Week 3 Lecture Recap

Lectures Included:

1. [Control Flow in Python](#_pfngv7dtvbzf) (April 30)
2. [Functions Revisited](#_yo904iqxzx65) (May 3) \*
3. [More Functions and Parameters](#_wahfnhpykusu) (May 5) \*

*\*Note that material from functions (May 3 and May 5) will not be the focus of this week's section*

# Control Flow in Python

Covered in this lecture:

1. [Conditions](#_bi3r3xxcpc5q)
2. [Guess Num and Sentinel Sum](#_1s7z21cxj7ox)
3. [Booleans](#_xemq8infnx6v)
4. [For Loops](#_l53hugn3m4wm)

## Conditions

While-loops and if-statements use conditions:

|  |  |
| --- | --- |
| **while** *condition*:  body | **if** *condition*:  body |

The condition should be a “boolean” which evaluates to either **True** or **False**

### Booleans

A boolean is an expression that evaluates to either **True** or **False**

Here is an example of a boolean expression: 1 < 2

* It asks the question: is the thing on the left smaller than the thing on the right?

Comparison Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example** | **Value** |
| == | equals | 1 + 1 == 2 | True |
| != | does not equal | 3.2 != 2.5 | True |
| < | less than | 10 < 5 | False |
| > | greater than | 10 > 5 | True |
| <= | less than or equal to | 126 <= 100 | False |
| >= | greater than or equal to | 5.0 >= 5.0 | True |

We can use comparison operators to make conditions:

For example:

|  |
| --- |
| **if** 1 < 2:  print("1 is less than 2") |

Another example:

|  |
| --- |
| num = int(input("Enter a number: "))  **if** num == 0:  print("That number is 0")  **else**:  print("That number is not 0") |

### If Else Revisited

**elif** is short for else if.

Here's an example of how it's used:

|  |
| --- |
| num = int(input("Enter a number: "))  **if** num == 0:  print("Your number is 0")  **elif** num > 0:  print("Your number is positive")  **else**:  print("Your number is negative") |

## GuessNum and Sentinel Sum

Here's a game called GuessMyNumber. When the program starts, the computer will choose a random number. Then you will guess the number, and it will tell you whether it's too high or too low until you get it right.

Here's a sample output of the GuessMyNumber game:

|  |
| --- |
| I am thinking of a number between 0 and 99…  Enter a guess: 50  Your guess is too high  Enter a new number: 25  Your guess is too high  Enter a new number: 40  Your guess is too low  Enter a new number: 45  Your guess is too low  Enter a new number: 48  Congrats! The number was: 48 |

Here's what the code looks like for this program, with comments to describe what each line does:

|  |
| --- |
| #generate a random number, store it in the variable secret\_number  secret\_number = random.randint(1, 99)  print("I am thinking of a number between 1 and 99…")  #get our first guess from the user  guess = int(input("Enter a guess: "))  # True if guess is not equal to secret number  while guess != secret\_number:  # True if guess is less than secret number  if guess < secret\_number:  print("Your guess is too low")  else:  print("Your guess is too high")  print("") # an empty line  # get another guess from the user  guess = int(input("Enter a new guess: "))  print("Congrats! The number was: " + str(secret\_number)) |

### Sentinel

**Sentinel**: A value that signals the end of user input

* **sentinel loop**: repeats until a sentinel value is seen

An example: Write a program that prompts the user for numbers until the user types -1, then output the total of the numbers

* In this case, -1 is the **sentinel value**

An example run of this program:

|  |
| --- |
| Type a number: 10  Type a number: 20  Type a number: 30  Type a number: -1  total is 60 |

Here's the code for this game, along with comments to explain what each line does:

|  |
| --- |
| def main():  #total is a variable to track the sum of the numbers  #at the start of the program, we initialize total to be 0  total = 0    # we get the first number from the user  num = int(input("Type a number: "))  # we have to first get a value for num because the next line of  # code checks for num's value  # -1 is our sentinel value. Our loop will stop if num == -1  while num != -1:  # get another number for the user  num = int(input("Type a number: "))    # update total to include our new number  total = total + num  print(total) #print out the total once the user enters -1 |

## Booleans

### Logical Operators

In order of precedence:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Result** |
| **not** | **not** (2 == 3) | **True** |
| **and** | (2 ==3) **and** (-1 < 5) | **False** |
| **or** | (2 == 3) **or** (-1 < 5) | **True** |

### Boolean Variables

Just like how you can have variables that store integer or string values, you can also have variables that store boolean values.

