PLTW COMPUTER SCIENCE

Activity 3.1.2

Variables and Conditionals: Combo Menu

goals

- Apply coding fundamentals in a text-based language
- Apply file naming conventions and version control
- Develop and test code incrementally
- Develop a program independently

description of TaskS

- Coding Fundamentals in Python: Create examples that build understanding of what coding concepts look like in *Python*
- Combo Menu: Create a program that automates the ordering of a meal by offering options in the menu, capturing the responses, and providing a summary of the order to the user

Essential Questions

1. Can I describe what an algorithm does to someone new to coding?
2. What are some mathematical and logical concepts that are used over and over?
3. What computer science terms keep confusing me?

essential Concepts

- Algorithms, Variables, Arguments, Procedures, Strings and Concatenation, and Logic
- Arithmetic Operators, Relational Operators, and Logical Operators, Conditionals and Event-driven Programming

Resources

Tracer Rounds

New Coding Language, Same Concepts

Your first steps in *Python* will be to collect input from a user and convert that input into a form that your program can use to make decisions.

To begin to learn syntax, you will start out experimenting with conditionals and functions. By the end of this activity, you will create a program that accepts orders off a menu from a user.

Note: As you work through this activity, remember to take note of new terms and concepts introduced within the procedure steps, as well as in the Runestone Interactive Textbook.

PLTW DEVELOPER'S JOURNAL In previous units, you were introduced to many types of conditionals. Take a moment to describe in your own words how you've used each of the following in this course:

- Conditional statement
- Boolean value
- Boolean expression
- Logical operator
- Chained conditionals
- Nesting
- Branching

Learning the Python Language

Much of the vocabulary you have learned to date now needs to be redefined for the *Python* language.

- Functions are blocks of organized, reusable code that perform a single action or related set of actions. *Python* has many built in functions making it easier to use. These are like the blocks in MIT App Inventor.
- Variables are reserved memory locations to store values.
- Values can be a number or word.
- Values are objects. In *Python*, everything is an object.
- In *Python*, objects can be classified as classes or data types.
- The words “classes”, “types”, and “data types” are often used interchangeably. You may learn

more about the relationship between classes and types if you continue your study of CS.
- Some examples of types or classes are integers, floating point numbers, and strings.
- Strings in *Python* can be enclosed in single, double, or triple quotes.

As you have learned throughout this course, there are many terms that coding professionals use interchangeably, which makes learning to program hard sometimes. With use over time, and more and more examples in things like your TEMP charts, you will begin to understand the language of coding professionals. Don't get frustrated if it doesn't all make sense at first. Remain patient, as over time, you will gain better understanding.

PLTW DEVELOPER'S JOURNAL Take a minute to look back at your first TEMP entries. Compare your thoughts and feelings when you see them now, versus when you saw them at the start of this course.

Python Variable Types and Functions in the Interpreter

Just like pre-built blocks in App Inventor, *Python* has pre-built **functions**, which allow programmers to use code without having to type all of it in the *interpreter*.

Python Functions

print()	In the *Python* terminal where you write code, anything after *print()* will show up in the *interpreter* when you are in the program.
type()	This function looks at what is inside the parentheses and returns what type of object or data type it is.
abs()	This function provides the mathematical absolute value of whatever numerical value is contained in the parentheses.

As described in previous units, a **variable**can be many different types of **data**.

The **type**of a variable refers to what kind of data is stored in that variable. In App Inventor, you didn't have to be overly concerned with what the type of a variable was—blocks either fit together or they didn't. In *Python*, code may be **syntactically**correct (formatted correctly using the conventions of the *Python* programming language), yet still produce unexpected results, if the variables you are using do not contain the type of data that you think they do.

