Conditionals and Raising Exceptions

What we will cover...

1. What are conditionals
2. if, elif, else
3. Using conditionals to raise exceptions

Control flow

Control flow allows us to execute certain **blocks** of code, based on a boolean value.

To create a block of code that should only execute when the boolean is true, we use `if`.

Note, as always, the whitespace!

```python
a = False

if a:
    print('hello!')
```

Control flow

Sometimes we want two blocks of code:

1. Only executes if the boolean is True.
2. Executes otherwise.

This is done with the `else` keyword.

Either the **if block** or the **else block** will execute. Never both!

```python
a = False

if a:
    print('hello!')
else:
    print('goodbye!')
```

Control flow

Of course, we can use comparison operators to create a boolean.

```
a = 5

if a > 5:
    print('hello!')
else:
    print('goodbye!')
```

Control flow

We can also use the `and` or the `or` operator to combine two booleans into a single boolean for use in control flow.

```
a = 5
b = 'foo'

if a > 5 or b == 'foo':
    print('hello!')
else:
    print('goodbye!')
```

Control flow

Sometimes we want several if clauses. We can achieve this with the `elif` keyword.

Once again, the **if block**, **elif block**, and **else block** are all mutually exclusive. Only one will execute!

What will this print for different values of `a`?

```
a = 5

if a > 5:
    print('hello!')
elif a > 0:
    print('eh')
else:
    print('goodbye!')
```

Truthy and Falsy

What if we try to give a non-boolean to the `if` statement?

Python will try to **cast** the value into a boolean, then use the result of that casting to perform the control flow.

This can be convenient, but it can also be dangerous!

```
a = 5

if a:
    print('hello!')
else:
    print('goodbye!')
```

Truthy and Falsy

What will this return?

```
a = 0

if a:
    print('hello!')
else:
    print('goodbye!')
```

Truthy and Falsy

Often we want to check if something exists, for which we can compare with `None`!

```python
a = 0

if a is not None:
    print(a)
else:
    print('goodbye!')
```

Exceptions

Exceptions are our friends.

We like our code to raise explicit, friendly, helpful suggestions whenever things aren't as they should be.

We can raise exceptions with the `raise` keyword, followed by an exception **type**. `Exception` is the most basic type of exception.

```
a = 0

if a is None:
    raise Exception('a should exist, but it doesnt!')
```

Exceptions

Exceptions take a **message** parameter, which is a string that describes what went wrong.

Helpful exception messages are an important part of writing good code!

```
a = 0

if a is None:
    raise Exception('a should exist, but it doesnt!')
```

Exceptions and functions

Good functions throw exceptions when they are given bad data.

Why?

```
def printer(a):
    if a is None:
        raise Exception('I can not print nothing!!')
    print(a)
```

Review

1. What are conditionals
2. if, elif, else
3. Using conditionals to raise exceptions