1:
        return x+x
    elif x<=2:
        return (x-1)+f(x-1)
    elif x<=5:
        return (x+3)+f(x-3)
    else:
        return x+f(x-1)
```

f(7) x=7

7+f(6) x=6

6+f(5) x=5

(5+3)+f(2) x=2

(2-1)+f(1) x=1

(1-1)+f(0) x=0

(0+0)

= 7+6+8+1+0+0

= 22

Summary

- Set methods
- Dictionaries
- Nested dict
- functions
  - Recursion*
  - Examples

Thank

THANK

Keep Hustling!