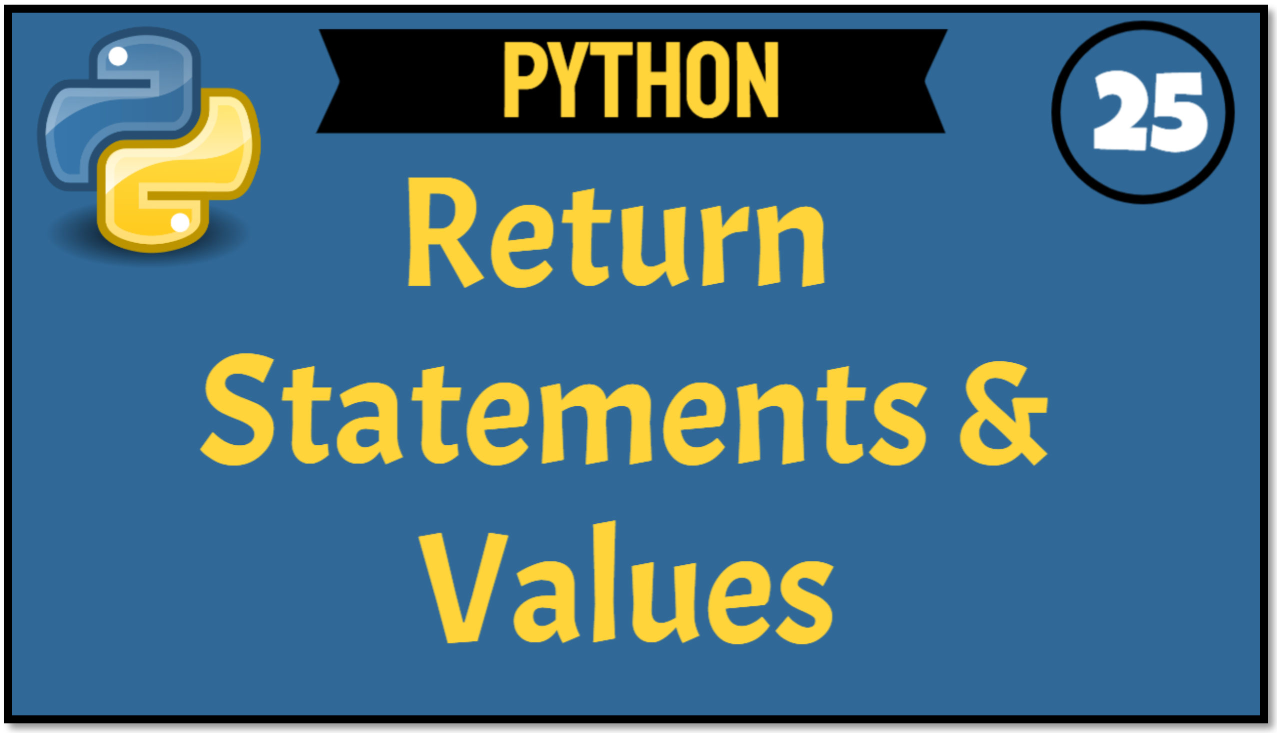
Rex Jones II

Python Return Statements & Values



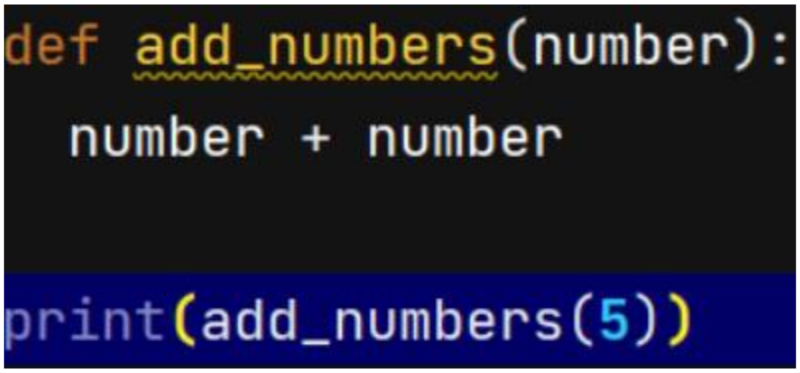
Python Video = https://youtu.be/Gc30ZnTvv60

Return Values & Statements

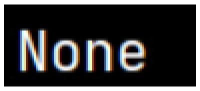
In this tutorial session, we are going to focus on returning a value from a function. A defined function can process information then send back a value to the calling function. The value is sent back using a return statement and the value is called a return value.

