Python

9/6/17

Preliminary Definition

Algorithm: A well ordered sequence of operations that produce a result from some input and halt in a finite amount of time.

Computing: The activity of using or creating algorithmic processes to complete some task

Computer Science: The study of algorithms including their mathematical properties, their linguistic realizations(programs), their physical realizations(computers), and their application.

Program: A formal specification of an algorithm that can be understood by computers and other programmers. (“A human-readable essay on computing”)

Different python program for the same algorithm.

Programming Language: A programming language is used to formally specify algorithms. Programming languages come in many different flavors and for different applications. In this course we will be using Python.

Interpretive language not compiler

How much water is there in an inch of rain?

One inch per acre

How many gallons per acre?

43560 sqft per acre

12 inches per feet

7.48052 gallons per cubic feet

9/11/17

Homework due September 25th

Find all the evn numbers between 10 and 20

[i for i in range (10, 21): if i%2 ==0]

In python: variable can contain data of different types, variables don’t have types

Data objects do have type

A=6 data type:int

A= “hello world” data type: string

Variable = A

A=3

27==27 🡪 true

my\_int = 27

my\_int == “27” 🡪 false

27 == 27.0 🡪 true

int (“27.5”) wont work 🡪 value error

How should you name variables

A name may be of arbitrary length and may contain letters, digits, and underscore

Every name must begin with a letter or underscore

A name must be a keyword (i.e: type)

Don’t use print as a variable

variable name that doesn’t change should we uppercase

ex: THIS\_IS\_A\_MAGIC\_NUMBER = 42

List:

x= [‘a’, ‘b’]

y=x

y[0] = ‘c’

now its [‘c’, ‘b’]

gretting = “hello”

id(greeting) = some id number #spcify the id (“memory location”) of variable

variable 🡪 objects in memory

a 🡪 object ID\_NUMBER, Int 42

b 🡪 str “fourty two”

a,b = b,a

a= str”fourtytwo”

b=42

how to remove an object from memory

c= None🡪 points to no objects in particular

frees up memory space

b=a two variables point to the same object

now is b==a? true

is operator, are they point to same object

a is b 🡪true

B=4

A+B= 7-integer

a/b=0.75 floating point number

a//b =0 cuts off remainder

a= hello

a

output: hello

b

output:2

3.0\*2: float type 6.0

x=5/2=2.5

int(2.5)=2

round up:

import math

Math.floor

Math.ceil(x)

Or int(x+1)

Can use double or single quotes for strings

Type(0.8)= float

Type(x)=str

Len(5) 🡪 type error

greeting.capitalize() 🡪’Hello’

every object has a type and this type never changes

variable do not have a type. They can point to any object

“x”\*3 🡪 xxx

a=5

b=5

a is b🡪 true

python reuses things

a= “abc”\*128

b=”abc”\*128

a is b 🡪 false ## too long string

a = [1,2,3]

b = [1,2,3]

a is b 🡪 false // are these variables attached to the same thing?

a == b 🡪 true // are the objects equal?

A = set([1,2,3])

X = [1,2,3,4]

X(0) = [‘a’,’b’,’c’,’d’]

x🡪[‘a’,’b’,’c’,’d’] 2,3,4]

y=x[0]

Y[0]= ‘d’ 🡪 [d,b,c,d]

x🡪 [‘d’,’b’,’c’,’d’] 2,3,4]

z.insert(0, ‘a’)🡪 adds at the beginning

**statement vs expression**

expression: are evaluated and return a value. The value need to be captured

Statement: can print something but doesn’t return a value

A= 27 🡪 statement

X=(a=27) 🡪 syntax error, right side must be an expression

Lecture 3: control

What makes computing powerful is control

Selection: making a decision about hot to proceed in a program based on the value of some variables.

