# Imperative programming with Python
Class #7

Facundo Carreiro

ILLC, University of Amsterdam

January 2015

## Basic File I/O
### Reading

- *File objects* are values representing files.
- We use the `open` function to read/write from files

```
>>> f = open('file.txt')
>>> print f
<open file 'file.txt', mode 'r' at 0x630b0>
```

  mode `'r'` means that the file is open for *reading*.
- We can read the file line by line using the `readline` method.

```
>>> f.readline()
'First line\n'
```

- The object remembers our position in the file, if you call `readline` again you get the following result

```
>>> f.readline()
'Second line\n'
```

# Basic File I/O

## Writing

- We use `open` with the `'w'` mode to write to files

```
>>> f = open('out.txt', 'w')
```

- The `write` method...well, writes to the file

```
>>> f.write('first thing')
>>> f.write('second thing')
```

- Always close a file when you are done with it. *Both* if you were reading or writing.

```
>>> f.close()
>>> print f
<closed file 'out.txt', mode 'w' at 0x632f0>
```

- *Note*: if we check out newly created file we see

```
first thingsecond thing
```

Newlines should be added explicitly! (with `\n`)

## Serialization

- *Serialization* is the process of translating data structures or object state into a format that can be reconstructed later in the same or another computer environment.
- Suppose you have a dictionary and you want to
  1. Send it through the internet, or
  2. Store it in a file, etc.
- One posibility is to go through each `(key, value)` pair and encode the dictionary as something like `k1,v1,k2,v2,...` or `k1,v1|k2,v2|...`
- What if we use `,` or `|` in some key or value?
- What if some value `vi` is also a dictionary?
- ...it is a pain in the 'back'. It gets worse with more complex types.

## Serialization

### The pickle module

- Python comes with several libraries to serialize objects! for example, the modules `marshal`, `json`, and `pickle`.
- They are used in a similar way (check the documentation)
- `pickle` is the most common, and works as follows

```
>>> import pickle
>>> s = pickle.dumps({'k1':1, 'k2':[1,2,3]})
>>> s
"(dp0\nS'k2'\np1\n(lp2\nI1\naI2\naI3\nasS'k1'\np3\nI1\ns."
>>> pickle.loads(s)
{'k2': [1, 2, 3], 'k1': 1}
```

## System arguments

- When executing a program you can also pass arguments to it

```
python program.py argument1 argument2
```

- This can be useful in situations without human interaction such as:
  - Servers (without keyboard)
  - One program calling another program
- You can use the `sys` module to access the aruments

```
import sys
print sys.argv
```

save that as '`program.py`' and execute

```
python program.py argument1 argument2
```

## System arguments

- You will get the complete list of arguments

```
['program.py', 'argument1', 'argument2']
```

- The first one is always the name of the program
- The rest, if they exist, are the other arguments
- You can get the number of arguments using

```
len(sys.argv)
```

  and access each one using

```
sys.argv[0]
sys.argv[1]
...
```

- More complex parsing in provided by the `argparse` module.

## References

- Chapters 8 and 9 of the book
  http://greenteapress.com/thinkpython/thinkpython.html
- String methods
  http://docs.python.org/library/stdtypes.html#string-methods
- File objects
  http://docs.python.org/library/stdtypes.html#file-objects
- The `sys` module
  https://docs.python.org/2/library/sys.html
- The `pickle` module
  https://docs.python.org/2/library/pickle.html
- The `argparse` module
  https://docs.python.org/3/library/argparse.html