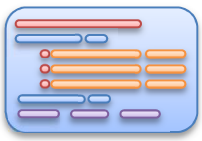


Computers are machines; they just do what they are told. If there are problems with the instructions given to the computer, then they won't behave in the way that you expect them to. Although there are occasionally issues with the things happening behind the scenes, most of the errors programmers have to deal with are caused by the programmer themselves.

***Aim:*** *To learn about errors and methods of debugging programs.*

## Task 1 – Types of Errors (or Bugs)

Three of the types of errors you might come across are described below. See if you can match each error to its cause. A simple search in a dictionary for the underlined words should help.

| | Type of Error | | | | Cause |
|---|---|---|---|---|---|
| 1 | **Syntax Error** | • | • | a | Problems with the logic. The program will run without an error being reported, but the results won't be as intended. |
| 2 | **Semantic Error** | • | • | b | The program will crash, often because of computer memory problems. There may be nothing wrong with the code. |
| 3 | **Runtime Error** | • | • | c | Spelling mistakes; incorrect punctuation or word order; inconsistent use of upper-case letters. These errors can cause the program to crash. |

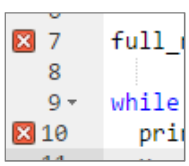
## Task 2 – Debugging Errors

**a.** The code below was meant to ask the user for their first and last names, then output the full name to the console **ten times**. It has at least 12 errors. List all those that you can find and suggest whether each is a syntax or semantic error.

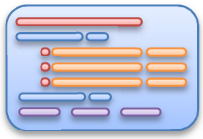
***Note:*** *Line 9 reads 'while the value of the variable, x, is less than 10 then do the following'. This line has no mistakes.*

```
1   x = 1
2
3   first_name = input("Enter your last name".)
4
5   last_name = input"Enter your first name.")
6
7   full_name = First_name & "  & lastname
8
9   while (x < 10):
10    Prin(fll_name)
11    x = x + 1
```

**b.** Recreate the program in *repl.it* and see if you can fix all the problems. Hover your mouse pointer over any red error messages that appear next to the code; these will tell you about the syntax errors. Semantic errors should be identified when you run the code. Save the program as '**10.2 Debugging**'.



# Errors and Tracing (page 2)



## Task 3 – Tracing Variables on Paper

Semantic errors are caused by problems in the logic. The program doesn't crash, but you don't get the results that you were hoping for. Semantic errors can often be investigated by tracing the changing variables through the algorithm to find out what is going wrong.

The program below was designed to accept a number inputted by the user, double it five times and then output the answer. The program isn't crashing, but it's not giving the correct answers.

Use a table to trace the values of the variables *x* and *number_int* as the program circulates through the loop. Assume a value of 1 is entered in the console and work out what is going wrong. We have started the process for you.

```
1  number = input("Enter an integer: ")
2  number_int = int(number)
3  x = 1
4
5  while(x < 5):
6    number_int = number_int * 2
7    x = x + 1
8
9  print("Answer = " + str(number_int))
```

**Line 9 converts the number to a string so that it can be concatenated before being displayed. You would otherwise get an error when you try and add text to an integer.**

| *x (at start of while loop)* | *number_int (after calc)* | *x (at end of while loop)* |
|---|---|---|
| 1 | 2 | 2 |
| 2 | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**Trace the values of *x* and *number_int* for an inputted value of 1**

## Task 4 – Outputting Values

A common way of investigating what might be going wrong is to display the values of the variables at different points in a program.

The output on the right is from the program in the previous task. Type the program in a new session and save as '**10.4 Tracing**'.

By adding extra lines of code, recreate the output exactly as shown. Use instructions like those below to display the variables and add extra blank lines. Fix the problem so that the integer entered is doubled five times.

```
print("")
```

**Output a blank line**

```
print(str(x) + " = x at end of loop \n")
```

**Output some text, followed by a new line (\n)**

```
Enter an integer:  1

1 = x at start of loop
2 = number_int at start of loop
2 = x at end of loop

2 = x at start of loop
4 = number_int at start of loop
3 = x at end of loop

3 = x at start of loop
8 = number_int at start of loop
4 = x at end of loop

4 = x at start of loop
16 = number_int at start of loop
5 = x at end of loop

Answer = 16
```

## Extension

1. You can create an infinite loop by deleting (or commenting out) line 7. The variable *x* never reaches 5 so the program never stops. You can try the Stop button, but you will probably have to restart your browser anyway. Infinite loops are a common semantic error.
2. Find out about 'Stepping' and 'Breakpoints' in terms of debugging computer programs. Work out whether there is any way to use these things in *repl.it*.