**CSC 401**

**Lecture Notes**

**Calculate a median:**

With paper and pencil:

Now with a calculator:

Now do it in Excel:

Now in Python:

>>> values = [12,40,8,52,16,4]

values.sort()

length = len(values)

oddeven = length % 2

if oddeven = = 1:

median = values[length // 2]

else:

median = (values[length // 2 – 1] + values[length // 2]) / 2

print(median)

PROBLEM SOLVING!!!

**Arithmetic Expressions**

Numeric values can be integer or real (floating point). (Some others, later)

3 + 7

12 / 5

12 // 7 integer division

12 % 7 integer remainder

3 \* 2 + 1 precedence??

(3 – 1) \* (4 + 1)

2 \*\* 3

OK, you’ve done a calculation. Now where do you hold the result?

A calculator has a few memory locations so you can hold a value temporarily.

A computer has millions (billions) of memory locations to hold values temporarily.

What if you have a computer with 4GB? What does that mean?

4GB = 4 gigabytes = 4 \* 1,073,741,824 bytes (a byte = 8 bits)

Most integers and reals need 4 bytes to hold a value (many variations here).

(Draw a large square representing memory. Put one integer box in the middle. What is the address of that square? Pretty big number for an address. Let’s use a variable name instead. Much easier for us humans. Computer maps variable name to actual address so we don’t have to.)

**Variable Names and Assignments**

<variable name > = <expression>

A-Z, a-z, \_ , 0-9 Must start with letter; case sensitive!

>>> x = 4 Use meaningful names. Draw x in box pointing to 4 in box.

>>> x

>>> y = x + 3

>>> y

**Strings**

>>> a = ‘hello’ Note: can use “ or ‘ to delimit

b = ‘world’

a

b

c = a + b concatenation

c

c = a + ‘ ‘ + b

c

c = ‘hello’ \* 3

c

**Python Files**

helloworld.py

Click on “File” and then “New Window”

a = ‘hello’

b = ‘world’ Make mistakes and try things!!

c = a + ‘ ‘ + b

print(c)

**Interactive Input**

name = input(“Enter your last name: “) Input ALWAYS returns a string!

x = eval(input(“Enter a value: “)) use eval to convert to number or list or …

Strings are not the same as numbers!

Example

Input radius, compute and print area and circumference. Assume radius is positive, non-zero

r = eval(input(“Enter a radius: “) vs. simply saying r = 4.5

area = 3.14159 \* (r \*\* 2)

circum = 2 \* 3.14159 \* r

print(“Area = “, area)

print(“Circumference = “, circum)

Hand out syllabus

Introduce textbook

Show lecture1 powerpoint slides

**Boolean Expressions**

>>> 3 < 2

3 > 2

x == 4 not x = 4

<=

>=

!=

2 < 3 and 4 > 5 Show truth tables for *and*, *or* (inclusive), *xor*

2 < 3 or 8 < 5

**Comments**

Start with a #

**Python Keywords**

and as assert break class continue del elif else except

exec finally from global if import in is not or

pass print raise return while with yield

**Decision Statements**

if <condition> :

<body> <body> must be indented!

if value <= 0:

print(“This value is not positive.”)

print(“Enter another value.”)

print(“We are not here”)

Example

User prompted to enter a value. If 1, change name, if 2, change address, if 3, change phone number.

value = eval(input(“Enter a value 1, 2, or 3: “))

if value == 1:

print(“So you want to change your name.”)

new\_name = input(“Please enter your new name: “)

if value == 2:

print(“So you want to change your address.”)

:

:

If value != 1 and value != 2 and value != 3:

print(“Invalid entry”)

What if someone enters a 6, or a -12, or 1002?

Example

Prompt a user to enter Yes or Y or yes.

answer = input(“Enter Yes to continue: “)

if answer = “Yes” or answer = “Y” or answer = “yes”:

or:

if answer in (‘Yes’,’yes’,’Y’,’y’):

**If Else**

if temp >= 86: 2-way if

print(“It is warm outside.”)

print(“Drink lots of fluids.”)

else:

print(“Not too hot.”)

print(“Drink fluids anyway.”)

print(“This is always printed.”)

grade = eval(input(“Enter a grade: “))

if grade >= 90:

print(‘A’)

else:

if grade >= 80:

print(‘B’) etc.

or:

if grade >= 90:

print(‘A’)

if grade < 90 and grade >= 80:

print(‘B’)

if grade < 80 and grade >= 70:

print(‘C’) etc.

**Types**

Python does not associate a type with an identifier (variable name). Associates a type with the object itself.

>>> a = 3

type(a)

a = 3.0

type(a)

a = ‘string’

type(a)

a = 1 == 1

type(a)

**Indexed Strings**

>>> str1 = ‘hello’

str1[1]

str1[0]

str1[4]

str1[-1]

Cannot say: str1[0] = ‘j’ Strings are immutable!

Substrings

str1[0:2]

str1[:2]

len(str1)

if ‘ll’ in str1:

print(“found it”)

**Lists**

Ordered collection of objects.

list1 = [1, 8, 7, 3, 6, 28, 4]

len(list1)

min(list1)

max(list1)

sum(list1)

list2 = [‘apple’, ‘banana’, ‘orange’]

list2[0] yields ‘apple’ Note: result is a string, not a list

Lists are mutable!!

list2[2] = ‘grape’

if ‘orange’ in list2:

list1 + list2 add 2 lists

2 \* list2

print(list2[-1]) prints last one in list and is not a list

print(list2[-1:]) prints last one in list and maintains a list!

[list2[-1]] also maintains a list

**List Methods**

list1.append(8) don’t say: list1 = list1.append(8)

list1.reverse() likewise

list1.sort() don’t say list1 = list1.sort()

value = list1.count(item) how many times does item occur in list

value = list1.index(item) index of first occurrence of item

list1.insert(index, item) insert ‘item’ into list at position ‘index’

list1.pop() removes last item (on right) from list

list1.remove(item) removes first occurrence of item

Examples

Finding a median (like we did on first day)

Practice problem 3.4: Assign a list of names. Prompt user for login name. If in list, print OK

users = [‘Joe’, ‘Sue’, ‘Al’, ‘Steve’, ‘Rashad’]

id = input(“Enter Login name: “)

if id in users:

print(“You are in!”)

else:

print(“User name unknown”)

**Iteration Structures or Loops**

Payroll (classic example)

Definite loops

Indefinite loops

Infinite loops

For Range Loop

for i in range(10):

print (i)

for i in range(2):

print(i)

for i in range(3,13):

print(i) prints 3 4 5 6 7 8 9 10 11 12

for i in range(0, 10, 2):

print(i) prints 0 2 4 6 8

Definite loop – For loop

list1 = [‘cat’, ‘dog’, ‘bird’]

for animal in list1:

print(animal)

list2 = [67, 34, 98, 63]

for i in list2:

print(i)

