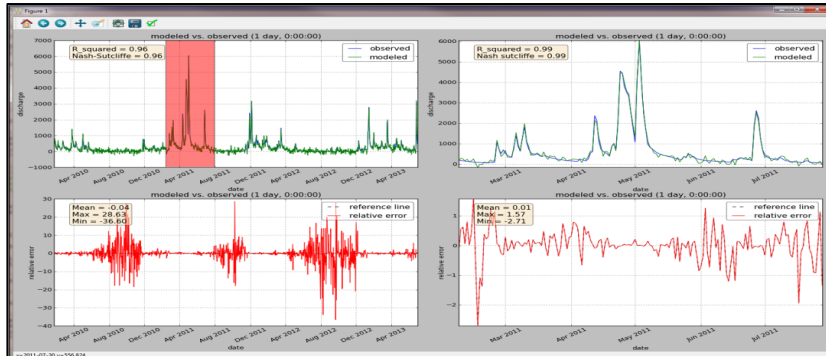
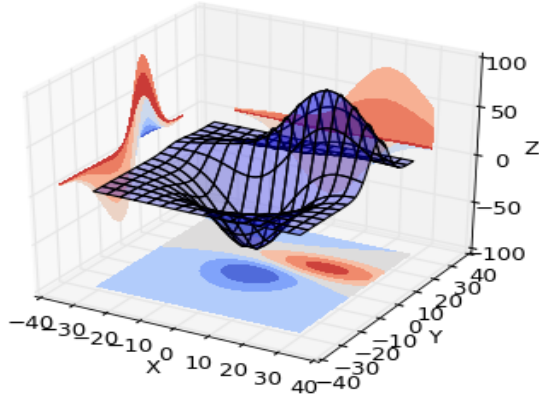


Scientific Computing Group

Programming with Python:

Functions, Read Measurements Project



```
# 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

The screenshot displays the TitanPad web application interface. The top navigation bar includes links for HOME, BLOG, HELP, and SIGN IN. The main header area features the TitanPad logo and the tagline "TitanPad lets people work on one document simultaneously", followed by the subtext "We are rescuing EtherPad for your use." A prominent blue button labeled "Create public pad" with the subtext "No sign-up, start writing instantly" is visible on the left.

The central workspace is divided into two main sections. The left section, titled "Meeting Notes", contains a rich text editor with a toolbar (bold, italic, underline, strikethrough, bulleted list, numbered list, link, unlink, image, undo, redo) and two text blocks: "Next Steps" with a bulleted item "Update 'Can I choose my own name or URL...' content on 'FAQ' page" and "New 'FAQ' text" with the paragraph "Yes! To create a custom name/URL for a pad, just type it in to your browser and visit that URL. You will then be prompted to create the pad." The right section is a chat area titled "David G." showing a list of participants (J.D., Aaron) and a chat log with messages from David G., Aaron, and J.D. with timestamps.

The footer area contains the quote "Everyone loves it!" and the attribution "— Anonymous, Internet". Below this, four feature highlights are presented: "True real-time" with a clock icon, "Eight distinctive author colours to choose from" with a color wheel icon, "Infinite Undo" with a circular arrow icon, and "Get your own private space" with a folder icon and the text "Get a space for your team, on a subdomain For FREE!". The bottom of the page includes copyright information "Copyright © 2010 TitanPad Authors" with links to "About" and "Impressum", and the text "Powered by TitanPad".

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+FS')

    - 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

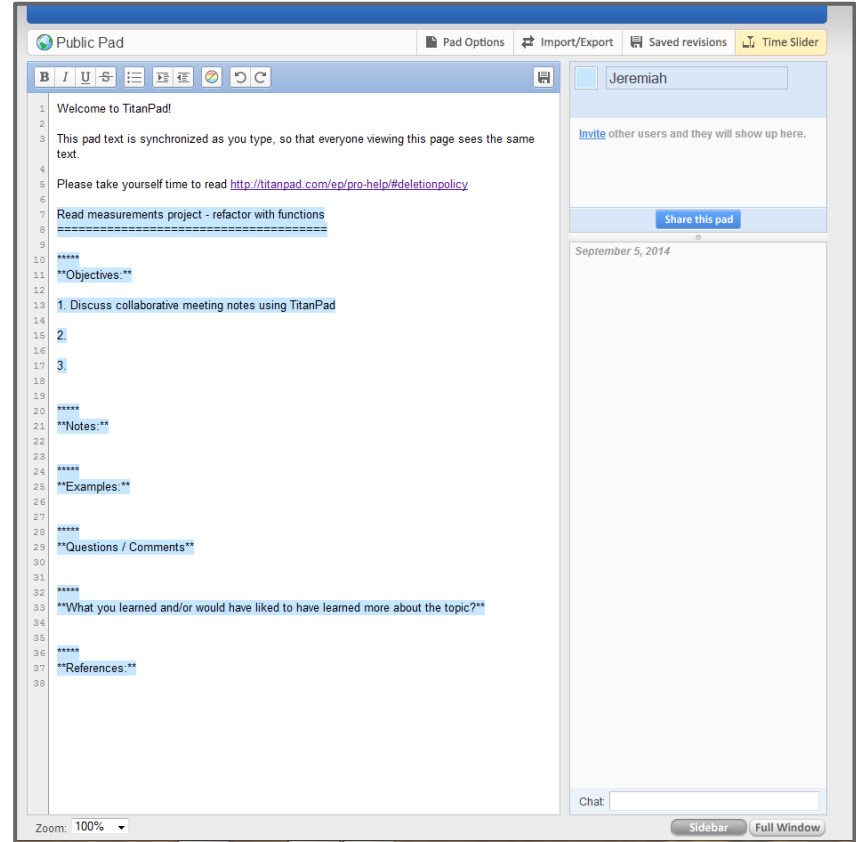
    2. use 'assert' statements in main program

        - '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](http://daringfireball.net/projects/markdown/) (or [HTML](http://daringfireball.net/projects/markdown/))” (<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.

<pre>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.</pre>	<pre><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
 line break</p> <p>Text attributes italic, bold, <code>monospace</code>.</p> <p>A link.</p> <p>Shopping list:</p> apples oranges pears <p>Numbered list:</p> apples oranges pears <p>The rain&mdash;not the reign&mdash;in Spain.</p></pre>	<p>Heading</p> <hr/> <p>Sub-heading</p> <hr/> <p>Traditional html title</p> <p>Paragraphs are separated by a blank line.</p> <p>Let 2 spaces at the end of a line to do a line break</p> <p>Text attributes <i>italic</i>, bold, <code>monospace</code>.</p> <p>A link.</p> <p>Shopping list:</p> <ul style="list-style-type: none">• apples• oranges• pears <p>Numbered list:</p> <ol style="list-style-type: none">1. apples2. oranges3. pears <p>The rain—not the reign—in Spain.</p>
--	---	--

Markdown - online editors

The screenshot displays the Markable online editor interface. The top navigation bar includes links for 'Export', 'Help', 'Login', and 'Signup for more features'. The document title is 'Python - alias, debugging, testing'. The left pane shows the raw Markdown source code, which includes a list of objectives, notes on debugging and testing, and a code example for a function. The right pane shows the rendered HTML output, where the objectives and notes are formatted as lists and the code is highlighted. The rendered document includes sections for 'Objectives:', 'Notes:', and 'Testing functions:'. The code example in the rendered document shows a function 'appender' that appends a number to a list and returns None.

Export ▾ Help ▾ Login Signup for more features

Markableβ

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.

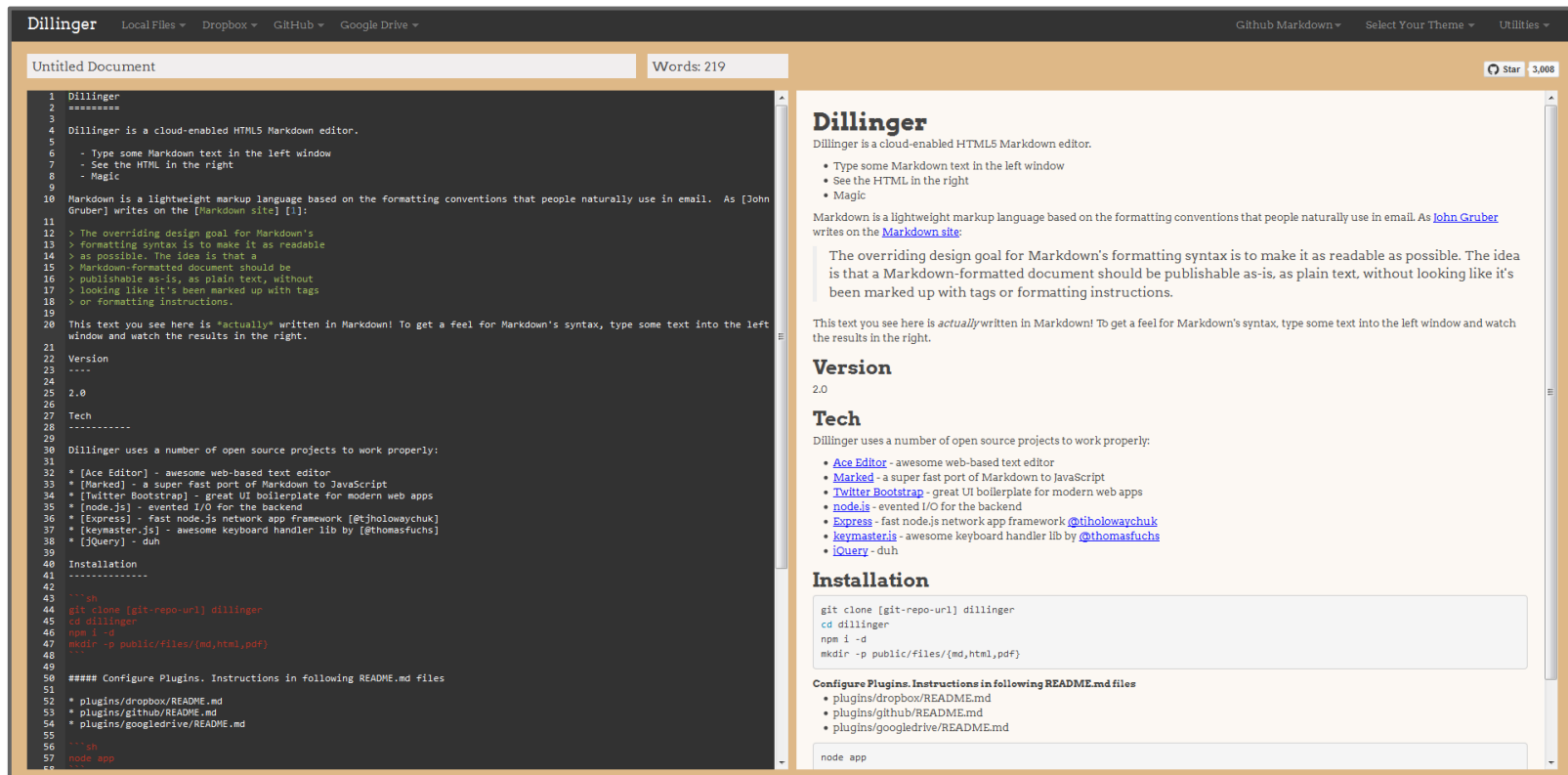
Notes:

- 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

