점프 투 파이썬

마지막 수정일시 :2010년 12월 20일 2:58:11 오후
작성자 : pahkey

책 설명

파이썬이란 1990년 암스테르담의 귀도 반 로섬에 의해 만들어진 인터프리터 언어이다. 귀도는 이 파이썬이라는 이름을 어린이를 위한 프로그램인 ‘Monty Python's Flying Circus’라는 코미디 쇼에서 따왔다고 한다. 파이썬(Python)의 사전적인 뜻은 고대 신화 속의 파르나수스(Parnassus) 산의 동굴에 살던 큰 뱀으로서, 아폴로가 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있다. 대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있다. 현재 파이썬은 국내에서는 많이 알려져 있지 않지만 외국에서는 학습의 목적뿐만 아니라 실용적인 부분에서도 많이 사용되고 있는데 그 대표적인 예를 보면 레드햇 리눅스의 설치 프로그램인 아나콘다, 구글(Google)이나 인포시크(Infoseek)에서 사용되는 검색 프로그램, 야후의 많은 인터넷 서비스 프로그램 등이 파이썬으로 개발되었다. 또한 파이썬 프로그램은 공동작업과 유지보수가 매우 쉽고 편하기 때문에 이미 다른 언어로 작성된 많은 프로그램과 모듈들이 파이썬으로 다시 재구성되고 있는 상황이다. 국내에서도 그 가치를 인정받아 사용자층이 더욱 넓어져 가고 있고, 파이썬을 이용한 프로그램을 개발하는 기업체들이 늘어가고 있는 추세이다.

목 차

00. 점프 투 파이썬

2001년 출간 [정보게이트]
저자 : 박응용

"강력하고 배우기 쉬운 스크립트 언어 "파이썬" 입문서"

**** 공지사항 (새소식) ****

[2010.12.23] PDF파일을 제공합니다.
위키독스가 2010년 년말에 기분좋은 선물을 드리고자 합니다. ^^
2010.12.22일 기준으로 제작된 "점프 투 파이썬" PDF파일을 제공합니다.
B5형태로 제작되었으니 제본하실 분들은 참고하시기 바랍니다.

- PDF 다운로드

[2010.12.21] django챕터가 추가되었습니다.
위키독스에 게시되었던 "초간단 장고 매뉴얼"이 파이썬 챕터로 추가되었습니다.

- http://wikidocs.net/read/1511

[2010.05.10] 안드로이드폰에서 점프 투 파이썬을 읽어 보세요.
안드로이드용 "점프 투 파이썬" 앱이 만들어졌습니다.
안드로이드 마켓에서 "점프 투 파이썬"으로 검색하시면 무료로 다운로드 받을 수 있습니다.

[2009.07.22] 점프 투 파이썬 작가를 모십니다.
점프 투 파이썬의 각 챕터 또는 새로운 챕터를 수정 및 집필해 주실 작가분을 모십니다.
(신규 챕터는 자유롭게 설정하시면 됩니다.) 참여를 원하시는 분은 아래 이메일로 연락주시기

바랍니다.
위키독스의 공동집필 기능을 이용하여 함께 수정 및 추가하는 형태로 진행됩니다.
많은 참여 부탁드립니다. ^^

박응용: pahkey@gmail.com

[2009.02] 다음과 같은 모토의 Endless Edition으로 점프 투 파이썬이 새롭게 변화해 갑니다.

> "어제의 점프 투 파이썬과 오늘의 점프 투 파이썬은 다르다."

01. 들어가기전에

> 점프 투 파이썬은 위키독스(http://wikidocs.net)를 이용하여 작성합니다.
> 빠른 페이지 이동을 위해서는 **목차보기(F2)**를 이용하는 것이 편리합니다.

누구를 위한 책인가?
이 책은 파이썬이란 언어를 처음 접해보는 독자들과 프로그래밍을 한 번도 해 본적이 없는 사람들을 대상으로 한다. 프로그래밍을 할 때 사용되는 전문적인 용어들을 알기 쉽게 풀어서 쓰려고 노력하였으며, 파이썬이란 언어의 개별적인 특성만을 강조하지 않고 프로그래밍 전반에 관한 사항을 파이썬이란 언어를 통해 알 수 있도록 알기 쉽게 설명하였다. 파이썬에 대한 기본적인 지식을 알고 있는 사람이라도 이 책은 파이썬 프로그래밍에 대한 흥미를 가질 수 있는 좋은 안내서가 될 것이다.

이 책의 목표는 독자가 파이썬을 통해 프로그래밍에 대한 전반적인 이해를 갖게하는 것이며, 또 파이썬이라는 도구를 이용하여 원하는 프로그램을 쉽고 재미있게 만들 수 있게 하는 것이다.

<책의 구성>

1. 들어가기전에
이 책을 쓴 목적 및 책을 읽는 독자의 대상에 대해서 말하고 책의 목차를 리뷰한다.

2. 파이썬이란 무엇인가?
파이썬에 대한 소개와 특징, 그리고 파이썬 프로그래밍을 시작하기 위해서 갖추어야 할 것들에 대해서 다룬다.

3. 자료형과 제어문
프로그래밍 언어를 배우기 위해서 반드시 익혀야 하는 자료형과 제어문에 대해서 다룬다.

4. 입출력
함수와 여러가지 입출력등에 대해서 다룬다.

5. 파이썬 날개달기
객체 중심의 프로그래밍을 하기 위한 필수 자료형인 클래스에 대해서 알아보고, 모듈과 예외처리등의 여러기법에 대해서 알아본다.

6. 어디서부터 시작할 것인가?
도대체 어디서부터 프로그래밍을 시작할 것인지 망설이는 독자들을 위해서 간단한 예제와 함수들을 소개한다.

7. 테스트 주도 개발
Simple한 코드를 만들기 위한 개발방법인 TDD에 대해서 간단하게 알아보고, 파이썬으로는 어떻게 테스트 주도적 개발을 할 수 있는지 알아본다.

8. 인터넷 프로그래밍
인터넷 프로그래밍을 하기에 앞서 꼭 알아두어야 할 기본적인 것들에 대해서 다루고, 파이썬 웹 프레임워크 중 가장 유명한 장고(django)에 대해서 살펴본다.

9. 파이썬 Tips
파이썬으로 할 수 있는 여러 유용한 Tip들에 대해서 살펴본다.

10. 파이썬 Quiz
유명한 Quiz들을 독자들과 함께 풀어본다.

90. 부록
파이썬에 대해서 더 자세하게 알고 싶은 독자를 위해서 더 많은 정보를 얻을 수 있는 방법을 알려준다.

1) Endless-Edition을 계획하며

Endless Edition이란 **끊임없이 변화하는 책**을 의미합니다.

현재 점프 투 파이썬은 끊임없이 변화해 가고 있는 중입니다.
독자들의 무수한 댓글로 점프 투 파이썬은 조금씩 계속해서 성장해 가고 있습니다. 또한 "점프 투 파이썬"에 집착했던 그동안의 관점에서 벗어나 파이썬으로 할 수 있는 다방면의 많은 지식들을 다룰 예정입니다. 본인의 지식만이 아닌 많은 사람들의 지식을 얻기위한 노력도 함께 하고자 합니다. (이에 대해 좋은 의견이 있다면 이곳에 댓글로 알려주시기 바랍니다.)

- 기고 부탁하기
- 기 작성된 파이썬 문서 "점프 투 파이썬"에 등록하기

Endless Edition은 다음과 같은 모토로 온라인 상에서 계속 진화해 나갈 예정입니다.

> "어제의 점프 투 파이썬과 오늘의 점프 투 파이썬은 다르다"

2009.02
박응용

2) 머리말

필자는 파이썬의 '>>>'프롬프트를 처음 보았던 순간부터 지금까지 파이썬과 함께 지내온 듯 하다. 항상 " 프로그래밍은 어렵고 지루한 것이다. " 라는 고정관념을 가지고 있었던 필자에게 파이썬은 커다란 충격으로 다가왔다. 파이썬의 깔끔한 문장구조와 프로그래밍이 정말로 즐겁게 느껴지게 하는 야릇한 매력은 지금껏 경험해 보지 못한 것이었다. 아직 파이썬의 향기를 맡지 못한 독자라면 꼭 파이썬이란 언어를 느껴볼 것을 당부한다. 분명 파이썬의 매력에 흠뻑 젖게 될 것이다.

파이썬은 프로그래밍을 잘하고 싶은 사람보다는 프로그래밍을 즐기고 싶은 사람에게 어울리는 언어임에 분명하다.

이 책을 쓸 수 있도록 도와주신 많은 분들께 감사의 말을 전하고 싶다.

이 책이 나올 수 있도록 여러모로 힘써 주신 정보게이트 관계자 여러분, 특히 무엇보다도 처음으로 책을 써 보는 필자에게 큰 힘이 되어주신 유혜규 기획실장님께 감사를 드린다.

그리고 한번도 프로그래밍을 해 본적이 없는 작은형이 지적해준 사항은 이 책을 초보자가 이해하기 쉽게 쓰도록 해 주었는데, 꼼꼼하게 원고를 살펴보아 주었던 석이형에게 이 책을 통해서 감사의 마음을 표하고 싶다. 또한 내가 편안한 마음으로 책을 쓸 수 있게 많은 배려를 해 주었던 가족들에게 고마운 마음을 전하고 싶다.

항상 격려와 도움을 주었던 학교 친구들에게도 고마운 마음을 전하고 싶다. 특히 내가 힘들어 할 때 나의 넋두리를 받아주고 많은 충고를 해 주었던 형석, 승용에게 이 책을 빌어 고마운 마음을 전하고 싶다.

마지막으로 바쁜 와중에도 책의 내용을 검토해 주시고 서문을 써 주신 리눅스 코리아의 이만용 이사님께 진심으로 감사의 뜻을 전하고 싶다. 또한 국내 파이썬 보급과 발전에 많은 힘을 기울이는 파이썬 사용자모임의 여러분들 모두에게 감사의 마음을 전하고 싶다.

2001. 06

3) 추천사

리눅스코리아 CTO 이만용
yong@linuxkorea.co.kr

프로그래머에게 있어 언어의 선택은 중요하다. 특히, 프로그래머가 되고자 마음먹은 사람에게 있어 '첫 번째' 언어의 선택은 더더욱 중요하다. 화가에게 양질의 붓과 캔버스가 필요하고 음악가에게 훌륭한 악기가 필요하듯, 프로그래머에게는 프로그래밍이라는 창조적 두뇌 작업을 도와 줄 훌륭한 프로그래밍 언어가 필요하다. 인간은 도구의 동물이라 하지 않았던가? 인간이 도구를 만들 듯, 도구는 또 다시 인간을 만들어준다는 의미에서 도구의 선택 행위는 인간의 가장 중요한 선택 행위 중 하나이다.

수많은 프로그래머 지망생들은 프로그래밍 언어를 선택함에 있어 처음부터 장벽에 부딪히기 시작한다. 내게 어떤 프로그래밍 언어를 해야 할 지 물어오는 많은 고등학생, 대학생들은 마이크로소프트사나 썬 마이크로시스템즈사가 프로그래밍 잡지를 통해 열심히 홍보하는 주류 언어만을 나열하면서 '낙점'을 해 달라고 요청한다. 역시 비주얼C++을 해야겠죠? 자바(Java)가 대세이니 자바를 먼저 해야겠죠? 아니면 주위 얘기를 들어보니 프로그래머의 근본을 갖추기 위해서는 C언어부터 다시 해야겠죠? (비주얼 C++은 제품 이름일 뿐인데 C++ 언어와 착각하는 사람도 많다. 핸재 많은 사람들이 제품과 언어를 착각하곤 한다.) 그들의 질문은 크게 다르지 않다. 그들은 빨리 프로그래밍을 하고 싶어하면서도 동시에 누구나 다 사용하는 현실적인 언어를 선택하고 싶어한다. 누군들 그러고 싶지 않은 사람이 있을까?

지난 1999년 파이썬(Python)이라는 언어를 알기 전까지 이 모든 대답에 통쾌한 답변을 주지 못했다. 언어란 결국 도구이므로 자기에게 가장 알맞은 것을 선택하라는 답변 뿐 현실적인 대안을 제시하기 힘들었다. 그러나 이제는 자신감을 가지고 고민하는 모든 프로그래머와 프로그래머가 되고자 마음먹은 초보 프로그래머들에게 파이썬이라는 언어를 추천할 수 있다.

파이썬은 그 자체로 매우 훌륭한 설계를 가지고 있으며(수학적 정확성 속에서 탄생하였다) 배우기 쉽고 현대적인 설계를 가지고 있으며 또한 '현실적'이고자 하는 많은 사람들을 위해 현실적인 강력함도 고루 갖춘 언어이다. 다른 언어와 비교하지 않아도 그 자체로 가치있는 프로그래밍 언어이다(본인과 본인의 회사 개발팀은 글자 그대로 100% 모든 프로젝트를 파이썬으로 진행하고 있다.)

그러나 이 글의 필자와 본인은 그냥 또 다른 언어 중 하나를 여러분에게 추천하는 것이 아니다. 이 세상에는 이미 프로그래밍 언어가 넘쳐난다. 이 글과 책은 여러분이 파이썬을 통해 2000년대의 프로그래밍 언어가 어떤 것이어야 하며, 프로그래머에게 있어 언어 선택이란 어떤 것인지, 그리고 결국에는 파이썬이든 그 무엇이든 상관없이 자기의 필요에 따라 어떻게 언어를 선택하고 언어를 창조해 나갈 수 있는지를 보여주고자 할 것이다.

프로그래밍에 있어 중요한 것 중 하나가 바로 스타일(Style)이다. 여러분이 생각하고 코딩하는 스타일을 제대로 배우지 못하면 즐길 줄 아는 프로그래머가 되기 어렵다. 훌륭한 스타일은 좀 더 높은 상상력을 낳고 프로그래밍을 재미(fun)로서 향유할 수 있는 여유를 가져다 준다. 직업 프로그래머가 되고자 하는 사람이라면 파이썬이든 자바든 하나의 언어만 가지고 살아갈 수는 없다. 여러분의 의지와 상관없이 독점적 지위를 가진 업체에서 언어를 하나 새롭게 만들어서 추진하면 배울 수밖에 없는 것이

현실이다(예를 들어 마이크로소프트사의 C#이 그러하다). 따라서 더더욱 중요한 것은 그 어떤 언어에도 쉽게 적응할 수 있도록 해 주는 기본기와 스타일이다. 바로 그것을 파이썬 언어에서 배울 수 있기를 기대한다. 무작정 C++이나 자바를 먼저 배우는 것보다 여유를 가지고 파이썬을 배우고 점차로 C++이나 자바를 배워간다면 오히려 더 빨리 더 많은 것을 성취할 수 있다. 그것이 바로 스타일의 힘이다.

사실 이 책을 들고 있는 독자의 대부분은 어디에선가 파이썬의 '명성'을 들었을 것이다. 1~2년전에는 꿈도 꾸지 못할 일이다. 리눅스처럼 강력한 오픈 소스 프로젝트 중 하나가 되어버린 파이썬은, 언어의 창조자 Guido van Rossum의 노력은 물론이고 파이썬을 현실적으로도 강력한 언어가 되도록 모듈을 만들어내고 있는 전세계 자발적인 프로그래머, 그리고 한국 파이썬 사용자모임에서 각종 세미나와 홍보 활동을 적극 벌이고 있는 회원들의 노력(이미 한국은 전세계적으로도 파이썬 열기가 가장 높은 나라 중 하나이다)을 통해 많은 사람들에게 알려져가고 있기 때문이다. 여러분도 파이썬을 자기 것으로 만든 후, 이 대열에 동참할 수 있기를 바란다. 파이썬을 배우면서 파이썬의 창조자 대열에 설 수 있다면 여러분은 이미 최고의 프로그래머가 되었다고 말할 수 있다.

이 글을 쓴 저자와 본인은 아직 한 번도 만나지 못했다. 그러나 같이 파이썬에 미쳐서 프로그래밍의 참 재미(fun)를 만끽할 수 있는 사람이라는 동지애를 느끼지 않을 수 없다. 그 동안 훌륭한(또는 대충 훌륭한) 외국 파이썬 서적이 몇 권 나왔다. 그러나 언어적 장벽 때문인지 초보 파이썬 입문자에게는 어려운 점이 없지 않았다. 추천사를 쓰기 위해 원고를 모두 읽으면서 몇 가지 미흡한 점을 발견하지 않은 것은 아니나, 적어도 이 책의 내용은 저자가 내면화한 뒤 다시 써내려간 내용이므로 의미를 갖는다. 안타까운 국내 출판 실정 때문에 나올 수밖에 없는 조잡한 번역서류에서 얻을 수 없는 것은 분명히 얻을 수 있다. 한국인 저자가 한국인을 위해 쓴 첫 번째책 중 하나라는 사실을 높이 평가하고 싶다.

파이썬은 여러분에게 그 자체로 흥미로운 언어라고 확신한다. 그러나 무엇보다도 프로그래머는 주체적으로 언어를 선택하고 새로운 언어를 계속 습득해야 한다는 깨달음을 얻기 바란다. 그것이 저자와 본인의 똑 같은 소망일 것이다.

마지막으로 파이썬을 통해 프로그래밍이란 원래 지적인 재미(fun)에 미치는 일이라는 사실을 느낄 수 있기 바란다. 본인은 사람들에게 매번 이 말을 강조한다. "똑똑한 사람은 결국 성실한 사람을 이길 수 없다. 그러나 성실한 사람이라도 이길 수 없는 사람이 있으니, 바로 그 일에 미친 사람이다." 여러분이 개인적으로나 사회적으로나 성공하는 전문 프로그래머가 되기 위해서는 미치지 않으면 안된다. 그 일을 파이썬이 도와 줄 것이다.

이만 본인은 파이썬 프로젝트 Na에 대한 구상을 시작하고자 한다. 한국 파이썬 사용자모임에서 만날 수 있기를 바란다.

2001년 7월
Happy Python!

02. 파이썬이란 무엇인가?

파이썬이란 1990년 암스테르담의 귀도 반 로섬에 의해 만들어진 인터프리터 언어이다. 귀도는 이 파이썬이라는 이름을 어린이를 위한 프로그램인 ‘Monty Python's Flying Circus’라는 코미디 쇼에서 따왔다고 한다. 파이썬(Python)의 사전적인 뜻은 고대 신화 속의 파르나수스(Parnassus) 산의 동굴에 살던 큰 뱀으로서, 아폴로가 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있다. 대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있다.

현재 파이썬은 국내에서는 많이 알려져 있지 않지만 외국에서는 학습의 목적뿐만 아니라 실용적인 부분에서도 많이 사용되고 있는데 그 대표적인 예를 보면 레드햇 리눅스의 설치 프로그램인 아나콘다, 구글(Google)이나 인포시크(Infoseek)에서 사용되는 검색 프로그램, 야후의 많은 인터넷 서비스 프로그램 등이 파이썬으로 개발되었다.

또한 파이썬 프로그램은 공동작업과 유지보수가 매우 쉽고 편하기 때문에 이미 다른 언어로 작성된 많은 프로그램과 모듈들이 파이썬으로 다시 재구성되고 있는 상황이다. 국내에서도 그 가치를 인정받아 사용자층이 더욱 넓어져 가고 있고, 파이썬을 이용한 프로그램을 개발하는 기업체들이 늘어가고 있는 추세이다.

이제 파이썬의 특징과 장단점, 실제로 파이썬 프로그래밍을 하기 위한 환경을 구축하는 법에 대해서 알아보고 실제로 간단한 파이썬 프로그램을 작성해볼 것이다.

1) 파이썬의 특징

필자는 파이썬을 무척이나 좋아한다. 하지만 다른 언어들에 대한 혹평을 하지는 않는다. 각기 나름대로의 장점이 있기 때문이다. 하지만 파이썬에서는 다른 언어들에서 쉽게 찾아볼 수 없는 파이썬만의 독특한 특징들이 있다. 이 특징들을 살펴보면 파이썬의 매력을 흠뻑 느낄 수 있을 것이다. 파이썬의 특징들을 알아보는 과정을 통해서 왜 파이썬을 공부해야 하는지, 과연 이것에 시간을 투자할 만큼 가치가 있는 것인지에 대한 독자의 판단을 분명하게 해줄 것이라 믿는다.

인간다운 언어이다
프로그래밍이란 컴퓨터에 인간이 생각하는 것을 입력시키는 행위라고 할 수 있다. 앞으로 살펴볼 파이썬 문법들에서도 볼 수 있겠지만 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있도록 해주는 언어이다. 따라서 프로그래머는 굳이 컴퓨터식 사고 방식으로 프로그래밍을 하려고 애쓸 필요가 없다. 이제 곧 어떤 프로그램을 구상하자마자 생각한대로 쉽게 술술 써내려가는 당신의 모습에 놀라게 될 것이다. 아래 예문을 보면 이 말이 더 쉽게 이해될 것이다.

```
if 4 in [1,2,3,4]: print "4가 있습니다."
```

위의 예제는 다음처럼 읽을 수 있다.

> 만약 4가 1,2,3,4중에 있으면 "4가 있습니다."를 출력한다.

프로그램을 모르더라도 직관적으로 무엇을 뜻하는지 알 수 있지 않겠는가? 마치 영어문장을 읽는 듯한 착각에 빠져든다.

문법이 쉬워 빠르게 학습할 수 있다
어려운 문법과 수많은 규칙들에 둘러싸인 언어에서 탈피하고 싶지 않은가? 파이썬은 문법 자체가 아주 쉽고 간결하며, 사람의 사고 방식과 매우 닮아있다. 배우기 쉬운 언어, 활용하기 쉬운 언어가 가장 좋은 언어가 아닐까? 참고로 프로그래밍 경험이 있는 어떤 사람은 파이썬을 공부한지 단 하루만에 자신이 원하는 프로그램을 작성할 수 있었다고 한다.

> 보통 프로그래밍 경험이 있는 평범한 사람이라면 자료형에 대한 공부, 함수, 클래스 만드는 법, 라이브러리 및 내장함수 사용방법등에 대해서 익히는 데에는 1주일이면 충분하리라 생각된다.

강력하다
파이썬으로 프로그래머는 대부분의 모든 일들을 해낼 수가 있다. 물론 시스템 프로그래밍, 하드웨어 제어, 매우 복잡하고 많은 반복연산 등은 파이썬으로 개발하기가 어렵다. 하지만 이러한 몇 가지를 제외하면 파이썬으로 할 수 없는 것은 거의 없다고 해도 과언이 아니다.

또한 파이썬은 위의 약점을 극복할 수 있게끔 다른 언어로 만든 모듈을 파이썬 프로그램에 포함할 수가 있다. 파이썬과 C는 찰떡궁합이란 말이 있다. 즉, 빠른 속도를 필요로 하는 부분을 C로 만들어서 파이썬에 포함시키고 프로그래밍의 전반적인 뼈대는 파이썬으로 하자는 것이다. (정말 놀라울 정도로 영악한 언어가 아닌가?) 사실 파이썬 모듈 중에는 순수 파이썬만으로 제작된 것도 많지만 C로 만들어진 것도 많다. C로 만들어진 것들은 대부분 속도가 빠르다.

무료이다
파이썬은 오픈소스이므로 당연히 무료이다. 언제 어디서든 파이썬 패키지를 다운로드해 쓸 수 있고, 사용료를 지불해야 할 필요가 없다. 그렇다고 자신이 만든 프로그램을 무조건 공개해야 하는 것은 아니다.

간결하다
파이썬은 간결하다. 이 간결함은 파이썬을 만든 귀도(Guido)의 의도적인 산물이다. 이 간결함으로 인해 파이썬 프로그래밍을 하는 사람들은 잘 정리되어 있는 소스코드를 볼 수 있게 되었다. 다른 사람들의 소스 코드가 한눈에 들어오기 때문에 이 간결함은 공동 작업에 매우 큰 역할을 하게 되었다. 다음은 파이썬 프로그램의 예제이다.

```
# simple.py
languages = ['python', 'perl', 'c', 'java']

for lang in languages:
   if lang in ['python', 'perl']:
       print "%6s need interpreter" % lang
   elif lang in ['c', 'java']:
       print "%6s need compiler" % lang
   else:
       print "should not reach here"
```

위의 프로그램 소스 코드를 이해하려 하지는 말자. 이것을 이해할 수 있다면 당신은 이미 파이썬 중독자 일 것이다. 그냥 한번 구경해 보도록 하자. 다른 언어들에서 늘 보이는 단락을 구분하는 브레이스({, })문자들이 보이지 않는 것을 확인 할 수 있다. 또한, 줄을 참 잘 맞춘 코드라는 것도 확인 할 수 있다. 줄을 맞추지 않으면 실행이 되지 않는다. 파이썬 프로그래머는 이쁘게 보일려고 저렇게 줄맞추어 코딩을 하지 않는다. 실행이 되게 하기 위해서 줄을 맞추는 것이다. 이렇듯 줄을 맞추어 코드를 작성하는 행위가 가독성에 얼마나 큰 도움을 주는지 곧 느끼게 될 것이다.

프로그래밍이 재밌다
이 부분을 가장 강조하고 싶다. 필자에게 파이썬만큼 프로그래밍을 하는 순간을 즐기게 해준 언어는 없었던 것 같다. 파이썬은 다른 것에 신경 쓸 필요 없이 내가 하고자 하는 부분에만 집중할 수 있게 해주기 때문이다. 억지로 만든 프로그램과 즐기면서 만든 프로그램, 과연 어떤 프로그램이 좋을까? 리누즈 토발즈는 재미로 리눅스를 만들었다고 하지 않는가? 파이썬을 배우고 나면 다른 언어로 프로그래밍을 하는 것에 지루함을 느끼게 될 지도 모른다. 조심하자! ^^

개발속도가 빠르다
마지막으로 다음의 재미있는 말로 파이썬의 특징을 마무리하려 한다.

```
Life is too short, You need python.
```

파이썬의 엄청나게 빠른 개발 속도를 두고 유행처럼 퍼진 말이다. 이 유머스러운 문장은 이 책에서 계속 예제로 사용할 것이다.

2) 무엇을 할 수 있나?

좋은 언어와 나쁜 언어는 이미 정해진 걸까? 그렇다면 어떤것이 최고의 언어일까? 가만히 살펴보면 어떤 언어든지 강한 부분과 약한 부분이 존재한다. 어떤 프로그램을 만들 것인지에 따라 선택해야 할 언어도 달라진다. 한 언어만을 고집하고 그 언어로만 모든 것을 하겠다는 생각은 현실과는 맞지 않는다. 따라서 자신이 만들고자 하는 프로그램을 가장 잘 만들 수 있게 도와주는 언어가 어떤 것인지 알아내고 선택하는 것은 중요한 일이다. 하지만 할 수 있는 일과 할 수 없는 일을 가리기는 쉽지 않다. 왜냐하면 어떤 언어든지 할 수 없는 일은 거의 없기 때문이다. 하지만 한 프로그래밍 언어가 어떤 일에 적합한지에 대해서 아는 것은 매우 중요하다. 따라서 파이썬으로 하기에 적당한 일과 적당하지 않은 일에 대해서 알아보는 것은 매우 가치있는 일이 될 것이다.

파이썬으로 할 수 있는 일
파이썬으로 할 수 있는 일은 너무나 많다. 대부분의 컴퓨터 언어가 하는 일을 파이썬은 쉽고 깔끔하게 처리한다. 이것들에 대해서 나열하자면 끝도 없겠지만 대표적인 몇 가지의 예를 들어보도록 하자.

1. 시스템 유틸리티
파이썬은 운영체제(윈도우즈, 리눅스등)의 시스템 명령어들을 이용할 수 있는 도구들을 갖추고 있기 때문에 이러한 것들을 바탕으로 갖가지 시스템 관련한 유틸리티를 만드는 데 유리하다. 여러분은 시스템에서 사용중인 다른 유틸리티성 프로그램들을 하나로 뭉쳐서 큰 힘을 발휘하게 하는 프로그램들을 무수히 만들어 낼 수 있다.

2. GUI(Graphic User Interface) 프로그램
GUI 프로그래밍이라는 것은 쉽게 말해서 윈도우즈 창같은 프로그램을 만드는 것이다. 파이썬으로 GUI프로그램을 작성하는 것은 다른 언어로 하는 것보다 훨씬 쉽다. 대표적인 것으로 파이썬 프로그램을 설치할 때 함께 설치되는 Tkinter를 들 수 있다. 실제로 Tkinter를 이용한 파이썬 GUI프로그램의 프로그램 소스는 매우 간단하다. 놀라운 사실은 Tkinter를 이용하면 소스코드 단 5줄만으로도 윈도우즈 창을 띄울 수 있다는 것이다. 이 외에도 wxPython, PyQT, PyGTK등의 Tkinter보다 빠른 속도와 미려한 윈도우 화면을 자랑하는 것들도 있다.

3. C/C++과의 결합
파이썬은 접착(glue)언어라고도 불리운다. 그 이유는 다른 언어와 함께 잘 어울릴 수 있기 때문이다. C로 만든 프로그램을 파이썬에서 쓸 수 있으며, 파이썬으로 만든 프로그램을 C에서 역시 쓸 수가 있다.

필자는 과거 증권사에서 파이썬과 C를 결합하여 효과를 본 경험이 있다. ChartDirector라는 그래프를 그려주는 파이썬으로 작성된 라이브러리와 기존 C모듈을 접착하여 활용했었다. 데이터를 가져오는 부분은 Legacy(기존프로그램)시스템을 이용할 수 있었기 때문에 추가 개발없이 매우 빠르게 많은 챠트 프로그램을 손쉽게 작성할 수 있었다.

4. CGI 프로그래밍
우리는 익스플로러나 넷스케이프라는 프로그램을 이용해서 WWW을 이용한다. 누구나 한번쯤 웹 서핑을 하면서 게시판이나 방명록에 글을 남겨 본 적이 있을 것이다. 그러한 게시판이나 방명록을 바로 CGI프로그램이라고 한다. 조금 어려운 말로 풀이 해 보면 CGI는 웹브라우져를 이용하는 사용자가 서버 프로그램(게시판, 방명록)등을 이용할 수 있게 해주는 도구라고 할 수 있다. CGI는 Common Gateway Interface의 약자이다. 이 책 인터넷 프로그래밍 파트에서는 파이썬으로 간단한 CGI프로그램을 실제로 작성해 볼 것이다.

5. 수치연산 프로그래밍
사실 파이썬은 수치연산 프로그래밍에 적합한 언어는 아니다. 왜냐하면 복잡하고 연산이 많다면 C와 같은 언어로 하는 것이 더 빠르기 때문이다. 하지만 파이썬에서는 Numeric Python이라는 수치 연산 모듈을 제공한다. 이 Numeric Python은 C로 작성되었기 때문에 매우 빠르게 수학연산을 수행한다. 이 모듈을 이용하면 파이썬에서 수치연산을 빠르게 할 수 있을 것이다.

6. 데이터베이스 프로그래밍
파이썬은 Sybase, Infomix, 오라클, MySQL, Postgresql등의 데이터 베이스에 접근할 수 있게 해주는 도구들을 제공한다. 이 책에서는 MySQL을 이용하는 파이썬 프로그램을 직접 만들어 볼 것이다. 또한 이런 굵직한 데이터베이스를 직접 이용하는 것 외에도 파이썬에는 재미있는 함수가 하나 있다. 바로 pickle이라는 모듈이다. 이 모듈은 파이썬에서 쓰이는 자료들을 변형없이 그대로 파일에 저장하고 불러오는 일들을 해 준다. 이 곳에서는 pickle을 어떻게 사용하고 활용하는지에 대해서도 알아본다.

파이썬으로 할 수 없는 일
파이썬으로 도스나 리눅스 같은 운영체제, 엄청난 횟수의 반복과 연산을 필요로 하는 프로그램 또는 데이터 압축 알고리즘 개발 프로그램등을 만들기는 어렵다. 즉, 대단히 빠른 속도를 요구하거나 하드웨어를 직접 건드려야 하는 프로그램에는 어울리지 않는다.

3) 파이썬 설치하기

자신의 컴퓨터에 파이썬을 설치해보자. 여기서는 윈도우즈와 리눅스의 경우만을 다루도록 하겠다. 다른 시스템에서는 파이썬 홈페이지의 설명서를 참고하기를 바란다.

윈도우에서 파이썬 설치
윈도우즈의 경우에는 설치가 정말 쉽다. 우선 http://www.python.org (파이썬 공식 홈페이지)에서 윈도우용 파이썬 언어 패키지를 다운로드한다. 윈도우즈 95, 98, NT, 2000, ME에 관계없이 다운받을 파이썬 패키지는 동일하고 또한 동일한 방법으로 설치를 한다. 이 파일을 받으면 확장자가 exe 또는 msi이다. 실행시키면 바로 설치가 시작된다.

설치할 때 선택하는 부분이 있는 데, 무슨 내용인지 잘 모른다면 모든 것을 선택하도록 하자. 설치가 끝난 후 해주어야 할 일은 어느 디렉토리에서나 파이썬이 실행될 수 있도록 환경변수에 다음과 같은 줄을 넣어준다.

```
PATH="%PATH%;C:\Python27"
```

위의 Python27이란 디렉토리를 설치할 때 경로나 이름을 다르게 하였다면 자신에게 맞게끔 고쳐야 한다.

리눅스에서 파이썬 설치
리눅스 사용자라면 이미 디폴트로 파이썬이 설치되어 있을 것이다.

```
$ python -V
```

위의 명령어를 치면 파이썬 버전을 확인할 수 있다. 만약 파이썬 버전이 2.7 보다 낮다면 최신 버전인 2.7를 설치하도록 하자. 역시 http://www.python.org에 접속해 Python-2.7.tgz를 다운로드한다.
여기서는 소스를 컴파일하여 설치하는 방법에 대해서 알아보도록 한다.

우선 다음처럼 압축을 푼다.

```
$ tar xvzf Python-2.7.tgz
```

다음에 해당 디렉토리로 이동한다.

```
$ cd Python-2.7
```

Makefile 파일을 만들기 위해서 configure를 실행한다.

```
$ ./configure
```

파이썬 소스를 컴파일 한다.

```
$ make
```

루트 계정으로 인스톨한다.

```
$ su
# make install
```

4) 파이썬 둘러보기

도대체 파이썬이란 언어는 어떻게 생겼는지, 어떤식으로 코드를 작성하는지 간단히 알아보자.
파이썬이란 언어를 자세히 탐구하기 전에 전체적인 모습을 한번 훑어보는 것은 매우 유익한 일이 될 것이다.

"백문이 불여 일견, 백견이 불여 일타"라 했다.
따라해 보자.

따라해 보기
먼저 파이썬이라는 것이 도대체 어떻게 생긴 것인지 보도록 하자.
[시작]메뉴에서 [프로그램] -> [Python 2.X] -> [Python (Command Line)]을 선택하자.

다음과 같은 화면을 볼 수 있다.

```
Python 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

위와 같은 것을 대화형 인터프리터라고 하는데 앞으로 이 책에서는 이 인터프리터로 파이썬 프로그래밍의 기초적인 사항들에 대해서 설명할 것이다. 대화형 인터프리터를 종료하는 방법은 Ctrl-Z 키를 누르는 것이다(유닉스 계열에서는 Ctrl + D).

또는 다음의 예와 같이 sys 모듈을 사용하는 방법이 있다.

```
>>> import sys
>>> sys.exit()
```

앞으로 보게 될 내용들은 2부에서 다시 자세하게 다룰 것이니 이해가 되지 않는다고 절망하거나 너무 오래동안 고심하지 말도록 하자.

다음과 같이 입력해 보자.
1 **더하기** 2는 3이라는 값을 출력한다.

```
>>> 1 + 2
3
```

나눗셈과 곱셈 역시 예상한대로의 결과값을 보여준다.

```
>>> 3 / 2.4
1.25
>>> 3 * 9
27
>>>
```

a에 1을 b에 2를 **대입**한다음 a와 b를 더하면 3이란 결과값을 보여준다.

```
>>> a = 1
>>> b = 2
>>> a + b
3
```

a 라는 변수에 "Python"이라는 값을 대입한 다음 print a를 해 주면 a의 값을 출력해 준다.

```
>>> a = "Python"
>>> print a
Python
```

파이썬에서는 **복소수**도 지원한다.

```
>>> a = 2 + 3j
>>> b = 3
>>> a * b
(6+9j)
```

a에 2+3j 라는 값을 설정하였다. 여기서 2+3j란 복소수를 의미하는 것이다. 보통 우리는 고등학교때 복소수를 표시할 때 알파벳 i를 이용해서 2 + 3i처럼 사용했지만 파이썬에서는 j를 사용한다. 위의 예는 2 + 3j 와 3을 곱하는 방법이다. 당연히 결과 값으로 6+9j가 될 것이다.

다음은 간단한 **조건문 if**를 이용한 예제이다.

```
>>> a = 3
>>> if a> 1:
...     print "a > 1"
```

```
...
a > 1
```

"..."이 의미하는 것은 아직 문장이 끝나지 않았음을 알려주는 것이다.

다음은 **for**를 이용해서 [1, 2, 3]안의 값들을 하나씩 출력해 주는 것을 보여준다.

```
>>> for a in [1,2,3]:
...     print a
...
1
2
3
```

위의 예는 대괄호([]) 사이에 있는 값들을 하나씩 출력해 준다. 위 코드의 의미는 "[1, 2, 3]에서 앞에서부터 하나씩 꺼내어 a라는 변수에 대입한 후 print a를 수행하라 " 이다. 당연히 a 에 차례로 1, 2, 3이란 값이 대입될 것이고 print a에 의해서 그 값이 차례대로 출력 될 것이다.

다음은 **while**을 이용하는 모습이다.

```
>>> i = 0
>>> while i < 3:
...     i = i+1
...     print i
...
1
2
3
```

while 이란 영어단어는 " ~인 동안 " 이란 뜻이다. 위의 예제는 "i값이 3보다 작다"인 동안 "i = i + 1"과 "print i"를 수행하라는 말이다. i = i + 1이란 문장은 i 의 값을 1씩 더하게 한다. i값이 3보다 커지게 되면 while문을 빠져 나가게 된다.

파이썬의 **함수**는 다음처럼 생겼다.

```
>>>def sum(a, b):
...    return a+b
...
>>>print sum(3,4)
7
```

파이썬에서 def는 함수를 만들 때 사용되는 명령어이다. 위의 예는 sum이란 함수를 만들고 그 함수를 어떻게 사용하는지에 대한 예를 보여준다. sum(a, b)에서 a, b는 입력값이고 a+b 는 리턴값이다. 즉 3, 4가 입력으로 들어오면 3+4를 수행하고 그 결과값인 7을 돌려준다.

이상과 같이 기초적인 파이썬 문법들에 대해서 간략하게 알아보았다.

에디터로 파이썬 프로그램 작성하기
이미 눈치를 챘을지도 모르지만 독자들이 유용한 파이썬 프로그램을 작성하기 위해서는 인터프리터보다는 에디터를 이용하여 파이썬 프로그램을 작성하는 것이 좋다. 에디터라는 것은 문서를 편집할 수 있는 프로그램을 말한다. 독자가 즐겨 사용하는 에디터가 없다면 필자는 에디트 플러스라는 프로그램을 적극 추천한다. 에디트 플러스 프로그램의 설치법과 사용법에 대한 사항이 부록에 자세하게 나와 있으니 참고하기 바란다.

다음과 같은 프로그램을 에디터로 직접 작성해 보자.

```
# hello.py
print "Hello world"
```

이러한 내용을 담은 파일을 "hello.py"라는 이름으로 저장하도록 하자. 위의 파일에서 "# hello.py"라는 문장은 주석이다. '#'로 시작하는 문장은 '#'부터 시작해서 그 줄 끝까지 프로그램 수행에 전혀 영향을 끼치지 않는다. 주석은 프로그래머를 위한 것으로 프로그램 소스에 설명문을 달 때 사용하게 된다.

에디터로 파이썬 프로그램을 작성한 후 저장할 때 파일 이름의 확장자명을 항상 py로 해주도록 하자. py는 파이썬 파일임을 알려주는 관례적인 확장자명이다. 이제 이 hello.py라는 프로그램을 실행시키기 위해서 도스창을 열고 hello.py라는 파일이 저장된 곳으로 이동한 후 다음과 같이 입력하자. (여기서는 hello.py라는 파일이 C:\Python이란 디렉토리에 있다고 가정을 하자.)

```
C:\WINDOWS〉 cd \Python
C:\Python〉 python hello.py
Hello world
```

위와 같은 결과값을 볼 수 있을 것이다. 위와 같은 결과 값을 볼 수 없는 독자라면 환경변수에 PATH설정이 제대로 되었는지, hello.py라는 파일이 이동한 디렉토리에 존재하는지에 대해서 다시 한번 살펴 보도록 하자. (참고 - 만약 에디트 플러스라는 에디터를 사용한다면 보다 쉽게 에디터로 작성한 파이썬 프로그램을 실행할 수 가 있다.)

위에서는 "Hello world"라는 문장을 출력하는 단순한 프로그램을 에디터로 작성했지만 보통 에디터로 작성하는 프로그램은 꽤 여러 줄로 이루어 질 것이다. 에디터로 만든 프로그램 파일은 언제나 다시 사용할 수 있다. 대화형 인터프리터에서 만든 프로그램은 인터프리터를 종료함과 동시에 사라지게 되지만 에디터로 만든 프로그램은 파일로 존재하기 때문에 언제고 다시 사용할 수 있다.

왜 대부분 에디터를 이용해서 파이썬 프로그램을 작성해야하는지 이제 이해가 될 것이다.

리눅스에서 파이썬 프로그램 실행하기

by 이규민 2010.01.11

(우분투 리눅스 9.10을 기준으로 합니다. 우분투나 그 계열에 속하는 쿠분투 등이 아닌 경우에 실행법이 다를 수도 있습니다.)
리 눅스에서 파이썬은 명령을 수행하는 '터미널'에서 실행됩니다. 우분투 리눅스의 경우, 파이썬을 수행하는 명령어는 python이며, 이를 명령하면

```
Python 2.6.4 (r264:75706, Dec 7 2009, 18:45:15) [GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

이라고 파이썬이 터미널 내에서 실행되게 됩니다.

파이썬으로 py 스크립트를 실행하기 위해서는, "python 파일.py"라는 명령어를 써 줘야 합니다. 기본 디렉토리가 home폴더이므로, home폴더에 있는 파일은 위의 명령어처럼 해주면 됩니다. 파일 이름에 공백이 있는 경우는 꼭 파일 이름을 ""로 감싸 줘야 합니다. 혹 다른 디텍토리에 있는 파일이라면, /home/(사용자 이름)/문서/hello.py 와 같이 디텍토리 전체를 입력하면 됩니다. 예를들어 ABC 사용자가 '문서' 폴더에 있는 'simple calculator.py'를 실행하고자 한다면, 터미널에 python /home/ABC/문서/"simple calculator.py" 라고 명령합니다. (영어 대소문자 구분을 해야 합니다. UNIX에서 파생된 Linux는 명령어에서 대소문자를 구분합니다.)

03. 자료형과 제어문

파이썬 프로그래밍에서 가장 기본이 되는 자료형과 제어문에 대해서 다룬다. 여기서 다룰 내용은 다소 딱딱하고 재미는 없겠지만 파이썬이란 언어를 능숙하게 다루기 위해서 꼭 필요한 부분이므로 인내심을 가지고 따라와 주기를 바란다.

1) 자료형

자료형이란 프로그래밍을 할 때 쓰이는 숫자, 문자열등의 자료 형태로 저장되는 그 모든 것을 뜻한다. 프로그램의 가장 기본이 되고 핵심적인 단위가 되는 것이 바로 자료형이다. 따라서 자료형을 충분히 이해하지 않고 프로그래밍을 시작하려는 것은 기초 공사가 마무리되지 않은 상태에서 빌딩을 세우는 것과 같다. 프로그래밍 언어를 배울 때 "그 언어의 자료형을 알게 된다면 이미 그 언어의 반을 터득한 것이나 다름없다"라는 말이 있다. 자료형은 가장 기초가 되는 중요한 부분이니 주의를 기울여 자세히 살펴보도록 하자.

우리가 이곳에서 기본적으로 알아야 할 자료형에는 숫자, 문자열, 리스트, 터플, 딕셔너리 등이 있다. 이것들에 대해서 하나씩 자세하게 알아 보도록 하자.

[1] 숫자형 (Number)

숫자형이란 숫자 형태로 이루어진 자료형으로 우리가 이미 아는 것들이다. 우리가 흔히 사용해 왔던 것들을 생각해 보자. 123과 같은 정수, 12.34 같은 실수, 공대생이라면 필수적으로 알아야 할 복소수(1 + 2j) 같은 것들도 있고, 드물게 쓰이긴 하지만 8진수나 16진수 같은 것들도 있다.

이런 숫자들을 파이썬에서는 어떻게 표현하고 적용하는 지 알아보자.

```
항목          사용 예
---------------------------------
정수          123, -345, 0
실수          123.45, -1234.5, 3.4e10
복소수        1 + 2j, -3j
8진수         034, 025
16진수        0x2A, 0xFF
```

위의 표는 숫자들이 파이썬에서 어떻게 표현되는지를 간략하게 보여준다.

정수형(Integer)
정수형이란 말 그대로 정수를 뜻하는 숫자를 말한다. 아래의 a는 각각의 숫자를 대입한 변수이다.

다음 예는 양의 정수와 음의 정수, 숫자 0을 변수에 대입하는 예제이다.

```
>>> a = 123
>>> a = -178
>>> a = 0
```

소수점 포함된 것(Floating-point)
아래의 첫 번째 예와 두 번째 예는 우리가 늘 사용하는 방식이다. 그리고 세 번째와 네 번째 방법은 컴퓨터식 지수 표현법이다. 지수 표현법으로 파이썬에서는 4.24e10 또는 4.24E10처럼 표현한다. (e, E둘중 어느 것을 사용해도 무방하다.)

```
>>> a = 1.2
>>> a = -3.45
>>> a = 4.24E10   # 4.24 * 10의 10승
>>> a = 4.24e-10  # 4.24 * 10의 마이너스 10승
```

8비트형 수(Octal)
8진수를 만들기 위해서는 숫자가 0(숫자 0)으로 시작하면 된다.

```
>>> a = 0177
```

16비트형 수(Hexadecimal)
16진수를 만들기 위해서는 숫자가 0x로 시작하면 된다.

```
>>> a = 0x8ff
>>> b = 0xABC
```

8진수나 16진수는 잘 사용하지 않는 형태의 숫자 자료형이다.

복소수 (Complex number)
보통 우리는 중고등학교 시절에 ‘j’ 대신 ‘i’를 사용했을 것이다. 파이썬은 ‘i’ 대신 ‘j’를 사용한다. ‘j’를 써도 되고 'J'를 써도 된다.

```
>>> a = 1+2j
>>> b = 3-4J
```

복소수를 활용하는 예들을 몇가지 보도록 하자. 복소수에는 복소수가 자체적으로 가지고 있는 내장함수가 있다. 그것들을 이용하면 좀더 다양한 방법으로 복소수를 사용할 수 있게 된다.

다음의 복소수 예들을 보자.
복소수.real은 복소수의 실수 부분을 돌려준다.

```
>>> a = 1+2j
>>> a.real
1.0
```

복소수.imag는 복소수의 허수 부분을 돌려준다.

```
>>> a = 1+2j
>>> a.imag
2.0
```

복소수.conjugate()는 복소수의 켤레 복소수를 돌려준다.

```
>>> a = 1+2j
>>> a.conjugate()
(1-2j)
```

abs(복소수)는 복소수의 절대값을 돌려준다. (1+2j의 절대값은 루트 1^2+2^2 이다.)

```
>>> a = 1+2j
>>> abs(a)
2.2360679774997898
```

숫자 연산
프로그래밍을 한번도 해 본적이 없는 독자라도 사칙연산(+, -, *, /)은 알고 있을 것이다.
파이썬에서도 역시 계산기와 마찬가지로 아래의 연산자를 이용하여 사칙연산을 수행한다.

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
```

여기서 주의해야 할 것은 나눗셈 연산자 '/'이다. 즉 3 / 4는 0.75라는 값이지만 둘 다 정수로 사용하면 소수점 이하의 자리를 무시하고 정수 형태의 값을 되돌려 주게 된다. 따라서 아래의 예 처럼 소수점 형태의 결과값을 얻기 위해서는 한 개의 값을 실수형으로 써주어야 한다.

```
>>> 3 / 4
0
```

```
>>> 3 / 4.0
0.75
>>> 3.0 / 4
0.75
```

또 다른 방법이 있다. 다음 예처럼 float라는 함수를 사용하면 소숫점 형태의 결과값을 얻을 수 있다.

```
>>> a = 3
>>> b = 4
>>> float(a) / b
0.75
```

이러한 방법을 ‘ 형변환 ’ 이라고 한다. 즉, float(a)는 a라는 변수에 대입된 3이라는 값을 실수값 3.0으로 바꾸어 준다. 파이썬에는 다양한 형변환 기법들이 존재하는데 그것들은 4장에서 자세히 다루도록 하자.

다음에 알아야 될 연산자로 ** 라는 연산자가 있다. 이것은 x ** y처럼 사용되었을 때 x의 y승 값을 돌려 준다. 다음의 예를 통해 알아보자.

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

프로그래밍을 접해 본 적이 없는 독자라면 %연산자는 본 적이 없을 것이다. %는 나머지 값을 반환하는 연산자이다. 7을 3으로 나누면 나머지는 1이 될 것이고 3을 7로 나누면 나머지는 3이 될 것이다. 다음의 예로 확인해 보자.

```
>>> 7 % 3
1
>>> 3 % 7
3
```

[2] 문자열 (String)

문자열이란 문장을 뜻한다. 예를 들어 다음과 같은 것들이 문자열이다.

```
"Life is too short, You need Python"
"a"
"123"
```

위의 예를 보면 이중 인용부호(“)로 둘러싸인 것은 모두다 문자열이 되는 것을 알 수 있다. “ 123"은 숫자인데 왜 문자열인가? 라는 의문이 드는 독자는 이중 인용부호(”)로 둘러싸인 것은 모두 문자열이라고 생각하면 될 것이다.

문자열 만드는 방법 4가지
위의 예에서는 문자열을 만들 때 이중 인용부호(“)만을 사용했지만 이 외에도 문자열을 만드는 방법은 세 가지가 더 있다. 파이썬에서 문자열을 만드는 방법은 다음과 같이 네 가지로 구분된다.

```
“ Hello World ”
'Python is fun'
"""Life is too short, You need python"""
'''Life is too shor, You need python'''
```

문자열을 만들기 위해서는 위의 예에서 보듯이 이중 인용부호(“)로 문자열의 양쪽을 둘러싸거나 단일 인용부호(') 또는 이중 인용부호나 단일 인용부호 세 개를 연속으로 쓰는(""", ''')를 양쪽으로 둘러싸면 된다. 그렇다면 왜 단순함을 좋아하는 파이썬에서 이렇듯 다양한 문자열 만드는 방법을 가지게 되었을까? 그 이유에 대해서 알아보도록 하자.

문자열 내에 (') 또는 (") 을 포함시키고 싶을 때
문자열을 만들어주는 주인공은 (') 와 (")이다. 하지만 문자열 내에 (') 와 (")를 포함시켜야 할 경우가 있다. 이 때는 좀 더 특별한 기술이 필요하다. 예제를 하나씩 살펴보면서 원리를 익혀보도록 하자.

예 1) 단일 인용부호(')를 포함시키고 싶을 때

```
Python's favorite food is perl
```

예 1과 같은 문자열을 변수에 저장하고 싶다고 가정해보자. Python's에서 보듯이 (')가 포함되어 있다. 이럴 때는 다음과 같이 문자열을 이중 인용부호(" ")로 둘러싸야 한다. 이중 인용부호(" ")안에 들어 있는 단일 인용부호(')는 문자열을 나타내기 위한 기호로 인식되지 않는다.

```
>>> food = "Python's favorite food is perl"
```

시험삼아 다음과 같이 (")이 아닌 (')로 문자열을 둘러싼 뒤 실행시켜 보자. 'Python' 이 문자열로 인식되어 에러(Syntax Error)가 날 것이다.

```
>>> food = 'Python's favorite food is perl'
```

예 2) 이중 인용부호(")를 포함시키고 싶을 때

```
"Python is very easy." he says.
```

예 2와 같이 이중 인용부호(")가 포함된 문자열이라면 어떻게 해야 (") 이 제대로 표현될까? 다음과 같이 그 문자열을 (' ')으로 둘러싸면 된다.

```
>>> say = '"Python is very easy." he says.'
```

이렇게 단일인용 부호(' ')안에 사용된 이중인용부호(")는 문자열을 나타내는 기호로 인식되지 않는다.

예 3) ‘ \ ’(역슬래시)로 (')과 (")를 문자열에 포함시키기

```
>>> food = 'Python\'s favorite food is perl'
>>> say = "\"Python is very easy.\" he says."
```

(')나 (")를 문자열에 포함시킬 수 있는 또 다른 방법은 '\'(역슬래시)를 이용하는 것이다. 즉 (\')가 문자열 내에 삽입되면 그것은 문자열을 둘러싸는 기호의 의미가 아니라 문자 (') 그 자체를 뜻하게 된다. (\") 역시 마찬가지이다. 어떤 것을 사용할 것인지는 각자의 선택이다. 대화형 인터프리터를 실행시킨 뒤 위의 예를 꼭 직접 실행해 보도록 하자.

여러 줄 짜리 문자열 처리
문자열이 항상 한 줄 짜리만 있는 것은 아니다. 다음과 같이 여러 줄 짜리 문자열이 있을 때는 어떻게 처리해야 할까?

```
Life is too short
You need python
Python is powerful language
```

이러한 문자열을 변수에 대입하려면 어떻게 하겠는가?

예1) 줄바꿈 문자인 '\n' 삽입

```
>>> multiline = "Life is too short\nYou need python\nPython is powerful language"
```

위의 예처럼 줄바꿈 문자인 '\n'을 삽입하는 방법이 있지만 읽기에 너무 불편하고 너무 줄이 길어지는 단점이 있다. 이것을 극복하기 위해 파이썬에서는 다음과 같이 (“ ” “)를 이용한다.

예 2) 연속된 이중인용부호 세 개(""") 이용

```
multiline=“ ” “
Life is too short
You need python
Python is powerful language
“ ” “
```

위 예에서도 확인할 수 있듯이 문자열이 여러줄일 경우 위와같은 방식이 상당히 유리하고 깔끔하다는 것을 알 수 있을 것이다.

```
[참고] 이스케이프 코드

여러 줄 짜리 문장을 처리할 때 '\n'과 같은 역슬래시 문자를 이용한
이스케이프 코드를 사용했다.

이와 같은 문자를 이스케이프 코드라고 부르는데,
출력물을 보기 좋게 정렬하거나 그 외 특별한 용도로 자주 이용된다.
몇 가지 이스케이프 코드를 정리하면 다음과 같다.
```

```
---------------------------------------
코드              설명
---------------------------------------
\n              개행 (줄바꿈)
\v              수직 탭
\t              수평 탭
\r              캐리지 리턴
\f              폼 피드
\a              벨 소리
\b              백 스페이스
\000            널문자
\\              문자 "\"
\'              단일 인용부호(')
\"              이중 인용부호(")
---------------------------------------

이중에서 활용빈도가 높은 것은 \n, \t, \\, \', \"이다.
나머지는 대부분의 프로그램에서 잘 쓰이지 않는다.
```

문자열 연산
파이썬에서는 문자열을 더하고 곱할 수 있다. 이것은 다른 언어에서는 쉽게 찾아 볼 수 없는 재미있는 기능이다. 우리의 생각을 그대로 반영해주는 파이썬만의 장점이라고 할 수 있다.

문자열을 더하거나 곱하는 방법에 대해 알아보기로 하자.

예 1) 문자열 합치기(Concatenation)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> print head + tail
Python is fun!
```

예 1의 세번 째 라인을 보자. 복잡하게 생각하지 말고 눈에 보이는 대로 생각해보자. “ Python"이라는 head변수와 " is fun"이라는 tail변수를 더한 것이다. 결과는 ‘ Python is fun! ’ 이다. 즉, head와 tail변수가 "+"에 의해 합쳐진 것이다.

직접 실행해보고 결과값이 제시한 것과 똑같이 나오는 지 확인해보자.

예 2) 문자열 곱하기

```
>>> a = "python"
>>> print a * 2
pythonpython
```

여기도 마찬가지로 ‘ * ’의 의미는 숫자 곱하기의 의미와는 다르게 사용되었다. 여기서 사용된 ‘ * ’는 문자열의 반복을 뜻하는 의미로 사용되었다. 굳이 예를 설명할 필요가 없을 정도로 직관적이다. "print a * 2"라는 문장은 a를 두번 반복해 출력하라는 뜻이다.

문자열 곱하기를 좀 더 응용해보자. 다음과 같은 소스를 에디터로 작성해 실행시켜보자.

```
# multistring.py

print "=" * 50
print "My Program"
print "=" * 50
```

결과값은 다음과 같이 나타날 것이다.

```
==================================================
My Program
==================================================
```

위와 같은 표현은 자주 사용하게 된다. 프로그램을 만들고 실행시켰을 때 출력되는 화면 제일 상단에 프로그램 제목으로 위처럼 만들면 보기 좋지 않겠는가?

인덱싱과 슬라이싱

인덱싱(indexing)이란 무엇인가를 ‘ 가리킨다 ’는 의미이고, 슬라이싱(slicing)은 무엇인가를 ‘ 잘라낸다 ’라는 의미이다. 이것들을 생각하면서 다음의 예를 따라해 보도록 하자.

```
>>> a = "Life is too short, You need Python"
```

```
Life is too short, You need Python
0         1         2         3
```

```
01234567890123456789012345678901234
```

각 문자열의 문자마다 번호를 매겨 보았다. 즉 "Life is too short, You need Python"이라는 문자열에서 'L'은 첫 번째 자리를 뜻하는 숫자인 0을 바로 다음인 'i'는 1을 이런식으로 계속 번호를 붙인 것이다. 중간쯤에 있는 "short"의 s는 12라는 번호가 된다.

그리고 다음 예를 실행해 보자.

```
>>> a = "Life is too short, You need Python"
>>> a[3]
'e'
```

a[3] 이 뜻하는 것은 a라는 문자열의 네 번째 문자인 'e'를 말한다. 프로그래밍을 처음 접하는 독자라면 a[3]에서 3이란 숫자는 세 번째인데 왜 네 번째 문자를 말한다는 것인지 의아할 것이다.

이 부분이 사실 필자도 가끔 많이 헷갈리는 부분인데, 이렇게 생각하면 쉽게 알 수 있을 것이다.

> “ 파이썬은 0부터 숫자를 센다 ” .

위의 문자열을 파이썬은 이렇게 바라보고 있는 것이다.

```
a[0]: 'L', a[1]: 'i', a[2]: 'f', a[3]: 'e', a[4]: ' ', a[5]: 'i', a[6]: 's',
a[7]: ' ',....
```

0부터 숫자를 센다는 것이 처음에는 익숙하지 않겠지만 이것도 하다 보면 자연스럽게 될 것이다. 위의 예에서 보듯이 a[3]이란 것은 문자열 내 특정한 값을 뽑아내는 역할을 해준다. 이러한 것을 인덱싱(Indexing)이라고 부른다.

몇가지를 인덱싱을 더 해 보도록 하자.

```
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

마지막의 a[-1]이 뜻하는 것은 뭘까? 눈치 빠른 독자는 이미 알겠지만 바로 문자열을 뒤에서부터 읽기 위해서 마이너스(-)기호를 붙이는 것이다. 즉 a[-1]은 뒤에서부터 세어서 첫 번째가 되는 문자를 말한다. a는 "Life is too short, You need Python"이라는 문장이므로 뒤에서부터 첫 번째 문자는 가장 마지막 문자인 'n'이 될 것이다.

뒤에서부터 첫 번째 문자를 표시 할 때 a[-0]이라고 해야 하지 않겠는가? 라는 의문이 들수도 있겠지만 잘 생각해 보자. 0과 -0은 똑같은 것이기 때문에 a[-0]이라는 것은 a[0]과 똑같은 값을 보여준다.

```
>>> a[-0]
'L'
```

계속해서 몇가지 예를 더 보자.

```
>>> a[-2]
'o'
>>> a[-5]
'y'
```

위의 첫 번째 예는 뒤에서부터 두 번째 문자를 가리키는 것이고 두 번째 예는 뒤에서부터 다섯 번째 문자를 가리키는 것이다. 그렇다면 "Life is too short, You need Python"이라는 문자열에서 단순히 한 문자만을 뽑아내는 것이 아니라 'Life'또는 'You'같은 단어들을 뽑아낼 수 있는 방법은 없을까?

다음과 같이 하면 될 것이다.

```
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
```

위의 방법처럼 할 수도 있겠지만 파이썬에서는 보다 더 좋은 방법을 제공한다. 바로 슬라이싱(Slicing)이라는 기법이다.

위의 예는 슬라이싱 기법으로 다음과 같이 간단하게 처리할 수 있다. (주의 - 지금까지의 예제는 계속 이어지는 것이기 때문에 이미 인터프리터를 닫고 다시 시작하는 독자라면 >>> a = "Life is too short, You need Python"이라는 것을 먼저 수행한 뒤 다음의 예제들을 따라하도록 하자)

```
>>> a[0:4]
'Life'
```

a[0:4]가 뜻하는 것은 a라는 문자열 즉, "Life is too short, You need Python"이라는 문장에서 0부터 4까지의 문자를 뽑아낸다는 뜻이다. 하지만 다음과 같은 의문이 들 것이다.

a[0]은 'L', a[1]은 'i', a[2]은 'f', a[3]은 'e'이니까 a[0:3]만으로도 'Life'라는 단어를 뽑아낼 수 있지 않을까?
다음의 예를 보도록 하자.

```
>>> a[0:3]
'Lif'
```

이렇게 되는 이유는 간단하다. a[시작번호: 끝번호] 처럼 쓰면 끝번호에 해당하는 것은 포함이 되지 않는다.

a[0:3]을 수식으로 나타내면 다음과 같다.

```
0 <= a < 3
```

즉 위의 수식을 만족하는 a는 a[0], a[1], a[2] 일 것이다. 따라서 a[0:3]은 'Lif'이고 a[0:4]는 'Life'가 되는 것이다. 이 부분이 문자열 연산에서 가장 혼동하기 쉬운 부분이니 스스로 많이 연습해 보기를 바란다.

슬라이싱의 예를 조금 더 보도록 하자.

```
>>> a[0:5]
'Life '
```

위의 예는 a[0] + a[1] + a[2] + a[3] + a[4]와 동일하다. a[4]라는 것은 공백문자 ' '이기 때문에 'Life'가 아닌 'Life '가 되는 것이다. 공백문자 역시 'L', 'i' , 'f', 'e'와 동일하게 취급되는 것을 잊지 말도록 하자. 'Life'와 'Life '는 완전히 다른 문자열이다.

항상 시작번호가 '0'일 필요는 없다.

```
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

a[시작번호:끝번호]에서 끝번호 부분을 생략하면 시작번호부터 그 문자열의 끝까지를 뽑아내게 된다.

```
>>> a[19:]
'You need Python'
```

a[시작번호:끝번호]에서 시작번호를 생략하면 그 문자열의 처음부터 끝번호까지 뽑아내게 된다.

```
>>> a[:17]
'Life is too short'
```

a[시작번호:끝번호]에서 시작번호와 끝번호를 생략하면

```
>>> a[:]
'Life is too short, You need Python'
```

a[시작번호:끝번호]에서 시작번호와 끝번호를 모두 생략했기 때문에 이것은 처음부터 끝까지를 말하게 되므로 위와 같은 결과를 보여주는 것이다.

슬라이싱에서 역시 인덱싱과 마찬가지로 '-'기호를 사용할 수가 있다.

```
>>> a[19:-7]
'You need'
```

a[19:-7]이 뜻하는 것은 a[19]에서부터 a[-7]까지를 말한다. 이것 역시 a[-7]은 포함하지 않는다.

자주 사용되는 슬라이싱 예
다음은 자주 사용하게 되는 슬라이싱 기법 중의 하나이다.

```
>>> a = "20010331Rainy"
>>> date = a[:8]
>>> weather = a[8:]
>>> date
'20010331'
>>> weather
'Rainy'
```

a 라는 문자열을 두 부분으로 나누는 기법이다. 동일한 숫자 '8'을 기준으로 a[:8], a[8:]처럼 사용을 하였다. a[:8]은 a[8]이 포함이 안되고 a[8:]은 a[8]을 포함하기 때문에 8을 기준으로 해서 두 부분으로 나누게 된 것이다. 위의 예에서는 "20010331Rainy"라는 문자열을 날짜를 나타내는 부분인 '20010331'과 날씨를 나타내는 부분인 'Rainy'로 나누는 방법을 보여준다.

