

Chapter 8 - User-Defined Functions

User-Defined Functions

Lab - Functions

Lab 1

Lab 1 - Building a Command Line Application

The next couple of labs will walk you through making a command line application that sorts a slightly modified version of this [movie data](#).

► **How has the CSV file been modified?**

This lab focuses on reading the information from a CSV file, and then printing the information in a human readable way.

Reading the CSV

The most important function is the one that reads the information from a CSV file. Once the file has been read, the information will be stored in the variable `movie_data`, and the file will be closed. There is no need to leave the file open for this program. You will need global variables for the path and file name of the CSV file. The program should print None.

```
import csv

movie_csv = "student_folder/.labs/movie_data.csv"

def fetch_movie_data(movie_csv):
    """Return movie data from a CSV file"""
    pass

movie_data = fetch_movie_data(movie_csv)
print(movie_data)
```

► What does “pass” mean?

Using `with open`, read the entire CSV file and then pass it to a `csv.reader`. Create the local variable `movie_info` and set it to an empty list. Use a `for` loop to iterate through the file and append each row to the list `movie_info`. Once done iterating through the file, return `movie_info`. Running the program now should return a list of lists with a bunch of information that is hard to understand.

```
import csv

movie_csv = "student_folder/.labs/movie_data.csv"

def fetch_movie_data(movie_csv):
    """Return movie data from a CSV file"""
    with open(movie_csv, "r") as movie_file:
        reader = csv.reader(movie_file)
        movie_info = []
        for row in reader:
            movie_info.append(row)
        return movie_info

movie_data = fetch_movie_data(movie_csv)
print(movie_data)
```

Printing the Movie Information

Since this is a command line application, the output should be easy to read for humans. The `print` command will print square brackets, and it

does not format the output. It is not sufficient. Create the function `print_movie_data` that takes the parameter `data`. Use the unpacking method to be able to reference each element of the list. The `{:10}` syntax adds padding to the right of the string. This will align all of the data in neat rows. A `$` needs to be added for gross so that the user knows that column relates to money. Add a function call for `print_movie_data` and remove `print(movie_data)`.

► Formatting a String with Padding

```
import csv
```

```
movie_csv = "student_folder/labs/movie_data.csv"
```

```
def fetch_movie_data(movie_csv):
```

```
    """Return movie data from a CSV file"""
```

```
    with open(movie_csv, "r") as movie_file:
```

```
        reader = csv.reader(movie_file)
```

```
        movie_info = []
```

```
        for row in reader:
```

```
            movie_info.append(row)
```

```
        return movie_info
```

```
def print_movie_data(data):
```

```
    """Print the movie data in easy to read columns"""
```

```
    for title, genre, rotten, gross, year in data:
```

```
        print("{:36} {:10} {:18} ${:16} {}".format(title, genre, rotten, gross, year))
```

```
movie_data = fetch_movie_data(movie_csv)
```

```
print_movie_data(movie_data)
```

Lab 2

Lab 2 - Sorting the Movie Data

You saw how to sort a list with the `sort` method. But sorting the movie data is a bit tricky because it is a list of lists.

```
movie_data
[
  ["Top Gun", "1986", "5 Stars"],
  ["E.T.", "1981", "3 Stars"],
  ["Watchmen", "2009", "4 Stars"]
]
```

Outer list is a list of all of the movies

Inner list is info for each movie

List of Lists

Using the `sort` method on the outer list will arrange the movies in alphabetical order. We want the application to be able to sort the movie data in a variety of ways.

```
movie_data.sort()
[
  ["E.T.", "1981", "3 Stars"],
  ["Top Gun", "1986", "5 Stars"],
  ["Watchmen", "2009", "4 Stars"]
]
```

Sorting the outer list arranges the inner lists by their first element.

Sort Outer List

What if, for example, the user wants to sort the data by date of release? Using the `sort` method on each inner list rearranges the order of the movie info. That means the function you just wrote to print the movie info will not work as intended. There also exists a case in which the star rating will come before the date. A movie with a rating of "1 star" will come after the release date if it is "2007".

