Intro to Coding with Python– Handling Exceptions

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

- Little algorithm practice
- Handling exceptions

Algorithm Practice

- Open the sorting demo on repl.it
- Add your code at the bottom of main()

- Use the time module to compare sort times on the three sorting algorithms we looked at last class
  - Ex.

```
# Record start time
start_time = time.time()
# Run the thing we want to time
sum = 0
for i in range(1000):
  sum += i

# Record end time
end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
```

- Which algorithm is fastest? Is that what you expected?

Lecture 10: some problems are obvious

this is called
an **`Exception`**

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Lecture 10: some problems are obvious

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

the kind of error gives you a **clue** about what the problem is

Lecture 10: some problems are obvious

it also tells you **where** the problem is
(but be careful!)

Python 3.6.5 Shell

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Discussion

But there's a drawback to when your program throws an **`Exception`**…

An example

What happens if the user enters a **negative** number?

```
1   import math
2
3   def main():
4
5     x = int(input("Enter an integer greater than 0: "))
6
7     print("The log is:", math.log(x))
8
9     print("Have a nice day!")
10
11  if __name__ == "__main__":
12    main()
```

An example

What happens if the user enters a **string**?

```python
import math

def main():

  x = int(input("Enter an integer greater than 0: "))

  if x < 0:
    print("Cannot take the log of a negative number.")
  else:
    print("The log is:", math.log(x))

  print("Have a nice day!")

if __name__ == "__main__":
  main()
```

The `try...except` block

- There are some cases where avoiding an **`Exception`** isn't possible
- In this case, we want tell Python:
  - what we **want** to happen (what to **`try`**)
  - how to **handle** it if things go wrong (**`except`**)

An example

```
1   import math
2
3   def main():
4
5     try:
6       x = int(input("Enter an integer greater than 0: "))
7       print("The log is:", math.log(x))
8
9     except:
10      print("Sorry, that is not a valid input.")
11
12    print("Have a nice day!")
13
14  if __name__ == "__main__":
15    main()
```

Okay python:
try to do this

An example

```python
import math

def main():

  try:
    x = int(input("Enter an integer greater than 0: "))
    print("The log is:", math.log(x))

  except:
    print("Sorry, that is not a valid input.")

  print("Have a nice day!")

if __name__ == "__main__":
  main()
```

except if you can't; then do this instead

Your turn!

Given: a (brittle) solution to A2: Clunky Calculator

Objective: find any places that might throw **`Exceptions`**, and handle them so the program doesn't crash!

Takeaways

- Even if you can't avoid all errors, you can design your program to **fail gracefully**
- You can handle multiple different kinds of **`Exceptions`**, and you can handle them differently
- Think about **edge cases** to provide specific feedback about what went wrong