IIT Madras

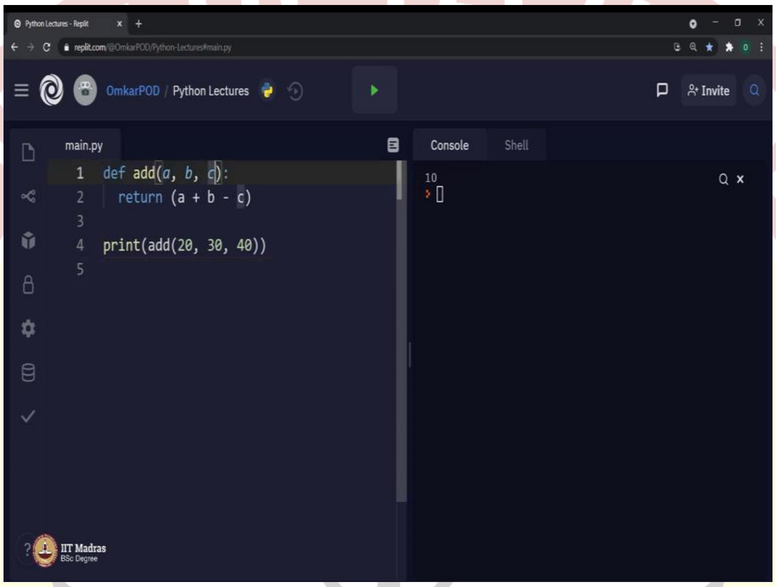
ONLINE DEGREE

Programming in Python

Professor. Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology, Ropar
Mr. Omkar Joshi
Course Instructor
Online Degree Programme
Indian Institute of Technology, Madras

Types of Function Arguments

(Refer Slide Time: 0:16)



```
def add(a, b, c):
    return (a + b - c)

print(add(20, 30, 40))
```

```
10
```

Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures#main.py
OmkarPOD / Python Lectures
Invite
main.py
Console
Shell
IIT Madras
BSc Degree

Hello python students. So far in this week’s lectures, professor has covered Python functions using various different examples like min, max, sorting, matrix multiplication and so on. Similar to previous week, the intension behind those lectures is to give you confidence in writing Python programs. But, in this lecture and in next few lectures of this week we will see some more details of functions.

So, let us start with different types of function arguments in Python. Let us look at this particular code block. At this point of time, this is a very basic and very straight forward code using function. And we all know what will be the output of this particular program. Before starting with types of arguments, let me revise few concepts which we have seen earlier.

Every function has various different components. The first component is called function definition which starts with keyword def. Next is called function call, this is the place where we are actually executing this defined function. The third component is arguments, the values which we pass along with the function call are referred as function arguments. And these variables which are supposed to store these arguments are called as parameters.

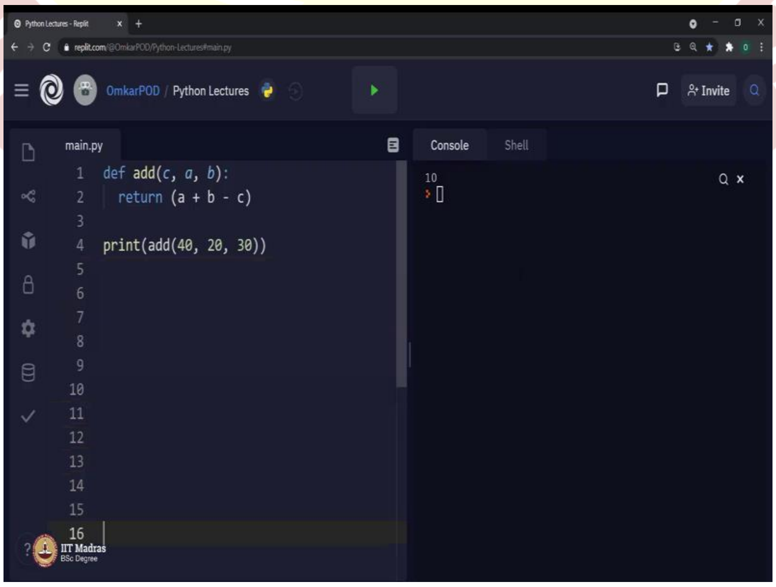
And once the entire function is executed as in, the function definition is executed, then we will have return statement. This return statement will pass the value back to the place where the function was called which is over here. Now, as we have sorted with all these terminologies like definition, call, parameter, arguments and return statement, now we are set to start with different types of arguments.