Here are some examples:

|  |
| --- |
| # Store expressions that evaluate to True/False  x = 1 < 2 #True  y = 5.0 == 4.0 #False  # Directly set to True/False  is\_sheltering = True  is\_raining = False  play\_again = input('Play again? "y" or "n"') == 'y'  if play again:  ... |

### For Loops

A for-loop repeats code for a specified number of times. For example, here's the code you could write to print "Python rocks socks!" 100 times:

for i in range(100):

print("Python rocks socks!")

What does the i in the for-loop mean?

i is a variable that keeps track of how many times the loop has run. In our loop above, for i in range(100) means that at the start of the loop, i is 0. Each time the for-loop runs, i increases by 1 and keeps going while i is less than 100. Once i reaches 100, the loop stops running.

Here's the equivalent of the for-loop above, written as a while-loop:

i = 0

while i < 100:

print("Python rocks socks!")

i += 1

Now that we know that i is a variable, we can see how we can use i inside our for-loop.

For example, here's how we can print even numbers:

for i in range(3):

print(i \* 2)

Here's another way to print even numbers:

for i in range(0, 6, 2):

print(i)

for i in range(0, 6, 2) says: i goes from 0 to 6, counting by 2 each time

*\*\*\* The following material is covered in lectures this week, but is not a main point of section \*\*\**