Repetition: Performing an operation over and over either on different objects or different elements of a list until some condition is met

Another built in data type: bool

Boolean- true or false

Boolean operators: not, and, or

X=5

X==5 🡪 true

Not x 🡪 false

Selection: The if statement

If Boolean expression:

Indented stuff gets executed if true

The indention must be 4 spaces long

Only in spyder is tab 4 spaces

In regular text editor you must do four spaces

Input() is an expression

Player\_move=Input(“please choose your move”)

Random\_int= random.random()

If Boolean expression:

Stuff

Elif:

Stuff

Else:

Other stuff

While Boolean expression:

Indented stuff

The break statement

While True:

If not (something):

continue

If something

Break

9/18/17

The for statement

for x in [“some”, “stuff” -42,True, None]:

Print (x)

For s in “hello 1006”

Print s

Results = []

For x in range (12):

If x %2 !=0

Results.append(x)

Print(results)

Li = [‘a’,’b’,’c’]

“ “.join(li)

a b c

collate conjecture

while n=!1

lecture 4

algorithm and program development

an algorithm should be

* Detailed
* Effective- eventually provide a solution in a reasonable amount of time
* Specific as to its behavior- clearly define outputs and inputs, should be able to predict output from input
* General purpose- algorithm apply to different objects share some properties. Sorting requires objects to be comparable

Style guide for python’

Import sys

Sys.exit(1) # exits program

Import string

String.digits

0123456789

09/20/17

largest\_int = li[0]

largest\_index =0

current\_index=1

while current\_index<= len(li)-1

if largest\_int < li[current index]

largest\_int = li[current\_index]

largest\_index = current\_index

break- gets you out of loop(?)

continue- brings you back to first line

runtime

what happens to run time depending on size of input

\n 🡪 new line

\t 🡪 tab

s[0:9]

s[0:8:2]

s[::-1]

If a is b then a == b, true

Once a variable is assigned to a value it can not be changed 🡪 false

Cant use dashes in variable name

Print is a built in function not a keyword

Ord(‘a’)

97

‘alex’ < ‘alexa’

“ some word or phrase”.title

s.find(y)

s.find(y,2)

s.find(‘a’,s.find(‘a’)+1)

x= “1,-2,-3,-4,-5”.split(‘,’) #splits a line in cvs format into a list

x= “1” , “-2”, “-3” etc

“,”.join(word\_list)

x= “{} is {} years old”

x.format(‘anne’, 28)

“pi is {:.4f}” .format(math.pi)

{:8.4f}// fits in line of 8

{:.2%}.format(.2510) // converts to percent 25.10%

s2 = reversed(s)

list s2

s[::-1]

s2\_lower = s2.lower()

s\_final = s2\_lower.replace(“ ”, “”)

9/27/17

**Introduction to functions**

Useful if you want some of the program to be reusable

* Function similar to math y=f(x)
* Function performs some operation given one or more parameters and return one value
* Functions encapsulates the performance of some particular operation so it can be reused by others

Functions are a way to solve a specific sub-problem. They **abstract** away from the actual solution to that sub problem

Import math 🡪 library

Math.sqrt🡪 function

You do not have to know how the math.sqrt function is implemented, just what it computes

You can share libraries you write so other people can use the functions

This encourage top-down problem solving

* Divide the problem into sub problems
* Solve each sub problem, then combine solutions

Functions make your program easier to read and maintain

A single function is also easier to make secure

As we will see later, functions are also a powerful computational device. They allow **recursion.**

**Defining and invoking Functions**

Def function\_name(parameter\_1, … , parameter\_n):

#docstring describing function

Statements

Return result

A function definition consist of

* A function name(follow variable rules)
* Zero or more separated parameters
* A function body or suit(list of indented statements)
* One or more return statements

Ex:

Def f(celsius\_float)

X = celsius\_float\*1.8+32

Return X

X=f(35) #converts 35 celsius to farenheit ## this x is different from return x

Once function is defined, it can be invoked (or called). A function must be defined *before* it is invoked.

When a function is invoked, arguments are passed into the functions parameters.

A function invocation is an expression whose value is the return value of the function

X=f(25.0) #25 is the argument passed into the Celsius\_float parameter

Functions have data type

A function can only return one **return value,** but this might be a compound object.

Def parse\_name(name\_string)

Parts = name\_string.split(“.”)

Last\_name = parts[0]

First\_name\_parts = parts[1].strip().split(“ “)

First\_name=first\_name\_parts[0]

Middle\_initial = first\_name\_parts[1]

Return [first\_name, middle\_initial, last\_name]

Result = parse\_name(name)

Print(“{} {} {}”.format(\*results)

X = parse\_name(name)

First, middle, last = x

<function\_main\_.f>

def fact(n):

“””

Compute n!

