I am d7

Built-in

Abdul - 1

↳ First class function
a. function as objects

pt(print)
print(print.__doc__)
#print(value, ..., sep=" ", end='\n',)
Print(print.__name__)
#print
show = print
show("Hello world")
#Hello world

~~take~~ = input
name = take("enter")
pt("welcome", name)
#enter : Abdul
welcome Abdul

1

def fun():
pt("My function")
f = fun
f()

// #My function

a. functions are treated as o
b. functions are access thei
properties
c. Assign to some variable
ther variable or be use
as functions

Nested Functions

→ Functions are object
→ Nested Functions
(Function inside Function)

Pros 1:-

```
def outer():
    def inner():
        pt("Hello")
    pt("outer")
    inner()
outer()
```

outer
Hello

pros 2 cubiod :-

```
def area(l, b, h):
    def area(d1, d2):
        return d1 * d2
    return 2 * (area(l * b) + area(b * h)
                + area(h * l))
area(2,3,4)
```

#52

Function as Parameter

Function takes function as parameter

```
def welcome():
    pt("welcome")

def fun():
    f()

fun(welcome)
# welcome
```

Prog 2

```
def add(x,y):
    return x+y

def sub(x,y):
    return x-y

def Calculator(f,x,y):
    return f(x,y)

pt(Calculator(sub,10,5)) # 5
pt(Calculator(add,10,5)) # 15
```

Module-2

lambda

Returning Functions :-

→ outer function returning inner function

```
def outer():
    def inner():
        pt("Hello")
    return inner

f = outer()
f() // Hello
```

Closure Function :-

a. Nested Function
b. Returning Function
c. Inner function access to outer Variables

```
msg = "welcome"
def inner():
    print("+" * 10)
    print(msg)
    print("+" * 10)

inner()
    # ++++++
    # welcome
    # ++++++
```

```
def outer():
    # msg = "welcome"
    def innee():
        print("+" * 10)
        print(msg)
        print("+" * 10)
    return innee

f = outer()
f()
// f = outer("welcome")
f()
```

Prog 3

```
count = 0
def Counter():
    global count
    count += 1
    return count
print(Counter())
print(Counter())
    2
```

```
def set_count():
    count = 0
    def counter():
        nonlocal count
        count += 1
        return count
    return counter

c1 = set_counter()
c2 = set_counter()
c1() c1() c1()
    # 1, 2, 3
c2() c2() c3()
    # 1, 2, 3
```

Rohit - IT
Abdul S

Decorators

a. Nested function
b. Returning function
c. Inner function access to outer values
d. function as parameter

lambda

```
def Outer(f):
    def inner():
        pt('*' * 10
        f()
        pt('-' * 10)
    return inner
def welcome():
    pt("welcome to decorator function")
f = Outer(welcome)
f()
```

```
*****
welcom to decorators
*****
```

Case 1:-

```
x = Outer(do
```

```
def outer(r):
    def inner():
        pt('*' * 10)
        f()
        pt('*' * 10)
    return inner
def display():
    pt("Display")
```

Case 1:-
```
r = outer(display)
r()
```

Case 2:-
```
display = outer(display)
display()
```

```
*****
Display
*****
```

Case 4:- Decorator

```
def outer(f):
    def inner():
        pt('*' * 10)
        f()
        pt('*' * 10)
    return inner
@outer
def display():
    pt("Display Function")
display()
```

```
*****
Display
*****
```

Lambda 4

Abdul

Case 5:-

```
def decorator(f):
    def inner():
        pt('*' * 10)
        f()
        pt('*' * 10)
    return inner

@decorator
def display():
    pt("Display function")

display()
**********
Display
**********
```

Lambda :-

a. Anonymous function
b. Simple function
c. Single line function
d. Functional programming

↳
def double(x):
    return x * 2

double(3)
# 6.

p = lamda x: * * 2
p(3) #6

↳ A = lamda x,y: x+y
pt(A(5,10))

↳ pt((lamda x: **2)(5))
#10

↳ L1 = [1 . . . 10]
f = filter(lamda x: x%.3==0, L1)
pt(list(f))
[3, 6, 9]

Map
L1 = [1,2,3,4]
L2 = list(map(lamda x: -x, L1))
p(L2) → [-1, -2, -3, -4]

// ↳ L1 = [1, . . . 10]
K = lamda x: x if x%.2==0 else -x
L2 = list(map(K, L1))
[-1, 2, . . . -9, 10]

↳
L1 = [[4, 2, 'six'], [1, 4, 'five'], [2, 2, 'four']]

pt (sorted (L1))
[[1, 4, 'five'], ...]

L2 = sorted (L1, key = lambda x: x[0] + x[1])
pt (L2)
[[2, 2, 'four'] ... [4, 2, 'six']]

Lends Abdul S

Caller x functions

#like closure function
#need function
#return function
#inner function
#inner function access outer variables

class Day:
    def __init__(self):
        self.days = {1: 'monday', ...}
    def __call__(self, dayno):
        return self.days[dayno]

d = Day()
pt (d[3]) # wednesday.