A Byte of Python (□□□□□□□□□□□□)

Contents

A Byte of Python **5**
- 파이썬 3 대응 . . . 5
- 한국어 번역 . . . 5
- 서평 . . . 6
- 수업 교재 . . . 9
- 권리 및 권한 . . . 10
- 책을 읽으세요! . . . 10
- 책 구입하기 . . . 10
- 내려받기 . . . 10
- 번역본 읽기 . . . 10

헌정 **11**

서문 **11**
- 이 책은 누가 읽으면 좋을까요? . . . 11
- 공식 홈페이지 . . . 11
- 생각할 것들 . . . 11

파이썬에 대하여 **11**
- 파이썬이라는 이름의 유래 . . . 12
- 파이썬의 특징 . . . 12
  - 단순함 . . . 12
  - 배우기 쉬운 언어 . . . 12
  - 자유, 오픈 소스 소프트웨어 . . . 12
  - 고수준 언어 . . . 12
  - 이식성 . . . 12
  - 인터프리터 언어 . . . 13
  - 객체 지향 언어 . . . 13
  - 확장성 . . . 13
  - 포함성 . . . 13
  - 확장 가능한 라이브러리 . . . 13
  - 요약 . . . 13
- 파이썬 2 vs 3 . . . 13
- 프로그래머들이 말하는 파이썬 . . . 14

설치 **14**
- 윈도우 환경에서의 설치 . . . 14
  - 명령 프롬프트 . . . 15
  - 윈도우에서 파이썬 프롬프트 실행하기 . . . 15
- Mac OS X 에서의 설치 . . . 15
- GNU/Linux 에서의 설치 . . . 16
- 요약 . . . 16

첫 걸음 **16**
인터프리터 프롬프트를 활용하는 방법 . . . 16
인터프리터 프롬프트를 종료하는 방법 . . . 17
편집기 선택하기 . . . 17
PyCharm . . . 17
Vim . . . 18
Emacs . . . 18
소스 파일을 활용하는 방법 . . . 18
도움 받기 . . . 19
요약 . . . 20

기초 **20**
주석 . . . 20
리터럴 상수 . . . 20
숫자형 . . . 21
문자열 . . . 21
작은 따옴표 . . . 21
큰 따옴표 . . . 21
따옴표 세 개 . . . 21
문자열은 수정이 불가능함 . . . 21
문자열 포맷팅 . . . 22
이스케이프(Escape) 문자 . . . 23
Raw 문자열 . . . 24
변수 . . . 24
식별자 이름 짓기 . . . 24
자료형 . . . 24
객체 . . . 24
파이썬 프로그램 작성하기 . . . 25
PyCharm을 이용할 경우 . . . 25
다른 편집기를 이용할 경우 . . . 25
예제: 변수와 리터럴 상수 사용하기 . . . 25
논리적/물리적 명령줄 . . . 26
들여쓰기 . . . 27
요약 . . . 27

연산자와 수식 **27**
연산자 . . . 27
연산 및 할당 연산자 . . . 29
연산 순서 . . . 29
연산 순서 변경 . . . 30
같은 연산 순서를 가질 경우 . . . 30
수식 예제 . . . 30
요약 . . . 31

흐름 제어 **31**
`if` 문 . . . 31
while 문 . . . 32
`for` 문 . . . 33
break 문 . . . 33
`continue` 문 . . . 34
요약 . . . 34

함수 **34**
함수의 매개 변수 . . . 35
지역 변수 . . . 35

global 문 . . . 35
기본 인수값 . . . 36
키워드 인자 . . . 36
VarArgs 매개 변수 . . . 37
return 문 . . . 37
DocString . . . 38
요약 . . . 38

모듈 **38**
바이트 컴파일된 .pyc 파일 . . . 39
from..import 문 . . . 39
모듈의 __name__ 속성 . . . 40
새로운 모듈 작성하기 . . . 40
dir 내장함수 . . . 41
패키지 . . . 42
요약 . . . 42

자료구조 **42**
리스트 . . . 42
객체와 클래스에 대한 간단한 소개 . . . 43
튜플 . . . 43
사전 (딕셔너리) . . . 44
열거형 . . . 45
집합 (Set) . . . 46
참조 . . . 46
문자열에 대한 좀 더 자세한 설명 . . . 47
요약 . . . 47

실생활 문제 해결 **47**
문제 . . . 47
첫번째 프로그램 . . . 48
두 번째 프로그램 . . . 49
세 번째 프로그램 . . . 49
네 번째 프로그램 . . . 49
더 많은 개선점 . . . 50
소프트웨어 개발 단계 . . . 50
요약 . . . 50

객체 지향 프로그래밍 **51**
self 에 대하여 . . . 51
클래스 . . . 52
메소드 . . . 52
__init__ 메소드 . . . 52
클래스 변수와 객체 변수 . . . 53
상속 . . . 54
요약 . . . 55

입력과 출력 **55**
사용자로부터 입력받기 . . . 55
연습 문제 . . . 55
파일 입/출력 . . . 56
피클 (Pickle) . . . 56
유니코드 . . . 56
요약 . . . 57

예외 처리 **57**
오류 (Error) . . . 57
예외 (Exception) . . . 58
예외 처리 . . . 58
예외 발생시키기 . . . 58
Try … Finally 문 . . . 59
with 문 . . . 59
요약 . . . 59

표준 라이브러리 **60**
`sys` 모듈 . . . 60
`logging` 모듈 . . . 60
'금주의 모듈' 시리즈 . . . 61
요약 . . . 61

더 많은 것들 **61**
튜플 넘기기 . . . 61
특별한 메소드들 . . . 62
한 줄짜리 블록 . . . 62
람다 (Lambda) 함수 . . . 62
리스트 축약 (Comprehension) . . . 63
함수 인자를 튜플이나 사전 형태로 넘겨받기 . . . 63
assert 문 . . . 63
데코레이터 . . . 64
파이썬 2와 3의 차이점 . . . 64
요약 . . . 64

남은 과제 **64**
더 많은 과제 . . . 65
예제 코드 읽기 . . . 65
참고 문서 . . . 65
파이썬 관련 영상 . . . 65
질문과 답변 . . . 65
튜토리얼 . . . 65
커뮤니티 . . . 65
새로운 소식 . . . 66
라이브러리 설치하기 . . . 66
홈페이지 제작 . . . 66
GUI 프로그램 만들기 . . . 66
그 외의 GUI 저작 도구들 . . . 66
다양한 파이썬 구현들 . . . 67
(고급 프로그래머를 위한) 함수형 프로그래밍 . . . 67
요약 . . . 67

부록: FLOSS **68**

부록: 끝맺음 **69**
이 책의 탄생 . . . 69
이 책의 십대 시절 . . . 69
현재 . . . 69
저자에 대하여 . . . 69

부록: 과거로부터의 교훈 **69**
책의 현재 . . . 70

부록: 리비전 기록 **70**

번역 **71**
Arabic . . . 72
Azerbaijani . . . 72
Brazilian Portuguese . . . 72
Catalan . . . 72
Chinese . . . 72
Earlier Chinese translation . . . 73
Chinese Traditional . . . 73
French . . . 74
German . . . 74
Greek . . . 74
Indonesian . . . 75
Italian (first) . . . 75
Italian (second) . . . 75
Japanese . . . 75
Korean . . . 75
Epsimatt (2019) . . . 75
Older . . . 75
Mongolian . . . 76
Norwegian (bokmål) . . . 76
Polish . . . 76
Portuguese . . . 76
Romanian . . . 76
Russian . . . 77
Ukranian . . . 77
Serbian . . . 77
Slovak . . . 77
Spanish . . . 77
Swedish . . . 78
Turkish . . . 78

번역 방법 **78**

피드백 **78**

A Byte of Python

"A Byte of Python" 은 무료로 제공되는 파이썬 프로그래밍 교재입니다. 이 책은 파이썬을 처음 접하는 분들을 위한 튜토리얼 및 가이드의 역할을 하도록 쓰여졌습니다. 컴퓨터로 텍스트 문서를 작성하고 저장할 줄만 알면 충분합니다! 이 책은 바로 그런 분들을 위해 쓰여진 책입니다.

파이썬 3 대응

이 책은 파이썬 3을 기반으로 서술되어 있습니다. 혹시라도 예전 버전의 파이썬 2 코드를 작성해야 하는 분들을 위한 가이드 또한 작성되어 있습니다.

한국어 번역

2022년 11월 박정빈 (jeongbin.park@pusan.ac.kr) 이 파이썬 3에 맞도록 책의 내용을 새롭게 갱신하였습니다. 이 책의 소스코드는 https://github.com/chaek-union/a_byte_of_python-korean_translation 에서 확인하실 수 있습니다.

이 책의 원 저자는 Swaroop C H 입니다. 책의 영어 원문은 http://www.swaroopch.com/notes/python/ 에서 확인하실 수 있습니다.

서평

아래는 독자들의 서평입니다:

> Somewhere around 2004 – 05 when I was convinced I wasn't smart enough to be a programmer , I came came across the original A Byte of #Python, and that changed my entire perspective on computing and life , I owe a lot to that book @swaroopch had written. – Rahul on Jul 30, 2020
>
> This is the book that got me into programming almost a decade ago. Thank you @swaroopch. You changed my life. – Stefan Froelich on Aug 2, 2019
>
> I am writing this email to thank you for the great help your book has done for me! It was a really good book that I enjoyed thoroughly. As a 15 year old who has never done programming before, trying to learn Python online was difficult and I couldn't understand anything. But I felt like your book gave was much easier to understand and eased me into the whole new world of programming. Thanks to you, I can now write a high level language with ease. I thought programming would be hard and boring, but with your book's help, I realised how fun and interesting yet simple it can be! I would like to thank you again for your hard work on helping out beginners like me. – Prottyashita Tahiyat on Sep 17, 2019
>
> This is the best beginner's tutorial I've ever seen! Thank you for your effort. – Walt Michalik
>
> The best thing i found was "A Byte of Python", which is simply a brilliant book for a beginner. It's well written, the concepts are well explained with self evident examples. – Joshua Robin
>
> Excellent gentle introduction to programming #Python for beginners – Shan Rajasekaran
>
> start to love python with every single page read – Herbert Feutl
>
> perfect beginners guide for python, will give u key to unlock magical world of python – Dilip
>
> I should be doing my actual "work" but just found "A Byte of Python". A great guide with great examples. – Biologist John
>
> Recently started reading a Byte of python. Awesome work. And that too for free. Highly recommended for aspiring pythonistas. – Mangesh
>
> A Byte of Python, written by Swaroop. (this is the book I'm currently reading). Probably the best to start with, and probably the best in the world for every newbie or even a more experienced user. – Apostolos
>
> Enjoying Reading #ByteOfPython by @swaroopch best book ever – Yuvraj Sharma
>
> A Byte of Python by @swaroopch is still the "Best newbie guide to python" – Nickson Kaigi
>
> Thank you so much for writing A Byte Of Python. I just started learning how to code two days ago and I'm already building some simple games. Your guide has been a dream and I just wanted to let you know how valuable it has been. – Franklin
>
> I'm from Dayanandasagar College of Engineering (7th sem, CSE). Firstly i want to say that your book "The byte of python" is too good a book for a beginner in python like me.The concepts are so well explained with simple examples that helped me to easily learn python. Thank you so much. – Madhura
>
> I am a 18 year old IT student studying at University in Ireland. I would like to express my gratitude to you for writing your book "A Byte of Python", I already had knowledge of 3 programming langagues – C, Java and Javascript, and Python was by far the easiest langague I have ever learned, and that was mainly because your book was fantastic and made learning python very

simple and interesting. It is one of the best written and easy to follow programming books I have ever read. Congratulations and keep up the great work. – Matt

Hi, I'm from Dominican Republic. My name is Pavel, recently I read your book A Byte of Python and I consider it excellent!! :). I learnt much from all the examples. Your book is of great help for newbies like me··· – Pavel Simo

I am a student from China, Now ,I have read you book A byte of Python, Oh it's beautiful. The book is very simple but can help all the first learnners. You know I am interesting in Java and cloud computing many times, i have to coding programm for the server, so i think python is a good choice, finish your book, i think its not only a good choice its must use the Python. My English is not very well, the email to you, i just wanna thank you! Best Wishes for you and your family. – Roy Lau

I recently finished reading Byte of Python, and I thought I really ought to thank you. I was very sad to reach the final pages as I now have to go back to dull, tedious oreilly or etc. manuals for learning about python. Anyway, I really appreciate your book. Samuel Young

Dear Swaroop, I am taking a class from an instructor that has no interest in teaching. We are using Learning Python, second edition, by O'Reilly. It is not a text for beginner without any programming knowledge, and an instructor that should be working in another field. Thank you very much for your book, without it I would be clueless about Python and programming. Thanks a million, you are able to break the message down to a level that beginners can understand and not everyone can. – Joseph Duarte

I love your book! It is the greatest Python tutorial ever, and a very useful reference. Brilliant, a true masterpiece! Keep up the good work! – Chris–André Sommerseth

First of all, I want to say thanks to you for this great book. I think it is a good book for those who are looking for a beginner's tutorial for Python. It is about two or there years ago, I think, when I first heard of this book. At that time, I was unable to read books in English yet, so I got a chinese translation, which took me into the gate of Python programming. Recently, I reread this book. This time, of course, the english version. I couldn't believe that I can read the whole book without my dictionary at hand. Of course, it all dues to your effort to make this book an easy–to–understand one. – myd7349

I'm just e–mailing you to thank you for writing Byte of Python online. I had been attempting Python for a few months prior to stumbling across your book, and although I made limited success with pyGame, I never completed a program. Thanks to your simplification of the categories, Python actually seems a reachable goal. It seems like I have finally learned the foundations and I can continue into my real goal, game development. ··· Once again, thanks VERY much for placing such a structured and helpful guide to basic programming on the web. It shoved me into and out of OOP with an understanding where two text books had failed. – Matt Gallivan

I would like to thank you for your book A Byte of Python which i myself find the best way to learn python. I am a 15 year old i live in egypt my name is Ahmed. Python was my second programming language i learn visual basic 6 at school but didn't enjoy it, however i really enjoyed learning python. I made the addressbook program and i was sucessful. i will try to start make more programs and read python programs (if you could tell me source that would be helpful). I will also start on learning java and if you can tell me where to find a tutorial as good as yours for java that would help me a lot. Thanx. – Ahmed Mohammed

A wonderful resource for beginners wanting to learn more about Python is the 110–page PDF tutorial A Byte of Python by Swaroop C H. It is well–written, easy to follow, and may be the best introduction to Python programming available. – Drew Ames

Yesterday I got through most of Byte of Python on my Nokia N800 and it's the easiest and most concise introduction to Python I have yet encountered. Highly recommended as a starting point

for learning Python. – Jason Delport

Byte of Vim and Python by @swaroopch is by far the best works in technical writing to me. Excellent reads #FeelGoodFactor – Surendran

"Byte of python" best one by far man (in response to the question "Can anyone suggest a good, inexpensive resource for learning the basics of Python?") – Justin LoveTrue

The Book Byte of python was very helpful ..Thanks bigtime :) Chinmay

Always been a fan of A Byte of Python – made for both new and experienced programmers. – Patrick Harrington

I started learning python few days ago from your book..thanks for such a nice book. it is so well written, you made my life easy..so you found a new fan of yours..thats me :) tons of thanks. – Gadadhari Bheem

Before I started to learn Python, I've acquired basic programming skills in Assembly, C, C++, C# and Java. The very reason I wanted to learn Python is it's popular (people are talking about it) and powerful (reality). This book written by Mr. Swaroop is a very good guide for both brand–new programmers and new python programmers. Took 10 half days to go through it. Great Help! – Fang Biyi (PhD Candidate ECE, Michigan State University)

Thank you ever so much for this book!! This book cleared up many questions I had about certain aspects of Python such as object oriented programming. I do not feel like an expert at OO but I know this book helped me on a first step or two. I have now written several python programs that actually do real things for me as a system administrator. They are all procedural oriented but they are small by most peoples standards. Again, thanks for this book. Thank you for having it on the web. – Bob

I just want to thank you for writing the first book on programming I've ever really read. Python is now my first language, and I can just imagine all the possibilities. So thank you for giving me the tools to create things I never would have imagined I could do before. – "The Walrus"

I wanted to thank you for writing A Byte Of Python (2 & 3 Versions). It has been invaluable to my learning experience in Python & Programming in general. Needless to say, I am a beginner in the programming world, a couple of months of self study up to this point. I had been using youtube tutorials & some other online tutorials including other free books. I decided to dig into your book yesterday, & I've learned more on the first few pages than any other book or tutorial. A few things I had been confused about, were cleared right up with a GREAT example & explanation. Can't wait to read (and learn) more!! Thank you so much for not only writing the book, but for putting it under the creative commons license (free). Thank goodness there are unselfish people like you out there to help & teach the rest of us. – Chris

I wrote you back in 2011 and I was just getting into Python and wanted to thank you for your tutorial "A Byte of Python". Without it, I would have fallen by the wayside. Since then I have gone on to program a number of functions in my organization with this language with yet more on the horizon. I would not call myself an advanced programmer by any stretch but I notice the occasional request for assistance now from others since I started using it. I discovered, while reading "Byte" why I had ceased studying C and C++ and it was because the book given to me started out with an example containing an augmented assignment. Of course, there was no explanation for this arrangement of operators and I fell on my head trying to make sense of what was on the written page. As I recall it was a most frustrating exercise which I eventually abandoned. Doesn't mean C or C++ is impossible to learn, or even that I am stupid, but it does mean that the documentation I worked my way through did not define the symbols and words which is an essential part of any instruction. Just as computers will not be able to understand a computer word or computer symbol that is outside the syntax for the language being used, a student new to any field will not grasp his subject if he encounters words or symbols for which there are no definitions. You get a "blue screen" as it were in either case. The solution is simple,

though: find the word or symbol and get the proper definition or symbol and lo and behold,the computer or student can proceed. Your book was so well put together that I found very little in it I couldn't grasp. So, thank you. I encourage you to continue to include full definitions of terms. The documentation with Python is good, once you know, (the examples are its strength from what I see) but in many cases it seems that you have to know in order to understand the documentation which to my mind is not what should be. Third party tutorials express the need for clarification of the documentation and their success largely depends on the words that are used to describe the terminology. I have recommended your book to many others. Some in Australia, some in the Caribbean and yet others in the US. It fills a niche no others do. I hope you are doing well and wish you all the success in the future. – Nick

hey, this is ankush(19). I was facing a great difficulty to start with python. I tried a lot of books but all were bulkier and not target oriented; and then i found this lovely one, which made me love python in no time. Thanks a lot for this "beautiful piece of book". – Ankush

I would like to thank you for your excellent guide on Python. I am a molecular biologist (with little programming background) and for my work I need to handle big datasets of DNA sequences and to analyse microscope images. For both things, programming in python has been useful, if not essential to complete and publish a 6–years project. That such a guide is freely available is a clear sign that the forces of evil are not yet ruling the world! :) – Luca

Since this is going to be the first language you learn, you should use A Byte of Python. It really gives a proper introduction into programming in Python and it is paced well enough for the average beginner. The most important thing from then on will be actually starting to practice making your own little programs. – "{Unregistered}"

Just to say a loud and happy thank you very much for publishing "A Byte of Python" and "A Byte of Vim". Those books were very useful to me four or five years ago when I starting learning programming. Right now I'm developing a project that was a dream for a long, long time and just want to say thank you. Keep walking. You are a source of motivation. All the best. – Jocimar

Finished reading A byte of Python in 3 days. It is thoroughly interesting. Not a single page was boring. I want to understand the Orca screen reader code. Your book has hopefully equipped me for it. – Dattatray

Hi, 'A byte of python' is really a good reading for python beginners. So, again, NICE WORK! i'm a 4 years experienced Java&C developer from China. Recently, i want to do some work on zim–wiki note project which uses pygtk to implement. i read your book in 6 days, and i can read and write python code examples now. thx for your contribution. plz keep your enthusiasm to make this world better, this is just a little encourage from China. – Lee

I am Isen from Taiwan, who is a graduating PhD student in Electrical Engineering Department of National Taiwan University. I would like to thank you for your great book. I think it is not only just easy to read but also comprehensive and complete for a new comer of Python. The reason I read your book is that I am starting to work on the GNU Radio framework. Your book let me catch most of important core ideas and skill of Python with a minimum time. I also saw that you do not mind that readers send you a thank note in your book. So I really like your book and appreciate it. Thanks. – Isen I–Chun Chao

The book is even used by NASA! It is used in their Jet Propulsion Laboratory with their Deep Space Network project.

수업 교재

이 책은 다음과 같은 교육 기관에서 교재로 이용되었거나, 이용되고 있습니다:

- 'Principles of Programming Languages' course at Vrije Universiteit, Amsterdam

- 'Basic Concepts of Computing' course at University of California, Davis
- 'Programming With Python' course at Harvard University
- 'Introduction to Programming' course at University of Leeds
- 'Introduction to Application Programming' course at Boston University
- 'Information Technology Skills for Meteorology' course at University of Oklahoma
- 'Geoprocessing' course at Michigan State University
- 'Multi Agent Semantic Web Systems' course at the University of Edinburgh
- 'Introduction to Computer Science and Programming' at MIT OpenCourseWare
- 'Basic programming at the Faculty of Social Sciences, University of Ljubljana, Slovenia' – Aleš Žiberna says "I (and my predecessor) have been using your book as the main literature for this course"
- 'Introduction to programming', Department of Information Sciences, University of Zadar, Croatia – Krešimir Zauder says "I would like to inform you that A Byte of Python is a mandatory read at my course"

권리 및 권한

이 책은 Creative Commons Attribution-ShareAlike 4.0 International License 허가서 아래 배포됩니다.

이것은 당신이 다음의 권리를 갖는 것을 뜻합니다:

- 이 책의 복제, 배포, 전시, 공연 및 공중송신을 할 수 있습니다.
- 이 책을 개작, 수정하거나 이차저작물을 작성할 수 있습니다 (특히 번역판을 제작할 수 있습니다).
- 이 책을 영리 목적으로 이용할 수 있습니다.

다음의 내용을 숙지해주시기 바랍니다:

- 이 책의 전자책/출력본을 판매하실 경우, 명백하고 눈에 잘 띄는 방법으로 이 책의 원 저자로부터 판매되는 것이 아님을 명시하지 않는 한 이 책을 판매하실 수 없습니다.
- 이러한 권리에 관련된 내용은 반드시 책의 도입부에 적혀 있어야 하며, 이 문서의 첫 페이지에는 https://python.swaroopch.com 로 연결되는 링크가 반드시 있어야 하고, 원 저자의 글을 이 곳에서 내려받을 수 있다는 사실을 명시해야 합니다.
- 따로 명시되어 있지 않는 한, 이 책에서 사용된 모든 코드 및 스크립트는 3-clause BSD License 아래에 배포됩니다.

책을 읽으세요!

https://python.swaroopch.com 에서 이 책을 온라인으로 읽으실 수 있습니다.

책 구입하기

종이 책을 좋아하시는 분들, 혹은 이 책의 발전과 개선을 위해 도움을 주시려는 분들께서는 이 책의 하드카피 출력본을 https://swaroopch.com/buybook 에서 구입할 수 있습니다.

내려받기

https://github.com/swaroopch/byte-of-python/releases/latest 에 방문하여 PDF 파일을 다운로드 받으세요 (데스크톱에 최적화되어 있습니다). 또는, EPUB 파일을 받을 수도 있습니다 (스마트폰, 태블릿, ebook 리더기 등에 최적화되어 있습니다).

https://github.com/swaroopch/byte-of-python 에 방문하여 책의 소스를 확인하실 수도 있습니다 (교정, 수정, 번역 등을 위해).

번역본 읽기

이 책의 번역본을 읽고 싶으신 분들, 혹은 이 책을 번역하는데 도움을 주시려는 분은 번역 챕터를 읽어 주세요.

헌정

이 책을 우리로 하여금 GNU/Linux 및 오픈 소스의 세계로 이끌어주신 Kalyan Varma, 그리고 PESIT의 다른 많은 분들께 헌정하는 바입니다.

또한 저의 좋은 친구이자 스승이 되어주셨던, Atul Chitnis를 기억하며 이 책을 바칩니다.

마지막으로 지금의 인터넷을 탄생시킨 주역들에게 이 책을 바칩니다. 이 책은 2003년도에 처음으로 작성되었습니다만, 여전히 많이 읽히고 있습니다. 이것은 바로 이들이 개척해 왔던 지식의 공유 정신 덕분입니다.

서문

파이썬은 간단하면서도 강력한 프로그래밍 언어입니다. 파이썬은 초보자뿐 아니라 숙련자에게도 유용하며, 또 무엇보다 파이썬으로 프로그래밍하는 것은 매우 즐겁습니다. 이 책은 여러분이 파이썬이라는 아름다운 프로그래밍 언어를 배울 수 있도록 돕고, 여러분이 하고자 하는 일을 빠르고 쉽게 해결하는 방법을 보여드리는 것을 목적으로 작성되었습니다.

이 책은 누가 읽으면 좋을까요?

이 책은 파이썬의 가이드 혹은 튜토리얼의 역할을 하도록 작성되었습니다. 프로그래밍에 대해 아무런 지식이 없는 '완전 초보자'들을 주 독자로 설정하였습니다. 물론 경험이 많은 프로그래머들에게도 유용한 책입니다.

이 책의 목표는 컴퓨터로 텍스트 문서를 저장하는 것밖에 모르는 사람도 이 책을 통해 파이썬을 배울 수 있도록 하는 것입니다. 물론 여러분이 이전에 프로그래밍 경험이 있다고 하더라도 이 책을 통해 파이썬을 익힐 수 있을 것입니다.

만약 여러분이 전에 프로그래밍을 해 본 경험이 있다면, 아마도 여러분은 여러분이 가장 좋아하는 언어와 파이썬이 어떻게 다른지에 대해 관심이 있을 것입니다. 저는 이 책에서 다른 언어와 파이썬과의 많은 차이점을 강조해 두었습니다. 주의하세요, 얼마 안 지나서 여러분이 가장 좋아하는 언어는 파이썬이 될 것입니다!

공식 홈페이지

이 책의 공식 홈페이지는 https://python.swaroopch.com 이며 여기서 이 책을 온라인으로 읽으실 수 있고, 최신 버전의 책을 내려받을 수 있으며 책을 구매하거나 피드백을 남길 수 있습니다.

생각할 것들

> 소프트웨어를 설계하는 데에는 두 가지 방법이 있습니다. 하나는 설계를 매우 간단하게 하여서 결함이 없도록 하는 것입니다. 또 하나는 설계를 굉장히 복잡하게 하여 눈에 띄는 결함이 없도록 만들어 버리는 것입니다 (There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other is to make it so complicated that there are no obvious deficiencies). – C. A. R. Hoare
>
> 삶에 있어 성공이라고 하는 것은 재능과 기회보다는 집중력과 참을성에 달려 있습니다 (Success in life is a matter not so much of talent and opportunity as of concentration and perseverance). – C. W. Wendte

파이썬에 대하여

파이썬은 간단 하면서도 강력 하다고 할 수 있을 만한 언어입니다. 곧 여러분은 문제를 해결하는데 파이썬이라는 언어의 문법과 구조에 별로 신경쓰지 않고도 문제를 푸는 것 자체에 쉽게 집중할 수 있다는 데 놀라게 될 것입니다.

다음은 파이썬의 공식 소개글입니다:

> 파이썬은 배우기 쉽고 강력한 프로그래밍 언어입니다. 파이썬은 효율적인 고수준 자료구조 및 간단하고 효과적인 객체 지향 프로그래밍 접근법을 갖추고 있습니다. 우아한 문법과 동적 타이핑, 그리고 인터프리팅

환경을 갖춘 파이썬은 다양한 분야에서, 또 대부분의 플랫폼에서 사용될 수 있고, 빠르게 프로그램을 개발할 수 있는 최적의 스크립팅 언어라 할 수 있습니다.

다음 섹션에서 이러한 파이썬의 특징에 대해 상세히 다룰 것입니다.

파이썬이라는 이름의 유래

파이썬의 창시자 귀도 반 로섬(Guido van Rossum)이 BBC에서 방영되던 "Monty Python's Flying Circus"라는 TV 프로그램의 이름을 따서 지었습니다. 파이썬은 뱀의 일종인데 사실 귀도는 뱀이라는, 긴 몸으로 다른 동물의 몸을 휘감아 으깨어 부수고 먹어치우는, 동물을 딱히 좋아하지는 않는다고 합니다.

파이썬의 특징

단순함

파이썬은 단순하고 최소화된 언어입니다. 잘 쓰여진 파이썬 프로그램을 읽는 것은 좀 딱딱하게 쓰여진 영어 문장을 읽는 것과 크게 다르지 않습니다! 이러한 프로그램 코드같지 않아 보이는 특성은 파이썬의 가장 강력한 특성들 중 하나입니다. 이로 인해 파이썬이라는 언어 자체보다 여러분이 풀고자 하는 문제에 더 쉽게 집중할 수 있습니다.

배우기 쉬운 언어

곧 알게 되시겠지만, 파이썬은 정말 배우기 쉬운 언어입니다. 위에서 이미 이야기했지만, 파이썬은 굉장히 쉬운 문법 체계를 갖고 있습니다.

자유, 오픈 소스 소프트웨어

파이썬은 FLOSS (Free/Libré and Open Source Software – 자유, 오픈 소스 소프트웨어)의 좋은 예입니다. 이것은 이 소프트웨어의 복사본을 마음대로 배포할 수 있고, 소스 코드가 공개되어 있어 언제든지 읽을 수 있으며, 필요한 부분을 고칠 수 있고, 새로운 자유 소프트웨어를 작성할 때 이 프로그램의 일부를 사용해도 된다는 것을 의미합니다. FLOSS는 지식을 공유하는 공동체를 기반으로 하고 있습니다. 이것은 왜 파이썬이라는 언어가 이렇게 좋은 언어가 되었는지를 설명하는 좋은 이유입니다. 파이썬은 좀 더 나은 파이썬을 만들고자 하는 공동체의 노력에 의해 지속적으로 개선되고 있기 때문입니다.

고수준 언어

여러분이 파이썬으로 프로그램을 작성할 때, 메모리를 관리한다던가 하는 저수준의 세부적인 사항들을 신경쓸 필요가 없습니다.

이식성

파이썬은 소스가 공개되어 있으므로, 여러 플랫폼을 지원하도록 수정되어 왔습니다. 따라서 여러분이 프로그램을 작성할 때 특정 플랫폼에서만 사용되는 몇몇 기능들을 사용하지 않으면, 작성한 모든 파이썬 프로그램은 어떤 수정 없이도 파이썬이 동작하는 모든 플랫폼 위에서 동작할 수 있습니다.

파이썬은 GNU/Linux(리눅스), Windows(윈도우즈), FreeBSD, Macintosh(맥킨토시), Solaris(솔라리스), OS/2, Amiga(아미가), AROS, AS/400, BeOS, OS/390, z/OS, Palm OS(팜 OS), QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation(플레이스테이션), Sharp Zaurus(샤프 자우르스), Windows CE(윈도우즈 CE), PocketPC(포켓 PC) 상에서 동작합니다!

혹은 Kivy 와 같은 플랫폼을 활용하면, 여러분의 컴퓨터 및 iPhone, iPad, Android상에서 동작하는 게임을 제작할 수도 있습니다.

인터프리터 언어

이 부분은 설명이 조금 필요합니다.

컴파일러 언어인 C 혹은 C++로 작성된 프로그램은 컴파일러에 여러 옵션과 플래그를 주어 프로그래머가 작성한 소스 코드로부터 컴퓨터가 사용하는 언어 (0과 1로 구성된 바이너리 코드)로 번역하게 하는 과정을 거칩니다. 이렇게 번역된 프로그램을 실행하면, 링커 또는 로더라고 불리우는 소프트웨어가 프로그램을 하드 디스크로부터 메모리로 불러들인 후 프로그램을 실행하게 됩니다.

반면에 파이썬은 이러한 컴파일 과정을 필요로 하지 않습니다. 파이썬 프로그램은 파이썬으로 된 소스 코드로부터 곧바로 실행됩니다. 이때 파이썬은 내부적으로 실행되는 소스 코드를 '바이트코드'라고 불리우는 중간 단계의 형태로 변환한 후, 이것을 다시 여러분의 컴퓨터가 사용하는 언어로 변환한 다음 실제로 실행하게 됩니다. 사실 이 모든 과정은, 여러분이 컴파일이라는 과정을 신경쓰지 않고서도, 즉 여러분이 필요한 라이브러리를 갖고 있는지 링크가 잘 되었는지 잘 로드 되었는지 등을 신경쓰지 않고서도 파이썬을 쉽게 사용할 수 있게 해 줍니다. 또한 이 특성은 여러분이 작성한 파이썬 프로그램이 여러 플랫폼에서 잘 동작하게 해 주므로, 다른 컴퓨터에 프로그램을 단순히 복사하기만 해도 곧바로 잘 동작하게 됩니다!

객체 지향 언어

파이썬은 절차 지향 프로그래밍 및 객체 지향 프로그래밍을 지원합니다. 절차 지향 언어에서는, 프로그램은 '프로시저' 또는 '함수'들로 구성되는데 이것들은 단순히 프로그램에서 많이 재사용되는 코드 조각들을 의미합니다. 반면 객체 지향 언어에서는, 프로그램은 '객체'로 구성되어 있는데 객체란 데이터와 기능이 결합된 하나의 대상을 의미합니다. 파이썬은 특히 C++이나 Java와 같은 언어들에 비해 매우 강력하고도 쉬운 방법으로 객체 지향을 지원합니다.

확장성

만약 여러분이 작성해야 하는 프로그램의 일부분이 빠른 속도로 동작해야 하거나 혹은 알고리즘의 일부를 공개하고 싶지 않은 경우, 코드의 일부분을 C 혹은 C++로 작성한 후 파이썬 프로그램에서 읽어들여 사용하도록 할 수 있습니다.

포함성

여러분의 C/C++ 프로그램에 파이썬을 포함하도록 하여 여러분의 프로그램을 사용하는 사용자들이 '스크립팅' 기능을 사용하도록 할 수 있습니다.

확장 가능한 라이브러리

파이썬은 방대한 표준 라이브러리를 갖추고 있습니다. 여기에는 정규 표현식, 자동 문서 생성, 유닛 테스트, 쓰레딩, 데이터베이스, 웹 브라우저, CGI, FTP, 전자메일, XML, XML-RPC, HTML, WAV 파일, 암호화 알고리즘, GUI (graphical user interfaces) 등등이 들어 있으며, 여러 시스템 관련 기능들 또한 포함되어 있습니다. 이러한 기능들은 파이썬이 설치되어 있는 어떤 시스템에서든지 사용 가능하다는 점을 기억하세요.

이러한 표준 라이브러리 외에도, 파이썬 패키지 인덱스 (PyPI, Python Package Index) 에 다양한 라이브러리가 공개되어 있습니다.

