Lists

Chapter 10

Today’s Outline

- Intro to list
- Create a list
- Access a list Element
- Modify a list
- for ... in ... Structure
- Split and join
- General “list madness”

Data structures

- Data structures are a way of organizing and storing data so that they can be accessed and worked with efficiently.
- The structure chosen, defines the relationship between the data, and the operations that can be performed on the data.
- There are many various kinds of data structures and we'll be looking at 3 of the most common in Python: lists, dictionaries, and tuples.

Data structures

- When you create a variable, x = 4, you are defining one value: the integer 4.
- A list is like a collection: you can carry many values around in one convenient package.
- A list has many values in a single variable.

Lists: a data structure

- Lists in Python are used to store collection of elements. In general these are homogenous, but they don’t have to be.
- Lists are mutable, which means that you can change their content without changing their identity.
- You can recognize lists by their square brackets that hold elements, separated by a comma **`[1,2]`**.

Lists: the basics

- A string can be regarded as an ordered sequence of characters… so what is a list?
- A list is a sequence of values. A single list may contain data types like integers, strings, or objects.
- The values in a list are called elements (or items).
- The elements in a list can be any type and are indexed like strings 0, 1, 2, 3…

Lists

- Enclose elements in square brackets, `[]` .
- Rule of thumb is: use lists when the items are similar: a sequence of 50 tree names is a great list.
- Use case: Natural Language Processing and turning a Tweet into a list of words to analyze.

```
text = "This is text from a #Tweet."
words = text.split()
print(words)
```

- What's the output?

Creating a List

- Homogenous lists…

```
cities = [“Jakarta”, “Mumbai”, “Nairobi”]
populations = [9608000, 18410000, 3134000]
empty = []
```

- Heterogeneous lists…

```
mixedlist = [“Jakarta”, 9608000, [12,15]]
```

  - What’s odd about this list?

Nested List !!!

List Operations (+, *)

- The ‘+’ operation concatenate lists into one. Print 1-5.

```
a=[1,2,3]
b=[4,5,6]
c=a+b
print(c)
```

- The ‘*’ repeats a list a given number of times. Print list a (above) three times.

```
[a] * 3
```

List Membership

- The in operator can check membership

```
features = ['point', 'line', 'polygon']
print(features)
'point' in features
```

- What's the output? What about…

```
’Point' in features
```

Strings & Lists

- **Join list elements to form a string**

```
historylist = ['Mumbai','was','formerly','Bombay.']
   mystr = '_'.join(historylist)
   print(mystr)
```

What's the output?

Replace w/ a space

- **Find a way to remove the underscores to get: "Mumbai was formerly Bombay"**

Accessing List Elements

- How do you access elements in a list?
  - Use the index! What are the indexes for the following?

```
cities = [“Jakarta”, “Mumbai”, “Nairobi”]
#Try it out in your shell.

cities [0] # Jakarta
cities [1] # Mumbai
cities [2] # Nairobi
```

Accessing List Elements

- Special case: slicing a list with “[::-1]” produces a reversed copy.
- Using this, how can I print this list in reverse?

```
cities = [“Jakarta”, “Mumbai”, “Nairobi”]
print(cities[::-1])
```

- Using the **.reverse()** method reverses the list itself. Until you run reverse again.

```
cities.reverse() #List is now in reverse order!
print(cities)
```

Lists are mutable!

Using Indexing to change an element in a list:

```
cities = ["Jakarta", "Mumbai", "Nairobi"]
print(cities)
```

Change Mumbai to its airport code: BOM.
Use this syntax `variable name[index] = "BOM"`

```
cities[1]="BOM"
print(cities)
```

Lists are mutable!

Using Indexing to change an element in a list:

```
numbers = [17, 123]
print(numbers)
```

Change 123 to 5

```
numbers[1]=5
print(numbers)
```

Lists are mutable!

Changing multiple elements in a list:

```
lamelist = [‘a’,’b’,’c’,’d’]
```

Insert elements… how can we change this to be a,x,y,d?

```
lamelist[1:3] = ['x','y']
print(lamelist)
```

Explain to me how we used the index.

Lists are mutable!

Insert elements in a list: how would you insert letters “b” and “c” in their proper places?

```
letterlist = ['a', 'd', 'f']
letterlist[1:1] = ['b', 'c']
print(letterlist)
```

What about doing the same with the letter ”e”?

```
letterlist[4:4] = ['e']
print(letterlist)
```

Explain how we’re using the index here.

Lists are mutable!

Delete elements in a list! how would delete letters “b” and “c” from the list?

```
letterlist = ['a', 'b', 'c', 'd', ‘f’]
letterlist[1:3] = []
```

You can also use the **`del`** operator.

```
a = ['mean', 'median', 'mode']
del a[…] #which index deletes ‘median’?
print(a)
```

Nested Lists: what is it?

- Use cases for nested lists might include lists that you want to group.

```
rhymes = [['cat', 'hat'], ['mouse', 'house’]]
```

- Or if you wanted to group things by condition: e.g. leaf colors

```
  # Where each list is an observation in the field.
  leaf_color = [[‘red’, ‘yellow'], [‘green’,
  ‘yellow’]]
```

Nested Lists / Indexing

- What are two ways to print “15” from this list?

```
mixedlist = [“Jakarta”, 9608000, [12,15]]

#Create variable using index to slice list
sublist = mixedlist[2]
# Get index for “15”
sublist[1]
```

Or…

```
mixedlist[2][1]
```

Somewhere…

baby mountain goats frolic.

Debugging

First, walk me through the code: explain what it's doing.
Code it up and check the output.
Now, debug this list to print all regions.

```
def printRegions():
    regions = ['north', 'east', 'south', 'west']
    i=0
    while i<3:
        print(regions[i])
        i = i+1
printRegions()
```

Change to 4. Why?

Using len() on a list

Create a function that prints a list of elements using len:

```
airports = ['DEN', 'LAX', 'BRU']

def printList(listname):
    i=0
    while i< len(listname):
        print(listname[i])
        i = i+1

printList(airports)
```

Lists & for Loops

- The `for… in…` structure can be really helpful. Run a block of code for every element in a list.

```
def printList(listVal):
    for listItem in listVal:
        print(listItem)

map_labels = [["red", "green", "blue"],
'yellow', 'cyan', 'magenta’]]

printList(map_labels)
```

Exercise

- Write a Python function that takes a list and returns a new list with unique elements of the first list.
  - Sample input : `[1, 2, 1, 1, 3, 4, 3, 3, 5]`
  - Sample output: `[1, 2, 3, 4, 5]`
- This is a common "data munging" problem: you just want to know the unique values for a given dataset.

Exercise Answer

```
def unique(list1):
    unique_list = []    # intilize empty list
    for x in list1:     # check if exists
        if x not in unique_list:
            unique_list.append(x)
    # to print
    for x in unique_list:
    print(x)

numlist = [1, 2, 1, 1, 3, 4, 3, 3, 5]
unique(numlist)
```

Summary

- Lists are very very useful!
- Lists are mutable, so you can access, change, insert and delete list elements.
- for ... in ... loop widely used in Python
- Useful functions (extend, join, split)