Lesson 12. Capstone project: name mashup

After reading lesson 12, you'll be able to

- Write code to solve a programming task
- Read requirements for a program
- Get input from the user for two first and last names, mash them up (combine them in some way), and show the user the result
- Systematically build up code to write program solutions

The problem

This is your first interactive programming task, so let's have some fun with the user! You want to write a program that automatically combines two names given by the user. That's an open-ended problem statement, so let's add a few more details and restrictions:

- Tell the user to give you two names in the format FIRST LAST.
- Show the user two possible new names in the format FIRST LAST.
- The new first name is a combination of the first names given by the user, and the new last name is a combination of the last names given by the user. For example, if the user gives you `Alice Cat` and `Bob Dog`, a possible mashup is `Bolice Dot`.

12.1. UNDERSTANDING THE PROBLEM STATEMENT

The checkpoint exercises you've seen so far have been simple. This is your first complicated program, and you'll have to think about how you'll accomplish the task rather than starting to code right away.

When you encounter a problem statement, you should look for the following:

- A general description of what the program should accomplish
- The inputs you should get from the user, if any
- What the program should output
- The behavior of the program in various situations

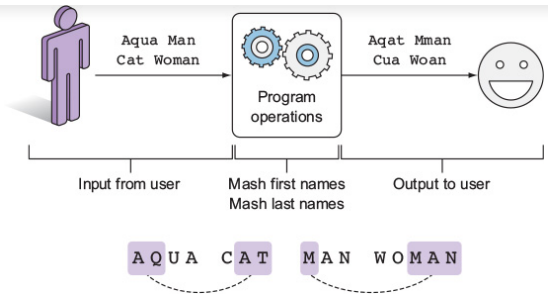
You should first organize your thoughts on the task you're given by using a method that works for you. Ideally, you'll do all three of the following:

- Draw sketches to understand what's being asked
- Come up with a couple of examples that you can use to test your code
- Abstract your drawing and examples into pseudocode

12.1.1. Drawing a sketch of the problem

In this problem, you're asked to get input from the user. The user will give you two names. You'll separate the names into first and last names. Then you'll take the two first names and mash them up. Similarly, you'll take the two last names and mash them up. Finally, you'll present the user with your new first- and last-name mashups. Figure 12.1 shows these three parts of the problem.

Figure 12.1. Drawing the inputs to the program, a sample mashup of the input names, and the output shown to the user



12.1.2. Coming up with a few examples

After you have an idea of the main parts of the program, you should come up with a few examples that you'll be able to use to test your program.

This is an important step. You, as a programmer, get to simulate what the user might input into the program. Users are unpredictable, and one of their favorite pastimes is to try to crash your program.

In this step, you should try to come up with as many different inputs as possible. Think about short names, long names, and combinations of different lengths of first and last names. Are there any unique names? Here are a few sample types of names for testing:

- A first/last name that has two letters (CJ Cool and AJ Bool)
- A first/last name that has many letters (Moonandstarsandspace Knight)
- A first/last name that has an even number of letters (Lego Hurt)
- A first/last name that has an odd number of letters (Sting Bling)
- A first/last name with the same letters (Aaa)
- Two names that are the same (Meg Peg and Meg Peg)

You should stick to examples that don't deviate from what you told the user to input. In this case, you ask the user for a first and last name. You don't make any guarantees about how your program works when the user puts in anything that doesn't match this. For example, a user who inputs `Ari L Mermaid` shouldn't expect the program to work as advertised.

12.1.3. Abstracting the problem into pseudocode

Now you're ready to divide your program into *blocks* of code. In this step, you start writing pseudocode: a mix of English and programming syntax. Each block will tackle a separate step in your program. Each step aims to gather data in a variable for use in a later step. Here are the main steps in this program:

1. Get user input and store it in variables.
2. Split up the full names into first and last names and store them in variables.
3. Decide how you'll split up the names. For example, find halfway points in each first name and last name. Store the first half of each in variables, and the last half of each in variables.
4. Combine the first half of one name with the second half of another name. Repeat for as many combinations as you want of first and last names.

The next few sections discuss each of these steps in detail.

12.2. SPLITTING UP FIRST AND LAST NAMES

You'll notice that the purpose of everything you've done so far has been to try to understand what's being asked in the problem. Coding should be the last step, to reduce the number of errors you might run into.

At this point, you can start to code up the individual blocks of statements. When writing interactive programs, you almost always start with getting input from the user. The following listing shows the lines of code that achieve this.

