Kendra Repo SI 106 Final Exam

Operators

\*\*:exponentation

/: produces a float if either operands are floats, if both are int truncates down.

//: integer division

%: yields remainder

Comparison Operators

==:produces Boolean value

!=: not equal

>=: greater or equal

Type Conversion Functions

int-can covert float/string(if#)

float- can convert int/string(#)

str- can covert all

Strings

* [ ] selects a single char
* Immutable, can slice/con

String Methods

* Ex: s.upper() to invoke

|  |  |  |
| --- | --- | --- |
| Method | Params | Description |
| upper | none | Returns string in all uppercase |
| lower | none | In all lowercase |
| capitalize | none | String w/ 1st char cap, rest lower |
| strip | none | Leading & trailing whitespace gone |
| lstrip | none | Leading whitespace gone |
| rstrip | none | Trailing whitespace gone |
| count | item | # occurrences of item |
| replace | old,new | Replaces all old w/ new |
| find | item | Leftmost index where item found |
| rfind | item | Rightmost index where item found |
| index | item | Like found but error if not found |
| split | item | Splits into list where item at. Removes item. |

Lists

* fruit[0] = “pear”, change
* del removes
* insert elements ex: [4:4]
* in order to concat, must add 2 lists.
* Append modifies original list. Concat creates new

List Methods

|  |  |  |
| --- | --- | --- |
| Method | Params | Description |
| append | item | Adds new item to end of list |
| insert | pos, item | Inserts item as given pos |
| pop | none | Removes and returns last item |
| pop | position | Removes&returns item at pos |
| sort | none | Modifies list to be sorted |
| reverse | none | Modifies list to be in reverse order |
| index | item | Returns pos of 1st occurrence of item |
| count | item | Returns # of occurrences of item |
| remove | item | Removes 1st occourance of item |

Accumulating a list ex:

Transform Sum

nums = [3,5,8]

accum = [] (accum = 0)

for w in nums: (same)

x= w\*\*2 (accum = accum+w)

accum.eppend(x)

Unix Commands

**ls:** displays directory contents

**cat**: displays contents of a file, can also concatenate files together, & redirect output (>), redirect input(<), append (>>)

**pipe |** the output of one to input of other

**grep** Ex: grep <word> <filename> prints each line w/ word in file

**less:** controlled

**diff** analyzes two files and prints the lines that are different

Files

open(filename, “r”) opens file for reading, returns ref to file obj

open(filename, “w”) for writing..

filevar.close()

filevar.write(asting) adds astring to end of file, remember astring + ‘\n’ if needed

filevar.read() returns a single string of file or n chars if n

filvevar.readline() returns one line up to newline char, or n

filevar.readlines() returns entire file as list of strings (each line), if n, line rounded up.

Dictionaries

* Unordered, key-value pairs.
* Empty dict = {}

Dictionary Methods

|  |  |  |
| --- | --- | --- |
| keys | None | Returns a list of keys in dict |
| values | None | Returns a list of values in dict |
| items | none | Returns list of key-value pairs |
| get | key | Returns value associated with key, none otherwise |
| get | Key, alt | Alt otherwise |

Sorted

* Returns a new list
* Optional reverse param
* Optional key param
* Lambda expression, syntax:

Lambda params: expression

Defining Functions

Syntax: def name(params)

Params ake arguments in the def are formal params

<expr>.<methodname>(params)

Listener Loop

Inside the while loop to get user input. Loop repeats until a particular input is received.

Ex: theSum = 0

x = -1

while x != 0:

x = int(raw\_input(“”))

theSum = theSum + x

print theSum

Defining Classes

Syntax: class Classname:

“””Docstring”””

def \_\_init\_\_(self….)

local vars

\_\_str\_\_\_ produces a string rep

used to set up attributes required, the self param references the new object

Requesting Data from the Internet

With unix curl:

* curl https://……..
* curl https://…. | less

With requests.get:

* import requests
* page = requests.get(“https…”)
* print page.text[:100]

returns a response object

JSON

* json.load() takes a string as input and produces a python object(list or dict) as output
* Ex: result = requests.get(“ht tps…”,params = {“format” : “json”})

D = json.loads(result.text) puts code into file.

requests.get("http://runestone.org/API/get\_score",params={'student\_id':374,

'assignment\_id':24})

my\_params['max\_id'] = min(ids) - 1

next\_five\_ids = [tweet['id'] for tweet in r.json()]

 ids = ids + next\_five\_ids

Try/Except

Provides a way to process a run-time error and continue execution.

Syntax: try:

<try code block>

except Exception:

<exception code block>

String Interpolation

Takes a format sting on the left and a value of tuples on the right.

