

direct = data  
comp = 12

Open ( ' ' data/12/POSCAR )

Open ( direct + comp + poscar  
direct/comp/poscar

PYTHON

PYTHON

Python ✓ (Algorithms)


(Getting Started with Python)

- Indentation is important (tab & space are different) <sup>↑ use (effective)</sup>
- elif statement
- try <sup>except</sup> ~~except~~ (used when in doubt, that statement gives an error)
- raw\_input(), write thing to displayed (value <sup>inputted</sup> by user can be saved in a variable)
- def hello(): (defining a function) - stored (& reused)
- ↳ hello() ← calling/invoking the function
- ↳ type string float int
- ↳ a = hello() → if hello returns a value, is stored in a
- ↳ anything after returns <sup>line</sup> in a function is left out
- while
- break - breaks the loop / iteration
- continue - breaks only the current iter<sup>n</sup> & jumps to the top of the loop & starts the next iteration


\* for (definite loop)  
→ for i in [5, 4, 3, 2, 1]:

= when the first value is Not decided

USE **NONE**

when comparing if a ==  95%

is operator

when using true, false none  
use if a is  1% of time

fh = open ("some.txt", "r")  
↑ read



(only) → length of string

string concatenation - a = 'hello', b = a + 'there' /  
print b → hellothere  
for space b = a + ' ' + there

fruit = 'banana'

library - zap = greet ~~lower()~~ lower() → greet

→ ») dir(stuff)  
['capitalize', 'center', 'count': ...]

→ `greet + Hello Bob'`  
`greet.replace('Bob', 'Jane')` | `greet.replace('o', 'x')`

\_\_\_\_\_

Prefixes

↑ this all is called  
passing  
→ find thing in a text, cutting  
repeating it, etc. ~

\* handle = open(filename, mode)      fhandle = open('mbor.txt', 'r')

file handle `open` for `read` can be treated as a sequence of (string) → where each FILE is a STRING in the sequence

extra new line  
 → find - new line (not a dot)  
 where you skip  
 make it automatic  
 takes the read  
 !  
 an extra new line  
 comes after 2 fr

for line in fhand:  
print line ← extra line — ~~next~~ next line b/c of file w/ an extra line — poss. come  
↓  
.. delete next line from file line by (line = line.rstrip())

`range()`  
`range(3)`  
`range(0, 3)`

Range friends = ['Joseph', 'Gina', 'Bella']  
 for friend in friends:  
     print 'happy new year', friend  
 for i in range(len(friends)):  
     friend = friends[i]  
     print 'happy new year', friend

Concatenate lists  
`A = [1, 2, 3]` `B = [4, 5, 6]` `C = A + B`  
`C` is `[1, 2, 3, 4, 5, 6]`

Lists can be sliced similar to strings.  
`a = [1, 2, 3]` → `a[1:3]` → `[2, 3]`  
 A ↑ not needed

→ `not in` `in`  
`1 in A` → True  
`2 not in A` → True

→ `sort()` `list.sort()`  
     ↑  
     sorts itself.

→ Built-in functions & lists  
`len()` `max()` `min()` `sum()`

→ Append → (add an item at end of list)  
`list.append(2)`

Best friends: LISTS and STRINGS.

`abc = "with three words"`  
`stuff = abc.split()` → looks at many spaces as equal to 1 space & separates them  
`stuff` → `['with', 'three', 'words']`

BUT if `string = 'A;B;C'`  
`list = string.split()` → `['A;B;C']`  
 (∵ no spaces, still single object)

What to do.

`list = string.split(';')` [Character other than space]  
`list` → `['A', 'B', 'C']`

Dictionary

Start with making an empty dictionary

`a = dict()`  
`a['m'] = 1`  
`a['n'] = 2`

or `a = {'m': 1, 'n': 2}`  
     ↑  
     key

→ `get()` (very useful in dict)

`counts[name] = counts.get(name, 0) + 1`

if name is there in dict  
 ignores, get the value and perform the operation  
 or else  
 adds the name to dict & assigns the default value & performs the operation



dict to list

[m:1, n:2]

Double iteration variable python in row

counts.keys() = [m, n]

counts.values() = [1, 2]

list(counts) = [m, n]

counts.items() = [('m', 1), ('n', 2)]

list

list of tuples

j = {'chuck': 1, 'fred': 42, 'jim': 100}

→ for aaa, bbb in j.items():

print aaa, bbb.

↓

jim 100

chuck 1

fred 42

converts dict to tuples

l = list()

dir(l)

['append', 'count', 'extend', 'index', 'insert', ..., 'remove', 'sort']

t = tuple()

dir(t)

['count', 'index']

tuples are

immutable list

stored as []

list

sorted()

list name

sorted(~~dict~~ items())

tuples

→ sorts the list

mp.sort()

tuples can be sorted based on keys

we can sort <sup>tuple</sup> based values, if we construct a list of tuple of form (value, key).