As we all know these arguments and thess parameters has a connection and this particular relation among arguments and parameters is based on this sequence in which they are written in the Python program. As in, the first argument over here is 20 that means this particular value 20 will be assigned to the first parameter over here, which means a will become 20, b will become 30 and c will become 40.

And because of that when we perform this expression a plus b minus c we are actually doing 20 plus 30 minus 40 and that is why the answer is 10. Now, you must be wondering what is so special about all this. This is something which is very basic at this point of time. So, let us introduce some complexity in this particular example.

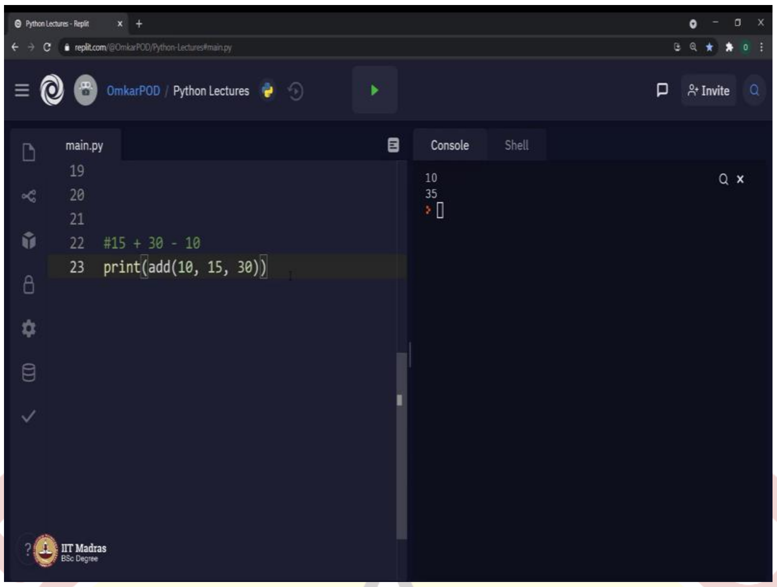
What if for some reason or even by mistake, instead of writing it a comma b comma c, I wrote it like c comma a comma b and executed this program. Now, the output is changed to 50 and we all know why that is happening. Because now variable c has value 20, a has 30 and b has 40. But I do not want this value 50. I was trying to get 10 as the output. In order to accommodate this change in the sequence of parameters, I also have to change the sequence of arguments over here.

(Refer Slide Time: 4:33)



```
def add(c, a, b):
    return (a + b - c)

print(add(40, 20, 30))
```

```
10
>
```



```
#15 + 30 - 10
print(add(10, 15, 30))
```

