Bugs

(Errors)

Bugs and error messages

If something's wrong with your code, Python will try to help by showing an error message.

These messages can seem a bit puzzling at first, but they'll give you clues about why your program isn't working and how to fix it.

An error message tells you what type of error has occurred and where to look in your code.

Syntax errors

When you get a syntax error message, it's a hint that you've typed something incorrectly. Perhaps your fingers slipped and hit a wrong letter? Don't worry—these are the easiest errors to fix. Check through your code carefully and try to spot what went wrong.

The closing bracket is missing—it needs another curved bracket here.

```
input('What is your name?'
```

The first quotation mark is missing. It needs to be a single quote to match.

```
print(It is your turn')
```

▷ Things to look out for

Are you missing a bracket or quotation mark? Do your pairs of brackets and quotation marks match? Have you made a spelling mistake? All these things can cause syntax errors.

This is a spelling mistake—it should be `short_shots`.

```
total_score = (long_shots * 3) + (shoort_shots * 2)
```

Indentation errors

Python uses indentation to understand where blocks of code start and stop. An indentation error means something is wrong with the way you've structured the code. Remember: if a line of code ends with a colon (`:`), the next line must be indented. Press the space bar four times to manually indent a line.

```
if weekday == True:
print('Go to school')
```

This line of code would trigger an indentation error message.

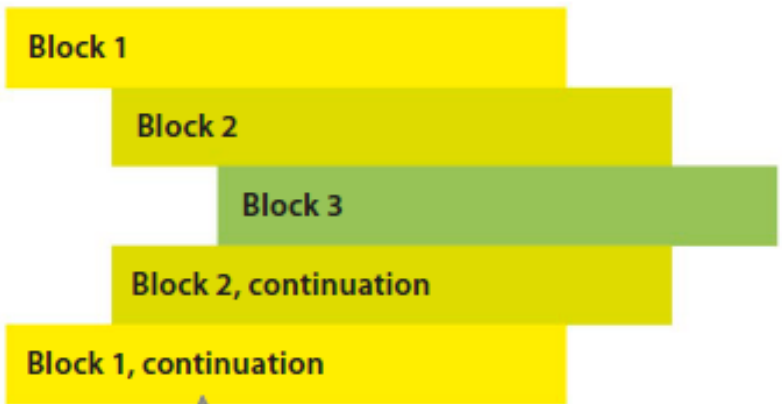
```
if weekday == True:
    print('Go to school')
```

Four spaces

You need to indent the code on the second line like this to fix the error.

▽ Indent each new block

In your Python programs, you'll often have one block of code within another block, such as a loop that sits inside a function. Every line in a particular block must be indented by the same amount. Although Python helps by automatically indenting after colons, you still need to check that each block is indented correctly.



The indents tell Python which lines of code belong to which block.

Type errors

A type error isn't a typing error—it means your code has mixed up one type of data with another, such as confusing numbers with strings. It's like trying to bake a cake in your refrigerator—it won't work, because the refrigerator isn't meant for baking! If you ask Python to do something impossible, don't be surprised if it won't cooperate!

```
budget = 'Fifty' * 'Five'
```

You can multiply two numbers in Python, but you can't do multiplication with strings.

```
hot_day = '20 degrees' > 15
```

Python can't check to see if a string is greater than a number, because they are different data types.

```
list = ['a','b','c']
find_biggest_number(list)
```

This function is expecting you to give it a list of numbers, but you've given it a list of letters instead!

◁ **Examples of type errors**

Type errors occur when you ask Python to do something that doesn't make sense to it, such as multiplying with strings, comparing two completely different types of data, or telling it to find a number in a list of letters.

Type errors

A type error isn't a typing error—it means your code has mixed up one type of data with another, such as confusing numbers with strings. It's like trying to bake a cake in your refrigerator—it won't work, because the refrigerator isn't meant for baking! If you ask Python to do something impossible, don't be surprised if it won't cooperate!

```
budget = 'Fifty' * 'Five'
```

You can multiply two numbers in Python, but you can't do multiplication with strings.

```
hot_day = '20 degrees' > 15
```

Python can't check to see if a string is greater than a number, because they are different data types.

```
list = ['a','b','c']
find_biggest_number(list)
```

This function is expecting you to give it a list of numbers, but you've given it a list of letters instead!

◁ **Examples of type errors**

Type errors occur when you ask Python to do something that doesn't make sense to it, such as multiplying with strings, comparing two completely different types of data, or telling it to find a number in a list of letters.

Type errors

A type error isn't a typing error—it means your code has mixed up one type of data with another, such as confusing numbers with strings. It's like trying to bake a cake in your refrigerator—it won't work, because the refrigerator isn't meant for baking! If you ask Python to do something impossible, don't be surprised if it won't cooperate!

```
budget = 'Fifty' * 'Five'
```

You can multiply two numbers in Python, but you can't do multiplication with strings.

```
hot_day = '20 degrees' > 15
```

Python can't check to see if a string is greater than a number, because they are different data types.

```
list = ['a','b','c']
find_biggest_number(list)
```

This function is expecting you to give it a list of numbers, but you've given it a list of letters instead!

◁ **Examples of type errors**

Type errors occur when you ask Python to do something that doesn't make sense to it, such as multiplying with strings, comparing two completely different types of data, or telling it to find a number in a list of letters.

Name errors

A name error message appears if your code uses the name of a variable or function that hasn't yet been created. To avoid this, always define your variables and functions before you write code to use them. It's good practice to define all your functions at the top of your program.

▷ **Name errors**

A name error in this code stops Python from displaying the message "I live in Moscow". You need to create the variable `hometown` first, before you use the `print()` function.

The `print()` instruction needs to come after the variable.

```
print('I live in ' + hometown)

hometown = 'Moscow'
```

Logic errors

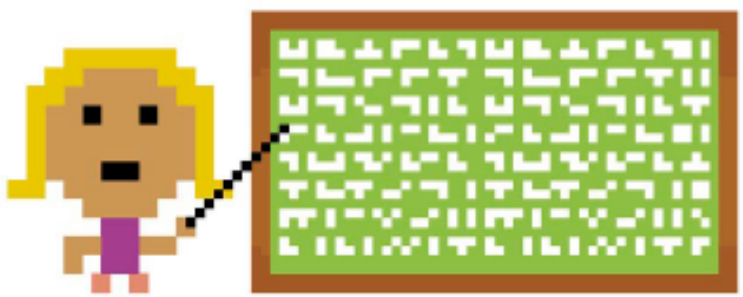
Sometimes you can tell something has gone wrong even if Python hasn't given you an error message, because your program isn't doing what you expected. It could be that you've got a logic error. You may have typed in the code correctly, but if you missed an important line or put the instructions in the wrong order it won't run properly.

```
print('Oh no! You've lost a life!')
print(lives)
lives = lives - 1
```

All the lines of code are correct, but two are in the wrong order.

◁ **Can you spot the bug?**

This code will run with no error messages, but there's a logic error in it. The value of `lives` is shown on the screen before the number of lives is reduced by one. The player of this game will see the wrong number of lives remaining! To fix it, move the instruction `print(lives)` to the end.



◁ **Line by line**

Logic errors can be tricky to find, but as you get more experienced you'll get good at tracking them down. Try to identify logic errors by checking your code slowly, line by line. Be patient and take your time—you'll find the problem in the end.