Listing 12.1. Getting user input

```
print("Welcome to the Mashup Game!")
name1 = input("Enter one full name (FIRST LAST): ")
name2 = input("Enter another full name (FIRST LAST):
```

- ***1*** **Asks the user for input in the desired format**

The user input is now stored in two variables, with appropriate names.

12.2.1. Finding the space between the first and last name

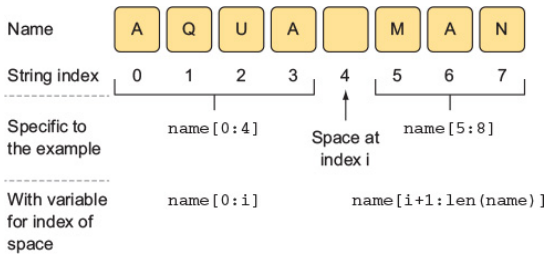
After getting user input, you should split it into first and last names. You'll have to mash up first names together and mash up last names together, so full names stored in one variable aren't helpful. The first step to splitting up the full name is to find the space between the first and last name.

In lesson 7, you learned about various operations you can do with strings. One operation, `find`, can tell you the index location of a particular character. In this case, you're interested in the index of the space character, `" "`.

12.2.2. Using variables to save calculated values

Now you'll save the first and last names into variables to use later. Figure 12.2 shows how to split the full name.

Figure 12.2. Splitting the full name into first and last names by using the index of the space



You first find the index location of the space. Everything from the start of the full name to the location of the space is the first name. Everything from one letter past the space is the last name.

You should store the first and last names so that you can work with them later. Listing 12.2 shows how to do this. You use the `find` operation on strings to get the index location of the space character.

Knowing this location, you can take all letters from the start of the full name (starting from index 0) until the space character index and store that substring as the first name. Recall that indexing into a string with `some_string[a:b]` means that you take all letters from index `a` to index `b - 1`. To get the last name, you start at one character past the `space` character index and take all letters until the end of the full name (up to and including the index of the last character in the string).

Listing 12.2. Storing the first and last names in variables

```
space = name1.find(" ")
name1_first = name1[0:space]
name1_last = name1[space+1:len(name1)]
space = name2.find(" ")
name2_first = name2[0:space]
name2_last = name2[space+1:len(name2)]
```

- ***1*** **Gets and stores the index of the space character, which delineates the first and last name**
- ***2*** **Takes all letters from the start of the full name until the space character to store the first name**
- ***3*** **Takes all letters from one character past the space until the end of the full name to store the last name**
- ***4*** **Repeats for second name**

At this point, you have the two first names and the two last names stored in variables.

12.2.3. Testing what you have so far

With the code in listing 12.2 written up, now is a good time to run the program on a couple of test cases. Testing the code so far means printing the values of the variables you created. You should check to make sure the output is what you expect. If you put in `Aqua Man` and `Cat Woman`, the `print` statements

```
print(name1_first)
print(name1_last)
print(name2_first)
print(name2_last)
```

will print this:

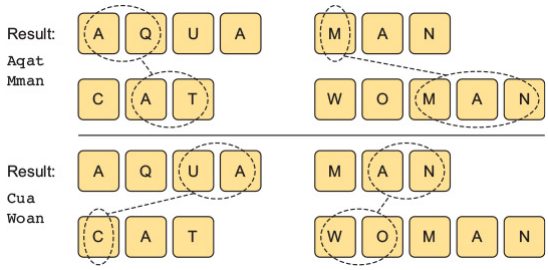
```
Aqua
Man
Cat
Woman
```

This checks out, so you can move on to the next part of the code.

12.3. STORING THE HALVES OF ALL NAMES

With what you know so far, you can't do any fancy letter detection to make the resulting mashup look and sound just right. Instead, you can do something simpler that works well most of the time. Given two first names, take

Figure 12.3. Two ways to mash up Aqua Man and Cat Woman. You take half of each of the first names and combine them. Then you take half of each of the last names and combine them.



12.3.1. Finding the midpoint of names

You want to find the index in the string that's halfway in a name. The user can enter names of any length, and names that have an even or an odd number of characters.

Names with an even number of characters

With names that have an even number of characters, dividing the name in half is easy. You take the length of the word and divide by 2 to give you a whole number for the halfway index point in the string. In figure 12.3, the name *Aqua* is an example of this.