Examples

Practice problem 3.5: Prompt user to enter list of words. Print all 4-letter words.

wordList = eval(input(“Enter word list: “))

for word in wordList:

if len(word) == 4:

print(word)

Problem 3.22 page 89: Prompt user for list of words. Print each word in list that is not the word ‘secret’.

words = eval(input(“Enter a list of words: “))

for word in words:

if word != “secret”:

print(word)

Problem 3.23 page 89: Prompt user for list of names. Print each name that starts between A and M (uppercase).

words = eval(input(“Enter list of names: “))

for word in words:

if word[0] >= ‘A’ and word[0] <= ‘M’:

print(word)

Example

Prompt user hours and hourly pay. Calculate gross pay. If gross pay > 100.00, taxes = 30%, else taxes = 20%.

hours = eval(input(“Enter hours worked: “))

hourly\_pay = eval(input(“Enter hourly pay: “))

gross\_pay = hours \* hourly\_pay

if gross\_pay > 100.00:

taxes = gross\_pay \* 0.30

:

**Functions**

Built-in functions:

values = [4, 8, 12, 2, 3]

smallest = min(values)

Write your own function!

Suppose you want a function that when you pass it a value, it adds 5 to that value and returns that value.

def fplus5(x): Place functions defs first in module

return x + 5 x is the parameter – the receiving variable

:

:

y = fplus5(10) 10 is the argument – it is being passed in

Can also do this in shell:

>>> def avg(x,y):

>>> print((x + y)/2)

>>> avg(2,4)

Example

Write a function which accepts a radius as a parameter and calculates and returns 2 \* pi \* r.

def perimeter(radius):

return 2 \* 3.14159 \* radius

:

:

r = eval(input(“Enter a radius: “))

value = perimeter(r)

print(value)

Import math module and use built-in operations

import math

def perimeter(radius):

return 2 \* math.pi \* radius

**Mutable and Immutable**

a = 3

b = 3.0

c = ‘hello’

d = [2,3,5,8,11]

d

c

b

a

[2,3,5,8,11]

3.0

‘hello’

3

What if we now say: a = 6

A new box with 6 will be created and the current box with 3 will be left stranded (garbage collection).

But lists are mutable!

a = [3, 4, 5]

b

a

b = a

[3, 4, 5]

b[1] = 8 So be careful!!

b

a

[3, 8, 5]

**Immutable Parameter Passing**

def stuff(x): Output: Note: x passed in but only value

print(x) 2 returned is in the name of the

x = 5 function

print(x) 5

return x

x = 2

value = stuff(x)

print(value) 5

print(x) 2

**Mutable Parameter Passing**

def stuff2(list1): Output:

list1[0] = 2 [1,2,3]

return list1 [2,2,3]

list2 = [1,2,3] [2,2,3]

print(list2)

print(stuff2(list2))

print(list2)

Example

Write a function that swaps the 2 elements in a list.

def swap(list1):

temp = list1[0]

list1[0] = list1[1]

list1[1] = temp

:

:

list1 = [42, 12]

print(list1)

swap(list1)

print(list1)

Example

Write function *negatives* that takes a list of integers passed and prints, one per line, negative values.

def negatives(negValues):

for value in negValues:

if value < 0:

print(value)

**More on Strings**

‘I am “sick”’

“I’m sick”

‘I\’m “sick”’ are all OK to use (\ is the escape character)

Over multiple lines:

poem = ‘’’This is a very long sentence that can span multiple lines.’’’

**Methods for Strings**

message = ‘’’This message is top secret and should not be divulged to anyone without

top secret clearance.’’’

print(message.find(‘top secret’)) yields 16

print(message.count(‘top secret’)) yields 2

message = message.replace(‘top’,’no’) replaces entire string message

message = message.capitalize() capitalize first character only

message = message.upper() change all chars to upper case

message = message.lower() change all chars to lower case

message = message.strip() leading and trailing blanks removed

lst = message.split() breaks sentence down giving a list of words

lst = ‘this is the text’.split() yields [‘this’, ‘is’, ‘the’, ‘text’]

table = str.maketrans(‘abcdef’,’uvwxyz’)

str1 = ‘fad’.translate(table) yields ‘zux’

str1 = ‘desktop’.translate(table) yields ‘xysktop’

Examples

Write a function that accepts one parameter – a string – and returns the number of words in the string.

def numwords(str1):

list1 = str1.split()

return len(list1)

Write a function that accepts one parameter – a string – and prints the first letter of each word.

def firstletter(str1):

list1 = str1.split()

for word in list1:

print(word[0])

Write a function that accepts one parameter – a string – and returns a list of two values: the number of words that start with A-M, and the number of words that start with N-Z. Case shouldn’t matter.

def breakup(str1):

str1 = str1.lower()

list1 = str1.split()

lessthan = 0

morethan = 0

for word in list1:

if word[0] <= ‘m’:

lessthan += 1

else:

morethan += 1

return [lessthan,morethan]

Modify the above code to return two lists: one with the words a-m and the second with the words n-z.

lesslist = []

morelist = []

for word in list1:

if word[0] <= ‘m’:

lesslist.append(word)

else:

morelist.append(word)

return [lesslist,morelist]

Write a function that accepts one parameter – a string – and removes all the punctuation marks. Replace the punctuation marks with null strings.

def removepunct(str1):

table = str.maketrans(‘,.;’,’%%%’)

str1 = str1.translate(table)

str1 = str1.replace(‘%’,’’)

return str1

**Files**

File organization can be messy. Luckily, Windows has a hierarchical file system, as well as Mac OS and Linux. Files can include disk drives, flash drives, solid state storage, tape backup, DVD/CD/…

Three basic operations for all files:

Open

Read/Write/Append

Close

In Python:

infile = open(‘example.txt’, ‘r’) Mode

‘r’ read (default)

‘example.txt’ is a relative filename ‘w’ write

Looks in current directory. ‘a’ append

Want it to look elsewhere? Use full name: ‘r+’ read and write

‘t’ text mode

infile = open(‘/Users/Smith/example.txt’,’r’) ‘b’ binary mode

Text mode (default): Treated as an encoded file

Binary mode: Treated as a sequence of bytes and not encoded

Write mode: If file already exists, its contents are wiped. If doesn’t exist, file created.

Open returns an object of Input Stream type or Output Stream type

**File Methods**

infile.read(n) reads n chars; returns a string

infile.read() reads until eof; returns a string

infile.readline() reads one line (until “ \n new line” or eof); returns string

infile.readlines() returns a list of lines (until eof)

Examples

1 The 3 lines in this file end with the new line character.

2

3 There is a blank line above this line.

infile = open(‘example.txt’,’r’)

infile.read(1) yields ‘T’

infile.read(5) yields ‘he 3 ‘

infile.readline() reads the rest of the first line

infile.read() reads the rest of the file

infile.close() Don’t forget the close!

