An informal introduction to Python

Inteligencia Artificial en los Sistemas de Control Autónomo
Máster en Ciencia y Tecnología desde el Espacio

Departamento de Automática

Universidad de Alcalá
ISG

Objectives

1. Understand the main Python features, strengths and weaknesses
2. Overview the main Python statements
3. Being able to program naïve Python scripts

Table of Contents

Introduction

What is Python?

Python is a general-purpose, high-level, interpreted programming language

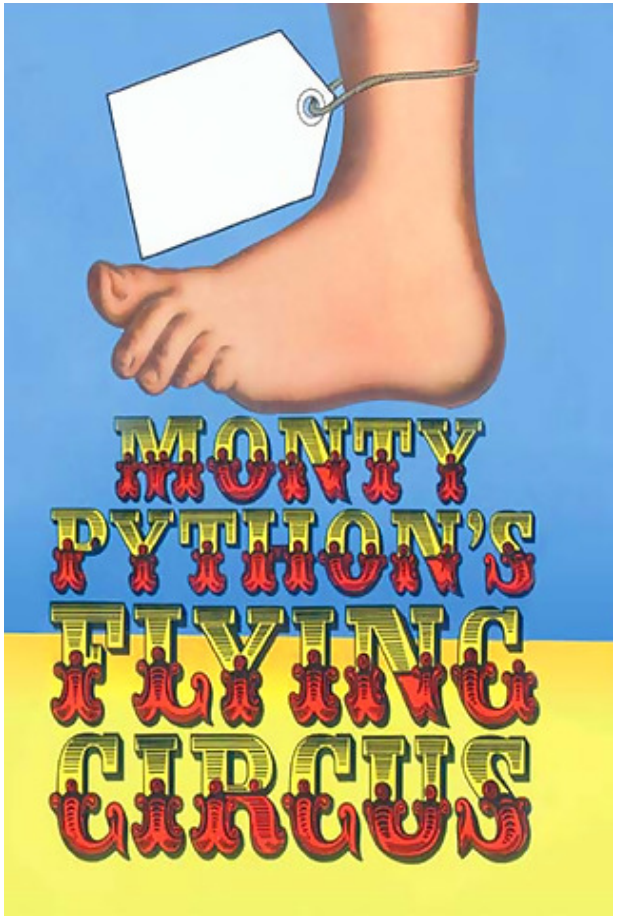
- *General-purpose*: Many applications.
- *High-level*: Abstract data structures, doing more with less code.
- *Interpreted*: No need to compile.

