Python Notes:

① Expression ≅ values and operators

Math Operators

** Exponent

% Modulus/Remainder

// Integer division

* Multiplication → this can be used with string + number

- Subtraction

+ Addition → Can be used with strings but not string + number. (concatenate

Rules of variables → statement

ⓐ Only one word

ⓑ letters, numbers and '_'

ⓒ Can't start with number

comparison Operators

== equal to

!= not equal to

< less than

Important functions
less
input() gets the input from the user
str() transform value to string
int() " " " int
float() " " " float
range() → used for loop

else if → 'elif'

Infinite loops
- break → gets outside the loop
- continue → resent the loop

for x in range(n) → something happens → this range of have a specific number interval (5, 6)

⇓

Values would be only 5 & 6

A third value can be also input that would be the value increment

Also values start from 0 and finishes in (n-1) value

range (s, en, st)

s: start, en: end, st: step

Modules → contains extra functions apart from (input(), len() int(); str(), float())

MUST BE IMPORTED!

Example

import random ⇒ import modules separated by commas

random.int(1, 10) → returns a random int

import random import * → allows you not put the "random." part (not recommended)

sys.exit → exits the program

3rd party programs → pip tool → pip install <modulename>

pyperclip → allows to copy & paste to clipboard

Other built in functions Abs(n) → returns Absolute value of n
round(n,e) round n to e Amount of decimals

```
def Function()
    global eggs → this is a change the variable global eggs
    print('Here is whatever the function does')   } function body
    eggs+= 1
```

Exception Handlers

```
def div42(divideby):
    try: 42/divideby
    except ZeroDivisionError:   ← can have no specific error type
        print('Error: You tried to divide by zero')
```

Types of errors - ValueError
ZeroDivisionError

no None ~ Null

Lists: uses [] numbered from 0 to n-1
can contain other lists inside [['A','B'], ['C','D']]
can be negative values to go backwards
AA [0:0] → refers to return values from 0 to a of the list AA.

len(list) will return nº of values in a list array

+ And * can be used to add lists and replicate them.

del → used to delete an item in an array list

.index('str') → returns an index value of a 'string' in a list
.insert (n, 'string')
.Append ('string')
.remove (str)
.sort()
.sort (reverse=true)
.sort (key = str.lower)

Strings & tuples

a[0] → returns 1 char of string A
you can also use -, n:m, in, not in

tuples → added with () instead of []

↳ Don't intend for sequence of values to change

Regular expressions → specifying text patterns

```
import re
re.compile() → create regex object
call regex objects .search() → create a match object
match object .group() → get the matched string

bat Regex = (r'Bat(wo)?man') → 0 or 1
            (r'Bat(wo)*man') → 0 to many 'wo'
            (r'Bat(wo)+man') → 1 to many 'wo'
            (r'(Ha){3}') → searching for HaHaHa
            (r'(Ha){3,5}') → searching between 3 to 5 repetition of Ha
                ↳ ? → transforms it to a non greedy search
```

\d : Numeric digit 0 to 9
\D : Not a numeric digit
\w : letter, digit or underscore "_"
\s : space, tab or newline

(r'[ ]) → define own possible characters

(r'^ ') → Matching must occur At start of text

(r' $') → Matching if string ends with the regex
(r'.*') → everything '.' is wild card
(r'.*', re.DOTALL) → match all characters

(r' ', re.I) → Case insensitive

.Sub ('replace', 'Phrase to replace the regex

you can use ''' to do Multiple lines and # to comment

(r'''(#Here is element 1
        #Here is element 2
)''', re.VERBOSE → ignore whitespace & comments

you can use | to use multiple properties
example (r' ', re.IGNORECASE | re.DOTALL | re.VERBOSE

Reading and Writing Files

os.path.abspath('.') → returns the absolute path of the pwd

os.getcwd → pwd

os.chdir(' ') → cd

path.isfile ⟶ boolean
path.isdir, .isfile, .exists

- .path.getsize → return file size
- .listdir → lists a ' folders inside
- .makedirs → mkd.

Reading & Writing Files:

open('filepath')

- .read() → read the content
- .readlines() → read the raw content
- .close() → closes the file

- .open('filepath', 'w') → writes new content on file
  'a' → Appends after the previous content
  } if file doesn't exist it creates one

- .write('content') → adds content, doesn't add a new line

shelve module:

```
import shelve

shelfFile = shelve.open('mydata')
shelfFile['example'] = ['Ex1', 'Ex2', 'Ex3']
shelfFile['example']
```

Anibal.solis@gmail.com

shutil Module

shutil.copy('file path', 'paste path') → cp

shutil.move('', '') → mv

How to delete files:

import os

1) os.unlink('single file path')
   os.rmdir('directory path') → Needs to be empty

2) import shutil
   shutil.rmtree('dir path') → deletes directory and contents

import send2trash

send2trash.send2trash('file path')

Walking A directory Tree

```
for foldername, subfolder, filenames in os.walk('root directory'):
    print('The folder is: ' + foldername)
    print('the subfolders in ' + foldername + 'Are ' + str(subfolders))
    print('the filenames in ' + foldername + 'Are ' + str(filenames))
```

