1.11. Exception Handling

There are two types of errors that typically occur when writing programs. The first, known as a syntax error, simply means that the programmer has made a mistake in the structure of a statement or expression. For example, it is incorrect to write a for statement and forget the colon.

```
>>> for i in range(10)
SyntaxError: invalid syntax (<pyshell#61>, line 1)
```

In this case, the Python interpreter has found that it cannot complete the processing of this instruction since it does not conform to the rules of the language. Syntax errors are usually more frequent when you are first learning a language.

The other type of error, known as a logic error, denotes a situation where the program executes but gives the wrong result. This can be due to an error in the underlying algorithm or an error in your translation of that algorithm. In some cases, logic errors lead to very bad situations such as trying to divide by zero or trying to access an item in a list where the index of the item is outside the bounds of the list. In this case, the logic error leads to a runtime error that causes the program to terminate. These types of runtime errors are typically called **exceptions**.

Most of the time, beginning programmers simply think of exceptions as fatal runtime errors that cause the end of execution. However, most programming languages provide a way to deal with these errors that will allow the programmer to have some type of intervention if they so choose. In addition, programmers can create their own exceptions if they detect a situation in the program execution that warrants it.

When an exception occurs, we say that it has been “raised.” You can “handle” the exception that has been raised by using a `try` statement. For example, consider the following session that asks the user for an integer and then calls the square root function from the math library. If the user enters a value that is greater than or equal to 0, the print will show the square root. However, if the user enters a negative value, the square root function will report a `ValueError` exception.

```
>>> anumber = int(input("Please enter an integer "))
Please enter an integer -23
>>> print(math.sqrt(anumber))
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    print(math.sqrt(anumber))
ValueError: math domain error
>>>
```

We can handle this exception by calling the print function from within a `try` block. A corresponding `except` block “catches” the exception and prints a message back to the user in the event that an exception occurs. For example:

lStructures.html)

```
>>> try:
       print(math.sqrt(anumber))
    except:
       print("Bad Value for square root")
       print("Using absolute value instead")
       print(math.sqrt(abs(anumber)))

Bad Value for square root
Using absolute value instead
4.79583152331
>>>
```

will catch the fact that an exception is raised by `sqrt` and will instead print the messages back to the user and use the absolute value to be sure that we are taking the square root of a non-negative number. This means that the program will not terminate but instead will continue on to the next statements.

It is also possible for a programmer to cause a runtime exception by using the `raise` statement. For example, instead of calling the square root function with a negative number, we could have checked the value first and then raised our own exception. The code fragment below shows the result of creating a new `RuntimeError` exception. Note that the program would still terminate but now the exception that caused the termination is something explicitly created by the programmer.

```
>>> if anumber < 0:
...    raise RuntimeError("You can't use a negative number")
... else:
...    print(math.sqrt(anumber))
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
RuntimeError: You can't use a negative number
>>>
```

There are many kinds of exceptions that can be raised in addition to the `RuntimeError` shown above. See the Python reference manual for a list of all the available exception types and for how to create your own.

(ControlStructures.html)

(DefiningFunctions.html)

© Copyright 2019 DCC IME USP. Created using Runestone (http://runestoneinteractive.org/) 3.4.5. | Back to top

lStructures.html)