

Python tutorial

Joris Gillis

August 3, 2014

Table of Contents

What is Python?

How to run Python?

Let's get coding

Table of Contents

What is Python?

How to run Python?

Let's get coding

Python's origin



- ▶ Guido Van Rossum (Netherlands)
- ▶ Created in 1991
- ▶ Named after Monty Python's flying circus

Python is a programming language
simple, powerful, fun



Python is a programming language
simple, powerful, fun

Python is mature yet in active development
Python Enhancement Proposals (PEP)

Python is a programming language
simple, powerful, fun

Python is mature yet in active development
Python Enhancement Proposals (PEP)

Python is a scripting language

- ▶ Dynamic typing
- ▶ Object oriented
- ▶ Interactive

Python is a programming language
simple, powerful, fun

Python is mature yet in active development
Python Enhancement Proposals (PEP)

Python is a scripting language

- ▶ Dynamic typing
- ▶ Object oriented
- ▶ Interactive

Python is cross-platform
Linux, Windows, Mac, android phones, ...

Python is glue

Easily interacts with

- ▶ filesystem, operating system
- ▶ command line tools
- ▶ compiled libraries (C, C++, Fortran)



Python is glue

Easily interacts with

- ▶ filesystem, operating system
- ▶ command line tools
- ▶ compiled libraries (C, C++, Fortran)

Python is a universe

- ▶ Batteries included
- ▶ Thousands of high-standard packages out there
- ▶ Open-source effort

Python is versatile

- ▶ Web Programming
- ▶ GUI Development
- ▶ **Scientific and Numeric**
- ▶ Software Development
- ▶ System Administration

Python happens to be suitable for science

- ▶ The “Scientific and Numeric” has been booming for 5 years
- ▶ Toolstack bundled as ‘scipy’
 - ▶ Theano, numexpr, numba (Python math compilers)
 - ▶ **Numpy** (linear algebra)
 - ▶ Sympy (computer algebra system)
 - ▶ Pandas (data analysis)
 - ▶ Fenics, Fipy (Python for PDEs)
 - ▶ Vistrails (Python scientific workflow, data exploration and visualization)
 - ▶ **matplotlib**, chaco (2D plotting)
 - ▶ Mayavi (3D visualization)
 - ▶ Pyomo, cvxpy (Python optimization)

Python is a free and open-source ecosystem

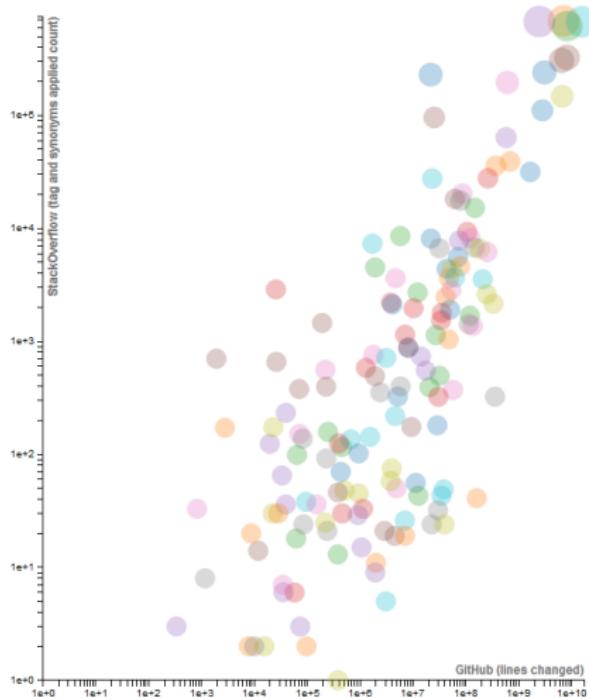
- ▶ Tutorials, documentation
- ▶ Rapid bug-fixing
- ▶ Nightly testing (Continuous integration)
- ▶ Survival of the fittest

Python is popular

- ▶ Lingua Franca
- ▶ A huge user-base to ask/answer questions
- ▶ For every nail you want to hit, there might be 5 hammers out there

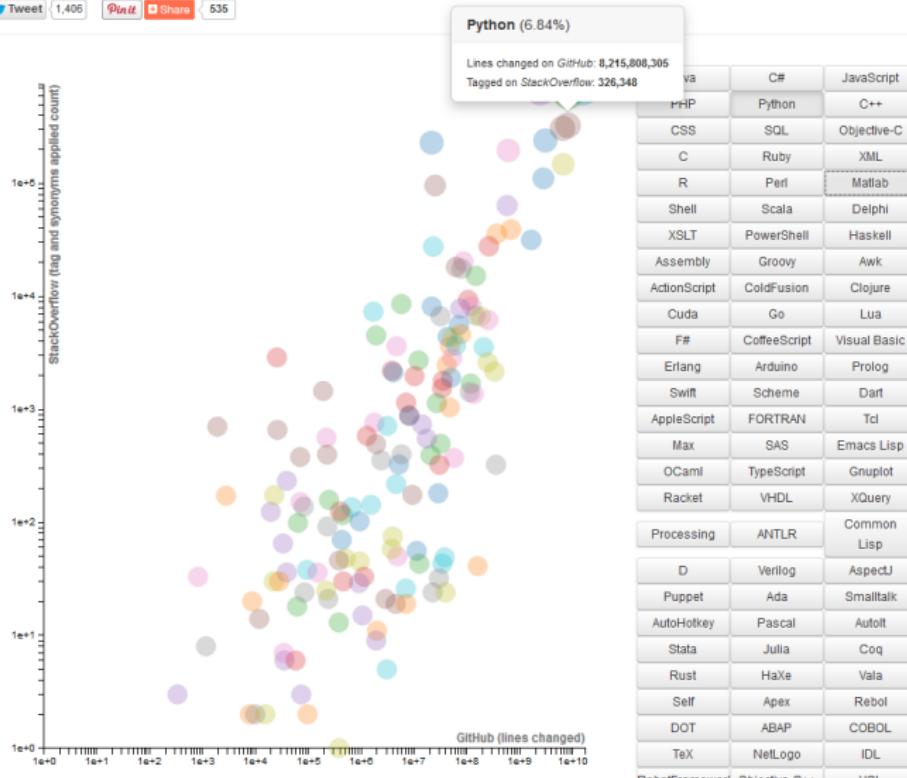
<http://langpop.corger.nl>

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535

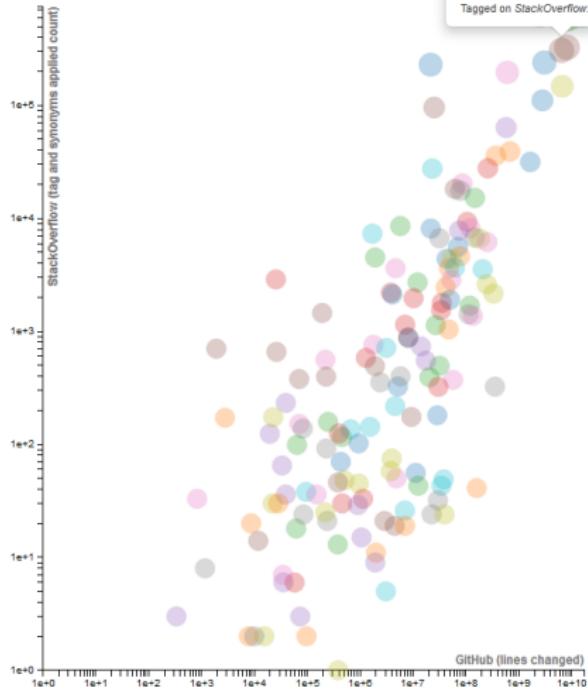


Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) 535 [Share](#)



[Twitter](#) 1,406 [Pin it](#) 535 [Share](#)



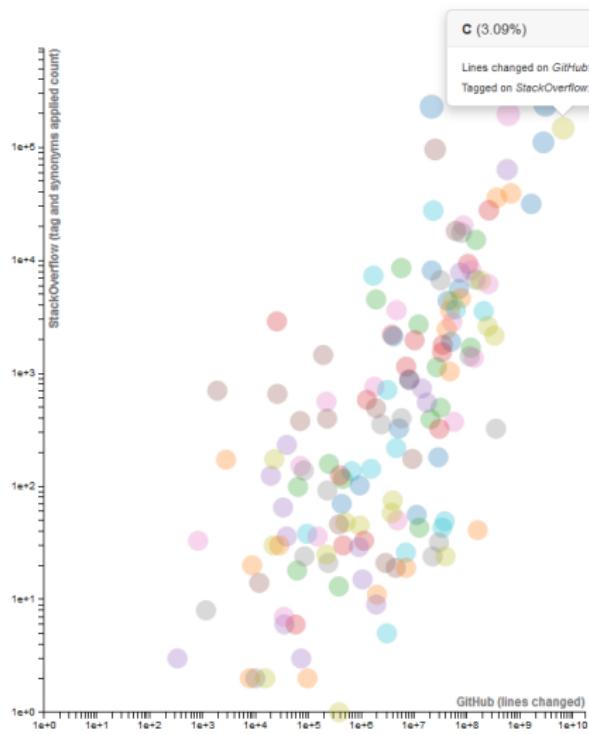
C++ (6.44%)

