PYTHON

LECTURE 20

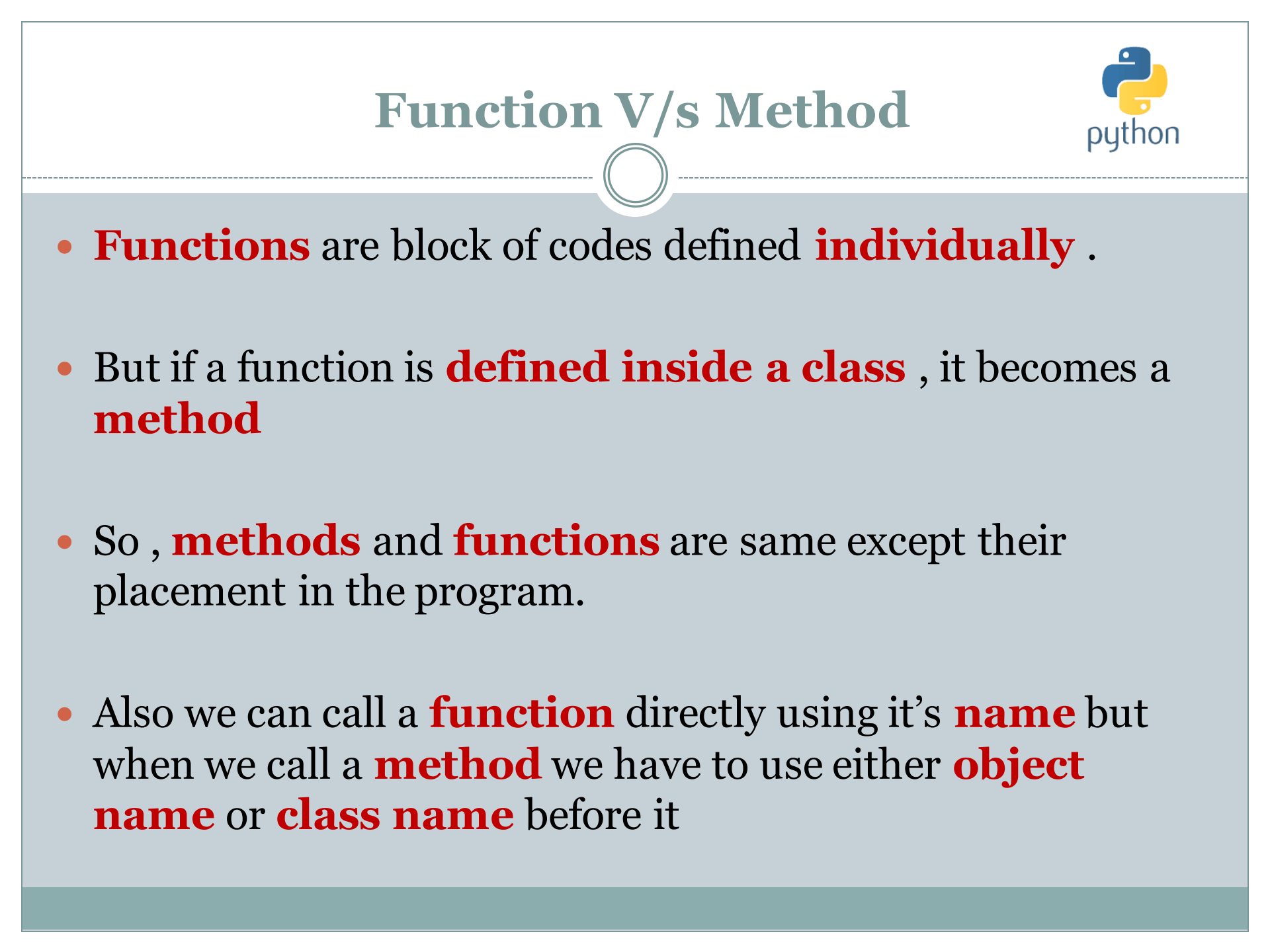
Today's Agenda

- **User Defined Functions**
  - What Is A Function ?
  - Function V/s Method
  - Steps Required For Developing User Defined Function
  - Calling A Function
  - Returning Values From Function

What Is A Function ?

- A **function** in **Python** is a **collection of statements** having a **particular name** followed by **parenthesis** .
- To **run** a function , we have to **call** it and when we call a function **all the statements** inside the **function** are **executed**.
- So we don't have to write the code again and again
- This is called **code re-usability**.

Function V/s Method



- **Functions** are block of codes defined **individually** .
- But if a function is **defined inside a class** , it becomes a **method**
- So , **methods** and **functions** are same except their placement in the program.
- Also we can call a **function** directly using it's **name** but when we call a **method** we have to use either **object name** or **class name** before it

Function V/s Method

For example:

- **print(“hello”)**

- Here **print( )** is a function as we are calling it directly

  - **message=“Good Morning”**
  - **print(message.lower())**

- Here **lower( )** is a method which belongs to the class **str** and so it is called using the object **message**

Steps Required For Function

- To create and use a function we have to take **2 steps**:

- **Function Definition**: Creating or writing the body of a function is called defining it. It contains the set of statements we want to run , when the function execute.