요약

파이썬은 흥미진진하고 강력한 언어입니다. 파이썬의 빠른 성능과 여러 기능들의 조화는 여러분이 재미있고 쉽게 파이썬으로 프로그램을 작성할 수 있도록 해 줍니다.

파이썬 2 vs 3

만약 여러분이 파이썬 2와 3의 차이에 대해 큰 관심이 없으시다면 이 섹션을 무시하셔도 좋습니다. 그렇지만 여러분이 어떤 버전의 파이썬을 사용하고 있는지는 알고 계시는 것이 좋습니다. 이 책은 파이썬 3에 대응하도록 작성되었습니다.

일단 여러분이 어느 한 버전을 잘 이해하고 사용하는 방법을 익히고 나면, 두 버전의 차이점에 대해 쉽게 배울 수 있고 다른 버전 또한 쉽게 사용할 수 있게 될 것입니다. 이것보다 더 중요한 부분은 프로그래밍을 배우고 또한 파이썬의 기본을

이해하는 것입니다. 이것이 바로 이 책의 목표이며, 일단 목표를 달성하고 나면 상황에 따라 파이썬 2나 3 모두 쉽게 사용할 수 있게 될 것입니다.

파이썬 2와 파이썬 3의 차이점에 대해 자세하게 알고 싶으시다면 다음을 참조하세요:

- The future of Python 2
- Porting Python 2 Code to Python 3
- Writing code that runs under both Python2 and 3
- Supporting Python 3: An in-depth guide

프로그래머들이 말하는 파이썬

ESR(Eric S. Raymond)과 같은 위대한 해커들이 파이썬을 뭐라고 표현했는지 알아보세요:

- Eric S. Raymond 는 "성당과 시장(The Cathedral and the Bazaar)" 의 저자이며, 또한 오픈 소스 (Open Source) 라는 단어의 창시자입니다. 그는 어떻게 파이썬이 자신이 가장 좋아하는 언어가 되었는지에 대한 글을 남겼습니다. 또 이 글은 제가 파이썬을 처음 시작하게 된 계기가 된 글이기도 합니다.
- Bruce Eckel 은 유명한 책 'Thinking in Java' 와 'Thinking in C++' 의 저자입니다. 그는 파이썬이 아닌 다른 어떤 언어도 그가 파이썬을 사용할 때만큼 생산적이도록 하지 못했다고 말합니다. 또 그는 프로그래머에게 있어서 문제를 쉽게 해결하는 데 초점을 맞추는 언어는 아마도 파이썬이 유일하지 않을까 라는 말도 남겼습니다. 좀 더 자세한 사항은 인터뷰 전문 을 읽어 보세요.
- Peter Norvig 은 유명한 Lisp 프로그래머이며 또 구글의 '검색 품질 책임자' 로 일하고 있습니다 (이 부분 지적해준 귀도 반 로섬에게 감사를 전합니다). 그에 따르면 파이썬으로 프로그래밍 하는 것은 마치 의사코드로 프로그램하는 것과 비슷하다 고 합니다. 그는 또한 파이썬은 언제나 구글의 가장 중요한 부분을 담당하고 있다는 사실도 밝혀 주었습니다. 이에 대해서는 여러분이 직접 구글 채용 (Google Jobs) 페이지에 방문해 보시면 소프트웨어 엔지니어로 채용되는 조건에 파이썬에 대한 지식이 필수 사항으로 되어 있는 것을 확인해보실 수 있을 것입니다.

설치

이 책에서 "파이썬 3" 이라고 부르는 것은 파이썬 버전 3.6.0 이상의 모든 버전을 의미합니다.

윈도우 환경에서의 설치

https://www.python.org/downloads/ 에 방문하셔서 최신 버전의 설치 프로그램을 내려받아 설치하세요. 이 책이 쓰여질 때 기준으로 최신 버전은 3.5.1 입니다. 설치 방법은 다른 소프트웨어를 설치할 때와 같습니다.

만약 여러분들의 윈도우 버전이 "윈도우 비스타" 이전 버전이라면, 여러분은 파이썬 3.4 이하의 버전만 설치할 수 있습니다.

주의: 설치할 때 `Add Python 3.5 to PATH` 옵션을 꼭 체크하세요.

설치 경로를 변경하려면 `Customize installation` 을 누르고, `Next` 버튼을 누른 뒤, `C:\\python35` 나 원하는 다른 경로를 입력하면 됩니다.

만약 앞서 `Add Python 3.5 PATH` 옵션을 체크하지 않고 지나쳤다면, `Add Python to environment variables` 을 누르세요. 이것은 `Add Python 3.5 to PATH` 를 체크하고 설치하는 것과 같은 효과가 있습니다.

설치 중 `install Launcher for all users` 옵션은 선택해도 되고 안 해도 됩니다. 런처는 서로 다른 파이썬 버전들이 설치되어 있을 때 어떤 버전의 파이썬을 실행될지 선택해주는 역할을 하는 프로그램입니다.

만약 `PATH` 환경변수가 제대로 설정되지 않았을 경우 (즉, 위에서 `Add Python 3.5 Path` 나 `Add Python to environment variables` 옵션을 선택해 주었을 때), 다음 섹션 에 이를 올바르게 설정할 수 있는 방법이 설명되어 있습니다. 그런 뒤 섹션을 읽어 보세요.

NOTE: For people who already know programming, if you are familiar with Docker, check out Python in Docker and Docker on Windows.

명령 프롬프트

여러분이 파이썬을 윈도우의 명령 프롬프트 (DOS 프롬프트) 상에서 사용하고 싶으시다면, PATH 환경 변수를 알맞게 설정해 주어야 합니다.

윈도우 2000, XP, 2003의 경우, -〉 -〉 -〉 로 들어가세요. 이제 시스템 변수 목록에 있는 PATH 를 선택한 뒤, 버튼을 누르고 ;C:\\Python35 (이 폴더가 실제로 존재하는지 다시 한번 확인하세요. 더 최신 버전의 파이썬을 설치한 경우 폴더 이름이 다를 수 있습니다) 이라는 문자열을 이미 있던 문자열의 맨 뒤에 추가하세요.

그 이전 버전의 윈도우를 사용하시는 분들은, C:\\AUTOEXEC.BAT 를 열고 맨 뒷줄에 PATH=%PATH%;C:\\Python35 라고 마지막에 한 줄 추가한 뒤 시스템을 재시작하세요. 윈도우 NT의 경우, AUTOEXEC.NT 파일을 대신 편집하세요.

윈도우 비스타:

- 메뉴를 클릭하고, 을 클릭하세요.
- 을 클릭하면, 오른쪽에 " " 창이 보일 것입니다.
- 왼쪽에는 작업 항목 아래 여러 메뉴들이 있는데, 이 중 " " 항목이 보일 것입니다. 이것을 클릭하세요.
- 그러면 대화상자의 탭이 보이게 됩니다. 오른쪽 아래에 있는 버튼을 클릭하세요.
- 아래쪽의 " "라고 적혀 있는 목록에 있는 PATH 항목을 선택하고, 버튼을 클릭하세요.
- 경로를 수정하세요.
- 시스템을 재시작하세요. 윈도우 비스타는 컴퓨터가 재시작되기 전까지 새로 지정한 환경 변수가 적용되지 않습니다.

For Windows 7 and 8:

- 바탕 화면에 있는 를 오른쪽 클릭하고 을 클릭하거나, 또는 메뉴를 클릭하고 을 선택한 뒤 의 을 클릭하세요. 화면 왼쪽에 보이는 항목을 클릭한 뒤 탭을 클릭하세요. 아래쪽에 보이는 밑에 있는 여러 변수들 중 PATH라는 변수를 찾아 선택한 뒤, 버튼을 누르세요.
- 이미 있던 문자열의 맨 끝에 ;C:\\Python35 를 추가하세요 (이 폴더가 실제로 존재하는지 다시 한번 확인하세요. 더 최신 버전의 파이썬을 설치한 경우 폴더 이름이 다를 수 있습니다).
- 만약 원래 있던 문자열이 %SystemRoot%\\system32;였다고 한다면, 변경된 문자열은 %SystemRoot%\\system32;C:\\Python3 이어야 합니다.
- 버튼을 누르면 시스템을 재시작하지 않아도 변경 사항이 곧바로 적용됩니다만, 현재 실행중인 명령 프롬프트는 종료 후 다시 시작해주어야 합니다.

윈도우 10:

윈도우의 〉 〉 〉 에 들어가서, 오른쪽의 을 누릅니다. 대화상자의 아래쪽 를 누르고 〉 Path 를 선택한 뒤 을 누른 뒤 〉 버튼을 누르고 파이썬이 설치된 경로를 입력합니다. 예를 들어, C:\\Python35\\ 라고 하면 됩니다.

윈도우에서 파이썬 프롬프트 실행하기

PATH 환경변수가 제대로 설정되어 있다면, 파이썬 인터프리터를 명령 프롬프트 상에서도 실행하실 수 있습니다.

윈도우 환경에서 터미널 창을 열기 위해서는, 메뉴를 누르고 버튼을 클릭하세요. 나타나는 대화상자에 cmd 를 입력하시고 [enter] 키를 입력하세요.

이제, python을 입력하고 파이썬 프롬프트가 잘 실행되는지 확인하세요.

Mac OS X 에서의 설치

Mac OS X 사용자의 경우, Homebrew를 설치한 후 다음 명령을 통해 파이썬을 설치할 수 있습니다: brew install python3.

잘 설치되었는지 확인하시려면, 먼저 [Command + Space] 키를 입력하여 Spotlight 검색 창을 여세요. 그리고 Terminal 이라 입력하시고 [enter] 키를 누르세요. 이제, python3 을 입력하고 문제가 없는지 확인하세요.

GNU/Linux 에서의 설치

GNU/Linux 사용자의 경우, 여러분이 사용 중인 리눅스 배포판에서 제공하는 패키지 매니저를 통해 파이썬을 설치할 수 있습니다. 예를 들어 데비안/우분투에서는 다음과 같이 합니다: `sudo apt-get update && sudo apt-get install python3`.

잘 설치되었는지 확인하시려면, 앱을 열거나 혹은 `Alt + F2` 키를 입력한 뒤 `gnome-terminal` 을 입력하여 터미널을 실행하세요. 만약 이 두 방법으로 터미널을 실행시킬 수 없으면, 여러분이 설치한 리눅스 배포판의 설명서를 참조하세요. 이제, `python3` 을 입력하고 문제가 없는지 확인하세요.

다음과 같이 입력하면 시스템에 설치된 파이썬의 버전을 확인할 수 있습니다.

```
$ python3 -V
Python 3.6.0
```

NOTE: `$` 는 쉘의 프롬프트를 의미합니다. 이것은 여러분의 컴퓨터에 설치된 운영 체제의 설정에 따라 바뀔 수 있습니다만, 이 책에서는 `$` 로 통칭하도록 하겠습니다.

CAUTION: 컴퓨터에 설치된 파이썬 버전에 따라 결과가 조금씩 다르게 나타날 수도 있습니다.

요약

이제 여러분의 시스템에 파이썬이 올바르게 설치된 것으로 간주하도록 하겠습니다.

다음으로는 우리의 첫번째 파이썬 프로그램을 작성해 봅시다.

첫 걸음

이제 그 유명한 'Hello World' 프로그램을 파이썬으로 어떻게 실행하는지 배워보도록 하겠습니다. 이를 통해 파이썬 프로그램을 어떻게 작성하고, 저장하고, 실행하는지를 배우게 될 것입니다.

파이썬에서 프로그램을 실행하는 방법은 두 가지가 있습니다. 첫째는 대화형 인터프리터 프롬프트를 이용하는 방법과, 둘째는 소스 파일을 이용하는 것입니다. 지금부터 두 방법 모두 알아보도록 하겠습니다.

인터프리터 프롬프트를 활용하는 방법

여러분이 설치한 운영 체제에서 제공되는 터미널을 실행하세요(실행하는 방법은 설치 챕터를 참조하세요). 그리고 `python3` 이라 입력한 뒤 `[enter]` 키를 눌러 인터프리터 프롬프트를 엽니다.

파이썬 프롬프트가 시작되면 >>> 이라는 문자열이 보이는데 이것은 여러분이 원하는 파이썬 명령을 입력할 수 있는 상태임을 뜻합니다. 이것을 우리는 파이썬 인터프리터 프롬프트 라 부릅니다.

파이썬 인터프리터 프롬프트에서 다음을 입력하세요:

```
print("Hello World")
```

`[enter]` 키를 입력하면, `Hello World` 라는 문자열이 화면상에 출력됨을 확인하실 수 있을 것입니다.

다음은 여러분이 Mac OS X 컴퓨터를 이용하고 있을 경우 볼 수 있을 것으로 예상되는 예제 화면입니다. 화면에 보이는 파이썬 프로그램에 대한 세부 사항은 컴퓨터에 따라 조금씩 다를 수 있습니다만, 프롬프트로부터 보이는 부분 (>>> 이후로 보이는 부분)은 여러분이 어떤 운영 체제를 이용하든지 동일할 것입니다.

```
$ python3
Python 3.6.0 (default, Jan 12 2017, 11:26:36)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
```

파이썬은 여러분이 입력한 것에 대한 결과물을 바로바로 출력해준다는 점을 확인하세요! 방금 여러분이 입력한 것은 하나의 파이썬 구문 입니다. 우리는 방금 `print` 를 이용하여 여기에 넘겨준 값을 출력하도록 한 것입니다 (별로 놀랍지는 않지만요). 다시 말하면 `Hello world` 라는 문자열을 넘겨 주었고, 이 결과가 곧바로 화면에 출력된 것입니다.

인터프리터 프롬프트를 종료하는 방법

여러분이 현재 GNU/Linux 혹은 Unix 쉘을 이용하는 중이라면, `[ctrl + d]` 를 누르거나 `exit()` 를 입력하여 프롬프트를 종료할 수 있습니다 (주의: exit 뒤에 괄호 `()` 를 붙인다는 점을 잊지 마세요).

만약 여러분이 윈도우 명령 프롬프트를 이용하는 중이라면, `[ctrl + z]` 키를 누르고 `[enter]` 키를 입력하여 프롬프트를 종료할 수 있습니다.

편집기 선택하기

여러분이 파이썬 프로그램을 실행할 때마다 매번 인터프리터 프롬프트를 실행하고 프로그램을 입력할 수는 없는 노릇입니다. 프로그램을 미리 파일로 저장해 두면, 언제든지 실행만 하면 되니 더 편리하겠죠.

파이썬 소스 코드 파일을 만들기 위해서는, 우선 글자를 입력할 수 있고 저장할 수 있는 편집기 프로그램이 필요합니다. 좋은 프로그래머들이 사용하는 편집기에는 소스 파일을 쉽게 작성할 수 있도록 돕는 여러 기능이 갖추어져 있습니다. 따라서 좋은 편집기를 고르는 것은 정말로 중요한 일입니다. 편집기를 고르는 것은 마치 여러분이 어떤 자동차를 구입할지 고르는 과정과도 같습니다. 좋은 편집기는 파이썬 프로그램을 쉽게 작성할 수 있도록 도와 주고, 여러분이 앞으로 떠날 여정을 좀 더 편리하게, 원하는 목적지에 닿을 때까지 (목표를 달성할 때까지) 더 빠르고 안전한 길로 안내할 것입니다.

가장 기초적인 필수 기능은 문법 강조 기능입니다. 이 기능은 여러분이 작성한 파이썬 프로그램의 각 부분을 여러가지 다른 색깔로 표시해주어 여러분이 프로그램을 쉽게 _파악_하고 어떻게 실행되는지 알 수 있도록 돕습니다.

만약 어떤 편집기를 선택할 지 잘 모르겠으면, 저는 PyCharm Educational Edition 을 이용할 것을 추천합니다. 이 소프트웨어는 윈도우, Mac OS X, GNU/Linux에서 모두 사용이 가능합니다. 더 자세한 사항은 다음 섹션에서 다루겠습니다.

혹시 여러분이 윈도우 사용자라면, 절대로 메모장을 사용하지 마세요. 메모장은 문법 강조 기능을 지원하지 않을 뿐 아니라, 앞으로 중요하게 다루어지게 될 자동 들여쓰기 기능을 지원하지 않기 때문에 매우 안 좋은 선택이라 할 수 있습니다. 이런 기능들을 지원하는 소프트웨어를 사용해 주세요.

여러분이 숙련된 프로그래머라면 아마도 여러분은 Vim 또는 Emacs 에 이미 익숙할 것입니다. 말할 필요도 없이, 이 두 편집기는 현존하는 최고의 편집기들이며 파이썬 프로그램을 작성하는데에도 여러 장점이 있습니다. 저도 프로그램을 작성할 때 이 두 프로그램을 주로 사용하며, 심지어 책 한 권을 Vim으로 작성하기도 했습니다.

혹시 여러분이 Vim 또는 Emacs를 시간을 들여 배우고 싶으신 경우, 저는 둘 중 하나라도 그 사용법을 익혀 두기를 추천하며, 이것은 장기간에 걸쳐 큰 도움이 될 것입니다. 그러나, 초보자분들의 경우 지금 시점에서는 위에서 언급한 PyCharm 을 사용하시고, 편집기 사용법을 익히는데 시간을 투자하기 보다는 파이썬을 배우는데 초점을 맞추시는 편이 더 나을 수도 있을 것입니다.

다시 한번 말씀드리지만, 제대로 된 편집기를 이용하시기 바랍니다. 그러면 파이썬 프로그램을 작성하는 것이 더 재미있고 쉽게 느껴질 것입니다.

혹시 이 주제에 좀 더 관심이 있으시다면, 다음을 읽어 보세요: Finding the Perfect Python Code Editor

PyCharm

PyCharm Educational Edition 은 여러분이 파이썬으로 프로그램을 작성할 때 사용할 수 있는 무료 편집기입니다.

PyCharm을 처음 실행하면 아래와 같은 화면을 보게 될 것입니다. `Create New Project` 를 누르세요:

When you open PyCharm

다음으로 `Pure Python` 을 선택합니다:

PyCharm New Project

이제 프로젝트 경로의 `untitled` 를 `helloworld` 로 변경해 주면 다음과 같은 화면을 볼 수 있을 것입니다:

PyCharm project details

`Create` 버튼을 누릅니다.

다음으로 사이드바에 있는 `helloworld` 를 오른쪽 버튼으로 클릭하고, 메뉴에서 `New -〉 Python File` 을 선택합니다:

PyCharm -〉 New -〉 Python File

그러면 파일명을 무엇으로 할 것인지 지정하는 창이 나타납니다. 여기서는 `hello` 라고 입력해 줍니다:

PyCharm New File dialog box

다음과 같이 새 파일이 생성되고 화면에 나타납니다:

PyCharm hello.py file

이제 자동으로 미리 입력되어 있는 것들을 모두 삭제한 뒤, 다음을 입력합니다:

```
print("hello world")
```

입력이 끝났으면 소스 코드 창에서 오른쪽 버튼을 클릭하고, `Run 'hello'` 버튼을 누릅니다:

PyCharm Run 'hello'

그러면 다음과 같이, 여러분이 방금 작성한 프로그램이 출력한 내용을 확인할 수 있을 것입니다:

PyCharm output

여기까지입니다! 지금은 처음이기 때문에 몇가지 초기 설정을 더 해 주었지만, 다음부터는 왼쪽 사이드바에서 `helloworld` 를 오른쪽 클릭 -〉 `New -〉 Python File` 을 선택한 후 위와 동일한 방법을 따라서 새 프로그램을 입력한 후 실행하면 됩니다.

PyCharm에 대해 더 자세히 알고 싶으시다면, PyCharm Quickstart 페이지 (영문) 를 참조하세요.

Vim

1. Vim 설치
    - Mac OS X 사용자들은 HomeBrew 를 통해 `macvim` 패키지를 설치하세요. *Windows 사용자들은 Vim website 에서 "self-installing executable"을 내려받아 설치하세요.
    - GNU/Linux 사용자들은 각 배포판의 소프트웨어 저장소에서 Vim을 내려받아 설치하여야 합니다. 예를 들어 데비안 혹은 우분투의 경우 `vim` 패키지를 설치하세요.
2. 자동완성 기능을 위해 jedi-vim 플러그인을 설치하세요.
3. 파이썬에서도 `jedi` 패키지를 설치해 줘야 합니다: `pip install -U jedi`

Emacs

1. Emacs 24+ 설치
    - Mac OS X 사용자들은 http://emacsformacosx.com 에서 Emacs를 내려받아 설치하세요.
    - Windows 사용자들은 http://ftp.gnu.org/gnu/emacs/windows/ 에서 Emacs를 내려받아 설치하세요.
    - GNU/Linux 사용자들은 각 배포판의 소프트웨어 저장소에서 Emacs를 내려받아 설치하여야 합니다. 예를 들어 데비안 혹은 우분투의 경우 `emacs24` 패키지를 설치하세요.
2. ELPY 를 설치하세요.

소스 파일을 활용하는 방법

이제 파이썬 프로그래밍으로 돌아갑시다. 아마 여러분이 어떤 언어를 배우던지, 'Hello World' 라는 프로그램을 처음 작성하고 실행하게 될 것입니다. 이 프로그램이 하는 일은 실행했을 때 단순히 'Hello World' 라는 문자열을 화면에

출력하는 것이 전부인데요, Simon Cozens[1] 에 따르면 “Hello World 란 프로그래밍 신에게 이 언어를 잘 배울 수 있도록 도와 달라는, 일종의 주문 같은 것이다” 라 합니다.

여러분이 선택한 편집기를 실행하시고, 다음과 같이 프로그램을 작성한 뒤 `hello.py` 라는 이름으로 저장하세요.

만약 여러분이 PyCharm을 선택하셨다면, 앞에서 다룬 PyCharm으로 파이썬 소스 파일을 저장하고 실행하는 방법 을 참고하세요.

다른 에디터를 선택하셨다면, 새 파일을 열고 다음을 입력한 뒤 `hellp.py` 라는 이름으로 저장하세요:

```
print("hello world")
```

파일을 어디에 저장해야 할까요? 아무 곳에나 저장해도 됩니다. 아무 곳이 어디에다 저장하라는 것인지 잘 모르겠는 경우에는 아래와 같이 새 폴더를 만들고, 우리가 앞으로 작성할 모든 파이썬 프로그램을 저장해 두도록 합시다:

- `/tmp/py` (Mac OS 환경)
- `/tmp/py` (GNU/Linux 환경)
- `C:\py` (Windows 환경)

폴더를 만드는 방법은 터미널에서 `mkdir` 명령을 이용하면 됩니다. 예를 들어 Max OS X 혹은 GNU/Linux의 경우 `mkdir /tmp/py` 라고 입력하면 됩니다.

중요: 파일의 확장자명을 `.py`로 지정했는지 언제나 다시 한번 확인하세요. 예를 들어 `foo.py` 와 같은 형태가 되어야 합니다.

파이썬 프로그램 실행하기:

1. 터미널 창을 여세요 (설치 챕터에서 터미널 창을 여는 법을 다루고 있습니다).
2. cd (**C**hange **d**irectory의 약어) 명령으로 파일을 저장한 경로로 이동합니다. 예를 들어 Max OS X 혹은 GNU/Linux의 경우 `cd /tmp/py` 와 같이 입력하세요.
3. `python hello.py` 라 입력하여 프로그램을 실행합니다. 실행 결과는 아래와 같습니다.

```
$ python hello.py
hello world
```

Screenshot of running program in terminal

위와 같은 결과물을 얻으셨나요? 축하합니다! 여러분은 여러분의 첫 파이썬 프로그램을 성공적으로 실행시킨 것입니다. 방금 여러분은 프로그래밍을 배울 때 접하는 가장 어려운 부분을 지나온 것입니다. 그것은 바로 첫 프로그램을 성공적으로 실행시키는 것입니다!

만약 결과물 대신 오류 메시지가 출력되는 경우에는, 다시한번 프로그램을 정확하게 적혀진 그대로 입력했는 지 확인한 뒤 프로그램을 다시 실행해 보세요. 파이썬은 대/소문자를 구분합니다. 예를 들면 `print` 는 `Print` 와 같지 않습니다. 전자의 `p` 는 소문자이고, 후자의 `P` 는 대문자임에 유의하세요. 또한, 모든 줄의 첫번째 문자 앞에 공백이나 탭이 입력되어 있지는 않은지 확인하세요. 곧 왜 이것이 중요한지 다룰 것입니다.

동작 원리

파이썬 프로그램은 구문 들로 구성됩니다. 여러분이 작성한 첫 프로그램에서는, 단 한 가지의 구문만이 사용되었습니다. 여기서는 `print` 라고 하는 구문 을 불러 “Hello World”라는 문자열을 출력하도록 한 것입니다.

도움 받기

여러분이 파이썬이 제공하는 여러 함수나 명령들에 대한 정보를 얻고 싶으실 경우, 파이썬에 내장된 `help` 명령을 사용할 수 있습니다. 이 기능은 인터프리터 프롬프트를 이용할 때 특히 유용합니다. 예를 들면, `help('len')`라고 입력해 보세요. 그러면 항목의 개수를 세는 데 사용되는 `len` 함수에 대한 도움말이 화면에 표시됩니다.

TIP: `q` 를 눌러 도움말을 종료할 수 있습니다.

[1] 각 사람의 정보를 담는 클래스를 하나 만드세요. 그리고 각 사람의 이름을 키로 하고 사람의 객체를 값으로 하는 사전에 데이터를 저장해 둡니다. 그리고 pickle 모듈을 사용하여 이 객체를 여러분의 하드 디스크에 저장해 두세요. 또 사전의 내장 메소드를 활용하여 사람을 추가하고 삭제하거나 수정하는 기능을 구현하세요.

비슷한 방법으로, 파이썬 내에 있는 거의 모든 항목에 대한 정보를 얻을 수 있습니다. `help()` 를 이용하여 `help` 라는 명령 자체에 대한 설명도 확인해 보세요!

혹시 여러분이 `return` 과 같이 실행 가능한 명령에 대한 정보를 얻고 싶은 경우, 좌우에 따옴표를 붙여줄 필요가 있습니다. 즉, `help('return')` 과 같이 해 주어 우리가 무엇을 하려고 하는 것인지 명확하게 해 줍니다.

요약

이제 여러분은 쉽게 파이썬 프로그램을 작성하고, 저장하고, 실행할 수 있을 것입니다.

이제 여러분은 한 명의 파이썬 사용자입니다. 이제 파이썬의 다른 기능들에 대해서도 배워 봅시다.

기초

화면에 `hello world` 를 출력하는 것만으로는 부족하지요? 여러분은 아마 더 많은 것들을 해 보고 싶을 것입니다. 뭔가 정보를 입력받고, 처리한 뒤 결과물을 출력하는 프로그램을 만들고 싶으실 테지요. 파이썬에서는 상수들과 변수들을 이용하여 이러한 일을 할 수 있습니다. 이 장에서는 이외에도 몇 가지 기본 기능들에 대해서도 다뤄 볼 것입니다.

주석

주석 은 # 문자 뒤에 따라오는 짧은 문장입니다. 주로 소스 코드를 읽는 사람들을 위해 주석을 남기는 용도로 빈번하게 사용됩니다.

예제:

```
print('hello world') # Note that print is a function
```

또다른 예제:

```
# Note that print is a function
print('hello world')
```

여러분이 프로그램을 작성할 때, 주석을 가능한 많이 사용하시기 바랍니다. 특히 다음과 같은 상황에서 사용하면 좋습니다:

- 앞서 가정하고 넘어간 것들에 대한 설명
- 중요한 결정사항에 대한 설명
- 중요한 세부사항에 대한 설명
- 해결하고자 하는 문제에 대한 설명
- 앞으로 극복하려고 하는 문제들에 대한 설명 등등.

코드는 어떻게? 라는 물음에 답하지만, 주석은 왜? 라는 물음에 답해야 합니다 (Code tells you how, comments should tell you why).

주석은 여러분의 프로그램을 읽는 사람들에게 여러분이 작성한 프로그램이 무엇을 하는 프로그램인지 쉽게 파악할 수 있도록 도움을 주는 역할을 합니다. 프로그램을 작성하고 한 6개월쯤 뒤에는 여러분이 작성한 주석에 도움을 받는 사람이 여러분 자신이 될 수도 있다는 점을 꼭 기억하세요!

리터럴 상수

리터럴 상수는 `5`, `1.23` 과 같은 숫자나, `'This is a string'` 혹은 `"It's a string!"` 과 같은 문자열 등을 말합니다..

이것들이 리터럴 상수라고 불리우는 이유는 이것들이 프로그램 내에 직접 문자 형태로(literally) 지정되는 값들이기 때문입니다. 이러한 값들은 한번 지정되면 변하지 않습니다. 예를 들면 숫자 2 는 언제나 자기 자신이 2라는 숫자임을 나타내며 어떤 다른 의미도 갖지 않습니다. 이들은 한번 지정되면 그 값을 변경할 수 없기 때문에 상수 입니다. 그 중에서도 특별히 이러한 값들을 리터럴 상수라고 부릅니다.

숫자형

숫자형에는 정수형(Integer)과 부동 소수점 숫자형(Float)의 두 가지 종류가 있습니다.

정수형 숫자의 예는 `2` 입니다. 이것은 단순히 2 라는 숫자를 의미하는 것입니다.

부동 소수점 숫자의 예는 `3.23`, `52.3E-4`와 같은 값입니다. `E` 표기법은 E뒤의 값이 10의 지수임을 나타냅니다. 예를 들어 `52.3E-4` 는 `52.3 * 10-4` 라는 값을 의미합니다.

> **숙련된 프로그래머들을 위한 주석**
>
> 파이썬에서는 `long` 형이 따로 없습니다. 대신, `int` 형에 어떤 크기의 정수든지 담을 수 있습니다.

문자열

문자열이란 문자 의 나열 을 뜻합니다. 문자열은 간단하게 말하자면 문자들의 집합입니다.

여러분은 아마 앞으로 작성하게 될 거의 모든 파이썬 프로그램에서 문자열을 사용하게 될 것입니다. 따라서, 아래 항목들을 주의깊게 살펴보세요.

작은 따옴표

여러분은 작은 따옴표를 이용하여 문자열을 지정할 수 있습니다. 예를 들어 `'Quote me on this'` 와 같이 하면 됩니다.

모든 공백 문자, 즉 띄어쓰기나 탭 등은 입력한 그대로 유지됩니다.

큰 따옴표

큰 따옴표로 둘러싸인 문자열은 작은 따옴표로 둘러싸인 문자열과 완전히 동일하게 취급됩니다. 예를 들면, `"What's your name?"` 과 같습니다 (큰 따옴표로 둘러싸인 문자열 안에 작은 따옴표가 포함되어도 됩니다).

따옴표 세 개

여러 줄에 걸친 문자열은 세 개의 따옴표로 표현할 수 있습니다 – (`"""` 또는 `'''`). 세 개의 따옴표로 묶여진 문자열 안에서는 작은 따옴표든 큰 따옴표든 마음대로 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

문자열은 수정이 불가능함

여러분이 문자열을 한번 만들면, 그 문자열의 내용은 더이상 변경할 수 없습니다. 이것은 어떤 면에서는 좀 불편할 수 있다고 느낄 수 있겠습니다만, 사실은 그렇지 않습니다. 책의 뒷부분에서 여러 프로그램 예시를 통해 왜 이것이 큰 제약이 아닌지 살펴볼 것입니다.

> **C/C++ 프로그래머들을 위한 주석**
>
> 파이썬에서는 `char` 형이 따로 구분되어 있지 않습니다. 파이썬에서는 이것이 딱히 필요가 없습니다. 곧 여러분도 `char` 형을 찾지 않게 될 것입니다.
>
> **Perl/PHP 프로그래머들을 위한 주석**
>
> 파이썬에서는 작은 따옴표와 큰 따옴표로 둘러싸인 문자열을 동일하게 취급합니다. 둘 사이에 어떤 차이도 없습니다.

문자열 포맷팅

문자열을 생성하려고 할 때, 종종 다른 정보들을 포함하여 생성하고 싶을 때가 있습니다. 이것을 문자열 포맷팅이라고 하며, 이를 위해 `format()` 을 이용합니다.

다음을 `str_format.py` 라는 이름으로 저장하세요:

```
age = 20
name = 'Swaroop'

print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
```

실행 결과:

```
$ python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

동작 원리

먼저 중괄호로 표현된 특별한 표시들이 포함된 문자열을 만들고, 그 후에 문자열의 `format` 메소드를 사용하여 이 표시들을 `format` 메소드에 주어진 인자들로 치환한 것입니다.

위 예시에서는 문자열 내에서 첫번째로 `{0}` 이 사용되었으며 이것은 format 메소드에 주어진 첫번째 인자, 즉 변수 `name` 에 해당됩니다. 마찬가지로, 두번째 사용된 표시는 `{1}` 이며 이것은 format 메소드에 주어진 두번째 인자인 `age` 에 해당됩니다. 파이썬은 숫자를 셀 때 항상 0 부터 세기 시작한다는 점에 유의하세요. 즉, 첫번째 인자의 인덱스는 0 이며, 두번째는 1 입니다.

또한 다음과 같이 문자열 더하기를 이용하여 동일한 결과를 얻을 수도 있습니다:

```
name + ' is ' + str(age) + ' years old'
```

그러나 이것은 보기에도 깔끔하지 못하며, 작성 중 실수하기도 쉽습니다. 또 이 경우 각 변수를 일일이 명시적으로 문자열로 변환해주어야 하지만, `format` 메소드를 이용할 경우에는 알아서 자동으로 변환해 줍니다. 또 `format` 메소드를 이용할 경우 변수들을 신경쓰지 않고 문자열의 내용을 수정하기 쉽고, 문자열에 신경쓰지 않고도 변수의 위치나 순서 등을 변경하기가 더 쉽습니다.

이 때 중괄호 내에 주어진 숫자는 생략할 수 있습니다. 다음 예제를 확인하세요:

```
age = 20
name = 'Swaroop'

print('{} was {} years old when he wrote this book'.format(name, age))
print('Why is {} playing with that python?'.format(name))
```

위 프로그램 또한 동일한 결과를 출력합니다.

또한 인자에 이름을 지정해 줄 수도 있습니다:

```
age = 20
name = 'Swaroop'

print('{name} was {age} years old when he wrote this book'.format(name=name, age=age))
print('Why is {name} playing with that python?'.format(name=name))
```

위 프로그램 또한 동일한 결과를 출력할 것입니다.

파이썬 3.6 에서는 인자에 이름을 지정하여 문자열을 포맷하는 더 간단한 방법인 “f-string” 을 도입했습니다:

```
age = 20
name = 'Swaroop'
```

```
print(f'{name} was {age} years old when he wrote this book')  # notice the 'f' before the string
print(f'Why is {name} playing with that python?')  # notice the 'f' before the string
```

위 프로그램을 실행해도 역시 동일한 결과를 출력합니다.

파이썬의 format은 중괄호 표시의 위치에 주어진 인자들의 값을 치환해 넣습니다. 이때, 중괄호 표시에 다음과 같이 좀 더 상세히 세부사항을 지정할 수도 있습니다.