위의 문자열 "20010331Rainy"라는 것을 연도인 2001과 월과 일을 나타내는 0331 그리고 날씨를 나타내는 Rainy를 세 부분으로 나누려면 다음과 같이 할 수 있다.

```
>>> a = "20010331Rainy"
>>> year = a[:4]
>>> day = a[4:8]
>>> weather = a[8:]
>>> year
'2001'
>>> day
'0331'
>>> weather
'Rainy'
```

위의 예는 4와 8이란 숫자로 "20010331Rainy"라는 문자열을 세 부분으로 나누는 방법을 보여준다.

이상과 같이 인덱싱과 슬라이싱에 대해서 살펴 보았다. 인덱싱과 슬라이싱은 프로그래밍을 할 때 매우 자주 사용되는 기법이니 꼭 반복해서 연습을 해 두도록 하자.

“Pithon”이란 문자열을 “Python"으로 바꾸려면?
위의 제목과 같이 "Pithon"이란 문자열을 "Python"으로 바꾸려면 어떻게 해야 할까? 제일 먼저 떠오르는 생각은 다음과 같을 것이다.

```
>>> a = "Pithon"
>>> a[1]
'i'
>>> a[1] = 'y'
```

위의 예에서 보듯이 우선 a라는 변수에 "Pithon"이란 문자열을 대입하고 a[1]이란 값이 'i'니까 a[1]을 위의 예처럼 'y'로 바꾸어 준다는 생각이다. 하지만 결과는 어떻게 나올까?

당연히 에러가 나고 실패하게 될 것이다. 에러가 나는 이유는 문자열의 요소 값은 바꿀 수 있는 값이 아니기 때문이다. 왜 바꿀 수 없는가? 라는 질문을 하지는 말자. 그냥 바꿀 수 없는 것이다. 하지만 앞서 살펴보았던 슬라이싱 기법을 이용해서 "Pithon"이란 문자열을 "Python"으로 바꿀 수 있는 방법이 있다.

다음의 예를 보자.

```
>>> a = "Pithon"
>>> a[:1]
'P'
>>> a[2:]
'thon'
```

```
>>> a[:1] + 'y' + a[2:]
'Python'
```

위의 예에서 보듯이 슬라이싱을 이용해서 먼저 'Pithon'이라는 문자열을 'P'부분과 'thon'부분으로 나눌 수 있기 때문에 그 사이에 'y'라는 문자를 추가하여 'Python'이라는 문자열을 만들면 된다.

문자열 포매팅(Formatting)
문자열에서 알아야 할것으로는 또하나, 문자열 포매팅이라는 것이 있다. 이것을 공부하기에 앞서 다음과 같은 문자열을 출력하는 프로그램을 작성했다고 가정해보자.

> " 현재 온도는 18도입니다. "

하지만 시간이 지나서 20도가 된다면 아래와 같은 문장을 출력하는 프로그램.

> " 현재 온도는 20도입니다 "

이러한 프로그램을 어떻게 만들 수 있는지에 대해서는 생각하지 말고 출력해주는 문자열에만 주목해 보자. 출력된 문자열은 모두 같은데 20이라는 숫자와 18이라는 숫자만이 다르다. 이렇게 문자열 내의 어떤 특정한 값을 변화시키는 것이 필요한 경우가 생기는데 이것을 가능하게 해 주는 것이 바로 문자열 포매팅 기법이다.

문자열 포매팅이란 문자열 내에 어떤 값을 삽입하는 방법이다. 다음의 예들을 따라해 보면서 그 사용법을 알아보자.

예 1) 숫자 바로 대입

```
>>> print "I eat %d apples." % 3
I eat 3 apples.
```

위의 예제 결과 값을 보면 알겠지만 위의 예는 문자열 내에 3이라는 정수값을 삽입하는 방법을 보여준다. 삽입할 3이라는 숫자는 가장 뒤에 " % " 문자 다음에 쓰였다. 그리고 문자열 내에 3이라는 값을 넣고 싶은 자리에 "%d"라는 문자를 넣어주었다.

하지만 꼭 문자열 내에 숫자만 넣으라는 법은 없다. 이번에는 숫자 대신 문자열을 넣어 보자.

예 2)문자열 바로 대입

```
>>> print "I eat %s apples." % "five"
I eat five apples.
```

위와 예에서 보는 것과 같이 문자열 내에 또다른 문자열을 삽입하기 위해서는 앞서 사용했던 %d 가 아닌 %s를 썼음을 알 수 있다. 그렇다면 위와 같은 사실로 독자는 유추할 수 있을 것이다. 숫자를 삽입하게 위해서는 %d를 써야 하고 문자열을 삽입하기 위해서는 %s를 써야 한다는 사실을.

예 3) 숫자 변수로 대입

```
>>> number = 3
>>> print "I eat %d apples." % number
I eat 3 apples.
```

예 1처럼 숫자를 바로 대입하나 예 3처럼 숫자값을 나타내는 변수를 대입하나 같은 결과가 나온다. 하지만 실제 프로그래밍에서 문자열 포매팅을 이용하는 대부분의 경우는 변수를 사용한다.

그렇다면 문자열 안에 한 개가 아닌 여러개의 값을 삽입하고 싶다면 어떻게 해야 할까?

예 4) 두 개 이상의 값을 치환

```
>>> number = 10
>>> day = "three"
>>> print "I eat %d apples. so I was sick for %s days." % (number, day)
I eat 10 apples. so I was sick for three days.
```

예 4처럼 두 개 이상의 값을 치환하려면 위에서 보듯이 마지막 % 다음에 ( ) 사이에 콤마로 구분하여 변수를 넣어 주어야만 한다.

문자열 포맷 코드
예 4는 대입시키는 자료형으로 정수와 문자열을 사용했지만 이 이외에도 다양한 것들을 대입시킬 수 있다. 문자열 포맷 코드로는 다음과 같은 것들이 있다.

```
---------------------------------------------
코드   설명
---------------------------------------------
%s    문자열 (String)
%c    문자 한개(character)
%d    정수 (Integer)
%f     부동소수 (floating-point)
```

```
%o    8진수
%x    16진수
%%   Literal % (문자 '%s' 자체)
--------------------------------------------------
```

여기서 재미있는 것은 %s 포맷 코드로 이것은 어떤 형태로든 변환이 가능하다. 무슨 말인지 예를 통해 확인해 보자.

```
>>> print "I have %s apples" % 3
I have 3 apples
>>> print "Todays rate is %s" % 3.234
Todays rate is 3.234
```

3을 문자열 내에 삽입하려면 문자열 내에 %d가 있어야 하고 3.234를 삽입하려면 문자열 내에 %f가 있어야 하지만 %s를 사용하면 이런 것을 생각하지 않아도 된다. 왜냐하면 %s는 자동으로 %d, %f로 바뀌기 때문이다. 따라서 항상 %s를 사용하면 프로그램 소스를 읽을 때 조금 헷갈리겠지만 에러는 나지 않을 것이다.

[참고] 포맷팅 연산자 %d와 %를 같이 쓸 때는 %%를

```
>>> print "Error is %d%." % 98
```

당연히 결과값으로 "Error is 98%."가 출력될 것이라고 예상하겠지만 파이썬은 에러 메시지를 보여준다. 이유는 문자열 포맷코드인 %d 와 % 가 같은 문자열 내에 존재하면 %를 나타내기 위해서는 반드시 %%로 해야 %로 되는 법칙이 있다. 이건 꼭 기억해두어야 한다. 하지만 문자열 내에 %d 같은 포맷팅 연산자가 없으면 %는 홀로 쓰여도 상관이 없다.

따라서 위 예를 제대로 실행될 수 있게 하려면 다음과 같이 해야 한다.

```
>>> print "Error is %d%%." % 98
Error is 98%.
```

포맷 코드의 또 다른 기능
위에서 보았듯이 %d, %s등의 포맷코드는 문자열 내에 어떤 값을 삽입하기 위해서 사용됨을 알 수 있었다. 하지만 포맷코드를 숫자와 함께 사용하면 더 유용하게 사용할 수 있다. 다음의 예를 보고 따라해 보도록 하자.

예 1) 정렬과 공백

```
>>> print "%10s" % "hi"
       hi
^^^^^^^^^^
```

즉 "%10s"의 의미는 길이가 10개인 문자열 공간에서 오른쪽으로 정렬하고 그 앞의 나머지는 공백으로 남겨 두라는 의미이다. 그렇다면 반대쪽인 왼쪽 정렬은 "%-10s"가 될 것이다. 확인해 보자.

```
>>> print "%-10s jane." % 'hi'
hi        jane.
^^^^^^^^^^
```

왼쪽으로 정렬하고 나머지는 공백으로 채웠음을 볼 수 있다.

예 2) 소숫점 표현

```
>>> print "%0.4f" % 3.42134234
3.4213
```

즉, 3.42134234를 소수점 4번째까지만 나타내고 싶을 경우에 위와 같이 하였다. 즉 여기서 '.'의 의미는 소수점 포인트를 말하고 그 뒤의 숫자 4는 뒤에 나올 숫자의 개수를 말한다. '.'앞의 숫자는 이전의 예에서와 같이 오른쪽 또는 왼쪽 정렬을 말하는 숫자이다.

```
>>> print "%10.4f" % 3.42134234
3.4213
^^^^
```

위의 예는 3.42134234라는 숫자를 10개의 문자열 공간에 오른쪽으로 정렬하고 소수점은 4번째 자리까지만 표시하게 하는 예를 보여준다. 위의 예와의 차이점은 숫자를 오른쪽으로 정렬했다는 점이다.

위에서 알아본 것들은 주로 결과값을 깔끔하게 정리할 목적으로 많이 사용한다.

지금까지 문자열을 가지고 할 수 있는 일들에 대해서 대부분 알아보았지만 문자열을 가지고 할 수 있는 일들에 대해서 공부해야 할 사항들이 아직 많이 남아 있다. 지겹다면 잠시 책을 접고 휴식을 취하도록 하자.

소문자를 대문자로 바꾸기(upper)

```
>>> a = "hi"
>>> a.upper()
'HI'
```

앞서 복소수에서 복소수가 자체적으로 가지고 있는 복소수 관련함수가 있었던 것처럼 문자열 역시 자체적으로 가지고 있는 관련함수들이 몇 개 있다. 그 것들을 사용하기 위해서는 문자열 변수이름 뒤에 '.'을 붙인 다음에 관련함수 이름을 써 주면 된다. 위의 예의 upper()함수는 소문자를 대문자로 바꾸어준다. 만약 문자열이 이미 대문자라면 아무런 변화도 일어나지 않을 것이다.

문자 갯수 세기(count)

```
>>> a = "hobby"
>>> a.count('b')
2
```

문자열 중 문자 ‘b’의 개수를 반환한다.

위치 알려주기1(find)

```
>>> a = "Python is best choice"
>>> a.find('b')
10
```

문자열 중 문자 'b'가 처음으로 나온 위치를 반환한다. 만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환한다.

위치 알려주기2(index)

```
>>> a = "Life is too short"
>>> a.index('t')
8
```

문자열 중 문자 't'가 처음으로 나온 위치를 반환한다. 만약 찾는 문자나 문자열이 존재하지 않는다면 에러를 발생시킨다. 위의 find함수와 다른 점은 없는 문자를 찾으려고 하면 에러가 발생한다는 점이다.

문자열 삽입 (join)

```
>>> a= ","
>>> a.join('abcd')
'a,b,c,d'
```

"abcd"라는 문자열의 각각의 문자사이에 변수 a의 값인 ','을 삽입한다.

대문자를 소문자로 바꾸기(lower)

```
>>> a = "HI"
>>> a.lower()
'hi'
```

대문자를 소문자로 바꾸어준다.

왼쪽 공백 지우기 (lstrip)

```
>>> a = " hi"
>>> a.lstrip()
'hi'
```

문자열중 가장 왼쪽의 연속된 공백들을 모두 지운다. 여기서 lstrip에서 'l'이 의미하는 것은 left이다.

오른쪽 공백 지우기 (rstrip)

```
>>> a= "hi "
>>> a.rstrip()
'hi'
```

문자열중 가장 오른쪽의 연속된 공백들을 모두 지운다. 여기서 rstrip에서 'r'이 의미하는 것은 right이다.

양쪽 공백 지우기 (strip)

```
>>> a = " hi "
>>> a.strip()
'hi'
```

양쪽의 연속된 공백을 모두 지운다.

문자열 바꾸기 (replace)

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

replace(바뀌게 될 문자열, 바꿀 문자열)처럼 사용해서 문자열 내의 특정한 값을 다른 값으로 치환해 준다.

문자열 나누기 (split)

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> a = "a:b:c:d"
>>> a.split(':')
['a', 'b', 'c', 'd']
```

a.split()처럼 괄호안에 아무런 값도 넣어주지 않으면 공백을 기준으로 문자열을 나누어 준다.

만약 a.split(':')처럼 괄호안에 특정한 값이 있을 경우에는 괄호안의 값을 구분자로 해서 문자열을 나누어 준다. 이 나눈 값은 리스트에 하나씩 들어가게 된다. ['Life', 'is', 'too', 'short']나 ['a', 'b', 'c', 'd']는 리스트라는 것인데 조금 후에 자세히 알아볼 것이니 너무 신경쓰지 말도록 하자.

대문자와 소문자를 서로 바꾸기(swapcase)

```
>>> a = "Hi man"
>>> a.swapcase()
'hI MAN'
```

대문자와 소문자를 서로 바꾸어 준다.

위에서 소개한 문자열 관련 함수들은 문자열 처리에서 매우 사용 빈도가 높은 것들이고 유용한 것들이다. 이 외에도 몇 가지가 더 있지만 자주 사용되는 것들은 아니다.

지금껏 알아보았던 문자열 관련 함수를 다음과 같이 표로 정리했다.
문자열 a의 관련함수 (여기서 a라는 변수는 임의로 설정한 문자열 변수이다)

함수	설명
a.upper()	문자열 a를 모두 대문자로 바꾸어 준다.
a.count(x)	문자열 a중 x와 일치하는 것의 갯수를 반환한다.
a.find(x)	문자열 a중 문자 x가 처음으로 나온 위치를 반환한다. 없으면 -1을 반환한다.
a.index(x)	문자열 a중 문자 x가 처음으로 나온 위치를 반환한다. 없으면 에러를 발생시킨다.
a.join(s)	s라는 문자열의 각각의 요소 문자사이에 문자열 a를 삽입한다.
a.lower()	문자열 a를 모두 소문자로 바꾸어 준다.

a.lstrip()	문자열 a의 왼쪽 공백을 모두 지운다.
a.rstrip()	문자열 a의 오른쪽 공백을 모두 지운다.
a.strip()	문자열 a의 양쪽 공백을 모두 지운다.
a.replace(s, r)	문자열 a의 s라는 문자열을 r이라는 문자열로 치환한다.
a.split([s])	문자열 a를 공백으로 나누어 리스트값을 돌려준다.
a.swapcase()	문자열 a의 대문자는 소문자로, 소문자는 대문자로 각각 바꾸어 준다.

[3] 리스트 (List)

지금까지 우리는 숫자와 문자열에 대해서 알아보았다. 하지만 이러한 것들로 프로그래밍을 하기엔 부족한 점이 많다. 예를 들어 1부터 10까지의 숫자들 중 홀수들의 모임인 1, 3, 5, 7, 9라는 집합을 생각해 보자. 이것들을 숫자나 문자열로 표현 하기는 쉽지가 않다. 파이썬에는 이러한 불편함을 해소할 수 있는 자료형이 존재한다. 그것이 바로 이곳에서 공부하게 될 리스트라는 것이다.

리스트를 이용하면 1, 3, 5, 7, 9라는 숫자의 모임을 다음과 같이 간단하게 표현할 수 있다.

```
>>> odd = [1,3,5,7,9]
```

리스트를 만들 때는 위에서 보는 것과 같이 대괄호([ ])로 감싸주고 안에 들어갈 값들은 쉼표로 구분해준다.

여러 가지 리스트의 생김새를 살펴보면 다음과 같다.

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

a 처럼 리스트는 아무 것도 포함하지 않는 빈 리스트([])일 수도 있고 b처럼 숫자를 그 요소 값으로 가질 수도 있고, c처럼 문자열을 요소값으로 가질 수 있고 d처럼 숫자와 문자열을 함께 요소값으로 가질 수 있으며 또한 e처럼 리스트 자체를 그 요소 값으로 가질 수 도 있다. 즉, 리스트 내에는 어떠한 자료형도 포함시킬 수 있다.

리스트의 인덱싱과 슬라이싱
리스트의 경우에도 문자열처럼 인덱싱과 슬라이싱이 가능하다. 백문이 불여일견. 말로 설명하는 것보다 직접 예를 따라해보면서 리스트의 기본 구조를 이해하는 것이 쉽다. 대화형 인터프리터로 예를 따라하며 알아보자.

리스트의 **인덱싱** 알아보기
a 변수에 [1, 2, 3] 이라는 값을 세팅한다.

```
>>> a = [1, 2, 3]
>>> a
```

```
[1, 2, 3]
```

리스트 역시 문자열에서처럼 인덱싱이 적용된다.

```
>>> a[0]
1
```

a[0]는 리스트 'a'의 첫 번째 요소값을 말한다.

아래의 예는 리스트의 첫 번째 요소인 a[0]와 세 번째 요소인 a[2]의 값을 더해 주었다.

```
>>> a[0] + a[2]
4
```

이것은 1 + 3로 해석되어서 4라는 값을 출력해 준다.

문자열을 공부할 때 이미 알아보았지만 컴퓨터는 숫자를 0부터 세기 때문에 a[1]이 리스트 a의 첫 번째 요소가 아니라 a[0]가 리스트 a의 첫 번째 요소임을 명심하도록 하자. a[-1]은 문자열에서와 마찬가지로 리스트 a의 마지막 요소를 말한다.

```
>>> a[-1]
3
```

이번에는 아래의 예처럼 리스트 a를 숫자 1, 2, 3과 또다른 리스트인 ['a', 'b', 'c']를 포함하도록 만들어 보자.

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

다음의 예를 따라해 보자.

```
>>> a[0]
1
```

```
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
```

예상한 대로 a[-1]은 마지막 요소값인 ['a', 'b', 'c']를 나타낸다. a[3]는 리스트 a의 네 번째 요소를 나타내기 때문에 마지막 요소를 나타내는 a[-1]과 동일한 결과값을 보여준다.

그렇다면 여기서 리스트 a에 포함된 ['a', 'b', 'c']라는 리스트의 'a'라는 값을 인덱싱을 이용하여 끄집어 낼 수 있는 방법은 없을까?
다음의 예를 보도록 하자.

```
>>> a[-1][0]
'a'
```

위처럼 하면 'a'를 끄집어 낼 수가 있다. a[-1]은 ['a', 'b', 'c']이고 다시 이것의 첫 번째 요소를 불러오기 위해서 [0]을 붙여준 것이다.

아래의 예도 역시 마찬가지 경우이므로 이해가 될 것이다.

```
>>> a[-1][1]
'b'
>>> a[-1][2]
'c'
```

조금은 복잡하지만 다음의 예를 따라해 보자

```
>>> a = [1, 2, ['a', 'b', ['Life', 'is']]]
```

리스트 a안에 리스트 ['a', 'b', ['Life', 'is']]라는 리스트가 포함되어 있고 그 리스트 안에 역시 리스트 ['Life', 'is']라는 리스트가 포함되어 있다. 삼중 리스트 구조이다.

'Life'라는 문자열만을 끄집어 내기 위해서는 다음과 같이 해야 한다.

```
>>> a[2][2][0]
'Life'
```

즉 위의 예는 리스트 a의 세 번째 요소인 리스트['a', 'b', ['Life', 'is']]의 세 번째 요소인 리스트 ['Life', 'is']의 첫 번째 요소를 말하는 것이다. 이렇듯 리스트를 중첩해서 쓰면 혼란스럽기 때문에 자주 사용되지는 않지만 알아두는 것이 좋을 것이다.

리스트의 **슬라이싱** 알아보기
문자열에서와 마찬가지로 리스트에서도 슬라이싱 기법이 적용된다. 슬라이싱이라는 것은
" 나눈다 " 라는 뜻이라고 했었다.
자, 그럼 리스트의 슬라이싱에 대해서 살펴보도록 하자.

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
```

이것을 문자열에서 했던 방법과 비교해서 생각 해보자.

```
>>> a = "12345"
>>> a[0:2]
'12'
```

두 가지가 완전히 동일하게 사용됨을 독자는 이미 눈치 챘을 것이다. 문자열에서 했던 것과 사용법이 완전히 동일하다.

몇가지 예를 더 들어 보도록 하자.

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> c = a[2:]
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

b라는 변수는 리스트 a의 처음 요소부터 2번째 요소까지를 나타내는 리스트이다. 여기서는 a[2] 값인 '3'이 포함되지 않는다. c라는 변수는 리스트 2번째 요소부터 끝까지를 나타내는 리스트이다.

리스트가 포함된 리스트역시 똑같이 적용된다.

```
>>> a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
>>> a[2:5]
[3, ['a', 'b', 'c'], 4]
>>> a[3][:2]
['a', 'b']
```

위의 예에서 a[3]은 ['a', 'b', 'c']를 나타내기 때문에 a[3][:2]는 ['a', 'b', 'c']의 a[0]에서 a[2]까지의 값 즉, ['a', 'b']를 나타내는 리스트가 된다.

리스트를 더하고(+) 반복하기(*)
리스트 역시 + 기호를 이용해서 더할 수가 있고 * 기호를 이용해서 반복을 할 수가 있다.
문자열과 마찬가지로 리스트에서도 되는지 한번 직접 확인해 보도록 하자.

예 1) 리스트 합치기

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

두 개의 리스트를 '+' 기호를 이용해 합치는 방법이다. 리스트 사이에서 '+' 기호는 두 개의 리스트를 합치는 기능을 한다. 문자열에서 "abc" + "def" = "abcdef"가 되는 것과 같은 이치이다.

예 2) 리스트 반복

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

위에서 보듯이 [1, 2, 3]이란 리스트가 세 번 반복되어 새로운 리스트를 만들어 내는 것을 볼 수 있다. 문자열에서 "abc" * 3 = "abcabcabc" 가 되는 것과 같은 이치이다.

리스트의 수정 변경과 삭제

참고 - 다음의 예들은 서로 연관되어 있으므로 따로따로 실행해 보지 말고 예 1부터 예 4까지 차례대로 진행해 나가야 한다.

예 1) 리스트 수정1

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

a[2]의 요소값 '3'을 '4'로 바꾸었다.

예 2) 리스트 수정2

```
>>> a[1:2]
[2]
>>> a[1:2] = ['a', 'b', 'c']
>>> a
[1, 'a', 'b', 'c', 4]
```

a[1:2] 는 a[1]부터 a[2]까지를 말하는데 a[2]를 포함하지 않는다고 했으므로 a = [1, 2, 4]에서 2값만을 말한다. 즉, a[1:2]를 ['a', 'b', 'c']로 바꾸었으므로 a 리스트에서 2라는 값대신에 ['a', 'b', 'c']라는 값을 대입하게 되는 것이다.

참고 - 여기서 a[1] = ['a', 'b', 'c']라고 하는 것과는 전혀 다른 결과값을 갖게 되므로 주의 하도록 하자. a[1] = ['a', 'b', 'c']는 리스트 a의 두 번째 요소를 ['a', 'b', 'c']로 바꾼다는 말이고 a[1:2]는 a[1]에서 a[2]사이의 리스트를 ['a', 'b', 'c']로 바꾼다는 말이다. 따라서 a[1] = ['a', 'b', 'c']처럼 하면 위와는 달리 리스트 a가 [1, ['a', 'b', 'c'], 4]라는 값으로 변하게 된다.

예 3) 리스트 요소 삭제1

```
>>> a[1:3] = []
>>> a
[1, 'c', 4]
```

지금까지의 리스트 a의 값은 [1, 'a', 'b', 'c', 4]였다. 여기서 a[1:3]은 a[1]부터 a[3]까지를 나타내므로 a[1:3] 은 ['a', 'b']이다. 그런데 위의 예에서 보듯이 a[1:3]을 []으로 바꾸어 주었기 때문에 a 는 a에서 ['a', 'b']가 사라진 [1, 'c', 4]가 되는 것이다.

예 4) 리스트 요소 삭제 2

```
>>> a
[1, 'c', 4]
>>> del a[1]
>>> a
[1, 4]
```

del a[x]는 x번째 요소값을 삭제한다. del 함수는 파이썬이 자체적으로 가지고 있는 내장 함수로 다음과 같이 사용되어 진다.

```
del 객체
```

(참고 - 객체란 파이썬에서 사용되는 모든 자료형을 말한다.)

del a[x:y]는 x번째부터 y번째 요소 사이의 값을 삭제한다. 예 4에서는 a[1]을 삭제하는 방법을 보여준다.

[참고] 초보가 범하기 쉬운 리스트 연산 오류
우선 다음과 같은 예를 먼저 만들어보자.

```
>>> a = [1, 2, 3]
>>> a[2] + "hi"
Traceback (innermost last):
File "", line 1, in ?
a[2] + "hi"
TypeError: number coercion failed
```

TypeError 에러가 발생했다. 에러의 원인은 무엇일까?a[2]는 3이라는 정수인데 "hi"는 문자열이다. 두 값(정수와 문자열)을 더한다는 것은 상식적으로 맞지 않는 방법이다. 그래서 Type 에러가 발생하는 것이다. 숫자와 문자열을 더할 수는 없다.

만약 숫자와 문자열을 더해서 '3hi'처럼 만들고 싶다면 숫자 3을 문자 '3'으로 바꾸어 주어야 한다.

그 방법에는 두 가지가 있다.

```
>>> str(a[2]) + "hi"
>>> `a[2]` + "hi"
```

첫 번째 방법은 str 함수로 정수를 문자열로 바꾸어 주는 방법이고 두 번째 방법은 (`) Back Quote 문자를 이용한 것인데, 위에서 Back Quote문자(`)는 str 함수를 사용한 것과 완전히 동일한 결과값을 돌려준다.

리스트 관련 함수들

문자열과 마찬가지로 리스트 변수명 뒤에 '.'을 붙여서 여러 가지 리스트의 관련 함수들을 이용할 수가 있다. 유용하게 쓰이는 리스트 관련 함수들 몇 가지에 대해서만 알아보기로 하자.

리스트에 요소 추가 (append)

append의 뜻이 무엇인지 안다면 아래의 예가 금방 이해가 될 것이다. append(x)는 리스트의 맨 마지막에 x를 추가시키는 함수이다.

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

리스트내에는 어떤 자료형도 추가시킬 수가 있다. 아래의 예는 리스트에 다시 리스트를 추가시킨 결과를 보여준다.

```
>>> a.append([5,6])
>>> a
```

```
[1, 2, 3, 4, [5, 6]]
```

리스트 정렬(sort)
sort 함수는 리스트의 요소를 순서대로 정렬하여 정렬된 값을 돌려준다.

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

문자 역시 마찬가지로 알파벳 순서로 정렬이 가능하다.

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

문자열과 숫자가 섞여있는 경우에도 순서대로 정렬함을 보여준다.

```
>>> a = ['abc', 123, 'you need python']
>>> a.sort()
>>> a
[123, 'abc', 'you need python']
```

숫자가 가장 앞에 오게 되고 문자열의 경우는 문자열끼리 각각의 첫 번째 문자부터 비교해서 작은 것을 앞에 오게 한다. 위의 경우에서는 123이 숫자이니까 가장 먼저 오고 다음에 'abc'와 'you need python'이란 문자열의 첫 번째 문자인 'a' 와 'y'를 먼저 비교하여 'a'가 'y'보다 작은 값이기 때문에 'abc'라는 문자열을 숫자 다음에 놓는 것이다.

리스트 뒤집기(reverse)
reverse 함수는 리스트를 역순으로 뒤집어준다. 하지만 이것이 의미하는 것이 먼저 순서대로 정렬한 다음에 다시 역순으로 정렬하는 것은 아니다. 그저 리스트 그대로를 거꾸로 뒤집는 일을 할 뿐이다.

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

위치 반환 (index)
index(x) 함수는 리스트에 x라는 값이 있으면 그 위치를 돌려준다.

```
>>> a = [1,2,3]
>>> a.index(3)
2
>>> a.index(1)
0
```

위의 예에서 리스트 a에 있는 3이라는 숫자는 a[2]이므로 2를 돌려주고, 1이라는 숫자는 a[0]이므로 0을 돌려준다.

아래의 예에서 0 이라는 값은 a 리스트에 존재하지 않기 때문에 에러가 난다.

```
>>> a.index(0)
Traceback (innermost last):
File "", line 1, in ?
a.index(0)
ValueError: list.index(x): x not in list
```

Tracebace이란 문장부터 ValueError라는 문장까지가 에러메시지이다.

리스트에 요소 삽입 (insert)
insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수이다.

```
>>> a = [1,2,3]
>>> a.insert(0, 4)
[4, 1, 2, 3]
```

위의 예에서는 0번째 자리 즉 첫 번째 자리에 4 라는 값을 삽입하라는 뜻이 된다.

아래의 예는 리스트 a의 a[3], 즉 네 번째 자리에 5라는 값을 삽입하라는 뜻이다.

```
>>> a.insert(3, 5)
[4, 1, 2, 5, 3]
```

리스트 요소 제거 (remove)
remove(x)는 첫번째 나오는 x 를 삭제하는 함수이다. 아래의 예는 a가 3이라는 값을 두개 가지고 있을경우 첫번째 3만을 제거하는 것을 보여준다.

```
>>> a = [1,2,3,1,2,3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

다시 또 3을 삭제한다.

```
>>> a.remove(3)
[1, 2, 1, 2]
```

리스트 요소 끄집어내기(pop)
pop() 함수는 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제한다.

```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
```

위의 예에서 보듯이 a 리스트 [1,2,3]에서 3을 끄집어내어서 최종적으로 [1, 2]만 남는 것을 볼 수 있다.

pop(x)는 리스트의 x번째 요소를 돌려주고 그 요소는 삭제한다.

```
>>> a = [1,2,3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

위의 예에서 보듯이 a.pop(1)는 a[1]의 값을 끄집어낸다.

갯수세기 (count)
count(x)는 리스트 중에서 x가 몇 개 있는지를 조사하여 그 갯수를 돌려주는 함수이다.

```
>>> a = [1,2,3,1]
>>> a.count(1)
2
```

위의 예에서는 1이라는 값이 리스트 a에 두 개가 들어 있으므로 2를 돌려준다.

리스트 확장(extend)
extend(x)에서 x에는 리스트만 올 수 있다. 원래의 a 리스트에 x 리스트를 더하게 된다.

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
```

a.extend([4,5])는 a + [4,5]와 동일하다.

※ 다음은 위에서 알아본 리스트에 관련한 함수들을 표로 만들어 본 것이다.
리스트 a의 관련함수 (여기서 a라는 변수는 임의로 설정한 리스트 변수이다.)

함수	설명
a.append(x)	리스트 a의 마지막에 x추가
a.sort()	리스트 a를 정렬
a.reverse()	리스트 a의 순서를 거꾸로 만든다.
a.index(x)	리스트 a에서 x를 찾아서 그 위치 반환

a.insert(i, x)	리스트 a에서 i 위치에 x 삽입
a.remove(x)	리스트 a에서 처음 나오는 x 삭제
a.pop()	리스트 a의 맨 마지막 요소 반환하고 마지막 요소 삭제
a.count(x)	리스트 a 안에 x가 몇 개 있는지를 반환
a.extend(x)	리스트 a에 리스트 x를 더함(확장)

[4] 터플 (tuple)

터플 또는 튜플이라고 부른다.
터플이란 리스트와 몇 가지 점을 제외하곤 모든 것이 동일하다. 그 다른 점은 다음과 같다.

- 리스트는 '[' 과 ']' 으로 둘러싸지만 터플은 '('과 ')'으로 둘러싼다.
- 리스트는 그 값을 생성, 삭제, 수정이 가능하지만 터플은 그 값을 변화시킬 수 없다.

터플은 다음과 같은 모습이다.

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1,2,3)
>>> t4 = 1,2,3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

리스트와 생김새가 거의 비슷하지만, 특이할 만한 점이라면 단지 한 개의 요소만을 갖는 터플은 t2 = (1,)처럼 한 개의 요소와 그 뒤에 콤마(',')를 넣어야 한다는 점과 네 번째 보기 t4 = 1, 2, 3 처럼 괄호()를 생략해도 무방하다는 점이다.

얼핏 보면 터플과 리스트는 비슷한 역할을 한다. 하지만 프로그래밍을 할 때 터플과 리스트는 구분해서 사용하는것이 유리하다. 터플과 리스트의 가장 큰 차이는 값을 변화시킬 수 있는 지 없는 지의 차이라고 했다. 리스트의 항목 값은 변화가 가능하고 터플의 항목 값은 변화가 불가능하다. 따라서 프로그램이 진행되는 동안 그 값이 항상 변하지 않기를 바란다면 또는 바뀔까 걱정하고 싶지 않다면 주저하지 말고 터플을 사용해야 할 것이다. 이와는 반대로 수시로 그 값을 변화시켜야 할 경우라면 리스트를 사용해야 한다. 실제 프로그램에서는 평균적으로 터플보다는 리스트를 훨씬 많이 쓰게 된다.

터플의 인덱싱, 슬라이싱, 더하기와 반복
값을 변화시킬 수 없다는 점만 제외하면 리스트와 완전히 동일하므로 간단하게만 살펴 보기로 하자. 아래의 예제는 서로 연관 되어 있으므로 예1부터 차례대로 수행해 보기를 바란다.

예 1) 인덱싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
```

```
'b'
```

문자열, 리스트와 마찬가지로 t1[0], t1[3]처럼 인덱싱이 가능하다.

예 2) 슬라이싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

슬라이싱의 예이다.

예 3) 터플 더하기(합)

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

터플을 합하는 방법을 보여준다.

예 4) 터플 곱(반복)

```
>>> t2 * 3
(3, 4, 3, 4, 3, 4)
```

터플의 곱(반복)을 보여준다.

터플의 요소 값은 변경시킬 수 없다 터플의 요소값은 한 번 정하면 지우거나 변경할 수 없다. 다음에 소개하는 두 개의 예를 살펴보면 좀 더 이해가 쉬울 것이다.

예 1) 터플 요소 지우려고 할 때의 에러

```
>>> del t1[0]
Traceback (innermost last):
File "", line 1, in ?del t1[0]
```

```
TypeError: object doesn't support item deletion
```

터플의 요소를 리스트처럼 del 함수로 지우려고 한 예이다.
터플은 요소를 지우는 행위가 지원되지 않는다는 메시지를 확인 할 수 있다.

예 2) 터플 요소값 변경시 에러

```
>>> t1[0] = 'c'
Traceback (innermost last):
File "", line 1, in ?t1[0] = 'c'
TypeError: object doesn't support item assignment
```

터플의 요소 값을 변경하려고 해도 마찬가지로 에러가 발생하는 것을 확인할 수 있다.

[5] 딕셔너리 (Dictionary)

‘사람’을 예로 들면 누구든지 “이름” = “홍길동”, “생일” = “몇 월 몇 일” 등으로 구분할 수 있다. 파이썬은 영리하게도 이러한 대응관계를 자료형으로 만들었다. 이것은 요즘 나오는 대부분의 언어들도 갖고 있는 자료형으로 Associative array, Hash라고도 불린다.

딕셔너리란 단어 그대로 해석하면 사전이란 뜻이다. 즉, people 이란 단어에 ‘사람’, baseball 이라는 단어에 ‘야구’라는 뜻이 부합되듯이 딕셔너리는 Key와 Value라는 것을 한 쌍으로 갖는 자료형이다. 위의 예에서 보면 Key가 'baseball'이라면 Value는 '야구'가 될 것이다.

딕셔너리는 리스트나 터플처럼 순차적으로(sequential) 해당 요소 값을 구하지 않고 key를 통해 value를 얻는다. 딕셔너리의 가장 큰 특징이라면 key로 value를 얻어낸다는 점이다. baseball이란 단어의 뜻을 찾기 위해서 사전의 내용을 순차적으로 모두 검색하는 것이 아니라 baseball이라는 단어가 있는 곳만을 펼쳐보는 것이다.

딕셔너리는 어떻게 생겼을까?
다음은 기본적인 딕셔너리의 모습이다.

```
{Key1:Value1, Key2:Value2, Key3:Value3,,,,}
```

즉, Key와 Value쌍들이 여러개가 '{'과 '}'으로 둘러싸이고 각각의 요소는 Key : Value형태로 이루어져 있고 쉼표(',')로 구분되어져 있음을 볼 수 이다.

다음의 딕셔너리 예를 보도록 하자.

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

위에서 key는 각각 ‘name’, ‘phone’, ‘birth’이고 그에 해당하는 value는 ‘pey’, ‘0119993323’, ‘1118’이 된다.

이것을 보기 좋게 테이블로 만들어 보았다.

<딕셔너리 dic의 정보>

key	value
name	pey
phone	01199993323
birth	1118

```
>>> a = {1: 'hi'}
```

위의 예는 Key로 정수값 1을 Value로 'hi'란 문자열을 사용한 예이다.

```
>>> a = { 'a': [1,2,3]}
```

또한 위의 예처럼 Value에 리스트도 넣을 수 있다.

딕셔너리 사용하기
딕셔너리는 주로 어떤 것을 표현하는 데 쓸 수 있을까? 라는 의문이 들 것이다. 4명의 사람이 있는데 각각의 사람의 특기를 표현할 수 있는 좋은 방법에 대해서 생각해 보자. 리스트나 문자열로는 표현하기가 상당히 까다로울 것이다. 하지만 파이썬의 딕셔너리를 사용한다면 위의 상황을 표현하는 것은 정말 쉽다.

다음의 예를 보자.

```
{ " 홍길동 " : " 칼싸움 " , " 임꺽정 " : " 주먹질 " , " 김두한 " : " 발길질 " , " 귀도 " : " 파이썬 " }
```

사람이름과 특기를 한쌍으로 하는 딕셔너리이다. 정말 간편하지 않은가? 지금껏 우리는 딕셔너리를 만드는 방법에 대해서만 살펴 보았는데 이것들을 제대로 활용하기 위해서 알아야 할 것들이 있다. 그것들에 대해서 알아보도록 하자.

Key이용하여 Value얻기
다음의 예를 따라해 보도록 하자.

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']
10
>>> grade['julliet']
99
```

리스트나 터플이나 문자열은 요소값을 얻어내기 위해서 인덱싱이나 슬라이싱이라는 기법을 사용했지만 딕셔너리는 단 한가지의 방법만이 있을 뿐이다. 바로 Key를 이용해서 Value를 얻어내는 방법이다.

위의 예에서처럼 Value를 얻기 위해서는 "딕셔너리변수[Key]"와 같이 하여 Value를 얻을 수 있다.

몇가지 예를 더 보도록 하자.

```
>>> a = {1:'a', 2:'b'}
>>> a[1]
'a'
>>> a[2]
'b'
```

먼저 a라는 변수에 {1:'a', 2:'b'}라는 딕셔너리를 대입하였다. 위의 예에서 보듯이 a[1]은 'a'라는 값을 돌려준다. 여기서 a[1]이 의미하는 것은 리스트나 터플의 a[1]과는 아주 다른 것이다. 여기서 [ ] 안의 숫자 1은 몇번째 요소를 뜻하는 것이 아니라 Key에 해당하는 1을 나타낸다. 딕셔너리는 리스트나 터플의 인덱싱 방법이란 것이 존재하지 않는다. 따라서 a[1]은 딕셔너리 {1:'a', 2:'b'}에서 Key가 1인것의 Value인 'a'를 돌려주게 된다. a[2] 역시 마찬가지이다.

```
>>> a = {'a':1, 'b':2}
>>> a['a']
1
>>> a['b']
2
```

이번에는 a라는 변수에 위에서 사용했던 딕셔너리의 Key와 Value를 뒤집어 놓은 딕셔너리를 대입해 보았다. 역시 a['a'], a['b']처럼 Key를 이용해서 Value를 얻을 수 있다. 이상 정리해 보면 딕셔너리 a 는 a[Key] 처럼 해서 Key에 해당하는 Value를 얻을 수 있다.

다음은 위에서 한번 언급했던 딕셔너리인데 Key를 이용하여 Value를 얻는 방법을 잘 보여주고 있다.

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> dic['name']
'pey'
>>> dic['phone']
'0119993323'
>>> dic['birth']
'1118'
```

딕셔너리 쌍 추가, 삭제하기
딕셔너리 쌍을 추가하는 방법은 Key를 이용해 Value를 호출했던 것처럼 새로운 Key에 Value를 설정하면 바로 딕셔너리에 추가된다. 예 1부터 예 3까지는 딕셔너리를 추가하는 예를 보여준다. 딕셔너리는 순서를 따지지 않는다. 예에서 알 수 있듯이 추가되는 순서는 원칙이 없다. 중요한 것은 " 무엇이 추가되었는가 " 이다.

다음의 예를 함께 따라해 보자.

예 1) 딕셔너리 쌍 추가1

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{2: 'b', 1: 'a'}
```

{1: 'a'}라는 딕셔너리에 a[2] = 'b'와 같이 사용해서 2 : 'b' 라는 딕셔너리 쌍을 추가하였다.

예 2) 딕셔너리 쌍 추가2

```
>>> a['name'] = 'pey'
{'name':'pey', 2: 'b', 1: 'a'}
```

딕셔너리 a에 'name': 'pey'라는 쌍을 추가한 모습이다.

예 3) 딕셔너리 쌍 추가3

```
>>> a[3] = [1,2,3]
{'name': 'pey', 3: [1, 2, 3], 2: 'b', 1: 'a'}
```

Key는 3 Value는 [1, 2, 3]을 가지는 한 쌍을 또 추가하였다.

예 4) 딕셔너리 요소 삭제 1

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

예 4는 딕셔너리의 요소를 지우는 방법을 보여준다. del a[key]하면 그에 해당하는 key:value 쌍이 삭제된다.

딕셔너리 주의사항

딕셔너리를 만들때 주의해야 할 사항은 Key는 고유한 값이므로 중복되는 값을 설정해 놓으면 하나를 제외한 나머지의 것들은 무시된다는 점이다. 다음 예에서 보듯이 Key가 동일한 것이 존재할 경우 1:'a'라는 쌍이 무시된다. 이때 꼭 딕셔너리를 만들 때 앞에 썼던 것이 무시되는 것은 아니고 어떤 것이 무시될지는 예측이 불가능하다. 결론은 중복되는 Key를 사용하지 말라는 것이다.

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

이렇게 중복되었을 때 한 개를 제외한 나머지의 Key:Value값이 무시되는 이유는 딕셔너리는 Key를 통해서 Value를 얻게 되는데 만약 동일한 Key가 존재 한다면 어떤 Key에 해당하는 Value를 불러야 할지 알 수가 없기 때문이다.

또 한 가지 주의해야 할 사항으로는 Key에 리스트는 쓸 수가 없다는 것이다. 하지만 터플은 Key로 쓸 수가 있다. 딕셔너리의 Key로 쓸 수 있고 없고의 구별은 Key가 변하는 값인지 변하지 않는 값인지에 달려 있다. 리스트를 Key로 사용한다면 그 값이 변할 수 있기 때문에 리스트를 Key로 쓸 수 없는 것이다. 아래 예처럼 리스트를 Key로 설정하면 TypeError가 난다.

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
File "", line 1, in ?
TypeError: unhashable type
```

따라서 딕셔너리의 Key를 딕셔너리로 할 수 없음은 당연한 얘기가 될 것이다. Value에는 변하는 값이든 변하지 않는 값이든 상관없이 아무 값이나 넣을 수 있다.

딕셔너리 관련함수

딕셔너리를 자유자재로 사용하기 위해 딕셔너리가 자체적으로 가지고 있는 관련 함수들을 사용해 보도록 하자.

Key리스트 만들기(keys)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
['birth', 'name', 'phone']
```

a.keys()는 딕셔너리 a의 Key만을 모아서 리스트로 만든다. 만들어진 리스트 요소들의 순서는 일정한 규칙이 있는 것이 아니라 상황에 따라서 바뀌게 된다. 따라서 독자의 결과값과 위의 예제의 결과값인 ['birth', 'name', 'phone']순서는 바뀔 수도 있다. 딕셔너리는 Key로 value를 찾기 때문에 그 순서는 상관하지 않는다.

Value리스트 만들기 (values)

```
>>> a.values()
['1118', 'pey', '0119993323']
```

마찬가지 방법으로 value만을 얻고 싶다면 a.values()처럼 values 함수를 사용하면 된다.

Key, Value 쌍 얻기(items)

```
>>> a.items()
[('birth', '1118'), ('name', 'pey'), ('phone', '011993323')]
```

items 함수는 key와 value의 쌍을 터플로 묶은 값을 리스트로 돌려준다.

Key: Value 쌍 모두 지우기(clear)

```
>>> a.clear()
>>> a
{}
```

clear() 함수는 딕셔너리 안의 모든 요소를 삭제한다. 위에서 보듯이 빈 리스트가 [] 빈 터플이 ()인 것과 마찬가지로 빈 딕셔너리도 {}과 같이 표현된다.

Key로 Value얻기 (get)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

get(x) 함수는 x 라는 key에 대응되는 value를 돌려준다. 앞서 살펴 보았듯이 a.get('name')은 a['name']처럼 사용하는 것과 완전히 동일한 결과값을 돌려 받는다. 어떤 것을 사용하는가는 독자의 선택이다.

해당 Key가 있는지 조사 (has_key)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.has_key('name')
True
>>> a.has_key('email')
False
```

a.has_key(x) 함수는 딕셔너리 a에 x라는 key가 존재하는지의 참, 거짓을 판단하여 존재하면 1을 존재하지 않는다면 0을 반환한다.

위에서 살펴본 딕셔너리 관련 함수를 정리하여 표로 만들어 보았다.
딕셔너리의 관련 함수 (여기서 a라는 변수는 임의로 설정한 딕셔너리 변수이다.)

함수	설명
a.keys()	딕셔너리 a의 Key들을 모아놓은 리스트를 돌려준다.
a.values()	딕셔너리 a의 Value들을 모아놓은 리스트를 돌려준다.
a.items()	딕셔너리 a의 (Key, Value)쌍의 터플을 모아놓은 리스트를 돌려준다.
a.clear()	딕셔너리 a의 모든 Key:Value 쌍들을 삭제한다.
a.get(x)	딕셔너리 a의 Key가 x인 것의 Value를 돌려준다.
a.has_key(x)	딕셔너리 a에 x라는 Key가 있는지 조사하여 참, 거짓을 돌려준다.

이상과 같이 파이썬에서 사용되는 가장 기본이 되는 자료형인 숫자, 문자열, 리스트, 터플, 딕셔너리에 대해서 알아 보았다. 여기까지 잘 따라온 독자라면 파이썬에 대해서 대략 50% 정도 습득했다고 보아도 된다. 그만큼 위에서 언급한 자료형들은 중요하기 때문이다. 책에 있는 예제들만 따라하지 말고 직접 여러 가지 예들을 만들어 보고 테스트해 가며 반복해서 위의 자료형들에 익숙해지기를 당부한다. 왜냐하면 자료형은 프로그램의 근간이 되기 때문에 확실하게 해 놓지 않으면

좋은 프로그램을 만들 수 없기 때문이다.

[6] 참과 거짓

자료형에 참과 거짓이 있다?
조금 이상한 말처럼 들리지만 참과 거짓은 분명히 있다. 이것은 매우 중요하며 자주 쓰이게 된다. 자료형의 참과 거짓을 구분짓는 기준을 다음과 같이 표로 정리했다.

자료형	참 or 거짓
""가 아닌 문자열 (예: "python")	참
""	거짓
[]가 아닌 리스트 (예: [1,2.3])	참
[]	거짓
()	거짓
{}	거짓
0 이 아닌 숫자 (예: 1)	참
0	거짓
None	거짓

문자열, 리스트, 터플, 딕셔너리 등의 값이 비어 있으면("", [], (), {}) 거짓이 된다. 당연히 비어 있지 않으면 참이 된다. 숫자에서는 그 값이 0일 때 거짓이 된다. 위의 표를 보면 None이란 것이 있는데 이것에 대해서는 아직은 신경쓰지 말도록 하자. 그저 None이란 것은 거짓을 뜻한다고만 알아두자.

다음의 예를 보고 참과 거짓이 프로그램에서 어떻게 쓰이는지에 대해서 간단히 알아보자.

```
>>> a = [1,2,3,4]
>>> while a:
...     a.pop()
...
4
3
2
1
```

먼저 a = [1,2,3,4]라는 리스트를 하나 만들었다.

while문은 뒤에서 자세히 다루겠지만 간단히 알아보면 다음과 같다.

```
while <조건문>:
   <수행할 문장>
```

<조건문>이 참인 동안에 <수행할 문장>을 계속 수행한다. 즉, 위의 예에서 보면 a가 참인 경우에 a.pop()을 계속 실행하라는 의미이다. a.pop()이란 함수는 리스트 a의 마지막 요소를 끄집어내는 함수이므로 a가 참인 동안(리스트 내에 요소가 존재하는 한)에 마지막 요소를 계속해서 끄집어 낼 것이다. 결국 더 이상 끄집어 낼 것이 없으면 a가 빈 리스트([])가 되어 거짓이 될 것이다. 따라서 while문은 거짓에 의해서 중지된다. 위에서 본 예는 파이썬 프로그래밍에서 매우 자주 쓰이는 기법중 하나이다.

위의 예가 너무 복잡하다고 생각하는 독자는 다음의 예를 보면 쉽게 이해가 될 것이다.

```
>>> if []:
. . .     print "True"
. . . else:
. . .     print "False"
. . .
False
```

if문에 대해서 잘 모르는 독자라도 위의 문장을 다음과 같이 해석하는 데 무리가 없을 것이다. (if문에 대해서는 바로 조금후에 자세히 다루게 된다.) 만약 []가 참이면 "True"라는 문자열을 출력하고 그렇지 않으면 "False"라는 문자열을 출력해라. []는 위의 테이블에서 보듯이 거짓이기 때문에 " False"란 문자열이 출력된다.

```
>>> if [1,2,3]:
. . .     print "True"
. . . else:
. . .     print "False"
. . .
True
```

위 코드를 해석해 보면 다음과 같다. 만약 [1,2,3]이 참이면 "True"라는 문자열을 출력하고 그렇지 않으면 "False"라는 문자열을 출력해라. [1,2,3]은 요소 값이 있는 리스트이기 때문에 참이다. 따라서 "True"를 출력한다.

[7] 변수

우리는 이미 변수들을 사용해 왔다. 다음 예와 같은 a, b, c를 파이썬 변수라고 한다.

```
>>> a = 1
>>> b = "python"
>>> c = [1,2,3]
```

변수를 만들 때는 위의 예에서와 같이 '='(assignment) 기호를 사용한다.

앞으로 설명할 부분은 프로그래밍 초보자가 얼른 이해하기 어려운 부분이므로 당장 이해가 되지 않는다면 그냥 건너뛰어도 무방하다. 파이썬에 대해서 공부하다 보면 자연스럽게 알 게 될 것이다.

변수란
변수는 객체를 가리키는 것이다. 객체란 우리가 지금껏 보아왔던 자료형을 포함한 파이썬에서 사용되는 그 모든 것을 말하는 말이다.

```
>>> a = 3
```

만약 위의 코드처럼 a = 3 이라고 했다면 3이라는 값을 가지는 정수 자료형(객체)이 자동으로 메모리에 생성된다. a는 변수 이름으로서 3이라는 정수형 객체가 저장된 메모리 위치를 가리키게 된다. 즉, a는 레퍼런스(Reference)이다.

이해가 잘 되지 않는다면 이렇게 생각해보자.

```
a --> 3
```

즉, a라는 변수는 3이라는 정수형 객체를 가리키고 있다.

다음 예를 보자.

```
>>> a = 3
>>> b = 3
>>> a is b
```

```
True
```

a가 3을 가리키고 b도 3을 가리킨다. 즉 a = 3을 입력하는 순간 3이라는 정수형 객체가 생성되고 변수 a는 3이란 객체의 메모리 번지를 가리킨다. 다음에 변수 b가 동일한 객체인 3을 가리킨다.

즉 3이라는 정수형 객체를 가리키고 있는 변수는 2개가 된다. 이 두 변수는 가리키고 있는 대상이 동일하다. 따라서 동일한 객체를 가리키고 있는지 아닌지에 대해서 판단하는 파이썬 내장함수인 is 함수를 a is b처럼 실행했을 때 참(True)을 리턴하게 된다. 3이라는 객체를 가리키고 있는 변수의 개수는 2개이다. 이것을 조금 어려운 말로 레퍼런스 카운트(Reference Count, 참조갯수)가 2개라고 한다. 만약 c = 3이라고 한번 더 입력한다면 레퍼런스 카운트는 3이 될 것이다.

변수 없애기

3이라는 정수형 객체가 메모리에 생성된다고 했다. 그렇다면 이 값을 메모리에서 없앨 수 있을까? 3이라는 객체를 가리키는 변수들의 개수를 레퍼런스 카운트라 하였는데, 이 레퍼런스 카운트가 0이 되는 순간 3이라는 객체는 자동으로 사라진다. 즉 3이라는 객체를 가리키고 있는 것이 하나도 없을 때 이 3이라는 객체는 메모리에서 사라지게 되는 것이다. 이것을 또한 어려운 말로 쓰레기 수집 – 가비지 콜렉션(Garbage collection)이라고도 한다.

다음의 예는 특정한 객체를 가리키는 변수를 없애는 예를 보여준다.

```
>>> a = 3
>>> b = 3
>>> del(a)
>>> del(b)
```

위의 예를 살펴보면 a와 b가 3이란 객체를 가리켰다가 del이란 파이썬 내장함수에 의해서 가리키는 변수 a, b가 사라지게 되고 따라서 레퍼런스 카운트가 0이 되어서 객체 3도 메모리에서 사라지게 된다.

변수를 만드는 여러 가지 방법

방법 1)

```
>>> a, b = 'python', 'life'
```

방법 1처럼 터플로 a, b에 값을 대입할 수 있다. 이 방법은 아래 소개된 방법 2와 완전히 동일하다.

방법 2)

```
>>> (a, b) = ('python', 'life')
```

방법 1과 방법 2는 사실상 같다. 터플 부분에서도 언급했지만 터플은 괄호를 생략해도 된다.

방법 3)

```
>>> [a,b] = ['python', 'life']
```

방법 3처럼 리스트로 만들 수도 있다.

방법 4)

```
>>> a = b = 'python'
```

여러 개의 변수에 같은 값을 대입할 수도 있다.

위의 방법을 이용하여 파이썬에서는 두 변수 값을 바꾸는 매우 간단하고 쉬운 방법을 쓸 수 있다.

```
>>> a = 3
>>> b = 5
>>> a, b = b, a
>>> a
5
>>> b
3
```

처음에 a 에는 3, b에는 5라는 값이 대입되어 있었지만 a, b = b, a라는 문장을 수행한 뒤 그 값이 서로 바뀌었음을 확인 할 수 있다.

리스트 복사

여기서는 리스트라는 자료형에서 가장 혼동하기 쉬운 부분을 설명하려고 한다. 예를 보며 알아보도록 하자.

```
>>> a = [1,2,3]
>>> b = a
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```

위의 예를 유심히 살펴보면 b라는 변수에 a가 가리키는 리스트를 대입하였다. 그런 다음, a 리스트의 a[1]을 4라는 값으로 바꾸었을 때, a리스트만이 바뀌는 것이 아니라 b리스트도 똑같이 바뀌게 된다. 그 이유는 a, b 모두 같은 리스트인 [1, 2, 3]을 가리키고 있었기 때문이다. a, b는 이름만 다를 뿐이지 완전히 동일한 리스트를 가리키고 있는 변수이다.

그렇다면 b 변수를 생성할 때 a와 같은 값을 가지면서 a 가 가리키는 리스트와는 다른 리스트를 가리키게 하는 방법은 없을까? 다음의 두 가지 방법이 있다.

방법 1) [:] 이용

첫 번째 방법으로는 아래와 같이 리스트 전체를 가리키는 [:]을 이용하는 것이다.

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a[1] = 4
>>> a[1, 4, 3]
>>> b[1, 2, 3]
```

위의 예에서 보듯이 a 리스트 값을 변형하더라도 b리스트에는 영향을 끼치지 않음을 볼수 있다.

방법 2) copy 모듈 이용

두 번째 방법은 copy 모듈을 이용하는 방법이다. 다음의 예제에서 보면 from copy import copy라는 처음 보는 형태가 나오는데 이것은 2장의 모듈 부분에서 자세히 다루게 된다. 여기서는 단순히 copy라는 함수를 쓰기 위해서 사용되는 것이라고만 알아두자.

```
>>> from copy import copy
>>> b = copy(a)
```

위의 예 b = copy(a) 는 b = a[:]과 동일하다.

두 변수가 같은 값을 가지면서 다른 객체를 제대로 생성했는 지를 확인하려면 다음과 같이 하면 된다. 즉, is 함수를 이용한다. 이 함수는 서로 동일한 객체인지 아닌지에 대한 판단을 하여 참과 거짓을 돌려준다.

```
>>> b is a
False
```

위의 예에서 b is a 가 False를 돌려주므로 b와 a 가 다른 객체임을 알 수 있다.

2) 제어문

이제는 **if**, **while**, **for**등의 **제어문**에 대해서 알아볼 것이다.

이러한 것들을 알아보기 전에 집을 짓는 경우를 생각해 보자. 나무, 돌, 시멘트 등은 집을 짓기 위한 재료가 될 것이고, 철근 같은 것은 집의 뼈대가 될 것이다. 이것은 프로그램의 경우와 매우 비슷하다. 즉 나무, 돌, 시멘트 등의 재료는 바로 자료형이 될 것이고 집의 뼈대를 이루는 철근이 바로 우리가 이 곳에서 알아볼 제어문이 될 것이다. 즉, 자료형을 그 근간으로 하여 그것들의 흐름을 원활히 효율적으로 만들어 주는 것, 이것이 바로 지금부터 공부할 제어문이다.

[1] if문

다음과 같은 상상을 해 보자.

> “돈이 있으면 택시를 타고 가고 돈이 없으면 걸어 간다.”

위와 같은 상황은 우리 주변에서 언제든지 일어 날 수 있는 상황중의 하나이다. 프로그래밍이란 것도 사람이 만드는 것이라서 위와 같은 문장처럼 조건을 판단해서 그 상황에 맞게 처리해야 할 경우가 생기게 된다. 이렇듯 조건을 판단하여 해당 조건에 맞는 상황을 수행하는데 쓰이는 것이 바로 if문이다.

위와 같은 상황을 파이썬에서는 다음과 같이 만들 수 있다.

```
>>> money = 1
>>> if money:
...     print "택시를 타고 가라"
... else:
...     print "걸어가라"
...
택시를 타고 가라
```

if문의 기본 구조
다음의 구조가 if와 else를 이용한 기본적인 구조이다.

```
if <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   ...
else:
   <수행할 문장A>
   <수행할 문장B>
   ...
```

조건문을 테스트 해서 참이면 if문 바로 다음의 문장들을 수행하고 조건문이 거짓이면 else문 다음의 문장들을 수행하게 된다.

들여쓰기(indentation)
if 문을 만들 때는 다음 처럼 if <조건문>: 다음의 문장부터 if문에 속하는 모든 문장들에 들여쓰기를 해 주어야 한다.

```
if <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   <수행할 문장3>
```

위에서 보는 것과 같이 조건문이 참일 경우 <수행할 문장1>을 들여쓰기 하였고 <수행할 문장2>, <수행할 문장3>도 들여쓰기를 해 주었다. 이것은 파이썬을 처음 배우는 사람들에게 매우 혼동스러운 부분이기도 하니 여러번 연습을 해 보는 것이 좋다.

다음처럼 하면 에러가 난다.

```
if <조건문>:
   <수행할 문장1>
<수행할 문장2>
   <수행할 문장3>
```

<수행할 문장2>가 들여쓰기가 되지 않았다.

또는

```
if <조건문>
   <수행할 문장1>
   <수행할 문장2>
     <수행할 문장3>
```

<수행할 문장3>이 들여쓰기는 되었지만 <수행할 문장1>이나 <수행할 문장2>와의 들여쓰기의 깊이가 틀리다. 즉 들여쓰기는 언제나 같은 깊이로 해 주어야 한다.

그렇다면 들여쓰기는 공백으로 하는 것이 좋을까? 아니면 탭으로 하는 것이 좋을까? 이것에 대한 논란은 파이썬을 사용하는 사람들 사이에서 아직도 계속되고 있는 것인데 탭으로 하자는 쪽, 공백으로 하자는 쪽 모두다 일치하는 내용은 단 하나, 둘을 혼용해서 쓰지 말자는 것이다. 공백으로 할 거면 항상 공백으로 하고 탭으로 할 거면 항상 탭으로 하자는 말이다. 탭이나 공백은 프로그램 소스에서 눈으로 보이는 것이 아니기 때문에 혼용해서 쓰면 에러의 원인이 되기도 한다. 주의 하도록 하자.

[참고] ':'을 잊지 말자
if(조건문) 다음에는 반드시 ':'이 오게 된다. 이것은 특별한 의미라기보다는 파이썬의 문법 구조이다. 왜 하필이면 ':'인가에 대해서 궁금하다면 파이썬을 만든 Guido에게 직접 물어보아야 할 것이다. while이나 for, def, class 에도 역시 그 문장의 끝에 ':'이 항상 들어간다. 이 ':'을 가끔 빠뜨리는 경우가 많은데 주의하도록 하자.

파이썬이 다른 언어보다 보기가 쉽고 소스코드가 간결한 이유가 바로 ':'을 사용하여 들여쓰기(indentation)를 하게끔 만들기 때문이다. 하지만 숙련된 프로그래머들이 파이썬을 접할 때 가장 혼란스러운 부분이기도 하다.

if 문을 다른 언어에서는 { } 이런 기호로 감싸지만 파이썬에서는 들여쓰기로 해결한다.

조건문이란 무엇인가?
if <조건문>에서 사용되는 조건문이란 참과 거짓을 판단하는 문장을 말한다. 자료형의 참과 거짓에 대해서는 이미 알아 보았지만 몇가지만 다시 알아보면 다음과 같은 것들이 있다.

자료형의 참과 거짓

	참	거짓
숫자	0 이 아닌 숫자	0
문자열	"abc"	""
리스트	[1,2,3]	[]
터플	(1,2,3)	()
딕셔너리	{"a":"b"}	{}

따라서 위의 예에서 보았던

```
>>> money = 1
>>> if money:
```

에서 조건문은 money가 되고 money는 1이기 때문에 참이 되어 if문 다음의 문장을 수행하게 되는 것이다.

비교연산자
하지만 조건판단을 하는 경우는 자료형보다는 비교 연산자(<, >, ==, !=, >=, <=)를 쓰는 경우가 훨씬 많다. 다음의 테이블은 비교연산자를 잘 설명해 준다.

x < y	x가 y보다 작다
x > y	x가 y보다 크다
x == y	x와 y가 같다
x != y	x와 y가 같지 않다
x >= y	x가 y보다 크거나 같다
x <= y	x가 y보다 작거나 같다

예를 통해서 위의 연산자들에 대해서 알아보자.

```
>>> x = 3
>>> y = 2
>>> x > y
True
>>>
```

x에 3을 y에 2를 대입한 다음에 x > y라는 조건문을 수행하니까 True를 리턴한다.
그 이유는 x > y라는 조건문이 참이기 때문이다.

```
>>> x < y
False
```

위의 조건문은 거짓이기 때문에 False를 리턴한다.

```
>>> x == y
False
```

x와 y는 같지 않다. 따라서 위의 조건문은 거짓이다.

```
>>> x != y
True
```

x와 y는 같지 않다. 따라서 위의 조건문은 참이다.

앞의 경우를 다음처럼 바꾸어 보자.

> " 만약 3000원 이상의 돈을 가지고 있으면 택시를 타고 그렇지 않으면 걸어가라 "

위의 상황을 다음처럼 프로그래밍 할 수 있을 것이다.

```
>>> money = 2000
>>> if money >= 3000:
...     print "택시를 타고 가라"
... else:
...     print "걸어가라"
...
걸어가라
>>>
```

money >= 3000 이란 조건문이 거짓이 되기 때문에 else문 다음의 문장을 수행하게 된다.

and, or, not
또다른 조건 판단에 쓰이는 것으로 and, or, not이란 것이 있다.

각각의 연산자는 다음처럼 동작을 한다.

예	설명
x or y	x와 y 둘중에 하나만 참이면 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

다음의 예를 통해 위의 사항을 반영해 보도록 하자.

