Python course materials

# Functions

## Introduction to Functions

This lecture will consist of explaining what a function is in Python and how to create one. Functions will be one of our main building blocks when we construct larger and larger amounts of code to solve problems.

### What is a function?

Formally, a function is a useful device that groups together a set of statements so they can be run more than once. They can also let us specify parameters that can serve as inputs to the functions.

On a more fundamental level, functions allow us to not have to repeatedly write the same code again and again. If you remember back to the lessons on strings and lists, remember that we used a function len() to get the length of a string. Since checking the length of a sequence is a common task you would want to write a function that can do this repeatedly at command.

Functions will be one of most basic levels of reusing code in Python, and it will also allow us to start thinking of program design (we will dive much deeper into the ideas of design when we learn about Object Oriented Programming).

### Why even use functions?

Put simply, you should use functions when you plan on using a block of code multiple times. The function will allow you to call the same block of code without having to write it multiple times. This in turn will allow you to create more complex Python scripts. To really understand this though, we should actually write our own functions!

## Function Topics

* def keyword
* simple example of a function
* calling a function with ()
* accepting parameters
* print versus return
* adding in logic inside a function
* multiple returns inside a function
* adding in loops inside a function
* tuple unpacking
* interactions between functions

### def keyword

Let’s see how to build out a function’s syntax in Python. It has the following form:

def name\_of\_function(arg1,arg2):  
 '''  
 This is where the function's Document String (docstring) goes.  
 When you call help() on your function it will be printed out.  
 '''  
 # Do stuff here  
 # Return desired result

We begin with def then a space followed by the name of the function. Try to keep names relevant, for example len() is a good name for a length() function. Also be careful with names, you wouldn’t want to call a function the same name as a [built-in function in Python](https://docs.python.org/3/library/functions.html) (such as len).

Next come a pair of parentheses with a number of arguments separated by a comma. These arguments are the inputs for your function. You’ll be able to use these inputs in your function and reference them. After this you put a colon.

Now here is the important step, you must indent to begin the code inside your function correctly. Python makes use of *whitespace* to organize code. Lots of other programing languages do not do this, so keep that in mind.

Next you’ll see the docstring, this is where you write a basic description of the function. Using Jupyter and Jupyter Notebooks, you’ll be able to read these docstrings by pressing Shift+Tab after a function name. Docstrings are not necessary for simple functions, but it’s good practice to put them in so you or other people can easily understand the code you write.

After all this you begin writing the code you wish to execute.

The best way to learn functions is by going through examples. So let’s try to go through examples that relate back to the various objects and data structures we learned about before.

### Simple example of a function

def say\_hello():  
 print('hello')

### Calling a function with ()

Call the function:

say\_hello()

hello

If you forget the parenthesis (), it will simply display the fact that say\_hello is a function. Later on we will learn we can actually pass in functions into other functions! But for now, simply remember to call functions with ().

say\_hello

<function \_\_main\_\_.say\_hello>

### Accepting parameters (arguments)

Let’s write a function that greets people with their name.

def greeting(name):  
 print(f'Hello {name}')

greeting('Jose')

Hello Jose

## Using return

So far we’ve only seen print() used, but if we actually want to save the resulting variable we need to use the **return** keyword.

Let’s see some example that use a return statement. return allows a function to *return* a result that can then be stored as a variable, or used in whatever manner a user wants.

### Example: Addition function

def add\_num(num1,num2):  
 return num1+num2

add\_num(4,5)

9

# Can also save as variable due to return  
result = add\_num(4,5)

print(result)

9

What happens if we input two strings?

add\_num('one','two')

'onetwo'

## Very Common Question: “What is the difference between *return* and *print*?”

**The return keyword allows you to actually save the result of the output of a function as a variable. The print() function simply displays the output to you, but doesn’t save it for future use. Let’s explore this in more detail**

def print\_result(a,b):  
 print(a+b)

def return\_result(a,b):  
 return a+b

print\_result(10,5)

15

# You won't see any output if you run this in a .py script  
return\_result(10,5)

15

**But what happens if we actually want to save this result for later use?**

my\_result = print\_result(20,20)

40

my\_result

type(my\_result)

NoneType

**Be careful! Notice how print\_result() doesn’t let you actually save the result to a variable! It only prints it out, with print() returning None for the assignment!**

my\_result = return\_result(20,20)

my\_result

40

my\_result + my\_result

80

# Adding Logic to Internal Function Operations

So far we know quite a bit about constructing logical statements with Python, such as if/else/elif statements, for and while loops, checking if an item is **in** a list or **not in** a list (Useful Operators Lecture). Let’s now see how we can perform these operations within a function.