Examples

Write a function with one parameter – a filename. Open that file and return the number of chars in the file.

def numChars(filename):

infile = open(filename,’r’)

content = infile.read()

infile.close()

return len(content)

Write a function that accepts two params – a filename and a string. Return the number of occurrences of string in filename.

def stringCount(filename,str1):

infile = open(filename,’r’)

content = infile.read()

infile.close()

return content.count(str1)

Write a function that splits a string of words into a list. Print the word list and return the number of words in the list.

def splitit(filename):

infile = open(filename,’r’)

content = infile.read()

wordList = content.split()

print(wordList)

return len(wordList)

Read a file and get a list of lines:

infile = open(filename,’r’)

lineList = infile.readlines()

Read one line at a time:

infile = open(…

for line in infile:

print(line, end = ‘ ‘)

What if we want to read numbers from a file?

One number per line (50):

infile = open(“sample.txt”)

str = infile.readline() yields str = ‘50\n’

value = int(str) yields value = 50

Multiple numbers per line (30, 40, 50):

infile = open(…)

lst = []

str = infile.readline() yields ’30 40 50\n’

values = str.split() yields [‘30’, ‘40’, ‘50\n’]

for value in values:

lst.append(int(value)) yields [30, 40, 50]

**Writing to a File**

Must open for writing:

outfile = open(‘test.txt’, ‘w’) No test.txt? Will be created.

Already exists? Will be wiped clean.

outfile.write(‘Hello’)

outfile.write(‘add this’)

outfile.write(‘ and this too. \n’) \n = end of line marker

outfile.write(‘Next line.’)

Non-string values must first be converted to string:

outfile.write(‘Add this number: ‘ + str(123))

outfile.close() Don’t forget to close!! Buffer is written.

Example

Case study page 121: Create a log file which records every time a file is opened.

def openLog(filename, mode):

infile = open(filename, mode)

outfile = open(‘log.txt’, ‘a’)

outfile.write(‘File {} opened. \n’.format(filename))

outfile.close()

Note that this case study makes extensive use of *time* module (pages 123-125).

**Formatted Output**

print(‘Al’, ‘Joe’, ‘Sue’, sep = ‘;’) yields: Al ; Joe ; Sue

print(‘One’, ‘Two’, ‘Three’, sep = ‘, ‘) yields: One, Two, Three

print(1,2,3, sep = ‘\n’) yields: 1

2

3

for name in [‘Joe’, ‘Sue’, ‘Al’]: prints one name per line

print(name) since \n is default

for name in [‘Joe’, ‘Sue’, ‘Al’]: prints: Joe! Sue! Al!

print(name, end = ‘!’)

hour = 4

minute = 26

second = 18

print(‘ {} : {} : {}’.format(hour, minute, second)) yields: 4: 26 : 18

weekday = ‘Tuesday’ etc.

print(‘{} , {} {} , {} at {} : {} : {}’.format(weekday, month, day, year, hour, minute, second))

yields: Wednesday, May 9, 2013 at 11:34:22

print(‘{0 : 3} , {1 : 5}’.format(12,345)) 0 and 1 are position identifiers

3 and 5 are column widths

Or you can leave off the position identifiers:

print(‘{ : 10} { :10}’.format(‘Bill’,’Gates’))

{ 0 : 8.4 } 0 is the position identifier; 8 is the column width; 4 digits total

(both before and after the decimal point!)

What if you want to print something like money with 2 decimal places rounded?

n = 3.45577

print(‘{:.2f}’.format(n))

n = 13

print(‘{ :b}’.format(n)) prints in binary: 1101

print(‘{ :c}’.format(n)) prints Unicode character

{ :d} decimal

{ :o} octal

{ :x} hex in lower case

{ :X} hex in upper case

Examples

Given: first = ‘Marlena’, last = ‘Sigel’, middle = ‘Mae’, create one formatted output statement for each of the following:

a. Sigel, Marlena Mae

b. Sigel, Marlena M.

c. Marlena M. Sigel

d. M. M. Sigel

**Errors and Exceptions**

Syntax errors: (3 + 4]

if x == 5

print ‘hello’

list = [4;5;6]

for i in range(10):

print(i)

Execution errors: 4/0 Division by zero (ZeroDivisionError)

list1 = [14,15,16]

list1[4] Index out of range (IndexError)

x + 5 What is x?? (NameError)

‘2’ \* ‘3’ Can’t multiply by non-int type ‘str’ (TypeError)

int(‘4.5’) ValueError

Exceptions KeyboardInterrupt ctrl-c

OverflowError only with float, not int!

ZeroDivisionError

IOError

IndexError

NameError

TypeError

ValueError

Exception Handlers will be covered in Chapter 7.

**Three-Way Decision**

if temp > 86: Show corresponding flowchart.

print(“It is hot!”)

elif temp > 32:

print(“It is cool”)

else:

print(“It is freezing!”)

if temp > 32: The order matters!!!

print(“Cool”)

elif temp > 86:

print(“Hot”)

else:

print(“Freezing”)

Example

Prompt user for height and weight. Calculate BMI = (weight \* 703) / height2. Then print: If BMI < 18.5, underweight. BMI >= 18.5 and <= 25,normal. BMI >= 25, overweight.

height = eval(input(“Please enter a height (in inches): “))

weight = eval(input(“Please enter a weight: “))

bmi = (weight \* 703) / height \*\* 2

if bmi < 18.5:

print(“Underweight”)

elif bmi <= 25:

print(“Normal”)

else:

print(“Overweight”)

Example

Prompt user for grade. If grade >= 90, print A. etc.

**While Loops – Indefinite Loop**

Probably the most popular loop.

while <condition>:

body

Examples

Read values, add to total, keep looping is more data = ‘Y’

sum = 0

moredata = input(“Do you have data?”) Initialization

while moredata == ‘Y’: Test

score = eval(input(“Enter next score: “))

sum = sum + score

moredata = input(“Continue – Y or N? “) Update

Better yet:

while moredata in [‘Y’,’y’,’Yes’,’yes’]:

Stop on sentinel value, such as -999.

value = eval(input(“Enter a value, -999 to stop: “) Initialization

while value != -999: Test

:

:

value = eval(input(“Enter a value, -999 to stop: “) Update

Example – Write a function that prompts a user to enter a list of cities. Stop when city = null string ‘’. Return the list when done.

def cities():

lst = []

city = input(“Enter a city (null string to stop)): “)

while city != ‘’:

lst.append(city)

city = input(“Enter a city (null string to stop)): “)

return lst

Let’s revisit *for* loops

n = 5

for i in range(n):

print(i, end = ‘ ‘) yields 0 1 2 3 4

n = 5 Nested loops

for j in range(n):

for i in range(n):

print(i, end = ‘ ‘) yields 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4

Practice Problem 5.8 Pg 145 Bubble Sort

def bubbleSort(values): pass a list of values

n = len(values)

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

for j in range (0,n-1):

if values[j] > values[j + 1]:

temp = values[j]

values[j] = values[j + 1]

values[j + 1] = temp

**Two-Dimensional Lists (Matrices)**

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 7 | 2 | 5 |
| 5 | 1 | 9 | 2 |
| 8 | 3 | 6 | 6 |

[row#][col#]

t = [[4, 7, 2, 5], [5, 1, 9, 2][8, 3, 6, 6]]

t[0] yields [4, 7, 5, 2]

t[1] yields [5, 1, 9, 2]

t[0][0] yields 4

How do we reference the 9?