```
# decimal (.) precision of 3 for float '0.333'
print('{0:.3f}'.format(1.0/3))
# fill with underscores (_) with the text centered
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
# keyword-based 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

실행 결과:

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

지금까지 문자열 포맷팅에 대해 알아보았습니다. 여기서 print는 언제나 주어진 문자열의 끝에 "줄바꿈" 문자 (\n) 을 덧붙인다는 것 또한 기억하세요. 따라서 print를 호출할 때마다 인자로 주어진 내용들은 항상 그 다음 줄에 출력됩니다. 이것을 막기 위해서, print 함수에 (end) 문자가 무엇이 되어야 할 지 추가로 지정해 줄 수 있습니다.

```
print('a', end='')
print('b', end='')
```

실행 결과:

```
ab
```

또는 (end) 문자를 공백으로 할 수도 있습니다:

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

실행 결과:

```
a b c
```

이스케이프(Escape) 문자

여러분이 작은 따옴표(')를 포함하고 있는 문자열 하나를 정의하고 싶다고 해 봅시다. 이 경우 어떻게 이 문자열을 정의하면 될까요? 예를 들면 What's your name? 과 같은 문자열을 정의하는 것입니다. 물론 "What's your name?"이라고 하면 되겠지만, 'What's your name?'과 같은 방식을 사용할수는 없습니다. 왜냐면 이 경우 문자열의 시작과 끝이 어디부터 어디까지인지 모호해지기 때문이죠. 따라서 우리는 문자열 안에 포함된 작은 따옴표가 문자열의 끝을 의미하는 것이 아니라는 것을 파이썬에게 알려줘야 합니다. 이것은 이스케이프 문자 라 불리우는 것을 이용하면 해결할 수 있습니다. 사용법은 작은 따옴표 앞에 \ 문자(enter 키 위에 있습니다)를 붙여 \' 와 같은 방식으로 표기하면 됩니다. 이를 이용하면, 위의 문자열은 'What\'s your name?' 과 같이 표기할 수 있습니다.

위 문자열을 정의하는 또 다른 방법은 큰 따옴표를 사용하여 "What's your name?" 과 같이 표기하는 것입니다. 큰 따옴표로 지정된 문자열 안에 포함된 큰 따옴표도 마찬가지로 이스케이프 문자를 이용하여 표기할 수 있습니다. 또한, 여러분이 \ 문자를 표기하고 싶을 경우에는 \\ 라고 표기하면 됩니다.

한편 여러분이 두 줄짜리 문자열을 정의하고 싶을 경우 어떻게 하면 될까요? 한가지 방법은 위에서 다루었듯이 따옴표 세 개로 문자열을 정의하거나, 혹은 이스케이프 문자를 이용하여 줄바꿈 문자 (newline character) \n 을 사용하여 줄바꿈을 표현할 수 있습니다. 다음 예제를 확인하세요:

```
'This is the first line\\nThis is the second line'
```

또 한가지 유용한 이스케이프 문자는 \t로 표현되는 탭 문자입니다. 이외에도 여러가지 이스케이프 문자를 이용한 유용한 다른 표기들이 있습니다만, 일단은 가장 유용한 것 몇가지를 알려 드리는 것이니 알아두세요.

또 한가지 유용한 것은 문자열을 정의할 때 줄의 끝에 \ 문자를 붙여 주면, 그 다음 줄에 정의된 문자열을 끊김없이 이어 붙여 문자열을 정의하게 됩니다. 예를 들면 다음과 같습니다.

```
"This is the first sentence. \\
This is the second sentence."
```

위 예제는 다음 예제와 동일합니다.

```
"This is the first sentence. This is the second sentence."
```

Raw 문자열

문자열 내에 포함된 이스케이프 문자 등을 처리하지 않고 그대로 출력하고 싶을 때, 문자열 앞에 r 또는 R 문자를 붙여 Raw 문자열임을 표기합니다. 다음 예제를 확인하세요.

```
r"Newlines are indicated by \\n"
```

> **정규 표현식 사용자를 위한 주석**
>
> 정규 표현식을 사용할 때는 항상 raw 문자열을 사용하세요. 그렇지 않으면 문자열 내에 이스케이프 문자가 너무 많아져 알아볼 수 없게 될지도 모릅니다. 예를 들어 raw 문자열을 사용하면, '\\1' 라고 해야 할 것을 r'\1'로 표기가 가능합니다.

변수

리터럴 상수만 사용하여 프로그램을 작성할 수는 없습니다. 뭔가 정보를 담고, 수정할 수 있는 어떤 공간이 필요할 것입니다. 즉, 변수 를 이용하는 것이 좋습니다. 변수는 이름 그대로 변할 수 있는 공간을 말하며, 여기에는 무엇이든 저장할 수 있습니다. 변수들은 단순히 정보를 저장할 때 사용되는 컴퓨터의 기억 장치의 한 부분을 가져다가 적당한 이름을 붙여 사용하는 것입니다. 리터럴 상수와는 달리, 변수들은 프로그램 내에서 여러 방법을 통해 변경되고 사용되기 때문에 한눈에 알아보기 쉬운 이름을 지어 줍시다.

식별자 이름 짓기

변수 이름은 식별자의 한 예입니다. 식별자 란 무언가 를 식별하기 위해 주어진 그것의 이름을 말합니다. 식별자를 짓는데는 다음과 같은 규칙이 있습니다.

- 식별자의 첫 글자는 문자 (ASCII 대/소문자 혹은 유니코드 문자) 이거나 밑줄 (_) 이어야 합니다.
- 나머지는 문자 (ASCII 대/소문자 혹은 유니코드 문자), 밑줄 (_), 또는 숫자 (0-9)가 될 수 있습니다.
- 식별자는 대/소문자를 구분합니다. 예를 들어, myname 과 myName 은 다릅니다. 전자의 n은 소문자이고, 후자의 N은 대문자입니다.
- 올바른 식별자 이름은 i, name_2_3 등과 같습니다. 올바르지 않은 식별자 이름은 2things, this is spaced out, my-name, >a1b2_c3 등입니다.

자료형

변수는 여러 가지 자료형 의 값을 담을 수 있습니다. 가장 간단한 자료형의 예는 앞에서 이야기한 숫자형과 문자열입니다. 뒷장에서, 클래스를 활용하여 사용자 정의 자료형을 만드는 법 또한 배우게 될 것입니다.

객체

파이썬에서 사용되는 모든 것은 객체 입니다. 무엇이든 "어떤 것" 이라고 표현하는 대신, "어떤 객체" 라고 부릅니다.

> **객체 지향 프로그래머들을 위한 주석**:
>
> 파이썬은 강력한 객체 지향 언어로써 숫자, 문자, 함수 등등 모든 것을 객체로 취급합니다.

이제 리터럴 상수들과 함께 변수를 사용하는 방법을 알아보도록 하겠습니다. 다음 예제를 저장한 후 실행하여 봅시다.

파이썬 프로그램 작성하기

이제부터, 다음과 같이 파이썬 프로그램을 저장하고 실행해 보도록 합시다:

PyCharm을 이용할 경우

1. PyCharm 을 실행합니다.
2. 적당한 파일명을 입력하고, 새 파일을 만듭니다.
3. 예제로 주어진 프로그램 코드를 입력합니다.
4. 현재 파일을 오른쪽 클릭하고 실행(Run)을 누릅니다.

NOTE: 프로그램을 실행할 때 명령줄 인수와 함께 실행하고 싶을 경우, 메뉴의 `Run -〉 Edit Configurations` 을 누르고 `Script parameters:` 섹션에 인수를 입력해 준 뒤 `OK` 버튼을 누르면 됩니다:

PyCharm command line arguments

다른 편집기를 이용할 경우

1. 사용하고자 하는 편집기를 실행합니다.
2. 예제 프로그램을 입력합니다.
3. 적당한 파일 이름을 짓고 저장합니다.
4. 터미널에서 `python program.py` 과 같이 실행하여 파이썬 인터프리터를 통해 프로그램을 실행합니다.

예제: 변수와 리터럴 상수 사용하기

다음 프로그램을 입력 후 실행하세요:

```
# Filename : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''This is a multi-line string.
This is the second line.'''
print(s)
```

실행 결과:

```
5
6
This is a multi-line string.
This is the second line.
```

동작 원리

위 프로그램의 동작 원리는 다음과 같습니다. 먼저, 리터럴 상수 `5` 라는 값을 변수 `i` 에 할당 연산자 (=)를 이용하여 할당하였습니다. 이러한 한 줄을 명령이라고 부르는데, 이 경우 변수명 `i` 를 값 `5` 에 할당하는 행위를 하는 하나의 명령을 뜻합니다. 다음으로, `i` 에 할당된 값을 `print` 명령을 이용하여 출력합니다. 그러면 변수에 지정된 값이 화면에 나타납니다.

그리고 `i` 에 할당된 값에 `1` 을 더한 후 그 값을 다시 변수에 할당합니다. 이제 이 값을 출력하면, 예상대로, `6` 이라는 값이 출력됨을 알 수 있습니다.

리터럴 문자열 상수 또한 앞에서 설명한 과정과 동일한 과정을 거쳐 변수 `s` 에 저장된 후 화면에 출력됩니다.

정적 언어 프로그래머들을 위한 주석

파이썬에서는 변수에 값을 할당함으로써 자동으로 해당 변수가 생성되며 곧바로 사용할 수 있습니다. 따로 변수의 자료형을 지정할 필요도 없고, 미리 변수를 선언할 필요도 없습니다.

논리적/물리적 명령줄

물리적 명령줄이란 프로그램 코드 내에 직접 표현된 한 줄을 의미하는 반면, 논리적 명령줄은 파이썬 인터프리터 관점에서의 한 명령 단위를 의미합니다. 일반적으로 파이썬은 각각의 물리적 명령줄 이 곧 논리적 명령줄 일 것이라고 내부적으로 간주하고 프로그램을 실행합니다.

논리적 명령줄이란 예를 들면 `print('hello world')` 같은 것입니다. 만약 이것이 실제 코드 상으로도 한 줄로 표현되어 있다면 (편집기에서 보이는 그대로를 말합니다), 이 한 줄은 물리적 명령줄이라고도 말할 수 있을 것입니다.

일반적으로 파이썬으로 프로그래밍할 경우, 한 명령을 한 행에 적어 전체적인 코드를 파악하기 쉽게 작성하기를 권합니다.

만약 여러분이 한 물리적 명령줄에 둘 이상의 논리적 명령줄을 넣고 싶다면, 세미콜론 (;)을 이용하여 하나의 물리적 명령줄을 여러 개로 나누어 줄 수 있습니다. 다음 예제를 확인하세요.

```
i = 5
print(i)
```

위 예제는 다음 예제와 같습니다.

```
i = 5;
print(i);
```

이것은 다음 예제와도 같습니다.

```
i = 5; print(i);
```

또한 다음 예제와도 같습니다.

```
i = 5; print(i)
```

하지만, 저는 여러분이 한 물리적 명령행에 두개 이상의 논리적 명령행을 사용하지 말 것 을 강력히 권합니다. 즉, 세미콜론을 사용하지 말아 주세요. 사실, 저는 파이썬 프로그램을 작성할 때 세미콜론을 한번도 사용해 본 적이 없고, 또 다른 사람이 사용하는 것을 본 적도 없습니다.

한 명령줄이 너무 길어져서 보기가 불편한 경우, 백슬래시 문자(\)를 이용하여 한 논리적 명령줄을 여러 물리적 명령줄로 나눌 수 있습니다. 이를 명시적 행간 결합 이라 부릅니다:

```
s = 'This is a string. \\
This continues the string.'
print(s)
```

실행 결과:

```
This is a string. This continues the string.
```

다음과 같이 쓸 수도 있습니다.

```
i = \\
5
```

위 예제는 다음과 같습니다.

```
i = 5
```

가끔 백슬래시 없이 행간을 합칠 수 있는 경우도 있습니다. 이것은 명령줄의 중간에 괄호가 있을 때, 즉 대괄호나 중괄호를 열었을 경우 괄호를 닫을 때까지 백슬래시 없이도 모두 같은 명령줄로 간주됩니다. 이것은 비명시적 행간 결합이라고 부릅니다. 뒷장에서 리스트를 사용하여 프로그램을 작성할 때 이런 경우를 보게 될 것입니다.

들여쓰기

파이썬에서 공백은 중요한 역할을 합니다. 사실, 한 행의 앞에 붙어있는 공백이 정말 중요합니다. 이것을 들여쓰기 라 부릅니다. 한 논리적 명령행의 앞에 붙어있는 공백 (빈 칸 혹은 탭)은 논리적 명령행의 들여쓰기 단계를 의미하며, 이것은 한 명령의 범위를 구분하는 데 사용됩니다.

이것은 같은 들여쓰기 단계에 있는 명령들은 반드시 같은 들여쓰기를 사용해야 함을 의미합니다. 이러한 같은 들여쓰기를 사용하고 있는 명령들의 집합을 블록(block) 이라고 부릅니다. 뒷장에서 예제를 통해 블록에 대해 다루게 될 것입니다.

지금 여러분이 기억하셔야 할 것은 잘못된 들여쓰기는 오류를 일으킨다는 것입니다. 다음 예제를 봅시다.

```
i = 5
#           !                .
 print('Value is', i)
print('I repeat, the value is', i)
```

위 예제를 실행하면 다음과 같이 오류가 발생합니다.

```
  File "whitespace.py", line 3
    print('Value is', i)
    ^
IndentationError: unexpected indent
```

두번째 행 앞에 공백이 한칸 있다는 점을 확인하세요. 위와 같은 오류는 파이썬이 우리에게 프로그램의 문법이 잘못되었음을, 즉 프로그램이 뭔가 잘못 작성되었다는 것을 알려 주는 것입니다. 이 오류가 의미하는 것은 여러분이 마음대로 새 블록을 시작할 수 없음 을 의미합니다. 새 블록을 시작할 수 있는 경우에 대해 흐름 제어 챕터에서 다루게 될 것입니다.

> **들여쓰기 하는 법**
>
> 들여쓰기를 할 때에는 공백 4개를 이용하세요. 이것은 파이썬 언어에서 공식적으로 추천하는 방법입니다. 좋은 편집기들은 이 점을 자동으로 준수합니다. 또, 들여쓰기를 할 때에는 항상 같은 개수의 공백을 사용해야 한다는 점에 유의하시기 바랍니다.
>
> **정적 언어 프로그래머들을 위한 주석**
>
> 파이썬은 블록 구분을 위해 들여쓰기를 사용하며, 중괄호를 사용하지 않습니다. 파이썬에서 `from __future__ import braces` 명령을 실행하여 자세한 사항을 확인하세요.

요약

지금까지 파이썬의 여러 기본적인 특징에 대해 배워보았습니다. 이제 흐름 제어와 같이 좀 더 재미있는 부분에 대해 배워보도록 하겠습니다. 다음 챕터로 넘어가기 전, 이 챕터에서 배운 내용에 대해 미리 익숙해져 두기를 바랍니다.

연산자와 수식

여러분이 앞으로 작성하게 될 거의 모든 명령문 (논리적 명령줄)은 수식 을 포함하게 될 것입니다. 아주 간단한 수식의 한 예는 `2 + 3` 입니다. 수식은 연산자와 피연산자로 나누어집니다.

연산자 란 무언가를 계산할 때 쓰이는 특정 기능을 뜻하며, +와 같이 기호 또는 또는 특별한 키워드로 나타내어집니다. 또 연산자는 계산에 사용될 데이터를 필요로 하는데, 이들을 피연산자 라고 부릅니다. 이 경우, 피연산자는 2 와 3 이 됩니다.

연산자

이제 연산자의 사용법에 대해 알아보도록 하겠습니다.

파이썬 인터프리터 프롬프트 상에서도 수식을 계산할 수 있습니다. 다음과 같이 파이썬 인터프리터 프롬프트 상에서 `2 + 3`이라는 수식을 입력해 봅시다:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

이제 파이썬에서 사용 가능한 연산자들의 종류에 대해 간단히 알아봅시다:

- + (덧셈 연산자)
  - 두 객체를 더합니다.
  - `3 + 5` 는 8 을 반환합니다. `'a' + 'b'` 는 `'ab'` 를 반환합니다.
- - (뺄셈 연산자)
  - 한 숫자에서 다른 숫자를 뺍니다. 첫 번째 피연산자가 주어지지 않으면, 0으로 간주됩니다.
  - `-5.2` 는 음수를 표현합니다. `50 - 24` 는 26 을 반환합니다.
- * (곱셈 연산자)
  - 두 숫자의 곱 혹은 지정된 숫자만큼 반복된 문자열을 반환합니다.
  - `2 * 3` 은 6 을 반환합니다. `'la' * 3` 는 `'lalala'` 를 반환합니다.
- ** (거듭제곱 연산자)
  - x 의 y제곱을 반환합니다.
  - `3 ** 4` 는 81 을 반환합니다 (이 결과는 `3 * 3 * 3 * 3` 과 같습니다)
- / (나눗셈 연산자)
  - x를 y로 나눕니다.
  - `13 / 3` 은 `4.333333333333333` 을 반환합니다.
- // (나누기 및 내림 연산자)
  - x를 y로 나누고 그 결과를 가장 가까운 정수로 내림 합니다. 두 숫자 중 하나가 부동 소수점 숫자인 경우 결과 또한 부동 소수점 숫자입니다.
  - `13 // 3` 은 4 를 반환합니다.
  - `-13 // 3` 은 -5 를 반환합니다.
  - `9//1.81` 은 `4.0` 을 반환합니다.
- % (나머지 연산자)
  - x를 y로 나눈 나머지를 반환합니다.
  - `13 % 3` 은 1 을 반환합니다. `-25.5 % 2.25` 은 `1.5` 를 반환합니다.
- << (왼쪽 시프트 연산자)
  - 지정된 숫자에 대해 지정된 비트 수 만큼 왼쪽 시프트 연산합니다 (모든 숫자는 메모리상에서 0 또는 1 의 비트로 구성된 이진수로 표현됩니다).
  - `2 << 2` 는 8 을 반환합니다. 2 는 이진법으로 `10` 으로 표현됩니다.
  - 이를 왼쪽으로 두 번 시프트 연산하면 `1000` 이 되며, 이 결과는 십진법으로 표현하면 8 이 됩니다.
- >> (오른쪽 시프트 연산자)
  - 지정된 숫자에 대해 지정된 비트 수 만큼 오른쪽 시프트 연산합니다.
  - `11 >> 1` 은 5 를 반환합니다.
  - `11` 은 이진법으로 `1011` 로 표현되며, 오른쪽으로 한 번 시프트 연산하면 `101` 이 되고, 이는 십진법으로 표현하면 5 가 됩니다.
- & (비트 AND 연산자)
  - 비트 AND 연산값을 반환합니다: 두 비트가 모두 `1` 인 경우, 결과는 `1` 입니다. 아닌 경우, `0` 입니다.
  - `5 & 3` 은 1 을 반환합니다 (`0101 & 0011` 은 `0001` 이 됩니다)
- | (비트 OR 연산자)
  - 비트 OR 연산값을 반환합니다: 두 비트가 모두 `0` 인 경우, 결과는 `0` 입니다. 아닌 경우, `1` 입니다.
  - `5 | 3` 은 7 을 반환합니다 (`0101 | 0011` 은 `0111` 이 됩니다)
- ^ (비트 XOR 연산자)
  - 비트 XOR 연산값을 반환합니다: 두 비트 (`1` `0`) 가 서로 같은 경우, 결과는 `0` 입니다. 아닌 경우, it's `1` 입니다.
  - `5 ^ 3` gives 6 (`0101 ^ 0011` 은 `0110` 이 됩니다)
- ~ (비트 반전 연산자)
  - 숫자 x의 비트 반전 연산값, 즉 -(x+1) 을 반환합니다.
  - `~5` 는 -6 을 반환합니다. 자세한 사항은 http://stackoverflow.com/a/11810203 을 참고하세요.

- < (작음 연산자)
  - x가 y보다 작은지 여부를 반환합니다. 모든 비교 연산자는 `True ( )` 또는 `False ( )`을 반환합니다. 각 반환값의 첫글자는 대문자라는 점에 유의하세요.
  - `5 < 3` 는 `False` 를 반환합니다. `3 < 5` 는 `True` 를 반환합니다.
  - 다음과 같이 여러 숫자에 대해 한꺼번에 비교 연산을 수행할 수 있습니다. `3 < 5 < 7` 은 `True` 를 반환합니다.
- > (큼 연산자)
  - x 가 y 보다 큰지 여부를 반환합니다.
  - `5 > 3` 은 `True` 를 반환합니다. 만약 두 피연산자가 모두 숫자라면, 같은 숫자형으로 변환된 후 크기가 비교됩니다. 피연산자가 숫자형이 아닐 경우, 항상 False 를 반환합니다.
- <= (작거나 같음 연산자)
  - x 가 y 보다 작거나 같은지 여부를 반환합니다.
  - `x = 3; y = 6; x <= y` 는 `True` 를 반환합니다.
- >= (크거나 같음 연산자)
  - x 가 y 보다 크거나 같은지 여부를 반환합니다.
  - `x = 4; y = 3; x >= 3` 은 `True` 를 반환합니다.
- == (같음 연산자)
  - 두 객체가 같은지 여부를 반환합니다.
  - `x = 2; y = 2; x == y` 는 `True` 를 반환합니다.
  - `x = 'str'; y = 'stR'; x == y` 는 `False` 를 반환합니다.
  - `x = 'str'; y = 'str'; x == y` 는 `True` 를 반환합니다.
- != (같지 않음 연산자)
  - 두 객체가 같지 않은지 여부를 반환합니다.
  - `x = 2; y = 3; x != y` 는 `True` 를 반환합니다.
- `not` (불리언 NOT 연산자)
  - `x` 가 `True` 라면, `False` 를 반환합니다. `x` 가 `False` 라면, `True` 를 반환합니다.
  - `x = True; not x` 는 `False` 를 반환합니다.
- `and` (불리언 AND 연산자)
  - `x and y` 를 계산할 경우, x 가 `False` 이면 `False`를 반환하며 아닌 경우 y 가 참인지 거짓인지를 판별하여 반환합니다.
  - `x = False; y = True; x and y` 를 계산할 경우, x 가 거짓이므로 y값과 관계없이 `False` 가 반환됩니다. 이 경우, 이미 'and' 연산의 왼쪽 값이 거짓이므로 전체 식의 결과가 거짓임이 자명하므로, 오른쪽에 해당하는 y 가 참인지 거짓인지를 판별할 필요가 없고, 따라서 y 는 연산되지 않습니다. 이것을 단축 계산(short-circuit evalulation)이라고 부릅니다.
- `or` (불리언 OR 연산자)
  - x 가 `True` 이면 True 가 반환되며, `False` 이면 y 가 참인지 거짓인지를 판별하여 반환합니다.
  - `x = True; y = False; x or y` 를 계산할 경우 `True` 가 반환됩니다. 여기서도 위와 같이 단축 계산이 적용됩니다.

연산 및 할당 연산자

아래 예제와 같이, 변수의 값에 어떤 연산을 한 뒤 다시 그 변수에 연산값을 할당하는 경우가 자주 발생합니다:

```
a = 2
a = a * 3
```

이런 경우, 아래와 같이 연산과 할당을 한번에 줄여 쓸 수 있습니다:

```
a = 2
a *= 3
```

즉 `( ) = ( ) ( ) ( )` 이 `( ) ( )= ( )` 의 형태가 됩니다.

연산 순서

`2 + 3 * 4`와 같은 수식을 계산한다고 합시다. 덧셈이 먼저일까요, 곱셈이 먼저일까요? 초중고등학교 시절에 이미 배우셨겠지만, 곱셈을 먼저 계산해야 합니다. 이것은 곱셈 연산이 덧셈 연산보다 연산 순서에서 우위에 있기 때문입니다.

아래 표는 파이썬에서의 연산 순서를 나타내고 있습니다. 맨 위부터 가장 늦게 계산되는 순서대로 나열한 것입니다. 이것은 특정한 수식이 주어졌을 때, 파이썬은 이 표의 가장 아래에 위치한 연산부터 차례대로 계산하게 된다는 것을 의미합니다.

아래 표는 파이썬 레퍼런스 매뉴얼 에서 가져온 것입니다. 연산 순서를 적절히 조절하기 위해서는 괄호를 적당한 위치에 사용하는 것이 좋습니다. 또, 적절한 괄호의 사용은 프로그램을 좀 더 읽기 쉽게 해 줍니다. 아래의 연산 순서 변경 챕터을 통해 이에 대해 좀 더 자세히 알아보시기 바랍니다.

- `lambda` : 람다 식
- `if - else` : 조건 식
- `or` : 불리언 OR
- `and` : 불리언 AND
- `not x` : 불리언 NOT
- `in, not in, is, is not, <, <=, >, >=, !=, ==` : 비교 연산, 요소 연산, 관계 연산
- `|` : 비트 OR
- `^` : 비트 XOR
- `&` : 비트 AND
- `<<, >>` : 시프트 연산
- `+, -` : 덧셈 및 뺄셈
- `*, /, //, %` : 곱셈, 나눗셈, 나누기 및 내림 연산, 나머지 연산
- `+x, -x, ~x` : 양수 표현, 음수 표현, 비트 NOT
- `**` : 거듭제곱
- `x[index], x[index:index], x(arguments...), x.attribute` : 원소 접근, 슬라이스, 함수 호출, 속성 참조
- `(expressions...), [expressions...], {key: value...}, {expressions...}` : 괄호 또는 튜플, 리스트, 사전, 집합

아직 이 연산자들에 대해 모두 다루지 않았지만, 곧 다루게 될 것입니다.

같은 연산 순서 를 갖는 연산자들은 위 표에서 같은 행에 위치하고 있습니다. 예를 들어, + 연산자와 - 연산자는 같은 연산 순서를 가지고 있습니다.

연산 순서 변경

괄호를 사용하여 수식을 좀 더 읽기 쉽게 할 수 있습니다. 예를 들어, `2 + (3 * 4)` 라고 쓰면 `2 + 3 * 4` 로 쓰는 것에 비해 연산자 순서를 잘 모르는 사람조차도 쉽게 읽을 수 있을 것입니다. 그렇지만, 괄호를 적당히 사용하는 것도 중요합니다. `(2 + (3 * 4))` 와 같이 괄호를 너무 많이 사용하는 것은 피하세요.

또 괄호를 사용하면 연산의 순서를 바꿀 수 있습니다. 예를 들어 위 수식에서 덧셈을 곱셈보다 먼저 계산하고 싶을 경우 `(2 + 3) * 4` 라고 적을 수 있습니다.

같은 연산 순서를 가질 경우

기본적으로 연산자는 왼쪽에서 오른쪽으로 차례대로 계산됩니다. 즉, 같은 연산 순서를 가진 연산자들의 경우 왼쪽에서 오른쪽으로 순서대로 계산됨을 의미합니다. 예를 들어, `2 + 3 + 4` 는 `(2 + 3) + 4` 와 같이 계산됩니다. 다만, 할당 연산자와 같은 몇몇 특별한 연산자들은 오른쪽에서 왼쪽으로 계산됩니다. 예를 들어 a = b = c 는 a = (b = c) 와 같이 계산됩니다.

수식 예제

예제 (`expression.py` 로 저장하세요):

```
length = 5
breadth = 2

area = length * breadth
print('Area is', area)
print('Perimeter is', 2 * (length + breadth))
```

출력 결과:

```
$ python expression.py
Area is 10
Perimeter is 14
```

동작 원리

먼저 직사각형의 높이와 너비가 각각 length 와 breadth 라는 변수에 저장됩니다. 이제 앞서 배운 수식을 이용하여, 두 변수를 이용해 직사각형의 면적과 둘레를 계산합니다. 첫번째로 `area` 라는 변수에 `length * breadth` 라는 수식의 결과가 저장되며, 이 값은 `print` 함수에 의해 화면에 출력됩니다. 두번째로는 print 함수에 곧바로 `2 * (length + breadth)` 라는 수식을 직접 입력하고, 그 결과를 화면에 출력합니다.

파이썬이 얼마나 예쁘게 결과를 보여주는지 확인하세요. 우리가 `'Area is'` 라는 문자열이나 `area` 라는 변수에 공백을 지정하지 않았음에도, 파이썬이 그 둘 사이에 자동으로 공백을 넣어 줌으로 결과물이 깔끔하고 멋지게 출력될 수 있도록 해 주기 때문에 좀 더 읽기 쉬운 프로그램을 작성할 수 있습니다. 이것은 파이썬이 프로그래머의 삶을 좀 더 쉽게 만들어주는 하나의 좋은 예라고도 할 수 있겠습니다.

요약

지금까지 연산자, 피연산자, 수식에 대해 알아보았습니다. 이들은 여러분이 앞으로 작성할 프로그램의 기본적인 골격이 되어 줄 것입니다. 다음으로는, 여러분의 프로그램에서 명령문을 이용하여 지금까지 배운 것들을 응용하는 방법에 대해 알아보겠습니다.

흐름 제어

지금까지 우리가 본 파이썬 프로그램들은 전부 맨 윗줄부터 차례대로 실행되기만 하는 것들 뿐이었습니다. 이러한 실행 흐름을 바꿀 수 있다면 어떨까요? 예를 들어, 프로그램이 현재 시간에 따라 'Good Morning' 혹은 'Good Evening'을 출력하는 결정을 내리도록 할 수 있게 하면 좋지 않을까요?

예상하시는대로 지금부터 배울 흐름 제어문들을 활용하면 이러한 프로그램을 제작할 수 있습니다. 파이썬에서는 `if`, `for`, `while` 이라는 세 종류의 흐름 제어문을 사용할 수 있습니다.

`if` 문

`if` 문은 조건을 판별할 때 사용됩니다. if (만약) 조건이 참이라면, if 블록 의 명령문을 실행하며 else (아니면) else 블록 의 명령문을 실행합니다. 이 때 else 블록은 생략이 가능합니다.

예제 (`if.py` 로 저장하세요):

실행 결과:

동작 원리

이 프로그램에서는, 사용자로부터 숫자를 입력받아 그 숫자가 프로그램에 지정된 숫자와 같은지 확인합니다. 먼저 `number` 변수에 원하는 숫자를 넣습니다. 여기서는 23 입니다. 그리고, `input()` 함수를 통해 사용자로부터 입력을 받습니다. 여기서 함수란 재사용 가능한 프로그램 조각을 의미합니다. 다음 장에서, 함수에 대해 좀 더 자세히 배울 것입니다.

파이썬에 내장된 `input` 함수에 문자열을 넣어 주면 화면에 이 문자열이 출력되며, 또 사용자의 입력을 기다리게 됩니다. 이제 사용자가 무엇인가를 입력하고 kbd:[enter] 키를 누르면, `input()` 함수는 사용자가 입력한 것을 문자열의 형태로 반환해 줍니다. 이제 `int` 를 이용하여 이것을 정수형으로 변환한 뒤, 그 값을 변수 `guess`에 대입합니다. 사실 여기에서 사용된 `int`는 클래스라는 것입니다만, 일단 여기서는 이것이 문자열을 숫자형으로 변환해 준다는 것만 기억하셔도 됩니다 (다만 이 때 사용된 문자열은 올바른 숫자를 포함하고 있어야 합니다).

다음으로, 사용자가 입력한 숫자와 우리가 고른 숫자를 비교합니다. 만약 이 두 숫자가 같으면, 성공했다는 메시지를 화면에 출력합니다. 이때 들여쓰기를 이용하여 어디부터 어디까지가 이 블록에 해당하는지를 표시했다는 것을 확인하세요. 이러한 이유로 파이썬에서 들여쓰기는 굉장히 중요합니다. 앞서 말씀드렸듯이 여러분이 "일관성 있게 들여쓰는" 습관에 익숙해져 있었으면 좋겠네요. 이미 그렇게 하고 계시지요?

또한 if 문의 뒷부분에 콜론(:)이 붙어 있는 것을 확인하세요. 콜론은 그 다음 줄부터 새로운 블록이 시작된다는 것을 의미합니다.

이제, 사용자가 입력한 숫자가 우리가 고른 숫자에 비해 작다면, 사용자에게 좀 더 큰 숫자를 입력하라고 알려 줍니다. 여기에 사용된 것은 elif 절인데, 이것은 두 개의 if 문을 중첩해서 사용해야 할 경우 (즉 if else를 쓰고 밑에 또 다시 if else를 써야 될 경우) 이것을 if-elif-else로 한번에 줄여서 쓸 수 있게 해 주는 것입니다. 즉 elif를 쓰면 프로그래밍을 좀 더 짧게 만들 수 있고 더 많은 들여쓰기를 해야 하는 수고도 줄여 줍니다.

elif 와 else 문을 사용할 경우, 논리적 명령줄의 마지막에는 항상 콜론이 붙어 있어야 하며 그 다음 줄에는 다른 들여쓰기 단계로 시작되는 새로운 명령문 블록이 시작되어야 합니다.

또한 if문의 if 블록 안에 또다른 if문을 넣고, 또 넣고 하는 식으로 작성할 수도 있습니다. 이 경우 이것을 중첩 if 문이라 부릅니다.

elif 와 else 절은 생략이 가능합니다. 최소한의 올바른 if 문 사용법은 다음과 같습니다:

```
if True:
    print('Yes, it is true')
```

if - elif - else 문의 실행이 끝나면, 파이썬은 if 문을 담고 있는 블록의 다음 줄부터 실행을 재개합니다. 위 예제의 경우 그 블록은 최상위 블록 (프로그램이 실행된 시점의 블록)이 되며, 따라서 그 다음에 실행될 명령문은 print('Done') 이 됩니다. 그 이후는 프로그램의 끝이고 실행이 종료되게 됩니다.

지금 여러분이 본 것은 굉장히 간단한 프로그램이지만, 이를 통해서도 충분히 많은 것들에 대해 이야기했습니다. 하지만 이 모든 내용은 상당히 직관적입니다 (만약 여러분이 C/C++ 경험이 있다면 훨씬 더 쉽다고 느껴지기까지 할 것입니다). 지금 당장은 여러분이 이 모든 내용을 익혀야 하겠지만, 몇번 연습을 해보고 나면 아마 좀 더 편하게 받아들여질 것이며 곧 '자연스럽게' 여기게 될 것입니다.

> **C/C++ 프로그래머를 위한 주석**
>
> 파이썬에는 switch 문이 없습니다. 그 대신 if..elif..else 문을 이용하여야 합니다 (몇몇 상황에서는 사전을 이용하는 것이 더 편리합니다).

while 문

while 문은 특정 조건이 참일 경우 계속해서 블록의 명령문들을 반복하여 실행할 수 있도록 합니다. while 문은 반복문의 한 예입니다. 또한 while 문에도 else 절이 따라올 수 있습니다.

예제 (while.py로 저장하세요):

실행 결과:

동작 원리

이 프로그램 또한 숫자 알아맞히기 게임입니다만, 더 나은 점은 사용자가 답을 맞출 때까지 계속 숫자를 입력할 수 있다는 것입니다. 즉, 이전 섹션에서 작성한 프로그램처럼 다른 숫자를 입력해 보기 위해 프로그램을 또 실행시킬 필요가 없습니다. 이 예제는 while문의 사용법을 잘 보여줍니다.