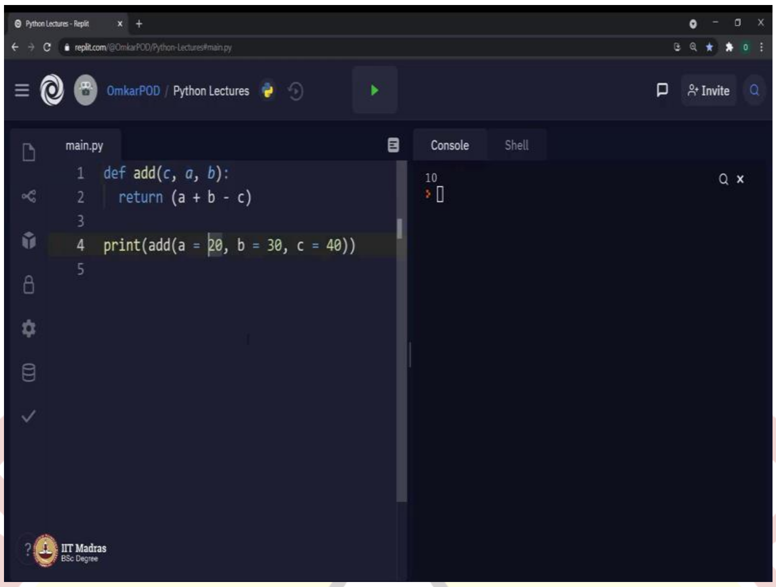
```
10
35
```

Now, this will solve my problem and I will get the output as 10. So far this is fine. What if I kept writing a Python program for next 20-25 lines and now, I want to use that particular function add over here. Print add and I want to execute this particular expression 15 plus 30 minus 10. Now, what should be the sequence of these arguments 15, 30 and 10? Do you remember, was it c a b, b a c, c b a or something else? What was the exact sequence of those parameters in the function definition? I do and not remember it at all.

So, in this case, I always have to go back and see the variable c is first with followed by a and then b. So, it is c a b. So, in that case I should write 10 first, then 15 and then 30. Then only I will get the expected output which is 35. But do not you think it is not a nice way to every time go back to function definition, see what is the sequence of these parameters, go back to the place where you actually want to use it and then try to map this argument order with the parameter order and this is not at all productive.

All these arguments are mapped against parameters based on its position and because of that this type of arguments are called as positional arguments where the position of each argument is critical with respect to the execution of the function definition. And that is exactly the problem we are facing over here.

(Refer Slide Time: 6:45)



```python
def add(c, a, b):
    return (a + b - c)

print(add(a = 20, b = 30, c = 40))
```

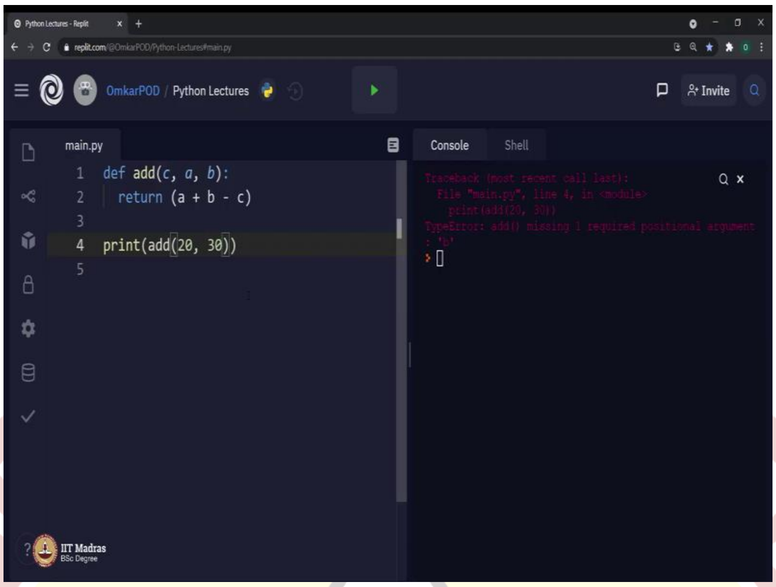
```
10
```

In order to solve this problem, we have the second type of function arguments called as keyword arguments. Let us go back to original code and see how to solve this particular problem. Earlier when we move this particular parameter c over here, we had to move this 40 as well. But now that is not required. Let us keep 40 over here and we can assign a is equal to 20, b is equal to 30 and c is equal to 40 over here itself. And now if we execute we will get the expected output as 10.

The value 20 is explicitly assigned to variable a, 30 to b and 40 to c. Because of that now the position or the sequence of these arguments has nothing to do with the sequence of parameters over here. The dependency between these two sequences has been eliminated completely. Now, we can give these arguments in any order as we like provided we mention the parameter name along with the argument over here in function definition itself. And this type of function arguments are referred as keyword arguments.

