Git for the DSI quick reference

What you want to do	**Git commands at command line**
Clone a repo on Github locally	`git clone <url_from_github>`
Check to see what files are changed	`git status`
Show file differences that haven't been staged	`git diff`
See what branch you are on	`git branch`
Add a file to the staging area (for tracking)	`git add <filename>`
Remove a file from the staging area	`git reset <filename>`
Show file differences between staged and last commit	`git diff --staged`
Commit changes in staging area to a checkpoint	`git commit -m "<a short, descriptive commit message>"`
See where on Github you will push your commit	`git remote -v`
Push your commit to Github	`git push <name_of_remote> <name_of_branch>`
Pull your latest changes on Github down locally	`git pull <name_of_remote> <name_of_branch>`
For pair programming	
Assumes you and your partner are working on the master (default) branch	
After your partner has added you as a collaborator on his/her Github repository:	
Add your partner as a remote (for push/pulling)	`git remote add <their_name> <url_of_their_repo_on_Github>`
Pull their changes into your master branch	`git pull <their_name> master`
You do some work (you "drive" aka code)	
Add your changes to the staging area	`git add <filename_1> <filename_2>`
Commit changes in staging area to a checkpoint	`git commit -m "<a short, descriptive commit message>"`
Push your commit to your partner's remote	`git push <their_name> master`
Push your commit to your remote	`git push origin master`
Now your partner starts to drive (you navigate)	
First your partner pulls your changes down	`git pull origin master`
For group case studies	
Assumes you are following the Feature Branch Workflow	
https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow	
*In this workflow **you never work on the master branch** (for final code & code to be shared)*	
One of the group members (A) forks the Galvanize case study, and then clones it locally.	
On Github, A adds everyone as collaborators, and then they clone A's repository (not Galvanize's).	
Each member (incl. A) makes a branch named after them	`git branch <your_name>`
Now each member (incl. A) checks out their branch	`git checkout <your_name>`
Now everyone does their work	
You have some code you want to share with the group, so:	
Add your changes to the staging area	`git add <filename>`
Commit changes in staging area to a checkpoint	`git commit -m "<a short, descriptive commit message>"`
Push your commit (on your branch) to A's remote	`git push origin <your_name>`
Now A will pull down your branch, and merge it into master	
A gets on the master branch	`git checkout master`
A pulls down whatever is new on the master branch	`git pull origin master`
A verifies that your branch exists on the remote	`git branch -r`
A fetches your branch	`git fetch origin <your_name>:<your_name>  #  Github:local`
A verifies he/she is on the master branch	`git checkout master`
A merges your branch into master	`git merge <your_name>`
A pushes the master branch up to Github	`git push origin master`
Others can pull down the master branch to access your code	`git pull origin master`
To use it, you and they should make a branch off master	`git checkout master`
Delete the old branch (if all changes have been committed)	`git branch -d <your_name>`
Make a new branch	`git branch <your_name>`
Check it out (to work on it)	`git checkout <your_name>`
Fetching a remote branch	
Clone a repo on Github locally	`git clone <url_from_github>`
Fetch the remote branch	`git fetch origin <remote_branch>:<remote_branch>  # Github:local`
If you want to merge a branch into master (after you've fetched it above)	
Make sure you are on master	`git checkout master`
Merge the branch into the master branch	`git merge <branch_name>`
Delete the branch	`git branch -d <branch_name>`

CHEATSHEET FOR DOCKER CLI

RUN A NEW CONTAINER

Start a new Container from an Image
```
docker run IMAGE
docker run nginx
```

...and assign it a name
```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port
```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports
```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background
```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname
```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry
```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container
```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/:/usr/share/nginx/html nginx
```

...but change the entrypoint
```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

MANAGE CONTAINERS

Show a list of running containers
```
docker ps
```

Show a list of all containers
```
docker ps -a
```

Delete a container
```
docker rm CONTAINER
docker rm web
```

Delete a running container
```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers
```
docker container prune
```

Stop a running container
```
docker stop CONTAINER
docker stop web
```

Start a stopped container
```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host
```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container
```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container
```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container
```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container
```
docker commit CONTAINER
docker commit web
```

MANAGE IMAGES

Download an image
```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository
```
docker push IMAGE
docker push myimage:1.0
```

Delete an image
```
docker rmi IMAGE
```

Show a list of all Images
```
docker images
```

Delete dangling images
```
docker image prune
```

Delete all unused images
```
docker image prune -a
```

Build an image from a Dockerfile
```
docker build DIRECTORY
docker build .
```

Tag an image
```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile
```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file
```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file
```
docker load -i TARFILE
docker load -i nginx.tar
```

INFO & STATS

Show the logs of a container
```
docker logs CONTAINER
docker logs web
```

Show stats of running containers
```
docker stats
```

Show processes of container
```
docker top CONTAINER
docker top web
```

Show installed docker version
```
docker version
```

Get detailed info about an object
```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container
```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container
```
docker port CONTAINER
docker port web
```

Visual Studio Code

Keyboard shortcuts for macOS

General

⇧⌘P, F1	Show Command Palette
⌘P	Quick Open, Go to File...
⇧⌘N	New window/instance
⌘W	Close window/instance
⌘,	User Settings
⌘K ⌘S	Keyboard Shortcuts

Basic editing

⌘X	Cut line (empty selection)
⌘C	Copy line (empty selection)
⌥↓ / ⌥↑	Move line down/up
⇧⌥↓ / ⇧⌥↑	Copy line down/up
⇧⌘K	Delete line
⌘Enter / ⇧⌘Enter	Insert line below/above
⇧⌘\	Jump to matching bracket
⌘] / ⌘[	Indent/outdent line
Home / End	Go to beginning/end of line
⌘↑ / ⌘↓	Go to beginning/end of file
^PgUp / ^PgDn	Scroll line up/down
⌘PgUp /⌘PgDn	Scroll page up/down
⌥⌘[/ ⌥⌘]	Fold/unfold region
⌘K ⌘[/ ⌘K ⌘]	Fold/unfold all subregions
⌘K ⌘0 / ⌘K ⌘J	Fold/unfold all regions
⌘K ⌘C	Add line comment
⌘K ⌘U	Remove line comment
⌘/	Toggle line comment
⇧⌥A	Toggle block comment
⌥Z	Toggle word wrap

Multi-cursor and selection

⌥ + click	Insert cursor
⌥⌘↑	Insert cursor above
⌥⌘↓	Insert cursor below
⌘U	Undo last cursor operation
⇧⌥I	Insert cursor at end of each line selected
⌘L	Select current line
⇧⌘L	Select all occurrences of current selection
⌘F2	Select all occurrences of current word
^⇧⌘→ / ←	Expand / shrink selection
⇧⌥ + drag mouse	Column (box) selection
⇧⌥⌘↑ / ↓	Column (box) selection up/down
⇧⌥⌘← / →	Column (box) selection left/right
⇧⌥⌘PgUp	Column (box) selection page up
⇧⌥⌘PgDn	Column (box) selection page down

Search and replace

⌘F	Find
⌥⌘F	Replace
⌘G / ⇧⌘G	Find next/previous
⌥Enter	Select all occurrences of Find match
⌘D	Add selection to next Find match
⌘K ⌘D	Move last selection to next Find match

Rich languages editing

^Space	Trigger suggestion
⇧⌘Space	Trigger parameter hints
⇧⌥F	Format document
⌘K ⌘F	Format selection
F12	Go to Definition
⌥F12	Peek Definition
⌘K F12	Open Definition to the side
⌘.	Quick Fix
⇧F12	Show References
F2	Rename Symbol
⌘K ⌘X	Trim trailing whitespace
⌘K M	Change file language

Navigation

⌘T	Show all Symbols
^G	Go to Line...
⌘P	Go to File...
⇧⌘O	Go to Symbol...
⇧⌘M	Show Problems panel
F8 / ⇧F8	Go to next/previous error or warning
^⇧Tab	Navigate editor group history
^- / ^⇧-	Go back/forward
^⇧M	Toggle Tab moves focus

Editor management

⌘W	Close editor
⌘K F	Close folder
⌘\	Split editor
⌘1 / ⌘2 / ⌘3	Focus into 1st, 2nd, 3rd editor group
⌘K ⌘← / ⌘K ⌘→	Focus into previous/next editor group
⌘K ⇧⌘← / ⌘K ⇧⌘→	Move editor left/right
⌘K ← / ⌘K →	Move active editor group

File management

⌘N	New File
⌘O	Open File...
⌘S	Save
⇧⌘S	Save As...
⌥⌘S	Save All
⌘W	Close
⌘K ⌘W	Close All
⇧⌘T	Reopen closed editor
⌘K Enter	Keep preview mode editor open
^Tab / ^⇧Tab	Open next / previous
⌘K P	Copy path of active file
⌘K R	Reveal active file in Explorer
⌘K O	Show active file in new window/instance

Display

^⌘F	Toggle full screen
⌥⌘0	Toggle editor layout (horizontal/vertical)
⌘= / ⇧⌘-	Zoom in/out
⌘B	Toggle Sidebar visibility
⇧⌘E	Show Explorer / Toggle focus
⇧⌘F	Show Search
^⇧G	Show Source Control
⇧⌘D	Show Debug
⇧⌘X	Show Extensions
⇧⌘H	Replace in files
⇧⌘J	Toggle Search details
⇧⌘U	Show Output panel
⇧⌘V	Open Markdown preview
⌘K V	Open Markdown preview to the side
⌘K Z	Zen Mode (Esc Esc to exit)

Debug

F9	Toggle breakpoint
F5	Start/Continue
F11 / ⇧F11	Step into/ out
F10	Step over
⇧F5	Stop
⌘K ⌘I	Show hover

Integrated terminal

^`	Show integrated terminal
^⇧`	Create new terminal
⌘C	Copy selection
⌘↑ / ↓	Scroll up/down
PgUp / PgDn	Scroll page up/down
⌘Home / End	Scroll to top/bottom

Other operating systems' keyboard shortcuts and additional unassigned shortcuts available at aka.ms/vscodekeybindings

Visual Studio Code

Keyboard shortcuts for Linux

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↓ / ↑	Move line down/up
Ctrl+Shift+K	Delete line
Ctrl+Enter / Ctrl+Shift+Enter	Insert line below/ above
Ctrl+Shift+	Jump to matching bracket
Ctrl+] / Ctrl+[	Indent/Outdent line
Home / End	Go to beginning/end of line
Ctrl+ Home / End	Go to beginning/end of file
Ctrl+ ↑ / ↓	Scroll line up/down
Alt+ PgUp / PgDn	Scroll page up/down
Ctrl+Shift+ [/]	Fold/unfold region
Ctrl+K Ctrl+ [/]	Fold/unfold all subregions
Ctrl+K Ctrl+0 / Ctrl+K Ctrl+J	Fold/Unfold all regions
Ctrl+K Ctrl+C	Add line comment
Ctrl+K Ctrl+U	Remove line comment
Ctrl+/	Toggle line comment
Ctrl+Shift+A	Toggle block comment
Alt+Z	Toggle word wrap

Rich languages editing

Ctrl+Space	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Ctrl+Shift+I	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Ctrl+Shift+F10	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Multi-cursor and selection

Alt+Click	Insert cursor*
Shift+Alt+ ↑ / ↓	Insert cursor above/below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt + →	Expand selection
Shift+Alt + ←	Shrink selection
Shift+Alt + drag mouse	Column (box) selection

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ = / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+C	Open new command prompt/terminal
Ctrl+K Ctrl+H	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurrences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match

Navigation

Ctrl+T	Show all Symbols
Ctrl+G	Go to Line...
Ctrl+P	Go to File...
Ctrl+Shift+O	Go to Symbol...
Ctrl+Shift+M	Show Problems panel
F8	Go to next error or warning
Shift+F8	Go to previous error or warning
Ctrl+Shift+Tab	Navigate editor group history
Ctrl+Alt+-	Go back
Ctrl+Shift+-	Go forward
Ctrl+M	Toggle Tab moves focus

Editor management

Ctrl+W	Close editor
Ctrl+K F	Close folder
Ctrl+	Split editor
Ctrl+ 1 / 2 / 3	Focus into 1st, 2nd, 3rd editor group
Ctrl+K Ctrl + ←	Focus into previous editor group
Ctrl+K Ctrl + →	Focus into next editor group
Ctrl+Shift+PgUp	Move editor left
Ctrl+Shift+PgDn	Move editor right
Ctrl+K ←	Move active editor group left/up
Ctrl+K →	Move active editor group right/down

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+W	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window/instance

Debug

F9	Toggle breakpoint
F5	Start / Continue
F11 / Shift+F11	Step into/out
F10	Step over
Shift+F5	Stop
Ctrl+K Ctrl+I	Show hover

Integrated terminal

Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Shift+C	Copy selection
Ctrl+Shift+V	Paste into active terminal
Ctrl+Shift+ ↑ / ↓	Scroll up/down
Shift+ PgUp / PgDn	Scroll page up/down
Shift+ Home / End	Scroll to top/bottom

* The Alt+Click gesture may not work on some Linux distributions. You can change the modifier key for the Insert cursor command to Ctrl+Click with the “editor.multiCursorModifier” setting.

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

CC BY SA

VIM

Cursor movement

h - move left
j - move down
k - move up
l - move right
w - jump by start of words (punctuation considered words)
W - jump by words (spaces separate words)
e - jump to end of words (punctuation considered words)
E - jump to end of words (no punctuation)
b - jump backward by words (punctuation considered words)
B - jump backward by words (no punctuation)
0 - (zero) start of line
^ - first non-blank character of line
$ - end of line
G - Go To command (prefix with number - 5G goes to line 5)
Note: Prefix a cursor movement command with a number to repeat it. For example, 4j moves down 4 lines.

Insert Mode - Inserting/Appending text

i - start insert mode at cursor
I - insert at the beginning of the line
a - append after the cursor
A - append at the end of the line
o - open (append) blank line below current line (no need to press return)
O - open blank line above current line
ea - append at end of word
Esc - exit insert mode

Editing

r - replace a single character (does not use insert mode)
J - join line below to the current one
cc - change (replace) an entire line
cw - change (replace) to the end of word
c$ - change (replace) to the end of line
s - delete character at cursor and subsitute text
S - delete line at cursor and substitute text (same as cc)
xp - transpose two letters (delete and paste, technically)
u - undo
. - repeat last command

Marking text (visual mode)

v - start visual mode, mark lines, then do command (such as y-yank)
V - start Linewise visual mode
o - move to other end of marked area
Ctrl+v - start visual block mode
O - move to Other corner of block
aw - mark a word
ab - a () block (with braces)
aB - a {} block (with brackets)
ib - inner () block
iB - inner {} block
Esc - exit visual mode

Visual commands

> - shift right
< - shift left
y - yank (copy) marked text
d - delete marked text
~ - switch case

Cut and Paste

yy - yank (copy) a line
2yy - yank 2 lines
yw - yank word
y$ - yank to end of line
p - put (paste) the clipboard after cursor
P - put (paste) before cursor
dd - delete (cut) a line
dw - delete (cut) the current word
x - delete (cut) current character

Exiting

:w - write (save) the file, but don't exit
:wq - write (save) and quit
:q - quit (fails if anything has changed)
:q! - quit and throw away changes

Search/Replace

/pattern - search for pattern
?pattern - search backward for pattern
n - repeat search in same direction
N - repeat search in opposite direction
:%s/old/new/g - replace all old with new throughout file
:%s/old/new/gc - replace all old with new throughout file with confirmations

Working with multiple files

:e filename - Edit a file in a new buffer
:bnext (or :bn) - go to next buffer
:bprev (of :bp) - go to previous buffer
:bd - delete a buffer (close a file)
:sp filename - Open a file in a new buffer and split window
ctrl+ws - Split windows
ctrl+ww - switch between windows
ctrl+wq - Quit a window
ctrl+wv - Split windows vertically

Python Standard Library Debugger (pdb)

Start the debugger by inserting a **`breakpoint()`** in your code. For example:

