CS177 Python Programming

Recitation 8 – Debugging
Announcements

• Project #2 starts this Friday
Roadmap

• What are computer bugs?
• When are computer bugs discovered?
  – Compile Time and Runtime
• What kind of bugs are discovered?
  – Syntax, Arithmetic and Logic Errors
• Are there tools to help us find bugs?
  – Print statements and Python debugger
Debugging

Early computers used vacuum tubes. The tubes would get hot and attracted moths. A moth was zapped and interfered with the circuitry. The bug had to be removed to fix the computer. Some say this is how the word “debugging” came into use.
Debugging

• What is a computer bug?
  – A computer bug is a problem that causes a computer to produce an incorrect or unexpected result.
Debugging

• Computer bugs can manifest themselves at different phases of the program execution including:
  – Compile Time
  – Runtime
When Are Bugs Discovered?

Compile Time, Load Time, & Runtime Bugs
Compile Time Bug

• Compile time bugs show up when the source code is converted into computer code
• A common compile time error is:
  – Syntax Error
• A syntax error means that the source code does not meet the source code specification. For example:
  – Missing a ‘:’ at the end of you def statement
Compile Time Bug Example

```python
>>> def t2()
SyntaxError: invalid syntax
```

- Notice the missing `:`
- When you run this statement, Python immediately knows that it is invalid code.
Load Time Bug

• Load time bugs, in the context of Python, often have to do with the libraries that are imported
  – The permissions may not be set correctly on an imported library file
  – The library file may not be found
Load Time Bug Example

>>> import foo
ImportError: No module named foo

• In this case a library named foo does not exist
Runtime Bug

• Runtime bugs show up when the computer code is executed

• A common runtime error is:
  – NameError

• A name error means that a function or variable was used that wasn’t defined
Runtime Bug Example

def t1():
    print(a)

>>> t1()

• Notice that the variable 'a' is not defined
• When you save the file, Python does not report an error, but when you call the function...
Runtime Bug Example

def t1():
    print(a)

>>> t1()
NameError: global name 'a' is not defined

• The NameError is produced when the t1 function is called
What are Some Common Bugs?

Syntax, Arithmetic, and Logic Errors
Syntax Bugs

• Syntax errors are often discovered by Python at compile time but not always

• Likely you have encountered many of these:
  – Incorrect indentation
  – Missing elements (like ':')
Syntax Bug

• Incorrect indentation:

```python
def t1():
t = 1

Invalid syntax
```

Python catches this syntax error
Syntax Bug

• Incorrect indentation:

```python
>>> def t1()
SyntaxError: invalid syntax
```

Missing ':'
Python will catch this syntax error
Arithmetic Bugs

• We will only focus on one, but a few more exist

• One important arithmetic bug is a divide-by-zero error
  – By definition of division, you can't divide by zero
Arithmetic Bug

• Division by zero:

```python
>>> 4/0
ZeroDivisionError: int division or modulo by zero
```

you can't divide by 0

Python will catch this arithmetic error
Logic Bugs

• Logic bugs are usually not caught automatically by the computer like Syntax Errors or Name Errors.
• The bug may be subtle and manifest itself in peculiar ways.
• Usually takes human source code analysis to track down the bug
Logic Bug

• Infinite loop:

```python
>>> i = 0
>>> while (i<5):
    i = 1
```

- i is not getting incremented
- Python will not catch this
Logic Bug

• Off-by-one error (or fencepost error):

```python
>>> x = [21, 22, 23, 24]
>>> i = 0
>>> while i <= len(x):
    s = s + x[i]
    i = i + 1
IndexError: list index out of range
```

- x[4] does not exist
- Python catches this logic error
Logic Bug

• Off-by-one error:

```python
>>> x = [21, 22, 23, 24]
>>> i = 1
>>> s = 0
>>> while i < len(x):
    s = s + x[i]
    i = i + 1
```

s will not be 90, it will be 69
Python will not catch this
Are there tools to help us find bugs?

Print Statements and Python Debugger
Print Statements

• Strategically places print() statements can be placed in the source code to verify values
• Advantage: Using print statements (or equivalents) to debug works in every language, no language specific tool must be learned
• Disadvantage: Not everything is printable
Using Print Statements

• Verify input and output

```python
def sort3(x, y, z):
    print("Input: x=", x, "y=", y, "z=", z)
    r = sorted([x, y, z])
    print("Output:", r)

>>> sort3(8, 11, 3)
Input: x= 8 y= 11 z= 3
Output: [3, 8, 11]
```
Using Print Statements

• Print intermediate live values

```python
def t():
    s = 0
    for i in range(3):
        ns = s + i
        print(ns, "=" , s, "+" , i)
        s = ns

>>> t()
0 = 0 + 0
1 = 0 + 1
3 = 1 + 2
```
Python Debugger

• Many programming languages have debuggers available

• A debugger lets you analyze the program state after each statement
  – Called stepping
Python Debugger

• To launch the Python debugger from IDLE:
  – From the IDLE command window choose the menu: Debug->Debugger
  – Your command window will show [DEBUG ON]
  – Then run commands as normal and you should see the debugger window...
Using Print Statements

def sort(r):
    print(t.py:11: t0)
    print

def t():
    s = 0
    for i in range(1, 10):
        ns = s + i
        print(ns, "=", s, "+", i)
        s = ns
Python Debugger

• Options
  – Stack: Current running function
  – Source: Show me in the source what statement is currently running
  Locals: What are the values of the local variables
  – Global: What are the values of global variables
ANY QUESTIONS?