Python Applications

debugging code, solving problems

CS101 Lecture #14

2016-11-16

Administrivia

Administrivia

- Homework #6 is due Today, Nov. 16.
- Prof. Li Er-Ping and Prof. Philip Krein will be out of town tomorrow, so I will (very happily) show up – *again* – on tomorrow's ENG100 course.
- Any questions for the lab sessions? suggestions or critiques?

Warmup Quiz

Question #1

```
a = [ 'alpha','beta','gamma' ]
b = [ 2,3,4 ]
for i,j in enumerate( a ):
    print( i,j )
```

Which of the following answers is a possible line of output of the above code?

A `4 gamma`

B `2 gamma`

C `gamma 4`

D `gamma 2`

(*What does the English word "enumerate" mean, and the function definition in python?*)

Question #1

```
a = [ 'alpha','beta','gamma' ]
b = [ 2,3,4 ]
for i,j in enumerate( a ):
    print( i,j )
```

Which of the following answers is a possible line of output of the above code?

A `4 gamma`

B `2 gamma` ⋆

C `gamma 4`

D `gamma 2`

Question #2

```
a = [ 'alpha','beta','gamma' ]
b = [ 2,3,4 ]
for i,j in zip( a,b ):
    print( i,j )
```

Which of the following answers is a possible line of output of the above code?

A `4 gamma`

B `2 gamma`

C `gamma 4`

D `gamma 2`

Question #2

```
a = [ 'alpha','beta','gamma' ]
b = [ 2,3,4 ]
for i,j in zip( a,b ):
    print( i,j )
```

Which of the following answers is a possible line of output of the above code?

