SYMBOLS

#! Shebang: path to program you’re trying to use

// integer division

% remainder of integer division

== is equal to

!= is not equal to

<= is less than or equal to

>= is greater than or equal to

& and

| or

! not

\*\* exponent

SYNTAX

set a variable: var = 12; var = “this is a string”

pass argument to command: command (argument)

list methods (function specific to the object): object. (press tab after the period)

DEFINITIONS  
 list🡪an ordered sequence of values

element🡪a value within a list

tuples🡪lists where the elements can’t be changed or reordered

set🡪collections of distinct objects, no duplicate entries, have special operators

union: value in set 1 or set 2

intersection: values in set 1 and set 2

difference: values unshared between set 1 and 2

dictionary🡪elements indexed by keys (order does not matter)

local variable 🡪 only exists in one part of code, then disappears (like in a function)

module 🡪 a single file containing multiple functions

package 🡪 collection of modules that can be imported into python

mutable 🡪 variables containing mutable types can be modified

to copy: b = a.copy()

to copy more complex objects: b = a.deepcopy()

immutable 🡪 variables containing immutable types cannot be modified

to copy: b = a

function scope 🡪 name of a variable within a function

global scope 🡪 name of a variable outside a function

FUNCTIONS

python3: change terminal from bash to python

help: get help

ctrl+d: return to bash terminal

“thing to print”: print a string

If the text string contains single and double quotes, “““ ‘words and “words”’ ”””

vars(): show user-defined variables

“string” + “string”: add a string to another string (result: “stringstring”)

str(thing): turn a variable into a string

int(thing): turn a variable into an integer

len(string): length of a string (in characters)

abs(value): absolute value

pow(base, power): base^power

print(variable): print variable to string

round(num): round to integer

help(command): get help

help(object.method):get help on a method

Things you can do with a string

object.replace(“a”, “b”): a replaced with b

object.find(“a”): find the first occurrence of a (starts with character 0)

object.count(“a”): count the number of a’s that appear

object.split(): split string at space

object.split(“a”): split string at a

object.strip(): remove spaces at beginning and end

object.upper(): make upper case

object.lower(): make lower case

“something”.join(list): combine list into one string, elements separated by something

alternatively 🡪 “string”.method(): do the same thing but string not saved as variable

name\_of\_list = []: create empty list

list(“numbers”): create a list of one-digit numbers from a string like this: “18734623”

name\_of\_list[number]: get the element at place #number in the list (starting at zero)

name\_of\_list[number1:number2]: get a range of elements (doesn’t include second)

name\_of\_list[:]: get all elements

name\_of\_list[:number]: get all elements before spot #number

name\_of\_list[number:]: get all elements after and including spot #number

name\_of\_list[-number]: get the elements that is #number from the end

Things you can do with a list

list.append(number): append element to end of the list

new\_list = list.copy(): copy list to a new list

list.clear(): clear all items from a list

list.count(“thing”): count number of occurrences of a thing

list.index(“thing”): return the index of the first matching element

list.pop(): remove last element of a list and return it

list.sort(): sort the elements in the list (best for numbers/characters)

list.reverse(): reverse the order of the elements

del(list[number:number]): delete a range of elements in a list

dictionary={}: make an empty dictionary

dictionary={“key”: “value”, “key2”, “value2”, …}: create a populated dictionary

dictionary[“key”]: retrieve the value of the called key

dictionary[“new\_key”]=new\_value: add a new key with its own value

Things you can do with a dictionary

dictionary.copy(): copy a dictionary

dictionary.clear(): clear all elements

dictionary.get(“key”, value): get value from a key, if dne returns default value

dictionary.keys(): create a list of all the keys

dictionary.values(): create list of all the values

dictionary.pop(“key”): remove the specified key and return its value

dictionary.update(dictionary): combine two dictionaries, order matters, values that are

different will be overwritten

tuple=(value, value, value): create tuple (same as list but can’t be changed)

tuple[element]: call value of element

Things to do with a tuple

tuple.count(thing): count occurrences of thing

tuple.index(thing): index of first occurrence of thing

Things to do with a set

a&b: intersection

a|b: union

a^b: difference

type(variable): determine what type of variable it is

max(thing): return largest item in string, list, or tuple

min(thing): return smallest item in a string, list, or tuple

sum(thing): return sum of the elements of a list or set

thing in thing: see if a thing is in another thing (returns true or false)

conditional branching 🡪

if condition:

thing to do

else:

thing to do

elif condition:

thing to do

#else if, works like a nested if else statement

looping 🡪

while condition

thing to do

break: stop the cycle

continue: skip the remaining code in the loop, move on to next iteration

for thing in thing:

thing to do

range(start, stop(non-inclusive), step) loop through a range of values

enumerate: creates a list of tuples

list comprehension – example:

a = [1, 2, 5, 14, 42, 132]

b = [x \*\* 2 for x in a]

