

# 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.

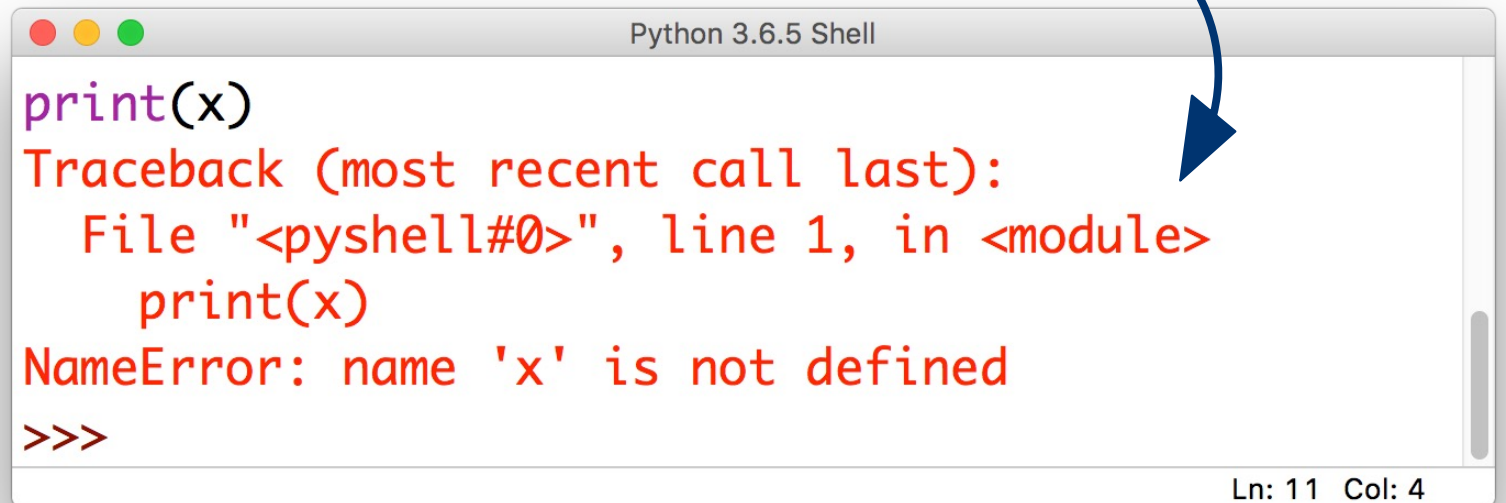
```
# Record start time
start_time = time.time()
# Run the thing we want to time
sum = 0
for i in range(1000):
    sum += i

# 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**

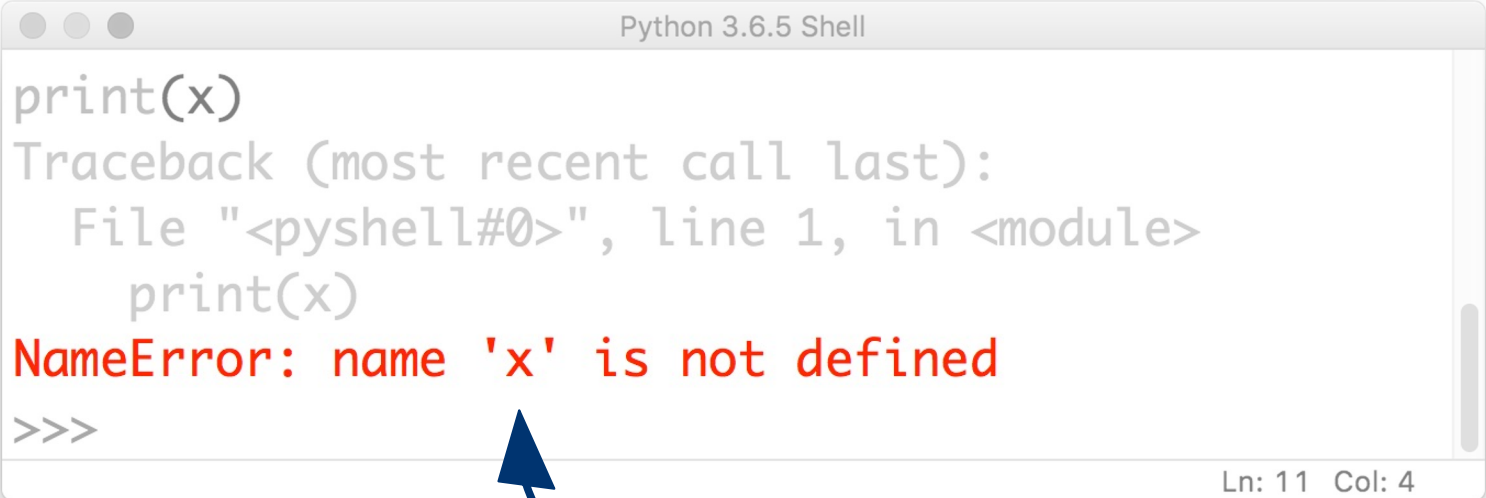
A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text: 

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

 A blue arrow points from the text "this is called an Exception" to the "NameError" line. The bottom right corner of the window shows "Ln: 11 Col: 4".

