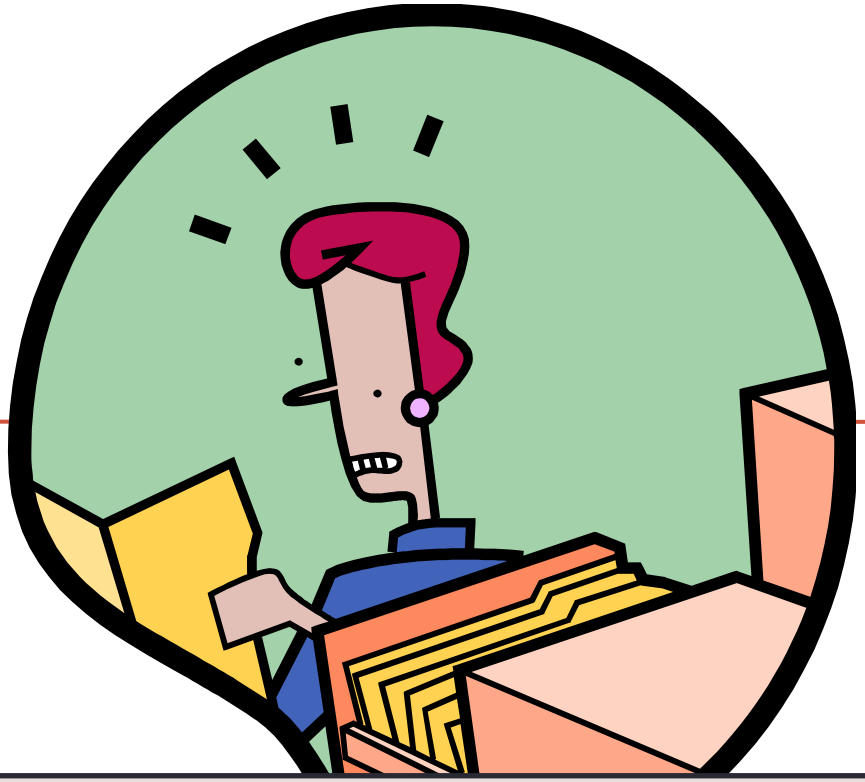
EXCEPTIONS



```
Traceback (most recent call last):
  File "E:/CSCI 135 - CS I/Fall 2019/Examples/AddNums1.py", line 9, in <module>
    num2 = int(sys.argv[2])
ValueError: invalid literal for int() with base 10: '3.5'
```

Outline

- Exceptions
  - An important part of writing defensive code
    - Defending against bad input
    - Handling unexpected events
      - e.g. File is missing
      - e.g. Trying to parse "$56.89" as a float

Adding Two Numbers

- Goal: Defend against all types of bad input
  - Problem 1: Crashes if less than 2 arguments

```
import sys

num1 = int(sys.argv[1])
num2 = int(sys.argv[2])
sum  = num1 + num2
print(str(num1) + " + " + str(num2) + " = " + str(sum))
```

```
% python AddNums1.py 1
Traceback (most recent call last):
  File "E:/CSCI 135 - CS I/Fall 2019/Examples/AddNums1.py", line 9, in <module>
    num2 = int(sys.argv[2])
IndexError: list index out of range
```

Adding Two Numbers

- Goal: Defend against all types of bad input
  - Fix 1: Add conditional to check there are 2 args (plus the arg for the program name)
  - Problem 2: Crashes if passed a non-integer arg

```
import sys

if len(sys.argv) != 3:
    print("AddNums <integer 1> <integer 2>")
else:
    num1 = int(sys.argv[1])
    num2 = int(sys.argv[2])
    sum  = num1 + num2
    print(str(num1) + " + " + str(num2) + " = " + str(sum))
```

```
% python AddNums2.py 2 3.5
Traceback (most recent call last):
  File "E:/CSCI 135 - CS I/Fall 2019/Examples/AddNums1.py", line 9, in <module>
    num2 = int(sys.argv[2])
ValueError: invalid literal for int() with base 10: '3.5'
```

Adding Two Numbers

- How to check for invalid input to `int()`?
  - e.g. 1.0, 192.168.1.4, $1, 123., one

```
import sys

if len(sys.argv) != 3:
    print("AddNums <integer 1> <integer 2>")
else:
    num1 = int(sys.argv[1])
    num2 = int(sys.argv[2])
    sum  = num1 + num2
    print(str(num1) + " + " + str(num2) + " = " + str(sum))
```

Python Exceptions

- When things go wrong:
  - Python raises an exception
  - You get to decide if program can recover or not
    - Rather than always crashing with a runtime error

```
% python AddNums2.py 2 3.5
Traceback (most recent call last):
  File "E:/CSCI 135 - CS I/Fall 2019/Examples/AddNums1.py", line 9, in <module>
    num2 = int(sys.argv[2])
ValueError: invalid literal for int() with base 10: '3.5'
```

try-except Block

```
try:
    # Do some risky things

    # Do some more risky things
except <ErrorNameHere>:
    # Try and recover from the problem
```

If something goes horribly wrong on a line in the `try` block, flow of control immediately jumps to the `except` block.

Add Code to catch all Exceptions

```
import sys

try:
    num1 = int(sys.argv[1])
    num2 = int(sys.argv[2])
    sum  = num1 + num2
    print(str(num1) + " + " + str(num2) + " = " + str(sum))
except BaseException:
    print("Something went wrong!")
print("End of program")
```

```
% python AddNums.py 2 3.5
Something went wrong!
End of program
```

```
% python AddNums.py
Something went wrong!
End of program
```

Not an ideal solution:

How is the user suppose to know what to do differently?

