Writing error-free code

When writing programs, code should be as legible and error free as possible. Debugging helps keep code free of errors and documenting helps keep code clear enough to read.

Errors and documenting code

When programs are written, it is likely that at least some errors will creep in. Errors in programs are often referred to as bugs.

It is vital that programs are as free of errors as possible. Errors can cause a program to produce unexpected results, or crash.

There are two types of error that need to be considered:

- syntax error
- logic error

Each type of error is different, and each is solved in different ways.

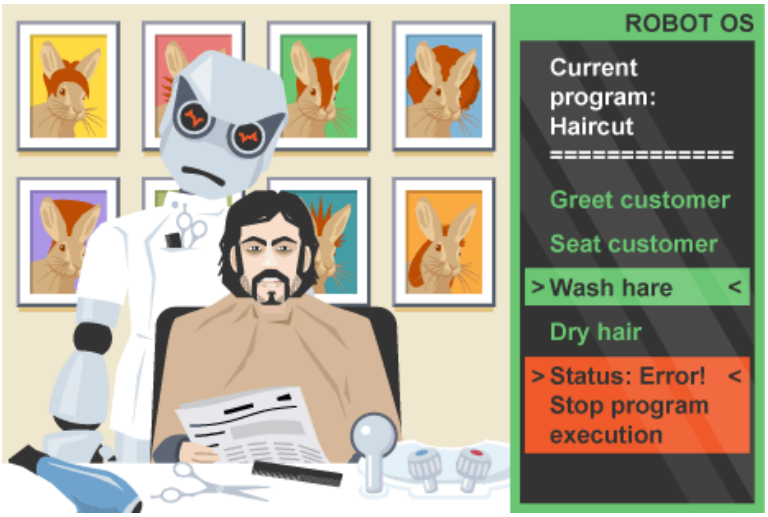
It is also good practice to document code. Documentation helps keep code easy to read and easy to understand.

Syntax errors

Syntax is the spelling and grammar of a programming language. Programming languages have rules for spelling, punctuation and grammar, just like the English language. In programming, a syntax error occurs when:

- there is a spelling mistake
- there is a grammatical mistake

Syntax errors will cause a program to crash or not run at all. The program may run until it encounters a syntax error, then it will stop.



Types of syntax error

More than one type of syntax error may exist. There may be:

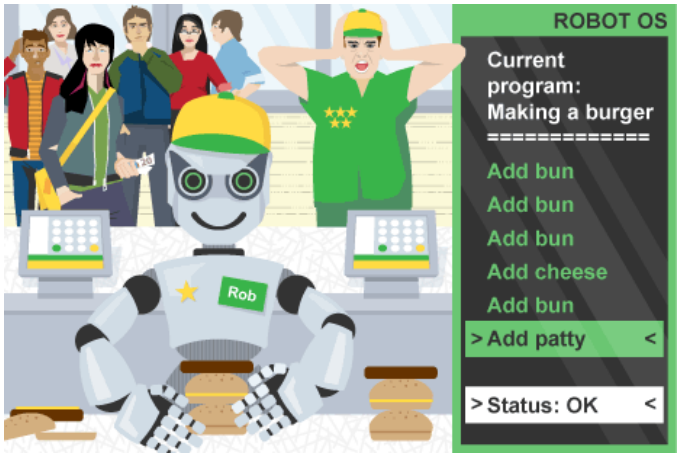
- incorrectly spelled statements
- incorrectly spelled variables
- missing punctuation (quotes, brackets, etc)

Any one or more of these errors may exist in a program, and each will cause the program to crash or not run at all.

Logic errors

Logic errors occur when there is a fault in the logic or structure of the problem.

Logic errors do not usually cause a program to crash. However, logic errors can cause a program to produce unexpected results.



Types of logic error

More than one type of logic error may exist. Parts of the program may:

- be in the wrong sequence
- have the wrong Boolean expression
- use the wrong data type
- be missing altogether

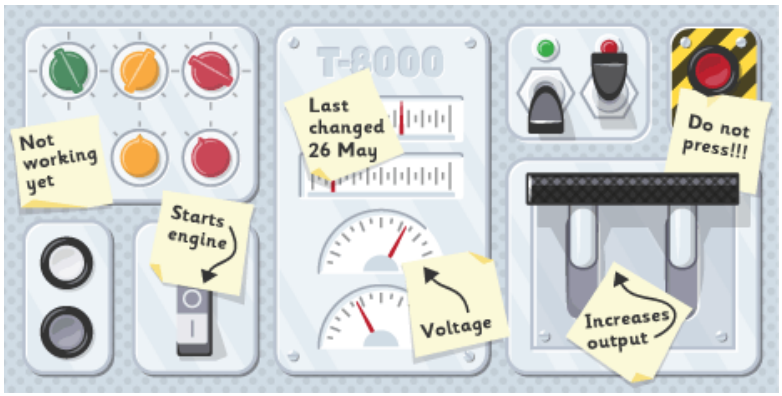
Any one or more of these errors may exist in a program, and each will cause the program to behave unexpectedly.

Documenting code

Code that is hard to read makes it difficult to understand what the program is trying to do. It also makes it difficult to understand the purpose of
any variables, procedures and functions. Errors are easier to fix when we understand the code we are reading through.

When we document code, we make it easier to read and understand. We also document code in case:

- we need to come back to it at a later date
- someone else needs to change or fix it



How to document code

Documenting code is very straightforward. It involves:

- giving meaningful names for variables, procedures and functions
- placing comments within the code to explain the purpose of each step

Having properly documented code makes it much easier for a programmer to understand and, if necessary, debug the program.

Using meaningful names

Wherever possible, meaningful names should be used
for variables, procedures and functions. Look at this simple Python program:

```
x = int(input("Enter x: ")) y = x * 4 print("y = ") print(y)
```

At first glance it may be hard to understand what this program does. It is clear that one number is inputted and another number is outputted, but beyond that it is not clear what the purpose of the program is.

Now look at this simple Python program:

```
length = int(input("Enter the length of one side: "))

perimeter = length * 4

print("Perimeter of your square is: ")

print(perimeter)
```

This time it is much easier to determine the purpose of the program (ie to calculate the perimeter of a square). The only difference between the two programs is that the second program uses meaningful names for variables (and in messages).

A meaningful name is one that allows us to easily understand what the variable, procedure or function does or is used for.

Meaningless names make code hard to understand. Sensible and meaningful names make the purpose of code easy to decipher.

Using comments

A comment is one or more sentences that explain the purpose of a section of code. A comment is placed just before (or after) the code to which it refers.

Well-written comments explain:

- the name of the program and its author
- what the programmer intends the code to do
- whether the programmer thinks the code could be better written
- where the program may be incomplete or need updating

In Python, comments start with a hash symbol – '#'. Look at this simple Python program:

```
Print("Hello Everyone")# This prints a message hello everyone
```

Comments make your code easier to understand. Try to include meaningful comments in your code wherever possible.

Be aware, though, that including too many comments can actually make code HARDER to read - so use them appropriately.