Lines changed on GitHub: 6,447,068,673
Tagged on StackOverflow: 307,302

Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

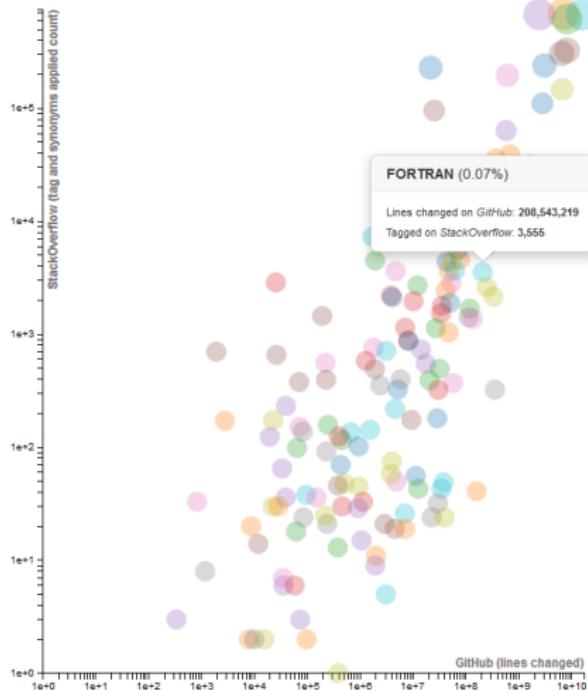
http://langpop.corger.nl

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



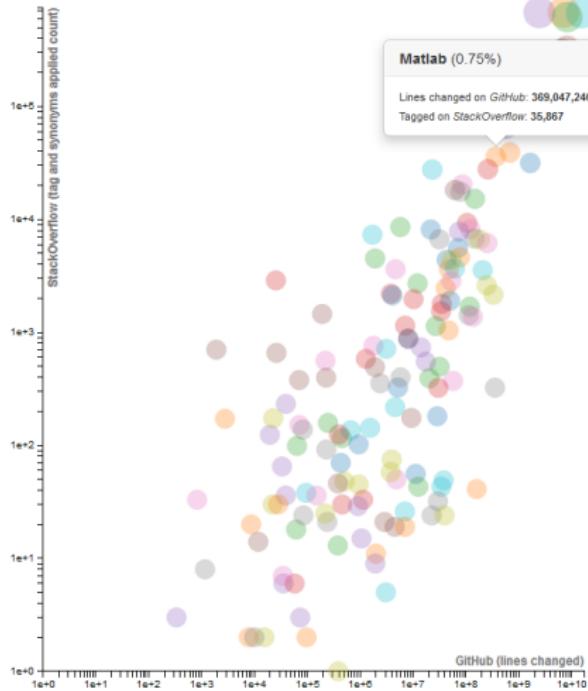
Java	C#	JavaScript
HP	Python	C++
SS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



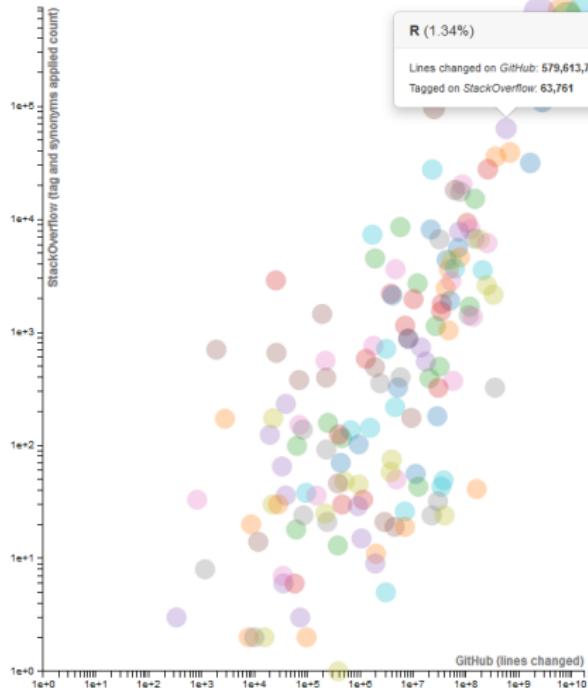
Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



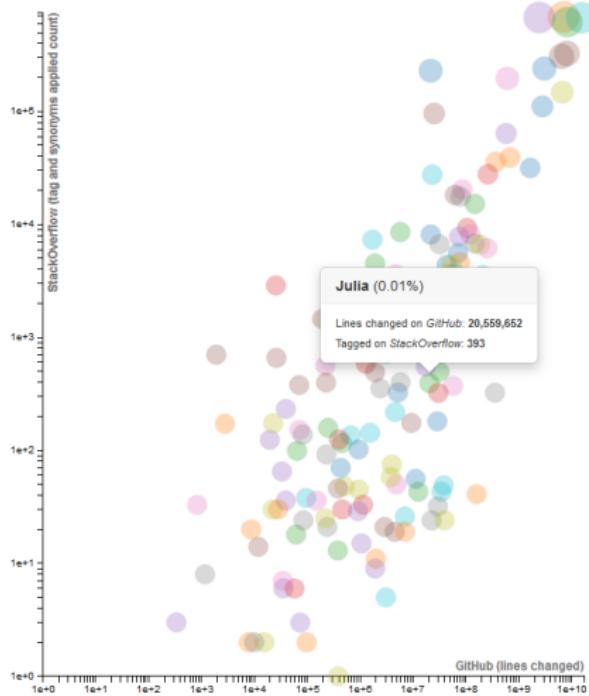
Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



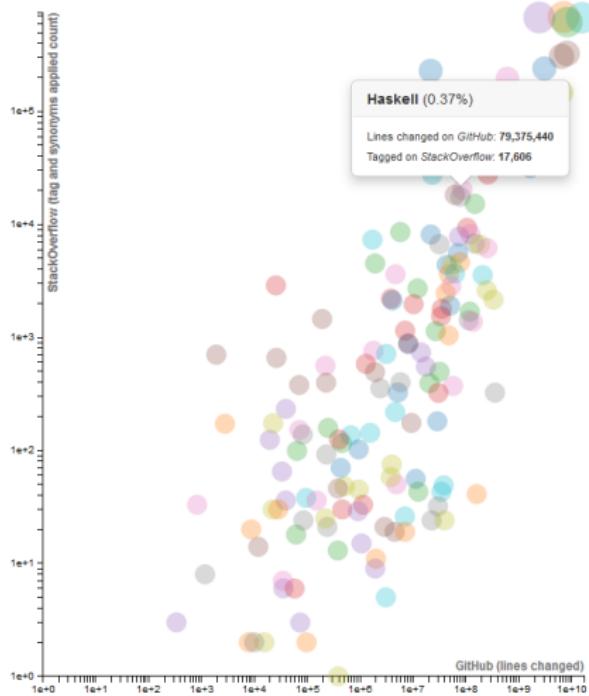
Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) [Share](#) 535



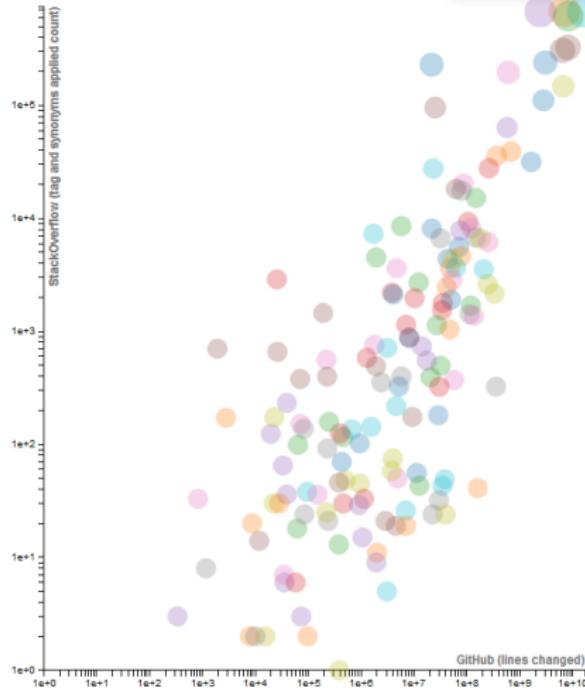
Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) 535 [Share](#)

PHP (12.95%)

