Python 101

Respondo

Which version 2.7 or 3.5

Most of the examples are in python 2.7 not that hard to switch over.

http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html

- print is a function and needs() in v3.
- raw_input() is input() in v3.
- xrange() is range() in v3. there is no xrange in v3

Know the version of you're running with python -V

Why Python for Cybersecurity?

Python is widely popular in the field of cybersecurity due to several key reasons.

- Versatility
- Readability
- Extensive Library Support
- Rapid Development and Prototyping
- Cross-Platform Compatibility
- Community Support and Resources

Versatility

Python is a versatile programming language that can be used for a wide range of cybersecurity tasks. It supports various paradigms, including procedural, object-oriented, and functional programming, allowing flexibility in designing and implementing solutions.

Readability

Python emphasizes code readability with its clean and concise syntax. Its use of indentation for block structuring promotes code clarity, making it easier for cybersecurity professionals to understand and maintain code, especially when collaborating on projects.

Extensive Library Support

Python boasts a vast ecosystem of libraries and frameworks specifically designed for cybersecurity. These libraries provide pre-built functions and modules to address common cybersecurity tasks, such as network scanning, web scraping, cryptography, and more. The availability of such libraries allows professionals to leverage existing tools and accelerate their development processes.

Rapid Development and Prototyping

Python's simplicity and ease of use contribute to rapid development and prototyping in the cybersecurity domain. Its concise syntax and extensive library support enable professionals to quickly build proof-of-concepts, automate tasks, and develop prototypes for security tools and utilities.

Cross-Platform Compatibility

Python is a cross-platform language, meaning that Python code can run on multiple operating systems, including Windows, macOS, and various Linux distributions. This cross-platform compatibility ensures that cybersecurity solutions built with Python can be deployed across different systems and environments.

Community Support and Resources

Python has a large and active community of developers, including cybersecurity professionals, who actively contribute to open-source projects, share knowledge, and provide support. The availability of resources, tutorials, forums, and online communities makes it easier for individuals to learn, troubleshoot, and stay updated with the latest advancements in Python and cybersecurity.

Learning Python for Free

References:

- Free access Lynda.com (video learning)
  - Miami-Dade Public Library Card
  - Broward NSU e-card
- https://www.w3schools.com/python
- https://www.tutorialspoint.com/python
- https://www.codecademy.com/learn/learn-python-3
- https://www.geeksforgeeks.org/python-programming-language
- https://edube.org/
- https://inventwithpython.com/
- Coursera & edX
- Meetup like this one. :)

pyenv - what is this for?

- pyenv help control the python versions in your system.
- This way you can have different versions of python install on the same machine.
- Easy to switch python versions.

```
# install pyenv
$ curl https://pyenv.run | bash

# list installable python versions
$ pyenv install —-list
```

```
> pyenv install 3.10.6
> pyenv global 3.10.6
> pyenv versions
  system
* 3.10.6 (set by /Users/jrespeto/.pyenv/version)
> python -V
Python 3.10.6
```

Installing pyenv deps – MacOS

```
# Install xcode
$ xcode-select --install


# Install homebrew https://brew.sh/
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"


# Install deps from pyenv
$ brew install openssl readline sqlite3 xz zlib
```

Setting up jupyter lab

Python - pip - virtualenv
Python is install on - Linux and OSx your good

```
sudo python -m ensurepip
```

```
python -m venv python101
source python101/bin/activate
pip install jupyter
```

```
mkdir pynotes
cd pynotes
jupyter lab
```

Browse should open.

IDE’s take you pick...

Not in any order...

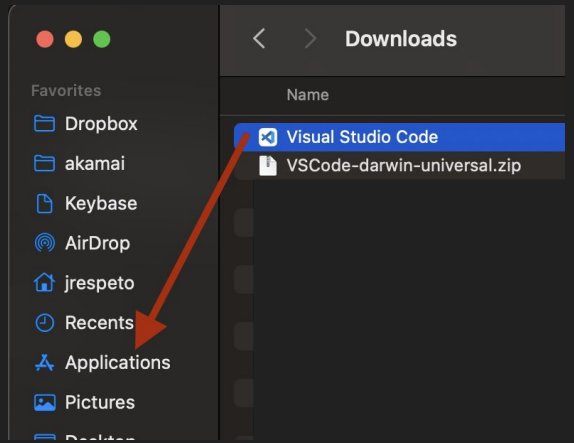
- vscode
- jupyter
- Vim
- …

```
.vimrc
filetype plugin indent on
syntax on
set autoindent
set smartindent
set tabstop=4
set softtabstop=4
set shiftwidth=8
expandtab
set number
set modeline
colorscheme murphy

Make sure file has .py extension
:ggVG=  # reindent file
https://wiki.python.org/moin/Vim
```

Visual Studio Code

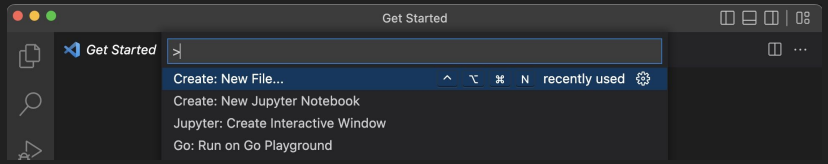
- All around good IDE and Free
- Supports all OS’s
- Syncs settings
- Very extendable
- Cool collaboration features.

