Functions as Objects:

First Class Functions🡪 Function as objects

print(print)

#<built-in function print>

print(print.\_\_doc\_\_)

#print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

A screenshot of a computer program

AI-generated content may be incorrect.

print(print.\_\_name\_\_)

#print

show=print

show("Hello World")

#Hello World

take=input

name=take("Enter your name: ")

print("Welcome",name)

#Enter your name: Abdul

#Welcome Abdul

def fun():

    print("My Function1")

f=fun

f()

#My Function1

* Functions are treated as objects
* Functions are access their properties
* Assign to some variable . That variable can be used as functions

Nested Functions:

* Functions are object
* Nested Functions

(Function inside Function)

def Outer():

    def Inner():

        print("Hello from the Inner function!")

    print("Outer")

    Inner()

Outer()

# Inner()

# # NameError: name 'Inner' is not defined

# Result:

# Outer

# Hello from the Inner function!

####################

# Cubiod=2\*((l\*b)+(b\*h)+(h\*l))

def totalarea(l,b,h):

    def area(d1,d2):

        return d1\*d2

    return 2\*((area(l,b))+(area(b,h))+(area(h,l)))

print(totalarea(2,3,4))

# 52

Function as parameter:

Function takes function as parameter

def welcome():

    print("Welcome to function parameter example!")

def fun(f):

    f()

fun(welcome)

# Welcome to function parameter example!

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)

print(calculator(add,10,5))

# 15

print(calculator(sub,10,5))

# 5

Returning Functions:

Outer function return inner function

def Outer():

    def Inner():

        print("Hello from the Inner function!")

    return Inner

f=Outer()

f()

# Hello from the Inner function!

Closure Functions:

Closure Function:

* Nested Function
* Return Function
* Inner Function access to outer variables

msg="welcome"

def inner():

    print("+" \* 10)

    print(msg)

    print("+" \* 10)

inner()

# ++++++++++

# welcome

# ++++++++++

Prog 2:

###########################3

def Outer():

    msg="welcome"

    def inner():

        print("+" \* 10)

        print(msg)

        print("+" \* 10)

    return inner

f=Outer()

f()

# ++++++++++

# welcome

# ++++++++++

Prog 3:

def Outer():

    # msg="welcome"

    def inner():

        print("+" \* 10)

        print(msg)

        print("+" \* 10)

    return inner

f=Outer("Welcome")

f()

# ++++++++++

# welcome

# ++++++++++

Prog 4:

count=0

def Counter():

    global count

    count +=1

    return count

print(Counter())

#1

print(Counter())

#2

print(Counter())

#3

Prog 5:

def get\_counter():

    count=0

    def Counter():

        nonlocal count

        count +=1

        return count

    return Counter

c1=get\_counter()

c2=get\_counter()

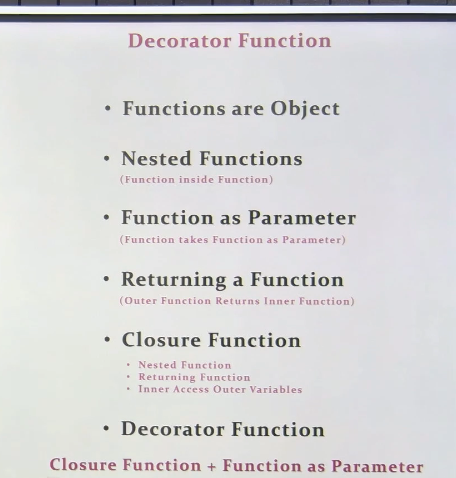
print(c1(),c1(),c1())

# 1 2 3

print(c2(),c2(),c2())

# 1 2 3

Decorators:



**# Nested Function**

**# Return Function**

**# Inner Function access to outer variables**

**# Function as parameter**

def Outer(f):

    def Inner():

        print("\*" \* 10)

        f()

        print("\*" \* 10)

    return Inner

def welcome():

    print("Welcome to Decorator Function")

f=Outer(welcome)

f()

# \*\*\*\*\*\*\*\*\*\*

# Welcome to Decorator Function

# \*\*\*\*\*\*\*\*\*\*

####################33

def Outer(f):

    def Inner():

        print("\*" \* 10)

        f()

        print("\*" \* 10)

    return Inner

def display():

    print("Display Function")

#case 1

# r=Outer(display)

# r()

#case 2

display=Outer(display)

display()

#case 3

# display=Outer(display)

display()

# \*\*\*\*\*\*\*\*\*\*

# Display Function

# \*\*\*\*\*\*\*\*\*\*

Case 4:

# case 4 :Decorator

def Outer(f):

    def Inner():

        print("\*" \* 10)

        f()

        print("\*" \* 10)

    return Inner

@Outer

def display():

    print("Display Function")

display()

# #\*\*\*\*\*\*\*\*\*\*

# Display Function

# \*\*\*\*\*\*\*\*\*\*

Case 5:

# case 5

def decorator(f):

    def Inner():

        print("\*" \* 10)

        f()

        print("\*" \* 10)

    return Inner

@decorator

def display():

    print("Display Function")

display()

# \*\*\*\*\*\*\*\*\*\*

# Display Function

# \*\*\*\*\*\*\*\*\*\*

Lambda:

A white paper with red text

AI-generated content may be incorrect.

def double(x):

    return x \* 2

print(double(3))

# 6

p=lambda x: x \* 2

print(p(3))

# 6

Case 2:

A=lambda x,y: x + y

print(A(5,10))

# 15

print((lambda x: x \*2) (5))

# 10

Case 3:

L1=[1,2,3,4,5,6,7,8,9,10]

f=filter(lambda x: x%3==0,L1)

print(list(f))

# [3, 6, 9]

Case 4:

L1=[1,2,3,4]

L2=list(map(lambda x: -x,L1))

print(L2)

# [-1, -2, -3, -4]

Case 5:

L1=[1,2,3,4,5,6,7,8,9,10]

k=lambda x: x if x%2==0 else -x

L2=list(map(k,L1))

print(L2)

# [-1, 2, -3, 4, -5, 6, -7, 8, -9, 10]

Case 6:

L1=[[4,2,"Six"],[1,4,"Five"],[2,2,"Four"]]

print(sorted(L1))

# [[1, 4, 'Five'], [2, 2, 'Four'], [4, 2, 'Six']]

L2=sorted(L1,key=lambda x: x[0]+x[1])

print(L2)

# [[2, 2, 'Four'], [1, 4, 'Five'], [4, 2, 'Six']]

#Caller Class

    # like Closure function

#Need function

#Return function inner function

#Inner function access outer variabels

class Day:

    def \_\_init\_\_(self):

        self.days={1:"Monday",2:"Tuesday",3:"Wednesday",

                   4:"Thursday",5:"Friday",6:"Saturday",7:"Sunday"}

    def \_\_call\_\_(self,dayno):

        return self.days[dayno]

d=Day()

print(d(3))

# Wednesday