- **Function Call**: A function never runs automatically . So to execute it's statements we must call it

Syntax Of Function Definition

```
def function_name(param 1,param 2, . . .):
        statement(s)
```

- Keyword **def** marks the start of **function header**.
- It is followed by a **function name** to uniquely identify it.
- **Parameters** (arguments) through which we pass values to a function. They are **optional**.
- A **colon** (:) to mark the end of **function header**.
- One or more valid **python statements** that make up the function body . All the statements must have same indentation level

Example Of Function Definition

```
def add(a,b):
    print("Values are",a,"and",b)
    c=a+b
    print("Their sum is",c)
```

How To Call A Function ?

- Once we have **defined** a function, we can **call** it from another **function, program** or even the **Python prompt**.

- To call a function we simply type the function name with appropriate parameters.

- **Syntax:**

  - function_name(arguments)

Complete Example

```
def add(a,b):
    print("Values are",a,"and",b)
    c=a+b
    print("Their sum is",c)
add(5,10)
add(2.5,5.4)
```

Output:

```
Values are 5 and 10
Their sum is 15
Values are 2.5 and 5.4
Their sum is 7.9
```

Returning Values From Function

- To return a value or values from a function we have to write the keyword **return** at the end of the function body along with the value(s) to be returned

- **Syntax:**

  - return <expression>

Complete Example

```
def add(a,b):
    c=a+b
    return c
x=add(5,10)
print("Sum of 5 and 10 is",x)
y=add(2.5,5.4)
print("Sum of 2.5 and 5.4 is",y)
```

Output:

```
Sum of 5 and 10 is 15
Sum of 2.5 and 5.4 is 7.9
```

Exercise

- **Write a function called absolute( ) to accept an integer as argument and return it's absolute value. Finally call it to get the absolute value of -7 and 9**

- **Sample Output:**

```
absolute of -7 is 7
absolute of 9 is 9
```

Solution

```
def absolute(n):
    if n>0:
        return n
    else:
        return -n

x=absolute(-7)
print("absolute of -7 is",x)
y=absolute(9)
print("absolute of 9 is",y)
```

Exercise

- **Write a function called factorial( ) which accepts a number as argument and returns it's factorial. Finally call the function to calculate and return the factorial of the number given by the user.**

- ```
  Enter an int:4
  Factorial of 4 is 24
  ```
  ```

Solution

```
def factorial(n):
    f=1
    while n>1:
        f=f*n
        n=n-1
    return f

x=int(input("Enter an int:"))
y=factorial(x)
print("Factorial of",x,"is",y)
```

Guess The Output

```
def greet(name):
    print("Hello",name)

greet("sachin")
greet()
```

Output:

```
Hello sachin
Traceback (most recent call last):
  File "func5.py", line 5, in <module>
    greet()
TypeError: greet() missing 1 required positional argument: 'name'
```

Guess The Output

```
def greet(name):
    print("Hello",name)

greet("sachin", "amit")
```

Output:

```
TypeError: greet() takes 1 positional argument but 2 were given
```

Guess The Output

```
def greet(name):
    print("Hello",name)
    return
    print("bye")

greet("sachin")
```

Output:

```
Hello sachin
```

Guess The Output

```
def greet(name):
    print("Hello",name)


x=greet("sachin")
print("value in x is",x)
```

Output:

```
Hello sachin
value in x is None
```

Returning Multiple Values From Function

- In languages like **C** or **Java** , a function can return only one value . However in **Python** , a function can return **multiple values** using the following syntax:

- **Syntax:**
  - return <value 1, value 2, value 3, . . . >

- **For example:**
  - return a,b,c

- When we do this , **Python** returns these values as a **tuple**, which just like a **list** is a collection of multiple values.

Receiving Multiple Values

- To receive multiple values returned from a function , we have **2 options:**
- **Syntax 1:**
  - var 1,var 2,var 3=<function_name>()
- **Syntax 2:**
  - var=<function_name>()
- In the **first case** we are receiving the values in **individual variables** . Their **data types** will be set according to the **types of values** being **returned**
- In the **second case** we are receiving it in a **single variable** and **Python** will automatically make the data type of this variable as **tuple**

Complete Example

```
def calculate(a,b):
    c=a+b
    d=a-b
    return c,d
x,y=calculate(5,3)
print("Sum is",x,"and difference is",y)
z=calculate(15,23)
print("Sum is",z[0],"and difference is",z[1])
```

Here Python will automatically set x and y to be of int type and z to be of tuple type

Output:

```
Sum is 8 and difference is 2
Sum is 38 and difference is -8
```