이 예제에서는 input 과 if 문들을 while 안으로 옮겼고, 반복절이 실행되기 전 변수 running을 (True)으로 설정했습니다. 이제 running 변수가 True로 설정되어 있으므로, while 문에 딸려 있는 while 블록이 실행됩니다. 이 블록의 실행이 끝나면, while 문은 변수 running 의 값을 다시 한번 확인합니다. 이 때 이 값이 True 인 경우 while 블록을 다시 한번 실행하며, 거짓인 경우 else 블록을 실행한 뒤 블록을 빠져나오게 되고 그 아래의 명령문이 계속 실행되게 됩니다.

else 블록은 while 반복절의 조건이 (False) 인 경우 실행됩니다. 물론 이 조건을 처음으로 확인했을 경우에도 이 블록이 실행될 수 있습니다. while 반복절에 else 절이 딸려있는 경우, break 명령으로 루프를 강제로 빠져나오지 않는 이상 이 블록은 항상 실행되게 됩니다.

여기서 (True) 과 (False) 을 담는 자료형을 (Boolean)형이라고 하며, 각각은 숫자로 변환하면 1 과 0 으로 간주됩니다.

C/C++ 프로그래머를 위한 주석

while 루프에 else 절이 사용될 수 있음을 기억하세요.

for 문

for..in 문은 객체의 열거형(Sequence)을 따라서 반복하여 실행할 때 사용되는 파이썬에 내장된 또 하나의 반복문으로, 열거형에 포함된 각 항목을 하나씩 거쳐가며 실행합니다. 열거형](./data_structures.md#sequence)에 대해서는 이후에 좀 더 자세히 다룰 것입니다. 일단 여기서는, 열거형이란 여러 항목이 나열된 어떤 목록을 의미한다고 생각하시기 바랍니다.

예제 (for.py로 저장하세요):

실행 결과:

동작 원리

이 프로그램은 화면상에 숫자의 나열을 출력합니다. 파이썬에 내장된 range 함수를 통해 이러한 숫자의 나열을 생성합니다.

여기서는 range 함수에 두 개의 숫자를 넣어 주었으며, 그러면 이 함수는 첫 번째 숫자 이상, 그리고 두 번째 숫자 미만까지의 숫자 목록(리스트)을 반환합니다. 예를 들어, range(1,5) 는 리스트 [1, 2, 3, 4] 를 반환합니다. 기본적으로, range 는 1씩 증가하는 숫자의 리스트를 반환합니다. 그러나 range 에 세 번째 숫자를 입력하면, 이 세 번째 숫자만큼씩 증가하는 숫자들의 리스트를 얻을 수 있습니다. 예를 들어, range(1,5,2) 는 [1,3] 을 반환합니다. 반환되는 리스트는 두번째 숫자 미만 까지 반환되며, 이하 까지 반환되는 것이 아니라는 점을 꼭 기억하세요.

range 함수는 호출될 때 해당 범위 내의 숫자를 모두 한번에 생성하는 것이 아니라 한번에 하나씩만 생성하여 반환해 주기 때문에, 모든 생성될 숫자의 목록을 얻고 싶은 경우 range()의 결과를 list() 에 넣어주면 됩니다. 예를 들어, list(range(5)) 와 같이 하면 [0, 1, 2, 3, 4] 가 반환될 것입니다. 리스트에 대해서는 자료구조 챕터에서 좀 더 자세히 설명할 것입니다.

위 예제의 for 루프는 range 함수에서 반환된 리스트를 따라 반복하여 실행됩니다. 즉, for i in range(1,5) 는 for i in [1, 2, 3, 4] 와 같습니다. 이것은 리스트에 들어 있는 각각의 숫자 (각 숫자는 곧 객체이기도 합니다) 를 한번에 하나씩 i 에 대입하고, 이렇게 대입된 각 i 값을 이용하여 for 에 딸린 블록을 실행합니다. 이 경우, for 블록에서 하는 일은 단순히 i 값을 화면에 출력해 주는 것입니다.

또한, 원하는 경우 추가로 else 절을 포함시켜 줄 수 있습니다. 이것이 포함되면, break 문으로 루프를 강제로 빠져나오지 않는 한 for 반복절을 다 마치고 난 뒤에는 이 절이 항상 실행되게 됩니다.

for..in 반복문은 어떤 종류의 열거형 자료형과도 함께 사용될 수 있습니다. 여기서는 range 라는 내장 함수를 통해 숫자 리스트를 생성하여 사용하였습니다만, 일반적으로는 아무 종류의 객체를 담고 있는 아무 열거형이나 사용이 가능합니다! 이 책의 후반부에서 이에 대해 좀 더 자세히 다뤄 보겠습니다.

C/C++/Java/C# 프로그래머를 위한 주석

파이썬의 for 루프는 C/C++ 에서 제공하는 for 루프와는 근본적으로 다릅니다. C# 프로그래머분들은 파이썬의 for 문이 C# 의 foreach 루프와 비슷하며, 자바 프로그래머분들은 이것이 Java 1.5의 for (int i : IntArray)와 비슷하다는 점을 확인하실 수 있을 것입니다.

C/C 처럼 for (int i = 0; i < 5; i) 와 같이 사용하고 싶은 경우, 파이썬에서는 단순히 for i in range(0,5)라고 입력하기만 하면 됩니다. 보시다시피, 파이썬의 for 루프는 더 단순하며 명확하고, 따라서 오류가 발생하기도 어렵습니다.

break 문

break 문은 루프 문을 탈출할 때 (break out), 즉 아직 루프 조건이 False 가 되지 않았거나 열거형의 끝까지 루프가 도달하지 않았을 경우에 루프 문의 실행을 강제로 정지시키고 싶을 때 사용됩니다.

중요한 점은 만약 여러분이 break 문을 써서 for 루프나 while 루프를 빠져나왔을 경우, 루프에 딸린 else 블록은 실행되지 **않습니다**.

예제 (break.py로 저장하세요):

실행 결과:

동작 원리

이 프로그램에서는 사용자의 입력을 반복해서 받고, 매 반복시 입력받은 문자열의 길이를 출력합니다. 다만 사용자가 입력한 문자열이 'quit' 일 경우, break문으로 루프를 빠져나와 프로그램을 끝내도록 조건을 넣어 주었습니다.

입력받은 문자열의 길이는 내장함수 len 을 이용하여 계산할 수 있습니다.

break 문은 for 루프 내에서도 이용될 수 있음을 기억하세요.

Swaroop의 파이썬 시

예제에서 입력한 것은 이 책의 저자가 작성한 작은 시입니다:

```
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
```

continue 문

continue 문은 현재 실행중인 루프 블록의 나머지 명령문들을 실행하지 않고 곧바로 다음 루프로 넘어가도록 (continue) 합니다.

예제 (continue.py 로 저장하세요):

실행 결과:

동작 원리

이 프로그램에서는 사용자로부터 입력을 받습니다만, 입력받은 문자열의 길이가 적어도 3 이상인 경우에만 문자열을 처리합니다. 즉, 내장함수 len 을 통해 입력받은 문자열의 길이를 알아낸 후 그 길이가 3보다 작으면, continue 문을 이용하여 그 이하의 명령문을 실행하지 않고 다음 루프로 넘어가도록 합니다. 입력받은 문자열의 길이가 3 이상일 경우에만 그 이하의 명령문이 실행되고, 지정된 작업이 실행됩니다.

continue 문은 for 루프 내에서도 이용될 수 있음을 기억하세요.

요약

이 챕터에서는 if, while, for 세 종류의 흐름 제어문에 대해 배워 보았습니다. 또한 그와 같이 이용할 수 있는 break 과 continue 문에 대해서도 배웠습니다. 이 명령문들은 파이썬에서 가장 많이 사용되는 명령문 중 일부라고 볼 수 있으며, 따라서 이 명령문들에 필수로 친숙해져야 할 필요가 있습니다.

다음으로, 함수를 만들고 사용하는 방법에 대해 배워 보겠습니다.

함수

함수는 재사용 가능한 프로그램의 조각을 말합니다. 이것은 특정 블록의 명령어 덩어리를 묶어 이름을 짓고, 그 이름을 프로그램 어디에서건 사용함으로써 그 블록에 포함된 명령어들을 몇번이고 다시 실행할 수 있게 하는 것입니다. 이를 보고 함수를 호출한다고 합니다. 사실 우리는 이미 앞에서 len 이나 range 와 같은 많은 내장 함수들을 사용해 왔습니다.

이러한 함수들은 프로그램을 작성할 때 가장 중요한 단위가 될 것입니다 (아마 어떤 프로그래밍 언어에서라도 그럴 것입니다). 따라서 이 챕터에서는 함수라는 것을 다양한 관점에서 살펴보도록 하겠습니다.

함수는 def 키워드를 통해 정의됩니다. def 뒤에는 함수의 식별자 이름을 입력하고, 괄호로 감싸여진 함수에서 사용될 인자(arguments)의 목록을 입력하며 마지막으로 콜론을 입력하면 함수의 정의가 끝납니다. 새로운 블록이 시작되는 다음 줄부터는 이 함수에서 사용될 명령어들을 입력해 줍니다. 복잡해 보이지만, 아래 예제를 통해 함수를 정의하는 것이 얼마나 간단한지 알아봅시다:

예제 (function1.py 로 저장합니다):

실행 결과:

동작 원리

여기에서는 위에서 설명한 문법을 이용하여 say_hello 라는 함수를 정의하였습니다. 이 함수는 어떤 인자도 넘겨받지 않으므로, 괄호 내에 매개 변수를 정의하지 않습니다. 함수의 인수란 함수로 넘겨지는 입력값들을 말하며, 함수는 이 값을 처리하여 결과를 넘겨줍니다.

함수를 두 번 호출하는 것은 같은 코드를 두 번 작성하는 것과 같은 효과를 가진다는 것을 알아두세요.

함수의 매개 변수

함수를 정의할 때 매개 변수를 지정할 수 있습니다. 매개 변수란 함수로 넘겨지는 값들의 이름을 말하며, 함수는 이 값들을 이용해서 뭔가를 할 수 있습니다. 매개 변수는 변수와 거의 비슷한 개념으로 취급되지만, 매개 변수의 값들은 함수가 호출되어질때 넘겨받은 값들로 채워지며 함수가 실행되는 시점에서 이미 할당이 완료되어 있다는 점이 다릅니다.

매개 변수는 함수를 정의할 때 괄호 안에 쉼표로 구분하여 지정합니다. 함수를 호출할 때에는, 동일한 방법으로 함수에 값을 넘겨 줍니다. 이 때 함수를 정의할 때 주어진 이름을 매개 변수 라 부르고, 함수에 넘겨준 값들을 인자 라 부릅니다.

예제 (function_param.py 로 저장하세요):

실행 결과:

동작 원리

여기서는 두 매개 변수 a 와 b 를 받는 print_max 라는 함수를 정의합니다. 그리고 간단한 if..else 문을 이용하여 크기를 비교하고 둘 중에 더 큰 값을 출력합니다.

print_max 함수를 처음 호출할 때에는 값을 직접 인자로 입력하여 넘겨주었습니다. 반면 두 번째 호출시에는 변수를 인자로 입력하여 주었습니다. 이것은 print_max(x, y) 는 변수 x 에 지정된 값을 변수 a 에 입력해 주고 변수 y 의 값을 변수 b 에 입력해 주는 것을 의미합니다. 따라서 두 경우 모두 print_max 함수는 동일한 결과를 줍니다.

지역 변수

여러분이 정의한 함수 안에서 변수를 선언하고 사용할 경우, 함수 밖에 있는 같은 이름의 변수들과 함수 안에 있는 변수들과는 서로 연관이 없습니다. 이러한 변수들을 함수의 지역 (local) 변수라고 하며, 그 범위를 변수의 스코프 (scope) 라고 부릅니다. 모든 변수들은 변수가 정의되는 시점에서의 블록을 스코프로 가지게 됩니다.

예제 (function_local.py 로 저장하세요):

실행 결과:

동작 원리

먼저 함수의 첫번째 줄에서 x 라는 이름을 가진 변수에 담긴 값을 출력합니다. 이 때 함수를 정의하는 부분의 상단에서 정의되었던 값을 함수의 매개 변수 x로 넘겨받은 값이 출력됩니다.

다음으로, x 에 값 2 를 대입합니다. 그러나 x 는 함수의 지역 변수이므로, 함수 안에서 x 의 값이 대입된 값으로 변하는 반면 메인 블록의 x 는 변하지 않고 그대로 남아 있습니다.

프로그램에서 사용된 마지막 print 함수를 통해 메인 블록의 x 값을 출력해 보면, 그 이전에 호출된 함수 안에서 시행된 지역 변수값의 변화가 적용되지 않았음을 확인할 수 있습니다.

global 문

함수나 클래스 내부에서 상위 블록에서 선언된 변수의 값을 변경하고 싶을 경우, 파이썬에게 이 변수를 앞으로 지역 변수가 아닌 전역 (global) 변수로 사용할 것임을 알려 주어야 합니다. 이때 global 문을 이용합니다. global 문을 사용하지 않으면, 함수 외부에서 선언된 변수의 값을 함수 내부에서 변경할 수 없습니다.

함수 안에서 동일한 이름으로 선언된 변수가 없을 경우, 함수 밖의 변수값을 함수 안에서 읽고 변경할 수도 있습니다. 그러나, 이것은 프로그램을 읽을 때 변수가 어디서 어떻게 선언되었는지 파악하기 힘들게 만들기 때문에 추천할만한 방법이 아니며, 가능한 이런 경우를 피하시기 바랍니다. `global` 문을 사용하면 그 블록의 밖에서 그 변수가 선언되어 있음을 알려 주므로 좀 더 프로그램이 좀 더 명확해집니다.

예제 (`function_global.py` 로 저장하세요):

실행 결과:

동작 원리

`global` 문을 통해 `x` 가 전역 변수임을 파이썬에게 알려 줍니다. 따라서, 이후로 `x` 에 값을 대입하면 메인 블록의 `x` 값 또한 변경됩니다.

하나의 `global` 문으로 여러 개의 전역 변수를 동시에 지정해 줄 수도 있습니다. `global x, y, z` 와 같이 하면 됩니다.

기본 인수값

어떤 특별한 경우, 함수를 호출할 때 인수를 선택적으로 넘겨주게 하여 사용자가 값을 넘겨주지 않으면 자동으로 기본값을 사용하도록 하는 것이 편할 때가 있습니다. 이런 경우, 기본 인수값을 지정하면 됩니다. 함수를 선언할 때 원하는 매개변수 뒤에 대입 연산자 (=)와 기본값을 입력하여 기본 인수값을 지정합니다.

이 때, 기본 인수값은 반드시 상수이어야 합니다. 좀 더 정확히 말하자면, 불변값이어야 합니다. 불변값에 대해서는 뒤 챕터에서 다룰 것입니다. 일단 지금은 그래야 한다는 것만 기억해 두기 바랍니다.

예제 (`function_default.py` 로 저장하세요):

실행 결과:

동작 원리

함수 `say` 는 지정된 숫자 만큼 문자열을 반복하여 출력하는 합수입니다. 숫자를 지정하지 않으면 기본값이 적용되어, 문자열이 한 번만 출력됩니다. 이 결과는 매개 변수 `times` 의 기본값을 `1` 로 지정해 줌으로써 얻어집니다.

프로그램에서 처음 `say` 를 호출할 때에는 함수에 문자열만 넘겨 주어 한 번 출력하게 합니다. 두 번째 호출에서는 문자열과 인수 `5` 를 넘겨 주어 함수가 문자열을 5 번 반복하게 합니다.

> CAUTION
>
> 매개 변수 목록에서 마지막에 있는 매개 변수들에만 기본값을 지정해 줄 수 있습니다. 즉, 기본값을 지정하지 않은 매개 변수의 앞에 위치한 매개 변수에는 기본을 지 정할 수 없습니다.
>
> 이것은 함수 를 호출할 때 매개 변수의 위치에 맞도록 값이 지정되기 때문입니다. 예를 들어, `def func(a, b=5)` 는 옳은 함수 정의이지만 `def func(a=5, b)` 는 올바르지 않은 정의입니다.

키워드 인자

여러 개의 매개 변수를 가지고 있는 함수를 호출할 때, 그 중 몇 개만 인수를 넘겨주고 싶을 때가 있습니다. 이때 매개변수의 이름을 지정하여 직접 값을 넘겨줄 수 있는데 이것을 키워드 인자 라 부릅니다. 함수 선언시 지정된 매개 변수의 순서대로 값을 넘겨주는 것 대신, 매개 변수의 이름 (키워드) 를 사용하여 각각의 매개 변수에 인수를 넘겨 주도록 지정해 줍니다.

키워드 인자를 사용하는 데 두가지 장점이 있습니다. 첫째로, 인수의 순서를 신경쓰지 않고도 함수를 쉽게 호출할 수 있는 점입니다. 둘째로는, 특정한 매개 변수에만 값을 넘기도록 하여 나머지는 자동으로 기본 인수값으로 채워지게 할 수 있습니다.

예제 (`function_keyword.py` 로 저장하세요):

실행 결과:

동작 원리

위의 func 라 이름 지어진 함수는 기본 인수값이 지정되지 않은 한 개의 매개 변수와, 기본 인수값이 지정된 두 개의 매개 변수, 총 세 개의 매개 변수를 가지고 있습니다.

첫 번째 호출 func(3, 7) 에서, 매개 변수 a 는 값 3, 매개 변수 b 는 7 을 넘겨 받으며, c 에는 기본값 10 이 주어집니다.

두 번째 호출 func(25, c=24) 에서는, 첫 번째 인수인 25 가 변수 a 에 넘겨집니다. 그리고 매개 변수 c 는 키워드 인수를 통해 값 24 가 지정되며, 변수 b 에는 기본값 5 가 주어집니다.

세 번째 호출 func(c=50, a=100) 에서는, 모든 값을 지정하는 데 키워드 인자가 사용됩니다. 함수 정의에는 a 다음에 c 가 정의되어 있지만, 키워드 인수를 사용하면 그 순서에 상관없이 c 를 먼저 지정하고 a 를 나중에 지정할 수도 있다는 점을 확인하세요.

VarArgs 매개 변수

가끔 함수에 임의의 개수의 매개 변수를 지정해주고 싶을 때가 있습니다. 이때 VarArgs 매개 변수를 사용합니다. 아래 예제와 같이 별 (*) 기호를 사용하여 정해지지 않은 개수의 (Variable) 인수(Arguments) 를 표현합니다 (function_varargs.py 로 저장하세요):

실행 결과:

동작 원리

앞에 별 기호가 달린 매개 변수, 예를 들어 *param 과 같이 매개 변수를 지정해 주면 함수에 넘겨진 모든 위치 기반 인수들이 'param' 이라는 이름의 튜플로 묶여서 넘어옵니다.

또 이와 비슷하게 앞에 별 두 개가 달린 매개 변수, 예를 들어 **param 과 같이 매개 변수를 지정해 주면 함수에 넘겨진 모든 키워드 인수들이 'param' 이라는 이름의 사전으로 묶여서 넘어옵니다.

튜플과 사전에 대해서는 뒤에서 좀 더 자세히 다뤄보도록 하겠습니다.

return 문

return 문은 함수로부터 반환될 (return) 때, 즉 함수를 빠져 나올 때 사용됩니다. 이 때 return 뒤에 값을 지정해 주면, 함수가 종료될 때 그 값을 반환하도록 할 수 있습니다.

예제 (function_return.py 로 저장하세요):

실행 결과:

동작 원리

여기에서 사용된 maximum 함수는 매개 변수들 중 최대 값을 반환합니다. 위 경우에는 함수에 넘겨진 숫자들 중 최대값을 반환합니다. 간단한 if..else 구문을 통해 더 큰 값을 찾고, 최종 값을 반환(return) 합니다.

return 문 뒤에 아무 값도 지정하지 않는 경우, return None 을 실행하는 것과 같습니다. None 이란 파이썬에서 사용되는 특별한 형식으로 아무것도 없음을 의미합니다. 예를 들면, 어떤 변수의 값이 None 이라는 것은 변수에 할당된 값이 없다는 것을 의미합니다.

여러분이 return 문을 함수에 지정하지 않으면, 함수는 끝날 때 자동으로 return None 구문을 암시적으로 호출합니다. 아래 예제에서 return 문이 지정되지 않은 some_function 이라는 함수를 선언하고 print some_function() 을 실행하여 그 결과를 확인해 보시기 바랍니다:

```
def some_function():
    pass
```

pass 문은 아무 기능이 없는 구문입니다. 이것은 빈 블록을 지정해 줄 때 사용됩니다.

> TIP: 사실 파이썬에는 '최대값을 찾는' 에 쓸 수 있는 내장 함수 max 가 이미 포함되어 있습니다. 따라서 가능하면 이 내장 함수를 사용하시기 바랍니다.

DocString

파이썬은 DocString (Documentation String) 이라는 편리한 기능을 가지고 있습니다. DocString은 여러분이 만든 프로그램을 알아보기 쉽게 해 주고, 또 후에 프로그램에 대한 설명서를 작성할 때 유용하게 사용될 수 있는 중요한 도구입니다. 아래 예제와 같이, DocString은 프로그램이 실행중일 때도 읽어올 수 있습니다.

예제 (`function_docstring.py` 로 저장하세요):

실행 결과:

동작 원리

함수에 포함된 첫 논리적 명령줄에 적어둔 문자열은 함수의 DocString 이라고 불리우는 것입니다. 여기에서 설명하는 DocString은 모듈 과 클래스 에도 똑같이 적용됩니다. 각각에 대해서는 각 챕터에서 좀 더 자세히 알아보도록 하겠습니다.

DocString은 일반적으로 첫째줄의 첫 글자는 대문자로, 마지막 문자는 마침표로 끝나도록 작성합니다. 그리고 두번째 줄은 비워 두고, 세번째 줄부터는 이것이 어떤 기능을 하는지에 대해 상세하게 작성합니다. 저는 앞으로 여러분이 함수의 DocString를 작성할 때 이 규칙을 따르기를 강력히 권합니다.

`print_max` 함수의 DocString은 함수의 `__doc__` 속성을 통해 접근할 수 있습니다 (밑줄이 두 개 입니다). `__doc__` 은 함수 객체가 갖고 있는 기본 속성입니다. 파이썬에서는 함수를 포함한 모든 것이 객체로 간주된다는 점을 기억하세요. 이에 대해서는 클래스 챕터에서 좀 더 자세히 알아볼 것입니다.

파이썬에서 `help()` 를 활용해 보셨다면, 여러분은 이미 DocString을 본 적이 있는 것입니다! `help()` 가 하는 일은 주어진 대상의 `__doc__` 속성을 가져와서 화면에 보여주는 것이 다입니다. 따라서 위에서 만든 함수에 대해서도 마찬가지로 동작합니다. 여러분의 프로그램에 `help(print_max)` 라고 한 줄 추가해 보시기 바랍니다. `help` 창을 닫으려면 `q` 키를 누르세요.

이러한 방식을 활용하여 여러분의 프로그램에 대한 명세서를 자동으로 만들어주는 프로그램들이 있습니다. 따라서, 저는 여러분이 어떤 함수를 작성하시던지 DocString을 작성할 것을 강력히 권합니다. 파이썬과 함께 설치되는 `pydoc` 또한 `help()` 와 비슷한 방법으로 DocString을 활용하여 동작합니다.

요약

지금까지 함수에 대해 많은 것들을 알아 보았습니다만, 사실 아직 모든 기능을 다 알아본 것은 아닙니다. 그러나, 실질적으로 많이 활용되는 기능에 대해서는 거의 모든 것을 설명해 드렸기 때문에 아마 당장 프로그램을 작성할 때는 별 무리가 없을 것입니다.

다음으로는, 모듈을 작성하고 사용하는 방법에 대해 알아보도록 하겠습니다.

모듈

앞에서 함수를 통해 여러분의 프로그램 안에서 코드를 재사용하는 방법에 대해서 배워 보았습니다. 그러면 여러 함수들을 한꺼번에 불러들여 재사용하는 방법은 없을까요? 네, 이럴 때 모듈을 이용합니다.

모듈을 작성하는 데에는 여러가지 방법이 있습니다만, 가장 간단한 방법은 `.py` 확장자를 가진 파일을 하나 만들고 그 안에 함수들과 변수들을 정의해 두는 것입니다.

모듈을 작성하는 또 한 가지 방법은 여러분이 현재 사용중인 파이썬 인터프리터를 만드는데 사용되는 프로그래밍 언어로 모듈을 작성하는 것입니다. 예를 들어, 표준 파이썬 인터프리터를 사용 중인 경우 C 언어 를 이용하여 모듈을 작성하고 컴파일하면 파이썬에서 이것을 불러와 사용할 수 있습니다.

다른 프로그램에서 import 명령을 통해 모듈을 불러와 사용할 수 있습니다. 파이썬 표준 라이브러리 또한 동일한 방법을 통해 이용이 가능합니다. 일단, 표준 라이브러리를 불러와 사용하는 방법을 알아보도록 하겠습니다.

예제 (`module_using_sys.py` 로 저장하세요):

실행 결과:

동작 원리

먼저, 표준 라이브러리 중 하나인 sys 모듈을 import 문을 통해 불러왔습니다. 이것은 단순히 파이썬에게 이 모듈을 앞으로 사용할 것이라고 알려주는 과정이라고 생각하시면 편합니다. 여기서 사용된 sys 모듈은 파이썬 인터프리터와 인터프리터가 실행중인 환경, 즉 시스템(**sys**tem)에 관련된 기능들이 담겨 있습니다.

파이썬이 import sys 문을 실행하는 시점에서, 파이썬은 sys 모듈을 찾습니다. 이 경우에는 내장 모듈을 불러오는 것이므로, 파이썬이 이미 어디서 이것을 불러와야 하는지 알고 있습니다.

만약 그렇지 않은 경우, 예를 들어 파이썬으로 여러분이 직접 작성한 모듈인 경우, 파이썬 인터프리터는 sys.path 변수 에 정의된 디렉토리들에 해당 모듈이 있는지 검색합니다. 여기서 모듈을 찾는 데 성공하면, 그 모듈 내부에 적혀있는 명령들이 읽어들여지게 되고 모듈이 사용가능하게 됩니다. 이러한 로딩 과정은 첫번째로 모듈을 불러올 때만 이루어진다는 것을 기억하세요.

sys 모듈의 argv 변수를 불러올 때, 마침표를 이용하여 sys.argv 와 같이 접근합니다. 이와 같이 마침표를 이용함으로써 뒤에 온 이름이 sys 모듈에 존재한다는 것을 명시적으로 알려 주고, 또한 여러분의 프로그램에 사용될지 모를 argv 라는 이름을 가진 다른 변수와 충돌이 일어나지 않도록 합니다.

sys.argv 변수는 문자열의 입니다 (리스트에 대해서는 뒤 챕터에서 좀 더 자세히 다룰 것입니다). 좀 더 구체적으로 sys.argv 가 담고 있는 것은 명령줄 인수들의 리스트인데, 이것은 명령줄로부터 프로그램을 실행시킬 때 함께 넘어온 인수들을 담고 있는 것입니다.

여러분이 프로그램을 작성하고 실행할 때 IDE(Integrated Development Environment, 통합 개발 환경)를 활용하는 경우, IDE 상에서 프로그램을 실행할 때 를 지정하는 방법에 대해 알아보시기 바랍니다.

여기서 우리는 직접 python module_using_sys.py we are arguments 명령을 통해 프로그램을 실행하였습니다. 이것은 python 명령을 통해 module_using_sys.py 모듈을 실행시키고 함께 적어준 명령줄 인수들을 실행될 프로그램에 넘겨준 것과 같습니다. 이렇게 하면 프로그램이 실행될 때 파이썬은 sys.argv 변수에 이 인수들을 저장해 주어 프로그램에서 사용할 수 있도록 합니다.

이 때, 여러분이 실행한 스크립트의 이름이 sys.argv 리스트의 첫번째 항목이 됨을 기억하세요. 즉, 이 경우 'module_using_sys.py' 가 sys.argv[0] 에 해당하며, 'we' 는 sys.argv[1] 에, 'are' 는 sys.argv[2] 에, 'arguments' 는 sys.argv[3] 에 해당합니다. 파이선은 숫자를 0부터 센다는 점을 기억하세요 (1이 아닙니다!).

sys.path 변수에는 모듈을 불러올 때 찾게 되는 디렉토리들의 리스트가 담겨 있습니다. sys.path 변수의 첫 번째 항목이 공백 문자열임을 확인하세요. 공백 문자열은 현재 디렉토리를 의미하는데, 따라서 이것은 현재 디렉토리 또한 sys.path 의 일부임을 의미하며, 이것은 PYTHONPATH 환경변수와 같은 값입니다. 즉, 여러분은 현재 디렉토리에 들어 있는 파이썬 모듈을 불러와 사용할 수 있는 것입니다. 그렇지 않은 경우, sys.path 에 지정된 디렉토리들 중 하나에 해당 모듈이 들어 있어야 합니다.

이때 현재 디렉토리란 프로그램을 실행할 때 위치하고 있는 디렉토리를 말합니다. import os; print os.getcwd() 를 실행하여 여러분의 프로그램에서 현재 디렉토리가 어디인지 확인할 수 있습니다.

바이트 컴파일된 .pyc 파일

모듈을 불러오는 것은 상대적으로 무거운 작업이기 때문에, 파이썬은 약간의 트릭을 써서 이 과정을 더 빠르게 수행할 수 있도록 합니다. 그것은 바로 .pyc의 확장자를 가지는 바이트 컴파일된, 일종의 중간 단계를 만들어 두는 것입니다 (입문 섹션에서 파이썬의 인터프리터 환경에 대해 설명했었던 것을 기억하시죠?). 이렇게 한번 만들어진 .pyc 파일은 다른 프로그램에서 그 모듈을 다시 필요로 할 때 사용되며, 이 경우 모듈을 읽어들이는 데 필요한 몇가지 선행 작업을 수행하지 않아도 되므로, 더 빠른 속도로 모듈을 불러올 수 있습니다. 이렇게 바이트 컴파일된 파일은 플랫폼에 구애받지 않습니다.

NOTE: .pyc 파일은 .py 파일이 저장되어 있는 디렉토리에 새롭게 생성됩니다. 파이썬이 이 디렉토리에 쓰기 권한을 가지고 있지 못한 경우, .pyc 파일은 생성되지 않을 것입니다.

from..import 문

매번 sys. 를 입력하지 않고서도 argv 변수를 프로그램에서 곧바로 불러와서 사용할 수도 있습니다. 이런 경우, from sys import argv 와 같은 구문을 이용합니다.

> WARNING: 그러나 식별자 이름들끼리의 충돌을 피하고 프로그램을 좀 더 읽기 쉽게 작성하기 위해서, 가능하면 from..import 를 피하고 import 문을 사용하기를 권합니다.

예제:

```
from math import sqrt
print("Square root of 16 is", sqrt(16))
```

모듈의 __name__ 속성

파이썬에서 모든 모듈은 이름을 가지고 있으며, 모듈 내에 포함된 명령을 통해 그 모듈의 이름을 알아올 수 있습니다. 이것은 현재 모듈이 다른 프로그램에 의해서 불러들여져서 사용되고 있는지 아니면 인터프리터에서 곧바로 실행된 것인지를 구분하는데 편리하게 활용될 수 있습니다. 이전에 언급하였듯이 모듈 내부의 코드는 모듈이 첫번째로 불러들여졌을 때 한번 전체적으로 실행되게 됩니다. 이 점을 활용하면, 모듈이 외부로부터 불러들여졌을 때와 곧바로 실행되었을 때에 서로 각각 다르게 처리되도록 할 수 있습니다. 이를 위해 __name__ 속성을 사용합니다.

예제 (module_using_name.py 로 저장하세요):

실행 결과:

동작 원리

모든 파이썬 모듈은 __name__ 속성을 가지고 있습니다. 만약 그 이름이 __'main'__일 경우, 이것은 모듈이 직접 실행된 것임을 의미하며 따라서 이에 맞는 적절한 처리를 해 줄 수 있습니다.

새로운 모듈 작성하기

모듈을 작성하는 것은 쉽습니다. 사실 이것은 여러분이 지금까지 해 왔던 것과 별 다를게 없습니다! 왜냐하면 모든 파이썬 프로그램은 곧 모듈이기 때문입니다. 즉 .py 확장자를 가진 이름으로 저장되기만 하면 됩니다. 아래 예제를 통해 이에 대해 좀 더 분명하게 알 수 있을 것입니다.

예제 (mymodule.py 로 저장하세요):

위 예제는 의 예시입니다. 보이는 바와 같이, 지금까지 작성해왔던 파이썬 프로그램들에 비해 별다른 특별한 것이 없습니다. 다음으로는 다른 파이썬 프로그램으로부터 모듈을 불러와서 사용하는 방법을 알아보겠습니다.

아래 예제를 실행시키기 전에, 위 예제 프로그램 파일이 아래 예제 프로그램과 같은 디렉토리에 있거나 혹은 sys.path 중 하나에 있어야 모듈을 제대로 불러올 수 있음을 기억하세요.

또다른 모듈 (mymodule_demo.py 로 저장하세요):

실행 결과:

동작 원리

앞서 알아보았던 것과 동일하게 모듈의 구성 요소에 접근할 대에도 마침표를 활용합니다. 파이썬은 같은 일을 할 때 같은 표기법을 사용하며, 이는 직관적이고 '파이썬스러운' 느낌을 줍니다. 새로운 일을 하기 위해 새로운 것을 또 배울 필요가 없습니다.

아래는 from..import 구문을 사용하는 예제입니다 (mymodule_demo2.py로 저장하세요):

mymodule_demo2.py 를 실행한 결과는 mymodule_demo.py 를 실행한 결과와 같습니다.

만약 mymodule을 불러올 때 __version__ 이라는 이름이 이미 모듈에 선언되어 있었다면 충돌이 일어날 것입니다. 사실 많은 경우 모듈의 버전을 이 이름으로 주로 선언해 두기 때문에 실제로 자주 일어나는 일이기도 합니다. 따라서, 프로그램이 조금 길어지는 한이 있더라도 왠만하면 import 문을 이용하여 프로그램을 작성하기를 권합니다.

아래와 같이 사용할 수도 있습니다:

```
from mymodule import *
```

이것은 say_hi 와 같이 모듈에 포함된 모든 public으로 공개된 이름들을 불러옵니다. 그러나 밑줄 두 개로 시작하는 이름들, 예를 들어 __version__ 과 같은 이름들은 불러오지 않습니다.

> WARNING: 가능하면 from mymodule import * 처럼 사용하는 것을 피하시기 바랍니다.

Zen of Python (파이썬 정신)

파이썬의 여러 지향점 중 하나는 "명시적인 것이 암시적인 것 보다 낫다 (Explicit is better than Implicit)" 입니다. 파이썬에서 import this 를 실행하여 이러한 지향점들에 대해 알아보세요.