It emphasizes code **readibility** and programmer's productivity

Introduction

Features

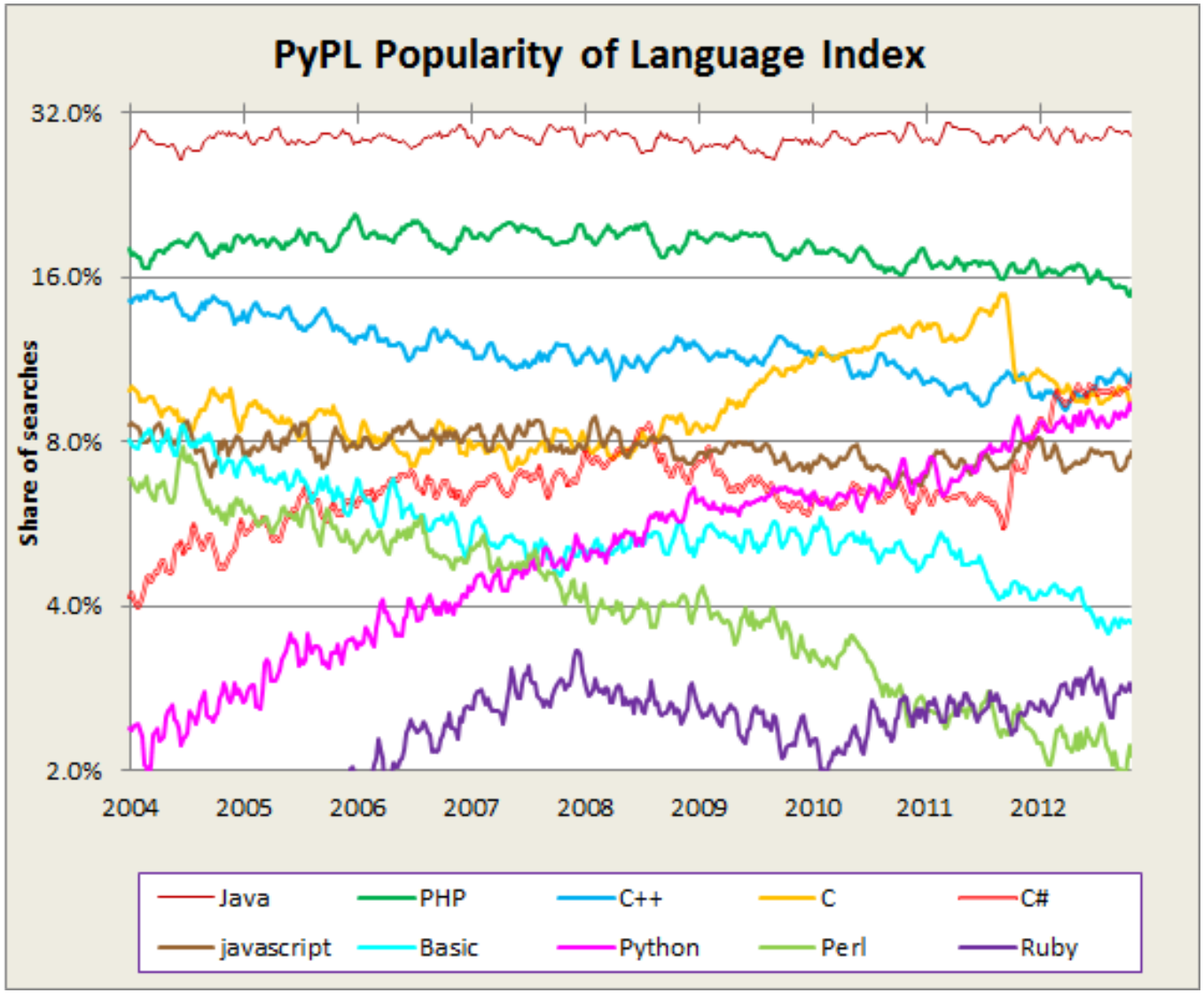
- Several paradigms
  - OO, imperative and functional
- Dynamic typing
- Interpreted
- Minimalistic syntax
- Portable
- Extensible - Bindings to other languages
- Embeddable
- Application domains
  - Web, robotics, data science, game development, admin ...



Want to know other Monty Python's contribution to Computer Science? Click here

Introduction

Why Python? (I)



(Source)

Introduction

Why Python? (II)

Hello world! examples

Python

```
#!/usr/bin/python

print("Hello, world!")
```

Java

```
public class HelloWorld {
  public static void main(String[]
      args) {
    System.out.println("Hello, world
        !");
  }
}
```

C

```
#include <stdio.h>

int main()
{
  printf("Hello, world!\n");
}
```

C++

```
#include <iostream>

int main()
{
  std::cout << "Hello, world!\n"
      ;
}
```

Introduction

Why Python? (III)

More reasons to love Python

- Very easy to learn ...
  - ... yet extremely powerful
  - Clearner syntax compared to almost anything else
- Facilities in development
  - Wide standard library: `http://docs.python.org/library/`
  - Great number of modules.
  - Almost any software library has its associated wrapping in order to its access from Python.
- Interactive mode
  - Rapid testing and development
- Most languages are made to make big and fast programs
  - Python was designed to ease programmers' life
- It is free software!

Introduction

Where Python is used?

- In Google. One of the development oficial languages
- In YouTube.
- In BitTorrent.
- In animation: DreamWorks Animation, Pixar, Industrial Light & Magic.
- Red HatFedora Installer (Anaconda).
- And much more ...: `http://www.python.org/about/success/`

Introduction

Where python is not suitable?

But ... Python is not perfect.

- It is not good for ...
  - Applications that require high computing capacity.
  - Programming of low level (system-programming): programming of kernels, *drivers*, etc.
  - Very big programs (open discussion).

Introduction

History

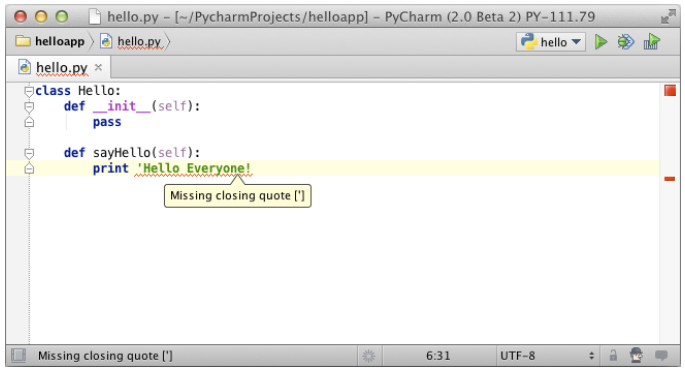
- Python was created by Guido van Rossum in the Netherlands
- Python 2.0: Released on 2000
- Python 3.0: Released on 2008. Backwards-incompatible