Example

Add two matrices s and t, both the same size.

nrows = len(s)

ncols = len(s[0])

for i in range (nrows):

for j in range (ncols):

sumit[i][j] = s[i][j] + t[i][j] But wait! sumit is not initialized!!

Initialize a two-dim list:

nrows = 3 or however many rows and cols there are

ncols = 4

sumit = [[0 for j in range(ncols)] for i in range(nrows)]

Example

5.35 from page 165 Write function pixels() that takes a 2-dim list of integers and returns the number of positive pixels (not dark).

def pixels(p):

nrows = len(p)

ncols = len(p[0])

counter = 0

for i in range (nrows):

for j in range(ncols):

if p[i][j] > 0:

counter += 1

return counter

**Dictionaries**

Wouldn’t it be nice if we had a list of employee info and to reference an employee’s data, you indexed via employee’s SSN? (Like a database works)

print(employee[‘987654321’]) yields [‘Jones’, ‘Stanley’]

That would be one, big list!

So use the built-in container called dictionary.

employee = {‘864-20-9753’ : [‘Anna’, ‘Karenina’],

‘987-65-4321’ : [‘Jones’, ‘Stanley’],

‘100-01-0010’ : [‘Hans’, ‘Castorp’] }

print(employee[‘987-65-4321’]) yields [‘Jones’,’Stanley’]

General form: {<key 1> : <value 1> , <key 2> : <value 2> , ….

The keys are immutable, so they cannot be lists! The key can be one thing.

The values can be essential any one thing: lists, strings, numbers, ...

days = {‘Mo’ : ‘Monday’ , ‘Tu’ : ‘Tuesday , ‘We’ : ‘Wednesday’, ‘Th’ : ‘Thursday’}

print(days[‘We’]) prints ‘Wednesday’

print(days[2]) error!!

Dictionaries are mutable:

days[‘Fr’] = ‘Fridayyyy’

days[‘Fr’] = ‘Friday’

print(days) prints {‘Fr’:’Friday’,’Mo’:’Monday’,’Tu’:’Tuesday’,’We’,’Wednesday’,

‘Th’:’Thursday’}

Order does not matter and can change.

print(len(days)) yields 5

if ‘Fr’ in days: yields True

if ‘Su’ in days: yields False

No slices! Since there is no order.

Example

Write a function reversePhone which prompts for a phone number and prints the name.

def reversePhone():

rphonebook = {‘(123)456-7890’: ‘Sue Smith’, etc

pnumber = input(‘enter a phone number in the format (xxx)xxx-xxxx: ‘)

if pnumber in rphonebook:

print(rphonebook[pnumber])

else:

print(‘Sorry, no such number’)

**Dictionary Methods**

pop removes a specific key:value pair

days.pop(‘Tu’)

print(days) no Tuesday will print

update merges 2 dictionaries, with no duplicates

favorites = {‘Fr’:’Friday’, ‘Sa’:’Saturday’}

days.update(favorites)

print(days) prints Friday, Monday, Wed, thurs, Fri, Sat

keys returns just the keys

someKeys = days.keys() yields dict\_keys([‘Fr’, ‘Mo’, ‘We’, ‘Th’, ‘Fr’, ‘Sa’])

Note: dict\_keys is a new type

values returns the values

for value in days.values():

print(value, end = ‘, ‘) prints Friday, Monday, Wednesday, etc

items can be used to iterate over the (key:value) pairs

for item in days.items():

print(item, end = ‘; ‘) prints (‘Fr’,’Friday’); (‘Mo’,’Monday’); etc

Note: these are shown as tuples

**Dictionary as a Collection of Counters**

Suppose we have a list of names. Determine the frequency of each name in the list.

def frequency(names):

counters = {} # make a null dictionary

for name in names: # look at each word in list

if name in counters: # if word exists, inc counter

counters[name] += 1

else: #if word does not exist, create counter

counters[name] = 1

return counters

names = [‘Bill’, ‘Sue’, ‘Joe’, ‘Sue’, ‘Debbie’, ‘Bill’]

print(frequency(names))

Example

Take a string and count the occurrences of each letter.

str1 = “this is a test to see if this works”

letters = {}

strlen = len(str1)

for i in range(strlen):

if str1[i] in letters:

letters[str1[i]] += 1

else:

letters[str1[i]] = 1

for letter in letters:

print(“The letter “, letter, “ occurs “, letters[letter], “ times.”)

Example

Pass a file name to a function. Read the file and count the occurrences of each word in the file.

Example

Pass a three-letter abbreviation for the name of a month to a function and the function returns the full month name.

First try – no dictionaries:

def month(abbrev):

if abbrev == ‘Jan’:

return ‘January’

if abbrev == ‘Feb’:

return ‘February’

etc.

Second try – with a dictionary

def month(abbrev):

months = {‘Jan’: ‘January’, ‘Feb’:’February’,’Mar’:’March’, etc

return months[abbrev]

What about state names? Any other examples like this?

**Character Encodings**

EBCDIC 8-bit

ASCII 7-bit

print(ord(‘a’)) yields the ordinal value 97 (in ASCII)

print(chr(97)) yields the ord ‘a’

UNICODE 16-bit almost all languages

See unicode-table.com

Page 192:

print(‘\u0409\u0443\u0431\u043e\u043c\u0438\u0440’) Cyrillic

print(‘\u0393\u03b5\u03b9\u03b1’) Greek

print(‘\u4e16\u754c\u60a8\u597d’) Chinese

UTF-8 8-bit code based on UNICODE

**Tuples**

Very much like a list but is immutable. Order is maintained.

days = (‘Mo’, ‘Tu’, ‘We’) parantheses are optional

print(days) (‘Mo’, ‘Tu’, ‘We’)

print(type(days)) <class ‘tuple’>

print(days[2]) ‘We’

days[2] = ‘Th’ Error!! Tuples are immutable

Lists cannot be keys in a dictionary <key:value> pair (because lists are mutable and keys must be immutable.)

