Python in a Nutshell
Overview of the Basic Concepts

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it

Python's Whys & Hows
What is Python

Python is a general-purpose high-level programming language.
- it pushes code readability and productivity;
- it best fits the role of scripting language.

Python supports multiple programming paradigms
- imperative (functions, state, ...);
- object-oriented/based (objects, methods, inheritance, ...);
- functional (lambda abstractions, generators, dynamic typing, ...).

Python is
- interpreted, dynamic typed and object-based;
- open-source.

Python's Whys & Hows
How to Use Python

We are considering Python 3+
- versions >3 is incompatible with previous versions;
- version 2.6 is the current version.

A python program can be:
- edited and run through the interpreter

```
[23:31]cazzola@ulik:~/esercizi-pa>vim hello.py
[23:32]cazzola@ulik:~/esercizi-pa>cat hello.py
print("Hello World!!!")
[23:32]cazzola@ulik:~/esercizi-pa>python3 hello.py
Hello World!!!
[23:32]cazzola@ulik:~/esercizi-pa>
```

- edited in the python shell and executed step-by-step by the shell.

```
[23:34]cazzola@ulik:~/esercizi-pa>python3
Python 3.1.1 (r311:74480, Aug 17 2009, 21:52:33)
[GCC 4.4.1] on linux2
Type help, copyright, credits or license for more information.
>>> print("Hello World!!!")
Hello World!!!
>>>
[23:35]cazzola@ulik:~/esercizi-pa>
```

Python's Whys & Hows
How to Use Python (Cont'd)

The python shell can be used to get interactive help.

```
[23:35]cazzola@ulik:~/esercizi-pa>python3
Python 3.1.1 (r311:74480, Aug 17 2009, 21:52:33)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.1!  This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help> ^D
You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> ^D
[23:40]cazzola@ulik:~/esercizi-pa>
```

Overview of the Basic Concepts
Our First Python Program

humanize.py

```
SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'],
            1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']}

def approximate_size(size, a_kilobyte_is_1024_bytes=True):
    ''' Convert a file size to human-readable form. '''
    if size < 0:
        raise ValueError('number must be non-negative')
    multiple = 1024 if a_kilobyte_is_1024_bytes else 1000
    for suffix in SUFFIXES[multiple]:
        size /= multiple
        if size < multiple:
            return '{0:.1f} {1}'.format(size, suffix)
    raise ValueError('number too large')

if __name__ == '__main__':
    print(approximate_size(1000000000000, False))
    print(approximate_size(1000000000000))
```

Running the program:

```
[16:00]cazzola@ulik:~/esercizi-pa>python3 humanize.py
1.0 TB
931.3 GiB
[16:01]cazzola@ulik:~/esercizi-pa>
```

Overview of the Basic Concepts
Declaring Functions

Python has function

- no header files à la C/C++
- no interface/implementation à la Java.

```
def approximate_size(size, a_kilobyte_is_1024_bytes=True):
```

function definition keyword — function name — comma separate argument list — default value

Note

- no return type, it always return a value (`None` as a default);
- no parameter types, the interpreter figures out the parameter type;

Overview of the Basic Concepts
Calling Functions

Look at the bottom of the humanize.py program:

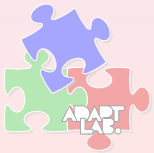
```
1  if __name__ == '__main__':
2      print(approximate_size(1000000000000, False))
3      print(approximate_size(1000000000000))
```

2. in this call to `approximate_size()`, the `a_kilobyte_is_1024_bytes` parameter will be **`False`** since you explicitly pass it to the function;
3. in this row we call `approximate_size()` with only a value, the parameter `a_kilobyte_is_1024_bytes` will be **`True`** as defined in the function declaration.

Value can be passed by name as in:

```
approximate_size(a_kilobyte_is_1024_bytes=False, size=1000000000000)
```

Parameters' order is not relevant.



Overview of the Basic Concepts
Writing Readable Code

Documentation Strings

A Python function can be documented by a documentation string (docstring for short).

''' Convert a file size to human-readable form. '''

Triple quotes delimit a single multi-string.

- if it immediately follows the function's declaration it is the docstring associated to the function.
- docstrings can be retrieved at run-time (they are attributes).

Case-Sensitive

All names in Python are case-sensitive.