Names with an odd number of characters

What should you do when a name has an odd number of characters? In figure 12.3, Cat and Woman are examples. Dividing an odd number by 2 in Python gives you a floating-point number, such as 3.5 or 5.5. A floating-point number can't be used as an index into the string.

Recall that you can cast a floating-point number to an integer. For example, `int(3.5)` gives you 3. Now, names that have an odd number of letters will have indices that are rounded down for the first half of the name; *Man* in the top part of figure 12.3 is an example. Therefore, names with an odd number of letters will start one index early for the second half of the name: *Woman* in the top part of figure 12.3 is an example.

The code to save halves into variables

Listing 12.3 shows how to store halves of each name. The user entered two full names. You have to find halves of each name, but the same basic process is repeated for each name.

You first find the halfway point in the name. You use the casting function in Python to deal with names that have an odd number of letters. If a name has five letters, the first half will have two letters, and the second half will have three letters. Casting to an int doesn't affect names with an even number of letters because, for example, `int(3.0)` is 3.

Listing 12.3. Storing halves of each name

```
len_name1_first = len(name1_first)
len_name2_first = len(name2_first)
len_name1_last = len(name1_last)
len_name2_last = len(name2_last)
index_name1_first = int(len_name1_first/2)
index_name2_first = int(len_name2_first/2)
index_name1_last = int(len_name1_last/2)
index_name2_last = int(len_name2_last/2)

lefthalf_name1_first = name1_first[0:index_name1_firs
righthalf_name1_first = name1_first[index_name1_first
lefthalf_name2_first = name2_first[0:index_name2_firs
righthalf_name2_first = name2_first[index_name2_first

lefthalf_name1_last = name1_last[0:index_name1_last]
righthalf_name1_last = name1_last[index_name1_last:le
lefthalf_name2_last = name2_last[0:index_name2_last]
righthalf_name2_last = name2_last[index_name2_last:le
```

- ***1*** **Stores lengths of the first and last names extracted from the input**
- ***2*** **Stores halfway indices of each name by casting the halfway point to an integer to round down to get a whole-number index**
- ***3*** **Name from start to halfway**
- ***4*** **Name from halfway to the end**

Now you have all the halves of names stored. The final thing left to do is to combine them.

12.4. COMBINING THE HALVES

step is now simple because you already computed and stored everything necessary.

The code for this is in listing 12.4. In addition to combining the halves, you should also make sure to capitalize the relevant half so that it looks like a name—for example, `Blah` and not `blah`. You can use the `capitalize` operation on the first half to capitalize the first letter only. You use the `lower` operation on the second half to make all of its letters lowercase.

A final thing to note about the code in listing 12.4 is the use of the backslash on some lines. The backslash is used to break up statements in your code over more than one line. If you insert a line break in a line of code, the backslash tells Python to keep reading the next line to find the rest of the statement; without the backslash, you'll get an error when you run the program.

Listing 12.4. Combining the names

```
newname1_first = lefthalf_name1_first.capitalize() +
righthalf_name2_first.lower()
newname1_last = lefthalf_name1_last.capitalize() +  \
righthalf_name2_last.lower()

newname2_first = lefthalf_name2_first.capitalize() +
righthalf_name1_first.lower()
newname2_last = lefthalf_name2_last.capitalize() +  \
righthalf_name1_last.lower()

print("All done! Here are two possibilities, pick the
print(newname1_first, newname1_last)
print(newname2_first, newname2_last)
```

- ***1* Capitalizes the first half string**
- ***2* Ensures that the second half string is all lowercase**
- ***3* Shows the user two possible names**

This code repeats the same thing four times: to get two new first-name combinations and then to get two new last-name combinations. First, you take the left half of the first name from the first user input and use the `capitalize` operation to ensure that the first letter is capitalized and all others are lowercase. Then you take the second half of the first name from the second user input and ensure that all letters are lowercase. The backslash tells Python that the statement spans two lines.

After you combine the halves, the final remaining step is to show the user the results. Use the `print` operation and display the new names. Try playing around with the program and different input names!

SUMMARY

In this lesson, my objective was for you to write a program that asks the user for two names in a specific format. You manipulated the names so that you created variables to hold halves of each first name and each last name. You combined the halves to mash up the input names and then showed the user the results. Here are the main takeaways:

- The user can provide input more than once in your program.
- You can use the `find` operation to find the location of substrings in the user input.
- You saved manipulated strings as variables, and you used the + operator to concatenate strings together.
- You used the `print` operation to show output to the user.