```
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

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three small circles on the left and the text "Python 3.6.5 Shell" on the right. The main area contains the following text: 

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

 The error message "NameError: name 'x' is not defined" is highlighted in red. At the bottom right of the window, it says "Ln: 11 Col: 4".

```
Python 3.6.5 Shell

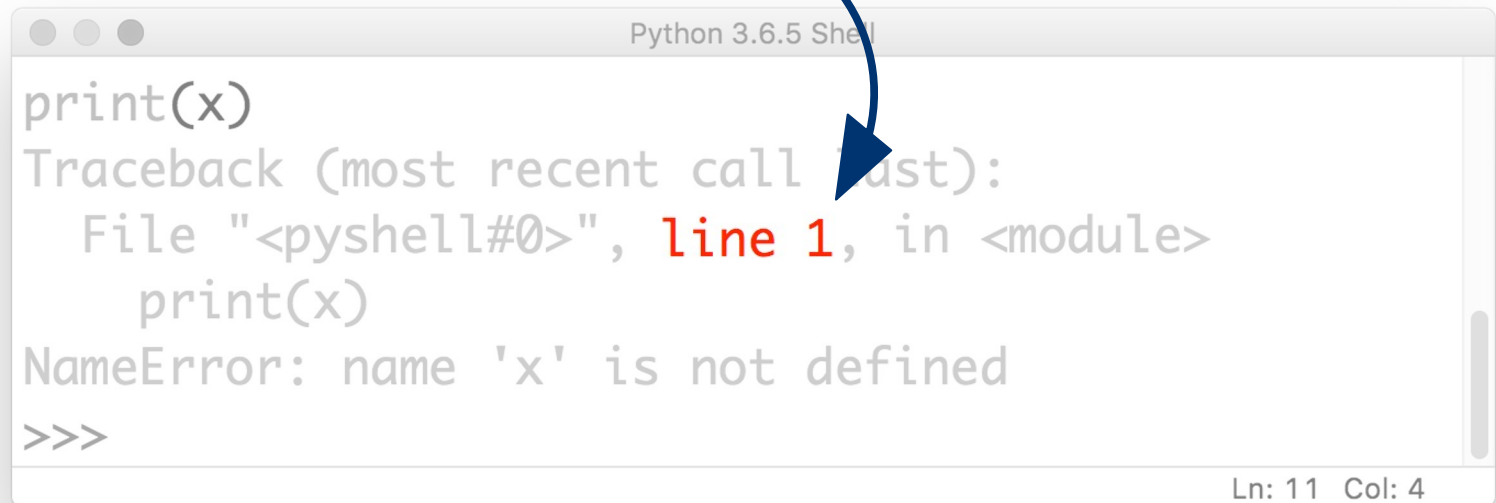
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

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
>>>
```

The image shows a screenshot of a Python 3.6.5 Shell window. The window title is "Python 3.6.5 Shell". The content of the window shows a Python script that has just executed the command `print(x)`, which has resulted in a `NameError: name 'x' is not defined`. A traceback is displayed above the error message, showing the file "`<pyshell#0>`", line 1, in `<module>`. A blue arrow points from the text "(but be careful!)" to the word "line" in the traceback, highlighting the location of the error.


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

```
1 import math
2
3 def main():
4
5     x = int(input("Enter an integer greater than 0: "))
6
7     if x < 0:
8         print("Cannot take the log of a negative number.")
9     else:
10        print("The log is:", math.log(x))
11
12    print("Have a nice day!")
13
14 if __name__ == "__main__":
15    main()
```

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




# 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

```
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()
```

**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