A1: Breaking Bad

- Assignment A1 should be done alone or with a partner.
- To begin, go to “File” and Select “Make a Copy...”
- You should seek help completing assignment A1 from the TAs at the evening lab.

Learning Objectives:

- Continue to explore the PyCharm IDE
- Practice editing, building, and running Python programs.
- Learn to interpret error messages from the PyCharm IDE in order to help to debug your computer program.
- Learn the rules for naming identifiers in Python
- Learn to keep an error log
- Write your first program

If you are working with a partner, please indicate both of your names below. If you work in a pair, you must work together the whole time.

Partner 1:	
Partner 2:	

Be sure to read chapters 1 and 2 of the text before beginning this assignment. You’ll find them very helpful, and you need to read them for Friday’s quiz anyways.

Background and Terminology

To create a computer program, one needs an editor to create the **source code** of the program. The source code contains the sequence of steps the programmer designed for the program (the **algorithm**), written in a specific language. The computer cannot yet execute this code at this stage because it is written in a high-level language designed for human readability, and it must be translated for the computer to understand it. This translation happens via the **IDE** or an **interpreter**.

Rarely do programs work the first time they are written. We must have some way to check what happens when our program fails to work correctly, due to a programming error called a **bug**. According to folklore, the first computer "bug" was an actual moth discovered by Grace Hopper in 1947 that was trapped between two electrical relays of the Mark II Aiken Relay Calculator and caused the machine to **crash**. A "bug" is a term used to describe a situation where a program, for a variety of reasons, does not do what it is expected to do. Some bugs are found by the IDE, others require a utility program called a **debugger** to find and correct these mistakes, and some must be found by you, the programmer. Debuggers enable programmers to stop running

their program at any point and examine the contents of any of the **variables** to ensure that their values are what they are expected to be.

Define each of the bold words in the text above, in your own words: ● Source code ● Algorithm ● IDE ● Interpreter ● Bug ● Crash ● Debugger ● Variables If you need to look up a word, feel free to do so, but cite your sources (even if it's the course textbook)!	1. Source Code: The basis of a program that allows it to run 2. Algorithm: a list of steps that can be taken to achieve a result. 3. IDE: A program can be used to develop software 4. Interpreter: Someone or something that translates information in a way in which you can understand it. 5. Bug: Anything that causes a problem in a program 6. Crash: This is what happens when there are too many problems and a program either closes suddenly or stops working altogether 7. Debugger: A program that analyzes and fixes bugs in programs 8. Variable: Something that when applied can change the outcome of something else

Instructions

1. Download the following file: t0_chocolate.py. You may recognize it from T0.
2. Open the file in PyCharm. Refer back to the T0 instructions if you need help remembering where to place the file so it is part of a PyCharm project and how to open them.
3. Run the program a few times, taking note of the interactions that occur in the Python Console.

Now, let's create some bugs.

Fundamentals of Programming Errors (bugs)

There are many kinds of programming errors or bugs:

- A **syntax error** is an error in the grammar, structure, or order of the elements in the source code. Syntax errors are typically detected by the IDE and flagged with a red squiggly underline in PyCharm. Normally, your program will not run at all with a syntax error in the source code.
- **Run-time errors** are generally detected by the IDE, but only when the user attempts to run the program. One standard example of a run-time error is where the source code has an instruction to divide a value by zero. The syntax may be correct (hence, no red

squiggly line), but the program will terminate early (i.e., crash) if the denominator has the value of 0. Usually run-time errors will cause the running program to crash, but they sometimes just generate warnings during the program execution.

- **Semantic errors or logic errors** are the most subtle, where the program is coded correctly and does not crash but do not perform the correct operation to produce the expected results. Errors of this kind cannot be found automatically by the IDE or easy to determine by the program's output. They are usually due to errors in the design or how the algorithm was translated into the source code.

Creating an Error Log

This assignment is designed for you to become more familiar with the error detection built into the PyCharm IDE. In particular, you will create a catalog of typical errors that may arise, and this document may hopefully help you in the future to identify the actual problem based upon the error message the IDE produces.

