python

Administration

- Instructor:
  - 曾學文
  - Office: Room 908
  - Email: hwtseng@nchu.edu.tw
  - Tel: 04-22840497 ext. 908

- Office Hours:
  - (Wednesday) 13:30~15:00

- Grade:
  - Homework 40%
  - Computer-based Test 30%
  - Final Project 30 %

Outline

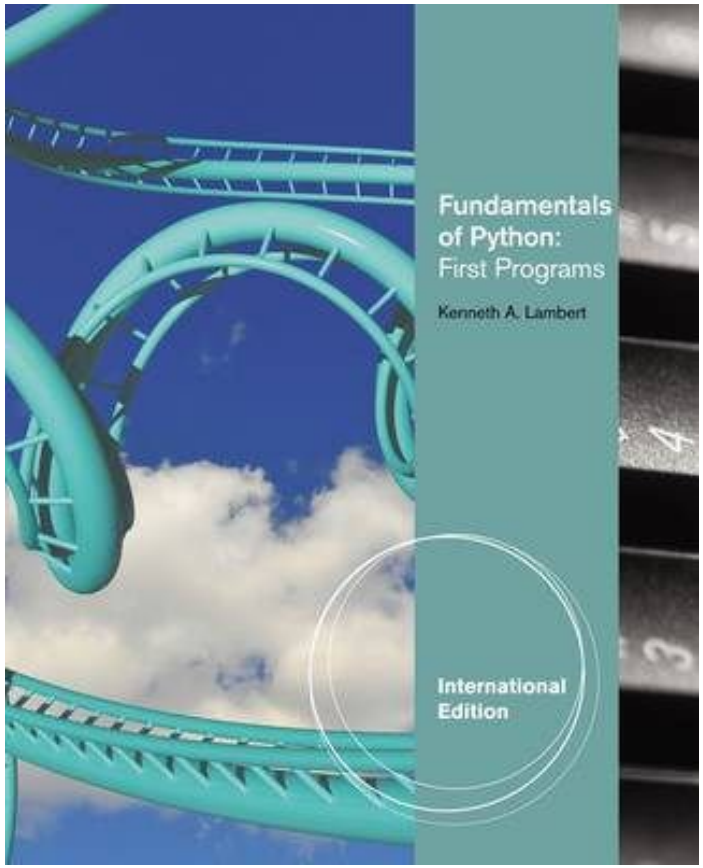
1. Python Introduction and Operation
2. Python Statement and Data Structure
3. Function and Module
4. Input and Output
5. Errors and Exception
6. Objects and Classes
7. Python GUI Programming
8. Python Network Programming
9. Thread
10. Python implement Mechanical Learning
11. Demo for Final Project

Introductory

- Raise your hand is always welcome!
- Slides are not enough. To master the materials, page-by-page reading is necessary.
- No phone, walk, sleep, and late during the lecture time.
- Do not copy the homeworks.
- You must spend time for programming.

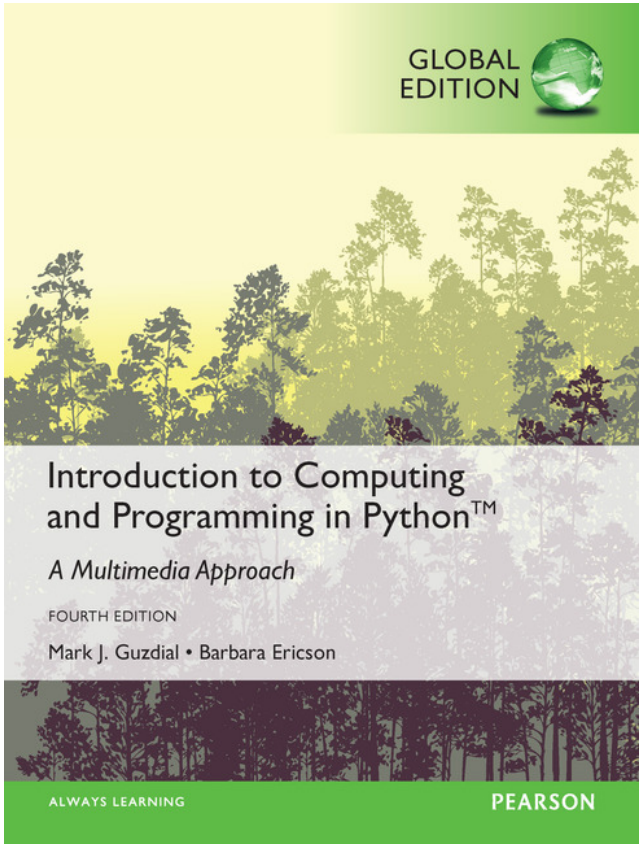
Reference Book

- Fundamentals of Python: First Programs, “Kenneth A. Lambert”, International Edition, ISBN: 1111822700



