PLTW COMPUTER SCIENCE

Activity 3.1.3

Data Types, Lists, and Elements: Social Media Posts

goals

- Learn to create, manipulate, and access lists in *Python*
- Develop and test using tracer rounds
- Pair Program to develop a program that solves a problem and generates understanding

description of TASKS

- Combo Menu Revisited: Modify the program from the previous activity to create a list for each order placed and store it to be accessed later.
- Social Media Posts: Modify a program that posts messages and gets posts to understand of how social media works.

Essential Questions

1. Why do conditional statements always have only two outcomes? (Yes or No. True or False)
2. Why is computer science considered a form of art and creative expression by many?
3. What are some essential operations you do over and over with lists or collections?

essential Concepts

- Tracer Bullet Development
- Classes and Objects
- Lists and Elements

- Data and Information

Resources

Activity 3.1.3 Student Files

Cloud9 Debugging Guide

Lists and Classes

A **list object** is a collection of data values. As you have seen previously, lists are useful whenever you want to manage several pieces or collections of related data. The individual values can be added, removed, or changed, so each individual entry in a list is called an **element**and has a specific identifying index number, based on its position in the list.

The elements can be any data type**,** such as strings, integers, or floats. At this point when you read these example classes, you might even begin to connect examples of the same concept that is shown in different ways, such as int = integer = 9, but not 9.5. These data types individually are examples of built-in **classes**. Python identifies them as *str*, *int*, and *float*, but no matter the programming language, they still have the same basic class properties.

Classes describe what types of data and functions are to be included in all objects of that type, whereas objects are specific instances or occurrences of a class. In other words, objects are created from a class definition, like a house is created from an architectural design or blueprint. For example, consider the *float* class. It creates float objects. When you think 9.5, the object 9.5 does not match up with the class properties of an integer (it has a decimal). It does match the *float* class properties, therefore, the 9.5 object is an example of a *float* class. The terms “class” and “data type” are often used interchangeably in *Python*, because each data type is a built-in class.

In general, a class groups functions and associated data into a cohesive unit. In other words, classes specify the functions, attributes, and properties that are related in some way. In MIT App Inventor, classes formed underlying objects for many of the tools you used, from buttons to Web APIs or Sprites. The difference is that in previous activities in this course, you used objects that were already created. For example, you did not have to define a button class to specify how it looked or behaved; you just created a button object and gave its text property some value like "Click me!"

Using an object that has already been created for you is another example of **abstraction**. It lets you focus on what you need, the button and its text, without worrying about low level details, like setting the shape of the button or a shadow effect. As you progress in this course, it is important to start thinking about these lower levels of abstraction.

Creating a User-defined Class

In this activity you’ll perform some basic operations on built-in data types like integers and strings, and then you will manipulate some lists. The lists will contain a new class of objects called **posts**, which represent the basic data contained in a social media post.

Finally, you’ll move on to create a tracer round of your own design that uses lists, as you did in the

last activity. Remember, a tracer program is a program that implements only the most basic and necessary structures used as a proof of concept to show that something difficult (high- or low-level) can be done, or to follow one particular path (again high- or low-level detail) through a larger application.

Lists in Python

Lists in *Python* are not limited in length; they can be as long as they need to be. They can also contain as many different data types as needed. This allows programmers, like you, great flexibility. How you handle lists in your program can impact the program's performance.

The data contained in a list are called elements, and the positions of an element within the list is called its index. In *Python*, the first index in a list is 0.

1. Navigate to Cloud9 and open a workspace as directed by your teacher.
2. Create a python (*.py*) file in Cloud9. Be sure to use a name that will help you find it quickly, if you want to review these later. Your teacher may have a specific naming scheme to follow.
3. Enter the following lists into a python file:

```
world = ['w','o','r','l','d']
integers = [-3,-2,-1,0,1,2,3]
floats = [-5.0, 0.5]
multi_type = ["hello", 2, "the", world]
empty = []
```

4. Take a moment to predict what you think each line will look like when you run the program.
5. Add print statements to the code so you can see the list contents that will be returned. Were your guesses correct?

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

6. Nest a *type ()* function in the world list to identify what data type 'w' is.

```
variable = [type('a_list_element'), 'list_element']
```

7. The *multi_type* list should return a message. What does it say? Hint: The message did not

change when you added the nested *type ()* function.

Important: For additional examples of lists, go to **Interactive Python**.

Accessing Elements in a List

Each data element in your list has a specific position. That position is its index. The index comes from the order they fall in the list. To find data you need, retrieve specific elements by their index. In *Python*, the first index—the location of the first element—in a list is at index 0.

