Error Handling and Simulation

CMSC 104 Section 2
October 28, 2025

Administrative notes

Classwork 7 will be on Thursday

Error handling

We've talked a number of times about errors that can crash your program

- Entering a name where a number is needed
- Trying to calculate the square root of a negative number
- Entering a negative number when you're trying to calculate a Fibonacci sequence
- Etc.

In general, a program crashing is considered poor form

- It's a sign of bad coding skills

“try…except”

Python has a built in way of handling a set of common errors

Your program doesn’t crash, it just lets the user know that something went wrong

```
try:
    x = int(input("Please enter a number: "))
    break
except ValueError:
    print("Oops!  That was no valid number.  Try again...")
```

If the user enters something that cannot be converted to an integer, Python throws an error of type “ValueError”

Syntax and semantics

Syntax:

```
try:
    #code that you want to run, that may crash
except <type of error>:
    #code to run when that error is detected by Python
```

Semantics:

It's pretty much like an "if" statement

- If your code works and no error condition is detected, the "try" section gets executed and the error handling is ignored
- If your code fails - like an "else" statement - anything in the "try" section after the error is not executed and the "except" clause - the error handling code - is executed
    - Anything in the "try" section prior to the error gets executed and not rolled back

Multiple error codes:

What if your code may cause one or more types of errors?

- Just like an “if..elif..else” statement, you can have multiple exceptions in your code

```
try:
        x = int(input("Please enter a number: "))
        y = "UMBC"
        print(x + y + x)

except ValueError:
   print("Oops!  That was no valid number.  Try again...")
except TypeError:
   print("Oops!  That was a list; not a valid number.  Try again...)"
```

Where are these error codes defined?

See the official Python documentation at:
https://docs.python.org/3/library/exceptions.html

Common types:

exception **TypeError**

Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch. This exception may be raised by user code to indicate that an attempted operation on an object is not supported, and is not meant to be. If an object is meant to support a given operation but has not yet provided an implementation, `NotImplementedError` is the proper exception to raise. Passing arguments of the wrong type (e.g. passing a `list` when an `int` is expected) should result in a `TypeError`, but passing arguments with the wrong value (e.g. a number outside expected boundaries) should result in a `ValueError`.

More error types

exception **ValueError**

Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as `IndexError`.

exception **ZeroDivisionError**

Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.