Python's Whys & Hows

What is Python

- 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.

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

```python
print("Hello World!!!")
```

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

```bash
cazzola@ulik:~/esercizi-pa>python3 hello.py
Hello World!!!
cazzola@ulik:~/esercizi-pa>
```

The python shell can be used to get interactive help.

```bash
cazzola@ulik:~/esercizi-pa>python3
[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".
To get a list of modules whose summaries contain a given word such as "spam", type "modules spam".
>>> help()
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.
```

```bash
[23:35]cazzola@ulik:~/esercizi-pa
```
Slide 5 of 14

Overview of the Basic Concepts
Our First Python Program

humanize.py

```python
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
```

Slide 6 of 14

Overview of the Basic Concepts
Declaring Functions

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

```python
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))
```

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

Parameters' order is not relevant.

Value can be passed by name as in:

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

Slide 7 of 14

Overview of the Basic Concepts
Calling Functions

Look at the bottom of the humanize.py program:

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

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

Parameters' order is not relevant.

Slide 8 of 14

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.'
```

```
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')
```

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 in Python is an Object

Everything is an Object, Functions included

- **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

Indenting Code

No explicit block delimiters
- the only delimiter is a column (':') and the code indentation.

```python
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 ['MB', 'KB', 'B'];
        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 handle.

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 excep-
  tion name with an optional argument;
- exceptions are realized by classes.

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

```python
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:

```python
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__`

```python
>>> import humanize
>>> humanize.__name__
'humanize'
```

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