# Python Data Science Toolbox (Part 1)

* 3 hours
* 12 Videos
* 46 Exercises
* 346,950 Participants
* 3,650 XP

### Course Description

It's time to push forward and develop your Python chops even further. There are tons of fantastic functions in Python and its library ecosystem. However, as a data scientist, you'll constantly need to write your own functions to solve problems that are dictated by your data. You will learn the art of function writing in this first Python Data Science Toolbox course. You'll come out of this course being able to write your very own custom functions, complete with multiple parameters and multiple return values, along with default arguments and variable-length arguments. You'll gain insight into scoping in Python and be able to write lambda functions and handle errors in your function writing practice. And you'll wrap up each chapter by using your new skills to write functions that analyze Twitter DataFrames.

1. 1

#### Writing your own functions

0%

In this chapter, you'll learn how to write simple functions, as well as functions that accept multiple arguments and return multiple values. You'll also have the opportunity to apply these new skills to questions commonly encountered by data scientists.

##### User-defined functions

50 xp

##### Strings in Python

50 xp

##### Recapping built-in functions

50 xp

##### Write a simple function

100 xp

##### Single-parameter functions

100 xp

##### Functions that return single values

100 xp

##### Multiple parameters and return values

50 xp

##### Functions with multiple parameters

100 xp

##### A brief introduction to tuples

100 xp

##### Functions that return multiple values

100 xp

##### Bringing it all together

50 xp

##### Bringing it all together (1)

100 xp

##### Bringing it all together (2)

100 xp

##### Congratulations!

**Daily XP150**

# User-defined functions

**50 XP**

This course is also available on the mobile app

## 1. User-defined functions

Welcome to the course! My name is Hugo Bowne-Anderson and I am a Data Scientist at DataCamp. In this course, the first of the Python Data Science Toolbox courses, you'll learn to write your very own functions and you'll have the opportunity to apply these newfound skills to questions that commonly arise in Data Science contexts.

## 2. You’ll learn:

Specifically, in this video and in the interactive exercises that follow it, you will learn to do the following: define functions without parameters, define functions with single parameters, and define functions that return a single value. In the next section, you'll learn how to pass multiple arguments to functions, as well as return multiple values from them. Let's begin!

## 3. Built-in functions

Let's check out Python's built-in function str, which accepts an object such as a number and returns a string object. You can assign a call to str to a variable to store its return value. While built-in Python functions are cool, as a Data Scientist, you'll need functions that have functionality specific to your needs. Fortunately, you can define your own functions in Python!

## 4. Defining a function

We'll now see how to define functions via an example, a function that squares a number. The function name square will be perfect for this. To define the function, We begin with the keyword def, followed by the function name square; this is then followed by a set of parentheses and a colon. This piece of code is called a function header. To complete the function definition, let's write the function body by squaring a value, say 4, and printing the output. Right now, our square function does not have any parameters within the parentheses. We will add them later. Now, whenever this function is called, the code in the function body is run. In this case, new\_value is assigned the value of 4 \*\* 2 and then printed out. You can call the function as you do with pre-built functions: square. This should yield the value, 16.

## 5. Function parameters

What if you wanted to square any other number besides 4, though? To add that functionality, you add a parameter to the function definition in between the parentheses. Here you see that we've added a parameter value and in the new function body, the variable new\_value takes the square of value, which is then printed out. We can now square any number that we pass to the function square as an argument. A quick word on parameters and arguments: when you define a function, you write parameters in the function header. When you call a function, you pass arguments into the function.

## 6. Return values from functions

The function square now accepts a single parameter and prints out its squared value. But what if we don't want to print that value directly and instead we want to return the squared value and assign it to some variable? You can have your function return the new value by adding the return keyword, followed by the value to return. Now we can assign to a variable num the result of the function call as you see here.

## 7. Docstrings

There's another essential aspect of writing functions in Python: docstrings. Docstrings are used to describe what your function does, such as the computations it performs or its return values. These descriptions serve as documentation for your function so that anyone who reads your function's docstring understands what your function does, without having to trace through all the code in the function definition. Function docstrings are placed in the immediate line after the function header and are placed in between triple quotation marks. An appropriate Docstring for our function square is 'Returns the square of a value'.

## 8. Let's practice!

You've now just learned the basics of defining your own functions! Now it's your turn. In the next few exercises, you will try your hand at defining and using your own functions.

**Daily XP200**

##### Exercise

##### Exercise

# Strings in Python

In the video, you learned of another standard Python datatype, **strings**. Recall that these represent textual data. To assign the string 'DataCamp' to a variable company, you execute:

company = 'DataCamp'

You've also learned to use the operations + and \* with strings. Unlike with numeric types such as ints and floats, the + operator concatenates strings together, while the \* concatenates multiple copies of a string together. In this exercise, you will use the + and \* operations on strings to answer the question below. Execute the following code in the shell:

object1 = "data" + "analysis" + "visualization"

object2 = 1 \* 3

object3 = "1" \* 3

What are the values in object1, object2, and object3, respectively?

##### Instructions

**50 XP**

##### Possible Answers

* 

object1 contains "data + analysis + visualization", object2 contains "1\*3", object3 contains 13.

* 

object1 contains "data+analysis+visualization", object2 contains 3, object3 contains "13".

* 

object1 contains "dataanalysisvisualization", object2 contains 3, object3 contains "111".

Correct answer is **c**

##### Possible Answers

* 

They are all str types.

* 

x is a float, y1 is an float, and y2 is a str.

* 

x is a float, y1 is a str, and y2 is a NoneType.

* 

They are all NoneType types.

x is a float; y1 is a str and y2 is a NoneType

In [1]:

y1 = str(x)

In [2]:

y2 = print(x)

4.89

In [3]:

type(y1)

Out[3]:

str

In [4]:

type(y2)

Out[4]:

NoneType

In [5]:

type(x)

**Daily XP300**

##### Exercise

##### Exercise

# Write a simple function

In the last video, Hugo described the basics of how to define a function. You will now write your own function!

Define a function, shout(), which simply prints out a string with three exclamation marks '!!!' at the end. The code for the square() function that we wrote earlier is found below. You can use it as a pattern to define shout().

def square():

new\_value = 4 \*\* 2

return new\_value

Note that the function body is indented 4 spaces already for you. Function bodies need to be indented by a consistent number of spaces and the choice of 4 is common.

This course touches on a lot of concepts you may have forgotten, so if you ever need a quick refresher, download the [*Python for Data Science Cheat Sheet*](https://datacamp-community-prod.s3.amazonaws.com/0eff0330-e87d-4c34-88d5-73e80cb955f2) and keep it handy!

##### Instructions

**100 XP**

* Complete the function header by adding the appropriate function name, shout.
* In the function body, **concatenate** the string, 'congratulations' with another string, '!!!'. Assign the result to shout\_word.
* Print the value of shout\_word.
* Call the shout function.

# Define the function shout

def \_\_\_\_():

    """Print a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    # Print shout\_word

    print(\_\_\_\_)

# Call shout

# Define the function shout

def shout():

    """Print a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    shout\_word = 'congratulations' + '!!!'

    # Print shout\_word

    print(shout\_word)

# Call shout

shout()

# Define the function shout

def shout():

"""Print a string with three exclamation marks"""

# Concatenate the strings: shout\_word

shout\_word = 'congratulations' + '!!!'

# Print shout\_word

print(shout\_word)

# Call shout

shout()

congratulations!!!

<script.py> output:

congratulations!!!



**Daily XP400**

##### Exercise

##### Exercise

# Single-parameter functions

Congratulations! You have successfully defined and called your own function! That's pretty cool.

In the previous exercise, you defined and called the function shout(), which printed out a string concatenated with '!!!'. You will now update shout() by adding a parameter so that it can accept and process any string argument passed to it. Also note that shout(word), the part of the header that specifies the function name and parameter(s), is known as the signature of the function. You may encounter this term in the wild!

##### Instructions

**100 XP**

* Complete the function header by adding the parameter name, word.
* Assign the result of concatenating word with '!!!' to shout\_word.
* Print the value of shout\_word.
* Call the shout() function, passing to it the string, 'congratulations'.

# Define shout with the parameter, word

def shout(\_\_\_\_):

    """Print a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    \_\_\_\_ = \_\_\_\_ + '!!!'

    # Print shout\_word

    print(\_\_\_\_)

# Call shout with the string 'congratulations'

# Define the function shout def shout(): """Print a string with three exclamation marks""" # Concatenate the strings: shout\_word shout\_word = 'congratulations' + '!!!' # Print shout\_word print(shout\_word) # Call shout shout()

# Define shout with the parameter, word

def shout(word):

    """Print a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    shout\_word = word + '!!!'

    # Print shout\_word

    print(shout\_word)

# Define shout with the parameter, word

def shout(word):

"""Print a string with three exclamation marks"""

# Concatenate the strings: shout\_word

shout\_word = word + '!!!'

# Print shout\_word

print(shout\_word)

# Call shout with the string 'congratulations'

shout('congratulations')

congratulations!!!

<script.py> output:

congratulations!!!

# Call shout with the string 'congratulations'

shout('congratulations')

# Functions that return single values

You're getting very good at this! Try your hand at another modification to the shout() function so that it now returns a single value instead of printing within the function. Recall that the return keyword lets you return values from functions. Parts of the function shout(), which you wrote earlier, are shown. Returning values is generally more desirable than printing them out because, as you saw earlier, a print() call assigned to a variable has type NoneType.

##### Instructions

**100 XP**

* In the function body, concatenate the string in word with '!!!' and assign to shout\_word.
* Replace the print() statement with the appropriate return statement.
* Call the shout() function, passing to it the string, 'congratulations', and assigning the call to the variable, yell.
* To check if yell contains the value returned by shout(), print the value of yell.

# Define shout with the parameter, word

def shout(word):

    """Return a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    # Replace print with return

    print(shout\_word)

# Pass 'congratulations' to shout: yell

# Print yell

# Define shout with the parameter, word def shout(word): """Print a string with three exclamation marks""" # Concatenate the strings: shout\_word shout\_word = word + '!!!' # Print shout\_word print(shout\_word) # Call shout with the string 'congratulations' shout('congratulations')

# Define shout with the parameter, word

def shout(word):

    """Return a string with three exclamation marks"""

    # Concatenate the strings: shout\_word

    shout\_word = word + '!!!'

    # Replace print with return

    return shout\_word

# Pass 'congratulations' to shout: yell

yell = shout('congratulations')

# Print yell

print(yell)

# Define shout with the parameter, word

def shout(word):

"""Return a string with three exclamation marks"""

# Concatenate the strings: shout\_word

shout\_word = word + '!!!'

# Replace print with return

return shout\_word

# Pass 'congratulations' to shout: yell

yell = shout('congratulations')

# Print yell

print(yell)

congratulations!!!

<script.py> output:

congratulations!!!

This course is also available on the mobile app

## 1. Multiple Parameters and Return Values

Welcome back! You're doing great at defining your own functions. Good job! At this point, you already know how to define your own functions and even return values from them. What you'll learn next is how to pass multiple arguments to functions, as well as return not just one, but multiple values from them.

## 2. Multiple function parameters

Let's tweak the square function we've been working on a little bit more. Suppose that, instead of simply squaring a value, we'd like to raise a value to the power of another value that's also passed to the function. We can do this by having our function accept two parameters instead of just one. You should also change your function name AND docstrings to reflect this new behavior. raise\_to\_power is an appropriate function name. Notice that there are now two parameters in the function header instead of one, value1 and value2. In the lines after that, the behavior of the overall function was also changed by raising value1 to the power of value2. You can call the function by passing in two arguments because the function has two parameters, as declared in the function header. The order in which the arguments are passed correspond to the order of the parameters in the function header. This means that when we call raise\_to\_power(2, 3), when the function is executed, 2 would be assigned to value1 and 3 to value2. Looking at the function body, this means that the computation value1 to the power of value2 translates to 2 to the power of 3. This function call then returns the value 8.

## 3. A quick jump into tuples

You can also make your function return multiple values. You can do that by constructing objects known as tuples in your functions. A tuple is like a list, in that it can contain multiple values. There are some differences, however. Firstly, unlike a list, a tuple is immutable, that is, you cannot modify the values in a tuple once it has been constructed. Secondly, while lists are defined using square brackets, tuples are constructed using a set of parentheses.

## 4. Unpacking tuples

Here we construct a tuple containing 3 elements. You can also unpack a tuple into several variables in one line. Doing so means that you assign to the variables a, b, and c the tuple values, in the order that they appear in the tuple.

## 5. Accessing tuple elements

Additionally, you can also access individual tuple elements like you do with lists. Doing this here accesses the second element of the tuple. Why is that? Recall that with lists, you can use zero-indexing to access list elements. You can do the same thing with tuples! Pretty cool, right?

## 6. Returning multiple values

Let's now modify the behavior of your raise function. Instead of returning just the value of value1 raised to the power of value2, let's also return the value of value2 raised to the power of value1. You thus need to make raise return two values instead of one. We can use what we now know of tuples to do this! We first change the name of our function and the docstring to reflect the new behavior of our function. We then, in the function body, construct a tuple consisting of the values we want the function to return and, also in the function body, we return the tuple! Calling the function constructed demonstrates that it does exactly what we want!

## 7. Let's practice!

Now it's your turn to play with writing function that accept multiple arguments and return multiple values. Enjoy!

**Daily XP650**

##### Exercise

##### Exercise

# Functions with multiple parameters

Hugo discussed the use of multiple parameters in defining functions in the last lecture. You are now going to use what you've learned to modify the shout() function further. Here, you will modify shout() to accept two arguments. Parts of the function shout(), which you wrote earlier, are shown.

##### Instructions

**100 XP**

* Modify the function header such that it accepts two parameters, word1 and word2, in that order.
* Concatenate each of word1 and word2 with '!!!' and assign to shout1 and shout2, respectively.
* Concatenate shout1 and shout2 together, in that order, and assign to new\_shout.
* Pass the strings 'congratulations' and 'you', in that order, to a call to shout(). Assign the return value to yell.

