Chapter 2: Lists, Arrays and Dictionaries

1. Higher order organization of data

In the previous chapter, we have seen the concept of scalar variables that define memory space in which we store a scalar, i.e. single numbers or strings. Scalar values however are usually insufficient to deal with current data. Imagine writing a program that analyzes the whole human genome. Since it contains approx. 30,000 genes, we would have to create 30,000 variables to store their sequences! Initializing these variables would already be a daunting task, but imagine that you wanted to count the number of times the sequence ATG appears in each gene, you would have to write one line of code for each gene, hence 30,000 lines, changing only the variable name for the gene!
Fortunately, Python thinks that laziness is a virtue, and would never tolerate that you have to write 30,000 lines of code. Two special types of variables exist to help managing long lists of items, namely arrays and dictionaries. These variables store lists of data, and each piece of data is referred to as an element. In this chapter, we will look in details at what are lists, and how they are stored and manipulated within arrays and dictionaries.

We are all familiar with lists: think about a shopping list, a soccer team roster, all integers between 1 and 10, genes in the human genomes,…A list can be ordered (increasing or decreasing values for numbers, lexicographic order for strings), or unordered. Python has two types of lists, **tuples** and **lists**. Tuples are **immutable**, i.e. they cannot be modified once created, while lists are **mutable**, i.e. they can be modified once created.

2. Tuples

2.1 Tuples in Python

By definition, a tuple is a set of comma-separated values enclosed in parentheses.

Examples:

- (1,2,3,4,5,6,7,8,9,10) is the tuple of integers between 1 and 10
- (‘Monday’,’Tuesday’,’Wednesday’,’Thursday’,’Friday’,’Saturday’,’Sunday’) is the tuple containing the days in the week.

2.2 Accessing tuple values

We have seen how to build a tuple. Another thing that is useful is to be able to access a specific element or set of elements from a tuple. The way to do this is to place the number or numbers of the element (s) we want in square brackets after the tuple. Let us look at an example:

```
>>> print (‘Monday’,’Tuesday’,’Wednesday’,’Thursday’,’Friday’,’Saturday’,’Sunday’)[2]
```

Which day do you think will be printed? If you try it, you will get:

'Wednesday'

Why didn't we get "Tuesday", the second element of the list? This is because Python starts counting from 0 and not 1!! This is something important to remember.

The element you want does not have to be literal: it can be a variable as well. As an exercise, write a small program that reads in a number between 1 and 7, and outputs the corresponding day of the week. The answer is on the next page.

```
>>> # Read in day considered
>>> day=int(raw_input("Enter a number from 1 to 7 : "))
>>> #
>>> # print element 'day' of the list of days of the week:
>>> #
>>> print ('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday')[day]
```

The examples above show you how to single out one element of a tuple; if you wanted more than one element, you can "splice" the tuple, the same way we splice strings. For example,

```
>>>print ('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday')[0:2]
```

will print

('Monday','Tuesday')

Remember that the range i:j means from position i to position j, j not included.

3. Lists

A list in Python is created by enclosing its elements in brackets:

```
>>> ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]
```

Elements in a list are accessed the same way elements are accessed in tuples.

Special lists: ranges

Often the lists we use have a simple structure: the numbers from 0 to 9, or the numbers from 10 to 20. We do not need to write these lists explicitly: Python has the option to specify a range of numbers. The two examples cited would be written in Python as:

```
>>> range(10)
[0,1,2,3,4,5,6,7,8,9]
>>> range(10,21)
[10,11,12,13,14,15,16,17,18,19,20]
>>> range(1,10,2)
[1,3,5,7,9]
```

Note that lists (and tuples) in Python can be mixed: you can include strings, numbers, scalar variables and even lists in a single list!

4. Arrays

There is not much we can do with lists and tuples, except print them. Even when you print them, the statements can become cumbersome. Also, there is no way to manipulate directly a list: if we wanted to create a new list from an existing list by removing its last element, we could not. The solution offered by Python is to store lists and tuples into arrays.