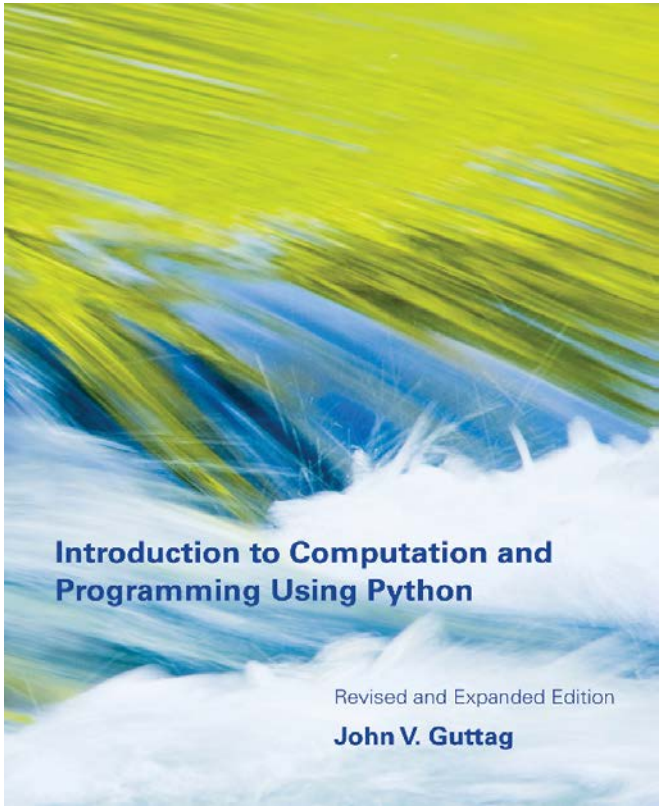
Reference Book

- Introduction to Computing and Programming in Python, “Mark J. Guzdial, Barbara Ericson”, Global Edition (4e), ISBN: 9781292109862

