

Programming in Python

01 – Python 개요, 프로그래밍 환경



2016년 8월, 국민대학교 컴퓨터공학부

100만 명이 넘는 사람들이 Python 을 이용하고 있지만...



py·thon

US [ˈpaɪθɑːn] | UK [ˈpaɪθən]

Derivation 형용사형 pythonic

Publisher ? **Oxford** Dong-A YBM E-E Dict.

Noun

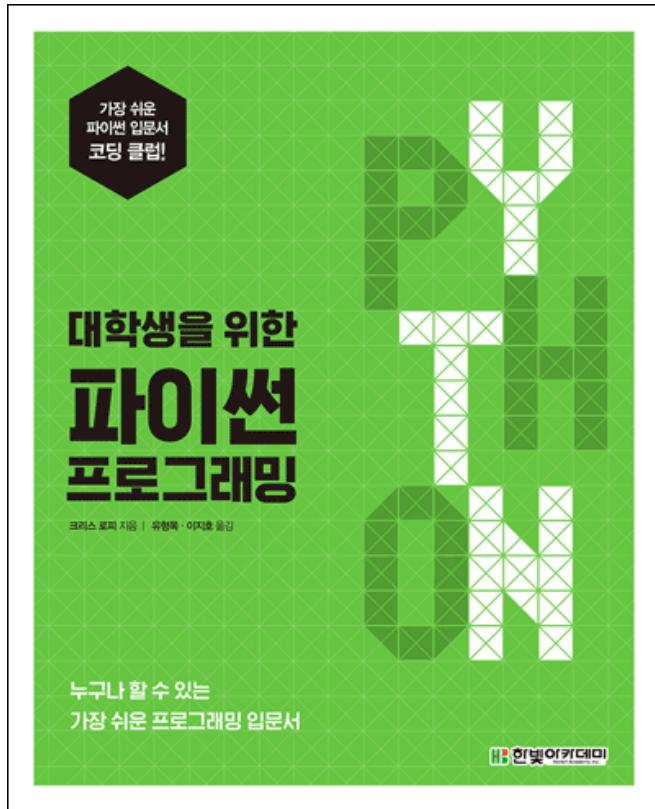
Noun

비단뱀

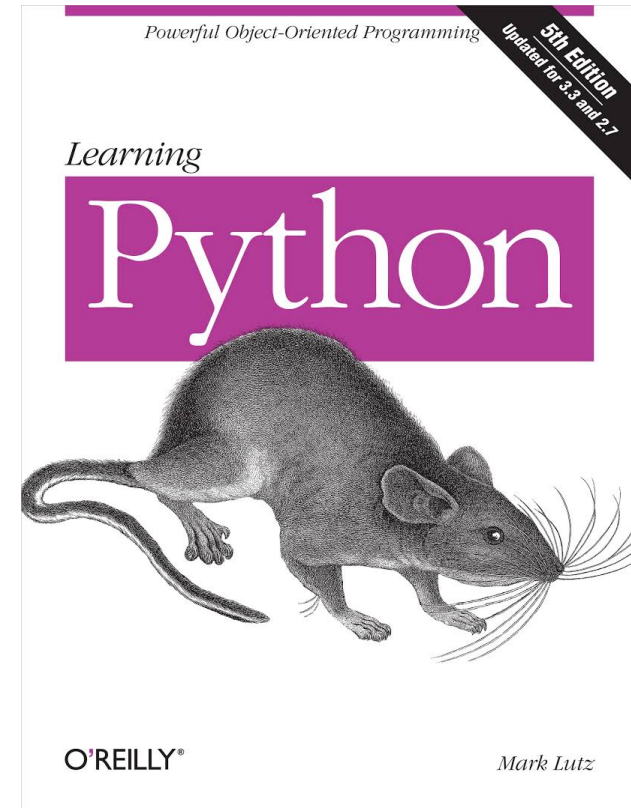


Monty Python's Flying Circus, 1969

참고자료



크리스 로피 지음, 유형목, 이지호 옮김,
대학생을 위한 파이썬 프로그래밍.
한빛아카데미



Mark Lutz, *Learning Python*, 5th Ed.
O'Reilly

Python 2.7 - <https://docs.python.org/2.7/contents.html>
Python 3 - <https://docs.python.org/3/contents.html>

왜 하필 (내가 왜?)

- Software quality
 - 코드가 읽고 이해하기 쉬우며, 재사용성이 높다.
- Developer productivity
 - 프로그래머가 작성해야 하는 코드가 짧고 (C++, Java 등의 1/5 수준) 익히기 쉽다.
- Program portability
 - 존재하는 (거의) 모든 플랫폼에서 실행 가능하다.
- Support libraries
 - 안정적인 라이브러리가 (거의 모두 공개) 풍부하게 제공된다.
- Component integration
 - C, C++, Java, .NET, COM, SOAP, XML-RPC, CORBA, ...
- Enjoyment
 - 재미있다?