Complete each action below in the t0_chocolate.py file, and log the response from PyCharm on the right. Indicate three things:

1. **Classification:** Either **SYNTAX** , **RUNTIME**, **SEMANTIC/LOGIC** (only record the first error, with a very brief explanation) or **NONE** (for no errors). Remember that just because the IDE doesn't catch it, does not mean there is not an error.
2. **Error Indication:** Note the first error indication or message if one appears. Then copy or describe the resultant error and/or warning message you got into your Error log below.
3. **Correspondence**: How obvious was it to you that you made an error? In other words, did PyCharm identify the **actual** problem? Was the message clear, or cryptic and hard to understand? Briefly explain why or why not?
4. Finally, repair each change that you just made to restore the program to its original state. Thus, after making each of the charges, you will begin again with the original working code. The undo command (Ctrl + Z) is often useful here.

Example 0: **Change:** Remove the second quotation mark from the line: `entered_name = raw_input("Please enter your name: ")`	**Classification:** Syntax Error because it is a grammatical error **Error Indication(s):** On the line where the error exists, a red squiggly line appeared under the text. When I hold my mouse over the text, it says “Missing Closing Quote [“]”. The next line also has a red squiggly at the end of it, that says “End of statement expected” **Correspondence**: Excellent. The problem we created was that the string never ended. The warning symbol

	appeared on the correct line and the caret appeared just after where the problem was.

1. Change the line:
`oz_per_lb = 16`
to
`16 = oz_per_lb`
	Classification: syntax error
	
	Error Indication(s): cannot assign to literal
	
	Correspondence: I do not know what this means. I don't know what the "literal" is.

Remove the word `int` from the line: `num_lbs = int(raw_input("How many lbs of chocolate in a box? "))`	**Classification:** Syntax Error **Error Indication(s):** TypeError: unorderable types: str() > int() **Correspondence:** I understand this. The program is trying to compare a string variable with a integer variable which it cannot do.
Change both occurrences of `oz_per_lb` to `16oz_per_lb`	**Classification:** syntax error **Error Indication(s):** end of statement expected **Correspondence:** I think that it expects the line of code to end directly after the 16

Change the line
`print("Hello "+ entered_name+"! \n")`
to
`print("Hello "+ entered_name+"! /n")`
	Classification: none
	
	Error Indication(s): The /n showed up in the program when run
	
	Correspondence: Instead of starting a new line it just printed the /n

Change the line `if oz_choc > 500:` by changing the > to a <	**Classification:** none

	Error Indication(s): It flops the result of whether you are a chocaholic or not. **Correspondence:** I understood this fully. I knew exactly what was going to happen.
Remove the colon from the line `if oz_choc > 500:`	**Classification:** syntax error **Error Indication(s):** unexpected indent **Correspondence:** The print command was indented when it shouldn't be

Change the algorithm by changing the line
`oz_choc = lbs_choc * oz_per_lb`
to
`oz_choc = lbs_choc / oz_per_lb`
	Classification: none
	
	Error Indication(s): The resulting number was significantly lower than it should have been.
	
	Correspondence: I understand this because instead of multiplying to get your final answer, you are dividing.

Deliberately create a **syntax** error which differs from all of the above. Describe how to create the error and then create an error log for the error.	**Change: Remove the 6th apostrophe in line 40** **Classification:** Syntax error **Error Indication(s):** Missing end quote **Correspondence:** It is missing that apostrophe to finish the quote
Deliberately create a **semantic/logic** error which differs from all of the above. Describe how to create the error and then create an error log for the error.	**Change: Remove line 35** **Classification:** Semantic **Error Indication(s):** lines 34 and 36 do not print if the number is below 500 and if it is above 500 both lines print

	Correspondence: It creates a situation in which lines 34 and 36 will only be printed if the number is above 500.
Deliberately create a **run-time** error which differs from all of the above. Describe how to create the error and then create an error log for the error.	**Change:** add a /0 after the 500 in line 35 **Classification:** run-time **Error Indication(s):** ZeroDivisionError: division by zero **Correspondence:** I understand this because the program is trying to divide 500 by 0 which can't be done.

Your First Code - Python Conditionals

"Everything has beauty, but not everyone sees it" - Confucius[1]

The Chinese Zodiac, according to its Wikipedia page, is a " classification scheme that assigns an animal to each year in a repeating twelve-year cycle." The scheme is quite complex, so we will only be considering a small portion of the scheme for this assignment. Namely, the relationship between the twelve animals, and your year of birth:

Animal	Birth Year
Rat	…, 1972, **1984**, 1996, ...
Ox	…, 1973, **1985**, 1997, ...
Tiger	…, 1974, **1986**, 1998, ...
Rabbit	…, 1975, **1987**, 1999, ...
Dragon	…, 1976, **1988**, 2000, ...
Snake	…, 1977, **1989**, 2001, ...
Horse	…, 1978, **1990**, 2002, ...
Goat	…, 1979, **1991**, 2003, ...