4.1 Assigning arrays

Names for arrays follow the same rules as those defined for scalar variables. We store a list into an array the same way we store a scalar into a scalar variable, by assigning it with =:

```
>>> days=('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday')
```

for a tuple, or

```
>>> days=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
```

for a list.

Note: the name of an array does not indicate if it contains a list or a tuple: try to use names that are explicit enough that there are no ambiguities.

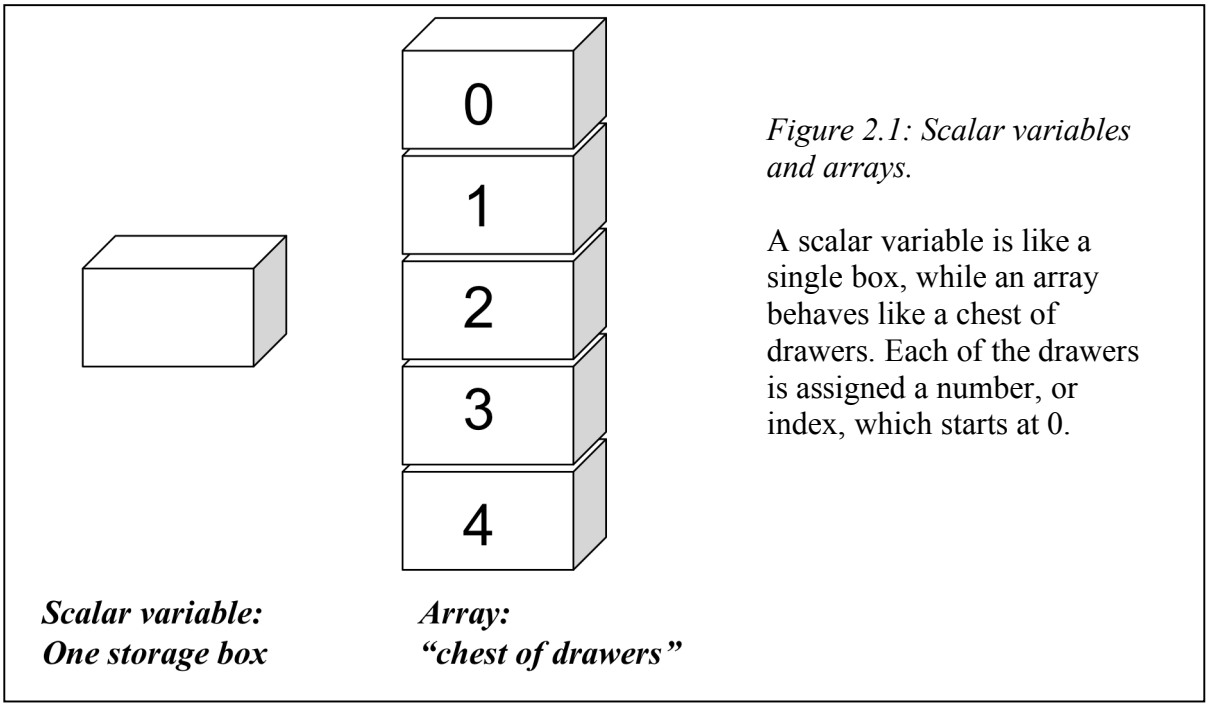

Figure 2.1: Scalar variables and arrays.

A scalar variable is like a single box, while an array behaves like a chest of drawers. Each of the drawers is assigned a number, or index, which starts at 0.

Once we have assigned a list to an array, we can use it where we would use a list. For example,

```
>>> print days
```

will print:

['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']

4.2 Accessing one element in an array

We can access one element in an array by using the index of the drawers it has been assigned to. Remember how we access an element in a list:

['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'][0]

gives us the first element in the list, i.e. 'Monday'. We could do:

days[0];

to access the first element of the array days.