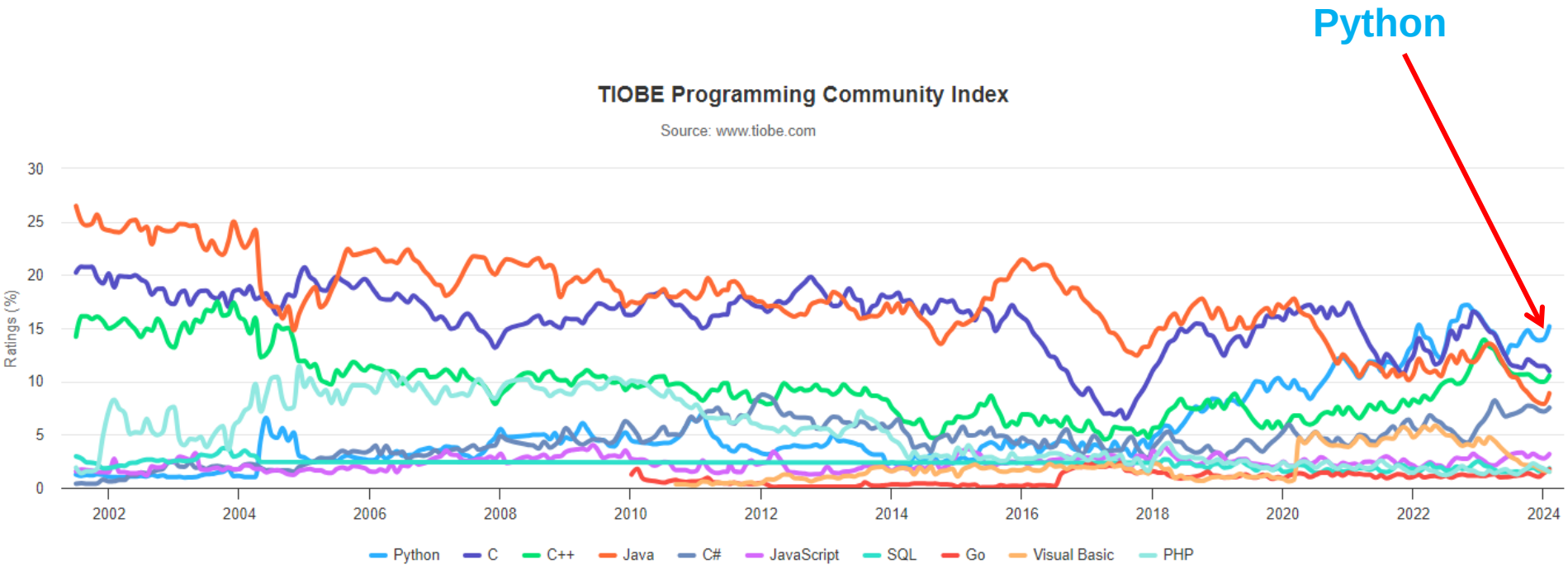


Reference Book

- Introduction to Computation and Programming Using Python, ” John V. Guttag ”, Revised And Expanded Edition, ISBN: 9780262316644




Microsoft
Language
Pascal
ABAP
C/AL
Cg
Rust
GDL
JOSS
Bistro
Cesil
Scala
SYMPL
Genie
Inform
HAL/S
SIMSCRIPT
Revolution
SenseTalk
Sawzall
Blue
Chapel
Rapira
CSKA
SIMPLE
ALGOL
DATATRIEVE
PL/I
ELAN
Ajax
Agora
Clojure
Not
Adenine
Trac
Object
Microcode
SAM76
code
Redcode
Autolt
Lynx
Logo
SALSA
ATS
Fjölnir
Fuxi
Assembly
TACPOL
FP
shell
language
E
A+
Lisp
VB
MicroScript
Squirrel
Windows/Dos
Kaleidoscope
IMP
Fancy
SuperTalk
P
K
Visual
JavaScript
FLJavaFX
R
Forth
b
T
Newspeak
LC-3
ABC
L
XOTcl
dc
Cayenne
Go
Delphi
DASL
D
S
Caml
Z
GameMonkey
ECMAScript
Prolog
GJ
JASS
bc
GM
Constraint
EusLisp
ObjectScript
Oxygene
G
PEARL
Programming
BLISS
notation
Ruby
EXEC
RPG
M
Cyclone
RPL
Script
PHP
CorVision
Action
REXX
F
Little
NET
Q
Simula
Boomerang
COBOL
GraphTalk
A
ChucK
Common
Agena
Strand
Turing
ISWIM
S/SL
PL/SQL
Arc
Moby
ORCA/Modula-2
Uniface
Python
J
Robot
SP/k
rex
HaXe
Eiffel
Coral
WATFOR


Python
TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
Python
C
C++
Java
C#
JavaScript
SQL
Go
Visual Basic
PHP

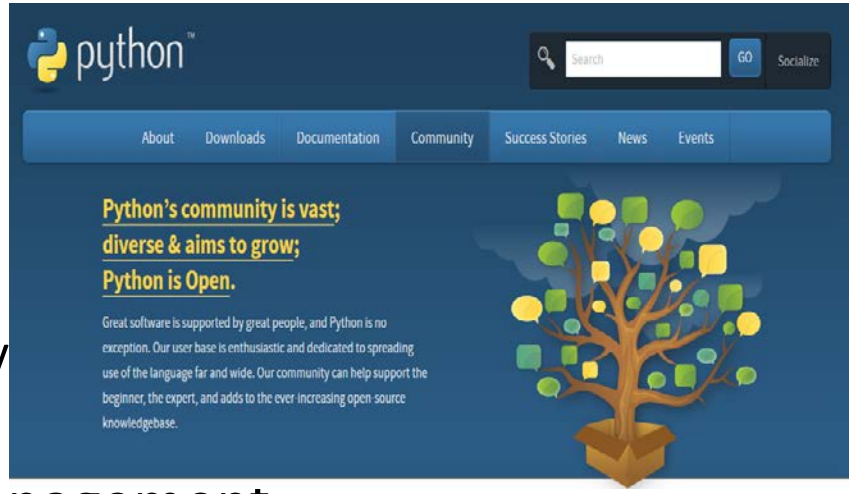
Feb 2024	Feb 2023	Change	Programming Language	Ratings	Change
1	1		Python	15.16%	-0.32%
2	2		C	10.97%	-4.41%
3	3		C++	10.53%	-3.40%
4	4		Java	8.88%	-4.33%
5	5		C#	7.53%	+1.15%
6	7	^	JavaScript	3.17%	+0.64%
7	8	^	SQL	1.82%	-0.30%
8	11	^	Go	1.73%	+0.61%
9	6	v	Visual Basic	1.52%	-2.62%
10	10		PHP	1.51%	+0.21%

