0
–
설정

이 장에는 코드가 없어요. 그저 여러분의 컴퓨터에서 파이썬을 실행할 준비를 마치면 됩니다. 지금부터 하는 설명은 가능한 한 정확하게 따르세요.

윈도우 파워셸이나 맥OS의 터미널, 리눅스의 '배시'를 쓸 줄 모른다면, 먼저 사용 방법을 배워야 합니다. 제 책 『명령줄(Command line) 완전 정복』 요약판을 부록에 넣어 뒀습니다(285쪽). 이 장을 시작하기 전에 이 부록을 꼭 먼저 보고 돌아오세요.

0.1 맥OS

다음 작업을 모두 마치세요.

1. *https://www.python.org/downloads/release/python-360/*으로 가서 macOS 64-bit/32-bit installer 버전을 받아 다른 프로그램을 설치하는 것처럼 설치하세요.
2. *https://atom.io*로 가서 Atom(텍스트 편집기)을 받아 설치하세요. Atom이 손에 맞지 않는다면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.
3. Atom을 독(dock)에 두세요. 이제 쉽게 실행할 수 있습니다.
4. 터미널 프로그램을 찾으세요. 검색해 보면 찾을 수 있습니다.
5. 터미널도 독에 두세요.
6. 터미널을 실행하세요.
7. 터미널에서 `python3.6`을 실행하세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 리턴 키를 누르면 됩니다.
8. `quit()`을 입력하고, 리턴 키를 눌러 파이썬에서 빠져 나오세요.

0
–
설정

이 장에는 코드가 없어요. 그저 여러분의 컴퓨터에서 파이썬을 실행할 준비를 마치면 됩니다. 지금부터 하는 설명은 가능한 한 정확하게 따르세요.

윈도우 파워셸이나 맥OS의 터미널, 리눅스의 '배시'를 쓸 줄 모른다면, 먼저 사용 방법을 배워야 합니다. 제 책 『명령줄(Command line) 완전 정복』 요약판을 부록에 넣어 뒀습니다(285쪽). 이 장을 시작하기 전에 이 부록을 꼭 먼저 보고 돌아오세요.

0.1 맥OS

다음 작업을 모두 마치세요.

1. *https://www.python.org/downloads/release/python-360/*으로 가서 macOS 64-bit/32-bit installer 버전을 받아 다른 프로그램을 설치하는 것처럼 설치하세요.
2. *https://atom.io*로 가서 Atom(텍스트 편집기)을 받아 설치하세요. Atom이 손에 맞지 않는다면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.
3. Atom을 독(dock)에 두세요. 이제 쉽게 실행할 수 있습니다.
4. 터미널 프로그램을 찾으세요. 검색해 보면 찾을 수 있습니다.
5. 터미널도 독에 두세요.
6. 터미널을 실행하세요.
7. 터미널에서 `python3.6`을 실행하세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 리턴 키를 누르면 됩니다.
8. `quit()`을 입력하고, 리턴 키를 눌러 파이썬에서 빠져 나오세요.

9. python3.6이라고 입력하기 전과 비슷한 상태로 돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 터미널에서 디렉터리 만드는 법을 배우세요.
11. 터미널에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 터미널로 돌아오세요.
14. 터미널로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

0.1.1 맥OS: 실행 결과

제 컴퓨터로 터미널에서 실행한 결과를 보여드리겠습니다. 결과는 다를 수 있으니, 책에 실린 결과와 직접 실행한 결과 사이에 무엇이 다른지 모두 찾아내보세요.

```
$ python3.6
Python3.6.0 (default, Feb 2 2017, 12:48:29)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin Type "help",
"copyright", "credits" or "license" for more information. >>>
~ $ mkdir lpthw
~ $ cd lpthw
lpthw $ ls
# ... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ....
lpthw $ ls
test.txt
lpthw $
```

0.2 윈도우

1. *https://atom.io*로 가서 Atom을 받아 설치하세요. 관리자 권한이 없어도 됩니다.
2. Atom을 바탕화면이나 빠른 실행에 두고 쉽게 실행할 수 있도록 하세요. 두 가지 모두 설치할 때 고를 수 있는 선택사항입니다. 컴퓨터가 느려서 Atom을 쓸 수 없으면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.

3. 시작 메뉴에서 파워셸(PowerShell)을 실행하세요. 검색하고 엔터 키를 누르면 실행할 수 있습니다.
4. 편리하게 쓰려면 꼭 바탕화면이나 빠른 실행에 바로가기를 만드세요.
5. 파워셸을 실행하세요(나중에는 터미널이라고 부를 거에요).
6. *https://www.python.org/downloads/release/python-360/*으로 가서 파이썬을 설치하세요. "Add Python 3.6 to PATH" 박스에 꼭 체크하세요.
7. 파워셸(터미널) 프로그램에서 파이썬을 실행해보세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 엔터 키를 누르면 됩니다. `python`이라고 입력했는데도 실행이 안 되면 파이썬을 다시 설치해야 합니다. 꼭 "Add Python 3.6 to PATH." 박스에 체크하세요. 작으니까 꼼꼼히 살펴야 합니다.
8. `quit()`을 입력하고 엔터 키를 눌러 파이썬을 종료합니다.
9. `python`을 입력하기 전에 보던 것과 비슷한 프롬프트로 되돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 파워셸(터미널)에서 디렉터리를 만드는 법을 배우세요.
11. 파워셸(터미널)에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 파워셸(터미널)로 돌아오세요. 할 줄 모르면 찾아보세요.
14. 파워셸(터미널)로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

이후에 "터미널"이나 "셸"이라고 하면 파워셸이라는 뜻이고 여러분은 파워셸을 쓰면 됩니다. 윈도우에서 작업한다면 이 책에서 `python3.6`을 입력해 실행할 때 여러분은 `python`만 입력해야 합니다.

0.2.1 윈도우: 실행 결과

```
> python
>>> quit()
> mkdir lpthw
> cd lpthw
... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ....
>
> dir

 Volume in drive C is
 Volume Serial Number is 085C-7E02

Directory of C:\Documents and Settings\you\lpthw

04.05.2010  23:32 <DIR>        .
04.05.2010
04.05.2010  23:32 <DIR>        ..

            23:32              6 test.txt

            1 File(s)              6 bytes

            2 Dir(s) 14 804 623 360 bytes free

>
```

이 화면과 다르게 보여도 잘 되고 있는 것이지만, 비슷하기는 해야 합니다.

0.3 리눅스

리눅스는 다양한 배포판이 있어서 소프트웨어 설치법도 다양합니다. 여러분이 리눅스를 쓰고 있다면 소프트웨어 패키지를 설치할 줄은 알고 있다고 생각할게요.

1. 패키지 관리자로 python3.6을 설치하세요. 불가능한 경우 *https://www.python.org/downloads/release/python-360/*에서 소스코드를 받아 빌드하세요.
2. 패키지 관리자로 Atom 텍스트 편집기를 설치하세요. Atom이 손에 맞지 않는다면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.

3. Atom 편집기를 쉽게 쓸 수 있도록 창 관리자의 메뉴에 두세요.
4. 터미널 프로그램을 찾으세요. GNOME 터미널, Konsole, xterm 같은 이름 가운데 하나일 수도 있습니다.
5. 독(dock)에 터미널을 두세요.
6. 터미널을 실행하세요.
7. 터미널에서 `python3.6`을 실행하세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 엔터 키를 누르면 됩니다. `python3.6`을 실행할 수 없으면 그냥 `python`을 실행해보세요.
8. `quit()`을 입력하고 엔터 키를 눌러 파이썬을 종료합니다.
9. `python`이라고 입력하기 전과 비슷한 상태로 돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 터미널에서 디렉터리를 만드는 법을 배우세요.
11. 터미널에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 터미널로 돌아오세요.
14. 터미널로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

0.3.1 리눅스: 실행 결과

```
$ python
>>> quit()
$ mkdir lpthw
$ cd lpthw
# ... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ...
$ ls
test.txt
$
```

이 화면과 다르게 보여도 잘 되고 있는 것이지만, 비슷하기는 해야 합니다.

0.4 인터넷에서 찾아보기

이 책에서 배우는 주요한 내용 가운데 하나는 프로그래밍에 관한 내용을 인터넷에서 찾아보고 공부하는 방법입니다. '이 내용을 인터넷에서 찾아보세요.'라는 부분이 보이면 검색 엔진으로 답을 찾아봐야 하는 거에요. 답을 그냥 알려주는 대신 인터넷에서 찾아보게 하는 이유는 여러분이 이 책을 마쳤을 즈음엔 더 이상 책이 필요하지 않은, 자립해서 공부할 수 있는 사람이 되기를 바라기 때문입니다. 인터넷에서 답을 찾아낼 수 있게 되면 제 도움이 필요하지 않은 수준에 한 발짝 더 다가가게 되고, 그게 바로 제 목표랍니다.

구글과 같은 검색 엔진에 힘입어 여러분은 제가 요구하는 어떤 것도 쉽게 찾을 수 있습니다. 책에 '인터넷에서 파이썬(python) 리스트(list) 함수(function)에 대해 찾아보세요.'라고 적혀 있으면 다음과 같이 하세요.

1 *http://google.com*에 접속한다.
2. '파이썬 리스트 함수'나 'python list functions'라고 검색한다.
3. 검색 결과에서 가장 좋은 답을 찾아본다.