# Define shout with parameters word1 and word2

def shout(\_\_\_\_, \_\_\_\_):

    """Concatenate strings with three exclamation marks"""

    # Concatenate word1 with '!!!': shout1

    # Concatenate word2 with '!!!': shout2

    # Concatenate shout1 with shout2: new\_shout

    # Return new\_shout

    return new\_shout

# Pass 'congratulations' and 'you' to shout(): yell

# Print yell

print(yell)

# Define shout with the parameter, word

def shout(word):

"""Return a string with three exclamation marks"""

# Concatenate the strings: shout\_word

shout\_word = word + '!!!'

# Replace print with return

return shout\_word

# Pass 'congratulations' to shout: yell

yell = shout('congratulations')

# Print yell

print(yell)

# Define shout with parameters word1 and word2

def shout(word1, word2):

    """Concatenate strings with three exclamation marks"""

    # Concatenate word1 with '!!!': shout1

    shout1 = word1 + '!!!'

    # Concatenate word2 with '!!!': shout2

    shout2 = word2 + '!!!'

    # Concatenate shout1 with shout2: new\_shout

    new\_shout = shout1 + shout2

    # Return new\_shout

    return new\_shout

# Pass 'congratulations' and 'you' to shout(): yell

yell = shout('congratulations', 'you')

# Print yell

print(yell)

# Define shout with parameters word1 and word2

def shout(word1, word2):

"""Concatenate strings with three exclamation marks"""

# Concatenate word1 with '!!!': shout1

shout1 = word1 + '!!!'

# Concatenate word2 with '!!!': shout2

shout2 = word2 + '!!!'

# Concatenate shout1 with shout2: new\_shout

new\_shout = shout1 + shout2

# Return new\_shout

return new\_shout

# Pass 'congratulations' and 'you' to shout(): yell

yell = shout('congratulations', 'you')

# Print yell

print(yell)

congratulations!!!you!!!

<script.py> output:

congratulations!!!you!!

**Daily XP750**

##### Exercise

##### Exercise

# A brief introduction to tuples

Alongside learning about functions, you've also learned about tuples! Here, you will practice what you've learned about tuples: how to construct, unpack, and access tuple elements. Recall how Hugo unpacked the tuple even\_nums in the video:

a, b, c = even\_nums

A three-element tuple named nums has been preloaded for this exercise. Before completing the script, perform the following:

* Print out the value of nums in the IPython shell. Note the elements in the tuple.
* In the IPython shell, try to change the first element of nums to the value 2 by doing an assignment: nums[0] = 2. What happens?

##### Instructions

**100 XP**

* Unpack nums to the variables num1, num2, and num3.
* Construct a new tuple, even\_nums composed of the same elements in nums, but with the 1st element replaced with the value, 2.

# Unpack nums into num1, num2, and num3

# Construct even\_nums

# Define shout with parameters word1 and word2

def shout(word1, word2):

"""Concatenate strings with three exclamation marks"""

# Concatenate word1 with '!!!': shout1

shout1 = word1 + '!!!'

# Concatenate word2 with '!!!': shout2

shout2 = word2 + '!!!'

# Concatenate shout1 with shout2: new\_shout

new\_shout = shout1 + shout2

# Return new\_shout

return new\_shout

# Pass 'congratulations' and 'you' to shout(): yell

yell = shout('congratulations', 'you')

# Print yell

print(yell)

# Unpack nums into num1, num2, and num3 num1,num2, num3 = nums # Construct even\_nums even\_nums = 2,4, 6 print(even\_nums)

**Daily XP850**

##### Exercise

##### Exercise

# Functions that return multiple values

In the previous exercise, you constructed tuples, assigned tuples to variables, and unpacked tuples. Here you will return multiple values from a function using tuples. Let's now update our shout() function to return multiple values. Instead of returning just one string, we will return two strings with the string !!! concatenated to each.

Note that the return statement return x, y has the same result as return (x, y): the former actually packs x and y into a tuple under the hood!

##### Instructions

**100 XP**

* Modify the function header such that the function name is now shout\_all, and it accepts two parameters, word1 and word2, in that order.
* Concatenate the string '!!!' to each of word1 and word2 and assign to shout1 and shout2, respectively.
* Construct a tuple shout\_words, composed of shout1 and shout2.
* Call shout\_all() with the strings 'congratulations' and 'you' and assign the result to yell1 and yell2 (remember, shout\_all() returns 2 variables!).

# Define shout\_all with parameters word1 and word2

def shout\_all(\_\_\_\_, \_\_\_\_):

    # Concatenate word1 with '!!!': shout1

    # Concatenate word2 with '!!!': shout2

    # Construct a tuple with shout1 and shout2: shout\_words

    # Return shout\_words

    return shout\_words

# Pass 'congratulations' and 'you' to shout\_all(): yell1, yell2

# Print yell1 and yell2

print(yell1)

print(yell2)

Define shout\_all with parameters word1 and word2

def shout\_all(word1, word2):

    # Concatenate word1 with '!!!': shout1

    shout1 = word1 + '!!!'

    # Concatenate word2 with '!!!': shout2

    shout2 = word2 + '!!!'

    # Construct a tuple with shout1 and shout2: shout\_words

    shout\_words = (shout1, shout2)

    # Return shout\_words

    return shout\_words

# Pass 'congratulations' and 'you' to shout\_all(): yell1, yell2

yell1, yell2 = shout\_all('congratulations', 'you')

# Print yell1 and yell2

print(yell1)

print(yell2)

# Define shout\_all with parameters word1 and word2

def shout\_all(word1, word2):

# Concatenate word1 with '!!!': shout1

shout1 = word1 + '!!!'

# Concatenate word2 with '!!!': shout2

shout2 = word2 + '!!!'

# Construct a tuple with shout1 and shout2: shout\_words

shout\_words = (shout1, shout2)

# Return shout\_words

return shout\_words

# Pass 'congratulations' and 'you' to shout\_all(): yell1, yell2

yell1, yell2 = shout\_all('congratulations', 'you')

# Print yell1 and yell2

print(yell1)

print(yell2)

congratulations!!!

you!!!

<script.py> output:

congratulations!!!

you!!!

## . Bringing it all together

Congrats! So now you're able to write functions

## 2. You’ve learned:

that accept multiple parameters and return multiple values, it is time to see how these learnt skills can be valuable in a Data Science context. In the following exercises, you'll write a function that analyzes a DataFrame of twitter data. The function that you'll write will return a dictionary containing data of how many times each language was used across all the tweets in the DataFrame. We'll see later in this course that we can generalize such a function to count occurrences of any items in a DataFrame column. Let's recap the basic ingredients of a function.

## 3. Basic ingredients of a function

We have a function header which begins with the keyword def. This is followed by the function name, parameters in parentheses and a colon. We then have the function body, which contains docstrings enclosed in triple quotation marks; docstrings describe what the function does; the rest of the function body performs the computation that the function does; the function body closes with the keyword return, followed by the value or values returned by the function. That's it from me.

## 4. Let's practice!

Happy function-writing!

**Daily XP1000**

##### Exercise

##### Exercise

# Bringing it all together (1)

You've got your first taste of writing your own functions in the previous exercises. You've learned how to add parameters to your own function definitions, return a value or multiple values with tuples, and how to call the functions you've defined.

In this and the following exercise, you will bring together all these concepts and apply them to a simple data science problem. You will load a dataset and develop functionalities to extract simple insights from the data.

For this exercise, your goal is to recall how to load a dataset into a DataFrame. The dataset contains Twitter data and you will iterate over entries in a column to build a dictionary in which the keys are the names of languages and the values are the number of tweets in the given language. The file tweets.csv is available in your current directory.

Be aware that this is real data from Twitter and as such there is always a risk that it may contain profanity or other offensive content (in this exercise, and any following exercises that also use real Twitter data).

##### Instructions

**100 XP**

* Import the pandas package with the alias pd.
* Import the file 'tweets.csv' using the pandas function read\_csv(). Assign the resulting DataFrame to df.
* Complete the for loop by iterating over col, the 'lang' column in the DataFrame df.
* Complete the bodies of the if-else statements in the for loop: **if** the key is in the dictionary langs\_count, add 1 to the value corresponding to this key in the dictionary, **else** add the key to langs\_count and set the corresponding value to 1. Use the loop variable entry in your code.

# Import pandas

# Import Twitter data as DataFrame: df

df = \_\_\_\_

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df['lang']

# Iterate over lang column in DataFrame

for entry in \_\_\_\_:

    # If the language is in langs\_count, add 1

    if entry in langs\_count.keys():

        \_\_\_\_

    # Else add the language to langs\_count, set the value to 1

    else:

        \_\_\_\_

# Print the populated dictionary

print(langs\_count)

# Define shout\_all with parameters word1 and word2 def shout\_all(word1, word2): # Concatenate word1 with '!!!': shout1 shout1 = word1 + '!!!' # Concatenate word2 with '!!!': shout2 shout2 = word2 + '!!!' # Construct a tuple with shout1 and shout2: shout\_words shout\_words = (shout1, shout2) # Return shout\_words return shout\_words # Pass 'congratulations' and 'you' to shout\_all(): yell1, yell2 yell1, yell2 = shout\_all('congratulations', 'you') # Print yell1 and yell2 print(yell1) print(yell2)

# Import pandas

import pandas as pd

# Import Twitter data as DataFrame: df

df = pd.read\_csv('tweets.csv')

#print(df['lang'].head())

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df['lang']

# Iterate over lang column in DataFrame

for entry in col:

    # If the language is in langs\_count, add 1

    if entry in langs\_count.keys():

       langs\_count[entry] +=1

    # Else add the language to langs\_count, set the value to 1

    else:

        langs\_count[entry] = 1

# Print the populated dictionary

print(langs\_count)

# Import pandas

import pandas as pd

# Import Twitter data as DataFrame: df

df = pd.read\_csv('tweets.csv')

#print(df['lang'].head())

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df['lang']

# Iterate over lang column in DataFrame

for entry in col:

# If the language is in langs\_count, add 1

if entry in langs\_count.keys():

langs\_count[entry] +=1

# Else add the language to langs\_count, set the value to 1

else:

langs\_count[entry] = 1

# Print the populated dictionary

print(langs\_count)

{'en': 97, 'et': 1, 'und': 2}

<script.py> output:

{'en': 97, 'et': 1, 'und': 2}

**Daily XP1100**

##### Exercise

##### Exercise

# Bringing it all together (2)

Great job! You've now defined the functionality for iterating over entries in a column and building a dictionary with keys the names of languages and values the number of tweets in the given language.

In this exercise, you will define a function with the functionality you developed in the previous exercise, return the resulting dictionary from within the function, and call the function with the appropriate arguments.

For your convenience, the pandas package has been imported as pd and the 'tweets.csv' file has been imported into the tweets\_df variable.

##### Instructions

**100 XP**

* Define the function count\_entries(), which has two parameters. The first parameter is df for the DataFrame and the second is col\_name for the column name.
* Complete the bodies of the if-else statements in the for loop: **if** the key is in the dictionary langs\_count, add 1 to its current value, **else** add the key to langs\_count and set its value to 1. Use the loop variable entry in your code.
* Return the langs\_count dictionary from inside the count\_entries() function.
* Call the count\_entries() function by passing to it tweets\_df and the name of the column, 'lang'. Assign the result of the call to the variable result.

# Define count\_entries()

def \_\_\_\_(\_\_\_\_, \_\_\_\_):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: langs\_count

    langs\_count = {}

    # Extract column from DataFrame: col

    col = df[col\_name]

    # Iterate over lang column in DataFrame

    for entry in col:

        # If the language is in langs\_count, add 1

        if entry in langs\_count.keys():

            \_\_\_\_

        # Else add the language to langs\_count, set the value to 1

        else:

            \_\_\_\_

    # Return the langs\_count dictionary

# Call count\_entries(): result

# Print the result

print(result)

* IPython Shell
* Slides
* Notes

# Import pandas

import pandas as pd

# Import Twitter data as DataFrame: df

df = pd.read\_csv('tweets.csv')

#print(df['lang'].head())

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df['lang']

# Iterate over lang column in DataFrame

for entry in col:

# If the language is in langs\_count, add 1

if entry in langs\_count.keys():

langs\_count[entry] +=1

# Else add the language to langs\_count, set the value to 1

else:

langs\_count[entry] = 1

# Print the populated dictionary

print(langs\_count)

# Define count\_entries()

def count\_entries(df, col\_name):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: langs\_count

    langs\_count = {}

    # Extract column from DataFrame: col

    col = df[col\_name]

    # Iterate over lang column in DataFrame

    for entry in col:

        # If the language is in langs\_count, add 1

        if entry in langs\_count.keys():

            langs\_count[entry] += 1

        # Else add the language to langs\_count, set the value to 1

        else:

            langs\_count[entry] =1

    # Return the langs\_count dictionary

    return langs\_count

# Call count\_entries(): result

result = count\_entries(tweets\_df , 'lang')

# Print the result

print(result)

# Define count\_entries()

def count\_entries(df, col\_name):

"""Return a dictionary with counts of

occurrences as value for each key."""

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over lang column in DataFrame

for entry in col:

# If the language is in langs\_count, add 1

if entry in langs\_count.keys():

langs\_count[entry] += 1

# Else add the language to langs\_count, set the value to 1

else:

langs\_count[entry] =1

# Return the langs\_count dictionary

return langs\_count

# Call count\_entries(): result

result = count\_entries(tweets\_df , 'lang')

# Print the result

print(result)

{'en': 97, 'et': 1, 'und': 2}

<script.py> output:

{'en': 97, 'et': 1, 'und': 2}

## 1. Congratulations!

Congratulations, you're now a bonafide Python function writer. On top of that, you have just written your very first Data Sciencey function.

## 2. Next chapters:

At this point, although you can write basic functions, you've really just touched the surface of function writing capabilities. In the following Chapters, you'll learn how to write functions that have default arguments so that when you call them, you don't always have to specify all the parameters; you'll learn how to write functions that can accept an arbitrary number of parameters and how to nest functions within one another; on top of this, you'll learn how to handle errors when writing functions, which will make your functions as robust as they need to be. Moreover, you'll see the importance of such techniques in Data Science by writing functions that are pertinent to the Data Science sphere like the Twitter DataFrame analysis that you just performed.