History

Programming Language	2024	2019	2014	2009	2004	1999	1994	1989
Python	1	4	8	6	11	27	22	-
C	2	2	1	2	2	1	1	1
C++	3	3	4	3	3	2	2	3
Java	4	1	2	1	1	13	-	-
C#	5	6	5	8	8	30	-	-
JavaScript	6	8	9	9	9	20	-	-
Visual Basic	7	19	-	-	-	-	-	-
PHP	8	7	6	5	6	-	-	-
SQL	9	9	-	-	7	-	-	-
Assembly language	10	11	-	-	-	-	-	-
Objective-C	27	10	3	38	48	-	-	-
Lisp	33	28	14	17	15	12	7	2
(Visual) Basic	-	-	7	4	5	3	3	7

Python

- Simple
  - Python is a simple and minimalistic language in nature
  - Reading a **good** python program should be like reading English
  - Its Pseudo-code nature allows one to concentrate on the problem rather than the language
- Easy to Learn
- Free & Open source
  - Freely distributed and Open source
  - Maintained by the Python community http://www.python.org/community/
- High Level Language – memory management
- Portable – runs on anywhere and combines with c code



Python

- Interpreted
  - You run the program straight from the source code.
  - Python program →Bytecode →a platform native language
  - You can just copy over your code to another system and it will auto-magically work with python platform
- Object-Oriented
  - Simple and additionally supports procedural programming
- Extensible – easily import other code
- Embeddable – easily place your code in non-python programs
- Extensive libraries
  - (i.e. reg. expressions, doc generation, CGI, ftp, web browsers, ZIP, WAV, cryptography, etc...) (wxPython, Twisted, Python Imaging library)

Python Timeline/History

- Python was conceived in the late 1980s.
  - Guido van Rossum (吉多·范羅蘇姆),
  - Benevolent Dictator For Life (仁慈獨裁者)
  - Rossum is Dutch, born in Netherlands
  - Descendant of ABC, he wrote glob() func in UNIX
  - M.D. @ U of Amsterdam, worked for CWI, NIST, CNRI, Google
  - Also, helped develop the ABC programming language
  - Monty Python's Flying Circus (蒙提·派森的飛行馬戲團)
- In 1991 python 0.9.0 was published and reached the masses through alt.sources
  - The alt.sources newsgroup is intended to be a repository for source-code of all sorts that people wish to distribute and share with other people.

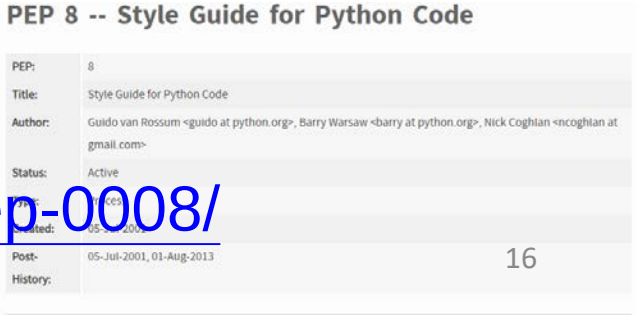
ABC is an imperative general-purpose programming language and programming environment developed at CWI, Netherlands by Leo Geurts, Lambert Meertens, and Steven Pemberton. It is interactive, structured, high-level, and intended to be used instead of BASIC, Pascal, or AWK.