Lines changed on GitHub: 8,284,596,823

Tagged on StackOverflow: 618,129



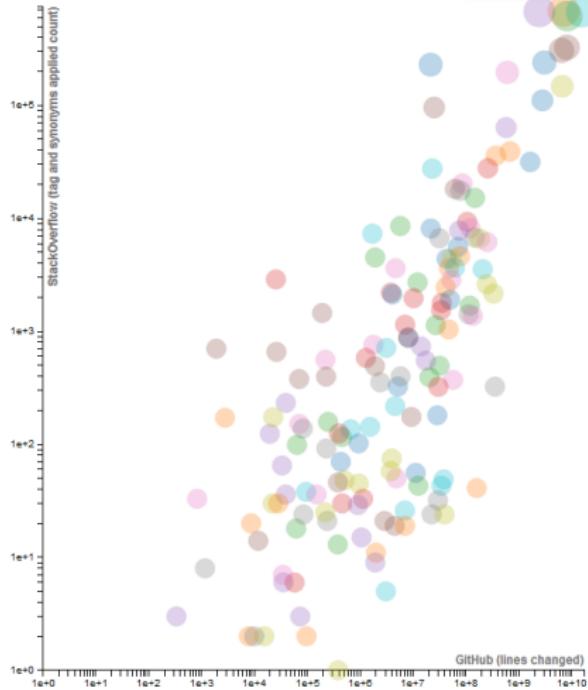
Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

[Twitter](#) 1,406 [Pin it](#) 535 [Share](#)

JavaScript (14.14%)

Lines changed on GitHub: 15,622,453,074

Tagged on StackOverflow: 674,781



Java	C#	JavaScript
PHP	Python	C++
CSS	SQL	Objective-C
C	Ruby	XML
R	Perl	Matlab
Shell	Scala	Delphi
XSLT	PowerShell	Haskell
Assembly	Groovy	Awk
ActionScript	ColdFusion	Clojure
Cuda	Go	Lua
F#	CoffeeScript	Visual Basic
Erlang	Arduino	Prolog
Swift	Scheme	Dart
AppleScript	FORTRAN	Tcl
Max	SAS	Emacs Lisp
OCaml	TypeScript	Gnuplot
Racket	VHDL	XQuery
Processing	ANTLR	Common Lisp
D	Verilog	AspectJ
Puppet	Ada	Smalltalk
AutoHotkey	Pascal	AutoIt
Stata	Julia	Coq
Rust	HaXe	Vala
Self	Apex	Rebol
DOT	ABAP	COBOL
TeX	NetLogo	IDL

IEEE Spectrum's 2014 Ranking

Language Rank	Types	Spectrum Ranking
1. Java	🌐💻🖱️	100.0
2. C	💻🖱️	99.3
3. C++	💻🖱️	95.5
4. Python	🌐💻	93.4
5. C#	🌐💻🖱️	92.4
6. R	💻	74.2
7. MATLAB	💻	72.9
8. PERL	🌐💻	70.3
9. SQL	💻	70.3
10. Visual Basic	💻	64.6
11. Objective-C	📱💻	64.5
12. Shell	💻	62.3
13. Go	🌐💻	60.0
14. Processing	🌐💻	51.9
15. D	💻🖱️	50.2
16. Lua	🌐💻	47.8
17. Fortran	💻	46.7
18. LISP	💻	44.3
19. Haskell	💻🖱️	44.1
20. Delphi	📱💻	43.2
21. Prolog	💻	38.2
22. Clojure	🌐💻	37.2

Table of Contents

What is Python?

How to run Python?

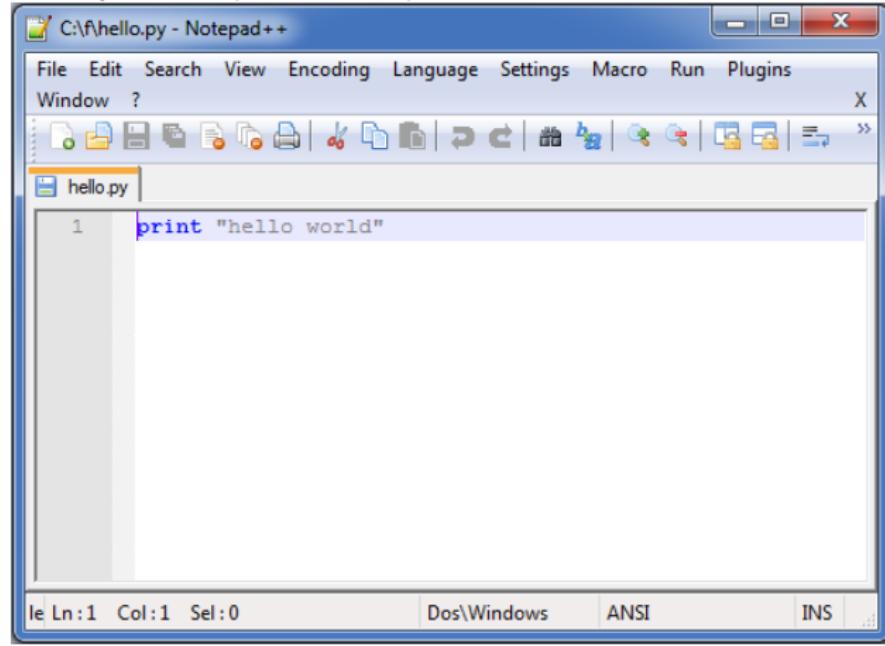
Let's get coding

Option A: as command line argument

```
$ python -c "print 'hello world'"
```

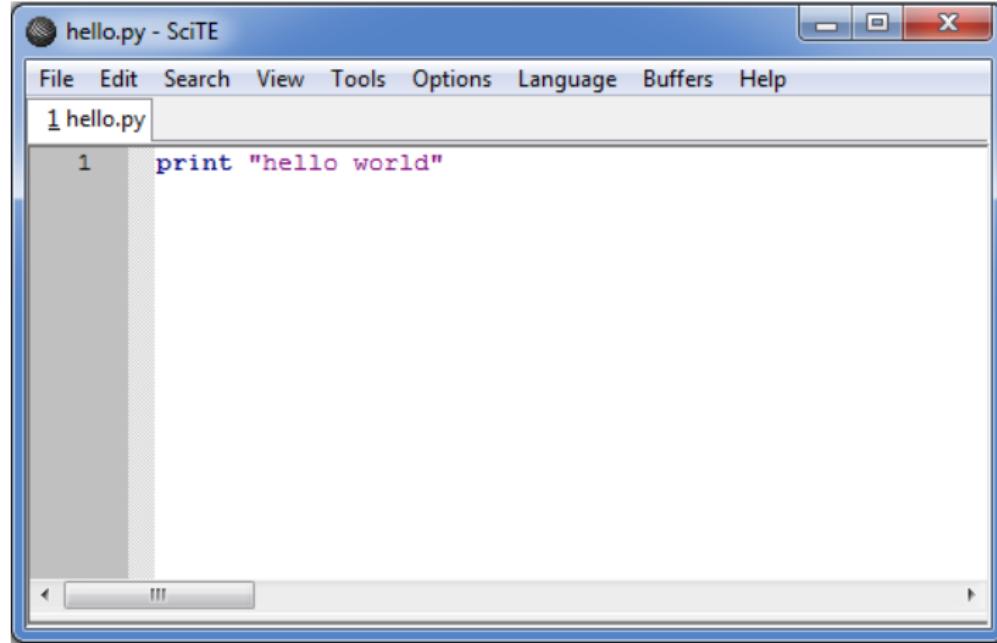
Option B: editor + command line invocation

Notepad++ (Windows)



Option B: editor + command line invocation

Scite (Windows, Linux, Mac)



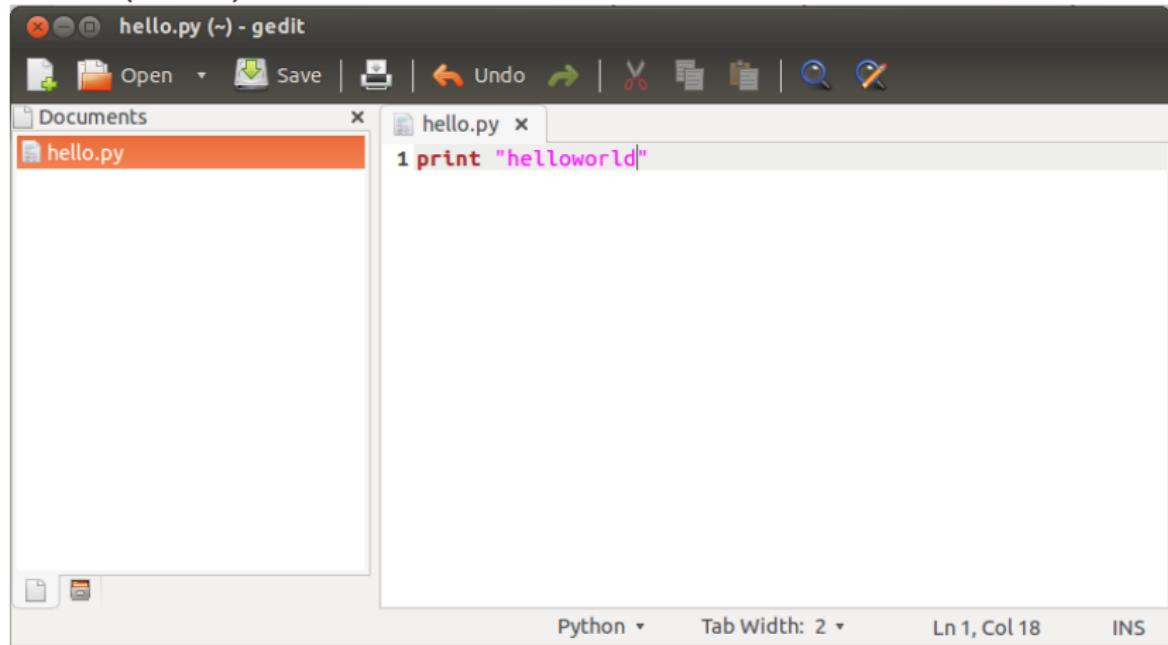
The screenshot shows the SciTE 2.0 Python editor window. The title bar reads "hello.py - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current file tab is "hello.py". The code editor displays the following Python code:

```
1 print "hello world"
```

The SciTE interface includes a status bar at the bottom with various icons for file operations like Open, Save, and Print.