Python 을 이용하여 서버, 클라이언트 등을 포함한
거의 모든 소프트웨어의 개발이 가능하고
실제로 그런 개발이 이루어지고 있음

Python 으로 할 수 있는 일들


- Systems programming
 - 시스템 관리 도구 및 유틸리티 등
- GUIs
 - tkinter (Tk), PyQt, PyGTK 등을 이용한 그래픽 사용자 인터페이스 구성이 가능
- Internet scripting
 - 서버 구축 및 클라이언트 스크립트에 유연하게 활용 가능
- Component integration
 - 다른 프로그래밍 언어로 이루어진 소프트웨어 및 상용 소프트웨어와 통합
- Database programming
 - Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite 등 인터페이스 제공
- Rapid prototyping
 - 우선 Python 을 이용하여 검증한 후 다른 프로그래밍 언어로 (일부를) 재작성
- Numeric and scientific programming
 - Numpy, scipy 등을 이용하여 복잡한 계산을 손쉽게 프로그래밍
- Gaming, images, serial ports, XML, robots, and more

Python 의 기술적 장점

- Object-oriented
 - Polymorphism, operator overloading, multiple inheritance, ...
- Free
 - 무료로 이용할 수 있다, 하지만 막강한 기술 지원이 이루어지고 있다.
- Portable
 - Python 이 실행될 수 없는 플랫폼을 발견한 사람?
- Powerful
 - 기계에게 친한 프로그래밍 언어 vs 사람에게 친한 프로그래밍 언어
- Mixable
 - 다른 프로그래밍 언어로 된 프로그램과 통합하기가 용이
- Easy to use
 - 바로 시작할 수 있음!
- Easy to learn
 - 이미 프로그래밍에 익숙한 사람
 - 처음 프로그래밍을 시작하려는 사람

주의사항!

- 프로그래밍은 눈과 귀로 배우는 것이 아님
 - 악기 연주나 스포츠를 책과 동영상으로 배울 수 있는가?
- 베껴 쓰더라도 직접 입력해 보고 실행해 보는 것과 그렇지 않은 것은 크게 다름
- 직접 프로그램을 향상시켜보고, 이리저리 코드를 변경하면서 결과를 확인하는 것은 절대 필요

하루 네 시간 앉아 있는 것도 지겨운데
그 이외 시간에 프로그램을 짜본다고?  말도 안됨

가능한 한 수업 시간 이내에 실습을 해본다!

Python 인터프리터

당신이 작성한 Python 프로그램을 읽어들이어 실행해 주는 프로그램

비교:
컴파일러
링커

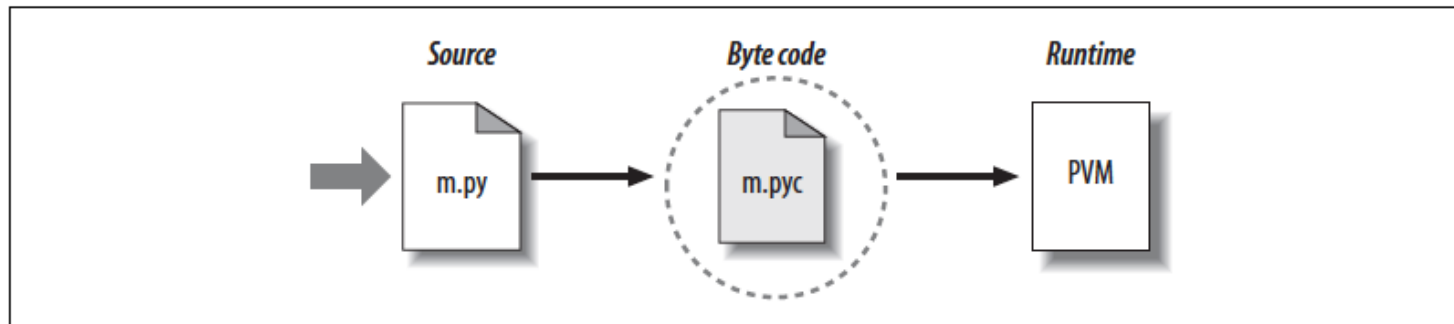
- Windows
 - 설치 파일을 얻어다가 설치 (대부분 “Yes” 또는 “Next” 클릭)
- Linux / Mac OS X
 - 이미 설치되어 있을 가능성이 매우 높음
 - 그렇지 않은 경우 - RPM 또는 macport 등의 방법?
- 이것도 저것도 아닌 경우
 - 소스를 얻어다가 컴파일하고 설치?

```
[sheayun@Sheayuns-MacBook-Air ~]$ python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Python 프로그램이 실행될 때

- Byte code compilation
 - Python 이 소스 코드를 바이트 코드로 번역 (*.pyc)
 - 같은 프로그램을 (변경 없이) 또다시 실행할 때에는 이 과정은 생략됨 (성능 향상을 위하여)
- PVM (Python Virtual Machine) 이 코드를 실행
 - 바이트 코드를 읽어들이며, 의도된 기능을 실행
 - PVM 은 Python 의 런타임 엔진
- 프로그램의 실행 성능
 - C/C++ 등의 컴파일 및 빌드가 필요한 프로그래밍 언어와 비교
- 소프트웨어 개발 유연성
 - 개발 환경과 실행 환경이 동일



Interactive Python Prompt

