Lists in Python

Leigh J. Little SUNY Brockport November 9, 2020

Contents

1 Introduction

This document will discuss the list structure in Python and how to perform basic operations with it.

2 Lists

In MATLAB, the most basic non-scalar data type is an array. In Python, there is no direct equivalent to an array that is built into the language (but an `array` type can be imported from one of the libraries we will use later); rather Python works with *lists*. In some respects, a list is similar to an array, but has much more flexibility in the list elements.

Over the years, many arguments about they way computers should behave have emerged. One of the oldest is: Should indexed/subscripted variables (such as arrays and lists) start indexing at 0 or at 1? This argument is probably 60 years old and is still going on despite the fact that everything that can be said has been said. Both approaches have good and bad merits, but in the long run the issue should not be that significant of a concern. In any case, the key thing to be aware of when working with lists is that the first element of the list is element 0, not element 1 as in MATLAB. Most languages start indexing at element 0. In Fortran, indexing starts at element 1, though this can be changed to start at any element.

To generate a simple list in Python, use the [ ] constructors

```
>>> x = [4, 6, 7, -3, 2]    # Note the commas
>>> x
[4, 6, 7, -3, 2]
>>> x[0]
4
>>> x[4]
2
```

Some important items to note from the above example:

a) The commas between list elements are required.

b) The list doesn't have an orientation (row or column).

c) The elements are referenced using [ ] instead of ( ).

d) The first element in the list is `x[0]`.

e) The list has 5 elements and the last element is `x[4]`.

2.1 Example 1 - Summing the elements in a list

We can use a list of numerical values in much the same way we have used vectors in MATLAB. Enter the following in a script called `testlists.py`

```
x = [4, 6, 7, -3, 2]
n = len(x)
total = 0
for i in range(0,n):
    total = total + x[i]
print('Sum of vector = ',total)
```

Note that the `len` function is used to get the length of the list.

2.2 The `range` Function

The basic form of the range function is

```
range(n)
```

This will generate a list of `n` consecutive integers, starting at 0. Thus, `range(5)` will give the list 0,1,2,3,4. When you modify the `range` function with an optional initial argument like

```
range(1,n)
```

The list will start at 1, but still end at `n-1`. The reason for this unusual behavior is tied into the fact that the first element of a list is element 0.

3 Unusual Lists

What makes lists so flexible in Python is that an element of a list can be anything. In this aspect, a list is like a cell array in MATLAB. For example, the list `a` below is valid

```
>>> a = [4, 5.2, 'Car', [1,2,3]]
```

This list contains 4 elements. `a[0]` is the integer 4, `a[1]` is the double 5.2, `a[2]` is the string 'Car' and `a[3]` is the list [1, 2, 3].

Because lists like this commonly occur, there is an alternative method to loop over the elements in a list:

```
a = [4, 5.2, 'Car', [1,2,3]]
for i in a:
    print(i)

4
5.2
Car
[1, 2, 3]
```

Note that instead of looping over a set of integers, the index variable `i` directly assumes the value of the list elements.

An equivalent code would be

```
a = [4, 5.2, 'Car', [1,2,3]]
for i in range(0,4)
    print(a[i])

4
5.2
Car
[1, 2, 3]
```

Here, `i` assumes the values 0, 1, 2, 3 and the `print` statement displays the corresponding value of `a`.

3.1 Adding Elements to an Existing List

In MATLAB, if we have an array that we need to add elements to, we can do

```
>> b = [1 2 3 4 5]
b =
     1     2     3     4     5
>> b(6) = 6;
>> b(7) = 7
b =
     1     2     3     4     5     6     7
```

If you attempt to do this in Python, an error will be generated

```
>>> b = [1,2,3,4,5]
>>> b
[1, 2, 3, 4, 5]
>>> b[5] = 6
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    b[5] = 6
IndexError: list assignment index out of range
```

In order to add an element to an array, you need to use the `append` command.

```
>>> b = [1,2,3,4,5]
>>> b
[1, 2, 3, 4, 5]
>>> b.append(6)
>>> b
[1, 2, 3, 4, 5, 6]
```

This is important, so it will be stated again: in order to add an element to an array, you need to use the `append` command.

