Errors

Common Errors

SyntaxError

NameError

IndexError

TypeError

ValueError

KeyError

raise

We can raise our own exceptions (force them happen)
whenever we want, using the **raise** keyword

raise

You can also provide a specific message when raising an exception

try/except

You can also provide a specific message when raising an exception

try/except

```
try:
    <code that could generate error>
except:
    <code that runs if error raised>
```

We can use the try and except keywords to handle exceptions. If an exception is raised in the try block, the except block will run.

try/except

```
try:
    num = int(input("Please enter a number: "))
except ValueError:
    print("Oh no, that isn't a number!")
```

Usually it's better to except a specific exception and handle it, rather than handling any possible exception that could occur.

Multiple Excepts

```
try:
    num = int(input("Enter an integer: "))
    print(10/num)
except ValueError:
    print("That's not an int!")
except ZeroDivisionError:
    print("Can't divide by zero!")
```

EAFP

Easier to ask for forgiveness than permission

"Assume things exist or will work, and catch exceptions if you're wrong"
Coding style characterized by lots of try/except blocks

LBYL

Look Before You Leap

Coding style where you explicitly test for pre-conditions before making calls or "leaping". Characterized by lots of if statements

LBYL

"Look"
Check to see if year is numeric.

```
year = input("Enter a year:")
if year.isnumeric():
    year = int(year)
else:
    year = 2025
```

"Leap"
Convert year to int once we know it's safe

EAFP

Assume it'll work
Try converting year to an integer

```
try:
    year = int(input("Enter a year:"))
except ValueError:
    year = 2025
```

Catch exception if you're wrong!
This code runs if year can't be cast to an int