```
[sheayun@bravo ~]$ python
Python 2.7 (r27:82500, Feb 26 2013, 10:58:16)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

```
% python
```

```
>>> print('Hello world!')
```

```
Hello world!
```

```
>>> print(2 ** 8)
```

```
256
```

```
>>> lumberjack = 'okay'
```

```
>>> lumberjack
```

```
'okay'
```

```
>>> 2 ** 8
```

```
256
```

962 자리의 계산 결과

```
>>> 3 ** 2016
752401261168257532212338322982623966392653752881885657086518737276593165208172505776691057435398355424417484057197214967
846728298473394114826403316359624716040468166831792025372036584703399181245100692449699128027511480154252332057467657551
092012096510343509470424812536008807721829628756972379102766371760156403041764394684699944030238138090035041832236412688
350514951695446482758356693560837830476406273763706608052458450549266307606256837091889322882430394266759809550318192384
195628388185890329889438005735505863867357672684768110409873754519742969726362649105439678363030113282588240900749814120
160362863413923484856171375926415836633406087134137713786658342744395194231324994795746012993302971461353413761628167058
091775454971268871659859720198806625443580008485778392820710079143930884192322815926103333907824776879867720971743083575
840364151483414698038397696019015500041156826462929809641888942477175130159471390506095902455254180352118267760139102867
21L
>>> █
```

Interactive Prompt 를 이용하는 잇점

- 프로그램은 당장 입력하여 실행할 수 있지만, 아무 것도 저장되지 않음
 - 프로그램을 작성했다고 할 수 있을까?
 - 그런데, 왜 interactive prompt 를 이용할까?
- Experimenting
 - >>> 'Spam!' * 8
- Testing
 - >>> import os
 - >>> os.getcwd()
- 프로그램 테스트의 중요성
 - Python 프로그램은 실행되기 이전에는 에러가 발견되지 않음
 - 테스트되지 않은 부분에 에러 (또는 버그) 가 존재할 가능성이 늘 있음

비교:

```
print 'Spam!' * 8
```

```
if x > 8:  
    y = 12  
else:
```

```
    alkdjflqner
```

Python 프로그램 스크립트

일반적으로 .py 의 확장자를 부여한 텍스트 파일

script1.py

```
# -*- coding: utf-8 -*-  
# A first Python script  
import sys  
print(sys.platform)  
print(2 ** 100)  
x = 'Spam!'  
print(x * 8)
```

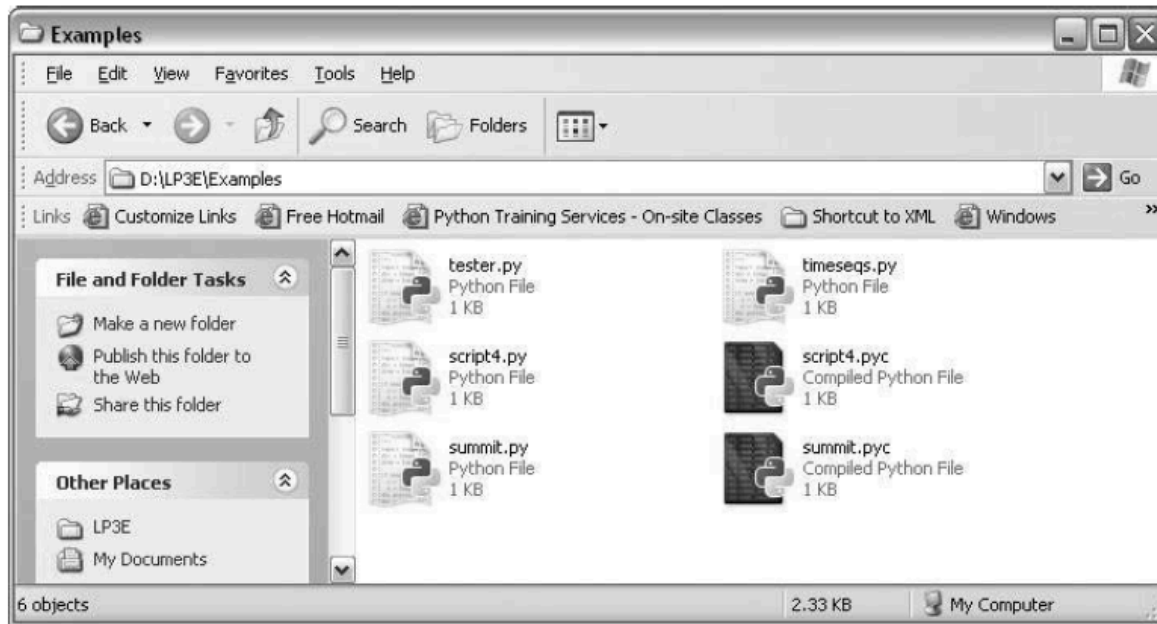
```
% python script1.py  
darwin  
1267650600228229401496703205376L  
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!
```