Download here:
https://code.visualstudio.com/
OSx Install: Unzip it, drag to Application

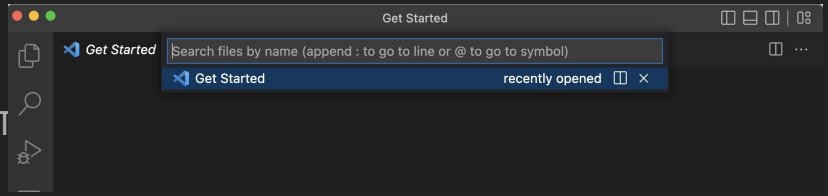


Shortcuts - you need to know

`Command + shift + p`



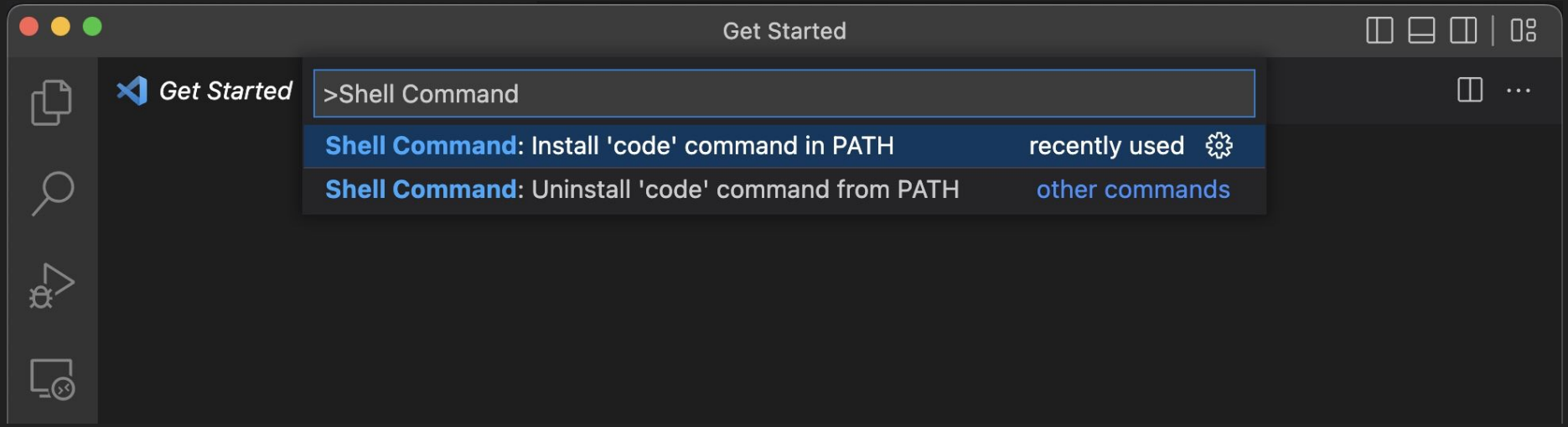
`Command + p`



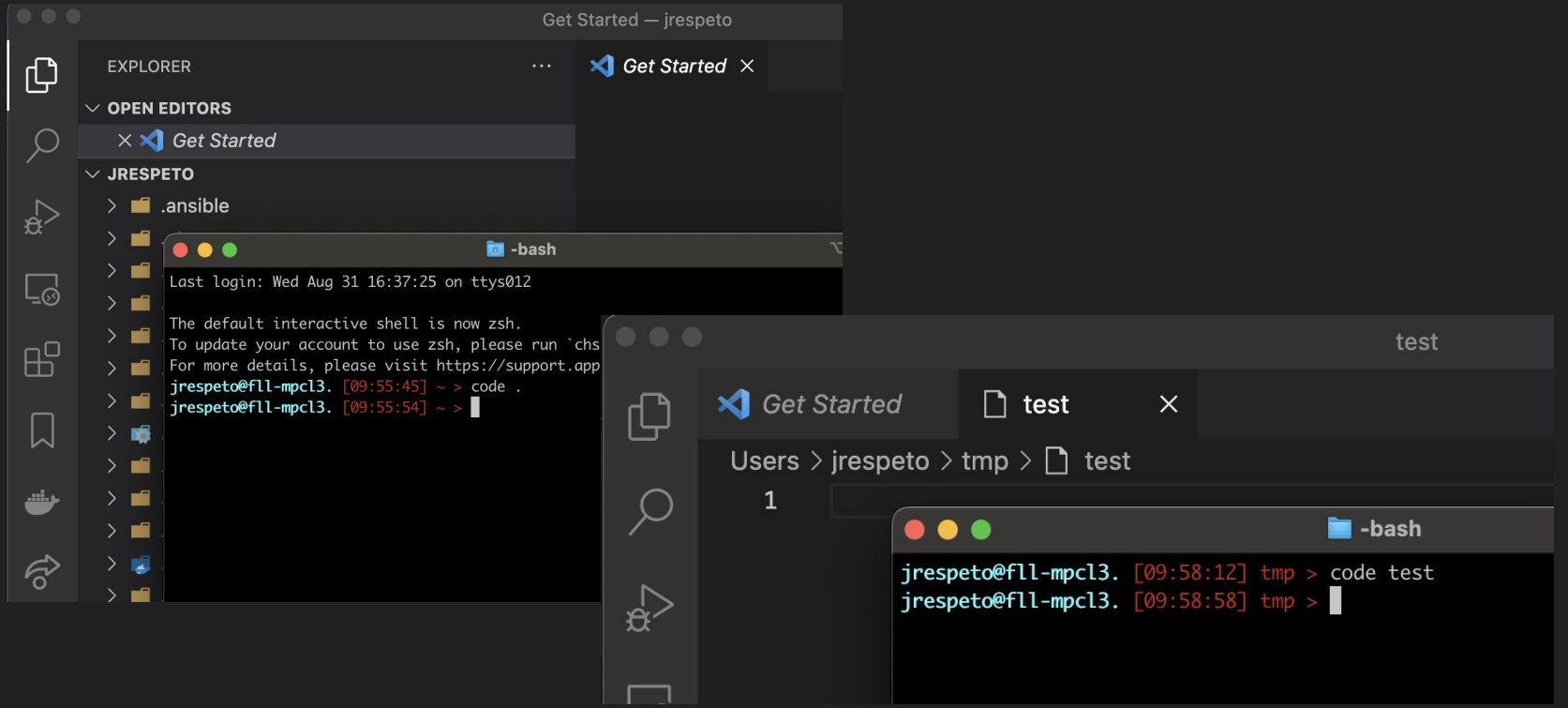
If you open the wrong one just add or remove the > in the front 😀

Adding code to your path

This lets you open files or dir in code from the command line.
Also lets you install extensions from the command line.

