Python Data Science Toolbox (Part 1)

Chapter 1 - Writing your own functions

Here you will learn how to write your very own functions. 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 newfound skills to questions that commonly arise in Data Science contexts.

Strings in Python

Strings 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?

In [ ]:

```
object1 = 'data' + 'analysis' + 'visualization'
object2 = 1*3
object3 = "1"*3

print(object1)
print(object2)
print(object3)
```

```
dataanalysisvisualization
3
111
```

Recapping built-in functions

In the video, Hugo briefly examined the return behavior of the built-in functions print() and str(). Here, you will use both functions and examine their return values. A variable x has been preloaded for this exercise. Run the code below in the console. Pay close attention to the results to answer the question that follows.

- Assign str(x) to a variable y1: y1 = str(x)
- Assign print(x) to a variable y2: y2 = print(x)
- Check the types of the variables x, y1, and y2.

What are the types of x, y1, and y2?

In [ ]:
```
x = 4.89
```

In [ ]:
```
y1 = str(x)
y2 = print(x)
```

```
4.89
```

In [ ]:
```
print(type(x))
print(type(y1))
print(type(y2))
```

```
<class 'float'>
<class 'str'>
<class 'NoneType'>
```

Correct! It is important to remember that assigning a variable y2 to a function that prints a value but does not return a value will result in that variable y2 being of type NoneType.

Write a simple 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 is found below. You can use it as a pattern to define shout().

```
def square():
    new_value = 4 ** 2
    return new_value
```

Function bodies need to be indented by a consistent number of spaces and the choice of 4 is common

In [ ]:
```
# 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!!!
```

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!

In [ ]:

```
# 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!!!
```

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.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

In [ ]:

```
# 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!!!
```

Great work! Here it made sense to assign the output of shout('congratulations') to a variable yell because the function shout actually returns a value, it does not merely print one.

Functions with multiple parameters

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.

In [ ]:
```
# 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!!!
```

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.

In [ ]:
```
nums = (3, 4, 6)
```

In [ ]:
```
type(nums)
```

```
Out[ ]: tuple
```

```
In [ ]: # Unpack nums into num1, num2, and num3
        num1, num2, num3 = nums
```

```
In [ ]: print(num1)
        print(num2)
        print(num3)
```

```
3
4
6
```

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!

```
In [ ]: # 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!!!
```

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.

In [ ]:
```python
# Import pandas
import pandas as pd

# Import Twitter data as DataFrame: df
df = pd.read_csv('C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course Dat

#df = pd.read_csv('tweets.csv')

# 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}
```

ringing 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.

In [ ]:
```python
# 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(df, 'lang')

# Print the result
print(result)
```

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

Chapter 2 - Default arguments, variable-length arguments and scope

In this chapter, you'll learn to write functions with default arguments, so that the user doesn't always need to specify them, and variable-length arguments, so that they can pass to your functions an arbitrary number of arguments. These are both incredibly useful tools! You'll also learn about the essential concept of scope. Enjoy!

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()?

```
In [ ]: num = 6
```

```
In [ ]: def func1():
            num = 3
            print(num)
```

```
In [ ]: func1()
```

```
3
```

```
In [ ]: def func2():
            global num
            double_num = num * 2
            num = 6
            print(double_num)
```

```
In [ ]: func2()
```

```
12
```

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.

In [ ]:
```
# 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
```

Nested Functions I

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!

In [ ]:
```
# 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!!!')
```

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!

In [ ]:

```
# 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
```

The keyword nonlocal and nested functions

Let's once again work further on your mastery of scope! In this exercise, you will use the keyword nonlocal within a nested function to alter the value of a variable defined in the enclosing scope.

In [ ]:
```python
# 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')
```

```
hellohello
hellohello!!!
```

Quite something, that nonlocal keyword!

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.

In [ ]:
```python
# 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 = echo*word1

    # 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!!!
```

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!

In [ ]:

```
# 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

    # Capitalize echo_word if intense is True
    if intense is True:
        # Capitalize 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
```

```python
# Call shout_echo() with "Hey", echo=5 and intense=True: with_big_echo
with_big_echo = shout_echo("Hey", 5, 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!!!
```

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.Within the function definition, args is a tuple.

In [ ]:

```python
# 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
```

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.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.

In [ ]:
```python
# 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
```

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.

In [ ]:
```python
# 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(df, 'lang')

# Call count_entries(): result2
result2 = count_entries(df, '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="nofoll
ow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>': 26, '<a href="htt
p://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twitter.com" rel="nofollow">Twi
tter for BlackBerry</a>': 2, '<a href="http://www.google.com/" rel="nofollow">Google</a>': 2, '<a href="http://twitter.com/#!/dow
nload/ipad" rel="nofollow">Twitter for iPad</a>': 6, '<a href="http://linkis.com" rel="nofollow">Linkis.com</a>': 2, '<a href="ht
tp://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">PlumeÂ forÂ Android</a>': 1}
```

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!

In [ ]:

```python
# 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(df, 'lang')

# Call count_entries(): result2
result2 = count_entries(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.face
book.com/twitter" rel="nofollow">Facebook</a>': 1, '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andr
oid</a>': 26, '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>': 33, '<a href="http://www.twit
ter.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">Linki
s.com</a>': 2, '<a href="http://rutracker.org/forum/viewforum.php?f=93" rel="nofollow">newzlasz</a>': 2, '<a href="http://ifttt.c
om" rel="nofollow">IFTTT</a>': 1, '<a href="http://www.myplume.com/" rel="nofollow">PlumeÂ forÂ Android</a>': 1}
```

Chapter 3 - Lambda functions and error-handling

Herein, you'll learn about lambda functions, which allow you to write functions quickly and on-the-fly. You'll also get practice at handling errors that your functions, at some point, will inevitably throw. You'll wrap up once again applying these skills to Data Science questions.