```
def bubblesort(lst):
    '''Bubblesort, more sparse and easier to debug'''
    breakpoint()
    num_el = len(lst)
    num_passes = num_el - 1
```

Common debugger commands:

command	description
p	Print the value of an expression.
pp	Pretty-print the value of an expression.
n	Continue execution until the next line in the current function is reached or it returns.
s	Step into a function/class
c	Continue execution and only stop when a breakpoint is encountered.
unt	Continue execution until the line with a number greater than the current one is reached. With a line number argument, continue execution until a line with a number greater or equal to that is reached.
l	List source code for the current file. Without arguments, list 11 lines around the current line or continue the previous listing.
ll	List the whole source code for the current function or frame.
b	With no arguments, list all breaks. With a line number argument, set a breakpoint at this line in the current file
w	Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame.
u	Move the current frame count (default one) levels up in the stack trace (to an older frame).
d	Move the current frame count (default one) levels down in the stack trace (to a newer frame).
h	See a list of available commands.
h pdb	Show the full pdb documentation.
q	Quit the debugger and exit.
!	Will override a command to see a variable value. For example, to see the value of variable n instead of n for next: !n

Python Notes/Cheat Sheet

Comments
from the hash symbol to the end of a line

Code blocks
Delineated by colons and indented code; and not the curly brackets of C, C++ and Java.

```
def is_fish_as_string(argument):
    if argument:
        return 'fish'
    else:
        return 'not fish'
```

Note: Four spaces per indentation level is the Python standard. Never use tabs: mixing tabs and spaces produces hard-to-find errors. Set your editor to convert tabs to spaces.

Line breaks
Typically, a statement must be on one line. Bracketed code - (), [] or {} - can be split across lines; or (if you must) use a backslash \ at the end of a line to continue a statement on to the next line (but this can result in hard to debug code).

Naming conventions

Style	Use
StudlyCase	Class names
joined_lower	Identifiers, functions; and class methods, attributes
_joined_lower	Internal class attributes
__joined_lower	Private class attributes # this use not recommended
joined_lower **ALL_CAPS**	Constants

Basic object types (not a complete list)

Type	Examples	
None	None	# singleton null object
Boolean	True, False	
integer	-1, 0, 1, sys.maxint	
long	1L, 9787L	# arbitrary length ints
float	3.14159265 inf, float('inf') -inf nan, float('nan')	 # infinity # neg infinity # not a number
complex	2+3j	# note use of j
string	'I am a string', "me too" '''multi-line string''', """+1""" r'raw string', b'ASCII string' u'unicode string'	
tuple	empty = () (1, True, 'dog')	# empty tuple # immutable list
list	empty = [] [1, True, 'dog']	# empty list # mutable list
set	empty = set() set(1, True, 'a')	# the empty set # mutable
dictionary	empty = {} {'a': 'dog', 7: 'seven', True: 1}	# mutable object
file	f = open('filename', 'rb')	

Note: Python has four numeric types (integer, float, long and complex) and several sequence types including strings, lists, tuples, bytearrays, buffers, and range objects.

Operators

Operator	Functionality
+	Addition (also string, tuple, list, and other sequence concatenation)
-	Subtraction (also set difference)
*	Multiplication (also string, tuple, list replication)
/	Division
%	Modulus (also a string format function, but this use deprecated)
//	Integer division rounded towards minus infinity
**	Exponentiation
=, -=, +=, /=, *=, %=, //=, **=	Assignment operators
==, !=, <, <=, >=, >	Boolean comparisons
and, or, not	Boolean operators
in, not in	Containment test operators
is, is not	Object identity operators
|, ^, &, ~	Bitwise: or, xor, and, compliment
<<, >>	Left and right bit shift
;	Inline statement separator # inline statements discouraged

Hint: float('inf') always tests as larger than any number, including integers.

Modules
Modules open up a world of Python extensions that can be imported and used. Access to the functions, variables and classes of a module depend on how the module was imported.

Import method	Access/Use syntax
import math	math.cos(math.pi/3)
import math as m **# import using an alias**	m.cos(m.pi/3)
from math import cos, pi **# only import specifics**	cos(pi/3)
from math import * **# BADish global import**	log(e)

Global imports make for unreadable code!!!

Oft used modules

Module	Purpose
datetime **time**	Date and time functions
math	Core math functions and the constants pi and e
pickle	Serialise objects to a file
os **os.path**	Operating system interfaces
re	A library of Perl-like regular expression operations
string	Useful constants and classes
sys	System parameters and functions
numpy	Numerical python library
pandas	R DataFrames for Python
matplotlib	Plotting/charting for Python

If - flow control

```
if condition:   # for example: if x < 5:
    statements
elif condition: # optional – can be multiple
    statements
else:           # optional
    statements
```

For - flow control

```
for x in iterable:
    statements
else:           # optional completion code
    statements
```

While - flow control

```
while condition:
    statements
else:           # optional completion code
    statements
```

Ternary statement

id = expression if *condition* else *expression*

```
x = y if a > b else z - 5
```

Some useful adjuncts:

- pass - a statement that does nothing
- continue - moves to the next loop iteration
- break - to exit for and while loop

Trap: break skips the else completion code

Exceptions – flow control

```
try:
    statements
except (tuple_of_errors): # can be multiple
    statements
else:            # optional no exceptions
    statements
finally:         # optional all
    statements
```

Common exceptions (not a complete list)

Exception	Why it happens
AsserionError	Assert statement failed
AttributeError	Class attribute assignment or reference failed
IOError	Failed I/O operation
ImportError	Failed module import
IndexError	Subscript out of range
KeyError	Dictionary key not found
MemoryError	Ran out of memory
NameError	Name not found
TypeError	Value of the wrong type
ValueError	Right type but wrong value

Raising errors

Errors are raised using the raise statement

```
raise ValueError(value)
```

Creating new errors

```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
```

Objects and variables (AKA identifiers)

- Everything is an object in Python (in the sense that it can be assigned to a variable or passed as an argument to a function)
- Most Python objects have methods and attributes. For example, all functions have the built-in attribute __doc__, which returns the doc string defined in the function's source code.
- All variables are effectively "pointers", not "locations". They are references to objects; and often called identifiers.
- Objects are strongly typed, not identifiers
- Some objects are immutable (int, float, string, tuple, frozenset). But most are mutable (including: list, set, dictionary, NumPy arrays, etc.)
- You can create our own object types by defining a new class (see below).

Booleans and truthiness

Most Python objects have a notion of "truth".

False	True
None	
0 **int(False) # → 0**	Any number other than 0 int(True) # → 1
"" **# the empty string**	" ", 'fred', 'False' # all other strings
() [] {} set() **# empty containers**	[None], (False), {1, 1} # non-empty containers, including those containing False or None.

You can use bool() to discover the truth status of an object.

```
a = bool(obj)          # the truth of obj
```

It is pythonic to use the truth of objects.

```
if container:          # test not empty
    # do something
while items:           # common looping idiom
    item = items.pop()
    # process item
```

Specify the truth of the classes you write using the __nonzero__() magic method.

Comparisons

Python lets you compare ranges, for example

```
assert(1 <= x <= 100)
```

Tuples

Tuples are immutable lists. They can be searched, indexed and iterated much like lists (see below). List methods that do not change the list also work on tuples.

```
a = ()                        # the empty tuple
a = (1,)  # ← note comma      # one item tuple
a = (1, 2, 3)                 # multi-item tuple
a = ((1, 2), (3, 4))          # nested tuple
a = tuple(['a', 'b'])         # conversion
```

Note: the comma is the tuple constructor, not the parentheses. The parentheses add clarity.

The Python swap variable idiom

```
a, b = b, a    # no need for a temp variable
```

This syntax uses tuples to achieve its magic.

String (immutable, ordered, characters)

```
s = 'string'.upper()          # STRING
s = 'fred'+'was'+'here'       # concatenation
s = ''.join(['fred', 'was', 'here'])        # ditto
s = 'spam' * 3                # replication
s = str(x)                    # conversion
```

String iteration and sub-string searching

```
for character in 'str':       # iteration
    print (ord(character))    # 115 116 114
for index, character in enumerate('str')
    print (index, character)
if 'red' in 'Fred':           # searching
    print ('Fred is red')     # it prints!
```

String methods (not a complete list)

capitalize, center, count, decode, encode, endswith, expandtabs, find, format, index, isalnum, isalpha, isdigit, islower, isspace, istitle, isupper, join, ljust, lower, lstrip, partition, replace, rfind, rindex, rjust, rpartition, rsplit, rstrip, split, splitlines, startswith, strip, swapcase, title, translate, upper, zfill

String constants (not a complete list)

```
from string import *  # global import is not good
print ([digits, hexdigits, ascii_letters,
    ascii_lowercase, ascii_uppercase,
    punctuation])
```

Old school string formatting (using % oper)

```
print ("It %s %d times" % ('occurred', 5))
# prints: 'It occurred 5 times'
```

Code	Meaning
s	String or string conversion
c	Character
d	Signed decimal integer
u	Unsigned decimal integer
H or h	Hex integer (upper or lower case)
f	Floating point
E or e	Exponent (upper or lower case E)
G or g	The shorter of e and f (u/l case)
%	Literal '%'

```
import math
'%s' % math.pi        # '3.14159265359'
'%f' % math.pi        # '3.141593'
'%.2f' % math.pi      # '3.14'
'%.2e' % 3000         # '3.00e+03'
'%03d' % 5            # '005'
```

New string formatting (using format method)

Uses: 'template-string'.format(arguments)
Examples (using similar codes as above):

```
import math
'Hello {}'.format('World')   # 'Hello World'
'{}'.format(math.pi)         # ' 3.14159265359'
'{0:.2f}'.format(math.pi)    # '3.14'
'{0:+.2f}'.format(5)         # '+5.00'
'{:.2e}'.format(3000)        # '3.00e+03'
'{:0>2d}'.format(5)          # '05' (left pad)
'{:x<3d}'.format(5)          # '5xx' (rt. pad)
'{:,}'.format(1000000)       # '1,000,000'
'{:.1%}'.format(0.25)        # '25.0%'
'{0}{1}'.format('a', 'b')    # 'ab'
'{1}{0}'.format('a', 'b')    # 'ba'
'{num:}'.format(num=7)       # '7' (named args)
```

List (mutable, indexed, ordered container)

Indexed from zero to length-1

```
a = []                        # the empty list
a = ['dog', 'cat', 'bird']    # simple list
a = [[1, 2], ['a', 'b']]      # nested lists
a = [1, 2, 3] + [4, 5, 6]     # concatenation
a = [1, 2, 3] * 456           # replication
a = list(x)                   # conversion
```

List comprehensions (can be nested)

Comprehensions: a tight way of creating lists

```
t3 = [x*3 for x in [5, 6, 7]] # [15, 18, 21]
z = [complex(x, y) for x in range(0, 4, 1)
          for y in range(4, 0, -1) if x > y]
# z --> [(2+1j), (3+2j), (3+1j)]
```

Iterating lists

```
L = ['dog', 'cat', 'turtle']
for item in L
    print (item)
for index, item in enumerate(L):
    print (index, item)
```

Searching lists

```
L = ['dog', 'cat', 'turtle']
value = 'cat'
if value in L:
    count = L.count(value)
    first_occurrence = L.index(value)
if value not in L:
    print 'list is missing {}'.format(value)
```

List methods (not a complete list)

Method	What it does
l.append(x)	Add x to end of list
l.extend(other)	Append items from other
l.insert(pos, x)	Insert x at position
del l[pos]	Delete item at pos
l.remove(x)	Remove first occurrence of x; An error if no x
l.pop([pos])	Remove last item from list (or item from pos); An error if empty list
l.index(x)	Get index of first occurrence of x; An error if x not found
l.count(x)	Count the number of times x is found in the list
l.sort()	In place list sort
l.reverse(x)	In place list reversal

List slicing

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8] # play data
x[2]     # 3rd element - reference not slice
x[1:3]   # 2nd to 3rd element (1, 2)
x[:3]    # the first three elements (0, 1, 2)
x[-3:]   # last three elements
x[:-3]   # all but the last three elements
x[:]     # every element of x – copies x
x[1:-1]  # all but first and last element
x[::3]   # (0, 3, 6, 9, …) 1st then every 3rd
x[1:5:2] # (1,3) start 1, stop >= 5, by every 2nd
```

Note: All Python sequence types support the above index slicing (strings, lists, tuples, bytearrays, buffers, and xrange objects)

Set (unique, unordered container)

A Python set is an unordered, mutable collection of unique hashable objects.

```
a = set()                      # empty set
a = {'red', 'white', 'blue'} # simple set
a = set(x)                     # convert list
```

Trap: {} creates empty dict, not an empty set

Set comprehensions

```
# a set of selected letters...
s = {e for e in 'ABCHJADC' if e not in 'AB'}
# --> {'H', 'C', 'J', 'D'}
# a set of tuples ...
s = {(x,y) for x in range(-1,2)
                for y in range (-1,2)}
```

Trap: set contents need to be immutable to be hashable. So you can have a set of tuples, but not a set of lists.