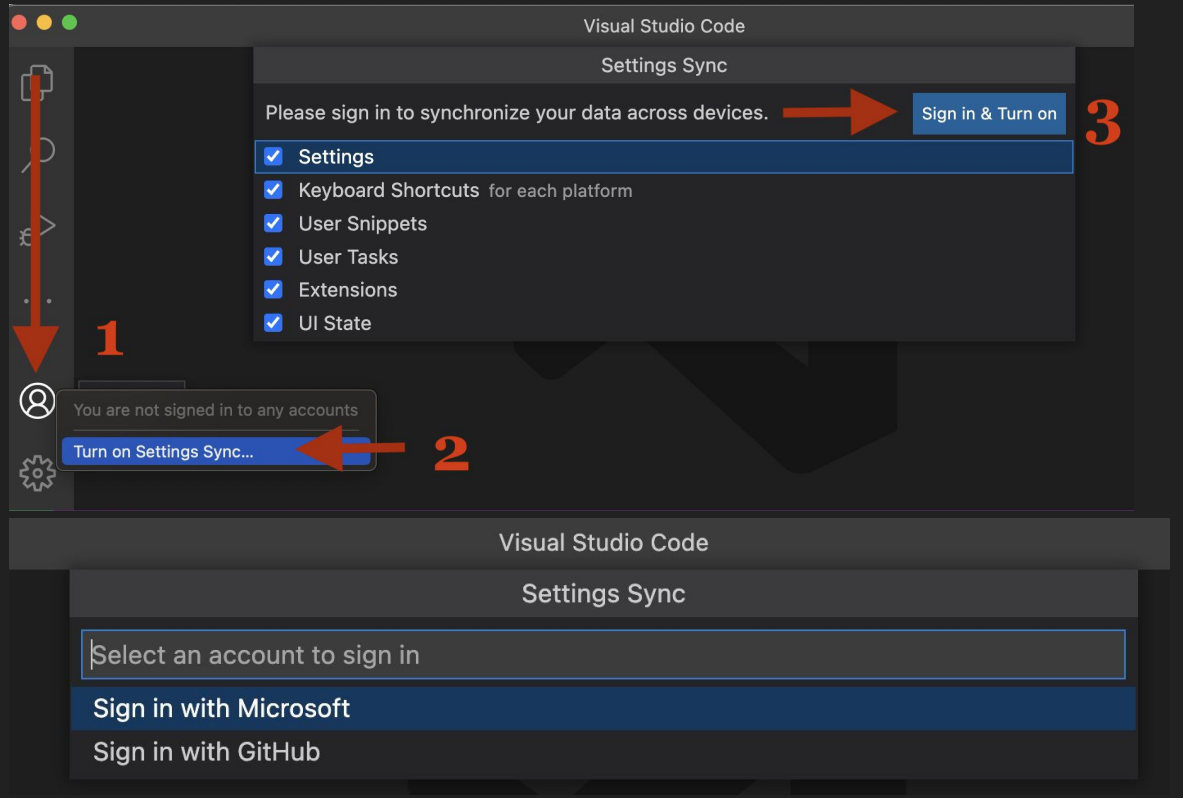


Opening file/dir from the cli



Settings Sync

Convenient if you use multiple systems.
Quick to setup.
Browser opens to login into Microsoft or GitHub.
Done



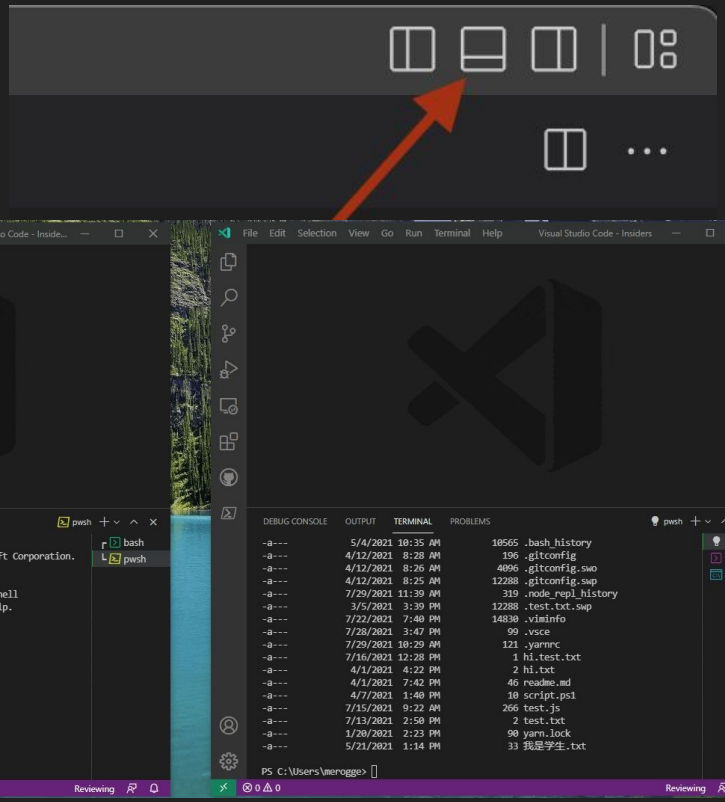
Terminal

Quick access to the terminal.