First step is to create a 'def' defined function with a name like add_numbers(): At this point, Python allow us to set the amount of parameters for our function. The purpose is to provide how many values to send to this add_numbers() function. Our parameter will be number. We are going to add number + number. Now when it comes to this function, it will add number by itself (number + number) Now, let's call the add_numbers() function then pass in 5 as the argument and also print() the value.

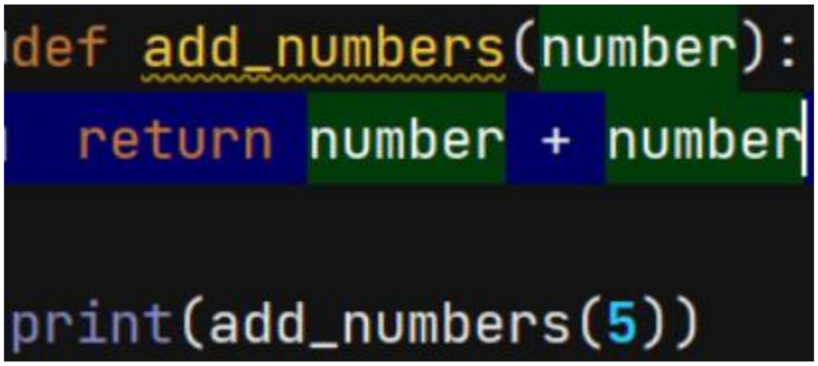
Rex
Jones II


```
def add_numbers(number):
  number + number

print(add_numbers(5))
```

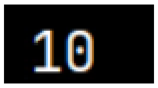
When we run the console shows None.


```
None
```

None is a reserved keyword that serves as the absence of a value. We see None because our defined function does not have a return keyword before number + number.

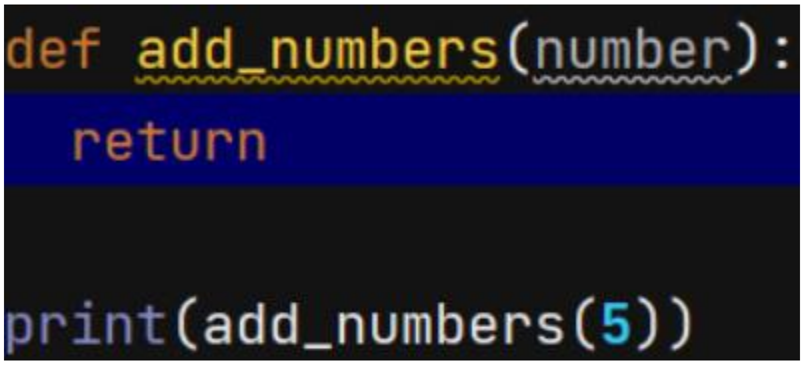

```
def add_numbers(number):
  return number + number

print(add_numbers(5))
```

This line is a return statement which consist of a return keyword and a value or expression. number + number is an expression that will be returned to the calling function. Let's execute and this time we see 10 in the console.

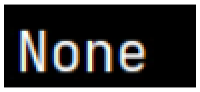

```
10
```

The reserved keyword None also shows up if we forget to add a value or expression after the return keyword. Remove number + number then Run.