Iterating a set

```
for item in set:
    print (item)
```

Searching a set

```
if item in set:
    print (item)
if item not in set:
    print ('{} is missing'.format(item))
```

Set methods (not a complete list)

Method	What it does
len(s)	Number of items in set
s.add(item)	Add item to set
s.remove(item)	Remove item from set. Raise KeyError if item not found.
s.discard(item)	Remove item from set if present.
s.pop()	Remove and return an arbitrary item. Raise KeyError on empty set.
s.clear()	Remove all items from set
item in s	True or False
item not in s	True or False
iter(s)	An iterator over the items in the set (arbitrary order)
s.copy()	Get shallow copy of set
s.isdisjoint(o)	True if s has not items in common with other set o
s.issubset(o)	Same as set <= other
s.issuperset(o)	Same as set >= other
s.union(o[, ...])	Return new union set
s.intersection(o)	Return new intersection
s.difference(o)	Get net set of items in s but not others (Same as set – other)

Frozenset

Similar to a Python set above, but immutable (and therefore hashable).

```
f = frozenset(s)               # convert set
f = frozenset(o)               # convert other
```

Dictionary (indexed, unordered map-container)

A mutable hash map of unique key=value pairs.

```
a = {}                         # empty dictionary
a = {1: 1, 2: 4, 3: 9}         # simple dict
a = dict(x)                    # convert paired data
# next example – create from a list
l = ['alpha', 'beta', 'gamma', 'delta']
a = dict(zip(range(len(l)), l))
# Example using string & generator expression
s = 'a=apple,b=bird,c=cat,d=dog,e=egg'
a = dict(i.split("=") for i in s.split(","))
# {'a': 'apple', 'c': 'cat', 'b': 'bird',
#    'e': 'egg', 'd': 'dog'}
```

Dictionary comprehensions

Conceptually like list comprehensions; but it constructs a dictionary rather than a list

```
a = { n: n*n for n in range(7) }
# a -> {0:0, 1:1, 2:4, 3:9, 4:16, 5:25,6:36}
odd_sq = { n: n*n for n in range(7) if n%2 }
# odd_sq -> {1: 1, 3: 9, 5: 25}
# next example -> swaps the key:value pairs
a = { val:key for key, val in a.items() }
# next example -> count list occurrences
l = [11,12,13,11,15,19,15,11,20,13,11,11,12,10]
c = { key: l.count(key) for key in set(l) }
```

Iterating a dictionary

```
for key in dictionary.keys():
    print (key)
for key, value in dictionary.items():
    print (key, value)
for value in dictionary.values():
    print(value)
```

Searching a dictionary

```
if key in dictionary:
    print (key)
```

Merging two dictionaries

```
merged = dict_1.copy()
merged.update(dict_2)
```

Dictionary methods (not a complete list)

Method	What it does
len(d)	Number of items in d
d[key]	Get value for key or raise the KeyError exception
d[key] = value	Set key to value
del d[key]	deletion
key in d	True or False
key not in d	True or False
iter(d)	An iterator over the keys
d.clear()	Remove all items from d
d.copy()	Shallow copy of dictionary
d.get(key[, def])	Get value else default
d.items()	Dictionary's (k,v) pairs
d.keys()	Dictionary's keys
d.pop(key[, def])	Get value else default; remove key from dictionary
d.popitem()	Remove and return an arbitrary (k, v) pair
d.setdefault(k[,def])	If k in dict return its value otherwise set def
d.update(other_d)	Update d with key:val pairs from other
d.values()	The values from dict

Key functions (not a complete list)

Function	What it does
abs(num)	Absolute value of num
all(iterable)	True if all are True
any(iterable)	True if any are True
bytearray(source)	A mutable array of bytes
callable(obj)	True if obj is callable
chr(int)	Character for ASCII int
complex(re[, im])	Create a complex number
divmod(a, b)	Get (quotient, remainder)
enumerate(seq)	Get an enumerate object, with next() method returns an (index, element) tuple
eval(string)	Evaluate an expression
filter(fn, iter)	Construct a list of elements from *iter* for which *fn()* returns True
float(x)	Convert from int/string
getattr(obj, str)	Like obj.str
hasattr(obj, str)	True if obj has attribute
hex(x)	From in to hex string
id(obj)	Return unique (run-time) identifier for an object
int(x)	Convert from float/string
isinstance(o, c)	Eg. isinstance(2.1, float)
len(x)	Number of items in x; x is string, tuple, list, dict
list(iterable)	Make a list
long(x)	Convert a string or number to a long integer
map(fn, iterable)	Apply fn() to every item in iterable; return results in a list
max(a,b) **max(iterable)**	What it says on the tin
min(a,b) **min(iterable)**	Ditto
next(iterator)	Get next item from an iter
open(name[,mode])	Open a file object
ord(c)	Opposite of chr(int)
pow(x, y)	Same as x ** y
print (objects)	What it says on the tin takes end arg (default \n) and sep arg (default ' ')
range(stop) **range(start,stop)** **range(fr,to,step)**	integer list; stops < stop default start=0; default step=1
reduce(fn, iter)	Applies the two argument fn(x, y) cumulatively to the items of iter.
repr(object)	Printable representation of an object
reversed(seq)	Get a reversed iterator
round(n[,digits])	Round to number of digits after the decimal place
setattr(obj,n,v)	Like obj.n = v #name/value
sorted(iterable)	Get new sorted list
str(object)	Get a string for an object
sum(iterable)	Sum list of numbers
type(object)	Get the type of object
xrange()	Like range() but better: returns an iterator
zip(x, y[, z])	Return a list of tuples

Using functions

When called, functions can take positional and named arguments.

For example:

```
result = function(32, aVar, c='see', d={})
```

Arguments are passed by reference (ie. the objects are not copied, just the references).

Writing a simple function

```
def funct(arg1, arg2=None, *args, **kwargs):
    """explain what this function does"""
    statements
    return x     # optional statement
```

Note: functions are first class objects that get instantiated with attributes and they can be referenced by variables.

Avoid named default mutable arguments

Avoid mutable objects as default arguments. Expressions in default arguments are evaluated when the function is defined, not when it's called. Changes to mutable default arguments survive between function calls.

```
def nasty(value=[]):       # <-- mutable arg
    value.append('a')
    return value
print (nasty ()) # --> ['a']
print (nasty ()) # --> ['a', 'a']

def better(val=None):
    val = [] if val is None else val
    value.append('a')
    return value
```

Lambda (inline expression) functions:

```
g = lambda x: x ** 2      # Note: no return
print(g(8))               # prints 64
mul = lambda a, b: a * b  # two arguments
mul(4, 5) == 4 * 5        # --> True
```

Note: only for expressions, not statements.
Lambdas are often used with the Python functions filter(), map() and reduce().

```
# get only those numbers divisible by three
div3 = filter(lambda x: x%3==0,range(1,101))
```

Typically, you can put a lambda function anywhere you put a normal function call.

Closures

Closures are functions that have inner functions with data fixed in the inner function by the lexical scope of the outer. They are useful for avoiding hard constants. Wikipedia has a derivative function for changeable values of dx, using a closure.

```
def derivative(f, dx):
    """Return a function that approximates
       the derivative of f using an interval
       of dx, which should be appropriately
       small.
    """
    def _function(x):
        return (f(x + dx) - f(x)) / dx
    return _function #from derivative(f, dx)

f_dash_x = derivative(lambda x: x*x,0.00001)
f_dash_x(5) # yields approx. 10 (ie. y'=2x)
```

An iterable object

The contents of an iterable object can be selected one at a time. Such objects include the Python sequence types and classes with the magic method __iter__(), which returns an iterator. An iterable object will produce a fresh iterator with each call to iter().

```
iterator = iter(iterable_object)
```

Iterators

Objects with a next() or __next__() method, that:

- returns the next value in the iteration
- updates the internal note of the next value
- raises a StopIteration exception when done

Note: with the loop for x in y: if y is not an iterator; Python calls iter() to get one. With each loop, it calls next() on the iterator until a StopIteration exception.

```
x = iter('XY')  # iterate a string by hand
print (next(x)) # --> X
print (next(x)) # --> Y
print (next(x)) # --> StopIteration exception
```

Generators

Generator functions are resumable functions that work like iterators. They can be more space or time efficient than iterating over a list, (especially a very large list), as they only produce items as they are needed.

```
def fib(max=None):
    """ generator for Fibonacci sequence"""
    a, b = 0, 1
    while max is None or b <= max:
        yield b    # ← yield is like return
        a, b = b, a+b

[i for i in fib(10)] # → [1, 1, 2, 3, 5, 8]
```

Note: a return statement (or getting to the end of the function) ends the iteration.
Trap: a yield statement is not allowed in the try clause of a try/finally construct.

Messaging the generator

```
def resetableCounter(max=None):
    j = 0
    while max is None or j <= max:
        x = yield j # ← x gets the sent arg
        j = j+1 if x is None else x

x = resetableCounter(10)
print x.send(None)      # → 0
print x.send(5)         # → 5
print x.send(None)      # → 6
print x.send(11)        # → StopIteration
```

Trap: must send None on first send() call

Generator expressions

Generator expressions build generators, just like building a list from a comprehension. You can turn a list comprehension into a generator expression simply by replacing the square brackets [] with parentheses ().

```
[i for i in range(10)]      # list comprehension
list(i for i in range(10))  # generated list
```

Classes

Python is an object-oriented language with a multiple inheritance class mechanism that encapsulates program code and data.

Methods and attributes

Most objects have associated functions or "methods" that are called using dot syntax:

```
obj.method(arg)
```

Objects also often have attributes or values that are directly accessed without using getters and setters (most unlike Java or C++)

```
instance = Example_Class()
print (instance.attribute)
```

Simple example

```
import math
class Point:
    # static class variable, point count
    count = 0

    def __init__(self, x, y):
        self.x = float(x)
        self.y = float(y)
        Point.count += 1

    def __str__(self):
        return \
        '(x={}, y={})'.format(self.x, self.y)

    def to_polar(self):
        r = math.sqrt(self.x**2 + self.y**2)
        theta = math.atan2(self.y, self.x)
        return(r, theta)

    # static method – trivial example ...
    def static_eg(n):
        print ('{}'.format(n))
    static_eg = staticmethod(static_eg)

# Instantiate 9 points & get polar coords
for x in range(-1, 2):
    for y in range(-1, 2):
        p = Point(x, y)
        print (p)   # uses __str__() method
        print (p.to_polar())
print (Point.count) # check static variable
Point.static_eg(9)  # check static method
```

The self

Class methods have an extra argument over functions. Usually named 'self'; it is a reference to the instance. It is not used in the method call; and is provided by Python to the method. Self is like 'this' in C++ & Java

Public and private methods and variables

Python does not enforce the public v private data distinction. By convention, variables and methods that begin with an underscore should be treated as private (unless you really know what you are doing). Variables that begin with double underscore are mangled by the compiler (and hence more private).

Inheritance

```
class DerivedClass1(BaseClass):
    statements
class DerivedClass2(module_name.BaseClass):
    statements
```

Multiple inheritance

```
class DerivedClass(Base1, Base2, Base3):
    statements
```

Decorators

Technically, decorators are just functions (or classes), that take a callable object as an argument, and return an analogous object with the decoration. We will skip how to write them, and focus on using a couple of common built in decorators.

Practically, decorators are syntactic sugar for more readable code. The *@wrapper* is used to transform the existing code. For example, the following two method definitions are semantically equivalent.

```
def f(...):
    ...
f = staticmethod(f)

@staticmethod
def f(...):
    ...
```

Getters and setters

Although class attributes can be directly accessed, the property function creates a property manager.

```
class Example:
    def __init__(self):
        self._x = None

    def getx(self):
        return self._x
    def setx(self, value):
        self._x = value
    def delx(self):
        del self._x
    x = property(getx, setx, delx,"Doc txt")
```

Which can be rewritten with decorators as:

```
class Example:
    def __init__(self):
        self._x = None

    @property
    def x(self):
        """Doc txt: I'm the 'x' property."""
        return self._x

    @x.setter
    def x(self, value):
        self._x = value

    @x.deleter
    def x(self):
        del self._x
```

Magic class methods (not a complete list)

Magic methods (which begin and end with double underscore) add functionality to your classes consistent with the broader language.