`dir` 내장함수

`dir()` 내장함수를 이용하여 객체에 정의되어 있는 식별자들의 목록을 불러올 수 있습니다. 예를 들어, 모듈의 경우 함수와 클래스 및 변수들의 식별자 이름이 정의되어 있을 것입니다.

이 함수는 인자를 받을 수 있습니다. 만약 모듈 이름을 넘겨 주면, 모듈 안에 선언된 식별자 이름들의 목록을 반환해 줍니다. 아무것도 넘겨주지 않는 경우, 현재 모듈에 선언된 식별자 이름들의 목록이 반환됩니다.

예제:

```
$ python
>>> import sys

# sys
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
'version', 'version_info']
#

#
>>> dir()
['__builtins__', '__doc__',
'__name__', '__package__', 'sys']

#      'a'
>>> a = 5

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys', 'a']

#
>>> del a

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
```

동작 원리

먼저 `dir` 함수를 통해 앞서 불러온 `sys` 모듈의 식별자 정보를 읽어옵니다. 그러면 내부에 포함된 긴 식별자 이름 목록이 나타나는 것을 확인할 수 있습니다.

다음으로, `dir` 함수를 아무 인수도 넘기지 않고 실행시킵니다. 그러면 기본값으로, 현재 모듈에 선언된 식별자 목록이 나타납니다. 이 때 앞서 불러온 모듈의 이름도 이 목록에 포함되어 있음을 확인하세요.

이제 `dir` 이 잘 작동하는지 확인하기 위해, 새로운 변수 `a` 를 선언하고 값을 할당해 준 뒤 `dir` 함수를 실행시켜 반환되는 목록에 새로 생성된 변수가 포함되어 있는지 확인합니다. `del` 문을 이용하여 현재 모듈에 선언된 변수 혹은 속성을 제거하면 `dir` 함수의 출력에 그 결과가 반영됨을 또한 알 수 있습니다.

`del` 문은 변수 혹은 이름을 삭제(delete)하는데 사용됩니다. 이 구문이 실행된 이후에는, 즉 위 예제의 경우 `del a` 를 실행한 이후에는, 더 이상 변수 `a` 에 접근할 수 없습니다. 마치 처음부터 이러한 변수가 존재하지 않았던 것처럼 말입니다.

`dir()` 함수는 어떤 객체에도 사용될 수 있습니다. 예를 들어, `dir(str)` 를 실행하면 `str` (문자열) 클래스의 속성에 대해서 알아볼 수 있습니다.

파이썬에는 정의된 속성과 그 값을 읽어오는 데 사용될 수 있는 `vars()` 함수 또한 존재합니다만, 이 함수는 언제나 모든 경우에서 동작하지는 않습니다.

패키지

지금까지 여러분은 파이썬 프로그램의 계층 구조에 대해 어느정도 파악이 되셨을 것입니다. 변수는 함수 내부에 존재하며, 함수와 전역 변수는 모듈 안에 존재합니다. 그렇다면 모듈은 어디에 포함되는 것일까요? 파이썬에서는 패키지라는 단위가 이에 해당됩니다.

패키지란 그냥 모듈이 들어 있는 폴더를 말하는 것입니다만, 파이썬에게 이 폴더는 파이썬 모듈을 담고 있다는 것을 알려주는 역할을 하는 `__init__.py` 라는 특별한 파일을 한 개 포함하고 있습니다.

여러분이 'asia', 'africa'라는 하위 패키지를 포함하고 있는 'world' 라는 패키지를 만들고 싶다고 가정해 봅시다. 또한 각각의 하위 패키지는 'india', 'madagascar' 등등의 하위 패키지를 하나씩 더 갖고 있습니다.

이 경우, 아래와 같이 폴더 구조를 만들어 주면 됩니다:

```
- <some folder present in the sys.path>/
    - world/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - foo.py
        - africa/
            - __init__.py
            - madagascar/
                - __init__.py
                - bar.py
```

패키지는 계층적으로 모듈을 관리할 수 있게 편의상 구성하는 것입니다. 표준 라이브러리에서 이러한 계층 구조를 많이 확인하실 수 있을 것입니다.

요약

함수가 재사용 가능한 프로그램의 한 부분인 것과 같이 모듈은 재사용 가능한 프로그램을 말하며, 패키지는 모듈을 구성하는 계층 구조를 말합니다. 파이썬과 함께 설치되는 표준 라이브러리는 이러한 패키지와 모듈로 이루어진 좋은 예제입니다.

지금까지 모듈을 사용하고 작성하는 방법에 대해 알아보았습니다.

다음으로, 자료 구조의 개념에 대해 알아보도록 합시다.

자료구조

자료구조란 간단하게, 어떤 자료를 담는 구조를 말합니다. 다른 말로 하면, 서로 연관있는 어떤 자료들의 집합을 저장하는 데 사용됩니다.

파이썬에는 네 종류의 자료구조가 있는데, 각각 리스트, 튜플, 사전(dictionary), 집합(set) 입니다. 이제 앞으로 각각의 사용법에 대해 알아보고 또 각각이 얼마나 편리한지 알아보도록 하겠습니다.

리스트

란 순서대로 정리된 항목들을 담고 있는 자료 구조입니다. 즉, 리스트에는 항목의 목록을 저장할 수 있습니다. 이것은 쉽게 말하자면 장 보러 갈 때 적는 일종의 장바구니 목록 같은 것인데, 아마도 여러분은 각 품목들을 한줄씩 적는 데 익숙하시겠지만 리스트에서는 줄 대신 쉼표로 각 항목을 구분한다는 점이 다릅니다.

리스트를 정의할 때는 대괄호 [] 를 이용해서 파이썬에게 이것이 리스트를 의미한다는 것을 알려 줍니다. 한번 리스트를 만들어 두면 여기에 새로운 항목을 추가하거나 삭제할 수 있으며, 특정 항목이 존재하는지 검색할 수도 있습니다. 이 때 항목을 추가 및 삭제가 가능하다는 특징을 비정적(mutable)이라고 하며, 리스트는 비정적 자료구조의 한 예로 내부 항목을 변경할 수 있는 자료형입니다.

객체와 클래스에 대한 간단한 소개

객체와 클래스에 대해서는 좀 더 나중에 다룰 예정이지만, 여기서 여러분이 리스트에 대해 좀 더 잘 이해하실 수 있도록 이에 대한 간단한 소개를 하도록 하겠습니다. 이들에 대해서는 뒤 챕터에서 좀 더 자세하게 다루겠습니다.

리스트는 객체와 클래스가 사용된 한 예입니다. 변수 `i` 를 선언하고 예를 들어 `5` 라는 정수형 값을 할당해 주는 것은, `int` 라는 클래스 (또는 타입) 의 객체 (또는 인스턴스) `i` 를 만드는 것입니다. 이에 대해 좀 더 자세히 알아보려면 `help(int)` 를 읽어보시기 바랍니다.

클래스는 메소드를 가질 수 있는데, 여기서 메소드란 그 클래스 내에 정의된 고유의 내장 함수들을 말합니다. 또 이러한 내장 함수들은 클래스로 객체를 생성했을 때에야 비로소 사용할 수 있습니다. 예를 들어, 파이썬의 `list` 클래스에는 `append` 라는 메소드가 포함되어 있으며 이는 리스트의 마지막에 항목을 한 개 추가할 때 사용되는 메소드입니다. 즉 `mylist.append('an item')` 라 하면 리스트 `mylist` 의 마지막에 해당 문자열을 추가해 줍니다. 이 때 객체의 메소드에 접근할 때에도 마침표를 이용한다는 점을 기억하세요.

또 클래스는 필드를 가질 수 있는데 이것은 단순히 그 클래스 내에 정의된 변수들을 의미합니다. 메소드와 마찬가지로 이러한 변수들은 클래스로 객체를 생성했을 때에야 비로소 사용할 수 있습니다. 필드도 메소드와 마찬가지로 마침표를 이용하여 접근합니다. 예를 들면 `mylist.field` 와 같습니다.

예제 (`ds_using_list.py` 로 저장하세요):

실행 결과:

동작 원리

어떤 사람이 마트에 장 보러 갈 때 장바구니 목록을 변수 `shoplist` 에 담아 두었다고 합시다. 이 `shoplist` 에는 구매할 항목들의 이름들, 즉 문자열들이 담겨 있을 것이지만, 사실 리스트에는 어떤 종류의 객체 든지 담을 수 있어서 숫자도 담을 수 있고, 심지어는 리스트 안에 다른 리스트도 담을 수 있습니다.

여기서는 `for..in` 반복문을 사용하여 리스트 안에 담겨 있는 항목들을 하나씩 반복하여 읽어왔습니다. 이제쯤 되면 리스트는 열거형이라는 점을 알고 계셔야 합니다. 열거형에 대해서는 뒤에 좀 더 자세히 다루겠습니다.

print 함수에 추가한 `end` 매개변수는 출력될 내용 뒤에 줄바꿈 기호 대신 공백을 출력하도록 지정해 준 것입니다.

다음으로, 앞서 설명했듯이 리스트 객체의 `append` 메소드를 이용해서 리스트에 항목을 한 개 추가합니다. 그리고, 추가된 항목이 제대로 추가되었는지 확인하기 위해 print 함수에 리스트를 넘겨 주어 리스트의 내용을 화면에 예쁘게 출력해 줍니다.

이제 리스트의 `sort` 메소드를 이용하여 리스트의 내용을 정렬해 줍니다. 여기서 이 메소드는 해당 리스트 자체를 변화시키며 수정된 리스트를 또 반환해주지 않는데 이 점을 이해하는 것이 중요합니다. 정적(immutable) 인 문자열과 달리 리스트는 변화될 수 있는 성질을 지녔으며 이것을 비정적(mutable) 이라 합니다.

다음으로 물건을 구매한 뒤에는 장바구니 목록에서 해당 항목을 지워야 할 것입니다. 이 때 `del` 문을 사용하여 항목을 삭제합니다. 항목을 삭제할 때는 어떤 항목을 지울지 del 문에 알려 주면 리스트로부터 해당 항목이 삭제됩니다. 여기서는 `del shoplist[0]` 이라고 입력하여 첫 번째 항목을 삭제할 것임을 알려 주었습니다 (파이썬은 숫자를 0부터 센다는 점을 기억하시기 바랍니다).

리스트 객체가 갖고 있는 모든 메소드에 대해 알고 싶으시면, `help(list)` 를 입력해 보시기 바랍니다.

튜플

튜플은 여러 개의 객체를 모아 담는 데 사용됩니다. 튜플은 리스트와 비슷하지만, 리스트 클래스에 있는 여러가지 기능이 없습니다. 또 튜플은 수정이 불가능하며, 그래서 주로 문자열과 같이 비정적인 객체들을 담을 때 사용됩니다.

튜플은 괄호로 묶인 쉼표로 구분된 여러 개의 항목으로 정의되며, 괄호는 생략이 가능합니다.

튜플에 저장된 값들은 수정이 불가능하기 때문에, 단순히 값들의 목록을 저장해두고 쓸 때 주로 사용됩니다.

예제 (ds_using_tuple.py 로 저장하세요):

실행 결과:

동작 원리

변수 zoo 는 여러 항목들을 담고 있는 튜플입니다. 보시는 바와 같이 len 함수를 통해 튜플의 길이를 알아올 수 있습니다. 튜플 또한 열거형 의 한 예입니다.

이제 동물원(zoo) 안의 동물들을 새로운 동물원으로 옮겨야 한다고 해 봅시다. 이를 위해 new_zoo 튜플이라는 새로운 동물원을 만들어 주고, 새 동물원에 원래 있던 동물들에 추가로 기존의 동물원에 있던 동물들을 옮겨 왔습니다. 이 때 예제와 같이 튜플 안에 튜플을 담아도 튜플의 성질을 잃지 않습니다.

리스트에서 했던 것과 같이, 튜플 안에 있는 항목의 위치를 대괄호로 묶어 지정해주면 각 항목에 접근할 수 있습니다. 이를 인덱싱 연산자라고 부릅니다. new_zoo 의 세 번째 항목에 접근하려면 new_zoo[2] 와 같이 하며, 이 세 번째 항목은 튜플이므로 이것의 세 번째 항목에 접근하려면 new_zoo[2][2] 와 같이 합니다. 익숙해지면 쉽게 느껴질 것입니다.

> **빈 튜플과 한 개짜리 튜플**
>
> 빈 튜플은 괄호 안에 아무것도 적지 않고 myempty = () 와 같이 생성할 수 있습니다. 그러나, 항목 한 개만 담고 있는 튜플을 정의할 때는 주의해야 합니다. 이 경우 첫 번째 항목의 뒤에 쉼표를 붙여 주어 파이썬에게 이것이 숫자 연산에 사용되는 괄호가 아니라 객체를 담는 튜플을 의미하는 것이라는 것을 구분할 수 있도록 단서를 주어야 합니다. 예를 들어, 항목 2 를 담고 있는 튜플을 정의하려면 singleton = (2 ,) 와 같이 합니다.
>
> **펄 프로그래머를 위한 주석**
>
> 리스트 안의 리스트는 리스트의 성질을 잃지 않습니다. 리스트는 펄에서처럼 flatten 되지 않습니다. 이 성질은 튜플 안의 튜플이나 리스트 안의 튜플 혹은 튜프 안의 리스트의 경우 모두에 적용됩니다. 파이썬에서는 이들은 단지 다른 객체 안에 저장된 객체들일 뿐입니다.

사전 (딕셔너리)

사전(딕셔너리)은 이를테면 전화번호부 같은 것인데, 누군가의 이름을 찾으면 그 사람의 주소와 연락처를 알 수 있는 것과 같습니다. 이 때 그 사람의 이름에 해당하는 것을 키 라 부르고, 주소와 연락처 등 세부사항에 해당하는 것을 값 이라 부릅니다. 전화번호부에 동명이인이 있을 경우 어떤 정보가 맞는 정보인지 제대로 알아낼 수 없듯이, 사전의 키는 사전에서 유일한 값이어야 합니다.

사전의 키는 정적 객체 (문자열 등등) 이어야 하지만, 값으로는 정적 객체나 비정적 객체 모두 사용할 수 있습니다. 이것을 간단하게 다시 말하면 사전의 키로는 단순 객체만 사용할 수 있다고 표현합니다.

사전을 정의할 때 키와 값의 쌍은 d = {key1 : value1, key2 : value2 } 와 같이 지정해 줍니다. 이 때 키와 값은 콜론으로 구분하며 각 키-값 쌍은 쉼표로 구분하고 이 모든 것을 중괄호 {}로 묶어 준다는 것을 기억하시기 바랍니다.

여기서 사전의 키-값 쌍은 자동으로 정렬되지 않으며, 파이썬 3.6 이상에서는 값이 저장된 순서대로 순서가 지정됩니다. 다르게 정렬된 값을 원한다면, 사용하기 전에 먼저 직접 정렬을 해 주어야 합니다.

앞으로 여러분이 사용하게 될 사전은 dict 라는 클래스의 인스턴스/객체입니다.

예제 (ds_using_dict.py 로 저장하세요):

실행 결과:

동작 원리

먼저 앞서 설명한대로 사전 ab 를 생성합니다. 그러면 리스트와 튜플을 설명할 때 언급했었던 인덱싱 연산자 (대괄호, []) 에 키를 지정해 주어 사전의 키-값 쌍에 접근할 수 있습니다. 간단하지요?

키-값 쌍도 리스트에서 했던 것과 똑같이 del 문으로 삭제할 수 있습니다. del 을 적고 사전의 이름을 적은 뒤, 인덱싱 연산자 안에 삭제하고 싶은 키를 지정해 주기만 하면 됩니다. 이 때 그 키에 해당하는 값은 지정해 줄 필요가 없습니다.

다음으로, 사전의 `items` 메소드를 사용하여 각 키-값 쌍에 접근합니다. 이 메소드는 키와 값 순으로 구성된 튜플들을 묶은 튜플 하나를 반환해 줍니다. 그 후 `for..in` 반복문을 사용하여 키와 값 각각을 변수 `name` 과 `address` 에 지정해 주게 하고 그 값을 출력합니다.

위 예제의 'Guido' 와 같이 인덱싱 연산자를 사용하여 새로운 키-값 쌍을 추가할 수도 있습니다.

또, 사전 안에 키-값 쌍이 존재하는지 `in` 연산자를 통해 확인할 수 있습니다.

`dict` 클래스의 모든 메소드 목록을 확인하시려면 `help(dict)` 를 입력하시기 바랍니다.

> **키워드 인수와 사전의 관계**
>
> 함수를 호출할 때 키워드 인수를 사용해 보셨다면, 여러분은 이미 사전을 사용해 본 것입니다. 여기서 함수의 매개 변수들의 이름은 사전의 키에 대응하고, 각 매개 변수에 넘겨주는 값은 사전의 값에 대응합니다. 이러한 사전과 같은 것을 심볼 테이블 이라고 부릅니다.

열거형

열거형들은 리스트, 튜플, 문자열 같은 것입니다. 그러면 열거형이란 무엇이고 열거형에서는 무엇이 중요할까요?

열거형의 주요한 두 가지 기능은 멤버십 테스트 (`in` 과 `not in` 연산), 즉 어떤 항목이 열거형 안에 있는지 없는지 확인하는 것, 그리고 열거형 안의 특정 항목을 지정하여 가져올 수 있는 인덱싱 연산 입니다.

여기에 더하여 위에서 설명한 리스트, 튜플, 문자열의 세 가지 열거형은 슬라이스 연산 기능을 가지고 있는데, 이것은 열거형의 일부분을 잘라(slice) 반환하는 연산을 말하며, 다시 말해 열거형의 부분집합을 반환해 주는 연산입니다.

예제 (`ds_seq.py` 로 저장하세요):

실행 결과:

동작 원리

먼저, 열거형의 각 항목을 얻어오기 위해 어떻게 인덱스 연산을 사용하는지 보겠습니다. 이를 다른 말로 서브스크립션 연산 이라고도 합니다. 위 예제에서 보인 것과 같이 대괄호 내에 특정 숫자를 지정해 주면, 파이썬은 열거형에서 그 위치에 있는 항목을 얻어옵니다. 여기서 파이썬은 숫자를 0부터 센다는 점을 기억하시기 바랍니다. 따라서 `shoplist[0]` 과 `shoplist[3]` 은 각각 열거형 `shoplist` 의 첫 번째와 네 번째 항목을 읽어오는 연산을 의미합니다.

인덱스에는 음수가 지정될 수도 있습니다. 이 경우, 열거형의 마지막부터 그 위치가 계산됩니다. 따라서, `shoplist[-1]` 은 열거형의 마지막 항목을 의미하며 `shoplist[-2]` 는 열거형의 마지막 항목 바로 뒤의 항목을 의미합니다.

슬라이스 연산은 대괄호 안에 콜론으로 구분한 숫자들을 입력해 주는 것입니다. 슬라이스 연산은 앞서 설명한 인덱싱 연산과 매우 비슷합니다. 이 경우 숫자는 반드시 지정해 줄 필요는 없지만 콜론은 반드시 들어가야 합니다.

슬라이스 연산에서 콜론 앞의 첫 번째 숫자는 슬라이스를 시작할 위치를 의미하며 콜론 뒤의 두 번째 숫자는 슬라이스를 멈출 위치를 지정합니다. 만약 첫 번째 숫자가 지정되지 않았을 경우, 파이썬은 열거형의 맨 처음부터 슬라이스를 시작합니다. 두 번째 숫자가 지정되지 않았을 경우, 파이썬은 열거형의 맨 끝에서 슬라이스를 멈춥니다. 이 때 슬라이스는 시작 위치부터 슬라이스를 시작 하며 끝 위치의 직전까지 수행됩니다. 즉, 시작 위치에 해당하는 항목은 슬라이스에 포함되나 마지막 위치에 해당하는 항목은 포함되지 않습니다.

따라서, `shoplist[1:3]` 은 위치 1 에 해당하는 항목부터 시작하여 위치 2 에 해당하는 항목을 포함하지만, 위치 3 에 해당하는 항목은 포함하지 않습니다. 따라서 두 개의 항목의 슬라이스 가 반환됩니다. 이와 비슷하게, `shoplist[:]` 는 전체 열거형의 복사본이 반환됩니다.

슬라이스 숫자로도 음의 위치를 지정해 줄 수 있습니다. 음수는 열거형의 마지막부터 위치를 계산하는 것을 의미합니다. 예를 들어, `shoplist[:-1]` 은 마지막 항목을 제외한 모든 항목을 포함하고 있는 슬라이스를 반환해 줍니다.

슬라이스 숫자에 세 번째 인수를 지정해 줄 수 있는데, 이것을 슬라이스의 스텝 이라 합니다 (기본값은 1 입니다):

```
>>> shoplist = ['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::1]
['apple', 'mango', 'carrot', 'banana']
>>> shoplist[::2]
['apple', 'carrot']
```

```
>>> shoplist[::3]
['apple', 'banana']
>>> shoplist[::-1]
['banana', 'carrot', 'mango', 'apple']
```

보시는 바와 같이 스텝이 2일 경우 위치 0, 2, ⋯ 에 해당되는 항목들이 반환되며 스텝이 3일 경우 0, 3, ⋯ 에 해당되는 항목들이 반환됩니다.

파이썬 인터프리터에서 여러 가능한 슬라이스 숫자의 조합들을 시험해 보시면 그 결과를 곧바로 확인해 보실 수 있습니다. 이 모든 사항은 모든 열겨형에 적용되므로, 튜플, 리스트, 문자열의 경우 모두 동일한 방법을 사용할 수 있습니다.

집합 (Set)

집합은 정렬되지 않은, 복잡하지 않은 객체들의 묶음입니다. 집합은 포함된 객체들의 순서나 중복에 상관없이 객체를 묶음 자체를 필요로 할 때 주로 사용합니다.

집합끼리는 멤버십 테스트를 통해 한 집합이 다른 집합의 부분집합인지 확인할 수 있으며, 두 집합의 교집합 등을 알아낼 수도 있습니다.

```
>>> bri = set(['brazil', 'russia', 'india'])
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bric.issuperset(bri)
True
>>> bri.remove('russia')
>>> bri & bric # OR bri.intersection(bric)
{'brazil', 'india'}
```

동작 원리

이 책을 읽는 여러분들은 아마도 학교에서 기초 집합론에 대해 이미 배우셨을 것이므로 위 예제에 대해서는 딱히 설명할 것이 없습니다. 만약 아니라면, “집합론” 이나 “벤 다이어그램” 등을 검색해 보시면 파이썬에서 집합을 어떻게 활용할 수 있는지 좀 더 자세히 알 수 있을 것입니다.

참조

객체를 생성하고 변수에 할당해 줄 때, 사실 실제 객체가 변수에 할당되는 것은 아닙니다! 변수에는 객체의 참조 가 할당됩니다. 참조란, 그 변수의 이름이 여러분의 컴퓨터 메모리 어딘가에 저장되어 있는 실제 객체의 위치를 가리키는 것을 말합니다. 이러한 것을 객체에 이름을 바인딩 한다고 합니다.

일반적으로는 참조에 대해 크게 신경 쓸 필요가 없습니다만, 이로 인해 발생하는 몇 가지 현상에 대해 알고 계실 필요가 있습니다:

예제 (ds_reference.py 로 저장하세요):

실행 결과:

동작 원리

주석에 거의 모든 설명을 달아 두었습니다.

리스트와 같은 어떤 열거형이나 복잡한 객체 (정수형과 같이 단순한 객체 들을 제외한 것들) 의 복사본을 생성하고 싶을 때에는, 슬라이스 연산자를 이용하여 복사본을 생성해야 합니다. 단순히 한 변수를 다른 변수에 할당하게 되면 두 변수는 같은 객체를 “참조” 하게 되며 실제 복사본이 생성되지 않습니다. 따라서 이 점을 유의하지 않으면 문제가 발생할 수 있습니다.

펄 프로그래머를 위한 주석

이미 존재하는 리스트를 다른 변수에 할당하는 구문은 복사본을 만드는 것이 **아닙니다**. 열거형의 복사본을 만들려면 반드시 슬라이스 연산자를 사용하시기 바랍니다.

문자열에 대한 좀 더 자세한 설명

앞서 문자열에 대해 이미 상세히 다루었지만, 몇 가지 더 알아두면 좋을 것들이 있습니다. 문자열도 객체이므로 여러 메소드를 가지고 있는데 이를 통해 문자열의 앞 뒤 공백을 제거한다거나 하는 일들을 할 수 있습니다. 사실 우리는 이미 문자열의 메소드를 한번 써 본 적이 있습니다… 바로 `format` 메소드입니다.

파이썬에서 사용되는 모든 문자열은 `str` 클래스의 객체입니다. 아래에 이 객체가 제공하는 몇가지 유용한 메소드들의 예제가 있습니다. str 클래스의 모든 메소드의 목록을 확인해 보시려면 `help(str)` 을 실행해 보시기 바랍니다.

예제 (`ds_str_methods.py` 로 저장하세요):

실행 결과:

동작 원리

여기서는 문자열이 제공하는 여러 메소드에 대해 알아보았습니다. `startswith` 메소드는 문자열이 주어진 문자열로 시작하는지 여부를 반환합니다. `in` 연산자는 문자열에 주어진 문자열이 포함되어 있는지 확인하는데 사용합니다.

`find` 메소드는 문자열 내에 포함된 특정 문자열의 위치를 반환합니다. 이 때 주어진 문자열을 찾지 못한 경우 `find` 는 `-1` 을 반환합니다. 또 `str` 클래스는 `join` 이라는 유용한 메소드를 가지고 있는데, 이것은 주어진 문자열들을 해당 문자열을 구분자로 하여 결합한 하나의 큰 문자열을 만들어 반환해 주는 메소드입니다.

요약

이 챕터에서는 파이썬의 여러 내장 자료구조들에 대해 상세히 알아보았습니다. 이 자료 구조들은 프로그램을 짧고 보기 쉽게 작성하는데 꼭 필요한 구성 요소들입니다.

이제 여러분은 파이썬의 여러 기본적인 문법에 익숙해졌을 것입니다. 이 다음부터는 실제 파이썬 프로그램을 설계하고 작성해 보도록 하겠습니다.

실생활 문제 해결

지금까지 파이썬이라는 언어의 여러 가지 구성 요소에 대해 배워 보았습니다. 이제는 지금까지 배운 것들을 토대로, 뭔가 유용한 것을 하는 프로그램을 만들어 보도록 합시다. 이 챕터의 목표는 여러분이 직접 파이썬 스크립트를 만들고 사용하는 법을 배우는 것입니다.

문제

다음과 같은 문제를 해결해 봅시다:

> 내 중요한 파일들을 백업해두는 프로그램을 만들고 싶어요.

이것은 간단한 문제이지만, 아직 어떻게 접근하면 좋을지 정보가 부족합니다. 따라서 약간 분석을 해 보도록 합시다. 예를 들어, 어떤 파일을 백업해야 할까요? 파일들은 어떻게, 또 어디에 저장되어야 할까요?

문제에 대해 분석한 이후에는, 프로그램을 설계해야 합니다. 이를 위해 우리가 만들 프로그램이 어떻게 동작하면 좋을지 그 목록을 만들어 봅시다. 저는 아래와 같이 저의 방식대로 목록을 만들었습니다. 그러나 모든 사람이 서로 다른 생각을 가지고 있는 것이 당연하므로, 여러분이 목록을 만들어도 저와 같은 항목들로 구성되어 있지 않을 수 있겠죠. 달라도 아무 상관 없습니다.

- 백업할 파일과 디렉토리들은 리스트의 형태로 지정해 둔다.
- 백업된 대상들은 백업 디렉토리 안에 저장되어야 한다.
- 백업된 파일들은 zip 파일로 압축한다.
- zip 파일의 이름은 현재 날짜와 시간으로 한다.

- GNU/Linux 환경이나 Unix 환경에서 기본으로 제공되는 zip 명령을 이용한다. (참고: 명령줄 인터페이스에서 사용할 수 있는 어떤 압축 유틸리티든지 사용이 가능합니다)

  윈도우 사용사를 위한 주석

  윈도우 사용자는 GnuWin32 프로젝트 페이지 에서 zip 명령을 내려받아 설치 할 수 있습니다. 또한 설치한 후 파이썬 명령을 어디서든 실행할 수 있도록 해 주었던 것처럼 C:\\Program Files\\GnuWin32\\bin 폴더를 시스템의 PATH 환경변수에 추가해 주면 어디서든 zip 명령을 사용할 수 있습니다.

첫번째 프로그램

일단은 프로그램을 안정적이게 설계한 것 같으므로, 코드를 입력하여 프로그램으로 구현 해 보도록 합시다.

backup_ver1.py 로 저장하세요:

실행 결과:

이제 테스트 단계로 넘어와서 프로그램이 잘 동작하는지 테스트 해보아야 합니다. 만약 예상대로 프로그램이 동작하지 않으면, 프로그램의 버그(bug)를 제거하는 과정, 즉 디버그(debug) 과정을 거쳐야 합니다.

위 예시 프로그램이 잘 동작하지 않는 경우, 실행 결과의 Zip command is 줄 뒤의 내용을 복사한 후 쉘 (GNU/Linux 나 Mac OS X) 혹은 cmd (윈도우 환경) 에 입력해본 뒤 무엇이 문제인지 확인하고 고쳐 보도록 합시다. 또 무엇이 잘못되었는지 확인하기 위해 zip 명령 설명서 또한 확인해 보시기 바랍니다. 만약 위 명령이 올바르게 실행된다면, 문제는 파이썬 프로그램 자체에 있는 것으로 추정되므로 프로그램을 올바르게 정확히 작성했는지 다시 한번 확인해 보시기 바랍니다.

동작 원리

아래에서 단계별로 어떻게 우리의 설계 를 코드 로 바꾸었는지에 대해 설명할 것입니다.

먼저 os 와 time 모듈을 불러왔습니다. 그리고, 백업할 파일들과 디렉토리들을 source 라는 리스트에 담아 두었고, 또 백업을 저장해 둘 대상 디렉토리는 target_dir 변수에 지정해 주었습니다. zip 파일의 이름은 현재 날짜와 시간으로 할 것이므로 time.strftime() 함수를 사용하여 현재 날짜와 시간을 얻어온 후 .zip 확장자를 붙여 target_dir 디렉토리에 저장하도록 했습니다.

여기서 os.sep 변수를 주목하시기 바랍니다. 이것은 여러분의 운영 체제에 다른 디렉토리 구분자를 나타내는 것으로 GNU/Linux, Unix, macOS 환경에서는 '/' 일 것이고, 윈도우 환경에서는 '\\\\' 일 것입니다. 이러한 문자들을 직접 사용하지 않고 os.sep 을 사용함으로써 여러분의 프로그램에 범용성을 제공할 수 있고, 따라서 여러 운영체제에서 수정 없이 사용할 수 있게 할 수 있습니다.

위 프로그램에서 사용한 것처럼 time.strftime() 함수는 특별한 형식을 인수로 넘겨 받습니다. %Y 는 네 자리의 연도로 치환되며, %m 은 두 자리의 달 즉 01 과 12 사이의 숫자를 의미합니다. 날짜 형식에 대해서는 파이썬 레퍼런스 매뉴얼 을 참고하시기 바랍니다.

이제 대상 zip 파일의 이름을 생성하기 위해 문자열 결합 연산자인 더하기 연산자를 사용하여 두 문자열을 합쳐서 하나의 문자열로 만듭니다. 그리고, zip_command 라는 문자열을 만들어 실제로 실행하게 될 문자열을 만듭니다. GNU/Linux 터미널이나 DOS 프롬프트에서 이 문자열을 실행시켜 보고 잘 동작하는지 확인해 볼 수 있을 것입니다.

앞으로 우리가 사용할 zip 명령은 몇 가지 옵션과 매개 변수를 필요로 합니다. -r 옵션은 zip 명령이 주어진 디렉토리에 대해 회귀적(recursive)으로, 즉 해당 디렉토리가 포함하고 있는 모든 하위 디렉토리와 파일들을 포함하도록 하는 명령입니다. 여기서는 두 가지 옵션을 결합하여 하나의 축약 옵션 -qr 을 지정해 주었습니다. zip 명령 뒤에는 차례로 이 옵션이 지정되고, 그 뒤에는 생성될 zip 파일의 이름이, 마지막으로는 압축될 대상 디렉토리들과 파일들이 지정됩니다. 이를 위해 source 리스트를 앞서 설명한 join 메소드를 통해 문자열로 바꿔서 넘겨 주었습니다.

그러면, 최종적으로 os.system 함수를 통해 이 명령을 실제 시스템(system) 의 쉘에서 실행시킵니다. 그러면 프로그램이 성공적으로 실행된 경우 0 이 반환되며 그렇지 않으면 숫자로 된 오류 코드가 반환됩니다.

이제 명령의 실행 결과에 따라, 적절한 메시지를 출력해 주고 백업이 성공했는지 실패했는지를 화면에 출력해 줍니다.

여기까지입니다. 이제 우리는 중요한 파일을 백업하는 스크립트를 성공적으로 만들었습니다!

윈도우 사용자를 위한 주석

이스케이프 문자 백슬래시를 두 번씩 입력하는 것보다 raw 문자열을 이용할 수도 있습니다. 예를 들어, 'C:\\\\Documents' 는 r'C:\\Documents' 로도 쓸 수 있습니다. 그러나, 'C:\\Documents' 와 같이 사용할 수는 없습니다. 이것은 \\D 라는 알 수 없는 이스케이프 문자를 의미하는 것으로 오류가 발생합니다.

이제 잘 동작하는 백업 스크립트를 만들었으므로, 이 스크립트를 언제든지 사용하여 중요한 파일들을 백업할 수 있습니다. 이 단계를 소프트웨어의 운영(Operation) 단계 혹은 배포(Deployment) 단계라고 합니다.

위 프로그램은 잘 동작하지만, 첫번째로 만든 프로그램은 (종종) 예상한 대로만은 실행되지 않습니다. 예를 들어 프로그램 설계를 잘못했다든지 코드를 입력할 때 실수를 했다던지 할 수 있습니다. 이런 경우 상황에 맞춰서 설계 단계로 돌아가거나 프로그램을 디버깅해야 합니다.

두 번째 프로그램

첫 번째로 만든 프로그램은 일단 잘 동작합니다. 그러나, 프로그램을 매일매일 잘 쓰기 위해 좀 더 개선의 여지가 있습니다. 이 단계를 소프트웨어의 유지보수(maintenance) 단계라고 합니다.

제가 이 프로그램을 쓰다가 느낀 한가지 개선점은 파일에 이름을 짓는 부분에 대한 것인데, 주 백업 디렉토리에 날짜로 된 하위 디렉토리를 만들고 시간으로 된 압축 파일들을 그 안에 넣는 것입니다. 이렇게 하면 백업된 파일들이 계층적으로 저장되므로 좀 더 쉽게 관리할 수 있을 것입니다. 또한, 파일명도 좀 더 짧아집니다. 마지막으로 각각 디렉토리에 백업 파일이 나뉘어 저장되므로 어떤 날에 백업을 했는지 여부를 그 날짜에 해당하는 디렉토리가 있는지 여부만으로 쉽게 확인할 수 있을 것입니다.

backup_ver2.py 로 저장하세요:

실행 결과:

동작 원리

많은 부분은 이전과 그대로입니다. 변경된 부분은 주 백업 디렉토리 안에 그 날짜에 해당하는 디렉토리가 있는지 여부를 os.path.exists 함수로 확인하는 부분입니다. 해당 디렉토리가 없으면, os.mkdir 함수를 통해 디렉토리를 새로 만듭니다.

