



In [6]:

```
def happy():
    print("Happy Birthday to you !!")

happy()
Happy Birthday to you !!
```



Hello, Functions!

We define a function using the def keyword:

```
>>> def say_hello():
...     print('Hello')
...
```

Once the function is defined, you can call it:

```
>>> say_hello()
Hello
```

A screenshot of a video conference interface. In the top right corner, there is a video feed of a man wearing glasses and a dark shirt, labeled "Host". The main area shows a code editor with Python pseudocode:

```
def name(parameters):
    statements
    return value      # optionally
```

Below the code, three definitions are listed:

- name: name of function (in `snake_case`)
- parameters: information passed into function
- return: information given back from the function

The video player interface at the bottom includes a progress bar from 13:00 to 47:44, a volume slider, and other control icons.

A screenshot of a video conference interface. In the top right corner, there is a video feed of a man wearing glasses and a dark shirt, labeled "Host". The main area shows a slide with the title "Scope".

The text on the slide states: "A variable is only available from inside the region it is created. This is called scope."

The video player interface at the bottom includes a progress bar from 13:00 to 47:44, a volume slider, and other control icons.

The screenshot shows a video conference interface. In the top right corner, there is a video feed of a man with glasses and a beard, labeled "Host". The main content area displays a presentation slide with the title "Scope". Below the title, there is a bulleted list of points:

- The name that identifies a variable has certain visibility throughout the program
- Three fundamental levels of scope
 - Global ✓
 - Local ✓

At the bottom of the slide, there is a navigation bar with icons for back, forward, and search, along with the URL "www.goclasses.in" and a timestamp "21:02 - 39:42".

The screenshot shows a Jupyter Notebook cell. The code in the cell is:

```
In [17]: def myfunc():
    x = 300
    print(x)

myfunc()
print(x)
```

The output of the code is:

```
300
```

Below the code, a traceback for a NameError is shown:

```
NameError
Cell In[17], line 6
  3     print(x)
  5 myfunc()
----> 6 print(x)

NameError: name 'x' is not defined
```

Overlaid on the code and output is a hand-drawn diagram. It consists of a pink rectangular box containing two smaller boxes: one labeled "300" and another labeled "x". To the right of the pink box, the text "myfunc" is written.

In the top right corner, there is a video feed of the same man as in the previous screenshot, labeled "Host".

GO CLASSES



```
In [18]: def myfunc(x):
    print(x)

myfunc(300)

300
```

```
In [19]: def myfunc(x):
    print(x)

myfunc(300)
print(x)

300
```

```
NameError                                 Traceback (most recent call last)
Cell In[19], line 5
      2     print(x)
      3 myfunc(300)
----> 5 print(x)

NameError: name 'x' is not defined
```

24:42 -36:02



local variable

```
In [18]: def myfunc(x):
    print(x)

myfunc(300)

300
```

x is a local variable

```
In [19]: def myfunc(x):
    print(x)

myfunc(300)
print(x)

300
```

we can't access outside the func.

```
NameError                                 Traceback (most recent call last)
Cell In[19], line 5
      2     print(x)
      3 myfunc(300)
----> 5 print(x)

NameError: name 'x' is not defined
```

25:16 -35:28

A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a host wearing glasses and a dark shirt, sitting in front of a microphone. The video content displays a Python code snippet:

```
def mySum(x,y):  
    sum = x + y # Sum is a local variable  
    return sum
```

An orange arrow points from the word 'local' in the explanatory text below to the word 'sum' in the code. To the right of the code, there is a bulleted list:

- x and y are local variables that only exist in the scope of the function mySum

The Go Classes logo is visible at the bottom of the video frame.