0.5 초보자 주의사항

이번 장이 끝났습니다. 컴퓨터에 얼마나 익숙하냐에 따라 어려웠을지도 몰라요. 어려웠다면 시간을 내서 읽고 알아보고 따라 해보세요. 이런 기본적인 일을 해낼 수 없다면 프로그래밍을 아주 잘 하기는 어려울 수도 있어요.

책에서 어떤 장까지만 보라거나, 어디는 건너뛰라고 얘기하는 사람이 있으면 듣지 말아야 합니다. 지식을 감추려 들거나, 더 나쁜 경우 여러분이 스스로의 노력으로 배우는 대신 그들의 지식에 의존하도록 만드는 사람들의 이야기를 들으면, 여러분의 능력은 그 정도 수준으로 한정되고 맙니다. 스스로 공부하는 법을 배울 수 있도록 그런 데 귀 기울이지 말고 모두 해보세요.

맥이나 리눅스를 쓰라는 사람도 있을지도 몰라요. 글꼴이나 타이포그래피를 좋아한다면 맥을, 통제권과 풍성한 턱수염을 좋아한다면 리눅스를 권하겠죠. 다시 한번 말하지만, 어떤 컴퓨터든 당장 갖고 있고 사용할 수 있는 것을

쓰세요. 필요한 것은 편집기와 터미널과 파이썬뿐입니다.

마지막으로 지금까지 한 준비는 앞으로 책을 보는 동안 다음 네 가지를 확실히 할 수 있도록 만들기 위해서입니다.

1. Atom으로 예제 코드를 입력할 수 있다.
2. 직접 쓴 예제를 실행할 수 있다.
3. 코드가 망가졌을 때는 수정할 수 있다.
4. 반복한다.

다른 모든 것은 헷갈리기만 할 뿐이니 신경 쓰지 말고, 계획대로 따라오세요.

0.6 다른 텍스트 편집기

프로그래머에게 텍스트 편집기란 대단히 중요하지만, 초보자인 여러분에게는 단순한 프로그래머용 편집기만 있으면 됩니다. 이런 편집기는 소설이나 책을 쓸 때와 달리 컴퓨터 코드에 필요한 특별한 기능을 갖고 있어요. 책에서는 무료이고 거의 어디서든 돌아가는 Atom을 권하고 있습니다. 하지만 여러분이 가진 컴퓨터에서 Atom이 잘 동작하지 않는다면, 다음의 다른 편집기도 써보세요.

편집기	지원 환경	다운로드
Visual Studio Code	윈도우, 맥, 리눅스	*https://code.visualstudio.com*
Notepad++	윈도우	*https://notepad-plus-plus.org*
gEdit	리눅스, 맥, 윈도우	*https://github.com/GNOME/gedit*
Textmate	맥	*https://github.com/textmate/textmate*
SciTE	윈도우, 리눅스	*http://www.scintilla.org/SciTE.html*
jEdit	리눅스, 맥, 윈도우	*http://www.jedit.org*

편집기의 순서는 잘 동작할 것 같은 순서입니다. 어떤 프로젝트는 버려지거나 죽거나 여러분 컴퓨터에서 더 이상 돌아가지 않을지도 모른다는 점을 명심하세요. 하나를 해 보고 잘 안 되면 다른 걸 해보세요. '지원 환경'도 마찬가

지로 잘 동작하는 순서대로입니다. 윈도우에서 사용할 편집기를 찾고 있다면 '지원 환경' 열에서 윈도우가 처음으로 나오는 편집기를 찾아보세요.

Vim이나 Emacs를 이미 쓸 줄 안다면 마음껏 쓰세요. 하지만 아직 한번도 써 본 적이 없다면 쓰지 마세요. Vim이나 Emacs를 써야 한다고 설득하려 드는 프로그래머들이 있을지도 모르는데, 그저 갈 길에서 멀어질 뿐입니다. 여러분의 목표는 파이썬을 배우는 거에요. Vim이나 Emacs를 배우는 게 아니고요. Vim을 써보려다가 끄지도 못하게 되었으면 키보드로 :q! 나 ZZ를 입력하세요. 여러분에게 Vim을 써보라던 사람이 이것조차도 알려주지 않았다면, 왜 그 사람들 말을 들으면 안 되는지 알겠죠?

이 책으로 공부하는 동안은 통합 개발 환경(IDE, Integrated Development Environment)은 쓰지 마세요. 통합 개발 환경에 의존한다는 얘기는, 앞으로 새 언어를 배우려면 그 언어에 맞는 통합 개발 환경을 만들어주는 회사가 나타날 때까지는 그 언어로는 프로그래밍할 수 없다는 말입니다. 새 언어가 나오면 탄탄한 수익을 낼 수 있는 고객층이 생길 만큼 커지기 전까지는 써보지도 못한다는 얘기지요. 프로그래머용 텍스트 편집기(Vim, Emacs, Atom 등)만 갖고도 프로그래밍할 수 있다는 자신이 있으면 다른 개발도구를 기다릴 필요가 없습니다. (거대한 코드 기반 위에서 작업하는 경우처럼) 통합 개발 환경이 유리한 상황도 있지만, 거기에만 얽매이면 여러분의 가능성도 거기에 얽매입니다.

IDLE도 역시 쓰지 말아야 합니다. 심각한 제약 아래에서 동작하고, 그다지 잘 만들어진 소프트웨어도 아닙니다. 여러분에게 필요한 건 간단한 텍스트 편집기와 셸과 파이썬뿐입니다.

연습 1

첫 번째 프로그램

잊지 마세요. 여러분은 앞에서 텍스트 편집기를 설치해서 실행해 보고, 터미널을 실행하고, 두 가지를 함께 쓰는 법을 배우느라 상당한 시간을 써야 했답니다. 아직 그 과정을 마치지 못했다면 여기서 멈추세요. 앞 부분을 마쳐야지만 다음을 즐겁게 해 나갈 수 있습니다. 이번 장에서만 건너뛰거나 앞서 나가지 말라는 경고를 하고 이제 시작하겠습니다.

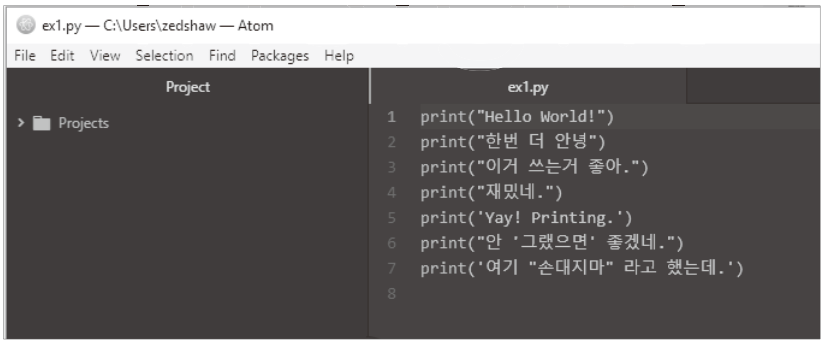
파일을 하나 만들어 이름을 ex1.py로 짓고 다음 내용을 써넣으세요. 중요한 내용입니다. 파이썬은 파일 이름이 .py로 끝나야 잘 동작하거든요.

0장을 건너뛰면 이 책을 제대로 공부하는 게 아닙니다. IDLE이나 IDE로 하려구요? 0장에서 쓰지 말라고 얘기했으니 쓰지 말아야 합니다. 0장을 건너뛰었으면 꼭 되돌아가서 읽어보세요.

ex1.py

```
1   print("Hello World!")
2   print("한번 더 안녕")
3   print("이거 쓰는거 좋아.")
4   print("재밌네.")
5   print('Yay! Printing.')
6   print("안 '그랬으면' 좋겠네.")
7   print('여기 "손대지마" 라고 했는데.')
```

어떤 환경에서든 Atom 편집기는 다음과 같이 보여야 합니다.



편집기가 정확히 똑같이 보이지 않는다고 걱정하지는 마세요. 하지만 비슷하기는 해야 합니다. 창머리나 색깔이 조금 다를 수 있습니다. 창 왼쪽에도 'zedshaw'라고 보이는 대신 여러분이 파일을 저장한 디렉터리가 보이겠지요. 이런 차이점은 모두 괜찮습니다.

파일을 만들 때 중요한 점은 다음 내용입니다.

1 왼쪽에 있는 줄 번호는 입력한 게 아닙니다. '5행을 보세요'처럼, 줄을 정확하게 가리키기 위해 책에만 출력한 것입니다. 파이썬 스크립트에는 쓰지 말아야 합니다.
2 모든 행은 `print`로 시작하며, 위에서 보여드린 예제와 정확히 똑같습니다. '정확히'라는 말은 글자 하나도 틀리지 않고 모두 똑같다는 뜻이에요. 완전히 똑같아야만 동작합니다. 하지만 색은 모두 달라도 됩니다. 색은 상관없어요. 입력한 글자만 중요합니다.