Magic method	What it does
__init__(self,[...])	Constructor
__del__(self)	Destructor pre-garbage collection
__str__(self)	Human readable string for class contents. Called by str(self)
__repr__(self)	Machine readable unambiguous Python string expression for class contents. Called by repr(self) Note: str(self) will call __repr__ if __str__ is not defined.
__eq__(self, other)	Behaviour for ==
__ne__(self, other)	Behaviour for !=
__lt__(self, other)	Behaviour for <
__gt__(self, other)	Behaviour for >
__le__(self, other)	Behaviour for <=
__ge__(self, other)	Behaviour for >=
__add__(self, other)	Behaviour for +
__sub__(self, other)	Behaviour for -
__mul__(self, other)	Behaviour for *
__div__(self, other)	Behaviour for /
__mod__(self, other)	Behaviour for %
__pow__(self, other)	Behaviour for **
__pos__(self, other)	Behaviour for unary +
__neg__(self, other)	Behaviour for unary -
__hash__(self)	Returns an int when hash() called. Allows class instance to be put in a dictionary
__len__(self)	Length of container
__contains__(self, i)	Behaviour for in and not in operators
__missing__(self, i)	What to do when dict key i is missing
__copy__(self)	Shallow copy constructor
__deepcopy__(self, memodict={})	Deep copy constructor
__iter__(self)	Provide an iterator
__nonzero__(self)	Called by bool(self)
__index__(self)	Called by x[self]
__setattr__(self, name, val)	Called by self.name = val
__getattribute__(self, name)	Called by self.name
__getattr__(self, name)	Called when self.name does not exist
__delattr__(self, name)	Called by del self.name
__getitem__(self, key)	Called by self[key]
__setitem__(self, key, val)	Called by self[key] = val
__delitem__(self, key)	del self[key]

Introductory Notes: Matplotlib

Preliminaries

Start by importing these Python modules

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
```

Which Application Programming Interface?

The two worlds of Matplotlib
There are two broad ways of using matplotlib's pyplot:

1. The first (and most common) way is not pythonic. It relies on global functions to build and display a global figure using matplotlib as a global state machine. (This is easy for interactive use).

2. The second way is pythonic and object oriented and it is the best approach for programmatic use:
   a. obtain an empty Figure from a global factory, then build the plot using the methods of the Figure and Axes classes; or
   b. obtain a populated Axes (and its Figure container) using a plot method from the pandas data analysis library, which you can then pretty-up and save to file.

These notes focus on the 2a approach. For 2b, see the pandas cheat sheet (http://bit.ly/python_cs). First, let's look at using pyplot as a global state machine.

Using matplotlib as a global state machine

1. Get data – we will fake up a monthly time series

```
x = pd.period_range('1980-01-01', periods=450,
    freq='M').to_timestamp().to_pydatetime()
y = np.random.randn(len(x)).cumsum()
```

2. Plot the data

```
plt.plot(x, y, label='FDI')
```

3. Add your labels and pretty-up the plot

```
plt.gcf().set_size_inches(8, 3)
plt.title('Fake Data Index')
plt.xlabel('Date')
plt.ylabel('Index')
plt.grid(True)
plt.legend(loc='best', framealpha=0.5,
    prop={'size':'small'})
plt.tight_layout(pad=1)
```

4. Save the figure and close it

```
plt.savefig('filename.png')
plt.close()
```

Alternatively, show the figure
With iPython, follow steps 1 to 3 above then

```
plt.show()    # Note: also closes the figure
```

Introducing the Figure and Axes classes

The Figure
Figure is the top-level container for everything on a canvas. It was obtained from the global Figure factory.

```
fig = plt.figure(num=None, figsize=None,
      dpi=None, facecolor=None,
      edgecolor=None)
```

num – integer or string identifier of figure
    if num exists, it is selected
    if num is None, a new one is allocated
figsize – tuple of (width, height) in inches
dpi – dots per inch
facecolor – background; edgecolor – border

Iterating over the open figures

```
for i in plt.get_fignums():
    fig = plt.figure(i) # get the figure
    print (fig.number)  # do something
```

Close a figure

```
plt.close(fig.number) # close figure
plt.close()      # close the current figure
plt.close(i)     # close figure numbered i
plt.close(name)  # close figure by str name
plt.close('all') # close all figures
```

An Axes or Subplot (a subclass of Axes)
An Axes is a container class for a specific plot. A figure may contain many Axes and/or Subplots. Subplots are laid out in a grid within the Figure. Axes can be placed anywhere on the Figure. There are a number of methods that yield an Axes, including:

```
ax = fig.add_subplot(2,2,1) # row-col-num
ax = fig.add_axes([0.1,0.1,0.8,0.8])
```

All at once
Use the subplots factory to get a Figure and Axes.

```
fig, ax = plt.subplots()
fig,(ax1,ax2,ax3) = plt.subplots(nrows=3,
    ncols=1, sharex=True, figsize=(8,4))
```

Iterating the Axes within a Figure

```
for ax in fig.get_axes():
    pass # do something
```

From an Axes get its Figure

```
fig = ax.figure
```

Remove an Axes from a Figure

```
fig.delaxes(ax)
```

Build a line plot with Matplotlib and raw (*x*, *y*) data

A single line plot in matplotlib

```
# --- fake up some data
x = np.linspace(0, 4*np.pi, 800)
y = np.sin(x)

# --- Select a style
plt.style.use('classic')

# --- get an empty Figure and add an Axes
fig = plt.figure(figsize=(8,4))
ax = fig.add_subplot(1,1,1) # row-col-num

# --- line plot data on the Axes
ax.plot(x, y, 'b-', linewidth=2,
    label=r'$y=\sin(x)$')

# --- add title and axis labels
ax.set_title('The Sine Wave')
ax.set_ylabel(r'$y$', fontsize=16)
ax.set_xlabel(r'$x$', fontsize=16)

# --- change the default plot limits
ax.set_xlim((-0.05, 4*np.pi + 0.05,))
ax.set_ylim((-1.05, 1.05))

# --- plot a legend in the best location
ax.legend(loc='best')

# --- add grid – not in default classic style
ax.grid(True)

# --- improve the layout
fig.tight_layout(pad=1)

# --- save and close
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Build a multi-line plot from raw data in matplotlib

A multi-line plot with markers and line-styles

```
# --- select a style
plt.style.use('classic')

# --- get the Figure and Axes all at once
fig, ax = plt.subplots(figsize=(8,6))

# --- plot some lines
N = 8 # the number of lines we will plot
styles =  ['-', '--', '-.', ':']
markers = list('+ox^psDv')
x = np.linspace(0, 100, 20)
for i in range(N): # add line-by-line
    y = x + x/5*i + i
    s = styles[i % len(styles)]
    m = markers[i % len(markers)]
    ax.plot(x, y,
      label='Line '+str(i+1)+' '+s+m,
      marker=m, linewidth=2, linestyle=s)

# --- add grid, legend, title and save
ax.grid(True)
ax.legend(loc='best', prop={'size':'large'})
ax.set_title('A Simple Line Plot')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Scatter plots – using matplotlib's ax.scatter()

A simple scatter plot

```
x = np.random.randn(100)
y = x + np.random.randn(100) + 10
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 3))
ax.scatter(x, y, alpha=0.5, color='orchid')
ax.set_title('Example Scatter Plot')
ax.grid(True)
fig.tight_layout(pad=1)
fig.savefig('filename1.png', dpi=125)
```

Add a regression line (using statsmodels)

```
import statsmodels.api as sm
x = sm.add_constant(x) # intercept
# Model: y ~ x + c
model = sm.OLS(y, x)
fitted = model.fit()
x_pred = np.linspace(x.min(), x.max(), 50)
x_pred2 = sm.add_constant(x_pred)
y_pred = fitted.predict(x_pred2)
ax.plot(x_pred, y_pred, '-',
    color='darkorchid', linewidth=2)
fig.savefig('filename2.png', dpi=125)
```

Add confidence bands for the regression line

```
y_hat = fitted.predict(x)
y_err = y - y_hat
mean_x = x.T[1].mean()
n = len(x)
dof = n - fitted.df_model - 1
from scipy import stats
t = stats.t.ppf(1-0.025, df=dof) # 2-tail
s_err = np.sum(np.power(y_err, 2))
conf = t * np.sqrt((s_err/(n-2))*(1.0/n +
    (np.power((x_pred-mean_x),2) /
    ((np.sum(np.power(x_pred,2))) -
    n*(np.power(mean_x,2))))))
upper = y_pred + abs(conf)
lower = y_pred - abs(conf)
ax.fill_between(x_pred, lower, upper,
    color='#888888', alpha=0.3)
fig.savefig('filename3.png', dpi=125)
```

Add a prediction interval for the regression line

```
from statsmodels.sandbox.regression.predstd\
    import wls_prediction_std
sdev, lower, upper =
wls_prediction_std(fitted,
    exog=x_pred2, alpha=0.05)
ax.fill_between(x_pred, lower, upper,
    color='#888888', alpha=0.1)
fig.savefig('filename4.png', dpi=125)
plt.close('all')
```

Note: The confidence interval relates to the location of the regression line. The predication interval relates to the location of data points around the regression line.

Quick scatter and regression plots using Seaborn

Use seaborn

```
import seaborn as sns

x = np.random.randn(100)
y = x + np.random.randn(100) + 10
df = DataFrame([x, y], index=['x', 'y']).T

g = sns.lmplot(x='x', y='y', data=df,
    fit_reg=True)
ax = g.axes[0, 0]
ax.set_title('Title')
fig = ax.figure
fig.set_size_inches(8, 3)
fig.tight_layout(pad=1)
fig.savefig("filename.png")
plt.close('all')
```

Scatter plots – more options

Changing the marker size and colour

```
# --- import a colour map
import matplotlib.cm as cm

# --- get some data
N = 100
x = np.random.rand(N)
y = np.random.rand(N)
size = ((np.random.rand(N) + 1) * 8) ** 2
colour = np.random.rand(N)

# --- plot
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 6))
l = ax.scatter(x, y, s=size, c=colour,
    cmap=cm.viridis) # choose a colormap

# --- attach the colour bar
clb = fig.colorbar(l,
    orientation='horizontal')
clb.ax.set_title('Colour bar title')

# --- do the size legend by hand ...
sz = [size.min(), size.mean(), size.max()]
handles = [ax.scatter([],[],s=sz[i]) for i in
    range(len(sz))]
labels = [str(int(round(x, 0))) for x in sz]
fig.legend(handles=handles, labels=labels,
    loc='upper right', scatterpoints=1,
    title='Size')

# --- grid, plot-limits, title and save
ax.grid(True)

ax.set_xlim((-0.05, 1.05))
ax.set_ylim((-0.05, 1.05))

ax.set_title('Dramatic Scatter Plot')

fig.tight_layout(pad=1);
fig.savefig('filename6.png', dpi=125)
plt.close('all')
```

Note: there are many colormaps to choose from.

Changing the marker symbol

```
# --- get the Figure and Axes classes
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8,8))

# --- add scatter plots
markers = list('ov^<>12348sphHdD+x*|_')
N = len(markers)
for i, m in enumerate(markers):
    x = np.arange(N)
    y = np.repeat(i+1, N)
    ax.scatter(x, y, marker=m, label=m,
        s=50, c='cornflowerblue')

# --- tidy up and save to file
ax.set_xlim((-1,N))
ax.set_ylim((0,len(markers)+1))
ax.legend(loc='upper left', ncol=3,
    prop={'size':'xx-large'},
    shadow=True, title='Marker Legend')
ax.get_legend().get_title().set_color("red")
ax.set_title('Markers ' +
    '(with an oversized legend)')
fig.tight_layout(pad=1);
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Bar plots – using ax.bar() and ax.barh()

A simple bar chart

```
# --- get the data
N = 5
labels = list('ABCDEFGHIJKLM'[0:N])
data = np.array(range(N)) + np.random.rand(N)

# --- plot the data
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 3))
width = 0.8
tickLocations = np.arange(N)
ax.bar(tickLocations, data, width,
    color='wheat', edgecolor='#8B7E66',
    linewidth=4.0, align='center')

# --- pretty-up the plot
ax.set_xticks(ticks= tickLocations)
ax.set_xticklabels(labels)
ax.set_xlim(min(tickLocations)-0.6,
    max(tickLocations)+0.6)
ax.set_yticks(range(N)[1:])
ax.set_ylim((0,N))
ax.yaxis.grid(True)

# --- title and save
ax.set_title("Bar Plot w/ Oversized Edges")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Side by side bar chart

```
# --- get the data
before = np.array([10, 11, 9, 12])
after = np.array([11, 12, 8, 17])
labels=['Group '+x for x in list('ABCD')]
# --- the plot – left then right
fig, ax = plt.subplots(figsize=(8, 5))
width = 0.4 # bar width
xlocs = np.arange(len(before))
ax.bar(xlocs-width, before, width,
     color='cornflowerblue', label='Males')
ax.bar(xlocs, after, width,
     color='hotpink', label='Females')
# --- labels, grids and title, then save
ax.set_xticks(ticks=range(len(before)))
ax.set_xticklabels(labels)
ax.yaxis.grid(True)
ax.legend(loc='best')
ax.set_ylabel('Mean Group Result')
ax.set_title('Group Results by Gender')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Stacked bar

```
# --- get some data
alphas = np.array( [23, 44, 52, 32] )
betas = np.array( [38, 49, 32, 61] )
labels = ['Sydney', 'Melb', 'Canb', 'Bris']

# --- the plot
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 5))
width = 0.8
xlocations = np.array(range(len(alphas)))
ax.bar(xlocations, alphas, width,
    label='alpha', color='tan',
    align='center')
ax.bar(xlocations, betas, width,
    label='beta', color='wheat',
align='center',
    bottom=alphas)

# --- pretty-up and save
ax.set_xticks(ticks=xlocations)
ax.set_xticklabels(labels)
ax.yaxis.grid(True)
ax.legend(loc='best', prop={'size':'small'})
ax.set_title("Stacked Nonsense")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Horizontal bar charts