## 3. Let's practice!

I am pumped for this and can't wait to see you in the next chapter!

**Daily XP1250**

# Scope and user-defined functions

**50 XP**

This course is also available on the mobile app

## 1. Scope and user-defined functions

Wow! At this point, you know how to define your own functions - but not only that, you know how to write functions with multiple parameters and can return multiple values using tuples. Good job!

## 2. Crash course on scope in functions

We'll now talk about the idea of scope in the context of user-defined functions. You've been defining variables in your programs and so far, you've been using these variables without any problems. However, one thing you should know is that not all objects that you define are always accessible everywhere in a program. Enter the idea of scope, which tells you which part of a program an object or a name may be accessed. Names refer to the variables or, more generally, objects such as functions that are defined in your program, for example, a variable x has a name, as does the function sum. There are three types of scope that you should know. The first one is the idea of the global scope. A name that is in the global scope means that it is defined in the main body of a script or a Python program. The second one, is the local scope. A name that is in a local scope means that it is defined within a function. Once the execution of a function is done, any name inside the local scope ceases to exist, which means you cannot access those names anymore outside of the function definition. The third is something called the built-in scope: this consists of names in the pre-defined built-ins module Python provides, such as print and sum. You'll play around with the built-ins module in the interactive exercises.

## 3. Global vs. local scope (1)

Let's look at a couple of examples to clarify these definitions. Let's check out our example function square from earlier. We define the function and then call it. If we then try to access the variable name new\_val after function execution, the name is not accessible. This is because it was defined only within the local scope of the function. The name new\_val was not defined globally.

## 4. Global vs. local scope (2)

Now what if we define the name globally before defining and calling the function? In short, any time we call the name in the global scope, it will access the name in the global, such as you see here. Any time we call the name in the local scope of the function, it will look first in the local scope. That's why calling square(3) results in 9 and not 10. If Python cannot find the name in the local scope, it will then and only then look in the global scope.

## 5. Global vs. local scope (3)

Here, for example, we access new\_val defined globally within the function square. Note that the global value accessed is the value at the time the function is called, not the value when the function is defined. Thus, if we re-assign new\_val and call the function square, we see that the new value of new\_val is accessed. To recap, when we reference a name, first the local scope is searched, then the global. If the name is in neither, then the built-in scope is searched.

## 6. Global vs. local scope (4)

Now what if we want to alter the value of a global name within a function call? This is where the keyword global comes in handy. To look at how it works, let's look at another example. Within the function definition, we use the keyword global followed by the name of the global variable that we wish to access and alter. For example, here we change new\_val to its square. The function call works as one would expect. Now calling new\_val, we see that the global value has indeed been squared by running the function square.

## 7. Let's practice!

Now it's your turn to play with all things scope, local, global and built-in. Have fun!

# Pop quiz on understanding scope

In this exercise, you will practice what you've learned about scope in functions. The variable num has been predefined as 5, alongside the following function definitions:

def func1():

num = 3

print(num)

def func2():

global num

double\_num = num \* 2

num = 6

print(double\_num)

Try calling func1() and func2() in the shell, then answer the following questions:

* What are the values printed out when you call func1() and func2()?
* What is the value of num in the global scope after calling func1() and func2()?

##### Instructions

**50 XP**

##### Possible Answers

* 

func1() prints out 3, func2() prints out 6, and the value of num in the global scope is 3.

* 

func1() prints out 3, func2() prints out 3, and the value of num in the global scope is 3.

* 

func1() prints out 3, func2() prints out 10, and the value of num in the global scope is 10.

* 

**Correct answer**: func1() prints out 3, func2() prints out 10, and the value of num in the global scope is 6.

# Define count\_entries() def count\_entries(df, col\_name): """Return a dictionary with counts of occurrences as value for each key.""" # Initialize an empty dictionary: langs\_count langs\_count = {} # Extract column from DataFrame: col col = df[col\_name] # Iterate over lang column in DataFrame for entry in col: # If the language is in langs\_count, add 1 if entry in langs\_count.keys(): langs\_count[entry] += 1 # Else add the language to langs\_count, set the value to 1 else: langs\_count[entry] =1 # Return the langs\_count dictionary return langs\_count # Call count\_entries(): result result = count\_entries(tweets\_df , 'lang') # Print the result print(result)

In [1]:

print(func1)

<function func1 at 0x7fd22fb13940>

In [2]:

print(func2)

<function func2 at 0x7fd22fb139d0>

In [3]:

print(num)

5

In [4]:

func1()

3

In [5]:

func2()

10

**Daily XP1350**

##### Exercise

##### Exercise

# The keyword global

Let's work more on your mastery of scope. In this exercise, you will use the keyword global within a function to alter the value of a variable defined in the global scope.

##### Instructions

**100 XP**

* Use the keyword global to alter the object team in the global scope.
* Change the value of team in the global scope to the string "justice league". Assign the result to team.
* Hit the Submit button to see how executing your newly defined function change\_team() changes the value of the name team!

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

    """Change the value of the global variable team."""

    # Use team in global scope

    # Change the value of team in global: team

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

* ython Shell
* Slides
* Notes

# Define count\_entries()

def count\_entries(df, col\_name):

"""Return a dictionary with counts of

occurrences as value for each key."""

# Initialize an empty dictionary: langs\_count

langs\_count = {}

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over lang column in DataFrame

for entry in col:

# If the language is in langs\_count, add 1

if entry in langs\_count.keys():

langs\_count[entry] += 1

# Else add the language to langs\_count, set the value to 1

else:

langs\_count[entry] =1

# Return the langs\_count dictionary

return langs\_count

# Call count\_entries(): result

result = count\_entries(tweets\_df , 'lang')

# Print the result

print(result)

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

    """Change the value of the global variable team."""

    # Use team in global scope

    global team

    # Change the value of team in global: team

    team = "justice league"

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

"""Change the value of the global variable team."""

# Use team in global scope

global team

# Change the value of team in global: team

team = "justice league"

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

teen titans

justice league

**Daily XP100**

##### Exercise

##### Exercise

# Python's built-in scope

Here you're going to check out Python's built-in scope, which is really just a built-in module called builtins. However, to query builtins, you'll need to import builtins 'because the name builtins is not itself built in…No, I’m serious!' ([Learning Python, 5th edition](http://shop.oreilly.com/product/0636920028154.do), Mark Lutz). After executing import builtins in the IPython Shell, execute dir(builtins) to print a list of all the names in the module builtins. Have a look and you'll see a bunch of names that you'll recognize! Which of the following names is NOT in the module builtins?

##### Instructions

**50 XP**

##### Possible Answers

* 

'sum'

* 

'range'

* 

'array'

* 

'tuple'

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

"""Change the value of the global variable team."""

# Use team in global scope

global team

# Change the value of team in global: team

team = "justice league"

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

Ctrl+O

**Daily XP150**

# Nested functions

**50 XP**

This course is also available on the mobile app

## 1. Nested functions

Now that you've come to grips with scope, both local and global, it's time to dive a bit deeper!

## 2. Nested functions (1)

What if we have a function inner defined within another function outer and we reference a name x in the inner function? The answer is intuitive: Python searches the local scope of the function inner, then if it doesn't find x, it searches the scope of the function outer, which is called an enclosing function because it encloses the function inner. If Python can't find x in the scope of the enclosing function, it only then searches the global scope and then the built-in scope. But whoa, hold on there for a second, why are we even nesting functions?

## 3. Nested functions (2)

There are a number of good reasons to do so. Let's say that we want to use a process a number of times within a function. For example, we want a function that takes 3 numbers as parameters and performs the same function on each of them. One way would be to write out the computation 3 times

## 4. Nested functions (3)

but this definitely does not scale if you need to perform the computation many times. What we can do instead is define an inner function within our function definition, such as we do here, and call it where necessary. This is called a nested function. The syntax for the inner function is exactly the same as that for any other function.

## 5. Returning functions

Let's now look at another important use case of nested functions. In this example, we define a function raise\_vals, which contains an inner function called inner. Now look at what raise\_vals returns: it returns the inner function inner! raise\_vals takes an argument n and creates a function inner that returns the nth power of any number. That's a bit complicated and will be clearer when we use the function raise\_vals. Passing the number 2 to raise\_vals creates a function that squares any number. Similarly, passing the number 3 to raise\_vals creates a function that cubes any number. One interesting detail: when we call the function square, it remembers the value n=2, although the enclosing scope defined by raise\_val and to which n=2 is local, has finished execution. This is a subtlety referred to as a closure in Computer Science circles and shouldn't concern you too much. It is worth mentioning, however, as you may encounter it out there in the wild.

## 6. Using nonlocal

Recall from our discussion of scope that you can use the keyword global in function definitions to create and change global names; similarly, in a nested function, you can use the keyword nonlocal to create and changes names in an enclosing scope. In this example, we alter the value of n in the inner function; because we used the keyword nonlocal, it also alter the value of n in the enclosing scope. This is why calling the function outer prints the value of n as determined within the function inner. You'll have practice using the keyword nonlocal in the interactive exercises.

## 7. Scopes searched

To summarize: name references search at most four scopes, the local scope, then those of enclosing functions, if there are any; then global, then built-in. This is known as the LEGB rule, where L is for local, E for enclosing, G for global and B for built-ins! Also, remember that assigning names will only create or change local names, unless they are declared in global or nonlocal statements using the keyword global or the keyword nonlocal, respectively.

## 8. Let's practice!

Now it's your turn to write some nested functions. Enjoy!

# Nested Functions I

You've learned in the last video about nesting functions within functions. One reason why you'd like to do this is to avoid writing out the same computations within functions repeatedly. There's nothing new about defining nested functions: you simply define it as you would a regular function with def and embed it inside another function!

In this exercise, inside a function three\_shouts(), you will define a nested function inner() that concatenates a string object with !!!. three\_shouts() then returns a tuple of three elements, each a string concatenated with !!! using inner(). Go for it!

##### Instructions

**100 XP**

* Complete the function header of the nested function with the function name inner() and a single parameter word.
* Complete the return value: each element of the tuple should be a call to inner(), passing in the parameters from three\_shouts() as arguments to each call.
* # Define three\_shouts
* def three\_shouts(word1, word2, word3):
* """Returns a tuple of strings
* concatenated with '!!!'."""
* # Define inner
* def \_\_\_\_(\_\_\_\_):
* """Returns a string concatenated with '!!!'."""
* return word + '!!!'
* # Return a tuple of strings
* return (\_\_\_\_, \_\_\_\_, \_\_\_\_)
* # Call three\_shouts() and print
* print(three\_shouts('a', 'b', 'c'))
* IPython Shell
* Slides
* Notes

# Create a string: team

team = "teen titans"

# Define change\_team()

def change\_team():

"""Change the value of the global variable team."""

# Use team in global scope

global team

# Change the value of team in global: team

team = "justice league"

# Print team

print(team)

# Call change\_team()

change\_team()

# Print team

print(team)

# Define three\_shouts

def three\_shouts(word1, word2, word3):

    """Returns a tuple of strings

    concatenated with '!!!'."""

    # Define inner

    def inner(word):

        """Returns a string concatenated with '!!!'."""

        return word + '!!!'

    # Return a tuple of strings

    return (inner(word1), inner(word2), inner(word3))

# Call three\_shouts() and print

print(three\_shouts('a', 'b', 'c'))

# Define three\_shouts

def three\_shouts(word1, word2, word3):

"""Returns a tuple of strings

concatenated with '!!!'."""

# Define inner

def inner(word):

"""Returns a string concatenated with '!!!'."""

return word + '!!!'

# Return a tuple of strings

return (inner(word1), inner(word2), inner(word3))

# Call three\_shouts() and print

print(three\_shouts('a', 'b', 'c'))

('a!!!', 'b!!!', 'c!!!')

<script.py> output:

('a!!!', 'b!!!', 'c!!!')

**Daily XP150**

##### Exercise

##### Exercise

# Nested Functions II

Great job, you've just nested a function within another function. One other pretty cool reason for nesting functions is the idea of a **closure**. This means that the nested or inner function remembers the state of its enclosing scope when called. Thus, anything defined locally in the enclosing scope is available to the inner function even when the outer function has finished execution.

Let's move forward then! In this exercise, you will complete the definition of the inner function inner\_echo() and then call echo() a couple of times, each with a different argument. Complete the exercise and see what the output will be!

##### Instructions

**100 XP**

* Complete the function header of the inner function with the function name inner\_echo() and a single parameter word1.
* Complete the function echo() so that it returns inner\_echo.
* We have called echo(), passing 2 as an argument, and assigned the resulting function to twice. Your job is to call echo(), passing 3 as an argument. Assign the resulting function to thrice.
* Hit Submit to call twice() and thrice() and print the results.

# Define echo

def echo(n):

    """Return the inner\_echo function."""

    # Define inner\_echo

    def \_\_\_\_(\_\_\_\_):

        """Concatenate n copies of word1."""

        echo\_word = word1 \* n

        return echo\_word

    # Return inner\_echo

# Call echo: twice

twice = echo(2)

# Call echo: thrice

# Call twice() and thrice() then print

print(twice('hello'), thrice('hello'))

# Define three\_shouts

def three\_shouts(word1, word2, word3):

"""Returns a tuple of strings

concatenated with '!!!'."""

# Define inner

def inner(word):

"""Returns a string concatenated with '!!!'."""

return word + '!!!'

# Return a tuple of strings

return (inner(word1), inner(word2), inner(word3))

# Call three\_shouts() and print

print(three\_shouts('a', 'b', 'c'))

# Define echo

def echo(n):

    """Return the inner\_echo function."""

    # Define inner\_echo

    def inner\_echo(word1):

        """Concatenate n copies of word1."""

        echo\_word = word1 \* n

        return echo\_word

    # Return inner\_echo

    return inner\_echo

# Call echo: twice

twice = echo(2)

# Call echo: thrice

thrice = echo(3)

# Call twice() and thrice() then print

print(twice('hello'), thrice('hello'))

# Define echo

def echo(n):

"""Return the inner\_echo function."""

