wtech
Teknolojide Kadın Derneği

Introduction to Machine Learning with Python

Lists

Dr. Süha Tuna
İTÜ Informatics Institute

CONTENT

- Basics

Basics

Creating lists Here is a simple list:

```
L = [1,2,3]
```

Use square brackets to indicate the start and end of the list, and separate the items by commas.

The empty list The empty list is `[]`. It is the list equivalent of 0 or `''`.

Long lists If you have a long list to enter, you can split it across several lines, like below:

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
        32, 33, 34, 35, 36, 37, 38, 39, 40]
```

Input We can use `eval(input())` to allow the user to enter a list. Here is an example:

```
L = eval(input('Enter a list: '))
print('The first element is ', L[0])
```

```
Enter a list: [5,7,9]
The first element is 5
```

Printing lists You can use the `print` function to print the entire contents of a list.

```
L = [1,2,3]
print(L)
```

```
[1, 2, 3]
```

An important feature

Data types Lists can contain all kinds of things, even other lists. For example, the following is a valid list:

```
[1, 2.718, 'abc', [5,6,7]]
```

Similarities to `strings`

- `len` — The number of items in `L` is given by `len(L)`.
- `in` — The `in` operator tells you if a list contains something. Here are some examples:

```
if 2 in L:
    print('Your list contains the number 2.')
if 0 not in L:
    print('Your list has no zeroes.')
```

- Indexing and slicing — These work exactly as with strings. For example, `L[0]` is the first item of the list `L` and `L[:3]` gives the first three items.
- `index` and `count` — These methods work the same as they do for strings.
- `+` and `*` — The `+` operator adds one list to the end of another. The `*` operator repeats a list. Here are some examples:

Expression	Result
`[7,8]+[3,4,5]`	`[7,8,3,4,5]`
`[7,8]*3`	`[7,8,7,8,7,8]`
`[0]*5`	`[0,0,0,0,0]`

The last example is particularly useful for quickly creating a list of zeroes.

- Looping — The same two types of loops that work for strings also work for lists. Both of the following examples print out the items of a list, one-by-one, on separate lines.

```
for i in range(len(L)):
    print(L[i])
```

```
for item in L:
    print(item)
```

The left loop is useful for problems where you need to use the loop variable `i` to keep track of where you are in the loop. If that is not needed, then use the right loop, as it is a little simpler.

Functions and `list` methods

There are several built-in functions that operate on lists. Here are some useful ones:

Function	Description
`len`	returns the number of items in the list
`sum`	returns the sum of the items in the list
`min`	returns the minimum of the items in the list
`max`	returns the maximum of the items in the list

For example, the following computes the average of the values in a list `L`:

```
average = sum(L)/len(L)
```

Method	Description
`append(x)`	adds `x` to the end of the list
`sort()`	sorts the list
`count(x)`	returns the number of times `x` occurs in the list
`index(x)`	returns the location of the first occurrence of `x`
`reverse()`	reverses the list
`remove(x)`	removes first occurrence of `x` from the list
`pop(p)`	removes the item at index `p` and returns its value
`insert(p,x)`	inserts `x` at index `p` of the list

Important note There is a big difference between list methods and string methods: String methods do not change the original string, but list methods do change the original list. To sort a list `L`, just use `L.sort()` and not `L=L.sort()`. In fact, the latter will not work at all.

A tricky implementation

Making copies of lists Making copies of lists is a little tricky due to the way Python handles lists. Say we have a list `L` and we want to make a copy of the list and call it `M`. The expression `M=L` will not work for reasons covered in Section 19.1. For now, do the following in place of `M=L`:

```
M = L[:]
```

Changing lists Changing a specific item in a list is easier than with strings. To change the value in location 2 of `L` to 100, we simply say `L[2]=100`. If we want to insert the value 100 into location 2 without overwriting what is currently there, we can use the `insert` method. To delete an entry from a list, we can use the `del` operator. Some examples are shown below. Assume `L=[6,7,8]` for each operation.

Operation	New `L`	Description
`L[1]=9`	`[6,9,8]`	replace item at index 1 with 9
`L.insert(1,9)`	`[6,9,7,8]`	insert a 9 at index 1 without replacing
`del L[1]`	`[6,8]`	delete second item
`del L[:2]`	`[8]`	delete first two items

Examples

```
from random import randint
L = []
for i in range(50):
    L.append(randint(1,100))
```