맥OS 터미널이나 (아마도) 리눅스에서는 파일을 이렇게 실행합니다.

```
python3.6 ex1.py
```

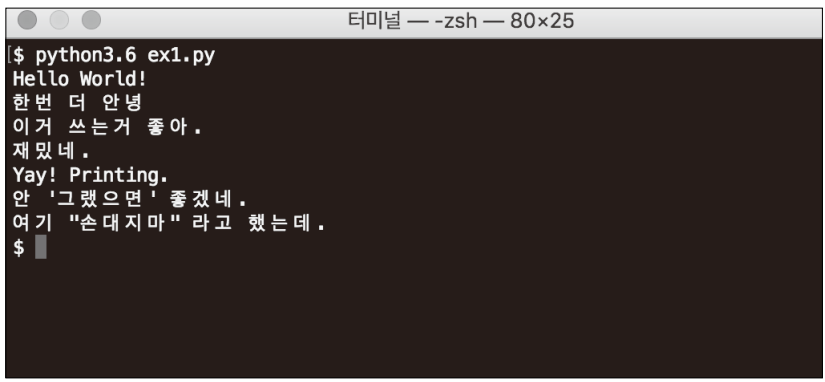
윈도우에서는 `python3.6` 대신 `python`이라고 입력한다는 걸 꼭 기억해 둡시다.

```
python ex1.py
```

올바르게 따라 했다면 다음의 실행 결과와 똑같은 결과가 나옵니다. 아니라면 무엇인가 잘못한 것입니다. 아니요. 컴퓨터가 잘못된 것은 아니에요.

1.1 실행 결과

맥OS 터미널에서는 이렇게 보입니다.



윈도우 파워셸에서는 이렇게 보입니다.

`python ex1.py` 이전 부분의 컴퓨터 이름이나 나머지 내용은 다를 수도 있습니다. 하지만 중요한 부분은 여러분이 명령어를 입력하자 앞의 화면과 똑같은 결과가 나왔다는 것입니다.

오류(error)가 있다면 이런 식일 것입니다.

```
$ python ex/ex1.py
    File "ex/ex1.py", line 3
        print "I like typing this.
                                  ^
   SyntaxError: EOL while scanning string literal
```

이 오류는 읽을 수 있어야 합니다. 이런 실수는 자주 할 테니까요. 저조차도 이런 실수를 자주 합니다. 한 줄씩 읽어봅시다.

1. 터미널에서 ex1.py 스크립트를 실행하도록 명령했습니다.
2. 파이썬이 ex1.py 파일의 셋째 줄에 오류가 있다고 알려줍니다.
3. 그 줄을 볼 수 있게 출력해 줍니다.
4. ^(캐럿, caret) 문자로 문제가 생긴 지점을 표시해줍니다. "(큰따옴표, double quote) 문자가 빠져있네요. 확인해보세요.
5. 마지막으로 'SyntaxError'를 출력하고 무엇 때문에 오류가 생겼을 수 있는지 알려줍니다. 몹시 아리송할 때가 많겠지만 검색 엔진에 복사해 넣으면 같은 오류를 겪은 다른 사람을 찾을 수 있습니다. 대부분은 고치는 방법까지 알아낼 수 있습니다.

1.2 더 해보기

각 장에는 '더 해보기' 절이 있습니다. '더 해보기'에는 여러분이 도전해 볼 만한 과제들이 있습니다. 모르겠으면 건너뛰었다 나중에 다시 보세요.

이번 장에서는 이런 것을 더 해보세요.

1. 한 줄 더 출력하도록 스크립트를 고쳐보세요.
2. 여러 줄 가운데 한 줄만 출력하도록 스크립트를 고쳐보세요.
3. #(해시) 문자를 줄 앞에 넣어보세요. 어떤 일이 일어났나요? 이 문자가 무슨 일을 하는지 알아내보세요.

지금부터는 특별히 다른 구성이 없는 한 각 장이 어떤 식으로 구성되어 있는지 설명하지 않겠습니다.

1.3 자주 묻는 질문

이 책의 온라인 페이지에서 댓글란에 학생들이 실제로 남긴 질문입니다. 여러분도 같은 문제를 겪을 수 있으니 자주 묻는 질문을 골라서 답해뒀습니다.

Q. IDLE을 써도 되나요?

A. 아니요. 책에 나온 대로 맥OS에서는 터미널을 쓰고 윈도우에서는 파워셸을 쓰세요. 어떻게 쓰는지 모른다면 부록 '명령줄 완전 정복'을 보세요.

Q. 편집기에 색은 어떻게 입히나요?

A. 먼저 ex1.py 같이 파일을 .py 파일로 저장하세요. 다음부터는 입력할 때마다 색이 입혀집니다.

Q. ex1.py를 실행하면 `SyntaxError: invalid syntax`가 나와요.

A. 파이썬을 한 번 실행하고 나서 그 안에서 다시 파이썬을 실행하려고 한 것입니다. 터미널을 닫고 다시 실행하자마자 python ex1.py라고만 입력하세요.

Q. `can't open file 'ex1.py': [Errno 2] No such file or directory.`가 나와요.

A. 파일을 만든 디렉터리에 들어가 있어야 합니다. cd 명령을 써서 해당 디렉터리로 가세요. 예를 들어 파일을 lpthw/ex1.py에 저장했다면 `python3.6 ex1.py`를 실행하기 전에 `cd lpthw/`부터 하세요. 무슨 뜻인지 모르겠으면 부록 '명령줄 완전 정복'을 보세요.

연습 2

주석과 해시 문자

프로그래밍에서 주석(comment)은 아주 중요합니다. 사람이 읽어야 할 내용을 사람이 쓰는 언어로 쓰는 것이 주석입니다. 프로그램 일부를 임시로 지워야 할 때 코드를 비활성화(disable)하기 위해서도 쓰입니다. 파이썬에는 주석을 다음과 같이 씁니다.

ex2.py

```
1   # 주석, 나중에 프로그램을 읽을 수 있게 도와줍니다.
2   # 파이썬에서 # 뒤에 쓰인 것은 무엇이든 무시됩니다.
3
4   print("이렇게 코드를 쓸 수 있습니다.") # 그리고 주석 뒤는 무시됩니다.
5
6   # 주석은 코드 일부를 주석 처리해 '비활성화'할 때도 사용할 수 있습니다.
7   # print("실행되지 않습니다.")
8
9   print("실행됩니다.")
```

지금부터는 이런 식으로 코드를 쓰겠습니다. 모든 것이 보이는 그대로일 필요는 없다는 점을 이해해야 해요. 화면과 프로그램이 눈에는 다르게 보일 수도 있지만, 중요한 내용은 여러분이 텍스트 편집기에서 파일에 입력하는 텍스트입니다. 다른 텍스트 편집기를 쓰더라도 결과는 똑같이 나와요.

2.1 실행 결과

Exercise 2 Session

```
$ python3.6 ex2.py
이렇게 코드를 쓸 수 있습니다.
실행됩니다.
```

다시 한번 말하지만, 책에서는 환경에 따라 다른 모든 터미널의 스크린샷을

보여주지 않겠습니다. 이 내용은 실행 결과를 눈에 보이는 그대로 보여주는 것이 아니라는 점을 명심하세요. 대신 첫 줄의 `$ python3.6`과 마지막 줄 $ 사이의 내용에 주목하세요.

책과 달리 내 터미널에서는 각 줄이 $로 시작하지 않나요? $ 앞부분은 복잡하고 컴퓨터마다 다르기 때문에 간단하게 표시하고 있습니다. 부록 '명령줄 완전 정복'을 먼저 보고 돌아오세요.

2.2 더 해보기

1. # 기호의 기능에 대해 추측했던 내용이 맞았나 확인하고 어떻게 부르는지 알아두세요.
2. ex2.py 파일을 거꾸로 한 줄씩 다시 보세요. 마지막 줄에서 시작해 한 단어씩 입력했던 순서와 반대로 확인하세요.
3. 실수한 게 더 있나요? 마저 고치세요.
4. 입력한 내용을 크게 읽으세요. 기호는 각각 그 이름으로 읽으세요. 실수한 게 더 있나요? 마저 고치세요.

2.3 자주 묻는 질문

Q. 왜 `print("안녕 # 여러분.")`의 #는 무시되지 않나요?

A. 그 #는 문자열 안에 있습니다. 따라서 문장 끝의 " 문자가 나타나기 전까지는 문자열의 일부입니다. 이 # 문자는 그냥 문자로 취급하고, 주석으로 처리하지 않습니다.

Q. 여러 줄은 어떻게 주석 처리하나요?

A. 매 줄마다 앞에 #를 붙이세요.

Q. 왜 코드를 거꾸로 읽어야 하죠?

A. 코드의 의미에 매달리지 않고 정말 코드를 한 부분씩 처리할 수 있도록 생각하게 도와주는 기법입니다. 이 방법으로 코드를 정확히 한 부분씩 비교해 오류를 찾을 수 있습니다. 간편한 오류 점검 기법이기도 합니다.

연습 3

수와 계산

모든 프로그래밍 언어에는 수(number)와 계산(math)을 다루는 방법이 있습니다. 걱정하지 마세요. 프로그래머들이란 사실은 그렇지도 않으면서 수학 천재인 척 굴며 거짓말하곤 합니다. 진짜 수학 천재였다면 버그 덩어리 웹 프레임워크를 만드는 대신 진짜 수학을 하고 있겠지요.

이번 장에서는 많은 수학 기호(symbol)를 다룹니다. 기호에 붙은 올바른 이름을 알아봅시다. 하나씩 입력하며 이름을 말해보세요. 지겹게 느껴지면 소리내기는 그만해도 됩니다. 기호 이름을 살펴봅시다.

