Exceptions

What we will cover...

1. What are errors/exceptions in Python.
2. EAFTP.
3. try/except blocks.

Plans

Usually, things go the way we planned.

Unfortunately, there are **exceptions** to this rule.

Our code needs to deal with both the plan A *and* the exception.

Errors

We've already seen that this will "throw an error".

Errors in Python are called **Exceptions**.

We like Exceptions. Paradoxically, we should try and expect them to happen. That makes for robust code!

```
a = 5
b = '10'

a + b
```

Try, except

A common pattern in Python is called **EAFTP**.

Easier to Ask Forgiveness Than Permission.

The basic idea is, rather than checking whether you can do something, just try and do it!

Then, if it doesn't work, have a backup plan.

```
a = 5
b = '10'

a + b
```

Try, except

Implementing this pattern involves two keywords that must be used together: `try` and `except`.

After the `try:` keyword, you have a **block**. This block, just like a function body, must be indented **4 spaces**. The same with the block after `except:`.

In the blocks, you can put any valid Python code!

```
a = 5
b = '10'

try:
    print(a + b)
except:
    print('all good, I knew that might happen')
```

Try, except

It's best practise to put a specific **type** of exception after the `except` keyword. That way, it only triggers for exceptions you were expecting, not for ones you weren't!

In this case, the block will only run for `TypeError` exceptions.

Why would we not want errors we weren't expecting to go to the except block?

```
a = 5
b = '10'

try:
    print(a + b)
except TypeError:
    print('all good, I knew that might happen')
```

Inside a function

Often, your try/except blocks go inside a function!

Note the whitespace, double nesting:

```
def adder(a, b):
    try:
        return a + b
    except TypeError:
        return None

adder(5, '10')
```

Review

1. What are errors/exceptions in Python.
2. EAFTP.
3. try/except blocks.