

Inf2a: Lab 2

Introduction to Python - Part 2

Paolo Besana, Laura Hutchins-Korte and Bonnie Webber

Week 3: 5–9 October 2009

- ▷ *Open a terminal window, and go to the folder “MyPython” that you should have created during the first lab.*
- ▷ *Launch your preferred editor (emacs, vim, kate,...) in the background (using &)*
- ▷ *launch the python shell*

1 More on Object Oriented Programming

Classes can have an initialisation method `__init()` similar to the class constructor in Java. This method is called when the class is instantiated and can have a set of parameters. In contrast with Java, which can have many different constructors, python allows only one such method per class.

```
▷ Type:
>>> class Greeter:
...     """A simple class"""
...     def __init__(self, greeting):
...         self.greeting = greeting
...     def greet(self, name):
...         return self.greeting + ", " + name
...
>>> c2 = Greeter("hi")
>>> c2.greet("tim")
```

A class can derive from another class:

```
▷ Type:
>>> class GreeterEx(Greeter):
...     """A derived class"""
...     def bye(self):
...         return "Bye Bye"
...
>>> c3 = GreeterEx('hello')
>>> c3.greet('mr smith')

>>> c3.bye()
```

This class will contain the methods defined in `Greeter`, plus the new `bye()` method.

EXERCISE 1

- ▷ Using your preferred editor, create a class that checks if a string (`infix`) is contained in other strings. The class must be initialised by passing the `infix` string, which must be stored in a class variable. The class must expose the method `check(string)` that verifies if `infix` is contained in the passed string. (You can use the operator `in` to verify if one string is contained in another one: `string1 in string2`.)

Remember to use `self` when trying to access to class methods and attributes.

- ▷ Import your module into python, and test its behaviour. (You must instantiate the class passing the `infix` string, and then call the method `check(string)` on different strings, to verify that it works as intended.)

If you use the statement `import modulename`, remember to use the `modulename` prefix in front of the class name. If you make an error in the class, and you need to reimport the module, use `reload(modulename)`: `import` will not reimport a module already imported. You will also have to re instantiate the class.

2 More on Data Types

2.1 Lists

It is possible to create nested lists:

- ▷ *Type:*

```
>>> L1 = [1, 2, 3]
>>> L2 = ['one', L1, 'two']
>>> L2
```

We have already seen the method `append()` for the list data type in the previous lab. We will see some other methods today:

- ▷ `insert(i,x)`
insert the item `x` at position `i` (where 0 is to the left of the first item, 1 is to the left of the second item, etc).

Type:

```
>>> L2 = ['a', 'b', 'd', 'e']
>>> L2.insert(1, 'c')
>>> L2
```

- ▷ `remove(x)`
removes the first item in the list whose value is `x`.

Type:

```
>>> L2.remove('d')
>>> L2
```

- ▷ `pop([i])`
returns (and removes) the last item in the list (or the item in position `i`).
Type:

```
>>> L2.pop()
>>> L2
```
- ▷ `sort()`
sort the items of the list, in place
Type:

```
>>> L2.sort()
>>> L2
```
- ▷ `reverse()`
reverse the elements of the list, in place.
Type:

```
>>> L2.reverse()
>>> L2
```
- ▷ `del`
can be used to remove items from a list using the index. It can also be used to remove slices from a list.
Type:

```
>>> del L2[1:3]
>>> L2
```

You can iterate over a list retrieving the index and the value at the same time using the `enumerate(list)` function.

- ▷ *Type:*

```
>>> for i, v in enumerate(['a', 'b', 'c', 'd']):
...     print i, v
```

EXERCISE 2

- ▷ Using your preferred editor, create a class named `Queue` that models a queue: the first element that enters is the first that exits (FIFO: First In, First Out). The class will use a list to maintain the data. It will expose the following methods:
 - ▷ `isempty()`: verifies if the queue is empty
 - ▷ `push(item)` inserts an element at the end of the queue
 - ▷ `pop()`: extracts and returns the first element in the queue (possibly only if the queue is not empty)
- ▷ Import the module into the python shell, and test it

Remember to create the list that contains the data before accessing it.

EXERCISE 3

- ▷ Using your preferred editor, create a class named **Stack** that models a stack: the last element that enters is the first that exits (LIFO: Last in, First Out). The class will use a list to maintain the data. It will expose the following methods:
 - ▷ `isempty()`: verifies if the stack is empty
 - ▷ `push(item)`: inserts an element at the end of the stack
 - ▷ `pop()`: extracts and returns the last element of the stack (possibly only if the stack is not empty)
- ▷ Import the module into the python shell, and test it

Remember to create the list that contains the data before accessing it.

Tuples

A *tuple* is composed of values separated by commas, and enclosed by parentheses.

