7.4 Try-Except Statements

In programming terminology an *exception* is a terminal event, typically caused either by an error in the code or by an unexpected input from the user. The usual behavior of a program in the event of an exception is to crash. The **try−except** statement allows you to catch exceptions before they cause your program to crash. The form of this statement is

```
try:
        <the try−block of code>
except:
        <the except−block of code>
```

When this statement is executed, the system executes the **try**-block. If this block of code causes no problems the **except**-block is ignored. However, if the **try**-block causes an error the system executes the **except**-block. The **except**-block should make up in some way for the fact that the **try**-block did not execute correctly.

For example, consider the following program:

```
def main():
    done = False
    while not done:
            x = eval(input( "Enter a number: " ))
            if x < 0:
                done = True
            else:
                print("100/%d = %d" %(x, 100/x))
main()
```

Program 7.4.1: A program that generates an exception

If the user enters a number such as 4 the program outputs "100/4=25". If the user enters a negative number the program terminates. However, if the user enters 0 the program crashes. We can use a **try−except** statement to avoid this as follows:

```
def main():
    done = False
    while not done:
        try:
            x = eval(input( "Enter a number: " ))
            if x < 0:
                done = True
            else:
                print("100/%d = %d" %(x, 100/x))
        except:
            print("I can't divide by that.")

main()
```

Program 7.4.2: A fix for the previous program

Now if the user enters a 0 the program responds with "I can't divide by that." The program goes back to the top of the input loop and asks for another number. There is no crash and the program continues to function.

It is possible to make **except**-clauses respond only to specific types of exceptions. For example, the exception that is raised when we divide by 0 is called a ZeroDivisionError. If we make the **except**-clause

```
except ZeroDivisionError:
```

then the block of code for this clause will be executed only for divisions by zero; any other type of exception will not be handled by this clause. Here is a third version of the program:

```
def main():
    done = False
    while not done:
        try:
            x = eval(input( "Enter a number: " ))
            if x < 0:
                done = True
            else:
                print("100/%d = %d" %(x, 100/x))
        except ZeroDivisionError:
            print("I can't divide by that.")
        except NameError:
            print("I said a NUMBER, doofus.")
        except:
            print("Now, that's  just dumb.")
main()
```

Program 7.4.3: Catching specific exceptions

Now, if the user enters 4 the program responds "100/4=25" If the user enters 0 the program responds "I can't divide by that." If the user enters the string "ten" the program responds "I said a NUMBER, doofus." Finally, if the user enters something silly like "23 skidoo" the program responds "Now, that is just dumb."