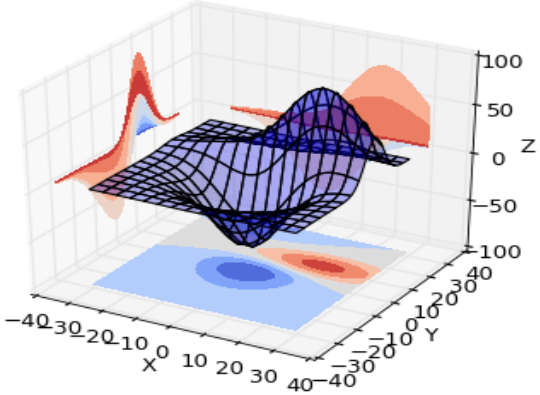
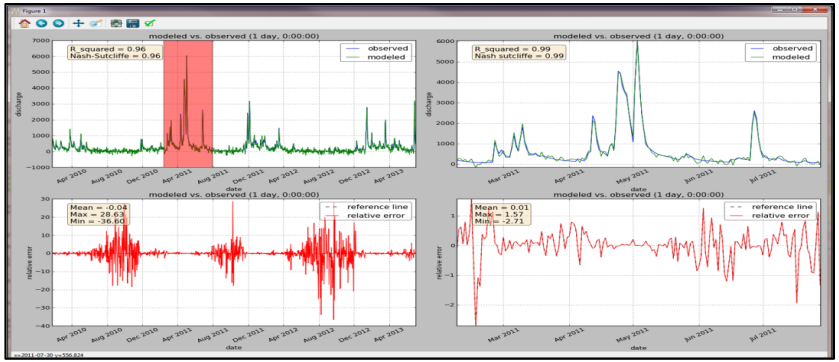
Scientific Computing Group
Programming with Python:
Functions, Read Measurements Project



python™

```
# Write Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```



Jeremiah Lant, Hydrologist
USGS Kentucky Water Science Center
jlant@usgs.gov

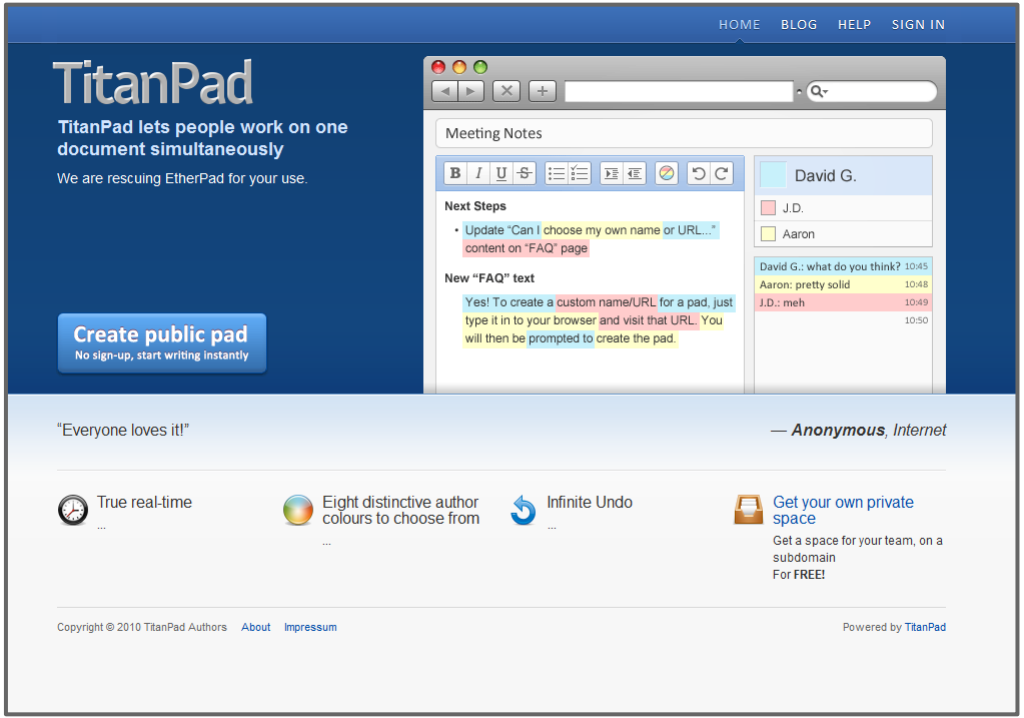
Last Meeting

- Reviewed aliasing with functions
- Debugged a program using the python debugger in the Spyder IDE
- Showed a few ways of writing tests for a function

Today’s Objectives

- Discuss collaborative meeting notes using TitanPad
- Discuss markdown
- Learn how to **clone** the scientific-computing-group repository on GitHub and **pull** latest changes.
- Refactor the read_measurements.py file in the “read measurements project” with functions.

Collaborative Meeting Notes



www.titanpad.com

Collaborative Meeting Notes Template

meeting notes template

```
Topic
=====

*****
**Objectives:**

1.