> "돈이 3000원 이상 있거나 풀러줄 시계가 있다면 택시를 타고 그렇지 않으면 걸어가라 "

```
>>> money = 2000
>>> watch = 1
>>> if money >= 3000 or watch:
...     print "택시를 타고 가라"
... else:
...     print "걸어가라"
...
택시를 타고 가라
>>>
```

money는 2000이지만 watch가 1이기 때문에 money >= 3000 or watch라는 조건문이 참이 되기 때문에 if문 다음의 문장이 수행된다. 여기서 watch = 1을 " 시계가 있다 " 라는 의미에 적용했음에 주목하도록 하자. 이것은 프로그래밍을 할 때 자주 사용되는 트릭이다. 그렇다면 " 시계가 없다 " 는 watch = 0으로 해야함은 두말할 나위가 없을 것이다.

x in s, x not in s

더 나아가서 파이썬에서는 조금 더 재미있는 조건문들을 제공한다. 바로 다음과 같은 것들이다.

- x in 리스트, x not in 리스트
- x in 터플, x not in 터플
- x in 문자열, x not in 문자열

'in' 이라는 영어단어가 '~안에'라는 뜻을 가졌음을 상기해 보면 다음의 예들이 쉽게 이해가 될 것이다.

```
>>> 1 in [1, 2, 3]
True
>>> 1 not in [1, 2, 3]
False
```

위의 첫 번째 예는 " [1, 2, 3]이라는 리스트 안에 1이 있는가? " 라는 조건문이다. 1은 [1, 2, 3]안에 있으므로 참이 되어 True를 리턴한다. 두 번째 예는 "[1, 2, 3]이라는 리스트 안에 1이 없는가? " 라는 조건문이다. 1은 [1, 2, 3]안에 있으므로 거짓이 되어 False를 리턴한다.

다음은 터플과 문자열의 적용예를 보여준다.

```
>>> 'a' in ('a', 'b', 'c')
True
>>> 'j' not in 'python'
True
```

위의 것들을 이용하여 우리가 계속 사용해온 택시예제에 적용시켜 보자.
"만약 주머니에 돈이 있으면 택시를 타고, 없으면 걸어가라 "

```
>>> pocket = ['paper', 'handphone', 'money']
>>> if 'money' in pocket:
...     print "택시를 타고 가라"
... else:
...     print "걸어가라"
...
택시를 타고 가라
>>>
```

['paper', 'handphone', 'money']라는 리스트에 안에 'money'가 있으므로 'money' in pocket은 참이 되어서 if문 다음의 문장이 수행되었다.

elif (다중 조건 판단)
if와 else만을 가지고서는 다양한 조건 판단을 하기가 어렵다. 다음과 같은 예만 하더라도 if와 else만으로는 조건 판단에 어려움을 겪게 된다.

"지갑에 돈이 있으면 택시를 타고, 지갑엔 돈이 없지만 시계가 있으면 택시를 타고,
돈도 없고 시계도 없으면 걸어가라 "

위의 문장을 보면 조건을 판단하는 부분이 두 군데가 있다. 먼저 지갑에 돈이 있는지를 판단해야 하고 지갑에 돈이 없으면 다시 시계가 있는지를 판단한다.

if와 else만으로 위의 문장을 표현 하려면 다음과 같이 할 수 있을 것이다.

```
>>> pocket = ['paper', 'handphone']
>>> watch = 1
>>> if 'money' in pocket:
...     print "택시를 타고가라"
... else:
...     if watch:
...             print "택시를 타고가라"
...     else:
...             print "걸어가라"
...
택시를 타고가라
>>>
```

언뜻 보기에도 이해하기가 쉽지 않고 산만한 느낌이 든다. 위와 같은 점을 보완하기 위해서 파이썬에서는 다중 조건 판단을 가능하게 하는 elif라는 것을 사용한다.

위의 예를 elif를 이용하면 다음과 같이 할 수 있다.

```
>>> pocket = ['paper', 'handphone']
>>> watch = 1
>>> if 'money' in pocket:
...     print "택시를 타고가라"
... elif watch:
...     print "택시를 타고가라"
... else:
...     print "걸어가라"
...
택시를 타고가라
```

즉, elif는 if <조건문>:에서 <조건문>이 거짓일 때 수행되게 된다. if, elif, else의 기본 구조는 다음과 같다.

```
If <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   ...
elif <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   ...
elif <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   ...
...
else:
  <수행할 문장1>
  <수행할 문장2>
  ...
```

위에서 보듯이 elif는 개수에 제한 없이 사용할 수 있다. (참고: 이것은 마치 C언어의 switch문과 비슷한 것이다.)

pass의 사용

가끔 조건문을 판단하고 참 거짓에 따라 행동을 정의 할 때 아무런 일도 하지 않게끔 설정을 하고 싶을 때가 생기게 된다. 다음의 예를 보자.

> "지갑에 돈이 있으면 가만히 있고 지갑에 돈이 없으면 시계를 끌러라 "

위의 예를 pass를 적용해서 구현해 보자.

```
>>> pocket = ['paper', 'money', 'handphone']
>>> if 'money' in pocket:
...     pass
... else:
...     print "시계를 끌른다"
...
```

pocket이라는 리스트 안에 'money'란 문자열이 있기 때문에 if문 다음문장인 pass가 수행되었고 아무런 결과값도 보여주지 않는 것을 확인 할 수 있다.

한줄 짜리 if문

위의 예를 보면 if문 다음의 수행할 문장이 한줄이고 else문 다음에 수행할 문장도 한줄이다. 이렇게 수행할 문장이 한줄일때 조금 더 간편한 방법이 있다. 위에서 알아본 pass를 사용한 예는 다음처럼 간략화 할 수 있다.

```
>>> pocket = ['paper', 'money', 'handphone']
>>> if 'money' in pocket: pass
... else: print "시계를 끌른다"
...
```

if문 다음의 수행할 문장을 ':'뒤에 바로 적어 주었다. else문 역시 마찬가지이다. 이렇게 하는 이유는 때때로 이렇게 하는 것이 보기에 편하게 때문이다.

[2] while문

반복해서 문장을 수행해야 할 경우 while문을 사용한다.

다음은 while문의 기본 구조이다.

```
while <조건문>:
   <수행할 문장1>
   <수행할 문장2>
   <수행할 문장3>
   ...
```

조건문이 참인 동안 while문 아래의 문장들을 계속해서 수행 하게 된다.

" 열 번 찍어 안 넘어 가는 나무 없다 " 라는 속담을 파이썬에 적용시켜 보면 다음과 같이 될 것이다.

```
>>> treeHit = 0
>>> while treeHit < 10:
. . .     treeHit = treeHit +1
. . .     print "나무를 %d번 찍었습니다." % treeHit
. . .     if treeHit == 10:
. . .         print "나무 넘어갑니다."
. . .
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

위의 예에서 while문의 조건문은 treeHit < 10 이다. 즉 treeHit가 10보다 작은 동안에 while 문 안의 문장들을 계속 수행하게 된다. whlie문 안의 문장을 보면 제일 먼저 treeHit = treeHit + 1로 treeHit값이 계속 1씩 증가한다. 그리고 나무를 treeHit번 만큼 찍었음을 알리는 문장을 출력하고 treeHit가 10이 되면 " 나무 넘어갑니다 " 라는 문장을 출력하고 treeHit < 10라는 조건문이 거짓이 되어 while문을 빠져 나가게 된다.

여기서 treeHit = treeHit +1은 프로그래밍을 할 때 매우 자주 쓰이는 기법으로 treeHit의 값을 1만큼씩 증가시킬 목적으로 쓰이는 것이다. 이것은 treeHit += 1처럼 쓰기도 한다.

무한루프(Loop)
이번에는 무한루프에 대해서 알아보기로 하자. 무한 루프라 함은 무한히 반복한다는 의미이다. 파이썬에서 무한루프는 while문으로 구현 할 수가 있다. 우리가 사용하는 프로그램들 중에서 이 무한루프의 개념을 사용하지 않는 프로그램은 하나도 없을 정도로 이 무한루프는 자주 사용된다. 다음은 무한루프의 기본적인 형태이다.

```
while 1:
   <수행할 문장1>
   <수행할 문장2>
   ...
```

while의 조건문이 "1"이므로 조건문은 항상 참이 된다. while은 조건문이 참인 동안에 while에 속해 있는 문장들을 계속해서 수행하므로 위의 예는 무한하게 while문 내의 문장들을 수행할 것이다.

다음의 예를 보자.

```
>>> while 1:
. . .     print "Ctrl-C를 눌러야 while문을 빠져 나갈 수 있습니다."
. . .
Ctrl-C를 눌러야 while문을 빠져 나갈 수 있습니다.
Ctrl-C를 눌러야 while문을 빠져 나갈 수 있습니다.
Ctrl-C를 눌러야 while문을 빠져 나갈 수 있습니다.
....
```

위의 문장이 영원히 출력될 것이다. Ctrl-C를 눌러서 빠져 나가도록 하자. 하지만 위처럼 무한루프를 돌리는 경우는 거의 없을 것이다. 보다 실용적인 예를 들어보자.

다음을 따라해 보도록 하자.

```
>>> prompt = """
. . . 1. Add
. . . 2. Del
. . . 3. List
. . . 4. Quit
```

```
. . .
. . . Enter number: """
>>>
```

먼저 위와같이 여러줄짜리 문자열을 만든다.

```
>>> number = 0
>>> while number != 4:
. . .     print prompt
. . .     number = int(raw_input())
. . .

1. Add
2. Del
3. List
4. Quit

Enter number:
```

다음에 number라는 변수에 "0"이라는 값을 먼저 대입한다. 이렇게 하는 이유는 다음에 나올 while문의 조건문이 number != 4인데 number라는 변수를 먼저 설정해 놓지 않으면 number라는 변수가 존재하지 않는다는 에러가 나기 때문이다. while문을 보면 number가 4가 아닌 동안에 prompt를 출력하게 하고 사용자로부터 입력을 받아 들인다. 위의 예는 사용자가 4라는 값을 입력하지 않으면 무한히 prompt를 출력하게 된다. 여기서 number = int(raw_input())은 사용자의 숫자 입력을 받아들이는 것이라고만 알아두자. int나 raw_input함수에 대한 사항은 뒤의 내장함수 부분에서 자세하게 다룰 것이다.

while문 빠져 나가기(break)
while 문은 조건문이 참인 동안 계속해서 while문 안의 내용을 수행하게 된다. 하지만 강제로 while문을 빠져나가고 싶을 때가 생기게 된다.

커피 자판기를 생각해 보자. 커피가 자판기 안에 충분하게 있을 때는 항상 " 돈을 받으면 커피를 줘라 " 라는 조건문을 가진 while문이 수행된다. 하지만 돈을 받더라도 커피가 다 떨어져서 커피를 주지 않는다면 사람들은 자판기를 마구 발로 걷어 찰 것이다. 자판기가 온전하려면 커피의 양을 따로이 검사를 해서 커피가 다 떨어지면 while문을 멈추게 하고 " 판매중지 " 란 문구를 자판기에 보여야 할 것이다. 이렇게 while문을 강제로 멈추게 하는 것을 가능하게 해 주는 것이 바로 break이다.

다음의 예는 위의 가정을 파이썬으로 표현해 본 것이다.

예) break의 사용

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print "돈을 받았으니 커피를 줍니다."
...     coffee = coffee -1
...     print "남은 커피의 양은 %d 입니다." % coffee
...     if not coffee:
...         print "커피가 다 떨어졌습니다. 판매를 중지합니다."
...         break
...
```

money가 300으로 고정되어 있으니까 while money:에서 조건문인 money는 0이 아니기 때문에 항상 참이다. 따라서 무한 루프를 돌게 된다. 그리고 while문의 내용을 한번 수행할 때 마다 coffee = coffee - 1에 의해서 coffee의 개수가 한 개씩 줄어들게 된다. 만약 coffee가 0이 되면 if not coffee: 라는 문장에서 not coffee가 참이 되므로 if문 다음의 문장들이 수행이 되고 break가 호출되어 while문을 빠져 나가게 된다.

하지만 실제 자판기는 위처럼 작동하지는 않을 것이다. 다음은 자판기의 실제 과정과 비슷하게 만들어 본 예이다. 이해가 안 되더라도 걱정하지 말자. 아래의 예는 조금 복잡하니까 대화형 인터프리터를 이용하지 말고 에디터를 이용해서 작성해 보자.

```
# -*- coding: euc-kr -*-
# coffee.py

coffee = 10
while 1:
   money = int(raw_input("돈을 넣어 주세요: "))
   if money == 300:
       print "커피를 줍니다."
       coffee = coffee -1
   elif money > 300:
       print "거스름돈 %d를 주고 커피를 줍니다." % (money -300)
       coffee = coffee -1
   else:
       print "돈을 다시 돌려주고 커피를 주지 않습니다."
       print "남은 커피의 양은 %d개 입니다." % coffee
   if not coffee:
       print "커피가 다 떨어졌습니다. 판매를 중지 합니다."
       break
```

위의 프로그램 소스를 따로 설명하지는 않겠다. 독자가 위의 것을 이해할 수 있다면 지금껏 알아온 if문이나 while문을 마스터 했다고 보면 된다. 다만 money = int(raw_input(" 돈을 넣어 주세요: "))라는 문장은 사용자로부터 입력을 받는 부분이고 입력 받은 숫자를 money라는 변수에 대입하는 것이라고만 알아두자.

만약 위의 프로그램 소스를 에디터로 작성해서 실행시키는 법을 모른다면 1장의 마지막 부분 또는 부록의 에디터 사용법을 참고하도록 하자.

while문 처음으로 되돌아가기(continue)

while문 안의 문장을 수행할 때 어떤 조건을 검사해서 조건에 맞지 않는 경우 while문을 빠져나가는 것이 아니라 다시 while문의 맨 처음으로 돌아가게 하고 싶을 경우가 생기게 된다. 만약 1부터 10까지의 수중에서 홀수만을 출력하는 것을 while문을 이용해서 작성한다고 생각해 보자. 독자는 어떤 방법을 사용할 것인가?

예) continue의 사용

```
>>> a = 0
>>> while a < 10:
...     a = a+1
...     if a % 2 == 0: continue
...     print a
...
1
3
5
7
9
```

위의 예는 1부터 10까지의 수 중 홀수만을 출력하는 예이다. a가 10보다 작은 동안 a는 1만큼씩 계속 증가한다. if a % 2 == 0 (2로 나누었을 때 나머지가 0인 경우)이 참이 되는 경우는 a가 짝수일 때이다. 즉, a가 짝수이면 continue 문장을 수행한다. 이 continue문은 while문의 맨 처음으로 돌아가게 하는 명령어이다. 따라서 위의 예에서 a가 짝수이면 print a는 수행되지 않을 것이다.

[3] for문

파이썬의 특징을 가장 잘 대변해주는 것이 바로 이 for문이다. for문은 매우 유용하고 사용할 때 문장 구조가 한눈에 들어오며 이것을 잘만 쓰면 프로그래밍이 즐겁기까지 하다.

for문의 기본구조
for 문의 기본적인 구조는 다음과 같다.

```
for 변수 in 리스트(또는 터플, 문자열):
   <수행할 문장1>
   <수행할 문장2>
   ...
```

리스트의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입해서 <수행할 문장1>, <수행할 문장2>,,,를 수행한다.

예제를 통해 for 알아보기
for문은 예제를 통해서 보는 것이 가장 알기 쉽다. 예제를 따라해 보도록 하자.

예 1) 전형적인 for문

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:
...     print i
...
one
two
three
```

['one', 'two', 'three']라는 리스트의 첫 번째 요소인 'one'이 먼저 i변수에 대입된 후 print i라는 문장을 수행한다. 다음에 'two'라는 두 번째 요소가 i변수에 대입된 후 print i문장을 수행하고 리스트의 마지막 요소까지 이것을 반복한다.

for문의 쓰임새를 알기 위해서 다음을 가정해 보자.

> “ 총 5명의 학생이 시험을 보았는데 시험점수가 60점이 넘으면 합격이고 그렇지 않으면 불합격이다. 합격인지 불합격인지에 대한 결과를 보여준다. ”

우선 5명의 학생의 시험성적을 리스트로 표현 해 보았다.

```
mark = [90, 25, 67, 45, 80]
```

1번 학생은 90점이고 5번 학생은 80점이다.

이런 점수를 차례로 검사해서 합격했는지 불합격했는지에 대한 통보를 해주는 프로그램을 만들어 보자. 역시 에디터로 만들어 보자.

```
# marks1.py
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
   number = number +1
   if mark >= 60:
       print "%d번 학생은 합격입니다." % number
   else:
       print "%d번 학생은 불합격입니다." % number
```

각각의 학생에게 번호를 붙이기 위해서 number라는 변수를 이용하였다. 점수 리스트인 marks에서 차례로 점수를 꺼내어 mark라는 변수에 대입하고 for안의 문장들을 수행하게 된다. 우선 for문이 한번씩 수행될 때마다 number는 1씩 증가하고 mark가 60이상이면 합격 메시지를 출력하고 60을 넘지 않으면 불합격 메시지를 출력한다.

for와 continue

while문에서 알아보았던 continue가 for문에서도 역시 동일하게 적용이 된다. 즉, for문 안의 문장을 수행하는 도중에 continue문을 만나면 for문의 처음으로 돌아가게 된다.

위의 예제를 그대로 이용해서 60점 이상인 사람에게는 축하 메시지를 보내고 나머지 사람에게는 아무런 메시지도 전하지 않는 프로그램을 만들어 보자.

```
# marks2.py
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
   number = number +1
   if mark < 60: continue
   print "%d번 학생 축하합니다. 합격입니다. " % number
```

점수가 60점 이하인 학생일 경우에는 mark < 60이 참이 되어 continue문이 수행된다. 따라서 축하 메시지를 출력하는 부분인 print문을 수행하지 않고 for문의 첫부분으로 돌아가게 된다.

for와 range함수

for문은 range라는 숫자 리스트를 자동으로 만들어 주는 함수와 함께 사용되는 경우가 많다. 다음은 range함수의 간단한 사용법이다.

```
>>> a = range(10)
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

위에서 보는 것과 같이 range(10)은 0부터 9까지의 숫자 리스트를 만들어 준다.

시작 번호와 끝 번호를 지정하려면 다음과 같이 해야 한다. 끝 번호는 포함되지 않는다.

```
>>> a = range(1, 11)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

위처럼 시작 숫자를 정해 줄 수도 있다.

for와 range를 이용하면 1부터 10까지 더하는 것을 다음과 같이 쉽게 구현할 수 있다.

예) 1부터 10까지의 합

```
>>> sum = 0
>>> for i in range(1, 11):
```

```
. . .     sum = sum + i
. . .
>>> print sum
55
```

range(1, 11)은 위에서 보았듯이 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]이라는 리스트를 만들어 준다. 따라서 위의 예에서 i변수에 리스트의 숫자들이 하나씩 차례로 대입되면서 sum = sum + i라는 문장을 수행하게 되어 sum은 최종적으로 55가 되게 된다.

또한 우리가 앞서 살펴 보았던 60점 이상이면 합격인 예제도 range함수를 이용해서 적용시킬 수도 있다. 다음을 보자.

```
#marks3.py
marks = [90, 25, 67, 45, 80]

for number in range(len(marks)):
   if marks[number] < 60: continue
   print "%d번 학생 축하합니다. 합격입니다." % (number+1)
```

len이라는 함수가 처음 나왔는데 len함수는 리스트의 요소 개수를 돌려주는 함수이다. 따라서 위에서 len(marks)는 5가 될것이고 range(len(marks))는 range(5)가 될 것이다. 따라서 number변수에는 차례로 0부터 4까지의 숫자가 대입될 것이고 makrs[number]는 차례대로 90, 25, 67, 45, 80이라는 값을 갖게 될 것이다.

다양한 for문의 사용

```
>>> a = [(1,2), (3,4), (5,6)]
>>> for (first, last) in a:
. . .     print first + last
. . .
3
7
11
```

위의 예는 a리스트의 요소 값이 터플이기 때문에 각각의 요소들이 자동으로 (first, last)라는 변수에 대입되게 된다.

이것은 이미 살펴보았던 터플을 이용한 변수 값 대입방법과 매우 비슷한 경우이다.

```
>>> (first, last) = (1, 2)
```

for와 range를 이용한 구구단
for와 range함수를 이용하면 단 4줄만으로 구구단을 출력해 볼 수가 있다.

```
>>> for i in range(2,10):
...     for j in range(1, 10):
...         print i*j,
...     print '\n'
...
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

위의 예를 보면 for가 두 번 사용되었다. range(2, 10)은 [2, 3, 4, 5, 6, 7, 8, 9]가 되고 각각의 숫자가 차례로 i에 대입된다. i가 처음 2일 때 for 문을 만나게 된다. range(1, 10)은 [1, 2, 3, 4, 5, 6, 7, 8, 9]가 되고 각각의 숫자는 j에 대입되고 그 다음 문장인 print i*j를 수행한다. 따라서 i가 2일 때 2*1, 2*2, 2*3, , , ,2*9 까지 차례로 수행되며 그 값을 출력하게 된다. 그 다음에는 i가 3일 때 역시 2일 때와 마찬가지로 수행될 것이고 i가 9일 때까지 계속 반복되게 된다.

위에서 print i*j, 처럼 print i*j뒤에 콤마(',')를 넣어준 이유는 해당 결과값을 출력할 때 다음 줄로 넘어가지 않고 그 줄에 계속해서 출력하기 위한 것이다. 이것은 잘 사용되지는 않지만 콤마 연산자라는 것이다. 그 다음의 print '\n'은 2단, 3단, , 등을 구분하기 위해서 두 번째 for문이 끝나면 결과 값을 다음 줄부터 출력하게 해주는 문장이다.

지금껏 우리는 프로그램의 흐름을 제어하는 if, while, for등에 대해서 알아보았다. 독자는 while과 for를 보면서 두 가지가 아주 흡사하다는 느낌을 받았을 것이다. 실제로 for문을 쓰는 부분을 while로 바꿀 수 있는 경우가 많고 while문을 쓴 부분을 for문으로 바꾸어서 사용할 수 있는 경우가 많다. 선택은 독자의 자유이다.

04. 입출력

지금껏 공부한 내용을 근간으로 하여 이제 함수, 입력과 출력, 파일처리방법등에 대해서 알아 보기로 하자.

입출력은 프로그래밍 설계와 관련이 있다. 프로그래머는 프로그램을 어떤식으로 동작하게 만들어야겠다는 디자인을 먼저 하게되는데 그 때 가장 중요한 부분이 바로 입출력의 설계이다. 특정 프로그램만 사용하는 함수를 만들 것인지 아니면 모든 프로그램들이 공통으로 사용하는 함수를 만들것인지, 더 나아가 오픈API로 공개하여 모든 외부 프로그램들이 사용할 수 있게끔 만들것인지에 대한 그 모든것들이 전부 이 입출력과 관련이 있다.

1) 함수

함수를 설명하기 전에 믹서기를 생각해보자. 우리는 믹서기에 과일을 넣는다. 그리고 믹서를 이용해서 과일을 갈아서 과일 쥬스를 만들어 낸다. 이러한 일들은 우리의 생활 주변에서 언제든지 찾아 볼 수 있는 일이다. 우리가 믹서기에 넣는 과일은 입력이 되고 과일 쥬스는 그 출력(리턴값)이 된다.
그렇다면 믹서기는 무엇인가?

(by http://www.wpclipart.com)

바로 우리가 여기서 알고자 하는 함수이다. 입력을 가지고 어떤 일을 수행한 다음에 결과물을 내어놓는 것, 이것이 함수가 하는 일이다. 우리는 어려서부터 함수가 무엇인지에 대해서 공부했지만 이것에 대해서 깊숙이 고민해 본 적은 별로 없다. 하지만 우리는 함수에 대해서 조금 더 생각해 보는 시간을 가져야 한다. 프로그래밍에 있어서 함수란 것은 정말 중요하기 때문이다.

일차함수 y = 2x + 3 이것도 함수이다. 하지만 우리는 이것을 수학시간에 배웠던 직선그래프로만 알고 있지 x에 어떤 값을 넣었을 때 어떤 변화에 의해서 y값이 나온다는 생각은 대부분 해보지 않았을 것이다.

생각을 달리하고 파이썬 함수에 대해서 깊이 들어가 보도록 하자.

함수를 사용하는 이유?
가끔 프로그래밍을 하다 보면 똑같은 내용을 자신이 반복해서 적고 있는 것을 발견할 때가 있다. 이 때가 바로 함수가 필요한 때이다. 여러 번 반복해서 사용된다는 것은 언제고 또다시 사용할 만한 가치가 있는 부분이라는 뜻이다. 즉, 이러한 경우 이것을 한 뭉치로 묶어서 " 어떤 입력값을 주었을 때 어떤 리턴값을 돌려준다 " 라는 식의 함수를 작성하는 것이 현명한 일일 것이다.

함수를 사용하는 또 다른 이유로는 자신이 만든 프로그램을 함수화 시켜 놓으면 프로그램의 흐름을 일목요연하게 감지할 수 있게 된다. 마치 큰 공장에서 한 물건이 나올 때 여러 가지 공정을 거쳐가는 것처럼 자신이 원하는 결과값이 함수들을 거쳐가면서 변화되는 모습을 볼 수 있다.

이렇게 되면 프로그램의 흐름도 잘 파악할 수 있게 되고 에러가 어디에서 나는지도 금방 알아차릴 수 있게 된다. 함수를 잘 이용하고 적절한 함수를 만들 줄 아는 사람을 훌륭한 프로그래머라고 할 것이다.

파이썬 함수의 구조
파이썬의 함수의 모습은 다음과 같다.

```
def 함수명(입력 인수):
   <수행할 문장1>
   <수행할 문장2>
   ...
```

def는 함수를 만들 때 사용된다. 함수명은 함수를 만드는 사람이 임의로 만드는 것이다. 마치 변수이름을 정하는 것과 같은 이치이다. 함수명 다음에 괄호 안에 있는 입력인수라는 것은 이 함수에 입력으로 넣어주는 값이다. 입력 인수의 개수에는 제한이 없다. 다음에 if, while, for문 등과 마찬가지로 수행할 문장을 수행한다.

가장 간단하지만 많은 것을 설명해 주는 다음의 예를 보도록 하자.

```
def sum(a, b):
   return a + b
```

위 함수의 의미는 다음과 같이 정의된다.

> " sum이라는 것은 함수명이고 입력값으로 두개의 값을 받으며 리턴값은 두 개의 입력값을 더한 값이다. "

여기서 return은 함수의 결과 값을 돌려주는 명령어이다. 직접 위의 함수를 만들어 보고 사용해 보자.

```
>>> def sum(a,b):
. . .     return a+b
. . .
>>>
```

위와 같이 sum함수를 먼저 만들자.

```
>>> a = 3
>>> b = 4
>>> c = sum(a, b)
>>> print c
7
```

a에 3, b에 4를 대입한 다음 이미 만들었던 sum함수에 a와 b를 입력값으로 주어서 c라는 함수의 리턴값을 돌려 받는다. print c로 c의 값을 확인할 수 있다.

함수의 입력값과 리턴값

프로그래밍을 공부할 때 어려운 부분 중 하나가 용어의 혼용이라고 할 수 있다. 많은 원서들을 보기도 하고 누군가의 번역본을 보기도 하면서 우리는 갖가지 용어들을 익힌다. 입력 값을 다른 말로 함수의 인수, 입력인수 등으로 말하기도 하고 리턴 값을 출력 값, 결과 값, 돌려주는 값 등으로 말하기도 한다. 이렇듯 많은 용어들이 다른 말로 표현되지만 의미는 동일한 경우가 많다. 이런 것들에 대해 기억해 놓아야만 머리가 덜 아플 것이다.

함수는 들어온 입력값을 가지고 어떤 처리를 하여 적절한 리턴값을 돌려주는 블랙박스와 같다.

```
입력값 ---> 함수(블랙박스) ----> 리턴값
```

함수에 들어오는 입력값과 리턴값에 대해서 자세히 알아보도록 하자.

평범한 함수

입력 값이 있고 리턴값이 있는 함수가 평범한 함수이다. 앞으로 독자가 프로그래밍을 할 때 만들 함수의 대부분은 이러한 형태일 것이다.

```
def 함수이름(입력인수):
   <수행할 문장>
   ...
   return 결과값
```

평범한 함수의 전형적인 예를 한번 보도록 하자.

```
def sum(a, b):
   result = a + b
   return result
```

두 개의 입력값을 받아서 서로 더한 결과값을 돌려주는 함수이다.

위의 함수를 사용하는 방법은 다음과 같다. 다음을 따라 해 보자. 우선 sum함수를 만들자.

```
>>> def sum(a, b):
. . .     result = a + b
. . .     return result
. . .
>>>
```

다음에 입력값을 주고 리턴값을 돌려 받아 보자.

```
>>> a = sum(3, 4)
>>> print a
7
```

위처럼 입력값과 리턴값이 있는 함수는 다음처럼 사용된다.

```
리턴값받을변수 = 함수명(입력인수1, 입력인수2, , ,)
```

입력값이 없는 함수
입력값이 없는 함수가 존재할까? 당연히 그렇다. 다음을 보자.

```
>>> def say():
. . .     return 'Hi'
. . .
>>>
```

say라는 이름의 함수를 만들었다. 하지만 함수이름 다음의 입력 인수부분을 나타내는 괄호 안이 비어있다.

이 함수를 어떻게 쓸 수 있을까? 다음과 같이 따라해 보자.

```
>>> a = say()
>>> print a
Hi
```

위의 함수를 쓰기 위해서는 say()처럼 괄호 안에 아무런 값도 넣어주지 않고 써야 한다. 위의 함수는 입력값은 없지만 리턴값으로 'Hi'라는 문자열을 돌려준다. 따라서 a = say()처럼 하면 a에는 'Hi'라는 문자열이 대입되게 되는 것이다.

즉, 입력값이 없고 리턴값만 있는 함수는 다음과 같이 사용된다.

```
리턴값받을변수 = 함수명()
```

리턴값이 없는 함수
리턴값이 없는 함수 역시 존재한다. 다음의 예를 보자.

```
>>> def sum(a, b):
. . .     print "%d, %d의 합은 %d입니다." % (a, b, a+b)
. . .
>>>
```

리턴값이 없는 함수는 돌려주는 값이 없기 때문에 다음과 같이 사용한다.

```
>>> sum(3, 4)
3, 4의 합은 7입니다.
```

즉, 리턴값이 없는 함수는 다음과 같이 사용된다.

```
함수명(입력인수1, 입력인수2, , ,)
```

실제로 리턴값이 없는지 알아보기 위해 다음의 예를 따라해 보자.

```
>>> a = sum(3, 4)
3, 4의 합은 7입니다.
```

아마도 독자는 다음과 같은 질문을 할지도 모른다. "3, 4의 합은 7입니다. " 라는 문장을 출력해 주었는데 리턴값이 없는 것인가? 이 부분이 초보자들이 혼동스러워 하는 부분이기도 한데 print 문은 함수내에서 사용되어지는 문장일 뿐이다. 돌려주는 값은 당연히 없다. 돌려주는 값은 return 명령어로만 가능하다.

이것을 확인해 보기 위해서 돌려받는 값을 a라는 변수에 대입하여 출력해 보면 리턴값이 있는지 알 수 있다.

```
>>> print a
None
```

a의 값이 None이다. 이 None이란 것은 거짓을 나타내는 자료형이라고 언급한 적이 있었다. sum함수처럼 돌려주는 값이 없을 때 a = sum(3, 4)처럼 쓰게 되면 함수 sum은 돌려주는 값으로 a변수에 None을 돌려주게 된다. 그렇다고 이것이 돌려주는 값이 있다는 걸로 생각하면 곤란하다.

입력값도 리턴값도 없는 함수
이것 역시 존재한다. 다음의 예를 보자.

```
>>> def say():
. . .     print 'Hi'
. . .
>>>
```

입력 값을 받는 곳도 없고 return문도 없으니 입력값도 리턴값도 없는 함수이다.

이것을 사용하는 방법은 단 한가지이다.

```
>>> say()
Hi
```

즉, 입력값도 리턴값도 없는 함수는 다음과 같이 사용된다.

```
함수명()
```

입력값이 몇 개가 될 지 모를 때는 어떻게 해야 할까?
입력값을 주었을 때 그 입력값들을 모두 더해주는 함수를 생각해 보자. 하지만 몇 개가 입력으로 들어올지 알 수 없을 때는 어떻게 해야 할지 난감할 것이다. 이에 파이썬에서는 다음과 같은 것을 제공한다.

```
def 함수이름(*입력변수):
  <수행할 문장>
  ...
```

입력인수 부분이 " *입력변수 " 로 바뀌었다.

다음의 예를 통해 사용법을 알아보자. sum_many(1,2) 하면 3을 sum_many(1,2,3)이라고 하면 6을 sum_many(1,2,3,4,5,6,7,8,9,10)은 55를 돌려주는 함수를 만들어 보자.

```
>>> def sum_many(*args):
. . .     sum = 0
. . .     for i in args:
. . .         sum = sum + i
. . .     return sum
. . .
>>>
```

위에서 만든 sum_many라는 함수는 입력값이 몇 개든지 상관이 없다. 그 이유는 args라는 변수가 입력값들을 전부 모아서 터플로 만들어 주기 때문이다. 만약 sum_many(1, 2, 3)처럼 이 함수를 쓴다면 args는 (1, 2, 3)이 되고 sum_many(1,2,3,4,5,6,7,8,9,10)처럼 하면 args는 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)이 된다. 여기서 *args라는 것은 임의로 정한 변수명이다. *pey, *python처럼 아무이름으로 해도 된다. 단 args라는 것은 입력인수를 뜻하는 영어단어인 arguments라는 영어의 약자로 관례적인 표기법임을 알아두도록 하자.

실제로 한번 테스트를 해 보도록 하자.

```
>>> result = sum_many(1,2,3)
>>> print result
6
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)
>>> print result
55
```

def sum_many(*args) 처럼 *args만이 입력 인수로 올 수 있는 것은 아니다. 다음의 예를 보도록 하자.

```
>>> def sum_mul(choice, *args):
. . .     if choice == "sum":
. . .         result = 0
. . .         for i in args:
. . .             result = result + i
. . .     elif choice == "mul":
. . .         result = 1
. . .         for i in args:
. . .             result = result * i
. . .     return result
. . .
>>>
```

위의 예는 입력 인수로 choice와 *args라는 것을 받는다. 따라서 다음과 같이 쓸 수 있을 것이다. sum_mul('sum', 1,2,3,4) 또는 sum_mul('mul', 1,2,3,4,5)처럼 choice부분에는 'sum'이나 'mul'이라는 문자열을 그리고 그 뒤에는 개수에 상관없는 숫자를 입력으로 준다.

실제로 한번 테스트를 해 보도록 하자.

```
>>> result = sum_mul('sum', 1,2,3,4,5)
>>> print result
15
>>> result = sum_mul('mul', 1,2,3,4,5)
>>> print result
120
```

함수의 리턴값은 언제나 하나이다.
먼저 다음의 함수를 만들어 보자.

```
>>> def sum_and_mul(a,b):
. . .     return a+b, a*b
```

이것을 다음과 같이 호출하면 어떻게 될까?

```
>>> a = sum_and_mul(3,4)
```

리턴값은 a+b와 a*b 두 개인데 리턴값을 받아들이는 변수는 a 하나만 쓰였으니 에러가 나지 않을까? 당연한 의문이다. 하지만 에러는 나지 않는다. 그 이유는 리턴값이 두 개가 아니라 하나라는 데 있다. 함수의 리턴값은 터플값 하나로 돌려주게 된다.

즉 a 변수는 위의 sum_and_mul 함수에 의해서 다음과 같은 값을 가지게 된다.

```
a = (7, 12)
```

즉, 리턴값으로 (7, 12)라는 터플 값을 갖게 되는 것이다.

이것을 두 개의 리턴값처럼 받고 싶다면 다음과 같이 호출하면 될 것이다.

```
>>> sum, mul = sum_and_mul(3, 4)
```

이렇게 호출한다면 이것의 의미는 sum, mul = (7, 12)가 되어서 sum은 7이 되고 mul은 12가 될 것이다.

또, 다음과 같은 의문이 생길 수도 있다.

```
>>> def sum_and_mul(a,b):
. . .     return a+b
. . .     return a*b
. . .
>>>
```

위와 같이 하면 두 개의 리턴값을 돌려주지 않을까? 하지만 파이썬에서 위와 같은 함수는 참 어리석은 함수이다.

위의 함수는 다음과 완전히 동일하다.

```
>>> def sum_and_mul(a,b):
...     return a+b
...
>>>
```

즉, 함수는 return문을 만나는 순간 return값을 돌려준 다음에 함수를 빠져나가게 된다.

return의 또 다른 쓰임새

특별한 경우에 함수를 빠져나가기를 원할 때 return만 단독으로 써서 함수를 즉시 빠져나갈 수 있다. 다음 예를 보자.

```
>>> def say_nick(nick):
...     if nick == "바보":
...         return
...     print "나의 별명은 %s 입니다." % nick
...
>>>
```

위의 함수는 입력값으로 nick이란 변수를 받아서 문자열을 출력하는 함수이다. 이 함수 역시 리턴값은 없다. (문자열을 출력한다는 것과 리턴값이 있다는 것은 전혀 다른 말이다. 혼동하지 말도록 하자, 함수의 리턴값은 오로지 return문에 의해서만 생성된다.) 만약에 입력값으로 '바보'라는 값이 들어오면 문자열을 출력하지 않고 함수를 즉시 빠져나간다. 위의 함수는 매우 쓸모 없는 함수이지만 이 return문으로 함수를 빠져나가는 방법은 실제 프로그래밍에서 자주 쓰인다.

입력값에 초기치 설정하기

이젠 좀더 다른 함수의 인수 전달 방법에 대해서 알아보자. 인수에 초기치를 미리 설정해 주는 경우를 보자.

```
def say_myself(name, old, sex=1):
   print "나의 이름은 %s 입니다." % name
   print "나이는 %d살입니다." % old
```

```
    if sex:
        print "남자입니다."
    else:
        print "여자입니다."
```

(※ 알아두기 - 앞으로 나올 프로그램 소스는 '>>>'표시가 없으면 항상 에디터로 작성하기로 하자.)

위의 함수를 보면 입력 인수가 세 개임을 알수 있다. 그런데 이상한 것이 나왔다. sex=1처럼 입력 인수에 미리 값을 넣어준 것이다. 이것이 바로 함수의 인수 초기치를 설정하는 방법이다. 항상 변하는 것이 아니라 아주 가끔 변하는 것일 때 이렇게 함수의 초기치를 미리 설정해 주는 것은 매우 유용하다.

위의 함수는 다음처럼 사용할 수 있다.

```
say_myself("박응용", 27)
say_myself("박응용", 27, 1)
```

즉 인수값으로 " 박응용 ", 27처럼 두 개를 주게 되면 name에는 "박응용 " 이 old에는 27이 sex라는 변수에는 입력값을 주지 않았지만 초기값인 1이라는 값을 갖게 되는 것이다. 따라서 위에서 함수를 사용한 두가지 방법은 모두 동일한 결과를 출력하게 된다.

```
say_myself("박응선", 27, 0)
```

초기값설정된 부분을 0으로 바꾸었으므로 이번에는 sex변수에 0이라는 값이 대입되게 된다.

함수 입력값 초기치 설정시 주의사항
함수의 초기치를 설정할 때 주의해야 할 것이 하나 있다.

만약 위의 함수를 다음과 같이 만들면 어떻게 될까?

```
def say_myself(name, sex=1, old):
    print "나의 이름은 %s 입니다." % name
    print "나이는 %d살입니다." % old
    if sex:
        print "남자입니다."
```

```
    else:
        print "여자입니다."
```

위의 함수와 바뀐 부분은 초기치를 설정한 인수의 위치이다. 결론을 미리 말하자면 이것은 실행시에 에러가 난다.

그냥 얼핏 생각하기에 위의 함수를 호출하려면 다음과 같이 하면 될 것 같다.

```
say_myself( " 박응용 " , 27)
```

하지만 위와 같이 함수를 호출한다면 name변수에는 "박응용 " 이 들어가겠지만 파이썬 인터프리터는 sex에 27을 대입해야 할지 old에 27을 대입해야 할지 알 수 없을 것이다.

에러메시지를 보면 다음과 같다.

```
SyntaxError: non-default argument follows default argument
```

위의 에러메시지는 초기 치를 설정해 놓은 입력 인수 뒤에 초기 치를 설정해 놓지 않은 입력 인수는 사용할 수 없다는 말이다. 즉 입력인수로 (name, old, sex=1)은 되지만 (name, sex=1, old)는 안된다는 것이다. 결론은 초기화 시키고 싶은 입력 변수들은 항상 뒤쪽에 위치시키라는 것이다.

함수 내에서 선언된 변수의 효력 범위
함수안에서 사용할 변수의 이름을 함수 밖에서 사용한 이름과 동일하게 사용한다면 어떻게 될까? 이런 궁금증이 떠올랐던 독자라면 이곳에서 확실하게 알 수 있을 것이다.

아래의 예를 보자.

```
a = 1
def vartest(a):
    a = a +1

vartest(a)
print a
```

먼저 a라는 변수를 생성하고 1을 대입한다. 그리고 입력으로 들어온 값을 1만큼 더해주고 리턴값은 돌려주지 않는 vartest 함수에 입력 값으로 a를 주었다. 그 다음에 a의 값을 출력하게 하였다. 당연히 vartest에서 a를 1만큼 더했으니까 2가 출력되어야 할 것 같지만 프로그램 소스를 작성해서 실행시켜 보면 결과 값이 1이 나온다.

그 이유는 함수안에서 새로 만들어진 변수는 함수안에서만 쓰여지는 변수이기 때문이다. 즉 def vartest(a)처럼 하면 이때 입력 인수를 뜻하는 변수 a는 함수안에서만 쓰이는 변수이지 함수 밖의 변수 a가 아니라는 말이다.

위에서 변수이름을 a 로한 vartest함수는 다음처럼 변수이름을 b로한 vartest와 완전히 동일한 것이다.

```
def vartest(b):
   b = b + 1
```

즉 함수에서 쓰이는 변수는 함수 밖의 변수이름들과는 전혀 상관 없다는 말이다. 따라서 위에서 보았던 vartest의 a는 1이란 값을 입력으로 받아서 1만큼 더해주어 a가 2가 되지만 그것은 함수 내의 a를 뜻하는 것이지 함수 밖의 변수 a와는 전혀 다르다는 말이다.

다음의 예를 보면 더욱 정확하게 이해할 수 있을 것이다.

```
def vartest(a):
   a = a + 1

vartest(3)
print a
```

위의 프로그램 소스를 에디터로 작성해서 실행시키면 어떻게 될까? 에러가 날 것이라 생각한 독자는 모든 것을 이해한 독자이다. vartest(3)을 수행하면 vartest라는 함수내에서 a는 4가 되지만 함수를 끝내고 난뒤에 print a라는 문장은 에러가 나게 된다. 그 이유는 a라는 변수는 어디에도 찾아 볼 수가 없기 때문이다. 다시 말하지만 함수 내에서 쓰이는 변수는 함수내에서 쓰일 뿐이지 함수 밖에서 사용될 수 있는 것이 아니다. 이것에 대해서 이해하는 것은 매우 중요하다.

그렇다면 vartest라는 함수를 이용해서 함수 외부의 a를 1만큼 증가시킬 수 있는 방법은 없을까? 라는 의문이 떠오르게 된다. 두 가지 해결 방법이 있을 수 있다.

첫 번째 방법

```
a = 1
def vartest(a):
   a = a +1
   return a

a = vartest(a)
print a
```

첫 번째 방법은 return을 이용하는 방법이다. vartest함수는 입력으로 들어온 값을 1만큼 더한 값을 돌려준다. 따라서 a = vartest(a)처럼 하면 a가 vartest함수의 리턴값으로 바뀌게 되는 것이다.
여기서도 물론 vartest함수 안의 a변수는 함수 밖의 a와는 다른 것이다.

두 번째 방법

```
a = 1
def vartest():
   global a
   a = a+1

vartest()
print a
```

두 번째 방법은 global이라는 명령어를 이용하는 방법이다. 위의 예에서 보듯이 vartest안의 global a라는 문장은 함수 안에서 함수 밖의 a변수를 직접 사용하겠다는 말이다. 보통 프로그래밍을 할 때 global이라는 것은 쓰지 않는 것이 좋다. 왜냐하면 함수는 독립적으로 존재하는 것이 좋기 때문이다. 외부 변수에 종속적인 함수는 그다지 좋은 함수가 아니다. 독자는 가급적 이 global을 쓰는 방식을 피해야 할 것이다. 따라서 당연히 두 번째 방법보다는 첫 번째 방법이 좋다.

2) 입력과 출력

우리들이 사용하는 대부분의 완성된 프로그램은 사용자의 입력에 따라서 그에 맞는 출력값을 내어주게끔 되어 있다. 예를 들어보면 게시판에 우리가 작성한 글을 입력한다는 “확인” 버튼을 눌러야만(입력) 우리가 작성한 글이 게시판에 올라가는 것(출력)을 확인 할 수 있게 되는 것이다.

```
사용자 입력 ---> 처리(프로그램, 함수 등) ---> 출력
```

우리는 이미 함수 부분에서 입력과 출력이 어떤 의미인지에 대해서 알아보았다. 지금부터는 좀 더 다양하게 사용자의 입력을 받는 방법과 파일을 읽고 쓰는 방법 등에 대해서 알아보도록 하자.

사용자 입력
어떤 변수에 사용자로부터 입력받은 값을 대입하고 싶을 때는 어떻게 해야 할까?

input의 사용

```
>>> a = input()
'Life is too short, you need python'
>>> a
Life is too short, you need python
>>> a = input()
3
>>> a
3
>>>
```

위의 예는 input의 사용법을 보여준다. input은 사용자의 입력을 받는 함수로서 사용자는 그것이 문자열일 때는 (')나 (")으로 둘러싸서 입력해야 에러가 나지 않는다. (달리 말하면 실제 프로그램 소스에 변수값을 입력하듯이 입력해야 한다.) 따라서 숫자 3은 (')나 (")으로 둘러쌀 필요가 없다.

```
>>> a = input()
you need python
Traceback (most recent call last):
File "", line 1, in ?
File "", line 1
```

```
you need python
^
SyntaxError: invalid syntax
```

위처럼 you need python 을 변수에 입력하듯이 'you need python'처럼 ''로 감싸주지 않았을 때는 에러가 발생했다.

raw_input의 사용

```
>>> a = raw_input()
Life is too short, you need python
>>> a
Life is too short, you need python
>>>
```

위의 경우는 raw_input의 사용예이다. 문자열을 입력할 때 (')나 (")을 필요로 하지 않는다. raw_input은 입력되는 모든 것을 문자열로 취급한다.

raw_input과 input의 차이점은?
input은 변수에 값을 대입할 때와 마찬가지로 입력을 넣어주어야 한다. 즉, 문자열 값을 넣어주려면 따옴표로 감싸주어야 한다. 하지만 raw_input은 입력하는 모든 값을 문자열형으로 보기 때문에 입력에 주의해야 할 필요가 없다. 다만 정수값을 입력해도 문자열값으로 변환된다.

```
>>> a = raw_input()
3
>>> a
'3'
>>>
```

위의 예에서 보듯이 3이라는 정수값을 입력했지만 a에는 '3'이라는 문자열 값이 들어가게 된다. 프로그래밍을 할 때에는 보통 raw_input을 많이 쓴다. 그 이유는 변수에 대입되는 값이 무조건 문자열형태이기 때문에 입력값이 무엇인지에 대해서 신경을 쓰지 않아도 되기 때문이다. 만약 숫자값을 raw_input으로 받았더라도 이것은 다시 형변환에 의해서 숫자의 형태(정수나 실수)로 바꿀 수 있다. 형변환에 대한 함수들은 3장에서 알아 볼 것이다.

프롬프트 추가하기

```
input(prompt), raw_input(prompt)
```

사용자로부터 입력을 받을 때 " 숫자를 입력하세요. " 라던지 " 이름을 입력하세요 " 라는 질문을 포함하고 싶을 것이다. input이나 raw_input이라는 함수에 입력으로 위의 질문을 포함시킬 수가 있다. 다음의 예를 따라해 보자.

```
>>> number = raw_input("숫자를 입력하세요: " )
숫자를 입력하세요:
```

위와 같은 질문을 볼 수 있을 것이다.

print 자세히 알기
우리가 앞서서 계속 사용해 왔던 print가 하는 일은 자료형을 출력하는 것이다. 지금껏 알아보았던 print의 사용예는 다음과 같다.

```
>>> a = 123
>>> print a
123
>>> a = "Python"
>>> print a
Python
>>> a = [1, 2, 3]
>>> print a
[1, 2, 3]
```

이제 이것보다 조금 더 자세하게 print에 대해서 알아 보기로 하자.
따옴표(")로 둘러싸인 문자열은 + 연산과 동일

```
>>> print "life" "is" "too short" -------------------- ①
lifeistoo short
>>> print "life"+"is"+"too short" -------------------- ②
lifeistoo short
```

위에서 ①과 ②는 완전히 동일한 결과값을 보여준다. 즉, 따옴표로 둘러싸인 문자열을 연속해서 쓰면 '+'연산을 한 것과 마찬가지이다.

문자열 띄어쓰기는 콤마로

```
>>> print "life", "is", "too short"
life is too short
```

콤마(,)를 이용하면 문자열간에 띄어쓰기가 된다.

한 줄에 출력하기
앞서 보았던 구구단 프로그램에서 보았듯이 한 줄에 결과값을 계속 출력하려면 print문과 콤마(,)를 함께 사용해야 한다.

```
>>> for i in range(10):
...     print i,
...
0 1 2 3 4 5 6 7 8 9
```

3) 파일 읽고 쓰기

우리는 지금껏 입력으로는 사용자가 입력하게 하는 방식을 사용하였고, 출력으로는 모니터 화면에 결과값을 출력하는 방식의 프로그래밍만 해 왔다. 하지만 입출력의 방법이 꼭 그것만 있는 것은 아니다. 이곳에서는 파일을 이용한 입출력 방법에 대해서 알아 볼 것이다. 먼저 파일을 새롭게 만들어서 프로그램에 의해서 만들어진 결과 값을 파일에 한번 적어보고, 또 적은 내용을 읽어보는 프로그램을 만드는 것으로 시작해 보자.

파일 생성하기
다음을 에디터로 작성해서 실행해 보면 프로그램을 실행한 디렉토리에 새로운 파일이 하나 생성되는 것을 확인할 수 있다.

```
# file1.py

f = open("새파일.txt", 'w')
f.close()
```

파일을 생성하기 위해서 우리는 open이란 파이썬 내장 함수를 썼다.

open함수는 다음과 같이 입력으로 파일이름과 파일열기모드라는 것을 받고 리턴값으로 파일 객체를 돌려준다.

```
파일객체 = open(파일이름, 파일열기모드)
```

파일 객체와 파일 이름은 프로그래머가 마음대로 바꿀 수 있지만 파일 열기모드는 정해진 값을 사용해야 한다.

파일 열기 모드에는 다음과 같은 것들이 있다.

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가시킬 때 사용

파일을 쓰기 모드로 열게 되면 해당 파일이 존재할 경우에는 원래있던 내용이 모두 사라지게 되고, 해당 파일이 존재하지 않으면 새로운 파일이 생성된다. 위의 예에서는 없는 파일인 " 새파일.txt"를 쓰기 모드로 열었기 때문에 새로운 파일인 "새파일.txt"라는 이름의 파일이 현재디렉토리에 생성되는 것이다.

만약 " 새파일.txt"라는 파일을 C:\Python이란 디렉토리에 생성하고 싶다면 다음과 같이 해야 할 것이다.

```
f = open("C:\Python\새파일.txt", 'w')
```

위의 예에서보면 f.close()라는 것이 있는데 이것은 열린 파일 객체를 닫아주는 것이다. 사실 이 문장은 생략해도 된다. 왜냐하면 파이썬 프로그램이 종료할 때 열린 파일 객체를 자동으로 닫아주기 때문이다. 하지만 직접 열린 파일을 닫아주는 것이 좋다. 쓰기모드로 열었던 파일을 닫지 않고 다시 사용하려고 할 경우에는 에러가 나기 때문이다.

파일을 쓰기 모드로 열어서 출력값 적기
위의 예에서는 파일을 쓰기모드로 열기만 했지 정작 아무런 것도 쓰지 않았다. 이번에는 프로그램의 출력값을 파일에 적어 보도록 하자.

```
# file2.py
f = open("새파일.txt", 'w')

for i in range(1, 11):
   data = "%d 번째 줄입니다.\n" % i
   f.write(data)

f.close()
```

위의 프로그램을 다음의 프로그램과 비교해 보자.

```
for i in range(1, 11):
   data = "%d 번째 줄입니다.\n" % i
   print data
```

두 프로그램의 서로 다른 점은 data를 출력시키는 방법이다. 두 번째 방법은 우리가 지금껏 계속 사용해 왔던 모니터 화면에 출력하는 방법이고 첫 번째 방법은 모니터 화면대신에 파일에 결과값을 적는 방법이다. 차이점이 금방 눈에 들어 올 것이다. 두번째 방법의 print대신에 파일객체 f의

write라는 함수를 이용했을 뿐이다. 첫 번째 예제를 에디터로 작성해서 실행시킨 다음 그 프로그램을 실행시킨 디렉토리를 살펴보면 " 새파일.txt"라는 파일이 생성되었음을 확인 할 수 있을 것이다. 그 파일이 어떤 내용을 담고 있는 지 확인 해 보자. 아마 다음과 같을 것이다.

새파일.txt의 내용

```
1 번째 줄입니다.
2 번째 줄입니다.
3 번째 줄입니다.
4 번째 줄입니다.
5 번째 줄입니다.
6 번째 줄입니다.
7 번째 줄입니다.
8 번째 줄입니다.
9 번째 줄입니다.
10 번째 줄입니다.
```

즉 두 번째 방법을 사용했을 때 모니터 화면에 출력될 내용이 파일에 고스란히 들어가 있는 것을 알 수 있다.

파일을 읽는 여러가지 방법
파이썬에는 파일을 읽는 여러가지 방법이 있다. 그것들에 대해서 자세히 알아보도록 하자.

첫 번째 방법
첫 번째 방법은 readline()을 이용하는 방법이다. 다음의 예를 보자.

```
# file2.py
f = open("새파일.txt", 'r')
line = f.readline()
print line
f.close()
```

이전에 만들었던 "새파일.txt"를 수정하거나 지우지 않았다면 위의 프로그램을 실행시켰을 때 " 새파일.txt"의 가장 첫 번째 줄을 읽어서 화면에 출력해 줄 것이다. 우선 f.open("새파일.txt", 'r')로 파일을 읽기 모드로 열어서 열린 파일을 나타내는 파일객체 f를 돌려준다. f객체를 이용해서

파일의 한줄을 읽는 파일 객체 관련함수인 readline()을 이용해서 읽은 파일의 첫 번째 줄을 읽어서 line 변수에 대입하고 출력해 보았다.

만약 모든 라인을 읽어서 화면에 출력하고 싶다면 다음과 같은 프로그램을 작성해야 한다.

```
# file3.py
f = open("새파일.txt", 'r')

while 1:
   line = f.readline()
   if not line: break
   print line


f.close()
```

while 1이란 무한루프를 이용해서 f.readline()을 이용해서 파일을 계속해서 한 줄씩 읽어 들인다. 만약 더 이상 읽을 라인이 없으면 break를 수행한다. 여기서 알아두어야 할 것은 f.readline()은 파일의 내용을 한 줄씩 읽어 들인다는 사실이다.

위의 프로그램을 다음 프로그램과 비교해 보자.

```
while 1:
   data = raw_input()
   if not data: break
   print data
```

위의 예는 사용자의 입력을 받아서 그 내용을 출력하는 예이다. 위의 파일을 읽어서 출력하는 예제와 비교해 보자. 입력을 받는 방식만 틀리다는 것을 금방 알 수 있을 것이다. 두 번째 예는 키보드를 통한 입력방법이고 첫 번째 방법은 파일을 이용한 입력 방법이다.

두 번째 방법
두 번째 방법은 readlines()를 이용하는 방법이다. 다음의 예를 보기로 하자.

```
# file4.py
f = open("새파일.txt", 'r')
lines = f.readlines()
```

```
for line in lines:
   print line

f.close()
```

f.readlines()라는 것은 읽기 모드로 열린 파일의 모든 라인을 한꺼번에 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려준다. 따라서 위의 예에서 lines는 [“ 1 번째 줄입니다. ” , “ 2 번째 줄입니다. ” , , , “ 10 번째 줄입니다. ”]라는 리스트가 된다. f.readlines()에서 f.readline()과는 달리 's'가 하나 더 붙어 있음에 주목하도록 하자.

세 번째 방법
세 번째 방법은 read()를 이용하는 방법이다. 다음의 예를 보기로 하자.

```
# file5.py
f = open("새파일.txt", 'r')
data = f.read()

print data

f.close()
```

f.read()는 파일을 전부 읽은 문자열을 돌려준다. 따라서 위의 예의 data는 파일의 전체내용이다.

파일에 새로운 내용 추가하기
‘ w ’ 모드로 파일을 연 경우에는 이미 존재하는 파일을 열 경우 그 파일의 내용이 모두 사라지게 된다고 했는데 원래 있던 값을 유지하면서 단지 새로운 값만 추가하기를 원할 수도 있다. 이런 경우에는 파일을 추가 모드('a')로 열면 된다. 다음의 예를 보도록 하자.

```
# file6.py
f = open("새파일.txt",'a')

for i in range(11, 20):
   data = "%d번째 줄입니다.\n" % i
   f.write(data)
```

```
f.close()
```

“새파일.txt"라는 파일을 추가모드('a')로 열은 파일 객체를 만든 다음, 파일 객체의 관련함수인 write를 이용해서 결과값을 파일에 적는다. 여기서 추가 모드로 파일을 열었기 때문에 ”새파일.txt"라는 파일이 원래 가지고 있던 내용 바로 다음에 결과값을 적기 시작한다. “새파일.txt"라는 파일을 읽어서 확인해 보면 원래있던 파일 뒷부분에 새로운 부분이 추가 되었음을 확인 할 수 있을 것이다.

[참고] tell과 seek
파일객체 관련 함수로 ‘tell’과 ‘seek’도 빼놓을 수 없다. ‘tell’은 지금 현재 파일 포인터의 위치를 반환하고, seek은 지정하는 곳으로 포인터의 위치를 변화시킬 수 있는 파일객체 관련 함수이다. 파일 포인터란 파일의 현재 위치를 가리키는 말이다. 대화형 인터프리터를 실행시키고 다음을 따라해 보자.

```
>>> f = open("test.txt", 'w')
>>> f.write("this is one line\n")
>>> f.write("two line\n")
>>> f.write("three line\n")
>>> f.close()
```

우선 test.txt라는 파일을 쓰기 모드로 열어서 파일 객체를 생성한후 write함수를 이용하여 총 세 개의 줄을 test.txt파일에 입력하고 파일 객체를 닫는다. test.txt파일은 다음과 같을 것이다.

```
this is one line
two line
three line
```

다음의 예를 계속해서 따라해 보자.

```
>>> f = open("test.txt", 'r')
>>> f.tell()
0
```

처음에 파일을 읽기 모드로 열었고, 그 파일 포인터 값을 알기 위해서 tell을 호출하였다. 물론 파일의 맨 처음이기 때문에 0을 반환했다.

```
>>> f.readline()
'this is one line\n'
>>> f.tell()
18
```

다음에 한 줄을 읽는다. 그 다음의 파일 포인터는 그 줄의 바이트 수만큼 포인터가 증가한다.
따라서 다시 tell을 호출했을 때 18이 된 것이다.

```
>>> f.readline()
'two line\n'
>>> f.tell()
28
```

마찬가지로 다시 한 줄을 읽었더니 파일 포인터의 위치는 28이 되었다.

```
>>> f.seek(0)
>>> f.readline()
'this is one line\n'
>>>
```

파일 포인터의 값을 변화시키기 위해서 seek를 사용하였다. f.seek(0)는 파일 포인터의 위치를 0으로 하라는 것이다. 따라서 다음에 다시 한 줄을 읽었을 때는 그 파일의 맨 처음 줄을 읽게 되는 것이다.

sys모듈 입력
명령행 입력이전에 도스를 사용해 본 독자라면 다음과 같은 명령어를 사용해 본적이 있을 것이다.

```
C:\> type a.txt
```

위의 type명령어는 뒤에 파일이름을 인수로 받아서 그 내용을 출력해 주는 도스 명령어이다.

```
도스명령어 [인수1 인수2]
```

많은 도스 명령어가 위와 같은 방식을 따른다. 즉 명령행(도스창)에서 입력인수를 직접 주어서 프로그램을 실행시키는 방식이다. 이러한 기능을 파이썬 프로그램에도 적용시킬 수가 있다.

파이썬에서는 sys란 모듈을 이용하여 이것을 가능하게 한다. sys라는 모듈을 쓰려면 아래의 예에서 같이 import sys처럼 import라는 명령어를 사용해야 한다. 모듈을 사용하고 만드는 방법에 대해서는 뒤에서 자세히 다룰 것이다.

```
#sys1.py
import sys

args = sys.argv[1:]
for i in args:
   print i
```

위의 프로그램을 C:\Python이란 디렉토리에 저장하고 윈도우즈 도스창을 열고 다음과 같이 입력해 보자.

```
C:\Python>python sys1.py aaa bbb ccc
```

다음과 같은 결과 값을 볼 수 있을 것이다.

결과값:

```
aaa
bbb
ccc
```

sys모듈의 argv는 명령창에서 입력한 인수들의 리스트를 나타낸다. 즉, argv[0]는 파일 이름인 sys1.py가 되고 argv[1]부터는 뒤에 따라오는 인수들이 차례로 argv의 요소가 된다. 위의 예는 입력받은 인수들을 for문을 이용해 차례대로 하나씩 출력하는 예이다.

위의 예를 이용해서 간단한 스크립트를 하나 만들어 보자.

```
#sys2.py
import sys
args = sys.argv[1:]
for i in args:
   print i.upper(),
```

문자열 관련함수인 upper()를 이용한 명령행에서 입력된 소문자를 대문자로 바꾸어 주는 간단한 프로그램이다. 도스창에서 다음과 같이 입력해 보자. (주의: sys2.py 파일이 C:\Python이란 디렉토리내에 있어야만 한다.)

```
C:\Python> python sys2.py life is too short, you need python
```

결과값:

```
LIFE IS TOO SHORT, YOU NEED PYTHON
```

05. 파이썬 날개달기

이제 프로그래밍의 꽃이라고 할 수 있는 클래스에 대해서 알아보고 모듈, 예외처리 및 파이썬 라이브러리에 대해서 알아보자.
이번장의 내용을 끝으로 여러분은 파이썬 프로그램을 작성하기 위해 배워야 할 대부분의 것들을 숙지했다고 보면 될 것이다.

1) 클래스

클래스(class)라는 것은 함수나 변수들을 모아놓은 집합체이다. 하지만 단순한 데이터 자료형이라고 하기엔 그 활용도가 무궁무진하다고 할 수 있다. 클래스를 어떻게 설계하고 그 관계를 어떻게 설정하는가에 의해서 복잡한 것을 단순하게 불분명한것을 명확하게 바꿀 수 있는 능력을 발휘하기도 한다.

다음은 파이썬 클래스의 가장 간단한 예이다.

```
class Simple:
   pass
```

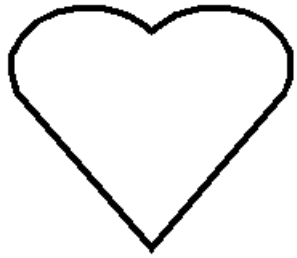
위의 클래스는 아무런 기능도 갖고 있지 않은 껍질 뿐인 클래스이다. 하지만 이렇게 껍질 뿐인 클래스도 인스턴스(instance)라는 것을 생성하는 기능은 가지고 있다. (인스턴스와 객체는 같은 말이다. 클래스에 의해서 생성된 객체를 인스턴스라고 부른다)

그렇다면 인스턴스는 무엇인가?
인스턴스는 클래스에 의해서 만들어진 객체로 한개의 클래스는 무수히 많은 인스턴스를 만들어 낼 수가 있다. 위에서 만든 클래스를 이용해서 인스턴스를 만드는 방법은 다음과 같다.

```
a = Simple()
```

바로 Simple()의 결과값을 돌려 받은 a가 인스턴스이다. 마치 함수를 사용해서 그 결과 값을 돌려 받는 모습과 비슷하다.