To retrieve data from a list, you use the square brackets and the index number. For example, using the previous list:

```
multi_type = ["hello", 2,"the", world]
```

Important: The element at *multi_type[0]* is the string "hello" and the element at *multi_type[1]* is the integer 2.

8. Enter the following examples into the code editor window to see some examples of how you can use indexing in a list to retrieve specific elements.

   ```
   food = ["Burger", "Shake", "Fries"]
   print food[1] + " at index 1"
   print "Is Burger in the list?:"
   print "Burger" in food
   ```

9. Based on the outputs, add additional code that shows which element is at index 0 and which element is at index 2.
10. Add additional code that checks whether “Shake”, “Fries”, and “Burger” are in the food list and returns a True or False response.
11. Add code to check that Pizza is not in the list.
12. Save your *Python* file so you can reference this information in the future. You may rename the file to help identify that it contains information about lists.
13. You may also want to add some comments to help identify what you have learned about lists. Such as:

    ```
    multi_type = [“hello”, 2, “the”, world] #The message is fixed!
    ```

Boolean Operators and Lists

You may use Boolean operators to check what is included or not included in a list. Based on what you have learned in other units, you could try something like this:

```
print "Is Pizza in the food list?:" + ("Pizza" in food)
```

However, that code will not work in *Python*. If you tried to add the line of code and run the program, you get back an error message like this:

```
TypeError: cannot concatenate 'str' and 'bool' objects
```

In *Python*, you cannot concatenate these two different data types, *string* and B*oolean*.

Important: Reading error messages will help you know what is wrong, and in which line, so you can correct your code. Reading errors in text-based languages is an important practice.

14. The following approach would work in *Python*. Comment the code to describe what is happening in each line.

```
in_list = "Pizza" in food
print "Is Pizza in the list?: " + str(in_list)
```

15. Use this approach to concatenate the strings so that each question and answer are displayed on the same line. You will create one line for “Burger”, one line for “Shake”, and one for “Fries”.

```
Is Pizza in the list?: False
Is Burger in the list?: True
Is Shake in the list?: True
Is Fries in the list?: True
```

Sample of a correct output

16. There are many other things you can do with lists. Take some time to research other functions and ways you are able to manipulate lists. Use Chapter 10 in the Runestone Interactive Textbook and an internet search to help with this research.

PLTW DEVELOPER’S JOURNAL Record some list functions that you are interested in or that you think you might use later.

Now you are going to implement some parts of a program to allow a user to manipulate posts on a social media site. While this code is not connected to any such site currently, the code that you will write to complete this program could be copied directly into a *Python* web project and used for that purpose.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Iterating on the Combo Menu

These files should show in your own Cloud9 workspace. Duplicating and renaming the files will break the links in them. If you corrupt your file, you can always download and upload them again. In a previous activity, you used variables and conditionals to make decisions about outcomes based on what food and drink combinations were ordered.

17. It is possible to change the file name listed after the "from" to access a different file if you

choose to save the file with a new name. You need to be careful and check for all import statements. You also need to check for errors with the imports when you try to run your code to know if the correct file is being accessed.

- Why would you want to use lists?
- Can you use lists to improve a program?

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

You will now modify your previous program to include lists. This approach will allow you create a new list for each order and store that list for use in the future, to know when to apply discounts, and print the order for the customer to check.

18. In your Cloud9 workspace where you can see all your files:

    - Right-click and duplicate your CSE_312_combo_menu file.
    - Rename it “CSE_313_combo_menu_list.py” .

      You need to practice file management as you move into the next development iteration; naming files in this way will you help identify the iteration of the files without losing previous work.

19. Modify the last combo menu activity in this duplicated and renamed file to include a list to store and manage data. There are different ways to use lists to accomplish this task. Review the two options below and what you would need to do:

    Option 1: Create a List with Placeholders to Define the Order and Data Types Expected

    Remember, each data element in your list has a specific position. That position is their index number. The index number comes from the order they fall in the list. However, each data element in the list could actually be a different data type. For example, the information at index 0 could be a string like “chicken” or “beef”, or it could be an integer like 2 or 3 to describe the number of ketchup packets at index 3.

    By using strings and variables in the actual list, this option will provide greater flexibility down the road.