2.

3.


*****
**Notes:**


*****
**Examples:**


*****
**Questions / Comments**


*****
**What you learned and/or would have liked to have learned more about the topic?**


*****
**References:**
```

meeting notes after meeting

```
Python - aliasing, debugging, testing
=====================================

*****
**Objectives:**

1. Discuss and review aliasing.

2. Debug a program using the Python debugger in the Spyder IDE.

3. Learn a few ways to test a function.


*****
**Notes:**

* Lists can be mutated in functions

* A **debugger** prevents you from having to change / alter your code with print statements

* Debugging keywords and keyboard shortcuts in [Spyder IDE]:

    - [Breakpoint] - an intentional stopping or pausing at a particular place in a program in order to debug the program;

    - Debug file (`Ctrl+F5`)

    - Run current line (`Ctrl+F10`)

    - Step into function or method of current line (`Ctrl+F11`)

    - Run until current function or method returns (`Ctrl+Shift+F11`)

    - Continue execution until next breakpoint (`Ctrl+F12`)

    - Exit debug mode (`Ctrl+Shift+F12`)

* A **function** *without* a return statement stills returns `None`

* **Testing functions** - A few methods:

    1. use `print()` function or `print` statements in main program and function to manually check validity

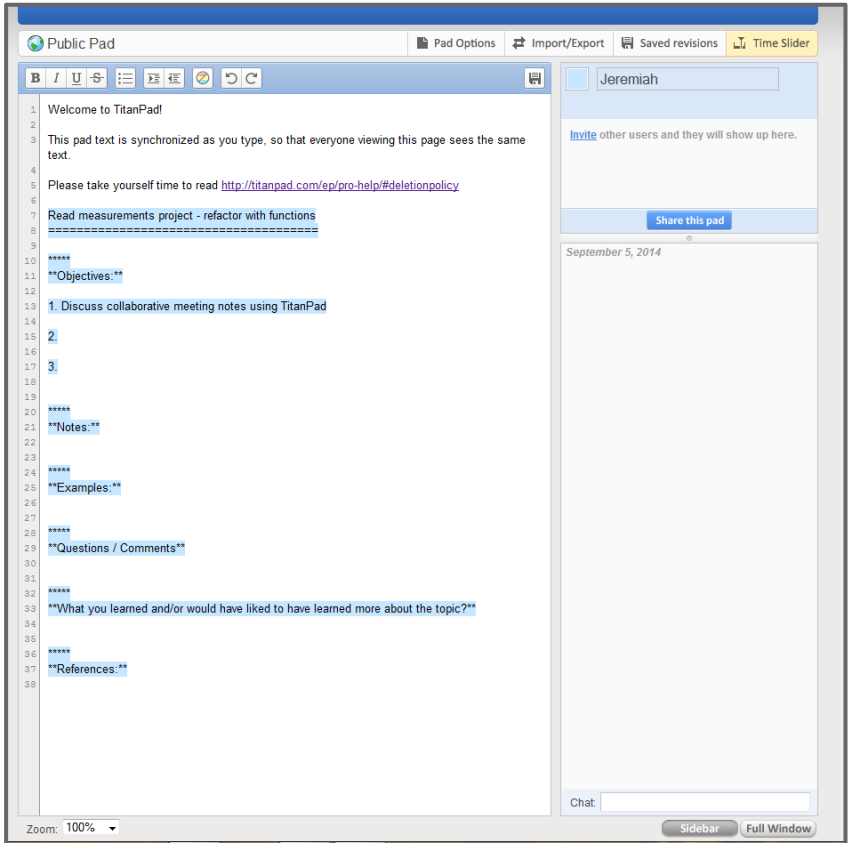
    2. use `assert` statements in main program

        - `assert actual == expected, "some error message"`

    3. write a **test function** which contains an assert statement; can have many test functions for a single function
```

Collaborative Meeting Notes Template

- Today's meeting notes: https://titanpad.com/gzJPoWrYY8



Markdown

- Markdown - a plain text formatting syntax that can be used to format readme files
  - created by John Gruber and Aaron Swartz
- John Gruber - “to write using an easy-to-read, easy-to-write plain text format, and optionally convert it to structurally valid XHTML (or HTML)” (http://daringfireball.net/projects/markdown/)

Example [edit]

The following shows text using Markdown syntax on the left, the corresponding HTML produced by a Markdown processor in the center, and the text viewed in a browser on the right.