- + 덧셈(plus)
- - 뺄셈(minus)
- / 나눗셈(slash)
- * 곱셈(asterisk)
- % 나머지(percent)
- 〈 작다 (less-than)
- 〉 크다 (greater-than)
- 〈= 작거나 같다(less-than-equal)
- 〉= 크거나 같다(greater-than-equal)

어떤 연산을 하는지가 빠졌죠? 이번 장에서 코드에 써본 다음, 돌아와서 각각 어떤 역할인지 찾아내 표를 완성하세요. 예를 들어 +는 덧셈 연산자입니다.

ex3.py

```
1   print("닭을 세어봅시다.")
2
3   print("암탉", 25 + 30 / 6)
4   print("수탉", 100 - 25 * 3 % 4)
5
6   print("이제 달걀도 세어봅시다.")
7
8   print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)
9
10  print("3 + 2 < 5 - 7 는 참인가요?")
11
12  print(3 + 2 < 5 - 7)
13
14  print("3 + 2 는 얼마죠?", 3 + 2)
15  print("5 - 7 은 얼마죠?", 5 - 7)
16
17  print("아하 이게 False인 이유네요.")
18
19  print("더 해볼까요.")
20
21  print("더 큰가요?", 5 > -2)
22  print("더 크거나 같나요?", 5 >= -2)
23  print("더 작거나 같나요?", 5 <= -2)
```

3.1 실행 결과

Exercise 3 Session

```
$ python3.6 ex3.py
닭을 세어봅시다.
암탉 30.0
수탉 97
이제 달걀도 세어봅시다.
6.75
3 + 2 < 5 - 7 는 참인가요?
False
3 + 2 는 얼마죠? 5
5 - 7 은 얼마죠? -2
아하 이게 False인 이유네요.
더 해볼까요.
더 큰가요? True
더 크거나 같나요? True
더 작거나 같나요? False
```

3.2 더 해보기

1. 줄마다 그 줄이 뭘 하는지 주석으로 써서 스스로에게 설명해보세요. 주석은 #를 이용해 쓸 수 있습니다.
2. '0. 설정' 장에서 파이썬을 처음 시작했을 때가 기억나세요? 그때처럼 파이썬을 다시 실행해 보고, 위에서 쓴 기호와 아는 내용을 이용해 파이썬을 계산기처럼 써보세요.
3. 계산할 거리를 찾아보고 그 계산을 하는 .py 파일을 만들어보세요.
4. ex3.py를 부동소수점을 이용해 더 정확하게 새로 만들어주세요. 20.0이 부동소수점입니다.

3.3 자주 묻는 질문

Q. 왜 % 문자는 '퍼센트'가 아니라 '나머지(modulus)'인가요?

A. 가장 큰 이유는 파이썬을 설계한 사람들이 그 기호를 그렇게 쓰기로 결정했기 때문이에요. 일반적인 글에서는 %를 '퍼센트'로 읽는 게 옳습니다. 프로그래밍에서 퍼센트는 보통 단순히 / 연산자로 나눠서 계산합니다. %는 퍼센트와는 상관없이 그냥 나머지 연산의 기호를 %로 정한 것일 뿐입니다.

Q. %는 어떻게 동작하나요?

A. 다르게 말하자면, "X를 Y로 나누면 나머지는 J이다."라고 할 수 있습니다. 예를 들어, "100을 16으로 나누면 나머지는 4이다." 같이요. %의 결과는 나머지 부분이라고 부르는 J입니다.

Q. 연산의 우선순위는 어떻게 되죠?

A. 수학에서 쓰는 순서는 괄호, 지수, 곱셈과 나눗셈, 덧셈과 뺄셈입니다. 파이썬도 이 순서를 따릅니다.

연습 4

변수와 이름

이제 여러분은 print로 출력(print)도 할 수 있고 계산도 할 수 있습니다. 다음 단계로 변수(variable)에 대해 배워봅시다. 프로그래밍에서 변수란 어떤 내용을 쓴 코드를 가리키는 이름일 뿐입니다. '제드 쇼'라는 제 이름이 '이 책을 쓴 사람'을 가리키는 이름인 것처럼요. 프로그래머의 기억이란 형편없기 때문에, 사람이 읽기 쉽도록 변수를 이용해 코드에 이름을 붙입니다. 좋은 변수 이름을 붙여가며 소프트웨어를 만들어야 코드를 다시 읽을 때 헤매지 않습니다.

이번 장을 하는 동안 막히는 부분이 있다면, 앞에서 배운, 차이를 찾고 세부에 집중하게 해주는 비법을 기억하세요.

1. 매 줄마다 그 줄이 무엇을 하는지 스스로에게 설명하는 주석을 글로 써보세요.
2. .py 파일을 거꾸로 읽어보세요.
3. .py 파일을 큰 소리로 읽어보세요. 기호까지도요.

ex4.py

```
1   자동차 = 100
2   차_안_공간 = 4.0
3   운전사 = 30
4   승객 = 90
5   운행_안하는_차 = 자동차 - 운전사
6   운행하는_차 = 운전사
7   총_정원 = 운행하는_차 * 차_안_공간
8   차당_평균_승객 = 승객 / 운행하는_차
9
10
11  print("자동차", 자동차, "대가 있습니다.")
12  print("운전자는", 운전사, "명뿐입니다.")
13  print("오늘은 빈 차가", 운행_안하는_차, "대일 것입니다.")
```

```
14  print("오늘은", 총_정원, "명을 태울 수 있습니다.")
15  print("함께 탈 사람은", 승객, "명 있습니다.")
16  print("차마다", 차당_평균_승객, "명 정도씩 타야 합니다.")[1]
```

차_안_공간 사이의 _는 밑줄(underscore) 문자라 불립니다. 어떻게 입력하는지 모르면 찾아보세요. 변수 이름에서 낱말 사이에 가상으로 띄어쓰기를 넣을 때 많이 쓰는 문자입니다.

4.1 실행 결과

Exercise 4 Session

```
$ python3.6 ex4.py
자동차 100 대가 있습니다.
운전자는 30 명 뿐입니다.
오늘은 빈 차가 70 대일 것입니다.
오늘은 120.0 명을 태울 수 있습니다.
함께 탈 사람은 90 명 있습니다.
차마다 3.0 명 정도씩 타야 합니다.
```

4.2 더 해보기

처음에 이 프로그램을 짜다가 실수했더니 파이썬이 이렇게 알려줬습니다.

```
Traceback (most recent call last):
  File "ex4.py", line 8, in <module>
    차당_평균_승객 = 총정원 / 승객
NameError: name '총정원' is not defined
```

이 오류를 직접 설명해보세요. 반드시 행 번호를 쓰며 이유를 설명하세요.

더 해봐야 하는 숙제도 많이 드리겠습니다.

1. `차_안_공간`의 값으로 `4.0`을 썼는데, 꼭 그래야 하나요? 그냥 `4`면 어떻게 되죠?
2. `4.0`이 '부동소수점(floating point)'이라는 것을 기억하세요. 그냥 소수점 달린 숫자인데, 부동소수점이기 때문에 `4` 대신 `4.0`이라고 써야 해요.

1 (옮긴이) 책에서 변수 이름 등은 특별한 관례가 없는 한 한국어로 썼습니다. 영어로 된 예제로 공부하고 싶으면 다음 주소에서 저자의 원래 코드를 참고할 수 있습니다. *https://github.com/zedshaw/learn-python3-thw-code*

3. 모든 변수 대입 위에 주석을 달아보세요.
4. =(등호)의 이름과 용도를 확실히 알아두세요. 다른 값(숫자, 문자열 등)에 이름(**운행하는_차, 승객**)을 붙일 때 씁니다
5. _는 밑줄 문자입니다. 잊지 마세요.
6. 앞에서처럼 터미널에서 python3.6을 실행해 계산기처럼 써보고, 계산에 변수 이름도 써보세요. 널리 쓰이는 변수 이름으로는 `i`, `x`, `j`가 있습니다.

4.3 자주 묻는 질문

Q. =(홑등호)와 ==(겹등호)는 무엇이 다른가요?

A. =(홑등호)는 오른쪽의 값을 왼쪽의 변수에 대입합니다. ==(겹등호)는 양쪽의 값이 같은지 비교합니다. 27장에서 배우겠습니다.

Q. x = 100 대신 x=100으로 쓸 수 있나요?

A. 네. 하지만 나쁜 형태입니다. 쉽게 읽을 수 있도록 연산자 주위는 띄어 써야 합니다.

Q. '파일을 거꾸로 읽어보세요'가 무슨 뜻인가요?

A. 아주 간단합니다. 16행짜리 코드가 있다고 생각해보세요. 16행부터 시작합니다. 그리고 예제 파일의 16행과 비교합니다. 15행에서도, 그 앞에서도 반복하고, 파일 전체를 다 읽을 때까지 계속합니다.

Q. 왜 차_안_공간 값에 4.0을 썼나요?

A. 가장 큰 이유는 여러분이 부동소수점 수가 무엇인지 찾아내고 이 질문을 하도록 하기 위해서였죠. '4.2 더 해보기'를 보세요.

이 책에서는 코드에서 대부분의 변수나 함수(나중에 배웁니다) 이름을 한글로 썼습니다. 영어에 익숙하지 않은 독자들이 영어와 파이썬에 모두 신경 쓰기보다는 익숙한 한국어로 보다 빠르게 파이썬을 배우기를 바라는 마음이기 때문이지요. 여러분이 프로그래밍의 세계에 본격적으로 발을 들이게 된다면, 실제로 프로그래밍은 대부분 영어로 하게 됩니다. 한국에서만 쓰는, 한국어 사용자를 위한 코드도 (아직은) 그렇습니다. 코드를 짜는 데는 사소해 보이지만 중요한 관습이 있습니다. 이를테면 프로그래밍 언어를 막론하고 대부분의 프로그래머는 단수와 복수 표현에 민감합니다. 똑같은 의미의 변수를 쓰더라도 변수 이름이 단수인지 복수인지에 따라 변수가 어떤 자료를 가리키고 있는지 의미가 달라지거든요. 파이썬 프로그래머들은 대소문자의 구분도 중요하게 생각합니다. 대소문자의 구분으로 코드를 더 읽기 쉽도록 하는 기법이 사용되기 때문이지요. 아쉽지만 코드를 한글로 옮기는 과정에서 한국어다움을 지키면서 온전히 옮길 수 있는 방법은 없었습니다. 단수와 복수의 구분은, 꼭 복수형으로 표시해야 하는 경우에 모두 복수 접미사 '들'을 써서 표현했습니다. 한글로 쓴 코드에 굳이 '들'이 들어가 있다면 의도적으로 구분한 부분이니 왜 들어갔는지 생각해보세요. 대소문자를 이용한 구분은 살릴 수 없었습니다. 책을 모두 마친 뒤에 영어로 된 예제 코드를 다시 살펴보고 파이썬 코드의 포맷에 대하여 더 알아보면 더욱 파이썬다운 코드를 짜는 데 도움이 될 것입니다.

