Advanced Programming

Federico Bruzzone

20 settembre 2022

Indice

1 Python

1.1 Python's whys & hows

1.1.1 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 support multiple programming paradigms

- imperative (function, state, ...);
- object-oriented/based (objects, methods, inheritance, ...);
- functional (lambda abstractions, generators, dynamic typing, ...).

Python is

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

1.1.2 How to use Python

We are condidering Python 3+

- version > 3 is incompatible with previus version;
- version 2.7 is the current version.

A python program can be:

- edited in the python shell and executed step-by-step by the shell;
- edited and run through the iterpreter.

1.2 Overview of the Basic Concepts

1.2.1 Our first Python program

```
1 SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'],
2             1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']}
3 def approximate_size(size, a_kilobyte_is_1024_bytes=True):
4   ''' Convert a file size to human-readable form. '''
5   if size < 0:
6     raise ValueError('number must be non-negative')
7   multiple = 1024 if a_kilobyte_is_1024_bytes else 1000
8   for suffix in SUFFIX[multiple]:
9     size /= multiple
10    if size < multiple:
11      return '{0:.1f} {1}'.format(size, suffix)
12    raise ValueError('number too large')
13
14 if __name__ == '__main__':
15   print(approximate_size(1000000000000, False))
16   print(approximate_size(1000000000000))
```

Listing 1: humanize.py

1.2.2 Declaring function

Python has function

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

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

1. **def**: function definition keyword;
2. **approximate_size**: function name;
3. **a_kilobyte_is_1024_bytes**: comma separate argument list;
4. =**True**: default value.

Python has function

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

1.2.3 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:

```
1 def approximate_size(a_kilobyte_is_1024_bytes=True, size=1000000000000)
```

Parameters' order is not relevant

1.2.4 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 immediatly follows the function's declaration it is the doc-string associated to the function;
- docstrings can be retrieved at run-time (they are attributes).

Case-Sensitive All names in Python are case-sensitive

1.2.5 Everything is an object

Everything in Python 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 the public functionality of the imported modules;
- the dot-notation can be used to access the attributes (e.g., the **__doc__**)
- **humanizeàpproximate_size.__doc__** gives access to the docstring of the **approximate_size()** function; the docstring is stored as an attribute.

1.2.6 Everything is an object (Cont'd)

In python is an object, better, is a first-calss object

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

---

```
1 h1 = humanize.approximate_size(9128)
2 h2 = humanize.approximate_size
```

- **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**.

1.2.7 Indenting code

No explicit block delimiters

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

1.2.8 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 rise an exception as in:

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

- syntax recalls function calls: **raise** statement followed by an exception name with an optional argument;

- exceptions are relized by classes.

No need to list the exceptions in the function declaration handling Exceptions

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

```
1 try:
2   from lxml import etree
3 except ImportError:
4   import xml.etree.ElementTree as etree
```

1.2.9 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__**

The value of __name__ depends on how you call it

- if imported it contains the name of the file without path and extension.

2 Primitive Datatypes & recursion in Python

2.1 Python's Native Datatypes

2.1.1 Introduction

In python **every value has a datatype**, but you do not need to declare it.

How does that work?

Based on each variable's assignment, python figures out what type it is and keeps tracks of that internally.

2.1.2 Boolean

Python provides two constants

- **True** and **False**

Operations on Booleans

Logic operations: *and or not*

Relational operators: == != < > <= >=

Note that python allows chains of comparisons

```
1 >>> x = 3
2 >>> 1<x<=5
3 True
```

2.1.3 Number

Two kinds of number: integer and floats

- no class declaration to distinguish them
- they can be distiguished by the presence/absence of the decimal point

```
1 >>> type(1)
2 <class 'int'>
3 >>> isinstance(1, int)
4 True
5 >>> 1+1
6 2
7 >>> 1+1.0
8 2.0
9 >>>type(2.0)
10 <class 'float'>
```

- **type()** function provides the type of any value or variable;
- **isinstance()** check if a value or variable is of a given type;
- adding an int to an yields another int but adding it to a float yields a float.