Intro to Databases in Python

Ken Harmon

2020 June 27

```
## Warning: package 'reticulate' was built under R version 4.0.2
```

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 and keep it handy!

Instructions 100 XP 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.

```python
# 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!

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(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. 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 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
    shout_word = word + "!!!"
    # Replace print with return
    return(shout_word)

# Pass 'congratulations' to shout: yell
yell =  shout("congratulations")

# Print yell
print(yell)
```

```
## congratulations!!!
```

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 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(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. 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 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.

```
nums = (3,4,6)

# Unpack nums into num1, num2, and num3
num1, num2, num3 = nums

# Construct even_nums
even_nums = (2, num2, num3)
```

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 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(word1, word2):
    """Return a tuple of strings"""
    # Concatenate word1 with '!!!': shout1
    shout1 = word1 + '!!!'
    # Concatenate word2 with '!!!': shout2
    shout2 = word2 + '!!!'
```

```python
    # 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)
```

```
## congratulations!!!
```

```python
print(yell2)
```

```
## 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.

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 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.

```python
# Import pandas
import pandas as pd

# Import Twitter data as DataFrame: df
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}
```

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 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.

```
tweets_df = pd.read_csv('tweets.csv')

# 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}
```