Tuples can be keys in a dictionary.

phonebook = { (‘Anna’, ‘Karenina’) : ‘(123)456-7890’, …

Example

Prompt user for first name and last name, create a tuple, use that tuple as a dictionary lookup for a phone number.

firstname = input(‘Enter first name: ‘)

lastname = input(‘Enter last name: ‘)

person = (firstname, lastname)

if person in phonebook:

print(phonebook[person])

else:

print(‘Sorry’)

**Module random and its Methods**

import random Don’t forget this!

randrange

>>> random.randrange(1,7) returns a rand number between 1 and 6 inclusive

uniform

>>>random.uniform(0,1) returns a random number bet 0 and 1

shuffle

>>> lst = [1,2,3,4,5]

random.shuffle(lst)

lst

random.shuffle(lst)

lst

choice

>>> lst = [‘cat’, ‘rat’, ‘bat’, ‘mat’]

random.choice(lst) returns a value randomly from lst

sample

>>> random.sample(lst,2) returns 2 values randomly from lst

Example - Practice Problem 6.9 Page 195 – Write function guess(n) which takes n and generates a random number between 0 and n. Then prompt the user to guess the number. If guessed, print “You got it”. If too high, print “Too high”. If too low, print “Too low”.

**Sets**

Unordered collection of items with no duplicates. A set is mutable, but items must be immutable.

phonebook1 = {‘(123)456-7890’, ‘(555)555-5555’}

names = {‘Bill’, ‘Al’, ‘Sue’, ‘Bill’, ‘Zygote’} Only 1 ‘Bill’ stored

Use the Set to remove duplicates from a List:

ages = [23,19,18,22,18,21,22,23,17,18,22]

ages = list(set(ages))

print(ages) yields [17,18,19,21,22,23]

To create an empty set:

phonebook2 = set() because phonebook2 = {} would create a dictionary!

**set Operators**

if ‘Bill’ in names:

len(names)

Comparison Operators

if set1 == set2: 2 sets with the exact same elements

also !=

set1 <= set2 set1 is a subset of set2 (two sets can be equal)

set1 < set2 set1 is a proper subset of set2 (not equal)

set1 | set2 yields union

set1 & set2 yields intersection

set1 – set2 yields difference between 2 sets

set1 ^ set2 symmetric diff: either in the first set of second set but not both

**set Methods**

add phonebook2.add(‘xyz blah blah’) add an item to a set

remove phonebook2.remove(‘blah blah’) remove an item from a set

clear phonebook3.clear() empties a set

Review

List: [2, 3, 4.0, ‘Hello’] Values ordered; mutable; anything can be in a list

Dictionary: {‘1234’:’Smith’} No order; key:value pairs; key must be immutable and one thing; value can be mutable but only one thing

Tuple: (23, 4, 8) Values ordered; immutable; good for keys in dictionary key:value pairs

Set: {‘dog’, ‘cat’, ‘bird’} Unordered; no duplicates kept; items in set must be immutable

Example (for entire chapter) – Blackjack

shuffle deck

deal 2 cards to player and house each

while user says ‘hit’, keep dealing cards

if >21 then player loses

while house < 17, dealer keeps taking cards

if >21, then dealer (house) loses

both stand? who is closer to 21?

function shuffledDeck() on page 200

function dealCard() on page 200

function total(hand) on page 201

Example

Caesar Cipher page 212

**Exceptions (Chapter 7)**

Try this:

strAge = input(‘Enter your age: ‘)

intAge = int(strAge)

print(‘You are {} years old.’.format(intAge))

What happens if you enter *twenty*?

try:

strAge = input(‘Enter your age: ‘)

intAge = int(strAge)

print(‘You are {} years old.’.format(intAge))

except:

print(‘Enter your age using digits 0 – 9! ‘)

strAge = input(‘Enter your age: ‘)

intAge = int(strAge)

print(‘You are {} years old.’.format(intAge))

Better yet: One of FEW places you can use while True…break

while True:

try:

x = int(input(‘Please enter a number: ‘))

break

except:

print(‘Oops! That was not a valid number. Try again.’)

**Multiple Exception Handlers**

try:

infile = open(filename)

strAge = infile.readline()

age = int(strAge)

print(‘Age is ‘, age)

except IOError:

print(‘Input/Output Error’)

except ValueError:

print(‘Value not an integer’)

except:

print(‘Some other error’)

Some of the more common exceptions:

* IOError
* ValueError – raised when a built-in operation or function receives an argument that has the right type but an inappropriate value (and not described by an IndexError)
* NameError – raised when a local or global name is not found
* IndexError
* KeyError – dictionaries
* ZeroDivisionError
* OverflowError
* EOFError
* KeyboardInterrupt
* TypeError – raised when an operation or function is applied to an object of inappropriate type

**Namespaces**

Functions are useful because:

* code reuse
* modularity (or procedural decomposition)
* encapsulation (or info hiding – hides the details)
* creates local variables

def func1():

x = 0

y = 1 both of these variables are local to func1 only

Example

def double(y):

x = 2

print(‘x = {}, y = {}’.format(x,y))

return x \* y

print(double(4))

print(‘x = {}, y = {}’.format(x,y))

What if leave out: x = 2 ? X is unknown.

What if we add x = 20 outside function double? x is now a *global* variable.

How do we define a variable?

1. Look first in the enclosing function call namespace

2. Then look in the global (module) namespace (not the function that called it, if there is one)

3. Finally look in the namespace of the builtin modules (print, sum, len,…)

How do we make a variable in a function global instead of local?

def f(b):

global a

a = 6

return a \* b

a = 0

print(‘f(3) = ‘, f(3))

print(‘a = ‘, a)

**Namespaces and Program Stack**

The OS helps us when a function is called from a main module, or a function calls another function. The OS maintains a stack of namespaces and return points. It uses pushes and pops to do this.

Example Show the program stack as main module calls f, then f calls g, then g calls h

def h(n):

print(‘Start h’)

print(1/n)

print(n)

def g(n):

print(‘Start g’)

h(n-1)

print(n)

def f(n):

print(‘Start f’)

g(n-1)

print(n)

>>> f(4)

Start f

Start g

Start h

0.5

2

3

4

**Modules as Namespaces**

>>> import math math is a module; when imported, it becomes a namespace

>>> dir(math) this will display all the module’s “attributes”

>>> math.pi namespace math has an attribute called pi

What happens when you do an import?

1. Looks for the file corresponding to the module. Python uses a search path, or list of directories

>>> import sys

>>> sys.path this will display the search path

What if your subdirectory is not in the list? Add it:

>>> import sys

>>> sys.path.append(‘C:/Users/me’)

>>> sys.path

2. Runs the module’s code to create the objects defined in the module

3. Creates a namespace where the names of these objects will live

**Classes as Namespaces**

>>> lst = [5, 3, 5, 8, 2, 1, 7]

lst.sort()

is actually interpreted as:

list.sort(lst) list is the namespace, sort the function/method in the namespace

lst.append(9)

is actually interpreted as:

list.append(lst,9)

**Break, Continue, Pass**

*Break* kicks you out of current loop; *continue* skips one of the iterations:

i = 0

while True:

i += 1

if i == 50:

continue # skips the 50th loop

print(i)

if i == 75:

break # terminates the current loop

*Pass*

if n % 2 == 0:

pass Reverse logic and skip else? Maybe as placeholder.

else:

print(n)

**Recursion** (Chapter 10)

Let’s write a program that prints:

5

4

3

2

1

Blastoff!!!

n = 5

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

print(i)

print(‘Blastoff!!!’)