Rex Jones II



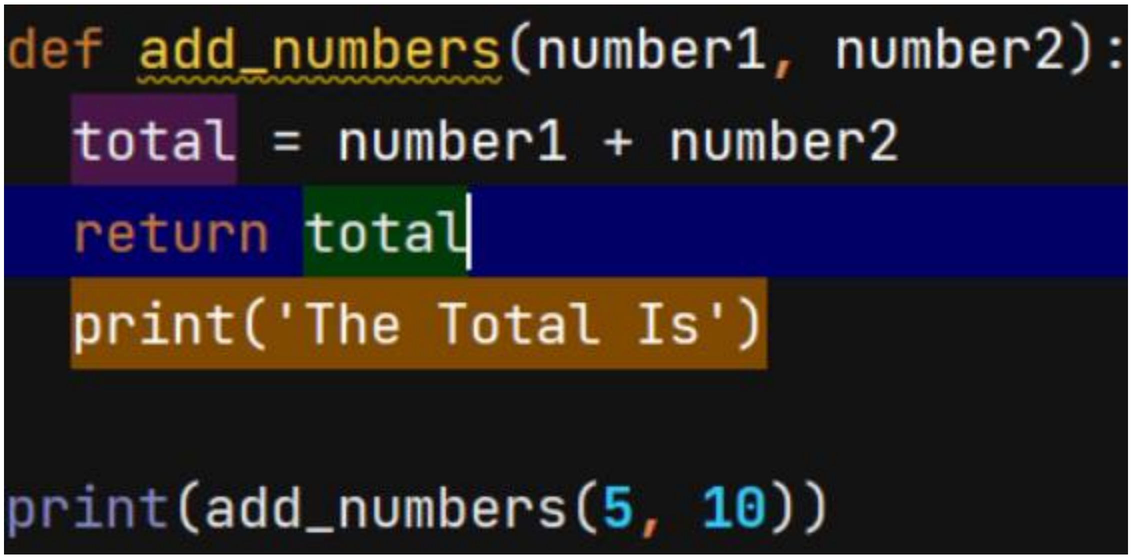
```
def add_numbers(number):
  return


print(add_numbers(5))
```



```
None
```

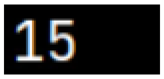
We see None in the console. Like the Parameters and Arguments tutorial session, we can also use our defined function to accept more than 1 parameter. The parameters will be number1, number2. Let's also add total = number1 + number2. Let's return a value and not an expression. The value will be total. Now when we run, we are going to call, this number1 parameter and number2 parameter but leave 5 as the number 1 argument and 10 for number2.

We must be careful to not generate unreachable code with a return statement. Unreachable code is also called dead code because the program will never execute that code. After the return statement, we write print("The Total Is ") then run.



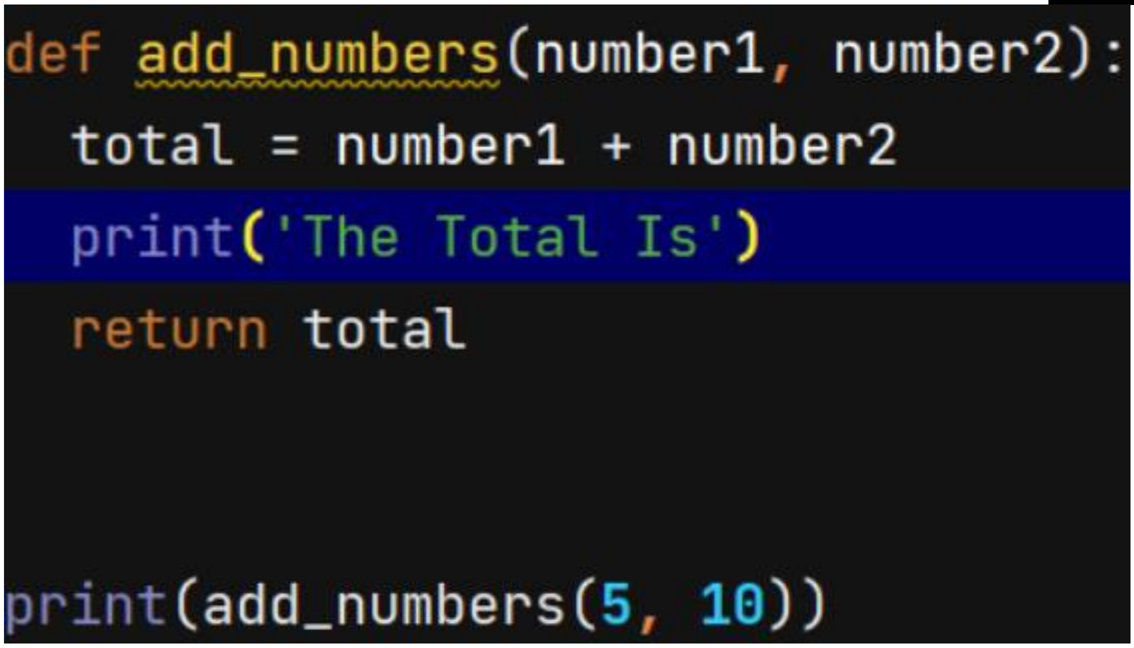
```
def add_numbers(number1, number2):
  total = number1 + number2
  return total
  print('The Total Is')


print(add_numbers(5, 10))
```

All we see is 15.