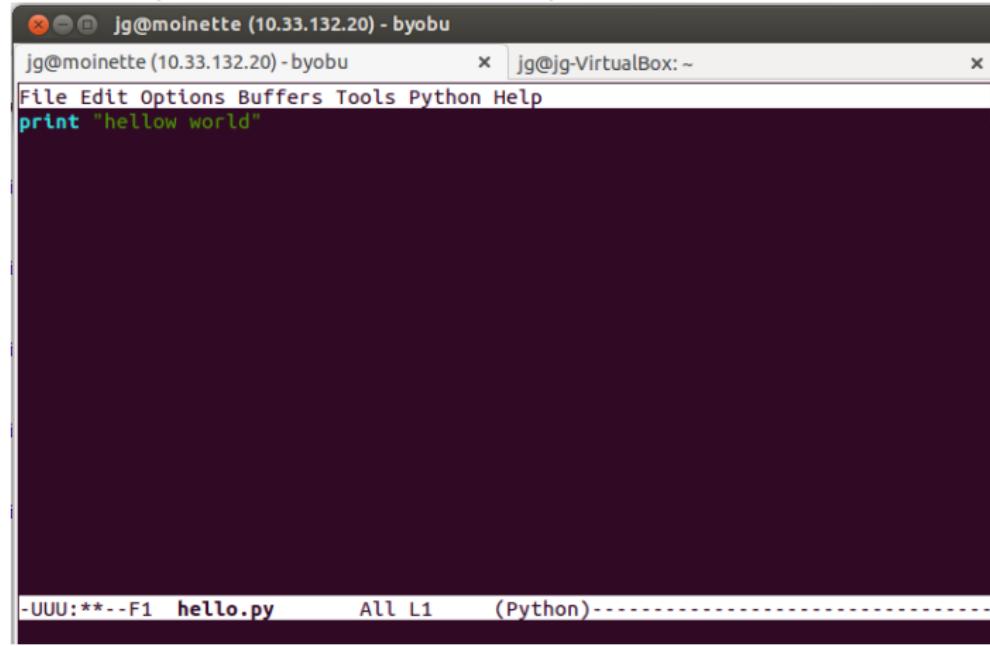
Option B: editor + command line invocation

Gedit (Linux)



Option B: editor + command line invocation

Emacs (Windows, Linux, Mac)



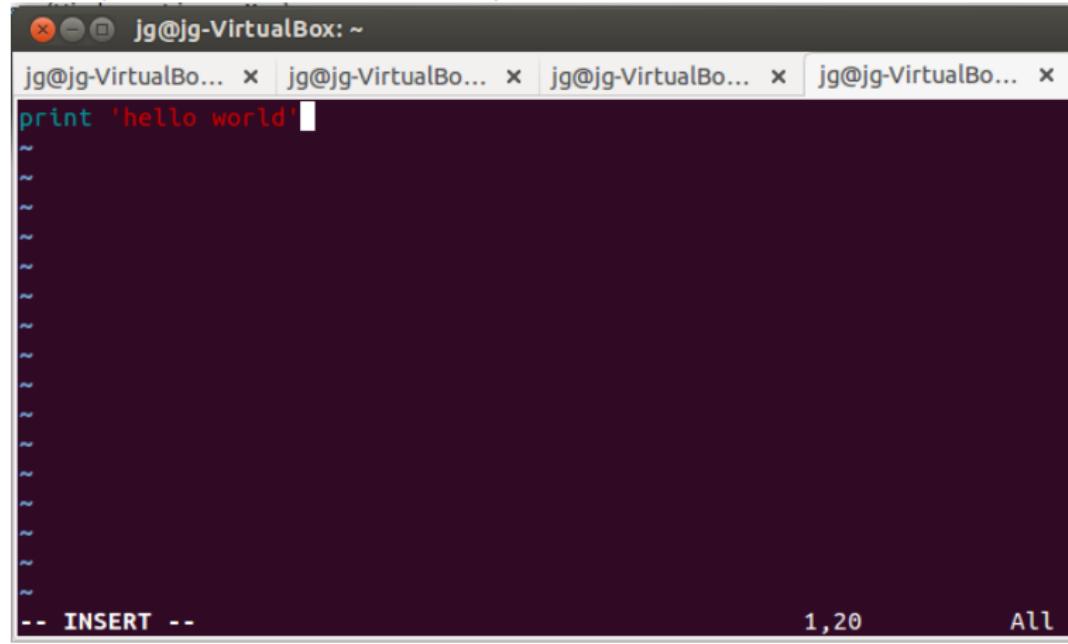
The screenshot shows an Emacs window titled "jg@moinette (10.33.132.20) - byobu". The buffer contains the following Python code:

```
File Edit Options Buffers Tools Python Help
print "hellow world"
```

The status bar at the bottom displays the file name "-UUU:***-F1 hello.py", the mode "All L1", and the language "(Python)".

Option B: editor + command line invocation

Vim (Windows, Linux, Mac)



A screenshot of the Vim text editor running in a terminal window. The terminal title bar says "jg@jg-VirtualBox: ~". There are four tabs open in the background, all labeled "jg@jg-VirtualBo...". The main Vim buffer contains the following Python code:

```
print 'hello world'
```

The status bar at the bottom shows "1,20" and "All". The bottom right corner of the terminal window has several small icons: back, forward, search, and others.

Option B: editor + command line invocation

```
$ python hello.py
```

What you get

- ▶ Syntax highlighting

Option C: interactive

IPython (Windows, Linux, Mac)

```
jg@jg-VirtualBox: ~
jg@jg-VirtualBox:~$ ipython
Python 2.7.4 (default, Apr 19 2013, 18:28:01)
Type "copyright", "credits" or "license" for more information.

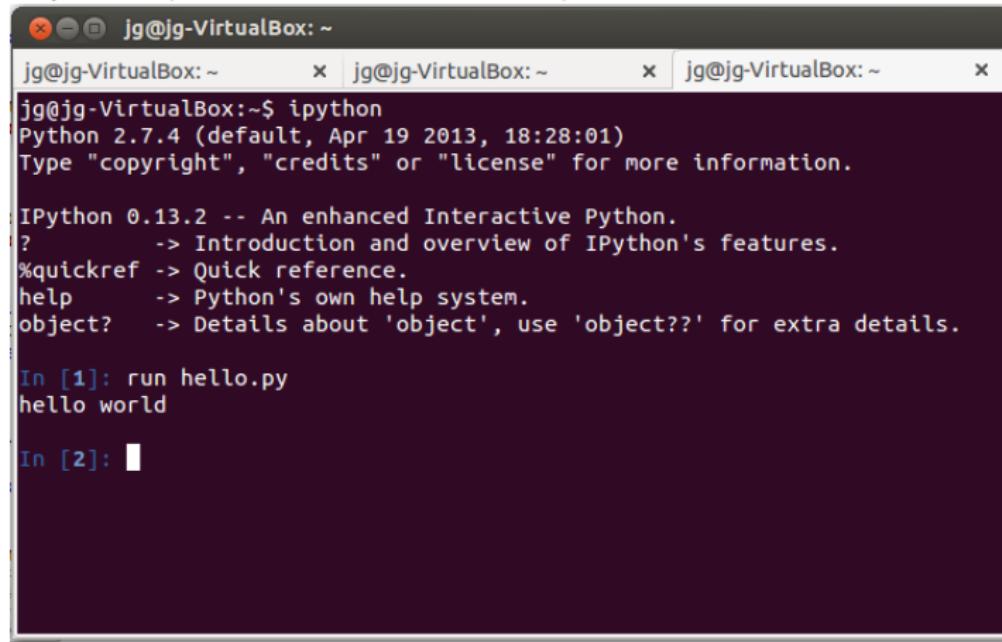
IPython 0.13.2 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: print "hello world"
hello world

In [2]:
```

Option C: interactive

IPython (Windows, Linux, Mac)



jg@jg-VirtualBox: ~

jg@jg-VirtualBox: ~

jg@jg-VirtualBox: ~

```
jg@jg-VirtualBox:~$ ipython
Python 2.7.4 (default, Apr 19 2013, 18:28:01)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.2 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: run hello.py
hello world

In [2]: 
```

Option C: interactive

IPython (Web)

The screenshot shows a Mozilla Firefox browser window with the title "Try IPython from your Browser: PythonAnywhere - Mozilla Firefox". The address bar displays the URL <https://www.pythonanywhere.com/try-ipython/>. The page content is titled "Try IPython from your browser!". It contains a text block about IPython's features and a list of keyboard shortcuts. On the right, there is a large black code editor window showing an IPython session. The session starts with a help command, followed by a print statement that outputs "hello world", and ends with an empty In [2] cell.