Now let’s try it using a recursive solution. In recursion, the function calls itself.

Let’s start n at 5 and print that value. Then decrement n by 1 and call ourselves.

When n is less than 1, we will print Blastoff!!! and return.

Note: There are no for-loops:

def countdown(n):

if n <= 0:

print(‘Blastoff!!!’) Base case – stops the recursive calls

else:

print(n)

countdown(n-1)

countdown(5)

Sometimes “thinking recursively” is a better and more “elegant” solution! But not always.

Example

Take an integer such as 3124 and print the digits individually down the page:

3

1

2

4

s = str(3124)

for i in s:

print(i)

Cheating! Let’s try to leave it as an integer:

3124 % 10 yields 4. 3124 // 10 yields 312. …

Let’s try a recursive solution. Take n, integer divide by 10, then pass it to ourselves. When n is less than 10 (a single-digit value), print n and return.

def vertical(n):

if n < 10: base case

print(n)

else:

vertical(n // 10)

print(n % 10)

Let’s trace the above problem and show the sequence of calls (page 357 from book).

Example

Take a multi-digit value and print the digits in reverse.

def reverse(n):

if n < 10: base case

print(n)

return

else:

print(n % 10)

reverse(n // 10)

reverse(1234)

Example

Write a recursive function numOnes() that takes a non-negative integer n as input and returns the number of 1s in the binary representation of n. Use the fact that this is equal to the number of 1s in the representation of n // 2, plus 1 if n is odd.

For example, take the number 14 and repeatedly divide it by 2 keeping track of the remainders.

def numOnes(n, count): *count* is a counter of number of 1s

if n == 1:

count += 1

return count

elif n % 2 == 0:

return numOnes(n // 2, count)

else:

count += 1

return numOnes(n // 2, count)

count = 0

print(numOnes(14, count))

Here is a slick solution that does not use a second parameter:

def numOnes2(n):

if n == 1:

return 1

elif n % 2 == 0:

return numOnes(n // 2) + 0

else:

return numOnes(n // 2) + 1

print(numOnes(1))

print(numOnes(14))

print(numOnes(151))

Example – Fractals (inefficient example):

def koch(n):

if n == 0:

return 'F'

return koch(n-1) + ‘L’ + koch(n-1) + ‘R’ + koch(n-1) + ‘L’ + koch(n-1)

Efficient example:

def koch(n):

if n == 0:

return 'F'

tmp = koch(n-1)

return tmp + 'L' + tmp + 'R' + tmp + 'L' + tmp

from turtle import Screen, Turtle

def drawKoch(n):

s = Screen()

t = Turtle()

directions = koch(n)

for move in directions:

if move == 'F':

t.fd(300/3\*\*n)

if move == 'L':

t.lt(60)

if move == 'R':

t.rt(120)

# s.bye()

drawKoch(4)

Let’s follow this by hand: try calling drawKoch(1), then maybe 2.

Virus Scanner Example

Use recursion to traverse tree directory. Go down a branch until you have an exception (can’t go any further).

import os

def scan(pathname, signatures):

for item in os.listdir(pathname):

next = os.path.join(pathname, item)

try:

scan(next, signatures)

except:

for virus in signatures:

if open(next).read().find(signatures[virus]) >= 0:

print(‘{}, found virus {}’.format(next, virus))

*or sub with:*

print(‘Scanning’)

scan(‘c:/some file name’, make up a value)

Exponent Example

How is 2\*\*4 calculated? Multiple multiplications?

def power(a,n):

res = 1

for i in range(n):

res \*= a

return res

Try it: power(2,5) which is 2\*\*5

What if n = 10,000? 2\*\*10000 gives us 10,000 multiplications. Is there a better way?

We know an is equal to an/2 \* an/2 (assuming n is even).

So let’s split the exponent in half and then multiply it by itself. And then let’s do this recursively:

def rpower(a,n):

if n == 0:

return 1 Base case, because anything raised to 0th power is 1

tmp = rpower(a, n // 2)

if n % 2 == 0:

return tmp \* tmp Even case

else:

return a \* tmp \* tmp Odd case; need one more multiplication

Let’s try it, and see if we can follow it on the board for:

rpower(2,10)

So which is more “efficient”: power() or rpower()

Put: *global counter* right after function def

And: *counter +=1* just before: return tmp\*tmp

and: *counter += 2* just before: return a\*tmp\*tmp

Fibonacci Sequence Example

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Code from Chapter 5:

def fib(n):

previous = 1

current = 1

i = 1

while i < n-1:

previous, current = current, previous+current

i += 1

return current

What about a recursive solution:

def rfib(n):

if n <= 2:

return 1

return rfib(n-1) + rfib(n-2)

Try it: rfib(1) rfib(3) rfib(5) rfib(20) rfib(35)

Really slow!!!

How about a better way to determine how long it takes for a function to run?

import time

def fib(n):

previous = 1

current = 1

i = 1

start = time.time()

while i < n:

previous, current = current, previous+current

i += 1

end = time.time()

print('time: ', end - start)

return current

print(fib(10))

More elaborate example of using the time method on pages 371-372:

(open and run file *TimingAnalysis.py*)

**Binary Search**

Show linear search (values sorted and values not sorted)

Show how binary search works (Figure 10.14 page 376)

def search(lst, target, i, j):

if i == j: # base case – empty list

return -1 # target cannot be found

mid = (i + j) // 2

if lst[mid] == target:

return mid

if target < lst[mid]:

return search(lst, target, i, mid)

else:

return search(lst, target, mid+1, j)

lst[2, 5, 11, 13, 16, 24, 28, 41, 42, 43, 45, 57, 72, 73, 89, 90, 99]

search(lst, 45, 0, len(lst))

**Object Oriented Programming** (Chapter 8)

What is OOP? A data object(s) and the methods that operate on it.

For example, design an object for a calculator / cell phone / cash register.

Need to design the methods that will operate on the object, and the container(s) that will store the data for the object.