### Check if a number is even

**Recall the mod operator % which returns the remainder after division, if a number is even then mod 2 (% 2) should be == to zero.**

2 % 2

0

20 % 2

0

21 % 2

1

20 % 2 == 0

True

21 % 2 == 0

False

\*\* Let’s use this to construct a function. Notice how we simply return the boolean check.\*\*

def even\_check(number):  
 return number % 2 == 0

even\_check(20)

True

even\_check(21)

False

### Check if any number in a list is even

Let’s return a boolean indicating if **any** number in a list is even. Notice here how **return** breaks out of the loop and exits the function

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # Otherwise we don't do anything  
 else:  
 pass

\*\* Is this enough? NO! We’re not returning anything if they are all odds!\*\*

check\_even\_list([1,2,3])

True

check\_even\_list([1,1,1])

\*\* VERY COMMON MISTAKE!! LET’S SEE A COMMON LOGIC ERROR, NOTE THIS IS WRONG!!!\*\*

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # This is WRONG! This returns False at the very first odd number!  
 # It doesn't end up checking the other numbers in the list!  
 else:  
 return False

# UH OH! It is returning False after hitting the first 1  
check\_even\_list([1,2,3])

False

\*\* Correct Approach: We need to initiate a return False AFTER running through the entire loop\*\*

def check\_even\_list(num\_list):  
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we return True  
 if number % 2 == 0:  
 return True  
 # Don't do anything if its not even  
 else:  
 pass  
 # Notice the indentation! This ensures we run through the entire for loop   
 return False

check\_even\_list([1,2,3])

True

check\_even\_list([1,3,5])

False

### Return all even numbers in a list

Let’s add more complexity, we now will return all the even numbers in a list, otherwise return an empty list.

def check\_even\_list(num\_list):  
   
 even\_numbers = []  
   
 # Go through each number  
 for number in num\_list:  
 # Once we get a "hit" on an even number, we append the even number  
 if number % 2 == 0:  
 even\_numbers.append(number)  
 # Don't do anything if its not even  
 else:  
 pass  
 # Notice the indentation! This ensures we run through the entire for loop   
 return even\_numbers

check\_even\_list([1,2,3,4,5,6])

[2, 4, 6]

check\_even\_list([1,3,5])

[]

## Returning Tuples for Unpacking

\*\* Recall we can loop through a list of tuples and “unpack” the values within them\*\*

stock\_prices = [('AAPL',200),('GOOG',300),('MSFT',400)]

for item in stock\_prices:  
 print(item)

('AAPL', 200)  
('GOOG', 300)  
('MSFT', 400)

for stock,price in stock\_prices:  
 print(stock)

AAPL  
GOOG  
MSFT

for stock,price in stock\_prices:  
 print(price)

200  
300  
400

**Similarly, functions often return tuples, to easily return multiple results for later use.**

Let’s imagine the following list:

work\_hours = [('Abby',100),('Billy',400),('Cassie',800)]

The employee of the month function will return both the name and number of hours worked for the top performer (judged by number of hours worked).

def employee\_check(work\_hours):  
   
 # Set some max value to intially beat, like zero hours  
 current\_max = 0  
 # Set some empty value before the loop  
 employee\_of\_month = ''  
   
 for employee,hours in work\_hours:  
 if hours > current\_max:  
 current\_max = hours  
 employee\_of\_month = employee  
 else:  
 pass  
   
 # Notice the indentation here  
 return (employee\_of\_month,current\_max)

employee\_check(work\_hours)

('Cassie', 800)

## Interactions between functions

Functions often use results from other functions, let’s see a simple example through a guessing game. There will be 3 positions in the list, one of which is an ‘O’, a function will shuffle the list, another will take a player’s guess, and finally another will check to see if it is correct. This is based on the classic carnival game of guessing which cup a red ball is under.

**How to shuffle a list in Python**

example = [1,2,3,4,5]

from random import shuffle

# Note shuffle is in-place  
shuffle(example)

example

[3, 1, 4, 5, 2]

**OK, let’s create our simple game**

mylist = [' ','O',' ']

def shuffle\_list(mylist):  
 # Take in list, and returned shuffle versioned  
 shuffle(mylist)  
   
 return mylist

mylist

[' ', 'O', ' ']

shuffle\_list(mylist)

[' ', ' ', 'O']

def player\_guess():  
   
 guess = ''  
   
 while guess not in ['0','1','2']:  
   
 # Recall input() returns a string  
 guess = input("Pick a number: 0, 1, or 2: ")  
   
 return int(guess)

player\_guess()

Pick a number: 0, 1, or 2: 1  
  
  
  
  
  
1

Now we will check the user’s guess. Notice we only print here, since we have no need to save a user’s guess or the shuffled list.

def check\_guess(mylist,guess):  
 if mylist[guess] == 'O':  
 print('Correct Guess!')  
 else:  
 print('Wrong! Better luck next time')  
 print(mylist)

Now we create a little setup logic to run all the functions. Notice how they interact with each other!

# Initial List  
mylist = [' ','O',' ']  
  
# Shuffle It  
mixedup\_list = shuffle\_list(mylist)  
  
# Get User's Guess  
guess = player\_guess()  
  
# Check User's Guess  
#------------------------  
# Notice how this function takes in the input   
# based on the output of other functions!  
check\_guess(mixedup\_list,guess)

Pick a number: 0, 1, or 2: 1  
Wrong! Better luck next time  
[' ', ' ', 'O']

Great! You should now have a basic understanding of creating your own functions to save yourself from repeatedly writing code!