IPython is an enhanced interactive Python interpreter, offering tab completion, object introspection, and much more. It's running on the right-hand side of this page, so you can try it out right now.

Here's a quick micro-tutorial to get you started with some of the fun stuff it provides:

- Type `imp` then tab to get `import` then type `nu` and tab to see which modules you can import that start with '`nu`'.

```
Type "copyright", "credits" or "license"
IPython 2.1.0 -- An enhanced Interactive
?      -> Introduction and overview of
%quickref -> Quick reference.
help     -> Python's own help system.
object?   -> Details about 'object', use

In [1]: print "hello world"
hello world

In [2]:
```

Copyright © 2014 PythonAnywhere LLP — [Terms](#) — [Privacy](#)
"Python" is a registered trademark of the Python Software Foundation.

Option C: interactive

What you get

- ▶ Tab completion
- ▶ Command history

To paste in IPython, type %paste

Option D: integrated development environment (IDE)

Eclipse (Windows, Linux, Mac)

The screenshot shows the Eclipse PyDev interface for developing Python code. The window title is "PyDev - Robots/src/robots/tests/test_robot.py - Eclipse SDK".

The left sidebar displays the "PyDev Package Explorer" showing the project structure:

- Robots
- src
- robots
- tests
- __init__.py
- test_robot.py
- core.py
- my.py

The "Outline" view on the left shows the class structure:

- unittest
- unresolved_import
- unused_import = unittest
- TestCase
- testRobot
- Robot (robots.core)
- methodNotWithSelf
- __main__

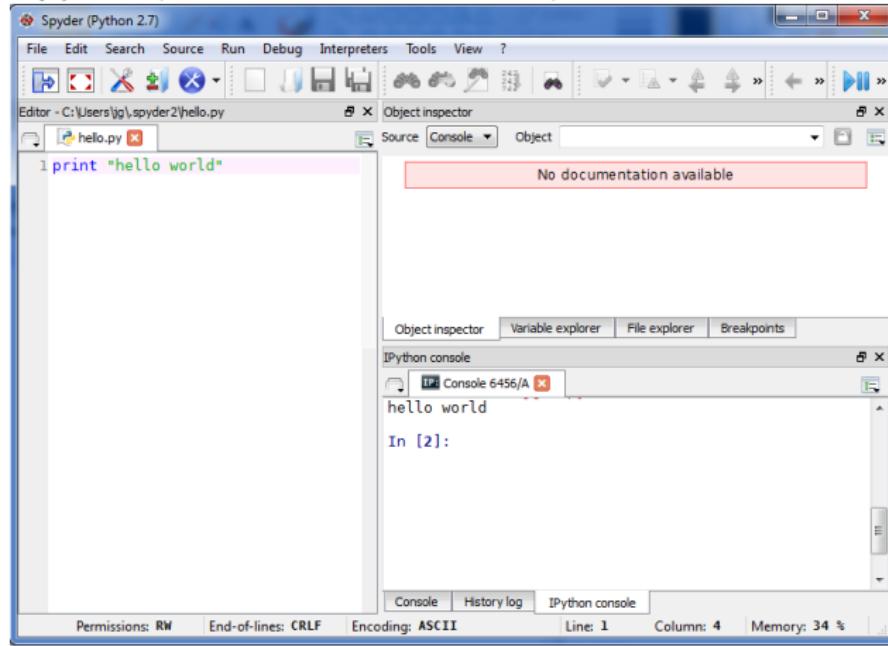
The main editor area contains the following Python code:

```
1 import unittest
2 import unresolved_import
3 import unittest as unused_import
4
5 #####
6 # TestCase
7 #
8 class TestCase(unittest.TestCase):
9
10     def testRobots(self):
11         from robots.core import Robot
12         robot = Robot()
13         robot.Walk()
14         robot.SayHello()
15
16         print undefined_variable
17
18
19     def methodNotWithSelf(foo):
20         pass
21
22
23 #####
24 # main
25
26
27 if __name__ == '__main__':
28     unittest.main()
```

The status bar at the bottom indicates "Writable" and "28: 20".

Option D: integrated development environment (IDE)

Spyder (Windows, Linux, Mac)



Option D: integrated development environment (IDE)

What you get

- ▶ Syntax highlighting
- ▶ Tab completion
- ▶ Command history
- ▶ Debugging tools
- ▶ Static code analysis

Table of Contents

What is Python?

How to run Python?

Let's get coding

Syntax: assignment

```
variable = object
```

Properties

- ▶ Every piece of data is an object
- ▶ A variable is a reference to an object
- ▶ The object stays alive as long as some variable points to it
(reference counted)

Datatypes: integer

```
a = 3
```

Properties

- ▶ Immutable

Datatypes: integer

```
a = 3
```

Properties

- ▶ Immutable

```
a = a + 1  
a += 1
```

Datatypes: integer

```
a = 3
```

Properties

- ▶ Immutable

```
a = a + 1  
a += 1
```

```
a = 1/2 # a = 0
```

Datatypes: bool

```
a = True  
a = False
```

Properties

- ▶ Immutable

Datatypes: float

```
a = 2.0
```

```
a = 2e-3
```

Datatypes: float

```
a = 2.0  
a = 2e-3
```

Properties

- ▶ Immutable

```
print a <= 0, a+1
```

```
False, 1.002
```

Datatypes: str

```
a = "spam"  
a = 'spam'  
a = "hello\nworld"
```

Properties

- ▶ Immutable

Datatypes: str

```
a = "spam"  
a = 'spam'  
a = "hello\nworld"
```

Properties

- ▶ Immutable

```
print a  
print "world" in a, "spam" + "eggs", "spam" * 3
```

```
hello  
world  
True, spameggs, spams(spamspam
```

Datatypes: str

```
a = "spam"  
a = 'spam'  
a = "hello\nworld"
```



```
hello  
world  
True, spameggs, spamspamspam
```

Datatypes: set

```
a = {}      # empty set
a = {"spam", 3}
a = {3}
a = {1, 2, 3}
```

Properties

- ▶ Unordered
- ▶ No duplicates
- ▶ Mutable

Datatypes: set

```
a = {}      # empty set
a = {"spam", 3}
a = {3}
a = {1, 2, 3}
```

Properties

- ▶ Unordered
- ▶ No duplicates
- ▶ Mutable

```
print 2 in a, 5 in a
print a.union({2, 5})
```

```
True, False
set([1, 2, 3, 5])
```

Datatypes: list

```
a = ["spam", 3]  
a = []      # empty list  
a = [1, 2, 3]
```

Properties

- ▶ Ordered
- ▶ Duplicates allowed
- ▶ Mutable

Datatypes: list

```
a = ["spam", 3]  
a = []      # empty list  
a = [1, 2, 3]
```

Properties

- ▶ Ordered
- ▶ Duplicates allowed
- ▶ Mutable

```
print a[0], 2 in a, a.index(2)
```

```
1, True, 1
```

Datatypes: tuple

```
a = (1, 2, 3)
a = ("spam", 3)
a = (3,) # note trailing comma
a = ()   # empty tuple
```

Properties

- ▶ Ordered
- ▶ Duplicates allowed
- ▶ Immutable

Datatypes: dict

```
a = {"spam": 3, "eggs": 1}
a = {3 : "spam", (1,2): "eggs"}
a = dict()    # empty dictionary
a = dict(spam=3, eggs=1)
```

Properties

- ▶ Unordered
- ▶ No duplicate keys allowed
- ▶ Immutable

Keys must be of immutable type



Datatypes: dict

```
a = dict(spam=3, eggs=1)
```

```
print "spam" in a, "eggs" in a, "foo" in a
print "spam" in a.keys()
print 3 in a, 3 in a.values()
```

```
True True False
True
False True
```

Syntax: assignment (2)

Recall

- ▶ Every piece of data is an object
- ▶ A variable is a reference to an object
- ▶ The object stays alive as long as some variable points to it
(reference counting)

```
a = {"spam": 3, "eggs": 1}  
a = 6
```

Syntax: assignment (2)

Recall

- ▶ Every piece of data is an object
- ▶ A variable is a reference to an object
- ▶ The object stays alive as long as some variable points to it
(reference counting)

```
a = [1, 2, 3]
b = a
b[0] = 7
print a
```

```
[7, 2, 3]
```