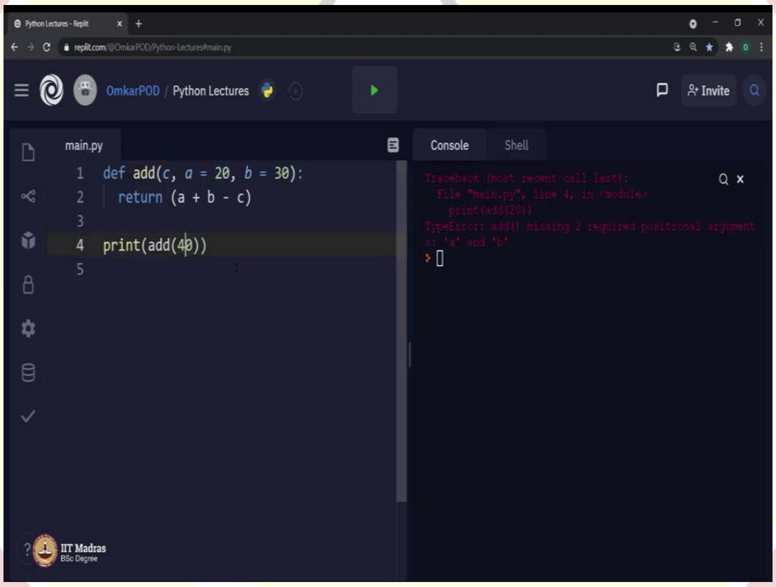
This type of argument will solve the problem which we faced earlier where we had to remember what is the actual sequence of parameters in the function definition. But still there is one more problem which we all must have faced many times while writing functions which is every time we have to remember how many parameters are there in the function definition. Let us go back to previous code. Currently we have three parameters and three arguments, hence the code is executing without an error.

(Refer Slide Time: 9:09)



```python
def add(c, a, b):
    return (a + b - c)

print(add(20, 30))
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(add(20, 30))
TypeError: add() missing 1 required positional argument
: 'b'
>
```



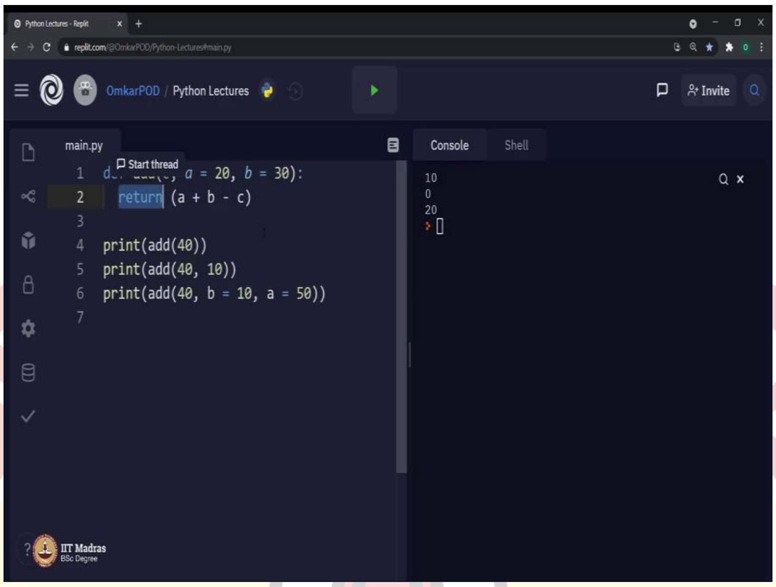
```python
def add(c, a = 20, b = 30):
    return (a + b - c)

print(add(40))
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(add(20))
TypeError: add() missing 2 required positional argument
s: 'a' and 'b'
>
```

But what if I remove this 40 from here and execute code like this? It says missing 1 required positional argument. As we have seen earlier, this type of arguments are called positional arguments. What if I remove this 30 as well? Now, it says missing 2 required positional argument. And this is very common, and it happens many times when we write a Python code.