A `4 gamma`

B `2 gamma`

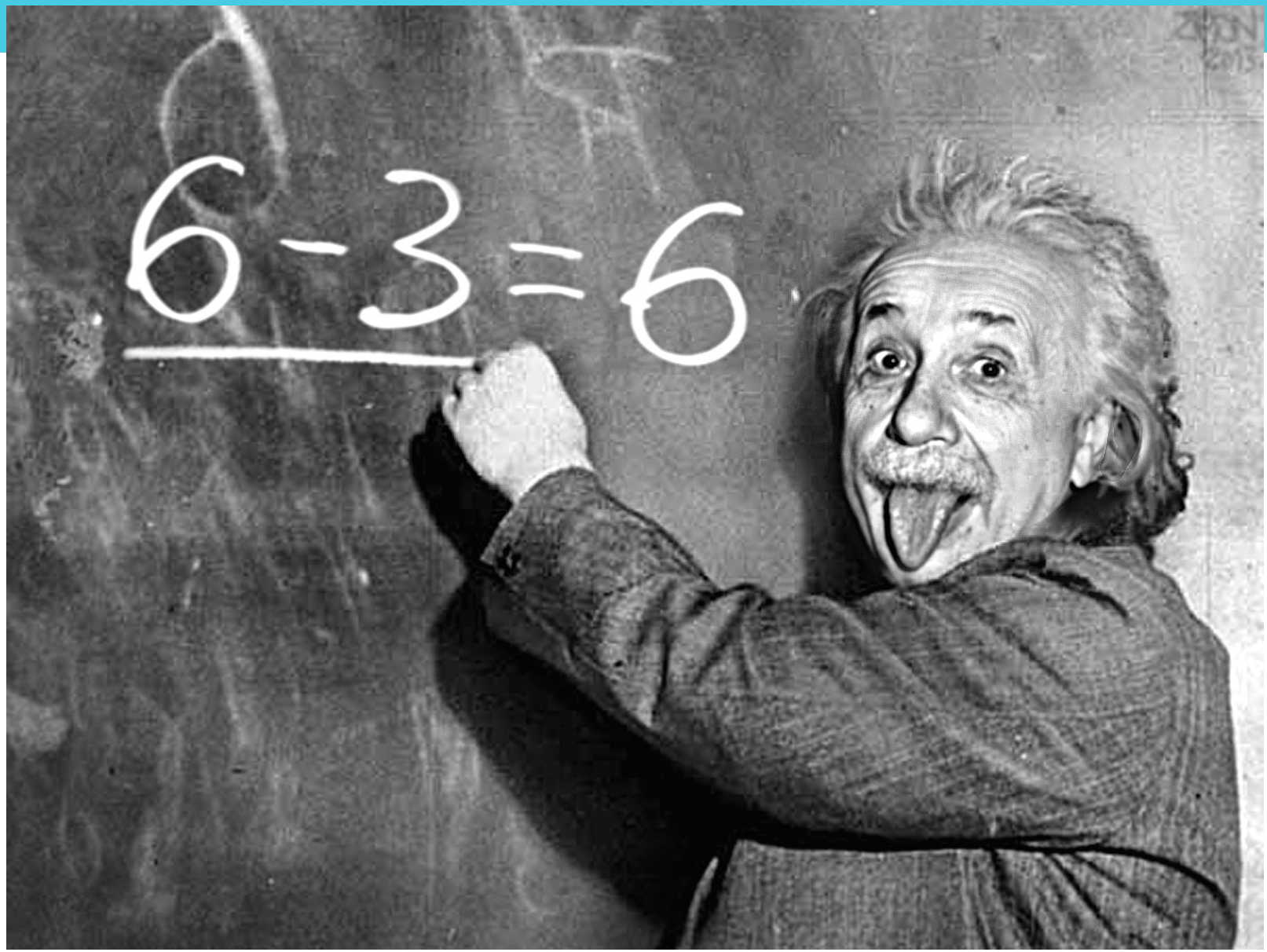
C `gamma 4` ⋆

D `gamma 2`

(*What if b = [2,3,4,5]?*)

When Things Go Wrong

My code doesn't work.



errors are cute

My experience with errors:

- You never learn to program correctly without making errors
- Some errors reveal the discrepancies between your way of thinking and the machine's logic
- A genuine intuition is always error-free – you make errors when you start to *think*.

Debugging

How do I know it isn't working?

- What do I expect it to do?
- What is my code doing instead?
- How to identify the source of error?

  ==> Debugging

Debugging

- A few working definitions:
  - **Exceptions**— unusual behaviors occurred in the execution of a program; caught by a try:{...}except e:{...} syntax. Most languages (Python, C++, Java) have this.

Debugging

- A few working definitions:
  - **Exceptions**— unusual behaviors occurred in the execution of a program; caught by a try:{...}except e:{...} syntax. Most languages (Python, C++, Java) have this.
  - **Errors**— exceptions that cause the program to be unrunnable

Debugging

- A few working definitions:
  - **Exceptions**— unusual behaviors occurred in the execution of a program; caught by a try:{...}except e:{...} syntax. Most languages (Python, C++, Java) have this.
  - **Errors**— exceptions that cause the program to be unrunnable
  - **Traceback**—listing of function calls on the stack at the time the exception (error) arises

Debugging

- A few working definitions:
  - **Exceptions**— unusual behaviors occurred in the execution of a program; caught by a try:{...}except e:{...} syntax. Most languages (Python, C++, Java) have this.
  - **Errors**— exceptions that cause the program to be unrunnable
  - **Traceback**—listing of function calls on the stack at the time the exception (error) arises
  - **Bugs**—errors and exceptions, but also miswritten, ambiguous, or incorrect code which in fact runs but does not advertise its miscreancy

Common exceptions

- `SyntaxError`
- `NameError`
- `TypeError`
- `ValueError`
- `IOError`
- `IndexError`
- `KeyError`
- `ZeroDivisionError`
- `IndentationError`
- `Exception:` subsumes all above and many others

Common exceptions

- `SyntaxError`—check missing colons or parentheses
- `NameError`—check for typos, function definitions
- `TypeError`—check variable types (coerce if necessary)
- `ValueError`—check function parameters
- `IOError`—check that files exist

Common exceptions

- `IndexError`—don't reference nonexistent list elements
- `KeyError`—similar to an IndexError, but for dictionaries
- `ZeroDivisionError`—three guesses...
- `IndentationError`—check that spaces and tabs aren't mixed
- `Exception`—generic error category

Comprehension Question

```
# calculate squares
d = list(range(10))
while i < 10:
    d[i] = d[i] ** 2.0
    i += 1
```

Which error would this code produce?

A SyntaxError
B IndexError
C ValueError
D NameError

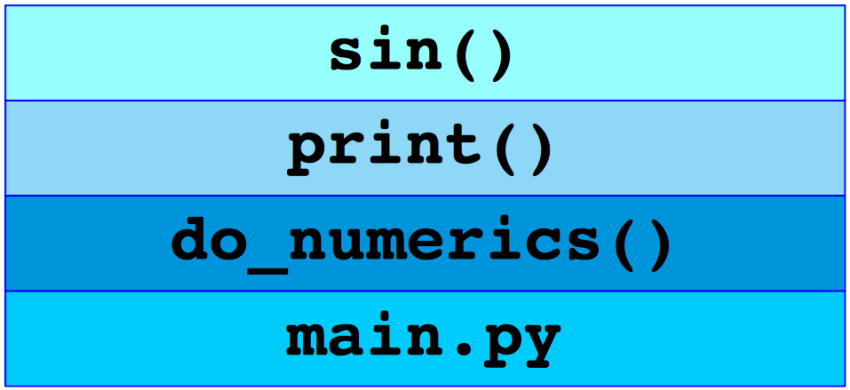
Comprehension Question

Which of the following would produce `TypeError`?

