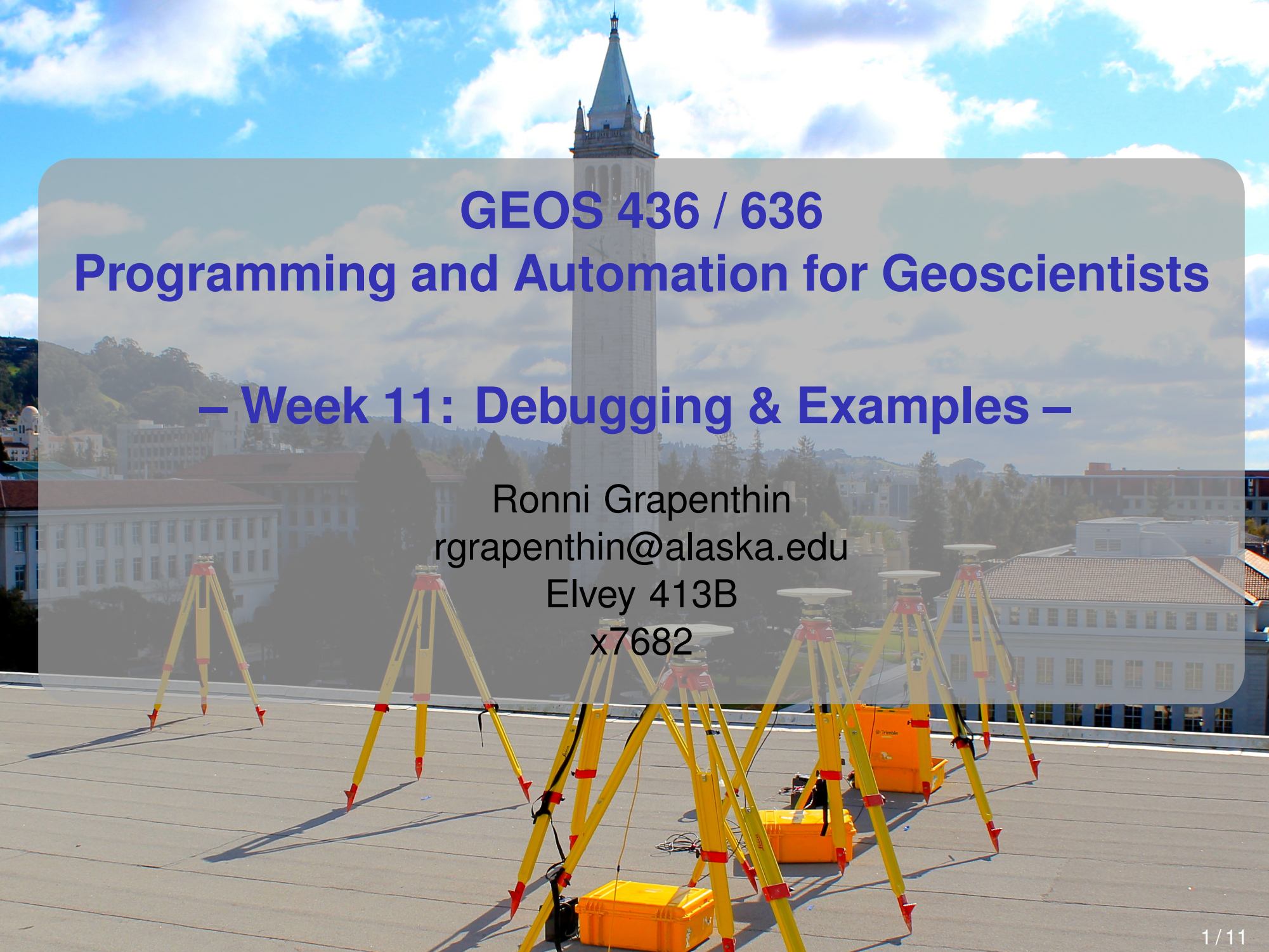

GEOS 436 / 636
Programming and Automation for Geoscientists

– Week 11: Debugging & Examples –

Ronni Grapenthin
rgrapenthin@alaska.edu
Elvey 413B
x7682

The code doesn’t work!

How do we fix it?

Review: Software Development Cycle

1. Design
2. Coding
3. Test
4. **Debugging**
5. go back to 1, 2, 3, . . .

This cycle is common to all programming strategies, independent of how involved each stage is.

What is “Debugging”?

Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

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.

Brian Kernighan

Simple Truths about Bugs and Debugging . . .

- Bugs are static – they won't run away.
- Often, the problem is simple.
- You created the code, so you made the bug! It's nobody else's fault - get to it!
- Debugging is a great way to learn being self-critical. Good luck!
- Be critical – did you mean '<', '<=', '>', '>='?
- Don't panic – be systematic!
- Sleep, go for a walk, come back later.

Debugging Styles / Helpful Tools

- **echoing:** place print statements at useful points in a program (function entry, exit)
- **unit testing:** write calls to particular function, throw artificial values at it, automate this to test units with every change of the code
- **exception handling:** in high level languages: sources of mistakes easier to spot
- **online debuggers:** for our purposes usually not necessary, useful if you want to step through your code, or for memory problems
- **version control:** have a tool keep track of changes you make; roll back or comparisons to bug-free code is simple (not covered here)

Debugging Styles: Echoing vs. Online Debuggers

. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.

From: Brian Kernighan, Rob Pike "The Practice of Programming"

Debugging Styles: Echoing

- at the simplest: supplement your code with `print()` or `echo` statements where necessary
- more sophisticated:
  - write a function that displays text only if a global DEBUG flag is set.
  - call this function whenever necessary: entry or exit of function, display variable values before / after critical operations, to follow the program flow, ...
  - turn DEBUG off when everything works

Debugging Styles: Unit Testing

- at the simplest: write a script that calls your functions with artificial values and compares results to expected outcome
- execute whenever code changes are made
- helps to detect errors due to changes in functions immediately
- possible for any language (fancy frameworks exist for some languages)

Debugging Styles: Unit Testing

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

https://docs.python.org/

Debugging Styles: Exception Handling

The motto: Keep the program running in cases where things go wrong (in recoverable ways).

```
>>> def this_fails():
...     x = 1/0
...
>>> try:
...     this_fails()
... except ZeroDivisionError as err:
...     print('Handling run-time error:', err)
...
Handling run-time error: division by zero
```

https://docs.python.org/