Chapter 1

Python support

Supports multiple programming paradigms, including object-oriented, procedural and functional styles.

- Proper indentation is a must. Blocks are represented with indentation, there is no need to use curly braces.
- You don't need to define the type of variables.
- You don't need to add semicolon at the end of the statements.
- Python is case sensitive
- a minimum set of Good Practices are mentioned in 1.1

1.1 Running python

- Interactively via an interpreter $*python*
- Execute a script - call from the command line $*python your_file.py*

We recommed the use of an IDE PyCharm or LilClipse. Students have free registration for PyCharm.

1.2 Basics

Main operators and built-in data types which you are going to use during the lab are mentioned in tables 1.2, 1.3, 1.4, 1.5 . For an exhaustive list, go to `https://www.tutorialspoint.com/python/index.htm` or `https://docs.python.org/2/tutorial/datastructures.html`

h

Table 1.1: Good practice

Variable, functions, methods, packages and modules	*lower_case_with_underscores*
Classes and exceptions	*CapWords*
Protected methods and internal functions	*_single_leading_underscore(self, ...)*
Constants	*ALL_CAPS_WITH_UNDERSCORE*

Table 1.2: Operators

Arithmetic	$+, -, *, /, \%. **$	$?2 * *3$ 8
Comparison	$==, ! =, <, >, >=, <=$	
Assignment	$=, + =, - =, * = ...$	$?a = 2$ $?a, b = 2,$ “$Hello''$ (multiple assignments)
Logical	*and or not*	

Table 1.3: Data types

Numbers	int, flat and complex classes
Strings	marked with single quotes or double quotes
Lists []	on ordered sequence of items which do not need to be of the same type) $?a = [3,' an', [3, 5]]$ $?a[0]$ 3 $?a[-1]$ $[3, 5]$
Tuples ()	an ordered sequence of items same as list; tuples once created cannot be modified $?a = (3,' an',' 10')$
Set {}	on ordered collection of unique items $?a = \{1, 2, 3, 0\}$
Dictionary	an unordered collection of ke-value pairs $?d = \{1 :' one', 2 :' two'\}$ $d[1] =' one'$
Observation: slicing operator [ ] works for string, lists and tuples.	

1.3 Exercises

1. Try basic operation in python

   (a) Invoke the interpreter in a terminal

   ```
   $ python
   ```

   (b) Try basic operators

   ```
   >>>1 + 3
   4
   >>> "IIA" == "iia"
   False
   >>> 'iia' + ' 3rd year'
   'iia 3rd year'
   >>> len('iia')
   3
   >>> s = 'iia'
   >>> print s
   >>> print "variable s has the value %s " % s
   variable s has the value iia
   >>> a,b = 2,5
   >>> a+b
   7
   ```

Table 1.4: Main operators on lists

len() Length	$?len([1,2,'3'])$ 3
Membership *in* , *not in*	$?a = [1,2]$ $?1\ in\ a$ True
Concatenation: +	$?a, b = [1,2],[3,4,5]$ $?b + a$ $[3,4,5,1,2]$
Negative indexing from back of the list:	$?a = [1,2,3]$ $?a[-1]$ 3
Slice operator for adjacent elements *list*[*start*, *stop*]	$?a = [1,2,3,4]$ $?a[1:3]$ [2, 3]

```
>>> (a,b) = (2, "iia")
>>> a
2
>>> b
'iia'
```

(c) Built-in data structures: test operators and methods mentioned in 1.4

```
>>> roles = ['arya', 'jon', 'daenerys', 'bran']
>>> roles[1]
'jon'
>>> roles[-2]
'daenerys'
>>> roles[2:]
['daenerys', 'bran']
>>> new_roles = ['cersei', 'sansa', 'tyrion']
>>> roles + new_roles
['arya', 'jon', 'daenerys', 'bran', 'cersei', 'sansa', 'tyrion']
>>> roles
['arya', 'jon', 'daenerys', 'bran']
>>> roles.append(new_roles)
>>> roles
['arya', 'jon', 'daenerys', 'bran', ['cersei', 'sansa', 'tyrion']]
>>> roles.remove(new_roles)
>>> roles
['arya', 'jon', 'daenerys', 'bran']
>>> roles.extend(new_roles)
>>> roles
['arya', 'jon', 'daenerys', 'bran', 'cersei', 'sansa', 'tyrion']
>>> roles.pop()
'tyrion'
>>> roles
['arya', 'jon', 'daenerys', 'bran', 'cersei', 'sansa']
>>>
```

