## DATA TYPES:

LISTS are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end. EG: Your many cats' names.
TUPLES are just like lists, but you can't change their values. The values that you give it first up, are the values that you are stuck with for the rest of the program. Each value is numbered starting from zero, for easy reference. EG: the names of the months of the year.
DICTIONARIES the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered. Key-value pairs EG: NUM = {'KYLE': 12, 'MICHELE': 13, 'AARON': 14, 'PRISCILLA': 15}

## LISTS[]:

Lists are extremely similar to tuples. Lists are modifiable (or ' mutable' , as a programmer may say), so their values can be changed. Most of the time we use lists, not tuples, because we want to easily change the values of things // if we need to. the code is exactly the same as a tuple, EXCEPT that all the values are put between square brackets, not parentheses. To add a value to a list, you use the '.append()' function. example: cats.append('Catherine')len() / range() / .append / split() / made for sorting / Mutable

```
Loops
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:',  friend)
```

```
split()
line = from a@gmail.com
word  = line.split()
email = word[1]     a@gmail.com
piece = email.split('@')    ['a', 'gmail.com']
print(piece[1])    'gmail.com'
```

## TUPLES():

Tuples are pretty easy to make. You give your tuple a name, then after that the list of values it will carry. For example, the months of the year:
months = ('January','February','March','April','May','June', 'July','August','September','October','November','December') //
You may have spaces after the commas if you feel it necessary - it doesn't really matter.

INDEXING IN TUPLES

Table 1 - tuple indicies

| Index | Value |
|---|---|
| 0 | January |
| 1 | Feb |
| 2 | Mar |

```
Immutable like strings
months = ('Jan', 'Feb')
Sorting
tup = (3, 8, 6)
sorted(tup)
```

## DICTIONARIES{}:

Name the data structure(s) that have a method called .items() Answer: Dictionaries.

WHAT IS THE OUTPUT AFTER THE FOLLOWING CODE HAS BEEN EXECUTED?

```
Diction = {"d":4,"e": {"g":4 }}
print(diction ["e"] ["g"]
Answer: 4
```

```
num = {'Kyle': 12, 'Michele': 13}
get()
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

```
Loops
counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for key in counts:
    print(key, counts[key])
```

NAME THE DATA STRUCTURE(S) THAT HAVE A METHOD CALLED .ITEMS()
Answer: Dictionaries

## GITHUB:

WHEN YOU CLONE A GITHUB REPO ON YOUR LOCAL WORKSTATION, YOU CANNOT CONTRIBUTE BACK TO THE UPSTREAM REPO UNLESS YOU ARE EXPLICITLY DECLARED AS "CONTRIBUTOR". Answer: False

BRANCHING is the way to work on different versions of a repository at one time.

HOW DO YOU SUPPLY A COMMIT MESSAGE TO A COMMIT?
Answer: git commit -m "description of what you're committing"

WHAT COMES FIRST, STAGING WITH GIT ADD . OR COMMITTING WITH GIT COMMIT?
Answer: staging your commits with git add

WHAT'S THE GIT COMMAND THAT DOWNLOADS YOUR REPOSITORY FROM GITHUB TO YOUR COMPUTER?
Answer: git clone

WHAT'S A SHORTCUT TO STAGING ALL THE CHANGES YOU HAVE?
Answer: git add-A .

WHAT IS THE CORRECT COMMIT SYNTAX FOR ALL CHANGES WITH A MESSAGE?
Answer: git commit -am"Your message"

WHAT'S THE OPPOSITE OF GIT CLONE, INSTEAD OF DOWNLOADING YOUR CODE FROM GITHUB, UPLOADS YOUR CHANGES AND CODE BACK TO GITHUB?
Answer: git push.

HOW DO YOU CREATE A COPY OF A LAB UNDER YOUR OWN GITHUB ACCOUNT SO THAT YOU CAN SOLVE THE LAB?
Answer: git fork

HOW DO YOU CHECK THE STATE OF YOUR LOCAL GIT REPOSITORY SINCE YOUR LAST COMMIT?
Answer: git status

What is the git command to see your activity?
Answer: git log

HOW DO YOU STAGE FILES FOR A COMMIT?
Answer: git add filename or -A

HOW DO YOU SAVE THE CURRENT STATE OF YOUR CODE INTO THE GIT VERSION CONTROL?
Answer: by committing the staged changes with git commit

| | |
|---|---|
| CLONE - | downloads your repository from github to your computer |
| INIT - | initializes the Git repository in which you will be adding, comiitting and pushing to |
| STATUS - | checks the state of your local git repository since your last commit |
| DIFF | changes to tracked files |
| ADD VS ADD . | git add will add in one particular file. git add . adds in everything in the directory |
| COMMIT | commits ur file to the interwebs of gitland |
| GIT LOG | shows all commits starting with the newest |
| GIT REMOTE -V | lists all currently configured remotes |
| GIT REMOTE ADD <SHORTNAME> <URL> | will give name for the URL on github profile |
| GIT PULL <REMOTE> <BRANCH> | pulls ur files from your online repo if u don't have them on ur local comp |
| GIT PUSH <REMOTE><BRANCH> | pushes your code to the web |

## PROJECT 1

```
mport os
import filecmp
import csv
import operator


def getData(file):
#Input: file name
#Ouput: return a list of dictionary objects where
#the keys will come from the first row in the data.