“””

product = 1

for i in range(n, 0, -1)

product\*=i

return product

def fact(n)

if n ==1

return 1

return n\*fact(n-1)

return is similar to break

**Function control Flow**

When a function is called, the main program(or another function) is temporarily suspended

Control is then passed to the function

Once the function finishes it returns to the main program

**Name space of a function**

Variables defined inside a function are not “visible” outside the function.

Some\_int =23

Def foo():

Some\_int = 42 #new variable in foo namespace

Print(some\_int)

Foo()

Print(some\_int)

Counter[0] #considered bad style!

Def foo():

Counter[0] =1

X=5

Def bar():

Print(x)

X = 7

Print(x)

#will give error, when variable in namespace is defined, it overshadow variable in outside namespace, wont print 5 and 7

Passing Arguments Call by object

Li[1,2,3]

li.append(4)

def addstuff(some\_list, new\_int):

somelist.append()

**Main Method**

If importing stuff from another file

Def main():

…

If\_name \_ ==”\_main\_”):

Main()

10/2/17

Files

Compile time error- can catch in advance- as opposed to syntax

Runtime error-user input

Parameter vs argument

Arg- actual value pass to parameter

Parameter acts as local variable

Function call interrupts program until user hits enter, then returns to main program

Namespace- between variable and value they point to

Small local namespace for functions, information doesn’t leak out of them

Chapter 6

A file is a collection of data that resides permanently on disk, it is independent of the program you’re running.

We often need to read and write data from files

Files have a name and location in your file system. Location = path of file

Path starts with “/”

R reading

Writing

Append

Test\_file = open(“test.txt”, “r”)

Textio wrapper – text handle lets us interact with it

Test\_file.readline()/reads line including leading/trailing whitespace, can get rid of using .strip()

Call readline again will go to second line

Test\_file.seek(11) #num character

Two program cant write to same file at same time

Cant read and write at same time

Test\_file.close()

For line in test\_file:

Print(line.strip())

Writing to file

Print(“something”, file = test\_file2)

//console also a file

Import sys

Sys.stdout

Print(“some”, file = sys.stdout)

Test2.write(“tuf”)

Return amount characters

Def check\_vowel(word)

Vowel= “aeiou”

Dic\_file = open (“dic.txt”, “r”)

For line in dic\_file

Word = line.strip()

If check\_vowel(word)

Print(word)

10/4/17

**Exception, Lists, Tuples**

But first:

**#say we have a file with data on contiguous US average temp**

Def read\_temps\_file(filename):

Temp\_file = open(filename, “r”)

#skips header 5 lines

For a in range(5):

Temps\_file.readline()

Data=[]

For line in Temps\_file:

Line = line.strip()

Fields = line.split(“,”) #makes list

Month\_years\_s= field[0]

Temperature\_s=field[1]

Temperature = float(temperature\_s)

Years\_s = month\_year\_s[:4] #first index inclusive second is not

Month\_s = month\_years\_s[4:]

Year = int(Year\_s)

Month = int(Month\_s)

Data\_row=[year,month,temperature]

Data.append[Data\_row]

Return data

Def compute\_average\_temp(data, target\_month):

Temp\_sum = 0.0

Counter=0

For row in data:

Year, month, temp = row[0], row[1], row[2] #or just row works too

If month == target\_month:

Temp\_sum +=temp

Counter+=1

Avg\_temp = Temp\_sum / Counter

Return avg\_temp

Def compute\_median\_temp(data, target\_month):

Temp\_list = get\_temp\_for\_month(data, target\_month):

Temp\_list.sort()

If len(temp\_list) % 2 ==1

Return len[len(temp\_list) //2]

Else:

Element 2 = len(temp\_list) //2

Element 1 = len(temp\_list) //2-1

Return (temp\_list[element 2] + temp\_list [element 1]) / 2.0

Def get\_temp\_for\_month(data, t\_month)

Temp\_list[]

For row in data:

Year, month, temp = row[0], row[1], row[2]

If month == t\_month

Temp\_list.append(temp)

Return temp\_list

Def compute\_average\_annual\_temp(data)

Avg\_per\_year = []

Avg\_this\_year = []

Current\_year = 0

For row in data:

Year, month, temp = row[0], row[1], row[2]