Syntax: “format string %s..” % (values)

.CVS Output

students = [("Jamal", 98, "A+"), ("Eloise", 87, "B+"),("Madeline", 99, "A+")]

outfile = open("grades.csv","w")

*# output the header row* outfile.write("Name”, “score”, “grade”**\n**") *# output each of the rows:*

**for** student **in** students:

outfile.write("*%s*, *%d*, *%s***\n**" % student)

outfile.close()

List Comprehensions

Map- transforms each item

Ex: things4 = map((lambda value: 4\*value), things)

Filter- Keep only certain items

New\_lst = filter(lambda num: num % 2 == 1, nums)

List comp: [<expression> for <item> in <sequence> if <condition>] if optional

Reduce- combine/sum all items

* Len, sum, max, join (“-“.join(str)

nums = [3, 4, 6, -7, 0, 1]

# count them; len

print reduce(lambda x, y: x +1, nums, 0)

# add them up; sum

print reduce(lambda x, y: x + y, nums)

#x refers to the result-so-far and y refers to the next element in the list

# find the largest; max

def greater(x, y):

if x > y:

return x

else:

return y

print reduce(greater, nums)

strings = ["Hello", "hi", "bye", "wonderful"]

# join the strings into one big string

print reduce(lambda x, y: x + "--" + y, strings)

Zip – takes multiple lists and turns them into a list of tuples pairing all 1st items as one tuple, ect.

Test Cases

Syntax: test.testEqual(actual, expected)

**Side effect**

**def** update\_counts(letters, counts\_dict):

**for** c **in** letters:

**if** c **in** counts\_dict:

counts\_dict[c] = counts\_dict[c] + 1 **else**: counts\_dict[c] = 1 counts\_dict = {'a': 3, 'b': 2} update\_counts("aaab", counts\_dict) test.testEqual(counts\_dict['a'], 6) test.testEqual(counts\_dict['b'], 3)

Define a function flatten that takes a list of sub-lists and returns a list that

contains all of the elements of the sub-lists.

def flatten(L):

res = []

for each\_list in L:

for item in each\_list:

res.append(item)

return res

Write code to sort L in order based on the last two characters in each string, so that 253 is first and 356 is last. L = ['154', '253', '356', '455']

L = sorted(L, key=lambda x: x[-2:])

Define a function sorted\_by\_keys that takes a dictionary as input and returns of a list of its values, sorted in alphabetic order based on the keys that have those values.

def sorted\_by\_keys(d):

pairs = d.items()

sorted\_list = sorted(pairs, key= lambda x: x[0])

return [x[1] for x in sorted\_list]

Define a function sleepiest that takes as input a list of strings and returns as output the string that has the most 'z's in it. Several possible answers shown below def sleepiest(L):

max\_zs = L[0]

for item in L:

if item.count('z') > max\_zs.count('z'):

max\_zs = item

return max\_zs

Define a function f that takes a list of strings and returns a list containing

the first letter of every word that contains the letter z. Make it pass the test

below. Your function must use manual accumulation (i.e., you may not use map,

filter, reduce, zip, or list comprehensions.)

def f(L):

res = []

for item in L:

if 'z' in item:

res.append(item[0])

return res

Now define the same function without using manual accumulation. Instead it should use some combination of map, filter, reduce, zip, and list comprehensions.

def f(L):

return [x[0] for x in L if 'z' in x]

Define a method enthusiast\_count for the Post class. It will return a count

of the number of people who both liked and commented on a post. For example,for the sample post, it would return 1, since only id '58763' both liked and commented on it.

def enthusiast\_count(self):

ls = self.likers()

cs = self.commenters()

return len([id for id in ls if id in cs])

Write a test (using test.testEqual) that checks whether the enthusiast\_count method is defined correctly.

test.testEqual(p1.enthusiast\_count(),1)

Which of the following are reasons to use a version control system like git and github? a. You want to be able to allow some people but not others to make changes to the version of the source code that gets distributed. d. You want to hold yourself accountable as an open source project team leader by making it possible for anyone, at any time, to fork the code and start a competing project that excludes you from leadership. e. You would like to be able to see or revert to any past version of any of the files in your project.

Yes, that does prevent you licensing your code with that restriction. The

viral clause of the GPL says that if you license your code, you have to

license it under GPL, which permits recipients to share.

“free” = freedom

Brooks’Law- adding people to a late project makes it later

max\_id param for paging

Overriding method

class Dog(Pet):

sounds = ['Woof', 'Ruff']

def mood(self):

if (self.hunger > self.hunger\_threshold) and (self.boredom > self.boredom\_threshold):

return "bored and hungry"

else:

return "happy"

Super class

class Dog(Pet):

sounds = ['Woof', 'Ruff']

def feed(self):

Pet.feed(self)

print "Arf! Thanks!"

class Bird(Pet):

sounds = ["chirp"]

def \_\_init\_\_(self, name="Kitty", chirp\_number=2):

Pet.\_\_init\_\_(self, name) # call the parent class's constructor

# basically, call the SUPER -- the parent version -- of the constructor, with all the parameters that it needs.

self.chirp\_number = chirp\_number # now, also assign the new instance variable