script2.py

```
#!/usr/local/bin/python  
print('The Bright Side ' + 'of Life...')
```

```
% script2.py  
The Bright Side of Life...
```

파일 아이콘을 이용한 실행



```
# -*- coding: utf-8 -*-  
# A first Python script  
import sys  
print(sys.platform)  
print(2 ** 100)  
input()
```



모듈과 import

.py 로 끝나는 Python 소스 코드 파일은 모두 모듈로 간주할 수 있음

- import 문장은 모듈을 로드하고, 이 모듈의 내용을 접근할 수 있게 함
 - 모듈의 내용? - attributes
 - import 의 마지막 단계로서, 해당 모듈의 스크립트를 실행

script1.py

```
# -*- coding: utf-8 -*-  
# A first Python script  
import sys  
print(sys.platform)  
print(2 ** 100)  
x = 'Spam!'  
print(x * 8)
```

```
% python  
>>> import script1  
darwin  
1267650600228229401496703205376L  
Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!  
>>> import script1  
>>> import script1  
>>> from imp import reload  
>>> reload(script1)  
???
```

IDLE

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.2
>>> 2 ** 100
1267650600228229401496703205376L
>>> 'spam!' * 10
'spam! spam! spam! spam! spam! spam! spam! spam! spam! '
>>> x = 'Spam'
>>> x + 'NI'
'SpamNI'
>>>
>>> import os
>>> os.getcwd()
'C:\Python25'
>>> ===== RESTART =====
>>>
I am: __main__
1
bb
>>> |
```

```
Python Shell
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr 9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> 2 ** 100
1267650600228229401496703205376L
>>> 'spam' * 10
'spamspamspamspamspamspamspamspamspam'
>>> r = 'Spam'
>>> r + 'NI'
'SpamNI'
>>> import os
>>> import sys
>>> sys.platform
'darwin'
>>> |
```

Ln: 15 Col: 4

Python 프로그램의 디버깅

- 에러 메시지를 잘 읽고 친숙해지도록 한다.
 - 프로그래밍을 계속 연습할수록 비슷한 종류의 에러 메시지를 반복해서 만나게 됨
- print 문장을 삽입하고 실행해 본다.
 - Print 'HERE' - 왜 이런 코드를?
- IDE GUI 디버거를 이용한다.
 - IDLE 에서도 제공하지만, 다른 툴들을 많이 이용 (예: Eclipse)
- pdb 커맨드라인 디버거를 이용한다.
 - 다른 프로그래밍 언어의 디버깅 환경에 익숙한 중/고급 프로그래머에게 적합
 - 익숙해지고 나면 매우 효과적/효율적으로 디버깅 가능

Easter Egg

```
% python  
>>> import this  
The Zen of Python, by Tim Peters
```

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Quiz

- 왜 많은 프로그래밍 언어들 중 Python 을 선택?
 - 그렇다면, Python 을 이용하지 않을 이유도 있는가?
- Python interpreter 란 무엇인가?
 - 소스 코드는 무엇이고, 바이트 코드는 무엇인가?
- Interactive interpreter session 과 스크립트 실행의 비교
 - 각각을 이용하는 경우는 어떤 것인가?
- Python 모듈이란 무엇인가?
 - import 문장이 실행되면 어떤 일이 벌어지는가?
- Python 프로그램을 디버깅하는 데에는 어떤 방법이 있는가?

Exercise

(1) "Hello, world!" 를 출력해 본다.

- A. Interactive session ("`>>>`" prompt) 에서
- B. Script 로 저장하고, `python scriptfilename.py` 를 입력해서
- C. `import` 를 이용해서

(2) 에러 메시지를 경험해 본다.

- A. `1 / 0`
- B. `print X`

Q & A