```
# --- the data
labels = ['Males', 'Females', 'Persons']
data = [6.3, 7.2, 6.8]
y = np.arange(len(data))

# --- plot
width = 0.8
fig, ax = plt.subplots(figsize=(8, 3.5))
ax.barh(y, data, width, color='wheat',
    align='center')

# --- tidy-up and save
ax.set_yticks(y)
ax.set_yticklabels(labels)
ax.xaxis.grid(True)
ax.set_ylabel('Gender');
ax.set_xlabel('Rate (Percent)')
ax.set_title("Horizontal Nonsense")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Pie Chart – using ax.pie()

As nice as pie

```
# --- get some data
data = np.array([5,3,4,6])
labels = ['bats', 'cats', 'gnats', 'rats']
explode = (0, 0.1, 0, 0) # explode cats
colrs=['khaki', 'goldenrod', 'tan', 'wheat']

# --- the plot – then tidy-up and save
fig, ax = plt.subplots(figsize=(8, 3.5))
ax.pie(data, explode=explode,
    labels=labels, autopct='%1.1f%%',
    startangle=270, colors=colrs)
ax.axis('equal') # keep it a circle
ax.set_title("Title")
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Polar – using ax.plot()

Polar coordinates

```
# --- theta
theta = np.linspace(-np.pi, np.pi, 800)

# --- get us a Figure
fig = plt.figure(figsize=(8,4))

# --- left hand plot
ax = fig.add_subplot(1,2,1, polar=True)
r = 3 + np.cos(5*theta)
ax.plot(theta, r)
ax.set_yticks([1,2,3,4])

# --- right hand plot
ax = fig.add_subplot(1,2,2, polar=True)
r = (np.sin(theta)) - (np.cos(10*theta))
ax.plot(theta, r, color='green')
ax.set_yticks([1,2])

# --- title, explanatory text and save
fig.suptitle('Polar Coordinates')
fig.text(x=0.24, y=0.05,
   s=r'$r = 3 + \cos(5 \theta)$')
fig.text(x=0.64, y=0.05,
   s=r'$r = \sin(\theta) - \cos(10' +
       r'\theta)$')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Plot spines

Hiding the top and right spines

```
# --- the data
x = np.linspace(-np.pi, np.pi, 800)
y = np.sin(x)

# --- the plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, y, label='Sine', color='red')

# --- background and spines
ax.set_facecolor('#e5e5e5')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['left'].set_position(
    ('outward',10))
ax.spines['bottom'].set_position(
    ('outward',10))
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

# --- tidy up and save
# do the ax.grid() after setting ticks
ax.grid(b=True, which='both',
    color='white', linestyle='-',
    linewidth=1.5)
ax.set_axisbelow(True)
ax.legend(loc='best', frameon=False)
ax.set_title('The Sine Wave')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Spines in the middle

```
# --- the data
x = np.linspace(-np.pi, np.pi, 800)
y = np.sin(x)

# --- the plot
fig, ax = plt.subplots(figsize=(8, 4))

# --- the spines
ax.plot(x, y, label='Sine')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position((
    'data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position((
    'data',0))
ax.grid(b=True, which='both',
     color='#888888', linestyle='-',
     linewidth=0.5)

# --- tidy-up and save
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Legends

Legend within the plot

Use the 'loc' argument to place the legend

```
# --- get the empty Figure and Axes classes
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

# --- plot something
N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1), label='Line'+str(j))

# --- place the legend
ax.legend(loc='upper left')

# --- save and close
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Note: for legend placement, often loc='best' works perfectly as expected.

Legend slightly outside of the plot

```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))
N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

ax.legend(bbox_to_anchor=(1.05, 1.05))

fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Legend to the right of the plot

```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

fig.tight_layout(pad=1)
box = ax.get_position()  # Shrink plot
ax.set_position([box.x0, box.y0,
    box.width * 0.8, box.height])
ax.legend(bbox_to_anchor=(1, 0.5),
    loc='center left')   # Put legend
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Legend below the plot

```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

fig.tight_layout(pad=1)
box = ax.get_position()
ax.set_position([box.x0,
    box.y0 + box.height * 0.15,
    box.width, box.height * 0.85])
ax.legend(bbox_to_anchor=(0.5, -0.075),
    loc='upper center', ncol=N)

fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Multiple plots on a canvas

Using Axes to place a plot within a plot

```
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)

# --- Main Axes
ax = fig.add_axes([0.1,0.1,0.8,0.8])
ax.text(x=0.01, y=0.01, s='Main Axes',
    color='red', ha='left', va='bottom',
    fontsize=20)
ax.set_xticks([]); ax.set_yticks([])

# --- Insert Axes
ax= fig.add_axes([0.15,0.65,0.2,0.2])
ax.text(x=0.01, y=0.01, s='Insert Axes',
    color='blue', ha='left', va='bottom',
    fontsize=20)
ax.set_xticks([]); ax.set_yticks([])
fig.suptitle('An Axes within an Axes')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Figure

Simple subplot grid layouts

```
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)

for i in range(4):
    # fig.add_subplot(nrows, ncols, num)
    ax = fig.add_subplot(2, 2, i+1)
    ax.text(x=0.01, y=0.01,
        s='Subplot 2 2 '+str(i+1),
        color='red', ha='left',
        va='bottom', fontsize=20)
    ax.set_xticks([]); ax.set_yticks([])

ax.set_xticks([]); ax.set_yticks([])
fig.suptitle('Subplots')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Figure

Using GridSpec layouts (like list slicing)

```
import matplotlib.gridspec as gs
gs = gs.GridSpec(3, 3) # nrows, ncols
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)
ax1 = fig.add_subplot(gs[0, :]) # row,col
ax1.text(x=0.2,y=0.2,s='0, :', color='b')
ax2 = fig.add_subplot(gs[1,:-1])
ax2.text(x=0.2,y=0.2,s='1, :-1', color='b')
ax3 = fig.add_subplot(gs[1:, -1])
ax3.text(x=0.2,y=0.2, s='1:, -1', color='b')
ax4 = fig.add_subplot(gs[-1,0])
ax4.text(x=0.2,y=0.2, s='-1, :0', color='b')
ax5 = fig.add_subplot(gs[-1,-2])
ax5.text(x=0.2,y=0.2, s='-1,:-2', color='b')
for a in fig.get_axes():
    a.set_xticks([])
    a.set_yticks([])

fig.suptitle('GridSpec Layout')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Figure

Style

Matplotlib has many styles
There are many different styles to choose from.

```
print(plt.style.available)
```

So let's choose one.

```
plt.style.use('ggplot')
```

Example line chart in three different styles

```
# --- fake up some data
x = pd.period_range('1980-01-01',
    periods=410, freq='M')
y = np.random.randn(len(x)).cumsum()
df = pd.DataFrame(y, index=x,
    columns=['Fake'])

# --- plot data using different styles
styles = ['classic', 'ggplot',
    'fivethirtyeight']
for style in styles:
    plt.style.use(style)
    ax = df.plot()
    ax.set_title('Style: ' + style)
    ax.set_xlabel('X Axis')
    ax.set_ylabel('Y Axis')
    fig = ax.figure
    fig.set_size_inches(8, 3)
    fig.tight_layout(pad=1)
    fig.text(0.99, 0.01, 'footnote',
        ha='right', va='bottom',
        fontsize=9, fontstyle='italic',
        color='#999999')
    fig.savefig('f-'+style+'.png', dpi=125)
    plt.close('all')
```

Roll your own style sheet

The Mark Graph Style Sheet

```
font.family: sans-serif
font.style: normal
font.variant: normal
font.weight: medium
font.stretch: normal
font.sans-serif: Helvetica, Arial, Avant
Garde, sans-serif
font.size: 14.0

lines.linewidth: 2.0
lines.solid_capstyle: butt

legend.fancybox: true
legend.fontsize: x-small
legend.framealpha: 0.5

axes.prop_cycle: cycler('color', ['ef5962',
    '5a9dd2', '7dc16d', 'f9a560', '9d69aa',
    'cd6f5a', 'd680b3', '737373'])
axes.facecolor: white
axes.titlesize: large  # fontsize of the axes
title
axes.labelsize: medium # x and y label size
axes.labelcolor: 111111
axes.axisbelow: true
axes.grid: true
axes.edgecolor: white
axes.linewidth: 0.01

patch.edgecolor: white
patch.linewidth: 0.01

svg.fonttype: path

grid.linestyle: -
grid.linewidth: 0.5
grid.color: e7e7e7

xtick.major.size: 0
xtick.minor.size: 0
xtick.labelsize: small
xtick.color: 333333
ytick.major.size: 0
ytick.minor.size: 0
ytick.labelsize: small
ytick.color: 333333

figure.figsize: 8, 4 # inches
figure.facecolor: white

text.color: black

savefig.edgecolor: white
savefig.facecolor: white
```

Which is saved as "markgraph.mplstyle"

Using your own style sheet

```
plt.style.use('./markgraph.mplstyle')
```

Cautionary notes

This cheat sheet was cobbled together by bots roaming the dark recesses of the Internet seeking ursine and pythonic myths. There is no guarantee the narratives were captured and transcribed accurately. You use these notes at your own risk. You have been warned.

Cheat Sheet: The pandas DataFrame Object

Preliminaries

Always start by importing these Python modules

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import DataFrame, Series
```

These are the standard import aliases most programmers use with Python pandas.

Check which version of pandas you are using

```
print(pd.__version__)
```

This cheat sheet was written for pandas version 0.23 and Python version 3.6.

The conceptual model

The **DataFrame object** is a two-dimensional table of data with column and row indexes (something like a spread sheet). The columns are made up of Series objects (see below).

A DataFrame has two Indexes:

- Typically, the column index (df.columns) is a list of strings (variable names) or (less commonly) integers
- Typically, the row index (df.index) might be:
  - Integers – for case or row numbers;
  - Strings – for case names; or
  - DatetimeIndex or PeriodIndex – for time series

The **Series object** is an ordered, one-dimensional array of data with an index. All the data is of the same data type. Series arithmetic is vectorised after first aligning the Series index for each of the operands.

Examples of Series Arithmatic

```
s1 = Series(range(0, 4))    # -> 0, 1, 2, 3
s2 = Series(range(1, 5))    # -> 1, 2, 3, 4
s3 = s1 + s2                # -> 1, 3, 5, 7

s4 = Series([1, 2, 3], index=[0, 1, 2])
s5 = Series([1, 2, 3], index=[2, 1, 0])
s6 = s4 + s5                # -> 4, 4, 4

s7 = Series([1, 2, 3], index=[1, 2, 3])
s8 = Series([1, 2, 3], index=[0, 1, 2])
s9 = s7 + s8                # NAN, 3, 5, NAN
```

Get your data into a DataFrame

Instantiate an empty DataFrame

```
df = DataFrame()
```

Load a DataFrame from a CSV file

```
df = pd.read_csv('file.csv') # often works
df = pd.read_csv('file.csv', header=0,
    index_col=0, quotechar='"', sep=':',
    na_values = ['na', '-', '.', ''])
```

Note: refer to pandas docs for all arguments

Get data from inline CSV text to a DataFrame

```
from io import StringIO
data = """, Animal, Cuteness, Desirable
row-1,      dog,    8.7,  True
row-2,      cat,    9.5,  True
row-3,      bat,    2.6,  False"""
df = pd.read_csv(StringIO(data), header=0,
    index_col=0, skipinitialspace=True)
```

Note: *skipinitialspace=True* allows for a pretty layout

Load DataFrames from a Microsoft Excel file

```
# Each Excel sheet in a Python dictionary
workbook = pd.ExcelFile('file.xlsx')
d = {} # start with an empty dictionary
for sheet_name in workbook.sheet_names:
    df = workbook.parse(sheet_name)
    d[sheet_name] = df
```

Note: the parse() method takes many arguments like read_csv() above. Refer to the pandas documentation.