조금 쉽게 이해해 보기
어린시절 뽑기를 해 본적이 있다면 아래 그림과 비슷한 모양의 뽑기 틀을 본적이 있을 것이다. 뽑기 아저씨가 뽑기를 불에 달군 후 평평한 바닥에 "탁"하고 소리나게 떨어뜨려 넙적하고 동그랗게 만든후에 아래와 비슷한 틀로 모양을 찍어준다. 찍어준 모양대로 모양을 만들어 오면 아저씨는 뽑기 한개를 더 해 준다.



이곳에서 설명할 클래스라는 것이 마치 위 뽑기의 틀(별모양, 하트모양)과 비슷하다. 별 모양의 틀(클래스)로 찍으면 별모양의 뽑기(인스턴스)가 생성되고 하트 모양의 틀(클래스)로 찍으면 하트모양의 뽑기(인스턴스)가 나오는 것이다.

클래스란 똑같은 무엇인가를 계속해서 만들어 낼 수 있는 설계도면 같은 것이고(뽑기 틀), 인스턴스란 클래스에 의해서 만들어진 피조물(별 또는 하트가 찍혀진 뽑기)을 뜻하는 것이다.

이야기 형식으로 클래스 기초쌓기
이곳에서는 클래스의 개념을 아직 잡지 못한 독자들을 위한 곳이다. 매우 쉽게 설명하려고 노력하였고 최대한 클래스의 핵심 개념에 근접할 수 있도록 배려를 하였다. 클래스에 대해서 잘 아는 독자라도 재밌게 한 번 읽어 보기를 바란다.

클래스 변수
좀 더 이해를 쉽게 하기 위해서 다음의 클래스를 보자.

```
>>> class Service:
. . .     secret = "영구는 배꼽이 두개다. “
```

위의 클래스의 이름은 Service이다. 우리는 위의 Service 클래스를 어떤 유용한 정보를 제공해 주는 한 인터넷 업체라고 해 두자. 이 인터넷 업체는 가입한 고객에게만 유용한 정보를 제공하려 한다. 자 그렇다면 가입을 해야만 이 인터넷 업체의 유용한 정보를 얻을 수 있을 것이다.

가입을 하는 방법은 다음과 같다.

```
>>> pey = Service()
```

위 처럼 하면 pey라는 아이디로 인터넷 서비스 업체인 Service클래스를 이용할 수가 있게 된다. 위의 Service 클래스는 마음이 좋아서 돈도 필요 없고 비밀 번호도 필요 없다고 한다.

자 이제 pey라는 아이디로 위 서비스 업체가 제공하는 정보를 얻어내 보자.

```
>>> pey.secret
"영구는 배꼽이 두 개다 "
```

아이디 이름에다가 서비스 업체가 제공하는 secret라는 변수를 '.'(도트 연산자)를 이용해서 호출하였더니 실로 어마어마한 정보를 얻을 수 있었다.

클래스 함수

하지만 이 Service라는 이름의 인터넷 서비스 제공업체는 신생 벤쳐 기업이라서 위처럼 단 하나만의 정보만을 제공하고 있다고 한다. 하지만 제공해 주는 정보의 양이 너무 적은 듯하여 가입한 사람들을 대상으로 설문조사를 하였더니 모두들 더하기를 잘 못한다고 대답을 했다. 그래서 이 서비스 업체는 더하기를 해주는 서비스를 제공하기로 마음을 먹었다.

두 수를 더하는 서비스를 제공해 주기 위해서 이 서비스 업체는 다음과 같이 클래스를 업그레이드하였다.

```
>>> class Service:
. . .     secret = "영구는 배꼽이 두 개다"
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s + %s = %s 입니다." % (a, b, result)
. . .
>>>
```

이제 서비스에 가입한 모든 사람들이 더하기 서비스를 제공받게 되었다.
서비스를 사용하는 방법은 다음과 같았다.

먼저 서비스 업체에 가입을 해서 아이디를 받는다.

```
>>> pey = Service()
```

다음에 더하기 서비스를 이용한다.

```
>>> pey.sum(1,1)
1 + 1 = 2 입니다.
```

그렇다면 이번에는 서비스 업체의 입장에서 생각해 보도록 하자. 서비스 업체는 오직 가입한 사람들에게만 서비스를 제공하고 싶어한다. 이를 위해서 그들은 더하기 제공하는 서비스에 트릭을 가했다. 위에서 보았던 더하기 해주는 함수를 다시 보면 다음과 같다.

```
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s + %s = %s 입니다." % (a, b, result)
```

누군가 이 서비스 업체의 더하기 서비스를 쓰고 싶다고 요청했을 때 이 사람이 가입을 한 사람인지 안한 사람인지 가리기 위해서 위처럼 sum이라는 함수의 첫 번째 입력 값으로 self라는 것을 집어넣었다.

누군가 다음처럼 더하기 서비스를 이용하려 한다고 생각을 해 보자.

```
>>> pey = Service()
>>> pey.sum(1, 1)
```

이렇게 하면 pey라는 아이디를 가진 사람이 이 서비스 업체의 sum이라는 서비스를 이용하겠다고 요청을 한다는 말이다. 위와 같이 했을 때 Service라는 인터넷 제공 업체의 더하기 함수(sum)는 다음처럼 생각한다.

" 어 누군가 더하기 서비스를 해 달라고 하네. 그럼 먼저 서비스를 해 주기 전에 이 사람이 가입을 한 사람인지 아닌지 판단해야 겠군. 자 그럼 첫 번째 입력 값으로 뭐가 들어오나 보자. 음... pey라는 아이디를 가진 사람이군. 음, 가입한 사람이군. 서비스를 제공해 주자 "

위에서 보듯이 서비스 업체는 sum함수의 첫 번째 입력 값을 통해서 가입 여부를 판단했다. 다시 sum 함수를 보자.

```
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s + %s = %s 입니다." % (a, b, result)
```

위의 sum함수는 첫 번째 입력 값으로 self라는 것을 받고 두 번째 세 번째로 더할 숫자를 받는 함수이다. 위의 sum함수는 입력으로 받는 입력 인수의 갯수가 3개이다.

따라서 pey라는 아이디를 가진 사람은 다음처럼 sum함수를 사용해야 할 것이다.

```
pey.sum(pey, 1, 1)
```

sum함수는 첫 번째 입력값을 가지고 가입한 사람인지 아닌지를 판단할 수 있다고 했었다. 따라서 첫 번째 입력 인수로 pey라는 아이디를 주어야지 sum함수는 이 사람이 pey라는 아이디를 가지고 있는 사람임을 알고 서비스를 제공해 줄 것이다. 하지만 위의 sum함수를 호출하는 방법을 보면 pey라는 것이 중복해서 사용되었다.

다음과 같은 문장만으로 sum함수는 이 사람이 pey라는 아이디를 가지고 있음을 알 수 있을 것이다.

```
>>> pey.sum(1, 1)
```

그래서 pey.sum(pey, 1, 1)이 아닌 pey.sum(1, 1)이라는 방식을 채택하게 된 것이다. 앞의 방법보다는 뒤의 방법이 쓰기도 쉽고 보기도 쉽지 않은가? pey.sum(1, 1)이라는 호출이 발생하면 self는 호출 시 이용했던 인스턴스(즉, pey라는 아이디)로 바뀌게 된다.

> **[참고]** self는 사실 다른 언어에서는 찾을 수 없는 파이썬만의 독특한 변수이다. 굳이 왜 'self'가 필요했는지는 파이썬 언어 그 자체를 살펴보아야 한다. 파이썬 언어 개발자가 아닌 우리는 그저 클래스내의 함수의 첫번째 인자는
>
> "무조건 self로 사용을 해야 인스턴스의 함수로 사용할 수 있다."
>
> 라고만 알아두자.

self제대로 알기

시간이 흘러 더하기 서비스를 계속 제공받던 사람들이 똑똑해 졌다고 한다. 그들은 매우 자신감에

넘치게 되어서 매우 잘난 척을 하기에 이르렀고 서비스 업체에 뭔가 색다른 서비스를 제공해 줄 것을 요구하기에 이르렀다. 그 요구는 황당하게도 더하기 서비스를 제공할 때 " 홍길동님 1 + 2 = 3 입니다. " 처럼 " 홍길동 " 이라는 자신의 이름을 넣어달라는 것이었다.

인터넷 서비스 업체인 Service는 가입자들의 요구가 참 우스웠지만 그래도 자신의 서비스를 이용해 주는 것에 고마운 마음으로 그러한 서비스를 제공해 주기로 마음을 먹었다.

그래서 다음과 같이 또 Service 클래스를 업그레이드 하게 되었다. 이름을 입력받아서 sum함수를 제공할 때 앞부분에 그 이름을 넣어 주기로 했다.

```
>>> class Service:
. . .     secret = "영구는 배꼽이 두 개다"
. . .     def setname(self, name):
. . .         self.name = name
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s님 %s + %s = %s입니다." % (self.name, a, b, result)
. . .
>>>
```

그리고 바뀐 점과 사용법에 대해서 가입한 사람들에게 일러주었다. 그래서 사람들은 다음처럼 위의 서비스를 이용할 수 있었다.

먼저 서비스에 가입을 해서 pey라는 아이디를 얻는다.

```
>>> pey = Service()
```

다음에 pey라는 아이디를 가진 사람의 이름이 " 홍길동 " 임을 서비스에 알려준다.

```
>>> pey.setname( " 홍길동 " )
```

다음에 더하기 서비스를 이용한다.

```
>>> pey.sum(1, 1)
홍길동님 1 + 1 = 2 입니다.
```

이제 서비스를 제공해 주는 서비스 업체의 입장에서 다시 한번 생각해 보도록 하자. 우선 이름을 입력 받는 함수 setname을 보자.

```
. . .     def setname(self, name):
. . .         self.name = name
```

아래처럼 pey라는 아이디를 부여 받은 사람이

```
>>> pey = Service()
```

이름을 설정하겠다는 요구를 다음과 같이 하였다.

```
>>> pey.setname("홍길동 " )
```

위와 같은 상황이 발생 했을 때 서비스 제공 업체의 setname함수는 다음과 같이 생각한다.

> " pey라는 아이디를 가진 사람이 자신의 이름을 '홍길동'으로 설정하려고 하는 구나. 그렇다면 앞으로 pey라는 아이디로 접근하면 이 사람의 이름이 '홍길동'이라는 것을 잊지 말아야 겠다. "

위와 같은 것을 가능하게 해 주는 것이 바로 self이다.

일단 pey라는 아이디를 가진 사람이 "홍길동 " 이라는 이름을 setname함수에 입력으로 주면 다음의 문장이 수행된다.

```
self.name = name
```

self는 첫 번째 입력 값으로 pey라는 아이디를 받게 되므로 다음과 같이 바뀔 것이다.

```
pey.name = name
```

name은 두 번째로 입력받은 "홍길동 " 이라는 값이므로 위의 문장은 다시 다음과 같이 바뀔 것이다.

```
pey.name = "홍길동 "
```

위의 말의 의미는 pey라는 아이디를 가진 사람의 이름은 이제 항상 " 홍길동 " 이다 라는 말이다. 이제 아이디 pey에 그 사람의 이름을 부여하는 과정이 끝이 났다.

다음으로 " 홍길동님 1 + 1 = 2 입니다. " 라는 서비스를 가능하게 해 주기 위한 더하기 함수를 보도록 하자.

```
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s님 %s + %s = %s입니다." % (self.name, a, b, result)
```

" 1 + 1 = 2 입니다. " 라는 서비스만을 제공했던 이전의 sum함수와 비교해 보면 다른 점은 단 하나, self.name이라는 것을 출력 문자열에 삽입한 것 뿐이다. 그렇다면 self.name은 무엇일까? 설명하기 전에 sum함수를 이용하기 까지의 과정을 먼저 보자.

pey라는 이름의 아이디를 가진 사람이 자신의 이름을 "홍길동 " 이라고 설정한 다음에 sum함수를 다음과 같이 쓰려고 요청했다고 해 보자.

```
>>> pey = Service()
>>> pey.setname( " 홍길동 " )
>>> pey.sum(1, 1)
```

이 때 서비스 업체의 sum함수는 다음과 같이 생각한다.

> " pey라는 아이디를 가진 사람이 더하기 서비스를 이용하려 하는군. 가입한 사람이 맞군. 아, 그리고 이 사람의 이름은 어디 보자 '홍길동'이군, 이름을 앞에 넣어 준 다음 결과 값을 돌려주도록 하자. "

여기서 우리가 알아야 할 사항은 " sum함수가 어떻게 pey라는 아이디를 가진 사람의 이름을 알아내게 되었을까? " 이다. 먼저 setname함수에 의해서 " 홍길동 " 이라는 이름을 설정해 주었기 때문에 setname함수에 의해서 pey.name이란 것이 " 홍길동 " 이라는 값을 갖게 된다는 사실을 이미 알아보았다. 따라서 sum함수에서도 self.name은 pey.name으로 치환되기 때문에 sum함수는 pey라는 아이디를 가진 사람의 이름을 알아채게 되는 것이다.

__init__ 이란 무엇인가?
자, 이제 또 시간이 흘러 가입자 수가 많아지게 되었다. 더하기 해 주는 서비스는 너무 친절하게 서비스를 제공해 주기 때문이었다. 그런데 가끔 이런 문제가 발생한다고 항의 전화가 빗발치듯 들어온다. 그 사람들의 말을 들어보면 다음과 같은 것이다.

```
>>> babo = Service()
>>> babo.sum(1, 1)
```

위와 같이 하면 babo.setname(" 나바보 ")와 같은 과정이 빠졌기 때문에 에러가 나는 것이라고 골백번 얘기를 하지만 항상 이런 실수를 하는 사람들로부터 항의 전화가 와서 서비스 업체에서는 여간 귀찮은 게 아니었다. 그래서 다음과 같은 아이디어를 떠올렸다. 지금까지는 사람들이 서비스 가입 시 바로 아이디를 주는 방식이었는데 아이디를 줄 때 그 사람의 이름을 입력받아야만 아이디를 주는 방식으로 바꾸면 babo.setname("나바보 ")와 같은 과정을 생략할 수 있을 거란 생각이었다.

위와 같이 해주기 위한 방법을 찾던 중 서비스 업체의 실력자 한사람이 __init__이란 함수를 이용하자고 제의를 한다. 그 방법은 다음과 같았다.

```
>>> class Service:
. . .     secret = "영구는 배꼽이 두 개다"
. . .     def __init__(self, name):
. . .         self.name = name
. . .     def sum(self, a, b):
. . .         result = a + b
. . .         print "%s님 %s + %s = %s입니다." % (self.name, a, b, result)
. . .
>>>
```

위의 Service클래스를 이전의 클래스와 비교해 보면 바뀐 부분은 딱 한가지이다. 바로 setname함수의 이름인 setname이 __init__으로 바뀐 것이다. 클래스에서 이 __init__이란 함수는 특별한 의미를 갖는다. 의미는 다음과 같다.

" 인스턴스를 만들 때 항상 실행된다. " 즉, 아이디를 부여받을 때 항상 실행된다는 말이다.

따라서 이제는 위의 서비스에 가입을 하기 위해서 다음처럼 해야 한다.

```
>>> pey = Service( " 홍길동 " )
```

이전에는 pey = Service()만을 하면 되었지만 이제는 __init__함수 때문에 pey = Service(“ 홍길동 ”)처럼 아이디를 부여받을 때 이름 또한 써 주어야 한다.

이전에 했던 방식은 다음과 같다.

```
>>> pey = Service()
>>> pey.setname( “ 홍길동 ” )
>>> pey.sum(1, 1)
```

이것이 __init__ 함수를 이용하면 다음처럼 간략하게 쓸 수 있게 된다.

```
>>> pey = Service( “ 홍길동 ” )
>>> pey.sum(1, 1)
```

따라서 빗발치던 항의 전화도 멈추게 되고 세 번 입력하던 것을 두 번만 입력하면 되니 모두들 기뻐하게 되었다고 한다.

이상과 같이 인스턴스와 self의 의미를 알기 위해서 이야기 형식으로 클래스에 대해서 알아보았다. 위의 내용은 클래스에 대한 정확한 설명이 아니지만 초보자가 인스턴스와 self의 의미, 그리고 __init__함수의 의미를 보다 쉽게 접근할 수 있었을 것이다. 위에서 알아본 pey = Service()로 해서 생성된 pey를 아이디라고 하였는데 이것이 바로 인스턴스라고 불리우는 것임을 잊지 말자.

이제 위에서 알아보았던 기초적인 사항을 바탕으로 클래스에 대해서 자세하게 알아보기로 하자.

클래스 자세히 알기

클래스란 함수나 변수 등을 포함하고 있는 집합체이다. 클래스의 함수는 일반적인 함수와는 다르게 매우 다양한 용도로 쓰일 수도 있다. 클래스는 프로그래머가 설계를 하는 것이다. 마치 함수를 만드는 것과 마찬가지로 프로그래머는 클래스를 만든다. 클래스를 이용하면 전역 변수를 쓸 필요가 없어서 좋고 클래스에 의해서 만들어지는 인스턴스(Instance)라는 것을 중심으로 프로그램을 작성할 수 있기 때문에 객체중심의 독립적인 프로그래밍을 할 수 있게 된다.

클래스란 인스턴스(Instance)를 만들어 내는 공장과도 같다. 이 인스턴스를 어떻게 사용할 수 있는지를 알려면 클래스의 구조를 보면 알 수 있다. 즉, 클래스는 해당 인스턴스의 청사진(설계도)라고 할 수 있다. 사실 지금껏 알아온 자료형, 제어문, 함수들만으로도 우리가 원하는 프로그램을 작성하는데는 문제가 없다. 하지만 클래스를 이용하면 보다 아름답게 프로그램을 만들 수 있게 된다.

이제부터 객체지향 프로그래밍의 가장 중심이 되는 클래스에 대해서 자세히 알아보기로 하자. 아마도 2장에서 가장 어려운 부분이 될 것이다. 잘 이해가 안되더라도 낙심하지는 말자. 파이썬에 익숙해지다 보면 반드시 쉽게 이해가 될 것이다.

여러 가지 클래스를 만들어 보면서 클래스에 대해서 자세히 알아보도록 하자.

클래스의 구조
클래스는 다음과 같은 모습이다.

```
class 클래스이름[(상속 클래스명)]:
   〈클래스 변수 1〉
   〈클래스 변수 2〉
   ...
   def 클래스함수1(self[, 인수1, 인수2,,,]):
       〈수행할 문장 1〉
       〈수행할 문장 2〉
       ...
   def 클래스함수2(self[, 인수1, 인수2,,,]):
       〈수행할 문장1〉
       〈수행할 문장2〉
       ...
   ...
```

위에서 보듯이 class라는 명칭은 클래스를 만들 때 쓰이는 예약어이고 그 바로 뒤에는 클래스이름을 써 주어야 한다. 마치 함수에서와 같은 방식이다. 클래스 이름 뒤에 상속할 클래스가 있다면 상속할 클래스 이름을 쓴다. 클래스 내부에는 클래스 변수가 있고 클래스 함수들이 있다.

클래스가 무엇인지 감이 안 오더라도 걱정하지 말고 차근차근 다음의 예를보며 이해해 보자.

사칙연산 하는 클래스 만들기
사칙연산을 쉽게 해주는 클래스를 만들어 볼 것이다. 이 것을 알아보는 과정을 통해서 독자는 클래스를 만드는 방법에 대해서 알게 될 것이다. 우선 사칙연산이란 것은 더하기, 빼기, 나누기, 곱하기를 말한다. 따라서 우리가 만들 클래스는 이러한 기능을 가능하게 만들 것이다.

FourCal이란 사칙연산을 가능하게 하는 클래스가 다음처럼 동작한다고 가정하자. 먼저 a = FourCal()처럼해서 a라는 인스턴스를 만든다.

```
>>> a = FourCal()
```

다음에 a.setdata(4, 2)처럼해서 4와 2라는 숫자를 a라는 인스턴스에 지정해 주고

```
>>> a.setdata(4, 2)
```

a.sum()을 하면 두 수의 합한 결과(4 + 2)를 돌려주고

```
>>> print a.sum()
6
```

a.mul()하면 두수의 곱한 결과(4 * 2)를 돌려주고

```
>>> print a.mul()
```

8

a.sub()은 두 수를 뺀 결과(4 - 2)를 돌려주고

```
>>> print a.sub()
2
```

a.div()는 두 수를 나눈 결과(4 / 2)를 돌려준다.

```
>>> print a.div()
2
```

위와 같은 동작을 하게 만든는 FourCal이란 클래스를 만드는 것이 우리의 목표가 될 것이다.

> 참고 - 위와 같은 방식은 클래스를 먼저 만드는 것이 아니라 클래스에 의해서 만들어진 인스턴스를 중심으로 어떤 식으로 동작하게 할 것인지 미리 구상을 해 보는 방식이다. 그리고 생각했던 것을 하나씩 해결해 나가면서 클래스를 완성하게

> 된다. 위의 방법은 필자가 즐겨 사용하는 방법으로 클래스에 의해서 생성된 인스턴스가 어떻게 행동할지 미리 생각한 다음 실제적인 클래스를 만들어 가능 방식이다.

자! 그렇다면 위처럼 동작하는 클래스를 지금부터 만들어 보자. 제일 먼저 할 일은 a = FourCal()처럼 인스턴스를 만들 수 있게 해야 한다. 이것은 매우 간단하다. 다음을 따라해 보자.

```
>>> class FourCal:
. . .     pass
. . .
>>>
```

우선 대화형 인터프리터에서 위와 같이 pass란 문장만을 포함한 FourCal이란 클래스를 만든다. 위의 클래스는 아무런 변수나 함수도 포함하지 않지만 우리가 원하는 a = FourCal()로 인스턴스 a를 만들 수 있는 기능을 가지고 있다. 확인 해보자.

```
>>> a = FourCal()
>>> type(a)
<type 'instance'>
```

위에서 보듯이 a = FourCal()처럼해서 a라는 인스턴스를 먼저 만들고 그 다음에 type(a)로 a라는 변수가 어떤 형태인지 알아보았다. 역시 처럼 변수 a가 인스턴스임을 보여준다. (※ 참고 - type함수는 파이썬이 자체적으로 가지고 있는 내장함수로 객체의 종류를 보여준다.)

하지만 우리가 만들어낸 인스턴스 a는 아무런 기능도 가지고 있지 않다. 우리는 더하기, 나누기, 곱하기, 빼기 등의 기능을 가진 인스턴스를 만들어야 한다. 이러한 기능을 갖춘 인스턴스를 만들기 전에 우선적으로 해 주어야 할 일은 a라는 인스턴스에 더하기나 곱하기를 할 때 쓰일 두 개의 숫자를 먼저 부여해 주는 일이다.

다음과 같이 연산을 수행할 대상(4, 2)을 지정할 수 있게 만들어 보자.

```
>>> a.setdata(4, 2)
```

위의 사항이 가능하도록 하기 위해서는 다음과 같이 해야 한다.

```
>>> class FourCal:
. . .     def setdata(self, first, second):
. . .         self.first = first
. . .         self.second = second
. . .
>>>
```

이전에 만들었던 FourCal클래스에서 pass란 것은 당연히 없어져야 하고 그 대신에 setdata라는 함수를 만들었다. 클래스 내의 함수를 다른 말로 메써드라고도 한다. 어려운 용어이지만 익혀두도록 하자. (즉 setdata라는 함수는 FourCal클래스의 메써드이다.) 그렇다면 이제 setdata함수에 대해서 자세하게 알아보기로 하자.

보통 우리가 일반적인 함수를 만들 때는 다음과 같이 만든다.

```
def sum(a, b):
   return a+b
```

즉 입력 값이 있고 돌려주는 리턴 값이 있다. 클래스 내의 함수도 마찬가지이다. setdata함수를 다시 적어보면 아래와 같다.

```
def setdata(self, first, second):
   self.first = first
   self.second = second
```

입력 인수로 self, first, second이란 세 개의 입력 값을 받는다. 하지만 일반적인 함수와는 달리 클래스 내의 함수에서 첫 번째 입력 인수는 특별한 의미를 갖는다. 위에서 보면 바로 self가 특별한 의미를 갖는 변수이다.

다음의 예를 보면서 자세히 알아 보기로 하자.

```
>>> a = FourCal()
>>> a.setdata(4, 2)
```

위에서 보는 것처럼 a라는 인스턴스를 만든 다음에 a.setdata(4,2)처럼 하면 FourCal클래스 의 setdata함수가 호출되고 setdata함수의 첫 번째 인수에는 자동으로 a라는 인스턴스가 입력으로 들어가게 된다.

즉 setdata의 입력 인수는 self, first, second처럼 총 세 개이지만 a.setdata(4,2)처럼 두 개의 입력 값만 주어도 a라는 인스턴스가 setdata함수의 첫 번째 입력을 받는 변수인 self에 대입되게 되는 것이다.

```
self: 인스턴스 a, first: 4, second: 2
```

파이썬 클래스에서 가장 헷갈리는 부분이 바로 이 부분이다. setdata라는 함수는 입력 인수로 3개를 받는데 왜 a.setdata(4,2)처럼 두 개만을 주어도 되는가? 라는 부분이다. 이것에 대한 답변을 여러분은 이제 알았을 것이다.

그 다음으로 중요한 다음의 사항을 보자.
setdata 함수에는 두 개의 수행할 문장이 있다.

```
self.first = first
self.second = second
```

이것이 뜻하는 바에 대해서 알아보자. 입력인수로 받은 first는 4이고 second는 2라는 것은 이미 알아보았다.

그렇다면 다음과 같이 위의 문장은 다음과 같이 바뀔 것이다.

```
self.first = 4
self.second = 2
```

여기서 중요한 것은 바로 self이다. self는 a.setdata(4, 2)처럼 호출했을 때 자동으로 첫 번째 입력 인수로 들어오는 인스턴스 a라고 했다. 그렇다면 self.first의 의미는 무엇이겠는가? 당연히 a.first가 될 것이다. self.second는 당연히 a.second가 될 것이다.

따라서 위의 두 문장을 풀어서 쓰면 다음과 같이 될 것이다.

```
a.first = 4
a.second = 2
```

정말 이런지 확인 해 보도록 하자.

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> print a.first
4
>>> print a.second
2
```

지금껏 완성된 클래스를 다시 써보면 다음과 같다.

```
>>> class FourCal:
. . .     def setdata(self, first, second):
. . .         self.first = first
. . .         self.second = second
. . .
>>>
```

지금까지 한 내용이 바로 위의 4줄을 설명하기 위한 것이었다. 위에서 설명한 것들이 이해가 잘 되지 않는다면 다시 한번 읽어보는 것이 좋다. 지금껏 했던 것을 이해하지 못하면 다음에 할 것들도 이해하기가 어렵기 때문이다. 자! 그럼 이제 두 개의 숫자 값을 설정해 주었으니 두 개의 숫자를 더하는 기능을 위의 클래스에 추가해 보도록 하자.

우리는 다음과 같이 더하기를 할 수 있는 기능을 갖춘 클래스를 만들어야 한다.

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> print a.sum()
6
```

위의 것을 가능하게 하기 위해서 FourCal클래스를 다음과 같이 만들어야 한다.

```
>>> class FourCal:
. . .     def setdata(self, first, second):
. . .         self.first = first
. . .         self.second = second
. . .     def sum(self):
. . .         result = self.first + self.second
. . .         return result
```

```
. . .
>>>
```

추가된 것은 sum이란 함수이다. 이 함수만 따로 떼어내서 생각해 보도록 하자.

```
def sum(self):
   result = self.first + self.second
   return result
```

입력으로 받는 값은 self밖에 없고 돌려주는 값은 result이다. a.sum()처럼 하면 sum함수에 자동으로 인스턴스 a가 첫 번째 입력 인수로 들어가게 된다는 것을 명심하자. 따라서 sum함수의 첫 번째 입력 변수 self는 인스턴스 a가 된다.

그러면 이번에는 돌려주는 값을 보자.

```
result = self.first + self.second
```

위의 내용은 아래와 같이 해석 될 것이다.

```
result = a.first + a.second
```

위의 내용은 a.setdata(4, 2)에서

```
a.first = 4
a.second = 2
```

라고 이미 설정되었기 때문에 다시 다음과 같이 해석 될 것이다.

```
result = 4 + 2
```

따라서

```
>>> print a.sum()
```

위처럼 하면 6이란 값을 화면에 출력하게 한다.

여기까지 모두 이해한 독자라면 클래스에 대한 것 80% 이상을 알았다고 보아도 된다. 파이썬의 클래스는 그다지 어렵지 않다.
그렇다면 이번에는 곱하기, 빼기, 나누기 등을 할 수 있게 해보자.

```
>>> class FourCal:
. . .     def setdata(self, first, second):
. . .         self.first = first
. . .         self.second = second
. . .     def sum(self):
. . .         result = self.first + self.second
. . .         return result
. . .     def mul(self):
. . .         result = self.first * self.second
. . .         return result
. . .     def sub(self):
. . .         result = self.first - self.second
. . .         return result
. . .     def div(self):
. . .         result = self.first / self.second
. . .         return result
. . .
>>>
```

mul, sub, div모두 sum함수에서 했던 것과 마찬가지 방법이니 따로 설명을 하지는 않겠다. 정말로 모든 것이 제대로 동작하는지 확인해보자.

```
>>> a = FourCal()
>>> a.setdata(4, 2)
>>> a.sum()
6
>>> a.mul()
8
>>> a.sub()
2
```

```
>>> a.div()
2
```

우리가 목표로 했던 사칙연산을 해내는 클래스를 만들어 낸 것이다.

“ 박씨네 집 ” 클래스
이번에는 전혀 다른 내용의 클래스를 한 번 만들어 보자. 사칙 연산을 하는 클래스보다는 조금 재미있게 만들어 보자. “ 박씨네 집 ” 이라는 클래스를 만들어 보겠다. 먼저 클래스가 어떤 식으로 동작하게 할 지 생각해 보자.

클래스 이름은 HousePark으로 하기로 하자. pey라는 인스턴스를 다음처럼 만든다.

```
>>> pey = HousePark()
```

pey.lastname을 출력하면 “ 박씨네 집 ” 이기 때문에 "박 “ 이라는 성을 출력하게 만들기로 하자.

```
>>> print pey.lastname
박
```

이름을 설정하면 pey.fullname이 성을 포함한 값을 가지게 하도록 만들자.

```
>>> pey.setname("응용")
>>> print pey.fullname
박응용
```

travel이란 함수에 여행을 가고 싶은 곳을 입력으로 주면 다음과 같이 출력해 주는 travel함수도 만들어 보자.

```
>>> pey.travel("부산 “ )
박응용, 부산여행을 가다.
```

우선 여기까지만 만들어 보기로 하자. 어렵지 않을 것이다. 먼저 인스턴스만 단순히 생성할 수 있는 클래스는 다음처럼 만들 수 있다.

```
>>> class HousePark:
. . .     pass
. . .
>>>
```

이렇게 하면 pey = HousePark()처럼 해서 인스턴스를 만들어 낼 수 있게 된다. 이번에는 pey.lastname하면 "박"을 출력하게 하기 위해서 다음처럼 해보자.

```
>>> class HousePark:
. . .     lastname = "박"
. . .
>>>
```

lastname은 클래스 변수이다. 이 클래스 변수 lastname은 HousePark클래스에 의해서 생성되는 인스턴스 모두에 lastname은 "박 " 이라는 값을 갖게 하는 기능을 한다. 다음의 예를 보자.

```
>>> pey = HousePark()
>>> pes = HousePark()
>>> print pey.lastname
박
>>> print pes.lastname
박
```

위에서 보듯이 HousePark클래스에 의해서 생긴 인스턴스는 모두 lastname이 "박"으로 설정되는 것을 확인 할 수 있다.

참고 - 클래스 변수를 쓸 수 있는 또 하나의 방법

```
>>> print HousePark.lastname
박
```

다음에는 이름을 설정하고 print pey.fullname하면 성을 포함한 이름을 출력하게 하도록 만들어 보자.

```
>>> class HousePark:
. . .     lastname = "박"
. . .     def setname(self, name):
. . .         self.fullname = self.lastname + name
. . .
>>>
```

우선 이름을 설정하기 위해서 setname이란 함수를 이용하였다.

위의 함수는 다음처럼 사용될 것이다.

```
>>> pey = HousePark()
>>> pey.setname("응용 " )
```

위의 예에서 보듯이 setname함수에 "응용"이란 값을 인수로 주어서 결국 self.fullname에는 " 박 " + " 응용 " 이란 값이 대입되게 된다.

이 과정을 살펴 보면 다음과 같다.

```
self.fullname = self.lastname + name
```

우선 두 번째 입력 값 name 은 " 응용 " 이므로 다음과 같이 바뀔 것이다.

```
self.fullname = self.lastname + " 응용 "
```

다음에 self는 setname함수의 첫 번째 입력으로 들어오는 pey라는 인스턴스이기 때문에 다음과 같이 다시 바뀔 것이다.

```
pey.fullname = pey.lastname + "응용 "
```

pey.lastname은 클래스 변수로 항상 "박 " 이란 값을 갖기 때문에 다음과 같이 바뀔 것이다.

```
pey.fullname = "박 “  +  ” 응용 “
```

따라서 위와 같이 pey.setname(“ 응용 ”)을 한 다음에는 다음과 같은 결과를 볼 수 있을 것이다.

```
>>> print pey.fullname
박응용
```

이제 우리가 만들려고 했던 클래스의 기능 중 단 한가지만 남았다.
“ 박응용 ” 을 여행 보내게 하는 함수 travel을 HousePark 클래스에 구현해 보자.

```
>>> class HousePark:
. . .     lastname = "박"
. . .     def setname(self, name):
. . .         self.fullname = self.lastname + name
. . .     def travel(self, where):
. . .         print "%s, %s여행을 가다. “  % (self.fullname, where)
. . .
>>>
```

travel이란 함수를 추가시켰다. 입력 값으로 인스턴스와 where를 받는다. 그리고 해당 값들을 문자열 포맷팅 연산자를 이용하여 문자열에 삽입한 후 출력한다.

위의 클래스는 다음과 같이 사용되어 진다.

```
>>> pey = HousePark()
>>> pey.setname("응용 “ )
>>> pey.travel( “ 부산 ” )
박응용, 부산여행을 가다.
```

위의 과정을 travel함수의 입장에서 살펴보면 다음과 같다. 우선 travel함수의 입력변수인 self와 where은 다음과 같을 것이다.

```
self: pey
where :  “ 부산 “
```

따라서 self.fullname은 pey.fullname이 될 것이고 이 pey.fullname은 pey.setname(“ 응용 ”)에 의해서 만들어진 “ 박응용 ” 이 될 것이다. 따라서 pey.travel(“ 부산 ”)처럼 하게 되면 위의 예처럼 출력되게 되는 것이다.

초기값 설정하기
우리는 위에서 HousePark이라는 클래스를 이용해서 인스턴스를 만들었는데 이 인스턴스에 setname함수를 이용해서 이름을 설정해 주는 방식을 사용했었다.

하지만 위에서 만든 함수를 다음과 같이 실행해 보자.

```
>>> pey = HousePark()
>>> pey.travel( “ 부산 ” )
```

에러가 나게 된다. 에러의 이유는 travel함수에서 self.fullname이란 변수를 필요로 하는 데 self.fullname이란 것은 setname이란 함수에 의해서 생성되는 것이기 때문에 위의 경우 setname을 해주는 부분이 생략되어서 에러가 나게 되는 것이다. 클래스 설계시 이렇게 에러가 날 수 있는 상황을 만들면 좋은 클래스라고 보기 어렵다. 따라서 이런 에러가 나는 것을 방지하기 위해서 pey라는 인스턴스를 만드는 순간 setname함수가 동작하게 한다면 상당히 편리할 것이다. 이러한 생각으로 나오게 된 것이 바로 __init__ 이란 함수이다.

__init__ 함수, 초기치를 설정한다.
자 그렇다면 위의 클래스를 다음과 같이 바꾸어 보자.

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print "%s, %s여행을 가다." % (self.fullname, where)
...
>>>
```

setname함수의 이름이 __init__으로 바뀌기만 하였다.

이것이 어떤 차이를 불러오는지 알아보자. 다음처럼 해보자.

```
>>> pey = HousePark()
TypeError: __init__() takes exactly 2 arguments (1 given)
```

위와 같은 에러 메시지를 볼 수 있을 것이다. 에러 메시지의 원인을 보면 입력 인수로 두 개를 받아야 하는데 1개 만 받았다는 에러이다. 여기서 1개를 받았다는 것은 바로 pey라는 인스턴스이다. 자! 에러의 메시지를 보면 알 수 있다. pey = HousePark()이라고 하는 순간 __init__함수가 호출 되게 되는 것이다. 따라서 __init__(self, name)처럼 __init__함수는 두 개의 입력값을 필요로 하게 된다. 그렇다면 어떻게 인스턴스를 만들 때 __init__함수에 두 개의 입력값을 줄 수 있을까?

그것은 다음과 같다.

```
>>> pey = HousePark("응용 " )
```

마치 이전에 보았던 setname함수를 썼던 것과 마찬가지 방법이다. 다만 인스턴스를 생성하는 순간에 입력값으로 "응용 " 이란 입력값을 주는 점이 다르다. __init__ 함수를 이용하면 만들어지는 인스턴스에 초기값을 줄 수 있기 때문에 편리할 때가 많다.

다음과 같이 하면 에러 없이 잘 실행되는 것을 확인 할 수 있을 것이다.

```
>>> pey = HousePark("응용 " )
>>> print pey.travel( " 태국 " )
박응용, 태국여행을 가다.
```

__del__ 함수, 인스턴스가 사라질 때 호출된다.

이번에는 인스턴스가 사라질 때 특정한 행동을 취할 수 있게 하는 방법에 대해서 알아보도록 하자. 인스턴스가 사라진다함은 다음과 같은 경우를 말한다.

```
>>> pey = HousePark("응용 " )
>>> del pey
```

파이썬이 자체적으로 가지고 있는 내장 함수 del에 의해서 pey라는 인스턴스를 소멸시킬 수 있다. 또한 파이썬 프로그램이 다 수행되고 종료될 때 역시 만들어진 인스턴스들이 모두 사라지게 된다.

이렇게 프로그램 내에서 del함수에 의해서 사라지거나 프로그램이 종료될 때 특정한 행동을 취할 수 있도록 클래스를 재구성 해보자.

```
>>> class HousePark:
. . .     lastname = "박"
. . .     def __init__(self, name):
. . .         self.fullname = self.lastname + name
. . .     def travel(self, where):
. . .         print "%s, %s여행을 가다." % (self.fullname, where)
. . .     def __del__(self):
. . .         print "%s 죽네 “  % self.fullname
. . .
>>>
```

위와 같이 클래스를 다시 만든 다음에 다음과 같이 따라해 보자.

```
>>> pey = HousePark("응용 “ )
>>> del pey
박응용 죽네
```

즉 클래스 내에 사용되는 __del__ 이란 함수는 인스턴스가 사라질 때 호출되는 것임을 확인 할 수 있다.

상 속 (Inheritance)
상속이란 말은 “ 물려받다 ” 라는 뜻이다. “ 재산을 상속받다 ” 할 때의 상속과 같은 의미이다. 클래스에도 이런 개념을 쓸 수가 있다. 어떤 클래스를 만들 때 다른 클래스의 기능을 상속받을 수 있는 것이다. 우리는 “ 박씨네 집 ” 이라는 HousePark이라는 클래스를 만들었었는데 이번엔 “ 김씨네 집 ” 이라는 HouseKim 클래스를 만들어 보자. 상속의 개념을 이용하면 다음과 같이 간단하게 할 수 있다. HousePark클래스는 이미 만들어 놓았다고 가정을 한다. 다음의 예는 HouseKim이라는 클래스가 HousePark클래스를 상속하는 예제이다.

```
>>> class HouseKim(HousePark):
. . .      lastname = "김 “
. . .
>>>
```

성을 "김 “ 으로 고쳐 주었다. 그리고 아무런 것도 추가 시키지 않았다.

이 HouseKim이란 클래스는 위와 같이

```
class HouseKim(HousePark):
```

클래스명다음에 괄호안에 다른 클래스를 넣어주면 상속을 하게된다.

위 클래스는 마치 다음처럼 코딩한 것과 완전히 동일하게 동작한다.

```
>>> class HouseKim:
. . .     lastname = "김"
. . .     def __init__(self, name):
. . .         self.fullname = self.lastname + name
. . .     def travel(self, where):
. . .         print "%s, %s여행을 가다." % (self.fullname, where)
. . .     def __del__(self):
. . .         print "%s 죽네 “  % self.fullname
. . .
>>>
```

즉, HousePark클래스와 완전히 동일하게 행동하는 것이다.

다시 다음과 같이 HouseKim이라는 클래스를 만들자.

```
>>> class HouseKim(HousePark):
. . .      lastname = "김 “
. . .
>>>
```

그리고 아래와 같이 따라해 보자.

```
>>> julliet = HouseKim("줄리엣")
>>> julliet.travel("독도 “ )
김줄리엣, 독도여행을 가다.
```

HousePark클래스의 모든 기능을 상속받았음을 확인할 수 있다. 재미있지 않은가? 상속의 개념을 이용해서 독자는 “ 양씨네 집 ”, “ 이씨네 집 ” 등을 쉽게 만들 수 있을 것이다.

상속의 개념중 하나 더 알아야 할 것이 있는데 그것은 상속대상이 된 클래스의 함수중에서 그 행동을 달리 하고 싶을 때가 있을 것이다. 이럴땐 어떻게 해야 하는 것이까? 우리는 " 김씨네 집 " 이란 클래스가 " 박씨네 집 " 클래스를 상속받았는데 여기서 상속받은 travel함수를 다르게 설정하는 방법을 한번 보도록 하자.

```
>>> julliet = HouseKim("줄리엣 " )
>>> julliet.travel("독도 " , 3)
김줄리엣, 독도여행 3일 가네.
```

위처럼 행동할 수 있게끔 HouseKim 클래스를 만들어 보자.

```
>>> class HouseKim(HousePark):
. . .     lastname = "김"
. . .     def travel(self, where, day):
. . .         print "%s, %s여행 %d일 가네." % (self.fullname, where, day)
. . .
>>>
```

위처럼 travel함수를 다르게 설정하고 싶으면 동일한 이름의 travel함수를 HouseKim 클래스내에서 다시 설정해 주면 된다. 간단하다.

연산자 오버로딩
연산자 오버로딩이란 연산자(+, -, *, /, , ,)등을 인스턴스끼리 사용할 수 있게 하는 기법을 말한다. 쉽게 말해서 다음과 같은 것을 가능하게 만드는 것이다.

```
>>> pey = HousePark("응용")
>>> julliet = HouseKim("줄리엣 " )
>>> pey + julliet
박응용, 김줄리엣 결혼했네
```

즉, 인스턴스끼리 연산자 기호를 사용하는 방법을 말하는 것이다.

우선 이미 만들어 보았던 클래스들에 몇가지 사항을 추가하여 에디터를 이용해서 다음처럼 작성해 보자.

```
# house.py

class HousePark:
   lastname = "박"
   def __init__(self, name):
       self.fullname = self.lastname + name
   def travel(self, where):
       print "%s, %s여행을 가다." % (self.fullname, where)
   def love(self, other):
       print "%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname)
   def __add__(self, other):
       print "%s, %s 결혼했네" % (self.fullname, other.fullname)
   def __del__(self):
       print "%s 죽네" % self.fullname


class HouseKim(HousePark):
   lastname = "김"
   def travel(self, where, day):
       print "%s, %s여행 %d일 가네." % (self.fullname, where, day)

pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.love(julliet)
pey + julliet
```

위의 프로그램을 실행시키면 다음과 같은 결과를 보여 준다.

```
박응용, 김줄리엣 사랑에 빠졌네
박응용, 김줄리엣 결혼했네
박응용 죽네
김줄리엣 죽네
```

위의 프로그램의 실행 부분인 다음을 보자.

```
pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.love(julliet)
pey + julliet
```

먼저 pey = HousePark(“ 응용 ”)으로 pey라는 인스턴스를 만들고 julliet이라는 인스턴스 역시 하나 생성한다. 다음에 pey.love(julliet)이란 클래스 함수 love가 호출되었다.

love함수를 보면

```
    def love(self, other):
        print "%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname)
```

입력 인수로 두 개의 인스턴스를 받는 것을 알 수 있다. 따라서 pey.love(julliet)과 같이 호출하면 self에는 pey가 other에는 julliet이 들어가게 되는 것이다. 따라서 "박응용, 김줄리엣 사랑에 빠졌네 “ 라는 문장을 출력하게 된다.

자! 이제 다음과 같은 문장이다.

```
pey + julliet
```

더하기 표시기호인 '+'를 이용해서 인스턴스를 더하려고 시도한다. 이렇게 '+'연산자를 인스턴스에 사용하게 되면 클래스의 __add__ 라는 함수가 호출되게 된다.

HousePark에서 사용된 부분을 보면 다음과 같다.

```
    def __add__(self, other):
        print "%s, %s 결혼했네" % (self.fullname, other.fullname)
```

pey + julliet처럼 호출되면 __add__(self, other) 함수의 self는 pey가 되고 other는 julliet이 된다. 따라서 "박응용, 김줄리엣 결혼했네 “ 라는 문자열을 출력한다.

다음에는 프로그램이 종료하게 되는데 프로그램이 종료될 때 인스턴스가 사라지게 되므로 __del__ 함수가 호출되어 “ 박응용 죽네 ” , “ 김줄리엣 죽네 ” 라는 문자열이 출력되게 되는 것이다.

자, 이젠 지금껏 만들어본 클래스로 이야기를 만들어 보자. 스토리는 다음과 같다.

> “ 박응용은 부산에 놀러가고
> 김줄리엣도 우연히 3일 동안 부산에 놀러간다.
> 둘은 사랑에 빠져서 결혼하게 된다.

```
그러다가 바로 싸우고 이혼을 하게 된다. ”
```

참으로 슬픈 이야기이지만 연산자 오버로딩을 공부하기에 더없이 좋은 이야기이다. 다음처럼 동작시키면

```
pey = HousePark("응용 “ )
julliet = HouseKim("줄리엣 “ )
pey.travel("부산 “ )
julliet.travel("부산 “ , 3)
pey.love(julliet)
pey + julliet
pey.fight(julliet)
pey - julliet
```

결과값을 다음과 같이 나오게 만드는 클래스를 작성하는 것이 목표이다.

```
박응용 부산여행을 가다.
김줄리엣 부산여행 3일 가네.
박응용, 김줄리엣 사랑에 빠졌네
박응용, 김줄리엣 결혼했네
박응용, 김줄리엣 싸우네
박응용, 김줄리엣 이혼했네
박응용 죽네
김줄리엣 죽네
```

위에서 보면 fight라는 함수를 추가시켜야 하고 pey - julliet을 수행하기 위해서 __sub__ 이라는 '-'연산자가 쓰였을 때 호출되는 함수를 만들어 주어야 한다.

자, 최종적으로 만들어진 “ 박씨네 집 ” 클래스를 공개한다.

```
class HousePark:
   lastname = "박"
   def __init__(self, name):
       self.fullname = self.lastname + name
   def travel(self, where):
       print "%s, %s여행을 가다." % (self.fullname, where)
   def love(self, other):
```

```
        print "%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname)
    def fight(self, other):
        print "%s, %s 싸우네" % (self.fullname, other.fullname)
    def __add__(self, other):
        print "%s, %s 결혼했네" % (self.fullname, other.fullname)
    def __sub__(self, other):
        print "%s, %s 이혼했네" % (self.fullname, other.fullname)
    def __del__(self):
        print "%s 죽네" % self.fullname


class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print "%s, %s여행 %d일 가네." % (self.fullname, where, day)


pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.travel("부산")
julliet.travel("부산", 3)
pey.love(julliet)
pey + julliet
pey.fight(julliet)
pey - julliet
```

결과값은 예상한 대로 다음처럼 나올 것이다.

```
박응용, 부산여행을 가다.
김줄리엣, 부산여행 3일 가네.
박응용, 김줄리엣 사랑에 빠졌네
박응용, 김줄리엣 결혼했네
박응용, 김줄리엣 싸우네
박응용, 김줄리엣 이혼했네
박응용 죽네
김줄리엣 죽네
```

우리가 알아본 것 외에도 오버로딩 기법에 쓰이는 것들이 많이 있다. 다음의 표를 보고 어떤 것들을 사용할 수 있는지 알아보자.

[참고] 클래스내의 함수연산자 언제 호출되나?

함수	설명	예제 (X, Y는 인스턴스)
__init__	생성자(Constructor), 인스턴스가 만들어 질 때 호출	
__del__	소멸자(Destructor) 인스턴스가 사라질 때 호출	
__add__	연산자 "+"	X + Y
__or__	연산자 "|"	X | Y
__repr__	print	print X
__call__	함수호출 X()했을 때 호출	
__getattr__	자격부여	X.메소드
__getitem__	인덱싱	X[i]
__setitem__	인덱스 치환	X[key] = value
__getslice__	슬라이싱	X[i:j]
__cmp__	비교	X > Y

물론 위의 있는 것들이 전부는 아니다. 이외에도 여러 가지가 있다. 이것들이 필요할 때가 온다면 독자는 이미 상당한 수준의 파이썬 프로그래머일 것이다. 그 때는 직접 레퍼런스를 찾아가며 사용하자. 이제 이러한 연산자 오버로딩이 무엇이며 어떻게 사용해야 하는지 알게 되었을 것이다.

2) 모듈

모듈이란 함수나 변수들, 또는 클래스들을 모아놓은 파일이다. 다른 파이썬 프로그램에서 불러쓸 수 있게끔 만들어진 파이썬 파일을 모듈이라 부른다.

우리는 파이썬으로 프로그래밍을 할 때 굉장히 많은 모듈을 사용한다. 물론 이미 다른 사람들에 의해서 만들어진 파이썬 라이브러리들이 그 대표적인 것이 되겠지만 우리가 직접 만들어서 사용해야 할 경우도 생길 것이다. 여기서는 모듈을 어떻게 만들고 또 사용할 수 있는지에 대해서 자세하게 알아보기로 하자.

모듈 만들고 불러보기
우선 모듈에 대해서 자세히 살펴보기 전에 간단한 모듈을 한번 만들어 보기로 하자.

```
# mod1.py
def sum(a, b):
   return a + b
```

위와 같이 sum 함수만을 가지고 있는 파일 mod1py를 만들고 저장하여 보자. 그리고 MS도스창을 열고 mod1.py를 저장한 곳으로 이동한 다음에 대화형 인터프리터를 실행시키고 아래와 같이 따라해 보자. 꼭 mod1.py가 저장한 위치로 이동한 다음 다음을 실행해야 한다. 그래야만 대화형 인터프리터에서 mod1.py를 읽을 수 있다. import 는 현재 디렉토리에 있는 파일이나 파이썬 라이브러리가 저장되어진 디렉토리에 있는 파이썬 모듈만을 불러올 수 있다. 이 사항에 대해서는 잠시 후에 알아보도록 하자.

우리가 만든 mod1.py라는 파일을 파이썬에서 불러서 쓰려면 어떻게 할까? 다음은 import의 사용법이다.

```
import 모듈이름
```

여기서 모듈이름은 mod1.py에서 ".py"라는 확장자를 제거한 mod1만을 가리키는 것이다.

```
>>> import mod1
>>> print mod1.sum(3,4)
7
```

위처럼 mod1.py를 불러오기 위해서 import mod1과 같이 하였다. import mod1.py과 같이 사용하는 실수를 하지 않도록 주의 하자. import는 이미 만들어진 파이썬 모듈을 사용할 수 있게 해주는

것이다. mod1.py파일에 있는 sum함수를 이용하기 위해서는 위의 예에서와 같이 mod1.sum처럼 모듈이름 뒤에 '.'(도트 연산자)를 붙이고 함수이름을 써서 사용할 수 있다.

이번에는 mod1.py에 다음의 함수를 추가 시켜 보자.

```
def safe_sum(a, b):
   if type(a) != type(b):
       print "더할수 있는 것이 아닙니다."
       return
   else:
       result = sum(a, b)
   return result
```

위의 함수가 하는 역할은 서로 다른 타입의 객체끼리 더하는 것을 방지해 준다. 만약 서로 다른 형태의 객체가 입력으로 들어오면 " 더할 수 있는 값이 아닙니다 " 라는 메시지를 출력하고 return문만 단독으로 사용되어 None값을 돌려주게 된다.

이 함수를 mod1.py에 추가 시킨다음 다시 대화형 인터프리터를 열고 다음과 같이 따라하자.

```
>>> import mod1
>>> print mod1.safe_sum(3, 4)
7
```

import mod1으로 mod1.py파일을 불러온다. 다음에 mod1.safe_sum(3, 4)로 safe_num함수를 호출한다.

```
>>> print mod1.safe_sum(1, 'a')
더할 수 있는 값이 아닙니다.
None
>>>
```

만약 서로 다른 형태의 객체가 입력으로 들어오면 에러 메시지를 출력하고 단독으로 쓰인 return에 의해서 None이라는 것을 돌려주게 된다.

또한 sum함수도 다음처럼 바로 호출할 수 있다.

```
>>> print mod1.sum(10, 20)
30
```

때로는 mod1.sum, mod1.safe_sum 처럼 쓰기 보다는 그냥 sum, safe_sum 처럼 함수를 쓰고 싶은 사람도 있을 것이다. 이럴때는 " from 모듈이름 import 모듈함수 " 를 사용하면 된다.

```
from 모듈이름 import 모듈함수
```

다음과 같이 따라해 보자.

```
>>> from mod1 import sum
>>> sum(3, 4)
7
```

from ~ import ~를 이용하면 위에서 처럼 모듈이름을 붙이지 않고 바로 해당 모듈의 함수를 쓸 수 있다. 하지만 위와 같이 하면 mod1.py파일의 sum함수만을 사용할 수 있게 된다. 그렇다면 sum함수와 safe_sum함수를 둘다 사용하고 싶을 땐 어떻게 해야 할까? 두가지 방법이 있다.

```
from mod1 import sum, safe_sum
```

첫 번째 방법은 위에서 보는 것과 같이 from 모듈이름 import 모듈함수1, 모듈함수2 처럼 사용하는 방법이다. 콤마로 구분하여 필요한 함수를 불러올 수 있는 방법이다.

```
from mod1 import *
```

두 번째 방법은 위와같이 * 문자를 이용하는 방법이다. '*'가 모든 것을 뜻한다고 대부분 알고 있는데 파이썬에서도 마찬가지이다. 위의 from mod1 import *가 뜻하는 말은 mod1.py의 모든 함수를 불러서 쓰겠다는 말이다. mod1.py에는 함수가 2개 밖에 존재하지 않기 때문에 위의 두가지 방법은 동일하게 적용될 것이다.

if __name__ == "__main__": 의 의미
이번에는 mod1.py 파일에 다음과 같이 추가하여 보자.

```
# mod1.py
def sum(a, b):
   return a+b

def safe_sum(a, b):
   if type(a) != type(b):
       print "더할수 있는 것이 아닙니다."
       return
   else:
       result = sum(a, b)
   return result


print safe_sum('a', 1)
print safe_sum(1, 4)
print sum(10, 10.4)
```

위와 같은 mod1.py를 에디터로 작성해서 C:\Python이란 디렉토리에 저장을 했다면 위의 프로그램 파일을 다음처럼 실행할 수 있다.

```
C:\Python> python mod1.py
더할수 있는 것이 아닙니다.
None
5
20.4
```

결과값은 위처럼 나올 것이다. 하지만 문제는 이 mod1.py라는 파일을 import해서 쓰려고 할 때 생긴다.

도스창을 열고 다음을 따라해 보자.

```
C:\WINDOWS> cd \Python
C:\Python> python
Python 2.1 (#15, Apr 16 2001, 18:25:49) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license" for more information.
>>> import mod1
더할수 있는 것이 아닙니다.
None
5
20.4
```

위와 같은 결과를 볼 수 있을 것이다. 우리는 단지 mod1.py파일의 sum과 safe_sum함수만을 쓰려고 했는데 위처럼 import mod1을 하는 순간 mod1.py가 실행이 되어서 결과값을 출력한다. 이러한 것을 방지하기 위한 것이 있다.

mod1.py파일에서 마지막 부분을 다음과 같이 수정해 보자.

```
if __name__ == "__main__":
   print safe_sum('a', 1)
   print safe_sum(1, 4)
   print sum(10, 10.4)
```

이것이 뜻하는 의미는 C:\Python> python mod1.py 처럼 직접 이 파일을 실행시켰을 때는 __name__ == "__main__" 이 참이 되어 if문 다음 문장들이 수행되고 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 쓸때는 __name__ == "__main__"이 거짓이 되어 if문 아래문장들이 수행되지 않도록 한다는 뜻이다.

파이썬 모듈을 만든 다음 보통 그 모듈을 테스트하기 위해서 위와 같은 방법을 쓴다. 실제로 그런지 대화형 인터프리터를 열고 실행해 보자.

```
>>> import mod1
>>>
```

위와 같이 고친 다음에는 아무런 결과값을 출력하지 않는 것을 볼 수 있다.

클래스나 변수등을 포함한 모듈
위에서 알아본 모듈은 함수만을 포함하고 있지만 클래스나 변수등을 포함할 수도 있다. 다음의 프로그램을 작성해 보자.

```
# mod2.py
PI = 3.141592

class Math:
   def solv(self, r):
       return PI * (r ** 2)
```

```
def sum(a, b):
   return a+b

if __name__ == "__main__":
   print PI
   a = Math()
   print a.solv(2)
   print sum(PI , 4.4)
```

클래스와 함수, 변수등을 모두 포함하고 있는 파일이다. 이름을 mod2.py로 하고 C:\Python이란 디렉토리에 저장했다고 가정을 해 보자.

다음과 같이 실행할 수 있다.

```
C:\Python> python mod2.py
3.141592
12.566368
7.541592
```

이번에는 대화형 인터프리터를 열고 다음과 같이 따라해 보자.

```
C:\Python> python
Python 2.1 (#15, Apr 16 2001, 18:25:49) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license" for more information.
>>> import mod2
>>>
```

__name__ == "__main__"이 거짓이 되므로 아무런 값도 출력되지 않는다.

```
>>> print mod2.PI
3.141592
```

mod2.PI 처럼 mod2.py에 있는 PI라는 변수값을 사용할 수 있다.

```
>>> a = mod2.Math()
>>> print a.solv(2)
12.566368
```

위의 예는 mod2.py에 있는 클래스 Math를 쓰는 방법을 보여준다. 위에서 보듯이 모듈내에 있는 클래스를 이용하기 위해서는 '.'(도트연산자)를 이용하여 클래스이름 앞에 모듈이름을 먼저 써 주어야 한다.

```
>>> print mod2.sum(mod2.PI, 4.4)
7.541592
```

mod2.py에 있는 함수 sum역시 당연히 사용할 수 있다.

다른 프로그램 파일에서 만든 모듈 불러오기
만들어놓은 모듈 파일을 써 먹기 위해서 지금까지 대화형 인터프리터 만을 이용했는데 이번에는 새롭게 만들 파이썬 파일 내에서 이전에 만들어 놓았던 모듈을 불러서 쓰는 방법에 대해서 알아보기로 하자. 바로 이전에 만든 모듈인 mod2.py라는 파일을 새롭게 만들 파이썬 프로그램 파일에서 불러보도록 하자. 자! 그럼 에디터로 다음을 함께 작성해 보자.

```
# modtest.py
import mod2

result = mod2.sum(3, 4)
print result
```

위에서 보듯이 파일에서도 대화형 인터프리터에서 한 것과 마찬가지 방법으로 import mod2로 mod2 모듈을 불러와서 쓰면 된다. 여기서 중요한 것은 modtest.py라는 파일과 mod2.py라는 파일이 동일한 디렉토리에 있어야만 한다는 점이다.

모듈 불러오는 또다른 방법
우리는 지금껏 만든 모듈을 써먹기 위해서 도스창을 열고 모듈이 있는 디렉토리로 이동한 다음에나 쓸 수 있었다. 하지만 항상 이렇게 해야 하는 불편함을 해소할 수 있는 방법이 있다.

만약 독자가 만든 파이썬 모듈을 항상 C:\Python\Mymodules 라는 디렉토리에 저장했다면 그 디렉토리로 이동할 필요없이 모듈을 불러서 쓸 수 있는 방법에 대해서 알아보자.

우선 위에서 만든 mod2.py모듈을 C:\Python\Mymodules라는 디렉토리에 저장한 다음에 다음의 예를 따라해 보도록 하자.

먼저 sys 모듈을 불러보자.

```
>>> import sys
```

sys 모듈은 파이썬을 설치할 때 함께 오는 라이브러리 모듈이다. sys에 대한 얘기는 3장에서 또다시 다룰 것이다. 우리는 sys모듈을 이용해서 파이썬 라이브러리가 설치되어 있는 디렉토리를 확인 할 수 있다.

다음과 같이 해보자.

```
>>> sys.path
>>> sys.path
['', 'c:\\', 'c:\\python21\\dlls', 'c:\\python21\\lib', 'c:\\python21\\lib\\plat-win',
'c:\\python21\\lib\\lib-tk', 'c:\\python21']
```

sys.path는 파이썬 라이브러리들이 설치되어 있는 디렉토리들을 보여준다. 또한 파이썬 모듈이 위의 디렉토리에 들어있는 경우에는 해당 디렉토리로 이동할 필요 없이 바로 불러서 쓸 수가 있다.
그렇다면 sys.path에 C:\Python\Mymodues라는 디렉토리를 추가하면 아무데서나 불러쓸 수 있을까?

당연하다.

sys.path의 결과값이 리스트 였으므로 우리는 다음과 같이 할 수 있을 것이다.

```
>>> sys.path.append("C:\Python\Mymodules")
>>> sys.path
['', 'c:\\', 'c:\\python21\\dlls', 'c:\\python21\\lib', 'c:\\python21\\lib\\plat
-win', 'c:\\python21\\lib\\lib-tk', 'c:\\python21', 'C:\\Python\\Mymodules']
>>>
```

sys.path.append를 이용해서 C:\Python\Mymodules라는 디렉토리를 sys.path에 추가시키고 다시 sys.path를 보았더니 가장 마지막 요소에 C:\\Python\\Mymodules라고 추가 된 것을 확인 할 수 있었다.

그렇다면 실제로 모듈을 불러서 쓸 수 있는지 확인해보자.

```
>>> import mod2
>>> print mod2.sum(3,4)
7
```

이상 없이 불러서 쓸 수 있다. 이렇게 특정한 디렉토리에 있는 모듈을 불러서 쓰고 싶을 때 사용하는 것이 바로 sys.path.append(모듈디렉토리)의 방법이다.

reload

reload는 이미 불렀던(import한) 모듈에 변경 사항이 생겼을 때 다시 그 모듈을 불러서 새로운 사항을 적용시키는 것이다. 아마도 파이썬 초보자는 이 기능을 별로 사용할 일이 없을 것이다. 하지만 모듈을 바꾸어가며 대화형 인터프리터에서 테스트할 때 reload는 매우 유용하게 쓰일 것이다. 다음과 같이 대화형 인터프리터에서 따라해 보자.

```
>>> import mod2
>>> print mod2.PI
3.141592
```

대화형 인터프리터를 아직 닫지 말고 에디터로 mod2.py의 PI 부분을 다음 처럼 수정하자.

```
PI = 3.14
```

PI를 좀더 간단한 값(3.14)으로 바꾸었다. 다음에 대화형 인터프리터 모드로 돌아가서 다음과 같이 해보자.

```
>>> reload(mod2)
>>> print mod2.PI
3.14
```

바뀐 변수값으로 출력되었음을 확인할 수 있다. reload대신에 import를 다시 하여도 바뀐값이 적용되지 않을까? 라는 의심이 드는 독자들은 직접 한번 실행 해 보기를 바란다. reload를 하면 바뀐 값이 적용되지만 import는 이전의 값을 유지하고 있음을 알게 될 것이다. 하지만 위에서 mod2.py파일을 수정한 다음 대화형 인터프리터를 닫고 다시 대화형 인터프리터를 실행한 후 import mod2를 하면 변경된 사항이 적용되는 것은 당연한 일이다.

3) 예외처리

예외 처리(try, except)
프로그램을 만들다 보면 수없이 많은 에러가 난다. 물론 에러가 나는 이유는 프로그램이 오동작을 하지 않기 하기 위한 파이썬의 배려이다. 하지만 때때로 이러한 에러를 무시하고 싶을 때도 있고, 에러가 날 때 그에 맞는 적절한 처리를 하고 싶을 때가 생기게 된다. 이에 파이썬에는 try, except를 이용해서 에러를 처리할 수 있게 해준다. 에러 처리하는 방법에 대해서 알게 되면 매우 유연한 프로그래밍을 구사 할 수 있을 것이다.

에러는 어떤 때 일어나는가?
에러를 처리하는 방법을 알기 전에 어떤 상황에서 에러가 나는지 한번 보자. 오타를 쳤을 때 나는 구문 에러 같은 것이 아닌 실제 프로그램에서 잘 발생하는 에러를 보기로 하자. 먼저 없는 파일을 열려고 시도해 보자.