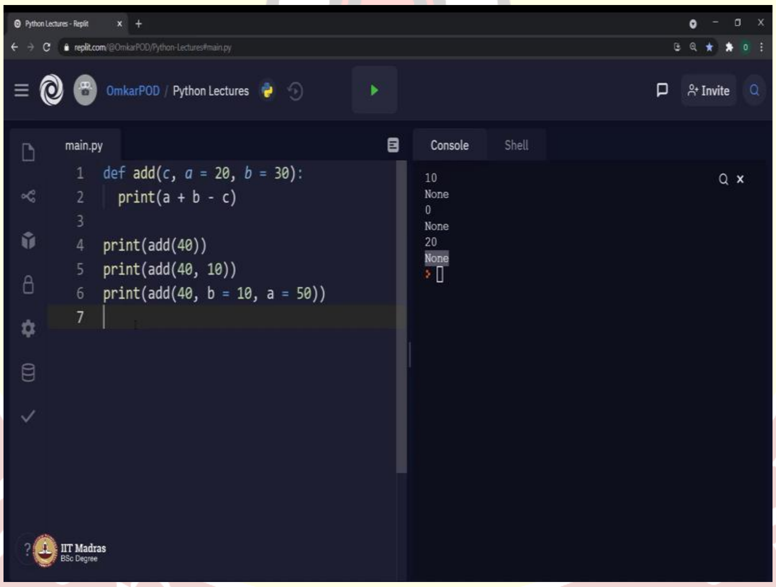
Logically we all know if there are 3 parameters, there should be 3 arguments. This number of parameters and number of arguments should be equal. But sometimes by mistake we write a code like this and end up getting errors. And this might become frustrating at a point. Therefore, Python has a way to solve this problem as well. And the solution to this problem is third type of function arguments call default arguments. For example, value for a was 20, for b it was 30 and for c let us say it was 40.

(Refer Slide Time: 10:28)



```
def add(c, a = 20, b = 30):
    return (a + b - c)

print(add(40))
print(add(40, 10))
print(add(40, b = 10, a = 50))
```

```
10
0
20
>
```


Python Lectures - Replit
replit.com/@OmkarPOD/Python-Lectures#main.py
OmkarPOD / Python Lectures
Invite
main.py
Console
Shell
IIT Madras
BSc Degree

```
def add(c, a = 20, b = 30):
    print(a + b - c)

print(add(40))
print(add(40, 10))
print(add(40, b = 10, a = 50))
```

```
10
None
0
None
20
None
>
```

If we execute this code, now it is executing without an error because in this case the value 40 is been passed to variable c. Whereas, as we are not passing any specific values for next two parameters, computer will assign these two default values to variables a and b respectively.

Now, we can write a code in any way we want. For example, if we write 40. Now, instead of a is equal to 20, what if I want to make it 10? I can always write like that. Let us say print add 40, 10, 50, still I can write that and if I execute this code, every single time I will get output 10 for first print, 0 for second print and 20 for third print. Because of this default arguments over here, now our code will execute even when we have one argument or 2 arguments or 3 arguments.

When we said 40, 10 and 50, computer will consider this 10 as a value for a, not 20. Similarly, for b instead of 30, computer will consider 50 because we have explicitly mentioned 10 and 50, whereas in this particular statement, we have mentioned the value for a. Hence, computer will consider 10 as a value for variable a, whereas for variable b it will still consider 30.

Which mean, if we explicitly give some value, then computer will consider that or else it will go back to its default value which is mentioned over here. Now, you must be wondering is it necessary to have only positional arguments or only keyword arguments or only default arguments or is it possible to have a different combination of these arguments? And the answer is it is possible to have different combinations of these arguments.

Let us say b is equal to 10 and a is equal to 50. Still this code will execute. Now, in this case, we have 40 as a positional argument, b and c as keyword arguments whereas, a and b were already defined as default arguments. Based on this demonstration, we can conclude that these 3 types of arguments can be used together as per our requirement, and it will provide lot of flexibility when we write a Python program using functions.

So far, we are talking about function definition, its parameters, function call, different types of arguments and so on, but we did not talked at all about the return statement. Therefore, before closing this lecture, let me give you a small task as kind of a homework. What if I remove this return statement and replace it with print. If I execute, still I will get output as 10, 0, 20 but there will be something else along with which is none, none and none.

Now it is your job to find out why this is printing none 3 times after every number 10, 0 and 20. Just to give you an hint, this particular print statement is printing these values 10, 0 and 20. Whereas, these 3 print statements are printing these values none over here. Now it is your task to find out why this particular none is getting printing over here. Thank you for watching this lecture. Happy learning.