Accessing an element in an array works both ways: we can either retrieve the value contained in the position considered, or assign a value to that position. For example,

```
numbers = [0,1,5];
```

Creates an array names numbers that contains the list [0,1,5]. This list can then be modified:

```
numbers[0]=3;
numbers[1]=4;
numbers[2]=5;
```

The array numbers now contains the list [3,4,5].

Important: you can change elements in a list, but you will get an error message if you try the same thing in a tuple.

4.3 Array manipulation

Python provides a list of functions that manipulates list. Let A be a list:

Type	Notation	Function
Adding values	**A.append(obj)**	Adds obj at the end of list A
	A.extend(list)	Adds list at the end of list A
	A.insert(index,item)	Adds item at position index in A, and move the remaining items to the right
Remove values	**del A[i]**	Removes element at position i in the list A
	Item=A.pop(index)	Removes object at position index in A, and stores it in variable item
	A.remove(item)	Search for item in A, and remove first instance
Reverse	**A.reverse()**	Reverses list A
Sorting	**A.sort()**	Sorts list A in place, in increasing order
Searching	**I=A.index(item)**	Search for item in list A, and puts index of first occurrence in i
Counting	**N=A.count(item)**	Counts the number of occurrence of item in A
Length	**N=len(A)**	Finds number of items in A, and stores in N

Important again: you can manipulate elements in a list, but you will get an error message if you try the same thing in a tuple.

4.4 ***From arrays to string and back***.

join: from array to string:

You can concatenate all elements of a list into a single string using the operator join. The syntax is:

```
>>> A=’’.join(LIST)
```

It concatenates all elements of LIST and stores them in the string A. Not the presence of ‘’ before join.

Split and list: from string to array

We have seen in chapter 1 that the function split will break down a string into a list, using a specified delimiter to mark the different elements. If the delimiter is omitted, split separates each word in the string. For example, if A=”This is a test”, A.split() will generate the list [“This”,”is”,”a”,”test”]. Split however cannot break down a word into characters: to do that, we use the function list.

For example, if we want to break a paragraph into sentences, a sentence into words, and a word into characters, we use:

```
>>> sentences = paragraph.split(‘.‘)
>>> words=sentence.split(‘ ‘)
>>> characters=list(word)
```

In all three cases, the result is a list and the input was a string.

5. Dictionary

5.1 Definition

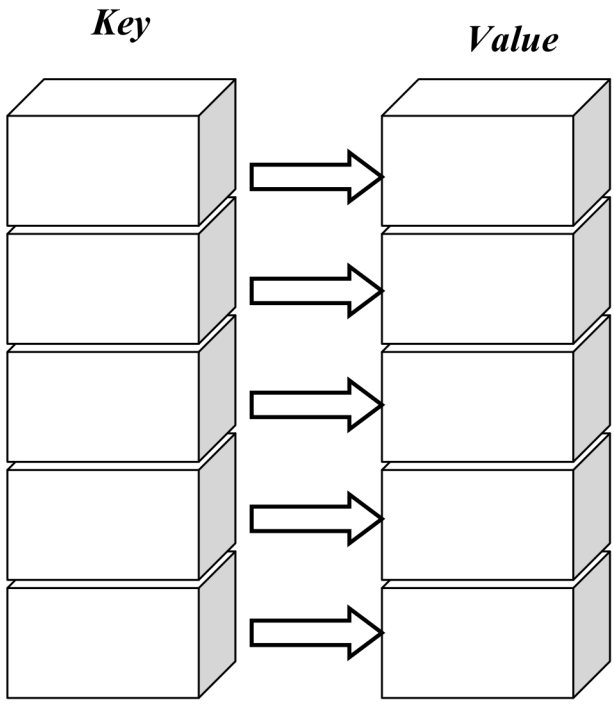

Figure 2.2: The dictionary variable

A dictionary is a special array for which each element is indexed by a string instead of an integer. The string is referred to as a key, while the corresponding item is the value. Each element of a dictionary is therefore a pair (key,value).