# Define inner\_echo

def inner\_echo(word1):

"""Concatenate n copies of word1."""

echo\_word = word1 \* n

return echo\_word

# Return inner\_echo

return inner\_echo

# Call echo: twice

twice = echo(2)

# Call echo: thrice

thrice = echo(3)

# Call twice() and thrice() then print

print(twice('hello'), thrice('hello'))

hellohello hellohellohello

<script.py> output:

hellohello hellohellohello

**Daily XP350**

# Default and flexible arguments

**50 XP**

This course is also available on the mobile app

## 1. Default and flexible arguments

Let's say that you're writing a function that takes multiple parameters and that there is often a common value for some of these parameters. In this case, you would like to be able to call the function without explicitly specifying every parameter. In other words, you would like some parameters to have default arguments that are used when it is not specified otherwise!

## 2. You'll learn:

In this video, you'll discover how to write function with default arguments, along with using flexible arguments, which allows you to pass any number of arguments to a function, as we'll soon see.

## 3. Add a default argument

First up, to define a function with a default argument value, in the function header we follow the parameter of interest with an equals sign and the default argument value. Notice that this function raises the first argument to the power of the second argument and the default 2nd argument value is 1. So we can call the function with two arguments as you would expect, however, if you only use one argument, the function call will use the default argument of 1 for the second parameter! Neat, huh? In the interactive exercises that follows, you'll gain expertise in writing functions with both single and multiple default arguments.

## 4. Flexible arguments: \*args (1)

Lets now look at flexible arguments: let's say that you want to write a function but aren't sure how many arguments a user will want to pass it; for example, a function that takes floats or ints and adds them all up, irrespective of how many there are. Enter flexible arguments! In this example, we write the function that sums up all the arguments passed to it. In the function definition, we use the parameter star followed by args: this then turns all the arguments passed to a function call into a tuple called args in the function body; then, in the function body, to write our desired function, we initialize our sum sum\_all to 0, loop over the tuple args and add each element of it successively to sum\_all and then return it.

## 5. Flexible arguments: \*args (2)

We can now call our function add\_all with any number of arguments to add them all up!

## 6. Flexible arguments: \*\*kwargs

You can also use a double star to pass an arbitrary number of keyword arguments, also called kwargs, that is, arguments preceded by identifiers. We'll write such a function called print\_all that prints out the identifiers and the parameters passed to them as you see here.

## 7. Flexible arguments: \*\*kwargs

Now to write such a function, we use the parameter kwargs preceded by a double star. This turns the identifier-keyword pairs into a dictionary within the function body. Then, in the function body all we need to do is to print all the key-value pairs stored in the dictionary kwargs. Note that it is NOT the names args and kwargs that are important when using flexible arguments, but rather that they're preceded by a single and double star, respectively.

## 8. Let's practice!

I know this is a lot to take in so it's now time to go and hack it out yourself. Get writing functions!

**Daily XP350**

# Default and flexible arguments

**50 XP**

This course is also available on the mobile app

## 1. Default and flexible arguments

Let's say that you're writing a function that takes multiple parameters and that there is often a common value for some of these parameters. In this case, you would like to be able to call the function without explicitly specifying every parameter. In other words, you would like some parameters to have default arguments that are used when it is not specified otherwise!

## 2. You'll learn:

In this video, you'll discover how to write function with default arguments, along with using flexible arguments, which allows you to pass any number of arguments to a function, as we'll soon see.

## 3. Add a default argument

First up, to define a function with a default argument value, in the function header we follow the parameter of interest with an equals sign and the default argument value. Notice that this function raises the first argument to the power of the second argument and the default 2nd argument value is 1. So we can call the function with two arguments as you would expect, however, if you only use one argument, the function call will use the default argument of 1 for the second parameter! Neat, huh? In the interactive exercises that follows, you'll gain expertise in writing functions with both single and multiple default arguments.

## 4. Flexible arguments: \*args (1)

Lets now look at flexible arguments: let's say that you want to write a function but aren't sure how many arguments a user will want to pass it; for example, a function that takes floats or ints and adds them all up, irrespective of how many there are. Enter flexible arguments! In this example, we write the function that sums up all the arguments passed to it. In the function definition, we use the parameter star followed by args: this then turns all the arguments passed to a function call into a tuple called args in the function body; then, in the function body, to write our desired function, we initialize our sum sum\_all to 0, loop over the tuple args and add each element of it successively to sum\_all and then return it.

## 5. Flexible arguments: \*args (2)

We can now call our function add\_all with any number of arguments to add them all up!

## 6. Flexible arguments: \*\*kwargs

You can also use a double star to pass an arbitrary number of keyword arguments, also called kwargs, that is, arguments preceded by identifiers. We'll write such a function called print\_all that prints out the identifiers and the parameters passed to them as you see here.

## 7. Flexible arguments: \*\*kwargs

Now to write such a function, we use the parameter kwargs preceded by a double star. This turns the identifier-keyword pairs into a dictionary within the function body. Then, in the function body all we need to do is to print all the key-value pairs stored in the dictionary kwargs. Note that it is NOT the names args and kwargs that are important when using flexible arguments, but rather that they're preceded by a single and double star, respectively.

## 8. Let's practice!

I know this is a lot to take in so it's now time to go and hack it out yourself. Get writing functions!

# Define shout\_echo

def \_\_\_\_(\_\_\_\_, \_\_\_\_):

    """Concatenate echo copies of word1 and three

     exclamation marks at the end of the string."""

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = \_\_\_\_

    # Concatenate '!!!' to echo\_word: shout\_word

    shout\_word = echo\_word + '!!!'

    # Return shout\_word

    return shout\_word

# Call shout\_echo() with "Hey": no\_echo

no\_echo = \_\_\_\_

# Call shout\_echo() with "Hey" and echo=5: with\_echo

with\_echo = \_\_\_\_

# Print no\_echo and with\_echo

print(no\_echo)

print(with\_echo)

# Define shout\_echo

def shout\_echo(word1 , echo=1):

    """Concatenate echo copies of word1 and three

     exclamation marks at the end of the string."""

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Concatenate '!!!' to echo\_word: shout\_word

    shout\_word = echo\_word + '!!!'

    # Return shout\_word

    return shout\_word

# Call shout\_echo() with "Hey": no\_echo

no\_echo = shout\_echo("Hey")

# Call shout\_echo() with "Hey" and echo=5: with\_echo

with\_echo = shout\_echo("Hey" , echo=5)

# Print no\_echo and with\_echo

print(no\_echo)

print(with\_echo)

* n Shell
* Slides
* Notes

# Define shout\_echo

def shout\_echo(word1 , echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo() with "Hey": no\_echo

no\_echo = shout\_echo("Hey")

# Call shout\_echo() with "Hey" and echo=5: with\_echo

with\_echo = shout\_echo("Hey" , echo=5)

# Print no\_echo and with\_echo

print(no\_echo)

print(with\_echo)

Hey!!!

HeyHeyHeyHeyHey!!!

<script.py> output:

Hey!!!

HeyHeyHeyHeyHey!!!

**Daily XP500**

##### Exercise

##### Exercise

# Functions with multiple default arguments

You've now defined a function that uses a default argument - don't stop there just yet! You will now try your hand at defining a function with more than one default argument and then calling this function in various ways.

After defining the function, you will call it by supplying values to all the default arguments of the function. Additionally, you will call the function by not passing a value to one of the default arguments - see how that changes the output of your function!

##### Instructions

**100 XP**

* Complete the function header with the function name shout\_echo. It accepts an argument word1, a default argument echo with default value 1 and a default argument intense with default value False, in that order.
* In the body of the if statement, make the string object echo\_word upper case by applying the method .upper() on it.
* Call shout\_echo() with the string, "Hey", the value 5 for echo and the value True for intense. Assign the result to with\_big\_echo.
* Call shout\_echo() with the string "Hey" and the value True for intense. Assign the result to big\_no\_echo.

# Define shout\_echo

def \_\_\_\_(\_\_\_\_, \_\_\_\_, \_\_\_\_):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Make echo\_word uppercase if intense is True

    if intense is True:

        # Make uppercase and concatenate '!!!': echo\_word\_new

        echo\_word\_new = \_\_\_\_ + '!!!'

    else:

        # Concatenate '!!!' to echo\_word: echo\_word\_new

        echo\_word\_new = echo\_word + '!!!'

    # Return echo\_word\_new

    return echo\_word\_new

# Call shout\_echo() with "Hey", echo=5 and intense=True: with\_big\_echo

with\_big\_echo = \_\_\_\_

# Call shout\_echo() with "Hey" and intense=True: big\_no\_echo

big\_no\_echo = \_\_\_\_

# Print values

print(with\_big\_echo)

print(big\_no\_echo)

# Define shout\_echo

def shout\_echo(word1 , echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo() with "Hey": no\_echo

no\_echo = shout\_echo("Hey")

# Call shout\_echo() with "Hey" and echo=5: with\_echo

with\_echo = shout\_echo("Hey" , echo=5)

# Print no\_echo and with\_echo

print(no\_echo)

print(with\_echo)

# Define shout\_echo

def shout\_echo(word1, echo=1, intense=False):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Make echo\_word uppercase if intense is True

    if intense is True:

        # Make uppercase and concatenate '!!!': echo\_word\_new

        echo\_word\_new = echo\_word.upper() + '!!!'

    else:

        # Concatenate '!!!' to echo\_word: echo\_word\_new

        echo\_word\_new = echo\_word + '!!!'

    # Return echo\_word\_new

    return echo\_word\_new

# Call shout\_echo() with "Hey", echo=5 and intense=True: with\_big\_echo

with\_big\_echo = shout\_echo("Hey", echo=5, intense=True)

# Call shout\_echo() with "Hey" and intense=True: big\_no\_echo

big\_no\_echo = shout\_echo("Hey" , intense=True)

# Print values

print(with\_big\_echo)

print(big\_no\_echo)

* ython Shell
* Slides
* Notes

# Define shout\_echo

def shout\_echo(word1 , echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo() with "Hey": no\_echo

no\_echo = shout\_echo("Hey")

# Call shout\_echo() with "Hey" and echo=5: with\_echo

with\_echo = shout\_echo("Hey" , echo=5)

# Print no\_echo and with\_echo

print(no\_echo)

print(with\_echo)

# Define shout\_echo

def shout\_echo(word1, echo=1, intense=False):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Make echo\_word uppercase if intense is True

if intense is True:

# Make uppercase and concatenate '!!!': echo\_word\_new

echo\_word\_new = echo\_word.upper() + '!!!'

else:

# Concatenate '!!!' to echo\_word: echo\_word\_new

echo\_word\_new = echo\_word + '!!!'

# Return echo\_word\_new

return echo\_word\_new

# Call shout\_echo() with "Hey", echo=5 and intense=True: with\_big\_echo

with\_big\_echo = shout\_echo("Hey", echo=5, intense=True)

# Call shout\_echo() with "Hey" and intense=True: big\_no\_echo

big\_no\_echo = shout\_echo("Hey" , intense=True)

# Print values

print(with\_big\_echo)

print(big\_no\_echo)

HEYHEYHEYHEYHEY!!!

HEY!!!

<script.py> output:

HEYHEYHEYHEYHEY!!!

HEY!!!

**Daily XP600**

##### Exercise

##### Exercise

# Functions with variable-length arguments (\*args)

Flexible arguments enable you to pass a variable number of arguments to a function. In this exercise, you will practice defining a function that accepts a variable number of string arguments.

The function you will define is gibberish() which can accept a variable number of string values. Its return value is a single string composed of all the string arguments concatenated together in the order they were passed to the function call. You will call the function with a single string argument and see how the output changes with another call using more than one string argument. Recall from the previous video that, within the function definition, args is a tuple.

##### Instructions

**100 XP**

* Complete the function header with the function name gibberish. It accepts a single flexible argument \*args.
* Initialize a variable hodgepodge to an empty string.
* Return the variable hodgepodge at the end of the function body.
* Call gibberish() with the single string, "luke". Assign the result to one\_word.
* Hit the Submit button to call gibberish() with multiple arguments and to print the value to the Shell.

# Define gibberish

def \_\_\_\_(\_\_\_\_):

    """Concatenate strings in \*args together."""

    # Initialize an empty string: hodgepodge

    # Concatenate the strings in args

    for word in args:

        hodgepodge += word

    # Return hodgepodge

    \_\_\_\_

# Call gibberish() with one string: one\_word

one\_word = \_\_\_\_

# Call gibberish() with five strings: many\_words

many\_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one\_word and many\_words

print(one\_word)

print(many\_words)

# Define shout\_echo

def shout\_echo(word1, echo=1, intense=False):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Make echo\_word uppercase if intense is True

if intense is True:

# Make uppercase and concatenate '!!!': echo\_word\_new

echo\_word\_new = echo\_word.upper() + '!!!'

else:

# Concatenate '!!!' to echo\_word: echo\_word\_new

echo\_word\_new = echo\_word + '!!!'

# Return echo\_word\_new

return echo\_word\_new

# Call shout\_echo() with "Hey", echo=5 and intense=True: with\_big\_echo

with\_big\_echo = shout\_echo("Hey", echo=5, intense=True)

# Call shout\_echo() with "Hey" and intense=True: big\_no\_echo

big\_no\_echo = shout\_echo("Hey" , intense=True)

# Print values

print(with\_big\_echo)

print(big\_no\_echo)

# Define echo\_shout()

def echo\_shout(word):

"""Change the value of a nonlocal variable"""

# Concatenate word with itself: echo\_word

echo\_word = word + word

# Print echo\_word

print(echo\_word)

# Define inner function shout()

def shout():

"""Alter a variable in the enclosing scope"""

# Use echo\_word in nonlocal scope

nonlocal echo\_word

# Change echo\_word to echo\_word concatenated with '!!!'

echo\_word = echo\_word + '!!!'

# Call function shout()

shout()

# Print echo\_word

print(echo\_word)

# Call function echo\_shout() with argument 'hello'

echo\_shout('hello')

# Functions with one default argument

In the previous chapter, you've learned to define functions with more than one parameter and then calling those functions by passing the required number of arguments. In the last video, Hugo built on this idea by showing you how to define functions with default arguments. You will practice that skill in this exercise by writing a function that uses a default argument and then calling the function a couple of times.

