An introduction to lists

Fabrice Rossi

Lecture notes

Python provides numerous data structures that can be used to store objects in an efficient way with respect to some specific tasks. One of the most fundamental of those types is the `list` one. Lists are used to store values in a specific order. A `list` object can evolve by storing/removing values. In addition Python provides several advanced list oriented constructions.

A list object can be specified by putting objects into brackets, as follows

```
ml = [1, 5, True, 'foo']
print(type(ml))
print(ml)
print(len(ml))
```

This program prints (as expected)

```
<class 'list'>
[1, 5, True, 'foo']
4
```

Lists are iterable and therefore can be used in *for loops*, as in the following example

```
ml = [1, 5, True, 'foo']
for k in ml:
   print(k)
```

which prints

```
1
5
True
foo
```

Exercise 1 (*Iterating over a list*)

Write a program that first create a list of integers (with specific values given in the program itself) and then computes with a `for` loop the minimal, maximal and average value of those integers. Do not use the built in Python functions `min`, `max` and `sum`.

Lecture notes

One can grow a list by appending objects to it, as in the following example

```
ml = [ ] # empty list
for k in range(5):
   ml.append(k) # adding k at the end of the list
print(ml)
```

which prints

```
[0, 1, 2, 3, 4]
```

Notice that `append` is a method with no (meaningful) return value.

Exercise 2 (*Appending to a list*)

Question 1 Using a `for` loop and list appending, write a program that asks to the user a number n and then asks to the user n strings to be stored in a list. Print the content of the list at the end of the program.

Question 2 Using a `while` loop and list appending, write a program that asks to the user to type in integer values until he/she gives a negative or null value. The program must store all the positive values in a list and print it before exiting.

Question 3 Using the `random` module (in particular the `randint` function), write a program that prints a list of a random number of random integers. The list length must be between 0 and 10, while the random integers must be between 0 and a user entered value.

Lecture notes

A fast way to create a `list` is to pass to the `list` function an iterable object as in the following example

```
ml = list(range(1,5))
print(type(range(1, 5)))
print(type(ml))
print(ml)
print(list('abcdef'))
```

which prints

```
<class 'range'>
<class 'list'>
[1, 2, 3, 4]
['a', 'b', 'c', 'd', 'e', 'f']
```

Exercise 3 (*List based calculation*)

Warning: this exercise does not provide an efficient way for computing the values specified below. The goal is to show case some list operations.

Python provides a `sum` function which computes the sum of the values contained in the parameter list (iterable).

Question 1 Using only this function and lists (no loops), write a program that calculates the following values, where n is given by the user:

$$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n$$

$$\sum_{k=0}^{n} (2k+1) = 1 + 3 + 5 + \ldots + (2n+1)$$

Question 2 Can this technique be extended to compute e.g. $\sum_{k=1}^{n} \frac{(-1)^k}{k}$?

Lecture notes

The most advanced list creation tool in Python is the *list comprehension*. It combines a `for` loop with a list syntax as in the following example

```
ml = [ x for x in range(4) ]
print(ml)
```

which prints

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

The general syntax of a (simple) list comprehension is

```
[ expression for variable in iterable ]
```

which is interpreted as follows: built a list of the values of `expression` obtained by setting in order `variable` to all the values of `iterable`. In other words,

```
ml = [ expression for variable in iterable ]
```

is roughly equivalent to

```
ml = [ ]
for variable in iterable:
   ml.append(expression)
```

Thus this program

```
print([ x + 1 for x in range(1, 6, 2)])
print([ a for a in 'a text'])
print([ 2 * y + 1 for y in [1, -2, 4, 5]])
```

prints

```
[2, 4, 6]
['a', ' ', 't', 'e', 'x', 't']
[3, -3, 9, 11]
```

Exercise 4 (*List comprehensions*)

Warning: this exercise does not provide an efficient way for computing the values specified below. The goal is to demonstrate some list operations.

Using only the function `sum` and lists (no loops), write a program that calculates the following values, where n is given by the user:

$$\sum_{k=1}^{n} \frac{(-1)^k}{k} = -1 + \frac{1}{2} + \ldots + \frac{(-1)^n}{n}$$

$$\sum_{k=0}^{n} (2k+1)^2 = 1 + 3^2 + 5^2 + \ldots + (2n+1)^2$$

Lecture notes

In addition to being iterable, Python lists are also indexable and sliceable, which means that this index operation `[]` and the associated slices `[::]` can be used on lists as in the following example

```
ml = list(range(1,6))
print(ml)
print(ml[-2])
print(ml[2])
ml2 = ml[1:3]
print(ml2)
```

which prints

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

Exercise 5 (*Indexing*)

Question 1 Write a program that ask for an integer n and generates at list of n random integers chosen between 0 and n (inclusive), using a list comprehension.

Question 2 Add to the program a `for` which prints the content of the list by iterating over a well chosen `range` object, rather than directly on the list. A typical output for $n = 4$ might be

```
0 1
1 1
2 4
3 2
```

where the first number is the index of the integer in the list and the second it the integer itself (so here, the list was `[1, 1, 4,   2]`.

Question 3 Using a well chosen slice, print the list in reverse order (as a list).

Exercise 6 (*All together*)

The goal of the exercise is to build a digital experiment about the random number generator of Python. The program shall simulate dice rolls with this generator and gather basic counts about them. More precisely write a program that performs the following tasks:

1. ask the user the integers n and p;
2. use `randint` from `random` to simulate n rolls of p standards six sided balanced dices, recording each time the *sum* of the dice values;
3. use ascii art to print a histogram of the frequencies of the sums (see below).

For instance $n = 100$ (100 rolls) and $p = 2$ (2 dices each time), a possible histogram might be

```
 2 **
 3 ******
 4 ******
 5 ***************
 6 ************
 7 *************
 8 **************
 9 ********
10 **********
11 *********
12 *****
```

Each value corresponds to one of the possible sums and the stars on its right corresponds to the number of rolls. In the above example, 12 was obtained 5 times, while 7 was rolled 13 times.