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'?

In [ ]:
```
add_bangs = lambda a : a + '!!!'
```

In [ ]:
```
add_bangs('hello')
```

Out[ ]:
```
'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.

In [ ]:
```
# Define echo_word as a lambda function: echo_word
echo_word = (lambda word1, echo : echo*word1)

# Call echo_word: result
result = echo_word('hey', 5)

# Print result
print(result)
```

```
heyheyheyheyhey
```

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.

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.

In [ ]:
```
# 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)
```

```
# Convert shout_spells into a list and print it
print(shout_spells_list)
```

```
['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.

In [ ]:
```
# 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)

# Convert result into a list and print it
print(result_list)
```

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

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.

```
In [ ]: # 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
```

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.

```
In [ ]: # 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 = echo * word1

                # 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.
Out[ ]: ''
```

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!

```
In [ ]: # 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 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)
```

```
Out[ ]: 'particleparticleparticleparticleparticle!!!'
```

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'.

```
In [ ]:
# Select retweets from the Twitter DataFrame: result
result = filter(lambda x: x[0:2] == 'RT', 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/pLk7rvoR
Sn https:/â€¦
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 tra
in. https:â€¦
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  httpsâ€¦
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 whateve
r... https:/â€¦
RT @Pjw20161951: NO KIDDING: #SleazyDonald just attacked Scott Walker for NOT RAISING TAXES in WI! #LyinTrump
#NeverTrump  #CruzCrew  httpsâ€¦
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 clearlyâ€¦
RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis   #tcot #P
JNET httpsâ€¦
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 whateve
r... https:/â€¦
RT @DennisApgar: Thank God I seen Trump at first stop in Wisconsin media doesn't know how great he is, advice watch live streamin
g https://â€¦
RT @paulbenedict7: How #Trump Sacks RINO Strongholds by Hitting Positions Held by Dems and GOP https://t.co/D7ulnAJhis   #tcot #P
JNET httpsâ€¦
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 streamin
g https://â€¦
RT @mitchellvii: So let me get this straight.  Any reporter can assault Mr Trump at any time and Corey can do nothing?  Michelle
is clearlyâ€¦
RT @sciam: Trump's idiosyncratic patterns of speech are why people tend either to love or hate him https://t.co/QXwquVgs3c http
s://t.co/P9Nâ€¦
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 http
s://t.co/P9Nâ€¦
RT @ggreenwald: The media spent all day claiming @SusanSarandon said she might vote for Trump. A total fabrication, but whateve
r... https:/â€¦
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
rethinkðŸ¤” httpâ€¦
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 screwe
d. That's nâ€¦
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/e1UWNNâ€¦
RT @RedheadAndRight: Report: Secret Service Says Michelle Fields Touched Trump https://t.co/c5c2sD8VO2

This is the only article you will nâ€¦
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/e1UWNNâ€¦
RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump
URGENT PUBLIC TRUMP ALERT:
COVERT KILL MEANS https:â€¦
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 nâ€¦
RT @JusticeRanger1: @realDonaldTrump @Pudingtane @DanScavino @GOP @infowars @EricTrump
URGENT PUBLIC TRUMP ALERT:
COVERT KILL MEANS https:â€¦
RT @Schneider_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Eman
cipation Pâ€¦
RT @RonBasler1: @DavidWhitDennis @realDonaldTrump @tedcruz

CRUZ SCREWS HOOKERS

CRUZ / CLINTON
RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his coll
ege educatâ€¦
RT @Schneider_CM: Trump says nobody had ever heard of executive orders before Obama started signing them. Never heard of the Eman
cipation Pâ€¦
RT @DonaldsAngel: Former Ms. WI just said that she is terminally ill but because of Trump pageant, her 7 yr. old son has his coll
ege educatâ€¦
RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to h
is wife @Tâ€¦
RT @RonBasler1: @DavidWhitDennis @realDonaldTrump @tedcruz

CRUZ SCREWS HOOKERS

CRUZ / CLINTON
RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #G
OP https:/â€¦
```

```
RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didnâ€™t come out of nowhere." â
€”Hillary
httpsâ€¦
RT @Dodarey: @DR8801 @SykesCharlie Charlie, let's see you get a straight "yes" or "no" answer from Cruz a/b being unfaithful to h
is wife @Tâ€¦
RT @HillaryClinton: "Every day, another Republican bemoans the rise of Donald Trump... but [he] didnâ€™t come out of nowhere." â
€”Hillary
httpsâ€¦
RT @RockCliffOne: Remember when the idea of a diabolical moron holding the world hostage was an idea for a funny movie? #Trump #G
OP https:/â€¦
RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, r
epeating tâ€¦
RT @immigrant4trump: @immigrant4trump msm, cable news attacking trump all day, from 8am to 10pm today, then the reruns come on, r
epeating tâ€¦
RT @GlendaJazzey: Donald Trumpâ€™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 theyâ€™ll vote for Hillary against Trump. We got a plan to beat Trump alread
y. Called Berâ€¦
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.

In [ ]:

```python
# 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(df, 'lang')

# Print result1
print(result1)
```

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

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.

In [ ]:

```
# 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(df)

# Print result1
print(result1)
```

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