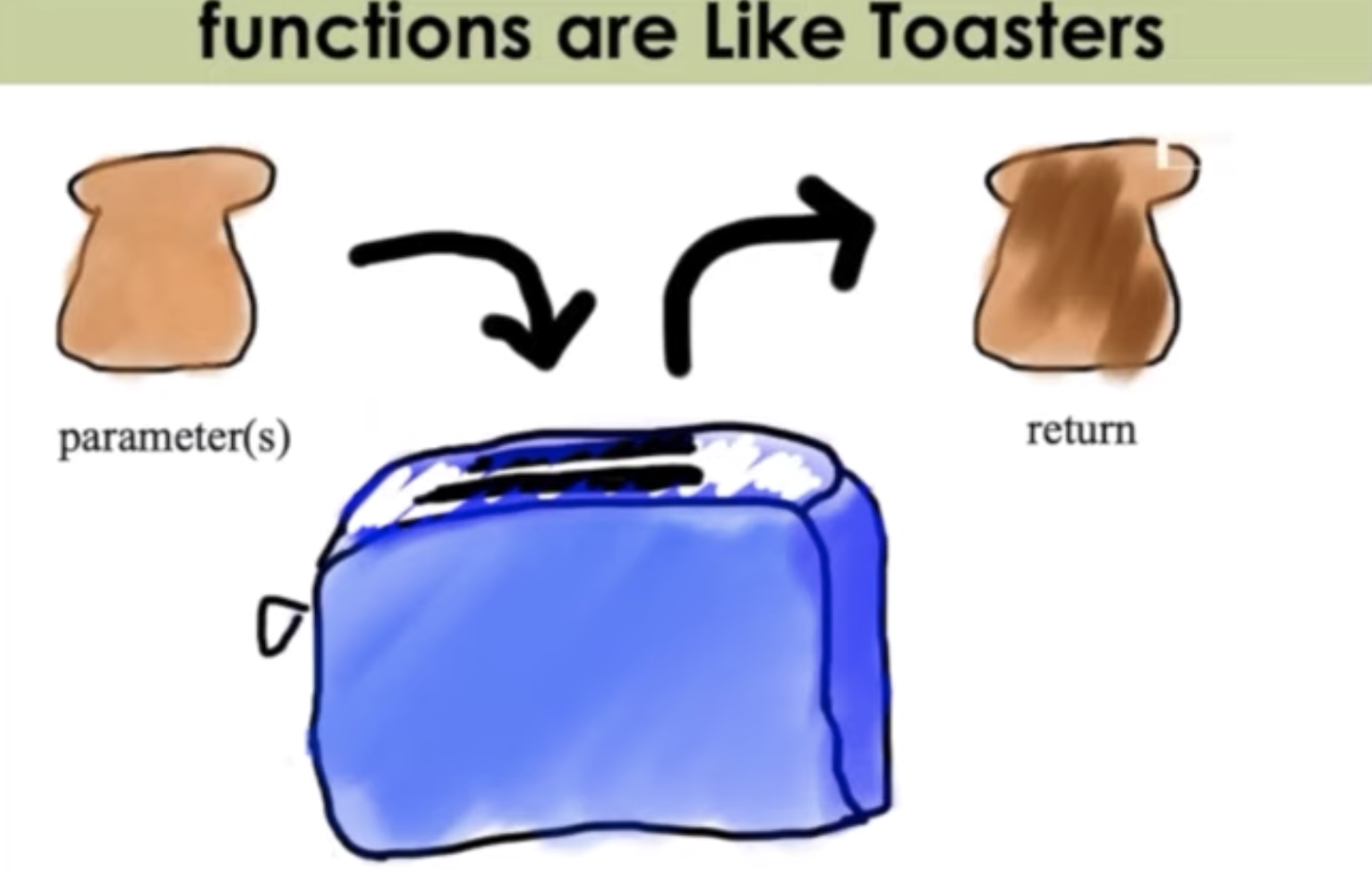
# Functions Revisited

Covered in this lecture:

1. [Analogy: Functions are like toasters](#_cq06v7u9s9n)
2. [Anatomy of a Function](#_xzmuh8ikr705)

## Analogy: Functions are like toasters

Just like toasters, functions can take in an input, change the input, and return a different output based on the input.

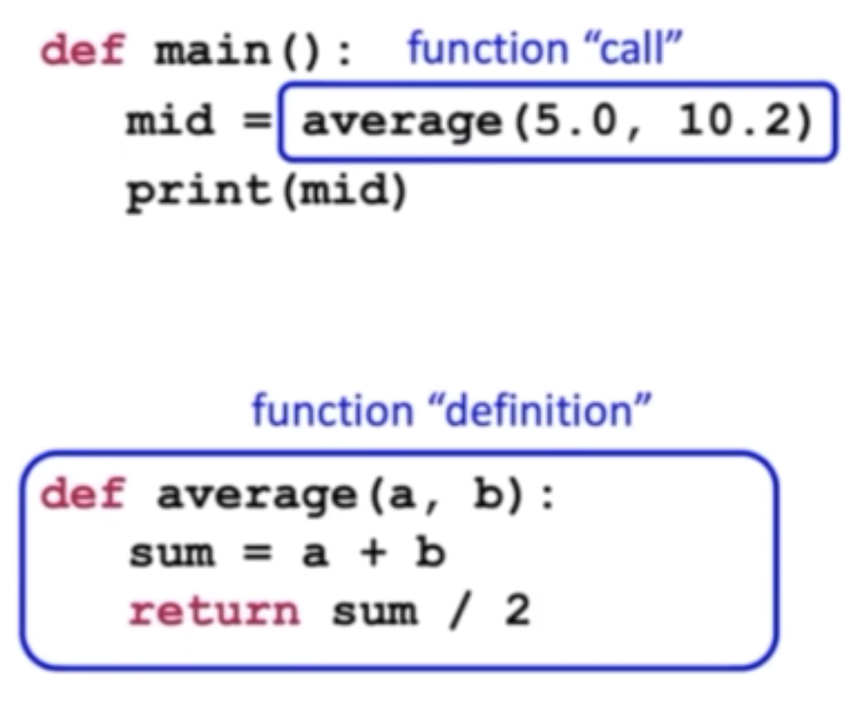


A function is also able to take in different parameters. For example, if you wanted to toast a bagel, you wouldn't get a new toaster. You can use the same toaster to toast bagels, too, in addition to bread.