Load a DataFrame from a MySQL database

```
import pymysql
from sqlalchemy import create_engine
engine = create_engine('mysql+pymysql://' +
    'USER:PASSWORD@HOST/DATABASE')
df = pd.read_sql_table('table', engine)
```

DataFrame from Series as rows (using a Python list)

```
s1 = Series(range(6))
s2 = s1 * s1
df = DataFrame([s1, s2])
df.index = ['n', 'n-squared']
```

DataFrame from Series as cols (using a dictionary)

```
s1 = Series({'Tom':1, 'Dick':2, 'Harry':3})
s2 = Series({'Tom':9, 'Dick':8, 'Mary':7})
df = DataFrame({'A': s1, 'B': s2 })
```

Note: misaligned rows; row order can change

DataFrame from Series as cols (using pd.concat())

```
df = pd.concat([s1, s2], axis=1, sort=False)
```

DataFrame from Series as cols (using transpose)

```
df = DataFrame([s1, s2], index=['A','B']).T
```

Note: data should all be of the same type

DataFrame from row data in a Python dictionary

```
# --- use helper method for data in rows
df = DataFrame.from_dict({ # data by row:
        # rows as python dictionaries
        'row0' : {'col0':0, 'col1':'A'},
        'row1' : {'col0':1, 'col1':'B'}
    }, orient='index')

df = DataFrame.from_dict({ # data by row:
        # rows as python lists
        'row0' : [1, 1+1j, 'A'],
        'row1' : [2, 2+2j, 'B']
    }, orient='index')
```

DataFrame of fake data – useful for testing

```
df = DataFrame(np.random.rand(500,5),
        columns=list('ABCDE'))
```

DataFrame of fake time-series data

```
df = DataFrame(np.random.rand(500,5)) - 0.5
df = df.cumsum()
df.index = pd.date_range('1/1/2017',
                periods=len(df), freq='D')
```

Fake data with alphabetic index and group variable

```
import string
import random
rows = 52
cols = 5
assert(1 <= rows <= 52) # min/max row count
df = DataFrame(np.random.randn(rows, cols),
   columns=['c'+str(i) for i in range(cols)],
   index=list((string.ascii_uppercase +
         string.ascii_lowercase)[0:rows]))
df['groupable'] = [random.choice('abcde')
    for _ in range(rows)]
```

Saving a DataFrame

Saving a DataFrame to a CSV file

```
df.to_csv('name.csv', encoding='utf-8')
```

Saving DataFrames to an Excel Workbook

```
from pandas import ExcelWriter
writer = ExcelWriter('filename.xlsx')
df1.to_excel(writer,'Sheet1')
df2.to_excel(writer,'Sheet2')
writer.save()
```

Saving a DataFrame to MySQL

```
import pymysql
from sqlalchemy import create_engine
e = create_engine('mysql+pymysql://' +
    'USER:PASSWORD@HOST/DATABASE')
df.to_sql('TABLE', e, if_exists='replace')
```

Note: if_exists → 'fail', 'replace', 'append'

Saving to Python objects

```
d = df.to_dict()        # to dictionary
str = df.to_string()    # to string
m = df.as_matrix()      # to numpy matrix
```

Working with the whole DataFrame

Peek at the DataFrame contents/structure

```
df.info()           # index & data types
dfh = df.head(i)    # get first i rows
dft = df.tail(i)    # get last i rows
dfs = df.describe() # summary stats cols
top_left_corner_df = df.iloc[:4, :4]
```

DataFrame non-indexing attributes

```
df = df.T       # transpose rows and cols
l = df.axes     # list row and col indexes
(r_idx, c_idx) = df.axes      # from above
s = df.dtypes   # Series column data types
b = df.empty    # True for empty DataFrame
i = df.ndim     # number of axes (it is 2)
t = df.shape    # (row-count, column-count)
i = df.size     # row-count * column-count
a = df.values   # get a numpy array for df
```

DataFrame utility methods

```
df = df.copy()  # copy a DataFrame
df = df.sort_values(by='col') # axis=1 cols
df = df.sort_values(by=['col1', 'col2'])
df = df.sort_values(by='row', axis=1)
df = df.sort_index()  # axis=1 to sort cols
df = df.astype(dtype) # type conversion
```

DataFrame iteration methods

```
df.iteritems()  # (col-index, Series) pairs
df.iterrows()   # (row-index, Series) pairs
# example ... iterating over columns ...
for (name, series) in df.iteritems():
    print('\nCol name: ' + str(name))
    print('1st value: ' + str(series.iat[0]))
```

Maths on the whole DataFrame (not a complete list)

```
df = df.abs()     # absolute values
df = df.add(o)    # add df, Series or value
s =  df.count()   # non NA/null values
df = df.cummax()  # (cols default axis)
df = df.cummin()  # (cols default axis)
df = df.cumsum()  # (cols default axis)
df = df.diff()    # 1st diff (col def axis)
df = df.div(o)    # div by df, Series, value
df = df.dot(o)    # matrix dot product
s =  df.max()     # max of axis (col def)
s =  df.mean()    # mean (col default axis)
s =  df.median()  # median (col default)
s =  df.min()     # min of axis (col def)
df = df.mul(o)    # mul by df Series val
s =  df.sum()     # sum axis (cols default)
df = df.where(df > 0.5, other=np.nan)
```

Note: methods returning a series default to work on cols

Select/filter rows/cols based on index label values

```
df = df.filter(items=['a', 'b'])  # by col
df = df.filter(items=[5], axis=0) # by row
df = df.filter(like='x') # keep x in col
df = df.filter(regex='x') # regex in col
df = df.select(lambda x: not x%5) # 5th rows
```

Note: select takes a Boolean function, for cols: axis=1
Note: filter defaults to cols; select defaults to rows

Working with Columns (and pandas Series)

Peek at the column/Series structure/contents

```
sh = df['col'].head(i) # get first i elements
st = df['col'].tail(i) # get last i elements
s = df['col'].describe() # summary stats
```

Get column index and labels

```
idx = df.columns         # get col index
label = df.columns[0]    # first col label
l = df.columns.tolist() # list of col labels
a = df.columns.values    # array of col labels
```

Change column labels

```
df = df.rename(columns={'old':'new','a':'1'})
df.columns = ['new1', 'new2', 'new3'] # etc.
```

Selecting columns

```
s =  df['colName']      # select col to Series
df = df[['colName']]    # select col to df
df = df[['a','b']]      # select 2-plus cols
df = df[['c','a','b']] # change col order
s =  df[df.columns[0]] # select by number
df = df[df.columns[[0, 3, 4]]] # by numbers
df = [df.columns[:-1]] # all but last col
s =  df.pop('c')        # get & drop from df
```

Selecting columns with Python attributes

```
s = df.a     # same as s = df['a']
df.existing_column = df.a / df.b
df['new_column'] = df.a / df.b
```

Trap: column names must be valid Python identifiers.
Trap: cannot create new columns by attribute.

Adding new columns to a DataFrame

```
df['new_col'] = range(len(df))
df['new_col'] = np.repeat(np.nan,len(df))
df['random'] = np.random.rand(len(df))
df['index_as_col'] = df.index
df1[['b','c']] = df2[['e','f']]
```

Trap: When adding a new column, only items from the new column series that have a corresponding index in the DataFrame will be added. The index of the receiving DataFrame is not extended to accommodate all of the new series.
Trap: when adding a python list or numpy array, the column will be added by integer position.

Add a mismatched column with an extended index

```
df = DataFrame([1, 2, 3], index=[1, 2, 3])
s = Series([2, 3, 4], index=[2, 3, 4])
df = df.reindex(df.index.union(s.index))
df['s'] = s # with NaNs where no data
```

Note: assumes unique index values

Dropping (deleting) columns (mostly by label)

```
df = df.drop('col1', axis=1)
df = df.drop(['col1', 'col2'], axis=1)
del df['col'] # even classic python works
df = df.drop(df.columns[0], axis=1)   #first
df = df.drop(df.columns[-1:], axis=1) #last
```

Swap column contents

```
df[['B', 'A']] = df[['A', 'B']]
```

Vectorised arithmetic on columns

```
df['proportion']=df['count']/df['total']
df['percent'] = df['proportion'] * 100.0
```

Apply numpy mathematical functions to columns

```
df['log_data'] = np.log(df['col1'])
```

Note: many many more numpy math functions
Hint: Prefer pandas math over numpy where you can.

Set column values set based on criteria

```
df['b'] = df['a'].where(df['a']>0, other=0)
df['d'] = df['a'].where(df.b!=0, other=df.c)
```

Note: where other can be a Series or a scalar

Data type conversions

```
df['float'] =  df['col'].astype(float)
df['int'] =  df['col'].astype(int)
s = df['col'].astype(str) # Series dtype
a = df['col'].values       # numpy array
l = df['col'].tolist()     # python list
```

Trap: index lost in conversion from Series to array or list

Common column-wide methods/attributes

```
value = df['col'].dtype    # type of data
value = df['col'].size     # col dimensions
value = df['col'].count()  # non-NA count
value = df['col'].sum()
value = df['col'].prod()
value = df['col'].min()
value = df['col'].max()
value = df['col'].mean()   # also median()
value = df['col'].cov(df['col2'])
s =     df['col'].describe()
s =     df['col'].value_counts()
```

Find first index label for min/max values in column

```
label = df['col1'].idxmin()
label = df['col1'].idxmax()
```

Common column element-wise methods

```
s = df['col'].isnull()
s = df['col'].notnull()    # not isnull()
s = df['col'].astype(float)
s = df['col'].abs()
s = df['col'].round(decimals=0)
s = df['col'].diff(periods=1)
s = df['col'].shift(periods=1)
s = df['col'].to_datetime()
s = df['col'].fillna(0)    # replace NaN w 0
s = df['col'].cumsum()
s = df['col'].cumprod()
s = df['col'].pct_change(periods=4)
s = df['col'].rolling(window=4,
    min_periods=4, center=False).sum()
```

Append a column of row sums to a DataFrame

```
df['Total'] = df.sum(axis=1)
```

Note: also means, mins, maxs, etc.

Multiply every column in DataFrame by a Series

```
df = df.mul(s, axis=0)  # on matched rows
```

Note: also add, sub, div, etc.

Selecting columns with .loc, .iloc

```
df = df.loc[:, 'col1':'col2'] # inclusive
df = df.iloc[:, 0:2]          # exclusive
```

Get the integer position of a column index label

```
i = df.columns.get_loc('col_name')
```

Test if column index values are unique/monotonic

```
if df.columns.is_unique: pass # ...
b = df.columns.is_monotonic_increasing
b = df.columns.is_monotonic_decreasing
```

Mapping a DataFrame column or Series

```
map = Series(['red', 'green', 'blue'],
        index=['r', 'g', 'b'])
s = Series(['r', 'g', 'r', 'b']).map(map)
# s contains: ['red', 'green', 'red', 'blue']

m = Series([True, False], index=['Y', 'N'])
df = DataFrame(np.random.choice(list('YN'),
        500, replace=True), columns=['col'])
df['col'] = df['col'].map(m)
```

Note: Useful for decoding data before plotting
Note: Sometimes referred to as a lookup function
Note: Indexes can also be mapped if needed.

Find the largest and smallest values in a column

```
s = df['A'].nlargest(5)
s = df['A'].nsmallest(5)
```

Sorting the columns of a DataFrame

```
df = df.sort_index(axis=1, ascending=False)
```

Working with rows

Get the row index and labels

```
idx = df.index          # get row index
label = df.index[0]     # first row label
label = df.index[-1]    # last row label
l = df.index.tolist()   # get as a list
a = df.index.values     # get as an array
```

Change the (row) index

```
df.index = idx             # new ad hoc index
df = df.set_index('A')     # col A new index
df = df.set_index(['A', 'B']) # MultiIndex
df = df.reset_index()      # replace old w new
# note: old index stored as a col in df
df.index = range(len(df)) # set with list
df = df.reindex(index=range(len(df)))
df = df.set_index(keys=['r1','r2','etc'])
```

Adding rows

```
df = original_df.append(more_rows_in_df)
```

Hint: convert row to a DataFrame and then append.
Both DataFrames must have same column labels.

Dropping rows (by name)

```
df = df.drop('row_label')
df = df.drop(['row1','row2']) # multi-row
```

Boolean row selection by values in a column

```
df = df[df['col2'] >= 0.0]
df = df[(df['col3']>=1.0) | (df['col1']<0.0)]
df = df[df['col'].isin([1,2,5,7,11])]
df = df[~df['col'].isin([1,2,5,7,11])]
df = df[df['col'].str.contains('hello')]
```

Trap: bitwise "or", "and" “not; (ie. | & ~) co-opted to be Boolean operators on a Series of Boolean
Trap: need parentheses around comparisons.

Selecting rows using isin over multiple columns

```
# fake up some data
data = {1:[1,2,3], 2:[1,4,9], 3:[1,8,27]}
df = DataFrame(data)

# multi-column isin
lf = {1:[1, 3], 3:[8, 27]} # look for
f = df[df[list(lf)].isin(lf).all(axis=1)]
```

Selecting rows using an index

```
idx = df[df['col'] >= 2].index
print(df.loc[idx])
```

Select a slice of rows by integer position

[inclusive-from : exclusive-to [: step]]
start is 0; end is len(df)

```
df = df[:]       # copy entire DataFrame
df = df[0:2]     # rows 0 and 1
df = df[2:3]     # row 2 (the third row)
df = df[-1:]     # the last row
df = df[:-1]     # all but the last row
df = df[::2]     # every 2nd row (0 2 ..)
```

Trap: a single integer without a colon is a column label for integer numbered columns.

Select a slice of rows by label/index

```
df = df['a':'c'] # rows 'a' through 'c'
```

Note: [inclusive-from : inclusive–to [: step]]
Trap: cannot work for integer labelled rows – see previous code snippet on integer position slicing.

Append a row of column totals to a DataFrame

```
# Option 1: use dictionary comprehension
sums = {col: df[col].sum() for col in df}
sums_df = DataFrame(sums,index=['Total'])
df = df.append(sums_df)

# Option 2: All done with pandas
df = df.append(DataFrame(df.sum(),
                    columns=['Total']).T)
```

Iterating over DataFrame rows

```
for (index, row) in df.iterrows(): # pass
```

Trap: row data type may be coerced.

Sorting the rows of a DataFrame by the row index

```
df = df.sort_index(ascending=False)
```

Sorting DataFrame rows based on column values

```
df = df.sort_values(by=df.columns[0],
                         ascending=False)
df = df.sort_values(by=['col1', 'col2'])
```

Random selection of rows

```
import random as r
k = 20 # pick a number
selection = r.sample(range(len(df)), k)
df_sample = df.iloc[selection, :] # get copy
```

Note: this randomly selected sample is not sorted

Drop duplicates in the row index

```
df['index'] = df.index    # 1 create new col
df = df.drop_duplicates(cols='index',
        take_last=True)   # 2 use new col
del df['index']           # 3 del the col
df = df.sort_index()      # 4 tidy up
```

Test if two DataFrames have same row index

```
len(a)==len(b) and all(a.index==b.index)
```

Get the integer position of a row or col index label

```
i = df.index.get_loc('row_label')
```

Trap: index.get_loc() returns an integer for a unique match. If not a unique match, may return a slice/mask.

Get integer position of rows that meet condition

```
a = np.where(df['col'] >= 2) #numpy array
```

Test if the row index values are unique/monotonic

```
if df.index.is_unique: pass # ...
b = df.index.is_monotonic_increasing
b = df.index.is_monotonic_decreasing
```

Find row index duplicates

```
if df.index.has_duplicates:
    print(df.index.duplicated())
```

Note: also similar for column label duplicates.