세 번째 프로그램

두 번째 프로그램도 잘 동작했지만, 백업을 많이 하고 싶을 때, 많은 백업 파일이 생성되므로 어떤 파일이 어떤 것의 백업인지 구분하기가 너무 어려웠습니다! 예를 들어, 어떤 문서나 프로그램에 큰 변화를 주었을 때 그 내용을 zip 파일의 이름에 추가로 달아 주면 좋을 것 같습니다. 이 문제는 zip 파일을 생성할 때 뒤에 사용자 정의 꼬리말을 달아 주는 기능을 추가하면 쉽게 해결될 것입니다.

WARNING: 아래 프로그램은 동작하지 않으니 놀라지 마시고 쭉 따라오시기 바랍니다. 이를 통해 뭔가를 배울 것입니다.

backup_ver3.py 로 저장하세요:

실행 결과:

동작(하지 않는) 원리

이 프로그램은 동작하지 않습니다! 실행시켜보면 구문 오류가 있다는 메시지가 출력되며, 이것은 위 스크립트가 파이썬의 문법 규칙을 만족하지 않는다는 것을 의미합니다. 파이썬이 출력해준 오류 메시지를 확인해 보면 어디에서 이 오류가 발생했는지를 알려 줍니다. 따라서 그 줄부터 디버깅 을 시작해 봅시다.

자세히 살펴보면, 한 개의 논리적 명령줄이 두개의 물리적 명령줄로 나뉘어 있지만 두 개의 물리적 명령줄이 사실 하나의 명령줄이라는 것을 파이썬에게 알려줄만한 뭔가가 누락되어 있습니다. 여기서 파이썬은 더하기 연산자 (+)를 발견했으나 그 논리적 명령줄에 피연산자가 없음을 발견하고는 어떻게 해야 할지 모르는 상황에 처하게 된 것입니다. 이 경우, 두 물리적 명령줄을 하나로 연결해 주기 위해서는 맨 뒤에 백슬래시를 추가해 주어야 한다고 배웠으므로 누락된 백슬래시를 추가해 줍니다. 프로그램에서 문제를 찾고 수정 하는 이러한 과정을 버그 수정 이라고 합니다.

네 번째 프로그램

backup_ver4.py 로 저장하세요:

실행 결과:

동작 원리

이제 프로그램이 잘 동작합니다! 이제 세 번째 프로그램을 작성할 때 추가했던 사항들에 대해 살펴보도록 합시다. 먼저 `input` 함수를 통해 사용자의 꼬릿말을 입력받은 후, 사용자가 뭔가를 입력했는지 여부를 `len` 함수를 통해 확인합니다. 만약 사용자가 아무것도 입력하지 않고 `enter` 키를 입력한 경우 (아마도 특별한 꼬릿말이 필요 없는 일상적인 백업을 할 경우에 해당될 것입니다), 이전과 동일하게 처리합니다.

그러나 사용자가 꼬릿말을 입력한 경우에는, zip 파일명을 생성할 때 뒤에 이 꼬릿말을 붙여 주고 `.zip` 확장자를 붙여 줍니다. 여기서 사용자가 입력한 꼬릿말에 포함된 공백 문자를 모두 밑줄로 치환하였는데, 이것은 나중에 파일들을 관리할 때 공백 문자가 없는 편이 관리가 더 쉽기 때문입니다.

더 많은 개선점

아마 네 번째 프로그램은 많은 경우에 만족스럽게 사용될 수 있겠지만, 언제나 개선할 사항은 넘쳐납니다. 예를 들면, 사용자가 `-v` 옵션을 지정하면 zip 명령의 상세함(verbosity) 단계를 지정하게 하여 프로그램이 실행될 때 처리되는 사항을 더 상세히 화면에 출력해주도록 할 수도 있고, `-q` 옵션을 통해 아무 출력 없이 조용히(quiet) 프로그램이 실행되도록 할 수도 있습니다.

또 다른 가능한 개선사항은 추가로 백업할 파일들이나 디렉토리들을 명령줄로부터 넘겨받아 함께 백업하게 하는 것을 생각해 볼 수 있습니다. 이러한 추가 파일들의 이름은 `sys.argv` 리스트를 이용하면 넘겨받을 수 있을 것이고, 이것을 `list` 클래스의 `extend` 메소드를 이용하여 `source` 리스트 뒤에 추가해 줄 수 있을 것입니다.

생각해볼 수 있는 가장 중요한 개선사항은 `os.system` 을 사용하지 않고 파이썬에서 제공되는 내장 모듈인 zipfile 이나 tarfile 을 이용하여 압축 파일을 생성하는 것입니다. 이들은 표준 라이브러리에 포함되어 있으며 zip 프로그램과 같은 추가 프로그램 등을 설치하지 않고서도 프로그램이 동작할 수 있도록 해 줄 수 있을 것입니다.

이 예제에서는 순전히 교육적인 목적에서 `os.system` 을 이용하여 백업 파일을 생성하였는데, 이것은 누구나 알아볼 수 있을 만큼 간단한 프로그램을 구현할 수 있으면서도, 또 실제로 사용할만 한 프로그램을 제작할 수 있기도 하기에 사용한 것입니다.

자, 이제 `os.system` 을 호출하지 않고 zipfile 모듈을 사용하여 다섯번째 프로그램을 만들어보지 않겠습니까?

소프트웨어 개발 단계

지금까지 소프트웨어를 개발하면서 여러 단계 들을 거쳐 왔습니다. 이 단계들은 다음과 같이 축약하여 설명할 수 있습니다:

1. 무엇을 만들 것인가? (분석 단계)
2. 어떻게 만들 것인가? (설계 단계)
3. 만들기 (구현 단계)
4. 테스트 하기 (테스트와 디버깅 단계)
5. 실제로 사용하기 (운영 또는 배포 단계)
6. 유지 및 보수하기 (개선 단계)

앞으로 프로그램을 작성할 때 지금까지 여러분이 백업 스크립트를 만들면서 거쳐 왔던 과정을 그대로 따라 하기를 추천합니다. 문제를 분석하고 프로그램을 설계하세요. 구현은 가장 단순한 프로그램으로 시작하세요. 테스트하고 디버그하세요. 한번 사용해보고 제대로 동작하는지 확인해보세요. 이제, 원하는 기능을 추가하고, 만들어보고 테스트해보고 사용해보는 일련의 과정들을 반복하며 프로그램을 개선해 나가세요.

기억하세요:

> 소프트웨어는 성장하는 것이며, 만들어지는 것이 아니다. – Bill de hÓra

요약

지금까지 직접 파이썬 프로그램/스크립트를 만드는 법과 이러한 프로그램을 만들기 위해 거쳐야 하는 여러가지 단계들에 대해 배워 보았습니다. 이 챕터에서 배운 것들을 통해 프로그램을 작성하는 방법을 익혀두면 유용할 것이며, 또 실제 문제를 해결하는 데 파이썬을 사용하는 것에 좀 더 익숙해질 수 있을 것입니다.

다음으로는, 객체 지향 프로그래밍에 대해 다루어 보겠습니다.

객체 지향 프로그래밍

지금까지 프로그램을 작성할 때, 우리는 데이터를 처리하는 명령들의 블록, 즉 함수들의 조합으로 프로그램을 구성하였습니다. 이러한 설계 방식을 절차 지향 프로그래밍 기법이라고 부릅니다. 이와 달리 데이터와 기능을 객체라고 불리우는 것으로 묶어서 프로그램을 구성하는 또 다른 기법이 있습니다. 이것을 객체 지향 프로그래밍 기법이라고 부릅니다. 아마도 대부분의 시간 동안 여러분은 절차 지향 프로그래밍 기법을 통해 프로그램을 작성하게 되겠지만, 큰 프로그램을 작성할 때나 객체 지향 기법을 이용하는 것이 더 편리하게 활용될 수 있는 문제를 해결해야 할 경우 이러한 객체 지향 프로그래밍 기법을 활용할 수 있습니다.

객체 지향 프로그래밍에서는 클래스와 객체라는 두 주인공이 있습니다. **클래스**는 어떠한 새로운 형식 을 정의하는 것이며, **객체** 란 클래스의 실현체, 다른 말로 **인스턴스** 를 의미하는 것입니다. 이것을 다시 표현하면 여러분이 `int` 라는 형식의 변수를 만들 수 있다는 것으로, 이것은 곧 정수형을 저장하는 변수는 `int` 클래스의 인스턴스 (객체) 를 변수에 할당하는 것이라고도 말할 수 있습니다.

> **정적 언어 프로그래머들을 위한 주석**
>
> 파이썬에서는 정수형조차도 객체로 다루어집니다 (in't 클래스의 객체입니다). C++이나 Java (버전 1.5 미만)처럼 정수형이 자체 기본 형식들 중 하나로 다루어지는 것과는 다릅니다.
>
> `help(int)` 를 입력하여 정수형 클래스에 대해 좀 더 자세히 알아보시기 바랍니다.
>
> C# 이나 Java 1.5 프로그래머들은 아마 이것이 boxing 과 unboxing 의 개념과 비슷하다는 점을 눈치채셨을 것입니다.

객체는 그 객체에 내장된 일반적인 변수들을 사용하여 데이터를 저장할 수 있습니다. 이 때 객체 혹은 클래스에 소속된 변수들을 **필드(field)** 라고 부릅니다. 객체는 또한 내장된 함수를 이용하여 어떤 기능을 갖도록 할 수 있는데 이것을 클래스의 **메소드(method)** 라고 부릅니다. 이들을 기존의 변수 및 함수와 달리 명칭을 구별하여 부르는 것이 중요한데, 이는 일반적인 변수와 함수와 달리 이들은 클래스나 객체에 소속되어 있는 대상들을 의미하는 것이기 때문입니다. 또, 이러한 필드와 메소드들을 통틀어 클래스의 **속성(attribute)** 이라 부릅니다.

필드는 두 가지 종류가 있습니다. 하나는 클래스의 인스턴스/객체에 내장되어 있는 것이고, 또 하나는 클래스 자체에 내장되어 있는 것입니다. 각각을 **인스턴스 변수** 와 **클래스 변수** 라 부릅니다.

클래스는 `class` 키워드를 통해 생성됩니다. 클래스의 필드와 메소드는 그 아래 들여쓰기 된 블록에 차례로 정의됩니다.

`self` 에 대하여

클래스 메소드는 일반적인 함수와 딱 한 가지 다른 점이 있는데, 그것은 메소드의 경우 매개 변수의 목록에 항상 추가로 한 개의 변수가 맨 앞에 추가되어야 한다는 점입니다. 또한 메소드를 호출할 때 이 변수에는 우리가 직접 값을 넘겨주지 **않으며**, 대신 파이썬이 자동으로 값을 할당합니다. 이 변수에는 현재 객체 자신의 참조가 할당되며, 일반적으로 이 변수는 `self` 라고 이름을 짓습니다.

사실 이 변수의 이름은 마음대로 지을 수 있지만, `self` 라는 이름을 사용할 것을 강력히 권합니다. 이것은 일종의 약속이며, 다른 이름을 사용하는 것은 다른 프로그래머들에게 눈총을 받는 일이 될 수 있기 때문입니다. `self`라는 표준적인 이름을 사용하면 여러분의 프로그램을 읽는 사람들로부터 이것이 바로 객체 자신의 참조를 의미함을 쉽게 알아보게 할 수 있고, 파이썬에 특화된 몇몇 IDE (Integrated Development Environment) 의 경우 이 이름을 활용하여 프로그래머에게 도움을 줄 수 있는 등 여러가지 장점이 있습니다.

> **C++/Java/C# 프로그래머를 위한 주석**
>
> 파이썬의 `self` 는 C++ 의 `this` 포인터와 같은 것이며, Java와 C# 의 `this` 참조와 같습니다.

아마 여러분은 파이썬이 `self` 에 어떻게 값을 할당하는 것인지 그리고 정말로 값을 직접 할당할 필요가 없는지가 궁금할 것입니다. 이해를 돕기 위해 예를 하나 들어 보겠습니다. 여러분이 `MyClass` 라는 클래스를 생성했고, 이 클래스의 객체를 `myobject` 라는 이름으로 생성했다고 해 봅시다. 이제 이 객체의 메소드를 호출할 때는 `myobject.method(arg1, arg2)` 와 같이 하며, 이것은 파이썬에 의해 자동적으로 `MyClass.method(myobject, arg1, arg2)` 의 형태로 바뀌게 됩니다. 이것이 `self` 가 동작하는 원리에 대한 모든 것입니다.

따라서 이것은 아무런 인수도 넘겨받지 않는 메소드를 정의할 때에도, self 라는 하나의 인수는 항상 추가해 주어야 한다는 것을 의미합니다.

클래스

가장 단순한 클래스의 예시가 아래 예제에 나타나 있습니다 (oop_simplestclass.py 로 저장하세요):

실행 결과:

동작 원리

먼저 class 문을 사용하여 새로운 클래스를 생성하였고 적당한 이름을 지어 주었습니다. 그 아래로는 들여쓰기 된 새로운 블록이 시작되며 이 블록은 클래스의 몸체를 구성합니다. 위 예제의 경우에는 pass 문으로 해당 블록이 빈 블록임을 나타내 주었습니다.

다음으로, 이 클래스의 이름 뒤에 괄호를 열고 닫아 주어 클래스의 객체/인스턴스를 만들었습니다 (다음 섹션에서 객체 초기화 에 대해 좀 더 자세히 배울 것입니다). 객체가 잘 생성되었는지 확인해 보기 위해, 정의한 변수명을 입력하여 결과를 확인해 봅니다. 그러면 이 객체는 __main__ 모듈에 속한 Person 클래스의 인스턴스임을 알 수 있습니다.

또 객체가 메모리상에 저장된 컴퓨터 주소값이 함께 반환되는 것을 확인하시기 바랍니다. 컴퓨터마다 그 객체를 저장하기 위한 공간이 위치한 곳이 다를 것이므로 컴퓨터마다 이 값은 다르게 출력될 것입니다.

메소드

앞서 클래스/객체는 메소드를 가질 수 있으며, 메소드는 추가된 self 변수를 제외하고는 함수와 똑같다는 것에 대해 이야기했습니다. 아래 예제를 통해 확인해 봅시다 (oop_method.py 로 저장하세요):

실행 결과:

동작 원리

위 예제는 self 가 어떻게 동작하는지 보여줍니다. 여기서 say_hi 메소드는 아무 매개 변수도 넘겨받지 않지만 함수 정의에 self 를 가지고 있음을 확인하시기 바랍니다.

__init__ 메소드

파이썬의 클래스에는 여러가지 특별한 메소드 이름이 존재합니다. 우선 그 중 __init__ 메소드의 중요성에 대해 알아보겠습니다.

__init__ 메소드는 클래스가 인스턴스화 될 때 호출됩니다. 따라서 이 메소드는 객체가 생성될 때 초기화가 필요한 경우, 즉, 객체에 초기값을 할당해 주어야 할 경우에 유용하게 사용됩니다 . 여기서 init의 앞과 뒤에 있는 밑줄은 두 번씩 입력해야 한다는 점에 유의하시기 바랍니다.

예제 (oop_init.py 로 저장하세요):

실행 결과:

동작 원리

먼저 매개 변수 name 을 넘겨 받는 __init__ 메소드를 정의합니다 (물론 self를 포함하여 정의합니다). 그리고, name 이라는 필드를 생성합니다. 이 때 두 개의 다른 변수의 이름으로 'name' 이라는 동일한 이름을 지정해 주었다는 점에 주목하시기 바랍니다. 이것이 문제가 되지 않는 이유는, 이 중 하나는 "self" 라 칭해지는 객체에 내장된 것으로 self.name 의 형태로 사용되며, 또 하나인 name 은 지역 변수를 의미하는 것으로 사용되기 때문입니다. 이렇게 프로그램 상에서 각각을 완전하게 구분할 수 있으므로 혼란이 일어나지 않습니다.

Person 클래스로부터 새로운 인스턴스 p 를 생성하기 위해, 먼저 클래스의 이름을 적고 괄호 안에 인수를 적어 줍니다: p = Person('Swaroop').

위 예제에서 중요한 것은, 우리가 init 메소드를 직접 호출해 주지 않는다는 점입니다.

이제, say_hi 메소드에서처럼 객체 내부에서 self.name 필드를 사용할 수 있습니다.

클래스 변수와 객체 변수

앞서 클래스와 객체의 기능적 부분, 즉 메소드에 대해 설명했습니다. 이제 객체의 데이터의 경우는 어떻게 해야 하는지 배워봅시다. 데이터, 즉 필드는 일반적인 변수와 다를 것이 없으나 딱 한 가지, 그 클래스 혹은 객체의 **네임스페이스** 에 묶여 있다는 점이 다릅니다. 이것은 필드의 이름은 그 클래스 혹은 객체 내부에서만 의미가 있음을 의미합니다. 그래서 이것을 이름이 통용되는 공간이라고 하여 네임스페이스 라고 부릅니다.

필드 에는 두 종류가 있는데, 클래스 변수와 객체 변수입니다. 각각은 그것을 _소유_하고 있는 대상이 클래스인지 객체인지에 따라 구분됩니다.

클래스 변수 는 공유됩니다. 즉, 그 클래스로부터 생성된 모든 인스턴스들이 접근할 수 있습니다. 클래스 변수는 한 개만 존재하며 어떤 객체가 클래스 변수를 변경하면 모든 다른 인스턴스들에 변경 사항이 반영됩니다.

객체 변수 는 클래스로부터 생성된 각각의 객체/인스턴스에 속해 있는 변수입니다. 이 경우에는, 각각의 객체별로 객체 변수를 하나씩 따로 가지고 있으며, 서로 공유되지 않고 각 인스턴스에 존재하는 같은 이름의 필드끼리 서로 어떤 방식으로든 간섭되지 않습니다. 아래 예제를 통해 좀 더 자세히 알아봅시다. (`oop_objvar.py` 로 저장하세요):

실행 결과:

동작 원리

예제가 좀 길지만, 클래스/객체 변수의 이해를 돕도록 구성되어 있습니다. 여기서 `population` 은 `Robot` 클래스에 속해 있는 클래스 변수입니다. 또, `name` 변수는 객체에 소속되어 있는 (즉 `self` 를 이용하여 접근할 수 있는) 객체 변수입니다.

따라서, `population` 클래스 변수는 `Robot.population` 과 같이 사용하며 `self.population` 과 같이 사용하지 않습니다. 반면 객체 변수 `name` 은 그 객체 안에서 `self.name` 과 같이 사용됩니다. 이러한 클래스 변수와 객체 변수의 차이점에 유의하시기 바랍니다. 또, 클래스 변수와 같은 이름을 가진 객체 변수는 클래스 변수를 감춘다는 점을 기억하세요!

`Ropot.population` 대신에 `self.__class__.population` 이라고도 사용할 수 있는데 이것은 모든 객체는 그 객체를 생성하는 데 사용되었던 클래스를 `self.__class__` 속성을 통해 참조하고 있기 때문입니다.

메소드 `how_many` 는 객체에 소속되어 있지 않고 클래스에 소속되어 있는 메소드입니다. 여기서 우리가 해당 클래스의 어떤 부분까지 알아야 할 지에 따라 메소드를 클래스 메소드 (`classmethod`) 로 정의할지 스태틱 메소드 (`staticmethod`) 로 정의할지 결정할 수 있습니다. 여기서는 클래스 변수를 사용할 것이므로, 클래스 메소드 를 사용합시다.

여기서는 `how_many` 메소드를 클래스 메소드로 만들어 주기 위해 데코레이터 를 이용하였습니다.

데코레이터는 어떤 일을 추가로 해 주는 더 큰 범위의 함수로 다른 함수를 감싸주는 것이라고 생각하면 됩니다. 즉, `@classmethod` 데코레이터는 아래처럼 호출하는 것과 같습니다:

```
how_many = classmethod(how_many)
```

`__init__` 메소드는 `Robot` 의 인스턴스를 초기화시킬 때 사용됩니다. 이 메소드를 통해 로봇이 하나 추가될 때마다 로봇의 개수를 의미하는 변수 `population` 을 1 씩 증가시켜 줍니다. 또한 각 생성된 객체별로 객체 변수 `self.name` 의 값을 따로따로 지정해 주었습니다.

객체에 속해 있는 변수와 메소드에 접근하기 위해서는 반드시 `self` 를 사용해야 한다는 점을 기억하시기 바랍니다. 이것을 다른 말로 속성 참조(attribute reference) 라 부릅니다.

프로그램을 살펴보면 메소드에 정의된 것 처럼 클래스에도 DocString 이 정의되어 있는 것을 보실 수 있습니다. 마찬가지로 이 DocString도 `Robot.__doc__` 을 통해 접근할 수 있고, 또 메소드의 DocString 은 `Robot.say_hi.__doc__` 과 같이 접근할 수 있습니다.

`die` 메소드가 실행되면, 간단히 `Robot.population` 을 하나 줄여 줍니다.

모든 클래스 멤버는 클래스 외부로 공개되어 있습니다. 한가지 예외가 있는데, 여러분이 밑줄 두 개로 시작하는 데이터 멤버를 정의하면, 즉 예를 들어 `__privatevar` 와 같이 하면, 파이썬이 이것을 클래스 외부로 드러나지 않도록 숨겨 줍니다.

이것은 클래스나 객체에 속해 있는 어떤 변수에나 적용됩니다. 클래스와 객체에 정의된 모든 이름은 밑줄로 시작하지 않는 이상 외부로 공개하고 다른 클래스나 객체에서 불러와 사용할 수 있도록 하는 규칙을 따르는 것이 좋습니다. 그러나 이것은 파이썬에서 강제하는 것이 아니며 (밑줄 두 개로 시작하는 경우를 제외하고) 프로그래머들끼리의 약속입니다.

C++/Java/C# 프로그래머를 위한 주석

모든 클래스 멤버는 (데이터 멤버를 포함하여) public 이며, 파이썬의 모든 메소드는 virtual 입니다.

상속

객체 지향 프로그래밍의 또 다른 큰 장점은 코드를 **재사용** 할 수 있다는 것이며, 이를 위한 한 가지 방법으로 **상속** 이 사용됩니다. 상속은 클래스 간의 형식과 세부 형식 을 구현하는 것이라고 생각해볼 수 있습니다.

어러분이 어떤 대학의 교수들과 학생들의 명부를 작성하는 프로그램을 작성한다고 해 봅시다. 이 때 교수와 학생 모두 공통적으로 이름, 나이, 주소 등의 성질을 가지고 있을 것이며, 교수에만 적용되는 성질로는 연봉, 과목, 휴가 등이 있을 것이고, 학생에만 적용되는 성질로는 성적, 등록금 등이 있을 것입니다.

따라서 여러분은 각각의 경우에 두 독립적인 클래스를 만들 수 있겠지만, 이 경우 각각의 공통적인 성질 또한 각각의 클래스에 두 번씩 반복해서 정의해 주어야 할 것입니다. 매우 불편합니다.

더 나은 방법은 `SchoolMember` 라는 이름으로 공통 클래스를 생성한 뒤 교수와 학생 클래스를 이 클래스로부터 상속 받아 생성하는 것입니다. 이 경우 상속받은 클래스들은 이를테면 상위 형식 (클래스) 의 세부 형식이 되는 것이고, 따라서 이 세부 형식에 각 상황에 맞는 세부적인 성질들을 추가해 줄 수 있는 것입니다.

이러한 접근 방식에는 많은 장점이 있습니다. 그 중 한 장점은 우리가 `SchoolMember` 에 새로운 기능을 추가하거나 혹은 있던 기능을 수정하게 되면, 그 하위 클래스인 교수와 학생 클래스에도 이러한 변경 사항이 자동으로 추가된다는 점입니다. 예를 들어 교수와 학생들에게 새로 출입증을 발급해야 할 경우 SchoolMember 클래스에 이를 적용해 주기만 하면 되는 것이죠. 반대로 하위 클래스에 적용된 변경 사항은 다른 하위 클래스에 적용되지 않습니다. 또 다른 장점은 여러분이 예를 들어 대학에 소속된 사람들의 모든 숫자를 파악해야 한다고 할 경우 교수와 학생 객체를 `SchoolMember` 객체로써 참조하여 사용할 수 있다는 점입니다. 이것을 **다형성** 이라고 부르는데, 하위 형식이 부모 형식을 필요로 하는 어떤 상황에서건 이를 대신하여 사용될 수 있다는 것을 의미합니다. 즉, 자식 클래스의 객체를 부모 클래스의 인스턴스인 것처럼 다루어질 수 있습니다.

따라서 상속을 활용하면 부모 클래스의 코드를 재사용할 수 있고 서로 완전히 독립적인 클래스들을 정의했을 때처럼 각각 다른 클래스에 이를 또 반복해서 써 줄 필요가 없다는 것입니다.

이 상황에서의 `SchoolMember` 클래스를 **베이스 클래스** 혹은 **슈퍼클래스** 라고 부릅니다. 또 `Teacher` 와 `Student` 클래스는 **파생 클래스** 혹은 **서브클래스** 라고 부릅니다.

다음 프로그램을 예제로 살펴 보겠습니다(`oop_subclass.py` 로 저장하세요):

실행 결과:

동작 원리

상속을 사용하기 위해, 예제에서 정의된 여러 베이스 클래스들의 이름들이 클래스 정의의 클래스명 옆 튜플에 지정됩니다 (예: `class Teacher(SchoolMember)`). 다음으로, 베이스 클래스의 `__init__` 메소드가 `self` 변수를 이용하여 명시적으로 호출되어, 객체의 베이스 클래스에 정의된 초기화 명령이 실행됩니다. 이 예제에서 `Teacher` 와 `Student` 서브클래스 각각 `__init__` 메소드를 구현할 경우, 이런 식으로 베이스 클래스의 생성자를 명시적으로 호출해 주지 않으면 파이썬은 베이스 클래스 `SchoolMember` 의 생성자를 자동으로 호출해 주지 않는다는 점을 기억하시기 바랍니다.

반면 서브클래스에서 `__init__` 메소드를 구현해 주지 않는 경우, 베이스 클래스의 생성자는 자동으로 호출되게 됩니다.

여기서 `SchoolMember` 클래스의 `tell` 메소드를 사용할 때, `Teacher` 나 `Student` 의 인스턴스들을 `SchoolMember` 의 인스턴스로써 사용할 수 간주할 수 있고 `SchoolMember` 의 `tell` 메소드를 `Teacher.tell` 또는 `Student.tell` 과 같이 접근할 수 있습니다. 그러나, 이 대신 또 다른 `tell` 메소드를 각각의 서브클래스에 구현해 주어 (이 때 `SchoolMember` 의 `tell` 메소드를 활용하여 일부를 구현했습니다) 각 서브클래스에 맞는 형태로 좀 더 다듬어 줄 수 있습니다. 이렇게 한 뒤 `Teacher` 의 `tell` 메소드를 찾으면 파이썬은 서브클래스와 슈퍼클래스에서 `tell` 메소드를 찾게 됩니다. 이 때, 파이썬은 항상 서브클래스에서 먼저 해당 메소드를 찾으며, 만약에 그 메소드를 찾지 못할 경우 그 서브클래스의 베이스 클래스에서 해당 메소드를 찾게 되며, 이는 클래스를 정의할 때 지정한 베이스 클래스들의 튜플에 지정된 순서대로 해당 메소드를 찾아 나가는 것과 같습니다 (이 예제에서 각 서브클래스는 한 개의 베이스 클래스만을 가지지만, 서브클래스는 여러 개의 베이스 클래스를 상속받을 수도 있습니다).

여기서 한 개 이상의 클래스를 상속받는 경우를 **다중상속** 이라고 부릅니다.

슈퍼 클래스의 `tell()` 메소드에서 print 문에 사용된 `end` 파라미터는 그 다음에 출력 될 내용을 새로운 줄에 출력하지 말고 그 줄에 이어서 출력하라는 것을 의미합니다. 이것은 `print` 가 `\n` (줄바꿈) 문자를 마지막에 입력하지 않게 하는 것과 같습니다.

요약

지금까지 클래스와 객체의 다양한 속성에 대해 알아 보았으며, 또 통용되는 용어들에 대해서도 알아 보았습니다. 또한 객체 지향 프로그래밍을 사용할 때의 장점과 주의해야 할 점에 대해서도 알아 보았 습니다. 파이썬은 고도의 객체 지향 언어로 이러한 개념을 잘 익혀 두면 여러분이 좋은 파이썬 프로그래머로서 꾸준히 성장할 수 있게 될 것입니다.

다음으로, 파이썬에서의 입/출력을 다루는 법과 파일을 읽고 쓰는 법에 대해 배워 보겠습니다.

입력과 출력

프로그램을 만들다 보면 간혹 프로그램이 사용자와 상호 작용을 해야 할 때가 있을 것입니다. 예를 들어, 사용자로부터 뭔가를 입력받고 처리 결과를 출력해 주는 것 같은 일이 필요할 때입니다. 파이썬에서는 이를 위해 각각 `input()` 함수와 `print` 문을 사용합니다.

결과를 출력해주기 위해서는 `str` (문자열) 클래스가 제공하는 여러 메소드를 사용할 수도 있습니다. 예를 들면, rjust 메소드를 사용하여 출력될 문자열이 특정 폭의 문자열 안에서 오른쪽 정렬되도록 할 수 있습니다. `help(str)` 을 실행하여 이들에 대해 자세히 알아보시기 바랍니다.

또 다른 입/출력의 형식은 파일을 다루는 것입니다. 파일을 생성하고, 읽고, 쓰는 것은 많은 프로그램에서 중요한 부분을 차지하고 있으며 이 챕터에서는 이러한 기능에 대해 자세히 알아보게 될 것입니다.

사용자로부터 입력받기

`io_input.py` 로 저장하세요:

실행 결과:

동작 원리

문자열을 뒤집기 위해서는 를 사용합니다. 앞서 보았듯이 열거형의 슬라이스 기능을 활용하여 `seq[a:b]` 와 같은 코드를 통해 위치 `a` 부터 위치 `b` 까지 문자열을 얻어올 수 있습니다. 슬라이스 숫자에 세 번째 인수를 넘겨 주어 슬라이스 스텝 을 지정해줄 수 있습니다. 스텝을 지정하지 않으면 기본값 `1` 이 지정되며, 이 경우 지정된 문자열을 차례로 슬라이스 하는 것을 의미합니다. 음의 스텝을 지정하면 열거형의 마지막부터 반대 방향으로 슬라이스가 진행되며, 예를 들어 `-1` 을 지정하면 뒤집혀진 문자열이 반환됩니다.

`input()` 함수는 인수로 넘겨받은 문자열을 화면에 표시해 줍니다. 그리고 나서는 사용자가 사용자가 무언가를 입력하고 `Enter` 키를 누를 때까지 기다립니다. 사용자가 입력을 마치고 `Enter` 키를 누르면 `input()` 함수는 사용자가 입력한 내용을 문자열로 반환해 줍니다.

이제 이 문자열을 받아서 뒤집어 줍니다 (참고로 여기서 뒤집혀진 문자열이 뒤집혀지지 않았을 때의 문자열과 동일한 경우를 영어로 palindrome 이라고 부릅니다).

연습 문제

어떤 문자열이 palindrome이라고 하는 것은 그 안에 포함된 문장 부호들과 공백 등을 제외한 문자들을 가지고서 판단해야 합니다. 예를 들어, “Rise to vote, sir.” 은 palindrome의 한 예이지만 위 예제 프로그램은 이것은 palindrome이 아니라고 판단할 것입니다. 위 프로그램을 고쳐서 이러한 문자열들을 palindrome으로 인식할 수 있는 프로그램을 작성해 보시기 바랍니다.

힌트가 필요하시면 주석을 읽어 보세요… [2]

[2] 각 사람의 정보를 담는 클래스를 하나 만드세요. 그리고 각 사람의 이름을 키로 하고 사람의 객체를 값으로 하는 사전에 데이터를 저장해 둡니다. 그리고 pickle 모듈을 사용하여 이 객체를 여러분의 하드 디스크에 저장해 두세요. 또 사전의 내장 메소드를 활용하여 사람을 추가하고 삭제하거나 수정하는 기능을 구현하세요.

파일 입/출력

입/출력을 위해 파일을 열고 사용하려면 `file` 클래스의 객체를 생성한 후 `read`, `readline`, `write` 와 같은 메소드들을 적절히 활용하면 됩니다. 파일을 열때 파일을 읽는 모드와 쓰는 모드를 따로 지정해 줄 수 있습니다. 마지막으로 파일을 읽거나 쓰는 일을 모두 마친 후에는, `close` 메소드를 호출하여 파이썬에게 파일 사용을 마쳤으니 이제 파일을 닫으라고 해 주어야 합니다.

예제 (`io_using_file.py` 로 저장하세요):

실행 결과:

동작 원리

먼저, 내장 함수 `open` 을 이용하여 파일을 열어 줍니다. 이 때 파일을 어떤 용도로 사용할 것인지도 함께 지정해 줍니다. 각 모드로는 읽기 모드 (`'r'`), 쓰기 모드 (`'w'`), 덧붙임 모드 (`'a'`) 등이 있습니다. 또한 우리가 다룰 파일을 일반적인 텍스트 모드 (`'t'`) 로 다룰 지 또는 바이너리 모드 (`'b'`) 로 다룰 지 여부도 함께 지정해 줄 수 있습니다. 이외에도 여러가지 다른 모드들이 있으며, help(open) 을 통해 그 목록을 확인해 볼 수 있습니다. 모드에 아무것도 지정하지 않으면, open() 은 기본적으로 파일을 텍스트('t'ext) 파일을 읽는 ('r'ead) 모드로 파일을 열어 줍니다.

위 예제에서는 먼저 파일을 쓰기/텍스트 모드로 열고 파일 객체의 `write` 메소드를 사용하여 파일에 문자열 변수 `poem` 에 들어 있는 텍스트를 써준 후 `close` 로 파일을 닫아 줍니다.

다음으로는, 똑같은 파일을 이번에는 읽기 모드로 엽니다. 이 때 아무 모드도 지정하지 않았는데 이렇게 하면 기본값인 '읽기/텍스트 모드' 가 지정됩니다. 파일을 연 후에는 반복문을 이용하여 파일의 `readline` 메소드를 통해 파일의 내용을 한 줄씩 읽어옵니다. 이 메소드는 파일 내용을 읽다가 줄바꿈 문자를 만날 때까지 한 줄을 읽어서 그 모든 내용을 반환해 줍니다. 만약 빈 문자열이 반환되었을 경우, 이것은 파일의 끝임을 의미하는 것이므로 'break' 문을 통해 반복문을 빠져나옵니다.

마지막으로, `close` 문으로 파일을 닫습니다.