Python Timeline/History

- In January of 1994 python 1.0 was released
  - Functional programming tools like lambda, map, filter, and reduce
  - comp.lang.python formed, greatly increasing python's user base
- In 1995, python 1.2 was released.
- By version 1.4 python had several new features
  - Keyword arguments (similar to those of common lisp)
  - Built-in support for complex numbers
  - Basic form of data-hiding through name mangling (easily bypassed)
    - private, protected, public
- Computer Programming for Everybody initiative
  - Make programming accessible to more people, with basic "literacy" similar to those required for English and math skills for some jobs.
  - Project was funded by DARPA (Defense Advanced Research Projects Agency)

Python Timeline/History

- In 2000, Python 2.0 was released.
  - Introduced list comprehensions similar to Haskells
    - Haskell is a modern functional language (like lisp)
  - Introduced garbage collection
- In 2001, Python 2.2 was released.
  - Included unification of types and classes into one hierarchy, making pythons object model purely object-oriented
  - Generators were added (function-like iterator behavior)
    - **iterator** is an object that enables a programmer to traverse a container.
- Standards
  - http://www.python.org/dev/peps/pep-0008/



PEP 8 -- Style Guide for Python Code

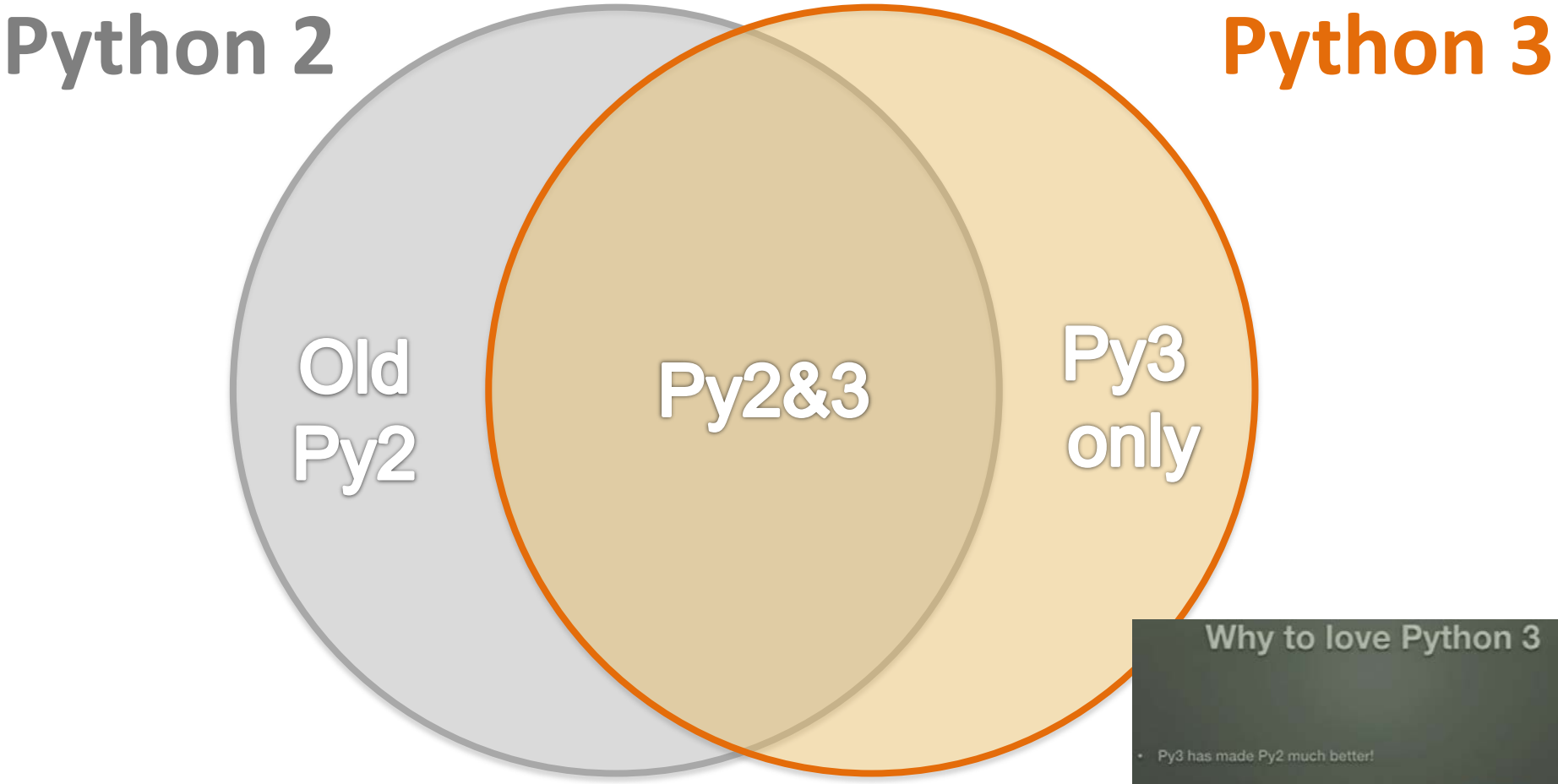
PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Version Release Dates