Objects

Observation

- ▶ Everything is an object ...
- ▶ ... so everything has methods and attributes

Objects

This is bashrc enabled bat.
The system cannot find the drive specified.

```
C:\Users\jg>ipython
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500
Type "copyright", "credits" or "license" for more information

IPython 2.1.0 -- An enhanced Interactive Python.
?            -> Introduction and overview of IPython's features
%quickref  -> Quick reference.
help        -> Python's own help system.
object?     -> Details about 'object', use 'object??' for e

In [1]: a = "spam"

In [2]:
```

Objects

This is bashrc enabled bat.
The system cannot find the drive specified.

```
C:\Users\jg>ipython
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500
Type "copyright", "credits" or "license" for more informa
IPython 2.1.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's featu
quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for e
In [1]: a = "spam"
In [2]: a.
```

Objects

```
ipython
The system cannot find the drive specified.

C:\Users\jg>ipython
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500
Type "copyright", "credits" or "license" for more informa
IPython 2.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's featu
%quickref -> Quick reference.
help     -> Python's own help system.
object?   -> Details about 'object', use 'object??' for e

In [1]: a = "spam"

In [2]: a.
a.capitalize a.find      a.isspace    a.partition a.rst
a.center      a.format    a.istitle    a.replace    a.spl
a.count       a.index     a.isupper   a.rfind     a.spl
a.decode      a.isalnum   a.join      a.rindex   a.sta
a.encode      a.isalpha   a.ljust     a.rjust    a.str
a.endswith   a.isdigit   a.lower    a.rpartition a.swa
a.expandtabs a.islower  a.lstrip   a.rsplit    a.tit

In [2]: a.
```

Objects

```
ipython
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500
Type "copyright", "credits" or "license" for more informa
IPython 2.1.0 -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's featu
quickref -> Quick reference.
help        -> Python's own help system.
object?     -> Details about 'object', use 'object??' for e

In [1]: a = "spam"

In [2]: a.
a.capitalize a.find      a.isspace    a.partition a.rst
a.center      a.format    a.istitle   a.replace    a.spl
a.count       a.index     a.isupper  a.replace    a.spl
a.decode      a.isalnum   a.join     a.rfind     a.spl
a.encode      a.isalpha   a.ljust    a.rindex   a.sta
a.endswith   a.isdigit   a.lower   a.rjust    a.str
a.expandtabs a.islower  a.lstrip  a.rpartition a.swa
a.rstrip      a.title

In [2]: a.capitalize()
Out[2]: 'Spam'

In [3]:
```

Objects

Observation

- ▶ Everything is an object ...
- ▶ ... so everything has methods
- ▶ ... and everything has documentation

Objects

```
ipython

In [1]: a = "spam"

In [2]: a.
a.capitalize a.find      a.isspace    a.partition  a.rstrip     a.tr
a.center      a.format    a.istitle    a.replace    a.split     a.up
a.count       a.index     a.isupper   a.rfind     a.splitlines a.zf
a.decode      a.isalnum   a.join      a.rindex    a.startswith
a.encode      a.isalpha   a.ljust     a.rjust    a.strip
a.endswith   a.isdigit   a.lower    a.rpartition a.swapcase
a.expandtabs  a.islower  a.lstrip    a.rsplit    a.title

In [2]: a.capitalize()
Out[2]: 'Spam'

In [3]: a.capitalize?
Type:           builtin_function_or_method
String form:   <built-in method capitalize of str object at 0x033A4300>
Docstring:
S.capitalize() -> string

Return a copy of the string S with only its first character
capitalized.

In [4]:
```

Objects

Observation

- ▶ Everything is an object ...
- ▶ ... so everything has methods
- ▶ ... and everything has documentation
- ▶ ... **everything has a type**

Objects

```
ipython

a.capitalize a.find      a.isspace    a.partition a.rstrip    a.tr
a.center     a.format    a.istitle    a.replace    a.split     a.up
a.count      a.index    a.isupper   a.rfind     a.splitlines a.zf
a.decode     a.isalnum  a.join     a.rindex    a.startswith
a.encode     a.isalpha   a.ljust    a.rjust     a.strip
a.endswith   a.isdigit  a.lower    a.rpartition a.swapcase
a.expandtabs a.islower a.lstrip   a.rsplit    a.title

In [2]: a.capitalize()
Out[2]: 'Spam'

In [3]: a.capitalize?
Type:           builtin_function_or_method
String form:   <built-in method capitalize of str object at 0x033A4300>
Docstring:
S.capitalize() -> string

Return a copy of the string S with only its first character
capitalized.

In [4]: type(a)
Out[4]: str

In [5]:
```

Objects

```
ipython
```

a.decode	a.isalnum	a.join	a.rindex	a.startswith
a.encode	a.isalpha	a.ljust	a.rjust	a.strip
a.endswith	a.isdigit	a.lower	a.rpartition	a.swapcase
a.expandtabs	a.islower	a.lstrip	a.rsplit	a.title

```
In [2]: a.capitalize()
Out[2]: 'Spam'

In [3]: a.capitalize?
Type:           builtin_function_or_method
String form:   <built-in method capitalize of str object at 0x033A4300>
Docstring:
S.capitalize() -> string

Return a copy of the string S with only its first character
capitalized.

In [4]: type(a)
Out[4]: str

In [5]: isinstance(a,tuple)
Out[5]: False

In [6]:
```

Objects

```
ipython
a.expandtabs a.islower      a.lstrip      a.rsplit      a.title
In [2]: a.capitalize()
Out[2]: 'Spam'

In [3]: a.capitalize?
Type:           builtin_function_or_method
String form:   <built-in method capitalize of str object at 0x033A4300>
Docstring:
S.capitalize() -> string

Return a copy of the string S with only its first character
capitalized.

In [4]: type(a)
Out[4]: str

In [5]: isinstance(a,tuple)
Out[5]: False

In [6]: isinstance(a,str)
Out[6]: True

In [7]:
```

Functions

```
def spam():
    return "eggs"

print spam()
```

```
eggs
```

Indentation determines scope

Function arguments

```
def spam(a,b,c):  
    return a+b+c  
  
print spam(1,2,3)
```

6

Function arguments

```
def spam(*args):
    print args
    return sum(args)

print spam(1, 2, 3)
print spam(1, 2, 3, 4)
```

```
(1, 2, 3)
6
(1, 2, 3, 4)
10
```

Function arguments

```
def spam(a,b,c="everybody"):  
    print "Greetings to %s! -- %d" % (c,a+b)  
  
spam(1,2)  
spam(3,4,c="you")
```

```
Greetings to everybody! -- 3  
Greetings to you! -- 7
```

Function arguments

```
def spam(*args, **kwargs):  
    print "Thanks for these arguments:"  
    print "  args:", args  
    print "  kwargs:", kwargs  
  
spam("eggs", 3, boilingtime=12, salt=True)
```

```
Thanks for these arguments:  
  args: ('eggs', 3)  
  kwargs: {'boilingtime': 12, 'salt': True}
```



Function arguments

Arguments are always passed by reference

```
def spam(eggs):  
    eggs.append(2)  
  
my_eggs = []  
spam(my_eggs)  
  
print my_eggs
```

```
[2]
```

Function arguments

```
def factorial(n):  
    if n==0:  
        return 1  
    else:  
        return n*factorial(n-1)  
  
print factorial(5)
```

24

Function arguments

```
def factorial(n,partials=[]):
    partials.append(n)
    print partials
    if n==0:
        return 1
    else:
        return n*factorial(n-1,partials)

print factorial(4)
```

```
[4]
[4, 3]
[4, 3, 2]
[4, 3, 2, 1]
[4, 3, 2, 1, 0]
24
```

A common pitfall



Function arguments

```
print factorial(4)
```

```
[4, 3, 2, 1, 0, 4]  
[4, 3, 2, 1, 0, 4, 3]  
[4, 3, 2, 1, 0, 4, 3, 2]  
[4, 3, 2, 1, 0, 4, 3, 2, 1]  
[4, 3, 2, 1, 0, 4, 3, 2, 1, 0]  
24
```

Function arguments

```
def factorial(n,partials=None):
    if partials is None:
        partials = []
    partials.append(n)
    print partials
    if n==0:
        return 1
    else:
        return n*factorial(n-1,partials)

print factorial(4)
print factorial(4)
```

```
[4]
[4, 3]
[4, 3, 2]
[4, 3, 2, 1]
[4, 3, 2, 1, 0]
24
[4]
```

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]

for p in teachers:
    print p.capitalize()
```

Moritz
Joel
Greg
Joris

Flow control

```
for i in range(4):  
    print i
```

```
0  
1  
2  
3
```

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]

for i in range(len(teachers)):
    print i, teachers[i]
```

```
0 moritz
1 joel
2 greg
3 joris
```

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]

for i in range(len(teachers)):
    print i, teachers[i]
```

```
0 moritz
1 joel
2 greg
3 joris
```



This is **unpythonic**

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]

for i, p in enumerate(teachers):
    print i, p
```

```
0 moritz
1 joel
2 greg
3 joris
```