↳ using for loop

to sort in descending order

tmp.sort(reverse=True)

very shorter version for sorting based on values

print sorted([(v,k) for k,v in c.items()])

↑

## Python to ACCESS WEB DATA

**Regular expression** (Language of characters)  
can be used in other languages  
(python, Java, etc...)

To use regular expression,  
Import regular expression library  
→ import re

**find()**  
hand = open('inbox-short.txt')  
for line in hand:  
line = line.rstrip()  
if line.find('From:'): **re.search()**  
print line  
states position  
returns true or false

**if line.startswith('From:'): if re.search('From:', line):**  
print line  
startswith (regular ex. begins, checks)

### Wild Card Characters

→ **X.\*:**  
states with  
Match any character  
any no. of times  
before colon

→ Fine-tuning your match

**X+:**  
one or more before times colon  
captured → non-blank char. (+)  
small (x) → blank char.  
oops.

**[0-9]+** → one or more type of characters  
↳ b/w sq. brackets is one character.

yes:-  
import re  
x = 'my 2 fav. no. are 7 & 13'  
y = re.findall('[0-9]+', x)  
print y  
['2', '7', '13']  
searches & finds what is stored

if you want or similar that in code  
use backslash  
e.g. dollars really \$

+ → greedy From: start to x!  
+? → non-greedy From: start to x!

### Fine-tuning string open

From stephen.marquard@uct.ac.za Sat Jan 5

1) **y = re.findall('\S+@\S+')**  
↑ ↑ ↑  
At least one non-whitespace character  
start end of what to extract

2) **y = re.findall('From (\S+@\S+)')**  
↑ ↑  
startswith if From. extract  
(return only how parentheses part)

3) **y = re.findall('[a-zA-Z]\*')**  
↑ ↑  
not space (include programming language)  
match many of them  
not blank character

4) **y = re.findall('From: [a-zA-Z]\*')**  
↑ ↑  
From: set  
match many of them  
not blank character

ever coded vector  
'From: [a-zA-Z]\*'



I.

connecting to servers in Python  
Using socket library

① `import socket`  
② `mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`  
becomes just a handle, like open for file.  
③ `mysock.connect(('www.pythinf.com', 80))`  
host port  
connects to other end

(a) → `mysock.send('GET http://www.pythinf.com/...txt HTTP/1.0\n\n')`

while True:

`data = mysock.recv(512)`

if (len(data) < 1):

break

print data

`mysock.close()`

Easier way to connect  
using urllib in python

① `import urllib`

② `fh = urllib.urlopen("http://www.pythinf.com/...txt")`

for line in fh:

`print line.strip()`

(DIFFERENCE)

in urllib - we DON'T get headers  
we only GET CONTENT

Passing data from HTML

Beautiful Soup library

download beautiful soup.py & place it  
in same dir as your python code.

`import urllib`

`from BeautifulSoup import *`

`url = raw_input('Enter - ')`

`html = urllib.urlopen(url).read()`

`soup = BeautifulSoup(html)`

# retrieve a list of anchor tags

# each tag is like a dictionary of HTML attributes

`tags = soup('a')`

for tag in tags:

`print tag.get('href', None)`

of all a tags

{a href=...}

XML

extensible markup language

`import xml.etree.ElementTree as ET`  
`import urllib`

`tree = ET.fromstring(urllib.urlopen(url).read())`

data: "1 2 3"  
xml data:

document tree

## JSON

JavaScript Object Notation

json.org

Python we make  
lists & dictionaries  
JavaScript  
arrays & Hashmaps  
JSON → Both

description

import json

info = json.loads(data)

print

what we get is dictionary

print 'Name:', info['name']

Accessing Program Interface

API library documentation

using json

API security & Rate limiting → API security key

Import OAuth to protect  
Import hidden from twitter

## GIT ESSENTIALS

## GIT ESSENTIALS

git init  
git add

git commit -m "Initial commit - a file for anything"

git log

git log -n 0 or 10...

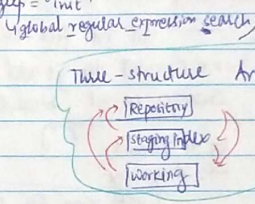
git log --since=2017-03-26

git log --until=2017-03-26

git log --author="Mobi's Name"

git log --grep="init"

main changes  
- add changed  
- commit changes



git was originally developed for source control management (scm)

Gives info about commits

Shortcut  
staging & committing at once  
git commit -a

Git uses - a checksum snapshot mechanism (SHA-1 hash algo)

generates checksum (40 char hexadecimal string 0-9, a-f)

levels of configurations → system / git config --system

User / git config --global

Project / git config

good habit to write messages - 1st line less than 50 char (descriptive but concise)

- present tense (optional)
- gap line + description (detailed) less than 72 char per line
- add tags (depends on what you're accomplishing, but keep it standard)