```
>>> f = open('나없는파일', 'r')
Traceback (most recent call last):
File "", line 1, in ?

IOError: [Errno 2] No such file or directory: '\xb3\xaa\xbe\xf8\xb4\xc2\xc6\xc4\xc0\xcf'
```

위의 예에서 보듯이 없는 파일을 열려고 시도하면 IOError라는 이름의 에러가 발생하게 된다. 위의 예에서 '나없는파일'이 한글이기 때문에 다음처럼 깨져 보이는 것이다. 이것은 신경쓰지 말도록 하자.

```
'\xb3\xaa\xbe\xf8\xb4\xc2\xc6\xc4\xc0\xcf'
```

이번에는 또 하나 자주 발생하는 에러로 0으로 어떤 다른 숫자를 나누는 경우를 생각해 보자.

```
>>> 4 / 0
Traceback (most recent call last):
File "", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>>
```

4를 0으로 나누려니까 ZeroDivisionError라는 이름의 에러가 발생한다.

마지막으로 한가지 에러만 더 들어 보자. 다음의 에러는 정말 빈번하게 일어난다.

```
>>> a = [1,2,3]
>>> a[4]
Traceback (most recent call last):
File "", line 1, in ?
IndexError: list index out of range
>>>
```

a는 [1, 2, 3]이란 리스트인데 a[4]는 a 리스트에서 구할 수 없는 값이기 때문에 IndexError가 나게 된다. 파이썬은 이런 에러가 나면 프로그램을 중단하고 에러메시지를 보여준다.

에러 처리하기
자, 이제 유연한 프로그래밍을 위한 에러처리의 기법에 대해서 살펴보자. 다음은 에러 처리를 위한 try, except문의 기본 구조이다.

```
try:
   ...
except [발생에러[, 에러메시지변수]]:
   ...
```

try문안의 수행할 문장들이 에러가 나지 않는다면 except문 다음의 문장들은 수행이 되지 않는다. 하지만 try문안의 문장들을 수행 중 에러가 발생하면 except문을 수행한다.

except문을 자세히 보자.

```
except [발생에러 [, 에러메시지변수]]:
```

위에서 보면 [발생에러 [, 에러메시지변수]]는 생략이 가능하다는 전형적인 표기법이다. 즉 다음처럼 try, except만 써도 되고
첫 번째

```
try:
   ...
except:
   ...
```

다음과 같이 발생에러만 포함한 except문을 써도 되고
두 번째

```
try
   ...
except 발생에러:
   ...
```

다음과 같이 발생에러와 에러메시지변수까지 포함한 except문을 써도 된다.
세 번째

```
try
   ...
except 발생에러, 에러메시지변수:
   ...
```

이중에서 한가지를 택해서 쓰게 되는데 첫 번째의 경우는 에러 종류에 상관없이 에러가 발생하기만 하면 except문 다음의 문장들을 수행한다는 말이고 두 번째 경우는 에러가 발생했을 때 except문에 미리 정해놓은 에러이름과 일치할 때만 except문 다음의 문장을 수행한다는 말이고 세 번째의 경우는 두 번째의 경우에다가 에러메시지를 담은 변수하나를 더 생성하게 하는 방법이다.

세 번째 방법의 예를 잠시 들어 보면 다음과 같다.

```
try:
   4 / 0
except ZeroDivisionError, e:
   print e
```

위처럼 4를 0으로 나누려고 하면 ZeroDivisioError가 발생하기 때문에 except문이 실행되고 위의 except문은 e라는 에러메시지를 담은 변수를 출력 시키기 때문에 결과값은 다음과 같을 것이다.

```
결과값: integer division or modulo by zero
```

에러처리하기 예제
자 이제 실제적인 에러 처리의 예를 들어보기로 하자. 어떤 프로그램을 만들었는데 만약 파일이 존재하면 읽기 모드로 열고 존재하지 않으면 쓰기 모드로 파일을 여는 프로그램을 만든다고 가정해 보자. 독자라면 어떻게 하겠는가?

필자는 다음과 같은 방식을 택할 것이다.

```
>>> try:
. . .     f = open("나없는파일.txt", 'r')
. . . except IOError:
. . .     print "쓰기모드로 파일을 엽니다."
. . .     f = open("나없는파일.txt", 'w')
. . .
쓰기모드로 파일을 엽니다.
```

위와 같이 하였을 때 try문을 우선 실행하게 되는데 try문의 f = open("나없는파일.txt " , 'r')처럼 없는 파일을 읽기 모드로 열려고 하면 IOError가 나게 된다. 바로 이 에러가 나는 순간에 except문의 에러이름과 똑같은지를 판단하게 된다. 만약 에러이름이 except문에서 정해놓은 에러이름과 일치할 때 except문 다음의 문장들을 수행하게 된다. 즉, 위의 예에서 보면 "쓰기모드로 파일을 엽니다 " 라는 문자열을 출력하고 " 나없는파일.txt " 라는 파일을 쓰기 모드로 열게 되는 것이다.

실제로 프로그램을 만들어 보면 위처럼 미리 에러를 예측하기가 쉽지 않다. 대부분의 에러처리는 실제 프로그램을 작성하고 실행될 때 발생하는 에러를 조사한 다음에 try, except를 이용해서 에러를 처리하게 된다.

에러 발생시키기(raise)
좀 이상하긴 하지만 에러를 일부러 발생시켜야 할 경우도 생기게 된다. 파이썬은 raise라는 명령어를 이용하여 에러를 강제로 발생시킨다. 예를 들어 Bird라는 클래스를 상속받는 자식 클래스는 반드시 fly라는 함수를 구현하게 만들고 싶은 경우(강제로 그렇게 하고 싶은경우)가 있을 수 있다.

```
class Bird:
   def fly(self):
       raise NotImplementedError
```

만약 위 Bird클래스를 상속받는 자식 클래스가 fly라는 함수를 구현하지 않은 상태에서 fly함수가 호출되면 반드시 위 에러가 발생하게 될 것이다.

Bird클래스의 fly함수를 구현한 자식클래스를 보자.

```
class Eagle(Bird):
   def fly(self):
       return "very fast"
```

4) 라이브러리

이제 우리는 파이썬에 대한 대부분의 것들을 알게 되었다. 이제 여러분은 자신이 원하는 프로그램을 직접 만들어볼 수 있을 것이다. 하지만 무엇인가를 만들기 전에 살펴보아야 할 것이 있다. 그것은 자신이 만들려는 프로그램을 이미 누군가가 만들어 놓았을지도 모른다는 사실이다.

물론 공부를 목적으로 누군가가 만들어 놓은 프로그램을 또 만들 수는 있지만 그런 목적이 아니라면 이미 만들어진 것을 다시 만드는 것은 어리석은 행동일 것이다. 그리고 이미 만들어진 것들은 테스트과정을 무수히 거친 훌륭한 것들이기도 하다. 따라서 무엇인가 새로운 프로그램을 만들기 전에 이미 만들어진 것들, 그 중에서도 특히 파이썬 배포본에 함께 들어 있는 파이썬 라이브러리들에 대해서 살펴보는 것은 매우 중요한 일일 것이다.

파이썬 라이브러리는 매우 광범위하고 훌륭한 것들이다. 이 책에서는 모든 라이브러리들을 살펴볼 수는 없고 자주 쓰이는 유용한 것들에 대해서만 다루기로 한다. 우선 라이브러리들을 살펴보기 전에 파이썬 내장함수를 먼저 보도록 하자.

[1] 내장함수

우리는 이미 몇 가지의 내장 함수들을 사용해 왔다. print, del, type 등이 바로 그것이다. 이러한 파이썬 내장 함수들은 외부 모듈과는 달리 import를 필요로 하지 않는다. 아무런 설정 없이 바로 사용할 수가 있다.

이곳에서 우리는 모든 내장함수에 대해서 알아보지는 않을 것이다. 다만 활용빈도가 높고 중요한 것들에 대해서만 간략히 알아볼 것이다. 여기서 설명하고 있지 않은 것들에 대해서는 라이브러리 레퍼런스를 참고하도록 하자.

abs
abs(x)는 숫자값을 입력값으로 받았을 때, 그 숫자의 절대값을 돌려주는 함수이다.

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(1+2j)
2.2360679774997898
>>>
```

복소수의 절대값은 다음과 같이 구해진다.

```
abs(a + bj) = sqrt (a^2+b^2)
```

apply
apply(function, (args))는 함수 이름과 그 함수의 인수를 입력으로 받아 간접적으로 함수를 실행시키는 명령어이다.

```
>>>def sum(a,b):
. . . return a+b
>>>
```

위와 같이 두 값의 합을 되돌려 주는 함수를 만들었다고 했을 때 아래와 같이 apply를 호출하여 쓸 수 있다.

```
>>> apply(sum, (3,4))
7
```

chr

chr(i)는 정수 형태의 아스키코드값을 입력으로 받아서 그에 해당하는 문자를 출력하는 함수이다.

```
>>> chr(97)
'a'
>>> chr(48)
'0'
>>>
```

cmp

cmp(x,y)는 두 개의 객체를 비교하는 함수이다. 만약 x가 크다면 양수값을 작다면 음수값을 반환한다. 같은 경우에는 0를 반환한다.

```
>>> cmp(4,3)
1
>>> cmp(3,4)
-1
>>> cmp(3,3)
0
```

dir

dir은 객체가 가지고 있는 변수나 함수를 리스트 형태로 보여준다. 아래의 예는 리스트와 딕셔너리 객체의 관련 함수들(메소드)을 보여주는 예이다. 우리가 앞서서 살펴보았던 관련함수들을 구경할 수 있을 것이다.

```
>>> dir([1,2,3])
['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys', 'setdefault', 'update', 'values']
```

divmod

divmod(a,b)는 두 개의 숫자를 입력값으로 받았을 때 그 몫과 나머지를 터플의 형태로 반환하는 함수이다.

```
>>> divmod(7,3)
(2,1)
>>> divmod(1.3, 0.2)
(6.0, 0.09999999999999978)
```

enumerate

enumerate는 입력값으로 시퀀스자료형(리스트, 터플, 문자열)을 입력으로 받아 enumerate객체를 리턴한다. enumerate객체는 첫번째로 그 순서값, 두번째로 그 순서값에 해당되는 시퀀스 자료형의 실제값을 갖는 객체이다.

enumerate는 아래의 예와같이 보통 for문과 함께 사용된다.

```
>>> for i, name in enumerate(['boby', 'foo', 'bar']):
...     print i, name
...
0 boby
1 foo
2 bar
```

반복구간에서 시퀀스 자료형의 값 뿐만 아니라 현재 어떤 위치에 있는지에 대한 인덱스값이 필요한 경우에 enumerate함수는 매우 유용하다.

eval
eval(expression)은 입력값으로 실행가능한 문자열(1+2, 'hi' + 'a' 같은 것)을 입력으로 받아서 문자열을 실행한 결과값을 반환하는 함수이다.

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4,3)')
(1, 1)
```

execfile
execfile(file)은 입력으로 파이썬 파일 이름을 받아서 그 파이썬 파일을 실행시키는 명령이다. 만약, 다음과 같은 스크립트를 sum.py라는 이름으로 만들어서 저장했다면,

```
# sum.py
def sum(a,b):
   return a+b

print sum(3,4)
```

다음처럼 인터프리터 모드에서 실행시킬 수 있다.

```
>>> execfile('sum.py')
7
```

filter
filter(function, list)는 함수와 리스트를 입력으로 받아서 리스트의 값이 하나씩 함수에 인수로 전달될 때, 참을 반환시키는 값만을 따로 모아서 리스트의 형태로 반환하는 함수이다. filter의 뜻은 무엇인가를 걸러낸다는 뜻이다. 이 의미가 filter 함수에서도 그대로 사용된다. 다음의 예를 보자.

```
#positive.py
def positive(l):
   result = []
```

```
   for i in l:
       if i > 0:
           result.append(i)
   return result

print positive([1,-3,2,0,-5,6])
```

결과값:

```
 [1, 2, 6]
```

즉, 위의 positive함수는 리스트를 입력값으로 받아서 각각의 요소를 판별해 양수값만 따로 리스트에 모아 그 결과값을 돌려주는 함수이다.

filter함수를 이용하면 아래와 같이 위의 내용을 간단하게 쓸 수 있다.

```
#filter1.py
def positive(x):
   return x > 0

print filter(positive, [1,-3,2,0,-5,6])
```

결과값:

```
 [1, 2, 6]
```

filter 함수는 첫 번째 인수로 함수명을, 두 번째 인수로는 그 함수에 차례로 들어갈 시퀀스 자료형(리스트, 터플, 문자열)을 받는다. filter 함수는 두 번째 인수인 각 리스트의 요소들이 첫 번째 인수인 함수에 들어갔을 때 리턴값이 참인 것만을 리스트로 묶어서 돌려준다. 위의 예에서는 1, 2, 6 만이 양수로 x > 0 이라는 문장이 참이 되므로 [1, 2, 6]이라는 결과 값을 돌려주게 된다.

lambda를 쓰면 더욱 간편하게 쓸 수 있다. (lambda함수는 잠시 후에 설명한다.)

```
>>> print filter(lambda x: x > 0, [1,-3,2,0,-5,6])
[1, 2, 6]
```

hex
hex(x)는 입력으로 정수값을 받아서 그 값을 십육진수값(hexadecimal)로 변환하여 돌려주는 함수이다.

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

id
id(object)는 객체를 입력값으로 받아서 객체의 고유값(레퍼런스)을 반환하는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

a, b 3이 모두 같은 객체를 가리키고 있음을 보여 준다.

4는 다른 객체이므로 당연히 id(4)는 다른 값을 보여 준다.

```
>>> id(4)
135072292
```

input
input([prompt])은 사용자 입력을 받는 함수이다. raw_input과 다른 점에 대해서는 "입력과 출력" 챕터에 설명되어 있다.
입력 인수로 문자열을 주면 아래의 세번째 예에서 보듯이 그 문자열은 프롬프트가 된다.

```
>>> a = input()
'hi'
>>> a
'hi'
>>> b = input("Enter: ")
Enter: 'hi'
```

위에서 입력 받은 문자열을 확힌해 보면 다음과 같다.

```
>>> b
'hi'
```

int
int(x)는 스트링 형태의 숫자나 소수점 숫자 등을 정수의 형태로 반환시켜 돌려준다. 정수를 입력으로 받으면 그대로 돌려준다.

```
>>> int('3')
3
>>> int(3.4)
3
```

int(x, radix)는 x라는 문자열을 radix(진수)형태로 계산한 값을 리턴한다.

'11'이라는 이진수 값에 대응되는 십진수 값은 다음과 같이 구한다.

```
>>> int('11', 2)
3
```

'1A'라는 십육신수 값에 대응되는 십진수 값은 다음과 같이 구한다.

```
>>> int('1A', 16)
26
```

isinstance

isinstance(object, class)는 입력값으로 인스턴스와 클래스 이름을 받아서 입력으로 받은 인스턴스가 그 클래스의 인스턴스인지를 판단하여 참이면 True, 거짓이면 False를 반환한다.

```
>>> class Person: pass
. . .
>>> a = Person()
>>> b = 3
>>> isinstance(a, Person)
True
```

위의 예는 a가 Person 클래스에 의해서 생성된 인스턴스임을 확인시켜 준다.

```
>>> isinstance(b, Person)
False
```

b는 Person 클래스에 의해 생성된 인스턴스가 아니다.

lambda

lambda는 함수를 생성할 때 사용되는 예약어로 def와 동일하나 보통 한줄로 간결하게 만들어 사용할 때 사용한다. lambda는 " 람다 " 라고 읽으며 보통 def를 쓸 정도로 복잡하지 않거나 def를 쓸 수 없는 곳에 쓰인다. lambda는 다음과 같이 정의된다.

```
lambda 인수1, 인수2,,, : 인수를 이용한 표현식
```

한 번 만들어 보자.

```
>>> sum = lambda a, b: a+b
>>> sum(3,4)
7
```

lambda를 이용한 sum함수는 인수로 a, b를 받고 a와 b를 합한 값을 돌려준다. 위의 lambda를 이용한 sum함수는 다음의 def를 이용한 함수와 하는 일이 완전히 동일하다.

```
>>> def sum(a, b):
. . .     return a+b
. . .
>>>
```

그렇다면 def가 있는데 왜 lambda라는 것이 나오게 되었을까? 이유는 간단하다. lambda는 def 대신 간결하게 사용할 수 있고 def로 쓸 수 없는 곳에 lambda는 쓰일 수 있기 때문이다. 리스트 내에 lambda가 들어간 경우를 살펴보자.

```
>>> l = [lambda a,b:a+b, lambda a,b:a*b]
>>> l
[at 0x811eb2c>, at 0x811eb64>]
```

즉 리스트 각각의 요소에 lambda 함수를 만들어 쓸 수 있다. 첫 번째 요소 l[0]은 두개의 입력값을 받아서 합을 돌려주는 lambda 함수이다.

```
>>> l[0]
at 0x811eb2c>
>>> l[0](3,4)
7
```

두 번째 요소 l[1]은 두개의 입력값을 받아서 곱을 돌려주는 lambda 함수이다.

```
>>> l[1](3,4)
12
```

프로그래밍을 하다 보면 lambda 함수의 사용용도는 무궁무진함을 알게 될 것이다.

len

len(s)은 인수로 시퀀스 자료형(문자열, 리스트, 터플)을 입력받아 그 길이(요소의 개수)를 돌려주는 함수이다.

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

list

list(s)는 인수로 시퀀스 자료형을 입력받아 그 요소를 똑같은 순서의 리스트로 만들어 돌려주는 함수이다. 리스트를 입력으로 주면 똑같은 리스트를 복사하여 돌려준다.

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
>>> a = [1,2,3]
>>> b = list(a)
>>> b
[1, 2, 3]
>>> id(a)
9164780
>>> id(b)
9220284
```

List(a)는 리스트를 복사해서 다른 리스트를 돌려주는 것을 위의 예에서 확인할 수 있다. 즉 a와 b의 id값이 서로 다르다.

long

long(x)은 숫자 형태의 문자열이나 숫자를 인수로 입력받아 큰 정수형(long integer)으로 돌려주는 함수이다.

```
>>> long('34567890')
34567890L
>>> long(34567890)
34567890L
```

map

map이라는 것은 함수와 시퀀스 자료형(리스트, 터플, 문자열)을 입력으로 받아서 시퀀스 자료형의 각각의 요소가 함수의 입력으로 들어간 다음 나오는 출력값을 묶어서 리스트로 돌려주는 함수이다.

다음의 함수를 보자.

```
def two_times(l):
   result = []
   for i in l:
       result.append(i*2)
   return result
```

이 함수는 리스트를 입력받아서 각각의 요소에 2를 곱한 결과값을 돌려주는 함수이다.

다음과 같이 쓰일 것이다.

```
# two_times.py
def two_times(l):
   result = []
       for i in l:
           result.append(i*2)
   return result

result = two_times([1,2,3,4])
print result
```

결과값:

```
[2, 4, 6, 8]
```

이것을 다음과 같이 해보자.

```
>>> def two_times(x): return x*2
. . .
>>> map(two_times, [1,2,3,4])
[2, 4, 6, 8]
```

즉 map이란 함수는 입력값으로 함수명과 그 함수에 들어갈 인수로 리스트 등의 시퀀스 자료형을 받는다. 이것은 다음과 같이 해석된다. 리스트의 첫 번째 요소인 1이 two_times 함수의 입력값으로 들어가서 1 * 2의 과정을 거쳐 2의 값이 결과값 리스트 []에 추가된다. 이 때 결과값은 [2]가 되고, 다음에 리스트의 두 번째 요소인 2가 two_times 함수의 입력값으로 들어가서 4가 된 다음 이 값이 또 결과값인 [2]에 추가되어 결과값은 [2, 4]가 된다. 총 4개의 요소값이 반복되면 결과값은 [2, 4, 6, 8]이 될 것이고 더 이상의 입력값이 없으면 최종 결과값인 [2, 4, 6, 8]을 돌려준다. 이것이 map 함수가 하는 일이다. 즉 map은 결과 값으로 리스트를 리턴한다.

위의 예는 lambda를 쓰면 다음처럼 간략화된다.

```
>>> map(lambda a: a*2, [1,2,3,4])
[2, 4, 6, 8]
```

[map을 이용해서 리스트의 값을 1씩 추가하는 예]

```
# map_test.py
def plus_one(x):
   return x+1
print map(plus_one, [1,2,3,4,5])
```

결과값: [2,3,4,5,6]

max

max(s)는 인수로 시퀀스 자료형(문자열, 리스트, 터플)을 입력받아 그 최대값을 돌려주는 함수이다.

```
>>> max([1,2,3])
3
>>> max("python")
'y'
```

min

min(s)은 max와는 반대로 시퀀스 자료형을 입력받아 그 최소값을 돌려주는 함수이다.

```
>>> min([1,2,3])
1
>>> min("python")
'h'
```

oct

oct(x)는 정수 형태의 숫자를 8진수 문자열로 바꾸어 돌려주는 함수이다.

```
>>> oct(34)
'042'
>>> oct(12345)
'030071'
```

open

open(filename, [mode])은 파일 이름과 읽기 방법을 입력받아 파일 객체를 돌려주는 함수이다. 읽기 방법(mode)이 생략되면 기본적으로 읽기 전용 모드('r')로 파일객체를 만들어 돌려준다. 그 외에 더 자세한 사항은 라이브러리 레퍼런스를 참조하자.

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

w+, r+, a+ 는 파일을 업데이트할 용도로 사용된다.

b는 w, r, a와 함께 사용된다.

```
>>> f = open("binary_file", "rb")
>>> fwrite = open("write_mode.txt", 'w')
>>> fread = open("read_mode.txt", 'r')
```

```
>>> fread2 = open("read_mode.txt")
```

fread와 fread2는 동일한 값을 나타낸다. 즉, 읽기모드 부분이 생략되면 기본값으로 'r'을 갖게 된다.

다음은 추가 모드로 파일을 여는 예이다.

```
>>> fappend = open("append_mode.txt", 'a')
```

ord
ord(c)는 문자의 아스키 값을 돌려주는 함수이다.

```
>>> ord('a')
97
>>> ord('0')
48
```

pow
pow(x, y)는 x의 y승을 한 결과값을 돌려주는 함수이다.

```
>>> pow(2, 4)
16
>>> pow(3, 3)
27
```

range
range([start,] stop [,step])는 for문과 잘 사용되는 것으로 인수로 정수값을 주어 그 숫자에 해당되는 범위의 값을 리스트의 형태로 돌려주는 함수이다.

인수가 하나일 경우

```
>>> range(5)
[0, 1, 2, 3, 4]
```

인수가 두 개일 경우 (입력으로 주어지는 두 개의 숫자는 시작과 끝을 나타낸다. 끝번호가 포함이 안된다는 것에 주의 하자.)

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
```

인수가 세 개일 경우 - 세 번째 인수는 시작번호부터 끝번호까지 가는데 숫자 사이의 거리를 말한다.

```
>>> range(1, 10, 2)
[1, 3, 5, 7, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

raw_input

raw_input([prompt])은 사용자 입력을 받는 함수로 prompt는 입력을 받는 메시지이다. 입력받은 값의 맨 마지막 문자('\n')를 없앤 값을 돌려준다.

```
>>> a = raw_input()
you need python
>>> a
'you need python'
>>> b = raw_input("--->")
--->you need python
>>> b
'you need python'
```

이미 설명했던 입출력 부분에 input과 raw_input에 대한 설명을 하였다.

reduce

reduce(function, sequence) 이것은 인수를 두 개 받는 함수인 function과 시퀀스 자료형을 인수로 입력받아서 시퀀스 요소의 왼쪽부터 차례대로 끝까지 감소시키며 입력받은 함수를 실행시키는 함수이다.

```
# reduce1.py
def test(x, y):
   return x+y

def test2(x, y):
   return x*y

print reduce(test, [1,2,3,4,5])
print reduce(test2, [1,2,3,4,5])
```

결과값:

```
 15
120
```

reduce(test, [1,2,3,4,5])가 적용되는 과정은 다음과 같다.

```
((((1+2)+3)+4)+5)
```

reduce(test2, [1,2,3,4,5])가 적용되는 과정은 다음과 같다.

```
((((1*2)*3)*4)*5)
```

reload

reload(module)는 이미 불러왔던(import) 모듈을 다시 부르는 함수로 모듈 객체를 돌려준다. 외부 에디터로 다음을 작성해 보자.

```
# test.py
def a():
   print "life is too short"
```

다음에 대화형 인터프리터에서 이 모듈을 불러보자.

```
>>> import test
>>> test.a()
'life is too short'
```

대화형 인터프리터를 닫지 말고 외부 에디터로 test.py 파일을 바꾸어보자.

```
#test.py
def a():
   print "you need python"
```

다음에 다시 대화형 인터프리터에서 다음과 같이 해보자.

```
>>> test.a()
'life is too short'
>>> reload(test)
>>> test.a()
'you need python'
```

즉, 위의 경우처럼 이미 불러온 모듈을 수정한 다음 바뀐 내용을 적용하고 싶을 때 주로 사용하게 된다.

repr

repr(object)은 객체를 출력할 수 있는 문자열 형태로 변환하여 돌려주는 함수이다. 이 변환된 값은 주로 eval 함수의 입력으로 쓰인다. str 함수와의 차이점이라면 str으로 변환된 값은 eval의 입력값이 될 수 없는 경우가 있다는 것이다.

```
>>> repr("hi".upper())
"'HI'"
>>> eval(repr("hi".upper()))
'HI'
>>> eval(str("hi".upper()))
Traceback (innermost last):
File "", line 1, in ? eval(str("hi".upper()))
File "", line 0, in ?
NameError: There is no variable named 'HI'
```

위의 경우처럼 str을 쓸 경우 eval 함수의 입력값이 될 수 없는 경우가 있다.

sorted

sorted 함수는 입력으로 받은 시퀀스 자료형을 소트한 후 그 결과를 리스트로 리턴하는 함수이다.

```
>>> sorted([3,1,2])
[1, 2, 3]
>>> sorted(['a','c','b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3,2,1))
[1, 2, 3]
```

리스트 자료형에도 sort라는 함수가 있다. 하지만 리스트 자료형의 sort함수는 리스트 객체 그 자체를 소트할 뿐이지 소트된 결과를 리턴하지는 않는다.

다음의 예제로 sorted와 리스트 자료형의 sort함수와의 차이점을 확인해 보자.

```
>>> a = [3,1,2]
>>> result = a.sort()
>>> print result
None
>>> a
[1, 2, 3]
```

str

str(object)은 객체를 출력할 수 있는 문자열 형태로 변환하여 돌려주는 함수이다. 단 문자열 그 자체로만 돌려주는 함수이다. 위의 repr함수와의 차이점을 살펴보자.

```
>>> str(3)
'3'
>>> str('hi')
'hi'
>>> str('hi'.upper())
'HI'
```

tuple

tuple(sequence)은 인수로 시퀀스 자료형을 입력받아 터플 형태의 자료로 바꾸어 돌려준다. 터플형이 입력으로 들어오면 그대로 돌려준다.

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1,2,3])
(1, 2, 3)
>>> tuple((1,2,3))
(1, 2, 3)
```

type

type(object)은 인수로 객체를 입력받아 그 객체의 자료형이 무엇인지 알려주는 함수이다.

```
>>> type("abc")
>>> type([])
>>> type(open("test", 'w'))
```

마지막 예에서 볼 수 있듯이 파일을 쓰기 모드로 연 객체의 자료형은 'file'임을 알 수 있다.

zip

zip 함수는 동일한 갯수의 요소값을 갖는 시퀀스 자료형을 묶어주는 역할을 한다. 예제로 확인해 보자.

```
>>> zip([1,2,3], [4,5,6])
[(1, 4), (2, 5), (3, 6)]
>>> zip([1,2,3], [4,5,6], [7,8,9])
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> zip("abc", "def")
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

[2] 외장함수

유용한 파이썬 라이브러리들
파이썬 사용에 날개를 달아보자. 전세계의 파이썬 사용자들에 의해서 이미 만들어진 프로그램들을 모아놓은 것이 바로 파이썬 라이브러리이다. ' 라이브러리 ' 는 ' 도서관 ' 이다. 즉, 찾아보는 곳이다. 모든 라이브러리를 공부할 필요는 없다. 그저 어떤 곳에 어떤 라이브러리를 써야 한다는 것만 알면 된다.

그러기 위해서 어떤 라이브러리들이 존재하고 어떻게 사용하는지에 대해서 알아야 할 필요가 있다. 이곳에서 파이썬의 모든 라이브러리를 다루지는 않을 것이다. 다만, 자주 쓰이고 꼭 알아야만 한다고 여겨지는 것들에 대해서만 다루도록 하겠다.

그리고 여기서는 주로 실례 위주로 설명할 것이다. 자세한 것은 파이썬과 함께 배포되는 파이썬 라이브러리 레퍼런스를 참고하도록 하자.
(※ 파이썬 라이브러리는 파이썬 설치시 자동으로 컴퓨터에 설치가 된다.)

명령행에서 인수를 전달(sys.argv)
sys 모듈은 파이썬 인터프리터가 제공하는 변수들과 함수들을 직접 접근하여 제어할 수 있게 해주는 모듈이다.

sys.argv

```
C:\Python21> python test.py abc pey guido
```

위의 예에서와 같이 python test.py뒤에 또 다른 값들을 함께 넣어주면 sys.argv라는 리스트에 그 값들이 추가되게 된다.

예제따라해 보기)

1. 우선 다음과 같은 파이썬 프로그램을 작성하자. (C:\Python\Mymoduels 라는 디렉토리에 저장했다고 가정을 한다.)

```
# argv_test.py
import sys
print sys.argv
```

2. 도스창에서 다음과 같이 해보자.

```
C:\Python\Mymodules> python argv_test.py you need python
['argv_test.py', 'you', 'need', 'python']
```

위처럼 python이란 명령어 뒤의 모든 것들이 공백을 기준으로 나뉘어서 sys.argv 리스트의 요소가 됨을 알 수 있다. 2장 입출력 부분의 명령행 입출력에서 자세한 내용을 다루고 있다.

강제 스크립트 종료법(sys.exit)

```
>>> sys.exit()
```

sys.exit는 Ctrl-Z나 Crtl-D를 눌러서 대화형 인터프리터를 종료하는 것과 같은 기능을 한다. 또, 프로그램 파일 내에서 쓰이면 프로그램을 중단하게 된다.

자신이 만든 모듈 불러서 쓰기(sys.path)
sys.path는 파이썬 모듈들이 저장되어 있는 위치를 나타낸다. 즉, 이 위치에 있는 xxx.py 파일들은 경로에 상관없이 어디에서나 불러올 수가 있다.

다음은 그 실행 결과이다.

```
>>> import sys
>>> sys.path
['', 'c:\python21', 'c:\python21\dlls', 'c:\python21\lib', 'c:\python21\l
ib\plat-win', 'c:\python21\lib\lib-tk']
>>>
```

위의 예에서 ''는 현재 디렉토리를 말한다.

```
# path_append.py
import sys
sys.path.append("C:\Python\Mymodules")
```

위와 같이 파이썬 프로그램 파일에서 sys.path.append를 이용해 경로명을 추가시킬 수 있다. 이렇게 하고 난 뒤에는 C:\Python\Mymodules라는 디렉토리에 있는 xxx.py 파일을 불러서 쓸 수가 있다.

객체를 그 상태 그대로 파일에 저장하고 싶을 때(pickle)
pickle 모듈은 객체의 형태를 그대로 유지하게 하여 파일에 저장시키고 또 불러올 수 있게 하는 모듈이다.

아래의 예는 파일을 쓰기 모드로 열어서 딕셔너리 객체인 data를 그대로 pickle.dump로 저장하는 전형적인 방법이다.

```
>>> import pickle
>>> f = open("test.txt", 'w')
>>> data = {1: 'python', 2: 'you need'}
>>> pickle.dump(data, f)
>>> f.close()
```

pickle.dump에 의해 저장된 파일을 열어 원래 있던 딕셔너리 객체(data)를 그대로 가져오는 전형적인 예이다.

```
>>> import pickle
>>> f = open("test.txt", 'r')
>>> data = pickle.load(f)
>>> print data
{2:'you need', 1:'python'}
```

위의 예에서는 딕셔너리 객체를 이용하였지만 어떤 자료형이든지 상관없다.

문자열 처리(string)
string 모듈은 문자열 처리에 쓰인다. 이 책 2장 파이썬 자료형 중 문자열에 관한 내용에서 이미 문자열 관련 함수를 알아보았다. 이 string 모듈의 관련함수들은 문자열 자체의 관련 함수들과 거의 일치하며 똑같이 동작한다. 여기서는 string 모듈에만 있는 함수 몇 개만 알아보기로 하자.

```
>>> import string
>>> dir(string)
['_StringType', '__builtins__', '__doc__', '__file__', '__name__', '_float', '_idmap', '_idmapL', '_int',
'_long', 'atof', 'atof_error', 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize', 'capwords',
```

```
'center', 'count', 'digits', 'expandtabs', 'find', 'hexdigits', 'index', 'index_error', 'join', 'joinfield
'letters', 'ljust', 'lower', 'lowercase', 'lstrip', 'maketrans', 'octdigits', 'printable', 'punctuation',
'replace', 'rfind', 'rindex', 'rjust', 'rstrip', 'split', 'splitfields', 'strip', 'swapcase', 'translate'
'upper', 'uppercase', 'whitespace', 'zfill']
```

위에서 보듯이 dir(string)을 한 결과 우리가 이미 알아본 문자열 관련 함수들과 거의 일치함을 확인할 수 있다. 즉, 다음은 완전히 동일하다.

```
>>> import string
>>> string.split("you need python")
['you', 'need', 'python']
>>> "you need python".split()
['you', 'need', 'python']
```

위의 예는 string 모듈의 split 함수를 이용한 것이고 밑의 예는 문자열 자체의 split함수를 이용한 것이다. string 모듈에만 있는 함수에는 다음과 같은 것들이 있다.

atof
문자열 형태의 숫자를 실수로 바꾸어 준다.

```
>>> string.atof('3')
3.0
```

atof는 실수로 변환 가능한 문자를 실수 형태로 바꾸어 주는 함수이다. 이것은 다음과 같은 의미이다.

```
>>> float('3')
3
```

atoi
위의 atof와 마찬가지로 정수로 변환 가능한 문자를 정수 형태로 바꾸어주는 함수이다. int 함수와 동일한 결과값을 출력한다.

```
>>> string.atoi('3')
3
>>> int('3')
3
```

zfill
이것은 주로 숫자를 표시하는 데 쓰이는 함수로 앞의 빈 공간을 0으로 채워 준다. zfill 은 zero + fill 즉 0으로 채운다는 의미이다.

```
>>> string.zfill(3, 8)
'00000003'
>>> string.zfill('a', 3)
'00a'
```

string.zfill(a, b)에서 첫 번째 인수(a)는 표시할 값이고 두 번째 인수(b)는 총 길이를 나타낸다. 우리는 보통 날짜를 저장할 때 2001-04-04 식으로 04월 04일처럼 앞에 0을 써주는 게 종종 필요하다. 이 때 zfill을 사용하면 매우 유용할 것이다.

파일 흉내내기(StringIO)
StringIO는 파일처럼 취급되는 객체를 만들어낸다. 단 실제 파일객체는 아니고 흉내를 낼 뿐이다.

[사용예제]

우선 StringIO 모듈을 사용하기 위해서 불러온다(import).

```
>>> import StringIO
```

그리고 StringIO의 함수인 StringIO를 이용해서 StringIO의 객체를 하나 생성한다. 이 객체 마치 파일 객체처럼 행동한다. 따라서 파일 객체에 쓰이는 함수들이 그대로 적용된다.

```
>>> f = StringIO.StringIO()
```

그리고 그 객체에 write를 이용하여 "life is too short"라는 문장을 써넣었다.

```
>>> f.write("life is too short")
```

그리고 나서 그 객체에 쓰여진 문자열을 getvalue를 이용하여 읽어서 변수 value에 대입하였다. 이 getvalue는 StringIO에서만 쓰이는 함수이다.

```
>>> value = f.getvalue()
>>> value
'life is too short'
```

그리고 최종적으로 close를 이용해서 파일 객체를 메모리에서 없애버린다.

```
>>> f.close()
```

> **[참고]** StringIO가 자주 쓰이는 이유
>
> 우리는 프로그래밍을 할 때 가끔 문자열 데이터를 파일에 저장한 다음에 여러 가지 처리를 한다. 하지만 그 파일이 다시 쓰이지 않고 꼭 저장될 필요가 없다면 굳이 파일에 저장할 필요는 없을 것이다. 이렇게 꼭 파일을 만들어서 처리를 할 필요가 없게 해주는 모듈이 바로 StringIO다. 물론 항상 이러한 용도로만 쓰이는 것은 아니지만 대부분 파일 처리를 하는 곳에서 이 StringIO는 매우 유용하다.

현재 내 시스템 환경변수값을 알고싶을 때는? (os.environ)
시스템은 제각기 다른 환경 변수 값들을 가지고 있는데 파이썬에는 이러한 환경 변수값 들을 보여주는 os모듈의 environ이다. 다음을 따라해 보자.

```
>>> import os
>>> os.environ
{'CMDLINE': 'WIN', 'PATH': 'C:\WINDOWS;C:\WINDOWS\COMMAND;C:\PROGRA~1\ULTRA
EDT;C:\JDK1.3\BIN;C:\ESSOLO.COM;C:\VIM\VIM\VIM57;C:\PYTHON21', 'BLASTER':
```

```
'A240 I10 D1', 'TEMP': 'C:\WINDOWS\TEMP', 'COMSPEC': 'C:\WINDOWS\COMMAND.CO
M', 'PROMPT': '$p$g', 'WINBOOTDIR': 'C:\WINDOWS', 'WINDIR': 'C:\WINDOWS', 'TMP
': 'C:\WINDOWS\TEMP'}
>>>
```

위의 결과값은 필자의 시스템 정보이다. os.environ은 딕셔너리 객체를 돌려준다. 자세히 보면 여러 가지 유용한 정보를 찾을 수 있다.

딕셔너리이기 때문에 다음과 같이 호출할 수 있다. 필자의 시스템의 PATH변수이다. 이것은 윈도우즈 C:\WINDOWS\autoexec.bat에서 설정해 놓은 정보이다.

```
>>> os.environ['PATH']
'C:\WINDOWS;C:\WINDOWS\COMMAND;C:\PROGRA~1\ULTRAEDT;C:\JDK1.3\BIN;C:\ESS
OLO.COM;C:\VIM\VIM\VIM57;C:\PYTHON21'
>>>
```

디렉토리에 대한 것들(os.chdir, os.getcwd)

아래와 같이 현재 디렉토리의 위치를 변경할 수 있다.

```
>>> os.chdir("C:\WINDOWS")
os.getcwd()
```

현재 자신의 디렉토리 위치를 돌려준다.

```
>>> os.getcwd()
'C:\WINDOWS'
```

시스템 명령(os.system, os.popen)

os.system
시스템의 유틸리티나 기타 명령어들을 파이썬에서 호출할 수 있다. os.system(" 명령어 ")처럼 사용해야 한다. 다음은 현재 디렉토리에서 dir을 실행하는 예이다.

```
>>> os.system("dir")
```

dir의 결과값 출력

os.popen
os.popen은 시스템 명령어를 실행시킨 결과값을 읽기 모드 형태의 파일객체로 돌려준다.

```
>>> files = os.popen("dir")
>>> files
>>>
```

읽은 파일 객체의 내용을 보기 위해서는 다음과 같이 하면 될 것이다.

```
>>> print files.read()
```

기타 유용한 os 관련 함수

함수	설명
os.mkdir(디렉토리)	디렉토리를 생성한다.
os.rmdir(디렉토리)	디렉토리를 삭제한다.단, 디렉토리가 비어있어야 삭제가 가능하다.
os.unlink(파일)	파일을 지운다.
os.rename(src, dst)	src라는 이름의 파일을 dst라는 이름으로 바꾼다.

파일 복사(shutil)
shutil은 파일을 복사해 주는 파이썬 모듈이다.

shutil.copy(src, dst)
src라는 이름의 파일을 dst로 복사한다. 만약 dst가 디렉토리 이름이라면 src라는 파일이름으로

dst이라는 디렉토리에 복사하고 그 이름이 존재하면 덮어쓰게 된다.

```
>>> import shutil
>>> shutil.copy("src", "dst")
```

디렉토리에 있는 파일들을 리스트로 만들려면 (glob)
가끔 파일을 읽고 쓰는 기능이 있는 프로그램들을 만들다 보면 해당 디렉토리의 파일들의 이름을 알아야 할 때가 있는데 이럴 때 쓰이는 모듈이 바로 glob이다.

glob(pathname)
glob모듈은 해당 디렉토리내의 파일들을 읽어서 리스트로 돌려준다. *, ?등의 메타문자를 써서 원하는 파일만을 읽어들일 수도 있다.

다음은 C:\Python이란 디렉토리에 있는 파일중 이름이 Q문자로 시작하는 파일들을 모두 찾아서 리스트로 돌려준 예이다.

```
>>> import glob
>>> glob.glob("C:\Python\Q*")
['C:\Python\quiz.py', 'C:\Python\quiz.py.bak']
>>>
```

임시파일 (tempfile)
임시적으로 파일을 만들어서 쓸 때 유용하게 쓰이는 모듈이 바로 tempfile이다. tempfile.mktemp()는 중복되지 않는 임시파일의 이름을 만들어서 돌려준다.

```
>>> import tempfile
>>> filename = tempfile.mktemp()
>>> filename
'C:\WINDOWS\TEMP\~-275151-0'
```

tempfile.TemporaryFile()은 임시적인 저장공간으로 사용될 파일 객체를 돌려준다. 기본적으로 w+b 의 모드를 갖는다. 이 파일객체는 f.close()가 호출될 때 자동으로 사라지게 된다.

```
>>> import tempfile
>>> f = tempfile.TemporaryFile()
```

시간에 관한 것들(time)
time 모듈에 관련된 유용한 함수는 굉장히 많다. 그 중에서 가장 유용한 몇 가지만 알아보도록 하자.

time.time
time.time()은 UTC(Universal Time Coordinated 협정 세계 표준시)를 이용하여 현재의 시간을 실수형태로 반환하여 주는 함수이다. 1970년 1월 1일 0시 0분 0초를 기준으로 지난 시간을 초단위로 돌려준다.

```
>>> import time
>>> time.time()
988458015.73417199
```

time.localtime
time.localtime은 time.time()에 의해서 반환된 실수값을 이용해서 년도, 달, 월, 시, 분, 초,.. 의 터플 형태로 바꾸어 주는 함수이다.

```
>>> time.localtime(time.time())
(2001, 4, 28, 20, 48, 12, 5, 118, 0)
```

time.asctime
위의 time.localtime에 의해서 반환된 터플 형태의 값을 인수로 받아서 알아보기 쉬운 날짜와 시간 형태의 값을 반환하여 주는 함수이다. 가장 자주 사용되는 시간관련 함수이다.

```
>>> time.asctime(time.localtime(time.time()))
'Sat Apr 28 20:50:20 2001'
```

time.ctime
위의 time.asctime은 간단하게 다음처럼 표현된다.

```
>>> time.ctime()
'Sat Apr 28 20:56:31 2001'
```

time.strftime
time.strftime('출력할 형식포맷코드', time.localtime(time.time())) strftime 함수는 시간에 관계된 것을 세밀하게 표현할 수 있는 여러 가지 포맷코드를 제공해 준다.

포맷코드	설명	예
%a	요일 줄임말	Mon
%A	요일	Monday
%b	달 줄임말	Jan
%B	달	January
%c	날짜와 시간을 출력함(로케일 설정에 의한 형식에 맞추어)	06/01/01 17:22:21
%d	날(day)	[00,31]
%H	시간(hour)-24시간 출력 형태	[00,23]
%I	시간(hour)-12시간 출력 형태	[01,12]
%j	1년 중 누적 날짜	[001,366]
%m	달	[01,12]
%M	분	[01,59]
%p	AM or PM	AM
%S	초	[00,61]
%U	1년 중 누적 주-일요일을 시작으로	[00,53]
%w	숫자로 된 요일	[0(일요일),6]

%W	1년 중 누적 주-월요일을 시작으로	[00,53]
%x	현재 설정된 로케일에 기반한 날짜 출력	06/01/01
%X	현재 설정된 로케일에 기반한 시간 출력	17:22:21
%Y	년도 출력	2001
%Z	시간대 출력	대한민국 표준시
%%	문자 %	
%y	세기부분을 제외한 년도 출력	01

```
>>> import time
>>> time.strftime('%x', time.localtime(time.time()))
'05/01/01'
>>> time.strftime('%c', time.localtime(time.time()))
'05/01/01 17:22:21'
```

time.sleep

time.sleep 함수는 보통 루프 안에서 많이 쓰이는데 일정한 시간 간격을 주기 위해서 주로 쓰이게 된다. 다음 예제를 보자.

```
#sleep1.py
import time
for i in range(10):
   print i
   time.sleep(1)
```

위 예제는 1초 간격으로 0부터 9까지의 숫자를 출력하게 된다. time.sleep 함수의 인수로는 실수 형태가 가능하다. 즉 1이면 1초이고 0.5 이면 0.5초가 되는 것이다.

파이썬에서 달력쓰기(calendar)
파이썬에서 달력을 볼 수 있게 해 주는 모듈이다.

2001년의 전체 달력을 볼 수가 있다.

```
>>> import calendar
>>> print calendar.calendar(2001)
```

위와 똑같은 결과값을 보여준다.

```
>>> calendar.prcal(2001)
```

2001년 4월의 달력만을 보여준다. 하지만 위의 것들은 프로그래밍 내에서는 거의 쓰이질 않는다. 다만 대화형 인터프리터에서 오늘 날짜를 확인하는 정도의 용도로만 쓰인다.

```
>>> calendar.prmonth(2001, 4)
April 2001
Mo Tu We Th Fr Sa Su
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

다음의 유용한 calendar 모듈의 함수를 보도록 하자. weekday(년도, 월, 일) 함수는 그 날짜에 해당하는 요일 정보를 돌려준다. 월요일은 0, 화요일은 1, 수요일은 2, 목요일은 3, 금요일은 4, 토요일은 5, 일요일은 6이라는 값을 돌려준다.

```
>>> calendar.weekday(2001, 4, 28)
5
```

위의 예에서 2001년 4월 28일은 토요일이 될 것이다. monthrange(년도, 월) 함수는 입력받은 달의 1일이 무슨 요일인지와 그 달이 몇 일까지 있는지에 대한 정보를 터플 형태로 돌려준다.

```
>>> calendar.monthrange(2001,4)
(6, 30)
```

위의 예는 2001년 4월의 1일은 일요일이고 30일까지 있다는 것을 보여준다. 주로 날짜 관련된 프로그래밍을 할 때 위의 두 가지 함수는 매우 유용하게 사용된다.

난수 발생시키기 (random)
random은 난수발생 모듈이다. random과 randint에 대해서 알아보자.

다음은 0.0에서 1.0 사이의 실수값 중에서 난수값을 돌려주는 예를 보여준다.

```
>>> import random
>>> random.random()
0.53840103305098674
```

1에서 10사이의 정수사이에서 난수값을 돌려준다.

```
>>> random.randint(1,10)
6
```

1에서 55 사이의 정수 사이의 난수 값을 돌려준다.

```
>>> random.randint(1,55)
43
```

이러한 난수를 이용해서 재미있는 함수를 하나 만들어보자.

```
# random_pop.py
import random
def random_pop(data):
   number = random.randint(0, len(data)-1)
```

```
   return data.pop(number)

if __name__ == "__main__":
   data = [1,2,3,4,5]
   while data: print random_pop(data)
```

결과값:

```
2
3
1
5
4
```

위의 random_pop 함수는 리스트의 요소 중에서 무작위로 하나를 선택하여 꺼낸 다음 그 값을 돌려주는 함수이다. 물론 꺼내어진 리스트의 요소는 사라진다.

파이썬에서의 쓰레드 (thread)
동작하고 있는 프로그램을 프로세스(Process)라고 한다. 보통 한 개의 프로세스는 한 가지의 일을 하지만, 이 쓰레드를 이용하면 한 프로세스 내에서 두 가지 또는 그 이상의 일을 동시에 할 수 있게 된다. 간단한 예제로 설명을 대신 하겠다.

```
# thread_test.py
import thread
import time
def say(msg):
   while 1:
       print msg
       time.sleep(1)
thread.start_new_thread(say, ('you',))
thread.start_new_thread(say, ('need',))
thread.start_new_thread(say, ('python',))
for i in range(100):
   print i
   time.sleep(0.1)
```

위 예제에서는 새로운 쓰레드를 세 개 만들었다. 1초마다 'you', 'need', 'python' 을 찍게 만드는 프로그램이다. 그리고 매 0.1초마다 0부터 99까지 숫자를 출력하게 하였다. 마지막 for가 사용된

부분이 Main 쓰레드가 된다.

결과값은 다음과 같을 것이다.

```
0
you
need
python
1
2
3
4
5
6
7
8
9
10
you
need
python
11
12
...
```

이것을 가능하게 해주는 것은 thread.start_new_thread로서 그 첫 번째 인수로는 함수명을, 두 번째 인수로는 그 함수의 입력 변수로 들어갈 터플 형태의 입력 인수를 받는다. 내장함수인 apply와 동일한 구조이다.

웹브라우저 실행시키기 (webbrowser)
webbrowser는 자신의 시스템에서 사용하는 기본 웹브라우저가 자동으로 실행되게 하는 모듈이다.

아래의 예는 웹브라우져를 자동으로 실행시키고 해당 URL인 http://www.yahoo.co.kr로 가게 해준다.

```
>>> import webbrowser
>>> webbrowser.open(http://www.yahoo.co.kr)
```

webbrowser의 open 함수는 웹브라우저가 실행된 상태이면 해당 주소로 이동하고 웹브라우저가 실행되지 않은 상태이면 새로이 웹브라우저가 실행되어 해당 주소로 이동한다.

open_new 함수는 이미 웹브라우저가 실행된 상태에서 새로운 창으로 해당 주소가 열리도록 한다.

```
>>> webbrowser.open_new(http://www.yahoo.co.kr)
```

06. 어디서부터 시작할 것인가?

이 곳에서는 아주 짤막한 스크립트나 함수들을 만들어 볼 것이다. 유용할 수도 있고 그렇지 않을 수도 있지만 독자의 프로그래밍 감각을 늘리는데는 더할 나위 없이 좋은 재료들이 될 것이다. 부디 이 책에 있는 것에 대해서만 생각하지 말고 자신이 새로운 것을 직접 만들어 보고 또 연구도 해가면서 파이썬을 공부하도록 하자.

이곳에 소개되는 모든 파이썬 프로그램 예제는 대화형 인터프리터가 아닌 에디터로 작성해야 한다. 스크립트라는 말이 처음 나왔는데 에디터로 작성한 파이썬 프로그램파일을 스크립트라고 부른다. 앞으로는 에디터로 작성한 파이썬 프로그램 파일을 계속 파이썬 스크립트라고 부를 것이니 혼동하지 말도록 하자.

[01] 내가 프로그램을 만들 수 있을까?

프로그램을 막 시작하려는 사람이 처음 느끼는 벽은 아마도 다음과 같을 것이다.

> “ 문법도 어느 정도 알겠고, 책의 내용도 대부분 다 이해하는데, 이러한 지식을 바탕으로 내가 도대체 어떤 프로그램을 만들 수 있을까? ”

이럴 때는 무턱대고 “ 어떤 프로그램을 짜야지 ” 라는 생각보다는 다른 사람들의 프로그램 파일들을 구경하고 분석하는데서 시작하는 것이 좋다. 그러면서 다른 사람들의 생각도 읽을 수 있고 거기에 더해 뭔가 새로운 아이디어가 떠오르기도 하는 것이다. 하지만 여기서 가장 중요한 것은 자신의 수준에 맞는 소스를 찾는 일이다. 그래서 이 책 5장에서는 아주 쉬운 예제부터 시작해서 차츰 어려워지는 실용적인 예제까지를 다루려고 노력하였다. 이것들을 어떻게 활용하는가는 독자의 몫이다.

이곳에 있는 예제들은 모두 필자가 만든 것인데, 대부분 다른 사람들이 만든 소스를 보고 아이디어를 얻은 것이 많다. 필자가 만든 예제들을 쉽게 이해하기 위해서는 필자가 프로그램을 만들 때 어떤 생각을 하면서 만들었는지 독자들이 알면 좋을 것이다. 그래서 여기에서는 필자의 프로그래밍 스타일을 잠시 소개하겠다.

필자는 이제 막 프로그래밍을 해보려는 사람에게 구구단 프로그램을 짜보라고 요구했던 적이 있었다. 생각보다 쉬운 퀴즈였는데 의외로 파이썬 문법도 다 공부한 사람이 프로그램을 어떻게 만들어야 할 지 갈피를 못잡고 있다는 사실은 놀라운 일이었다. 그래서 필자는 다음과 같은 해결책을 알려 주었다.

“ 입력과 출력 ” 을 생각하라는 것이었다. 우선 구구단 중 먼저 2단을 만들어야 하니까 2를 입력값으로 주었을 때 원하는 출력 값을 생각해 보라는 힌트였다. 그래도 그림이 그려지지 않는 듯 보여 직접 연습장에 그려주면서 설명을 해 주었다. 그것은 다음과 같았다. 이것이 필자가 프로그래밍을 하는 방식이다. 독자들도 함께 따라해 보기를 바란다.

먼저 에디터를 열고 다음과 같이 쓴다. 즉, GuGu라는 함수에 2라는 입력을 주면 result라는 결과 값을 준다.

```
result = GuGu(2)
```

그렇다면 이제 결과값을 어떤 형태로 받을 것인지를 고민한다. 2단이니까 2,4,6,,,18까지 갈 것이다. 아무래도 위와 같은 데이터는 리스트가 좋을 것 같다. 따라서 result = [2, 4, 6, 8, 10, 12, 14, 16, 18] 이런 결과를 얻는 것이 좋겠다는 생각을 먼저 하고 나서 프로그래밍을 시작하는 것이다. 그렇다면 의외로 생각이 가볍게 좁혀 지는 것을 느낄 수 있을 것이다. 일단 함수를 다음과 같이 만들어 보자.

```
def GuGu(n):
   print n
```

위와 같은 함수를 만들고 GuGu(2)처럼 하면 2라는 값을 출력하게 된다. 즉 입력으로 2를 받는 것을 확인을 하는 것이다.

다음에는 결과값을 담을 리스트를 하나 생성하자.

```
def GuGu(n):
   result = []
```

다음에는 result에 2, 4, 6,,, 18을 어떻게 넣어 주어야 할지 생각해 보자. 필자는 다음과 같이 하였다.

```
def GuGu(n):
   result = []
   result.append(n*1)
   result.append(n*2)
   result.append(n*3)
   ...
   result.append(n*9)
   return result
```

정말 무식한 방법이지만 입력값 2를 주었을 때 원하는 결과값을 얻을 수 있었다. 하지만 위의 함수는 반복이 너무 많다. 가만히 보니 1부터 9까지의 숫자만이 틀린 것을 볼 수 있지 않은가? 그렇다면 이번에는 1부터 9까지를 출력해 주는 것을 먼저 생각하자.

대화형 인터프리터를 열고 필자는 다음과 같이 테스트 해 보았다.

```
>>> i = 1
>>> while i < 10:
. . . print i
. . . i = i + 1
```

결과는 아주 만족이다. 따라서 위와 같은 것을 GuGu함수에 적용시키기로 결정했다.

이러한 생각을 바탕으로 만든 함수는 다음과 같다.

```
def GuGu(n):
   result = []
   i = 1
   while i < 10:
       result.append(n * i)
       i = i + 1
   return result
```

다음과 같이 테스트를 해 보았다.

```
print GuGu(2)
```

결과는 대 만족이다. 사실 위의 함수는 위와같은 과정을 거치지 않고서도 금방 생각할 수 있는 독자들이 많겠지마 위처럼 간단하지 않다면 필자가 했던 방식이 매우 도움이 된다는 것을 금방 알 수 있을 것이다.

즉, 필자가 강조하고 싶은 것은 프로그래밍이란 것은 위에서 보았듯이 매우 구체적으로 접근해야 머리가 덜 아프다는 얘기이다. 독자들 나름대로의 스타일이 있겠지만 도움이 됐으면 좋겠다.

앞으로 소개할 예제들을 위의 방식으로 설명하지는 않을 것이다. 하지만 필자가 위와 같은 사고 방식으로 아래의 예제들을 만들 수 있었을 거란 생각을 해주기 바란다. 자, 이제 4장에 있는 예제들을 보며 독자들 나름대로 멋진 생각들을 해보기 바란다.

[02] 간단한 메모장

파일에 원하는 메모를 저장하고 수정및 확인을 할 수 있는 간단한 메모장을 만들어 보도록 하자.

```
# memo.py
import sys
import time

def usage():
   print """
Usage
=====
%s -v : View memo
%s -a : Add memo
""" % (sys.argv[0], sys.argv[0])

if not sys.argv[1:] or sys.argv[1] not in ['-v', '-a']:
   usage()
elif sys.argv[1] == '-v':
   try: print open("memo.txt").read()
   except IOError: print "memo does not exist!"
elif sys.argv[1] == '-a':
   word = raw_input("Enter memo: ")
   f = open("memo.txt", 'a')
   f.write(time.ctime() + ': ' + word+'\n')
   f.close()
   print "Added"
```

사용자가 명령행에서 ‘ python memo.py -v ’ 라고 입력하면 지금까지의 메모를 출력해주고 만약 하나도 추가된 것이 없을 때는 "memo does not exist!"라는 문장을 출력해 준다. ‘ python memo.py -a ’ 라고 입력하면 메모를 입력받아서 memo.txt라는 파일에 입력된 내용과 현재의 시간을 함께 파일에 적는다.

사용자가 명령행에서 ‘ python memo.py -v ’ 나 ‘ python memo.py -a ’ 라고 입력하지 않을 경우에는 usage() 함수를 호출한다. sys.argv를 어떻게 활용하는지 그리고 try.. except 구문을 어떻게 활용했는지를 주목해서 보도록 하자.

[03] tab을 4개의 space로 바꾸기

이 스크립트는 한 문서파일을 읽어서 그 문서파일 내에 있는 탭(Tab)을 공백 네 개(4 Space)로 바꾸어 주는 스크립트이다. 필자는 대부분의 파이썬 스크립트를 리눅스의 VI 에디터를 이용하여 작성하는데 들여쓰기를 항상 탭으로 한다. 그런데 이 소스파일을 문서화시킬 때 탭 사이즈가 8이어서 읽기에 좀 불편했었다. 그런 이유로 이런 스크립트를 만들어 보았다.

```
#tabto4.py

import re
import sys

def usage():
   print "Usage: python %s filename" % sys.argv[0]

try: f = open(sys.argv[1])
except: usage(); sys.exit(2)

msg = f.read()
f.close()
p = re.compile(r'\t')
changed = p.sub(" "*4, msg)

f = open(sys.argv[1], 'w')
f.write(changed)
f.close()
```

위 스크립트에서 주목해서 볼 점은 re 모듈로 어떻게 탭을 네 개의 공백으로 바꾸었는가 이다. 우선 p = re.compile(r'\t')로 패턴을 만들었다. 여기서 r'\t'의 의미는 \t탭문자 그대로를 뜻한다. 여기서 r은 raw를 말한다. 이것은 이렇게 해도 무방하다.

```
p = re.compile('\\t')
```

그리고 만들어진 패턴 p의 함수인 sub를 이용해 패턴과 매칭되는 부분을 " "*4 즉, 공백 네 개로 바꾸었다. sub은 패턴과 매치되는 부분을 원하는 문자열로 바꾸어 주는 기능을 가지고 있다.

```
p.sub(" "*4, msg)
```

[04] 12345라는 숫자를 12,345처럼 바꾸기

이 스크립트는 숫자를 나타내는 문자열을 입력받아서 읽기 편한 형식인 123,456처럼 콤마가 섞인 숫자로 바꾸어서 돌려주는 함수이다.

```
# commanumber.py
import string
def comma_number(number):
   if number[0] in ['+', '-']:
       sign_mark, number = number[:1], number[1:]
   else:
       sign_mark = ''
   try:
       tmp = string.split(number, '.')
       num = tmp[0]; decimal = '.' + tmp[1]
   except:
       num = number; decimal = ''
   head_num = len(num) % 3
   result = ''
   for pos in range(len(num)):
       if pos == head_num and head_num:
           result = result + ','
       elif (pos - head_num) % 3 == 0 and pos:
           result = result + ','
       result = result + num[pos]
   return sign_mark + result + decimal

print comma_number("12345678.345678")
```

결과값:

```
12,345,678.345678
```

제일 먼저 '+', '-'가 있는지를 조사하여 있으면 sign_mark라는 변수에 그 기호값을 넣는다. 다음에 '.'이 포함되어 있으면 앞부분과 뒷부분으로 나누어서 앞부분은 ','가 들어갈 숫자부분이고 뒷부분은 소수점 부분이므로 뒷부분을 decimal 변수에 넣는다.

```
    head_num = len(num) % 3
```

'.' 앞부분 즉 콤마가 들어갈 숫자의 개수를 세어서 3으로 나눈값을 말한다.

```
    for pos in range(len(num)):
       if pos == head_num and head_num:
           result = result + ','
       elif (pos - head_num) % 3 == 0 and pos:
           result = result + ','
       result = result + num[pos]
```

이 부분이 핵심이 되는 알고리즘이다. 소수점 앞의 숫자의 길이를 세어서 3으로 나눈 나머지 값을 이용해서 맨 처음 나오는 ',' 앞에 몇 개의 숫자가 들어갈지 알 수 있다. 즉 콤마가 들어갈 숫자(num)의 첫 번째 요소부터 끝까지 for문을 통해 들어오면 그 개수를 세어서 head_num, 즉 총 숫자 개수를 3으로 나눈 나머지값의 갯수와 일치하고 head_num이 0이 아닌 값이면 ','를 삽입한다. 그런 다음에 3개 간격으로 ','를 추가한다. 위의 알고리즘은 얼핏보면 잘 이해가 가지 않는다. 곰곰히 생각해 보도록 하자.

한참 후에 이 코드를 다시 보고 설명을 읽어도 무슨 내용인지 이해가 가지 않았다. 독자들은 위와 같은 코드를 작성하지 말기 바란다. 누가 보아도 알기 쉽고 당연한 코드를 작성할 수 있도록 노력해야 한다. 그렇다면 어떻게 이 프로그램을 좀 더 Simple하게 작성할 수 있을까?

그것은 독자의 몫으로 남길까 한다.
이곳에 댓글로 자신의 코드를 남겨보는 것은 어떨까?

(at 2008.02 by 박응용)

```
import unittest
def comma_number(no):
   t = no.split(".")
   decimal = t[0]
   sosu = ""
   if len(t) > 1: sosu = t[1]
   if sosu:
       return comma(decimal) + "." + sosu
   else:
       return comma(decimal)

def comma(no):
   result = []
   numbers = list(str(no))
```

```
    numbers.reverse()
    for i, n in enumerate(numbers):
        if i%3 == 0 and i: result.insert(0, ",")
        result.insert(0, n)
    return "".join(result)

class CommaTest(unittest.TestCase):
    def test1(self):
        self.assertEquals("", comma_number(""))
        self.assertEquals("1", comma_number("1"))
        self.assertEquals("12", comma_number("12"))
        self.assertEquals("123", comma_number("123"))
        self.assertEquals("1,234", comma_number("1234"))
        self.assertEquals("1,234.02", comma_number("1234.02"))
        self.assertEquals("3,312,345.3234", comma_number("3312345.3234"))

if __name__ == "__main__":
    unittest.main()
```

[05] 하위디렉토리 검색

자신의 PC에서 특정 파일만을 찾아내어 특정 문장이 포함되어 있는 부분을 다른 문구로 수정하여 저장해야 한다고 생각해 보자.(이와 비슷한 상황은 실제 업무에서 매우 빈번하게 발생한다.)

파이썬 프로그래머라면 일일이 파일을 찾은 후에 파일을 열어서 수정한 후에 다시 저장하는 행위를 반복하는 어리석은 짓은 하지 않을 것이다.

다음의 소스를 보자.