Zip files: Required to create a 'ZipFile' object → zipfile.ZipFile('filepath')

```
import zipfile
exampleZip = zipfile.ZipFile('example.zip')
exampleZip.namelist() → lists the files contained
fileInfo = exampleZip.getinfo('file contained') → creates information object

fileInfo.file_size → get the uncompressed size
        .compress_size → get the compressed size
exampleZip.close() → stops using the object

exampleZip.extractall() → extract all the files contained
exampleZip.extract('filename', 'extract location')
```

How to create a zipfile:

```
import zipfile
newZip = zipfile.ZipFile('new.zip', 'w') → writable
newZip.write('filename', compress_type = zipfile.ZIP_DEFLATED) → Add a file
newZip.close()
```

```
raise exception("This is an error message") → throws an exception

Sanity check

Assert False, 'This is an error message'
```

Loggin Module

```
Import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s
- %(message)s

logging.debug('logging message')
```

Web scrapping:

Modules: webbrowser → opens A browser page
requests → downloads files and webpages
beautiful soup → parses HTML
selenium → launches & controls A Web browser

webbrowser.open('web address')

requests module:

- requires external installation (pip install requests)

res = requests.get('web address')
res.status_code → return status of website { 400 → Not found, 200 → found }
res.text → text of the website
bad res.raise_for_status() → raises the possible error code in case of failed download

playfile = open('file name', 'wb')
↳ open in binary (required for web txt)

for chunk in res.iter_content(n°bytes) { for loop to pass chunks of the data scrapped (avoid errors)
playfile.write(chunk)

Scrapping HTML (beautiful soup4 module

pip install beautifulsoup4
import bs4 → beautiful soup 4
import requests

```
res = requests.get('urladdress')
soup = bs4.BeautifulSoup(res.text)
```

Selenium Module:

```
from selenium import webdriver
browser = webdriver.Firefox()
type(browser) → browser starts up
```

PyPDF2:

```
pdfFile = open('pdf location')
reader = PyPDF2.PdfFileReader(pdfFile)
reader.numPages → return number of pages of the PDF
page = reader.getPage(n) returns the page n of the pdf
page.extractText() extract the text of the page object
```

CSV And JSON

```
import csv
eF = open('example.csv')
eR = csv.reader(eF)
eD = list(eR)
```

} Reader

```
import csv
oF = open('output.csv', 'w', newline='')
oW = csv.writer (oF)
oW = write row (['insert', 'data', 'here'])
oF.close()
```

} Writer

```
import json
jsonObject = json.loads(jsonString)
```
→ this, takes a string and loads it as python values

```
jsonObject = json.dumps(values)
```
→ translate python values as a json string

Time Module:

```
import time
```

time.time() → number of seconds since January 1st 1970

```
import cProfile
cprofile.run(comand/function)
```
→ gives time status of a command/function executed

```
time.sleep
round(variable, no. of decimals)
```
→ rounds a float to a certain no. of decimals

Datetime Module:

import datetime

datetime.datetime.now() → current datetime

dt = datetime.datetime(2015, 10, 21, 16, 29, 0) → creates a datetime object
(dt.year, dt.month, dt.hour, dt.minute, dt.second)

delta = datetime.timedelta(days=11, hours=10, minutes=9, seconds=8) → represents duration rather than moments

delta.days delta.seconds delta.microseconds → returns the value
delta.total_seconds() → returns the total amount expressed in seconds
str(delta) → transforms into str.

* timedelta values can be added with each other

while datetime.datetime.now() < Hallowen 2016: → condition that does not execute till
time.sleep(1)

strftime() → from datetime object to string

%Y → Year with century
%y → year without century
%m → number month
%B → name month
%d → day of the month
%w → number day of the week
%A → full week day

example dt.strftime('%A %B %d %y %I:%M %p')

converting str to datetime → .strptime('date string', 'string format')
(uses the same notation as strftime)

Threading:

Threading

```
import threading
threadObj = threading.Thread(target = <MethodName>)   → This allows more than 1 method to be executed at the same time
threadObj.start()

Passing arguments to the threaded method executed:
threadObj = threading.Thread(tAget = <MethodName>, args=[<Arguments>]

** You need to be aware of concurrency issues when multithreading
multipleThreads=[] #list of all the threaded objects
for i in range(0, <NoItems>, <items per thread>):
    multipleThread = threading.Thread(target=<MethodName>, Args(i, i+<NoItems>))
    multipleThreads.Append(multipleThread)
    multipleThread.start()
for multipleThread in multipleThreads:
    multipleThread.join()                  } This waits for the theads to stop
Print("Done")
```

Sub process:

```
import subprocess:
    subprocess.Popen(<program location>, '<Argument textfile>')   → (optional)

    subprocess.call("command")  → performs a command in the shell
    subprocess.Popen(<python location>, <python script location>)
```

Useful Methods: .poll .wait

program

open a default program:

```
import subprocess
subprocess.Popen (['start', '<file location>'], shell = True) → Windows

subprocess.Popen (['open'],'<file location>') → Unix
```

Sending Email & Text Msg:

SMTP → Simple Mail Transfer Protocol (Protocol used to send email)
IMAP → Protocol to retrieve Email sent back