Exceptions

- Suggested video Exceptions in Python by Socratica

Study the following code and ask, what will Python do if the user input's `0`? Test your hypothesis.

```
denominator = float(input("Please enter a number: "))
result = 100.0 // denominator
print(f"100 divided by {denominator} is {result}")
```

Exceptions are the error events that occur when your program is running that Python can not recover from on its own, so will result you your program crashing out.

We can avoid program crashes from exceptions. If we are creating code that we consider may result in an exception, we can preempt it in our code providing Python with an alternative to take. This may be referred to as catching the exception.

Catching an exception

Firstly, we can have a generic "catch every exception" response with a "try and except" set of blocks.

```
try:
    denominator = float(input("Please enter a number: "))
    result = 100.0 // denominator
    print(f"100 divided by {denominator} is {result}")
except:
    print("I can't do that!")
```

A warning

You should be aware that while possible, the failsafe catch-all of `except:` is considered poor practice. This is because it hides bugs in your code since any error in your code will create an exception.

The following excample will illustrate the problem...

```
try:
    denominator = float(input("Please enter a number: "))
    result = 100.0 // denoninator
    print(f"100 divided by {denominator} is {result}")
except:
    print("I can't do that!")
```

This code looks the same as before but has one critical different. I have inserted a deliberate bug... intentionally made hard to find... can you spot it? Hint: There is a spelling error on a variable name.

This bug means no matter what the user enters when they run the program, it will fail and produce the `I can't do that!` message. This would become very frustrating for you as the programmer as you won't understand why your code is not working.

Catching specific exceptions

A better solution is to specify which errors we are anticpating and to catch those specifically. To find the label to use for the exception statement, you can either run the code in such a way as to generate the error (and thus read it from the error statement), or check the Python Exceptions documentation at the link for the description of each official exception.

- https://docs.python.org/3/library/exceptions.html

A better solution to our problem is thus...

```
try:
    denominator = float(input("Please enter a number: "))
    result = 100.0 // denominator
    print(f"100 divided by {denominator} is {result}")
except ValueError:
    print("That wasn't a number")
except ZeroDivisionError:
    print("I can't divide by zero")
```

Rasing exceptions

Occassionally you might have a valid cause to want to Python to have an exception event.

We can use the `raise` statement to do this.

Raising a generic exception like shown here however is considered bad practice and should be avoided. For the same reason as catching generic exceptions... they hide bugs.

I generally only ever use this approach when I am developing a program and want to quickly test if a particular section works as intended or not, and so would want the program to fail. Deliberately causing your program to fail when in real-world use is very bad practice and lazy programming. You should program your way around it with code that responds to errors rather than causing them.

```
print("I am about to fail")
x = 10
if x > 5:
    raise Exception("Not allowed to have a number greater than 5")
print("You will never see me print")
```

Raising exceptions is all beyond the scope of your course. If you are interested in more on this, I suggest getting the relevant Python documentation at..

- https://docs.python.org/3/tutorial/errors.html#raising-exceptions

Problem set

Do not use the generic exception handler in your responses to these problems.

Q1. Create a quadratic formula calculator that uses exceptions...

- to ensure the `a`, `b` and `c` entered by the user are numbers
- to detect if `a` is zero (would result in a division by zero)
- to detect if there are no real solutions (square root of a negative number exception)

Q2. Question 5 of the Files problem set reads "*Read a list of strings from a text file. Tell the user how many lines there are and ask them to enter a line number indicating one they would like to read. Print just the content of that line to the user*".

- Add exception handling in case the file does not exist.
- Add exception handling in case the user asks for a line number beyond the limits of the list (ie: if there are only 5 lines and the user askes for line 6).