Arrays are very good for maintaining and manipulating lists. They have one limitations however: each element in an ARRAY is indexed with an integer value, varying from 0 to its last index, len(ARRAY). You can think about it as the array representing a street of houses, with each house defined by its number. There are instances however in which it would be more convenient to refer to the different houses using the name of its inhabitants: this is exactly what a dictionary variable does (see figure 2.2). Dictionaries are also referred to as associative arrays or hashes.

5.2 *Assigning values to dictionaries*

A dictionary is a set of pairs of value, with each pair containing a key and an item. Dictionaries are enclosed in curly brackets. For example:

```
>>> country = { “Paris”:”France”,  “Washington”: “USA”, “London”:”England”,
“Ottawa”:”Canada”,“Beijing”:”China”}
```

Creates a dictionary of countries with their capitals, where the capitals serve as keys.

Note that keys must be unique. If you try to add a new entry with the same key as an existing entry, the old one will be overwritten. Dictionary items on the other hand need not be unique.

Dictionaries can also be created by **zipping** two tuples:

```
>>> seq1=(“Paris”,”Washington”,”London”,”Ottawa”,”Beijing”)
>>> seq2=(“France”,”USA”,”England”,”Canada”,”China”)
>>> d = dict(zip(seq1,seq2))
>>> d
{ “Paris”:”France”,  “Washington”: “USA”, “London”:”England”, “Ottawa”:”Canada”,
“Beijing”:”China”}
```

5.2 Accessing elements in dictionaries

This is similar to looking inside an array, except that the positions in the dictionaries are indexed by their keys, and not by an integer index. Using the dictionary d defined above,

```
>>> d[“Paris”]
“France”
>>> d[“Beijing”]
“China”
```

If you give a key however that is not in the dictionary, Python will output an error message.
To circumvent this problem, it is often better to use the function get: if key does not exist, Python will output “None”, or an error message that you have defined:

```
>>> d.get(“Paris”)
“France”
>>> d.get(“Mexico City”)
>>> d.get(“Mexico City”,”This key is not defined”)
‘This key is not defined’
```

5.3 Manipulating dictionaries

Adding new key-value pairs to a dictionary is simply done by assignment. For example, we could have added Germany and Mexico in our dictionary d using:

```
>>> d[“Berlin”]=”Germany”
>>> d[“Mexico City”]=”Mexico”
```

We can also change the entries in a dictionary just by reassigning them.

To remove an entry in a dictionary, we need to use the function **del**. For example, to remove the key-value (“Paris”:”France”) from our dictionary d:

```
del d[“Paris”]
```

Other useful functions on dictionaries are illustrated in the example below:

```
>>> d = {“A”:”California”,”B”:”Nevada”,”C”:”Oregon”}
>>>
>>> d.keys()                          # list all keys of d
[‘A’,’B’,’C’]
>>>
>>> d.values()                        # list all values of d
[‘California’,’Nevada’,’Oregon’]
>>>
>>> d.has_key(‘A’)                    # check if a given key is known in d
True
>>>d.has_key(‘D’)
False
```

Exercises:

1. Write a program that prints all the numbers from 1 to 100. Your program should have much fewer than 100 lines of code!

2. Starting with the word GENE1=”ATGTTGATGTG”, write a Python program that creates the new words GENE2, GENE3, GENE4 and GENE5 such that:
    i. GENE2 only contains the last two letters of GENE1
    ii. GENE3 only contains the first two letters of GENE1
    iii. GENE4 only contains the letters at positions 2,4,6,8 and 10 in GENE1
    iv. GENE5 only contains the first 3 and last 3 letters of GENE1

3. Suppose you have a Python program that read in a whole page from a book into an array PAGE, with each item of the array corresponding to a line. Add code to this program to create a new array SENTENCES that contains the same text, but now with each element in SENTENCES being one sentence.

4. Let d be a dictionary whose pairs key:value are country:capital. Write a Python program that prints the keys and values of d, with the keys sorted in alphabetical order. Test your program on
   d = {“France”:”Paris”,”Belgium”:”Brussels”,”Mexico”:”Mexico City”,”Argentina”:”Buenos Aires”,”China”:”Beijing”}