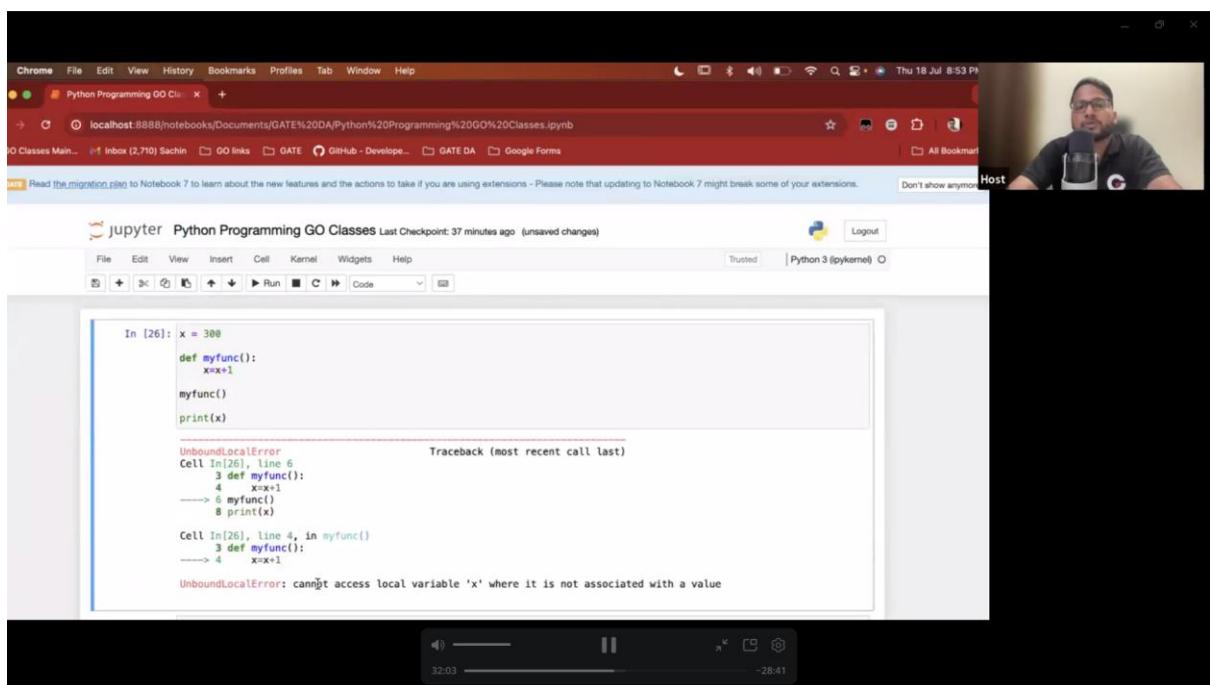
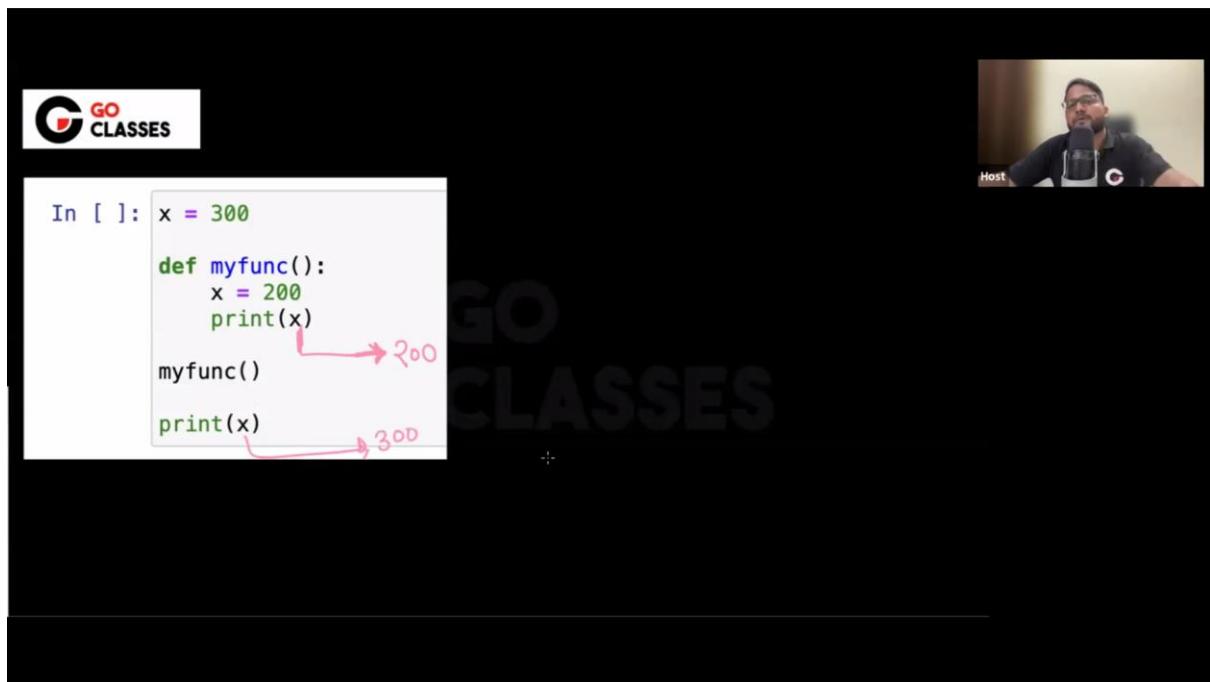
A screenshot of a video player interface for 'GO CLASSES'. The video frame shows a host wearing glasses and a dark shirt, sitting in front of a microphone. The video content displays a Python code snippet with handwritten annotations:

```
def mySum(x,y):  
    local  
    sum = x + y # Sum is a local variable  
    return sum
```