이제 `readline` 의 출력 결과를 통해 예제 프로그램이 `poem.txt` 파일에 올바른 내용을 쓰고 읽어들였는지를 다시 한번 확인해 보시기 바랍니다.

피클 (Pickle)

파이썬은 `pickle` 이라고 불리우는 모듈을 기본으로 제공하는데, 이것은 어떤 파이썬 객체이든지 파일로 저장해 두었다가 나중에 불러와서 사용할 수 있게 하는, 즉 객체를 영구히 저장해 두는 방법을 제공해 주는 모듈입니다.

예제 (`io_pickle.py` 로 저장하세요):

실행 결과:

동작 원리

파일에 객체를 저장하기 위해서 먼저 open 문을 이용하여 **쓰기 (w)** 및 **바이너리 (b)** 모드로 파일을 열어 준 후 `pickle` 모듈의 `dump` 함수를 호출하여 줍니다. 이 과정을 피클링 (pickling) 이라고 합니다.

다음으로 `pickle` 모듈의 `load` 함수를 이용하여 파일에 저장된 객체를 불러옵니다. 이 과정을 언피클링(unpickling) 이라고 합니다.

유니코드

지금까지 우리가 문자열을 화면이나 파일에 쓰거나 읽을 때 영어 알파벳 문자들만을 주로 이용해 왔습니다. 여러분이 영어가 아닌 다른 언어로 된 문자를 읽고 쓰게 될 경우, 파이썬은 이러한 문자들을 유니코드로 표현하는데 (이에 대한 좀 더 자세한 정보는 이 섹션의 마지막 부분에 있는 여러 글들을 읽어 보세요), 파이썬 3은 기본적으로 문자열 변수들에 저장된 값들 (예를 들어 작은 따옴표, 큰 따옴표, 따옴표 세 개를 활용한 문자열들)을 모두 유니코드로 표현합니다. 이 아래부터는 파이썬 2에 해당하는 설명입니다.

> NOTE: 만약 여러분이 파이썬 2를 사용중이며 영어가 아닌 문자를 표현해야 할 경우 unicode 형식을 이용할 필요가 있으며, 이것은 문자 `u` 를 앞에 붙여 주어 지정해 줍니다. 예를 들어 `u"hello world"` 와 같이 합니다.

```
>>> "hello world"
'hello world'
>>> type("hello world")
<class 'str'>
>>> u"hello world"
'hello world'
>>> type(u"hello world")
<class 'str'>
```

특히 데이터가 인터넷을 통해 전송될 때, 이러한 데이터는 컴퓨터가 쉽게 알아들을 수 있는 형식인 바이트 형식을 통해 전송됩니다. 이 때, 유니코드 (파이썬이 문자열을 저장하는 방식) 에서 바이트로 변환하는 것을 이라고 합니다. 현재 많이 쓰이는 인코딩의 예는 UTF-8 입니다. open 함수에 간단히 키워드 인자를 하나 넘겨주는 것을 통해 UTF-8 로 파일을 읽고 쓸 수 있습니다:

동작 원리

이 예제에서는 io.open 을 활용하며, 첫 번째 open 문에 encoding 인자를 넘겨 줌으로써 메시지를 인코딩하고, 두번째 open 문에서는 이를 디코딩합니다. 이 때 encoding 인자는 텍스트 모드로 파일을 열었을 때만 사용 가능하다는 점에 유의하세요.

유니코드 리터럴 상수를 활용할 때에는 (즉, 문자열의 앞에 u를 달아주어야 할 때), 파이썬에게 이 파일이 UTF-8을 쓴다는 점을 알려 주어야 합니다. 이렇게 하려면, `# encoding=utf-8` 이라는 주석을 프로그램의 맨 윗줄에 달아 주면 됩니다.

다음 글들을 읽어 보시면 더 많은 것들을 배울 수 있을 것입니다:

- "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets"
- Python Unicode Howto
- Pragmatic Unicode talk by Nat Batchelder

요약

지금까지 여러가지 종류의 입/출력 및 파일을 다루는 법, 그리고 pickle 모듈을 다루는 법과 유니코드에 대해 배워 보았습니다.

다음으로는 예외 처리의 개념에 대해 배워봅시다.

예외 처리

예외 (Exception) 란 말 그대로 프로그램에서 벌어지는 예외적인 상황을 뜻합니다. 예를 들자면 여러분이 파일을 읽고자 할 때 그 파일이 존재하지 않는 경우라던지, 또는 프로그램이 한참 실행중인데 그 파일을 갑자기 지워버렸다던지 하는 경우 등입니다. 이러한 상황을 ***예외 (Exception)** 를 활용하여 처리할 수 있습니다.

비슷하게 여러분의 프로그램에 부적절한 명령문이 있을 경우 어떻게 될까요? 이런 경우 파이썬은 프로그램에 오류 (**error**) 가 있음을 제기 (**raise**) 해 줍니다.

오류 (Error)

간단한 print 함수를 호출하는 상황을 생각해 봅시다. 이 때 print 를 Print 라고 잘못 입력했을 경우 어떻게 될까요 (대/소문자 구분에 유의해 주세요)? 이 경우, 파이썬은 구문 오류를 발생 (raise) 시킵니다.

```
>>> Print("Hello World")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

```
>>> print("Hello World")
Hello World
```

위와 같이 NameError 가 발생되었고 오류가 발생한 위치가 표시됩니다. 이렇게 오류를 처리해 주는 부분을 **오류 핸들러**라 합니다.

예외 (Exception)

이번에는 사용자로부터 뭔가를 입력 받는 것을 시도하는 (try) 경우를 생각해 봅시다. 아래 예제에서의 첫 줄을 입력하고 Enter 를 누릅니다. 여러분의 컴퓨터가 입력을 받기 위해 대기중일 때, 어떠한 것을 입력하는 대신 [ctrl-d] (Mac OS) 또는 [ctrl-z] (Windows) 를 누르고 어떻게 되는지 살펴봅시다 (만약 여러분이 Windows를 사용중인데 둘 다 안 되는 경우 [ctrl-c]를 대신 눌러 보세요. 다만 이 경우 아래와 달리 KeyboardInterrupt 오류가 표시될 것입니다).

```
>>> s = input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

그러면 파이썬은 EOFError 라는 오류를 발생시키는데, 이때 EOF란 파일의 끝 (end of file) 을 의미하며 (파일의 끝은 ctrl-d 에 의해 표현됩니다), 파이썬이 갑자기 파일의 끝이 올 것을 예상하지 못했기 때문에 위와 같은 오류가 발생하는 것입니다.

예외 처리

예외는 try..except 문을 통해 처리할 수 있습니다. 이것은 try 블록 안에 명령을 입력하고, 발생할 수 있는 각 예외 상황을 처리해줄 수 있는 오류 핸들러 를 except 블록에 입력해 주면 됩니다.

예제 (exceptions_handle.py 로 저장하세요):

실행 결과:

동작 원리

이 예제에서는 예외가 발생할 수 있는 명령문들을 try 블록에 넣어 주었으며 오류 또는 예외를 적절하게 처리해 줄 오류 핸들러를 except 절 (또는 블록) 에 넣어 주었습니다. except 절에서는 지정된 한 개의 오류 혹은 예외를 처리할 수도 있고, 괄호로 묶여진 모든 오류나 예외들의 목록을 처리해 줄 수도 있습니다. 만일 오류나 예외를 특별히 지정하지 않은 경우에는 모든 오류 및 예외를 처리하게 됩니다.

이 때 모든 try 절에는 적어도 한 개의 except 절이 있어야 합니다. 아니면 try 블록을 사용할 아무런 이유가 없겠지요?

만약 어떤 오류나 예외든지 이처럼 처리되지 않는 경우, 기본 파이썬 오류 핸들러가 호출되는데 그러면 이에 의해 프로그램의 수행이 중단되며 해당하는 오류 메시지가 출력됩니다. 위 예제에서는 기본적인 파이썬 오류 핸들러가 어떻게 동작하는지 확인해 보았습니다.

또한 try..except 블록에는 추가로 else 절을 붙여줄 수 있습니다. 이 때 else 절은 어떤 예외도 발생하지 않았을 경우 호출됩니다.

다음으로는 예외 객체를 얻어오는 방법과 이를 통해 예외에 대한 추가 정보를 얻어오는 방법에 대해 알아보겠습니다.

예외 발생시키기

raise 문에 오류나 예외의 이름을 넘겨 주는 것을 통해 예외를 직접 발생(raise) 시킬 수 있습니다. 그러면 예외 객체가 throw 됩니다.

이 때 발생시킬 수 있는 오류나 예외는 반드시 직접적으로든 간접적으로든 Exception 클래스에서 파생된 클래스이어야 합니다.

예제 (exceptions_raise.py 로 저장하세요):

실행 결과:

동작 원리

위 예제에서는 ShortInputException 이라고 하는 새로운 예외 형식을 직접 하나 만들어 보았습니다. 여기에는 두 개의 필드가 있습니다. 하나는 length 필드로 주어진 입력의 길이를 의미하며, 또 하나는 atleast 필드로 프로그램이 요구하는 최소한의 길이를 의미합니다.

이제 except 절에서 as 를 이용하여 해당 오류의 클래스를 좀 더 짧은 이름의 변수로 대신하여 사용할 수 있게 해 줍니다. 여기서 새로 정의한 예외 형식에 정의한 필드와 값의 관계는 마치 함수에서의 매개 변수와 인수의 관계와 비슷합니다. 마지막으로 이 오류를 처리해주는 except 절에서는 해당 예외 객체의 length 와 atleast 필드를 이용하여 사용자에게 적절한 결과를 출력해 줍니다.

Try … Finally 문

프로그램이 파일을 읽고 있는 상황을 가정해 봅시다. 이 때 예외가 발생할 경우, 예외의 발생 여부와 상관없이 파일 객체를 항상 닫아 주도록 할 수는 없을까요? 이를 위해 finally 블록을 사용합니다.

아래 프로그램을 exceptions_finally.py 로 저장하세요:

실행 결과:

동작 원리

위 예데는 단순히 파일을 읽는 코드이지만, 파일에서 한 줄을 읽어올 때마다 time.sleep 함수를 호출하여 2초씩 멈추게 하는 인위적인 코드를 집어넣어 프로그램이 천천히 실행되도록 해 주었습니다 (파이썬은 원래 굉장히 빠릅니다). 프로그램이 실행중일 때, ctrl + c 를 눌러 프로그램을 강제로 중단 (interrupt) 시켜 봅시다.

그러면 KeyboardInterrupt 예외가 발생되며 프로그램이 종료됩니다. 그러나 프로그램이 종료되기 전에 finally 절이 실행되므로 열어주었던 파일 객체는 항상 닫히게 됩니다.

여기서 0, None, 빈 열거형 객체 등은 파이썬에서 (False) 에 해당하는 것으로 처리됩니다. 위 예제에서 if f: 를 쓸 수 있는 이유입니다.

또한 여기서 print 문 뒤에 sys.stdout.flush() 를 사용하여 화면에 결과를 곧바로 출력하도록 해 주었습니다.

with 문

try 블록에서 어떠한 시스템 자원을 가져오고 finally 문에서 이를 해제하여 주는 것은 자주 활용되는 패턴입니다. 그렇지만, with 문을 활용하면 이것을 좀 더 깔끔하게 작성해 줄 수 있습니다:

exceptions_using_with.py 로 저장하세요:

동작 원리

위 예제는 이전의 예제와 동일한 결과를 출력합니다. 차이점은 open 함수를 사용할 때 with 문을 사용하였다는 것입니다. 그러면 파일을 직접 닫아 주지 않아도 with open 이 자동으로 파일을 닫아 줍니다.

그러면 with 문은 어떻게 자동으로 이러한 것들을 처리해 주는 것일까요? 우선 with 문은 open 문이 반환해 주는 객체를 받아 오는데, 일단 여기서는 이것을 "thefile" 이라고 해 봅시다.

with 문은 항상 thefile.__enter__ 함수를 호출한 뒤 해당 블록의 코드를 실행하며, 실행이 끝난 후에는 항상 thefile.__exit__ 를 호출합니다.

따라서 finally 블록에서 써 줬어야 했을 코드가 __exit__ 메소드에 의해 자동적으로 이루어지는 것입니다. 따라서 with를 쓰면 매번 try..finally 문을 명시적으로 쓰지 않고도 같은 일을 할 수 있습니다.

이에 대한 좀 더 자세한 설명은 이 책이 다루는 범위를 벗어납니다. 자세한 설명은 PEP 343 을 읽어 보시기 바랍니다.

요약

지금까지 try..except 문과 try..finally 문의 사용법을 배워 보았습니다. 또 사용자 정의 예외 형식을 만드는 법과 예외를 일으키는 법에 대해서도 알아 보았습니다.

다음으로, 파이썬 표준 라이브러리에 대해 알아 보겠습니다.

표준 라이브러리

파이썬 표준 라이브러리는 파이썬을 설치할 때 항상 함께 설치되는 다수의 유용한 모듈들을 말합니다. 파이썬 표준 라이브러리에 익숙해지면 이를 이용해 많은 문제를 좀 더 빠르고 쉽게 해결할 수 있습니다.

지금부터 표준 라이브러리에 포함된 많은 모듈 중에서 자주 사용되는 몇 가지 모듈에 대해 알아볼 것입니다. 파이썬 표준 라이브러리에 포함된 모든 모듈에 대한 자세한 설명은 파이썬과 함께 설치되는 설명서의 '라이브러리 레퍼런스' 섹션 에서 확인해 보실 수 있습니다.

여기서는 유용한 몇 개의 모듈을 다뤄 보겠습니다.

> CAUTION: 이 챕터에서 다루는 내용은 조금 어려울 수 있습니다. 그런 경우 일단 이 챕터를 읽지 말고 넘기세요. 그렇지만 여러분이 파이썬에 좀 더 익숙해지게 되면 이 챕터로 다시 돌아오기를 강력히 권합니다.

sys 모듈

sys 모듈에는 시스템의 기능을 다루는 여러 함수가 들어 있습니다. 예를 들어 sys.argv 리스트에는 명령줄 인수들이 들어 있습니다.

또 sys 모듈을 통해 현재 사용하고 있는 파이썬의 버전을 알 수 있습니다.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

동작 원리

sys 모듈에는 version_info라고 하는 파이썬의 버전 정보가 담겨 있는 튜플이 들어 있습니다. 첫 번째 항목은 주 버전을 의미합니다. 이제 이 정보를 읽어와 사용할 수 있습니다.

logging 모듈

여러분이 디버깅할 때 중간에 변수들의 내용 등을 출력하고 싶거나 실행 시 중요한 메시지를 어딘가에 저장해 두게 하여 여러분의 프로그램이 제대로 실행되고 있는지 확인하고 싶을 때 어떻게 하면 좋을까요? 어떻게 이러한 메시지들을 "어딘가에 저장해" 둘 수 있을까요? 이를 위해 logging 모듈을 사용합니다.

stdlib_logging.py 로 저장하세요:

실행 결과:

이 예제에서는 'test.log' 파일의 내용을 읽기 위해 cat 명령을 활용합니다. 여러분의 명령행 환경에서 cat 명령을 사용할 수 없을 경우, 아무 텍스트 에디터에서나 test.log 파일을 열어서 내용을 확인해 보실 수 있습니다.

동작 원리

위 예제에서는 표준 라이브러리에 있는 세 가지 다른 모듈을 사용하였습니다. 그중 하나는 시스템의 운영 체제와 상호작용할 때 쓰이는 os 모듈이고, 또 하나는 플랫폼(운영 체제라든지)의 정보를 알아 오는 데 사용되는 platform 모듈이며 마지막 하나는 정보를 기록(log) 하는 데 사용되는 logging 모듈입니다.

먼저, platform.platform() 이 반환해주는 문자열을 통해 현재 사용 중인 운영 체제가 무엇인지 알아 옵니다 (이 모듈에 대해 더 자세히 알아보려면 import platform; help(platform) 을 입력하세요). 이제 윈도우 플랫폼인 경우 홈 드라이브 및 홈 폴더를 알아내어 정보를 저장해 둘 파일 이름을 구성합니다. 다른 플랫폼의 경우, 현재 사용자의 홈 폴더만 알면 파일의 전체 경로를 구성해낼 수 있습니다.

다음으로 os.path.join() 함수를 이용하여 이 세 문자열을 하나의 경로 문자열로 합쳐 줍니다. 이때 문자열을 단순히 합치지 않고 이러한 특별한 함수를 이용하여 합쳐 준 것은 합쳐진 최종 경로가 현재 사용 중인 운영 체제의 형식에 맞는

형태로 생성되도록 해 두기 위함입니다. 참고로, 이 예제에서 활용한 `join()` 메소드는 `os` 모듈에 있는 것으로, 이 책의 다른 곳에 활용했던 문자열의 메소드인 `join()` 과는 다른 것입니다.

이제 `logging` 모듈을 이용하여 필요한 기록 사항들을 지정해준 파일에 지정된 형식대로 기록합니다.

마지막으로 각 메시지가 디버깅 정보인지, 단순한 정보인지, 경고 혹은 심각한 메시지인지 등에 따라 다르게 저장해 줍니다. 이 프로그램이 실행되어도 화면에는 아무것도 출력해 주지 않지만, 이 파일의 내용을 확인해 보면 프로그램이 실행되며 어떤 일들이 일어났는지 확인할 수 있습니다.

'금주의 모듈' 시리즈

파이썬 표준 라이브러리에는 다음과 같은 더 많은 모듈이 있습니다. 디버깅(debugging) 모듈, 명령행 옵션 관련 모듈, 정규 표현식 모듈 등등입니다.

파이썬 표준 라이브러리에 대해 좀 더 알아볼 수 있는 최고의 방법은 Doug Hellmann이 쓴 금주의 파이썬 모듈 시리즈 (책으로도 읽을 수 있습니다) 및 파이썬 공식 문서 를 읽는 것입니다.

요약

지금까지 파이썬 표준 라이브러리에 있는 몇 가지 모듈의 기능에 대해 다뤄 보았습니다. 이에 만족하지 말고, 파이썬 표준 라이브러리 문서를 읽고 사용할 수 있는 모든 모듈에 대한 정보를 알아보는 것을 추천해 드립니다.

다음으로는, 우리의 파이썬과 함께 하는 여정을 좀 더 완전하게 해 줄 파이썬의 여러 장점에 대해 알아보겠습니다.

더 많은 것들

지금까지 앞으로 여러분이 사용하게 될 파이썬의 여러 주요한 기능들에 대해 다뤄 보았습니다. 이 챕터에서는, 여러분이 앞으로 파이썬을 사용하면서 추가로 알아두면 좋을 몇 가지를 다뤄 보겠습니다.

튜플 넘기기

함수의 실행 결과로 두 개 이상의 값을 반환하고 싶을 때가 있지 않았나요? 파이썬에서는 할 수 있습니다. 단순히 튜플을 넘겨 주기만 하면 됩니다.

```
>>> def get_error_details():
...     return (2, 'details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'details'
```

위와 같이 `a, b = < >` 과 같이 해 주면 계산식의 결과로 넘어온 튜플이 자동으로 두 값에 알맞게 들어가게 됩니다.

이것을 활용하면 두 변수의 값을 바꾸어야 할 때 다음과 같이 할 수 있습니다:

```
>>> a = 5; b = 8
>>> a, b
(5, 8)
>>> a, b = b, a
>>> a, b
(8, 5)
```

특별한 메소드들

클래스에는 `__init__` 이나 `__del__` 메소드처럼 특별한 일을 하는 몇 개의 메소드들이 있습니다.

이러한 특별한 메소드들을 이용하면 파이썬에서 지원하는 내장 형식들이 동작하는 방식을 똑같이 흉내낼 수 있습니다. 예를 들어, 여러분이 새로 만든 클래스에서 `x[key]` 와 같은 형태의 인덱싱 연산을 가능하게 하고 싶을 경우 (리스트나 튜플처럼), 클래스에 `__getitem__()` 메소드를 구현해 주면 됩니다. 사실 파이썬에 내장된 `list` 클래스도 이러한 방식으로 구현되어 있습니다!

아래에 몇 개의 유용한 특별한 메소드의 목록이 있습니다. 모든 특별한 메소드들에 대해 알고 싶으시면, 공식 설명서를 읽어 보세요.

- `__init__(self, ...)`
  - 이 메소드는 객체가 새로 생성될 때 호출됩니다.
- `__del__(self)`
  - 이 메소드는 객체가 메모리에서 제거되기 직전에 호출됩니다 (그러나 이것은 언제 호출될 지 그 시점이 분명하지 않으므로 가능하면 사용을 피하세요).
- `__str__(self)`
  - `print` 문이라던가 `str()` 등이 사용될 경우 호출되어 객체의 문자열 표현을 반환합니다.
- `__lt__(self, other)`
  - 작음 연산자 (〈) 가 사용될 경우 호출됩니다. 이와 비슷하게, 모든 연산자(+, 〉, 등등)에 해당하는 특별한 메소드들이 하나씩 따로 존재합니다.
- `__getitem__(self, key)`
  - `x[key]`형태의 인덱싱 연산이 사용될 경우 호출됩니다.
- `__len__(self)`
  - 열거형 객체의 길이를 얻어오기 위한 내장 함수 `len()` 이 사용될 경우 호출됩니다.

한 줄짜리 블록

지금까지 여러분이 작성한 프로그램에서는 각 블록이 서로 다른 들여쓰기 단계에 따라 구분되어 있었을 것입니다. 그렇지만 한 가지 예외가 있습니다. 만약 블록에 딱 한 개의 명령만 존재하는 경우, 특히 조건문이나 반복문을 사용할 때, 한 줄에 해당 명령을 이어서 지정해 줄 수 있습니다. 아래 예제를 보면 이것을 좀 더 명확하게 이해할 수 있을 것입니다:

```
>>> flag = True
>>> if flag: print('Yes')
...
Yes
```

위와 같이, 한 줄짜리 블록은 새로 블록을 생성하지 않고 그 줄 뒤에 이어서 사용됩니다. 이러한 방식을 사용하면 여러분의 프로그램을 몇 줄 줄여줄 수는 있겠지만, 디버깅을 할 때와 같은 경우를 제외하고는 가능하면 이 방법을 사용하지 않기를 강력히 권합니다. 그 주된 이유는 적절한 들여쓰기를 사용할 경우, 그 아래에 추가 명령을 삽입하기가 좀 더 쉬워지기 때문입니다.

람다 (Lambda) 함수

`lambda` 문은 새 함수 객체를 만들 때 사용됩니다. 기본적으로 `lambda` 문은 한 줄짜리 수식으로, 파라미터를 받는 부분과 람다 함수 본체로 구성되어 있습니다. 이렇게 만들어진 함수를 호출하면, 지정된 수식의 계산 결과가 함수의 결과값으로 반환됩니다.

예제 (`more_lambda.py` 로 저장하세요):

실행 결과:

동작 원리

여기서 `list` 의 `sort` 메소드는 `key` 매개 변수를 인자로 받는데, 이것은 어떻게 리스트를 정렬할 것인지를 결정해주는 것입니다 (주로 오름차순으로 할 지 내림차순으로 할 지를 지정해 주는 데 쓰는 경우가 많습니다). 위 예제에서는 우리가 정의한 특별한 방식대로 정렬을 해 주려고 하며, 따라서 정렬 순서를 결정해 줄 수 있는 함수를 하나 만들어 주어야 합니다. 이 때 `def` 블록을 사용하여 함수를 생성하지 않고 lambda 식을 사용하여 새 함수를 그 자리에서 바로 만들어 주었습니다.

리스트 축약 (Comprehension)

리스트 축약 (또는 리스트 컴프리헨션) 은 이미 존재하는 하나의 리스트를 기반으로 또 다른 리스트를 생성할 때 사용됩니다. 예를 들어 숫자로 이루어진 리스트가 하나 있을 때 이 리스트의 모든 항목에 대해 각 항목이 2 보다 클 경우에만 2 를 곱해준 리스트를 생성하고 싶다고 해 봅시다. 리스트 축약은 이러한 종류의 상황에 적절하게 사용될 수 있습니다.

예제 (`more_list_comprehension.py` 로 저장하세요):

실행 결과:

동작 원리

위 예제에서는 기존 리스트에서 특정 조건 (`if i > 2`)을 만족하는 항목에 대해 2 를 곱해주는 조작을 가한 (`2*i`) 새 리스트를 생성하였습니다. 이 때 기존 리스트는 변경되지 않습니다.

리스트 축약을 사용하면 반복문을 사용하여 리스트에 있는 각각의 항목에 접근하고 새 리스트를 생성하는 등 많은 양의 코드를 한번에 줄여서 쓸 수 있는 장점이 있습니다.

함수 인자를 튜플이나 사전 형태로 넘겨받기

`*` 혹은 `**` 을 활용하면 함수의 매개 변수를 튜플이나 사전 형태로 넘겨받을 수도 있습니다. 이 방법은 함수의 인자의 개수가 정해지지 않은 함수를 정의하고 싶을 때 유용하게 사용됩니다.