Sorting the inner lists arranges the order of the movie info.

```
movie_data[0].sort()

[
    ["1986", "5 Stars", "Top Gun"],
    ["E.T.", "1981", "3 Stars"],
    ["Watchmen", "2009", "4 Stars"]
]
```

Sort Inner Lists

Thankfully Python has a way to sort `movie_data` by elements of the inner lists without messing up the output of the program. Import the `operator` module along with `csv`. Next, define the function `sort_movie_data` with the parameters `data` and `index`. Make the function body `pass`.

```
import csv, operator
```

```
movie_csv = "student_folder/.labs/movie_data.csv"
```

```
def fetch_movie_data(movie_csv):
```

```
    """Return movie data from a CSV file"""
```

```
    with open(movie_csv, "r") as movie_file:
```

```
        reader = csv.reader(movie_file)
```

```
        movie_info = []
```

```
        for row in reader:
```

```
            movie_info.append(row)
```

```
        return movie_info
```

```
def print_movie_data(data):
```

```
    """Print the movie data in easy to read columns"""
```

```
    for title, genre, rotten, gross, year in data:
```

```
        print("{:36} {:10} {:18} {:16} {}".format(title, genre, rotten, gross, year))
```

```
def sort_movie_data(data, index):
```

```
    """Sort movie data based on the column data"""
```

```
    pass
```

```
movie_data = fetch_movie_data(movie_csv)
```

```
print_movie_data(movie_data)
```

The `index` parameter lets Python know which element to use for sorting the order of the inner lists. Sorting all of `movie_data` will take the headers

from the CSV file and mix them up with the movies themselves. Instead create a list called header and set its value to data[0]. Then make another list called sorted_movies and set its value to data[1:]. The first row of movie_data (the column titles) is stored in header. And the rest of movie_data is stored in sorted_movies. Except the movie information has not yet been sorted. You are going to use the sort method on sorted_movies, but put the following code in between the parentheses key=operator.itemgetter(index). Then return data. Finally, change the function call for print_movie_data to print_movie_data(sort_movie_data(movie_data, 0)).

```
import csv, operator
```

```
movie_csv = "student_folder/.labs/movie_data.csv"
```

```
def fetch_movie_data(movie_csv):
```

```
    """Return movie data from a CSV file"""
```

```
    with open(movie_csv, "r") as movie_file:
```

```
        reader = csv.reader(movie_file)
```

```
        movie_info = []
```

```
        for row in reader:
```

```
            movie_info.append(row)
```

```
        return movie_info
```

```
def print_movie_data(data):
```

```
    """Print the movie data in easy to read columns"""
```

```
    for title, genre, rotten, gross, year in data:
```

```
        print("{:36} {:10} {:18} {:16} {}".format(title, genre, rotten, gross, year))
```

```
def sort_movie_data(data, index):
```

```
    """Sort movie data based on the column data"""
```

```
    header = data[0]
```

```
    sorted_movies = data[1:]
```

```
    sorted_movies.sort(key=operator.itemgetter(index))
```

```
    sorted_movies.insert(0, header)
```

```
    return sorted_movies
```

```
movie_data = fetch_movie_data(movie_csv)
```

```
print_movie_data(sort_movie_data(movie_data, 0))
```

Sorting like this, however, not work with when the index is 3. That is

because that column is read as a string, not number. That means sorting this column will put \$110 before \$20. When index is 3, then the sorting should use floats instead of strings. To do this, first create the function `get_money` which takes `gross` as its parameter. This function returns element 3 which has been typecast as a float.

```
def get_money(gross):  
    return float(gross[3])
```

Next, modify `sort_movie_data` with a conditional that tests for when `index` is 3. If true, sort the movies with `key` set to the function `get_money`. If `index` is not 3, then use the sort code from the example above.

```
def sort_movie_data(data, index):  
    """Sort movie data based on the column data"""  
    header = data[0]  
    sorted_movies = data[1:]  
    if index == 3:  
        sorted_movies.sort(key=get_money)  
    else:  
        sorted_movies.sort(key=operator.itemgetter(index))  
    sorted_movies.insert(0, header)  
    return sorted_movies
```