9. python3.6이라고 입력하기 전과 비슷한 상태로 돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 터미널에서 디렉터리 만드는 법을 배우세요.
11. 터미널에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 터미널로 돌아오세요.
14. 터미널로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

0.1.1 맥OS: 실행 결과

제 컴퓨터로 터미널에서 실행한 결과를 보여드리겠습니다. 결과는 다를 수 있으니, 책에 실린 결과와 직접 실행한 결과 사이에 무엇이 다른지 모두 찾아내보세요.

```
$ python3.6
Python3.6.0 (default, Feb 2 2017, 12:48:29)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin Type "help",
"copyright", "credits" or "license" for more information. >>>
~ $ mkdir lpthw
~ $ cd lpthw
lpthw $ ls
# ... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ....
lpthw $ ls
test.txt
lpthw $
```

0.2 윈도우

1. *https://atom.io*로 가서 Atom을 받아 설치하세요. 관리자 권한이 없어도 됩니다.
2. Atom을 바탕화면이나 빠른 실행에 두고 쉽게 실행할 수 있도록 하세요. 두 가지 모두 설치할 때 고를 수 있는 선택사항입니다. 컴퓨터가 느려서 Atom을 쓸 수 없으면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.

3. 시작 메뉴에서 파워셸(PowerShell)을 실행하세요. 검색하고 엔터 키를 누르면 실행할 수 있습니다.
4. 편리하게 쓰려면 꼭 바탕화면이나 빠른 실행에 바로가기를 만드세요.
5. 파워셸을 실행하세요(나중에는 터미널이라고 부를 거에요).
6. *https://www.python.org/downloads/release/python-360/*으로 가서 파이썬을 설치하세요. "Add Python 3.6 to PATH" 박스에 꼭 체크하세요.
7. 파워셸(터미널) 프로그램에서 파이썬을 실행해보세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 엔터 키를 누르면 됩니다. `python`이라고 입력했는데도 실행이 안 되면 파이썬을 다시 설치해야 합니다. 꼭 "Add Python 3.6 to PATH." 박스에 체크하세요. 작으니까 꼼꼼히 살펴야 합니다.
8. `quit()`을 입력하고 엔터 키를 눌러 파이썬을 종료합니다.
9. `python`을 입력하기 전에 보던 것과 비슷한 프롬프트로 되돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 파워셸(터미널)에서 디렉터리를 만드는 법을 배우세요.
11. 파워셸(터미널)에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 파워셸(터미널)로 돌아오세요. 할 줄 모르면 찾아보세요.
14. 파워셸(터미널)로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

이후에 "터미널"이나 "셸"이라고 하면 파워셸이라는 뜻이고 여러분은 파워셸을 쓰면 됩니다. 윈도우에서 작업한다면 이 책에서 `python3.6`을 입력해 실행할 때 여러분은 `python`만 입력해야 합니다.

0.2.1 윈도우: 실행 결과

```
> python
>>> quit()
> mkdir lpthw
> cd lpthw
... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ....
>
> dir

 Volume in drive C is
 Volume Serial Number is 085C-7E02

Directory of C:\Documents and Settings\you\lpthw

04.05.2010  23:32 <DIR>        .
04.05.2010
04.05.2010  23:32 <DIR>        ..

            23:32              6 test.txt

            1 File(s)              6 bytes

            2 Dir(s) 14 804 623 360 bytes free

>
```

이 화면과 다르게 보여도 잘 되고 있는 것이지만, 비슷하기는 해야 합니다.

0.3 리눅스

리눅스는 다양한 배포판이 있어서 소프트웨어 설치법도 다양합니다. 여러분이 리눅스를 쓰고 있다면 소프트웨어 패키지를 설치할 줄은 알고 있다고 생각할게요.

1. 패키지 관리자로 python3.6을 설치하세요. 불가능한 경우 *https://www.python.org/downloads/release/python-360/*에서 소스코드를 받아 빌드하세요.
2. 패키지 관리자로 Atom 텍스트 편집기를 설치하세요. Atom이 손에 맞지 않는다면 이 장 마지막 절 '다른 텍스트 편집기'를 참고하세요.

3. Atom 편집기를 쉽게 쓸 수 있도록 창 관리자의 메뉴에 두세요.
4. 터미널 프로그램을 찾으세요. GNOME 터미널, Konsole, xterm 같은 이름 가운데 하나일 수도 있습니다.
5. 독(dock)에 터미널을 두세요.
6. 터미널을 실행하세요.
7. 터미널에서 `python3.6`을 실행하세요. 터미널에서 무언가를 실행하려면 그 이름을 쓰고 엔터 키를 누르면 됩니다. `python3.6`을 실행할 수 없으면 그냥 `python`을 실행해보세요.
8. `quit()`을 입력하고 엔터 키를 눌러 파이썬을 종료합니다.
9. `python`이라고 입력하기 전과 비슷한 상태로 돌아와야 합니다. 아니라면 이유를 찾아보세요.
10. 터미널에서 디렉터리를 만드는 법을 배우세요.
11. 터미널에서 디렉터리를 바꿔 들어가는 법을 배우세요.
12. 편집기를 이용해 그 디렉터리에 파일을 하나 만드세요. 파일을 만들고 메뉴에서 [Save]나 [Save As ....]를 선택한 다음 그 디렉터리를 고르세요.
13. 키보드만 써서 창을 바꿔 터미널로 돌아오세요.
14. 터미널로 돌아와서 새로 만든 파일을 볼 수 있도록 디렉터리 내용을 출력해보세요.

0.3.1 리눅스: 실행 결과

```
$ python
>>> quit()
$ mkdir lpthw
$ cd lpthw
# ... 여기서 텍스트 편집기로 test.txt 파일을 편집합니다 ...
$ ls
test.txt
$
```

이 화면과 다르게 보여도 잘 되고 있는 것이지만, 비슷하기는 해야 합니다.

0.4 인터넷에서 찾아보기

이 책에서 배우는 주요한 내용 가운데 하나는 프로그래밍에 관한 내용을 인터넷에서 찾아보고 공부하는 방법입니다. '이 내용을 인터넷에서 찾아보세요.'라는 부분이 보이면 검색 엔진으로 답을 찾아봐야 하는 거에요. 답을 그냥 알려주는 대신 인터넷에서 찾아보게 하는 이유는 여러분이 이 책을 마쳤을 즈음엔 더 이상 책이 필요하지 않은, 자립해서 공부할 수 있는 사람이 되기를 바라기 때문입니다. 인터넷에서 답을 찾아낼 수 있게 되면 제 도움이 필요하지 않은 수준에 한 발짝 더 다가가게 되고, 그게 바로 제 목표랍니다.

구글과 같은 검색 엔진에 힘입어 여러분은 제가 요구하는 어떤 것도 쉽게 찾을 수 있습니다. 책에 '인터넷에서 파이썬(python) 리스트(list) 함수(function)에 대해 찾아보세요.'라고 적혀 있으면 다음과 같이 하세요.

1 *http://google.com*에 접속한다.
2. '파이썬 리스트 함수'나 'python list functions'라고 검색한다.
3. 검색 결과에서 가장 좋은 답을 찾아본다.

0.5 초보자 주의사항

이번 장이 끝났습니다. 컴퓨터에 얼마나 익숙하냐에 따라 어려웠을지도 몰라요. 어려웠다면 시간을 내서 읽고 알아보고 따라 해보세요. 이런 기본적인 일을 해낼 수 없다면 프로그래밍을 아주 잘 하기는 어려울 수도 있어요.

책에서 어떤 장까지만 보라거나, 어디는 건너뛰라고 얘기하는 사람이 있으면 듣지 말아야 합니다. 지식을 감추려 들거나, 더 나쁜 경우 여러분이 스스로의 노력으로 배우는 대신 그들의 지식에 의존하도록 만드는 사람들의 이야기를 들으면, 여러분의 능력은 그 정도 수준으로 한정되고 맙니다. 스스로 공부하는 법을 배울 수 있도록 그런 데 귀 기울이지 말고 모두 해보세요.

맥이나 리눅스를 쓰라는 사람도 있을지도 몰라요. 글꼴이나 타이포그래피를 좋아한다면 맥을, 통제권과 풍성한 턱수염을 좋아한다면 리눅스를 권하겠죠. 다시 한번 말하지만, 어떤 컴퓨터든 당장 갖고 있고 사용할 수 있는 것을

쓰세요. 필요한 것은 편집기와 터미널과 파이썬뿐입니다.