If current\_year != Year:

If avg\_this\_year: # same as avg\_this\_year != []

Avg\_tmp = sum(avg\_this\_year)/ len(avg\_this\_year)

Avg\_per\_year.append(avg\_temp)

Avg\_this\_year = []

Current year = year

Avg\_this\_year.append(temp)

Return avg\_per\_year

Temp\_data = Read\_temp\_file(“temperature.csv”)

**Exception**

we have already seen some

while = 27 is a syntax error

syntax errors are compile time error, program will not even run

print(hshad) is a name error

27 + “hello” is a type error

int(“hello”) is a value error, string should be formatted as an integer

same thing with int(“27.5”)

a careful programmer should be able to avoid these problems in their code

unfortunately when a program reads input from the user or from a file, exceptions may be out of our control

f = open(“something.txt”, “r”) is a file not found error

if you don’t have permission to write to a file you get a permission error

f.close()

f.readline is a value error since f is closed

rather than letting the program crash cause the user input the wrong thing/ bad file name

try-except construct allow you to capture and handle runtime errors

try:

f = open(“somefile.txt”, “r”)

print(“opened correctly”)

except FileNotFoundError

print(“sorry file does not exist”)

can have multiple exceptions for the same try

okay = false

while not okay:

try:

x = input(“please type an integer number”)

x\_int = int(x)

f = open(“some file. Txt”, “r”)

okay = true

except

except

10/9/17

Midterm practice – do before monaday and we’ll discuss on Monday

Hw3 – will have to take data from an outside file

(Review) Exceptions:

IO, index out of bounds (index error), file not found

Try/except blocks

First exception will be handled by the “except” block

-- don’t think of exceptions as python objects

**Lecture 9 – dictionaries and sets**

Dictionaries: python’s most powerful data type (i.e. mapping names and phone numbers)

Python à looking at elements with a specific index takes minimal time

Mention of numpy

Nested lists à

List\_of\_lists[1][2] ß second list, third second element of that list (you can nest even more.. so if you add another one that’ll be the letter index of the element you picked out, etc.

You can turn a list of numbers in a range into a list… anything in a for loop you can turn into a list?

I.e

List(range(1,10)) = [1,2,3,4,…,10]

What happens if this is compared

[1,2,3] < [1,’a’,3]

à error : doesn’t know how to compare them

lists are mutable

you can use sort(13) when 13 = [1,3,4,6,2] to sort mutable types

you can also use 13.sort()

this gives you 2:  len(lists)

if lists = [[1,2,3],[5,6,7]] (two elements that happen to be lists)

you can count how often an element in a list occurs:

i.e – l.count(0) à tells you how many times 0 occurs

you can override an element in a list

i.e.

x = [[1,2,3],[4,5,6]]

x[1] = “hello”

now

x = [[1,2,3],’hello’]

*lists can be changed, strings cant* – strings are immutable

l[:] = [1,2,3] replaces all elements in a list by 1,2,3.

Why can’t we just do l = [1,2,3] ? it’s cause this creates a new list called l, pointing at a new object, whereas the first way doesn’t create a new object.

b\_list = a\_list[:] #creates a COPY of a\_list

**Tuples**

Append function doesn’t work with tuple

To create a tuple, you use parenthesis instead of brackers

i.e x = (1,2,3)

or alternatively x = 1,2,3

this is not a tuple… x = (1), it’s an int!

if you add a comma inside, you create a tuple that’s only one element.

i.e.

x = (1,)

create an empty tuple:

x = tuple()

you can convert a list to a tuple and a tuple to a list, and vice versa

x = (1,2,3)

list(x) ß turns into list

etc.. (helpful so you can modify the list) à this creates a new object that’s a list and has the variable point at it…

Tuple

Mutable, as opposed to strings

10/11/17

Dictionary and sets

Dictionary, like list but not in sequence, “bag” of items, no structure

Sometimes called maps or associative arrays

Backwards phone book – name not unique but phone number is

You can write dictionary literal as a list of key:value pairs surrounded by {}

Contacts = {“bill”:”353-1234”, “rich”:”269-1234”, “jane”: “352-1234”}

Contacts[‘bill’] Output: 353-1234

Some\_dictionary = {} or = dict()

Dictionaries are mutable like lit unlike strings and tuples, dict can be modified

Can nest dictionaries

A = (1,2,3)

B = (1,2,3)

A is b = false

A == B true

Test\_dict = {a:42}

Test\_dict[b] = 42

Hash Maps

Hashing is a technique to create an integer identifier(the hash value) for an object that is as unique as possible. However, multiple objects can in theory have the same has value

Len(my\_dict) number of key:value pars

My\_dict = {“a” :1, “answer”: 42, 3:[x,y]}

“A” in my\_dict- true

Only check keys not values

How to iterate through values?