There are two ways to execute *Python* code in Cloud9:

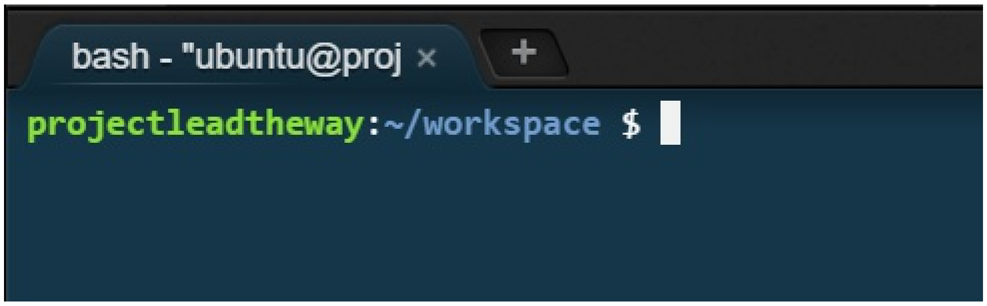
- You can create a *Python* file, type all the code in it, and then select the Run button.
- You can execute one statement at a time in the interactive *interpreter*.

In this activity, you will work with four types of data. Since you don't know any *Python* functions yet, you will use the *interpreter* to help you.

1. Log in to your Cloud9 account.

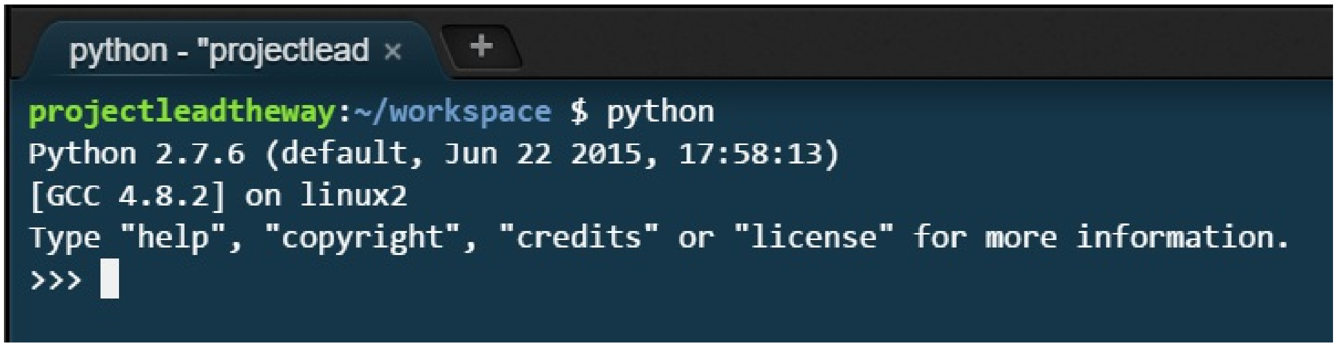
- Follow your teachers directions to setup a directory structure in a workspace that is shared with with your teacher.

2. Select the Cloud9 tab with *bash* in the title.
3. In the bash tab, type “python” after the dollar sign on the command line to start the *interpreter*.



```
projectleadtheway:~/workspace $ 
```

This will let you run *Python* scripts in the terminal.
When you press **Enter**, the program responds as follows:



```
projectleadtheway:~/workspace $ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

4. In the *interpreter*, enter the following statement after the >>> prompt.
5. `type("Hello World!")`

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

6. In the interactive *interpreter*, use the type function for each of the following statements and notice the data type the *interpreter* identifies. You should get the same data types as shown below.

Statement	Data Type Returned
1.8	float

"King Henry VIII"	str (string)
False	bool (Boolean)
5	int (integer)

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

Python Variable Types and Functions

This time, you will type the code into a text file and run it. You need to use the print statement so that the data type returned will show up in the terminalwindow. Just like before, you will see the output from your print statements in the terminal. The only difference is you are writing your code ahead of time rather than entering functions in the *interpreter*.

7. In your Lesson 3.1 directory folder in your Workspace on Cloud9, create a new *.py* file.

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

8. Can you guess what each data type is?

```
print(type("1"))
print(type(-10))
print(type(False))
print(type("False"))
print(type(3.14))
```

9. Type in the code above and run your program in the terminal to see whether you were right.

Converting Data Types

Sometimes when you are programming, you will want to change one data type into a different type. For example, what if you are trying to add an integer like the number 10 and a floating-point like the number .7? What type will the *Python interpreter* say is the answer?

10. Test the following in the interactive *interpreter*.