```
Heading
=======

Sub-heading
-----------

h3. Traditional html title

Paragraphs are separated
by a blank line.

Let 2 spaces at the end of a line to do a
line break

Text attributes *italic*,
**bold**, `monospace`.

A [link](http://example.com).
<<<   No space between ] and (  >>>

Shopping list:

  * apples
  * oranges
  * pears

Numbered list:

  1. apples
  2. oranges
  3. pears

The rain---not the reign---in
Spain.
```

```
<h1>Heading</h1>

<h2>Sub-heading</h2>

<h3>Traditional html title</h3>

<p>Paragraphs are separated
by a blank line.</p>

<p>Let 2 spaces at the end of a line to do a<br />
line break</p>

<p>Text attributes <em>italic</em>,
<strong>bold</strong>,
<code>monospace</code>.</p>

<p>A <a href="http://example.com">link</a>.</p>

<p>Shopping list:</p>

<ul>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ul>

<p>Numbered list:</p>

<ol>
<li>apples</li>
<li>oranges</li>
<li>pears</li>
</ol>

<p>The rain&mdash;not the
reign&mdash;in Spain.</p>
```

Heading

Sub-heading

Traditional html title

Paragraphs are separated by a blank line.

Let 2 spaces at the end of a line to do a line break

Text attributes *italic*, **bold**, `monospace`.

A link.

Shopping list:

- apples
- oranges
- pears

Numbered list:

1. apples
2. oranges
3. pears

The rain—not the reign—in Spain.

http://en.wikipedia.org/wiki/Markdown

Markdown - online editors

Export ▾ Help ▾ Login Signup for more features Markable β

```
Python - alias, debugging, testing
==================================

*****
**Objectives:**

1. Discuss and review aliasing.

2. Debug a program using the Python debugger in the Spyder IDE.

3. Learn a few ways to test a function.


*****
**Notes:**

* Lists can be mutated in functions

* A **debugger** prevents you from having to change / alter your code with print statements

* Debugging keywords and keyboard shortcuts in [Spyder IDE]:

    - [Breakpoint] - an intentional stopping or pausing at a particular place in a program in order to debug the program; set a breakpoint on a particular line using `F12` or double clicking line number

    - Debug file (`Ctrl+F5`)

    - Run current line (`Ctrl+F10`)

    - Step into function or method of current line (`Ctrl+F11`)

    - Run until current function or method returns (`Ctrl+Shift+F11`)

    - Continue execution until next breakpoint (`Ctrl+F12`)

    - Exit debug mode (`Ctrl+Shift+F12`)

* A **function** *without* a return statement stills returns `None`

* **Testing functions** - A few methods:

    1. use `print()` function or `print` statements in main program and function to manually check validity

    2. use `assert` statements in main program

        - `assert actual == expected, "some error message"`

    3. write a **test function** which contains an assert statement; can have many test functions for a single function

*****
**Examples:**

**Aliasing in functions** - a list can be mutated in a function

    def appender(x_list, number):
        x_list.append(number)

        return None
```

Python - alias, debugging, testing

Objectives:

1. Discuss and review aliasing.
2. Debug a program using the Python debugger in the Spyder IDE.
3. Learn a few ways to test a function.

Notes:

- Lists can be mutated in functions
- A **debugger** prevents you from having to change / alter your code with print statements
- Debugging keywords and keyboard shortcuts in Spyder IDE:
  - Breakpoint - an intentional stopping or pausing at a particular place in a program in order to debug the program; set a breakpoint on a particular line using `F12` or double clicking line number
  - Debug file (`Ctrl+F5`)
  - Run current line (`Ctrl+F10`)
  - Step into function or method of current line (`Ctrl+F11`)
  - Run until current function or method returns (`Ctrl+Shift+F11`)
  - Continue execution until next breakpoint (`Ctrl+F12`)
  - Exit debug mode (`Ctrl+Shift+F12`)
- A **function** *without* a return statement stills returns `None`
- **Testing functions** - A few methods:
  1. use `print()` function or `print` statements in main program and function to manually check validity
  2. use `assert` statements in main program
     - `assert actual == expected, "some error message"`
  3. write a **test function** which contains an assert statement; can have many test functions for a single function

http://markable.in/editor/

Markdown - online editors

Dillinger Local Files Dropbox GitHub Google Drive Github Markdown Select Your Theme Utilities

Untitled Document Words: 219 Star 3,008

```
Dillinger
========

Dillinger is a cloud-enabled HTML5 Markdown editor.