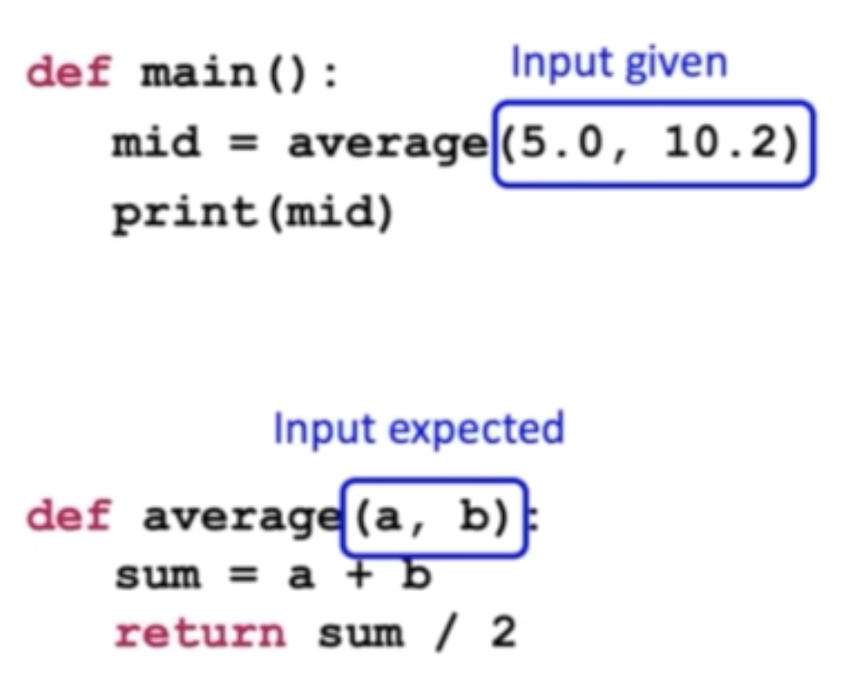
|  |  |
| --- | --- |
|  |  |

## Anatomy of a Function

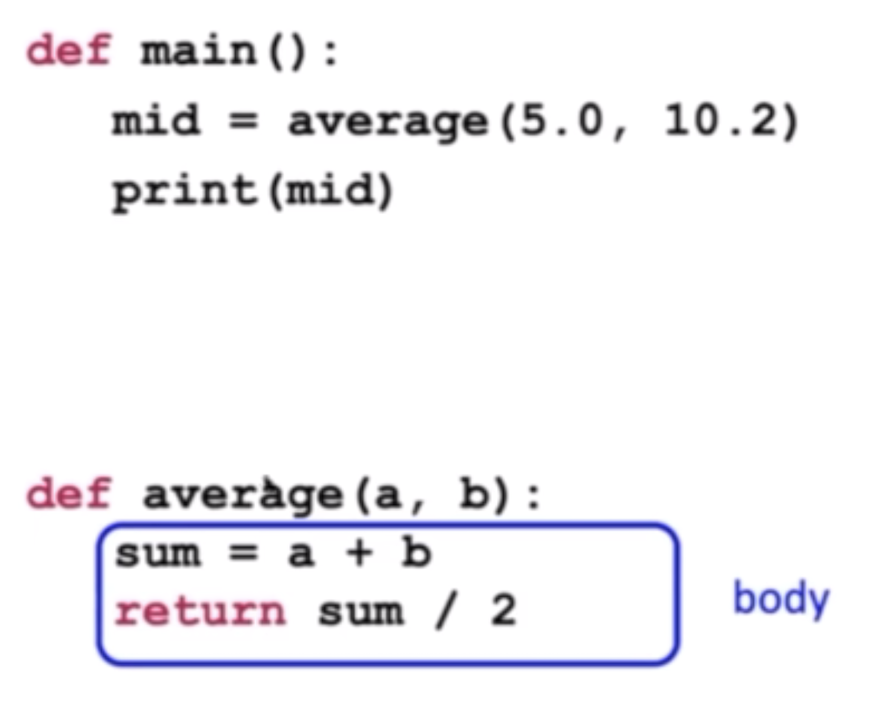
Starting from main, our function is called:



