DEBUGGING

Objectives

- Explain common errors and how they occur in Python
- Use pdb to set breakpoints and step through code
- Use try and except blocks to handle errors

You're going to make mistakes...

So how do you fix them?

Understand Them!

Let's start with some common errors!

SyntaxError

- Occurs when Python encounters incorrect syntax (something it doesn't parse).
- Usually due to typos or not knowing Python well enough

```
def first: # SyntaxError

None = 1 # SyntaxError

return # SyntaxError
```

NameError

This occurs when a variable is not defined,
i.e. it hasn't been assigned

```
test
# NameError: name 'test' is not defined
```

TypeError

Occurs when:

- An operation or function is applied to the wrong type
- Python cannot interpret an operation on two data types

```
len(5)
# TypeError: object of type 'int' has no len()

"awesome" + []
# TypeError: cannot concatenate 'str' and 'list' objects
```

IndexError

Occurs when you try to access an element in a list using an invalid index (i.e. one that is outside the range of the list or string):

```
list = ["hello"]
list[2]
# IndexError: list index out of range
```

ValueError

This occurs when a built-in operation or function receives an argument that has the right type but an inappropriate value:

```
int("foo")
# ValueError: invalid literal for int() with base 10: 'foo'
```

KeyError

This occurs when a dictionary does not have a specific key:

```
d = {}
d["foo"]
# KeyError: 'foo'
```

AttributeError

This occurs when a variable does not have an attribute:

```
"awesome".foo
# AttributeError: 'str' object has no attribute 'foo'
```

Raise Your Own Exception!

In python we can also throw errors using the raise keyword. This is helpful when creating your own kinds of exception and error messages.

```
raise ValueError('invalid value')
```

YOUR
TURN

Handle Errors!

In Python, it is **strongly** encouraged to use *try/except* blocks, to catch exceptions when we can do something about them. Let's see what that looks like.

```
try:
    foobar
except NameError as err:
    print(err)
```

Why Not Catch Them All?

```
try:
    colt
except:
    print("You tried to use a variable that was never declared!")
```

What we are doing here is catching **every** error, which means we are not able to correctly identify "what" went wrong. It is highly discouraged to do this.

Any Better?

```
try:
    colt
except NameError:
    print("You tried to use a variable that was never declared!
```

When you use try/except, make sure that a specific type of exception is being handled.

If you want to except a handful of exceptions, you can pass a tuple of errors into the except block as well:

```
try:
    colt.hello
except (TypeError, AttributeError):
    print("That doesn't work with this thing.")
```

Debugging with pdb

To set breakpoints in our code we can use *pdb* by inserting this line:

```
import pdb; pdb.set_trace()
```

import

This is how we include modules! We'll learn about these in quite a bit more depth later.

```
import pdb; pdb.set_trace()
```

For now, think of this as loading some external code and then running code right away

Using pdb

```
def add_then_multiply(num1, num2):
    sum = num1 + num2
    import pdb; pdb.set_trace()

    product = sum * num1 * num2

    return product
```

Inside of the debugger we can press **c** to continue and **q** to quit.

There are a few more shortcuts as well; you can read more about pdb here.

pdb Gotcha

```
def add_numbers(a, b, c, d):
    import pdb; pdb.set_trace()

    return a + b + c + d
```

Be careful with how you name your parameters!

Recap

- Python has many different types of errors
- You can use raise to throw your own errors
- using try and except is the best way to handle errors, just make sure you handle specific errors
- pdb is very useful for setting breakpoints and pausing execution of code
- certain characters have meaning in pdb so be careful with naming variables

YOUR TURN