Iterates through key then index to get value

For k in my dict

Print my\_dict[k]

My\_dict.keys() 🡪 3, answer,a

My\_dict.values()

My\_dict.items()

My\_dict[“greet”] = hello 🡪 added to my\_dict

Fruits\_to\_color= {banana:yellow, cherry:red, blueberries:blue, apple:red, lemon:yellow}

How to make color\_to\_fruit

Color\_to fruit={}

For fruit, color in fruit\_to\_color.items():

Color\_to\_fruit[color] = fruit

Print(color\_to\_fruit)🡪{yellow:lemon, red:apple, blue:blueberries] –banana and cherry gets overwritten

Instead, put in list

Color\_to fruit={}

For fruit, color in fruit\_to\_color.items():

Fruit\_list = []

Color\_to\_fruit[color] = fruit

Try:

Fruit\_list = Color\_to\_fruit[color]

Except: keyerror

Fruitlist = []

Color\_to\_fruit[color] = []

Fruit\_list.append(fruit)

Or

If color not in color\_to\_fruit:

Color to fruit[color] = []

UPLOAD TO COURSEWORKS

Def preprocess\_line(line):

Line = line.strip()

For character in line:

If character.isalpha() or character in [ “ “, “\t”]:

Replace\_line = replace\_line + character

Replace\_line = replace\_line.lower()

Return replace\_line.split

Def get\_word\_count(filename)

Input\_file = open[“filename”, “r”]

For line in input\_file:

Processed\_line = preprocessed\_line(line)

For word in processed\_line:

If word not in counter\_dict:

Counter\_dict[word] =0

counter\_dict[word] = counter\_dict[word] +1

return counter\_dict

word\_count = get\_word\_count(‘constitution.txt”)

word\_counter\_list = []

for word,count in ord\_count.items

10/23/17

1. [1,2,3] is [1,2,3] False
2. True, extra element
3. True

Sets: does not contain any duplicate elements, only one copy allowed

A\_set= {1,2,3,4,5}

Nullset = set()

B\_set = {1,1,2,2,1}

Sets are independent of order

A = {1,2,3}

B={2,3,4}

a.intersection(b) 🡪 {2,3}

a&b 🡪 {2,3}

a.union(b)🡪 {1,2,3,4}

a.difference(b)- {1}

a-b- {1}

b-a – {4}

issubset

issuperset

10/25/17

Lecture 11: Object Oriented Programming

Variables don’t have data type objects do.

Methods 🡪 l.append(5)

Built in Functions🡪 len(l)

Dir(l) 🡪 list of all the methods associated with object

Classes is a data type defined by programmer

* A name
* A set of attributes
  + Data field
  + Method- definition of operations

Example: Library System

Class name use Camel Notation

Class LibraryBook(object)

“””

Describe data in class or role in program

“””

pass

type(LibraryBook 🡪 type

Classes are blueprints for instances

Isinstance(my\_book, LibraryBook)

My\_book.\_\_class is libraryBook 🡪 Yes

Test2 = test

My\_book.title = “Harry Potter” // can define whenever you want

Def set\_title(self, new\_title) // self always pointing to object you called it on, so don’t need to write self parameter when calling it

Self.title = new\_title

Special Methods

Def \_\_init\_\_(self,title,author,pub\_year, call\_no, checked\_out= False)

Self.title = title

Self.author = author

Self.year = year

Self.call\_number = call\_no

Self.checked\_out = checked\_out

Def \_\_str\_\_(self):

Return “{} {} {}: {}”.format()

Print(new\_book) does. \_\_str\_\_\_

\_\_\_repr\_\_\_ 🡪 can just enter new\_book instead of print(new\_book)