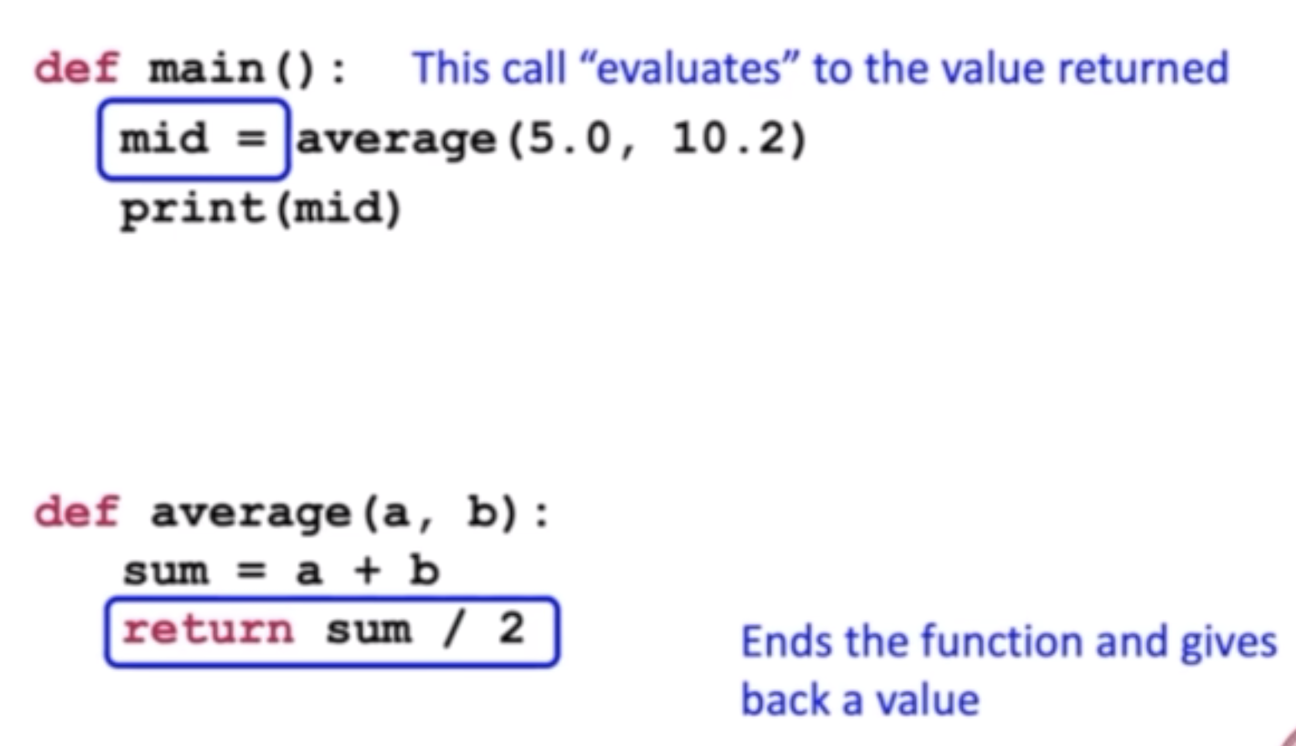
Inputs & parameters:



Now, that we have our parameters, we can go inside the body of our function:



At the end of the function, we return a value, which is saved in the mid variable in main