```
import os

def search(dirname):
   flist = os.listdir(dirname)
   for f in flist:
       next = os.path.join(dirname, f)
       if os.path.isdir(next):
           search(next)
       else:
           doFileWork(next)


def doFileWork(filename):
   ext = os.path.splitext(filename)[-1]
   if ext == '.py': print filename


search("d:/")
```

위 소스는 재귀호출을 이용하여 특정 디렉토리부터 시작하여 그 하위의 디렉토리 파일등을 검색하기 시작하는 프로그램이다. 만약 디렉토리일 경우에는 다시 search함수를 재귀로 호출하고 파일일 경우에는 doFileWork이라는 함수를 호출한다.

위의 예는 d드라이브 밑에 있는 파일 중 확장자가 .py인 파일을 모두 출력하는 예제이다.
만약 .py라는 확장자를 가진 모든파일에서 "ABC"를 "DEF"로 바꾸려면 doFileWork를 아래처럼 구현하면 될 것이다.

```
def doFileWork(filename):
   ext = os.path.splitext(filename)[-1]
   if ext != ".py": return
   f = open(filename)
```

```
    before = f.read()
    f.close()
    after = before.replace("ABC", "DEF")
    f = open(filename, "w")
    f.write(after)
    f.close()
```

[06] 얼마나 시간이 경과됐을까?

예전에는 게시판 리스트에 작성일이 "2007년 1월 1일" 처럼 항상 "년월일" 형식으로 보여 주었다.
하지만 요새 진보된 게시판 리스트에는 작성일에 이런식으로 표시해 준다.

```
1일 2시간 전
```

또는

```
1시간 48분 전
```

사용자들 위주로 UI들이 변하고 있는것이다. 물론 파이썬에서도 위와 같은 기능을 구현할 수 있다.

다음의 함수를 보자.

```
import datetime
def elapsed_time(sdate):
   e = datetime.datetime.now()
   if not sdate or len(sdate) < 14: return 0,0,0,0
   s = datetime.datetime(int(sdate[:4]), int(sdate[4:6]), int(sdate[6:8]),
       int(sdate[8:10]), int(sdate[10:12]), int(sdate[12:14]))
   days = (e-s).days
   sec = (e-s).seconds
   hour, sec = divmod(sec, 3600)
   minute, sec = divmod(sec, 60)
   return days, hour, minute, sec
```

날짜를 입력으로 받아서 현재 날짜로부터 며칠, 몇시간, 몇분, 몇초가 지났는지를 리턴해 주는 함수이다.
이 프로그램의 핵심은 datetime객체의 "-"연산자이다.

이번에는 오늘로부터 일주일 전의 날짜를 계산하는 방법에 대해서 알아보자.

```
(datetime.datetime.now() - datetime.timedelta(7)).strftime("%Y%m%d")
```

datetime의 now()라는 메써드와 timedelta(일자) 메써드를 이용하여 1주일 전의 datetime값을 구한 후 YYYYMMDD형식으로 리턴하는 예제이다.

여러분이 만약 웹 프로그래밍을 하게 된다면 즐겨 사용하게 될 datetime모듈을 주의깊게 살펴보기로 하자.

07. 테스트 주도 개발

지금까지 파이썬 언어 그 자체에 대해서 공부해 왔다. 이제 파이썬이란 언어를 더욱 깊이 이해하고 활용하는것은 독자의 몫이다. 이곳에서는 파이썬 언어 자체에 대한 것보다는 프로그래밍 언어를 가지고 보다 효율적이고 효과적으로 코딩을 할 수 있는 방법에 대해서 알아볼까 한다.

그 중 가장 강력한 개발 방법인 TDD(Test Driven Development)에 대해서 이야기 한다.

[1] 테스트 주도 개발이란 무엇인가?

수많은 하드웨어와 소프트웨어, 언어와 개발방법론, 프레임워크 등 끊임없이 쏟아져 나오는 IT업계의 신기술은 이 시대의 프로그래머에게 마냥 좋은 소식만은 아닌 것 같다. 프로그래머란 변화를 수용하지 않고는 가까운 미래마저 보장받기 어려운 직업이기 때문이다. 항상 새로운 것을 배우고 개척해야만 뒤쳐지지 않을 것이라는 부담이 우리를 억누르고 있고, 또한 일정한 나이가 지나면 지금껏 쌓아온 지식과 경험이 대부분 쓸모없는 것이 되어 버릴 것이라는 불안감이 늘 함께한다. 필자의 주관적인 느낌이지만, 여러분이 프로그래머라면 공감하는 부분이 분명 있을 것이다.

왜 TDD인가?
필자는 또 하나의 그저 그런 복잡하고 귀찮은 개발 방법론을 소개하려는 것이 아니다. 위와 같이 급변하는 IT환경에서도 변하지 않고 끊임없이 효력을 발휘할 수 있는 " 기민한 코드를 만드는 스타일"에 대해서 이 곳을 통해 얘기하려고 하는 것이다. 아마 경험이 많은 프로그래머라면 "TDD"의 명성에 대해서 한번쯤 들어보았을 것이다. TDD는 Test Driven Development의 줄임말로 한국어로 번역하면 "테스트 주도적 개발"이 된다. 말 그대로 테스트 주도적으로 프로그램을 개발 하는 방법이다.

필자가 TDD를 처음 접하고 느낀 첫 느낌은 "뭐, 이런 게 다 있어? 테스트를 먼저하고 실제 코드를 작성하라니! 웃기는 이야기군!"이었다. " 도대체 실제코드가 없는데 무얼 테스트한다는 말인가! " 하지만 TDD의 진가를 조금 씩 알게 될수록 신세계를 발견하는 것 같은 경이로움을 느꼈다.

> "세상에, 이런 식으로 코딩을 한다면 난 정말 두 다리 뻗고 잠을 잘 수 있을 거야!"

필자는 이곳을 통해 필자에게 큰 깨달음을 주었던 TDD에 대해서 여러분과 함께 생각해 보고 여러분이 그것을 "체험"할 수 있기를 무척이나 희망한다. TDD가 필자에게 주었던 그 "즐거움과 성취감"을 여러분이 조금이라도 느껴볼 수 있다면, 필자는 무한한 보람을 느낄 것이다.

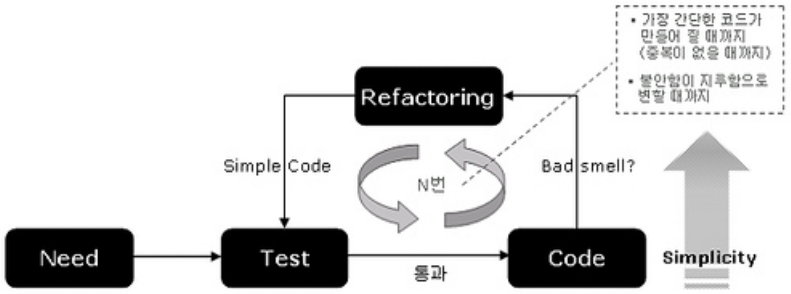
TDD 자세히 알기
전통적으로 우리는 프로그램 개발이 완료된 후에 테스트를 진행한다. 하지만 TDD는 테스트코드를 먼저 작성하고 그 테스트코드를 통과하는 실제코드를 나중에 만든다. 건물을 지을 때 벽돌을 쌓는 방법을 떠올려보자. 벽돌을 쌓을 때는 벽돌을 얼마만큼 쌓을 건지 특정영역에 실로 표시를 해 놓고 벽돌을 쌓다가 실까지 벽돌이 채워지면 쌓는 것을 중지한다.

TDD로 비유하면 공간에 실로 영역을 표시하는 것을 " 테스트코드 " 에, 실제 벽돌을 쌓는 것은 " 실제코드 " 에 비유할 수 있다. 벽돌을 쌓을 때 벽돌이 비뚤어지는지 정확히 쌓이는지 실에 의해서 판단이 가능한 것과 같은 이치로 테스트 코드는 실제 코드가 나아가야 할 방향을 알려주고 있는 것이다.

만약 벽돌이 조금 비뚤어지게 쌓였다면 반듯하게 다시 잡아가게 되는데 이것은 리팩토링에 비유할 수 있겠다. (리팩토링은 소스코드의 기능은 유지한 채로 소스코드의 디자인을 개선해 나가는 방법이다)

TDD의 흐름

TDD에 절대적인 방법이 있는 것은 아니지만, 일반적인 흐름은 있다. 그 흐름은 다음과 같다.



1. 무엇을 테스트할 것인가 생각한다.
2. 실패하는 테스트를 작성한다.
3. 테스트를 통과하는 코드를 작성한다.
4. 코드를 리팩토링한다. (테스트코드 또한 리팩토링한다)
5. 구현해야 할 것이 있을 때까지 위의 작업을 반복한다.

TDD의 목표

Simple Code는 TDD의 궁극적인 목표이다. 켄트벡(Kent Beck, ExtremeProgramming 창안자 중 한 사람. 사람들은 그를 프로그래밍의 달인이라고 부른다.), 워드 커닝험(Ward Cunningham)과 함께 익스트림 프로그래밍(Extreme Programming)의 아버지라 불리는 론 제프리즈(Ron Jeffries)는 Simple Code를 다음과 같이 표현하였다.

> Clean code that works! (작동하는 깨끗한 코드!)

Simple Code는 코드가 단순하다는 의미가 아니라 중복이 없고 누가 봐도 명확한 코드를 말한다. 가끔 프로젝트를 하다보면 정말 뛰어난 프로그래머들을 보게 된다. "오~ 이것을 이렇게 간단하고 이해하기 쉽게 표현하다니!"라는 감탄사와 함께 말이다. 직관력이 뛰어난 프로그래머는 복잡하게 할 것을 아주 간단하게 만들어버리는 능력이 있다. 하지만 TDD를 이용하면 특별한 능력 없이도 자연스럽게 가장 Simple한 코드를 만들어 나가게 된다.

직관력, 또는 갑자기 떠오르는 아이디어가 프로그램을 훌륭하게 만들 수는 있지만 항상 그러리란 보장은 없다. TDD는 그것을 추구해 나가는 가장 현실적인 방법이다.

이제부터 파이썬으로 테스트주도 개발을 해 나가는 방법에 대해서 구체적으로 알아보도록 하자.

[2] PyUnit

pyunit은 파이썬에서 기본적으로 제공하는 라이브러리로 테스트케이스를 만들고 테스트를 진행하는 데 도움을 주는 모듈이다.

가장 간단한 사용법을 먼저보면 다음과 같다.

```
import unittest

class SimpleTest(unittest.TestCase):
   def test1(self):
       self.assertEquals(1, 1)

if __name__ == "__main__":
   unittest.main()
```

아무런 배경지식없이 위 코드를 보면 어리둥절 할 수 있겠지만 자세히 한번 들여다 보자.

1. 우선 pyunit을 사용하기 위해 unittest모듈을 import한다.
2. unittest.TestCase를 상속하여 SimpleTest라는 임의의 클래스를 생성한다.
3. 실제 테스트를 수행할 메써드를 만든다(메써드명:test1, 메써드명은 무조건 test로 시작해야 하는 규칙이 있다. unittest.main()이라는 문장을 통해서 테스트가 진행될 때 unittest모듈은 메써드명이 test로 시작하는것만을 테스트 메써드로 판단하기 때문이다.).
4. self.assertEquals라는 TestCase의 메써드를 이용하여 1이란 값이 1이란 값과 일치하는지를 조사한다.
5. 끝

1과 1을 비교하는것은 의미가 없지만 위 샘플은 unittest를 어떻게 사용하는지 테스트를 어떻게 작성하는지를 보여준 가장 간단한 예시이다.

TestCase에는 assertEquals말고 테스트를 도와주는 더 많은 메써드들이 있다. 몇가지만 알아보도록 하자.

메써드명	설명
assertEquals(x,y)	x와 y의 값이 일치하는지를 조사한다
assertTrue(x)	x가 참인지를 조사한다

assertFalse(x)	x가 거짓인지를 조사한다
fail(msg)	무조건 실패하게 만든다, msg는 출력메시지
failIf(x)	x가 참이면 실패하게 만든다

다음 장에서 pyunit을 이용하여 실제로 TDD를 연습해 보자.

[3] SubDate

Sub Date

두 날짜(YYYYMMDD)의 차이를 구하는 프로그램. 예를 들면 다음과 같다.

```
20030515 sub 20030501 = 14
20030501 sub 20030515 = 14
20030301 sub 20030515 = 31 + 30 + 14
```

이와 같은 프로그램을 TDD를 이용하여 작성해 보자.

어디서부터 시작할 것인가?
이 문제는 아주 쉬워 보이지만 만만하지는 않을듯 하다. 일단 가장 걸림돌이 될만한 것은 윤달에 대한 것이다. 28일도 되었다가 29일도 되는 아주 특이한 것이므로,,

잠시 생각해 본 결과 이문제를 풀기 위한 가장 쉬운 해법은 다음과 같은 것이었다.

```
결과 = absolute(첫번째 날짜의 총 일수 - 두번째 날짜의 총 일수)
```

그렇다, 해당 날짜의 0년부터 지금껏 지나온 총 일수를 구한다면 문제가 쉽게 해결될 듯하다.

TODO

- 특정일자의 총 일수를 구한다.

조금 더 세분화 하여 특정일자의 총 일수를 구하기 위해서 다음의 세가지 항목을 추가하였다.

- 전년도까지의 총 일수를 구한다.
- 전달까지의 총 일수를 구한다.
- 해당일자까지의 총 일수를 구한다.

즉 다음과 같은 의도이다. 20030515라는 날짜의 총 일수를 구하고 싶다면 아래처럼 구할 수 있다는 생각이다.

> 2003년 5월 15일의 총일수 = 2002년까지의 총 일수 + 2003년 1월 부터 4월까지의 총 일수 + 15

TDD Start

TODO

- 두일자(YYYYMMDD)의 차이 일자를 구한다.
- 특정일자의 총 일수를 구한다.
- 전년도까지의 총 일수를 구한다.
- 전달까지의 총 일수를 구한다.
- 해당일자까지의 총 일수를 구한다.

위의 TODO리스트를 기반으로 TDD를 시작해 보자. TODO 리스트에서 전년도까지의 총 일수를 구하는 것을 먼저 하기로 한다. 어떻게 구현할 것인가 보다는 어떻게 테스트를 할 것인지 생각해야 한다.

첫번째 테스트 코드는 다음과 같다.

```
import unittest

class SubDateTest(unittest.TestCase):
   def testGetDayByYear(self):
       subdate = SubDate()
       self.assertEquals(0, subdate.getTotalDayByYear(1))
       self.assertEquals(365, subdate.getTotalDayByYear(2))


if __name__ == "__main__":
   unittest.main()
```

0년이란 것은 존재하지 않기 때문에 1년 까지의 총 일 수는 0이 되어야 한다. (최초 일자를 1년 1월 1일이라고 나름대로 설정한 것이다.) 2년 까지의 총 일 수는 1년 1월 1일부터 2년 1월 1일까지이므로 365일이 될 것이다.

이 테스트를 통과하기 위한 가장 빠른 방법은 다음과 같았다.

```
class SubDate:
   def getTotalDayByYear(self, year):
       if year==1: return 0
       else: return 365
```

다음에 테스트 코드에 다음의 한 줄을 삽입하였다.

```
self.assertEquals(365*3+366, subdate.getTotalDayByYear(5))
```

이 것을 통과하기 위해서는 1년부터 4년까지중 윤년이 있는지 조사해야 하기 때문에 위의 한줄을 잠시 주석처리하고 윤년인지 아닌지를 검사하는 테스트 코드를 작성하기로 한다.

TODO

- 윤년체크

테스트 코드는 다음과 같다.

```
    def testLeapYear(self):
       subdate = SubDate()
       self.assertTrue(subdate.isLeapYear(0))
       self.assertFalse(subdate.isLeapYear(1))
       self.assertTrue(subdate.isLeapYear(4))
```

테스트를 통과하기 위한 가장 빠른 실제코드는 다음과 같았다.

```
class SubDate:
   def getTotalDayByYear(self, year):
       if year==1: return 0
       else: return 365

   def isLeapYear(self, year):
       if (year == 0): return True
       if (year == 1): return False
```

```
        if (year == 4): return True
        return False
```

테스트 코드와 실제코드를 잘 살펴보면 duplication을 발견할 수 있다. 그것은 바로 0, 1, 4라는 숫자이다. 이 숫자를 유심히 관찰하면 다음과 같은 실제 코드로 리펙토링이 가능할 것이다.

```
    def isLeapYear(self, year):
        if year % 4 == 0: return True
        return False
```

4라는 중복 숫자가 남아 있긴 하지만 4라는 숫자는 의미가 있는 숫자이므로 일단은 그대로 놔두기로 한다.

테스트를 계속 하기전에 먼저 윤년이 무엇인지 개념이 확실해야 할 것 같다. 4로 나누어서 떨어지는 연도가 윤년이라는 것은 들어서 알고 있었지만 확실치 않으므로 윤년의 정의를 찾아보았다. 역시 다른것들이 있었다.

윤년의 정의는 다음과 같다.

> 400으로 나누어 떨어지는 년도는 윤년이다.
> 100으로 나누어 떨어지는 년도는 윤년이 아니다.
> 4로 나누어 떨어지는 년도는 윤년이다.
> 위에 속하지 않는 년도는 모두 윤년이 아니다.

즉 해석해 보면 1200년은 400으로 나누어 떨어지고 100으로도 나누어 떨어지지만 400을 먼저 생각하기 때문에 윤년이다. 700년은 100으로 나누어 떨어지기 때문에 윤년이 아니다. 즉 400, 100, 4라는 우선순위를 적용시켜야 한다는 점이다.

이것을 나타날 낼 수 있는 테스트 코드를 작성해 보면,

```
    def testLeapYear(self):
        subdate = SubDate()
        self.assertTrue(subdate.isLeapYear(0))
        self.assertFalse(subdate.isLeapYear(1))
```

```
        self.assertTrue(subdate.isLeapYear(4))
        self.assertTrue(subdate.isLeapYear(1200))
        self.assertFalse(subdate.isLeapYear(700))
```

컴파일은 되지만 self.assertFalse(subdate.isLeapYear(700))에서 실패한다.

테스트를 통과하기 위해서는 실제코드를 수정해야만 한다.

```
    def isLeapYear(self, year):
        if year % 100 == 0: return False
        if year % 4 == 0: return True
        return False
```

실제 코드를 위와 같이 수정하니 이번에는 assertTrue(subdate.isLeapYear(1200))에서 실패한다.
그래서 실제 코드는 다음과 같이 바뀌어야 했다.

```
    def isLeapYear(self, year):
        if year % 400 == 0: return True
        if year % 100 == 0: return False
        if year % 4 == 0: return True
        return False
```

이제 모든 테스트가 통과되었다.

다음 테스트로 넘어가기 전에 한가지 유념해야 할 것이 있다. 윤년을 체크하는 테스트 코드가 믿음을 주는가? 이다. 위의 테스트 코드로 윤년을 조사하는 로직이 완벽하다고 느낄 수 있냐는 점이다.
완벽하지 않다고 느낀다면 확신할 수 있는 테스트 코드를 작성하는 것이 좋다. 우리는 다음으로 그냥 넘어가도록 하자. (충분히 안정적이라고 생각한다. 본인은...)

이제 첫번째 테스트 코드에 주석으로 막아 놓았던 문장을 풀어보도록 하자.

```
#self.assertEquals(365*3+366, subdate.getTotalDayByYear(5))
```

테스트는 실패할 것이다,

우리는 미리 만들어 놓았던 윤년 체크 로직을 이용하여 쉽게 통과할 수 있을 것이다.

```
    def getTotalDayByYear(self, year):
       result = 0
       for i in range(1, year):
           if self.isLeapYear(i): result += 366
           else: result += 365
       return result
```

getTotalDayByYear라는 메써드명이 해당년까지의 총일수인지 전년도까지의 총일수인지를 명확하게 알려주지 못하므로 이름을 다음과 같이 바꾸어 주었다.

```
getTotalDayByYear => getTotalDayLessThanYearOf
```

자, 이젠 전월까지의 총일수를 구해보도록 하자. 테스트 코드는 다음과 같다.

```
    def testGetDayByMonth(self):
       subdate = SubDate()
       self.assertEuqlas(0, subdate.getTotalDayLessThanMonthOf(1))
       self.assertEquals(31, subdate.getTotalDayLessThanMonthOf(2))
```

테스트 코드를 통과한 실제 코드는 다음과 같다.

```
    def getTotalDayLessThanMonthOf(self, month):
       if month == 1: return 0
       else: return 31
```

데이터의 중복을 제거하기 전에 한가지 간과한 사실이 있다. 2월, 즉 윤달에 대한 처리를 어떻게 할 것인가? 하는 점이다. 테스트 코드에 그 의도를 담아보면,

```
    def testGetDayByMonth(self):
       subdate = SubDate()
       self.assertEquals(0, subdate.getTotalDayLessThanMonthOf(1, True))
       self.assertEquals(31, subdate.getTotalDayLessThanMonthOf(2, False))
```

의도는 2월이 윤달인지 아닌지를 파라미터로 보내겠다는 의도이다. 실제 코드는 다음과 같이 변해야 한다.

```
    def getTotalDayLessThanMonthOf(self, month, isLeap):
       if month == 1: return 0
       else: return 31
```

이제 테스트 코드에 윤달 부분을 테스트할 수 있도록 추가해 보자.

```
    def testGetDayByMonth(self):
       subdate = SubDate()
       self.assertEquals(31+28, subdate.getTotalDayLessThanMonthOf(3, False))
       self.assertEquals(31+29, subdate.getTotalDayLessThanMonthOf(3, True))
```

3월 전달까지의 총일수를 윤달이 낀 경우와 아닌경우를 테스트하는 것이다. 물론 이 테스트는 실패할 것이다.

테스트를 통과하기 위해서 실제코드는 다음과 같이 변할 것이다.

```
    def getTotalDayLessThanMonthOf(self, month, isLeap):
       monthDays = [31,28,31,30,31,30,31,31,30,31,30,31]
       result = 0
       for i in range(1, month):
           if isLeap and i==1:
               result += monthDays[i-1]+1
           else:
               result += monthDays[i-1]
       return result
```

다시 다음과 같이 리펙토링 하였다.

```
    monthDays = [31,28,31,30,31,30,31,31,30,31,30,31]
   def getTotalDayLessThanMonthOf(self, month, isLeap):
       result = 0
       for i in range(1, month):
```

```
            result += self.monthDays[i-1]
        if isLeap and month > 2: result += 1
        return result
```

monthDays는 변하지 않는 값이므로 global한값을 주었고, 루프에서 매번 윤달을 체크하던것을 루프를 다 돌고 난 다음에 한번만 체크하도록 하였다.

자 이젠 전달까지의 총일수를 구했으므로 일자의 총 일자를 구하면 될 것이다. 하지만 일자의 총 일자는 일자값 그 자체이므로 테스트가 필요없다고 느껴진다.

이제 어떤 날짜의 총일자를 구해보도록 하자.

```
    def testGetTotalDay(self):
        subdate = SubDate()
        self.assertEquals(1, subdate.getTotalDayOf("00010101"))
        self.assertEquals(366, subdate.getTotalDayOf("00020101"))
```

1년 1월 1일의 총 일수는 1일이 될것이고, 2년 1월 1일은 366일이 되어야 한다.

테스트를 통과하기 위해서 실제코드를 작성해 보면,

```
    def getTotalDayOf(self, date):
        year = int(date[:4]) month = int(date[4:6])
        day = int(date[6:])
        return self.getTotalDayLessThanYearOf(year) + \
            self.getTotalDayLessThanMonthOf(month, self.isLeapYear(year))+day
```

자 이제 마지막 두 날짜의 차이 일자를 구하는 우리의 마지막 테스트가 남아 있다. 테스트 코드는 명확하다.

```
    def testSubDate(self):
        subdate = SubDate()
        self.assertEquals(1, subdate.getSubDate("20021231", "20030101"))
        self.assertEquals(31+28+30+31+14, subdate.getSubDate("20030101", "20030515"))
        self.assertEquals(31+29+30+31+14, subdate.getSubDate("20040101", "20040515"))
```

2002년12월31일과 2003년1월1일의 차이 일자는 1일이다. 테스트를 통과하려면,

```
    def getSubDate(self, date1, date2):
        return abs(self.getTotalDayOf(date1) - self.getTotalDayOf(date2))
```

오류없이 모든 테스트가 통과될 것이다.

Congratulations!!

최종 테스트 코드

```
import unittest

class SubDateTest(unittest.TestCase):
    def testGetDayByYear(self):
        subdate = SubDate()
        self.assertEquals(0, subdate.getTotalDayLessThanYearOf(1))
        self.assertEquals(365, subdate.getTotalDayLessThanYearOf(2))
        self.assertEquals(365*3+366, subdate.getTotalDayLessThanYearOf(5))

    def testLeapYear(self):
        subdate = SubDate()
        self.assertTrue(subdate.isLeapYear(0))
        self.assertFalse(subdate.isLeapYear(1))
        self.assertTrue(subdate.isLeapYear(4))
        self.assertTrue(subdate.isLeapYear(1200))
        self.assertFalse(subdate.isLeapYear(700))

    def testGetDayByMonth(self):
        subdate = SubDate()
        self.assertEquals(31+28, subdate.getTotalDayLessThanMonthOf(3, False))
        self.assertEquals(31+29, subdate.getTotalDayLessThanMonthOf(3, True))

    def testGetTotalDay(self):
        subdate = SubDate()
        self.assertEquals(1, subdate.getTotalDayOf("00010101"))
        self.assertEquals(366, subdate.getTotalDayOf("00020101"))

    def testSubDate(self):
        subdate = SubDate()
        self.assertEquals(1, subdate.getSubDate("20021231", "20030101"))
```

```
        self.assertEquals(31+28+30+31+14, subdate.getSubDate("20030101", "20030515"))
        self.assertEquals(31+29+30+31+14, subdate.getSubDate("20040101", "20040515"))

if __name__ == "__main__":
    unittest.main()
```

최종 실제코드

```
class SubDate:
    def getTotalDayLessThanYearOf(self, year):
        result = 0

        for i in range(1, year):
            if self.isLeapYear(i): result += 366
            else: result += 365
        return result

    def isLeapYear(self, year):
        if year % 400 == 0: return True
        if year % 100 == 0: return False
        if year % 4 == 0: return True
        return False

    monthDays = [31,28,31,30,31,30,31,31,30,31,30,31]
    def getTotalDayLessThanMonthOf(self, month, isLeap):
        result = 0
        for i in range(1, month):
            result += self.monthDays[i-1]
        if isLeap and month > 2: result += 1
        return result

    def getTotalDayOf(self, date):
        year = int(date[:4])
        month = int(date[4:6])
        day = int(date[6:])
        return self.getTotalDayLessThanYearOf(year) + \
            self.getTotalDayLessThanMonthOf(month, self.isLeapYear(year))+day

    def getSubDate(self, date1, date2):
        return abs(self.getTotalDayOf(date1) - self.getTotalDayOf(date2))
```

[4] 미니 웹 서버

이번에는 HTTP프로토콜을 흉내내는 웹서버를 만들어 볼까 한다.

HTTP프로토콜은 우리가 일반적으로 부르는 www(World Wide Web)에서 사용하는 프로토콜로,
www.naver.com, www.google.com등이 HTTP프로토콜을 사용한 웹 서버의 예라고 할 수 있다.

다음과 같은 역할을 하는 프로그램을 만들 것이다.

- 미니웹서버는 웹브라우저(익스플로러 or firefox등)의 요청을 받아들인다.
- 미니웹서버는 웹브라우저로 응답을 보낸다.
- 미니웹서버는 웹브라우저에게 응답을 보낸후 웹브라우저와의 연결을 끊는다.

어디서부터 시작할까?
어떤것을 만들어야 할지 목표가 분명하므로 테스트 코드로 시작하자. 파이썬에서 테스트를 작성하는 기본 구조는 다음과 같다.

```
import unittest


if __name__ == "__main__":
   unittest.main()
```

unittest를 import하고 main()함수로 테스트를 시작하는 것이다.

이제 테스트를 추가해 보도록 하자.

```
class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
```

test1이라는 테스트를 하나 생성했고 MiniWeb이란 클래스의 인스턴스를 생성하겠다는 의도이다.
테스트를 수행(파이썬실행)해 보면 MiniWeb클래스가 없다는 에러가 발생한다.

그렇다면 MiniWeb클래스를 만들도록 하자.

```
class MiniWeb:
   def __init__(self, port):
       self.port = port


class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
```

미니웹서버의 시작과 정지
이제 미니웹서버를 구동할 차례이다. 미니웹서버는 8080포트로 요청을 기다리도록 만들자.

```
class MiniWeb:
   def __init__(self, port):
       self.port = port

   def start(self):
       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       s.bind(("localhost", self.port))
       s.listen(1)
       conn, addr = s.accept()


class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
       miniweb.start()
```

미니웹에 start라는 함수를 생성하고 socket모듈을 이용하여 8080포트로 요청을 기다리게 하였다.
테스트를 수행해 보면 실행은 되지만 더 이상 진행이 되지 않는다.

원인은 무엇일까?
s.accept()함수는 요청이 들어올 때까지 영원히 기다리므로 s.accept()이후는 진행이 되지 않는다.

그렇다! Thread가 필요한 시점이다.

```
import unittest
import socket
import threading

class MiniWeb(threading.Thread):
   def __init__(self, port):
       threading.Thread.__init__(self)
       self.port = port

   def run(self):
       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       s.bind(("localhost", self.port))
       s.listen(1)
       conn, addr = s.accept()


class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
       miniweb.start()


if __name__ == "__main__":
   unittest.main()
```

쓰레드를 사용하기 위해 import threading을 하고 미니웹 클래스가 Thread클래스를 상속하도록 수정한 후 start함수의 이름을 run으로 바꾸어 주었다. 왜냐하면 Thread클래스는 start()함수를 수행하면 run()함수가 자동으로 수행되기 때문이다.

테스트를 수행해 보면 테스트는 무사히 진행이 되나 끝나지를 않는다.

accept()함수는 다른 쓰레드에 의해서 실행되므로 이전과 같은 멈춤현상은 발생하지 않지만 accept()가 종료되지 않고 계속 기다리고 있기 때문이다. 당연한 결과다.

자! 이제 미니웹서버를 중지해야 할 시점이다.

```
import unittest
import socket
import threading

class MiniWeb(threading.Thread):
   def __init__(self, port):
       threading.Thread.__init__(self)
```

```
        self.port = port
        self.s = None

    def run(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s.bind(("localhost", self.port))
        self.s.listen(1)
        conn, addr = self.s.accept()

    def stop(self):
        if self.s: self.s.close()

class TestMiniWeb(unittest.TestCase):
    def test1(self):
        miniweb = MiniWeb(port=8080)
        miniweb.start()
        miniweb.stop()


if __name__ == "__main__":
    unittest.main()
```

위와 같이 stop함수를 생성하여 미니웹서버를 종료하려고 해도 이전과 동일하게 미니웹서버가 종료되지 않는다.

start와 동시에 stop이 일어나서 발생하는 간섭현상이란 판단하에 다음과 같이 start와 stop사이에 간격을 두었더니 정상적으로 종료되는 것을 확인할 수 있었다.

```
import time
...

class TestMiniWeb(unittest.TestCase):
    def test1(self):
        miniweb = MiniWeb(port=8080)
        miniweb.start()
        time.sleep(0.5)
        miniweb.stop()
```

정상적으로 종료는 되지만 Interupt관련한 에러가 발생한다. 에러처리를 하고 완벽한 쓰레드의 종료를 위해 self.join()을 했다. 또한 웹서버는 계속해서 동작해야 하기 때문에 while문을 추가하여 무한루프를 돌게 하였다.

다음은 지금까지 진행한 전체 코드이다.

```
import unittest
import socket
import threading
import time

class MiniWeb(threading.Thread):
   def __init__(self, port):
       threading.Thread.__init__(self)
       self.port = port
       self.s = None

   def run(self):
       self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       self.s.bind(("localhost", self.port))
       self.s.listen(1)
       while 1:
           try:
               conn, addr = self.s.accept()
               conn.close()
           except socket.error:
               break

   def stop(self):
       if self.s: self.s.close()
       self.join()

class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
       miniweb.start()
       time.sleep(0.5)
       miniweb.stop()


if __name__ == "__main__":
   unittest.main()
```

요청을 보내보자

이제 미니웹서버는 시작과 중지가 가능하며 요청을 받아들일 준비가 되어 있다. 테스트코드로 요청을

보내보도록 하자.

```
class TestMiniWeb(unittest.TestCase):
   def test1(self):
       miniweb = MiniWeb(port=8080)
       miniweb.start()

       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       s.connect(("localhost", 8080))
       s.send("abc")
       s.close()

       time.sleep(0.5)
       miniweb.stop()
```

클라이언트 소켓을 하나 생성하여 미니웹서버로 "abc"라는 문자열을 보내고 종료해 보았다. 이상없이 동작하는것을 확인할 수 있었다.

다음 테스트를 진행하기 전에 테스트의 편의와 중복을 제거하기 위해 setUp과 tearDown함수를 이용하여 테스트코드를 리팩토링 하였다.

```
class TestMiniWeb(unittest.TestCase):
   def setUp(self):

       self.server = MiniWeb(port=8080)
       self.server.start()

   def tearDown(self):
       time.sleep(0.5)
       self.server.stop()

   def test1(self):
       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       s.connect(("localhost", 8080))
       s.send("abc")
       s.close()
```

응답을 보내보자
응답은 웹 브라우저가 해석할 수 있는 형태로 하도록 하자. 웹 브라우저가 해석할 수 있는 형태란

HTTP프로토콜 규약을 따르는 응답을 말한다. 가장 간단한 형태의 응답은 다음과 같이 할 수 있을 것이다.

```
HTTP/1.1 200 OK
Server: SimpleHttpServer
Content-type: text/plain
Content-Length: 2

HI
```

위 프로토콜의 규약대로 응답을 하도록 하고 클라이언트가 요청한 문자열을 "HI"대신 삽입하여 응답을 해 보도록 하자.

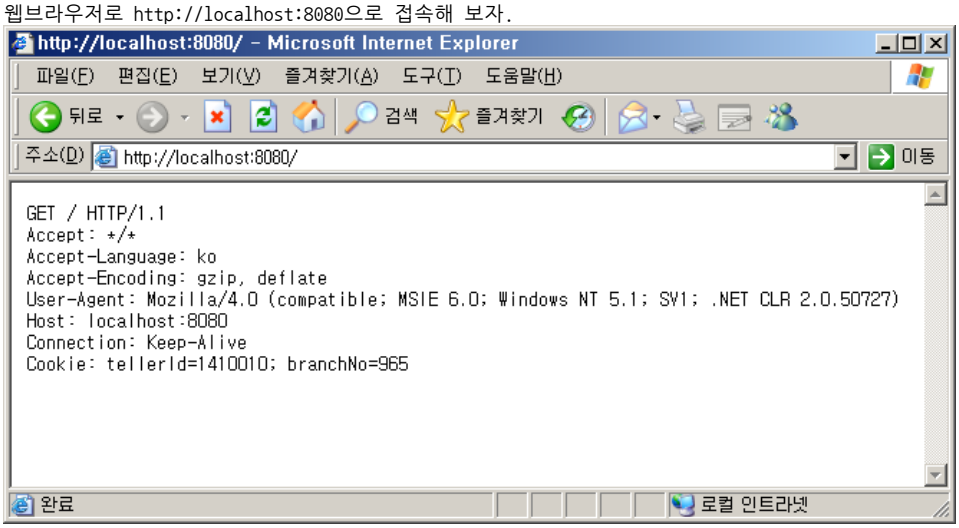
```
...
   def run(self):
       self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       self.s.bind(("localhost", self.port))
       self.s.listen(1)
       while 1:
           try:
               conn, addr = self.s.accept()
               recvmsg = conn.recv(1024)
               conn.send(self.simpleResponse(recvmsg))
               conn.close()
           except socket.error:
               break

   def simpleResponse(self, msg):
       return """HTTP/1.1 200 OK
Server: SimpleHttpServer
Content-type: text/plain
Content-Length: %s

%s""" % (len(msg), msg)
...
   def test1(self):
       s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       s.connect(("localhost", 8080))
       s.send("abc")
       self.assertEquals(self.server.simpleResponse("abc"), s.recv(1024))
       s.close()
...
```

웹브라우저로 테스트하기
미니 웹 서버는 가장 간단한 형태의 응답까지 줄 수 있는 수준이 되었다. 이제 최종적으로 미니웹서버를 데몬으로 띄우고 웹브라우저로 접속해 보도록 하자.

데몬으로 미니웹서버 띄우기

```
if __name__ == "__main__":
   #unittest.main()
   MiniWeb(port=8080).start()
```

웹브라우저로 http://localhost:8080으로 접속해 보자.



웹브라우저에 표시되는 문자들은 웹브라우저가 미니웹에 요청한 데이터들이다.

최종코드

```
miniweb.py

import unittest
```

```
import socket
import threading
import time

class MiniWeb(threading.Thread):
   def __init__(self, port):
       threading.Thread.__init__(self)
       self.port = port
       self.s = None

   def run(self):
       self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
       self.s.bind(("localhost", self.port))
       self.s.listen(1)
       while 1:
           try:
               conn, addr = self.s.accept()
               recvmsg = conn.recv(1024)
               conn.send(self.simpleResponse(recvmsg))
               conn.close()
           except socket.error:
               break

   def simpleResponse(self, msg):
       return """HTTP/1.1 200 OK
Server: SimpleHttpServer
Content-type: text/plain
Content-Length: %s

%s""" % (len(msg), msg)

   def stop(self):
       if self.s: self.s.close()
       self.join()

class TestMiniWeb(unittest.TestCase):
   def setUp(self):
       self.server = MiniWeb(port=8080)
       self.server.start()

   def tearDown(self):
       time.sleep(0.5)
       self.server.stop()
```

```
    def test1(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("localhost", 8080))
        s.send("abc")
        self.assertEquals(self.server.simpleResponse("abc"), s.recv(1024))
        s.close()


if __name__ == "__main__":
    #unittest.main()
    MiniWeb(port=8080).start()
```

08. 인터넷 프로그래밍

예전과 달리 이젠 컴퓨터에 랜선(LAN Cable)을 꼽지 않으면 할 게 없다. 아니 무얼 하려고 해도 할 수가 없다. 디스크 드라이브에 디스켓을 삽입하여 프로그램을 사용하던 시절과는 달리 이젠 인터넷 없이는 살기 힘든 세상이다.

인터넷이란 것은 모든것을 연결시켜준다는 의미에서 무한한 잠재력이 있다. 인터넷이 연결된 컴퓨터 앞에 앉아 있으면 어디라도 갈 수 있고, 누구와도 의사소통을 할 수 있지 않은가? 우린 참으로 놀라운 세상에 살고 있다.

image 출처: http://ds.informatik.rwth-aachen.de/events/sciencesummer06/

지금까지는 인터넷을 사용하기 위해 누군가 만들어 놓은 잘 닦인 프로그램들(익스플로러등)을 사용해왔지만, 파이썬을 통해 여러분도 나름대로의 새롭고 창조적인 길을 개척해 보길 바란다.

[1] 데이터베이스

이 곳에서는 MySQL 데이터베이스를 사용할 수 있게 해주는 MySQL에 대한 파이썬 바인딩중 하나인 MySQLdb모듈에 대해서만 다룰 것이다. 또한 이미 MySQL이 설치되어져 있는 시스템이고 독자가 기본적인 MySQL 명령에 대해서 알고 있다고 가정을 하고 설명할 것이다. 만약 MySQL에 대해서 잘 모른다면 관련 서적을 참고 하도록 하자.

MySQLdb 모듈 설치하기
MySQLdb모듈을 설치하기 이전에 Mysql패키지가 이미 컴퓨터에 설치되어 있어야 한다. 이것에 대한 내용은 이 책에서 다루지 않는다. 윈도우즈 사용자라면 www.mysql.com에서 최신 버전을 다운받아 설치하는 것이 아주 쉬울 것이다. 보통 C:\mysql이란 디렉토리에 기본적으로 설치되고 mysql 데몬을 구동시키기 위해서 도스창에서 다음과 같이 입력하기만 하면 된다.

```
C:\mysql\bin\mysqld
```

다음으로 파이썬에서 mysql을 사용하기 위해서 MySQLdb라는 모듈을 설치해야 한다. 다음을 따라해 가며 설치를 하자.

윈도우즈

다음의 MySQLdb 홈페이지에 가면 윈도우 버젼을 찾아서 받을 수가 있다.

- MySQLdb 홈페이지 - http://sourceforge.net/projects/mysql-python/files

windows installer라는 링크를 클릭하면 Gerhard's Homepage로 가게 된다. 그곳에서 최신 윈도우용 MySQLdb모듈을 다운받도록 하자. 도스창을 열고 MySQLdb-python-0.3.5-win32-2.zip(이책을 쓸 당시의 최신버젼)압축을 풀은 디렉토리로 가서 다음의 명령을 실행하자.

```
C:\temp\mysqldb-python-0.3.5>> python setup.py install
```

위의 과정으로 모든 설치가 마무리 될 것이다.

다음과 같이 했을때 아무런 메시지가 나오지 않는다면 설치에 성공을 한 것이다.

```
>>> import MySQLdb
>>>
```

리눅스

아래에서 최신 MySQLdb모듈을 다운 받도록 하자.

- MySQLdb 홈페이지 - http://sourceforge.net/projects/mysql-python

이 책을 쓰는 지금의 최신 버젼은 다음과 같다.

- MySQL-python-0.3.5.tar.gz

다음과 같은 순서로 설치를 하자.

1. $ tar xvzf MySQL-python-0.3.5.tar.gz
2. $ cd MySQL-python-0.3.5
3. $ python setup.py build
4. # python setup.py install

마지막 install 명령은 루트 퍼미션으로 해 주어야 한다.

설치가 제대로 되었는지 확인을 해보자.

```
>>> import MySQLdb
>>>
```

무소식이 희소식이라고 아무 메시지가 없으면 설치해 성공을 한 것이다.

간단한 데이터 베이스 작성하기

MySQLdb모듈 사용법에 대해서 알아보기 전에 간단한 데이터 베이스를 먼저 작성해 보자.

```
C:\WINDOWS>cd \
C:\>cd mysql
C:\mysql>cd bin
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.38
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql>
```

참고 - 위의 C:\mysql\bin>mysql을 하기 전에 mysqld를 먼저 실행하지 않았다면 mysqld라는 프로그램을 먼저 실행한 후에 mysql을 실행해야만 한다.

우선 book라는 데이터 베이스를 만든다.

```
mysql> CREATE DATABASE book;
Query OK, 1 row affected (0.01 sec)
```

book데이터 베이스를 사용하기 위해 use명령을 사용한다.

```
mysql> use book;
Database changed
```

테이블을 만든다.

```
mysql> create table urllink(category varchar(80), author varchar(80),
   -> subject varchar(80), url varchar(150));
Query OK, 0 rows affected (0.01 sec)
```

위의 테이블은 다음의 용도로 사용될 것이다.

- category : 웹 문서의 종류
- author : 웹 문서 작성자

- subject : 웹 문서 제목
- url : 웹 문서의 URL

```
mysql> describe urllink;
+----------+--------------+------+-----+---------+-------+
| Field        | Type          | Null   | Key |    Default | Extra |
+----------+--------------+------+-----+---------+-------+
| category   | varchar(80)     | YES  |      | NULL    |       |
| author      | varchar(80)     | YES  |      | NULL    |       |
| subject     | varchar(80)     | YES  |      | NULL    |       |
| url           | varchar(150)    | YES  |      | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.05 sec)
```

위와 같은 결과를 출력할 것이다. 그리고는 mysql을 빠져 나가자.

```
mysql> quit
```

위의 과정에 대한 설명은 생략하겠다. mysql문서를 참고하길 바란다.

MySQLdb 모듈 사용하기
우선 MySQLdb모듈을 불러온 다음, connect명령을 이용해서 host에는 localhost를 입력하고 db명은 우리가 만든 book을 입력으로 주어 book 데이터베이스와 연결된 객체 m을 돌려받는다.

```
>>> import MySQLdb
>>> m = MySQLdb.connect(host='localhost', db='book')
```

(참고 - 만약 user, passwd(비밀번호)등을 설정하였다면 다음과 같이 해 주어야 한다

```
m = MySQLdb.connect(host='localhost', user='유저', passwd='비밀번호', db='book')
```

연결 객체인 m의 cursor함수를 이용하여 커서 객체인 c를 만들어 낸다.

```
>>> c = m.cursor()
```

c 라는 커서 객체를 통해서 모든 대화가 이루어지게 될 것이다.

가장 먼저 다음을 입력해 보자.

```
>>> c.execute("select * from urllink")
0L
```

c 커서 객체의 함수인 execute를 이용해서 실제 mysql> 프롬프트에서 실행할수 있는 명령어를 직접 사용할 수가 있다. urllink라는 테이블로 부터 모든것을 검색했지만 입력사항이 하나도 없기 때문에0L을 반환한다. 여기서 0L은 'long int'형이다.

이제 프롬프트 창에서 데이터베이스에 값을 추가해 보도록 하자. 아래와 같은 정보를 추가해 보자.

- category : python
- author : Eung-Yong
- subject : EungYong HomePage
- url : http://tdd.or.kr/tddlog

```
mysql > insert into urllink values ('python', 'Eung-Yong', 'EungYong Homepage', 'http://tdd.or.kr/tddlog'
```

다음엔 입력시킨 데이터 베이스를 파이썬에서 어떻게 얻어낼 수 있는지를 보자. 커서 객체c의 execute함수를 이용하여 urllink테이블의 모든 데이터를 검색하는 명령어를 주었다. 하나의 값이 저장되었기 때문에 1L을 반환하는 것을 확인 할 수 있다.

```
>>> c.execute("select * from urllink")
1L
```

다음에 객체 c의 함수인 fetchall을 이용해서 이전에 실행했던 c.execute("select * from urllink") 의 결과 값을 불러내었다. 반환된 값이 터플의 터플 값임을 주의 깊게 보도록 하자.

```
>>> c.fetchall()
(('python', 'Eung-Yong', 'EungYong HomePage', 'http://tdd.or.kr/tddlog'),)
```

위와 같은 과정이 전형적인 MySQLdb 모듈의 사용법이다. 그 과정을 기술하면 다음과 같다.

1. import MySQLdb로 먼저 모듈을 부른다.
2. MySQLdb.connect 를 이용하여 데이터 베이스 객체를 생성한다.
3. 생성된 객체로 부터 커서 객체를 만든다.
4. 커서 객체를 이용하여 mysql명령을 사용하고 결과값이 있을 때는 fetchall로 결과값을 얻어낸다.

결과값을 한줄씩 얻어내기 위해서는 fetchone이란 함수가 있다. 마치 fetchall은 파일객체의 readlines 메써드와 비슷하고 fetchone은 readline과 비슷한 것이라 생각하면 될 것이다.

이상과 같이 파이썬에서 MySQL 데이터 베이스를 사용하는 방법에 대해서 알아 보았다.

[2] django

"**장고**"라고 읽는다.

자바에 스프링(Spring), 스트러츠(Struts), 루비에 ROR(Ruby On Rails)이 있다면 파이썬에는 django가 있다.
가장 주목받는 파이썬 웹 프레임워크인 django에 대해서 공부해 보자.

이 챕터는 django를 처음 접한 개발자가 도대체 django가 무엇인지 알 수 있게하고 쉽게 django에 다가설 수 있게 하기 위한 안내서이다.

물론 온라인상에 공개되어 있는 장고북(http://www.djangobook.com/en/2.0/)은 매우 훌륭하다. 또한 한글로 번역되어 있는 훌륭한 번역서 또한 존재한다. (배민효님의 "쉽고 빠른 웹개발 django")

그럼에도 불구하고 django에 대한 안내서를 작성하는 이유는 한국 파이썬 개발자가 장고가 무엇인지 간단하게 알아보고 가벼운 마음으로 보다 쉽게 django에 적응하길 희망하는 마음에서이다.

이 챕터는 저자가 django를 배워나가며 알게 된 것을 정리한(정리해 나가는) 책이다. django에 대한 좋은 책을 쓰기 위해 django에 대해서 잘 알아야만 한다는 것은 고정관념이 아닐까? 오히려 모르는 시점부터 차근차근 배워나가는 과정을 적어놓는 것이 django를 처음 접하는 독자에게 더욱 도움이 될것으로 믿는다. 다만 이 책은 온라인 상에 공개되는 책이라 본인이 django를 배워가는 시간과 정리하는 시간이 더디어 지루하게 여겨질까 걱정이 된다.

이 책을 지금 읽고 있고 저자와 함께 django를 공부해 나갈 독자라면 이 곳 위키독스를 통해 궁금한 점을 함께 공유하며 해결해 나가는 것은 어떨까? 저자의 바램이다.

이제 지루한 얘기는 그만하고 당장 django를 공부해 보도록 하자.

01) 따라해 보기

이 책을 읽고 있는 독자라면 django가 무엇인지 들어본 적이 있을 것이다.

아마도 다음과 같은 것이 아닐까?:

- 파이썬으로 만들어진 웹 프레임워크
- MVC모델을 지원한다

아마도 위의 것 이상을 알고 있는 독자라면 지금 이 책을 기술하고 있는 저자보다 훨씬 많은 배경지식을 지니고 있을 것으로 생각된다. 저자가 가지고 있는 **django 이외의 배경지식**은 다음과 같다 :

- 파이썬
- Web에 대한 이해(HTTP프로토콜)
- HTML
- 데이터베이스 사용

만약 위의 배경지식이 없는 독자라면 저자가 기술하는 글들을 어쩌면 쉽게 이해할 수 없을지도 모른다. 하지만 위의 것들을 전혀 모른다면 이것들에 대해서 먼저 공부해야 하지 않을까? 기초없는 부실공사는 오래가지 못한다.

django에 관심을 가진 독자라면 위에서 언급한 것들에 대한 기본적인 지식은 있을거라 희망하며 계속 앞으로 전진해 보도록 하자.

이 글의 진행을 원할히 할 수 있도록 유명한 django book을 참고자료로 활용하자 :

- http://www.djangobook.com/en/2.0/

위 링크를 따라가보면 알겠지만 위키독스처럼 목차도 있고 페이지이동도 쉬운 매우 정성이 들어간 자료임을 금방 알 수 있을 것이다. 영문자료라서 좀 힘이 들겠지만 용기를 내어 도전해 보도록 하자. 저자는 영어를 많이 못하는 편이다. 저자가 잘못 이해하거나 잘못 설명하고 있다면 이곳에 댓글로 혼내주기 바란다.

또한 이곳 위키독스에 게시된 "한날"님의 "날로먹는 Django 웹 프로그래밍"도 함께 참고하도록 하자:

- http://wikidocs.net/mybook/read/index?pageid=4901

먼저 django를 설치해 보자
아래의 링크에서 다운로드가 가능하다:

- http://www.djangoproject.com/download/

이 글을 쓰는 시점에서 다운로드 받을 수 있는 파일은 다음과 같았다:

- Django-1.1.tar.gz

윈도우용 설치 파일은 따로 없는 듯하다. 확장자가 tar.gz이지만 알집이나 토탈 커맨더를 이용한다면 쉽게 압축을 풀 수 있을 것이다.

다음과 같이 아래의 디렉토리에 압축을 풀었다.

```
C:\Django-1.1
```

그리고 설치하기 위해서 다음과 같은 명령을 실행하였다.

```
C:\Django-1.1>c:\python2.6\python setup.py install
```

상당히 많은 파일들이 설치가 되고 성공적으로 끝나는 것을 볼 수 있을 것이다.

참고하는 책을 보니 django-admin.py라는 파일을 이용해서 특정 파일을 자동으로 생성해 주어야 하는 듯 하다. django-admin.py라는 파일은 다음 위치에 복사되어 있는것을 확인할 수 있었다.

```
\python2.6\Lib\site-packages\django\bin
```

이 곳의 파이썬 파일(django-admin.py)을 실행하기 위해서 작업 디렉토리로 C:\work라는 디렉토리를 만든 후 다음과 같이 했다.

```
C:\work>c:\python2.6\python c:\python2.6\Lib\site-packages\django\bin\django-admin.py startproject mysite
```

위 명령을 실행해 보니 c:\work디렉토리 하위에 mysite라는 디렉토리가 자동으로 생성되는 것을 확인할 수 있었다.

디렉토리에는 다음과 같은 파일들이 자동으로 생성되었다.

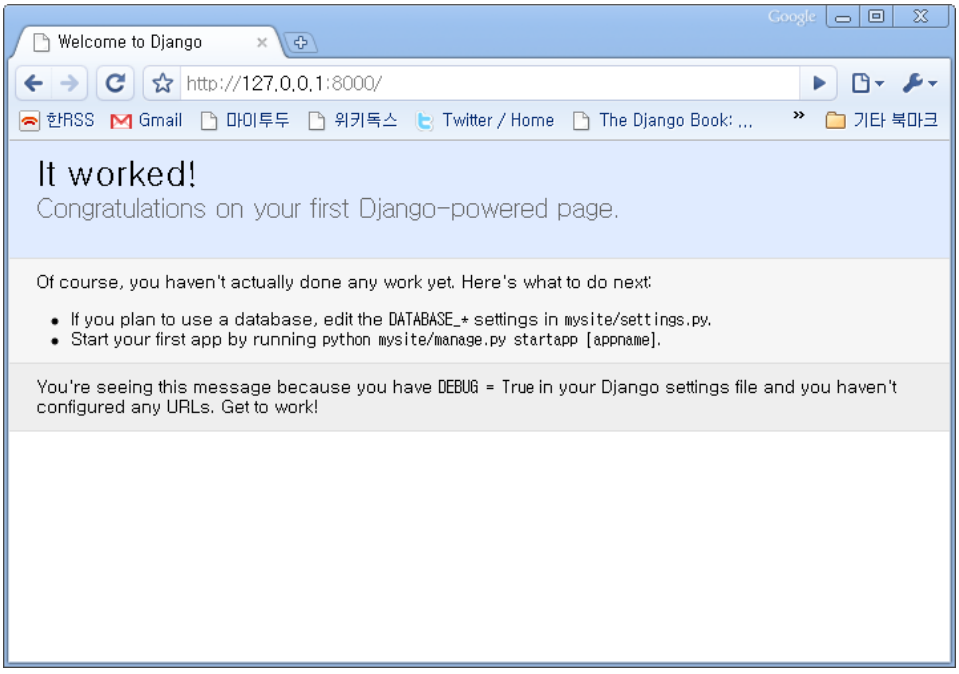
```
2009-07-29  오후 02:09               557 manage.py
2009-07-29  오후 02:09             2,852 settings.py
2009-07-29  오후 02:09               554 urls.py
2009-07-29  오후 02:09                 0 __init__.py
```

이제 드디어 서버를 실행해 볼 수 있게 된 듯 하다. 다음과 같이 서버를 실행해 보자

```
C:\work\mysite>c:\python2.6\python manage.py runserver
Validating models...
0 errors found

Django version 1.1, using settings 'mysite.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

정상적으로 서버가 실행 된 듯 하다.

로그를 보니 127.0.0.1에 8000번 포트로 서버가 떴다고 하니 브라우저로 접속해 보자.



위와 같은 화면을 볼 수 있을 것이다.

이제 Hello World를 출력해 보자.
우선 mysite라는 디렉토리 하위에 views.py라는 파일을 다음과 같이 만든다.

mysite\views.py

```
from django.http import HttpResponse

def hello(request):
   return HttpResponse("Hello world")
```

그리고 mysite디레토리내의 urls.py파일을 다음과 같이 수정한다.

```
from django.conf.urls.defaults import *
from mysite.views import hello

urlpatterns = patterns('',
   ('^hello/$', hello))
```

이렇게 만든 후에 http://127.0.0.1:8000/hello로 접속하면 "Hello World"가 출력되는 것을 확인할 수 있다.

예제를 따라하면서 유추할 수 있는 상황은 다음과 같은 것들이다:

- urls.py 라는 파일은 url규칙을 만들어 내는 파일이다. url규칙은 regex(정규표현식)을 이용한다
- views.py 라는 파일은 요청을 해석하여 응답을 처리하는 파일이다.
- HttpResponse라는 메써드는 화면에 문자열을 출력하는 기능을 가지고 있다.

Hello World까지 출력해 보았다.

02) 템플릿(Templates)

이번에 알아볼 것은 django의 템플릿 기능이다. 템플릿은 HTML페이지 내에서 파이썬 자료형의 값을 **편리하게 표현**하게 해 주는 django의 기법인 듯 하다.

템플릿 파일 작성하고 django에서 읽어보기
django에서 템플릿 파일을 사용하기 위해서는 mysite/settings.py파일을 다음과 같이 수정해야 한다.

mysite/settings.py

```
TEMPLATE_DIRS = (
   'C:/work/mysite/templates',
)
```

위 코드의 의도는 c:/work/mysite/templates라는 디렉토리에 템플릿 파일들을 넣겠다는 의도이다.

위와 같이 설정한 후에 위 디렉토리에 test1.html이라는 파일을 다음과 같이 작성하여 저장해 보자. (한글을 사용했기 때문에 파일은 **반드시 UTF-8 형태로 저장**해야 한다. 그렇지 않을 경우 인코딩 에러가 난다)

mysite/templates/test1.html

```
<html>
<head>
<title>django example #1</title>
</head>
<body>

<p>
한글을 출력해 본다.
</p>

<p>
문자열 대입은 이렇게 {{ message }}
</p>

</body>
</html>
```

그리고 views.py파일을 다음과 같이 수정한다.

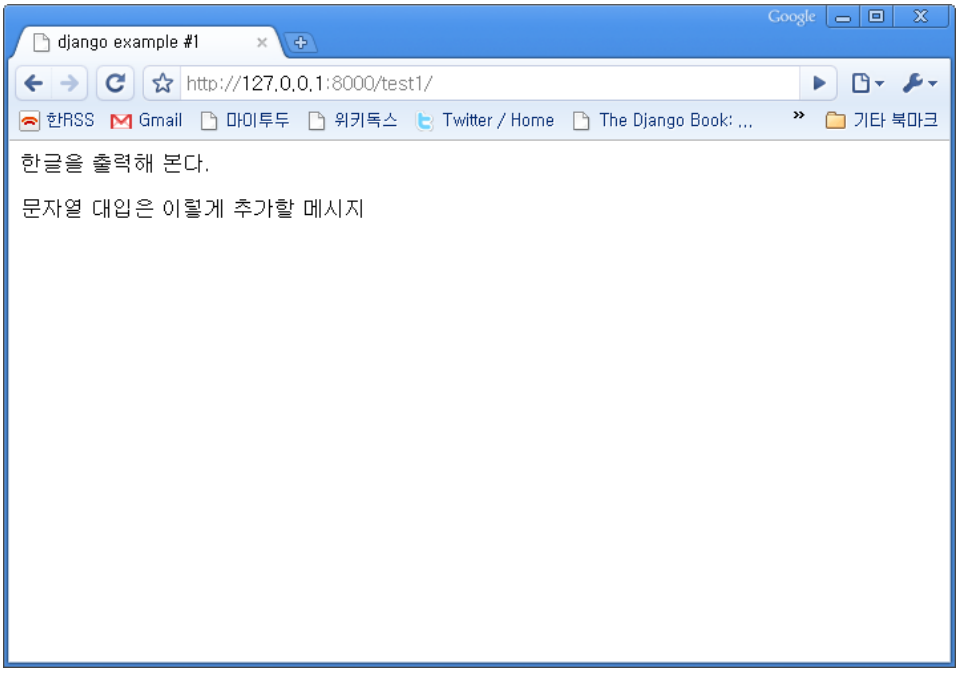
mysite/views.py

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from django.template.loader import get_template
from django.template import Context

def hello(request):
   return HttpResponse("Hello world")

def template_test(request):
   t = get_template('test1.html')
   html = t.render(Context({'message': "추가할 메시지"}))
   return HttpResponse(html)
```

그리고 urls.py파일도 다음과 같이 수정하자.
mysite/urls.py

```
from django.conf.urls.defaults import *
from mysite.views import hello, template_test

urlpatterns = patterns('',
   ('^hello/$', hello),
   ('^test1/$', template_test),)
```

그리고 http://127.0.0.1:8000/test1 으로 접속해 보자.
다음과 같은 결과를 볼 수 있을 것이다.



이상으로 유추할 수 있는 내용은 다음과 같았다:

- 한글 사용은 문제가 없다.
- get_template이라는 메써드를 이용하면 Template 객체를 얻을 수 있다.

  위 get_template은 다음과 동일하다

```
fp = open('c:/work/mysite/templates/test1.html')
t = Template(fp.read())
fp.close()
```

- Context객체를 이용하여 템플릿 파일내의 특정 문자열을 바꿔치기 할 수 있다.

템플릿은 게시판처럼 **반복적인 형태의 데이터**도 쉽게 처리할 수 있는 것 같다.
이것에 대해서도 알아보자.

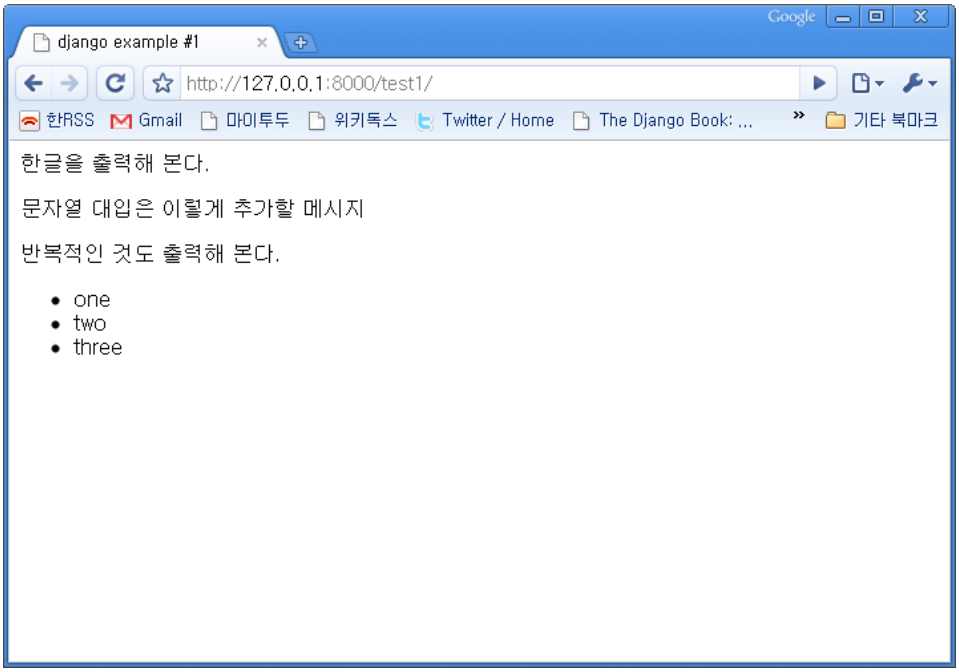
먼저 템플릿 파일을 아래와 같이 수정하자.
mysite/templates/test1.html

```
<html>
<head>
<title>django example #1</title>
</head>
<body>

<p>
한글을 출력해 본다.
</p>

<p>
문자열 대입은 이렇게 {{ message }}
</p>

<p>
반복적인 것도 출력해 본다.
</p>

<ul>
{% for item in item_list %}
 <li>{{ item }}</li>
{% endfor %}</ul>

</body>
</html>
```

그리고 views.py파일을 다음과 같이 수정하자.
mysite/views.py

```
def template_test(request):
   t = get_template('test1.html')
   c = Context({
            'message': "추가할 메시지",
            "item_list":["one", "two", "three"],
       })
   html = t.render(c)
   return HttpResponse(html)
```

그리고 다시 http://127.0.0.1:8000/test1을 실행하면 아래와 같은 결과를 볼 수 있다.



이상으로 알 수 있는 내용은 다음과 같다:

- 반복적인 것을 처리 하기 위해서는 {% for .. %}, {% endfor %}를 이용한다

이상과 같이 템플릿에 대해서 간단하게 알아보았다.

03) 모델(Models)

모델이란 무엇일까?
가장 단순하게 생각해 보면 데이터를 쉽게 다루기 위한 꾸러미 같은 것이다.

예를 들면 "User"라는 테이블에 "이름", "이메일", "주소"라는 컬럼이 있다면 이런 테이블의 데이터를 간편하게 다루기 위해서 아래와 같은 모델이 필요한 것이다.