#b ends up being equal to [1, 4, 25, 196, 1764, 17424]

open(“path\_to\_file\_to\_open”, “code”) #code can be w for write, r for read, a for append

filehandle = open(“path\_to\_file\_to\_open”, “code”)

filehandle.name: returns name of file

filehandle.mode: returns w, r, or a

filehandle.encoding: returns the way the file contents are interpreted

filehandle.read: return the whole file content as a single string

filehandle.readline: return the current line as a string

filehandle.readlines: return list of strings, each element being the corresponding line in the file

filehandle.write(string\_to\_write + “\n”): write a string then move to next line

filehandle.writelines([“line one\n”, “line two\n”, “line three\n”]): write multiple lines

filehandle.close(): close the file

with open(“path\_to\_file”, “code”) as filehandle:

things to do

#automatically opens and closes the file

filehandle.seek(0): return to beginning of file

filehandle.next(): skip over a line

enumerate(filehandle): I don’t really understand, but useful for creating iterator

import csv: use csv module

print(line.strip()): print line deleting leading/trailing spaces

csv.DictReader(filehandle)

print(dict(row)): print a row as dictionary

#check out page 115-116 for how to turn text into a dictionary

reader=csv.DictReader(filehandle)

header=reader.fieldnames #extract the header

writer=csv.DictWriter(filehandle, fieldnames=header, delimiter= “delimiter”)

writer.writerow(row) #write row

to define a function 🡪

def function\_name(user\_input\_variable):

function content

to set default values in function 🡪

def function\_name(usr\_input\_var1 = default\_val1, usr\_input\_value2 = default\_val2):

function content

import sys: allows us to read in variable from the bash command line into python

if \_\_name\_\_==”\_\_main\_\_”: basically “if there are arguments in command line”

var=type(sys.argv[1])

var=type(sys.argv[2]

#etc

#this is is similar to var=$1, var=$2

import module

from module import function

import module as new\_name

from module import \* (not recommended because it can overwrite functions of the same

names)

pickle objects 🡪

import pickle

pickle.dump(object, open(“object.pickle”, “wb”)

unpickle objects 🡪

name = pickle.load(open(“object.pickle”, “rb”))

print(object)

handling exceptions 🡪

try:

thing that may result in error

except name\_of\_exception:

what to do if an error is encountered

%pdb: turn on/off python debugger

Ipdb>: examine value of all variables from individual lines of code

assert: for debugging

scipy.random.seed(number): set seed

debugging tests: doctest, unittest, nose, mock

%run -p code.py: run profiler (to see what is taking most time)

global variable: do right after def function (next line) to change from function to global scope

Python debugger

import pdb: import python debugger

pdb.set\_trace(): put a break point in the code

next: execute next line of code

step: step into the next line of code (from here you can execute any other python code

you like, ie: figure out what a variable is set to at this point)

cont: continue executing code until the end of the script or next break point

numpy/scipy (use inside python3)

import numpy as np: get started with import

ARRAYS

np.arange(length): create 1d array similar to a list

(example: np.arange(4) 🡪 array([0, 1, 2, 3])

note: different behaviors than list, for example: you can’t add 2 to a list

array.shape: give length along each dimension

array.ndim: give number of dimensions

array.dtype.name: give data type of content

array.size: give number of elements

array.sum(): add elements

array.mean(): average elements

array.std(): standard deviation

array.min(): minimum

array.max(): maximum

np.sqrt(array): square root each element

np.exp(array): e^ each element

1d\_array = np.array([element, element, element])

2d\_array = np.array([[element, element], [element, element]], dtype = )

np.zeros((length\_of\_d1, length\_of\_d2), dtype = float): make array of zeros

np.ones((length\_of\_d1, length\_of\_d2), dtype = complex): array filled with 1+0i

array= np.arange(num); array.reshape((d1, d2)): rearrange vect of ints into array

np.random.random((d1, d2)): array of random values

array [n]: call the nth value in an array

array [:n]: call 0-nth value in an array

array[-n:]: call the last n values in an array

np.arange(n^2).reshape((n, n)): 2d array of nxn

array[val:val, val:val]: extract submatrix

array[n]: extract nth row

array[:, n]: extract nth column

array.sum(): sum whole matrix

array.sum(axis=0): sum by column

array.sum(axis=1): sum by row

READING DATA FROM A CSV

array = np.loadtxt(“filename.csv”, delimiter=””): load text from csv

IMAGE PROCESSING

typical structure: height (y), width (x), color (z)

color made of three elements 0-255 for RGB

io.imread: reads and image into an array

io.imshow: visualize an image, and the matplotlib to visualize the images