##### Instructions

**100 XP**

* Complete the function header with the function name shout\_echo. It accepts an argument word1 and a default argument echo with default value 1, in that order.
* Use the \* operator to concatenate echo copies of word1. Assign the result to echo\_word.
* Call shout\_echo() with just the string, "Hey". Assign the result to no\_echo.
* Call shout\_echo() with the string "Hey" and the value 5 for the default argument, echo. Assign the result to with\_echo.

# Define gibberish

def gibberish(\*args):

    """Concatenate strings in \*args together."""

    # Initialize an empty string: hodgepodge

    hodgepodge = ""

    # Concatenate the strings in args

    for word in args:

        hodgepodge += word

    # Return hodgepodge

    return hodgepodge

# Call gibberish() with one string: one\_word

one\_word = gibberish("luke")

# Call gibberish() with five strings: many\_words

many\_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one\_word and many\_words

print(one\_word)

print(many\_words)

# Define gibberish

def gibberish(\*args):

"""Concatenate strings in \*args together."""

# Initialize an empty string: hodgepodge

hodgepodge = ""

# Concatenate the strings in args

for word in args:

hodgepodge += word

# Return hodgepodge

return hodgepodge

# Call gibberish() with one string: one\_word

one\_word = gibberish("luke")

# Call gibberish() with five strings: many\_words

many\_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one\_word and many\_words

print(one\_word)

print(many\_words)

luke

lukeleiahanobidarth

<script.py> output:

luke

lukeleiahanobidarth

**Daily XP700**

##### Exercise

##### Exercise

# Functions with variable-length keyword arguments (\*\*kwargs)

Let's push further on what you've learned about flexible arguments - you've used \*args, you're now going to use \*\*kwargs! What makes \*\*kwargs different is that it allows you to pass a variable number of keyword arguments to functions. Recall from the previous video that, within the function definition, kwargs is a dictionary.

To understand this idea better, you're going to use \*\*kwargs in this exercise to define a function that accepts a variable number of keyword arguments. The function simulates a simple status report system that prints out the status of a character in a movie.

##### Instructions

**100 XP**

* Complete the function header with the function name report\_status. It accepts a single flexible argument \*\*kwargs.
* Iterate over the key-value pairs of kwargs to print out the keys and values, separated by a colon ':'.
* In the first call to report\_status(), pass the following keyword-value pairs: name="luke", affiliation="jedi" and status="missing".
* In the second call to report\_status(), pass the following keyword-value pairs: name="anakin", affiliation="sith lord" and status="deceased".

# Define report\_status

def \_\_\_\_(\_\_\_\_):

    """Print out the status of a movie character."""

    print("\nBEGIN: REPORT\n")

    # Iterate over the key-value pairs of kwargs

    for \_\_\_\_, \_\_\_\_ in kwargs.items():

        # Print out the keys and values, separated by a colon ':'

        print(\_\_\_\_ + ": " + \_\_\_\_)

    print("\nEND REPORT")

# First call to report\_status()

# Second call to report\_status()

report\_status(name=\_\_\_\_, affiliation=\_\_\_\_, status=\_\_\_\_)

# Define gibberish

def gibberish(\*args):

"""Concatenate strings in \*args together."""

# Initialize an empty string: hodgepodge

hodgepodge = ""

# Concatenate the strings in args

for word in args:

hodgepodge += word

# Return hodgepodge

return hodgepodge

# Call gibberish() with one string: one\_word

one\_word = gibberish("luke")

# Call gibberish() with five strings: many\_words

many\_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one\_word and many\_words

print(one\_word)

print(many\_words)

# Define report\_status

def report\_status(\*\*kwargs):

    """Print out the status of a movie character."""

    print("\nBEGIN: REPORT\n")

    # Iterate over the key-value pairs of kwargs

    for key, value in kwargs.items():

        # Print out the keys and values, separated by a colon ':'

        print(key + ": " + value)

    print("\nEND REPORT")

# First call to report\_status()

report\_status(name="luke", affiliation="jedi", status="missing")

# Second call to report\_status()

report\_status(name="anakin", affiliation="sith lord", status="deceased")



* IPython Shell
* Slides
* Notes

# Define gibberish

def gibberish(\*args):

"""Concatenate strings in \*args together."""

# Initialize an empty string: hodgepodge

hodgepodge = ""

# Concatenate the strings in args

for word in args:

hodgepodge += word

# Return hodgepodge

return hodgepodge

# Call gibberish() with one string: one\_word

one\_word = gibberish("luke")

# Call gibberish() with five strings: many\_words

many\_words = gibberish("luke", "leia", "han", "obi", "darth")

# Print one\_word and many\_words

print(one\_word)

print(many\_words)

# Define report\_status

def report\_status(\*\*kwargs):

"""Print out the status of a movie character."""

print("\nBEGIN: REPORT\n")

# Iterate over the key-value pairs of kwargs

for key, value in kwargs.items():

# Print out the keys and values, separated by a colon ':'

print(key + ": " + value)

print("\nEND REPORT")

# First call to report\_status()

report\_status(name="luke", affiliation="jedi", status="missing")

# Second call to report\_status()

report\_status(name="anakin", affiliation="sith lord", status="deceased")

BEGIN: REPORT

name: luke

affiliation: jedi

status: missing

END REPORT

BEGIN: REPORT

name: anakin

affiliation: sith lord

status: deceased

END REPORT

<script.py> output:

BEGIN: REPORT

name: luke

affiliation: jedi

status: missing

END REPORT

BEGIN: REPORT

name: anakin

affiliation: sith lord

status: deceased

END REPORT

This course is also available on the mobile app

## 1. Bringing it all together

It's now time to get you hands dirty using your newly acquired skills by applying them to the wonderful world of Data Science: recall that, in the previous chapter, you wrote a function that would perform the following: it would take a DataFrame of twitter data and return a dictionary containing languages as keys and the number of times a tweet was written in a given language as values.

## 2. Next exercises:

In the following exercises, you will write a function that processes a DataFrame and returns a dictionary with counts of occurrences in any column at all! By default, however, it will process a column called lang. This generalizes the previous function that you wrote. You will then generalize this further so that you can pass the function a DataFrame and any number of column names to perform the computation on an arbitrary number of columns.

## 3. Add a default argument

Let's remind ourselves of the techniques necessary: recall that to define a function with a default argument, all that you need to do is provide that argument in the function header as you can see here.

## 4. Flexible arguments: \*args (1)

Next up, to write a function that you can pass an arbitrary number of arguments to, that is, flexible arguments, we use the arguments args as here, and then we can loop over all elements of args in the function body. Now that we've had that quick refresher, let's write some more functions!

## 5. Let's practice!

Happy coding!

**Daily XP850**

##### Exercise

##### Exercise

# Bringing it all together (1)

Recall the Bringing it all together exercise in the previous chapter where you did a simple Twitter analysis by developing a function that counts how many tweets are in certain languages. The output of your function was a dictionary that had the language as the keys and the counts of tweets in that language as the value.

In this exercise, we will generalize the Twitter language analysis that you did in the previous chapter. You will do that by including a **default argument** that takes a column name.

For your convenience, pandas has been imported as pd and the 'tweets.csv' file has been imported into the DataFrame tweets\_df. Parts of the code from your previous work are also provided.

##### Instructions

**100 XP**

* Complete the function header by supplying the parameter for a DataFrame df and the parameter col\_name with a default value of 'lang' for the DataFrame column name.
* Call count\_entries() by passing the tweets\_df DataFrame and the column name 'lang'. Assign the result to result1. Note that since 'lang' is the default value of the col\_name parameter, you don't have to specify it here.
* Call count\_entries() by passing the tweets\_df DataFrame and the column name 'source'. Assign the result to result2.

# Define count\_entries()

def count\_entries(\_\_\_\_, \_\_\_\_):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Extract column from DataFrame: col

    col = df[col\_name]

    # Iterate over the column in DataFrame

    for entry in col:

        # If entry is in cols\_count, add 1

        if entry in cols\_count.keys():

            cols\_count[entry] += 1

        # Else add the entry to cols\_count, set the value to 1

        else:

            cols\_count[entry] = 1

    # Return the cols\_count dictionary

    return cols\_count

# Call count\_entries(): result1

result1 = \_\_\_\_

# Call count\_entries(): result2

result2 = \_\_\_\_

# Print result1 and result2

print(result1)

print(result2)



* IPython Shell
* Slides
* Notes

# Define report\_status

def report\_status(\*\*kwargs):

"""Print out the status of a movie character."""

print("\nBEGIN: REPORT\n")

# Iterate over the key-value pairs of kwargs

for key, value in kwargs.items():

# Print out the keys and values, separated by a colon ':'

print(key + ": " + value)

print("\nEND REPORT")

# First call to report\_status()

report\_status(name="luke", affiliation="jedi", status="missing")

# Second call to report\_status()

report\_status(name="anakin", affiliation="sith lord", status="deceased")

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Extract column from DataFrame: col

    col = df[col\_name]

    # Iterate over the column in DataFrame

    for entry in col:

        # If entry is in cols\_count, add 1

        if entry in cols\_count.keys():

            cols\_count[entry] += 1

        # Else add the entry to cols\_count, set the value to 1

        else:

            cols\_count[entry] = 1

    # Return the cols\_count dictionary

    return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df)

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, col\_name='source')

# Print result1 and result2

print(result1)

print(result2)

# Define report\_status

def report\_status(\*\*kwargs):

"""Print out the status of a movie character."""

print("\nBEGIN: REPORT\n")

# Iterate over the key-value pairs of kwargs

for key, value in kwargs.items():

# Print out the keys and values, separated by a colon ':'

print(key + ": " + value)

print("\nEND REPORT")

# First call to report\_status()

report\_status(name="luke", affiliation="jedi", status="missing")

# Second call to report\_status()

report\_status(name="anakin", affiliation="sith lord", status="deceased")

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

"""Return a dictionary with counts of

occurrences as value for each key."""

# Initialize an empty dictionary: cols\_count

cols\_count = {}

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df)

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, col\_name='source')

# Print result1 and result2

print(result1)

print(result2)

{'en': 97, 'et': 1, 'und': 2}

{'<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>': 24, '<a href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>': 26, '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com" rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/" rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com" rel="nofollow">Linkis.com</a>': 2, '<a href="http://rutracker.org/forum/viewforum.php?f=93" rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com" rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/" rel="nofollow">Plume\xa0for\xa0Android</a>': 1}

<script.py> output:

{'en': 97, 'et': 1, 'und': 2}

{'<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>': 24, '<a href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>': 26, '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com" rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/" rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com" rel="nofollow">Linkis.com</a>': 2, '<a href="http://rutracker.org/forum/viewforum.php?f=93" rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com" rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/" rel="nofollow">Plume\xa0for\xa0Android</a>': 1}

**Daily XP950**

##### Exercise

##### Exercise

# Bringing it all together (2)

Wow, you've just generalized your Twitter language analysis that you did in the previous chapter to include a default argument for the column name. You're now going to generalize this function one step further by allowing the user to pass it a flexible argument, that is, in this case, as many column names as the user would like!

Once again, for your convenience, pandas has been imported as pd and the 'tweets.csv' file has been imported into the DataFrame tweets\_df. Parts of the code from your previous work are also provided.

##### Instructions

**100 XP**

* Complete the function header by supplying the parameter for the DataFrame df and the flexible argument \*args.
* Complete the for loop within the function definition so that the loop occurs over the tuple args.
* Call count\_entries() by passing the tweets\_df DataFrame and the column name 'lang'. Assign the result to result1.
* Call count\_entries() by passing the tweets\_df DataFrame and the column names 'lang' and 'source'. Assign the result to result2.

# Define count\_entries()

def \_\_\_\_(\_\_\_\_, \_\_\_\_):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    #Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Iterate over column names in args

    for col\_name in \_\_\_\_:

        # Extract column from DataFrame: col

        col = df[col\_name]

        # Iterate over the column in DataFrame

        for entry in col:

            # If entry is in cols\_count, add 1

            if entry in cols\_count.keys():

                cols\_count[entry] += 1

            # Else add the entry to cols\_count, set the value to 1

            else:

                cols\_count[entry] = 1

    # Return the cols\_count dictionary

    return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(\_\_\_\_, \_\_\_\_)

# Call count\_entries(): result2

result2 = count\_entries(\_\_\_\_, \_\_\_\_, \_\_\_\_)

# Print result1 and result2

print(result1)

print(result2)



# Define count\_entries()

def count\_entries(df, col\_name='lang'):

"""Return a dictionary with counts of

occurrences as value for each key."""

# Initialize an empty dictionary: cols\_count

cols\_count = {}

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df)

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, col\_name='source')

# Print result1 and result2

print(result1)

print(result2)

# Define count\_entries()

def count\_entries(df, \*args):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    #Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Iterate over column names in args

    for col\_name in args:

        # Extract column from DataFrame: col

        col = df[col\_name]

        # Iterate over the column in DataFrame

        for entry in col:

            # If entry is in cols\_count, add 1

            if entry in cols\_count.keys():

                cols\_count[entry] += 1

            # Else add the entry to cols\_count, set the value to 1

            else:

                cols\_count[entry] = 1

    # Return the cols\_count dictionary

    return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, 'lang', 'source')

# Print result1 and result2

print(result1)

print(result2)

# Define count\_entries()

def count\_entries(df, \*args):

"""Return a dictionary with counts of

occurrences as value for each key."""