Command + j opens the bottom panel.

Window can be move around.

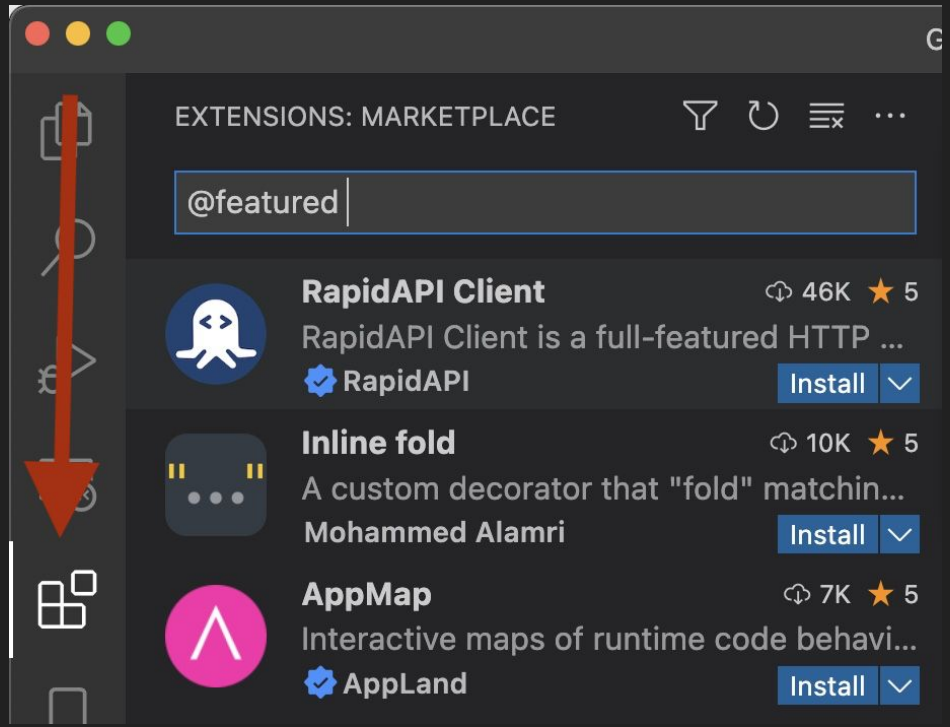
Ctrl+` also opens the terminal



https://stackoverflow.com/questions/59665958/vscode-open-terminals-in-a-separate-window

Installing extensions

You can search and add extension by clicking on the icon.



Installing extension from the command line.

Here is a list of 37 extension I use.
Most are for code highlighting.

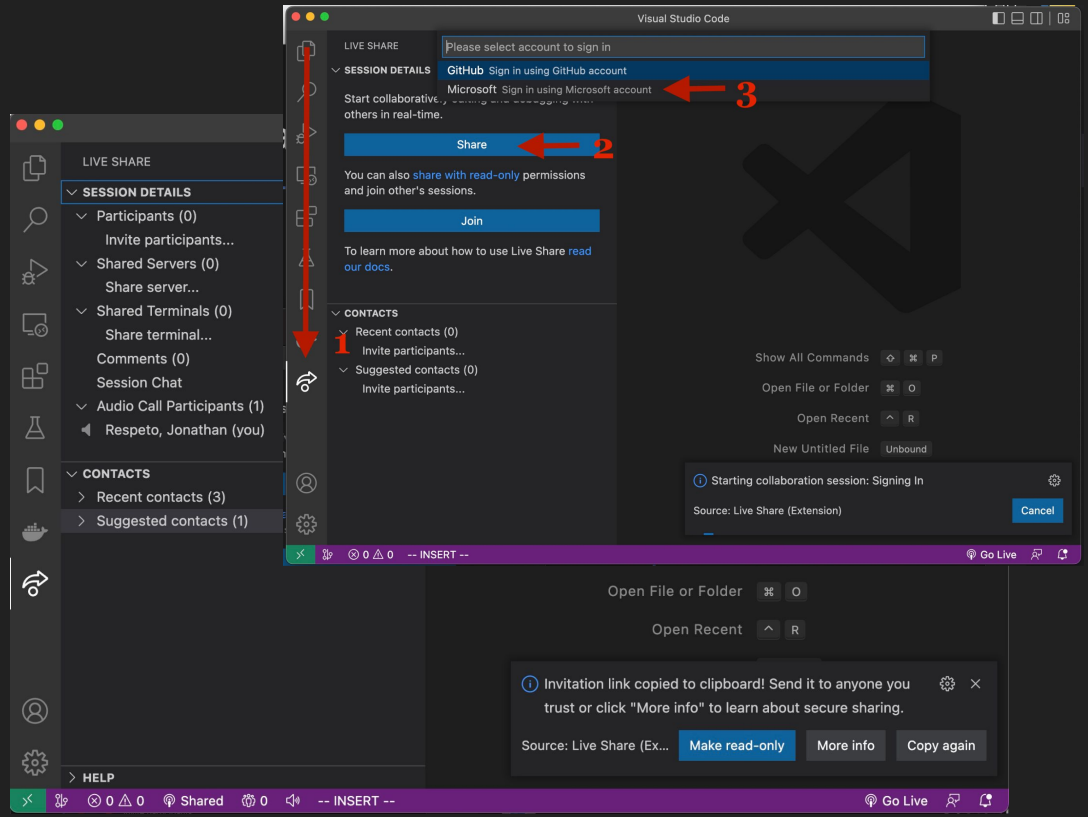
Short list of things covered.

```
code --install-extension aaron-bond.better-comments
code --install-extension alefragnani.Bookmarks
code --install-extension DavidAnson.vscode-markdownlint
code --install-extension eamodio.gitlens
code --install-extension esbenp.prettier-vscode
code --install-extension KevinRose.vsc-python-indent

code --install-extension ms-azuretools.vscode-docker
code --install-extension ms-python.python
code --install-extension ms-python.vscode-pylance
code --install-extension ms-toolsai.jupyter
code --install-extension ms-toolsai.jupyter-keymap
code --install-extension ms-toolsai.jupyter-renderers
code --install-extension ms-vscode-remote.remote-containers
code --install-extension ms-vscode-remote.remote-ssh
code --install-extension ms-vsliveshare.vsliveshare
code --install-extension quicktype.quicktype

code --install-extension shd101wyy.markdown-preview-enhanced
code --install-extension vscode-icons-team.vscode-icons
code --install-extension wk-j.vscode-httpie


code --install-extension vscodevim.vim # if you dont use vim dont install
```