마지막으로 지금까지 한 준비는 앞으로 책을 보는 동안 다음 네 가지를 확실히 할 수 있도록 만들기 위해서입니다.

1. Atom으로 예제 코드를 입력할 수 있다.
2. 직접 쓴 예제를 실행할 수 있다.
3. 코드가 망가졌을 때는 수정할 수 있다.
4. 반복한다.

다른 모든 것은 헷갈리기만 할 뿐이니 신경 쓰지 말고, 계획대로 따라오세요.

0.6 다른 텍스트 편집기

프로그래머에게 텍스트 편집기란 대단히 중요하지만, 초보자인 여러분에게는 단순한 프로그래머용 편집기만 있으면 됩니다. 이런 편집기는 소설이나 책을 쓸 때와 달리 컴퓨터 코드에 필요한 특별한 기능을 갖고 있어요. 책에서는 무료이고 거의 어디서든 돌아가는 Atom을 권하고 있습니다. 하지만 여러분이 가진 컴퓨터에서 Atom이 잘 동작하지 않는다면, 다음의 다른 편집기도 써보세요.

편집기	지원 환경	다운로드
Visual Studio Code	윈도우, 맥, 리눅스	*https://code.visualstudio.com*
Notepad++	윈도우	*https://notepad-plus-plus.org*
gEdit	리눅스, 맥, 윈도우	*https://github.com/GNOME/gedit*
Textmate	맥	*https://github.com/textmate/textmate*
SciTE	윈도우, 리눅스	*http://www.scintilla.org/SciTE.html*
jEdit	리눅스, 맥, 윈도우	*http://www.jedit.org*

편집기의 순서는 잘 동작할 것 같은 순서입니다. 어떤 프로젝트는 버려지거나 죽거나 여러분 컴퓨터에서 더 이상 돌아가지 않을지도 모른다는 점을 명심하세요. 하나를 해 보고 잘 안 되면 다른 걸 해보세요. '지원 환경'도 마찬가

지로 잘 동작하는 순서대로입니다. 윈도우에서 사용할 편집기를 찾고 있다면 '지원 환경' 열에서 윈도우가 처음으로 나오는 편집기를 찾아보세요.

Vim이나 Emacs를 이미 쓸 줄 안다면 마음껏 쓰세요. 하지만 아직 한번도 써 본 적이 없다면 쓰지 마세요. Vim이나 Emacs를 써야 한다고 설득하려 드는 프로그래머들이 있을지도 모르는데, 그저 갈 길에서 멀어질 뿐입니다. 여러분의 목표는 파이썬을 배우는 거에요. Vim이나 Emacs를 배우는 게 아니고요. Vim을 써보려다가 끄지도 못하게 되었으면 키보드로 :q! 나 ZZ를 입력하세요. 여러분에게 Vim을 써보라던 사람이 이것조차도 알려주지 않았다면, 왜 그 사람들 말을 들으면 안 되는지 알겠죠?

이 책으로 공부하는 동안은 통합 개발 환경(IDE, Integrated Development Environment)은 쓰지 마세요. 통합 개발 환경에 의존한다는 얘기는, 앞으로 새 언어를 배우려면 그 언어에 맞는 통합 개발 환경을 만들어주는 회사가 나타날 때까지는 그 언어로는 프로그래밍할 수 없다는 말입니다. 새 언어가 나오면 탄탄한 수익을 낼 수 있는 고객층이 생길 만큼 커지기 전까지는 써보지도 못한다는 얘기지요. 프로그래머용 텍스트 편집기(Vim, Emacs, Atom 등)만 갖고도 프로그래밍할 수 있다는 자신이 있으면 다른 개발도구를 기다릴 필요가 없습니다. (거대한 코드 기반 위에서 작업하는 경우처럼) 통합 개발 환경이 유리한 상황도 있지만, 거기에만 얽매이면 여러분의 가능성도 거기에 얽매입니다.

IDLE도 역시 쓰지 말아야 합니다. 심각한 제약 아래에서 동작하고, 그다지 잘 만들어진 소프트웨어도 아닙니다. 여러분에게 필요한 건 간단한 텍스트 편집기와 셸과 파이썬뿐입니다.

연습 1

첫 번째 프로그램

잊지 마세요. 여러분은 앞에서 텍스트 편집기를 설치해서 실행해 보고, 터미널을 실행하고, 두 가지를 함께 쓰는 법을 배우느라 상당한 시간을 써야 했답니다. 아직 그 과정을 마치지 못했다면 여기서 멈추세요. 앞 부분을 마쳐야지만 다음을 즐겁게 해 나갈 수 있습니다. 이번 장에서만 건너뛰거나 앞서 나가지 말라는 경고를 하고 이제 시작하겠습니다.

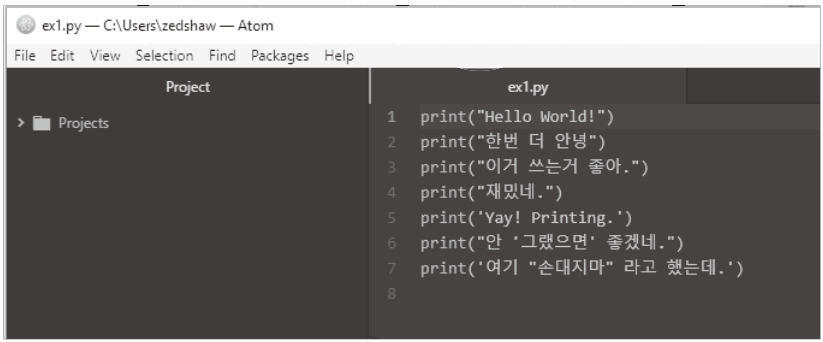
파일을 하나 만들어 이름을 ex1.py로 짓고 다음 내용을 써넣으세요. 중요한 내용입니다. 파이썬은 파일 이름이 .py로 끝나야 잘 동작하거든요.

0장을 건너뛰면 이 책을 제대로 공부하는 게 아닙니다. IDLE이나 IDE로 하려구요? 0장에서 쓰지 말라고 얘기했으니 쓰지 말아야 합니다. 0장을 건너뛰었으면 꼭 되돌아가서 읽어보세요.

ex1.py

```
1   print("Hello World!")
2   print("한번 더 안녕")
3   print("이거 쓰는거 좋아.")
4   print("재밌네.")
5   print('Yay! Printing.')
6   print("안 '그랬으면' 좋겠네.")
7   print('여기 "손대지마" 라고 했는데.')
```

어떤 환경에서든 Atom 편집기는 다음과 같이 보여야 합니다.



편집기가 정확히 똑같이 보이지 않는다고 걱정하지는 마세요. 하지만 비슷하기는 해야 합니다. 창머리나 색깔이 조금 다를 수 있습니다. 창 왼쪽에도 'zedshaw'라고 보이는 대신 여러분이 파일을 저장한 디렉터리가 보이겠지요. 이런 차이점은 모두 괜찮습니다.

파일을 만들 때 중요한 점은 다음 내용입니다.

1 왼쪽에 있는 줄 번호는 입력한 게 아닙니다. '5행을 보세요'처럼, 줄을 정확하게 가리키기 위해 책에만 출력한 것입니다. 파이썬 스크립트에는 쓰지 말아야 합니다.
2 모든 행은 `print`로 시작하며, 위에서 보여드린 예제와 정확히 똑같습니다. '정확히'라는 말은 글자 하나도 틀리지 않고 모두 똑같다는 뜻이에요. 완전히 똑같아야만 동작합니다. 하지만 색은 모두 달라도 됩니다. 색은 상관없어요. 입력한 글자만 중요합니다.

맥OS 터미널이나 (아마도) 리눅스에서는 파일을 이렇게 실행합니다.

```
python3.6 ex1.py
```

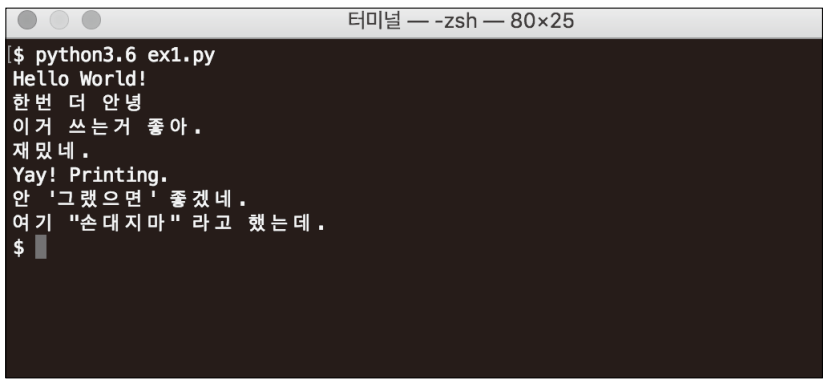
윈도우에서는 `python3.6` 대신 `python`이라고 입력한다는 걸 꼭 기억해 둡시다.

```
python ex1.py
```

올바르게 따라 했다면 다음의 실행 결과와 똑같은 결과가 나옵니다. 아니라면 무엇인가 잘못한 것입니다. 아니요. 컴퓨터가 잘못된 것은 아니에요.

1.1 실행 결과

맥OS 터미널에서는 이렇게 보입니다.



윈도우 파워셸에서는 이렇게 보입니다.