Working with cells

Getting a cell by row and column labels

```
value = df.at['row', 'col']
value = df.loc['row', 'col']
value = df['col'].at['row']      # tricky
```

Note: .at[] fastest label based scalar lookup
Note: at[] does not take slices as an argument

Setting a cell by row and column labels

```
df.at['row', 'col'] = value
df.loc['row', 'col'] = value
df['col'].at['row'] = value      # avoid!
```

Getting and slicing on labels

```
df = df.loc['row1':'row3', 'col1':'col3']
```

Note: the "to" on this slice is inclusive.

Setting a cross-section by labels

```
df.loc['A':'C', 'col1':'col3'] = np.nan
df.loc[1:2,'col1':'col2'] = np.zeros((2,2))
df.loc[1:2,'A':'C'] = othr.loc[1:2,'A':'C']
```

Remember: inclusive "to" in the slice

Getting a cell by integer position

```
value = df.iat[9, 3]         # [row, col]
value = df.iloc[0, 0]        # [row, col]
value = df.iloc[len(df)-1, len(df.columns)-1]
```

Getting a range of cells by int position

```
df = df.iloc[2:4, 2:4] # subset of the df
df = df.iloc[:5, :5]   # top left corner
s = df.iloc[5, :]      # return row as Series
df = df.iloc[5:6, :]   # returns row as row
```

Note: exclusive "to" – same as python list slicing.

Setting cell by integer position

```
df.iloc[0, 0] = value        # [row, col]
df.iat[7, 8] = value
```

Setting cell range by integer position

```
df.iloc[0:3, 0:5] = value
df.iloc[1:3, 1:4] = np.ones((2, 3))
df.iloc[1:3, 1:4] = np.zeros((2, 3))
df.iloc[1:3, 1:4] = np.array([[1, 1, 1],
                              [2, 2, 2]])
```

Remember: exclusive-to in the slice

Views and copies

From the manual: Setting a copy can cause subtle errors. The rules about when a view on the data is returned are dependent on NumPy. Whenever an array of labels or a Boolean vector are involved in the indexing operation, the result will be a copy.

Summary: selecting using the DataFrame index

Using the DataFrame index to select columns

```
s =  df['col_label']         # returns Series
df = df[['col_label']]    # returns DataFrame
df = df[['L1', 'L2']] # select cols with list
df = df[index]    # select cols with an index
df = df[s]     # select with col label Series
```

Note: scalar returns Series; list &c returns a DataFrame.

Using the DataFrame index to select rows

```
df = df['from':'inc_to']  # label slice
df = df[3:7]              # integer slice
df = df[df['col'] > 0.5]  # Boolean Series
df = df.loc['label']      # single label
df = df.loc[container]    # lab list/Series
df = df.loc['from':'to']  # inclusive slice
df = df.loc[bs]           # Boolean Series
df = df.iloc[0]           # single integer
df = df.iloc[container]   # int list/Series
df = df.iloc[0:5]         # exclusive slice
```

Trap: Boolean Series gets rows, label Series gets cols.

Using the DataFrame index to select a cross-section

```
# r and c can be scalar, list, slice
df.loc[r, c]   # label accessor (row, col)
df.iloc[r, c]  # integer accessor
df[c].iloc[r]  # chained – also for .loc
```

Using the DataFrame index to select a cell

```
# r and c must be label or integer
df.at[r, c]    # fast scalar label accessor
df.iat[r, c]   # fast scalar int accessor
df[c].iat[r]   # chained – also for .at
```

DataFrame indexing methods

```
v = df.get_value(r, c)    # get by row, col
df = df.set_value(r,c,v)  # set by row, col
df = df.xs(key, axis)     # get cross-section
df = df.filter(items, like, regex, axis)
df = df.select(crit, axis)
```

Note: the indexing attributes (.loc, .iloc, .at .iat) can be used to get and set values in the DataFrame.
Note: the .loc, and iloc indexing attributes can accept python slice objects. But .at and .iat do not.
Note: .loc can also accept Boolean Series arguments
Avoid: chaining in the form df[col_indexer][row_indexer]
Trap: label slices are inclusive, integer slices exclusive.

Some index attributes and methods

```
b = idx.is_monotonic_decreasing
b = idx.is_monotonic_increasing
b = idx.has_duplicates
i = idx.nlevels          # num of index levels
idx = idx.astype(dtype)# change data type
b = idx.equals(o)        # check for equality
idx = idx.union(o)       # union of two indexes
i = idx.nunique()        # number unique labels
label = idx.min()        # minimum label
label = idx.max()        # maximum label
```

Joining/Combining DataFrames

Three ways to join two DataFrames:

- merge (a database/SQL-like join operation)
- concat (stack side by side or one on top of the other)
- combine_first (splice the two together, choosing values from one over the other)

Merge on (row) indexes

```
df_new = pd.merge(left=df1, right=df2,
          how='outer', left_index=True,
          right_index=True)
```

How: 'left', 'right', 'outer', 'inner'
How: outer=union/all; inner=intersection

Merge on columns

```
df_new = pd.merge(left=df1, right=df2,
          how='left', left_on='col1',
          right_on='col2')
```

Trap: When joining on columns, the indexes on the passed DataFrames are ignored.
Trap: many-to-many merges on a column can result in an explosion of associated data.

Join on indexes (another way of merging)

```
df_new = df1.join(other=df2, on='col1',
                  how='outer')
df_new = df1.join(other=df2,on=['a','b'],
                  how='outer')
```

Note: DataFrame.join() joins on indexes by default. DataFrame.merge() joins on common columns by default.

Simple concatenation is often the best

```
df=pd.concat([df1,df2],axis=0)#top/bottom
df = df1.append([df2, df3])   #top/bottom
df=pd.concat([df1,df2],axis=1)#left/right
```

Trap: can end up with duplicate rows or cols
Note: concat has an ignore_index parameter

Combine_first

```
df = df1.combine_first(other=df2)

# multi-combine with python reduce()
df = reduce(lambda x, y:
        x.combine_first(y),
            [df1, df2, df3, df4, df5])
```

Uses the non-null values from df1. The index of the combined DataFrame will be the union of the indexes from df1 and df2.

Groupby: Split-Apply-Combine

Grouping

```
gb = df.groupby('cat') # by one columns
gb = df.groupby(['c1','c2']) # by 2 cols
gb = df.groupby(level=0) # multi-index gb
gb = df.groupby(level=['a','b']) # mi gb
print(gb.groups)
```

Note: groupby() returns a pandas groupby object

Note: the groupby object attribute .groups contains a dictionary mapping of the groups.
Trap: NaN values in the group key are automatically dropped – there will never be a NA group.

The pandas "groupby" mechanism allows us to split the data into groups, apply a function to each group independently and then combine the results.

Iterating groups – usually not needed

```
for name, group in gb:
    print (name, group)
```

Selecting a group

```
dfa = df.groupby('cat').get_group('a')
dfb = df.groupby('cat').get_group('b')
```

Applying an aggregating function

```
# apply to a column ...
s = df.groupby('cat')['col1'].sum()
s = df.groupby('cat')['col1'].agg(np.sum)
# apply to the every column in DataFrame
s = df.groupby('cat').agg(np.sum)
df_summary = df.groupby('cat').describe()
df_row_1s = df.groupby('cat').head(1)
```

Note: aggregating functions reduce the dimension by one – they include: mean, sum, size, count, std, var, sem, describe, first, last, min, max

Applying multiple aggregating functions

```
gb = df.groupby('cat')
# apply multiple functions to one column
dfx = gb['col2'].agg([np.sum, np.mean])
# apply to multiple fns to multiple cols
dfy = gb.agg({
    'cat': np.count_nonzero,
    'col1': [np.sum, np.mean, np.std],
    'col2': [np.min, np.max]
})
```

Note: gb['col2'] above is shorthand for df.groupby('cat')['col2'], without the need for regrouping.

Transforming functions

```
# transform to group z-scores, which have
# a group mean of 0, and a std dev of 1.
zscore = lambda x: (x-x.mean())/x.std()
dfz = df.groupby('cat').transform(zscore)
# replace missing data with group mean
mean_r = lambda x: x.fillna(x.mean())
dfm = df.groupby('cat').transform(mean_r)
```

Note: can apply multiple transforming functions in a manner similar to multiple aggregating functions above,

Applying filtering functions

Filtering functions allow you to make selections based on whether each group meets specified criteria

```
# select groups with more than 10 members
eleven = lambda x: (len(x['col1']) >= 11)
df11 = df.groupby('cat').filter(eleven)
```

Group by a row index (non-hierarchical index)

```
df = df.set_index(keys='cat')
s = df.groupby(level=0)['col1'].sum()
dfg = df.groupby(level=0).sum()
```

Pivot Tables: working with long and wide data

These features work with and often create hierarchical or multi-level Indexes; (the pandas MultiIndex is powerful and complex).

Pivot, unstack, stack and melt

Pivot tables move from long format to wide format data

```
# Let's start with data in long format
from StringIO import StringIO # python2.7
#from io import StringIO      # python 3
data = """Date,Pollster,State,Party,Est
13/03/2014, Newspoll, NSW,   red,   25
13/03/2014, Newspoll, NSW,   blue,  28
13/03/2014, Newspoll, Vic,   red,   24
13/03/2014, Newspoll, Vic,   blue,  23
13/03/2014, Galaxy,   NSW,   red,   23
13/03/2014, Galaxy,   NSW,   blue,  24
13/03/2014, Galaxy,   Vic,   red,   26
13/03/2014, Galaxy,   Vic,   blue,  25
13/03/2014, Galaxy,   Qld,   red,   21
13/03/2014, Galaxy,   Qld,   blue,  27"""
df = pd.read_csv(StringIO(data),
      header=0, skipinitialspace=True)

# pivot to wide format on 'Party' column
# 1st: set up a MultiIndex for other cols
df1 = df.set_index(['Date', 'Pollster',
    'State'])
# 2nd: do the pivot
wide1 = df1.pivot(columns='Party')

# unstack to wide format on State / Party
# 1st: MultiIndex all but the Values col
df2 = df.set_index(['Date', 'Pollster',
    'State', 'Party'])
# 2nd: unstack a column to go wide on it
wide2 = df2.unstack('State')
wide3 = df2.unstack() # pop last index

# Use stack() to get back to long format
long1 = wide1.stack()
# Then use reset_index() to remove the
# MultiIndex.
long2 = long1.reset_index()

# Or melt() back to long format
# 1st: flatten the column index
wide1.columns = ['_'.join(col).strip()
    for col in wide1.columns.values]
# 2nd: remove the MultiIndex
wdf = wide1.reset_index()
# 3rd: melt away
long3 = pd.melt(wdf, value_vars=
    ['Est_blue', 'Est_red'],
    var_name='Party', id_vars=['Date',
    'Pollster', 'State'])
```

Note: See documentation, there are many arguments to these methods.

Working with dates, times and their indexes

Dates and time – points and spans

With its focus on time-series data, pandas has a suite of tools for managing dates and time: either as a point in time (a Timestamp) or as a span of time (a Period).

```
t = pd.Timestamp('2013-01-01')
t = pd.Timestamp('2013-01-01 21:15:06')
t = pd.Timestamp('2013-01-01 21:15:06.7')
p = pd.Period('2013-01-01', freq='M')
```

Note: Timestamps can range from 1678 to 2261. (Check pd.Timestamp.max and pd.Timestamp.min).

A Series of Timestamps or Periods

```
ts = ['2015-04-01', '2014-04-02']

# Series of Timestamps
s = pd.to_datetime(pd.Series(ts))

# Series of Periods
s = s.dt.to_period('M') # from Timestamps
```

Note: While Periods make a very useful index; they may be less useful in a Series.
Trap: pd.to_datetime(list_of_timestamp_strings) returns a pandas DatetimeIndex object.

From non-standard strings to Timestamps

```
t = ['09:08:55.7654-JAN092002',
        '15:42:02.6589-FEB082016']
s = pd.Series(pd.to_datetime(t,
        format="%H:%M:%S.%f-%b%d%Y"))
```

Also: %B = full month name; %m = numeric month; %y = year without century; and more …

Dates and time – stamps and spans as indexes

An index of Timestamps is a DatetimeIndex.
An index of Periods is a PeriodIndex.

```
date_strs = ['2018-01-01', '2018-04-01',
             '2018-07-01', '2018-10-01']

dti = pd.DatetimeIndex(date_strs)

pid = pd.PeriodIndex(date_strs, freq='D')
pim = pd.PeriodIndex(date_strs, freq='M')
piq = pd.PeriodIndex(date_strs, freq='Q')

print (pid[1] - pid[0])  # 90  [days]
print (pim[1] - pim[0])  #  3  [months]
print (piq[1] - piq[0])  #  1  [quarter]

time_strs = ['2015-01-01 02:10:40.12345',
             '2015-01-01 02:10:50.67890']
pis = pd.PeriodIndex(time_strs, freq='U')

df.index = pd.period_range('2015-01',
    periods=len(df), freq='M')

dti = pd.to_datetime(['04-01-2012'],
    dayfirst=True) # Australian date format
pi = pd.period_range('1960-01-01',
    '2015-12-31', freq='M')
```

Hint: unless you are working in less than seconds, prefer PeriodIndex over DatetimeImdex.

Period frequency constants (not a complete list)

Name	Description
U	Microsecond
L	Millisecond
S	Second
T	Minute
H	Hour
D	Calendar day
B	Business day
W-{MON, TUE, …}	Week ending on …
MS	Calendar start of month
M	Calendar end of month
QS-{JAN, FEB, …}	Quarter start with year starting (QS – December)
Q-{JAN, FEB, …}	Quarter end with year ending (Q – December)
AS-{JAN, FEB, …}	Year start (AS - December)
A-{JAN, FEB, …}	Year end (A - December)

DatetimeIndex from DataFrame columns

```
datecols = ['year', 'month', 'day']
df.index = pd.to_datetime(df[datecols])
```

From DatetimeIndex to Python datetime objects

```
dti = pd.DatetimeIndex(pd.date_range(
  start='1/1/2011', periods=4, freq='M'))
s = Series([1,2,3,4], index=dti)
a = dti.to_pydatetime()       # numpy array
a = s.index.to_pydatetime()   # numpy array
```

From Timestamps to Python dates or times

```
df['py_date'] = [x.date() for x in df['TS']]
df['py_time'] = [x.time() for x in df['TS']]
```

Note: converts to datatime.date or datetime.time. But does not convert to datetime.datetime.