#Initialize an empty dictionary: cols\_count

cols\_count = {}

# Iterate over column names in args

for col\_name in args:

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, 'lang', 'source')

# Print result1 and result2

print(result1)

print(result2)

{'en': 97, 'et': 1, 'und': 2}

{'en': 97, 'et': 1, 'und': 2, '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>': 24, '<a href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>': 26, '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com" rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/" rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com" rel="nofollow">Linkis.com</a>': 2, '<a href="http://rutracker.org/forum/viewforum.php?f=93" rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com" rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/" rel="nofollow">Plume\xa0for\xa0Android</a>': 1}

<script.py> output:

{'en': 97, 'et': 1, 'und': 2}

{'en': 97, 'et': 1, 'und': 2, '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>': 24, '<a href="http://www.facebook.com/twitter" rel="nofollow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>': 26, '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com" rel="nofollow">Twitter for BlackBerry</a>': 2, '<a href="http://www.google.com/" rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com" rel="nofollow">Linkis.com</a>': 2, '<a href="http://rutracker.org/forum/viewforum.php?f=93" rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.com" rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/" rel="nofollow">Plume\xa0for\xa0Android</a>': 1}

o available on the mobile app

## 1. Lambda functions

You've written your very own Python functions using the def keyword, function headers, docstrings and function bodies.

## 2. Lambda functions

There's a quicker way to write functions on the fly and these are called lambda functions because you use the keyword lambda. Here we re-write our function raise\_to\_power as a lambda function. To do so, after the keyword lambda, we specify the names of the arguments; then we use a colon followed by the expression that specifies what we wish the function to return. Lambda functions allow you to write functions in a quick and potentially dirty way so I wouldn't advise you to use them all the time but there are situations when they can come in very handy.

## 3. Anonymous functions

For example, check out the map function, which takes two arguments, a function and a sequence such as a list and applies the function over all elements of the sequence. We can pass lambda functions to map without even naming them and in this case we refer to them as anonymous functions. In this example, we use map on a lambda function that squares all elements of a list and we'll store the result in square\_all. Printing square\_all reveals that it is actually a map object so to see what it contains we use the function list to turn it into a list and print the results to the shell. As expected, it's a list containing the squares of the elements in the original list!

## 4. Let's practice!

In the following interactive exercises, you'll become a boss at writing lambda functions and see a number of other cool uses for them. Get hacking!

**Daily XP50**

##### Exercise

##### Exercise

# Pop quiz on lambda functions

In this exercise, you will practice writing a simple lambda function and calling this function. Recall what you know about lambda functions and answer the following questions:

* How would you write a lambda function add\_bangs that adds three exclamation points '!!!' to the end of a string a?
* How would you call add\_bangs with the argument 'hello'?

You may use the IPython shell to test your code.

##### Instructions

**50 XP**

##### Possible Answers

* 

The lambda function definition is: add\_bangs = (a + '!!!'), and the function call is: add\_bangs('hello').

* **This is the correct answer**

The lambda function definition is: **add\_bangs = (lambda a: a + '!!!'**), and the function call is**: add\_bangs('hello').**

* 

The lambda function definition is: (lambda a: a + '!!!') = add\_bangs, and the function call is: add\_bangs('hello').

# Define count\_entries()

def count\_entries(df, \*args):

"""Return a dictionary with counts of

occurrences as value for each key."""

#Initialize an empty dictionary: cols\_count

cols\_count = {}

# Iterate over column names in args

for col\_name in args:

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, 'lang', 'source')

# Print result1 and result2

print(result1)

print(result2)

In [1]:

add\_bangs = lambda a : a + '!!!'

In [2]:

add\_bangs('hello')

Out[2]:

'hello!!!'

# Writing a lambda function you already know

Some function definitions are simple enough that they can be converted to a lambda function. By doing this, you write less lines of code, which is pretty awesome and will come in handy, especially when you're writing and maintaining big programs. In this exercise, you will use what you know about lambda functions to convert a function that does a simple task into a lambda function. Take a look at this function definition:

def echo\_word(word1, echo):

"""Concatenate echo copies of word1."""

words = word1 \* echo

return words

The function echo\_word takes 2 parameters: a string value, word1 and an integer value, echo. It returns a string that is a concatenation of echo copies of word1. Your task is to convert this simple function into a lambda function.

##### Instructions

**100 XP**

* Define the lambda function echo\_word using the variables word1 and echo. Replicate what the original function definition for echo\_word() does above.
* Call echo\_word() with the string argument 'hey' and the value 5, in that order. Assign the call to result.

# Define echo\_word as a lambda function: echo\_word

echo\_word = (\_\_\_\_)

# Call echo\_word: result

result = \_\_\_\_

# Print result

print(result)

# Define count\_entries()

def count\_entries(df, \*args):

"""Return a dictionary with counts of

occurrences as value for each key."""

#Initialize an empty dictionary: cols\_count

cols\_count = {}

# Iterate over column names in args

for col\_name in args:

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Call count\_entries(): result2

result2 = count\_entries(tweets\_df, 'lang', 'source')

# Print result1 and result2

print(result1)

print(result2)

# Define echo\_word as a lambda function: echo\_word

echo\_word = (lambda word1, echo : word1 \* echo)

# Call echo\_word: result

result = echo\_word('hey', 5)

# Print result

print(result)

# Define echo\_word as a lambda function: echo\_word

echo\_word = (lambda word1, echo : word1 \* echo)

# Call echo\_word: result

result = echo\_word('hey', 5)

# Print result

print(result)

heyheyheyheyhey

<script.py> output:

heyheyheyheyhey

**Daily XP200**

##### Exercise

##### Exercise

# Map() and lambda functions

So far, you've used lambda functions to write short, simple functions as well as to redefine functions with simple functionality. The best use case for lambda functions, however, are for when you want these simple functionalities to be anonymously embedded within larger expressions. What that means is that the functionality is not stored in the environment, unlike a function defined with def. To understand this idea better, you will use a lambda function in the context of the map() function.

Recall from the video that map() applies a function over an object, such as a list. Here, you can use lambda functions to define the function that map() will use to process the object. For example:

nums = [2, 4, 6, 8, 10]

result = map(lambda a: a \*\* 2, nums)

You can see here that a lambda function, which raises a value a to the power of 2, is passed to map() alongside a list of numbers, nums. The map object that results from the call to map() is stored in result. You will now practice the use of lambda functions with map(). For this exercise, you will map the functionality of the add\_bangs() function you defined in previous exercises over a list of strings.

##### Instructions

**100 XP**

* In the map() call, pass a lambda function that concatenates the string '!!!' to a string item; also pass the list of strings, spells. Assign the resulting map object to shout\_spells.
* Convert shout\_spells to a list and print out the list.

# Create a list of strings: spells

spells = ["protego", "accio", "expecto patronum", "legilimens"]

# Use map() to apply a lambda function over spells: shout\_spells

\_\_\_\_ = map(\_\_\_\_, \_\_\_\_)

# Convert shout\_spells to a list: shout\_spells\_list

\_\_\_\_

# Print the result

print(shout\_spells\_list)

# Define echo\_word as a lambda function: echo\_word echo\_word = (lambda word1, echo : word1 \* echo) # Call echo\_word: result result = echo\_word('hey', 5) # Print result print(result)

# Create a list of strings: spells

spells = ["protego", "accio", "expecto patronum", "legilimens"]

# Use map() to apply a lambda function over spells: shout\_spells

shout\_spells = map(lambda item : item + '!!!', spells)

# Convert shout\_spells to a list: shout\_spells\_list

shout\_spells\_list = list(shout\_spells)

# Print the result

print(shout\_spells\_list)

# Create a list of strings: spells

spells = ["protego", "accio", "expecto patronum", "legilimens"]

# Use map() to apply a lambda function over spells: shout\_spells

shout\_spells = map(lambda item : item + '!!!', spells)

# Convert shout\_spells to a list: shout\_spells\_list

shout\_spells\_list = list(shout\_spells)

# Print the result

print(shout\_spells\_list)

['protego!!!', 'accio!!!', 'expecto patronum!!!', 'legilimens!!!']

<script.py> output:

['protego!!!', 'accio!!!', 'expecto patronum!!!', 'legilimens!!!']

# Filter() and lambda functions

In the previous exercise, you used lambda functions to anonymously embed an operation within map(). You will practice this again in this exercise by using a lambda function with filter(), which may be new to you! The function filter() offers a way to filter out elements from a list that don't satisfy certain criteria.

Your goal in this exercise is to use filter() to create, from an input list of strings, a new list that contains only strings that have more than 6 characters.

##### Instructions

**100 XP**

* In the filter() call, pass a lambda function and the list of strings, fellowship. The lambda function should check if the number of characters in a string member is greater than 6; use the len() function to do this. Assign the resulting filter object to result.
* Convert result to a list and print out the list.
* Create a list of strings: fellowship
* fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']
* # Use filter() to apply a lambda function over fellowship: result
* \_\_\_\_ = filter(\_\_\_\_, \_\_\_\_)
* # Convert result to a list: result\_list
* \_\_\_\_
* # Print result\_list
* print(result\_list)

# Create a list of strings: spells spells = ["protego", "accio", "expecto patronum", "legilimens"] # Use map() to apply a lambda function over spells: shout\_spells shout\_spells = map(lambda item : item + '!!!', spells) # Convert shout\_spells to a list: shout\_spells\_list shout\_spells\_list = list(shout\_spells) # Print the result print(shout\_spells\_list)

# Create a list of strings: fellowship

fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']

# Use filter() to apply a lambda function over fellowship: result

result = filter(lambda member : (len(member) >6) , fellowship)

# Convert result to a list: result\_list

result\_list = list(result)

# Print result\_list

print(result\_list)

# Create a list of strings: fellowship

fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']

# Use filter() to apply a lambda function over fellowship: result

result = filter(lambda member : (len(member) >6) , fellowship)

# Convert result to a list: result\_list

result\_list = list(result)

# Print result\_list

print(result\_list)

['samwise', 'aragorn', 'boromir', 'legolas', 'gandalf']

<script.py> output:

['samwise', 'aragorn', 'boromir', 'legolas', 'gandalf']

**Daily XP400**

##### Exercise

##### Exercise

# Reduce() and lambda functions

You're getting very good at using lambda functions! Here's one more function to add to your repertoire of skills. The reduce() function is useful for performing some computation on a list and, unlike map() and filter(), returns a single value as a result. To use reduce(), you must import it from the functools module.

Remember gibberish() from a few exercises back?

# Define gibberish

def gibberish(\*args):

"""Concatenate strings in \*args together."""

hodgepodge = ''

for word in args:

hodgepodge += word

return hodgepodge

gibberish() simply takes a list of strings as an argument and returns, as a single-value result, the concatenation of all of these strings. In this exercise, you will replicate this functionality by using reduce() and a lambda function that concatenates strings together.

##### Instructions

**100 XP**

* Import the reduce function from the functools module.
* In the reduce() call, pass a lambda function that takes two string arguments item1 and item2 and concatenates them; also pass the list of strings, stark. Assign the result to result. The first argument to reduce() should be the lambda function and the second argument is the list stark.

# Import reduce from functools

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(\_\_\_\_, \_\_\_\_)

# Print the result

print(result)

* Notes

# Create a list of strings: fellowship

fellowship = ['frodo', 'samwise', 'merry', 'pippin', 'aragorn', 'boromir', 'legolas', 'gimli', 'gandalf']

# Use filter() to apply a lambda function over fellowship: result

result = filter(lambda member : (len(member) >6) , fellowship)

# Convert result to a list: result\_list

result\_list = list(result)

# Print result\_list

print(result\_list)

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2 : item1 + item2, stark)

# Print the result

print(result)

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2 : item1 + item2, stark)

# Print the result

print(result)

robbsansaaryabrandonrickon

<script.py> output:

robbsansaaryabrandonrickon

## 1. Introduction to error handling

When you use a function incorrectly, it should throw you an error. For example,

## 2. The float() function

check out the function float that returns a floating point from a number or string, under the condition that the string corresponds to a number.

## 3. Passing an incorrect argument

When I pass the function float an integer, the corresponding float is returned; similarly if I pass it the string '2.3'. However, if I pass it the string 'hello', Python will throw me an error telling me that it couldn't convert the string to a float. In this case, it threw me a ValueError and there are many types of errors.

## 4. Passing valid arguments

When we write our own functions, as we have been doing, we may wish to catch specific problems and write specific error messages. Let's check out this user-defined function that computes the square root of a number. It behaves as expected with integers.

## 5. Passing invalid arguments

What happens if we pass it a string such as 'hello'? Then it throws me an error corresponding to a line of code within the function definition. This error says it was some sort of TypeError but the message may not be particularly useful to a user of our function, so we should endeavor to provide useful error messages for the functions we write.

## 6. Errors and exceptions

This is an example of an error caught during execution, commonly called exceptions. The main way to catch such exceptions is the try-except clause, in which Python tries to run the code following try and if it can, all is well. If it cannot due to an exception, it runs the code following except.

## 7. Errors and exceptions

Let's now rewrite our square root function but this time catch any exceptions raised. So here, we try to execute x to the power of zero point five; using except, in the case of an exception, we print 'x must be an int or float'. Now we see that the resulting function behaves well for ints and floats and also prints out what we wanted it to for a string.

## 8. Errors and exceptions

We may also wish to only catch TypeErrors and let other errors pass through, in which case we would use except TypeError as you can see here. There are many other types of exceptions that can be caught and you can have a look at them in the Python documentation available online.

## 9. Errors and exceptions

More often than not, instead of merely printing an error message, we'll want to actually raise an error by using the keyword raise. For example, our square root function does something we may not desire when applied to negative numbers. It actually returns a complex number which we may not want. In fact, let's say that we don't wish our function to work for negative numbers. Then using an if clause, we can raise a ValueError for cases in which the user passes the function a negative number.

## 10. Errors and exceptions

Let's now see it in action! If we pass our new function a negative number, see it returns the prescribed ValueError! That's enough out of me.

## 11. Let's practice!

It's time to get you writing your very own error messages to help people use your functions!

**Daily XP500**

##### Exercise

##### Exercise

# Pop quiz about errors

In the video, Hugo talked about how errors happen when functions are supplied arguments that they are unable to work with. In this exercise, you will identify which function call raises an error and what type of error is raised.

Take a look at the following function calls to len():

len('There is a beast in every man and it stirs when you put a sword in his hand.')

len(['robb', 'sansa', 'arya', 'eddard', 'jon'])

len(525600)

len(('jaime', 'cersei', 'tywin', 'tyrion', 'joffrey'))

Which of the function calls raises an error and what type of error is raised?