Overview of the Basic Concepts
Everything is an Object

Everything in Python is an Object, functions included.

```
[15:51]cazzola@ulik:~/esercizi-pa>python3
Python 3.1.1 (r311:74480, Aug 17 2009, 21:52:33)
[GCC 4.4.1] on linux2
Type help, copyright, credits or license for more information.
>>> import humanize
>>> print(humanize.approximate_size(4096))
4.0 KiB
>>> print(humanize.approximate_size.__doc__)
Convert a file size to human-readable form.
>>>
[15:52]cazzola@ulik:~/esercizi-pa>
```

- **import** can be used to load python programs in the system as modules;
- the dot-notation gives access to the public functionality of the imported modules;
- the dot-notation can be used to access the attributes (e.g., the `__doc__`);
- `humanize.approximate_size.__doc__` gives access to the docstring of the `approximate_size()` function; the docstring is stored as an attribute.

Overview of the Basic Concepts
Everything is an Object (Cont'd)

In Python everything is an object, better, is a first-class object

- everything can be assigned to a variable or passed as an argument.

```
[16:30]cazzola@ulik:~/esercizi-pa>python3
Python 3.1.1 (r311:74480, Aug 17 2009, 21:52:33)
[GCC 4.4.1] on linux2
Type help, copyright, credits or license for more information.
>>> import humanize
>>> h1 = humanize.approximate_size(9128)
>>> h2 = humanize.approximate_size
>>> print("'{0}' is the stored value, '{1}' is the calculated value".format(h1, h2(9128)))
'8.9 KiB' is the stored value, '8.9 KiB' is the calculated value
>>>
[16:36]cazzola@ulik:~/esercizi-pa>
```

Note

- `h1` contains the string calculated by `approximate_size(9128)`;
- `h2` contains the "function" object `approximate_size`, the result is not calculated yet;
- to simplify the concept: `h2` can be considered as a new name of (alias to) `approximate_size`.

Overview of the Basic Concepts
Indenting Code

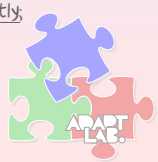
No explicit block delimiters

- the only delimiter is a column (':') and the code indentation.

```
def approximate_size(size, a_kilobyte_is_1024_bytes=True):
    if size < 0:
        raise ValueError('number must be non-negative')
    multiple = 1024 if a_kilobyte_is_1024_bytes else 1000
    for suffix in SUFFIXES[multiple]:
        size /= multiple
        if size < multiple:
            return '{0:.1f} {1}'.format(size, suffix)
    raise ValueError('number too large')
```

Note

- code blocks (i.e., functions, if statements, loops, ...) are defined by their indentation;
- white spaces and tabs are relevant: use them consistently;
- indentation is checked by the compiler.



Overview of the Basic Concepts
Exceptions

Exceptions are Anomaly Situations

- C encourages the use of return codes which you check;
- Python encourages the use of exceptions which you handles.

Raising Exceptions

- the **raise** statement is used to raise an exception as in

  raise ValueError('number must be non-negative')

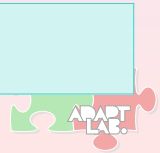
- syntax recalls function calls: **raise** statement followed by an exception name with an optional argument;
- exceptions are realized by classes.

No need to list the exceptions in the function declaration.

Handling Exceptions

- an exception is handled by a **try** ... **except** block.

```
try:
  from lxml import etree
except ImportError:
  import xml.etree.ElementTree as etree
```



Overview of the Basic Concepts
Running Scripts

Look, again, at the bottom of the `humanize.py` program:

```
1 if __name__ == '__main__':
2    print(approximate_size(1000000000000, False))
3    print(approximate_size(1000000000000))
```

Modules are Objects

- they have a built-in attribute `__name__`

```
[18:29]cazzola@ulik:~/esercizi-pa>python3
>>> import humanize
>>> humanize.__name__
'humanize'
>>> ^D
[18:30]cazzola@ulik:~/esercizi-pa>python3 humanize.py
1.0 TB
931.3 GiB
[18:30]cazzola@ulik:~/esercizi-pa>
```

The value of `__name__` depends on how you call it

- if imported it contains the name of the file without path and extension;
- if run as a stand-alone program it contains the `"main"` string.

References

- Jennifer Campbell, Paul Gries, Jason Montojo, and Greg Wilson.
  Practical Programming: An Introduction to Computer Science Using Python.
  The Pragmatic Bookshelf, second edition, 2009.
- Mark Lutz.
  Learning Python.
  O'Reilly, third edition, November 2007.
- Mark Pilgrim.
  Dive into Python 3.
  Apress*, 2009.