ERRORS

An Introduction to Computer Science

Let's learn about Errors.

Purpose

When something goes wrong with your program, Python will give you an error message. These messages are meant to help you find where your error is and what kind of error it is.

Types of Error Messages

- StopIteration
- SystemExit
- ArithmeticError
- OverflowError
- FloatingPointError
- ZeroDivisionError
- AssertionError
- AttributeError
- EOFError
- ImportError
- KeyboardInterrupt
- LookupError
- IndexError
- KeyError
- NameError
- UnboundLocalError
- EnvironmentError
- IOError
- OSError
- SyntaxError
- IndentationError
- SystemError
- SystemExit
- TypeError
- ValueError
- RuntimeError
- NotImplementedError

Python has many, many error messages.
It can take a long time to learn them all.
When you encounter an error message you are unfamiliar with, you should first read and think about the message.
If you are not sure what it means, you should look up the error's meaning in the documentation.
Then, you can debug your program.

Terminology

"Errors" or "Exceptions"

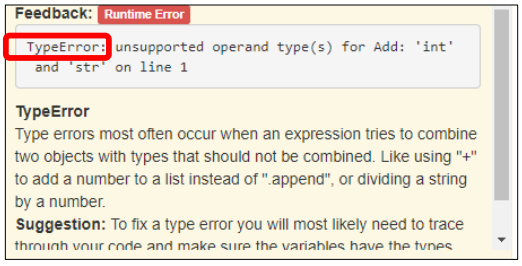
Just a minor point of terminology.
Errors are sometimes referred to as "Exceptions".
They're not quite identical, but for our purposes they might as well be.

Identify the error type

Thonny

```
Traceback (most recent call last):
  File "C:\Users\acbart\Projects\cisc108\cisc108-python\assignments\Programming 14- Errors\bad_code_example.py", line 1, in <module>
    5+""
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

BlockPy



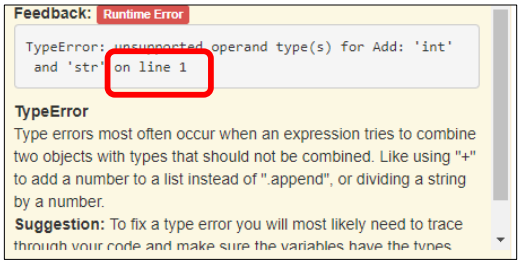
Let's imagine that we ran some code, and encountered an error message.
In the bottom left of the error message, you can see the type of error.
In the example here, you can see that we have a "TypeError".
After the error type, there is usually some further description of the error.

Find the Line

Spyder

```
Traceback (most recent call last):
  File "C:\Users\acbart\Projects\cis108-python\assignments\Programming 14- Errors\bad_code_example.py", line 1, in <module>
    5+""
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

BlockPy



It can be tricky to read, but the error message should also identify WHERE the error occured.
Of course, sometimes the error is actually caused by code on a previous line.
You will have to think very critically about whether the error message is correct.

Common Errors: NameError

```
student_grade = 10

print(studetn_grade)
```

This typo will cause a NameError!

Let's look at a few common errors.
First, here is a NameError.
A NameError comes up when you try to reference a variable that does not exist yet.
Sometimes, you mispelled the name of the variable.
Sometimes, you have not initialized the variable.
Sometimes, you initialized the variable, but you did so AFTER its first usage.
In this case, there is a typo in the name "student grade".

Common Errors: TypeError

```
print("7"+3)
```

TypeError: cannot concatenate 'str' and 'int' objects on line 1

Another common error is the TypeError, which occurs when you use an operator incorrectly.
For example, adding together a string and a number is not allowed.
When you read the error message that is produced, it will describe what went wrong.
Here, it says that we attempted to concatenate (which means combine) a string and an integer.

Common Errors: SyntaxError

```
name = "Klaus von Wagglyschwantz"
age = 17
is_good_dog = True
print("Klaus is a good dog?)
print(is_good_dog)
```

SyntaxError: bad token on line 4
Missing quotation mark

A syntax error means that you broke Python's rules of spelling, grammar, and punctuation.
There are many ways to break a programs' syntax.
Usually, Python is very good at suggesting where the error is.
Still, sometimes it can be very tricky to track down a SyntaxError.

Debugging Errors

We'll learn some more techniques to help you fix your broken problems in the coming weeks.
For now, think critically about the errors you receive.
When you see one you don't understand, you should ask "What does this error message mean?" instead of "What is wrong with my program?"