`python ex1.py` 이전 부분의 컴퓨터 이름이나 나머지 내용은 다를 수도 있습니다. 하지만 중요한 부분은 여러분이 명령어를 입력하자 앞의 화면과 똑같은 결과가 나왔다는 것입니다.

오류(error)가 있다면 이런 식일 것입니다.

```
$ python ex/ex1.py
    File "ex/ex1.py", line 3
        print "I like typing this.
                                  ^
   SyntaxError: EOL while scanning string literal
```

이 오류는 읽을 수 있어야 합니다. 이런 실수는 자주 할 테니까요. 저조차도 이런 실수를 자주 합니다. 한 줄씩 읽어봅시다.

1. 터미널에서 ex1.py 스크립트를 실행하도록 명령했습니다.
2. 파이썬이 ex1.py 파일의 셋째 줄에 오류가 있다고 알려줍니다.
3. 그 줄을 볼 수 있게 출력해 줍니다.
4. ^(캐럿, caret) 문자로 문제가 생긴 지점을 표시해줍니다. "(큰따옴표, double quote) 문자가 빠져있네요. 확인해보세요.
5. 마지막으로 'SyntaxError'를 출력하고 무엇 때문에 오류가 생겼을 수 있는지 알려줍니다. 몹시 아리송할 때가 많겠지만 검색 엔진에 복사해 넣으면 같은 오류를 겪은 다른 사람을 찾을 수 있습니다. 대부분은 고치는 방법까지 알아낼 수 있습니다.

1.2 더 해보기

각 장에는 '더 해보기' 절이 있습니다. '더 해보기'에는 여러분이 도전해 볼 만한 과제들이 있습니다. 모르겠으면 건너뛰었다 나중에 다시 보세요.

이번 장에서는 이런 것을 더 해보세요.

1. 한 줄 더 출력하도록 스크립트를 고쳐보세요.
2. 여러 줄 가운데 한 줄만 출력하도록 스크립트를 고쳐보세요.
3. #(해시) 문자를 줄 앞에 넣어보세요. 어떤 일이 일어났나요? 이 문자가 무슨 일을 하는지 알아내보세요.

지금부터는 특별히 다른 구성이 없는 한 각 장이 어떤 식으로 구성되어 있는지 설명하지 않겠습니다.

1.3 자주 묻는 질문

이 책의 온라인 페이지에서 댓글란에 학생들이 실제로 남긴 질문입니다. 여러분도 같은 문제를 겪을 수 있으니 자주 묻는 질문을 골라서 답해뒀습니다.

Q. IDLE을 써도 되나요?

A. 아니요. 책에 나온 대로 맥OS에서는 터미널을 쓰고 윈도우에서는 파워셸을 쓰세요. 어떻게 쓰는지 모른다면 부록 '명령줄 완전 정복'을 보세요.

Q. 편집기에 색은 어떻게 입히나요?

A. 먼저 ex1.py 같이 파일을 .py 파일로 저장하세요. 다음부터는 입력할 때마다 색이 입혀집니다.

Q. ex1.py를 실행하면 `SyntaxError: invalid syntax`가 나와요.

A. 파이썬을 한 번 실행하고 나서 그 안에서 다시 파이썬을 실행하려고 한 것입니다. 터미널을 닫고 다시 실행하자마자 python ex1.py라고만 입력하세요.

Q. `can't open file 'ex1.py': [Errno 2] No such file or directory.`가 나와요.

A. 파일을 만든 디렉터리에 들어가 있어야 합니다. cd 명령을 써서 해당 디렉터리로 가세요. 예를 들어 파일을 lpthw/ex1.py에 저장했다면 `python3.6 ex1.py`를 실행하기 전에 `cd lpthw/`부터 하세요. 무슨 뜻인지 모르겠으면 부록 '명령줄 완전 정복'을 보세요.

연습 2

주석과 해시 문자

프로그래밍에서 주석(comment)은 아주 중요합니다. 사람이 읽어야 할 내용을 사람이 쓰는 언어로 쓰는 것이 주석입니다. 프로그램 일부를 임시로 지워야 할 때 코드를 비활성화(disable)하기 위해서도 쓰입니다. 파이썬에는 주석을 다음과 같이 씁니다.

ex2.py

```
1   # 주석, 나중에 프로그램을 읽을 수 있게 도와줍니다.
2   # 파이썬에서 # 뒤에 쓰인 것은 무엇이든 무시됩니다.
3
4   print("이렇게 코드를 쓸 수 있습니다.") # 그리고 주석 뒤는 무시됩니다.
5
6   # 주석은 코드 일부를 주석 처리해 '비활성화'할 때도 사용할 수 있습니다.
7   # print("실행되지 않습니다.")
8
9   print("실행됩니다.")
```

지금부터는 이런 식으로 코드를 쓰겠습니다. 모든 것이 보이는 그대로일 필요는 없다는 점을 이해해야 해요. 화면과 프로그램이 눈에는 다르게 보일 수도 있지만, 중요한 내용은 여러분이 텍스트 편집기에서 파일에 입력하는 텍스트입니다. 다른 텍스트 편집기를 쓰더라도 결과는 똑같이 나와요.

2.1 실행 결과

Exercise 2 Session

```
$ python3.6 ex2.py
이렇게 코드를 쓸 수 있습니다.
실행됩니다.
```

다시 한번 말하지만, 책에서는 환경에 따라 다른 모든 터미널의 스크린샷을

보여주지 않겠습니다. 이 내용은 실행 결과를 눈에 보이는 그대로 보여주는 것이 아니라는 점을 명심하세요. 대신 첫 줄의 `$ python3.6`과 마지막 줄 $ 사이의 내용에 주목하세요.

책과 달리 내 터미널에서는 각 줄이 $로 시작하지 않나요? $ 앞부분은 복잡하고 컴퓨터마다 다르기 때문에 간단하게 표시하고 있습니다. 부록 '명령줄 완전 정복'을 먼저 보고 돌아오세요.

2.2 더 해보기

1. # 기호의 기능에 대해 추측했던 내용이 맞았나 확인하고 어떻게 부르는지 알아두세요.
2. ex2.py 파일을 거꾸로 한 줄씩 다시 보세요. 마지막 줄에서 시작해 한 단어씩 입력했던 순서와 반대로 확인하세요.
3. 실수한 게 더 있나요? 마저 고치세요.
4. 입력한 내용을 크게 읽으세요. 기호는 각각 그 이름으로 읽으세요. 실수한 게 더 있나요? 마저 고치세요.

2.3 자주 묻는 질문

Q. 왜 `print("안녕 # 여러분.")`의 #는 무시되지 않나요?

A. 그 #는 문자열 안에 있습니다. 따라서 문장 끝의 " 문자가 나타나기 전까지는 문자열의 일부입니다. 이 # 문자는 그냥 문자로 취급하고, 주석으로 처리하지 않습니다.

Q. 여러 줄은 어떻게 주석 처리하나요?

A. 매 줄마다 앞에 #를 붙이세요.

Q. 왜 코드를 거꾸로 읽어야 하죠?

A. 코드의 의미에 매달리지 않고 정말 코드를 한 부분씩 처리할 수 있도록 생각하게 도와주는 기법입니다. 이 방법으로 코드를 정확히 한 부분씩 비교해 오류를 찾을 수 있습니다. 간편한 오류 점검 기법이기도 합니다.

연습 3

수와 계산

모든 프로그래밍 언어에는 수(number)와 계산(math)을 다루는 방법이 있습니다. 걱정하지 마세요. 프로그래머들이란 사실은 그렇지도 않으면서 수학 천재인 척 굴며 거짓말하곤 합니다. 진짜 수학 천재였다면 버그 덩어리 웹 프레임워크를 만드는 대신 진짜 수학을 하고 있겠지요.

이번 장에서는 많은 수학 기호(symbol)를 다룹니다. 기호에 붙은 올바른 이름을 알아봅시다. 하나씩 입력하며 이름을 말해보세요. 지겹게 느껴지면 소리내기는 그만해도 됩니다. 기호 이름을 살펴봅시다.

- + 덧셈(plus)
- - 뺄셈(minus)
- / 나눗셈(slash)
- * 곱셈(asterisk)
- % 나머지(percent)
- 〈 작다 (less-than)
- 〉 크다 (greater-than)
- 〈= 작거나 같다(less-than-equal)
- 〉= 크거나 같다(greater-than-equal)

어떤 연산을 하는지가 빠졌죠? 이번 장에서 코드에 써본 다음, 돌아와서 각각 어떤 역할인지 찾아내 표를 완성하세요. 예를 들어 +는 덧셈 연산자입니다.

ex3.py

```
1   print("닭을 세어봅시다.")
2
3   print("암탉", 25 + 30 / 6)
4   print("수탉", 100 - 25 * 3 % 4)
5
6   print("이제 달걀도 세어봅시다.")
7
8   print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)
9
10  print("3 + 2 < 5 - 7 는 참인가요?")
11
12  print(3 + 2 < 5 - 7)
13
14  print("3 + 2 는 얼마죠?", 3 + 2)
15  print("5 - 7 은 얼마죠?", 5 - 7)
16
17  print("아하 이게 False인 이유네요.")
18
19  print("더 해볼까요.")
20
21  print("더 큰가요?", 5 > -2)
22  print("더 크거나 같나요?", 5 >= -2)
23  print("더 작거나 같나요?", 5 <= -2)
```

3.1 실행 결과

Exercise 3 Session

```
$ python3.6 ex3.py
닭을 세어봅시다.
암탉 30.0
수탉 97
이제 달걀도 세어봅시다.
6.75
3 + 2 < 5 - 7 는 참인가요?
False
3 + 2 는 얼마죠? 5
5 - 7 은 얼마죠? -2
아하 이게 False인 이유네요.
더 해볼까요.
더 큰가요? True
더 크거나 같나요? True
더 작거나 같나요? False
```

3.2 더 해보기

1. 줄마다 그 줄이 뭘 하는지 주석으로 써서 스스로에게 설명해보세요. 주석은 #를 이용해 쓸 수 있습니다.
2. '0. 설정' 장에서 파이썬을 처음 시작했을 때가 기억나세요? 그때처럼 파이썬을 다시 실행해 보고, 위에서 쓴 기호와 아는 내용을 이용해 파이썬을 계산기처럼 써보세요.
3. 계산할 거리를 찾아보고 그 계산을 하는 .py 파일을 만들어보세요.
4. ex3.py를 부동소수점을 이용해 더 정확하게 새로 만들어주세요. 20.0이 부동소수점입니다.