```
for i in range(len(L)):
    L[i] = L[i]**2
```

```
count = 0
for item in L:
    if item>50:
        count=count+1
```

```
frequencies = []
for i in range(1,101):
    frequences.append(L.count(i))
```

```
scores.sort()
print('Two smallest: ', scores[0], scores[1])
print('Two largest: ', scores[-1], scores[-2])
```

Lists and the `random` module

There are some nice functions in the `random` module that work on lists.

Function	Description
`choice(L)`	picks a random item from `L`
`sample(L,n)`	picks a group of `n` random items from `L`
`shuffle(L)`	Shuffles the items of `L`

Note The `shuffle` function modifies the original list, so if you don't want your list changed, you'll need to make a copy of it.

Example 1 We can use `choice` to pick a name from a list of names.

```
from random import choice
names = ['Joe', 'Bob', 'Sue', 'Sally']
current_player = choice(names)
```

Example 2 The `sample` function is similar to `choice`. Whereas `choice` picks one item from a list, `sample` can be used to pick several.

```
from random import sample
names = ['Joe', 'Bob', 'Sue', 'Sally']
team = sample(names, 2)
```

Further examples

Example 3 The `choice` function also works with strings, picking a random character from a string. Here is an example that uses `choice` to fill the screen with a bunch of random characters.

```
from random import choice
s='abcdefghijklmnopqrstuvwxyz1234567890!@#$%^&*()'
for i in range(10000):
    print(choice(s), end='')
```

Example 4 Here is a nice use of `shuffle` to pick a random ordering of players in a game.

```
from random import shuffle
players = ['Joe', 'Bob', 'Sue', 'Sally']
shuffle(players)
for p in players:
    print(p, 'it is your turn.')
    # code to play the game goes here...
```

Example 5 Here we use `shuffle` divide a group of people into teams of two. Assume we are given a list called `names`.

```
shuffle(names)
teams = []
for i in range(0,len(names),2):
    teams.append([names[i], names[i+1]])
```

split

```
s = 'Hi! This is a test.'
print(s.split())
```

```
['Hi!', 'This', 'is', 'a', 'test.']
```

```
from string import punctuation
for c in punctuation:
    s = s.replace(c, '')
```

Example Here is a program that counts how many times a certain word occurs in a string.

```
from string import punctuation

s = input('Enter a string: ')
for c in punctuation:
    s = s.replace(c, '')
s = s.lower()
L = s.split()

word = input('Enter a word: ')
print(word, 'appears', L.count(word), 'times.')
```

Optional argument The `split` method takes an optional argument that allows it to break the string at places other than spaces. Here is an example:

```
s = '1-800-271-8281'
print(s.split('-'))
```

```
['1', '800', '271', '8281']
```

join

The `join` method is in some sense the opposite of `split`. It is a string method that takes a list of strings and joins them together into a single string. Here are some examples, using the list `L = ['A','B','C']`

Operation	Result
`' '.join(L)`	`A B C`
`''.join(L)`	`ABC`
`', '.join(L)`	`A, B, C`
`'***'.join(L)`	`A***B***C`

`list` comprehensions

List comprehensions are a powerful way to create lists. Here is a simple example:

```
L = [i for i in range(5)]
```

This creates the list `[0,1,2,3,4]`. Notice that the syntax of a list comprehension is somewhat reminiscent of set notation in mathematics. Here are a couple more examples of list comprehensions. For these examples, assume the following:

```
string = 'Hello'
L = [1,14,5,9,12]
M = ['one', 'two', 'three', 'four', 'five', 'six']
```

List comprehension	Resulting list
`[0 for i in range(10)]`	`[0,0,0,0,0,0,0,0,0,0]`
`[i**2 for i in range(1,8)]`	`[1,4,9,16,25,36,49]`
`[i*10 for i in L]`	`[10,140,50,90,120]`
`[c*2 for c in string]`	`['HH','ee','ll','ll','oo']`
`[m[0] for m in M]`	`['o','t','t','f','f','s']`
`[i for i in L if i<10]`	`[1,5,9]`
`[m[0] for m in M if len(m)==3]`	`['o','t','s']`

As we see in the last two examples, we can add an `if` to a list comprehension. Compare the last example with the long way of building the list:

```
L = []
for m in M:
    if len(m)==3:
        L.append(m)
```

`list` comprehensions

```
L = [[i,j] for i in range(2) for j in range(2)]
```

```
[[0, 0], [0, 1], [1, 0], [1, 1]]
```

This is the equivalent of the following code:

```
L = []
for i in range(2):
    for j in range(2):
        L.append([i,j])
```

Here is another example:

```
[[i,j] for i in range(4) for j in range(i)]
```

```
[[1, 0], [2, 0], [2, 1], [3, 0], [3, 1], [3, 2]]
```

Using `list` comprehensions

Example 1 Write a program that generates a list `L` of 50 random numbers between 1 and 100.