```
1 Python - alias, debugging, testing
2 *****
3
4 *****
5 **Objectives:**
6
7 1. Discuss and review aliasing.
8
9 2. Debug a program using the Python debugger in the Spyder IDE.
10
11 3. Learn a few ways to test a function.
12
13 *****
14 **Notes:**
15
16 * Lists can be mutated in functions
17
18 * A debugger prevents you from having to change / alter your code with print statements
19
20 * Debugging keywords and keyboard shortcuts in [Spyder IDE]:
21
22   - [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
23
24   - Debug file ('Ctrl+F5')
25
26   - Run current line ('Ctrl+F10')
27
28   - Step into function or method of current line ('Ctrl+F11')
29
30   - Run until current function or method returns ('Ctrl+Shift+F11')
31
32   - Continue execution until next breakpoint ('Ctrl+F12')
33
34   - Exit debug mode ('Ctrl+Shift+F12')
35
36 * A function "without" a return statement stills returns 'None'
37
38 * Testing functions - A few methods:
39
40   1. use 'print()' function or 'print' statements in main program and function to manually check validity
41
42   2. use 'assert' statements in main program
43
44     - 'assert actual == expected, "some error message"'
45
46   3. write a test function which contains an assert statement; can have many test functions for a single function
47
48 *****
49 **Examples:**
50
51 **Aliasing in functions** - a list can be mutated in a function
52
53 def appender(x_list, number):
54     x_list.append(number)
55
56     return None
57
```

<http://markable.in/editor/>

Markdown - online editors



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

Untitled Document Words: 219 3,008

```
1 Dillinger
2 =====
3
4 Dillinger is a cloud-enabled HTML5 Markdown editor.
5
6 - Type some Markdown text in the left window
7 - See the HTML in the right
8 - Magic
9
10 Markdown is a lightweight markup language based on the formatting conventions that people naturally use in email. As [John
11 Gruber] writes on the [Markdown site] [1]:
12
13 > The overriding design goal for Markdown's
14 > formatting syntax is to make it as readable
15 > as possible. The idea is that a
16 > Markdown-formatted document should be
17 > publishable as-is, as plain text, without
18 > looking like it's been marked up with tags
19 > or formatting instructions.
20
21 This text you see here is "actually" written in Markdown! To get a feel for Markdown's syntax, type some text into the left
22 window and watch the results in the right.
23
24 Version
25 ----
26
27 2.0
28
29 Tech
30 -----
31
32 Dillinger uses a number of open source projects to work properly:
33
34 * [Ace Editor] - awesome web-based text editor
35 * [Marked] - a super fast port of Markdown to JavaScript
36 * [Twitter Bootstrap] - great UI boilerplate for modern web apps
37 * [node.js] - evented I/O for the backend
38 * [Express] - fast node.js network app framework [at]holowaychuk
39 * [keymaster.js] - awesome keyboard handler lib by [thomasfuchs]
40 * [jQuery] - duh
41
42 Installation
43 -----
44
45 git clone [git-repo-url] dillinger
46 cd dillinger
47 npm i -d
48 mkdir -p public/files/{md,html,pdf}
49
50 ##### Configure Plugins: Instructions in following README.md files
51
52 * plugins/dropbox/README.md
53 * plugins/github/README.md
54 * plugins/google drive/README.md
55
56
57 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/google drive/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