Live Share

code --install-extension ms-vsliveshare.vsliveshare

Share the link.
Keep in mind participants can see anything in the open workspace.
Participants can only see saved files. Not new files.



Setting up a python environment

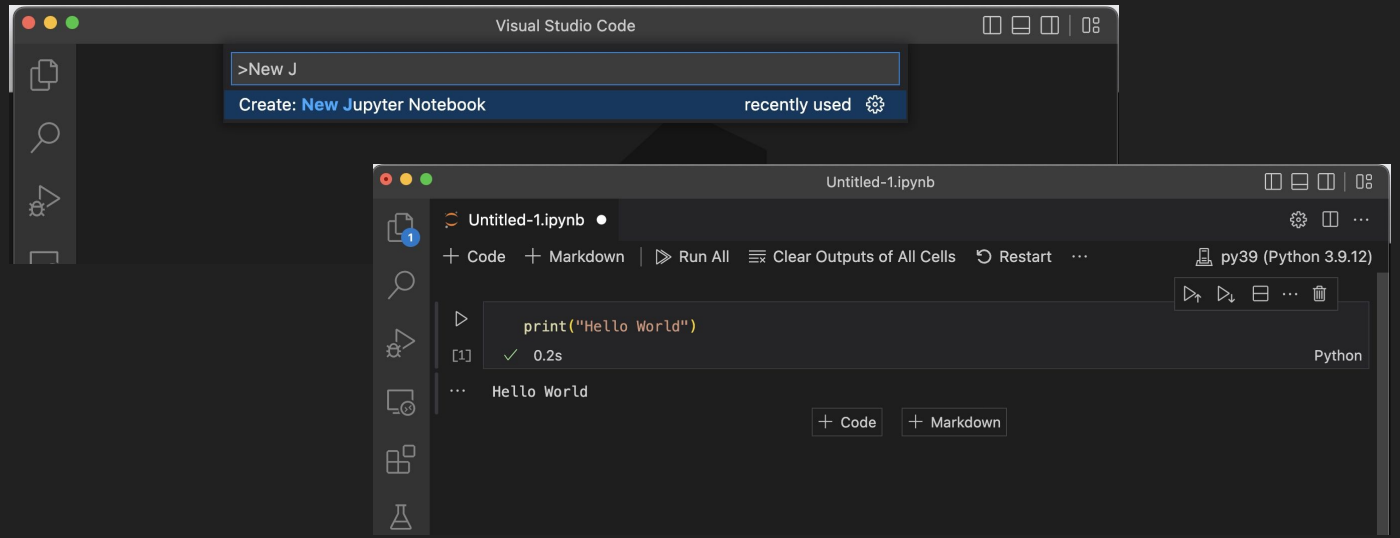
https://www.python.org/downloads/macos
Run the python installer package
In a Terminal window:
python -m venv ~/pyenv

In vscode -> settings -> search for “Default Interpreter Path”
~/pyenv/bin/python

Python - Jupyter

```
code --install-extension KevinRose.vsc-python-indent
code --install-extension ms-python.python
code --install-extension ms-python.vscode-pylance
code --install-extension ms-toolsai.jupyter
code --install-extension ms-toolsai.jupyter-keymap
code --install-extension ms-toolsai.jupyter-renderers
```

After you install the extensions.
Going to ask which python to run.



python and whitespace

```
no ;
no {}
no problems
```

Indentation indicate a block of code in python

Code blocks for, loops / try / def and class have ':' colon at the end of the first line.

    Then are tab indented.

Blocks end by un-indenting.

```
for i in range(5):
    print i
<- # don't type this!
```

comments

```
# Hash mark (pound sign) indicate the begin of comments to the end
of the line.
# there are no multiline comments.

print 'hello world' # this is printing hello world.


'''Docstrings are for documenting and are multiline'''
```

Hello World

```
Open a terminal
$ python
>>> print 'hello world!!'
```

Python builtins methods and getting help.

```
dir() - list methods
dir(__builtins__) - print python builtin methods
help() - prints help from docstrings in classes and functions.
type() - prints data type.
__doc__

import os
help(os)
print os.__doc__
dir(os)
```

Troubleshooting your script

Print often!
Embed a -debug or -verbose switch
Use the help() and dir()
Be use to reading errors.

variable name

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (age, Age and AGE are three different variables)