# More Functions and Parameters

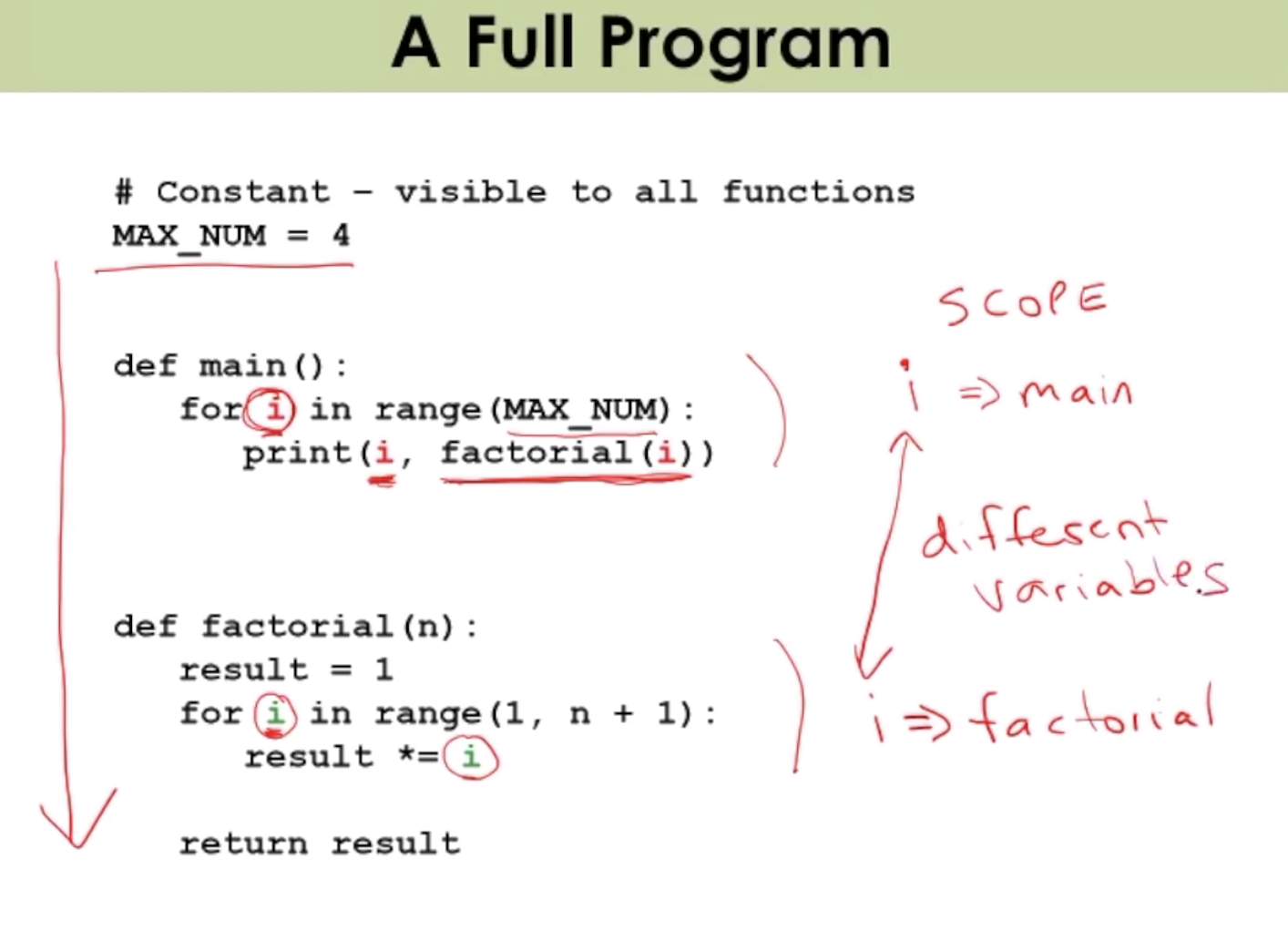
Covered in this lecture:

1. [Factorial & Variable Scope](#_g97kpyz7if2i)
2. [DocTests](#_axjzpqugvne0)