From DatetimeIndex to PeriodIndex and back

```
df = DataFrame(np.random.randn(20,3))
df.index = pd.date_range('2015-01-01',
       periods=len(df), freq='M')
dfp = df.to_period(freq='M')
dft = dfp.to_timestamp()
```

Note: from period to timestamp defaults to the point in time at the start of the period.

Working with a PeriodIndex

```
pi = pd.period_range('1960-01','2015-12',
                     freq='M')
a = pi.values    # numpy array of integers
p = pi.tolist() # python list of Periods
sp = Series(pi)  # pandas Series of Periods
s = Series(pi).astype(str) # Series of strs
l = Series(pi).astype(str).tolist()
```

Get a range of Timestamps

```
dr = pd.date_range('2013-01-01',
    '2013-12-31', freq='D')
```

Error handling with dates

```
# 1st example returns string not Timestamp
t = pd.to_datetime('2014-02-30')
# 2nd example returns NaT (not a time)
t = pd.to_datetime('2014-02-30', coerce=True)
# NaT like NaN tests True for isnull()
b = pd.isnull(t) # --> True
```

The tail of a time-series DataFrame

```
df = df.last("5M") # the last five months
```

Upsampling and downsampling

```
# upsample from quarterly to monthly
pi = pd.period_range('1960Q1',
        periods=220, freq='Q')
df = DataFrame(np.random.rand(len(pi),5),
        index=pi)
dfm = df.resample('M', convention='end')
# use ffill or bfill to fill with values

# downsample from monthly to quarterly
dfq = dfm.resample('Q', how='sum')
```

Time zones

```
t = ['2015-06-30 00:00:00',
      '2015-12-31 00:00:00']
dti = pd.to_datetime(t
      ).tz_localize('Australia/Canberra')
dti = dti.tz_convert('UTC')
ts = pd.Timestamp('now',
      tz='Europe/London')

# get a list of all time zones
import pyzt
for tz in pytz.all_timezones:
    print tz
```

Note: by default, Timestamps are created without time zone information.

Row selection with a time-series index

```
# start with the play data above
idx = pd.period_range('2015-01',
        periods=len(df), freq='M')
df.index = idx

february_selector = (df.index.month == 2)
february_data = df[february_selector]

q1_data = df[(df.index.month >= 1) &
   (df.index.month <= 3)]

mayornov_data = df[(df.index.month == 5) |
   (df.index.month == 11)]

totals = df.groupby(df.index.year).sum()
```

Also: year, month, day [of month], hour, minute, second, dayofweek [Mon=0 .. Sun=6], weekofmonth, weekofyear [numbered from 1], week starts on Monday], dayofyear [from 1], …

The Series.dt accessor attribute

DataFrame columns that contain datetime-like objects can be manipulated with the .dt accessor attribute

```
t = ['2012-04-14 04:06:56.307000',
     '2011-05-14 06:14:24.457000',
     '2010-06-14 08:23:07.520000']

# a Series of time stamps
s = pd.Series(pd.to_datetime(t))
print(s.dtype)      # datetime64[ns]
print(s.dt.second)  # 56, 24,  7
print(s.dt.month)   #  4,  5,  6
# a Series of time periods
s = pd.Series(pd.PeriodIndex(t,freq='Q'))
print(s.dtype)      # datetime64[ns]
print(s.dt.quarter) #  2,  2,  2
print(s.dt.year)    #  2012, 2011, 2010
```

Plotting from the DataFrame

Import matplotlib, choose a matplotlib style

```
import matplotlib.pyplot as plt
print(plt.style.available)
plt.style.use('ggplot')
```

Fake up some data (which we reuse repeatedly)

```
a = np.random.normal(0,1,999)
b = np.random.normal(1,2,999)
c = np.random.normal(2,3,999)
df = pd.DataFrame([a,b,c]).T
df.columns =['A', 'B', 'C']
```

Line plot

```
df1 = df.cumsum()
ax = df1.plot()

# from here down - standard plot output
ax.set_title('Title')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')

fig = ax.figure
fig.set_size_inches(8, 3)
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)

plt.close()
```

Box plot

```
ax = df.plot.box(vert=False)
# followed by the standard plot code as above
```

```
ax = df.plot.box(column='c1', by='c2')
```

Histogram

```
ax = df['A'].plot.hist(bins=20)
# followed by the standard plot code as above
```

Multiple histograms (overlapping or stacked)

```
ax = df.plot.hist(bins=25, alpha=0.5) # or...
ax = df.plot.hist(bins=25, stacked=True)
# followed by the standard plot code as above
```

Bar plots

```
bins = np.linspace(-10,15,26)
binned = pd.DataFrame()
for x in df.columns:
    y=pd.cut(df[x],bins,labels=bins[:-1])
    y=y.value_counts().sort_index()
    binned = pd.concat([binned,y],axis=1)
binned.index = binned.index.astype(float)
binned.index += (np.diff(bins) / 2.0)
ax = binned.plot.bar(stacked=False,
    width=0.8) # for bar width
# followed by the standard plot code as above
```

Horizontal bars

```
ax = binned['A'][(binned.index >= -4) &
    (binned.index <= 4)].plot.barh()
# followed by the standard plot code as above
```

Density plot

```
ax = df.plot.kde()
# followed by the standard plot code as above
```

Scatter plot

```
ax = df.plot.scatter(x='A', y='C')
# followed by the standard plot code as above
```

Pie chart

```
s = pd.Series(data=[10, 20, 30],
    index = ['dogs', 'cats', 'birds'])
ax = s.plot.pie(autopct='%.1f')

# followed by the standard plot output ...
ax.set_title('Pie Chart')
ax.set_aspect(1) # make it round
ax.set_ylabel('') # remove default

fig = ax.figure
fig.set_size_inches(8, 3)
fig.savefig('filename.png', dpi=125)

plt.close(fig)
```

Change the range plotted

```
ax.set_xlim([-5, 5])

# for some white space on the chart ...
lower, upper = ax.get_ylim()
ax.set_ylim([lower-1, upper+1])
```

Add a footnote to the chart

```
# after the fig.tight_layout(pad=1) above
fig.text(0.99, 0.01, 'Footnote',
    ha='right', va='bottom',
    fontsize='x-small',
    fontstyle='italic', color='#999999')
```

A line and bar on the same chart

In matplotlib, bar charts visualise categorical or discrete data. Line charts visualise continuous data. This makes it hard to get bars and lines on the same chart. Typically combined charts either have too many labels, and/or the lines and bars are misaligned or missing. You need to trick matplotlib a bit … pandas makes this tricking easier

```
# start with fake percentage growth data
s = pd.Series(np.random.normal(
    1.02, 0.015, 40))
s = s.cumprod()
dfg = (pd.concat([s / s.shift(1),
    s / s.shift(4)], axis=1) * 100) - 100
dfg.columns = ['Quarter', 'Annual']
dfg.index = pd.period_range('2010-Q1',
    periods=len(dfg), freq='Q')

# reindex with integers from 0; keep old
old = dfg.index
dfg.index = range(len(dfg))

# plot the line from pandas
ax = dfg['Annual'].plot(color='blue',
    label='Year/Year Growth')

# plot the bars from pandas
dfg['Quarter'].plot.bar(ax=ax,
    label='Q/Q Growth', width=0.8)

# relabel the x-axis more appropriately
ticks = dfg.index[((dfg.index+0)%4)==0]
labs = pd.Series(old[ticks]).astype(str)
ax.set_xticks(ticks)
ax.set_xticklabels(labs.str.replace('Q',
    '\nQ'), rotation=0)

# fix the range of the x-axis … skip 1st
ax.set_xlim([0.5,len(dfg)-0.5])

# add the legend
l=ax.legend(loc='best',fontsize='small')

# finish off and plot in the usual manner
ax.set_title('Fake Growth Data')
ax.set_xlabel('Quarter')
ax.set_ylabel('Per cent')

fig = ax.figure
fig.set_size_inches(8, 3)
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)

plt.close()
```

Working with missing and non-finite data

Working with missing data

Pandas uses the not-a-number construct (np.nan and float('nan')) to indicate missing data. The Python None can arise in data as well. It is also treated as missing data; as is the pandas not-a-time construct (pandas.NaT).

Missing data in a Series

```
s = Series( [8,None,float('nan'),np.nan])
           #[8,     NaN,   NaN,   NaN]
s.isnull() #[False, True,  True,  True]
s.notnull()#[True,  False, False, False]
s.fillna(0)#[8,     0,     0,     0]
```

Missing data in a DataFrame

```
df = df.dropna() # drop all rows with NaN
df = df.dropna(axis=1) # same for cols
df=df.dropna(how='all') #drop all NaN row
df=df.dropna(thresh=2) # drop 2+ NaN in r
# only drop row if NaN in a specified col
df = df.dropna(df['col'].notnull())
```

Recoding missing data

```
df.fillna(0, inplace=True) # np.nan → 0
s = df['col'].fillna(0)    # np.nan → 0
df = df.replace(r'\s+', np.nan,
      regex=True) # white space → np.nan
```

Non-finite numbers

With floating point numbers, pandas provides for positive and negative infinity.

```
s = Series([float('inf'), float('-inf'),
     np.inf, -np.inf])
```

Pandas treats integer comparisons with plus or minus infinity as expected.

Testing for finite numbers

(using the data from the previous example)

```
b = np.isfinite(s)
```

Working with Categorical Data

Categorical data

The pandas Series has an R factors-like data type for encoding categorical data.

```
s = Series(['a','b','a','c','b','d','a'],
                        dtype='category')
df['B'] = df['A'].astype('category')
```

Note: the key here is to specify the "category" data type.
Note: categories will be ordered on creation if they are sortable. This can be turned off. See ordering below.

Convert back to the original data type

```
s = Series(['a','b','a','c','b','d','a'],
                        dtype='category')
s = s.astype('string')
```

Ordering, reordering and sorting

```
s = Series(list('abc'), dtype='category')
print (s.cat.ordered)
s=s.cat.reorder_categories(['b','c','a'])
s = s.sort()
s.cat.ordered = False
```

Trap: category must be ordered for it to be sorted

Renaming categories

```
s = Series(list('abc'), dtype='category')
s.cat.categories = [1, 2, 3] # in place
s = s.cat.rename_categories([4,5,6])
# using a comprehension ...
s.cat.categories = ['Group ' + str(i)
    for i in s.cat.categories]
```

Trap: categories must be uniquely named

Adding new categories

```
s = s.cat.add_categories([4])
```

Removing categories

```
s = s.cat.remove_categories([4])
s.cat.remove_unused_categories() #inplace
```

Working with strings

Working with strings

```
# assume that df['col'] is series of strings
s = df['col'].str.lower()
s = df['col'].str.upper()
s = df['col'].str.len()

# the next set work like Python
df['col'] += 'suffix'       # append
df['col'] *= 2              # duplicate
s = df['col1'] + df['col2'] # concatenate
```

Most python string functions are replicated in the pandas DataFrame and Series objects.

Regular expressions

```
s = df['col'].str.contains('regex')
s = df['col'].str.startswith('regex')
s = df['col'].str.endswith('regex')
s = df['col'].str.replace('old', 'new')
df['b'] = df.a.str.extract('(pattern)')
```

Note: pandas has many more regex methods.

Basic Statistics

Summary statistics

```
s = df['col1'].describe()
df1 = df.describe()
```

DataFrame – key stats methods

```
df.corr()     # pairwise correlation cols
df.cov()      # pairwise covariance cols
df.kurt()     # kurtosis over cols (def)
df.mad()      # mean absolute deviation
df.sem()      # standard error of mean
df.var()      # variance over cols (def)
```

Value counts

```
s = df['col1'].value_counts()
```

Cross-tabulation (frequency count)

```
ct = pd.crosstab(index=df['a'],
        cols=df['b'])
```

Quantiles and ranking

```
quants = [0.05, 0.25, 0.5, 0.75, 0.95]
q = df.quantile(quants)
r = df.rank()
```

Histogram binning

```
count, bins = np.histogram(df['col1'])
count, bins = np.histogram(df['col'],
               bins=5)
count, bins = np.histogram(df['col1'],
               bins=[-3,-2,-1,0,1,2,3,4])
```

Regression

```
import statsmodels.formula.api as sm
result = sm.ols(formula="col1 ~ col2 +
          col3", data=df).fit()
print (result.params)
print (result.summary())
```

Simple smoothing example using a rolling apply

```
k3x5 = np.array([1,2,3,3,3,2,1]) / 15.0
s = df['A'].rolling(window=len(k3x5),
    min_periods=len(k3x5),
    center=True).apply(
         func=lambda x: (x * k3x5).sum())
# fix the missing end data ... unsmoothed
s = df['A'].where(s.isnull(), other=s)
```

Cautionary note

This cheat sheet was cobbled together by tireless bots roaming the dark recesses of the Internet seeking ursine and anguine myths from a fabled land of milk and honey where it is rumoured pandas and pythons gambol together. There is no guarantee the narratives were captured and transcribed accurately. You use these notes at your own risk. You have been warned. I will not be held responsible for whatever happens to you and those you love once your eyes begin to see what is written here.

Errors: If you find any errors, please email me at markthegraph@gmail.com; (but please do not correct my use of Australian-English spelling conventions).