Example

Let’s define a class for a Point in an x-y grid:

<draw a figure of x-y grid with a point on it>

So we need something to hold the coordinates, and we need to create methods that operate on it, such as:

set x

set y

get returns the x,y coordinates

move change the x,y coordinates to new location dx,dy

So how do we create our own Point class?

class Point:

def setx(self, xcoord): Note 2 params – one for the name of the class and one for the xcoord

self.x = xcoord Could use any var name but *self* is common w/Python users

Can’t say *x = xcoord* because then x would be local to only *setx*

def sety(self, ycoord):

self.y = ycoord

def get(self):

return(self.x, self.y)

def move(self, dx, dy):

self.x += dx

self.y += dy

Let’s try our new class. Enter the above code in a file, then add the following statements after the class definition and run the program.

a = Point() This creates an instance of Point and names it *a*

a.setx(3)

a.sety(4)

print(a.get()) would return and print (3, 4)

What if we created another Point?

b = Point()

b.setx(8)

b.sety(1)

print(b.get()) would return and print (8, 1)

Note that we now have 2 instances of Point: a and b. And each instance has its own x coord and y coord. (Show the memory spaces on the board)

**Overloaded Constructor Operator**

How did we create a point using the class Point?

a = Point()

and then we set the x and y coordinates in separate statements:

a.setx(3)

a.sety(4)

Wouldn’t it have been nice to do all the above with one statement:

a = Point(3,4)

You can, but you have to do the following:

class Point:

def \_\_init\_\_(self, xcoord, ycoord):

self.x = xcoord

self.y = ycoord

Now you can say:

a = Point(3,4) This will create a “blank” Point object first, then executes

\_\_init\_\_ and initializes the x and y variables.

Note: If you use \_\_init\_\_ with 3 params, then you have to use a = Point() with 3 parameters.

Unless, you do this:

def \_\_init\_\_ (self, xcoord = 0, ycoord = 0): The 0s are default values

Now you can say:

a = Point(3,4) get (3,4)

or b = Point() get (0,0)

or c = Point(5) get (5,0)

Example

Let’s make a class for *employees*. What methods might we make to support this class?

What containers will we store the data in?

How about:

setName text

setID text

giveRaise float?

givePromotion text

class Employee:

‘represents an employee’ this is a docstring

def \_\_init\_\_(self, name, ID):

‘constructor’

self.nam = name

self.idee = ID

def setName(self, name):

self.nam = name

def setID(self, ID):

self.idee = ID

def giveRaise(self, rais):

self.raize = rais

def givePromotion(self, promote):

self.pro = promote

def printEmpl(self):

print(self.nam, self.idee)

empl1 = Employee()

empl1.setName(‘Bill Smith’)

empl1.setID(‘123456’)

empl1.printEmpl()

Example

Let’s revisit our Deck (of Cards) and make a Deck class.

The methods that the Deck class should support are:

1. Deck() Constructor that inits the deck to contain a standard deck of 52 playing cards

2. shuffle() Shuffles the deck

3. dealCard() Pops and returns the card at the top of the deck

4. printDeck() Prints the current deck

What is the container? How about keeping the cards in a **list**.

import random

class Deck:

‘represents a deck of 52 cards’

ranks = {‘2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,’10’,’J’,’Q’,’K’,’A’} #ranks is a class variable

suits = {‘\u2660’,’\u2661’,’\u2662’,’\u2663’} # so is suits

# since ranks and suits won’t be modified and they are shared by all Deck instances,

# let’s make them class variable (more on class variables shortly)

def \_\_init\_\_(self):

self.deck = []

for suit in Deck.suits:

for rank in Deck.ranks:

self.deck.append(rank + ‘ ‘ + suit)

def dealCard(self):

‘deal (pop and return) card from the top of the deck’

return self.deck.pop() or pop(0) ??

def shuffle(self):

random.shuffle(self.deck)

def printDeck(self):

print(self.deck)

adeck = Deck()

adeck.printDeck()

adeck.shuffle()

adeck.printDeck()

print(adeck.dealCard())

print(adeck.dealCard())

print(adeck.dealCard())

**Class Variable**

You can create a variable inside a class.

Example

>>> class Test:

version = 1.02 #this is a class variable

a = Test()

b = Test()

a.version yields 1.02

b.version 1.02

Test.version 1.02

Test.version = 1.03

Test.version 1.03

a.version 1.03

b.version 1.03

a.version = ‘latest’

a.version ‘latest’

b.version 1.03

Example

Let’s make a class for a queue.

What is a queue? How does it compare to a stack? Stack examples: infix/postix arithmetic; TCP/IP protocol stack. Queue examples: routers, check-out lines, print queues

What are the methods? What is the container(s)?

class Queue:

def \_\_init\_\_(self):

self.q = [] Make the queue a list internally

def isEmpty(self):

return (len(self.q) == 0)

def enqueue(self, item):

self.q.append(item)

def dequeue(self):

return self.q.pop(0) # call isEmpty in main before you call dequeue

fruit = Queue()

fruit.enqueue(‘apple’)

What if we wanted to print the queue? Would this work:

print(fruit)

No. As we’ve seen, we could create a new function that prints the queue:

def printQueue(self):

print(self.q)

fruit.printQueue()

fruit.enqueue(‘banana’)

fruit.printQueue()

print(fruit.dequeue())

Let’s take a look at a better way of doing this.

**Overloaded Operators**

Take the + operator. We can use it to add two integers: 2 + 4

We can use it to concatenate two strings: ‘abc’ + ‘xyz’

We can also use it to join two lists: [1,2,3] + [5]

So the + operator is actually overloaded. Its true form is this:

int(2).\_\_add\_\_(4)

‘abc’.\_\_add\_\_(‘xyz’)

[1,2,3].\_\_add\_\_([5])

As an example of creating an overloaded operator, let’s go back to our Point class.

a = Point(3,4)

b = Point(3,4)

if a == b ? Shouldn’t they be equal? But they are not.

So add the following overloaded operator = \_\_eq\_\_ to the Point class:

def \_\_eq\_\_(self, other):

return self.x == other.x and self.y == other.y

Now try the Point class with the inclusion of the above def, then try:

a = Point(3,4)

b = Point(3,4)

if a == b:

print(‘The two points are equal.’)

else:

print(‘The two points are not equal.’)

Here is another example of what we can’t do.

What if you want to print a or b?

We already saw you can’t do that without writing a special function to handle prints.

Here is maybe a slightly better way to do that:

def \_\_repr\_\_(self):

return ‘Point({}, {})’.format(self.x, self.y)

Now you can say:

a = Point(4,2)

print(a)

You can also use the overloaded operator \_\_str\_\_ in place of \_\_repr\_\_.

def \_\_str\_\_(self):

return ‘Point({}, {})’.format(self.x, self.y)

If you have both, \_\_repr\_\_ is called in the interpreter shell window, while \_\_str\_\_ is called from a print statement.