4 String Variables

Python has string variables, which are denoted by either single or double quotes, for example,

```
>>> s = "Today is Monday"
>>> s
'Today is Monday'
>>> len(s)
15
>>> s[0]
'T'
>>> s[13]
'a'
>>> for i in s: print(i)
T
o
d
a
y

i
s

M
o
```

```
n
d
a
y
```

String variables are basically a special type of list. One important characteristic of string variables is that they are *immutable*. This means that once they are assigned, they cannot be modified. For example, if you attempt to reassign the value of `s[10]`, an error will be generated:

```
>>> s[10] = 'q'
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    s[10] = 'q'
TypeError: 'str' object does not support item assignment
>>>
```

Otherwise they behave like any other list.

5 Example: Tabulating a Function

Tables are important in scientific computing. We will see later on that there are many ways to create a table of values of a function, but it is instructive to see how we can do this using the basic building blocks we have so far. Suppose you wanted to tabulate $y(x) = \sin(x)\cos(x)$ on the interval $x \in [0, 2\pi]$ for some user input number of table values $n + 1$.

Recall from before that the hard part is generating the x-coordinates. Once these are known, the corresponding values of y are easy to obtain. In MATLAB, we had the `linspace` command. Here we will use the formulas directly. These are given by:

$$\begin{aligned} x_i &= a + (i-1)h, \quad i = 1, 2, \ldots, n+1 \\ y_i &= f(x_i) \end{aligned}$$

where

$$h = \frac{b-a}{n}.$$

Note that these formulas are written in a MATLAB friendly form. The index variable `i` varies from 1 to $n + 1$ because the first element in MATLAB arrays is 1. Type the following into a file called `table1.py`

```
import math as m
n = int(input("Input n "))
a = 0
b = 2*m.pi
h = (b-a)/n
x = []
y = []
for i in range(1,n+2):
     temp1 = a + (i-1)*h
     temp2 = m.sin(temp1)*m.cos(temp1)
     x.append(temp1)
     y.append(temp2)
     print(temp1,temp2)
```

When run for $n = 10$,, this gives

```
Input n 10
0.0 0.0
0.6283185307179586 0.4755282581475768
1.2566370614359172 0.29389262614623657
1.8849555921538759 -0.2938926261462365
2.5132741228718345 -0.4755282581475768
```

```
3.141592653589793 -1.2246467991473532e-16
3.7699111843077517 0.4755282581475767
4.39822971502571 0.2938926261462367
5.026548245743669 -0.2938926261462364
5.654866776461628 -0.47552825814757693
6.283185307179586 -2.4492935982947064e-16
```

which is correct. There are some important items to note about this example:

- The math library needs to be imported and the `m` prefix needs to be used for `pi`, `sin` and `cos`.
- The `int` function has been used to ensure that `n` is an integer.
- The statements

  ```
  x = []
  y = []
  ```

  are necessary. These tell Python that these variables are lists. Unlike MATLAB, we can't start putting elements into lists at will. We need to explicitly tell Python this information.
- The upper limit on the `range` function is `n+2`. This is because `i` needs to assume the values `1` through `n+1`.
- `temp1` is the current value of `x` and `temp2` is the current value of `y`. These are then inserted into the `x` and `y` lists using the `append` command. It is possible to perform these calculations directly in the `append` command using a syntax like

  ```
  x.append(a+(i-1)*h)
  ```

  but the process here is less likely to lead to formula transcription errors.

6 Example: Python Friendly Tabulation

The formulas from the previous example can be written in a more Python friendly format. Because Python indexing starts at 0 instead of 1, the formulas

$$\begin{aligned} x_i &= a + ih, \quad i = 0, 1, \ldots, n \\ y_i &= f(x_i) \end{aligned}$$

make more sense. Note that this formula for `{x_i` will generate the same set of x-coordinates as the previous formula.

The new version of the program is

```
import math as m
n = int(input("Input n "))
a = 0
b = 2*m.pi
h = (b-a)/n
x = []
y = []
for i in range(0,n+1):
     temp1 = a + i*h
     temp2 = m.sin(temp1)*m.cos(temp1)
     x.append(temp1)
     y.append(temp2)
     print(temp1,temp2)
```

The only changes are to the `range` function and the formula for `temp1`.