Data Types

bool - true or false

strings="Foo" - just a word

numbers=1 - int or float or complex 1000 or 1000.99 or 10e29 or 5j

lists=["foo","bar"] - array or changeable list - zero indexed

tuples=("foo","bar") - unchangeable list - zero indexed

sets={"foo","bar","foo"} - list of unique

dict={"foo":"bar","pet":"dog"} - key:value pair

Strings

String literals in python are surrounded by either single quotation marks, or double quotation marks.

```
month='July '
month="July "

sting have a list(array) like indexing
month[1] # will print u

like list you can also call substrings
month[2:4] # prints ly

And much more… dir(month) to see all you can do.
```

String formatting

```
a="hello"
b="world"

print "%s" % a

print "We are doing better than %s %s now" % (a, b)
```

```
%s - string
%d - number
%r - raw
```

String formatting 2

```
a="hello"
b="world"

print "We are doing better than {} {}".format(a, b)
```

Little tricks with LIST

```
foods=["burgers","tacos","pizza"]

a=foods.copy()

a=foods[:] # this is also a copy

b=foods[1:] #

c=foods[:-2] #

d=len(foods) #

n = ['hi']*3 #

o = n+foods #
```

```
food.pop() # returns and removes last
             element from the list

Question: how do you remove the first
          element?


dir(food)  # and much more...
```

Assignment Operators

=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Arithmetic Operators

+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Comparison Operators

`==`	Equal	`x == y`
`!=`	Not equal	`x != y`
`>`	Greater than	`x > y`
`<`	Less than	`x < y`
`>=`	Greater than or equal to	`x >= y`
`<=`	Less than or equal to	`x <= y`

Logical Operators

```
and     Returns True
        if both statements are true
        x < 5 and x < 10

or      Returns True
        if one of the statements is true
        x < 5 or x < 4

not     Reverse the result, returns False if the result is true
        not(x < 5 and x < 10)
```

Identity / Membership Operators

```
is          Returns true if both variables are the same object
        x is y

is not      Returns true if both variables are not the same object
        x is not y

in          Returns True if a sequence with the specified value is
present in the object
        x in y

not in      Returns True if a sequence with the specified value is
not present in the object
        x not in y
```

If / Elif / Else Statements

```
if b > a: print("b is greater than a")

if a in b == True:
    print (a ," is in b")
elif c is b:
    print (c+" is in d" )
else:
    print ("nothin to see here.")
```

Modules

Python code files with .py extension.

They can contain functions and all variable types.
(arrays, dicts, objects, …)

Modules are loaded into your script with the import keyword.

```
import os
```

To use a module function or class prefix the name of the module.

```
print os.path
path = os.path
print path
```

For and While Loops Examples

```
color = ["red", "white", "blue"]
for x in color:
  print x

pets = ["dog", "cat", "fish"]
for x in pets:
  if x == "cat":
     break # stop running the loop
  print x

for x in range(5):
  if x < 3:
     continue # go to the next loop
  print x
```

```
i = 1
while i < 6:
  print(i)
  i += 1

i = 1
while i < 6:
  print(i)
  if i == 3:
     break
  i += 1
```

Argv and Raw_input

```
python myscript.py a b c
```

```
import sys
print "script is: ", sys.argv[0]

print "\t",sys.argv[2],"\n"

print "Num of args: ", len(sys.argv)

print "The args are: " , str(sys.argv)
```

```
#python 2.#
name = raw_input("What is your name? ")
print "Hello, %s." % name
```

```
#python 3
name = input("What is your name? ")
print "Hello, %s." % name
```

Working with files

```
FH = open("demofile.txt", "rt")

print FH.read() # print whole file

print FH.read(10) # print 10 characters

print FH.readline() # read last line

for x in FH: # loop thur lines in a file
    print x

for i in range(2): # print first two lines
    line=FH.next().strip()
    print line
```

```
FH = open('filename.txt',wt)

text = "this is \n a file \n \t I
made."

FH.write(text)
FH.close()

Modes - t - text(default)| b - binary
r - read - errors does not exist

a - append - Adds to the end of file

w - write - creates if not exists
            overrides if exists
```

Functions

Block of code that runs when called.

```
# Just the basics
def my_function():
    print "Hello from function"

my_function()
```

```
# Parameters
this_func(my_pram):
    print my_pram

this_func("foo")
```

```
# Return values
this_func(a,b):
    return a+b

this_func(5,3)
```

Documenting

documenting functions with doc type

example: cat re.py

```
'''
__doc__

r'''
u'''
```

```
def my_function():
    '''this is a documentation'''
    print __doc__


help(my_function())
```

Decode this only using python __builtins__

1.
Iwt Rpthpg rxewtg xh dct du iwt tpgaxthi zcdlc pcs hxbeathi rxewtgh.

Decode this only using python __builtins__

2.

```
41 64 76 65 72 74 69 73 69 6e 67 20 70 65 6f 70 6c 65 20 77 68 6f
20 69 67 6e 6f 72 65 20 72 65 73 65 61 72 63 68 20 61 72 65 20 61
73 20 64 61 6e 67 65 72 6f 75 73 20 61 73 20 67 65 6e 65 72 61 6c
73 20 77 68 6f 20 69 67 6e 6f 72 65 20 64 65 63 6f 64 65 73 20 6f
66 20 65 6e 65 6d 79 20 73 69 67 6e 61 6c 73 2e
```

Decode this only using python __builtins__

3.