A Better Solution

```
import sys

if len(sys.argv) != 3:
    print("AddNums <integer 1> <integer 2>")
else:
    try:
        num1 = int(sys.argv[1])
        num2 = int(sys.argv[2])
    except ValueError:
        print("Something went wrong!")
    else:
        sum  = num1 + num2
        print(str(num1) + " + " + str(num2) + " = " + str(sum))
print("End of program")
```

Principle 1:
Don't catch exceptions you can handle with logic such as staying in bounds of an array.

Principle 2:
Catch exceptions based on their type, not just generically.

Else, the else Block

- Else block executes if no exception was raised
  - If no exception, runs after try-block
  - If exception occurs, does not run

```
try:
    turnOvenOn()
    x.bake()
except BakingException:
    # report an error
else:
    turnOvenOff()
```

Averaging Numbers

```
import sys

total = 0.0
count = 0

# Check if we need to print out command line help
if len(sys.argv) < 2:
    print("AvgNumsFile <filename>")
else:
    try:
        # Open up the text file for reading
        with open(sys.argv[1], encoding='utf-8') as f:
            contents = f.read()
    except FileNotFoundError:
        print(f"Sorry, the file {sys.argv[1]} does not exist.")
    else:
        numbers = contents.split()
        for num in numbers:
            total += float(num)
            count += 1
        f.close()

        # Print out the final average
        if count != 0:
            print(total / count)
        else:
            print("Cannot divide by 0")
```

New program: reads from filename given by `sys.argv[1]`.

```
% python AvgNumsFile1.py squares.txt
332833.5
```

Trying to Break it

```
import sys

total = 0.0
count = 0

# Check if we need to print out command line help
if len(sys.argv) < 2:
    print("AvgNumsFile <filename>")
else:
    try:
        # Open up the text file for
        with open(sys.argv[1], enco
            contents = f.read()
    except FileNotFoundError:
        print(f"Sorry, the file {sys.argv[1]} does not exist.")
    else:
        numbers = contents.split()
        for num in numbers:
            total += float(num)
            count += 1
        f.close()

        # Print out the final average
        if count != 0:
            print(total / count)
        else:
            print("Cannot divide by 0")
```

```
% python AvgNumsFile1.py noexist.txt
Sorry, the file NoExist.txt does not exist.
```

```
% python AvgNumsFile1.py mobydick.txt
Traceback (most recent call last):
  File "E:/CSCI 135 - CS I/Fall 2019/Examples/AvgNumsFile1.py",
line 24, in <module>
    total += float(num)
ValueError: could not convert string to float: 'Loomings'
```

Multiple except Blocks

```
import sys

total = 0.0
count = 0

# Check if we need to print out command line help
if len(sys.argv) < 2:
    print("AvgNumsFile <filename>")
else:
    try:
        # Open up the text file for reading
        with open(sys.argv[1], encoding='utf-8') as f:
            contents = f.read()
        numbers = contents.split()
        for num in numbers:
            total += float(num)
            count += 1
        f.close()
    except FileNotFoundError:
        print(f"Sorry, the file {sys.argv[1]} does not exist.")
    except ValueError:
        print("Invalid data in file!")
    else:

        # Print out the final average
        if count != 0:
            print(total / count)
        else:
            print("Cannot divide by 0")
```

```
% python AvgNumsFile2.py mobydick.txt
Invalid data in file!
```

Ignoring an Exception

```
import sys

total = 0.0
count = 0

# Check if we need to print out command line help
if len(sys.argv) < 2:
    print("AvgNumsFile <filename>")
else:
    try:
        # Open up the text file for reading
        with open(sys.argv[1], encoding='utf-8') as f:
            contents = f.read()
    except FileNotFoundError:
        print(f"Sorry, the file {sys.argv[1]} does
    else:
        numbers = contents.split()
        for num in numbers:
            try:
                total += float(num)
            except ValueError:
                pass
            else:
                #print(num)
                count += 1
        f.close()

        # Print out the final average
        if count != 0:
            print(total / count)
        else:
            print("Cannot divide by 0")
```

```
% python AvgNumsFile3.py TaleOfTwoCities.txt
799.5714285714286
```

User Defined Exceptions

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

Raising Exceptions

- How do exceptions start their life?
  - Somebody raises them:

```
raise NoCaffeineException()
```

  - Whoever is using the method should catch (“try-except”)
  - Or else that method can raise it
  - What if nobody catches it?
    - Causes compile error if exception is not caught

Python Exception Class Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

Finally, the finally Block

- Finally block executes no matter what
  - If no exception, runs after try- or else-block
  - If exception occurs, runs after except-block
  - Useful for doing cleanup that is always needed

```
try
   turnOvenOn()
   x.bake()
except BakingException:
   print(“Baking didn’t work”)
finally:
   turnOvenOff()
```

Summary

- Exceptions
  - An important part of writing defensive code
    - Defending against bad input
    - Handling unexpected events
      - e.g. File is missing
      - e.g. Trying to parse "$56.89" as a float

Try It!

- This is not really an extra credit activity, but what I want you to do is download the examples from today along with the text files. Particularly with the AvgNumsFile.java code, try running it with both text files and see what happens. You can even create your own text file with numbers in it and use it on the command line instead. For example, you could put the numbers:

1

2

3

- in a file and see if the program averages them out to 2.0 correctly.