```
class User:
   def __init__(self):
       self.name = ""
       self.email = ""
       self.address = ""
```

이렇게만 생각하면 모델이란 것은 참 쉬운 개념이다.

그렇다면 django의 모델은 어떤걸까?
데이터베이스에 접속해서 데이터를 가져오고 집어넣고 하는 일반적인 데이터를 처리하는 방법과 django의 가장 큰 차이점은 "쿼리"를 직접 사용하지 않는점을 꼽을 수 있었다. 쿼리를 직접사용하지 않고 django는 도대체 어떻게 데이터를 가져오고 집어넣고 하는 걸까?

당연히 데이터베이스의 SQL을 사용하지 않고서는 데이터베이스와 소통할 수 없다. 그렇다면 django역시 SQL을 사용해야만 한다. 다만 그 SQL을 프로그래머가 직접 작성하는 것이 아니라 **django가 SQL을 자동으로 만들어 주고 있는 것**이다.

하지만 한가지 깊이 생각해야 할 것이 있다. 바로 "**유연성**"이다. 만약 프로그래머가 SQL을 직접 사용할 수 있는 방법이 없다면 여러가지 난관에 봉착할 것이다. SI프로젝트에서 사용하는 업무 SQL들을 생각해 보자. 몇 백 라인되는 SQL을 django처럼 객체화하고 메써드화 할 수 있을까? 또 과연 그렇게 한다고 해서 그게 좋은 코드일까?

하지만 역시 django에서는 쿼리를 직접 사용할 수 있는 방법도 제공하는 것 같다. 다만 django의 쿼리를 대신하는 기능은 단순하지만 반복적인 쿼리문들을 간편하게 대체해 줄 수 있을것 같다.

django의 모델에 대해서 본격적으로 알아보기 전에 선행해야 할 것들이 있다:

1. 로컬PC에서 동작하는 데이터 베이스 설치하기
2. 파이썬과 데이터베이스를 연결할 수 있는 파이썬 모듈 설치하기

이곳에서는 데이터베이스로 **MySQL**을 사용할 예정이다.
MySQL은 아래에서 윈도우버전을 다운로드 받을 수 있다:

- http://dev.mysql.com/get/Downloads/MySQL-5.1/mysql-essential-5.1.36-win32.msi/from/pick#mirrors

MySQL과 파이썬을 연결하는 모듈은 아래에서 다운로드 받을 수 있다.

- http://sourceforge.net/projects/mysql-python/

위 파일들을 다운로드 받고 설치하도록 하자. (설치방법은 생략하도록 한다. 설치에 어려움이 있다면 이곳에 댓글로 남겨주기 바란다)

django & Models
다음과 같은 목표를 가지고 django의 Models를 알아보도록 하자.

우리가 앞으로 이곳에서 해결해야 할 것
우선 다음과 같은 테이블을 만들자:

테이블명 :

- 페이지 - **mypage_page**

컬럼정보 :

- 페이지 아이디 - **pageid** (int)
- 페이지명 - **page_name** (varchar(100))
- 페이지내용 - **page_content** (text)

그리고 이 테이블에 데이터를 인서트 하고 조회하는 프로그램을 작성하도록 하자.

당장 만들어 보자
우선 장고북을 참고해 보니 데이터베이스 **모델을 사용하기 위해서는 app라는 것을 먼저 만들어야 한다는 것**을 알 수 있었다. app를 만들어야만 하는 정확한 이유는 알 수 없지만 그냥 아래와 같이 app라는 것을 만들어 보자.

```
python manage.py startapp mypage
```

위 명령을 수행하니 아래와 같은 파일들이 생성되는 것을 확인할 수 있었다.

```
C:\work\mysite>cd mypage

C:\work\mysite\mypage>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: D075-1035

C:\work\mysite\mypage 디렉터리

2009-08-10  오후 05:18             .
2009-08-10  오후 05:18             ..
2009-08-10  오후 05:18              60 models.py
2009-08-10  오후 05:18             537 tests.py
2009-08-10  오후 05:18              27 views.py
2009-08-10  오후 05:18               0 __init__.py
               4개 파일                624 바이트
               2개 디렉터리   4,267,645,952 바이트 남음
```

mypage라는 디렉토리와 총 4개의 파일이 자동으로 생성되었다.

테이블 생성을 위해서 위에서 자동으로 만들어진 models.py파일을 다음과 같이 수정하도록 하자.

mysite/mypage/models.py

```
from django.db import models

# Create your models here.
class Page(models.Model):
   pageid = models.IntegerField(primary_key=True)
   page_name = models.CharField(max_length=100)
   page_content = models.TextField()
```

[참고] - 장고북에는 위처럼 primary_key라던가 TextField에 대한 설명이 없다. 그래서 실제 소스코드를 살펴보는 수 밖에 없었다. 아래 Field 클래스를 보면 생성자(__init__) 메써드에 primary_key인지에 대한 여부값을 전달받는것을 확인할 수 있었다. 또한 TextField, CharField등 **모든 데이터 클래스가 이 Field클래스를 상속**하여 구현했음을 확인할 수 있었다.

python2.6\Lib\site-packages\django\db\models\fields__init__.py

```
class Field(object):
   # Designates whether empty strings fundamentally are allowed at the
   # database level.
   empty_strings_allowed = True

   # These track each time a Field instance is created. Used to retain order.
   # The auto_creation_counter is used for fields that Django implicitly
   # creates, creation_counter is used for all user-specified fields.
   creation_counter = 0
   auto_creation_counter = -1

   def __init__(self, verbose_name=None, name=None, primary_key=False,
           max_length=None, unique=False, blank=False, null=False,
           db_index=False, rel=None, default=NOT_PROVIDED, editable=True,
           serialize=True, unique_for_date=None, unique_for_month=None,
           unique_for_year=None, choices=None, help_text='', db_column=None,
           db_tablespace=None, auto_created=False):
```

또한 데이터베이스를 사용하기 위해서 settings.py도 수정해야 한다.

mysite/settings.py

```
DATABASE_ENGINE = 'mysql'           # 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'
DATABASE_NAME = 'django'             # Or path to database file if using sqlite3.
DATABASE_USER = 'root'             # Not used with sqlite3.
DATABASE_PASSWORD = '1'         # Not used with sqlite3.
DATABASE_HOST = ''             # Set to empty string for localhost. Not used with sqlite3.
DATABASE_PORT = ''             # Set to empty string for default. Not used with sqlite3.

INSTALLED_APPS = (
   #'django.contrib.auth',
   #'django.contrib.contenttypes',
```

```
   #'django.contrib.sessions',
   #'django.contrib.sites',
   'mysite.mypage',
)
```

mysql을 사용하기 위한 설정인듯 하다. mysql인 경우는 위와 같이 설정할 수 있다. 데이터베이스 명과, 사용자, 패스워드는 자신의 설정과 맞게끔 수정하도록 하자. INSTALLED_APPS에는 mysite.mypage를 꼭 추가해 주어야만 한다.

그리고 다음의 명령을 수행해 보자.

```
C:\work\mysite>c:\python2.6\python manage.py sqlall mypage
c:\python2.6\lib\site-packages\MySQLdb\__init__.py:34: DeprecationWarnin
g: the sets module is deprecated
 from sets import ImmutableSet
BEGIN;
CREATE TABLE `mypage_page` (
   `pageid` integer NOT NULL PRIMARY KEY,
   `page_name` varchar(100) NOT NULL,
   `page_content` longtext NOT NULL
)
;
COMMIT;

C:\work\mysite>
```

manage.py의 sqlall명령을 이용하면 위와 같이 SQL문이 자동으로 생성되어 출력되는 것을 확인할 수 있다.

위 SQL을 긁어서 직접 수행할 수도 있겠지만 django는 조금은 더 편리한 방법을 제공해 준다.

```
C:\work\mysite>c:\python2.6\python manage.py syncdb

c:\python2.6\lib\site-packages\MySQLdb\__init__.py:34: DeprecationWarning: the sets module is deprecated
 from sets import ImmutableSet
Creating table mypage_page
```

위처럼 syncdb명령을 사용했더니 자동으로 테이블이 생성되어 버렸다.

```
mysql> desc mypage_page;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| pageid       | int(11)      | NO   | PRI |         |       |
| page_name    | varchar(100) | NO   |     |         |       |
| page_content | longtext     | NO   |     |         |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

이제 자동으로 생성된 테이블에 한건의 데이터를 인서트 해 보도록 하자. 장고는 파이썬 인터프리터로 직접 테스트를 할 수 있는 매우 편리한 방법을 제공한다. (장고는 정말 멋지다!)

아래의 명령을 수행해 보자.

```
C:\work\mysite>c:\python2.6\python manage.py shell
```

그러면 아래와 같은 장고 shell이 나타날 것이다.

```
c:\python2.6\lib\site-packages\MySQLdb\__init__.py:34: DeprecationWarning: the sets module is deprecated
 from sets import ImmutableSet
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] onwin32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

이제 장고 shell을 이용하여 데이터를 한건 인서트 해 보자.

```
from mypage.models import Page
>>> p1 = Page(pageid=1, page_name='test page', page_content='test page content')
>>> p1.save()
```

Page모델 객체의 save란 메써드를 이용하여 데이터 입력이 가능하다. 실제로 데이터베이스에 저장되었는지를 확인해 보았다.

```
mysql> select * from mypage_page;
+--------+-----------+-------------------+
| pageid | page_name | page_content      |
+--------+-----------+-------------------+
|      1 | test page | test page content |
+--------+-----------+-------------------+
1 row in set (0.00 sec)
```

잘 들어가 있는것이 확인된다.

만약 p1.save()라는 메써드를 한번 더 실행하면 어떻게 될까?
예상대로 데이터 중복에러가 날까?
궁금하다.

아래와 같이 p1.save()를 다시 실행해 보자.

```
>>> p1.save()
```

다시 실행해 보았더니 아무런 반응이 없다. 데이터베이스도 확인해 보니 아무런 값도 입력이 되지 않았다. 왜 그런지에 대해서는 장고북에 나오지 않는것 같다. 한번 save라는 메써드가 호출이 되면 두번 세번호출되더라도 무시되는 걸까? 일단 궁금증은 뒤로한채 새로운 모델을 만들어서 다시 save()를 호출해 보도록 하자.

p1객체와 동일한 내용을 가지고 있는 p2객체를 만들고 save해 보았다. 예상대로라면 중복키 에러가 발생해야 하지만 장고는 또다시 "**침묵**"한다.

```
>>> p2 = Page(pageid=1, page_name='test page', page_content='test page content')
>>> p2.save()
```

데이터베이스를 확인해 보아도 역시 아무런 값도 입력이 되지 않았다. 키 값을 달리하여 p3를 만들고 다시 save해 보자.

```
>>> p3 = Page(pageid=2, page_name='test page', page_content='test page content')
>>> p3.save()
```

이제서야 새로운 값이 인서트 되었다.

```
mysql> select * from mypage_page;
+--------+-----------+-------------------+
| pageid | page_name | page_content      |
+--------+-----------+-------------------+
|      1 | test page | test page content |
|      2 | test page | test page content |
+--------+-----------+-------------------+
2 rows in set (0.00 sec)
```

이로써 내릴 수 있는 결론은 장고는 기본적으로 **save라는 메써드 호출 시 중복에러가 발생할 경우 에러로 인식하지 않는다**는 사실이다.

한글도 정상적으로 입력이 될까? 궁금하다. 한번 입력해 보자.

```
>>> p4 = Page(pageid=3, page_name='한글', page_content='test page content')
>>> p4.save()
```

에러는 발생하지 않는다. 데이터베이스에도 정상적으로 입력된 것 같다. 한글 데이터가 깨져보이는 것은 도스창에서 mysql데이터를 보려고 했기 때문인듯 하다.

```
mysql> select * from mypage_page;
+--------+-----------+-------------------+
| pageid | page_name | page_content      |
+--------+-----------+-------------------+
|      1 | test page | test page content |
|      2 | test page | test page content |
|      3 | ?쒓?      | test page content |
+--------+-----------+-------------------+
3 rows in set (0.00 sec)
```

이제는 데이터를 조회해 보도록 하자.

```
>>> page_list = Page.objects.all()
>>> page_list
[<Page: Page object>, <Page: Page object>, <Page: Page object>]
```

```
>>> for page in page_list: print page.page_name
...
test page
test page
한글
```

한글도 정상적으로 출력되는 것을 확인할 수 있다.

이제 한가지 생각해야 할 점이 있다. 위 mypage_page테이블에 데이터를 인서트 할 경우 위의 예에서는 강제로 pageid값을 1,2,3을 알고서 입력했지만 일반적인 웹 프로그램이라면 max(pageid)+1 값을 읽은 후에 인서트하는 방식을 사용할 것이다.

장고에서는 이것을 어떻게 다룰 수 있는지 생각해보자.
장고의 order_by를 사용해 보자.

```
>>> pageid_list = Page.objects.order_by("pageid")
>>> pageid_list
[<Page: Page object>, <Page: Page object>, <Page: Page object>]
```

위와 같이 pageid로 order_by한 후에 가장 마지막 값의 pageid값이 현재 가장 큰 값임을 알 수 있다.

가장 마지막 값을 구하기 위해 -1 인덱싱을 했더니 재미있는 결과가 나왔다.

```
>>> pageid_list[-1].pageid
Traceback (most recent call last):
 File "", line 1, in
 File "c:\python2.6\lib\site-packages\django\db\models\query.py", line126, in __getitem__
   "Negative indexing is not supported."
AssertionError: Negative indexing is not supported.
```

바로 Negative 인덱싱을 허용되지 않는다는 에러이다. 현재로서는 왜 그런지 감이 오지 않는다. 다만 데이터베이스의 커서기능 때문이 아닐까? 유추해 볼 뿐이다.

아쉽지만 다음과 같이 가장 큰 값을 얻을 수는 있다.

```
>>> page_list[len(page_list)-1].pageid
3L
```

len(page_list)-1은 가장 마지막 값을 뜻하는 숫자값이다. 이제 가장 마지막 값을 알 수 있으니 3+1한 4라는 값을 다음 인서트시 사용하면 될 것이다. 하지만 실제 프로그램에서 이와같은 방법을 쓰는것은 어리석은 짓이다. max(pageid)+1 값을 알기 위해서 모든 데이터를 가져온 후 order by한 가장 마지막 값을 읽는 행위는 아마도 많이 혼날 것이다.

oder by desc limit 1을 사용하면 max와 비슷한 효과를 낼 수도 있다.

```
>>> Page.objects.order_by("-pageid")[0].pageid
3L
```

desc를 구현하기 위해서 장고는 컬럼명 앞에 "-" 문자를 입력하면 되도록 했다.

하지만 보다 더 근본적으로 아래와 같은 SQL문과 매핑되는 장고 메써드가 있는지 알아보자.

```
select max(pageid)+1 from mypage_page
```

장고북에서는 찾을 수 없었다. 구글로 검색해 보니 아래와 같은 방법을 찾을 수 있었다.

```
>>> from django.db.models import Max
>>> Page.objects.all().aggregate(Max("pageid"))
{'pageid__max': 3}
```

이상과 같이 장고의 모델에 대해서 간략하게 살펴 보았다.

장고북을 보니 이곳에서 알아 보았던 것들 말고도 filter라던가 get등의 메써드를 이용하여 데이터를 추출하는 방법을 보여준다. filter, get말고도 like라던지 join은 어떻게 하는지 subquery같은것은 어떻게 하는지 등등,,, 의문점이 많이 남아 있다. 이것들에 대해서는 기회가 되는데로 차츰 알아보기로 하자.

04) 폼(Forms)

모델 챕터에서 사용하던 테이블을 그대로 이용하여 폼(Forms)에 대하여 알아볼까 한다.

장고의 폼에 대해서 공부해 보기 전에 우선 폼(Forms)이 무엇인지 살펴보자. (대부분 알고 있는 이야기지만..)
폼은 당연히 HTML의 form태그를 말하는 것이다.

다음과 같은 HTML을 보자.

```
<form name="pageform" action="/save_page/" method="post">
<input type="text" name="page_name" id="page_name" />
<textarea name="page_content" id="page_content" rows="5" cols="20"></textarea>
</form>
```

위 폼은 페이지명, 페이지내용을 /save_page/라는 URL로 전송한다는 의도이다.

이제부터 장고가 폼데이터를 어떻게 처리하는지에 대해서 알아보자.
우선 위와같은 HTML을 화면에 렌더링하는 프로그램을 작성해보자.

mysite/views.py

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from django.template.loader import get_template
from django.template import Context
from mypage.models import Page

...
def page_form(request):
   t = get_template('page_form.html')
   return HttpResponse(t.render(Context({})))
```

mysite/urls.py

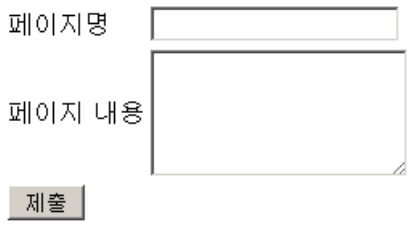
```
from django.conf.urls.defaults import *
from mysite.views import hello, page_form
```

```
urlpatterns = patterns('',
   ('^hello/$', hello),
   ('^page_form/$', page_form),
)
```

mysite/templates/page_form.html

```
<html>
<head>
<title>page form</title>
</head>
<body>
<form name="pageform" action="/save_page/" method="post">
<table>
<tr>
 <td>페이지명</td>
 <td>
   <input type="text" name="page_name" id="page_name" />
 </td>
</tr>
<tr>
 <td>페이지 내용</td>
 <td>
   <textarea name="page_content" id="page_content" rows="5" cols="20"></textarea>
 </td>
</tr>
<tr>
 <td colspan="2">
   <input type="submit" />
 </td>
</tr>
</table>
</form>
</body>
</html>
```

위와 같이 프로그램을 변경하고 생성한 후 http://localhost:8000/page_form/ 으로 접속해 보면 다음과 같은 화면을 볼 수 있다.



이제 "제출" 버튼을 클릭하면 /save_page라는 URL이 호출 될 것이다.
의도대로 페이지명과 페이지 내용을 정상적으로 저장 하려면 아래와 같이 프로그램들이 변경되어야 할 것이다.

mysite/views.py

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from django.template.loader import get_template
from django.template import Context
from mypage.models import Page
from django.db.models import Max

...

def save_page(request):
   page_name = request.POST['page_name']
   page_content = request.POST['page_content']
   pageid = int(Page.objects.all().aggregate(Max("pageid")).get("pageid__max")) + 1

   page = Page(pageid=pageid, page_name=page_name, page_content=page_content)
   page.save()

   return page_form(request)
```

post형태의 폼 데이터는 request.POST['폼데이터명'] 처럼 읽어들인다. 마찬가지로 get형태의 폼 데이터는 request.GET['폼데이터명']처럼 읽어야 한다.

mysite/urls.py

```
from django.conf.urls.defaults import *
from mysite.views import hello, page_form, save_page

urlpatterns = patterns('',
   ('^hello/$', hello),
   ('^page_form/$', page_form),
   ('^save_page/$', save_page),
)
```

위 처럼 프로그램들을 변경한 후 폼에 데이터를 입력한 후에 "Submit"버튼을 클릭하면 데이터가 입력되는 것을 확인할 수 있다.

장고는 폼을 객체화하여 사용할 수 있게끔 지원해 준다.
from django import forms 를 이용하는 방법인데 여기에서는 이 부분은 건너 띄기로 한다.
이 책은 장고를 가장 빨리 습득하고 이해하기 위한 초간단 매뉴얼이다.

05) 세션(Sessions)

이번엔 장고의 세션에 대해서 알아보자.

세션에 대해서 정확히 알려면 우선 **쿠키(Cookie)**가 무엇인지 알아야 한다.
쿠키가 도대체 왜 필요했는지를 알려면 **HTTP프로토콜 방식**에 대해서 또한 이해해야 한다.
세션을 알기위해서는 꼭 알아야 하는 내용이므로 "참고"를 읽어보자.

> **[참고 - 세션의 이해]**
>
> 사용자는 브라우저를 가지고 HTTP프로토콜을 구현한 서버에 접속하여 필요한 정보를 얻는다. 이것이 우리가 일반적으로 얘기하는 "웹서핑"이다.
>
> HTTP프로토콜은 접속이 유지되는 형태가 아니라 클라이언트가 한번 요청을 보낸 후 응답을 받으면 소켓이 끊어지는 형태이다. 최초 웹 서비스들은 HTML만 제공했었다. 따라서 브라우저가 요청하면 응답 HTML만 보내면 되었다. 정말 간단했다.
>
> 하지만 곧 사용자 정보가 필요해졌다. 누군가 어떤 페이지에 접속한 후 그 후속페이지에 접속할 경우에 이전 페이지에서 어떤 짓을 했는지 알아야 할 필요가 생긴 것이다. 이래서 만들어진 것이 그 유명한 쿠키(Cookie)이다.
>
> 쿠키는 클라이언트 기술이다. 웹 서버로 요청을 보내거나 받을 때 HTTP헤더에 Cookie를 세팅하는 방법이다. 즉 브라우저나 웹 서버가 Cookie값을 변경하며 서로 데이터를 주고 받는 기술이다. 하지만 쿠키는 데이터가 클라이언트(브라우저)에 저장되기 때문에 보안문제가 대두되게 되었다.
>
> 데이터를 클라이언트에 저장하지 않고 서버에 저장하는 기술이 바로 세션(Sessions)이다. 서버는 세션 데이터를 저장하고 서버의 세션 데이터를 이용할 수 있는 세션 키 값만 쿠키를 통해서 주고 받는다.

장고에서 세션을 사용하기 위해서 해주어야 할 것들이 있다.
우선 환경설정파일인 settings.py에서 다음 항목들을 확인해야한다.

mysite/settings.py

```
MIDDLEWARE_CLASSES = (
   ...
   'django.contrib.sessions.middleware.SessionMiddleware',
   ...
)

INSTALLED_APPS = (
   ...
   'django.contrib.sessions',
   ...
)
```

이렇게 설정한 후 다음의 명령을 이용하여 django_session이란 테이블을 생성해야만 한다.

```
C:\work\mysite>c:\python2.6\python manage.py syncdb
c:\python2.6\lib\site-packages\MySQLdb\__init__.py:34: DeprecationWarning: the sets module is deprecated
 from sets import ImmutableSet
Creating table django_session
```

생성된 테이블을 확인해 보자.

```
mysql> desc django_session;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| session_key  | varchar(40) | NO   | PRI |         |       |
| session_data | longtext    | NO   |     |         |       |
| expire_date  | datetime    | NO   |     |         |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

이제 세션을 사용할 수 있다.

세션에 어떻게 값을 저장하고 읽을 수 있는지를 확인할 수 있는 간단한 프로그램을 작성해 보자.

mysite/urls.py

```
from django.conf.urls.defaults import *
from mysite.views import hello, page_form, save_page, session_test

urlpatterns = patterns('',
   ...
   ('^session_test/$', session_test),
)
```

mysite/views.py

```
def session_test(request):
   if request.GET.has_key("data"):
       request.session['data'] = request.GET['data']
   if not request.session.has_key('data'):
       return HttpResponse(u"세션 데이터가 없습니다.")
   else:
       return HttpResponse(u"세션 데이터 : %s" % request.session['data'])
```

세션이 없는 경우는 "세션 데이터가 없습니다"를 출력하고 세션값은 Get방식으로 입력받는다.
데이터를 입력받은 경우에는 세션에 입력 받은 값을 저장한다.

세션값을 세팅하거나 변경하기 위한 URL호출 방법은 다음과 같다.

```
http://localhost:8000/session_test/?data=ok
```

위와 같이 호출할 경우 세션 data라는 키값에 "ok"라는 값이 저장된다.

장고는 기본적으로 세션 데이터를 2주일간 저장하는 것 같다.
django_session테이블의 expire_date라는 컬럼값을 읽어보니 한번 세션 데이터가 저장되면 2주일 이후시간이 세팅되는 것 같다.
만약 브라우저가 닫힐 때 세션값을 삭제하고 싶다면 아래와 같이 설정해야 한다.

mysite/settins.py

```
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

06) 마치며

이상과 같이 장고에 대한 간단한 설명을 마쳤다. 읽어보면 알겠지만 너무 많이 부족한 매뉴얼이다. 다만 장고가 무엇인지에 대한 감을 잡고 이제 어디서부터 시작해야 할 지 알게되었다면 그것으로 저자는 만족한다.

이제 해야 할 일은 django를 이용하여 실제 프로그램을 작성해 보는 일일 것이다.

백문이 불여일견이라 했던가!

직접 프로그램을 작성해 보지 않는다면 아무것도 얻을 수 없을것이다. 배가 고플 때 떡이 그려진 그림을 백날 쳐다 보아도 배가 부르지 않는 것처럼 지금 바로 django를 이용한 프로그램을 작성해 보기를 희망한다.

이 매뉴얼은 계속해서 발전시켜 나갈 것이며 또한 많은 도움을 필요로 하고 있다. 만약 이 문서를 함께 만들고 싶다면 저자는 언제나 "환영"이다. 아래의 이메일로 연락만 주면 바로 위키독스 "공동저자"로 설정할 것이다.

pahkey@gmail.com

09. 파이썬 Tips

파이썬 Tips에서는 파이썬을 활용하여 할 수 있는 여러 유용한 것들에 대해서 다룬다.

[01] 오픈API를 이용한 이미지 업로드

게시판류의 웹 서비스를 운영할 경우 이미지 업로드에 대해서 생각해야 한다. 웹 서버에 직접 이미지 파일을 업로드할 경우 그 파일에 대한 "관리의 책임"이 생기게 되기 때문이다. 만약 이미지를 대신 보관해 주고 그 링크를 제공해 주는 서비스가 있다면 굳이 자신의 서버에 이미지를 저장할 필요는 없을 것이다. 이곳에서는 OpenAPI를 이용한 이미지 업로드 방법에 대해서 소개한다.

1. 플리커를 이용한 이미지 업로드
플리커를 이용한 이미지 업로드방법에 대해서 설명한다.

우선 플리커 OpenAPI를 이용하기 위해서는 플리커 OpenAPI 키를 생성해야 한다.
아래 URL에서 키와 비밀키를 생성하고 인증 후에 호출할 URL을 등록시켜 놓는다.(플리커에서 해야 할 일은 여기까지이다.)

- http://flickr.com/services/api/keys/

(인증 후 호출할 URL예: http://lab.pyframe.org/example/openapi/flickrcallback)

이제 multipart/form을 이용하여 이미지를 업로드 해보자.

```
<form name="imageForm" action="/example/openapi/upload" method="post" enctype="multipart/form-data">
```

위 form을 submit하면 서버프로그램(위:/example/openapi/upload)이 실행될 것이다.
(여기서는 간편하게 요청된 이미지파일을 세션에 저장해 놓는 방식을 사용하기로 한다.)

/example/openapi/upload

```
import md5

...
def upload(self):
   filename, content = self.getfile("filename")
   session = self.getSession()
   session['file_content'] = content
   session['file_filename'] = filename
```

```
    m = md5.new()
    sig = "%sapi_key%sperms%s" % (API_SECRET, API_KEY, "write")
    m.update(sig)
    api_sig = m.hexdigest()
    authurl = "http://www.flickr.com/services/auth/?api_key=%s&perms=write&api_sig=%s" \
        % (self.getAppCfg().API_KEY, api_sig)
    self.redirect(authurl)
```

위 self.getfile메써드는 pyframe의 API중 하나로 파일명과 파일의 내용을 리턴한다.
(굳이 pyframe이 아니더라도 위와 같은 역할을 하도록 만들 수 있다.)
그런 후에 규칙에 따라 authurl을 만들고 authurl로 리다이렉트하여 호출하도록 한다.
authurl을 만들 때 사용하는 API_SECRET, API_KEY는 flickr사이트에서 생성한 키와 비밀키를 말한다.
인증이 이루어 진 후에는 flickr사이트에서 등록한 인증 후 호출되는 callback URL이 호출된다.
(예: /example/openapi/flickrcallback)

최종적으로 아래의 callback메써드에서 실제 플리커에 파일을 업로드하는 과정이 일어난다.

/example/openapi/flickrcallback

```
import flickrapi
...

def flickrcallback(self):
    m = md5.new()
    frob = self.get("frob")
    sig = "%sapi_key%sfrob%smethodflickr.auth.getToken" % \
        (API_SECRET, API_KEY, frob)
    m.update(sig)
    api_sig = m.hexdigest()
    authurl = "http://api.flickr.com/services/rest/?method=flickr.auth.getToken&api_key=%s&frob=%s&api_sig=
        (API_KEY, frob, api_sig)
    f = urllib2.urlopen(authurl)
    data = f.read()
    f.close()
    t = flickrapi.XMLNode.parseXML(data)
    flickr_token = t.auth[0].token[0].elementText

    flickr = flickrapi.FlickrAPI(API_KEY, API_SECRET)
    flickr.token = flickr_token
```

```
    session = self.getSession()
    content = session['file_content']
    filename = session['file_filename']

    r = flickr.upload(filename=filename, payload=content, callback=None)
    photoid = r.photoid[0].elementText

    xmlnode = flickr.photos_getSizes(photo_id=photoid)
    p = flickrapi.XMLNode.parseXML(xmlnode.xml)

    imageurl = str(p.sizes[0].size[-1].attrib["source"])

    del session['file_content']
    del session['file_filename']
```

위 함수에서 플리커 API를 사용하기 위해 flickrapi라는 라이브러리를 이용했다.

see also : http://flickrapi.sourceforge.net/flickrapi.html

flickr.upload메써드를 이용하여 손쉽게 파일(세션에 미리 저장해 놓았던)을 업로드 할 수 있다.
업로드 후에는 플리커 이미지 고유아이디인 photoid값과 imageurl을 알 수 있다. (리턴받은 photoid값을 이용하여 수정및 삭제등을 할 수 있다. 수정및 삭제에 대한 자세한 사항은 flickrapi를 참고하도록 하자)

이제 리턴받은 imageurl값을 이용하여 운영중인 웹 서비스에서 링크하여 표시하면 된다.

2. 피카사를 이용한 이미지 업로드
플리커를 이용한 이미지 업로드에 이어 두번째 피카사(picasa)를 이용한 이미지 업로드방법에 대해서 설명한다.

피카사 는 구글에서 제공하는 사진, 이미지 업로드 사이트로 기존에는 설치용 클라이언트 프로그램만을 이용하여 업로드가 가능했으나 지금은 open api를 통해 업로드를 하는 방식을 지원하고 있다.

우선 파이썬으로 picasa api를 사용하기 위해서는 gdata라이브러리를 다운로드 받아야 한다.

- http://code.google.com/p/gdata-python-client/downloads/list

Picasa의 경우 플리커처럼 키를 생성하거나 등록하는등의 절차가 필요없다.
다만 구글계정만 있으면 된다. 정말 간편하다.

이제 multipart/form을 이용하여 이미지를 업로드를 해보자.

```
<form name="imageForm" action="/example/openapi/upload" method="post" enctype="multipart/form-data">
```

위 form을 submit하면 서버프로그램(위:/example/openapi/upload)이 실행될 것이다.
여기서는 요청된 이미지파일을 세션에 저장해 놓는 방식을 사용하기로 한다.

/example/openapi/upload

```
import gdata.photos.service
import gdata.media
import gdata.geo

...
def upload(self):
   filename, content = self.getfile("filename")
   session = self.getSession()
   session['file_content'] = content
   session['file_filename'] = filename
   authSubUrl = self.picasaAuth()
   self.redirect(authSubUrl)


def picasaAuth(self):
   next = 'http://%s/example/openapi/picasaAuthAfter' \
       % self.getAppCfg().DOMAIN_NAME
   scope = 'http://picasaweb.google.com/data/'
   secure = False
   session = True
   gd_client = gdata.photos.service.PhotosService()
   return gd_client.GenerateAuthSubURL(next, scope, secure, session)
```

위와같이 upload메써드가 호출되면 우선 파일명과 파일내용을 세션에 잠시 저장해 놓는다. 그런후에 picasaAuth라는 메써드를 다시 호출한다. 이곳에서는 picasa api를 이용하여 인증을 거치도록 한다. 인증을 완료한 후에는 picasaAuthAfter라는 메써드가 다시 실행될 수 있도록 next URL을 설정한다.

/example/openapi/picasaAuthAfter

```
def picasaAuthAfter(self):
   authsub_token = self.get("token")
```

```
    # token update
    gd_client = gdata.photos.service.PhotosService()
    gd_client.auth_token = authsub_token
    gd_client.UpgradeToSessionToken()

    # add album
    #entry = gd_client.InsertAlbum('example', 'example')

    session = self.getSession()
    content = session['file_content']
    filename = session['file_filename']
    f = cStringIO.StringIO(content)

    album_url = '/data/feed/api/user/%s/album/%s' % ("default", "example")
    entry = gd_client.InsertPhotoSimple(album_url, filename,
        'Uploaded by example', f, content_type='image/jpeg')

    del session['file_content']
    del session['file_filename']

    self.redirect(entry.content.src)
```

인증이 완료된 후 호출된 picasaAuthAfter 메써드에서 위처럼 token값을 얻어내고 세션에 저장해 놓았던 이미지를 업로드 할 수 있다.

위 예제에서 보면 아래와 같이 새로운 앨범을 만드는 api가 있다.

```
gd_client.InsertAlbum('example', 'example')
```

버그인지 InsertAlbum은 정상적으로 동작하지 않는것 같다. 그래서 직접 picasa사이트에 접속하여 example이라는 앨범을 미리 만들어 놓아야만 했다. (어쩌면 이글을 작성하는 지금은 위 버그(?)가 사라졌을지도 모른다.) 확실히 구글의 api가 플리커의 api보다는 심플한 장점이 있다. 하지만 이미지의 품질이나 기타등등의 이유로 아직까지는 플리커의 손을 들어주고 싶다.

이상과 같이 플리커와 피카사의 API를 이용하여 이미지를 업로드하는 방법에 대해서 알아보았다.

[02] 저렴한 데이터베이스 백업

1. 데이터베이스 백업 파일을 생성
postgresql이라는 데이터베이스를 사용할 경우라고 가정한다.

backup.sh

```
#!/bin/bash
# ----------------------------------------------------------------
# pg_dump & restore
# [postgres postgres]$ pg_dump -Fc -f mydb.dump mydb
# [postgres postgres]$ dropdb mydb
# [postgres postgres]$ pg_restore -Fc -C -d template1 mydb.dump
# ----------------------------------------------------------------
BACKUP_DIR=/var/lib/postgresql/backup
DB_NAME=mydb
DATE=$(date +%Y%m%d)

/usr/bin/pg_dump -Fc -f $BACKUP_DIR/$DB_NAME.$DATE.dump $DB_NAME
```

위 backup.sh을 실행하면 mydb라는 데이터베이스가 /var/lib/postgresql/backup이라는 폴더에 mydb.YYYYMMDD.dump라는 이름으로 백업된다. (YYYYMMDD는 sh을 실행한 날짜가 된다) 백업한 파일을 복구하는 방법은 sh에 주석으로 표시되어 있는 방법을 사용하면 된다.

2. 백업한 파일을 지메일로 송신

email_backup.py

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-
# Import smtplib for the actual sending function
import smtplib
import time
import glob
import mimetypes

# Here are the email package modules we'll need
```

```
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email import encoders

BACKUP_DIR = "/var/lib/postgresql/backup"

def todaytime(fmt): return time.strftime(fmt)


# Create the container (outer) email message.
msg = MIMEMultipart()
msg['Subject'] = 'BACKUP MYDB [%s]' % todaytime("%Y/%m/%d")
# me == the sender's email address
# family = the list of all recipients' email addresses
msg['From'] = "보내는이 메일주소"
msg['To'] = "pahkey@gmail.com"

filenames = glob.glob("%s/*%s*" % (BACKUP_DIR, todaytime("%Y%m%d")))
for filename in filenames:
   ctype = 'application/octet-stream'
   maintype, subtype = ctype.split('/', 1)
   mb = MIMEBase(maintype, subtype)
   fp = open(filename, 'rb')
   mb.set_payload(fp.read())
   fp.close()
   encoders.encode_base64(mb)
   mb.add_header('Content-Disposition', 'attachment', filename=filename)

   msg.attach(mb)


mailServer = smtplib.SMTP("smtp.gmail.com")
mailServer.ehlo()
mailServer.starttls()
mailServer.ehlo()
mailServer.login("보내는이 메일주소", "보내는이 패스워드")
mailServer.sendmail(msg["From"], msg["To"], msg.as_string())
```

지메일(Gmail)로 파일을 첨부하여 메일송신하는 파이썬 스크립트이다. 백업폴더에 있는 파일 중 오늘 날짜만 보낸다.

3. crontab을 이용하여 자동화

```
00 03 * * * /var/lib/postgresql/backup.sh
20 03 * * * /var/lib/postgresql/email_backup.py
```

자동으로 새벽 3시에 backup.sh을 실행하고 3시 20분에 이메일로 송신한다.

[03] 블로그에 글 올리기

우리는 보통 블로그를 작성할 때 블로그에 접속하고 로그인 한 후에 글을 작성한다. 하지만 다른 프로그램에서 작성한 글을 블로그에 옮겨 적고 싶은 경우에는 어떻게 해야 할까?

보통 우리가 늘 사용했던 방식은 Copy & Paste이다. Copy & Paste로 옮긴 글들은 포맷이 맞지 않아 삐뚤빼뚤하게 보이기도 하니 그리 좋은 방법은 아니다. 이에 몇가지 방법들이 생기다가 표준화 된것이 바로 **MetaWebLogApi**이다.

MetaWebLogApi는 API, 즉 규칙이다. 국내 유명 블로그 사이트인 이글루스, 티스토리는 MetaWebLogApi를 따르고 있다. 그래서 미투데이나 스프링노트, 마이투두같은 사이트에서는 그곳에서 발생한 글들을 이글루스나 티스토리로 아주 쉽게 이동시켜 준다.

MetaWebLogApi의 규칙은 어떤것들이 있는지 알아보도록 하자.
(MetaWebLogApi는 XML-RPC라는 프로토콜을 사용한다. 이곳에서는 XML-RPC는 설명하지 않는다.)

1. 블로그 포스팅하기

```
metaWeblog.newPost (blogid, username, password, struct, publish)
returns string
```

metaWeblog는 XML-RPC규칙에 의해 생성된 객체이다.
파이썬은 다음과 같은 코드로 metaWeblog객체를 만들 수 있다.

```
metaWeblog=xmlrpclib.Server("http://pahkey.tistory.com/api").metaWeblog # 티스토리인 경우
```

입력값으로 사용하는 것들은 총 5가지이다.

1. blogid : 블로그 아이디 (티스토리인 경우, 관리자 페이지에서 확인할 수 있다. 이글루스는 이 값을 사용하지 않는다)
2. username : 사용자 아이디 (블로그 사이트 로그인 아이디)
3. password : 사용자 패스워드 (티스토리인 경우 블로그 사이트 로그인 패스워드, 이글루스는 관리자 페이지에서 확인 할 수 있다)
4. struct : 포스팅할 내용을 담고 있는 변수(제목, 내용등..)
5. publish : 포스팅한 내용을 공개할 것인지, 비공개로 할 것인지

struct에 올 수 있는 항목으로는 다음과 같은 것이 있다.

- category : 블로그에서 사용하는 카테고리명

- description : 포스팅 글 내용
- title : 포스팅 글 제목

(블로그 업체마다 이 struct는 약간씩 다르다. 티스토리의 경우 mt_keywords(태그명)를 사용할 수 있다.)

파이썬은 struct를 아래와 같이 딕셔너리로 구성하면 된다.

```
datastruct={'category': '', 'description': '내용입니다', 'title':'제목입니다.'}
```

출력값은 성공인 경우 postid(포스팅한 글의 id)를 받고 에러인 경우 에러내용을 받는다.

2. 블로그 수정하기

```
metaWeblog.editPost (postid, username, password, struct, publish)
returns true
```

입력값으로 사용하는 것들은 총 5가지이다.

- postid - 포스팅 성공 시 리턴받은 id값
- 나머지 4개는 상동

위와 동일하나 차이점은 첫번째 파라미터 값으로 포스팅 성공시 리턴받은 postid값을 송신해야 한다는 점이다.
응답으로는 블로그 수정이 성공했는지 실패했는지를 나타내는 boolean값을 받는다.

3. 블로그 읽기

```
metaWeblog.getPost (postid, username, password) returns struct
```

입력값으로 사용하는 것들은 총 3가지이다.

- postid - 포스팅 성공 시 리턴받은 id값
- 나머지 2개는 상동

응답으로는 struct를 받는다.

- category : 블로그에서 사용하는 카테고리명
- description : 포스팅 글 내용
- title : 포스팅 글 제목

더 자세한 사항은 다음의 URL을 확인하도록 하자.

- http://www.xmlrpc.com/metaWeblogApi

아래는 파이썬으로 MetaWebLog API를 테스트한 소스이다.

```
# -*- coding: utf-8 -*-
import xmlrpclib

# tistory
tistory=xmlrpclib.Server("http://pahkey.tistory.com/api")
datastruct={'category': '', 'description': "내용입니다.",
   'title':'제목입니다.', "mt_keywords":"mytodo,마이투두"}
no=tistory.metaWeblog.newPost("blogid", "username",
   "passwd", datastruct, True)
r = tistory.metaWeblog.getPost(no, "username", "passwd")

print r

# egloos
egloos=xmlrpclib.Server("https://rpc.egloos.com/rpc1")
datastruct={'category': '', 'description': "내용입니다.",
   'title':'제목입니다.'}
no=egloos.metaWeblog.newPost("", "username", "passwd", datastruct, True)
r = egloos.metaWeblog.getPost(no, "username", "passwd")

print r
```

[04] 구글 캘린더를 이용하여 무료로 SMS 보내기

이번에는 구글 캘린더를 이용하여 핸드폰으로 SMS를 보내는 기능에 대해서 살펴보겠다.

구글 캘린더에서 제공하는 SMS는 현재까지는 (어쩌면 앞으로도 계속) 무료이다. 물론 구글 캘린더에서 자신의 핸드폰이 아닌 다른 사람의 핸드폰으로 문자를 보낼 수는 없다. 하지만 다른 사람의 구글계정과 패스워드를 알면 그 사람에게 문자메시지를 보낼 수 있다. 구글은 여러개의 복수계정을 만드는 것을 허용하기 때문에 SMS수신전용 계정을 만들고 믿을 수 있는 사람들과 공유한다면 무료로 서로 문자메시지를 주고 받을 수 있게된다.

구글캘린더를 경유하여 문자메시지를 보내는 기능, 어디에 써 먹으면 좋을까?

활용할 수 있는 곳은 무궁무진 할 것이다.
이 기능이 매력적인 가장 큰 이유는 **무료**라는 점이다.

다음은 파이썬 코드이다.

```
# -*- coding: utf-8 -*-
try:
 from xml.etree import ElementTree # for Python 2.5 users
except ImportError:
 from elementtree import ElementTree
import gdata.calendar.service
import gdata.service
import atom.service
import gdata.calendar
import atom
import getopt
import sys
import string
import time


class GoogleCalendarSMS:
   def __init__(self, email, password):
       self.cs = gdata.calendar.service.CalendarService()
       self.cs.email = email
       self.cs.password = password
       self.cs.source = "mytodo"
       self.cs.ProgrammaticLogin()

   def send(self, calendar_name, sms_content, sms_time='', reminder_use=True):
       calendar = self.get_calendar(calendar_name)
       event = gdata.calendar.CalendarEventEntry()
```

```
        event.title = atom.Title(text=sms_content)
        event.content = atom.Content(text="")
        #event.where.append(gdata.calendar.Where(value_string=calendar_name))
        if not sms_time:
            sms_time = time.strftime('%Y-%m-%dT%H:%M:%S.000Z', time.gmtime(time.time()+60)) # 1 min later.
        event.when.append(gdata.calendar.When(
            start_time=sms_time, end_time=sms_time))
        if reminder_use:
            reminder = gdata.calendar.Reminder(minutes='0')
            reminder._attributes['method'] = 'method'
            reminder.method = 'sms'
            event.when[0].reminder.append(reminder)
        return self.cs.InsertEvent(event, calendar.GetAlternateLink().href)

    def senddate(self, calendar_name, sms_content, stime, etime):
        calendar = self.get_calendar(calendar_name)
        event = gdata.calendar.CalendarEventEntry()
        event.title = atom.Title(text=sms_content)
        event.content = atom.Content(text="")
        event.where.append(gdata.calendar.Where(value_string=calendar_name))
        event.when.append(gdata.calendar.When(
            start_time=stime, end_time=etime))
        return self.cs.InsertEvent(event, calendar.GetAlternateLink().href)

    def get_calendar(self, calendar_name):
        feed = self.cs.GetAllCalendarsFeed()
        for i, a_calendar in enumerate(feed.entry):
            if a_calendar.title.text == calendar_name:
                return a_calendar
        calendar = gdata.calendar.CalendarListEntry()
        calendar.title = atom.Title(text=calendar_name)
        calendar.summary = atom.Summary(text=calendar_name)
        return self.cs.InsertCalendar(new_calendar=calendar)


if __name__ == "__main__":
    gss = GoogleCalendarSMS("pahkey@gmail.com", "xxxxxxxx") # parameter : 구글계정, 패스워드
    gss.send("MY TODO", "안녕하세요") # parameter : 구글캘린더 캘린더명, 송신할 SMS메시지
```

정말 간단하다.

사용법은 아래코드와 같다.

```
gss = GoogleCalendarSMS("pahkey@gmail.com", "xxxxxxxx") # parameter : 구글계정, 패스워드
gss.send("MY TODO", "안녕하세요") # parameter : 구글캘린더 캘린더명, 송신할 SMS메시지
```

이 기능을 사용하기 위해서는 우선 구글 계정이 있어야 하고 구글 캘린더에서 SMS수신 설정을 해야한다.

다음의 URL을 참조하여 설정하자.

- http://googlekoreablog.blogspot.com/2008/08/google.html

위 파이썬 코드를 실행하기 위해서는 파이썬용 구글캘린더 API를 다운로드 받고 설치해야 한다.

- http://code.google.com/p/gdata-python-client/

※ 필자가 운영하는 마이투두(http://mytodo.org) 는 위 코드를 이용하여 작성한 TODO를 정해진 시간에 SMS 송신할 수 있도록 구현했다.

10. 파이썬 Quiz

치매예방 파이썬 Quiz.

Quiz마다 필자의 풀이도 참고용으로 포함시켜 놓았다.
하지만 그 풀이는 보지 말고 직접 퀴즈를 풀어 보도록 하자.
(풀이를 보게되면 그 방법에 집착하게 되어 새로운 방법을 찾기 어렵다)

[01] 절대적인 믿음을 줄 수 있는 테스트코드

다음은 애자일 컨설팅에 있는 자신의 프로그램에 목숨을 걸 수 있습니까 에 대한 내용중 일부이다.

[문제]

> 리스트는 유한개의 원소가 정해진 순서로 들어가 있는 자료구조입니다. 리스트에 포함된 원소는 하나의 숫자일수도 있지만 또다른 리스트일수도 있습니다. 이렇게 리스트 속에 리스트가 들어 있는 것을 중첩 리스트(nested list)라고 합시다.

> 리스트를 격자괄호로 둘러싸고, 각 원소는 쉼표로 구분해 표기하도록 하죠. 다음은 중첩 리스트의 예입니다.

```
[1, 2, [3, 4, [5]], 6, [[7,8]]]
```

> 예를 들어, 이런 리스트가 주어져 있을 때, 그걸 평평하게 만들어주는 프로그램 F를 작성해야 합니다.

> 즉, 결과가 [1, 2, 3, 4, 5, 6, 7, 8]로 나오면 됩니다.

[조건]

> 그런데 몇 가지 조건이 더 주어집니다. 자신은 예컨대 2시간 후에 수술을 받아야 합니다. 그때 사용하는 수술 기계에 이 프로그램이 중요한 역할을 하게 됩니다. 자신이 직접 작성한 프로그램으로 수술을 받는 것입니다. 프로그램이 오작동하면 자신은 죽을 수도 있습니다.

> 이 프로그램이 정상작동한다는 것을 최대한 보장하기 위해 프로그래밍 전, 프로그래밍 중, 프로그래밍 후 각각의 시기에 어떤 일을 하시겠습니까? (프로그램 F를 작성하는 것이 이 오디션의 목표는 아닙니다 -- 따라서 오디션 중에 그 프로그램을 작성할 필요까지는 없습니다)

[풀이]

```
result = []
def forLife(l):
   for item in l:
       if type(item) == list:
           forLife(item)
       else:
           result.append(item)

forLife([1, 2, [3, 4, [5]], 6, [[7,8]]])
print result
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[02] 넥슨 입사문제중에서

[문제]

어떤 자연수 n이 있을 때, d(n)을 n의 각 자릿수 숫자들과 n 자신을 더한 숫자라고 정의하자.

예를 들어 d(91) = 9 + 1 + 91 = 101

이 때, n을 d(n)의 제네레이터(generator)라고 한다. 위의 예에서 91은 101의 제네레이터이다.

어떤 숫자들은 하나 이상의 제네레이터를 가지고 있는데, 101의 제네레이터는 91 뿐 아니라 100도 있다. 그런데 반대로, 제네레이터가 없는 숫자들도 있으며, 이런 숫자를 인도의 수학자 Kaprekar가 셀프 넘버(self-number)라 이름 붙였다. 예를 들어 1,3,5,7,9,20,31 은 셀프 넘버 들이다.

1 이상이고 5000 보다 작은 모든 셀프 넘버들의 합을 구하라.

[풀이]

```
# -*- coding: euc-kr -*-
import unittest

def add_digit(no):
   return sum(map(int, ' '.join(str(no)).split()))

def f(no):
   return add_digit(no)+no

refer = {}
for i in range(1, 5001): refer[i] = f(i)

def getgen(no):
   result = []
   for n in range(1, no+1):
       #if f(n) == no: result.append(n)
```

```
        if refer[n] == no: result.append(n)
    return result

def selfno(limit):
    result = []
    for i in range(1, limit+1):
        if not getgen(i): result.append(i)
    return result


class GeneratorTest(unittest.TestCase):
    def test1(self):
        self.assertEquals(2, add_digit(200))
        self.assertEquals(10, add_digit(91))
        self.assertEquals(101, f(91))
        self.assertEquals(101, f(100))
        self.assertEquals([1], getgen(2))
        self.assertEquals([91, 100], getgen(101))
        self.assertEquals([1,3,5,7,9], selfno(10))
        print "result:", sum(selfno(5000))

if __name__ == "__main__":
    unittest.main()
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[03] PrimaryArithmetic

About PrimaryArithmetic

초등학생들이 여러 자리 수의 덧셈을 배울 때는 한 번에 한 자리씩 오른쪽에서 왼쪽으로 계산하도록 배운다. 그런데 그 자리 숫자의 합이 10을 넘어갈 때 그 윗자리 숫자에 1을 더해주는 것을 배울 때 많은 학생들이 힘들어한다. 일련의 덧셈 문제가 주어졌을 때 자리를 올리는 횟수를 세어서 선생님들이 학생들을 가르치는 데 도움을 줄 수 있는 프로그램을 만들어야 한다.

Input

> 각 행에는 열 자리 미만의 부호가 없는 정수가 두 개씩 입력된다. 마지막 줄에는 '0 0'이 입력된다.

output

> 마지막 줄을 제외한 각 줄에 대해 주어진 두 수를 더할 때 자리를 올려야 하는 횟수를 계산한 다음,
>
> 아래에 주어진 형식대로 결과를 출력한다.

Sample Input

```
123 456
```

```
555 555
```

```
123 594
```

```
0 0
```

Sample Output

```
No carry operation.
```

```
3 carry operations.
```

```
1 carry operation.
```

[풀이]

```
import unittest

class Number:
   def __init__(self, num):
       self.num = list(str(num))
   def __getitem__(self, index):
       try: return int(self.num[index])
       except IndexError: return 0
   def __len__(self):
       return len(self.num)

class Adder:
   def __init__(self):
       self.carries = [0] * 10
   def add(self, src, target):
       for i in range(1, max(len(src), len(target))+1):
           if src[-i]+target[-i]+self.carries[-i] >= 10:
               self.carries[-i-1] = 1
   def carryCount(self):
       return self.carries.count(1)


class CarryTest(unittest.TestCase):
   def test1(self):
       adder = Adder()
       adder.add(Number(1), Number(1))
       self.assertEquals(0, adder.carryCount())
   def test2(self):
       adder = Adder()
       adder.add(Number(9), Number(1))
       self.assertEquals(1, adder.carryCount())
```

```
    def test3(self):
        adder = Adder()
        adder.add(Number(99), Number(1))
        self.assertEquals(2, adder.carryCount())
    def test4(self):
        adder = Adder()
        adder.add(Number(899), Number(1))
        self.assertEquals(2, adder.carryCount())
    def test5(self):
        adder = Adder()
        adder.add(Number(555), Number(555))
        self.assertEquals(3, adder.carryCount())
    def test6(self):
        adder = Adder()
        adder.add(Number(123), Number(594))
        self.assertEquals(1, adder.carryCount())

import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(CarryTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[04] Spiral Array

Spiral Array

문제는 아주 간단하다.

```
6 6

 0   1   2   3   4   5
19  20  21  22  23   6
18  31  32  33  24   7
17  30  35  34  25   8
16  29  28  27  26   9
15  14  13  12  11  10
```

위처럼 6 6이라는 입력을 주면 6X6 매트릭스에 나선형 회전을 한 값을 출력하는 것이다.

[풀이]

제일 먼저 생각한 것은 방향이었다. 방향을 관장하는 클래스를 통해 현재 방향을 변경하는 것이다. 핵심은 방향을 언제 틀어야 하고, 어떤방향으로 틀어야 하는점.

처음에는 Direction이라는 클래스가 있었는데, 하는일이 적어서 Spiral Array클래스로 인라인 했다.

제일 리팩토링하고 싶은부분은 방향을 전환하는 메써드인 changeDirection메써드인데, 더 좋은 방법이 도무지 생각나지 않는다. 이 문제를 다 풀고나서 다른사람이 푼것을 구경해 보았는데 sin함수를 이용하여 좀 더 세련되게 방향을 트는 방법이 있는것 같다.

```
import unittest

"""
6 6

 0   1   2   3   4   5
19  20  21  22  23   6
18  31  32  33  24   7
17  30  35  34  25   8
16  29  28  27  26   9
15  14  13  12  11  10

"""
```

```
class SpiralArray:
   def __init__(self, row, col):
       self.maxcol = col-1
       self.maxrow = row-1
       self.factors = {
           "right":(0,1),
           "down":(1,0),
           "left":(0,-1),
           "up":(-1,0)
       }
       self.init()
   def init(self):
       self.cur = 0
       self.curPoint = 0,0
       self.map = {}
       for r in range(self.maxrow+1):
           for c in range(self.maxcol+1):
               self.map[r,c] = -1
       self.map[self.curPoint]=0
       self.setDirection("right")
   def setDirection(self, direction):
       self.directFactor = direction
   def getDirection(self):
       return self.directFactor
   def getCol(self):
       return self.curPoint[1]
   def getRow(self):
       return self.curPoint[0]
   def isDown(self, nextPoint):
       return self.getDirection() == "right" \
           and (self.getCol()==self.maxcol or self.hasValue(nextPoint))
   def isLeft(self, nextPoint):
       return self.getDirection() == "down" \
           and (self.getRow()==self.maxrow or self.hasValue(nextPoint))
   def isUp(self, nextPoint):
       return self.getDirection() == "left" \
           and (self.getCol()==0 or self.hasValue(nextPoint))
   def isRight(self, nextPoint):
       return self.getDirection() == "up" \
           and (self.getRow()==0 or self.hasValue(nextPoint))
   def changeDirection(self):
       nextPoint = self.nextPoint()
       if self.isDown(nextPoint): self.setDirection("down")
```

```
        elif self.isLeft(nextPoint): self.setDirection("left")
        elif self.isUp(nextPoint): self.setDirection("up")
        elif self.isRight(nextPoint): self.setDirection("right")
    def nextPoint(self):
        factor = self.factors[self.directFactor]
        return self.getRow()+factor[0], self.getCol()+factor[1]
    def allPointed(self):
        return self.map.values().count(-1) == 0
    def point(self):
        if self.allPointed():raise RuntimeError()
        self.changeDirection()
        self.curPoint = self.nextPoint()
        self.cur+=1
        self.setValue(self.curPoint)
    def hasValue(self, point):
        return self.getValue(point) != -1
    def setValue(self, point):
        self.map[point[0], point[1]] = self.cur
    def getValue(self, point):
        return self.map[point[0], point[1]]
    def run(self):
        while 1:
            try: self.point()
            except: break
    def __str__(self):
        result = []
        for row in range(self.maxrow+1):
            for col in range(self.maxcol+1):
                result.append("%4s" % self.map[row,col])
            result.append('\n')
        return ''.join(result)


class SpiralArrayTest(unittest.TestCase):
    def testChangeDirect(self):
        sa = SpiralArray(2,2)
        self.assertEquals((0,0), sa.curPoint)
        sa.setDirection("right")
        sa.point()
        self.assertEquals((0,1), sa.curPoint)
        sa.point()
        self.assertEquals(1, sa.getValue((0,1)))
        self.assertEquals((1,1), sa.curPoint)
        sa.point()
```

```
        self.assertEquals((1,0), sa.curPoint)
        try:
            sa.point()
            self.fail("should not reach here")
        except:
            pass
    def test1(self):
        sa = SpiralArray(6,6)
        sa.run()
        print sa

import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(SpiralArrayTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[05] Four Boxes

[문제]

4개의 직사각형이 평면에 있는데 밑변이 모두 가로축에 평행하다. 이 직사각형들이 차지하는 면적을 구하는 프로그램을 작성하시오. 이 네 개의 직사각형들은 서로 떨어져 있을 수도 있고 겹쳐 있을 수도 있다. 또한 하나가 다른 하나를 포함할 수도 있으며,변이나 꼭지점이 겹쳐질 수도 있다. 실행 파일의 이름은 RECT.EXE로 하시오.

입력형식

하나의 직사각형은 왼쪽 아래의 꼭지점과 오른쪽 위의 꼭지점의 좌표로 주어진다. 입력은 네 줄이며, 각 줄은 네 개의 정수로 하나의 직사각형을 나타낸다. 첫 번째와 두 번째의 정수는 사각형의 왼쪽 아래 꼭지점의 x좌표, y좌표이고, 세 번째와 네 번째의 정수는 사각형의 오른쪽 위 꼭지점의 x좌표, y좌표이다. 단, x좌표와 y좌표는 1 이상이고 1000 이하인 정수이다.

출력형식

화면에 4개의 직사각형이 차지하는 면적을 출력한다.

입력예제

```
1 2 4 4
```

```
2 3 5 7
3 1 6 5
7 3 8 6
```

출력예제

```
26
```

[풀이]

가장 많은 시간을 할애했던 부분은 다음을 유추해 내는 것이었다.

```
n(AUBUCUD) = n(A)+n(B)+n(C)+n(D)
           -(n(A∩B)+n(A∩C)+n(A∩D)+n(B∩C)+n(B∩D)+n(C∩D))
           +(n(A∩B∩C)+n(A∩B∩D)+n(A∩C∩D)+n(B∩C∩D))
           -n(A∩B∩C∩D)
```

intersect하고 난 후에 새로운 Rect객체를 리턴해 준것이 여러개의 intersect를 구하는데 많은 도움이 되었다. combination조합을 구하는 comb메써드는 다른사람의 것을 도용했다. (generator와 재귀를 이용한 기발한 방법이다.)

```
import unittest

class RectError(RuntimeError):
   pass

class Rect:
   def __init__(self, lx, ly, rx, ry):
       self.lx = lx
       self.ly = ly
       self.rx = rx
       self.ry = ry

   def size(self):
       return abs(self.rx - self.lx) * abs(self.ry-self.ly)

   def intersect(self, other):
       try:
           lx, rx = self.get(self.lx, self.rx, other.lx, other.rx)
```

```
            ly, ry = self.get(self.ly, self.ry, other.ly, other.ry)
            return Rect(lx,ly,rx,ry)
        except RectError:
            return Rect(0,0,0,0)

    def __eq__(self, other):
        return self.lx == other.lx \
            and self.ly == other.ly \
            and self.rx == other.rx \
            and self.ry == other.ry

    def __repr__(self):
        return str((self.lx,self.ly))+str((self.rx,self.ry))

    def get(self, sleft, sright, oleft, oright):
        if oleft <= sleft <= oright:
            if sright <= oright: return sleft, sright
            else: return sleft, oright
        elif sleft <= oleft <= sright:
            if oright <= sright: return oleft, oright
            else: return oleft, sright
        else:
            raise RectError("intersection does not exists!")


def comb(fromSet,choice):
    if choice==0:
        yield []
    else:
        for i, pivot in enumerate(fromSet):
            for each in comb(fromSet[i+1:],choice-1):
                yield [pivot]+each

def intersectSum(*rect):
    result = 0
    for rectGroup in rect:
        for targetRect in rectGroup:
            seed = targetRect[0]
            for t in targetRect[1:]:
                seed = seed.intersect(t)
            result += seed.size()
    return result

def calc(*rect):
```

```
    result = 0
    for i in range(len(rect)):
        sign = i%2 and -1 or 1
        result += sign * intersectSum(comb(rect, i+1))
    return result


class SumOfRectTest(unittest.TestCase):
    def testTwoRectNoIntersect(self):
        A = Rect(0,0,1,1)
        B = Rect(1,1,2,2)
        self.assertEquals(2, A.size()+B.size())
        self.assertEquals(0, A.intersect(B).size())

    def testTwoRectIntersect(self):
        A = Rect(0,0,2,2)
        B = Rect(1,1,3,3)
        self.assertEquals(8, A.size()+B.size())
        self.assertEquals(1, A.intersect(B).size())

    def testInterSect1(self):
        A = Rect(0,0,2,2)
        B = Rect(1,1,3,3)
        C = Rect(1,1,2,2)
        self.assertEquals(C, A.intersect(B))
        self.assertEquals(C, B.intersect(A))
        self.assertEquals(C, A.intersect(C))
        self.assertEquals(C, C.intersect(B))
        self.assertEquals(C, C.intersect(A))
        self.assertEquals(C, B.intersect(C))

    def testThreeRect1(self):
        A = Rect(-1,-1,1,1)
        B = Rect(0,0,2,2)
        C = Rect(1,1,3,3)
        self.assertEquals(4+4+4, A.size()+B.size()+C.size())
        self.assertEquals(1, A.intersect(B).size())
        self.assertEquals(1, B.intersect(C).size())
        self.assertEquals(0, A.intersect(C).size())
        self.assertEquals(0, A.intersect(B).intersect(C).size())

    def testThreeRect2(self):
        A = Rect(0,0,2,2)
        B = Rect(1,1,3,3)
```

```
        C = Rect(0,1,2,3)
        self.assertEquals(1, A.intersect(B).intersect(C).size())

    def testFourRect1(self):
        A = Rect(0,0,2,2)
        B = Rect(1,1,3,3)
        C = Rect(1,0,3,2)
        D = Rect(-1,0,1,3)
        self.assertEquals(0, A.intersect(B).intersect(C).intersect(D).size())
        self.assertEquals(1, A.intersect(B).size())
        self.assertEquals(2, A.intersect(C).size())
        self.assertEquals(2, A.intersect(D).size())
        self.assertEquals(2, B.intersect(C).size())
        self.assertEquals(0, B.intersect(D).size())
        self.assertEquals(0, C.intersect(D).size())
        self.assertEquals(1, A.intersect(B).intersect(C).size())
        self.assertEquals(4+4+4+6, A.size()+B.size()+C.size()+D.size())
        self.assertEquals(12, A.size()+B.size()+C.size()+D.size() -(1+2+2+2+0+0)+1)

    def testFourRect2(self):
        A = Rect(0,0,2,2)
        B = Rect(0,-1,2,1)
        C = Rect(-1,-1,1,1)
        D = Rect(-1,0,1,2)
        self.assertEquals(2, A.intersect(B).size())
        self.assertEquals(1, A.intersect(C).size())
        self.assertEquals(2, A.intersect(D).size())
        self.assertEquals(2, B.intersect(C).size())
        self.assertEquals(1, B.intersect(D).size())
        self.assertEquals(2, C.intersect(D).size())
        self.assertEquals(1, A.intersect(B).intersect(C).size())
        self.assertEquals(1, A.intersect(B).intersect(D).size())
        self.assertEquals(1, A.intersect(C).intersect(D).size())
        self.assertEquals(1, B.intersect(C).intersect(D).size())
        self.assertEquals(1, A.intersect(B).intersect(C).intersect(D).size())
        self.assertEquals(4+4+4+4, A.size()+B.size()+C.size()+D.size())
        self.assertEquals(9, 4+4+4+4-(2+1+2+2+1+2)+(1+1+1+1)-1)
        self.assertEquals(9, calc(A,B,C,D))

    def test1(self):
        A = Rect(0,0,2,1)
        B = Rect(0,0,1,2)
        C = Rect(-1,-1,1,1)
        D = Rect(-2,0,-1,3)
```

```
        self.assertEquals(9, calc(A,B,C,D))

    def test2(self):
        A = Rect(0,0,2,2)
        B = Rect(1,1,3,3)
        C = Rect(2,2,4,4)
        D = Rect(3,3,5,5)
        self.assertEquals(13, calc(A,B,C,D))

    def test3(self):
        A = Rect(1,2,4,4)
        B = Rect(2,3,5,7)
        C = Rect(3,1,6,5)
        D = Rect(7,3,8,6)
        E = Rect(1,2,4,4)
        self.assertEquals(26, calc(A,B,C,D,E))



import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(SumOfRectTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

테스트 코드를 리팩토링하지 않은채로 그대로 두었다. 테스트 코드는 현재 코드를 테스트할 수 있는 완벽한 상태여야 하고 또한 중복이 없어야 할 것이다.

아~ 중요한 테스트 코드

두번째 버젼

아래버전은 사각형을 cell로 보고 푸는 초간단 방법이다. 물론 속도가 느리고 실수연산은 불가능하지만, 이처럼 쉬운 방법을 생각해 낸다는건 쉬운 일이 아닌것 같다.