```
10 + .7
```

PLTW DEVELOPER'S JOURNAL

1. Is the output answer *int* or *float* data type?
2. What value does it return?

The numbers you have been entering into the type function are sometimes called **arguments**. Arguments are the values that are provided to a program. The numbers that are "returned" are called **return values**. To make sure you get the returned value output you want, you can convert one of the data types to match the other.

The four functions below convert to the data types you identified earlier in *Python*. These functions change the value inside the parentheses into the data type of the function.

Python Functions

int()	A *Python* function that converts the value to an **integer**.
str()	A *Python* function that converts the value to a **string**.
bool()	A *Python* function that converts the value to a **Boolean value**.
float()	A *Python* function that converts the value to a **float-point**.

11. Using these functions, enter code in the *interpreter* that will:
    - Change the floating-point number 3.14159 into an integer.
    - Change the integer 8675309 into a string.
    - Change the integer 0 into a Boolean value.
    - Change the integer 1 into a Boolean value.
    - Make sure the answer returned from 10 + .7 is a floating point number.
    - Make sure the answer returned from 10 + .7 is an integer.

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

12. Review a complete list of **all the functions built in to the *Python* programming language**. Explore what you can do based on the kinds of programs you have already written in this course.
13. Identify functions you might want to use later.

PLTW DEVELOPER'S JOURNAL Create a chart similar to the one below. Start with the four functions you learned so far (*int*, *str*, *bool*, *float*). Provide examples that will help you remember what they do. Remember, arguments are what you enter, and return values are the outputs the function produces based on those arguments.

Function	Description	Arguments	Return Value
type()	identifies the data type of what is in the ()	-10	int
int()			
str()			
bool()			
float()			

14. To check your understanding of example arguments and return values, visit Chapter 2.3 in the Runestone Interactive Textbook.

Variable Names and Assignment Statements in Python

At this point in the course, you should be accustomed to creating meaningful variable names to store values. Storing values is critical in any sufficiently complex program.

In *Python*, the syntax for variable name assignment is:

```
variable_name = value or expression
```

1. Is the equal sign an assignment operator or equality operator?

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

2. What do you predict the output of the following program would be?

```
mystery_value = type("Hello World!")
print mystery_value
```

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

In *Python*, the naming convention standard tends toward underscore_case, as opposed to the CamelCase that you saw in MIT App Inventor. This means that instead of camelcase, you are now going to see separate words within variable names appear in all lowercase with underscores to separate them. There are a few exceptions to the lowercase part of this convention that will be explained later. In general, follow the conventions of the language, or the conventions of the original author of the code you are working with.

15. Type and execute the program to verify the answer.

```
mystery_value = type("Hello World!")
print mystery_value
```

16.

PLTW DEVELOPER'S JOURNAL Explain in your own words what happens in this program.

17. Type the code below into a python file. Before you run the program, can you predict what the outputs will be?

```
student_count = int (21.7)
introduction_question = "What is your name?"
truth = (0+1+0)

print(student_count)
print(introduction_question)
print(bool(truth))
```

PLTW DEVELOPER'S JOURNAL

1. Why might it make sense to report the number of students in a class as an integer and not a floating-point?
2. Can you explain the return value you got for 0+1+0 ?

18. To check your understanding of value assignments, visit Chapter 2.4 in the Runestone Interactive Textbook.

The Input Function

There are many ways to gather user data in *Python*; the most straightforward way is through the command line. To collect typed user input at the command line, your program only needs to call the *raw_input* function. This could be useful if you want to prompt a user to enter "continue" or "quit" to determine whether they want to keep playing a game.

raw_input()	**The function in *Python* version 2.7 that gets user input. You will use this function in Cloud9.**
input()	The function in *Python* version 3.6 that gets user input. The online textbook uses this function.

19. In the interactive interpreter, type the following code. Can you guess what the code will do before you run it? Verify your results in the terminal:

```
x = raw_input("What is your name?:")
print "Hello " + x
```

20. What happens if you try the same code with just input()?