Below the code, there is more code:

```
# Global variables  
a = 10  
b = 20  
  
#c is also Global  
c = mySum(a,b)
```

The word 'local' is written above the line 'sum = x + y' with a red arrow pointing to it. The Go Classes logo and a navigation bar with a play button and time markers (27:05 and -33:39) are visible at the bottom of the video frame.



The screenshot shows a video player interface for 'GO CLASSES'. In the top left corner is the channel logo. In the top right corner, there is a small video frame showing a man, identified as the 'Host'. The main content area displays a Python code snippet in a terminal window:

```
In [7]: x = 300
def myfunc():
    x = 5
myfunc()
print(x)
```

A pink handwritten note next to the code asks: 'Q: how to modify global variable inside the function?' A yellow arrow points from the handwritten note to the line 'x = 5' in the code. At the bottom of the video player are standard controls: volume, play/pause, and a progress bar indicating the video is at 36:26 and has a total duration of 24:18.

The screenshot shows a video player interface for 'GO CLASSES'. The title 'The global keyword' is prominently displayed in large white text. In the top right corner, there is a small video frame showing a man, identified as the 'Host'. The main content area contains text explaining the behavior of variables within functions:

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the "global" keyword.

```
def myfunc():
    global x
    x = "fantastic"
myfunc()
print("Python is " + x)
```

At the bottom of the video player is a footer with the website address 'www.goclasses.in' and the Go Classes logo.

In [7]:

```
x = 300
def myfunc():
    x = 5
myfunc()
print(x)
```

Q: how to modify global variable inside the function?

Host

38:35 -22:09

In [1]: # Variable local to the function

```
def increment_x(x):
    x = x + 1
    print('This is the value of x inside the function:', x)
    return
x = 42
increment_x(x)
print('This is the value of x outside the function:', x)
```

This is the value of x inside the function: 43
This is the value of x outside the function: 42

www.goclasses.in

The screenshot shows a video player interface for Go Classes. On the left, a code editor window displays the following Python script:

```
y = 7
def f(x):
    print(x)
    print(y)
f(4)
print(y)
print(x)
```

On the right, a terminal window shows the output of running this script:

Output:

```
4
7
7
NameError: name 'x' is not defined
```

Below the terminal, a message reads "Hmmm....there seems to be an error". A yellow arrow points from the word "NameError" to the line "name 'x' is not defined".