## Factorial & Variable Scope

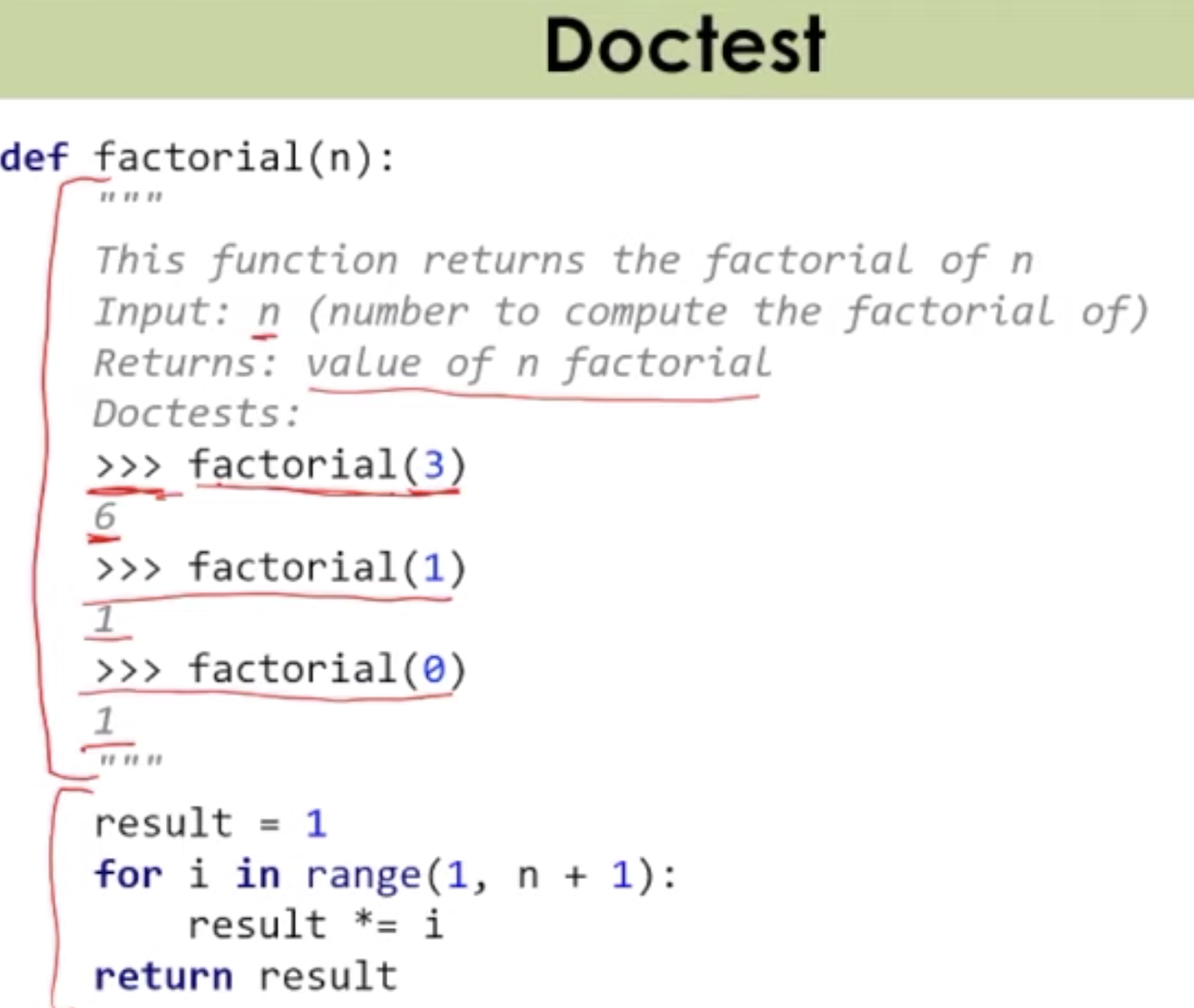
A scope of a variable is where in the program a variable lives, and where it can be accessed.

The scope of a variable is the function it is defined in. For example, in the following program that prints the factorial of a number, there are two variables named i: one in main, and one in our function called factorial. Even though the variables i have the same name, refer to different things.



## DocTests

In Python, you can use DocTests to test if your functions work the way you expect them to. You write your doctests at the top of the function. Here's how we can text our function factorial:



To run our doctests, type this in the terminal: python -m doctest filename.py -v