2.2 What can go wrong?

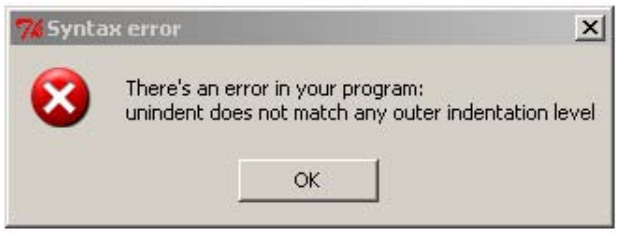
Let's look again at the first program we wrote in Section 2.1 and see what happens if we don't get it quite right. First, suppose we aren't consistent with our indentation:

```
# This asks for the user's name
# and prints a greeting.

def main():
        name = input( "Who are you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
main()
```

Program 2.2.1: Bad Indentation

In program 2.2.1 the line `name = input( "Who are you?")` is not fully indented. If you write a program inside Python's own editors it is pretty easy to avoid indentation problems, but if you write a program in a standard text editor (such as Notepad in Windows) mistakes like this can crop up. If we run this program in the Python system, we get the following error alert:



If we look at the code in IDLE, a portion of the line following the bad line is highlighted. Since the error message talks about indentation levels, it doesn't take too long to realize that there is a problem with the indentation somewhere, and if you look carefully at the code knowing that there is an indentation problem, the mistake isn't hard to spot.

Don't expect too much from error messages. Error detection in programs is an art form that is still far from perfect. It is fairly common for the system to flag the line after the one that actually contains the mistake. In this case the system misinterpreted our intention. "unindent does not match any outer indentation level" means that the system thought we were ending the main() function but failed to move to the previous indentation level, as in

```
def main():
    name = input( "Who are you? " )
if name == "bob":
          ⋮
```

That, of course, is not what we wanted, but it is close enough to allow us to find the real problem.

The Python error detection system stops as soon as it finds one problem. This makes error correction a cyclic affair: we fix one bug, try to rerun the program, fix the next bug that appears, and so forth. Experienced programmers often prefer systems that list all of the errors at once; it is faster to do multiple bug-fixes in one iteration than to handle them serially. The problem with systems like this is that one error can cause a whole sequence of errors. As we saw above, our improper indentation caused the Python system to think we were doing something completely different than we actually intended. If it had continued with the program it would have flagged the next line as being improperly indented (according to what it thought we intended) and these errors would continue through the rest of the program. It takes experience to look at a cascade of errors and determine what needs to be fixed. Python's system is much easier for beginners to use. It does have one offputting result. Each time you fix a bug it is the only bug you know about, so you believe the program will run correctly. This leads to a cycle of overconfidence followed by feelings of frustration. One way to minimize this is to keep debugging your program as you write it, so that with each run only a small piece of the code is new. We will discuss this at more length in a later section.

What happens if you forget a comment sign, as in the second line of the following program?

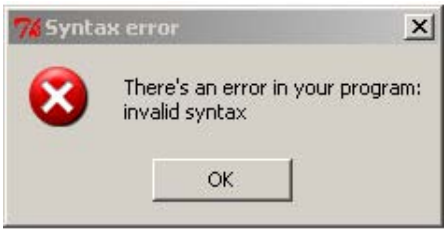
```
# This asks for the user's name
and prints a greeting.

def main():
    name = input( "Who are you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
    print( "Goodbye." )

main()
```

Program 2.2.2: Bad Comment

This time we get the cryptic error message:



This doesn't say much of anything. "invalid syntax" means that your program isn't constructed according to the grammar rules of Python. Fortunately, if we look at the code in Idle it shows us the problem. The second line of the code is highlighted:

```
and prints a greeting.
```

Now we know that there is a syntax problem with the second line, starting with the word **and** Of course, since this line is supposed to be a comment the system shouldn't be checking its syntax; this ought to lead us to realize that the problem is our missing comment symbol.

Many Python constructs use colons to separate various parts of a construction. An easy typing mistake is to leave off one of these colons. We do this in the following program, where the colon on"else" is omitted.

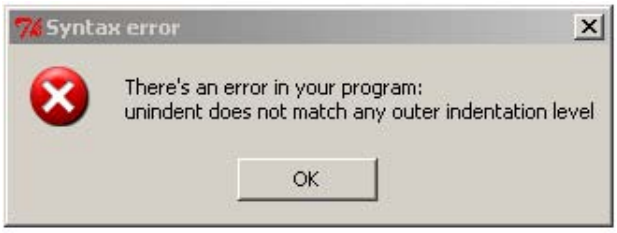
```
# This asks for the user's name
# and prints a greeting.

def main():
    name = input( "Who are  you?" )
    if name == 'bob':
        print( "Bob rules!")
    else
        print( "Hello, " + name + "!" )
    print( "Goodbye." )

main()
```

Program 2.2.3: Omitted colon

When we run this program the system again reports a syntax error:



Nothing here points to a missing colon as the problem, but if we look at the code the **else** line is highlighted. For someone who is familiar with how *if*-statements are constructed in Python (we will look at this in detail in Chapter 3) it is a small step to realize that the problem is a missing colon.

As a final example for this section, consider Program 2.2.4:

```
# This asks for the user's name
# and prints a greeting.

def Main():
    name = input( "Who are  you? " )
    if name == 'bob':
        print( "Bob rules!" )
    else:
        print( "Hello, " + name + "!" )
    print( "Goodbye." )

main()
```

Program 2.2.4: Case-sensitivity

In this program we define a function called Main() and at the end call a function called main(). Python, like most programming languages, is case-sensitive. Upper case letters are different from lower-case letters with the same name, so Main() and main() are different functions. If we try to run this program we four lines of red ink:

```
Traceback (most recent call last):
  File "C:\Documents and Settings\bob\My Documents\book\Program2.2
       line 13, in <module> main()
NameError: name 'main' is not defined
```

Don't be put off by the red ink or verbose text. The last two lines of this message have everything you need to know. On line 13 of the program:

```
main()
```

there is a problem because the word main isn't defined. Of course, it should have been defined in the definition at the top of the program, so as soon as we look at that definition we see that it defines Main() and not main(). Nothing is highlighted in your program because the system can't tell if you misspelled the name of the function in its definition or in its call, but this error message should be sufficient for you to figure out what the problem is and to correct your spelling.