65 100 118 101 114 116 105 115 105 110 103 32 112 101 111 112 108
101 32 119 104 111 32 105 103 110 111 114 101 32 114 101 115 101
97 114 99 104 32 97 114 101 32 97 115 32 100 97 110 103 101 114
111 117 115 32 97 115 32 103 101 110 101 114 97 108 115 32 119 104
111 32 105 103 110 111 114 101 32 100 101 99 111 100 101 115 32
111 102 32 101 110 101 109 121 32 115 105 103 110 97 108 115 46

Decode this only using one module and __builtins__

4.

```
NDEgNjQgNzYgNjUgNzIgNzQgNjkgNzMgNjkgNmUgNjcgMjAgNzAgNjUgNmYgNzAgNm
MgNjUgMjAgNzcgNjggNmYgMjAgNjkgNjcgNmUgNmYgNzIgNjUgMjAgNzIgNjUgNzMg
NjUgNjEgNzIgNjMgNjggMjAgNjEgNzIgNjUgMjAgNjEgNzMgMjAgNjQgNjEgNmUgNj
cgNjUgNzIgNmYgNzUgNzMgMjAgNjEgNzMgMjAgNjcgNjUgNmUgNjUgNzIgNjEgNmMg
NzMgMjAgNzcgNjggNmYgMjAgNjkgNjcgNmUgNmYgNzIgNjUgMjAgNjQgNjUgNjMgNm
YgNjQgNjUgNzMgMjAgNmYgNjYgMjAgNjUgNmUgNjUgNmQgNzkgMjAgNzMgNjkgNjcg
NmUgNjEgNmMgNzMgMmUg
```

Decode this...

