|  |  |
| --- | --- |
| Lab 6: Try | Name: Lex Baker  Date: 9/27/22 |

A **try statement** will attempt to execute the statements in a block of code. As soon as an error is encountered, execution in that block will stop immediately and the flow of execution will be diverted to statements in the appropriate except **clause**. The upshot of all this is that you can tell Python to do something constructive when an error occurs instead of having your whole program crash.

**1.** Open a new program file and add a header. Enter the following “program” and run it a few times with a variety of input types to make sure you see how it behaves in a variety of circumstances.

#Example of a bad way to get input

my\_number = int(input("I demand an integer."));

print("I got",my\_number);

**a.** For what inputs does this code fail?

**Any inputs that are not purely numbers, including decimals and letters**

**b.** There are three functions called in the program above. Which one of them is responsible for the failure?

**The int() function**

**c.** When this code fails, Python prints out an error message in the Shell. Copy down the last line of this error message word for word and see that you understand what it is saying.

**<< ValueError: invalid literal for int() with base 10: ‘example’ >>**

**This error is telling me that int() doesn’t know what to do with input ‘example’ because it can’t evaluate it into a base 10 number.**

**2.** Enter and run the following code to see the syntax of a try statement and to see what it does. Run it with a variety of different inputs. Remember to comment out the previous code.

#Example of a better way to get input

try:

my\_number = int(input("I demand an integer: "));

print("I got",my\_number);

except ValueError:

print("I don't have an integer, but I am not broken!");

When did you see ValueError before? Putting ValueError after the keyword except specifies which type of error will cause the statements in the except clause to be executed.

**3.** Wrapping the try statement in a while loop creates a persistent little code block that can get appropriate input from even the most inept of users. Explore the following code.

#Example of a robust way to get input

my\_number = None;

while my\_number == None:

try:

my\_number = int(input("I demand an integer: "));

except ValueError:

print("I am not broken, and I am not going away");

print("I got ",my\_number,". I knew I would win!!",sep = "");

Hopefully you appreciate this little construct and have some ideas of how it could improve your programs. Hopefully you also noticed the keyword None. None is both a type and a value that literally means “nothing here”. The existence of such an object seems at first preposterous, but it turns out to be quite useful. Here it allowed me to create a variable without associating a value with it. I needed the variable my\_number to exist to get into the while-loop, and as long as my\_number had no associated value, the loop kept iterating. Initializing my\_number to zero (or any integer value) is problematic in terms of the condition. Initializing my\_number to a string, even the empty string "", or a float or some other type is bad style. If a variable is going to be an int, it should never exist as anything but an int...or a None .

**Quick Check**

**Robust SSS**. Go back to problem 12 from Lab 5 (calculating the angles for a triangle given the side lengths). Make a copy of that file and replace the input lines with while/try constructs to make sure the user enters numbers.

**import math**

**a1 = None**

**a2 = None**

**a3 = None**

**while a1 == None or a2 == None or a3 == None:**

**s1 = float(input("Enter the first side length:\n"))**

**s2 = float(input("Enter the second side length:\n"))**

**s3 = float(input("Enter the third side length:\n"))**

**try:**

**a1 = math.acos((s2\*\*2 + s3\*\*2 - s1\*\*2) / (2\*s2\*s3)) \* (180/math.pi)**

**a2 = math.acos((s1\*\*2 + s3\*\*2 - s2\*\*2) / (2\*s1\*s3)) \* (180/math.pi)**

**a3 = math.acos((s1\*\*2 + s2\*\*2 - s3\*\*2) / (2\*s1\*s2)) \* (180/math.pi)**

**print("The angles, in order, are:", end=" ")**

**print(a1, a2, a3, sep=", ")**

**except ValueError:**

**print("The triangle lengths you input are invalid")**