    1. Set up the list to expect variables, strings, and integers in the list; create an index spot for each order item to become an element in the list.
    2. Create a list using empty strings for each future sandwich, beverage, and fries choice. The empty string tells the program to expect a string to be placed into the list at that index position.
    3. At index 3, however, place a 0 to indicate that an integer is expected (number of ketchup packets). Some starter code you might add to looks like this:

    ```
    order = ["","","",0 ]
    sandwich_index = 0
    ```

```
beverage_index = 1
fries_index = 2
ketchup_index = 3
```

Using the example code provided, you would need to outline your list and add code to add items at specific indices.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Option 2: Create and Append To an Empty List

In the previous exploration of functions, methods, and list manipulations, you may have come across the *.append()* method. Rather than creating an index spot and defining what data type to expect, you create an empty list at the beginning. Each time the user makes a choice, the *.append()* method adds whatever the user entered to the list.

```
order = [ ]  #Setting up the list
.append()    #a function that adds items to a list
```

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

Using the code sample, you would need to define an empty list and use the *.append()* method to add items to the list.

```
order = [ ]
order.append()
```

20. In the 3.1.3 python file that you created (the copy of 3.1.2 combo menu), add to the combo menu to have an ongoing list that is added to every time the program runs. Review the design requirements from Activity 3.1.2 to make sure the program does not lose any of those design specifications.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

21. Decide whether you are going to use Option 1 or Option 2 to add a list to your combo menu.
22. Modify the combo menu to collect the user's order in a list and then print the list at the end of the program after the order is complete.
23. Modify your conditionals to provide an appropriate response if the user tries to order something not on the menu.

Skeleton Code

In general, **skeleton code** refers to a program that has many missing pieces to be filled in. Often times skeleton code will contain comments to inform the programmer what kind of code needs to go where.

24. Download the **source files**. The following two are skeleton files:
    - CSE_313_post.py
    - CSE_313_skeleton.py

Now you are going to implement some parts of a program to allow a user to manipulate posts on a social media site. While this code is not connected to any such site currently, the code that you will write to complete this program could be copied directly into a *Python* web project and used for that purpose.

25. Form pairs for pair programming as directed by your teacher. Decide:
    - Who will create and share the workspace for this activity.
    - What the file name will be. It should allow you both to recognize the contents of the file, and be able to find the file again later for review.
    - Who will drive/navigate first.
26. In Cloud9, click **File** in the menu bar and select **Upload Local Files**.
27. Then select the *CSE_313_post.py* and *CSE_313_skeleton.py* files.

These files should show in your own Cloud9 workspace. Duplicating and renaming the files will break the links in them. If you corrupt your file, you can always download and upload them again.

Note: It is possible to change the file name listed after the *from* and access a different file. You will need to save the *CSE_313_post* file with a new name and put the new name in place of the *CSE_313_post*.

You will need to be careful and check for all *import* statements to pull the correct class from the correct file. When you try to run your code, you will also need to check for errors with the imports to know whether the correct file is being accessed.

```
#This line of code tells the Python interpreter that it needs to reference the
#post.py file in order to run the rest of the code in this file.
from CSE_313_post import Post
```

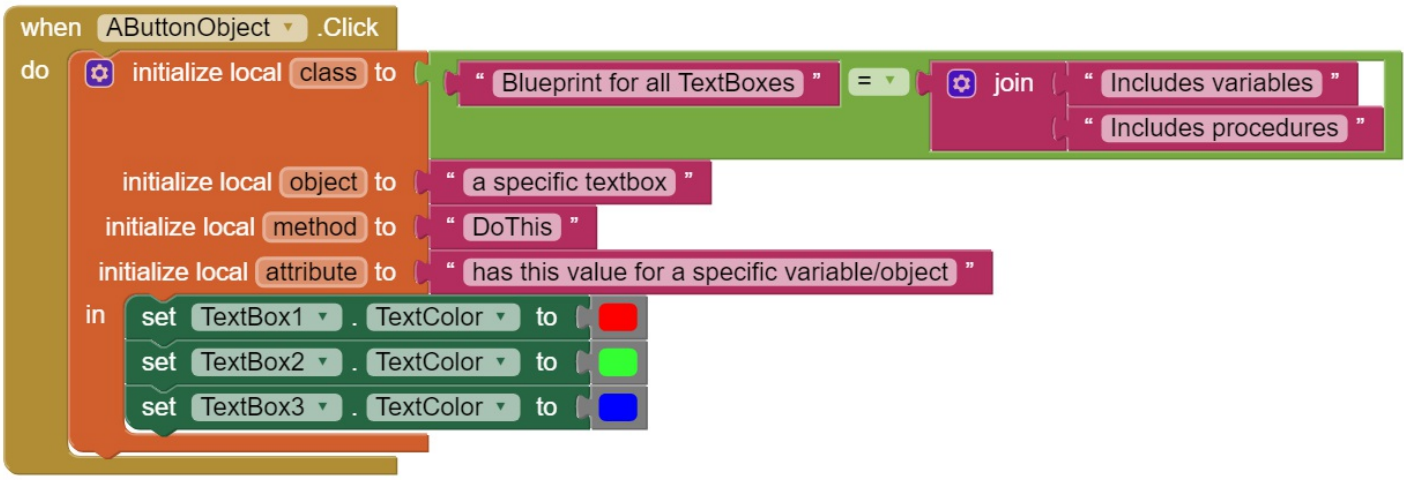
Class and Objects

The *CSE_313_post.py* and *CSE_313_skeleton.py* files use **Object-Oriented Programing (OOP)**, which focuses on objects that contain specific data and functions together. A **class** outlines the properties of that object. In turn, the **object** is an instance or example of data that fit the properties of a class.