3.3 자주 묻는 질문

Q. 왜 % 문자는 '퍼센트'가 아니라 '나머지(modulus)'인가요?

A. 가장 큰 이유는 파이썬을 설계한 사람들이 그 기호를 그렇게 쓰기로 결정했기 때문이에요. 일반적인 글에서는 %를 '퍼센트'로 읽는 게 옳습니다. 프로그래밍에서 퍼센트는 보통 단순히 / 연산자로 나눠서 계산합니다. %는 퍼센트와는 상관없이 그냥 나머지 연산의 기호를 %로 정한 것일 뿐입니다.

Q. %는 어떻게 동작하나요?

A. 다르게 말하자면, "X를 Y로 나누면 나머지는 J이다."라고 할 수 있습니다. 예를 들어, "100을 16으로 나누면 나머지는 4이다." 같이요. %의 결과는 나머지 부분이라고 부르는 J입니다.

Q. 연산의 우선순위는 어떻게 되죠?

A. 수학에서 쓰는 순서는 괄호, 지수, 곱셈과 나눗셈, 덧셈과 뺄셈입니다. 파이썬도 이 순서를 따릅니다.

연습 4

변수와 이름

이제 여러분은 print로 출력(print)도 할 수 있고 계산도 할 수 있습니다. 다음 단계로 변수(variable)에 대해 배워봅시다. 프로그래밍에서 변수란 어떤 내용을 쓴 코드를 가리키는 이름일 뿐입니다. '제드 쇼'라는 제 이름이 '이 책을 쓴 사람'을 가리키는 이름인 것처럼요. 프로그래머의 기억이란 형편없기 때문에, 사람이 읽기 쉽도록 변수를 이용해 코드에 이름을 붙입니다. 좋은 변수 이름을 붙여가며 소프트웨어를 만들어야 코드를 다시 읽을 때 헤매지 않습니다.

이번 장을 하는 동안 막히는 부분이 있다면, 앞에서 배운, 차이를 찾고 세부에 집중하게 해주는 비법을 기억하세요.

1. 매 줄마다 그 줄이 무엇을 하는지 스스로에게 설명하는 주석을 글로 써보세요.
2. .py 파일을 거꾸로 읽어보세요.
3. .py 파일을 큰 소리로 읽어보세요. 기호까지도요.

ex4.py

```
1   자동차 = 100
2   차_안_공간 = 4.0
3   운전사 = 30
4   승객 = 90
5   운행_안하는_차 = 자동차 - 운전사
6   운행하는_차 = 운전사
7   총_정원 = 운행하는_차 * 차_안_공간
8   차당_평균_승객 = 승객 / 운행하는_차
9
10
11  print("자동차", 자동차, "대가 있습니다.")
12  print("운전자는", 운전사, "명뿐입니다.")
13  print("오늘은 빈 차가", 운행_안하는_차, "대일 것입니다.")
```

```
14  print("오늘은", 총_정원, "명을 태울 수 있습니다.")
15  print("함께 탈 사람은", 승객, "명 있습니다.")
16  print("차마다", 차당_평균_승객, "명 정도씩 타야 합니다.")[1]
```

차_안_공간 사이의 _는 밑줄(underscore) 문자라 불립니다. 어떻게 입력하는지 모르면 찾아보세요. 변수 이름에서 낱말 사이에 가상으로 띄어쓰기를 넣을 때 많이 쓰는 문자입니다.

4.1 실행 결과

Exercise 4 Session

```
$ python3.6 ex4.py
자동차 100 대가 있습니다.
운전자는 30 명 뿐입니다.
오늘은 빈 차가 70 대일 것입니다.
오늘은 120.0 명을 태울 수 있습니다.
함께 탈 사람은 90 명 있습니다.
차마다 3.0 명 정도씩 타야 합니다.
```

4.2 더 해보기

처음에 이 프로그램을 짜다가 실수했더니 파이썬이 이렇게 알려줬습니다.

```
Traceback (most recent call last):
  File "ex4.py", line 8, in <module>
    차당_평균_승객 = 총정원 / 승객
NameError: name '총정원' is not defined
```

이 오류를 직접 설명해보세요. 반드시 행 번호를 쓰며 이유를 설명하세요.

더 해봐야 하는 숙제도 많이 드리겠습니다.

1. 차_안_공간의 값으로 4.0을 썼는데, 꼭 그래야 하나요? 그냥 4면 어떻게 되죠?
2. 4.0이 '부동소수점(floating point)'이라는 것을 기억하세요. 그냥 소수점 달린 숫자인데, 부동소수점이기 때문에 4 대신 4.0이라고 써야 해요.

1 (옮긴이) 책에서 변수 이름 등은 특별한 관례가 없는 한 한국어로 썼습니다. 영어로 된 예제로 공부하고 싶으면 다음 주소에서 저자의 원래 코드를 참고할 수 있습니다. *https://github.com/zedshaw/learn-python3-thw-code*

3. 모든 변수 대입 위에 주석을 달아보세요.
4. =(등호)의 이름과 용도를 확실히 알아두세요. 다른 값(숫자, 문자열 등)에 이름(**운행하는_차, 승객**)을 붙일 때 씁니다
5. _는 밑줄 문자입니다. 잊지 마세요.
6. 앞에서처럼 터미널에서 python3.6을 실행해 계산기처럼 써보고, 계산에 변수 이름도 써보세요. 널리 쓰이는 변수 이름으로는 `i`, `x`, `j`가 있습니다.

4.3 자주 묻는 질문

Q. =(홑등호)와 ==(겹등호)는 무엇이 다른가요?

A. =(홑등호)는 오른쪽의 값을 왼쪽의 변수에 대입합니다. ==(겹등호)는 양쪽의 값이 같은지 비교합니다. 27장에서 배우겠습니다.

Q. x = 100 대신 x=100으로 쓸 수 있나요?

A. 네. 하지만 나쁜 형태입니다. 쉽게 읽을 수 있도록 연산자 주위는 띄어 써야 합니다.

Q. '파일을 거꾸로 읽어보세요'가 무슨 뜻인가요?

A. 아주 간단합니다. 16행짜리 코드가 있다고 생각해보세요. 16행부터 시작합니다. 그리고 예제 파일의 16행과 비교합니다. 15행에서도, 그 앞에서도 반복하고, 파일 전체를 다 읽을 때까지 계속합니다.

Q. 왜 차_안_공간 값에 4.0을 썼나요?

A. 가장 큰 이유는 여러분이 부동소수점 수가 무엇인지 찾아내고 이 질문을 하도록 하기 위해서였죠. '4.2 더 해보기'를 보세요.

이 책에서는 코드에서 대부분의 변수나 함수(나중에 배웁니다) 이름을 한글로 썼습니다. 영어에 익숙하지 않은 독자들이 영어와 파이썬에 모두 신경 쓰기보다는 익숙한 한국어로 보다 빠르게 파이썬을 배우기를 바라는 마음이기 때문이지요. 여러분이 프로그래밍의 세계에 본격적으로 발을 들이게 된다면, 실제로 프로그래밍은 대부분 영어로 하게 됩니다. 한국에서만 쓰는, 한국어 사용자를 위한 코드도 (아직은) 그렇습니다. 코드를 짜는 데는 사소해 보이지만 중요한 관습이 있습니다. 이를테면 프로그래밍 언어를 막론하고 대부분의 프로그래머는 단수와 복수 표현에 민감합니다. 똑같은 의미의 변수를 쓰더라도 변수 이름이 단수인지 복수인지에 따라 변수가 어떤 자료를 가리키고 있는지 의미가 달라지거든요. 파이썬 프로그래머들은 대소문자의 구분도 중요하게 생각합니다. 대소문자의 구분으로 코드를 더 읽기 쉽도록 하는 기법이 사용되기 때문이지요. 아쉽지만 코드를 한글로 옮기는 과정에서 한국어다움을 지키면서 온전히 옮길 수 있는 방법은 없었습니다. 단수와 복수의 구분은, 꼭 복수형으로 표시해야 하는 경우에 모두 복수 접미사 '들'을 써서 표현했습니다. 한글로 쓴 코드에 굳이 '들'이 들어가 있다면 의도적으로 구분한 부분이니 왜 들어갔는지 생각해보세요. 대소문자를 이용한 구분은 살릴 수 없었습니다. 책을 모두 마친 뒤에 영어로 된 예제 코드를 다시 살펴보고 파이썬 코드의 포맷에 대하여 더 알아보면 더욱 파이썬다운 코드를 짜는 데 도움이 될 것입니다.