Lab 3

Lab 3 - Ascending or Descending Order

The default sort in Python sorts from smallest to largest. Or, if the search key is a string, Python sorts from A to Z. Both of these examples are ascending order. A user may want to sort the data in descending order (largest to smallest or Z to A). Modify the `sort_movie_data` function to have a third parameter called `descending`, which will be a boolean value.

```
def sort_movie_data(data, index, descending):  
    """Sort movie data based on the column data"""  
    header = data[0]  
    sorted_movies = data[1:]  
    if index == 3:  
        sorted_movies.sort(key=get_money)  
    else:  
        sorted_movies.sort(key=operator.itemgetter(index))  
    sorted_movies.insert(0, header)  
    return sorted_movies
```

Add an if statement to determine if `descending` is true. The conditional should come after when `sorted_movies` is sorted, but before when `header` is inserted into `sorted_movies`. Use `pass` as a placeholder for now.

```
def sort_movie_data(data, index, descending):  
    """Sort movie data based on the column data"""  
    header = data[0]  
    sorted_movies = data[1:]  
    if index == 3:  
        sorted_movies.sort(key=get_money)  
    else:  
        sorted_movies.sort(key=operator.itemgetter(index))  
    if descending:  
        pass  
    sorted_movies.insert(0, header)  
    return sorted_movies
```

Since Python always sorts in ascending order, the `reverse` method will be

used to arrange the data in descending order. Replace `pass` with `sorted_movies.reverse()`.

```
def sort_movie_data(data, index, descending):  
    """Sort movie data based on the column data"""  
    header = data[0]  
    sorted_movies = data[1:]  
    if index == 3:  
        sorted_movies.sort(key=get_money)  
    else:  
        sorted_movies.sort(key=operator.itemgetter(index))  
    if descending:  
        sorted_movies.reverse()  
    sorted_movies.insert(0, header)  
    return sorted_movies
```

Finally, the function call needs to be modified to accept the third parameter.

```
print_movie_data(sort_movie_data(movie_data, 0, True))
```

The output should be sorted by title.

challenge

What happens if you...

- Change the descending parameter to `False`?
- Change the `0` to a different number? The number represents the other columns of data, so it can only be a number between `0` and `4`.

Lab 4

Lab 4 - Command Line Interface

The next step is to build an interface for the user of this program. The interface should run continuously until the user tells the program to quite. Define a function `user_interface` with no parameters. Inside the function, have a `while True` loop. The body of the loop should be `pass`.

```
def user_interface():  
    """Ask user how they want to sort the movie data"""  
    while True:  
        pass
```

Here is the flow of the function `user_interface`:

- * Ask the user by which criteria they want to sort the data
- * If the user entered “6”, quit the program
- * Check to make sure the data entered is valid
- * If not, print a message and start again; the program should not crash
- * Ask the user if they want ascending or descending order
- * Check to make sure the data entered is valid
- * If not, print a message and start again; the program should not crash
- * Print the sorted data
- * Repeat until the user quits

Many of the above tasks will be put into their own functions. You will also need some conditionals to control the flow of the program. The code below is the skeleton of the above tasks.

```

def user_interface():
    """Ask user how they want to sort the movie data"""
    while True:
        column = ask_column()
        if column == "6":
            break
        if sanitize_column(column):
            order = ask_order()
            if sanitize_order(order):
                movie_data = fetch_movie_data(movie_csv)
                print_movie_data(sort_movie_data(movie_data, int(column) - 1, int(order) =

            else:
                print("Enter a number 1 or 2.\n")
        else:
            print("Enter a number 1 to 6.\n")

```



Remove the variable definition of `move_data` and the function call `print_movie_data`. In their place, call `user_interface` instead.

```
user_interface()
```

Here are a couple of things to take note of:

- * `column` is an integer that represents the column in the CSV file by which the data will be sorted.
- * `ask_column` is a function that presents the columns and asks the user to type 1 to 6.
- * `sanitize_column` is a function that returns `True` if the number is between 1 and 6, it returns `False` if not.
- * `order` is an integer (1 or 2) that represents ascending or descending order.
- * `ask_order` is a function that asks the user to type 1 or 2.
- * `sanitize_order` returns `True` if the user typed a 1 or 2, it returns `False` if not.
- * `column` and `order` are user input, which is stored as a string; they must be typecast as ints in order to use them.
- * `column` is a number 1 to 6 (with 6 being the command to quit). The CSV file has columns 0 to 4, so subtract 1 from `column` so it matches the CSV file.
- * `order` is an integer, but the parameter needs to be a boolean. Using a

boolean expression as a parameter ensures either `True` or `False` will be passed to `print_movie_data`.

Note, there is no button to run the code. Right now, the code would generate several errors. Continue to the next page to get the program in a workable state.

Lab 5

Lab 5 - Adding Helper Functions

On the previous page, the following functions were referenced in the program, but have not yet been declared. Create the functions before the `user_interface` function. Set the function bodies to `pass` for now.

```
def ask_column():  
    """Ask the user by which criteria they want to sort the data"""  
    pass  
  
def sanitize_column(column):  
    """Return True if the user entered a valid number, else return False"""  
    pass  
  
def ask_order():  
    """Ask the user how they want the data sorted: ascending or descending"""  
    pass  
  
def sanitize_order(order):  
    """Return True if the user entered a valid number, else return False"""  
    pass
```

The code technically works, but it is not a good idea to run it just yet. Because none of the above functions do anything, your program will be stuck in an infinite loop.

► Why is the program an infinite loop?

The function `ask_column` should ask the user to type in a number 1 to 6, with each number representing a choice. To make this readable, each choice should be on its own line. This is where the triple-quote makes printed formatted text easy. The function should return the value `column`.

```
def ask_column():
    """Ask the user by which criteria they want to sort the data"""
    column = input("""How do you want to sort the movie data? Enter '6' to exit the program

    1) By Film Title
    2) By Genre
    3) By Rotten Tomatoes Score
    4) By Worldwide Gross
    5) By Year
    6) Quit\n""")
    return column
```

The function `sanitize_column` returns `True` if the user typed in a valid choice. It returns `False` if they did not. Two things must be true for `column` to be a valid choice. First, it must be a number. Any data coming from `input` is captured as a string. The string "6" can be typecast as an `int`; the string "cat" cannot. If `column` cannot be typecast as an `int`, then there is a `ValueError`. So this function is going to use `try... except`. A `ValueError` should result in the function returning `False`. The second thing that must be true is that `column` must be greater than or equal to 1 and less than or equal to 6. Return the value of this boolean expression.

```
def sanitize_column(column):
    """Return True if the user entered a valid number, else return False"""
    try:
        int(column)
        return int(column) >= 1 and int(column) <= 5
    except ValueError:
        return False
```

The `ask_order` function is almost identical to the `ask_column` function. Instead, it asks the user to enter a 1 or 2.

```
def ask_order():
    """Ask the user how they want the data sorted: ascending or descending"""
    order = input("""How do you want the movie data ordered?

    1) Ascending Order
    2) Descending Order\n""")
    return order
```

Similarly, `sanitize_order` is almost identical to `sanitize_column`. Instead, it checks to see if `order` is either a 1 or 2.

```
def sanitize_order(order):  
    """Return True if the user entered a valid number, else return False"""  
    try:  
        int(order)  
        return int(order) >= 1 and int(order) <= 2  
    except ValueError:  
        return False
```

Your app should be complete. Click the button below to run it. Enter numbers that produce output and enter text that is incorrect.

► **Solution**

Lab Challenge

Write the function `to_upper` which accepts a string parameter. The function should return string, but in all caps.

The TRY IT button below will test your code with `hello`. If your code returns `HELLO` then you will receive the message that your code passed the test.