Debugging in Python

Debugging in Python

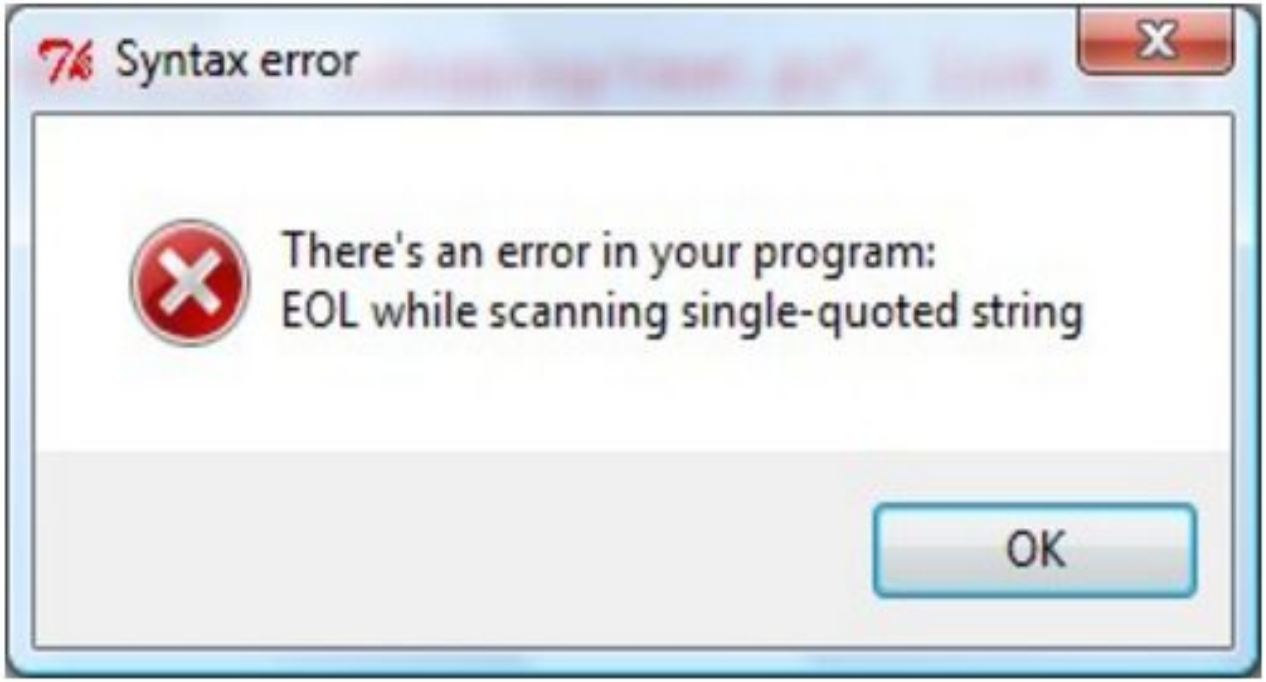
- **Syntax Errors:**
- These types of errors are usually **typing mistakes**, but more generally **it means that there is some problem with the structure of your program.**

- **Syntax errors** in Python will pop up a dialog box like the one below.

- The message in this box is *Syntax Error.*

- *There was an error in your program: EOL while scanning single-quoted string.*

print "hello world



- EOL stands for ***End Of Line*.**
- This error means that there was an open quote somewhere, but the line ended before a closing quote was found.

a = 3 + 5 7

- Another type of syntax error will simply say ***invalid syntax***.
- **An invalid syntax error means that there is a line that python doesn't know what to do with.**

- The last common type of syntax error you will likely encounter has to do **with indention**.

- **You may see *unindent does not match any outer indention level unexpected indent*.**

- **Solution:** When you press OK on the dialog box. **Python will attempt to highlight the offending line in your source code**.

- You should use this location as a hint for where to start looking for your problem.

- First check the area highlighted.

- Then check the entire line.

- Lastly, check the line or lines before the line highlighted.

- If you get an indention error, you should check that all of your lines of code are properly aligned in the correct columns.

Token Error (missing parenthesis

a = 3 + (4 + 5

- Token errors in Python will pop up a dialog box like the one below.

- This error usually means that there was an open parenthesis somewhere on a line, but not a matching closing parenthesis.

- Python reached the end of the file while looking for the closing parenthesis.
- **Solution:** When you press OK on the dialog box.

- Python will attempt to highlight the offending line in your source code.

