Introduction into Python

Python 1: Examples, Variables, Scalars and Lists

Daniel Polani

Why Python?

Properties:

- minimalistic syntax
- powerful
- high-level data structures built in
- scripting, rapid applications, and — as we will see — AI
- widely in use (a plus for you)

What we will mainly need:

- flexible data and algorithm structures
- functionality
- procedurality

Where to get Python and Learn More?

Web Site: `http://www.python.org/` (also main source for this introduction)

UH Availability: in PC labs A,B,C

Introduction: Remarks

Note:

- you will be taught the essentials for it to be useful and to solve the problems posed later
- intro is not meant to be complete
- read for yourselves if you want to learn more (there's no way around it)

Invoking Python

- To start: type `python` in command line
- it will look like

```
Python 2.4 (#1, Mar  8 2005, 19:08:08)
[GCC egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- you can now type commands in the line denoted by >>>
- To leave: type end-of-file character (`ctrl-D` on Unix, `ctrl-z` on Windows
- We call this: *interactive mode*

Appetizer Example

Task: Print all numbers in a given file

File:

numbers.txt

```
2.1
4.2
3.3
5.2
6.4
```

Code:

print.py

```
# Note: the code lines begin in the first column of the file.  In
# Python code indentation *is* syntactically relevant. Thus, the
# 'hash' # (which is a comment symbol, everything past a hash is
# ignored on current line) marks the first column of the code

data = open("numbers.txt", "r")

for d in data:
    print d

data.close()
```


Appetizer Example

Task: Print the sum of all the data in the file

sum.py

```
data = open("numbers.txt", "r")

for d in data:
    print d

print sum(float(d) for d in data)

data.close()
```


Appetizer Example

Task: Print the sum of all the data in the file (different way)

sum1.py

```
data = open("numbers.txt", "r")

s = 0

# What happens if you uncomment this one?
# for d in data:
#    print d

for d in data:
    s += float(d)

print s

data.close()
```

Bottom Line

Lessons:

1. easy access to data files
2. natural treatment of data sequences
3. variables come into existence when used

Note: Python can be used interactively! This makes it much easier to test programs and find mistakes

Example

Interaction:

Input

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
...
Be careful not to fall off!
```

Lessons:

- prompt >>> allows to enter command
- command is ended by newline
- variables (`the_world_is_flat`) need not be initialized or declared
- a colon "`:`" opens a *block*
- `...` prompt denotes that block is expected
- a block is indented
- by ending indentation, block is ended

Notation

Notation:

- >>> prompt means, your input
- `...` prompt means, indentation (block) expected or possible
- no prompt means, python output

Differences to Java or C

- interactive work possible
- no declaration of variables
- dynamic typing
- no brackets denote block, just indentation
  - this needs getting used to
  - a good editor (e.g. emacs) supports the style

Comments

Remark: a comment begins with a #. Everything after that is ignored

Code

```
# this is a comment
foobar = 1 # this is setting 'foobar' to 1, followed by a comment
mystring = "# not a comment, just a hash"
```


Interactive Use

Input

```
>>> 2 + 2
```

```
4
```

Input

```
>>> (50 - 5*6)/4
```

```
5
```

Input

```
>>> 7/3 # integer division returns floor
```

```
2
```

Input

```
>>> 7/-3
```

```
-3
```

Input

```
>>> 7.0 / 2 # floating point is recognized dynamically
```

```
3.5
```

Input

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
```

```
900
```

Input

```
>>> x = y = z = 0
```


Strings

Input

```
>>> 'spam eggs'
```

```
'spam eggs'
```

Input

```
>>> 'doesn\'t'
```

```
"doesn't"
```

Input

```
>>> "doesn't"
```

```
"doesn't"
```

Input

```
>>> '"Yes," he said.'
```

```
'"Yes," he said.'
```


Strings II

Input

```
>>> """Here you can do
... everything"""
```

```
'Here you can do\neverything'
```

Input

```
>>> "\"Yes,\" he said."
```

```
'"Yes," he said.'
```

Input

```
>>> '"Isn\'t," she said.'
```

```
'"Isn\'t," she said.'
```

Strings III

Input
```
>>> 'a' + 'b'
```

```
'ab'
```

Input
```
>>> 'and again ' * 5
```

```
'and again and again and again and again and again '
```

Input
```
>>> '   lost all spaces   '.strip()
```

```
'lost all spaces'
```


Strings IV

Input
```
>>> mystring = 'abcd'
>>> mystring[1]
```

```
'b'
```

Input
```
>>> mystring[0:2]
```

```
'ab'
```

Input
```
>>> mystring[2:4]
```

```
'cd'
```


Strings V

Input
```
>>> mystring[4]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

Input
```
>>> mystring[:2]
```

```
'ab'
```

Input
```
>>> mystring[2:]
```

```
'cd'
```


Strings VI

Note: individual characters in a Python string cannot be changed

Input
```
>>> mystring[0] = 'z'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
```

Instead use:

Input
```
>>> 'z' + mystring[1:]
```

```
'zbcd'
```

Strings VII

Input
```
>>> mystring[2:1]
```
```
''
```
Input
```
>>> mystring[-1]
```
```
'd'
```
Input
```
>>> mystring[-2]
```
```
'c'
```
Input
```
>>> mystring[-2:]
```
```
'cd'
```
Input
```
>>> mystring[:-2]
```
```
'ab'
```


Strings VIII

Input
```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
```
```
34
```
Input
```
>>> len(mystring[1:3])
```
```
2
```


Lists

Input
```
>>> mylist = ['foo', 'bar', 3, 1.5]
>>> mylist
```
```
['foo', 'bar', 3, 1.5]
```
Input
```
>>> mylist[1] # counting starts from 0
```
```
'bar'
```
Input
```
>>> mylist[-1] # negative index from end of list
```
```
1.5
```
Input
```
>>> mylist[1:-1] # sublist from second element to second last
```
```
['bar', 3]
```
Input
```
>>> mylist[:2] + ['bingo', 2*2] # concatenate and evaluate
```
```
['foo', 'bar', 'bingo', 4]
```
Input
```
>>> 3 * mylist[:2] # list with 3 copies of mylist
```
```
['foo', 'bar', 'foo', 'bar', 'foo', 'bar']
```


Lists II

Input
```
>>> mylist[2] += 2 # it is possible to change elements
>>> mylist
```
```
['foo', 'bar', 5, 1.5]
```
Input
```
>>> mylist[:2] = ['foobar'] # or even to replace sublists
>>> mylist
```
```
['foobar', 5, 1.5]
```
Input
```
>>> mylist[1:1] = ['foo', 'bar'] # or to insert
>>> mylist
```
```
['foobar', 'foo', 'bar', 5, 1.5]
```
Input
```
>>> len(mylist) # len works also for lists
```
```
5
```

Lists III

Input

```
>>> innerlist = ['in', 'ner']
>>> mylist[3] = innerlist # one can nest lists
>>> mylist
```

```
['foobar', 'foo', 'bar', ['in', 'ner'], 1.5]
```

Input

```
>>> mylist[3]
```

```
['in', 'ner']
```

Input

```
>>> mylist[3].append('list') # and operate on the inner lists
>>> mylist
```

```
['foobar', 'foo', 'bar', ['in', 'ner', 'list'], 1.5]
```

Input

```
>>> innerlist # mylist[3] and innerlist are the same object
```

```
['in', 'ner', 'list']
```