At the bottom of the video player, there is a progress bar indicating the video is at 42:19 of 18:25.

The screenshot shows a video player interface for Go Classes. On the left, a code editor window displays the following Python script:

```
a=3
b=2
def foo(x):
    return a+x
def bar(x):
    b=1
    return b+x
print(foo(3), bar(3))
```

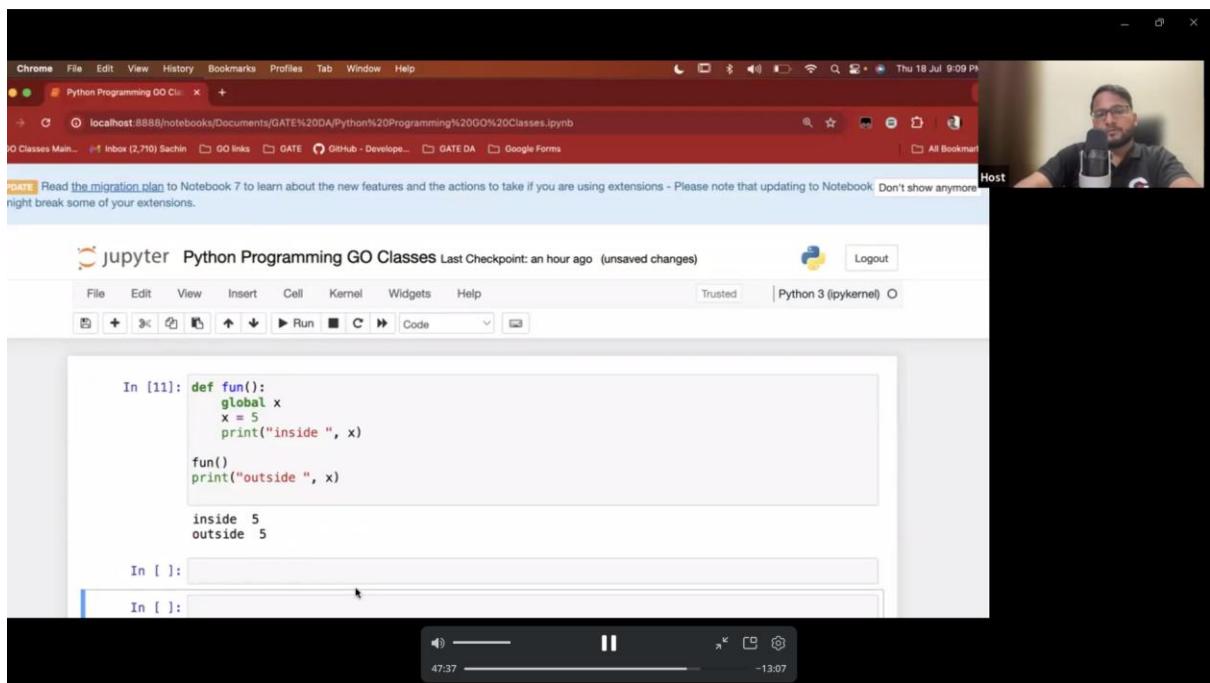
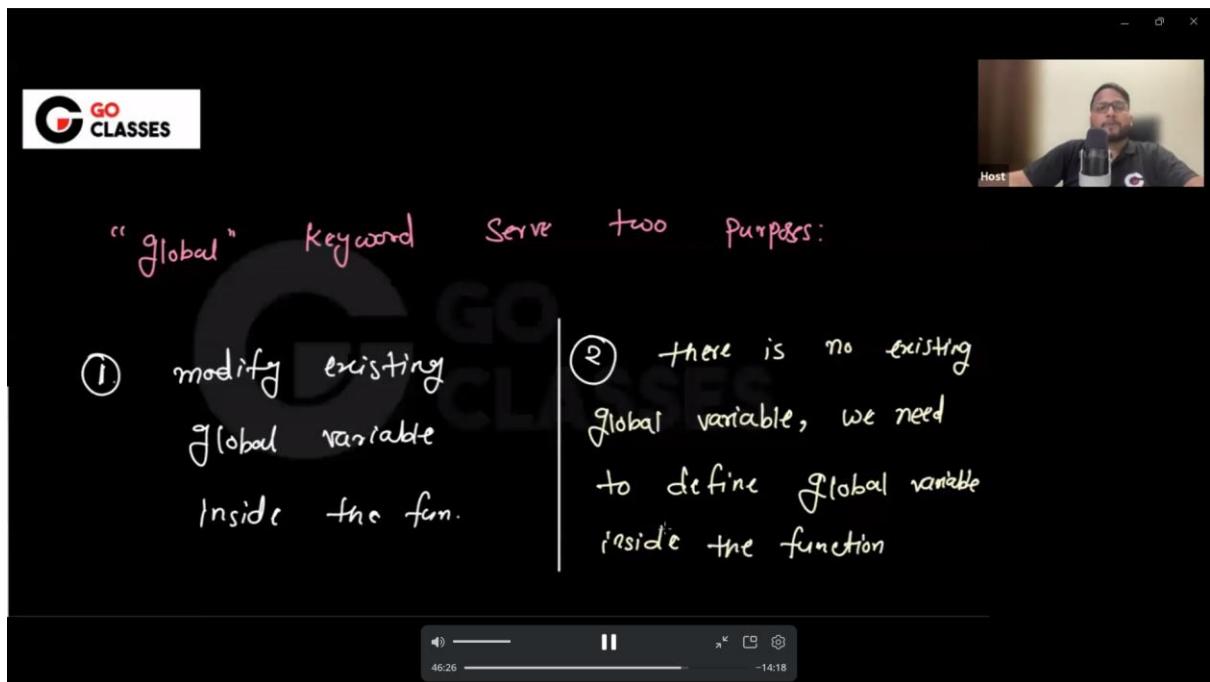
Annotations on the left side of the code explain variable scope:

- A blue arrow points to the line "b=1" with the text "b is now local" written below it.
- A blue arrow points to the line "return b+x" with the text "b is now local" written below it.

On the right, a list of bullet points explains the behavior of variables *a* and *b*:

- *a* and *b* are two global variables
- In function *foo*:
 - *a* is global, its value remains 3
- In function *bar*:
 - *b* is local, since it is redefined to be 1

At the bottom of the video player, there is a progress bar indicating the video is at 42:19 of 18:25.



Question: What will be the output of the following code ?

The screenshot shows a video player interface for 'GO CLASSES'. The video title is 'Question: What will be the output of the following code ?'. The video frame displays Python code and annotations. The code is:

```
a = 1  
b = 2  
  
def foo():  
    global a  
    a = 2  
    b = 3  
    print("In foo:", "a=", a, " b=", b)  
  
print("Outside foo: ", "a=", a, " b=", b)  
foo()  
print("Outside foo: ", "a=", a, " b=", b)
```

Annotations include:

- In foo:
 - A local variable *b*
 - A global variable *a*
 - The value of *a* changes by executing *foo()*

Handwritten notes:
a = 1 , b = 2
In foo a = 2 , b = 3

Below the video frame, there is a browser header with 'www.goclasses.in' and a progress bar indicating 50:47 of a 57-minute video.