```
x = input("What is your name?:")
print "Hello " + x
```

Important: *raw_input* treats any user input as a string, whereas *input* will not auto-convert to strings.

21. Modify the code above so that it works.

Nesting Functions in Python

Nesting means to put one thing inside of a similar thing.

22. Try out the following code in the *interpreter*.

Reminder: Copy/paste will only bring you frustrating errors. Always type the code.

```
type(raw_input("type in a name"))
type(raw_input("type in a number"))
type(raw_input("Enter anything:"))
```

The functions above are nested. The statement contains a call to both the *type()* and *raw_input()* functions, one inside the parentheses of the other. As in mathematics, operations inside parentheses take place first. This means that first the user is prompted with “Enter anything:”, then the user’s value is returned and passed into the *type()* function. The result of that call to the type function will be *str()* , because *raw_input()* interprets any user input as a string.

PLTW DEVELOPER’S JOURNAL

1. What do nested functions do?
2. How do nesting functions behave?
3. Which function executes first, and which last?

Boolean Conditionals and Logical Operators in Python

You have seen and used a number of conditional statements in other programming environments.

- The simplest form of a conditional statement or selection statement is the *if* statement. The *if* statement is a binary selection, because it only has two possible outputs.
- We have also used *if else* statements to make decisions in our programs throughout this course.
- In *Python*, the syntax for an *if else* statement might look like this:

```
x = raw_input("pick a number:")
if x < 10:
    print "too low"
else:
    print "too high"
```

23. If you try the code as written, it will not work. Based on what you learned about *int()* and *raw_input()* , fix the program so that it provides the correct outputs.

24. You can also use an *else if* conditional, which in *Python* is typed “elif”. Using specific variable naming and an *elif*, the code might look like this:

```
x = input("pick a number:")
if x < 10:
    print"too low"
elif x == 10:
    print "Exactly 10!"
else:
    print"too high"
```

    - What is the above code doing?

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

25. Read and work through the following examples in the Runestone Interactive Textbook:
    - Chapter 7.1 Boolean Values and Boolean Expressions
    - Chapter 7.2 Logical Operators
    - Chapter 7.3 Precedence of Operators

    Important: In *Python*, the Boolean data type only has two possible conditions: True or False. Note the capitalization. In *Python*, Boolean values must start with a capital letter, or the program will not recognize them as keywords.

PLTW DEVELOPER'S JOURNAL

1. Make notes of any key differences you notice between how *Python* handles Boolean values and operators and how MIT App Inventor handles them.
2. Examine the following table that contains six common logical operators. Determine whether each statement should be True or False. Note that in *Python*, operators must be written exactly as they are shown below. "<= " is not the same as "=<". Enter these statements and see whether you were right. In your journal, describe the operator in each statement.

Example Comparison Operator Code	Boolean Return Value (True or False)	How would I say it in Math?
print(3==3)		
print(3!= 3)		
print(5>6)		
print(5<6)		
print(5>=6)		
print(5<=6)		

The binary number system (where only 0s and 1s can be used) and the Boolean values of "True" and "False" can be thought of as numbers.

```
True == 1
False == 0
```

So if you were to say, "False + True == False", would you be correct? This would be the same as saying "0 + 1 = 0".

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

Commenting Python Code

As you work through the rest of this unit of study, make sure to leave appropriate comments in your code. This will make it much easier for others to help you if you get stuck and to refresh your memory on what you have done after a school break or between classes.

In *Python*, any line beginning with the # character is a comment, as well as any lines that fall between a matching pair of three double quotes (also known as “triple quotes” to *Python* programmers). Examples are shown below :