This is pythonic

List comprehension

```
teachers = "moritz joel greg joris".split(" ")  
  
namelengths = []  
for p in teachers:  
    namelengths.append(len(p))  
  
print namelengths
```

```
[6, 4, 4, 5]
```

This is **unpythonic**

List comprehension

```
teachers = "moritz joel greg joris".split(" ")  
  
namelengths = [len(p) for p in teachers]  
  
print namelengths
```

```
[6, 4, 4, 5]
```

This is pythonic

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]
homes = ["Freiburg", "Barcelona", "Freiburg", "Leuven"]

for i in range(len(teachers)):
    print "%s lives in %s" % (teachers[i], homes[i])
```

```
moritz lives in Freiburg
joel lives in Barcelona
greg lives in Freiburg
joris lives in Leuven
```

This is **unpythonic**

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]
homes = ["Freiburg", "Barcelona", "Freiburg", "Leuven"]

for t, p in zip(teachers, homes):
    print "%s lives in %s" % (t.capitalize(), p)
```

```
Moritz lives in Freiburg
Joel lives in Barcelona
Greg lives in Freiburg
Joris lives in Leuven
```

This is pythonic

Flow control

```
teachers = ["moritz", "joel", "greg", "joris"]
homes = ["Freiburg", "Barcelona", "Freiburg", "Leuven"]

print zip(teachers,homes)
```

```
[('moritz', 'Freiburg'), ('joel', 'Barcelona'),
 ('greg', 'Freiburg'), ('joris', 'Leuven')]
```



Flow control

```
[('moritz', 'Freiburg'), ('joel', 'Barcelona'),  
 ('greg', 'Freiburg'), ('joris', 'Leuven')]
```

```
for t, p in zip(teachers, homes):  
    code
```

Flow control

```
[('moritz', 'Freiburg'), ('joel', 'Barcelona'),  
 ('greg', 'Freiburg'), ('joris', 'Leuven')]
```

```
for t, p in zip(teachers, homes):  
    code
```

```
t, p = ('moritz', 'Freiburg')  
code  
t, p = ('joel', 'Barcelona')  
code  
...  
...
```

Flow control

```
d = dict()  
  
for t, p in zip(teachers,homes):  
    d[t] = p  
  
print d
```

```
{'joris': 'Leuven', 'joel': 'Barcelona',  
'greg': 'Freiburg', 'moritz': 'Freiburg'}
```

Flow control

```
d = dict(zip(teachers,homes))  
  
print d
```

```
{'joris': 'Leuven', 'joel': 'Barcelona',  
'greg': 'Freiburg', 'moritz': 'Freiburg'}
```

More pythonic

Flow control

```
for t, p in d.items():
    print "%s lives in %s" % (t.capitalize(), p)
```

```
Moritz lives in Freiburg
Joel lives in Barcelona
Greg lives in Freiburg
Joris lives in Leuven
```

More pythonic

Using packages

The creative cycle in Python

1. Imagine it (ascii art)
2. Google it (pyfiglet)
3. Install it (`pip install pyfiglet` – Windows, Linux, Mac)
4. Import it

Using packages

```
import pyfiglet

teachers = ["moritz", "joel", "greg", "joris"]

for t in teachers:
    print pyfiglet.figlet_format(t, font="starwars")
```

Using packages

```
import pyfiglet

teachers = ["moritz", "joel", "greg", "joris"]

for t in teachers:
    print pyfiglet.figlet_format(t, font="starwars")
```

```
  ____  ____ .  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \
 | \ / | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | \ / | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / |
```

```
  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \  ____  ____ \
 | \ / | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | \ / | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / | | / \ | / |
 | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / | | \ / | \ / |
```

Using packages

```
import pyfiglet as pf

teachers = ["moritz", "joel", "greg", "joris"]

for t in teachers:
    print pf.figlet_format(t, font="starwars")
```

A large, stylized representation of the word "moritz" using the "starwars" font. The letters are composed of various line segments and punctuation marks, such as slashes, dots, and dashes, arranged to form the characters 'm', 'o', 'r', 'z', and 'i'. The entire word is contained within a single rectangular frame.

A smaller, stylized representation of the word "moritz" using the "starwars" font. This version is less dense than the first, showing more individual characters and some internal structure. It is also contained within a rectangular frame.

Using packages

```
from pyfiglet import figlet_format

teachers = ["moritz", "joel", "greg", "joris"]

for t in teachers:
    print figlet_format(t, font="starwars")
```

```
 .--.   .--.   .--.   .--.   .--.   .--.   .--.   .--.
| | \ / | | / \ | | \ / | | \ / | | \ / | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / | | \ / | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / | | \ / | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / | | \ / | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / | | \ / | | \ / | | \ / |
```

```
.--.   .--.   .--.   .--.
| | \ / | | / \ | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / |
| | \ / | | / \ | | \ / | | \ / |
\ / \ / \ / \ / \ / \ / \ / \ /
```

Using packages

```
from pyfiglet import figlet_format as asciiart

teachers = ["moritz", "joel", "greg", "joris"]

for t in teachers:
    print asciiart(t, font="starwars")
```

A large, detailed ASCII art representation of the word "moritz" in a Star Wars font. The letters are composed of various symbols like slashes, dots, and underscores, arranged to form the shape of the letters. The entire word is contained within a rectangular border.A smaller, less detailed ASCII art representation of the word "moritz" in a Star Wars font. It is similar in style to the larger one but is scaled down. It is also enclosed in a rectangular border.

Using packages

```
from pyfiglet import *
teachers = ["moritz", "joel", "greg", "joris"]
for t in teachers:
    print figlet_format(t, font="starwars")
```

The watermark consists of two identical versions of the word 'Pyfiglet' rendered in a blocky, pixelated font. The characters are formed by various line segments and dots, giving them a digital, binary appearance. The two instances of the word overlap slightly in the center.

A single instance of the word 'Pyfiglet' rendered in a blocky, pixelated font, similar in style to the larger watermark above. It is positioned at the bottom left of the slide.

Slicing

```
teachers = ["moritz", "joel", "greg", "joris"]

print teachers[0]
print teachers[-1]
print teachers[:2]
print teachers[1:]
print teachers[::2]
print teachers[1::2]
```

```
moritz
joris
['moritz', 'joel']
['joel', 'greg', 'joris']
['moritz', 'greg']
['joel', 'joris']
```

Iterators

```
def fibonacci(a,b):  
    yield a  
    yield b  
    n2, n1 = a, b  
    while True:  
        yield n1+n2  
        n2, n1 = n1, n1+n2  
  
for i in fibonacci(1,1):  
    print i  
    if i > 20: break
```

```
1  
1  
2  
3  
5  
8  
13  
21
```

Scope

```
def simulate(a):  
    return a**2  
  
print simulate(3)
```

9

Scope

```
cache = {12: 144}

def simulate(a):
    if a in cache:
        return cache[a]
    else:
        print "working hard"
        return a**2

print simulate(3)
print simulate(12)
```

```
working hard
9
144
```

Scope

```
cache = {}

def simulate(a):
    if a in cache:
        return cache[a]
    else:
        print "working hard"
        cache[a] = result = a**2
    return result
```



Scope

```
print simulate(3)
print simulate(3)
cache = {}
print simulate(3)
```

```
working hard
```

```
9
```

```
9
```

```
working hard
```

```
9
```

Scope

```
cache = {}

def simulate(a, clear=False):
    if clear:
        cache = {}
    if a in cache:
        return cache[a]
    else:
        print "working hard"
        cache[a] = result = a**2
    return result
```

Scope

```
print simulate(3)
print simulate(3)
print simulate(3, clear=True)
```

```
4     if clear:
5         cache = {}
----> 6     if a in cache:
7         return cache[a]
8     else:
```

```
UnboundLocalError: local variable 'cache' referenced before
```

Scope

```
cache = {}

def simulate(a, clear=False):
    global cache
    if clear:
        cache = {}
    if a in cache:
        return cache[a]
    else:
        print "working hard"
        cache[a] = result = a**2
    return result
```

Scope

```
print simulate(3)
print simulate(3)
print simulate(3, clear=True)
```

```
working hard
```

```
9
```

```
9
```

```
working hard
```

```
9
```

Scope

```
def simulate(a, clear=False, cache={}):
    if clear:
        cache = {}
    if a in cache:
        return cache[a]
    else:
        print "working hard"
        cache[a] = result = a**2
    return result
```

Scope

```
print simulate(3)
print simulate(3)
print simulate(3, clear=True)
```

```
working hard
```

```
9
```

```
9
```

```
working hard
```

```
9
```

Functions are objects too

```
def speaker(message):
    def temp(receiver):
        print "Hey", receiver, "!", message
    return temp

python_promoter = speaker("Time to learn Python")

python_promoter("Moritz")
python_promoter("audience")
```

```
Hey Moritz ! Time to learn Python
Hey audience ! Time to learn Python
```

Functions are objects too

```
def speaker(message):
    return lambda r: "Hey " + r + " ! " + message

python_promoter = speaker("Time to learn Python")

print python_promoter("Moritz")
print python_promoter("audience")
```

```
Hey Moritz ! Time to learn Python
Hey audience ! Time to learn Python
```

Classes

```
class Model(object):
    """ Represents a model """
    def __init__(self, labels):
        self.labels = labels

pendulum = Model(["theta", "omega"])

print pendulum
```

```
<__main__.Model object at 0x0312A418>
```

Classes

```
type(pendulum)
```

```
__main__.Model
```

Classes

```
type(pendulum)
```

```
__main__.Model
```

```
pendulum?
```

```
Type: Model
```

```
String form: <__main__.Model object at 0x031AC050>
```

```
Docstring: Represents a model
```

Classes

```
class Model(object):
    """ Represents a model """

    def __init__(self, labels):
        self.labels = labels

    def __repr__(self):
        return "Model(" + str(self.labels) + ")"

pendulum = Model(["theta", "omega"])

print pendulum
```

```
Model(['theta', 'omega'])
```

Classes

pendulum?