- However, since it had reached the end of the file, it will highlight the last line in the file!

Runtime Errors

- Runtime errors occur as your program executes.

- Since Python is an interpreted language, these errors will not occur until the flow of control in your program reaches the line with the problem.

print hello
pring "hello"

- **Traceback (most recent call last): File "C:/Users/John/Documents/Teaching-BU/Python-debugging/test.py", line 7, in main() File**

- **"C:/Users/John/Documents/Teaching-BU/Python-debugging/test.py", line 5, in main print hello**

- **NameError: global name 'hello' is not defined**

- The first part tells you which file had the error.
- In the example above, the file is *test.py* and the error occurs on line 7.
- The next line shows the actual line of code where the error occurred.
- This line executes the main() function.
- Similarly, the next two lines say that the error occurred on line 5, within main , and that the line with the error is print hello.
- Lastly, the actual NameError says that *global name 'hello' is not defined*.

- **Usual Causes**:A mistyped variable or function name.
- Using a variable before it is defined.

- The name was intended to be enclosed in quotes.

Logic (semantic) errors

- Semantic or logic errors are problems with the design of your program.
- These usually do not produce any error message, but instead cause your program to behave incorrectly.
- These errors are often caused by accidentally using one variable in a place where a different variable is intended, or by simply doing some math incorrectly.

```
apples = 0
pickedApples = input("Pick some apples: ")
apples = apples + pickedApples
pickedApples = input("Pick some more apples: ")
apples = apples + pickedApples
print "You have a total of %d apples" %pickedApples
```

- The above code will execute, but it will not output the total number of apples picked.
- Instead, it will output the amount that was picked the last time!
- This simple example is easy to fix, but in a more complicated program it can be difficult to find such problems.

Experimental Debugging

- debugging is one of the most challenging, and interesting parts of programming.
- Debugging is also like an experimental science.

- Once you have an idea what is going wrong, you modify your program and try again.

- If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program.
- If your hypothesis was wrong, you have to come up with a new one.
- That is, programming is the process of gradually debugging a program until it does what you want.

Difference between Braces, brackets, and parentheses

- **Braces ("curly braces") : { }**

Are used in Python for creating Dictionaries.

- **Brackets ("square brackets") [ ]**

Are used in Python for creating List .

- **Parentheses: ( )**

Are used in Python for creating Tuples

formal and natural languages

- **Natural languages** are the **languages** that people speak, such as English, Spanish, and French.
- they evolved naturally.

- **Formal languages** are **languages** that are designed by people for specific applications.

- For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols.

- **Programming languages are formal languages that have been designed to express computations.**

- Formal languages tend to have strict rules about syntax.
- For example, 3+3=6 is a syntactically correct mathematical statement, but 3=+6$ is not.
- H_2O is a syntactically correct chemical name, but $_2Zz$ is not.
- Syntax rules come in two flavors, pertaining to **tokens** and **structure**.
- **Tokens** are the basic elements of the language, such as words, numbers, and chemical elements.

- The second type of syntax rule pertains to the **structure** of a statement.

- **that is, the way the tokens are arranged.**

- When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is. This process is called **parsing**.

- For **example**, when you hear the sentence, “The other shoe fell”, you understand that the other **shoe** is **the subject** and **fell** is the **verb**.

- Once you have parsed a sentence, you can figure out what it means, or the **semantics** of the sentence.

- Assuming that you know what a shoe is and what it means to fall, you will understand the general implication of this sentence.

- Although formal and natural languages have many features in common —
- **tokens, structure, syntax, and semantics**
- there are many differences:

- **Ambiguity**
- Natural languages are full of ambiguity.
- Formal languages are designed to be nearly or completely unambiguous,

- which means that any **statement has exactly one meaning, regardless of context.**

- **Redundancy**

- In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy.

- **Formal languages are less redundant and more concise.**

Loops in Python

```
primes = [2, 3, 5, 7]
for prime in primes:
print   (prime)
```

- # Prints out the numbers 0,1,2,3,4

```
for x in range(5):
    print(x)
```

- # Prints out 3,4,5

```
for x in range(3, 6):       #from 3 and less than 6
    print(x)
```

- # Prints out 3,5,7

```
for x in range(3, 8, 2):   #from 3 and less than 8 incremented by 2
    print(x)
```

"while" loops

- # Prints out 0,1,2,3,4

```
count = 0
while count < 5:
    print(count)
        count += 1
# This is the same as count = count + 1
```