  - Type some Markdown text in the left window
  - See the HTML in the right
  - Magic

Markdown is a lightweight markup language based on the formatting conventions that people naturally use in email.  As [John
Gruber] writes on the [Markdown site] [1]:

> The overriding design goal for Markdown's
> formatting syntax is to make it as readable
> as possible. The idea is that a
> Markdown-formatted document should be
> publishable as-is, as plain text, without
> looking like it's been marked up with tags
> or formatting instructions.

This text you see here is *actually* written in Markdown! To get a feel for Markdown's syntax, type some text into the left
window and watch the results in the right.

Version
----

2.0

Tech
-----------

Dillinger uses a number of open source projects to work properly:

* [Ace Editor] - awesome web-based text editor
* [Marked] - a super fast port of Markdown to JavaScript
* [Twitter Bootstrap] - great UI boilerplate for modern web apps
* [node.js] - evented I/O for the backend
* [Express] - fast node.js network app framework [@tjholowaychuk]
* [keymaster.js] - awesome keyboard handler lib by [@thomasfuchs]
* [jQuery] - duh

Installation
--------------

```sh
git clone [git-repo-url] dillinger
cd dillinger
npm i -d
mkdir -p public/files/{md,html,pdf}
```


Configure Plugins. Instructions in following README.md files

* plugins/dropbox/README.md
* plugins/github/README.md
* plugins/googledrive/README.md

```sh
node app
```
```

Dillinger

Dillinger is a cloud-enabled HTML5 Markdown editor.

- Type some Markdown text in the left window
- See the HTML in the right
- Magic

Markdown is a lightweight markup language based on the formatting conventions that people naturally use in email. As John Gruber writes on the Markdown site:

> The overriding design goal for Markdown's formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions.

This text you see here is *actually* written in Markdown! To get a feel for Markdown's syntax, type some text into the left window and watch the results in the right.

Version

2.0

Tech

Dillinger uses a number of open source projects to work properly:

- Ace Editor - awesome web-based text editor
- Marked - a super fast port of Markdown to JavaScript
- Twitter Bootstrap - great UI boilerplate for modern web apps
- node.js - evented I/O for the backend
- Express - fast node.js network app framework @tjholowaychuk
- keymaster.js - awesome keyboard handler lib by @thomasfuchs
- jQuery - duh

Installation

```
git clone [git-repo-url] dillinger
cd dillinger
npm i -d
mkdir -p public/files/{md,html,pdf}
```

Configure Plugins. Instructions in following README.md files

- plugins/dropbox/README.md
- plugins/github/README.md
- plugins/googledrive/README.md

```
node app
```

http://dillinger.io/

Clone the GitHub repo

```
$ mkdir scientific-computing-group
$ cd scientific-computing-group
$ git clone https://github.com/jlant-usgs/scientific-computing-group.git
$ git pull -u origin master
```

```
/scientific-computing-group /          # parent directory; call it what you like
    projects/
       my-hobbies/                            # repo for git lesson
       readmeasurements/                  # repo for python lesson
     recordings/
    scientific-computing-group/        # this is the scientific computing group's GitHub repo
        .git/
        data/
        meetings/
        presentations/
        resources/
        readme.md
```

Read Measurements Project

- Purpose - read and process measurement like data files, compute simple statistics for each parameter, and display results to the screen.
- Next step - refactor script with functions to make code more easily understandable for your future self and to others reading your code. In addition, allows for your code to be more flexible and robust.

date	discharge (cfs)	stage (ft)	temperature (celsius)
01/05/2014	100	12.2	5
02/08/2014	110	12.8	3
03/07/2014	105	12.5	10
04/01/2014	98	11.9	20
05/04/2014	92	11.5	25
06/01/2014	104	12.3	28
07/02/2014	97	11.8	32
08/03/2014	95	11.7	33
09/04/2014	96	11.7	27
10/05/2014	101	12.0	20
11/02/2014	112	13.2	15
12/03/2014	109	12.8	7

```
$ python read_measurements.py 2014_measurements_bob.txt
2014_measurements_bob.txt

  discharge (cfs):
     Average: 101.583
     Maximum: 112.0 occurred on 11/02/2014
     Minimum: 92.0 occurred on 05/04/2014

  stage (ft):
     Average: 12.200
     Maximum: 13.2 occurred on 11/02/2014
     Minimum: 11.5 occurred on 05/04/2014

  temperature (celsius):
     Average: 18.750
     Maximum: 33.0 occurred on 08/03/2014
     Minimum: 3.0 occurred on 02/08/2014
```

Next Meeting

- Continue to implement functions into the read_measurements.py script
- Learn about python scoping