Example

Let’s update the Queue class to add overloaded operators:

class Queue:

def \_\_init\_\_(self, q = None): Not sure I’m crazy about this. Maybe leave as earlier.

if q == None:

self.q = []

else:

self.q = q

def \_\_eq\_\_(self, other):

return self.q == other.q

def \_\_len\_\_(self):

return len(self.q)

def \_\_repr\_\_(self):

return ‘Queue: {} ‘.format(self.q)

def enqueue(self, item):

self.q.append(item)

def isEmpty(self):

return len(self.q) == 0

def dequeue(self): #call isEmpty in main before you call dequeue

return self.q.pop(0)

Now let’s try using the class Queue in an example. A router has one or more input ports (queues) and one or more output ports (queues). The choice of output port depends upon the IP address of the packet. Let’s create a router with one input port (port 1) and two output ports. IP addresses 184.37.200.0 will be sent to port 2, while all other IP addresses will be sent to port 3.

class Queue:

‘place the Queue class def here’

def inputPacket(packet):

‘take IP packet and place into input queue’

port1.enqueue(packet)

def processPacket():

‘take IP packet from input queue, process, and send to appropriate output queue’

packet = port1.dequeue()

#analyze packet

#move to appropriate queue

if packet == “184.37.200.0”:

port2.enqueue(packet)

else:

port3.enqueue(packet)

def outputPacket2():

‘take IP packet from output queue and transmit’

if not port2.isEmpty():

packet = port2.dequeue()

print(packet)

else:

print(‘Queue is empty’)

def outputPacekt3(): # Essentially same as outputPacket2

port1 = Queue()

port2 = Queue()

port3 = Queue()

:

while True:

if inputFlag:

inputPacket(packet)

if not port1.isEmpty():

processPacket()

if not port2.isEmpty():

outputPacket2()

if not port3.isEmpty():

outputPacket3()

**Inheritance**

What if we want to create a class of lists which supports all the list methods we know and love but then also has an additional method or two?

Do this:

import random

class MyList(list): Note the term list inside the ()

This is how we tell what the superclass is.

def choice(self): We are adding this method to our MyList

return random.choice(self)

We have just defined *MyList* as a subclass of the class *list* and thus supports all the methods that class list supports, along with a new method called *choice*.

If MyList is a subclass of the class list, then list is a superclass of the class MyList.

Example

Let’s create a class Bird which inherits the methods from class Animal.

Here is class Animal:

class Animal:

‘represents an animal’ this is a docstring

def \_\_init\_\_(self, species = ‘animal’, language = ‘make sounds’):

‘constructor’

self.spec = species

self.lang = language

def setSpecies(self, species):

‘sets the animal species’

self.spec = species

def setLanguage(self, language):

‘sets the animal language’

self.lang = language

def speak(self):

‘prints a sentence by the animal’

print(“I am a {} and I {}.’.format(self.spec,self.lang))

Now let’s create a subclass of Animal, Bird:

class Bird(Animal):

def speak(self):

print(‘{}! ‘.format(self.lang) \* 3)

So class Bird inherits all the methods of class Animal and adds a new method *speak*, which overrides the method *speak* in Animal.

In general, you can create a subclass which inherits all the methods from a superclass and nothing else (1), you can create a subclass which has a method that overrides the same method in the superclass (2), you can create a subclass which extends a method from a superclass (3), or you can create a subclass which inherits all the methods from a superclass and adds a brand new method (4).

class Super:

def amethod(self):

print(‘In Super method’)

(1) class Inheritor(Super):

pass

(2) class Replacer(Super):

def amethod(self):

print(‘In Replacer method’)

(3) class Extender1(Super):

def amethod(self):

print(‘Starting Extender method’)

Super.amethod(self)

print(‘Ending Extender method’)

(4) class Extender2(Super):

def brandNew(self):

print(‘This is a unique method’)

something = Inheritor()

something.amethod()

another = Replacer()

another.amethod()

third = Extender1()

third.amethod()

fourth = Extender2()

fourth.brandNew()

Example – Implementing a Queue Class by Inheriting from list

class Queue2(list):

‘a queue class, subclass of list’

def isEmpty(self):

return (len(self) == 0)

def dequeue(self):

return self.pop(0)

def enqueue(self, item):

self.append(item)

We don’t need \_\_repr\_\_ or \_\_len\_\_ methods because Queue2 is a subclass of list, which means Queue2 inherits ALL the list methods. But is that necessarily good?

Because now we can insert into the middle of the list (queue) or do anything we want to do to a list (queue). Some would say this defeats the concept of OOP encapsulation and information hiding.

Example

Develop a class BankAccount that supports these methods:

* \_\_init\_\_(): Initializes the bank account balance to the value of the input argument or to 0 if no input argument is given
* withdraw(): Takes an amount as input and withdraws it from the balance
* deposit(): Takes an amount as input and adds it to the balance
* balance(): Returns the balance on the account

Now create a subclass called CheckingAccount which limits the withdrawal to $10,000.00.

One last example (not from our book):

class Person:

def \_\_init\_\_(self, name, job = None, pay = 0):

self.name = name

self.job = job

self.pay = pay

Now, let’s test it. (Write a little code, test it. Write a little more, test it. …)

Place the following code in the same file (the main module) as the above class definition:

bob = Person(‘Bob Smith’)

sue = Person(‘Sue Jones’, job = ‘dev’, pay = 100000)

print(bob.name, bob.pay) OUCH! Should not reference *name* or *pay*!

print(sue.name, sue.pay)

Is there a way that we could insert the above test code into our main module, but it would only run when this class is in the main module and NOT when it has been imported into someone else’s work?

Yes:

class Person:

def \_\_init\_\_(self, name, job = None, pay = 0):

self.name = name

self.job = job

self.pay = pay

if \_\_name\_\_ == ‘\_\_main\_\_’:

bob = Person(‘Bob Smith’) #But now Bob has no job or pay and no way to update

sue = Person(‘Sue Jones’, job = ‘dev’, pay = 100000)

print(bob.name, bob.pay)

print(sue.name, sue.pay)

Next, what if we want to split the name into two parts and print only the last name? We could say the following in our test code:

names = bob.name.split()

print(names[-1])

But as we have seen, it is much better to create a method to do this for us. If nothing else, the code for splitting a name is now in ONE place, thus simplifying code maintenance.

class Person:

def \_\_init\_\_(self, name, job = None, pay = 0):

self.name = name

self.job = job

self.pay = pay

def lastName(self):

return self.name.split()[-1]

And let’s add an overloaded operator method:

def \_\_repr\_\_(self):

return ‘Person: {}, {}’.format(self.name, self.pay)

\_\_repr\_\_ is run automatically every time we convert something to a string, such as is done in a print statement.

Let’s add a method to Person to calculate a pay raise:

def giveRaise(self, percent):

self.pay = int(self.pay \* (1 + percent))

Let’s create a subclass – a manager :

class Manager(Person):

Now Manager has all the methods that Person has. What if we want to also add a method to calculate a pay raise for managers?

class Manager(Person):

def giveRaise(self, percent, bonus = 0.10):

self.pay = int(self.pay \* (1 + percent + bonus))

But what if the company changes its policy for granting raises? Then you have to change the code in TWO places. Not good programming.

So let’s augment the original giveRaise method:

class Manager(Person):

def giveRaise(self, percent, bonus = 0.10):

Person.giveRaise(self, percent + bonus)