Table 1.5: Main methods on lists

Name	Description	Example
$pop()$	Remove an elements from list	$?a = [1,' a',' three']$ $?a.pop()$ $'three'$ $?a$ $[1,' a']$
$reverse()$	Reverses items from list in place	$?a = ['start',' a',' three']$ $?a.reverse()$ $?a$ $['three',' a',' start']$
$sort()$	Sorts items from list in place	$?a = ['start',' a',' three']$ $?a.sort()$ $?a$ $['a',' start',' three']$
$count()$	Count of how many times obj occurs in list	$?[1, 2, 3, 2].count(2)$ 2
$index()$	The lowest index in list that obj appears	$?[1, 2, 3, 2].index(2)$ 1
$extend()$	Appends the contents of seq to list	$?a = [1, 2, 3]$ $a.extend([4])$ $?a$ $[1, 2, 3, 4]$
$append()$	Appends object to list	$?a = [1, 2, 3]$ $?a.extend([4])$ $?a$ $[1, 2, 3, [4]]$

```
Dictionaries
>>> actors = {'tyron':"peter", "daenerys":"emilia", "jaime":"nikolaj"}
>>> actors["tyron"]
'peter'
>>> actors.keys()
['daenerys', 'tyron', 'jaime']
>>> actors.values()
['emilia', 'peter', 'nikolaj']
>>> actors.items()
[('daenerys', 'emilia'), ('tyron', 'peter'), ('jaime', 'nikolaj')]
>>> actors.items()[1]
('tyron', 'peter')

>>> dict = {"two:":2, "zero":0,"three":3}
>>> dict["four"] = 4
>>> dict
{'four': 4, 'zero': 0, 'two:': 2, 'three': 3}
>>> min(dict, key = dict.get)
'zero'
```

(d) Control structures

```
>>> a = [("a", 1), ("b", 2), ("c", 3)]
>>> for pair in a:
      print "The pair is ", pair
      (what, value) = pair
      print "Elements of the pair are: ", what, " ", value

The pair is  ('a', 1)
Elements of the pair are:  a   1
The pair is  ('b', 2)
Elements of the pair are:  b   2
The pair is  ('c', 3)
Elements of the pair are:  c   3

>>>if (1==1):
       print "True"
   else:
       print "False"
```

2. **Pacman** framework - watch some movies with Pacman helped by search algorithm. At the end of the first four labs you will be able to solve similar problems.
   - why do we need search? `http://cs-gw.utcluj.ro/~anca/iia/why_search.ogv`,
   - different layouts solved with different search algorithms `http://cs-gw.utcluj.ro/~anca/iia/examples.ogv`,
   - comparison of more strategies on the same layout `http://cs-gw.utcluj.ro/~anca/iia/comparison_search_strategies.ogv`,
   - pacman and more ghosts `http://cs-gw.utcluj.ro/~anca/iia/multipacman.ogv`.

3. **Practice Python and autograder** Your activity from the first 4 laboratories can be auto-tested by using an autograder. It is possible to have more question for one exercise and for each question there are more test cases. The original projects are taken from `http://ai.berkeley.edu/project_overview.html`

   Copy the files for tutorial autograder from `https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/tutorial/v1/001/tutorial.zip` into your folder. The folder tutorial contains more files from which you need to modifiy only the files `addition.py`, `buyLotsOfFruit.py` and `shopSmart.py`.

   (a) Run the autograder and observe the results before any change to the files

   ```
   $python autograder.py
   ```

   (b) **Question 1** Change the function from `addition.py` such that it returns the addition of the two parameters. Run again the autograder and check that at Question 1 you get 1 point from 1.

   (c) **Question 2** Change the function *buyLotsOfFruit* in `byLotsOfFruit.py` such that it returns the total cost of he order.
   - Run

     ```
     $python buyLotsOfFruit.py
     ```

   - Run again the autograder and check that at Question 2 you get 1 out of 1 points.

(d) **Question 3** Class `shop.py` describes the main methods for accessing the cost per pound for a fruit and the price of the entire order (similar to teh previous question, but in an object oriented manner). There are more Fruit Shops and all items of an order must be bought from the same shop.

Change the function *shopSmart* from `shopSmart.py` such that it returns the Fruit-Shop where the order costs the least amount. Don't change `shop.py`.

Run

```
$python shopSmart.py
```

Run again the autograder and check that at Question 3 you get 1 out of 1 points.