A
```
'2' + 2
```
B
```
2 / 0
```
C
```
2e8 + (1+0j)
```
D
```
'2' * 2
```

Program stack

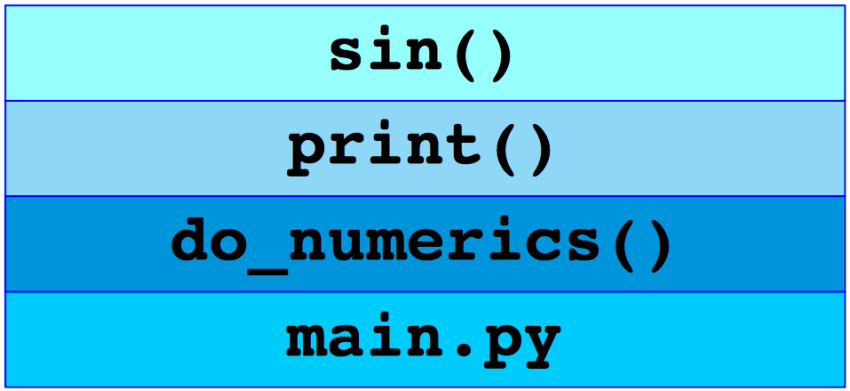
```
# in file `main.py`
def do_numerics():
    print(sin(5.0))

do_numerics()
```



Program stack trace

```
Traceback (most recent call last):
    File "main.py", line 7, in <module>
      do_numerics()
    File "main.py", line 4, in do_numerics
      print(sin(5.0))
NameError: name 'sin' is not defined
```

- Read these from end to beginning:
- `sin` in `do_numerics` in file `main.py`



Debugging

Debugging

Brian Kernighan:
"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Debugging example

This code should find all lines in a file whose first two letters are the same:

```
for line in open( "words.txt" ):
    line = line.strip()
    if len( line ) >= 2:
        a,b = line[1:3]
    if a == b:
        print( line )
```

Debugging example

This code should find all lines in a file whose first two letters are the same:

```
for line in open( "words.txt" ):
    line = line.strip()
    if len( line ) >= 2:
        print( line )
        a,b = line[1:3]
        print( a,b )
    if a == b:
        print( line )
```

Debugging example

This code should find all lines in a file whose first two letters are the same:

```
for line in open( "words.txt" ):
    line = line.strip()
    if len( line ) >= 2:
        a,b = line[0:2]
    if a == b:
        print( line )
```

Debugging example

This code should find all lines in a file whose first two letters are the same:

```
for line in open( "words.txt" ):
    line = line.strip()
    if len( line ) >= 2:
        a,b = line[0:2]
    if a == b:
        print( line )
```

Debugging strategies

- Start early.
- Read the problem statement carefully.
- Add print statements.
- Chart the flow of the program.
- Break the program down into functions.
- Document functions before writing them.
- Show it to someone else. People have blindspots.
- Make no assumptions! If your thinking is not precise, your code will not be precise.
- Start over from scratch. Take a fresh look at the problem.

Debugging strategies

- Start early.
- Read the problem statement carefully.
- Add print statements.
- Chart the flow of the program.
- Break the program down into functions.
- Document functions before writing them.
- Show it to someone else. Everyone has a blind spots someone else can easily see.
- Make no assumptions! If your thinking is not precise, your code will not be precise.
- Start over from scratch. Take a fresh look at the problem.

Style

Style

- What makes a good Python code?

```
import this
```

Style

- Use descriptive variable names.
- Why do we write comments?
- For the person who next looks at the code!

```
x_vals = [0,0.1,0.2,0.3,0.4]  # meters
faraday = 96485.3328959  # coulombs,
                         # electric charge
```

Style

- Document your code!
- Every function should have a docstring.

```
def warning( msg ):
    '''Display a warning message.'''
    print( 'Warning:  %s'%msg)
```

- Docstrings explain what the function does and what its parameters are.
- They always are triple-quoted strings on the first line of the function block.

```
help(warning)
```

Style

- Use functions to structure code (also called code refactory).
- This makes code more readable (and debuggable!).
- A `main` function lets you organize your program's logic succinctly.

Style

- A `main` function lets you organize your program's logic succinctly.
- We have a special way of writing these so that we can load our code as a module or execute it alone.
- A module's '__name__' (an environment variable) is set to '__main__' when read from standard input, a script, or from an interactive prompt, but not when it is *i*mported

```
def main():
    # your code here

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

Course outline

Course Summary:

- Python basics:
  - operators, expressions, data types, data structures
- Python applications:
  - workflow, I/O, ⋆debugging
- Scientific Python:
  - plotting, calculating, modeling, simulation, optimization
- MATLAB:
  - recap of the above

⋆(you are here)