For example, some of the tools used in App Inventor—like buttons, text boxes, list view, list picker,

and web viewer—are actually examples of classes. You have also used some built-in classes defined in *Python*, such as *str*, *int*, and *float*. Data types are examples of class types, because they define the properties of what is considered a string, float, or integer.

An object is created anytime things like textboxes are used in a program. They are constrained to the properties defined in the class, but individual textbox objects can have their own attributes, such as color, text, and placement.



An example of MIT App Inventor object-oriented programming that is mostly handled by programming abstraction

Another way to consider the object-oriented concept outside of the computer science world is with building a house:

- The class is the blueprint for the house. Many houses may be built using the same blueprint.
- The object is any house built from that template.

The Post Class

28. Examine the contents of the *CSE_313_post.py* file, which defines the Post class.

    You have yet to see the syntax for a class definition and its functions in *Python*. For now, just know that any line that begins with "def" and is followed by an indented block of code is a function.

    ```
    def function_name(arguments_passed_into_function):
         what the function will do when it is called
    ```

    PLTW DEVELOPER'S JOURNAL Summarize the code contained in the file. Practice using abstraction to your advantage by just focusing on the comments and ignoring implementation details.

You do not need to modify the *CSE_313_post.py* file to complete this activity. However,

understanding what the file is doing is important, because the file you will modify accesses information from the Post class.

29. Read the comments in the *CSE_313_skeleton.py* file. This is the only file that you will modify, so it is important to have an understanding of what the code is doing before you begin an iterative process of change.

PLTW DEVELOPER'S JOURNAL Record a summary of what you need to do to complete the program. Use the comments to help outline what needs to be done, and start looking at your notes for how to do that in *Python*.

30. Follow the comments in the code to make the file work. To help you fill in the tracer round skeleton code with functional code, here are some things to know and apply:
    - You will create Post objects and modify them using the class definition in *post.py*.

      Note: "variable_name" is a placeholder for a variable that you need to create.

    - To create a new Post object, use the following syntax: *variable_name = Post("username", "message")* .
    - To retrieve the user name associated with a given post, call the *get_user_name method: variable_name.get_user_name()* .
    - To remove a list item, use:
      - *del variable_name[index]*
      - Optional exploration: To check whether the index number is valid, use the *len()* function.
    - You may call a function inside another function call, such as nesting a call to *raw_input* inside a different function call, as in:

      ```
      (int(raw_input(…))
      ```

    - Review functions, methods, and conditionals as presented in this activity and previous activities.
    - You may need to add a few items to the post before you use the other features like *remove*.
    - When the user enters "quit", do not print anything out and just let the program end.
    - If in doubt, start by reading through the comments in the provided code, and then come back to this list.
31. Sometimes coding errors cause an error that can impact the terminal that you are testing your code in. Occasionally, it might be helpful to do some of the following steps:
    - In the terminal, right-click the menu option, then select **Clear Buffer**.
    - In the terminal, right-click **Restart All Terminal Sessions**.
    - Save and restart your Cloud9 window in the web browser.
    - Review the Cloud9 Debugging Guide.

      Cloud9 Debugging Guide

32. When you have completed your code, let your teacher know.

Conclusion

1. Describe a metaphor that highlights the relationship between the concepts of classes and objects.
2. What ethical consideration arise from the requirements for this program?
3. How did you interpret and respond to the **essential questions**? Capture your thoughts for future conversations.

Cloud9 Debugging Guide

1. 

   Check first that the file has a *.py* ending.

2. 

   Read any error messages and start with the first line that has an error.

3. 

   If no errors exist, verify that there is some form of print or return statement that would display to the user the end result.

4. 

   Add comments (# or ''' ''') to make some of the code inactive. This is called “commenting out” code and can help you isolate problems.

5.

Use a **Python visualizer** to see how the code is executing.

6.

If the thought is that Cloud9 is “broken” or “not working”, then stop running the program, make sure everything is saved, and refresh the screen before you try to run the program again.