Python 2.X is still very popular, but Python 3.X is the future (deprecated from 2020)

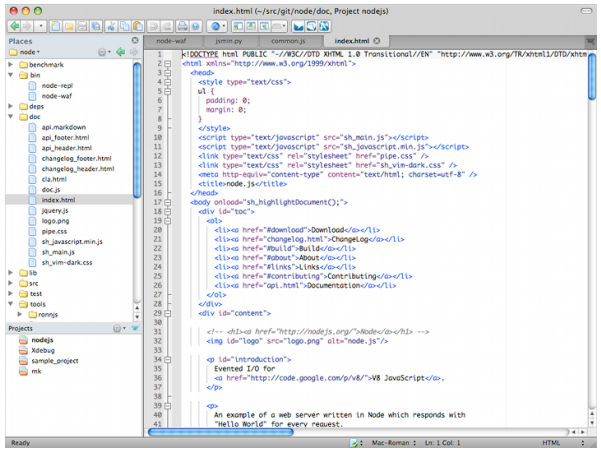
Introduction

Installation

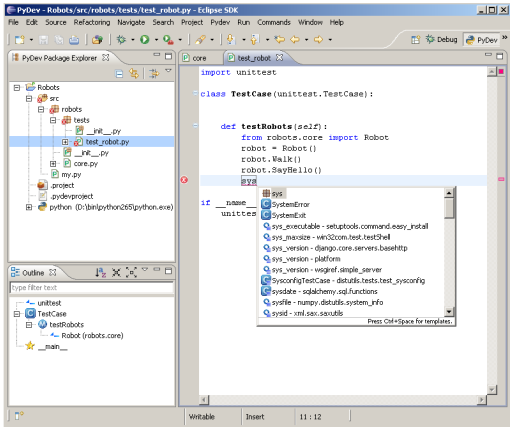
- If you have a good OS such as Linux or Mac, you already have Python!
- Otherwise (Windows), you have to install it
  - Visit `https://www.python.org/downloads/`
- Bad news: There is no "standard" IDE
  - PyCharm, Komodo, PyDev, ...
  - `http://wiki.python.org/moin/PythonEditors`



PyCharm



Komodo



PyDev

The Python interpreter

Python operation modes

Python is an interpreted language, i.e., it needs an interpreter.

- Interpreted = it is not complied = it needs no compilation.
- Faster development, slower execution.

Two operation modes:

- **Interactive**: The interpreter reads the program from the *stdin* (usually the keyboard).
- **Non-interactive**: The interpreter reads the program from a file (also known as script).

The Python interpreter

Non-interactive

The program is in a plain text file.

- It can be edited with any text editor.
- Extension ".py".
- Execution permission (`chmod u+x myscript.py`).
- By default, UTF-8 encoding.

The first line must be `#!/usr/bin/python`

- It is the interpreter location.
- If not present, the interpreted must be invoked.

script.py

```
#!/usr/bin/python

print("Hello, world!")
```

```
python script.py
./script.py
```

The Python interpreter

Interactive

Just run `python`

- Different names for different versions to avoid conflicts.
- `python`, `python3.4`, ...

```
localhost:~ user$ python3.4
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The programmer executes as he writes code down.

An informal introduction

Variables (I)

Variable: A name that refers a value.

- No need to declare variables (Python is weakly typed!).
- Python automatically assigns types.
- Basic types: Numbers, strings and booleans.

Complex data structures:

- Lists, tuples, dictionaries, associative arrays.

Variables

```
variable = value
```

An informal introduction

Variables (II)

Hint: `type()` returns data type.

```
>>> integer = 4
>>> float = 2.3
>>> integer + float
6.3
>>> string = "Spam"
>>> boolean = True
>>> a = b = c = 0
>>> b
0
>>> type(integer)
<type 'int'>
```

An informal introduction

Numbers (I)

Number types: Integer, float and complex.