#Note: The column headings will not change from the
#test cases below, but the the data itself will
#change (contents and size) in the different test
#cases.

 lst = list()

 with open(file, 'r') as f:
  reader = csv.DictReader(f)
  for row in reader:
   d = dict(row)
   lst.append(d)
 return(lst)


#Sort based on key/column
def mySort(data,col):
#Input: list of dictionaries
#Output: Return a string of the form firstName lastName

 sort = sorted(data, key = operator.itemgetter(col))
 firstName = sort[0]["First"]
 lastName = sort[0]["Last"]
 return firstName + " " + lastName

#Create a histogram
def classSizes(data):
# Input: list of dictionaries
# Output: Return a list of tuples ordered by
# ClassName and Class size, e.g
# [('Senior', 26), ('Junior', 25), ('Freshman', 21), ('Sophomore', 18)]

 d = {}
 for classes in data:
  d[classes["Class"]] = d.get(classes["Class"], 0) + 1
 class_list = d.items()
 return sorted(class_list, key = lambda tup: tup[1], reverse = True)


# Find the most common day of the year to be born
def findDay(a):
# Input: list of dictionaries
# Output: Return the day of month (1-31) that is the
# most often seen in the DOB

 d = {}
 for day in a:
  d[int(day["DOB"].split("/")[1])] = d.get(int(day["DOB"].split("/")[1]), 0) + 1
 lst = list(d.items())
 lst_1 = sorted(lst, key = lambda x: x[1], reverse = True)
 return int(lst_1[0][0])


# Find the average age (rounded) of the Students
def findAge(a):
# Input: list of dictionaries
# Output: Return the day of month (1-31) that is the
# most often seen in the DOB
 lst = []
 for d in a:
  bday_yr = int (d["DOB"].split('/')[2])
  age = 2017 - bday_yr
  lst.append(age)
 yr = 0
 for age in lst:
  yr += age
 return int(yr/len(lst))


#Similar to mySort, but instead of returning single
#Student, all of the sorted data is saved to a csv file.
def mySortPrint(a,col,fileName):
#Input: list of dictionaries, key to sort by and output file name
#Output: None

 sort = sorted(a, key = operator.itemgetter(col))
 f = open(fileName, 'w')

 person = sort[0]
 for person in sort:
  f.write(person["First"] + "," + person["Last"] + "," + person["Email"] + "," + "\n")
 f.close()
```

## PROJECT 2

```
## Import statements
import unittest
import requests
import re
from bs4 import BeautifulSoup


## Part 1 -- Define your find_urls function here.
## INPUT: any string
## RETURN VALUE: a list of strings that represents all of the URLs in the input string


## For example:
## find_urls("http://www.google.com is a great site") should return ["http://www.google.com"]
## find_urls("I love looking at websites like http://etsy.com and http://instagram.com and stuff")
## should return ["http://etsy.com","http://instagram.com"]
## find_urls("the internet is awesome #worldwideweb") should return [], empty list

def find_urls(s):
    url = re.findall('http\S*\.[a-z]+\S', s)
    return url

## PART 2  - Define a function grab_headlines.
## INPUT: N/A. No input.
## Grab the headlines from the "Most Read" section of
## http://www.michigandaily.com/section/opinion

def grab_headlines():
    #Opening from opinion.html
    # f = open('opinion.html', 'r')
    # file = f.read()
    # f.close()
    #Opening from Michigan Daily
    r = requests.get('http://www.michigandaily.com/section/opinion')
    soup = BeautifulSoup(r.text, 'html.parser')
    most_read = []
    s = ""

    for most in soup.find_all(class_ = 'view view-most-read view-id-most_read view-display-id-panel_pane_1/
     view-dom-id-99658157999dd0ac5aa62c2b284dd266'):
        s = most.text.strip()
        most_read = s.split("\n")