##### Instructions

**50 XP**

##### Possible Answers

* 

The call len('There is a beast in every man and it stirs when you put a sword in his hand.') raises a TypeError.

* 

The call len(['robb', 'sansa', 'arya', 'eddard', 'jon']) raises an IndexError.

* 

The call len(525600) raises a TypeError.

* 

The call len(('jaime', 'cersei', 'tywin', 'tyrion', 'joffrey')) raises a NameError

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2 : item1 + item2, stark)

# Print the result

print(result)

# Import reduce from functools from functools import reduce # Create a list of strings: stark stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon'] # Use reduce() to apply a lambda function over stark: result result = reduce(lambda item1,item2 : item1 + item2, stark) # Print the result print(result)

In [1]:

len('There is a beast in every man and it stirs when you put a sword in his hand.')

Out[1]:

76

In [2]:

len(['robb', 'sansa', 'arya', 'eddard', 'jon'])

Out[2]:

5

In [3]:

len(525600)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 1, in <module>

len(525600)

TypeError: object of type 'int' has no len()

In [4]:

len(('jaime', 'cersei', 'tywin', 'tyrion', 'joffrey'))

Out[4]:

5

# Error handling with try-except

A good practice in writing your own functions is also anticipating the ways in which other people (or yourself, if you accidentally misuse your own function) might use the function you defined.

As in the previous exercise, you saw that the len() function is able to handle input arguments such as strings, lists, and tuples, but not int type ones and raises an appropriate error and error message when it encounters invalid input arguments. One way of doing this is through exception handling with the try-except block.

In this exercise, you will define a function as well as use a try-except block for handling cases when incorrect input arguments are passed to the function.

Recall the shout\_echo() function you defined in previous exercises; parts of the function definition are provided in the sample code. Your goal is to complete the exception handling code in the function definition and provide an appropriate error message when raising an error.

##### Instructions

**100 XP**

* Initialize the variables echo\_word and shout\_words to empty strings.
* Add the keywords try and except in the appropriate locations for the exception handling block.
* Use the \* operator to concatenate echo copies of word1. Assign the result to echo\_word.
* Concatenate the string '!!!' to echo\_word. Assign the result to shout\_words

# Define shout\_echo

def shout\_echo(word1, echo=1):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Initialize empty strings: echo\_word, shout\_words

    # Add exception handling with try-except

    \_\_\_\_:

        # Concatenate echo copies of word1 using \*: echo\_word

        echo\_word = \_\_\_\_

        # Concatenate '!!!' to echo\_word: shout\_words

        shout\_words = \_\_\_\_

    \_\_\_\_:

        # Print error message

        print("word1 must be a string and echo must be an integer.")

    # Return shout\_words

    return shout\_words

# Call shout\_echo

shout\_echo("particle", echo="accelerator")

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2 : item1 + item2, stark)

# Print the result

print(result)

# Define shout\_echo

def shout\_echo(word1, echo=1):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Initialize empty strings: echo\_word, shout\_words

    echo\_word = ""

    shout\_words = ""

    # Add exception handling with try-except

    try:

        # Concatenate echo copies of word1 using \*: echo\_word

        echo\_word = word1 \* echo

        # Concatenate '!!!' to echo\_word: shout\_words

        shout\_words = echo\_word + '!!!'

    except:

        # Print error message

        print("word1 must be a string and echo must be an integer.")

    # Return shout\_words

    return shout\_words

# Call shout\_echo

shout\_echo("particle", echo="accelerator")

# Import reduce from functools

from functools import reduce

# Create a list of strings: stark

stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']

# Use reduce() to apply a lambda function over stark: result

result = reduce(lambda item1,item2 : item1 + item2, stark)

# Print the result

print(result)

# Define shout\_echo

def shout\_echo(word1, echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Initialize empty strings: echo\_word, shout\_words

echo\_word = ""

shout\_words = ""

# Add exception handling with try-except

try:

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_words

shout\_words = echo\_word + '!!!'

except:

# Print error message

print("word1 must be a string and echo must be an integer.")

# Return shout\_words

return shout\_words

# Call shout\_echo

shout\_echo("particle", echo="accelerator")

word1 must be a string and echo must be an integer.

''

<script.py> output:

word1 must be a string and echo must be an integer.

# Error handling by raising an error

Another way to raise an error is by using raise. In this exercise, you will add a raise statement to the shout\_echo() function you defined before to raise an error message when the value supplied by the user to the echo argument is less than 0.

The call to shout\_echo() uses valid argument values. To test and see how the raise statement works, simply change the value for the echo argument to a negative value. Don't forget to change it back to valid values to move on to the next exercise!

##### Instructions

**100 XP**

* Complete the if statement by checking if the value of echo is less than 0.
* In the body of the if statement, add a raise statement that raises a ValueError with message 'echo must be greater than or equal to 0' when the value supplied by the user to echo is less than 0.

# Define shout\_echo

def shout\_echo(word1, echo=1):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Raise an error with raise

    if \_\_\_\_:

        \_\_\_\_ \_\_\_\_(\_\_\_\_)

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Concatenate '!!!' to echo\_word: shout\_word

    shout\_word = echo\_word + '!!!'

    # Return shout\_word

    return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=5)

# Define shout\_echo

def shout\_echo(word1, echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Initialize empty strings: echo\_word, shout\_words

echo\_word = ""

shout\_words = ""

# Add exception handling with try-except

try:

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_words

shout\_words = echo\_word + '!!!'

except:

# Print error message

print("word1 must be a string and echo must be an integer.")

# Return shout\_words

return shout\_words

# Call shout\_echo

shout\_echo("particle", echo="accelerator")

# Define shout\_echo

def shout\_echo(word1, echo=1):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Raise an error with raise

    if echo <0:

        raise ValueError('echo must be greater than or equal to 0')

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Concatenate '!!!' to echo\_word: shout\_word

    shout\_word = echo\_word + '!!!'

    # Return shout\_word

    return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=-1)

# Define shout\_echo

def shout\_echo(word1, echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Raise an error with raise

if echo <0:

raise ValueError('echo must be greater than or equal to 0')

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=-1)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 20, in <module>

shout\_echo("particle", echo=-1)

File "<stdin>", line 8, in shout\_echo

raise ValueError('echo must be greater than or equal to 0')

ValueError: echo must be greater than or equal to 0

# Define shout\_echo

def shout\_echo(word1, echo=1):

    """Concatenate echo copies of word1 and three

    exclamation marks at the end of the string."""

    # Raise an error with raise

    if echo <0:

        raise ValueError('echo must be greater than or equal to 0')

    # Concatenate echo copies of word1 using \*: echo\_word

    echo\_word = word1 \* echo

    # Concatenate '!!!' to echo\_word: shout\_word

    shout\_word = echo\_word + '!!!'

    # Return shout\_word

    return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=5)

# Define shout\_echo

def shout\_echo(word1, echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Raise an error with raise

if echo <0:

raise ValueError('echo must be greater than or equal to 0')

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=-1)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 20, in <module>

shout\_echo("particle", echo=-1)

File "<stdin>", line 8, in shout\_echo

raise ValueError('echo must be greater than or equal to 0')

ValueError: echo must be greater than or equal to 0

Traceback (most recent call last):

File "script.py", line 72, in exceptionCatcher

raise exception

File "script.py", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "script.py", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "script.py", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "script.py", line 72, in exceptionCatcher

raise exception

File "script.py", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "script.py", line 20, in <module>

shout\_echo("particle", echo=-1)

File "script.py", line 8, in shout\_echo

raise ValueError('echo must be greater than or equal to 0')

ValueError: echo must be greater than or equal to 0

# Define shout\_echo

def shout\_echo(word1, echo=1):

"""Concatenate echo copies of word1 and three

exclamation marks at the end of the string."""

# Raise an error with raise

if echo <0:

raise ValueError('echo must be greater than or equal to 0')

# Concatenate echo copies of word1 using \*: echo\_word

echo\_word = word1 \* echo

# Concatenate '!!!' to echo\_word: shout\_word

shout\_word = echo\_word + '!!!'

# Return shout\_word

return shout\_word

# Call shout\_echo

shout\_echo("particle", echo=5)

'particleparticleparticleparticleparticle!!!'

## 1. Bringing it all together

You're now going to use your hard-earnt skills to write error messages into your DataFrame analyzer that you have been building up in previous chapters:

## 2. Errors and exceptions

let's say that a user of your function passes your function the name of a column that isn't a column in the DataFrame that they pass it; you'll want to let them know! In the following interactive exercises, you'll write error messages using two methods that you have learned: one, using the try-except syntax that you see here;

## 3. Errors and exceptions

two: explicitly raising errors using the keyword raise as in this example.

## 4. Let's practice!

Now that's enough out of me, happy function writing and error handling!

**Daily XP800**

##### Exercise

##### Exercise

# Bringing it all together (1)

This is awesome! You have now learned how to write anonymous functions using lambda, how to pass lambda functions as arguments to other functions such as map(), filter(), and reduce(), as well as how to write errors and output custom error messages within your functions. You will now put together these learnings to good use by working with a Twitter dataset. Before practicing your new error handling skills; in this exercise, you will write a lambda function and use filter() to select retweets, that is, tweets that begin with the string 'RT'.

To help you accomplish this, the Twitter data has been imported into the DataFrame, tweets\_df. Go for it!

##### Instructions

**100 XP**

* In the filter() call, pass a lambda function and the sequence of tweets as strings, tweets\_df['text']. The lambda function should check if the first 2 characters in a tweet x are 'RT'. Assign the resulting filter object to result. To get the first 2 characters in a tweet x, use x[0:2]. To check equality, use a Boolean filter with ==.
* Convert result to a list and print out the list.

# Select retweets from the Twitter DataFrame: result

result = filter(\_\_\_\_, \_\_\_\_)

# Create list from filter object result: res\_list

\_\_\_\_

# Print all retweets in res\_list

for tweet in res\_list:

    print(tweet)

# Define shout\_echo def shout\_echo(word1, echo=1): """Concatenate echo copies of word1 and three exclamation marks at the end of the string.""" # Raise an error with raise if echo <0: raise ValueError('echo must be greater than or equal to 0') # Concatenate echo copies of word1 using \*: echo\_word echo\_word = word1 \* echo # Concatenate '!!!' to echo\_word: shout\_word shout\_word = echo\_word + '!!!' # Return shout\_word return shout\_word # Call shout\_echo shout\_echo("particle", echo=5)

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x : x[0:2] == 'RT' , tweets\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

    print(tweet)

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x : x[0:2] == 'RT' , tweet\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

print(tweet)

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 2, in <module>

result = filter(lambda x : x[0:2] == 'RT' , tweet\_df['text'])

NameError: name 'tweet\_df' is not defined

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x : x[0:2] == 'RT' , tweets\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

print(tweet)

RT @bpolitics: .@krollbondrating's Christopher Whalen says Clinton is the weakest Dem candidate in 50 years https://t.co/pLk7rvoRSn https:/…

RT @HeidiAlpine: @dmartosko Cruz video found.....racing from the scene.... #cruzsexscandal https://t.co/zuAPZfQDk3

RT @AlanLohner: The anti-American D.C. elites despise Trump for his America-first foreign policy. Trump threatens their gravy train. https:…

RT @BIackPplTweets: Young Donald trump meets his neighbor https://t.co/RFlu17Z1eE

RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective amnisia.

RT @HouseCracka: 29,000+ PEOPLE WATCHING TRUMP LIVE ON ONE STREAM!!!

https://t.co/7QCFz9ehNe

RT @urfavandtrump: RT for Brendon Urie

Fav for Donald Trump https://t.co/PZ5vS94lOg

RT @trapgrampa: This is how I see #Trump every time he speaks. https://t.co/fYSiHNS0nT

RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective amnisia.

RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT RAISING TAXES in WI! #LyinTrump

#NeverTrump #CruzCrew https…

RT @urfavandtrump: RT for Brendon Urie

Fav for Donald Trump https://t.co/PZ5vS94lOg

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT RAISING TAXES in WI! #LyinTrump

#NeverTrump #CruzCrew https…

RT @trapgrampa: This is how I see #Trump every time he speaks. https://t.co/fYSiHNS0nT

RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump at any time and Corey can do nothing? Michelle is clearly…

RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis #tcot #PJNET https…

RT @DRUDGE\_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who has terminal illness... https://t.co/qt06aG9inT

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't know how great he is, advice watch live streaming https://…

RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis #tcot #PJNET https…

RT @DRUDGE\_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who has terminal illness... https://t.co/qt06aG9inT

RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't know how great he is, advice watch live streaming https://…

RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump at any time and Corey can do nothing? Michelle is clearly…

RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…

RT @Norsu2: Nightmare WI poll for Ted Cruz has Kasich surging: Trump 29, Kasich 27, Cruz 25. https://t.co/lJsgbLYY1P #NeverTrump

RT @thehill: WATCH: Protester pepper-sprayed point blank at Trump rally https://t.co/B5f65Al9ld https://t.co/skAfByXuQc

RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @DebbieStout5: Wow! Last I checked it was just 12 points &amp; that wasn't more than a day ago. Oh boy Trump ppl might want to rethink🤔 http…

RT @tyleroakley: i'm a messy bitch, but at least i'm not voting for trump

RT @vandives: Trump supporters r tired of justice NOT being served. There's no justice anymore. Hardworking Americans get screwed. That's n…

RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.

RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.