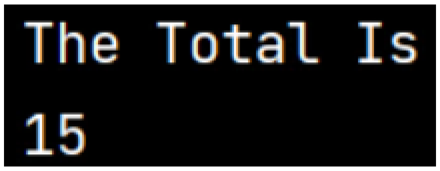
```
15
```

Hover the print statement and it says "This code is unreachable". Move the print statement before the return statement then Run.



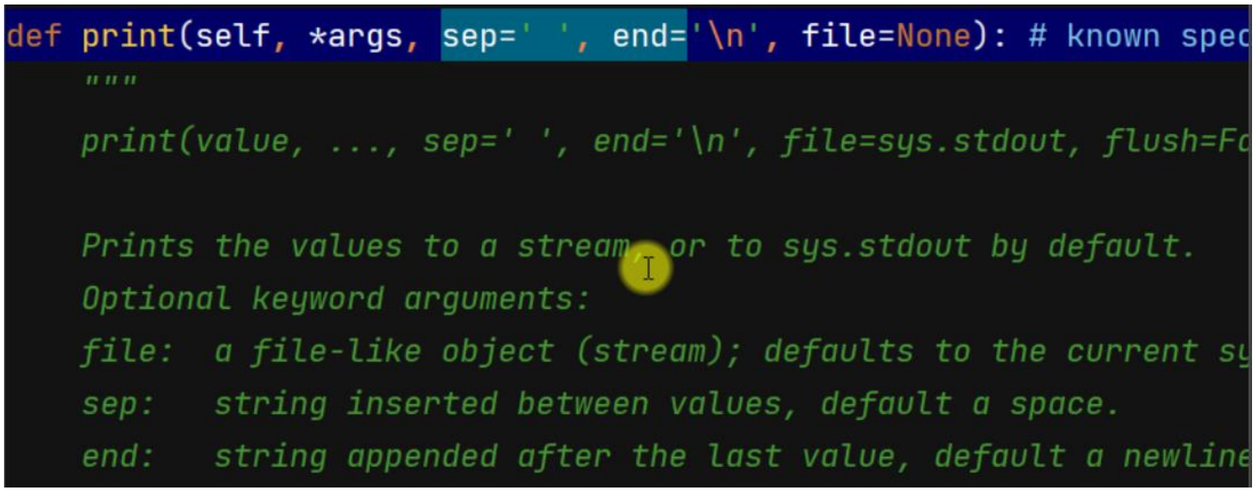
```
def add_numbers(number1, number2):
  total = number1 + number2
  print('The Total Is')
  return total


print(add_numbers(5, 10))
```

The console shows “The Total Is 15” but 15 is on a separate line.



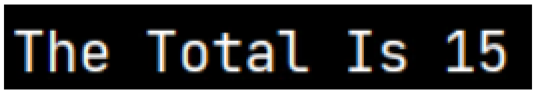
```
The Total Is
15
```

Let me show you how to print 15 be on the same line using a keyword argument. Do you recall in one of the previous sessions when we investigated the built-in print() function? It has parameters as sep and end. The docstring comment shows sep is a string inserted between the values, default a space. If we insert a comma then a comma will be inserted between each value. However, end is a string appended after the last value, default a newline.



```
def print(self, *args, sep=' ', end='\n', file=None): # known spec
    """
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=Fa

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sy
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline
```

Let’s go back to our program then add a comma end=””. This will no longer insert a new line. One more concept I want to show you. Right now, we are calling the function with print(add_number(5, 10)). If you want to, you can store the returned value in a variable by writing result = add_numbers(5, 10). We will get 15 from the defined function after adding 10 + 5. Next, the total 15 is returned to add_numbers(5, 10). Now, when we store the result into a variable. Let’s print(result). Execute and we see “The Total Is 15” on the same line.



```
The Total Is 15
```

Contact Info

- ✓ Email Rex.Jones@Test4Success.org
- ✓ YouTube https://www.youtube.com/c/RexJonesII/videos
- ✓ Facebook https://facebook.com/JonesRexII
- ✓ Twitter https://twitter.com/RexJonesII
- ✓ GitHub https://github.com/RexJonesII/Free-Videos
- ✓ LinkedIn https://www.linkedin.com/in/rexjones34/