    return most_read

## PART 3 (a) Define a function called get_umsi_data.  It should create a dictionary
## saved in a variable umsi_titles whose keys are UMSI people's names, and whose
## associated values are those people's titles, e.g. "PhD student" or "Associate
## Professor of Information"...
## Start with this page: https://www.si.umich.edu/directory?field_person_firstname_value=&field_person_lastname_value=&rid=All
## End with this page: https://www.si.umich.edu/directory?field_person_firstname_value=&field_person_lastname_value=&rid=All&page=12
## INPUT: N/A. No input.
## OUTPUT: Return umsi_titles
## Reminder: you'll need to use the special header for a request to the UMSI site, like so:
## requests.get(base_url, headers={'User-Agent': 'SI_CLASS'})

def get_umsi_data():
    umsi_titles = {}
    base_url = "https://www.si.umich.edu/directory?field_person_firstname_value=&field_person_lastname_value=&rid=All"
    r = requests.get(base_url, headers = {'User-Agent': 'SI_CLASS'})
    soup = BeautifulSoup(r.text, 'lxml')

    for i in range(13):
        if i == 0:
            base_url = 'https://www.si.umich.edu/directory?field_person_firstname_value=&field_person_lastname_value=&rid=All'
        else:
            base_url = 'https://www.si.umich.edu/directory?field_person_firstname_value=&field_person_lastname_value=&rid=All&page={}'.format(str(i))
        r = requests.get(base_url, headers = {'User-Agent': 'SI_CLASS'})
        soup = BeautifulSoup(r.text, 'lxml')
        field_name = soup.find_all('div', {'class': 'field-item even', 'property': 'dc:title'})
        field_position = soup.find_all('div', {'class': 'field field-name-field-person-titles field-type-text field-label-hidden'})
        for field_item in range(len(field_name)):
            umsi_titles[field_name[field_item].text] = field_position[field_item].text

    return umsi_titles

## PART 3 (b) Define a function called num_students.
## INPUT: The dictionary from get_umsi_data().
## OUTPUT: Return number of PhD students in the data.  (Don't forget, I may change the input data)
def num_students(data):
    phd_students = 0

    for key in data:
        if data[key] == 'PhD student':
            phd_students += 1

    return phd_students
```

## MISC..

WRITE A FOR LOOP TO DISPLAY ALL THE ELEMENTS OF A TUPLE.

```
sammy_shark = {'name': 'Sammy', 'animal': 'shark', 'color': 'blue', 'location': 'ocean'}
    for key in sammy_shark:
   print(key + ': ' + sammy_shark[key])

for key,value in sammy_shark.items():
print  (key,value)
Output:
(name, Sammy)
(animal,shark) .. etc.
v
(2.) continued
Output
name: Sammy
 animal: shark
 location: ocean
 color: blue
```

WHAT IS THE DIFFERENCE BETWEEN URLLIB.URLOPEN(URL) AND URLLIB.URLOPEN(URL).READ()
Answer: Opens url            reads the url as a string

WHAT HAPPENS IF YOU FORGET THE "-M" IN A GIT COMMIT COMMAND?
Answer: The VI editor will pop up and messy things will happen.
(There is another answer as well.)

Also, if you can explain the 2nd answer would be -
Or git commit -a
#for all local changes

DO YOU JSON.LOADS() OR JSON.DUMPS() TO CREATE A STRING?
Answer: json.dumps()

Databases
Database - contains many tables
Relation (or table) - contains tuples and attributes
Tuple (or row) - a set of fields that generally represents an "object" like a person or a music track
Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row

Given the following code, you want to sort lst1 on the third element of each list. Write the code for three different options for replacing ??? .
Using lambda. Key = lambda k: k[2]
Using itemgetter() key = itemgetter(2)
Using a separate function three differente

REMEMBER STRING METHODS AND FUNCTIONS LIKE LEN AND SPLIT!

Write a regular expression that describes a string with no digits in it with no digits in it that is at least 4 characters long.
Answer:
re.findall([^0-9+]{4,})

## REGULAR EXPRESSIONS

| | |
|---|---|
| ^ | Matches the beginning of a line |
| $ | Matches the end of the line |
| . | Matches any character |
| \s | Matches whitespace |
| \S | Matches any non-whitespace character |
| * | Repeats a character zero or more times |
| *? | Repeats a character zero or more times (non-greedy) |
| + | Repeats a character one or more times |
| +? | Repeats a character one or more times (non-greedy) |
| [aeiou] | Matches a single character in the listed set |
| [^XYZ] | Matches a single character not in the listed set |
| [a-z0-9] | The set of characters can include a range |
| ( | Indicates where string extraction is to start |
| ) | Indicates where string extraction is to end |

```
re.search()
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print (line)
find()
if line.find('From:') >= 0:
        print (line)
startswith()
if line.startswith('From:') :
        print (line)
re.findall()
re.findall('@([^ ]*)',line)
```