The screenshot shows a video player interface for 'GO CLASSES'. The video frame displays Python code and annotations. The code is identical to the one in the previous screenshot:

```
a = 1  
b = 2  
  
def foo():  
    global a  
    a = 2  
    b = 3  
    print("In foo:", "a=", a, " b=", b)  
  
print("Outside foo: ", "a=", a, " b=", b)  
foo()  
print("Outside foo: ", "a=", a, " b=", b)
```

Annotations include:

- In foo:
 - A local variable *b*
 - A global variable *a*
 - The value of *a* changes by executing *foo()*

Handwritten notes:
a = 1 , b = 2
In foo a = 2 , b = 3
Outside foo: a = 2 b = 2 }
In foo: a = 2 b = 3 }
Outside foo: a = 2 b = 2 }
Answers.

Below the video frame, there is a browser header with 'www.goclasses.in' and a progress bar indicating 50:47 of a 57-minute video.

GO CLASSES

```
In [12]: a = 100

def bar():
    a = a+1

bar()
print(a)

UnboundLocalError
Cell In[12], line 8
  4 def bar():
  5     a = a+1
----> 6 bar()
  7 print(a)

Cell In[12], line 5, in bar()
  4 def bar():
----> 5     a = a+1

UnboundLocalError: cannot access local variable 'a' where it is not associated with a value
```



Host



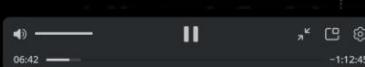
```
def mystery(a):
    print(a)
    for i in range(1, len(a)):
        a[i] += a[i-1]
    print(a)

mystery([8, 5, 0, -7, 4])
[8, 5, 0, -7, 4]
[8, 13, 0, -7, 4]
[8, 13, 13, -7, 4]
[8, 13, 13, 6, 4]
[8, 13, 13, 6, 10]
```



Host

<https://www.cs.middlebury.edu/~cs101/homework/cs101-midterm-sample-Qs-soln.pdf>



GO CLASSES

In [3]: `l = ['a', 'b']`

```
def fun(p,q):
    p.append(q)

fun(l,'c')
print(l)
```

['a', 'b', 'c']

Host

29:27 -50:00

`l = ['a', 'b'] ✓`

```
def func(p, q):
    p.append(q)

func(l, 'c')
print(l)
```

Diagram illustrating list mutation:

A list `l` is shown as a horizontal box containing elements `a` and `b`. A pointer from the variable `l` in the code points to the first element of the list. Inside a pink-bordered box labeled `func`, there is a local variable `p` pointing to the same list `l`. A local variable `q` contains the value `c`. An arrow shows the value `c` being appended to the list `l` at its end, resulting in the mutated list `a | b | c`.

Host

30:02 -49:25

Question: What will be the output of the following code ?

```

d = "Hello"

def func(p):
    p = p + ", world!"

func(d)
print(d)
    
```

www.goclasses.in

32:14 - 47:13

```

def h(L):
    L[1] = [7, 8]
    L[0] = [5, 6]
    print(L)

L = [[1, 2], [3, 4]]
h(L)
print(L)
    
```

40:42 - 38:45

The screenshot shows a video call interface. In the top right corner, there is a video window of a man wearing glasses and a light-colored shirt, identified as the 'Host'. The main area of the screen displays a Jupyter Notebook interface. The notebook has two code cells:

```
In [1]: def greeting(name = "friend"):
    print("Hello "+name)

greeting()
greeting("Mike ")
Hello friend
Hello Mike

In [2]: def greeting(name):
    print("Hello "+name)

greeting()
greeting("Mike ")

TypeError: greeting() missing 1 required positional argument: 'name'
```

Below the notebook, there is a media control bar with a volume icon, a play/pause button, and a progress bar indicating the video is at 47:11 and has a total duration of 32:16.

The screenshot shows a video call interface. In the top right corner, there is a video window of a man wearing glasses and a light-colored shirt, identified as the 'Host'. The main area of the screen displays a Jupyter Notebook interface running in a web browser. The browser's title bar shows 'Python Programming GO Classes'. The notebook has one code cell:

```
In [3]: def foo(x, y=10):
    print(x+y)

foo(5)
foo(5,100)
15
105
```

The browser's toolbar includes standard options like File, Edit, View, History, Bookmarks, Profiles, Tab, Window, Help, and a Logout button. The status bar at the bottom indicates the last checkpoint was an hour ago and shows 'unsaved changes'.