Type: Model

String form: Model(['theta', 'omega'])

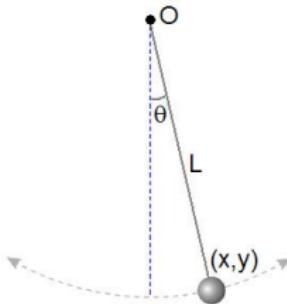
Docstring: Represents a model

Classes

```
class Pendulum(Model):
    def __init__(self, length=1.0):
        Model.__init__(self, ["theta", "omega"])
        self.length = length

    def describe(self):
        print "pendulum with length %d" % self.length

pendulum = Pendulum(2)
```



Classes

```
print pendulum
```

```
Model(['theta', 'omega'])
```

Classes

```
print pendulum
```

```
Model(['theta', 'omega'])
```

```
print pendulum.length
```

```
2
```

Classes

```
print pendulum
```

```
Model(['theta', 'omega'])
```

```
print pendulum.length
```

```
2
```

```
pendulum.length = 4
```

```
pendulum.describe()
```

```
pendulum with length 4
```

Getters and setters

```
class Pendulum(Model):
    def __init__(self, length=1.0):
        Model.__init__(self, ["theta", "omega"])
        self._length = length

    def getLength(self):
        return self._length

    def setLength(self, value):
        self._length = value

pendulum = Pendulum(2)
```

Getters and setters

```
pendulum.<tab>
```

```
pendulum.getLength  
pendulum.labels  
pendulum.setLength
```

Getters and setters

```
pendulum.setLength(5)  
print pendulum.getLength()
```

```
5
```

Getters and setters

```
pendulum.setLength("spam")
print pendulum.getLength()
```

```
spam
```

Getters and setters

```
pendulum.se  
print pendu
```

```
spam
```



Getters and setters

```
class Pendulum(Model):
    def __init__(self, length=1.0):
        Model.__init__(self, ["theta", "omega"])
        self._length = length

    def getLength(self):
        return self._length

    def setLength(self, value):
        if isinstance(value, int):
            self._length = value
        else:
            raise Exception("Expected number")
```

Getters and setters

```
pendulum = Pendulum(2)  
pendulum.setLength("spam")
```

Exception: Expected number

Getters and setters

```
def setLength(self, value):
    if isinstance(value, int):
        self._length = value
    else:
        raise Exception("Expected number")
```

This is **unpythonic**

Getters and setters

```
def setLength(self, value):  
    try:  
        self._length = int(value)  
    except:  
        raise Exception("Expected number")
```

This is pythonic

Getters and setters

```
def setLength(self, value):  
    try:  
        self._length = int(value)  
    except:
```

EAFF: Easier to ask for forgiveness than permission

This is pythonic

Getters and setters

```
pendulum.setLength(5)
```

Getters and setters

```
pendulum.setLength(5)
```

```
pendulum.setLength('5')
```

Getters and setters

```
pendulum.setLength(5)
```

```
pendulum.setLength('5')
```

```
pendulum.setLength('spam')
```

Exception: Expected number

Getters and setters

```
class Pendulum(Model):
    def __init__(self, length=1.0):
        Model.__init__(self, ["theta", "omega"])
        self.length = length

    @property
    def length(self):
        return self._length

    @length.setter
    def length(self, value):
        try:
            self._length = int(value)
        except:
            raise Exception("Expected number")

pendulum = Pendulum(2)
```

Getters and setters

```
pendulum.<tab>
```

```
pendulum.labels    pendulum.length
```

Getters and setters

```
pendulum.<tab>
```

```
pendulum.labels    pendulum.length
```

```
pendulum.length = "5"  
print pendulum.length
```

```
5
```

Getters and setters

```
pendulum.<tab>
```

```
pendulum.labels    pendulum.length
```

```
pendulum.length = "5"  
print pendulum.length
```

```
5
```

```
pendulum.length = "spam"
```

```
Exception: Expected number
```

Code reuse

Code Listing 1: models.py

```
class Model(object):  
    """ Represents a model """
```

...

```
class Pendulum(Model):
```

...

```
from models import Pendulum  
  
pendulum = Pendulum(3)
```

Code reuse

Code Listing 2: models.py

```
class Model(object):  
    """ Represents a  
    ...  
  
class PendulumModel:  
    ...
```

```
from models import Pendulum  
  
pendulum = Pendulum(3)
```



Code reuse

Code Listing 3: models.py

```
class Model(object):
    """ Represents a model """
    ...

class Pendulum(Model):
    ...

if __name__ == "__main__":
    p = Pendulum()
    p.describe()
```

Code reuse

```
$ python models.py
```

```
pendulum with length 1
```

Pickling

```
import pickle

teachers = ["moritz", "joel", "greg", "joris"]
namelengths = [len(p) for p in teachers]

results = (teachers, namelengths)
pickle.dump(results, file('results.pkl', 'w'))
```



Pickling

```
import pickle

teachers = ["moritz", "joel", "greg", "joris"]
namelengths = [len(p) for p in teachers]

results = (teachers, namelengths)
pickle.dump(results, file('results.pkl', 'w'))
```

```
import pickle

teachers, namelengths =
    pickle.load(file('results.pkl', 'r'))
print teachers, namelengths
```

```
['moritz', 'joel', 'greg', 'joris'] [6, 4, 4, 5]
```

Command line arguments

Code Listing 4: simulator.py

```
import numpy as np
import sys

N = int(sys.argv[1])
items = sys.argv[2:]

for i in range(N):
    n = len(items)
    print items[np.random.randint(n)]
```



Command line arguments

Code Listing 5: simulator.py

```
import numpy as np
import sys

N = int(sys.argv[1])
items = sys.argv[2:]

for i in range(N):
    n = len(items)
    print items[np.random.randint(n)]
```

```
$ python simulator.py 3 spam eggs
```

```
eggs
spam
spam
```

Command line arguments

```
$ python simulator.py spam eggs 3
```

```
ValueError                                     Traceback
C:\Users\jg\hello.py in <module>()
      2 import sys
      3
----> 4 N = int(sys.argv[1])
      5 items = sys.argv[2:]
      6
```

```
ValueError: invalid literal for int() with base 10: 'spam'
```

Command line arguments

Code Listing 6: simulator.py

```
try:  
    N = int(sys.argv[1])  
except:  
    raise Exception("""Oops. Expecting a number but  
                      you supplied me with '%s'.  
                      """ % sys.argv[1])
```

Exception: Oops. Expecting a number but
you supplied me with 'spam'.

Command line arguments

Code Listing 7: simulator.py

```
from argparse import *
import numpy as np

p = ArgumentParser(
    description="Simulate drawing from a collection")
p.add_argument('--N', type=int, default=10,
    help="The number of draws")
p.add_argument('items', type=str, nargs='+',
    metavar='item', help='the items to draw from')
p.add_argument('--special', action='store_true')

args = p.parse_args()

for i in range(args.N):
    n = len(args.items)
    print args.items[np.random.randint(n)]
```

Command line arguments

```
$ python simulator.py
```

```
usage: hello.py [-h] [--N N] [--special] item [item ...]
hello.py: error: too few arguments
```



Command line arguments

```
$ python simulator.py -h
```

```
usage: hello.py [-h] [--N N] [--special] item [item ...]

Simulate drawing from a collection

positional arguments:
  item      the items to draw from

optional arguments:
  -h, --help  show this help message and exit
  --N N      The number of draws
  --special
```

Command line arguments

```
$ python simulator.py --N 3 spamm eggs
```

```
spamm  
eggs  
eggs
```

Interacting with the shell

```
import subprocess

items = ["spam", "eggs"]

N = 3
p = subprocess.Popen(
    ["python", "simulator.py", "--N", "%d" % N] + items,
    stdout=subprocess.PIPE, stderr=subprocess.PIPE)

stdout, stderr = p.communicate()
print stdout
```

```
eggs
spam
eggs
```

Interacting with the shell

Conclusion

This should be enough background to explore the Python universe



Happy coding!