11/1/17

Even more on OOP

Three different aspects of OOP

Encapsulation-hiding specific implementation of object

Polymorphism- use same algorithm with same object with different functionalities

Inheritance- (scroll down, talks about it later)

Built in methods- constructor(\_\_init\_\_)

Self- allows u to access instance of object, similar to “this” from java but in java we declare the varibles for the class, in python u don’t, so you have to keep writing “self” to know where to look for the object

Class Rational #lecture code

Writes constructor, writes code to compute GCD outside class, now can reduce object parameters by GCD, writes reduce function to reduce fraction to simplest form, writes function for LCM, uses GCD, can now write function to add fractions \*\*has some trouble printing new fraction\*\*

Isinstant checks if object is instance of class

Writes rock,paper,scissor game by creating classes for each #lecture code

Repr vs string – repr gices what the func outputs and print will print what user wants from it

Inheritance- recall relationship between a class instance and class

Class Clownfish

Nemo = Clownfish()

Isinstance(nemo, Clownfish) 🡪 True

“Is a” relationship

you can have more than one parent class per object (unlike java), must have atleast one parent

every clownfish is a fish and is also an animal – two parent classes

the topmost parent class is object

everything has a “is a” relationship to object

children classes inherit methods in parent class(es) up to object

can override this method if you make a method with the same name

when you do \_\_.str\_\_() you are overriding the method in the object class

11/8/17

SRI Model

%matplotlib inline for spyder

y,x axis

display method

take map and turn it into image of correct format

list of list

every list is a row and you have 150 of those

(0,0) (0,1)

[ [ (R,G,B) tuple of three values ] row zero

[ ]

[ ] ] row n

column 0 column m

for do multiplication

two nested for loops, make all black, if state is I then red

process method

if cell in dictionary in state s, can become infective

if random number is less than variatity then become infected if greater than nothing happens

later also take care about recovery and dying

timestep method

process interate through all the cells get adjacent and doing process

mortality

m = 3

sd = 1

random

list comprehension

[i for i in range(20) if i%2 == 1]

[(I, i\*\*2, i\*\*3) for I in range(10) if i%2 ==0]

x\*\*2 if x%2 == 0 else “odd” for I in range (20)

string.

11/13/17

Recursion

How rabbits populate?

Fibonacci- 0🡪 1

1🡪1+0=1

2 🡪1+1=2 second gen

3🡪1+2=3

4 🡪2+3=5 third gen

generator expression

while true

yield fib1

11/20/17

def f(x)

return x\*\*2 + 2\*x + 5

How to plot this?

Provide range for x coord

X\_val = list(range(-5,5))

Y\_val = [f(x) for x in x\_values]

Plt.xlabel(“x”)

Plt.ylabel(“y”)

Plt.plot(x\_val, y\_val)

U=np.array([1,2,3])

V =np.array([4,5,6])

Np.dot(u,v)

32

print u 🡪 no commas

11/27/17

square % 2 makes it 0 or 1

square%2 == 0 – makes true or false array

np.vstack 🡪 stacks vertically, change v to h for horizontal

np.transpose()

machine learning

Training-testing- development data

Np.genfrotxt(file, delimiter = “,”, skip\_header = 1)

Np.random.shuffle(data)

Test = data[:15]

Train = data[15 :]

Len train = 135

Nearest centroid classifier

Idea is to create a input example for each class

Strip off labels of training data

Setosa [0][1:] 🡪 without label

Def makecentroid

Sum each column and divde by number = average

Average flower looks

Then make centroid of each type of flower

Nearest centroid model

**Lecture 20: Graphical User Interfaces with tkInter**

Back end 🡪 front end

Event handler

online

**Lecture 21**

**FINAL LECTURE- FINAL REVIEW**

Selection vs repetition

* If else
* Booleans, empty list empty string 0 empty dictionaries – false

Ternary operator x =(a if condtions else b)

Statements: do not evaluate to anything, x=y, break, functions, and expression: evaluates to some value, arithmetic operations, comparison,

Writing main method🡪 only runs if calling that file directly

Objects and data type

objects- attributes, methods, data fields

namespace – where the python interpreter stores variables and their objects, maps them

shadowing, hiding outer a with inner a

dictionaries can not have mutable data types as keys because hashing

string operations

replace returns a new string, strings are immutable

find