```
L = [randint(1,100) for i in range(50)]
```

Example 2 Replace each element in a list `L` with its square.

```
L = [i**2 for i in L]
```

Example 3 Count how many items in a list `L` are greater than 50.

```
len([i for i in L if i>50])
```

Example 4 Given a list `L` that contains numbers between 1 and 100, create a new list whose first element is how many ones are in `L`, whose second element is how many twos are in `L`, etc.

```
frequencies = [L.count(i) for i in range(1,101)]
```

Another example The `join` method can often be used with list comprehensions to quickly build up a string. Here we create a string that contains a random assortment of 1000 letters.

```
from random import choice
alphabet = 'abcdefghijklmnopqrstuvwxyz'
s = ''.join([choice(alphabet) for i in range(1000)])
```

Using `list` comprehensions

One more example Suppose we have a list whose elements are lists of size 2, like below:

```
L = [[1,2], [3,4], [5,6]]
```

If we want to flip the order of the entries in the lists, we can use the following list comprehension:

```
M = [[y,x] for x,y in L]
```

```
[[2, 1], [4, 3], [6, 5]]
```

Two-dimensional `lists`

There are a number of common things that can be represented by two-dimensional lists, like a Tic-tac-toe board or the pixels on a computer screen. In Python, one way to create a two-dimensional list is to create a list whose items are themselves lists. Here is an example:

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

Indexing We use two indices to access individual items. To get the entry in row `r`, column `c`, use the following:

```
L[r][c]
```

Printing a two-dimensional list To print a two-dimensional list, you can use nested for loops. The following example prints a 10×5 list:

```
for r in range(10):
    for c in range(5):
        print(L[r][c], end=" ")
    print()
```

Another option is to use the `pprint` function of the `pprint` module. This function is used to "pretty-print" its argument. Here is an example to print a list `L`:

```
from pprint import pprint
pprint(L)
```

The `pprint` function can be used to nicely print ordinary lists and other objects in Python.

Working with two-dimensional `lists`

```
count = 0
for r in range(10):
    for c in range(5):
        if L[r][c]%2==0:
            count = count + 1
```

This can also be done with a list comprehension:

```
count = sum([1 for r in range(10) for c in range(5) if L[r][c]%2==0])
```

Creating large two-dimensional lists To create a larger list, you can use a list comprehension like below:

```
L = [[0]*50 for i in range(100)]
```

This creates a list of zeroes with 100 rows and 50 columns.

Picking out rows and columns To get the row r of `L` (starting at row $r = 0$), use the following:

```
L[r]
```

To get the column c of `L` (starting at column $c = 0$), use a list comprehension:

```
[L[i][c] for i in range(len(L))]
```

Working with two-dimensional `lists`

Flattening a list To flatten a two-dimensional list, that is, return a one-dimensional list of its elements, use the following:

```
[j for row in L for j in row]
```

For instance, suppose we have the following list:

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

The flattened list will be:

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

Higher dimensions Creating and using 3-dimensional and higher lists is similar. Here we create a $5 \times 5 \times 5$ list:

```
L = [[[0]*5 for i in range(5)] for j in range(5)]
```

It is a list whose items are lists of lists. The first entry in the list is