- Python 1.0 - January 1994
  - Python 1.5 - December 31, 1997
  - Python 1.6 - September 5, 2000
- Python 2.0 - October 16, 2000
  - Python 2.1 - April 17, 2001
  - Python 2.2 - December 21, 2001
  - Python 2.3 - July 29, 2003
  - Python 2.4 - November 30, 2004
  - Python 2.5 - September 19, 2006
  - Python 2.6 - October 1, 2008
  - Python 2.7 - July 3, 2010
- Python 3.0 - December 3, 2008
  - Python 3.1 - June 27, 2009
  - Python 3.2 - February 20, 2011
  - Python 3.3 - September 29, 2012
  - Python 3.4 - March 16, 2014
  - Python 3.5 - September 13, 2015
  - Python 3.6 - December 23, 2016
  - Python 3.7 - June 27 2018
  - Python 3.8 - October 14 2019
  - Python 3.9 - October 05 2020
  - Python 3.10 - October 04 2021
  - Python 3.11 - October 24 2022
  - Python 3.12 - October 02 2023

Python Taiwan
https://www.facebook.com/groups/pythontw/10152295869513438/

Python (Python 2.7 & Python 3.x) Standard Syntax



Source: PyCon Australia (2014), Writing Python 2/3 compatible code by Edward Schofield

https://www.youtube.com/watch?v=KOqk8j11aAI

Running Python

- There are three different ways to start Python:

(1) Interactive Interpreter:

  – You can enter **python** and start coding right away in the interactive interpreter by starting it from the command line.

```
$python              # Unix/Linux

or

python%              # Unix/Linux

or

C:>python            # Windows/DOS
```

Interactive Interpreter

- Here is the list of all the available command line options:

Option	Description
-d	provide debug output
-O	generate optimized bytecode (resulting in .pyo files)
-S	do not run import site to look for Python paths on startup
-v	verbose output (detailed trace on import statements)
-X	disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6
-c cmd	run Python script sent in as cmd string
file	run Python script from given file

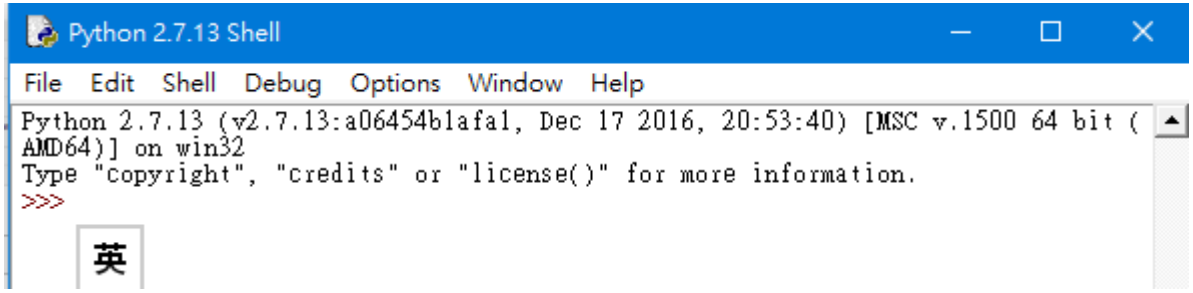
Script from the Command-line

- A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

```
$python  script.py          # Unix/Linux

or

python% script.py           # Unix/Linux

or

C:>python script.py         # Windows/DOS
```

Integrated Development Environment (IDE)

- You can run Python from a graphical user interface (GUI) environment as well.
  - All you need is a GUI application on your system that supports Python.
- **Unix:** IDLE is the very first Unix IDE for Python.
- **Windows:** PythonWin is the first Windows interface for Python and is an IDE with a GUI.
- **Macintosh:** The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.