```
# This is a single line comment

""" This
Comment Spans
Multiple Lines """


print("If a string is too long \
adding the \
at the end allows you to hit enter \
and continue the string. \
It will all print together in one line.")
```

Iteration Through Tracer Programs or Tracer Bullet Development

You will now create a tracer program that allows a user to select some options from a combo menu at a restaurant. A tracer program is a highly abstracted program that implements only the most basic and necessary structures without including implementation of more specific details. Abstraction helps programmers focus on a particular level of coding, so as to avoid extra complexity and only focus on the immediately relevant parts of a problem. Using *Python* creates an abstracted version of giving the computer directions in base 2, but not as abstracted as a user who just provides an input and waits for an output. Read more about tracer rounds.

Tracer Rounds

Combo Menu Practice

26. Using what you’ve learned about *Python* so far, create a program named “combo_menu”.

    Important: How are you going to store this value and future values?

    The Combo Menu program should behave as follows:

    - ☐ Ask the user for a type of sandwich (chicken $5.25, beef $6.25, tofu $5.75).
    - ☐ Ask the user whether they would like a beverage (yes, no).

- ☐ If they say yes, ask what size beverage they would like (small $1.00, medium $1.75, large $2.25).
- ☐ Ask the user whether they would like french fries (yes, no).
- ☐ If they say yes, ask what size french fries they would like (small $1.00, medium $1.50, large $2.00).
- ☐ If they say “small”, ask the user whether they’d like to Mega-Size their fries (yes, no), and if they say yes, give them large fries at the large fries price instead of their small fries.
- ☐ Ask the user how many ketchup packets they would like (enter a positive integer; cost is $0.25 per packet).
- ☐ If the user selected a sandwich, french fries, and a beverage, reduce the total cost of the order by $1.00.
- ☐ When the user is done entering input, the program informs them of their menu selections, for only the items they ordered and the total cost of the order.

Important: Use variables and Boolean logic to help with the dollar discount. What would the default values be? When would they change? What conditions should the final part of the app check for?

27. Save your code.
28. Notify your teacher when your program is done.

Challenge

1. Read and work through the examples found at the following links:
   - Chapter 7.4 Conditional Execution Binary Selection
   - Chapter 7.5 Omitting the Else Clause
   - Chapter 7.6 Nested Conditionals
   - Chapter 7.7 Chained Conditionals
2. Complete the “Check your understanding” questions.
3. Make notes in your PLTW Developer’s Journal of any key differences you notice between how *Python* handles conditionals and how App Inventor handles them.

Conclusion

1. Where have you seen data types and conditionals before?
2. Why is *Python* considered to have a high level of abstraction even though you don’t see any blocks?
3. How did you interpret and respond to the essential questions? Capture your thoughts for future conversations.

Tracer Rounds/Bullets

What Are Tracer Rounds?

Tracer rounds are highly abstracted programs that implement only the most basic and necessary structures or proof of concept without including implementation of more specific or peripheral details.

Programmers will develop these tracer round programs quickly to identify if they can build out an entire program as they predicted, or if they need to change some of the structures of the program before developing it out completely.

This kind of development could take two approaches. This particular example using the combo menu has you developing the logical structure of flow of control throughout the program without concerning yourself with any of the purely functional elements such as user interface, error handling, connecting to a database, or data analysis. This could be thought of as a horizontal integration. A second approach would be to connect all moving parts of the finished product previously mentioned together as a proof of concept while leaving the details of logical structure and control flow for a later time. This might be thought of as a vertical integration.

In short, tracer rounds lay the groundwork for future development.

Why Are They Used?

- Sometimes, to express the goal of a program and test whether it will work, a simple version is created first. A tracer round, or tracer bullet development approach, helps to check the basics of a program before developing it fully.
- Tracer rounds can help you develop the flow and outline for your programs to save you time before you over-develop a part that may need to be changed when a different part of the program is developed.
- Tracer rounds can simplify debugging to focus on the key aspects of a program first before hours of work are invested in a program that will not meet the development goals.

When Are They Used?

Skeleton code is a program that has many missing pieces to be filled in. Often skeleton code will contain comments to inform the programmer as to what kind of code needs to go where, allowing others to turn the skeleton code into their own tracer round by modifying it to meet their own needs.