[1] https://www.brainyquote.com/quotes/quotes/c/confucius104254.html

Monkey	…, 1980, **1992**, 2004, ...
Rooster	…, 1981, **1993**, 2005, ...
Dog	…, 1982, **1994**, 2006, ...
Pig	…, 1983, **1995**, 2007, ...

So, for example, I was born in 1982, which makes me a dog. Try to keep the jokes to a minimum…

In class, we took a brief tour of the following code: A1_pets.py. There are lots of good hints in this code that you'll want to use as you solve the problem for this assignment.

Your tasks

You have three tasks for this assignment. I suggest solving them in order, and not moving on from each subtask until you solve the prior task.

Download this starter code, A1_stubs.py, which includes notes for all tasks. HOWEVER, keep reading this document to make sure you complete all tasks!

Task 1

1. Create a Python program that asks the user for input. Namely, ask the user to put in the year they were born.
2. Print to the screen that user's Chinese Zodiac animal that corresponds to their birth year. For me, you would print something like "Your animal is the dog!"
   NOTE: For now, you only need to consider years between 1985 and 1994, as the vast majority of the class will fall in that range.

Task 2

In addition to the Chinese Zodiac showing your animal, it also shows who you are "most compatible" with, on this compatibility grid. So for example, I'm a Dog (1982), and if you are a Dragon (1988), we are not a good match. Sorry Dragons!

(I'll still try to get along with you in the class!)

Alternatively, you also have animals you are highly compatible with. In my case, I am most compatible with the Rabbit (1987), Tiger (1986), and Horse (1990). We're going to get along great!

Implementing **ALL** of these combinations would be far too much, especially this early in the course. Also, it'd very tedious by the time you got to the end, and you wouldn't learn more by doing all of them. **So, you will only need to implement the "No match" and "Best match" for your own animal.** So, I would only implement the "Dog" column on the table, as I described above.

1. In the same Python program, after the Task 1 code, ask the user to input a friend's birth year.
2. Print to the screen that friend's Chinese Zodiac animal that corresponds to their birth year. Again, only worry about the years between 1985 and 1994.

Task 3

1. Using the compatibility grid, print out one of two things:
    a. If your animal and your friend's animal are a "No match", print to the screen a witty message indicating why you aren't a good match.
    b. If your animal and your friend's animal are a "Best match", print to the screen an even wittier message about why you are the best of friends.
    c. Else, print to the screen the wittiest message about why you are "okay" friends.

A sample output is provided here, so you can see one possible output of your program.

```
Enter your birth year (e.g., 1982): 1994
You were born in 1994
You are a dog
Enter your friend's birth year (e.g., 1982): 1987
Your friend was born in 1987
Your friend is a rabbit
You two are besties!
```

Additional challenges (not graded)

1. Make the program work for any year, not just 1985 through 1996.
2. Make the program work for the entire compatibility table.
3. Minimize the number of if/elif/else statements.
4. Implement the month, day, and/or hour animal signs, as described in the Wikipedia page

Submission Instructions

1. Download this document as a PDF. To do this, go to File >> Download as...
2. Rename the document to ***A1_usernames.pdf.*** Replace *usernames* with your Berea username(s). For example, John and my document would be named **A1_heggens_hellrungj.pdf.**
   NOTE: From now on**,** incorrect filenames will automatically reduce your grade by 1 point

for each assignment. Fortunately, the format is always the same no matter what the assignment.

3. Rename your Python code to A1_usernames.py. Replace *usernames* with your Berea username(s). For example, John and my document would be named **A1_heggens_hellrungj.py.**
4. Zip the two files together. If you do not know how to zip two files together, refer to this short tutorial. Also, TA's in the evening lab will be happy to show you how to do this.
5. If you worked in pairs, one of your do step 6, the other do step 7.
6. (All) Upload the zip folder to Moodle by the due date listed on the course website: https://trello.com/b/w7bIrLoV/.
7. (Partners only) Upload a text file with both you and your partner's name in the document. See the example below:

Filename: A1_heggens_hellrungj.txt Scott Heggen John Hellrung