INSTALL ANACONDA

DOWNLOAD ANACONDA

- https://www.anaconda.com/products/individual

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\user

C:\Users\user\untitled5.py

untitled5.py*

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 19 15:13:18 2021

@author: user
"""


print("Hello")
```

Source Console Object

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

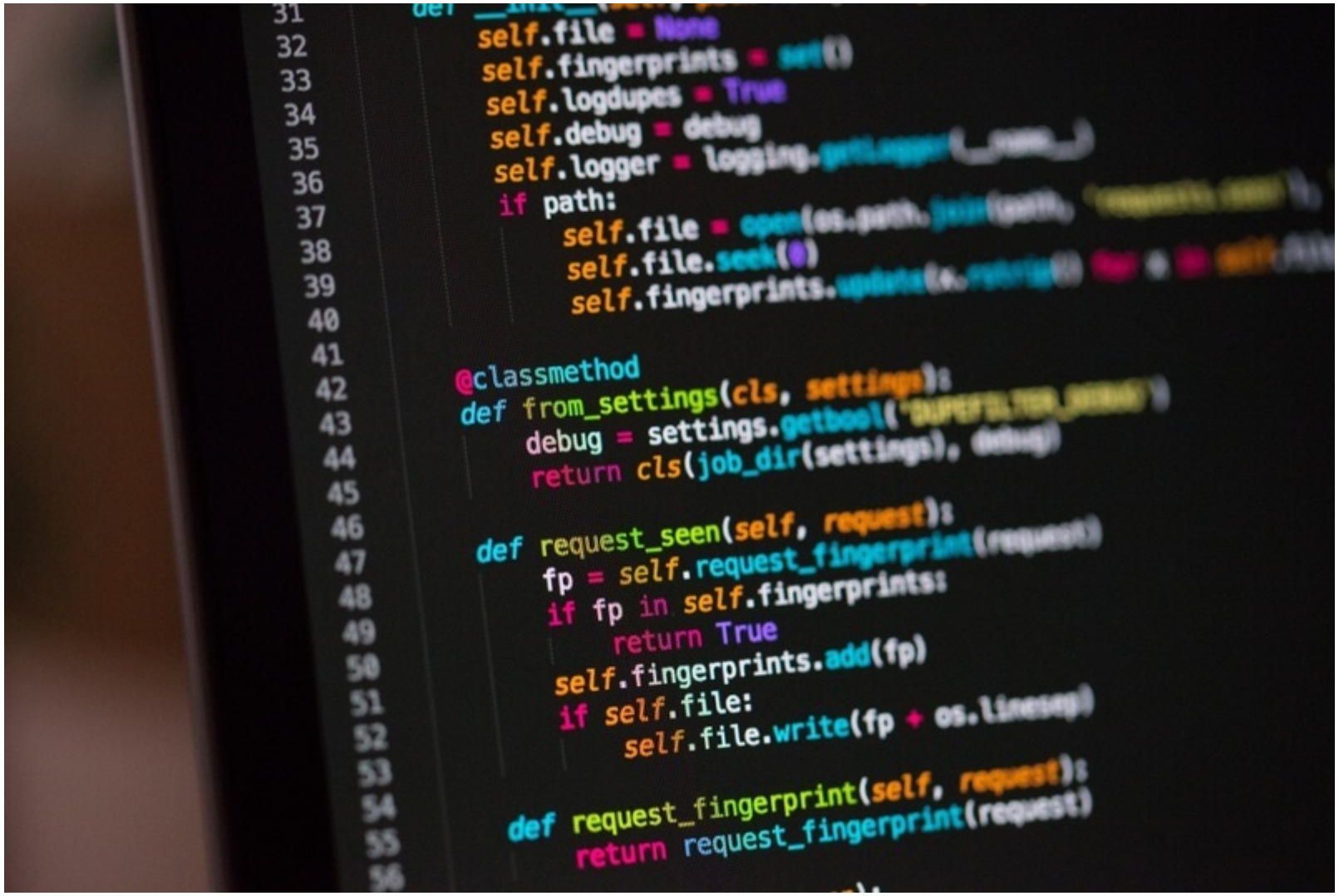
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Variable explorer Help Plots Files

Console 1/A

```
In [5]: runfile('C:/Users/user/untitled5.py', wdir='C:/Users/user')
Hello

In [6]:
```


self.file = None
self.fingerprints = set()
self.logdupes = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
self.file = open(os.path.join(path, 'requests.seen'),
self.file.seek(0)
self.fingerprints.update(x.rstrip()
@classmethod
def from_settings(cls, settings):
debug = settings.getbool('DUPEFILTER_DEBUG')
return cls(job_dir(settings), debug)
def request_seen(self, request):
fp = self.request_fingerprint(request)
if fp in self.fingerprints:
return True
self.fingerprints.add(fp)
if self.file:
self.file.write(fp + os.linesep)
def request_fingerprint(self, request):
return request_fingerprint(request)