- ▷ *Type:*

```
>>> T = (1,2,'three')
>>> T

>>> T[2]
```

Tuples can be nested.

- ▷ *Type:*

```
>>> T1 = (1,2,(3,5))
>>> T1
```

Tuples, like strings (but unlike lists), are immutable and can not be changed once created. If you try, you will get an error message.

- ▷ *Type:*

```
>>> s = "hello"
>>> s[1] = "u"

>>> T1[2] = 3
```

Dictionaries

A *dictionary* is indexed by a key. A key can be any immutable object (number, string, tuple). You cannot use a list to index a dictionary, since a list is mutable.) A *dictionary* can be seen an unordered set of *key:value* pairs, with the constraint that each key must be unique. (Values need not be unique.) The main operations performed on dictionaries are storing and retrieving values by their keys.

```

▷ Type:
>>> num = {'one':1, 'two':2, 'three':3, 'four':4}
>>> num['three']

>>> num

```

You can easily add a new item to the dictionary:

```

▷ Type:
>>> num['five'] = 5
>>> num

```

You can delete one item from the dictionary using the built in function `del(item)`:

```

▷ Type:
>>> del(num['three'])
>>> num

```

To list all the keys from a dictionary, you use the method `keys()`. To check if a key belongs to the dictionary you use the method `has_key()`.

```

▷ Type:
>>> num.has_key('one')

>>> num.keys()

```

You can iterate over a dictionary, retrieving the keys and their corresponding values using the method `iteritems()`

```

▷ Type:
>>> for k, v in num.iteritems():
...     print k,v
...

```

EXERCISE 4

- ▷ Create a class for managing a phone book. The user must be able to:
 - ▷ insert a name and their phone number,
 - ▷ obtain a number when given a name,
 - ▷ verify if name is in the phone book,
 - ▷ list all the names and phone numbers in the phone book,
 - ▷ delete a name from the phone book
 - ▷ as *optional feature*, the user should be able to print in alphabetical order the names and their phone numbers
- ▷ Import your class into the python shell, and test it. (Remember to instantiate the class.)

Use a dictionary to store the data, and remember to create the dictionary before using it. You can use the method `keys()` to obtain the list of all keys. Then you can apply any method available for lists on the list you have obtained.

3 Pattern matching

The `re` module provides a tools for regular expressions.

▷ *Type:*

```
>>> import re
```

▷ `match(pattern, string)`

If zero or more characters *at the beginning* of `string` match the regular expression `pattern`, `match` returns a corresponding `MatchObject` instance. It returns nothing if the pattern does not match the start of the string.

Type:

```
>>> re.match("(aa|bb)+", "aabbaa")
```

```
>>> re.match("(aa|bb)+", "abba")
```

▷ `search(pattern, string)`

If zero or more characters *anywhere* in `string` match the regular expression `pattern`, `search` returns a corresponding `MatchObject` instance. It returns nothing if the pattern does not match anywhere in the string.

Type:

```
>>> re.search("(aa|bb)+", "aabbaa")
```

```
>>> re.search("(aa|bb)+", "abba")
```

▷ `findall(pattern, string)`

It returns a list of all non-overlapping matches of `pattern` in `string`.

Type

```
>>> re.findall("[a-z]*th[a-z]*", "I think this is the right one")
```

```
>>> re.findall("(aa|bb)", "aabbaa")
```

```
>>> re.findall("(aa|bb)+", "aabbaa")
```

▷ `sub(pattern, repl, string)`

It returns the string obtained by replacing the leftmost non-overlapping occurrence of `pattern` in `string` by the replacement `repl`.

Type:

```
>>> re.sub("[a-z]*th[a-z]*", "TH-word", "I think this is the right one" )
```

EXERCISE 5

- ▷ Create a regular expression that checks if a string starts with 3 binary digits. Test it: `010asda` must be recognised, while `1aa` must be rejected.
- ▷ Using a regular expression, write a python statement that finds all the words that end with “ly” in strings. Test it using, for example, the string ‘‘it is likely to happen rarely’’.
- ▷ Using a regular expression, write a python statement that replaces all the words that start with “wh” with “WH-word”. Test it using, for example, the string ‘‘who should do what?’’.

4 Passing parameters

It is possible to pass parameters to a script.

▷ *create in your editor a file named `test.py`*

▷ *exit from the python shell*

▷ *Type in the editor:*
`import sys`

```
    for arg in sys.argv:  
        print arg
```

▷ *Save the file*

▷ *Type in the shell:*
`python test.py these are the arguments`

The arguments are stored in the variable `sys.argv`, that is a list of string. `sys.argv[0]` contains the name of the script, while the following elements contains the arguments.