**Introduction to Loops**

Loops are handy to repeatedly perform a task on different subset of our data. For example, if we want to print out our cards, it is very time consuming to execute print on each element one at a time. Using our cards list from previous classes, we can write:

print ("Card is " + cards[0])

print ("Card is " + cards[1])

print ("Card is " + cards[2])

…

Clearly, it is more handy to loop through the list, and loops are useful for many other tasks too. The main loop type in R is the for loop.

The for loop

The for loop can iterate over an element such as a list.

Here, there are two components: the name of the list and the name of the variable that takes on the current element each time we go through the loop body.

for draw in cards:

print ("Card is " + draw)

print ("Its suit is " + draw[-1])

Above, we loop over the list. We also treat draw, a string, like a list by taking its last element.

Often, we will extract information from a subset of elements within a group.

capitals=("ottawa", "washington","istanbul", "beijing")

long\_c=list()

for capital in capitals:

print ("The city is " + capital)

if (len(capital)>8):

long\_c.append(capital)

print ("There were " + str(len(long\_c)) + " long capitals: ",end="")

print (" ".join(long\_c))

We can also loop over a string.

draw="5H"

for letter in draw:

print ("one character is " + letter)

print ("end")

We can also use the split command to make the string an array, although here it is complicated because we do not have a separator. We can use the list command to make the string a list, and loop over this list.

print (list(draw))

We can also iterate over lines in a file by using a file object. We obtain a file object with the open command that requires the path and filename.

Here is the content of a file superheroes.txt:

Batman Bruce Wayne Hero DC

Catwoman Selina Kyle Villain DC

Invisible Woman Susan Storm Richards Hero Marvel

Supergirl Linda Danvers Hero DC

Superman Clark Kent Hero DC

Wonder Woman Diana Prince Hero DC

file= open ("superheroes.txt")

file= open ("/Users/lewislukens/Teaching /Bioinfclasses/BINF6410/superheroes.txt")

for line in file:

print (line)

file.close()

Note we can also tell python the filename we want it to open. sys.argv is a list containing all the arguments typed by the user on the command line when executing the script. The argv list includes the script name itself.

Here is a program (e.g. loop.py) that outputs the content of a file that the user provides.

e.g. python3 loop.py superheroes.txt

import sys

print (sys.argv[0]) #this is the program name. If you need to add a path, put the path in quotes and concatenate sys.argv[0]

file= open (sys.argv[1]) #sys.argv[1] contains the 2nd argument

for i in file:

print (i)

print ("goodbye")

Note that file objects are exhaustible. Once we have iterated a file object, Python “knows” it is at the end, and there are no more lines to read. A solution is to close and reopen the file every time we want to read it. We can use the file method readlines()

file= open ("superheroes.txt")

all\_lines=file.readlines()

file.close()

for line in all\_lines:

print ("Hello " + line)

for line in all\_lines:

print ("Hello again " + line)

We can also iterate over a list of numbers. The range() function can be used to generate a series of numbers. With a single argument, range calculates up to that number, starting at 0. It will exclude the final number.

for number in range(5):

print (number)

With two numbers, range will count up from the first to the second, excluding the second.

for number in range(6,10):

print (number)

With three numbers, range will count from the first to the second using the step size of the third

for number in range(6,10,2):

print (number)

Finally, while loops combine the concept of conditionals with repeated tasks.

list = [1, 3, 5, 7, 9]

length = len(list)

i = 0

while i < length: #the while loop runs while a condition is met

print(list[i])

i += 1

Another example:

$x=1;

while ($x <1000) {

print "$x\n";

$x+=$x; #doubles x

}

Note for loops can often do the same thing as while loops and often do it better.

**Introduction to Dictionaries**

Let’s write a program that takes a superhero name as an argument and gives you the superhero’s real name. Perhaps called unmask.py

from sys import exit #we will use the exit function

from sys import argv

if (len(argv)>1):

guess=argv[1]

else:

exit ("I need a superhero name after program name")

print ("Let’s see…")

nicknames=["Batman","Doctor Doom", "Doctor Octopus", "Wonder Woman"]

real\_names=["Bruce Wayne", "Victor von Doom", "Otto Octavius", "Diana Prince"]

for i in range(len(nicknames)):

if (guess == nicknames[i]):

print ("Name " + nicknames[i] + ". Realname "+ real\_names[i])

else:

print ("Don’t know that name. ")

This approach works but is not a good solution for storing associations. If we want to add alias-name pairs, we need to add them to both lists. Also, lists become inefficient as data sets becomes larger- we need to look through one-half the records on average before finding the right one.

*What are dictionaries?*

Dictionaries are a data type. Like a dictionary, you have a word you want to look up and a definition for the word. The dictionary will not have the same lookup word twice, although the definitions of the words may not be unique.

*How do dictionaries work?*

A dictionary associates pairs of items. A dictionary contains keys and values. The key is the lookup word. Once we create the key-value pair, we can quickly access the value with the key.

*Making a dictionary*

Here we can define a dictionary called “alias” and enter key, value pairs

alias = {

'Batman' : 'Bruce Wayne',

'Doctor Doom' : 'Victor von Doom',

'Doctor Octopus':'Otto Octavius',

'Wonder Woman':'Diana Prince'

}

We use dictionary\_name[key] to access the dictionary.

print (alias['Batman'])

print (alias['Doctor Doom'])

Now, our program is better:

print ("My dictionary says " + alias[guess] + " is the real name")

**Working with dictionaries**

*Adding and removing key-value pairs*

To add pairs to a dictionary:

alias['Susan Storm Richards'] = 'Invisible Woman'

alias['Clark Kent']= 'Superman'

alias.clear #deletes the dictionary contents

To replace a value with a different value:

alias['Susan Storm Richards'] = 'The Invisible Woman'

print(alias['Susan Storm Richards']) #to confirm name changed

To remove a key-value pair use the pop method.

alias.pop('Susan Storm Richards')

Removing a key-value pair can create an error message if the key is used.

print(alias['Susan Storm Richards'])

Traceback (most recent call last):

File "dict.py", line 37, in <module>

print(alias['Susan Storm Richards'])

KeyError: 'Susan Storm Richards'

We can avoid this error in two ways.

if 'Susan Storm Richards'in alias:

#this treats alias like a list of keys

print(alias['Susan Storm Richards'])

else:

print ("Not found")

Another way is to use the get method on the dictionary.

print (alias.get('Wonder Woman','no match'))

print (alias.get('Susan Storm Richards','no match'))

One of the most common things to do with a dictionary is to get a list of all of the keys or all of the values. If we use the dictionary as a list, the dictionary yields key values as a list. This can be explicit, or one uses the dictionary name as a list. Note there is no innate order to dictionaries. Keys and values come out in a seemingly random way.

keys=list(alias)

for key in keys:

print ("here is a superhero name " + key)

Here is program to count the number of words entered.

from sys import argv

words=argv[1:len(argv)]

word\_dict={}

for word in words:

if word in word\_dict:

word\_dict[word]+=1

else:

word\_dict[word]=1

for key in word\_dict:

print ("Occurrences of " + key + " are " + str(word\_dict[key]))