```
>>> num = 1+3j
>>> num
(1+3j)
```

Sign	Operator	Sign	Operator
=	Assignment	//	Floor division[1]
+	Add	**	Exponent
-	Substration	+=	Assign +
*	Multiplication	-=	Assign -
/	Division	*=	Assign *
%	Modulus	/=	Assign /

[1] Only Python 3.x

An informal introduction

Numbers (II)

ArithmeticDemo.py

```
a = int(input("Number: "))
b = float(input("Number: "))
c = (a * b) / 2
c += 1
d = c ** 2
print("Result c: ", c)
print("Result d: ", d)
```

New Python elements:

- The `input()` function.
- The `int()` and `float()` functions.

An informal introduction

Strings (I)

```
>>> 'hello'
'hello'
>>> "hello"
'hello'
```

Strings are delimited with single or double quotes, they can be used together.

Triple quotes to define multi-line strings.

```
>>> """hello
... there are multiple lines"""
'hello\nthere are multiple lines'
```

As C, C++ or Java, '\n' means carriage return.

An informal introduction

Strings (II)

Of course, variables can contain strings.

```
>>> text = "hello"
>>> print(text)
hello
```

New Python elements:

- The `print()` function.

An informal introduction

Strings (III)

Strings contatenation

```
>>> "hello" + " there"
'hello there'
>>> "hello" "there"
'hellothere'
```

Variables with strings

```
>>> a = "hello"
>>> b = " there"
>>> a + b
'hello there'
```

String length

```
>>> len("hello")
5
```

An informal introduction

Strings (IV)

Strings can be used as a sequence of characters: *Slice notation*.

- Quite common in Python data structures.
- It uses indices (as an array). First index is 0.

```
>>> a = "hello"
>>> a[2]
'l'
>>> a[2:]
'llo'
>>> a[:2]
'he'
>>> a[2:] + a[:2]
'llohe'
>>> a[2:4]
'll'
```

An informal introduction

Lists (I)

List: An ordered collection of mutable data.

- Very powerful data structure, similar to an array.
- *Ordered*: Data in the list have a location.
- *Mutable*: Data can be modified.
- Data types can be different.

List initialization

```
variable = [data1, data2, ..., dataN]
```

An informal introduction

Lists (II)

Definition example

```
>>> a = ['spam', 'eggs', 123]
>>> a
['spam', 'eggs', 123]
```

Slice notation and the `len()` function work on lists

```
>>> a[2]
123
>>> a[1:]
['eggs', 123]
>>> a + a[2:len(a)]
['spam', 'eggs', 123, 123]
```

Control flow

Conditions (I)

Conditional statements implement decision making

- Decide some code has to be executed or not.
- The result is a boolean.
- Execute code if condition is satisfied.

`if` statement

```
if condition:
    # Some code
else:
    # Some other code
```

New Python elements:

- Comments begin with '#'.
- Indentation plays a mayor role: It defines code bodies.

Control flow

Conditions (II)

Condition example

```
temperature = float(input('What is the temperature? '))
if temperature > 70:
    print('Wear shorts.')
else:
    print('Wear long pants.')
print('Get some exercise outside.')
```

(Source)

New Python element:

- Comparison operators.

Control flow

Conditions (III)

Sign	Operator	Sign	Operator
==	Equal	`and`	Logical and
!=	Not equal	`or`	Logical or
>	Greater	`not`	Logical not
<	Lower		
>=	Greater or equal		
<=	Lower or equal		

Example: `((age > 18) or (name == 'Biggus Dickus'))`

Control flow

While loop

Fibonacci series

```
#!/usr/bin/python

a, b = 0, 1 # Init variables

while b < 10: # This is a loop
   print("b = ", b)
   print("a = ", a)  # Identation is very important here!
   a, b = b, a+b
```

New Python elements:

- Multiple assignments.

Examples

Example 1: Multiplication table

multi.py

```
table = 8
start = 1
max = 10
s = '-' * 20
print (s)
print ('The table of 8')
print (s)
i = start
while i <= max:
    result = i * table
    print (i, ' * ', table, ' =' , result)
    i = i + 1
print (s)
print ('Done counting...')
print (s)
```

Source

Bibliographic references I


[van Rosum, 2012] G. van Rossum, Jr. Fred L. Drake.
Python Tutorial Release 3.2.3, chapters 2 and 3.
Python Software Foundation, 2012.


[Bahit, 2008] E. Bahit.
Curso: Python para principiantes.
Creative Commons Atribución-NoComercial 3.0, 2012.


[Swaroop, 2008] C H. Swaroop.
A Byte of Python.
Creative Commons Attribution-ShareAlike 3.0, 2008.


[Pilgrim, 2004] M. Pilgrim.
Dive into Python.
Ed. Prentice Hall, 2004.