```
L[0][0][0]
```

Exercises

1. Write a program that asks the user to enter some text and then counts how many articles are in the text. Articles are the words `'a'`, `'an'`, and `'the'`.

2. Write a program that allows the user to enter five numbers (read as strings). Create a string that consists of the user's numbers separated by plus signs. For instance, if the user enters 2, 5, 11, 33, and 55, then the string should be `'2+5+11+33+55'`.

3. (a) Ask the user to enter a sentence and print out the third word of the sentence.
   (b) Ask the user to enter a sentence and print out every third word of the sentence.

4. (a) Write a program that asks the user to enter a sentence and then randomly rearranges the words of the sentence. Don't worry about getting punctuation or capitalization correct.
   (b) Do the above problem, but now make sure that the sentence starts with a capital, that the original first word is not capitalized if it comes in the middle of the sentence, and that the period is in the right place.

5. Write a simple quote-of-the-day program. The program should contain a list of quotes, and when the user runs the program, a randomly selected quote should be printed.

Exercises

6. Write a simple lottery drawing program. The lottery drawing should consist of six different numbers between 1 and 48.

7. Write a program that estimates the average number of drawings it takes before the user's numbers are picked in a lottery that consists of correctly picking six different numbers that are between 1 and 10. To do this, run a loop 1000 times that randomly generates a set of user numbers and simulates drawings until the user's numbers are drawn. Find the average number of drawings needed over the 1000 times the loop runs.

8. Write a program that simulates drawing names out of a hat. In this drawing, the number of hat entries each person gets may vary. Allow the user to input a list of names and a list of how many entries each person has in the drawing, and print out who wins the drawing.

9. Write a simple quiz game that has a list of ten questions and a list of answers to those questions. The game should give the player four randomly selected questions to answer. It should ask the questions one-by-one, and tell the player whether they got the question right or wrong. At the end it should print out how many out of four they got right.

10. Write a censoring program. Allow the user to enter some text and your program should print out the text with all the curse words starred out. The number of stars should match the length of the curse word. For the purposes of this program, just use the"curse" words *darn*, *dang*, *freakin*, *heck*, and *shoot*. Sample output is below:

```
Enter some text: Oh shoot, I thought I had the dang problem
figured out. Darn it. Oh well, it was a heck of a freakin try.

Oh *****, I thought I had the **** problem figured out.
**** it. Oh well, it was a **** of a ****** try.
```

Exercises

11. Section 8.3 described how to use the `shuffle` method to create a random anagram of a string. Use the `choice` method to create a random anagram of a string.

12. Write a program that gets a string from the user containing a potential telephone number. The program should print `Valid` if it decides the phone number is a real phone number, and `Invalid` otherwise. A phone number is considered valid as long as it is written in the form *abc-def-hijk* or *1-abc-def-hijk*. The dashes must be included, the phone number should contain only numbers and dashes, and the number of digits in each group must be correct. Test your program with the output shown below.

```
Enter a phone number: 1-301-447-5820
Valid
Enter a phone number: 301-447-5820
Valid
Enter a phone number: 301-4477-5820
Invalid
Enter a phone number: 3X1-447-5820
Invalid
Enter a phone number: 3014475820
Invalid
```

13. Let `L` be a list of strings. Write list comprehensions that create new lists from `L` for each of the following.

    (a) A list that consists of the strings of `s` with their first characters removed

    (b) A list of the lengths of the strings of `s`

    (c) A list that consists of only those strings of `s` that are at least three characters long

14. Use a list comprehension to produce a list that consists of all palindromic numbers between 100 and 1000.

15. Use a list comprehension to create the list below, which consists of ones separated by increasingly many zeroes. The last two ones in the list should be separated by ten zeroes.

```
[1,1,0,1,0,0,1,0,0,0,1,0,0,0,0,1,....]
```

Exercises

16. Let `L=[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]`. Use a list comprehension to produce a list of the gaps between consecutive entries in `L`. Then find the maximum gap size and the percentage of gaps that have size 2.

17. Write a program that finds the average of all of the entries in a 4×4 list of integers.

18. Write a program that creates a 10×10 list of random integers between 1 and 100. Then do the following:

    (a) Print the list.

    (b) Find the largest value in the third row.

    (c) Find the smallest value in the sixth column.

19. Write a program that creates and prints an 8×8 list whose entries alternate between 1 and 2 in a checkerboard pattern, starting with 1 in the upper left corner.

20. Write a program that checks to see if a 4×4 list is a magic square. In a magic square, every row, column, and the two diagonals add up to the same value.

21. Write a program that asks the user to enter a length. The program should ask them what unit the length is in and what unit they would like to convert it to. The possible units are inches, yards, miles, millimeters, centimeters, meters, and kilometers. While this can be done with 25 if statements, it is shorter and easier to add on to if you use a two-dimensional list of conversions, so please use lists for this problem.

22. The following is useful as part of a program to play *Battleship*. Suppose you have a 5×5 list that consists of zeroes and ones. Ask the user to enter a row and a column. If the entry in the list at that row and column is a one, the program should print `Hit` and otherwise it should print `Miss`.