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to the specified power.'''
...     total = 0
...     for i in args:
...         total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25
>>> powersum(2, 10)
100
```

변수 `args` 앞에 `*` 을 붙여 주면, 함수로 넘겨진 모든 다른 인수들이 args 라는 튜플에 담겨진 형태로 함수에 넘어오게 됩니다. `*` 대신 `**` 을 앞에 붙여 주면, 이번에는 인수들이 사전의 형태, 즉 키/값 쌍의 형태로 변환되어 넘어오게 됩니다.

assert 문

`assert` 문은 어떤 조건이 참인지 확실하게 하고 싶을 때 사용됩니다. 예를 들어, 리스트에 반드시 적어도 한 개의 항목이 담겨 있어야 하는 상황을 가정해 봅시다. 이 때 그렇지 않은 경우 오류 메시지가 나타나야 할 것입니다. 이러한 경우 assert 문을 활용할 수 있으며, 이는 주어진 조건이 참이 아닌 경우, AssertionError 를 발생시킵니다.

아래 예제의 `pop()` 메소드는 리스트의 마지막 아이템을 반환한 뒤 이를 삭제하는 메소드입니다.

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

`assert` 문은 신중하게 사용하여야 합니다. 보통 이렇게 하기보다는 예외 처리 구문을 작성하여, 문제가 무엇인지 확인하고 사용자에게 오류 메시지를 보여준 뒤 프로그램을 종료하게 하는 과정을 거치도록 하는 것이 낫습니다.

데코레이터

데코레이터는 어떤 함수를 감싸 주는 함수를 축약하여 간단하게 표현한 것입니다. 이것은 특정 코드를 "감싸는" 일을 계속 반복하여 해야 할 경우 유용하게 사용됩니다. 아래 예제에서는 어떤 함수가 실행되는 중에 오류가 발생하면 최대 5 번까지, 일정 간격을 두고 재실행하게 하는 `retry` 데코레이터를 만들어 주었습니다. 이러한 데코레이터는 여러분이 원격 컴퓨터에 네트워크를 통해 접속을 시도하거나 하는 상황 등에서 유용하게 활용될 수 있습니다:

실행 결과:

동작 원리

다음을 참고하세요:

- Video : Python Decorators Made Easy
- http://www.ibm.com/developerworks/linux/library/l-cpdecor.html
- http://toumorokoshi.github.io/dry-principles-through-python-decorators.html

파이썬 2와 3의 차이점

다음을 참고하세요:

- "Six" library
- Porting to Python 3 Redux by Armin
- Python 3 experience by PyDanny
- Official Django Guide to Porting to Python 3
- Discussion on What are the advantages to python 3.x?

요약

이 챕터에서는 파이썬의 여러 기능에 대해 좀 더 다양하게 살펴보았습니다. 아직 우리가 파이썬의 모든 기능을 다 짚고 넘어온 것은 아니지만, 여러분이 실전에서 사용할 수 있을 만큼은 충분히 다루었습니다. 이제 앞으로 무엇을 더 배워야 할지에 대해서는 앞으로 여러분이 어떤 프로그램을 만들게 될 지에 따라 여러분이 직접 결정하면 될 것입니다.

다음으로 파이썬을 좀 더 자세히 공부해 볼 수 있는 방법에 대해 알아보겠습니다.

남은 과제

지금까지 이 책의 내용을 잘 따라오신 분들이라면 아마 많은 파이썬 프로그램을 작성해 보는 연습을 해 보셨을 것이고, 파이썬이 더 편안해지고 친숙해졌을 것입니다. 또 이 책의 예제 프로그램 이외에 이것저것 다른 것들도 시도해 보신 분들도 계시겠죠. 혹시 아니시라면, 지금이라도 그렇게 하셔야 합니다. 자, 이제 남은 질문은 하나입니다. '앞으로 무엇을 하면 좋을까요?'

아래 문제를 한번 해결해 보시기 바랍니다:

> 명령줄에서 실행되는 주소록 프로그램을 만들어 보세요. 여기에는 친구, 가족, 동료 등을 카테고리별로 나눠 저장해 두며, 또 항목을 검색하거나 새로 추가 혹은 삭제할 수 있고 각 사람의 메일 주소, 전화번호 등의 정보를 담을 수 있도록 하세요. 또한 이러한 정보들을 저장해 두었다가 언제든 불러와 사용할 수 있게 하세요.

지금까지 살펴본 다양한 내용들을 다시 되짚어 보면 위 문제 정도는 쉽게 해결할 수 있을 것입니다. 힌트를 원하신다면 아래 꼬리말을 참조하세요 [3].

위 문제를 성공적으로 해결하셨다면, 여러분은 이제 한 사람의 파이썬 프로그래머라고 불리기에 아무 손색이 없을 것입니다. 이제 저에게 이런 좋은 책을 써서 고맙다는 메일 한 통을 보내 주세요 ;-). 또, 이 책의 지속적인 발전을 위해 책을 구입하시는 것도 고려해 주세요.

위 문제가 너무 쉬웠다면, 아래 문제도 해결해 보시기 바랍니다:

[3] 각 사람의 정보를 담는 클래스를 하나 만드세요. 그리고 각 사람의 이름을 키로 하고 사람의 객체를 값으로 하는 사전에 데이터를 저장해 둡니다. 그리고 pickle 모듈을 사용하여 이 객체를 여러분의 하드 디스크에 저장해 두세요. 또 사전의 내장 메소드를 활용하여 사람을 추가하고 삭제하거나 수정하는 기능을 구현하세요.

replace 명령을 직접 구현해 보세요. 이 명령은 주어진 파일들 내부의 모든 특정 문자열을 다른 문자열로 치환하는 데 사용됩니다.

여러분의 취향에 따라 구현은 단순히 문자열을 치환해 주는 식으로 구현될 수도 있고 패턴 검색 (정규 표현식)을 통해 좀 더 유연하게 구현될 수도 있습니다.

더 많은 과제

위 문제들이 너무 쉬웠다면, 다음 사이트에 주어진 과제들을 프로그램으로 구현해 보시기 바랍니다: https://github.com/thekarangoel/Pr
(또는 Martyr2의 Mega Project List)

또 다음을 확인해 보세요:

- 프로그래머가 되기 위한 연습: 코딩 스킬을 기르기 위한 57가지 도전과제 (Exercises for Programmers: 57 Challenges to Develop Your Coding Skills)
- 파이썬 중급 프로젝트 (Intermediate Python Projects).

예제 코드 읽기

프로그래밍 언어를 배우는 가장 좋은 방법은 코드를 직접 많이 써보고 또 많이 읽는 것입니다:

- Python Cookbook 은 파이썬을 이용하여 어떤 문제를 해결할 때 유용한 해결 방법과 팁을 정리해 놓은 굉장히 좋은 사이트입니다. 파이썬 사용자라면 반드시 한 번쯤 읽어 볼 필요가 있습니다.
- Python Module of the Week 또한 한 번쯤 읽어 봐야 할 매우 잘 쓰인 표준 라이브러리 가이드입니다.

참고 문서

- 파이썬을 여행하는 히치하이커를 위한 안내서(The Hitchhiker's Guide to Python!)
- 파이썬 스타일의 기본요소 (The Elements of Python Style)
- 파이썬의 큰 그림(Python Big Picture)
- 전자책 "Writing Idiomatic Python" (유료)

파이썬 관련 영상

- 플라스크를 활용한 풀스택 웹 개발 (Full Stack Web Development with Flask)
- PyVideo

질문과 답변

- Official Python Dos and Don'ts
- Official Python FAQ
- Norvig's list of Infrequently Asked Questions
- Python Interview Q & A
- StackOverflow questions tagged with python

튜토리얼

- Hidden features of Python
- What's the one code snippet/python trick/etc did you wish you knew when you learned python?
- Awaretek's comprehensive list of Python tutorials

커뮤니티

파이썬을 사용하다가 도저히 해결하지 못할 문제에 직면한 경우, python-tutor list (영문) 혹은 Python Korea 페이스북 그룹 (한글) 에 질문하시면 좋습니다.

질문하기 전에 먼저 직접 문제를 해결하려고 노력해 보신 후에 질문하시기 바라며, 다음을 읽어 보시기 바랍니다: 좋은 질문을 하는 방법(영문).

새로운 소식

파이썬 세계에서 벌어지고 있는 최신 정보들을 접하고 싶으시면 공식 Python Planet 사이트 (영문) 를 확인하세요.

라이브러리 설치하기

파이썬 패키지 인덱스 (Python Package Index, PyPI) 에 여러분의 프로그램에서 사용할 수 있는 수많은 오픈 소스 라이브러리가 존재합니다.

pip를 활용하면 이 라이브러리들을 쉽게 설치하고 사용할 수 있습니다.

홈페이지 제작

플라스크(Flask) 를 이용하여 홈페이지를 만들 수 있습니다. 다음을 읽어 보세요:

- Flask Official Quickstart
- The Flask Mega-Tutorial
- Example Flask Projects

GUI 프로그램 만들기

파이썬 바인딩을 제공하는 여러 GUI(Graphical User Interface) 라이브러리를 사용하여 GUI 프로그램을 제작할 수 있습니다. 바인딩이란 C, C++ 혹은 다른 언어로 제작된 라이브러리를 파이썬에서 불러와 사용할 수 있도록 하는 일종의 연결 모듈을 말합니다.

다음은 파이썬에서 사용할 수 있는 GUI 라이브러리 목록입니다:

- Kivy
  - http://kivy.org
- PyGTK
  - 이것은 GNOME을 제작할 때 사용된 GTK+ 의 파이썬 바인딩입니다. GTK+은 처음에는 사용하기 불편할 수 있지만 한번 익숙해지면 GUI 프로그램을 빠르게 제작할 수 있게 됩니다. 이때 Glade라는 디자인 도구를 많이 사용합니다. GTK+ 를 사용하여 자유/독점 소프트웨어를 모두 작성할 수 있습니다. PyGTK를 사용하시려면 PyGTK 튜토리얼을 읽어 보세요.
- PyQt
  - 이것은 KDE을 제작할 때 사용된 Qt의 파이썬 바인딩입니다. Qt는 Qt Designer라는 디자인 도구와 방대한 문서 덕택에 굉장히 사용이 쉽고 강력한 도구입니다. PyQt는 오픈 소스 (GPL 하에서) 소프트웨어를 작성하려고 할 때 무료로 사용할 수 있습니다만, 독점 소프트웨어를 작성하려고 할 때는 사용권을 구입하여야 합니다. 또 Qt 4.5부터는 GPL이 아닌 소프트웨어 또한 작성할 수 있습니다. PyQt를 사용하시려면 PySide 를 읽어 보세요.
- wxPython
  - 이것은 wxWidgets의 파이썬 바인딩입니다. wxPython은 조금 어렵기 때문에 익숙해 지기까지 조금 시간이 필요할 수 있습니다. 그러나, wxPython으로 작성된 프로그램은 GNU/Linux, Windows, Mac 등 여러 플랫폼을 지원하며 심지어 임베디드(embedded) 플랫폼에서도 사용이 가능합니다. 또한 SPE (Stani's Python Editor) 나 wxGlade 와 같은 IDE 혹은 GUI 디자인 도구들을 사용할 수 있습니다. wxPython을 이용하여 독점 소프트웨어 또한 자유롭게 작성이 가능합니다. wxPython을 사용하시려면 wxPython 튜토리얼을 읽어 보세요.

그 외의 GUI 저작 도구들

그 외의 다른 도구들에 대해서는 파이썬 공식 사이트의 GuiProgramming 위키 페이지를 참조하세요.

지금까지는 파이썬을 위한 "표준" GUI 저작 도구는 없습니다. 따라서, 위 목록에서 여러분의 상황에 맞는 도구를 하나 골라 사용하시는 것을 추천합니다. 아마 첫 번째 고려해야 할 점은 여러분이 선택한 GUI 저작 도구를 구입할지 여부일 것이고, 두 번째 고려해야 할 점은 여러분의 프로그램이 윈도우 환경이나 맥, 리눅스 중 하나에서만 동작해도 되는지 아니면 모든 환경에서 잘 동작해야 하는지를 결정해야 할 것입니다. 이때 여러분이 리눅스 환경을 선택했다면 여러분이 KDE를 사용하는지 GNOME을 사용하는지도 고려 대상이 될 것입니다.

이에 대한 좀 더 상세하고 포괄적인 분석을 원하신다면, 'The Python Papers, Volume 3, Issue 1' (PDF) 의 26페이지를 참조하시기 바랍니다.

다양한 파이썬 구현들

프로그래밍 언어는 크게 두 부분으로 나뉘는데, 그 하나는 언어이고 또 하나는 소프트웨어입니다. 여기서 언어란 어떻게 프로그램을 작성하는지 정의해 둔 것을 말하며, 소프트웨어란 무엇이 실제로 이렇게 작성된 프로그램을 실행시키는지를 말합니다.

지금까지 우리는 여러분이 작성한 프로그램을 실행시키기 위해 CPython 이라는 소프트웨어를 사용해 왔습니다. 이것은 C 언어로 작성되었기 때문에 CPython이라고 하며, 이는 가장 기본적인 파이썬 인터프리터 입니다.

그렇지만, 우리가 작성한 파이썬 프로그램을 실행할 수 있는 다양한 다른 소프트웨어들도 존재합니다:

- Jython
  - 이것은 자바 플랫폼상에서 동작하는 파이썬 구현입니다. 이를 이용하면 파이썬 언어 안에서 자바 라이브러리 및 클래스를 불러와 사용할 수 있으며, 그 반대도 가능합니다.
- IronPython
  - 이것은 .NET 플랫폼상에서 동작하는 파이썬 구현입니다. 이를 이용하면 파이썬 언어 안에서 .NET 라이브러리 및 클래스를 불러와 사용할 수 있으며, 그 반대도 가능합니다.
- PyPy
  - PyPy는 파이썬으로 작성된 파이썬 구현입니다! 이것은 C, Java, C# 등과 같은 정적인 언어를 배제한 동적 언어로 구현된 인터프리터가 어디까지 빨라질 수 있으며 또 얼마나 쉽게 구현할 수 있는지 확인해보려는 연구 프로젝트입니다.

이외에도 CLPython 이라는 Common Lisp로 작성된 파이썬 구현이 있습니다. 또 자바스크립트 인터프리터 상에서 동작하는 Brython 이 있는데, 이를 이용하면 자바스크립트 대신 파이썬을 이용하여 웹 브라우저상에서 동작하는 Ajax 프로그램을 제작할 수도 있습니다.

이러한 각각의 파이썬 구현은 각 분야에서 유용하게 사용됩니다.

(고급 프로그래머를 위한) 함수형 프로그래밍

여러분이 큰 프로그램을 제작해야 할 경우, 앞서 객체 지향 프로그래밍 챕터에서 배웠던 클래스에 기반한 객체 지향적 프로그래밍 대신 함수형 프로그래밍 방법에 대해서도 한 번쯤 배워 볼 필요가 있습니다:

- Functional Programming Howto by A.M. Kuchling
- Functional programming chapter in 'Dive Into Python' book
- Functional Programming with Python presentation
- Funcy library
- PyToolz library

요약

이제 여러분은 이 책의 마지막에 다다랐습니다. 그러나, 이것은 또 다른 시작일 뿐입니다! 여러분은 이제 열의에 가득 찬 한 명의 파이썬 프로그래머이고, 파이썬을 활용해 더 많은 문제를 해결할 준비가 되어 있을 것입니다. 이전에는 생각하지 못했던 여러 자동화 스크립트를 작성해 보거나, 직접 게임을 만들어 보는 등 여러 가지 시도를 해 보기 바랍니다. 자, 이제 시작해 봅시다!

부록: FLOSS

NOTE: 이 섹션은 2003년에 작성되었습니다. 따라서 몇몇 사항은 최신의 내용이 아닐 수 있습니다.

"자유/오픈 소스 소프트웨어(Free/Libre and Open Source Software)", 줄여서 FLOSS 란 공동체의 공유 정신, 특히 지식의 공유 정신에 의해 개발되는 소프트웨어를 의미합니다. FLOSS는 자유롭게 사용할 수 있고 수정할 수 있으며 재배포할 수 있습니다.

여러분이 이 책을 읽으셨다면 FLOSS에 이미 익숙할 것입니다. 왜냐면 지금까지 여러분이 사용해 왔던 파이썬이 바로 오픈 소스 소프트웨어이기 때문입니다!

아래 공동체에 의해 개발되는 몇가지 FLOSS의 예가 있습니다:

리눅스 (Linux): 이것은 GNU/Linux 운영 체제에서 사용되는 FLOSS 운영 체제 커널입니다. 리눅스, 즉 커널은 학생 시절의 리누스 토발즈 (Linus Torvalds)에 의해 시작되었습니다. 안드로이드도 리눅스를 기반으로 합니다. 또 현재 여러분이 접속하여 사용하는 거의 모든 웹 사이트들도 리눅스를 기반으로 동작하고 있습니다.

우분투 (Ubuntu): 캐노니컬(Canonical) 사의 후원을 받는 우분투는 커뮤니티에 의해 개발되는 GNU/Linux 배포판이며, 현재 가장 널리 사용되고 있습니다. 우분투는 여러 FLOSS를 누구나 쉽게 설치하여 사용할 수 있도록 해 줍니다. 가장 좋은 점은 CD를 넣고 컴퓨터를 재부팅하는 것만으로 GNU/Linux를 바로 사용해 볼 수 있다는 점입니다! 즉, 여러분의 컴퓨터에 설치하기 전에 완전한 새 OS를 사용해 볼 수 있습니다. 그러나 우분투는 완전한 자유 소프트웨어는 아닙니다. 우분투에는 몇가지 독점 드라이버, 펌웨어, 소프트웨어 등이 함께 포함되어 있습니다.

리브레오피스 (LibreOffice): 커뮤니티에 의해 개발되는 리브레오피스는 문서 편집기, 프리젠테이션, 스프레드시트, 그리기 도구 등 여러 기능을 갖춘 뛰어난 오피스 프로그램입니다. 리브레오피스를 이용하면 MS 워드, MS 파워포인트 등의 파일들 또한 쉽게 열고 편집할 수 있습니다. 리브레오피스는 모든 플랫폼 상에서 동작하며 완전히 무료이고, 완전한 자유/오픈 소스 소프트웨어입니다.

모질라 파이어폭스 (Mozilla Firefox): 이것은 최고의 웹 브라우저입니다. 모질라 파이어폭스는 무시무시하게 빠르며, 감각적이고 인상적인 기능으로 인해 많은 찬사를 받아 왔습니다. 확장 기능을 통해 여러 플러그인을 사용할 수 있습니다.

모노 (Mono): 모노는 마이크로소프트 (Microsoft) 닷넷 (.NET) 플랫폼의 오픈 소스 구현입니다. 이를 활용하면 .NET 소프트웨어를 GNU/Linux, 윈도우, FreeBSD, Mac OS 및 다른 플랫폼들에서 개발하고 구동할 수 있습니다.

아파치 웹 서버 (Apache web server): 이것은 지구상에서 가장 많이 사용되는 오픈 소스 웹 서버입니다! 현존하는 모든 웹사이트의 절반 이상이 아파치 웹 서버를 기반으로 동작하고 있습니다. 아파치는 마이크로소프트 IIS를 포함한 모든 경쟁자들이 구동하는 홈페이지 수를 모두 합친 수보다 많은 홈페이지를 구동하고 있습니다.

VLC 플레이어 (VLC Player): 이것은 DivX, Mp3, Ogg, VCD, DVD 등등 수많은 형식을 지원하는 동영상 재생기입니다. 누가 오픈 소스 소프트웨어가 지루하다고 하던가요? ;-)

위 목록은 여러분에게 FLOSS에 대해 간략히 소개해드리기 위한 것일 뿐, Perl 이나 PHP 와 같은 언어들, 웹 사이트 관리 시스템 드루팔(Drupal), 데이터베이스 서버 PostgreSQL, 레이싱 게임 TORCS, KDevelop IDE, 동영상 재생기 Xine, 편집기 VIM 과 Quanta+, 음악 재생기 밴시(Banshee), 사진 및 그림 편집기 GIMP 등등 셀 수 없이 많은 뛰어난 FLOSS들이 존재합니다.

FLOSS의 최신 소식이 궁금하시다면 다음 웹 사이트들을 확인해 보세요:

- OMG! Ubuntu!
- Web Upd8
- DistroWatch
- Planet Debian

FLOSS에 대해 좀 더 알고 싶으시다면 다음 웹 사이트들을 확인해 보세요:

- GitHub Explore
- Code Triage
- SourceForge
- FreshMeat

이제 언제나 열려 있는 자유롭고 방대한 FLOSS의 세계로 떠나 보세요!

부록: 끝맺음

이 책을 제작하는데 사용된 거의 모든 소프트웨어는 FLOSS 입니다.

이 책의 탄생

이 책의 초판은 Red Hat 리눅스 9.0 을 기반으로 한 시스템에서 발행되었으며, 6 번째 판부터는 Fedora Core 3 리눅스를 기반으로 한 시스템을 사용하여 발행되었습니다.

또 서문의 history lesson 에서 언급한 것과 같이, 이 책은 KWord를 사용하여 처음으로 작성되었습니다.

이 책의 십대 시절

이후에는 Kate를 사용하여 DocBook XML으로 문서를 편집하기 시작했지만 이것은 조금 복잡하고 어려워서 문서 포맷팅에 필요한 여러 뛰어난 기능이 제공되는 OpenOffice를 사용해 보았지만, 이것은 PDF에 비해 HTML로는 문서를 예쁘게 잘 만들어 주지 못했습니다.

그러던 중 XEmacs라는 좋은 툴을 발견하게 되었고, 그래서 저는 DocBook XML을 사용하여 처음부터 다시 책을 작성하기로 했습니다.

책의 6판부터는 Quanta+ 편집기를 이용하기 시작했고, 또 이 때는 Fedora Core 3 리눅스와 함께 제공되는 기본 XSL 스타일시트를 사용하였습니다. 그렇지만 이 때는 HTML 페이지에 여러 색과 스타일을 주기 위해 CSS 문서들을 작성해 주어야 했습니다. 또한 예제 프로그램의 문법 강조 기능을 위해 파이썬으로 문법 분석기를 직접 작성하여야 했습니다.

책의 7판에 이르러서는 MediaWiki를 사용해 보았습니다. 이를 통해 문서를 온라인으로 편집할 수 있고 위키 웹 사이트를 통해 독자가 직접 내용을 읽고/편집하고/토론할 수 있게 할 수 있었지만, 저는 책을 작성하기 보다 스팸과 싸우는 데 시간을 더 많이 할애해야 했습니다.

책의 8판에서는 Vim, Pandoc, 그리고 Mac OS X를 이용했습니다.

책의 9판에서는 Emacs 24.3, tomorrow theme, Fira Mono font and adoc-mode 등을 이용하여 AsciiDoc format으로 책을 재작성했습니다.

현재

2016: AsciiDoctor의 몇가지 렌더링 문제에 지쳐 버렸습니다. 예를 들어 `C/C++` 에서 ++ 부분이 사라진다던지 하는 문제였는데, 이런 사소한 문제들을 확인하고 수정하는 것이 너무 힘들었습니다. 여기에 더해서, Asciidoc 포맷은 너무 복잡하여 책을 편집하기가 점점 꺼려지는 문제가 있었습니다.

책의 10판에서, 저는 Spacemacs editor를 활용해서 책을 편집하고, 포맷은 Markdown + GitBook 으로 변경했습니다.

2020년 11월: Gitbook이 더 이상 오픈 소스로 소프트웨어를 제공하지 않기로 함에 따라, Honkit (커뮤니티가 관리하는 Gitbook의 마지막 오픈 소스 버전) 을 사용하는 것으로 변경했습니다.

저자에 대하여

https://swaroopch.com/about 에 방문해 보세요.

부록: 과거로부터의 교훈

제가 파이썬을 처음 시작하게 된 계기는 전에 작성했던 'Diamond'라는 소프트웨어를 좀 더 쉽게 설치할 수 있게 하기 위해 설치 프로그램을 작성하려고 했던 일이었습니다. 당시에 Qt 라이브러리를 사용하고 싶었는데, 이를 위해 파이썬과 펄 중 하나를 골라야 했었지요. 그래서 인터넷을 찾아보았는데 유명한 해커 Eric S. Raymond가 쓴 글을 발견하게 되었습니다. 거기에는 파이썬이 어떻게 자신이 가장 좋아하는 프로그래밍 언어가 되었는지에 대해 적혀 있었지요. 또한 당시 Perl-Qt 바인딩에 비해 PyQt 바인딩이 좀 더 안정적이었기도 해서 저는 파이썬을 선택하게 되었습니다.

다음으로 저는 좋은 파이썬 책이 있는지 찾아보기 시작했습니다. 그런데, 아무것도 찾을 수가 없었습니다! 몇 권의 오라일리(O'Reilly) 책을 찾았지만 너무 비싸고 초보자를 위한 가이드이기보다는 레퍼런스 매뉴얼에 가까웠습니다. 그래서, 저는 파이썬 문서를 검색해보기 시작했습니다. 그렇지만 제가 찾은 것들은 너무 간략하거나 짧은 것들 뿐이었습니다. 저는 이런 문서들을 통해 기본적인 것들에 대한 도움을 받을 수 있었지만 파이썬에 대해 완전히 알 수는 없었습니다. 또 저는 프로그래밍을 전에 해 보았기 때문에 이런 짧은 문서들을 겨우겨우 이해해나갈 수 있었지만, 초보자에게는 불가능한 일이라는 것도 알게 되었습니다.

제가 파이썬을 처음 시작하고 6 개월 정도 지난 뒤, 저는 레드햇 리눅스(Red Hat Linux) 9.0을 설치하였는데 불현듯 KWord를 사용하여 파이썬에 대해 뭔가 적어보자는 생각이 떠올랐습니다. 처음에는 몇 장으로 시작했지만 곧 30 페이지 정도의 문서가 되었습니다. 그렇게 되니 좀 더 내용을 정리해서 책의 형태로 만들면 좀 더 좋겠다고 생각했습니다. 그 후 많은 재작성 끝에, 파이썬이라는 언어를 익히는 데 유용한 가이드북을 만들 수 있었습니다. 저는 이 책을 오픈 소스 공동체에 대한 저의 기여와 헌정품으로 여깁니다.

이 책은 파이썬에 대한 저의 개인 노트에서 출발했고, 앞으로도 같은 방식으로 편집될 것입니다. 물론 다른 사람들이 이 책을 읽기 쉽게 하기 위한 노력 또한 계속 기울일 것이고요 :)

또 오픈 소스 정신을 통해, 많은 열정적인 독자들의 제안과 비평 그리고 피드백을 바탕으로 이 책이 지금까지 성장할 수 있었음을 알립니다.

책의 현재

이 책은 책의 좋지 않은 부분, 이해하기 어려운 부분, 또는 잘못된 부분들을 지적하는 독자들의 도움이 필요합니다. 저자 또는 역자 에게 여러분의 의견과 지적 사항들을 보내 주시기 바랍니다.

부록: 리비전 기록

- No version change
  - 06 Nov 2020
  - Migrated from abandoned GitBook to community-maintained Honkit, a fork of GitBook legacy
- 4.0
  - 19 Jan 2016
  - Switched back to Python 3
  - Switched back to Markdown, using GitBook and Spacemacs
- 3.0
  - 31 Mar 2014
  - Rewritten for Python 2 using AsciiDoc and adoc-mode.
- 2.1
  - 03 Aug 2013
  - Rewritten using Markdown and Jason Blevins' Markdown Mode
- 2.0
  - 20 Oct 2012
  - Rewritten in Pandoc format, thanks to my wife who did most of the conversion from the Mediawiki format
  - Simplifying text, removing non-essential sections such as `nonlocal` and metaclasses
- 1.90
  - 04 Sep 2008 and still in progress
  - Revival after a gap of 3.5 years!
  - Rewriting for Python 3.0
  - Rewrite using http://www.mediawiki.org[MediaWiki] (again)
- 1.20
  - 13 Jan 2005
  - Complete rewrite using Quanta+ on Fedora Core 3 with lot of corrections and updates. Many new examples. Rewrote my DocBook setup from scratch.
- 1.15

    - 28 Mar 2004
    - Minor revisions
- 1.12
    - 16 Mar 2004
    - Additions and corrections
- 1.10
    - 09 Mar 2004
    - More typo corrections, thanks to many enthusiastic and helpful readers.
- 1.00
    - 08 Mar 2004
    - After tremendous feedback and suggestions from readers, I have made significant revisions to the content along with typo corrections.
- 0.99
    - 22 Feb 2004
    - Added a new chapter on modules. Added details about variable number of arguments in functions.
- 0.98
    - 16 Feb 2004
    - Wrote a Python script and CSS stylesheet to improve XHTML output, including a crude-yet-functional lexical analyzer for automatic VIM-like syntax highlighting of the program listings.
- 0.97
    - 13 Feb 2004
    - Another completely rewritten draft, in DocBook XML (again). Book has improved a lot - it is more coherent and readable.
- 0.93
    - 25 Jan 2004
    - Added IDLE talk and more Windows-specific stuff
- 0.92
    - 05 Jan 2004
    - Changes to few examples.
- 0.91
    - 30 Dec 2003
    - Corrected typos. Improvised many topics.
- 0.90
    - 18 Dec 2003
    - Added 2 more chapters. OpenOffice format with revisions.
- 0.60
    - 21 Nov 2003
    - Fully rewritten and expanded.
- 0.20
    - 20 Nov 2003
    - Corrected some typos and errors.
- 0.15
    - 20 Nov 2003
    - Converted to DocBook XML with XEmacs.
- 0.10
    - 14 Nov 2003
    - Initial draft using KWord.

번역

번역에 자원해주신 많은 지칠 줄 모르는 번역자 분들 덕택에, 이 책은 여러 다른 언어로 번역되어 있습니다!

번역에 도움을 주실 분들께서는 아래 번역 목록을 확인하시고 기존의 번역 프로젝트에 기여를 할 것인지 아니면 새로운

번역을 시작할 것인지를 결정해 주세요.

새로운 번역을 시작하시려는 분들께서는 번역 방법 챕터를 읽어 주세요.

Arabic

Below is the link for the Arabic version. Thanks to Ashraf Ali Khalaf for translating the book, you can read the whole book online at http://www.khaledhosny.org/byte-of-python/index.html or you can download it from sourceforge.net for more info see http://itwadi.com/byteofpython_arabi.

Azerbaijani

Jahangir Shabiyev (c.shabiev@gmail.com) has volunteered to translate the book to Azerbaijani. The translation is in progress at https://www.gitbook.com/book/jahangir-sh/piton-sancmasi

Brazilian Portuguese

There are two translations in various levels of completion and accessibility. The older translation is now missing/lost, and newer translation is incomplete.

Samuel Dias Neto (samuel.arataca@gmail.com) made the first Brazilian Portuguese translation (older translation) of this book when Python was in 2.3.5 version. This is no longer publicly accessible.

Rodrigo Amaral (rodrigoamaral@gmail.com) has volunteered to translate the book to Brazilian Portuguese, (newer translation) which still remains to be completed.

Catalan

Moises Gomez (moisesgomezgiron@gmail.com) has volunteered to translate the book to Catalan. The translation is in progress.

> Moisès Gómez - I am a developer and also a teacher of programming (normally for people without any previous experience).
>
> Some time ago I needed to learn how to program in Python, and Swaroop's work was really helpful. Clear, concise, and complete enough. Just what I needed.
>
> After this experience, I thought some other people in my country could take benefit from it too. But English language can be a barrier.
>
> So, why not try to translate it? And I did for a previous version of BoP.
>
> I my country there are two official languages. I selected the Catalan language assuming that others will translate it to the more widespread Spanish.

Chinese

In 2017 which is after 11 years, Mo Lun (i@molun.net) re-translated the book from the beginning based on Version 4.0. And the translation is storaged in GitHub and Gitbook. He is keeping follow this translated edition and ready to fix it if there is any wrong or mistake in the translated BoP.

The 2017 translation edition is available in https://bop.molun.net.

Mo Lun Says:

> I am a common journalism student from CYU, Beijing. And actually, I am an absolute newbie in Python programming when I start to translate this book. Initially, it was just a whim, but when I done this work, I realized that a decision triggered by interest had prompted me to go so far.

With the help of my predecessors' translations and the vast amount of information provided by the developed Internet, and with the help of my friends, I prudently presented this translation edition. I just hope my translation work will help other newcomers in learning Python.

At the same time, I am always waiting for my translation of the comments and suggestions, and ready to change or improve this superficial work.

Earlier Chinese translation

In 2005, Shen Jieyuan translated this book with version 1.20 to Chinese and published it to the Internet. This is the first Chinese edition. In BoP official site, he was called Juan Shen, with Email adderss orion_val@163.com。This edition has been disseminated on the network widely, and the links provided by BoP official site are not available anymore, so that its original source is unable to find. Therefore in here can't provide a certain address. But you can try to search keywords like "□明Python□程 沈□元" to find a copy.

Juan Shen says:

I am a postgraduate at Wireless Telecommunication Graduate School, Beijing University of Technology, China PR. My current research interest is on the synchronization, channel estimation and multi-user detection of multicarrier CDMA system. Python is my major programming language for daily simulation and research job, with the help of Python Numeric, actually. I learned Python just half a year before, but as you can see, it's really easy-understanding, easy-to-use and productive. Just as what is ensured in Swaroop's book, 'It's my favorite programming language now'.

'A Byte of Python' is my tutorial to learn Python. It's clear and effective to lead you into a world of Python in the shortest time. It's not too long, but efficiently covers almost all important things in Python. I think 'A Byte of Python' should be strongly recommendable for newbies as their first Python tutorial. Just dedicate my translation to the potential millions of Python users in China.

Chinese Traditional

Fred Lin (gasolin@gmail.com) has volunteered to translate the book to Chinese Traditional.

It is available at http://code.google.com/p/zhpy/wiki/ByteOfZhpy.

An exciting feature of this translation is that it also contains the executable chinese python sources side by side with the original python sources.

Fred Lin – I'm working as a network firmware engineer at Delta Network, and I'm also a contributor of TurboGears web framework.

As a python evangelist (:-p), I need some material to promote python language. I found 'A Byte of Python' hit the sweet point for both newbies and experienced programmers. 'A Byte of Python' elaborates the python essentials with affordable size.

The translation are originally based on simplified chinese version, and soon a lot of rewrite were made to fit the current wiki version and the quality of reading.

The recent chinese traditional version also featured with executable chinese python sources, which are achieved by my new 'zhpy' (python in chinese) project (launch from Aug 07).

zhpy(pronounce (Z.H.?, or zippy) build a layer upon python to translate or interact with python in chinese(Traditional or Simplified). This project is mainly aimed for education.

French

Gregory (coulix@ozforces.com.au) has volunteered to translate the book to French.

Gérard Labadie (gerard.labadie@gmail.com) has completed to translate the book to French.

This translation was later ported to the markdown format, updated to match the last version of the book, and published on GitBook by Romain Gilliotte (rgilliotte@gmail.com).

It can be found at https://rgilliotte.gitbook.io/byte-of-python/

German

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) and Christoph Zwerschke (cito@online.de) have volunteered to translate the book to German.

Translation can be found at http://cito.github.io/byte_of_python/

Lutz Horn says:

> I'm 32 years old and have a degree of Mathematics from University of Heidelberg, Germany. Currently I'm working as a software engineer on a publicly funded project to build a web portal for all things related to computer science in Germany.The main language I use as a professional is Java, but I try to do as much as possible with Python behind the scenes. Especially text analysis and conversion is very easy with Python. I'm not very familiar with GUI toolkits, since most of my programming is about web applications, where the user interface is build using Java frameworks like Struts. Currently I try to make more use of the functional programming features of Python and of generators. After taking a short look into Ruby, I was very impressed with the use of blocks in this language. Generally I like the dynamic nature of languages like Python and Ruby since it allows me to do things not possible in more static languages like Java.I've searched for some kind of introduction to programming, suitable to teach a complete non-programmer. I've found the book 'How to Think Like a Computer Scientist: Learning with Python', and 'Dive into Python'. The first is good for beginners but to long to translate. The second is not suitable for beginners. I think 'A Byte of Python' falls nicely between these, since it is not too long, written to the point, and at the same time verbose enough to teach a newbie. Besides this, I like the simple DocBook structure, which makes translating the text a generation the output in various formats a charm.

Bernd Hengelein says:

> Lutz and me are going to do the german translation together. We just started with the intro and preface but we will keep you informed about the progress we make. Ok, now some personal things about me. I am 34 years old and playing with computers since the 1980's, when the "Commodore C64" ruled the nurseries. After studying computer science I started working as a software engineer. Currently I am working in the field of medical imaging for a major german company. Although C++ is the main language I (have to) use for my daily work, I am constantly looking for new things to learn.Last year I fell in love with Python, which is a wonderful language, both for its possibilities and its beauty. I read somewhere in the net about a guy who said that he likes python, because the code looks so beautiful. In my opinion he's absolutly right. At the time I decided to learn python, I noticed that there is very little good documentation in german available. When I came across your book the spontaneous idea of a german translation crossed my mind. Luckily, Lutz had the same idea and we can now divide the work.I am looking forward to a good cooperation!

Greek

The Greek Ubuntu Community translated the book in Greek, for use in our on-line asynchronous Python lessons that take place in our forums. Contact @savvasradevic for more information.

Indonesian

Daniel (daniel.mirror@gmail.com) is translating the book to Indonesian at http://python.or.id/moin.cgi/ByteofPython.

Wisnu Priyambodo (cibermen@gmail.com) also has volunteered to translate the book to Indonesian.

Also, Bagus Aji Santoso (baguzzzaji@gmail.com) has volunteered.

Italian (first)

Enrico Morelli (mr.mlucci@gmail.com) and Massimo Lucci (morelli@cerm.unifi.it) have volunteered to translate the book to Italian.

The Italian translation is present at http://www.gentoo.it/Programmazione/byteofpython.

> Massimo Lucci and Enrico Morelli – we are working at the University of Florence (Italy) – Chemistry Department. I (Massimo) as service engineer and system administrator for Nuclear Magnetic Resonance Spectrometers; Enrico as service engineer and system administrator for our CED and parallel / clustered systems. We are programming on python since about seven years, we had experience working with Linux platforms since ten years. In Italy we are responsible and administrator for www.gentoo.it web site for Gentoo/Linux distrubution and www.nmr.it (now under construction) for Nuclear Magnetic Resonance applications and Congress Organization and Managements.That's all! We are impressed by the smart language used on your Book and we think this is essential for approaching the Python to new users (we are thinking about hundred of students and researcher working on our labs).

Italian (second)

An Italian translation has been created by Calvina Bice & colleagues at http://besthcgdropswebsite.com/translate/a-byte-of-python/.

Japanese

Shunro Dozono (dozono@gmail.com) is translating the book to Japanese.

Korean

Epsimatt (2019)

Epsimatt has started a new Korean translation:

- Read online at https://epsimatt.gitbook.io/byte-of-python/
- Follow progress at https://github.com/epsimatt/byte-of-python/issues/16

Older

Jeongbin Park (pjb7687@gmail.com) has translated the book to Korean – https://github.com/pjb7687/byte_of_python

> I am Jeongbin Park, currently working as a Biophysics & Bioinformatics researcher in Korea.
>
> A year ago, I was looking for a good tutorial/guide for Python to introduce it to my colleagues, because using Python in such research fields is becoming inevitable due to the user base is growing more and more.
>
> But at that time only few Python books are available in Korean, so I decided to translate your ebook because it looks like one of the best guides that I have ever read!

Currently, the book is almost completely translated in Korean, except some of the text in introduction chapter and the appendixes.

Thank you again for writing such a good guide!

Mongolian

Ariunsanaa Tunjin (luftballons2010@gmail.com) has volunteered to translate the book to Mongolian.

Update on Nov 22, 2009 : Ariunsanaa is on the verge of completing the translation.

Norwegian (bokmål)

Eirik Vågeskar is a high school student at Sandvika videregående skole in Norway, a blogger and currently translating the book to Norwegian (bokmål).

Eirik Vågeskar: I have always wanted to program, but because I speak a small language, the learning process was much harder. Most tutorials and books are written in very technical English, so most high school graduates will not even have the vocabulary to understand what the tutorial is about. When I discovered this book, all my problems were solved. "A Byte of Python" used simple non-technical language to explain a programming language that is just as simple, and these two things make learning Python fun. After reading half of the book, I decided that the book was worth translating. I hope the translation will help people who have found themself in the same situation as me (especially young people), and maybe help spread interest for the language among people with less technical knowledge.

Polish

Dominik Kozaczko (dominik@kozaczko.info) has volunteered to translate the book to Polish. Translation is in progress and it's main page is available here: Ukąś Pythona.

Update : The translation is complete and ready as of Oct 2, 2009. Thanks to Dominik, his two students and their friend for their time and effort!

Dominik Kozaczko – I'm a Computer Science and Information Technology teacher.

Portuguese

Artur Weber (arturweberguimaraes@gmail.com) has completed a translation of this book to Portuguese (as of Feb 21, 2018) at https://www.homeyou.com/~edu/introducao.

Artur Weber: My students study at the Polytechnic faculty in Ecological University in the city of Curitiba (Brazil) and some of them are interested in different papers.

As they are writing course and academic papers, they always look for interesting articles and pages. I also do my best to find interesting materials which can be the sources for their university works.

I found the materials from your site useful for some of my students who are writing papers based on programming on Python. Actually, that is why I made a decision to perform a Portuguese translation to let my students who don't know English read exciting articles in their mother tongue (in Portuguese).

Romanian

Paul-Sebastian Manole (brokenthorn@gmail.com) has volunteered to translate this book to Romanian.

> Paul-Sebastian Manole – I'm a second year Computer Science student at Spiru Haret University, here in Romania. I'm more of a self-taught programmer and decided to learn a new language, Python. The web told me there was no better way to do so but read ''A Byte of Python''. That's how popular this book is (congratulations to the author for writing such an easy to read book). I started liking Python so I decided to help translate the latest version of Swaroop's book in Romanian. Although I could be the one with the first initiative, I'm just one volunteer so if you can help, please join me.

Russian

Vladimir Smolyar (v_2e@ukr.net) has completed a Russian translation at http://wombat.org.ua/AByteOfPython/.

Ukranian

Averkiev Andrey (averkiyev@ukr.net) has volunteered to translate the book to Russian, and perhaps Ukranian (time permitting).

Serbian

"BugSpice" (amortizerka@gmail.com) has completed a Serbian translation:

> This download link is no longer accessible.

More details at http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona.

Slovak

Albertio Ward (albertioward@gmail.com) has translated the book to Slovak at http://www.fatcow.com/edu/python-swaroopch-sl/ :

> We are a non-profit organization called "Translation for education". We represent a group of people, mainly students and professors, of the Slavonic University. Here are students from different departments: linguistics, chemistry, biology, etc. We try to find interesting publications on the Internet that can be relevant for us and our university colleagues. Sometimes we find articles by ourselves; other times our professors help us choose the material for translation. After obtaining permission from authors we translate articles and post them in our blog which is available and accessible to our colleagues and friends. These translated publications often help students in their daily study routine.

Spanish

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) and Cristian Bermudez Serna (crisbermud@hotmail.com) have volunteered to translate the book to Spanish.

Gustavo Echeverria says:

> I work as a software engineer in Argentina. I use mostly C# and .Net technologies at work but strictly Python or Ruby in my personal projects. I knew Python many years ago and I got stuck inmediately. Not so long after knowing Python I discovered this book and it helped me to learn the language. Then I volunteered to translate the book to Spanish. Now, after receiving some requests, I've begun to translate "A Byte of Python" with the help of Maximiliano Soler.

Cristian Bermudez Serna says:

I am student of Telecommunications engineering at the University of Antioquia (Colombia). Months ago, i started to learn Python and found this wonderful book, so i volunteered to get the Spanish translation.

Swedish

Mikael Jacobsson (leochingkwake@gmail.com) has volunteered to translate the book to Swedish.

Turkish

Türker SEZER (tsezer@btturk.net) and Bugra Cakir (bugracakir@gmail.com) have volunteered to translate the book to Turkish. "Where is Turkish version? Bitse de okusak."

번역 방법

1. 이 책의 소스 코드는 https://github.com/swaroopch/byte-of-python 에 공개되어 있습니다.
2. 책의 저장소를 fork 하세요.
3. 그리고 저장소의 내용을 컴퓨터로 받아옵니다. 이를 위해 Git 의 사용법을 익히세요.
4. Honkit 문서를, 특히 Markdown 섹션을 주목하여 읽으세요.
5. `.md` 파일을 편집하여 번역을 시작하세요.
6. INSTALL.md 파일을 읽고 책의 웹사이트, PDF, EPUB 파일을 만드는 방법에 대해 알아보세요.

피드백

이 책은 책의 좋지 않은 부분, 이해하기 어려운 부분, 또는 잘못된 부분들을 지적하는 독자들의 도움을 필요로 합니다. 저자 또는 역자 에게 여러분의 의견과 지적 사항들을 보내 주시기 바랍니다.