RT @urfavandtrump: RT for Jerrie (Little Mix)

Fav for Donald Trump https://t.co/nEVxElW6iG

RT @urfavandtrump: RT for Jerrie (Little Mix)

Fav for Donald Trump https://t.co/nEVxElW6iG

RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…

RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…

RT @AIIAmericanGirI: VIDEO=&gt; Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face

https://t.co/GeEryMDuDY

RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…

RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump

URGENT PUBLIC TRUMP ALERT:

COVERT KILL MEANS https:…

RT @AIIAmericanGirI: VIDEO=&gt; Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face

https://t.co/GeEryMDuDY

RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…

RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump

URGENT PUBLIC TRUMP ALERT:

COVERT KILL MEANS https:…

RT @Schneider\_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…

RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his college educat…

RT @Schneider\_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…

RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his college educat…

RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to his wife @T…

RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #GOP https:/…

RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didn’t come out of nowhere." —Hillary

https…

RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to his wife @T…

RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didn’t come out of nowhere." —Hillary

https…

RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #GOP https:/…

RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, repeating t…

RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, repeating t…

RT @GlendaJazzey: Donald Trump’s Campaign Financing Dodge, @rrotunda https://t.co/L8flI4lswG via @VerdictJustia

RT @TUSK81: LOUDER FOR THE PEOPLE IN THE BACK https://t.co/hlPVyNLXzx

RT @loopzoop: Well...put it back https://t.co/8Yb7BDT5VM

RT @claytoncubitt: Stop asking Bernie supporters if they’ll vote for Hillary against Trump. We got a plan to beat Trump already. Called Ber…

RT @akaMaude13: Seriously can't make this up. What a joke. #NeverTrump https://t.co/JkTx6mdRgC

<script.py> output:

RT @bpolitics: .@krollbondrating's Christopher Whalen says Clinton is the weakest Dem candidate in 50 years https://t.co/pLk7rvoRSn https:/…

RT @HeidiAlpine: @dmartosko Cruz video found.....racing from the scene.... #cruzsexscandal https://t.co/zuAPZfQDk3

RT @AlanLohner: The anti-American D.C. elites despise Trump for his America-first foreign policy. Trump threatens their gravy train. https:…

RT @BIackPplTweets: Young Donald trump meets his neighbor https://t.co/RFlu17Z1eE

RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective amnisia.

RT @HouseCracka: 29,000+ PEOPLE WATCHING TRUMP LIVE ON ONE STREAM!!!

https://t.co/7QCFz9ehNe

RT @urfavandtrump: RT for Brendon Urie

Fav for Donald Trump https://t.co/PZ5vS94lOg

RT @trapgrampa: This is how I see #Trump every time he speaks. https://t.co/fYSiHNS0nT

RT @trumpresearch: @WaitingInBagdad @thehill Trump supporters have selective amnisia.

RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT RAISING TAXES in WI! #LyinTrump

#NeverTrump #CruzCrew https…

RT @urfavandtrump: RT for Brendon Urie

Fav for Donald Trump https://t.co/PZ5vS94lOg

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT RAISING TAXES in WI! #LyinTrump

#NeverTrump #CruzCrew https…

RT @trapgrampa: This is how I see #Trump every time he speaks. https://t.co/fYSiHNS0nT

RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump at any time and Corey can do nothing? Michelle is clearly…

RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis #tcot #PJNET https…

RT @DRUDGE\_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who has terminal illness... https://t.co/qt06aG9inT

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't know how great he is, advice watch live streaming https://…

RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis #tcot #PJNET https…

RT @DRUDGE\_REPORT: VIDEO: Trump emotional moment with Former Miss Wisconsin who has terminal illness... https://t.co/qt06aG9inT

RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't know how great he is, advice watch live streaming https://…

RT @mitchellvii: So let me get this straight. Any reporter can assault Mr Trump at any time and Corey can do nothing? Michelle is clearly…

RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…

RT @Norsu2: Nightmare WI poll for Ted Cruz has Kasich surging: Trump 29, Kasich 27, Cruz 25. https://t.co/lJsgbLYY1P #NeverTrump

RT @thehill: WATCH: Protester pepper-sprayed point blank at Trump rally https://t.co/B5f65Al9ld https://t.co/skAfByXuQc

RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either to love or hate him https://t.co/QXwquVgs3c https://t.co/P9N…

RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whatever... https:/…

RT @DebbieStout5: Wow! Last I checked it was just 12 points &amp; that wasn't more than a day ago. Oh boy Trump ppl might want to rethink🤔 http…

RT @tyleroakley: i'm a messy bitch, but at least i'm not voting for trump

RT @vandives: Trump supporters r tired of justice NOT being served. There's no justice anymore. Hardworking Americans get screwed. That's n…

RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.

RT @AP: BREAKING: Trump vows to stand by campaign manager charged with battery, says he does not discard people.

RT @urfavandtrump: RT for Jerrie (Little Mix)

Fav for Donald Trump https://t.co/nEVxElW6iG

RT @urfavandtrump: RT for Jerrie (Little Mix)

Fav for Donald Trump https://t.co/nEVxElW6iG

RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…

RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…

RT @AIIAmericanGirI: VIDEO=&gt; Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face

https://t.co/GeEryMDuDY

RT @NoahCRothman: When Walker was fighting for reforms, Trump was defending unions and collective bargaining privileges https://t.co/e1UWNN…

RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump

URGENT PUBLIC TRUMP ALERT:

COVERT KILL MEANS https:…

RT @AIIAmericanGirI: VIDEO=&gt; Anti-Trump Protester SLUGS Elderly Trump Supporter in the Face

https://t.co/GeEryMDuDY

RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will n…

RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump

URGENT PUBLIC TRUMP ALERT:

COVERT KILL MEANS https:…

RT @Schneider\_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…

RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his college educat…

RT @Schneider\_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Emancipation P…

RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his college educat…

RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to his wife @T…

RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #GOP https:/…

RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didn’t come out of nowhere." —Hillary

https…

RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to his wife @T…

RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didn’t come out of nowhere." —Hillary

https…

RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #GOP https:/…

RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, repeating t…

RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, repeating t…

RT @GlendaJazzey: Donald Trump’s Campaign Financing Dodge, @rrotunda https://t.co/L8flI4lswG via @VerdictJustia

RT @TUSK81: LOUDER FOR THE PEOPLE IN THE BACK https://t.co/hlPVyNLXzx

RT @loopzoop: Well...put it back https://t.co/8Yb7BDT5VM

RT @claytoncubitt: Stop asking Bernie supporters if they’ll vote for Hillary against Trump. We got a plan to beat Trump already. Called Ber…

RT @akaMaude13: Seriously can't make this up. What a joke. #NeverTrump https://t.co/JkTx6mdRgC

# Bringing it all together (2)

Sometimes, we make mistakes when calling functions - even ones you made yourself. But don't fret! In this exercise, you will improve on your previous work with the count\_entries() function in the last chapter by adding a try-except block to it. This will allow your function to provide a helpful message when the user calls your count\_entries() function but provides a column name that isn't in the DataFrame.

Once again, for your convenience, pandas has been imported as pd and the 'tweets.csv' file has been imported into the DataFrame tweets\_df. Parts of the code from your previous work are also provided.

##### Instructions

**100 XP**

* Add a try block so that when the function is called with the correct arguments, it processes the DataFrame and returns a dictionary of results.
* Add an except block so that when the function is called incorrectly, it displays the following error message: 'The DataFrame does not have a ' + col\_name + ' column.'.

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Add try block

    \_\_\_\_:

        # Extract column from DataFrame: col

        col = df[col\_name]

        # Iterate over the column in DataFrame

        for entry in col:

            # If entry is in cols\_count, add 1

            if entry in cols\_count.keys():

                cols\_count[entry] += 1

            # Else add the entry to cols\_count, set the value to 1

            else:

                cols\_count[entry] = 1

        # Return the cols\_count dictionary

        return cols\_count

    # Add except block

    \_\_\_\_:

        \_\_\_\_

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Print result1

print(result1)

# Select retweets from the Twitter DataFrame: result result = filter(lambda x : x[0:2] == 'RT' , tweets\_df['text']) # Create list from filter object result: res\_list res\_list = list(result) # Print all retweets in res\_list for tweet in res\_list: print(tweet)

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Add try block

    try:

        # Extract column from DataFrame: col

        col = df[col\_name]

        # Iterate over the column in DataFrame

        for entry in col:

            # If entry is in cols\_count, add 1

            if entry in cols\_count.keys():

                cols\_count[entry] += 1

            # Else add the entry to cols\_count, set the value to 1

            else:

                cols\_count[entry] = 1

        # Return the cols\_count dictionary

        return cols\_count

    # Add except block

    except:

        print('The DataFrame does not have a ' + col\_name + ' column.')

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Print result1

print(result1)

# Select retweets from the Twitter DataFrame: result

result = filter(lambda x : x[0:2] == 'RT' , tweets\_df['text'])

# Create list from filter object result: res\_list

res\_list = list(result)

# Print all retweets in res\_list

for tweet in res\_list:

print(tweet)

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

"""Return a dictionary with counts of

occurrences as value for each key."""

# Initialize an empty dictionary: cols\_count

cols\_count = {}

# Add try block

try:

# Extract column from DataFrame: col

col = df[col\_name]

# Iterate over the column in DataFrame

for entry in col:

# If entry is in cols\_count, add 1

if entry in cols\_count.keys():

cols\_count[entry] += 1

# Else add the entry to cols\_count, set the value to 1

else:

cols\_count[entry] = 1

# Return the cols\_count dictionary

return cols\_count

# Add except block

except:

print('The DataFrame does not have a ' + col\_name + ' column.')

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df, 'lang')

# Print result1

print(result1)

{'en': 97, 'et': 1, 'und': 2}

<script.py> output:

{'en': 97, 'et': 1, 'und': 2}

**Daily XP1000**

##### Exercise

##### Exercise

# Bringing it all together (3)

In the previous exercise, you built on your function count\_entries() to add a try-except block. This was so that users would get helpful messages when calling your count\_entries() function and providing a column name that isn't in the DataFrame. In this exercise, you'll instead raise a ValueError in the case that the user provides a column name that isn't in the DataFrame.

Once again, for your convenience, pandas has been imported as pd and the 'tweets.csv' file has been imported into the DataFrame tweets\_df. Parts of the code from your previous work are also provided.

##### Instructions

**100 XP**

* If col\_name is not a column in the DataFrame df, raise a ValueError 'The DataFrame does not have a ' + col\_name + ' column.'.
* Call your new function count\_entries() to analyze the 'lang' column of tweets\_df. Store the result in result1.
* Print result1. This has been done for you, so hit 'Submit Answer' to check out the result. In the next exercise, you'll see that it raises the necessary ValueErrors.
* # Define count\_entries()
* def count\_entries(df, col\_name='lang'):
* """Return a dictionary with counts of
* occurrences as value for each key."""
* # Raise a ValueError if col\_name is NOT in DataFrame
* if col\_name not in df.columns:
* \_\_\_\_
* # Initialize an empty dictionary: cols\_count
* cols\_count = {}
* # Extract column from DataFrame: col
* col = df[col\_name]
* # Iterate over the column in DataFrame
* for entry in col:
* # If entry is in cols\_count, add 1
* if entry in cols\_count.keys():
* cols\_count[entry] += 1
* # Else add the entry to cols\_count, set the value to 1
* else:
* cols\_count[entry] = 1
* # Return the cols\_count dictionary
* return cols\_count
* # Call count\_entries(): result1
* \_\_\_\_
* # Print result1
* print(result1)

# Define count\_entries()

def count\_entries(df, col\_name='lang'):

    """Return a dictionary with counts of

    occurrences as value for each key."""

    # Raise a ValueError if col\_name is NOT in DataFrame

    if col\_name not in df.columns:

        raise ValueError ('The DataFrame does not have a ' + col\_name + ' column.')

    # Initialize an empty dictionary: cols\_count

    cols\_count = {}

    # Extract column from DataFrame: col

    col = df[col\_name]

    # Iterate over the column in DataFrame

    for entry in col:

        # If entry is in cols\_count, add 1

        if entry in cols\_count.keys():

            cols\_count[entry] += 1

            # Else add the entry to cols\_count, set the value to 1

        else:

            cols\_count[entry] = 1

        # Return the cols\_count dictionary

    return cols\_count

# Call count\_entries(): result1

result1 = count\_entries(tweets\_df,'lang')

# Print result1

print(result1)

In [1]:

count\_entries(tweets\_df, 'lang')

Out[1]:

{'en': 97, 'et': 1, 'und': 2}

In [2]:

count\_entries(tweets\_df, 'lang1')

Traceback (most recent call last):

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3361, in run\_ast\_nodes

if (await self.run\_code(code, result, async\_=asy)):

File "<stdin>", line 3458, in run\_code

self.showtraceback(running\_compiled\_code=True)

File "<stdin>", line 2066, in showtraceback

self.\_showtraceback(etype, value, stb)

File "<stdin>", line 72, in exceptionCatcher

raise exception

File "<stdin>", line 3441, in run\_code

exec(code\_obj, self.user\_global\_ns, self.user\_ns)

File "<stdin>", line 1, in <module>

count\_entries(tweets\_df, 'lang1')

File "<stdin>", line 17, in count\_entries

raise ValueError('The DataFrame does not have a ' + col\_name + ' column.')

ValueError: The DataFrame does not have a lang1 column.

## 1. Congratulations!

Well done. You're now well on your way to being a Pythonista Data Science ninja.

## 2. What you’ve learned:

You're now able to write functions in Python that accept single and multiple arguments and can return as many values as you please. You're also adept at using default and flexible arguments and keyword arguments. You've gained insight into scoping in Python, can write lambda functions and handle errors in your very own function writing practice. You've also gained invaluable practice in using all of these techniques to write functions that are useful in a Data Science context. You have come a long way in your developing practice as a budding Pythonista Data Scientist.

## 3. There’s more to learn!

There are more basic skills that you will need to learn in Python to be valuable as a working Data Scientist and many of these we'll cover in the sequel to this course so if you're finding yourself still thirsty for more Pythonista Data Science chops, I'd head over there right now. There you'll learn all about list comprehensions, which allow you to wrangle data in lists to create other lists, a tool utilized by all Data Scientists working in Python. You'll also learn about iterators, which you have already seen in the context of for loops without having necessarily known it. Iterators are everywhere in PythonLand and, to put it simply, allow you to rapidly iterate Data Science protocols and procedures over sets of objects; these are a couple of the cool functionalities in PythonLand you'll encounter in the sequel to this course, which will conclude with an entire chapter devoted to a case study in which you'll apply time and time again techniques learnt in both of these courses.

## 4. Let's practice!

I'm looking forward to seeing you there and congratulations once again!