/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAkGBw0SDRANDRAQDw8NEA8QDQ0REBAQEA0OFREWGBURExUYHSggGRolGxgVITEjJSkrLi8uFx8zODMtQygtLisBCgoKDg0OGxAQGzAmHR4rLS03LSstLS0tKy8rLSsuMCstLS0vLTAtKy0tLS0rLSsvLS0rKy0tKy8rKy0
rLSsrLf/AABEIAOEA4QMBEQACEQEDEQH/xAAcAAEBAAMBAQEBAAAAAAAAAAAAAAQQFBgcDAgj/xABHEAACAQEDBQgPBgYCAwAAAAAAAQIDBAURERiExQVEGB1JhcZGT0RMUFRYXIjIzU3JzgZGZkx4SM0oaKz0yQ1QoKywXTwQ2LC/8QAGwEBAQADAQEBAAAAAAAAAAAAAA
EDBAUCBgf/xAAzEQEAAQICBgcIAwEBAAAAAAAAAAAQIDBBESFCExUXEFMlJhkbHRFTNBgaHB4fATIjRyQv/aAAwDAQACEQMRAD8A9xAAAAAAAAAAfitVhCLnOUYRisZTk1GMVtbegsRM7ISZiNsuTvXxfAsdNuNCMrTJa14lPH1nnfuTRtUYOurrbGrXKj1ulz1zFt3wbwni
qSpUFqyYZcl75Yp8xs0403G/a1qsXcndsaa0bpLxn5dqr/2TdP8IYGaLNuN1MMU3rk76pYVS3V5eVWqy9apN/NnrRpj4PGnVx18Jsb0tvleJ6RFNrQ2uR4EV96dvtEfIrVo+rVnH5Mk0U8F06uMs2zbp7yp+Ra6398uy/54mObNud9L3F65G6r9+bdWHfFFt0MFWhSrx
15nTm/7o5vymGrB0Tu2M10LrjftdXdG76wVmo1XKzTfpcOxt8VRZkvWwNavC107trZoxVW/wmmmmsU1nTW1Gs2VAAAAAAAAAAAAAAAGFfN6UrNQlaKz8WOZRV3VJvRCK2v66j3btzXVow8XLKUU6vH7/3Qm1zyqKaeNOhFvsdPZ60uN/hovas0242
eLk3btVydvg1RlYkIIACoBCAFAIQArebmt1Fpsckot1KDfj2eT8XjcH/S/wAHrRhu2abnPizWr1Lwlwev3VeVG00I2ihLKhP3SjGS1NHLromicJpcab0Hc/pdZoOuKozhmHl6AAAAAAAAAAAAAAAAHmG+fb5StdOzY+JQpio87ZngIiauLm4yrQuKede1bKtz7kEeDjDcab0Hc
fuJozowtVtTm6qUqVDFxjGDzqU8M7b04aMHn4uffxUxOjQ37GGiY0q/B0dt3HXbUg49gjTeqdL7OUXtzZn70XzXpxNyJzzbFWGtzGXg8r3RXPUslplZ5vKWClTqYYKpTeOEsNTzNNdbUdO1ci5TpQ5tz23NurRlrDIxoQAoBCAFQCEV1u9tfEqNtVnk/srWQNZLxJe
/yePGOw1sVb0qNL4w2MNXo15fCXrhzXSAAAAAAAAAAAAHke+N/M5+zpf4nVwununxKfvHLzJ/k2hvb7hHLbvW/Xz10Q/vbzQHV7Aib/ucDiHxxfQK+i5KthE5SkF3ChLRTt3NeItzSlaxlrsfMr6+rNkyMTuaQnT1ZxR3C1NdhWaTaXLuwqi0HAUyHLVwqtmGHwaDTGmqZ8HQt15RRpc
0/7VbvNqYnEaH9ad/k8tqTlKTlJuUpnUpNuUm9LbelnTUbJbIdd2CuZZbmGkPhp6/GgIWJGhXmB5TXl51KDvmIcd0VaBbBoAxs3pIybEDWaaNa75JJlK8NzqwqHRPSEPatuOFeBeQUKkwMrEqqW2WmZ2gqo5GVc53NUlWsNvEbUtrEoMM3ndvqH1NtPtW6xW10+ZT8/9dk78r6oHE7Y8ADEqdJcdPJMbJsJb8UZLd
yTpIFAIRUAY4Z1pWgZzG2N8Gwboe3oH2WvW3K/gqeI3tUuyrVaJ8OWvRPsB8oDPQnFQ9ry09t4ztZ51HXYqyR+1kHg22bDhS3aj9UHFLa8gJzHQNjKQ1mXcBZTw0b9j/OKI1fTe0+Vt5zAARfcETUTFiCBP8tbufAQ+zBWIkxgBwUyAzUHbu1CvhwdXnX3AuKGKC9JLaErbMBUiT8Ar4UmdzyJIxXUqIRJLaKiDGm2n0kSX/Phc/
aKZNCkAWYAH4MA6H+GQF2K5E8Esy6H4iXBGp1e1Vvvu5fyk1dFFnVkx9UnfkE4PlKnhNd17rA4OfN09jJPPBZNXpLPxx6MaJw62WD1Pwt+1ZCzSoFfnsO2XZbtSyOL2eJzpoD3wlvmBxfVn3RfZyEAHJ8AM4ZJKrvPP8lC8bUtKy/M4Mj0PLzNMLTKqgUv7Y9OFqjK1wG7uYlUoebV+RqRacTKKdNR6h48mD+yXw6MVaIYlm2cqH
mbjjVyaxpqpGr0wcBdFBtlS+LAcPcJVIy1b5ZwHD1Y56VKwRWTT9mqCzOcuG6GSgVMg4B8pbBoB6dG8bmQiUnamnbbIyGMHqr2sS4s8+hn5pqVVFXzm6byWVDeD4NAzuK7SGdvBGUiuKd+x2DcXhSdUkJtV/vIJGZQAzwTEZFR4OQdHsjK3vGq3O0D7ZRUzYYEQbSD/Yu/BzrN/p5XYsxtWCsbCnJ5VrddAE45HscqchqQ/Y8w
OpYRbEbd9B7xBnDQYyyUVPhVbcrZs2SGI6aXVJVLZzBfwzZQvLXH9uqQhZt8wYtWqNFKUqPMa3MSmDuDJj5twrHO8aVHz9OnVe+u4Lx7w6/LgdVhxDZwbBoGmU7CE8cVXjVQ2BbHMvE+N6zMV7f+Dwx3jEdW3X2W0LY91vbHp5HOd9lPWcmoULQw0d1wcTf6RvjtE9RnyuzTrFLpBzDgSXN5fqW7YgDyZYm1IxW3fd8JDUHN9kTiicDD+
ri1xTdtqB8GM2XA2Ha9bCPn3G+sj68Lpx1cs7LdFA4Zug+p+Y+uz8qh2XymEi6oQ3yjOuK1S+B+6Kba6VTr76TemOwZMGSh7uwXZSHPg7HKaqVIOyJ9h1jkXVZZN9dSKZQgXHZ3kQlCXQ8OVwLBF3hd2uWOaRKEHSjrVu7WHoEakbOZmFS5E8g7aMgLGrjC5i3zOWfJxXx4MdNdW3dX4BBcE8XQ2Fk
Il5ZlV5uXnNNw/3q7++AHoUGvgPPEQpkMRxUNAhjLJqZGbvdQ9RFGdzEPk81u2ZtPQJLoHe0qhEnxeEZ7zWQ7ZjytiEDVGUdOxBnGIRvQGqJMBHrh7htV81hcBvkm7Dsy2S3s/fh1qtLu6e5TP7w0nvjW4jCW/AF4xcUj5OTl4ZHoZDFudAfLnS4IvJqV3DVm5Bvv9+Kws+2IDs9ebfRCb0eJiaTu1fV1sbj7
PxMsbJLZGAeIvoUgvPbLwuncZH3mS8LqBTLiZMXvfAAHaKAJRaONjWm56PSgxeGd5bRhYeNpsvwAqxN98kjmTOQeyE09cVOTH/YuU89g6xcOHZ0Yc1PcFoeMVyTUwJB3mFiAbI1ab+SPJi8zqgFRVfgB1NpiK8FabG/iZsGD/D2djMt1WE2WZF6mQkVCn78zV8AegeMjv2FagFtNQefE9JyT6dtHzl2IlIftz9QnlLlFw0kuiBFiL2
eFpfrxuMxCHmRu3JVRJGGLQMZ3F84Tk2dVxxNWWiD6hfXnvFtFb6hxf/kvqnXS7xAxH+VveAjK/2lsu2z7bPc/HzMPbIY9+SUm8xKnDDLAaS4kCAqL8kYPlf4XjPLfwxsOtNRBjlYqD6CWlpd5bz+eeyPwh0EcDpTRl6kD1qvmzqjvrlV1ssXCrZD7MbmkfaEuR6P+LN+jtMAS4lfOHUoKAgAkAmlyYKSGU7bdB+NCJzHHXlOCMsgmm2lFTL0JVzwU4wN2cLc+xnNgYBO0IHnXKIWO1EkWmzVqXmy3DUVO1FS1LGHSKNV6ZB0i4RxIgUmYFE3r3DeG5dwVDHRtwhGnNRBjF26+EZBxVYN3b0v1ZM6jVdGwZjmaFtwvlBLCJ5RXf1Vl3DhnQpEsJNiiYhv7I4G8vqeJCSHMktW1tVcZyiW0vMCAtSiLy40MIbMN6FdVW6ltjbWjvVvlWxVVrQZHUSkIPjLvVZ0QqkJNlkDaMHXv7O6BhSkAt5ahLGHE+eMXl8F5bkWGSVhiIPAW+2nSUAxLINZY2dfl0YqIAGjIVjn

What else is there

- Lambda Functions
- Decorators
- Handling Exceptions
- Classes and Objects
- Generators
- and other stuff.