Demonstrating default argument values

```
def say(s, times=1):
    print(s * times)

say('Hello')
say('World', 3)
```

The code defines a function `say` that prints a string `s` repeated `times` times. If no value is provided for `times`, it defaults to 1. The first call `say('Hello')` prints "Hello". The second call `say('World', 3)` prints "WorldWorldWorld". Handwritten annotations show arrows from the function call parameters to the output words.

www.goclasses.in

49:53 -29:34

Order of Default Arguments:

Consider the following function definition:

```
def fun(x=1, y, z=2):
    print(x,y,z)
```

Suppose we call the function with `fun(5,6)` then what are the values of `x,y,z` inside function?

www.goclasses.in

50:03 -29:24

The screenshot shows a video conference interface. In the top right corner, there is a video feed of a man wearing glasses and a grey polo shirt, labeled "Host". The main area displays a presentation slide with a dark background. At the top left of the slide is the "GO CLASSES" logo. The title of the slide is "Using Default Argument Values". Below the title, a text block states: "Only those parameters which are at the end of the parameter list can be given default argument values." To the right of this text is a small, semi-transparent watermark or logo that looks like a stylized letter 'S'.

Only those parameters which are at the end of the parameter list can be given default argument values.

- ▶ We cannot have a parameter with a default argument value before a parameter without a default argument value, in the order of parameters declared, in the function parameter list.
- ▶ This is because values are assigned to the parameters by position.

Example

- ▶ `def func(a, b=5)` is valid
- ▶ `def func(a=5, b)` is not valid

53:39 -25:48

The screenshot shows a Jupyter Notebook interface running in a browser window. The title bar of the browser says "Python Programming GO Classes". The notebook cell content is as follows:

```
In [4]: def fun(x=1, y, z=2):  
    print("x = ", x)  
    print("y = ", y)  
    print("z = ", z)  
  
fun(5,6) I  
Cell In[4], line 1  
  def fun(x=1, y,  
          ^  
SyntaxError: non-default argument follows default argument
```

54:13 -25:14

GO CLASSES

```
In [3]: def fun(x, y=1, z=2):
    print("x = ", x)
    print("y = ", y)
    print("z = ", z)

fun(5)

x = 5
y = 1
z = 2
```

```
In [4]: def fun(x, y=1, z=2):
    print("x = ", x)
    print("y = ", y)
    print("z = ", z)

fun(5, 9)

x = 5
y = 9
z = 2
```

```
In [5]: def fun(x, y=1, z=2):
    print("x = ", x)
    print("y = ", y)
    print("z = ", z)

fun(5, 9, 84)

x = 5
y = 9
z = 84
```

Host

put all default arguments at the end.

57:28 -21:59

GO CLASSES

```
In [5]: def func(a, b=5, c=10, d=20):
    print("a =", a)
    print("b =", b)
    print("c =", c)
    print("d =", d)

# Example usage:
func(1)
func(1, 2)
func(1, 2, 3)
func(1, 2, 3, 4)
```

Host

www.goclasses.in

GO CLASSES

```
In [5]: def func(a, b=5, c=10, d=20):
    print("a =", a)
    print("b =", b)
    print("c =", c)
    print("d =", d)

# Example usage:
func(1)           # This will use the default values for b, c, and d
func(1, 2)        # This will use the default values for c and d
func(1, 2, 3)     # This will use the default value for d
func(1, 2, 3, 4)  # This will use the provided values for all parameters
```

a = 1
b = 5
c = 10
d = 20

a = 1
b = 2
c = 3
d = 20

a = 1
b = 2
c = 3
d = 4

✓

SES

Host