```
class RectBox:
    def __init__(self):
        self.box = {}
    def add(self, x1, y1, x2, y2):
        for x in range(x1, x2):
```

```
            for y in range(y1, y2):
                self.box[(x,y)] = 1
    def area(self):
        return len(self.box)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[06] Slurpy

Slurpy

스러피(Slurpy)란 어떠한 속성이 존재하는 문자열이다. 문자열을 읽어서 스러피가 존재하는지를 판단하는 프로그램을 작성해야 한다.

스럼프(Slump)는 다음 속성을 갖는 문자열이다.

1. 첫 번째 문자가 'D' 또는 'E'이다.
2. 첫 번째 문자 뒤에는 하나 이상의 'F'가 나온다.
3. 하나 이상의 'F' 뒤에는 또 다른 스럼프나 'G'가 온다. 스럼프는 'F' 끝에 오는 스럼프나 'G'로 끝난다. 예를 들어, DFFEFFFG는 첫 번째 문자가 'D'로 시작하고 두 개의 'F'가 나오며, 또 다른 스럼프 'EFFFG'로 끝난다.
4. 위의 경우가 아니면 스럼프가 아니다.

스림프(Slimp)는 다음 속성을 갖는 문자열이다.

1. 첫 번째 문자는 'A'이다.
2. 두 개의 문자로만 된 스림프면, 두 번째 문자는 'H'이다.
3. 두 개의 문자로 된 스림프가 아니면 다음 형식 중의 하나가 된다.
   - 'A' + 'B' + 스림프 + 'C'.
   - 'A' + 스럼프 + 'C'.
4. 위의 경우가 아니면 스림프가 아니다.

스러피(Slurpy)는 스림프(Slimp) 뒤에 스럼프(Slump)로 구성되는 문자열이다.

다음은 그 예이다.

```
Slumps : DFG, EFG, DFFFFFG, DFDFDFDFG, DFEFFFFFG
Not Slumps: DFEFF, EFAHG, DEFG, DG, EFFFFDG
Slimps: AH, ABAHC, ABABAHCC, ADFGC, ADFFFFGC, ABAEFGCC, ADFDFGC
Not Slimps: ABC, ABAH, DFGC, ABABAHC, SLIMP, ADGC
Slurpys: AHDFG, ADFGCDFFFFFG, ABAEFGCCDFEFFFFFG
Not Slurpys: AHDFGA, DFGAH, ABABCC
```

입력

입력될 문자열의 개수를 나타내는 정수 N 이 1 ~ 10의 범위로 우선 입력된다.

다음 줄부터 N개의 문자열이 입력된다. 문자열은 1 ~ 60 개의 알파벳 문자로 구성된다.

출력

첫 줄에는 "SLURPYS OUTPUT"을 출력한다. N 개의 문자열 입력에 대해서 각 문자열이 스러피인지를 "YES" 또는 "NO"로 표기한다. 마지막으로 "END OF OUTPUT"를 출력한다.

Sample Input

```
2
AHDFG
DFGAH
```

Sample Output

```
SLURPYS OUTPUT
YES
NO
END OF OUTPUT
```

[풀이]

```
# -*- coding: euc-kr -*-

'''
```

```
* slurpy (slimp and slump)

* slump - (D or E) and F+ and (slump or G)
* slimp - A and (H or ((B and slimp and C) or (slump and C))

** find slurpy!

[example]
Slumps : DFG, EFG, DFFFFFG, DFDFDFDFG, DFEFFFFFG
Not Slumps: DFEFF, EFAHG, DEFG, DG, EFFFFDG
Slimps: AH, ABAHC, ABABAHCC, ADFGC, ADFFFFGC, ABAEFGCC, ADFDFGC
Not Slimps: ABC, ABAH, DFGC, ABABAHC, SLIMP, ADGC
Slurpys: AHDFG, ADFGCDFFFFFG, ABAEFGCCDFEFFFFFG
Not Slurpys: AHDFGA, DFGAH, ABABCC
'''

import unittest


class UnitPattern:
   def __init__(self, *args):
       self.args = args
       self._remain = ''

   def match(self, target):
       raise NotImplementedError

   def remain(self):
       return self._remain


class Word(UnitPattern):
   def match(self, target):
       if not target: return False
       self._remain = target[1:]
       return self.args[0][0] == target[0]


class And(UnitPattern):
   def match(self, target):
       for arg in self.args:
           if not arg.match(target):
               return False
```

```
            target = arg.remain()
        self._remain = target
        return True


class Or(UnitPattern):
    def match(self, target):
        for arg in self.args:
            if arg.match(target):
                self._remain = arg.remain()
                return True
        return False


class More(UnitPattern):
    def match(self, target):
        if not target: return False
        moreword = self.args[0][0]
        for count, t in enumerate(target):
            if t != moreword:
                if count == 0 : return False
                break
        self._remain = target[count:]
        return True


class MultiPattern:
    def match(self, target):
        return self.pat.match(target)

    def remain(self):
        return self.pat.remain()


class Slump(MultiPattern):
    ''' slump - (D or E) and F+ and (slump or G) '''

    def __init__(self):
        self.pat = And(
                Or(Word('D'), Word('E')),
                More('F'),
                Or(self, Word('G'))
            )
```

```
class Slimp(MultiPattern):
   ''' slimp - A and (H or ((B and slimp and C) or (slump and C)) '''

   def __init__(self):
       self.pat = And(
               Word('A'),
               Or(
                   Word('H'),
                   Or(
                       And(Word('B'), self, Word('C')),
                       And(Slump(), Word('C'))
                   )
               )
           )


class Slurpy(MultiPattern):
   ''' slurpy (slimp and slump) '''

   def __init__(self):
       self.pat = And(Slimp(), Slump())



def isSlurpy(target):
   pat = Slurpy()
   result = pat.match(target)
   if pat.remain(): return False
   return result



### test code   ##########################################################


class SlurpyTest(unittest.TestCase):
   def testWord(self):
       word = Word('D')
       self.assertEquals(True, word.match('DEF'))
       self.assertEquals('EF', word.remain())

   def testAnd(self):
       D = Word('D')
```

```
        E = Word('E')
        andDE = And(D,E)
        self.assertEquals(True, andDE.match('DE'))

    def testMore(self):
        self.assertEquals(True, More('F').match('FFFF'))
        self.assertEquals(True, And(Word('D'), More('F')).match('DFFF'))
        more = More('F')
        more.match('FGHG')
        self.assertEquals('GHG', more.remain())

    def testOr(self):
        self.assertEquals(True, Or(Word('F'), Word('E')).match('F'))
        self.assertEquals(True, Or(Word('F'), Word('E')).match('E'))
        self.assertEquals(True,
            And(More('K'), Or(Word('F'), Word('E'))).match('KKKKE'))

    def testSlump(self):
        #Slumps : DFG, EFG, DFFFFFG, DFDFDFDFG, DFEFFFFFG
        self.assertEquals(True, Slump().match('DFG'))
        self.assertEquals(True, Slump().match('EFG'))
        self.assertEquals(True, Slump().match('DFFFFFG'))
        self.assertEquals(True, Slump().match('DFDFDFDFG'))
        self.assertEquals(True, Slump().match('DFEFFFFFG'))
        #Not Slumps : DFEFF, EFAHG, DEFG, DG, EFFFFDG
        self.assertEquals(False, Slump().match('DFEFF'))
        self.assertEquals(False, Slump().match('EFAHG'))
        self.assertEquals(False, Slump().match('DEFG'))
        self.assertEquals(False, Slump().match('DG'))
        self.assertEquals(False, Slump().match('EFFFFDG'))

    def testSlimp(self):
        #Slimps: AH, ABAHC, ABABAHCC, ADFGC, ADFFFFGC, ABAEFGCC, ADFDFGC
        self.assertEquals(True, Slimp().match('AH'))
        self.assertEquals(True, Slimp().match('ABAHC'))
        self.assertEquals(True, Slimp().match('ABABAHCC'))
        self.assertEquals(True, Slimp().match('ADFGC'))
        self.assertEquals(True, Slimp().match('ADFFFFGC'))
        self.assertEquals(True, Slimp().match('ABAEFGCC'))
        self.assertEquals(True, Slimp().match('ADFDFGC'))
        #Not Slimps: ABC, ABAH, DFGC, ABABAHC, SLIMP, ADGC
        self.assertEquals(False, Slimp().match('ABC'))
        self.assertEquals(False, Slimp().match('ABAH'))
        self.assertEquals(False, Slimp().match('DFGC'))
```

```
        self.assertEquals(False, Slimp().match('ABABAHC'))
        self.assertEquals(False, Slimp().match('SLIMP'))
        self.assertEquals(False, Slimp().match('ADGC'))

    def testSlurpy(self):
        #Slurpys: AHDFG, ADFGCDFFFFFG, ABAEFGCCDFEFFFFFG
        self.assertEquals(True, isSlurpy('AHDFG'))
        self.assertEquals(True, isSlurpy('ADFGCDFFFFFG'))
        self.assertEquals(True, isSlurpy('ABAEFGCCDFEFFFFFG'))
        #Not Slurpys: AHDFGA, DFGAH, ABABCC
        self.assertEquals(False, isSlurpy('AHDFGA'))
        self.assertEquals(False, isSlurpy('DFGAH'))
        self.assertEquals(False, isSlurpy('ABABCC'))


import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(SlurpyTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[07] Eight Queen

Eight Queen Problem

체스게임에서 Queen은 직선과 대각선을 자유롭게 움직일 수 있는 유닛이다.

8 X 8 체스판이 있다고 가정하고 이곳에 총 8개의 Queen을 놓아야 한다. 단 Queen간에 충돌이 없어야 한다. (각각의 Queen이 좌우, 대각선으로 움직일때 다른 Queen과 충돌이 없어야한다.)

다음과 같은 예를 들수 있다. (X는 Queen의 위치를 의미한다.)

```
X O O O O O O O
O O O O O O X O
O O O O X O O O
O O O O O O O X
O X O O O O O O
O O O X O O O O
O O O O O X O O
O O X O O O O O

O O O O O O X O
O O O X O O O O
O X O O O O O O
O O O O X O O O
O O O O O O O X
X O O O O O O O
O O X O O O O O
O O O O O X O O
```

가지수는 총 XX가지이다. 궁금하다면 바로 문제를 풀어보도록 하자.

[풀이]

```
import unittest

count = 0

class Board:
   def __init__(self, row, col):
       self.row = row
       self.col = col
```

```
        self.map = {}
        for r in range(self.row):
            for c in range(self.col):
                self.map[r,c] = 'O'
    def isQueenAvailable(self, point):
        if self.isQueenExistAtLine(point): return False
        if self.isQueenExistAtDiagonal(point): return False
        return True
    def setQueen(self, point):
        self.map[point] = 'X'
    def isExistQueen(self, point):
        return self.map[point] == 'X'
    def isQueenExistAtLine(self, point):
        for c in range(self.col):
            if self.isExistQueen((point[0],c)): return True
        for r in range(self.row):
            if self.isExistQueen((r, point[1])): return True
        return False
    def isQueenExistAtDiagonal(self, point):
        if self.isQueenExistAtDiagonalLeftUp(point): return True
        if self.isQueenExistAtDiagonalRightUp(point): return True
        return False
    def getStartForDiagonalSearchLeftUp(self, (row,col)):
        while 1:
            if row==0 or col==0: break
            row = row-1
            col = col-1
        return row, col
    def getStartForDiagonalSearchRightUp(self, (row,col)):
        while 1:
            if row==0 or col==self.col-1: break
            row = row-1
            col = col+1
        return row, col
    def isQueenExistAtDiagonalLeftUp(self, point):
        startRow, startCol = self.getStartForDiagonalSearchLeftUp(point)
        for i in range(8):
            r = startRow+i
            c = startCol+i
            if r == self.row: break
            if c >= self.col: break
            if self.isExistQueen((r,c)): return True
        return False
    def isQueenExistAtDiagonalRightUp(self, point):
```

```
        startRow, startCol = self.getStartForDiagonalSearchRightUp(point)
        for i in range(8):
            r = startRow+i
            c = startCol-i
            if r == self.row: break
            if c < 0: break
            if self.isExistQueen((r,c)): return True
        return False
    def searchNext(self, (row,col)):
        for c in range(col, self.col):
            if self.isQueenAvailable((row, c)):
                self.copyAndRerun((row,c))
    def run(self):
        for r in range(0, self.row):
            for c in range(0, self.col):
                if self.isQueenAvailable((r,c)):
                    self.searchNext((r, c+1))
                    self.setQueen((r,c))
        self.printEightQueen()
    def printEightQueen(self):
        if self.map.values().count('X')==self.row:
            global count
            count += 1
            print count
            print self
    def copyAndRerun(self, nextPoint):
        other = Board(self.row, self.col)
        other.map = self.map.copy()
        other.setQueen(nextPoint)
        other.run()
    def __str__(self):
        result = []
        for r in range(self.row):
            for c in range(self.col):
                result.append("%2s" % self.map[r,c])
            result.append('\n')
        return ''.join(result)


class EightQueenTest(unittest.TestCase):
    def testSetQueen(self):
        bd = Board(8,8)
        bd.setQueen((0,0))
        self.failUnless(bd.isExistQueen((0,0)))
```

```
    def assertCantSet(self, bd, point):
        self.failIf(bd.isQueenAvailable(point))
    def assertSet(self, bd, point):
        self.failUnless(bd.isQueenAvailable(point))
    def testLineSet(self):
        bd = Board(8,8)
        bd.setQueen((0,0))
        self.assertCantSet(bd, (0,1))
        self.assertCantSet(bd, (1,0))
    def testDiagonalSetLeftUp(self):
        bd = Board(8,8)
        bd.setQueen((4,4))
        self.assertCantSet(bd, (1,1))
        self.assertCantSet(bd, (7,7))
        self.assertCantSet(bd, (3,5))
        self.assertCantSet(bd, (5,3))
    def testDiagonalSetRightUp(self):
        bd = Board(8,8)
        bd.setQueen((2,3))
        self.assertCantSet(bd, (0,1))
        self.assertCantSet(bd, (5,6))
        self.assertCantSet(bd, (0,5))
        self.assertCantSet(bd, (5,0))
    def testNormalSet(self):
        bd = Board(8,8)
        self.assertSet(bd, (0,0))
        self.assertSet(bd, (1,2))
        self.assertSet(bd, (2,6))
    def testRun(self):
        bd = Board(8,8)
        bd.run()


import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(EightQueenTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[08] Tug Of War

About TugOfWar

사무실 야유회에서 줄다리기를 하기로 했다. 야유회에 참가한 사람들을 두 편으로 공평하게 나눈다. 모든 사람들이 둘 중 한 편에 참여해야 하며, 두 편의 사람 수는 한 명이 넘게 차이가 나면 안 된다. 그리고 양 편에 속한 사람들 체중의 총합 차를 최소한으로 줄여야 한다.

Input

> 첫번째 줄에는 테스트 케이스의 수가 입력되고, 그 다음 줄은 빈 줄이다.

> 각 케이스의 첫번째 줄에는 야유회에 참가한 사람의 수인 n이 입력된다.그리고 그 밑으로 n줄에 걸쳐서 야유회에 참가한 사람의 체중이 입력된다. 이 값은 1 이상 450 이하의 정수이다. 야유회에 참가하는 사람의 수는 최대 100명이다.

> 서로 다른 테스트 케이스 사이에는 빈 줄이 하나씩 입력된다.

output

> 각 테스트 케이스마다 한 줄씩의 결과를 출력하며, 각 줄마다 두 개씩의 정수가 출력된다. 첫번째 수는 한편에 속한 사람들의 체중의 총합, 다른 수는 다른 편에 속한 사람들의 체중의 총합이다. 이 두 값이 서로 다르면 작은 값을 먼저 출력한다.

> 두 개의 서로 다른 케이스에 대한 결과는 빈 줄로 구분한다.

Sample Input

```
5

```

```
3
100
90
200
```

```

```

```
6
45
55
70
60
50
75
```

```

```

```
4
92
56
47
82
```

```

```

```
5
2
3
4
7
8
```

```

```

```
4
50
50
100
200
```

Sample Output

```
190 200
```

```
175 180
```

```
138 139
```

```
12 12
```

```
150 250
```

worst case 총 100개의 랜덤 데이타. 1부터 450 사이의 정수.

```
80
```

```
434
```

```
253
```

```
166
```

```
441
102
```

```
356
107
144
93
380
387
14
302
330
1
298
262
154
184
362
381
231
76
52
84
115
135
153
129
359
438
131
394
427
213
321
276
256
24
79
175
418
150
415
387
143
38
252
222
```

```
326
331
147
78
378
232
17
228
316
360
140
178
301
384
239
78
286
77
385
420
51
88
428
371
217
206
267
133
140
267
25
438
442
10
217
300
127
280
88
149
244
89
330
405
```

```
140
149
26
134
384
435
```

이 자료를 TugOfWar 프로그램에 넣으면 (심사 서버에서) 10초 이내에 다음 결과가 나와야 한다.

worst case output

```
11402 11403
```

[풀이]

알고리즘

```
총 몸무게의 입력을 두개의 팀으로 나누어 분류한다. 바꾸었을 때 팀별 몸무게
총합차를 가장 많이 줄일수 있는 사람을 각 팀에서 한사람씩 뽑아 서로
바꾼다. 균형이 맞을 때까지 바꾸기를 계속한다. 더 이상 바꿀필요가 없으면
바꾸기를 중단한다.
```

```
import unittest

class Team:
   def __init__(self, weights):
       self.weights = weights
   def sumOfWeight(self):
       return sum(self.weights)
   def pull(self, index):
       return self.weights.pop(index)
   def push(self, weight):
       self.weights.append(weight)
   def __str__(self):
       return "Sum:[%d] ==> %s" % (self.sumOfWeight(), str(self.weights))
```

```
class Judge:
   def __init__(self, blue, red):
       self.blue = blue self.red = red
   def changeWeight(self, blueIndex, redIndex):
       blueWeight = self.blue.pull(blueIndex)
       redWeight = self.red.pull(redIndex)
       self.blue.push(redWeight)
       self.red.push(blueWeight)
   def forBestBalance(self):
       currentGap = abs(self.blue.sumOfWeight()-self.red.sumOfWeight())
       minGap = 450 * len(self.blue.weights)
       found = False
       blueSum = self.blue.sumOfWeight()
       redSum = self.red.sumOfWeight()
       for blueIndex, blueWeight in enumerate(self.blue.weights):
           for redIndex, redWeight in enumerate(self.red.weights):
               gap = blueWeight - redWeight
               nextGap = abs((blueSum-gap)-(redSum+gap))
               if nextGap < currentGap and nextGap < minGap:
                   minGap = nextGap
                   found = True
                   foundBlueIndex = blueIndex
                   foundRedIndex = redIndex
       if found: return foundBlueIndex, foundRedIndex
   def makeBalacedTeam(self):
       count = 0
       while True:
           foundIndex = self.forBestBalance()
           if foundIndex:
               blueIndex, redIndex = foundIndex
               self.changeWeight(blueIndex, redIndex)
               count += 1
               #print "count: %d, gap:%d" % (count,
               #    abs(self.blue.sumOfWeight()- self.red.sumOfWeight()))
           else:
               break


def tugOfWar(data):
   all = map(int, data.split())
   half = len(all)/2
   blue, red = Team(all[:half]), Team(all[half:])
   Judge(blue, red).makeBalacedTeam()
   result = []
```

```
    result.append('*' * 78)
    result.append(str(all))
    result.append('-' * 78)
    result.append(str(blue))
    result.append(str(red))
    result.append('*' * 78)
    print
    print '\n'.join(result)

import random
class TugWarTest(unittest.TestCase):
    def testTeam(self):
        A = Team([1,2,3])
        self.assertEquals(6, A.sumOfWeight())
        self.assertEquals(1, A.pull(0))
        self.assertEquals(5, A.sumOfWeight())
    def testJudge(self):
        A = Team([1,2,3])
        B = Team([4,5,6])
        judge = Judge(A,B)
        judge.changeWeight(1,0)
        self.assertEquals([1,3,4], A.weights)
        self.assertEquals([5,6,2], B.weights)
    def testJudgeforBestBalance(self):
        blue = Team([1,2,3])
        red = Team([4,5,6])
        judge = Judge(blue,red)
        self.assertEquals((0,1),judge.forBestBalance())
        judge.changeWeight(0,1)
        self.assertEquals(10, blue.sumOfWeight())
        self.assertEquals(11, red.sumOfWeight())
    def testJudgeUntilBalance(self):
        blue = Team([1,2,3])
        red = Team([4,5,6])
        judge = Judge(blue,red)
        judge.makeBalacedTeam()
        self.assertEquals([2,3,5], blue.weights)
        self.assertEquals([4,6,1], red.weights)
    def getRandomData(self, n):
        return ' '.join([str(random.randint(1, 450)) for i in range(n)])
    def testTugOfWar(self):
        tugOfWar("""1 2 3 4 5 6""")
        tugOfWar("""100 90 200 """)
        tugOfWar("""45 55 70 60 50 75 """)
```

```
        tugOfWar("""92 56 47 82 """)
        tugOfWar("""2 3 4 7 8""")
        tugOfWar("""50 50 100 200""")
        tugOfWar(self.getRandomData(100))
        #tugOfWar(self.getRandomData(500))
        #tugOfWar(self.getRandomData(1000))

import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TugWarTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

[09] LCD Display

LCD Display

한 친구가 방금 새 컴퓨터를 샀다. 그 친구가 지금까지 샀던 가장 강력한 컴퓨터는 공학용 전자 계산기였다. 그런데 그 친구는 새 컴퓨터의 모니터보다 공학용 계산기에 있는 LCD 디스플레이가 더 좋다며 크게 실망하고 말았다. 그 친구를 만족시킬 수 있도록 숫자를 LCD 디스플레이 방식으로 출력하는 프로그램을 만들어보자.

입력

> 입력 파일은 여러 줄로 구성되며 표시될 각각의 숫자마다 한 줄씩 입력된다. 각 줄에는 s와 n이라는 두개의 정수가 들어있으며 n은 출력될 숫자(0<= n <= 99,999,999), s는 숫자를 표시하는 크기(1<= s < 10)를 의미한다. 0 이 두 개 입력된 줄이 있으면 입력이 종료되며 그 줄은 처리되지 않는다.

출력

> 입력 파일에서 지정한 숫자를 수평 방향은 '-' 기호를, 수직 방향은 '|'를 이용해서 LCD 디스플레이 형태로 출력한다. 각 숫자는 정확하게 s+2개의 열, 2s+3개의 행으로 구성된다. 마지막 숫자를 포함한 모든 숫자를 이루는 공백을 스페이스로 채워야 한다. 두 개의 숫자 사이에는 정확하게 한 열의 공백이 있어야 한다.

> 각 숫자 다음에는 빈 줄을 한 줄 출력한다. 밑에 있는 출력 예에 각 숫자를 출력하는 방식이 나와있다.

예

> **입력 예**

> 2 12345

> 3 67890

```
0
```

```
```

출력 예

```
   --  --    --
|  |  || ||
|  |  || ||
  --  --  --  --
||   |  |  |
||   |  |  |
  --  --    --
```

```
```

```
 ---  ---  ---  ---  ---
|    || || || |
|    || || || |
|    || || || |
---     ---  ---
| |  || |  || |
| |  || |  || |
| |  || |  || |
---     ---  ---  ---
```

```
```

[풀이]

```
import unittest

class Number:
   @staticmethod
```

```
    def get(num):
        info = {
            0:['U','D','LU','LD','RU','RD'],
            1:['RU','RD'],
            2:['U','C','D','LD','RU'],
            3:['U','C','D','RU','RD'],
            4:['C','LU','RU','RD'],
            5:['U','C','D','LU','RD'],
            6:['U','C','D','LU','LD','RD'],
            7:['U','RU','RD'],
            8:['U','C','D','LU','LD','RU','RD'],
            9:['U','C','D','LU','RU','RD']
        }
        return Number(info[num])
    def __init__(self, info):
        self.info = info
    def hasValue(self, value):
        return self.info.count(value)

class Lcd:
    def __init__(self):
        self.numbers = []
        self.result = []
    def add(self, number):
        self.numbers.append(number)
    def init(self, *numbers):
        for number in numbers:
            self.add(Number.get(number))
    def makeHorizon(self, zoom, C):
        for number in self.numbers:
            self.result.append(' ')
            if number.hasValue(C): self.result.append('-' * zoom)
            else: self.result.append(' ' * zoom)
            self.result.append(' ' * 2)
        self.result.append('\n')
    def makeVertical(self, zoom, L, R):
        for i in range(zoom):
            for number in self.numbers:
                if number.hasValue(L): self.result.append('|')
                else: self.result.append(' ')
                self.result.append(' '*zoom)
                if number.hasValue(R): self.result.append('| ')
                else: self.result.append('  ')
            self.result.append('\n')
```

```
   def make(self, zoom):
       self.makeHorizon(zoom, 'U')
       self.makeVertical(zoom, 'LU', 'RU')
       self.makeHorizon(zoom, 'C')
       self.makeVertical(zoom, 'LD', 'RD')
       self.makeHorizon(zoom, 'D')
   def getResult(self):
       return ''.join(self.result)[:-1]


class LcdTest(unittest.TestCase):
   def testNormal(self):
       lcd = Lcd()
       lcd.add(Number.get(8))
       lcd.make(1)
       expect = """
-
| |
-
| |
-
"""
       self.assertEquals(expect[1:-1], lcd.getResult())
   def testZoom(self):
       lcd = Lcd()
       lcd.add(Number.get(8))
       lcd.make(2)
       expect = """
--
|  |
|  |
--
|  |
|  |
--
"""
       self.assertEquals(expect[1:-1], lcd.getResult())
   def testAllNumberWithZoom(self):
       lcd = Lcd()
       lcd.init(1,2,3,4,5,6,7,8,9)
       lcd.make(2)
       print
       print lcd.getResult()
```

```
import sys
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(LcdTest))
unittest.TextTestRunner(verbosity=2, stream=sys.stdout).run(suite)
```

이곳에 자신이 작성한 코드를 공개 해 보자.

90. 부록

이젠 무엇을 할 것인가?
이곳에서는 파이썬 관련정보를 주로 다루게 될 것이다. 우선 이 책을 읽은 후에 관심을 가질 만한 것들에 대해서 얘기한다. 파이썬 배포본과 함께 제공되는 많은 문서와 하우투 문서들, 그리고 온라인상에 있는 많은 유용한 파이썬 문서들에 대해서 얘기한다. 다음에 파이썬 뉴스그룹과 파이썬 공식 홈페이지인 www.python.org와 한국 파이썬 사용자 모임인 파이썬 정보광장에 대한 얘기를 한다. 그리고 마지막으로 Swig, Zope, Jython, PIL, Numeric Python 에 대해서 간략하게 알아 볼 것이다.

파이썬 문서
파이썬을 제대로 알고 잘 활용하기 위해서는 직접 많은 프로그램들을 만들어 보는 것도 중요하지만 관련 서적과 참고 문헌들을 언제나 가까이 하고 읽는 것 또한 매우 중요한 일일 것이다. 이 책을 읽고 난 후에 다음의 문서들을 읽어본다면 독자는 한층 더 깊이있는 파이썬 프로그래머가 될 것이다.

파이썬 튜토리얼
파이썬을 만든 귀도가 직접 만든 문서로서 파이썬 배포본에 함께 들어있는 문서이다. 튜토리얼인만큼 쉽게 다가설 수 있지만 내용이 매우 함축적이어서 초보자가 얼른 이해하기 힘든 부분이 많이 있다. 그리고 무엇보다도 영문이기 때문에 어려움이 많다. 한국 파이썬 사용자 모임인 파이썬 정보광장에서 이만용씨가 튜토리얼 문서를 번역한 것을 찾아 볼 수도 있다. 꼭 한번은 읽어야만 될 문서이다.

파이썬 라이브러리 레퍼런스
이 문서의 중요성을 새삼 거론할 필요는 없을 것 같다. 이 문서 없이는 제대로 된 파이썬 프로그램을 작성할 수가 없다. 항상 가까이에 두고 늘 찾아보아야 하는 문서이다. 이 문서를 처음부터 끝까지 대충이라도 꼭 한번 읽어보도록 하자. 어떤 모듈들이 있는지 조차 모른다면 프로그래밍을 할 때 많은 어려움을 겪게 될 것이다. 또한 프로그래밍을 할 때 어떤 모듈을 사용할 것인지에 대한 감각을 익히기 바란다.

파이썬 하우투들
http://www.python.org/doc/howto 에 가면 파이썬 하우투 문서들을 찾아 볼 수가 있다. 특정한 주제에 대해서 라이브러리 레퍼런스의 내용보다 자세한 설명을 해 준다. 다음과 같은 문서들이 있다.

Advocacy
파이썬을 사용하는 이유에 대한 토론, 파이썬으로 하기에 적당한 일과 그렇지 않은 것들에 대한 이야기등.

Curses Programming with Python
파이썬과 Curses를 이용한 텍스트 프로그래밍 소개

Editor Configuration for Python
파이썬 프로그래밍에 사용되는 에디터 설정법.

Regular Expression HOWTO
정규표현식 하우투 문서.

Socket Programming HOWTO
소켓 프로그래밍 하우투 문서.

Sorting Mini-HOWTO
내장 함수인 sort()메소드를 이용한 많은 소팅 기법소개.

XML HOWTO
파이썬으로 하는 XML 처리

유용한 온라인 문서들
온라인 상에 있는 파이썬에 관련한 유용한 문서들을 소개한다.

Why Python? Eric. S. Raymond
http://www2.linuxjournal.com/lj-issues/issue73/3882.html

에릭 레이먼드의 글로 자신이 파이썬을 사용하게 된 계기와 파이썬을 사용하는 이유, 그리고 파이썬을 사용함으로써 얻을 수 있었던 것들에 대한 자신의 경험을 위주로 한 글이다.

Learning to Program - by Alan Gauld
http://www.crosswinds.net/~agauld/

컴퓨터 프로그래밍에 대한 기본 원리를 설명하고 프로그래밍이 무엇인지에 대한 내용과 문제 해결을 위한 기초 테크닉들을 설명한다. 대부분 모든 설명이 파이썬을 기반으로 이루어져 있다.

Non-Programmers Tutorial - Josh Cogliati
http://www.honors.montana.edu/~jjc/easytut/easytut/

프로그래밍에 대한 경험이 없는 사람을 대상으로 하는 파이썬 프로그래밍에 대한 문서이다.

Instant Python - by Magnus Lie Hetland
http://www.hetland.org/python/instant-python.php

파이썬을 빠르게 배울수 있는 간단한 튜토리얼 문서이다.

Python for experienced programmers - Mark Pilgrim
http://diveintopython.org/toc.html

객체지향 프로그래밍과 파이썬 프로그래밍에 대한 경험이 있는 사람들을 대상으로 한 보다 깊이 있는 파이썬 프로그래밍 설명 문서이다.

How to think like a computer scientist Python Version by Allen B.Downey and Jeffrey Elkner
http://www.ibiblio.org/obp/thinkCSpy/

제목그대로 프로그래밍을 할 때 컴퓨터 과학자처럼 생각하는 방법에 대한 설명 문서이다. 객체지향 프로그래밍에 대한 설명도 포함한다.

comp.lang.python
이곳에는 파이썬에 대한 질문과 답변들, 새로운 소식들이 하루에도 100건 이상씩 올라온다. 이 뉴스그룹을 잘만 활용한다면 많은 도움이 될 것이다. 이 뉴스그룹에 글을 올리는 사람들 중에는 잘 알려진 유명한 사람들이 많다. 그러한 사람들과 서로 뉴스그룹 상에서 대화할 수 있는 기회를 가져보도록 하자. comp.lang.python을 활용하는 방법으로 데자 뉴스를 이용하는 것도 좋은 방법이다. 다음의 URL에 접속하면 comp.lang.python의 기사를 더욱 쉽게 읽을 수 있다. 데자 뉴스는 질문과 답변이 잘 정리되어 있다.

http://groups.google.com/groups?oi=djq&as_ugroup=comp.lang.python

www.python.org

파이썬 공식 홈페이지로 파이썬에 대한 대부분의 모든 정보를 이곳에서 구할 수 있다. 파이썬 공식 패키지를 다운받을 수 있고, 파이썬에 관한 무수히 많은 문서들을 볼 수 가 있다. Topic Guide에서는 Tkinter, XML, Databases, Web Programming등 굵직한 주제를 다루고 SIGs(Special Interest Groups)에서는 자신이 관심있는 프로그래밍 부분에 대한 자료를 충분히 구할 수 있고 또한 참여 할 수 있으며, 메일링 리스트를 이용하여 정보를 교환할 수 있을 것이다.

파이썬 마을(python.kr)
한국 파이썬 사용자들이 대부분 있는 곳이다. 파이썬을 공부해가며 모르는 부분에 대해서 질문을 할 수 있는 게시판등이 있고 각종 번역물이나 파이썬 관련 정보를 손쉽게 구할 수 있다. 수많은 글들이 올라오고 많은 문서들이 생겨나는 매우 바쁜 곳이다. 아마도 필연적으로 독자는 이 사이트에서 많은

시간을 보내게 될 것이다.

파이썬과 에디터
파이썬 코드를 작성하기 위해서 어떤 에디터를 선택할 것인가?
이문제에 대한 해답은 자신에게 익숙한 에디터를 선택하는 것이 정답일 것이다.

하지만 아직 마땅히 사용할 만한 에디터가 없는 독자에게 필자는 몇가지 추천하고픈 에디터가 있다. 윈도우즈 사용자라면 에디트 플러스나 울트라 에디터를, 리눅스 사용자라면 당연히 VI에디터를 추천한다. 물론 리눅스에는 이맥스라는 좋은 에디터가 있긴 하지만 초보자에겐 어울리지 않을 듯 하다.

여기서는 에디트 플러스를 설치하고 파이썬 사용에 필요한 몇가지 설정 사항에 대해서 알아보기로 하자.

에디트 플러스(Edit Plus)
이 프로그램은 공개 소프트웨어는 아니기 때문에 평가판을 이용해야 한다. 다운받은 후부터 한달 간 쓸 수 있다.

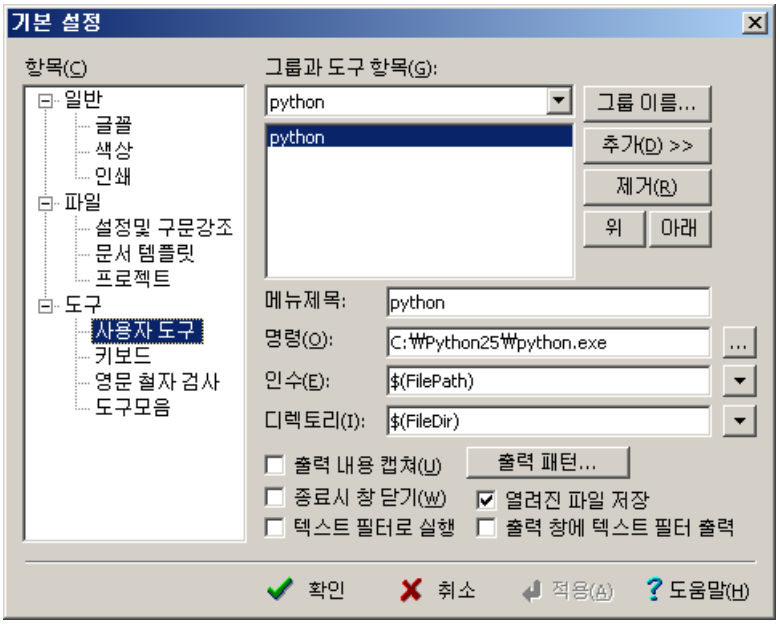
- http://www.editplus.com/kr

위 URL에서 파일을 다운로드하고 설치를 하자.

이제 파이썬을 에디트 플러스에서 잘 쓰기 위해 어떻게 설정을 해야하는지 보도록 하자.
에디트 플러스를 실행시키자. 평가판은 "동의함 " 버튼을 눌러야만 쓸 수 있다.

에디트 플러스에서 파이썬을 편하게 사용하기 위한 몇가지 설정을 먼저 하도록 하자.

1. 첫 번째 설정
우선 에디트 플러스를 실행시키고 **메뉴**바중 **도구**를 선택한 다음 **기본설정**항목을 선택하자.
여러 가지 선택 메뉴중에서 사용자 도구를 선택하자. 다음과 같은 화면을 볼 수 있을 것이다.



위의 창처럼 내용이 채워지도록 수정한다.

==> 그룹이름 버튼을 눌러서 그룹이름을 "Python"으로 바꾼다. 다음에 추가 버튼을 누른 다음 프로그램을 선택하고 메뉴제목에 " python " 이라고 적고 명령란에는 파이썬 프로그램이 있는 경로를 적어준다. 인수란에는 옆의 화살표버튼을 누르고 파일이름을 선택하면 자동으로 아래의 그림처럼 설정된다. 디렉토리란에는 옆의 화살표버튼을 누르고 파일 디렉토리를 선택하면 아래의 그림처럼 역시 설정된다. 다음에 적용버튼을 누르도록 하자. 이 설정을 하는 이유는 파이썬 소스코드를 작성한 다음에 자동으로 실행을 시키기 위해서이다.

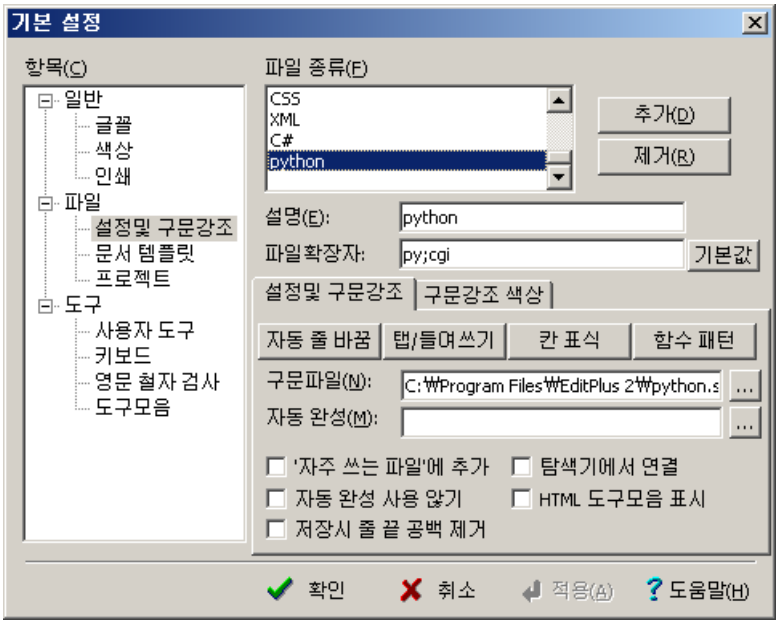
2. 두 번째 설정

에디트 플러스에서 파이썬 코드를 입력하면 검정색으로만 표시가 된다. 파이썬 코드에 색깔을 입히고 싶다면 파이썬 Syntax파일을 다운받아야 한다. 파이썬 Syntax파일은 다음에서 다운받을 수 있다.

- http://www.editplus.com/kr/javacpp.html

이곳에서 pythonfiles.zip파일을 다운받도록 하자. 이 압축된 파일을 풀면 python.stx라는 파일이 생긴다. 이것을 에디트 플러스의 디렉토리인 C:\Program Files\EditPlus 2(또는 EditPlus 3)로 복사를 하자. 다음에 이 Syntax파일을 사용하기 위해서 에디트 플러스의 **메뉴바**중 **도구**메뉴의 **기본 설정**을 선택하자. 여기서 **설정 및 구문 강조**를 선택하도록 하자.

아래와 같은 화면을 볼 수 있을 것이다.



위의 창처럼 내용이 채워지도록 수정한다.

우선 추가 버튼을 누르고 아래의 그림과 같이 설명란에 추가할 파이썬 Syntax이름을 넣는다. " Python"이라고 입력하자. 파일 확장자에는 'py'라고 입력을 하자. 다음에 구문파일(N)란에 우리가 미리 저장한 Python Syntax파일이 있는 디렉토리 경로인 C:\Program Files\EditPlus 2\python.stx를 입력해주자. 옆의 ...버튼을 이용하면 손쉽게 입력할 수가 있을 것이다. 그런다음에 창 가장 아래에 있는 적용버튼을 누르자. 아래의 그림은 이미 적용버튼을 누른상태의 그림이다. 이런 설정을 하는 이유는 파이썬 코드를 칼라로 보여주기 위해서이다. 그냥 검정색 글씨만을 보길 원하다면 이 두 번째 설정을 할 필요가 없다.

이제 에디트 플러스로 test.py와 같은 프로그램을 작성하고 저장한 후 Ctrl-1을 실행하면 파이썬이 자동으로 실행되는 것을 확인할 수 있을 것이다.

파이썬과 다른 언어와의 비교
Guido van Rossum저. (이강성 역)

Python은 흔히 Java, JavaScript, Perl, Tcl 또는 Samlltalk와 같은 인터프리터 언어와 비교된다. C++, Common Lisp 그리고 Scheme과 같은 비교되 부각된다. 이 절에서 난 이들 언어를 간단히 비교할 것이다. 이 비교는 언어적인 측면에서만 다룬다. 실제적으로, 프로그래밍 언어의 선택은 다른 실세계

제약(비용, 유용성, 학습 그리고 선행 투자비용 혹은 감성적인 친근감까지도)에 의해 자주 언급되기도 한다. 이러한 면들은 아주 가변적이고, 이러한 면으로 비교를 한다는 것은 시간 낭비에 가깝다.

Java (자바)
일반적으로 Python 프로그램은 Java 프로그램 보다는 느리게 수행된다. 하지만 Python 프로그램은 개발하는 시간이 훨씬 적게 걸린다. Python 프로그램은 Java 프로그램보다도 3-5배 정도 코드가 짧다. 이 차이는 Python의 내장 고수준 데이터 형과 동적인 형결정 기능에서 기인한다고 생각한다. 예를 들면, Python 프로그래머는 인수나 변수의 형을 선언하는데 시간을 허비하지 않고, 구문적인 지원이 언어안에 내장되어 있는 Python의 강력한 다형질의 리스트(polymorphic list)와 사전 형은 거의 모든 Python 프로그램에서 유용하게 활용되고 있다. 실행시간 형결정으로 인해서 Python의 실행시간에 Java가 하는 것보다 좀더 많은 일을 한다. 예를 들면, a+b와 같은 식을 계산할 때, 먼저 컴파일시에 알려지지 않은 a와 b 객체를 검사하여 그들의 형을 알아내야 한다. 그리고 나서 적절한 덧셈 연산을 호출한다. 그 덧셈 연산은 객체에 따라 사용자에 의해 오버로드(overloaded)된 것일 수 있다. 반면에, Java는 효과적인 정수형, 실수형 덧셈을 한다. 하지만 a와 b의 변수선언을 요구하고, + 연산자에 대한 사용자 정의 연산자 오버로딩을 허용하지 않는다.

이러한 이유들로, Python은 '접착' 언어로서 아주 적당한 반면, Java는 저수준 구현 언어로 특성화 지을 수 있다. 사실 이 두 언어는 아주 훌륭한 조합을 이룬다. Java에서 개발된 요소(components)들이 Python에서 활용된다; Python 역시 Java로 구현되기 전에 그 프로토타입을 정하는데 활용된다. 이러한 형의 개발을 지원하기 위해, Java로 쓰여진 Python 구현(implementation)이 개발중이다. 이것은 Java에서 Python을 호출하고 그 반대도 가능하게 해준다. 이 구현으로, Python 소스코드는 Java 바이트코드로 (Python의 동적 의미를 지원하기 위한 실행시간 라이브러리의 도움으로)번역된다.

Javascript (자바스크립트)
Python의 '객체기반' 부분 집합이 대략 JavaScript와 동일하다. JavaScript와 같이 (그러나 Java와는 다르게), Python은 클래스안에 정의하지 않아도 되는, 단순한 함수와 변수를 사용하는 프로그래밍 스타일을 지원한다. 하지만 JavaScript는 이것이 지원하는 전부이다. Python은, 반면에, 훨씬 큰 프로그램을 클래스와 상속이 중요한 역할을 하는 진정한 객체 지향 프로그래밍 스타일을 통하여 더 좋은 코드 재사용을 하도록 지원한다.

Perl (펄)
Python과 Perl은 비슷한 배경에서 개발되었다(유닉스 스크립트언어에서 성장했다). 그리고 많은 비슷한 기능을 지원한다. 그러나 철학은 다르다. Perl은 보편적인 응용지향 태스크를 지원하는데 중심을 두었지만 (예:내장 정규식 표현, 파일 스캐닝과 보고서 생성 기능들), Python은 보편적인 프로그래밍 방법론 (자료구조 설계 및 객체지향 프로그래밍)을 지원한다. 그리고 프로그래머가 우아하고(elegant) 암호같지 않은 코드를 통해 읽기 쉽고 관리하기 쉽도록 한다. 결과적으로, Python이 Perl과 가깝지만 그 원래 응용 영역을 침범하는 일은 많지 않다. 하지만 Python은 Perl의 적합한 응용분야 외에 많은 부분에서 적용성을 갖는다.

Tcl (티클)
Python과 같이 Tcl은 독립적인 프로그래밍 언어 분 아니라, 응용 확장언어(extension language)로도 사용된다. 하지만, 전통적으로 모든 데이터를 문자열로 처리하는 Tcl은 자료구조에 약하고 Python

보다 실행에 시간이 많이 걸린다. Tcl은 또한 모듈러 이름영역(name space)와 같은, 큰 프로그램을 쓰기에 적합한 특징들을 가지고 있지 않다. 따라서 Tcl을 사용하는 전형적인 큰 응용 프로그램은 특별히 그 응용에 필요한 C나 C++로 확장된 부분을 갖는다. 이에 반해서 Python 응용 프로그램은 '순수한 Python'으로만 흔히 기술된다. 물론 순수한 Python을 이용한 개발은 C나 C++부분을 쓰고 디버깅하는 것보다도 훨씬 빠르다. Tcl의 결점을 매우는 부분이 Tk 툴킷이다. Python은 Tk을 표준 GUI 라이브러리로 쓰도록 적용했다.

Tcl 8.0은 바이트코드를 도입하여 빠른 처리를 했고, 제한된 데이터 형 지원과 이름영역을 지원한다고 하지만 여전히 거추장스러운 프로그래밍 언어이다.

Smalltalk (스몰토크)
아마도 Python과 Smalltalk의 가장큰 차이는 Python이 보다 더 '주류(mainstream)' 구문을 가진다는 것이다. Python은 Smalltalk과 같이 동적인 형결정과 결합(binding)을 한다. Python의 모든 것은 객체이다. 하지만, Python은 내장 객체 형과 사용자 정의 클래스를 구별하고, 내장 형으로부터의 상속은 현재로서 허용하지 않는다. Smalltalk의 데어터 타입의 표준 라이브러리 모음은 훨씬 섬세한 반면, Python의 라이브러리는 인터넷과 WWW 세계 (email, HTML, FTP등) 에 적응하기 좋은 많은 기능을 제공한다.

Python은 개발환경과 코드 배포에 있어서 다른 철학을 갖는다. Smalltalk이 환경과 사용자 프로그램을 포함하는 통일된 '시스템 이미지'를 갖는데 반해, Python은 표준 모듈과 사용자 모듈을 다른 파일에 저장하여 쉽게 재배열되고 시스템 밖으로 배포될 수 있다. 한 결과를 예를 들면, GUI가 시스템 안에서 설계된 것이 아니므로, GUI를 붙이기 위한 한가지 이상의 선택이 Python 프로그램에 있다.

C++
Java에 대해서 이야기 한 대부분이 C++에도 적용된다. Python코드가 Java 코드보다 3-5배 짧으며, C++코드에 비해 5-10배 짧다!! 일 예로 한명의 Python 프로그래머는 C++프로그래머 두 명이 1년에 끝낼 수 없는 일을 두달만에 끝낼 수 있다. Python은 C++로 쓰여진 코드를 사용하는 접착 언어로 빛을 발한다.

Common Lisp and Scheme
이들 언어는 동적인 의미해석에서 Python에 가깝다. 그러나 구문해석 접근은 너무 달라서 매우 심한 논쟁거리가 될 만한 비교가 된다: Lisp의 구문적인 부족함이 장점일까 단점일까? Python은 Lisp과 같은 내성적인 능력(capabilities)이 있고, Python 프로그램은 아주 쉽게 프로그램 부분을 구성해서 실행할 수 있다는 것을 밝혀야겠다. 일반적으로, 실세계 실체가 결정적이다: Common Lisp은 크다(어떠한 관점에서도 그렇다). Scheme 세계는 많은 어울리지 않는 버전들로 나누어져있다. 그에 반해서 Python은 하나이고, 무료이고, 작게 구현되었다. 더 자세한 Scheme와의 비교에 관해선 Moshe Zadka가 쓴 Python vs. Scheme을 보라.

01. 파이썬 3.0에서 새로워 진 점

출처 : http://bluekyu.textcube.com/111

작성자 : Bluekyu

** 파이썬 3를 완벽하게 이해하지 못한 부분에서 오류가 발생하는 것을 최소화 하기 위해 모든 부분에서 직접 실행을 해보고 적었습니다. 그래도 오류가 발생한다면 죄송합니다.

1. PEP 3105 : Print Is A Function

Print 구문이 함수로 바뀌었습니다. 함수의 원형은 아래와 같이 변경되었습니다.

```
print([object, ...], *, sep=' ', end='\n', file=sys.stdout)
```

예를 들면 print "Hello, Python.", "I like Python" 과 같이 쓴 코드를 3.0 버전으로 다시 쓰면 print("Hello, Python.", "I like Python") 처럼 쓸 수 있습니다. 그리고 sep 키워드의 값은 앞에 나온 객체들을 붙여 쓰는 방식을 지정합니다. 기본값이 ' ' 으로 되어 있으므로 위의 예에서는 Hello, Python. I like Python 처럼 쓰여지지만 sep='' 라고 지정하면 Hello, Python.I like Python 과 같이 쓰여집니다. 예전에는 연속해서 나오는 문자열을 붙여서 쓸려면 print 문에서 바로 해결이 안됐지만 sep 의 도입으로 가능해졌습니다.

참고로 더 이상 print 함수에서는 "softspace"가 지원되지 않습니다.(softspace 란 print 문에서 컴마를 사용해서 한 줄에 입력할 때 자동으로 출력값 사이에 스페이스를 넣어주는 기능입니다.)

2. Views And Iterators Instead Of Lists

몇몇의 API들이 더 이상 리스트를 리턴하지 않게 됐습니다. 먼저, 사전 객체의 메소드인 dict.keys(), dict,itmes(), dict,values() 는 views 객체를 리턴하게 됩니다. 따라서 k = d.keys(), k.sort() 와 같은 기법은 사용할 수 없게 됩니다. 대신에 k = sorted(d) 를 사용해야 합니다.

참고로 views 객체는 사전에 존재하는 객체인데, 사전의 변경이 이들의 객체에 직접 반영됩니다. 예를 들어 a = {'one' : 1, 'two' : 2}를 하고, b = a.keys() 를 했을 때 a 값을 변경하게 되면 그 변경이 b 에도 영향을 미칩니다. 이 외에도 views 객체는 개개의 데이터를 산출하도록 반복될(iterated) 수 있고, 멤버쉽 테스트를 지원합니다. 그리고 dict.iterkeys() 와 dict.iteritmes(), dict.itervalues() 메소드가 더 이상 지원되지 않습니다. 따라서 이들을 계속 쓰려면 keys나 items, values 메소드가 리턴하는 views 객체에 iter() 함수를 사용해야 할 듯 싶습니다. 또, map()과 filter()는 반복자를 리턴합니다. 따라서 리스트를 반드시 얻어야 한다면 list() 함수로 리스트를 구해야 할 것입니다. 그리고 range() 함수가 임의의 크기를 갖는 값과 작동하는 경우를 제외하고는 xrange() 처럼 작동합니다. 따라서 xrange() 는 더 이상 존재하지 않게 됩니다. 마지막으로 zip()도 반복자를

리턴합니다.

3. Ordering Comparisons

파이썬 3.0 에서는 비교 방식이 더 단순해졌습니다. 일반적으로 의미 없는 비교(<, >, <=, >=)를 가질 때 TypeError 예외를 일으키도록 변경 됐습니다. 예를 들어 1 < 'a', 0 < None 과 같이 예전에는 False 를 리턴했지만 3.0 부터는 에러를 일으킵니다. 따라서 각각의 요소들이 비교될 수 없는 리스트끼리의 비교는 에러를 일으키게 됩니다. 단, = 과 != 비교는 이 규칙을 따르지 않기 때문에 여전히 비교가 가능합니다.
내장 함수인 sorted() 와 list.sort() 함수가 더 이상 cmp 인자를 받지 않습니다. 대신 key 인자를 사용해야 합니다. 그리고 key 와 reverse 는 오직 키워드만 받는 인자가 되었습니다. 마지막으로 cmp 함수와 __cmp__() 메소드가 더 이상 지원되지 않습니다. 따라서 __lt__(), __eq__(), __hash__() 나 다른 비교 함수를 사용해야 합니다. 그리고 cmp() 함수가 필요하다면 (a > b) - (a < b) 와 같이 사용하면 cmp() 와 동등하게 사용할 수 있습니다.

4. Integers

long 이 int 로 이름이 변경되었습니다. 즉, 모든 내장 정수는 int 형 하나 뿐입니다. 그러나 여전히 long 형처럼 사용할 수 있습니다. 또, 1/2 와 같은 분수형 표현이 실수형을 리턴합니다. 만약, 예전처럼 정수값을 얻고 싶다면 1//2 와 같이 사용해야 합니다. 그리고 정수형이 더 이상 제한이 없기 때문에 sys.maxint 상수가 제거되었습니다. 대신 sys.maxsize 를 사용해서 예전의 sys.maxint 와 같은 값을 얻을 수 있습니다. long 형이 제거 되었기 때문에 repr() 을 사용함으로써 long 형 뒤에 생기는 L 은 더 이상 생기지 않습니다. 마지막으로 8진수 형태가 0123 이 아니라 0o123 으로 변경 되었습니다.

5. Text Vs. Date Instead Of Unicode Vs. 8-bit

1) 파이썬 3.0부터는 문자열을 다루는 방식이 달라졌습니다. 예전과는 달리 모든 텍스트는 유니코드가 되었습니다. (단, 인코딩된 유니코드는 바이너리 데이타로 표현 됩니다.) 텍스트로 사용된 타입은 str 클래스이고, 데이타로 사용된 타입은 바이트 클래스입니다. 만약, 실질적인 사용법을 알고 싶다면 아래를 참조하세요.

- http://diveintopython3.org/strings.html#byte-arrays

2) 파이썬 2 버전과 가장 큰 차이점이라고 하면 텍스트와 데이터를 섞으려는 시도가 있을 경우 TypeError 를 일으킵니다. 파이썬 2버전에서는 유니코드와 8비트 문자열을 섞으려고 하면 8비트 문자열이 아스키 코드이었다면 작동을 했지만 8비트 문자열에 비 아스키 코드가 있었다면 UnicodeDecodeError 를 일으켰습니다.

3) 모든 텍스트가 유니코드인 만큼 더 이상 u'...' 표현은 사용할 수 없습니다. 대신 바이너리 데이타에서는 반드시 b'...' 표현을 사용해야 합니다.

4) 텍스트와 바이트가 섞일 수 없기 때문에 서로 전환을 해야 하는데, 텍스트에서 바이트로는

str.encode() 나 bytes(s, encoding = ...) 를 사용하면 되고, 바이트에서 텍스트로는 bytes.decode()나 str(b, encoding=...)을 사용하면 됩니다.

5) 텍스트 표현이 변경 불가능인 것처럼 바이트 표현도 변경 불가능 입니다. 대신, bytearray라는 타입이라는 변경 가능한 타입이 있습니다. bytes 를 받아 들이는 거의 대부분의 API 들이 bytearray 도 받아들입니다. 변경 가능한 API 는 collections.MutableSequence 에 기본을 둡니다.

6) raw 형식의 문자열에서 모든 백슬래쉬가 문자 그대로 해석됩니다. 예전에는 ur'\u20ac' 를 유로 문자로 해석했지만 3 버전부터는 r'\u20ac' 를 6개의 문자로 해석을 합니다. 단, '\u20ac' 의 경우에는 여전히 유로 문자로 해석됩니다.

7) 2.3 버전에서 도입된 basestring 타입이 제거 됩니다. 대신 str를 사용하면 됩니다. 2to3 에서는 str로 변환을 합니다.
(basestring 은 2버전에서의 문자열이나 유니코드와 인스턴스 비교를 할 때 사용된 타입이라고 합니다.)

8) 텍스트 파일로 열리는 파일들은 (메모리에서)문자열과 (디스크에서)바이트 사이를 매치하는 인코딩을 사용합니다. 그리고 b 모드를 사용해서 열은 바이너리 파일은 메모리에서 바이트를 사용합니다. 즉, 파일을 잘못된 모드나 인코딩을 사용해서 열게 되면 I/O가 잘못된 데이터를 넘겨주는 것이 아니라 실패를 일으킵니다. 또, 유닉스 사용자들도 올바른 모드를 사용해야 합니다. (파이썬 책에 보니까 유닉스는 모든 파일을 2진(바이너리) 파일로 처리한다고 되어 있네요. 그런데 문자열과 바이너를 매치 시켜서 사용한다고 하니까 무조건 모드를 적어주어야 하겠네요.) 그리고 인코딩도 고려해야 하기 때문에 디폴트 인코딩에 의존하는 것은 자제해야 해야 합니다. (언어 환경 변수를 설정할 수 있는 유닉스 계열의 경우 대부분(전부가 아닌) 디폴트 인코딩이 UTF-8 이라고 하는데, 그래도 이것에 의존해서는 안되겠죠.) 그리고 인코딩을 지정해서 쓰기 때문에 더 이상 codecs 모듈에 있는 인코딩 스트림들은 사용할 필요가 없게 되었습니다.

9) 파일 이름들은 유니코드를 사용하게 됩니다. 그렇기 때문에 바이트 문자열을 사용하는 일부 플랫폼에서는 문제를 일으킬 수 있습니다. 대신 다른 방법으로, 파일 이름을 받아들이는 대부분의 API(open() 이나 os 모듈)들은 바이트 객체도 받아들입니다. 그리고 몇몇의 API들도 바이트 리턴값을 요구하는 방법을 가지고 있습니다. 즉, os.listdir()은 인자가 바이트일 경우 바이트의 리스트를 리턴합니다. 참고로, os.listdir() 이 문자열을 리턴할 때 디코드될 수 없는 파일 이름들은 UnicodeError 예외를 일으키보다는 생략됩니다.

10) os.environ 이나 sys.argv 와 같은 일부 시스템 API의 경우, 바이트가 기본 인코딩으로 해석될 수 없을 때 문제를 일으킬 수 있습니다. 이 경우에는 언어 환경 변수를 설정한 뒤 재실행 하는 것이 가장 좋은 방법입니다.

11) PEP 3138
비 아스키 코드 문자열이 repr 에 의해서 더 이상 escape 되지 않습니다. 그러나 출력될 수 없는

control character 와 code points 는 여전히 escape 됩니다. 즉, 예전에는 repr('한글') 이라고 하게 되면 "'한글'" 이 아니라 "'\\xxx\\xxx\\xxx\\xxx'" 와 같이 표현 되었습니다. 그러나 이제는 "'한글'" 로 표현이 됩니다. 다만, '\n' 과 같은 문자는 여전히 "'\\n'" 로 표시가 됩니다.

12) PEP 3120, PEP 3131
기본 소스 인코딩은 UTF-8 입니다. 그리고 비 아스키 문자열들도 변수로 허용이 됩니다. 다만, 표준 라이브러리들은 오직 아스키로만 남아 있습니다. (However, the standard library remains ASCII-only with the exception of contributor names in comments.)

13) StringIO 와 cStringIO 모듈이 제거 되었습니다. 대신에, io 모듈을 임포트해서 io.StringIO 와 io.BytesIO 를 사용하면 됩니다.

6. Overview Of Syntax Changes

< 새로운 문법 >
1) 먼저, PEP 3107 내용입니다. 여기서는 함수 인자와 리턴 값의 주석을 다는 문법이 추가되었습니다. 사용하는 방법은

```
def foo(a: expression, b: expression = 5) -> expression:
   ...
```

와 같이 인자 오른쪽에 : expression 이라고 사용하면 되고, 리턴 값의 주석은 -> expression 과 같이 사용하면 됩니다.
(참고로 expression 은 의미 그대로 식이기 때문에 1+2 와 같이 파이썬 식을 넣으면 주석을 볼 때 식의 결과가 나옵니다.)

그리고 위 주석들은 함수의 __annotations__라는 속성에 사전 형식으로 추가가 됩니다. 다만, 문법의 소개에서는 실행 중에 __annotations__ 속성을 사용해서 주석을 보는 경우 외에는 이러한 주석 방식은 의미가 없다고 합니다. 그리고 이러한 의도는 메타 클래스나 장식자, 프레임 워크를 통한 사용을 권장하는 것이라고 합니다.

2) 다음으로 PEP 3102 입니다. 함수의 인지가 키워드로만 사용할 수 있도록 하는 문법이 추가되었습니다. 예를 들어서

```
def f(a, b, c=key) :
   ...
```

가 있을 때, c가 오직 키워드로만 사용되게 하고 싶으면, c 인자 앞에 가변인수(*args)를 넣어주면 됩니다. 예전 버전에서는 가변인수가 앞에 오게 되면 에러를 일으켰는데, 이러한 문법을 허용하고, 가변 인수 뒤에 오는 키워드 인자들은 오직 키워드만 받게 됩니다. 또, 가변인수를 받고 싶지 않을 경우에는 * 만 사용해서 가변인수 없이 오직 키워드 인자만 사용할 수 있습니다. 사용의 예는 다음과

같습니다.

```
def f(a, b, *, c=key) :
  ...
```

3) PEP 3104 내용입니다. 이 내용에서는 nonlocal 이라는 문(예약어)가 등장했습니다. 이름 그대로 이것은 글로벌 지역이 아닌 바깥 지역을 검색할 수 있게 합니다. 열혈강의 파이썬에 있는 예 중에서

```
>>> def bank_account1(initial_balance) :
   balance = initial_balance
   def deposit(amount) :
       balance = balance + amount
       return balance
   def withdraw(amount) :
       balance = balance - amount
       return balance
   return deposit, withdraw

>>> d, w = bank_account1(100)
>>> print(d(100))
```

와 같은 코드가 있을 때, balance = balance + amount 줄에서 에러가 발생하게 됩니다. balance 는 지역 내에 선언된 것도 아니고, 글로벌에 선언된 것도 아니기 때문에 중간에 있는 변수를 참조 할 수 없기 때문입니다. 그래서 책에서는 이것을 해결하기 위해 balance 를 리스트로 바꾸어서 해결을 했습니다. 그러나 nonlocal 의 추가로 이것을 간단하게 해결할 수 있게 되었습니다.

```
>>> def bank_account1(initial_balance) :
   balance = initial_balance
   def deposit(amount) :
       nonlocal balance
       balance = balance + amount
       return balance
   def withdraw(amount) :
       nonlocal balance
       balance = balance - amount
       return balance
   return deposit, withdraw

>>> d, w = bank_account1(100)
>>> print(d(100))
200
```

와 같이 nonlocal balance 를 중간에 삽입해주게 되면 함수 중간에 있는 balance 변수를 참조할 수 있게 됩니다.

4) PEP 3132 내용입니다. 이 내용에서는 반복 언팩킹 기능이 확장되었습니다. 함수에서 인자를 받을 때 여분의 인자는 *arg 와 같이 받는 형식처럼 언팩킹을 할 때 여분의 인자를 받을 수 있도록 바뀌었습니다. 예를 들어

```
a, *rest, b = range(5)
```

와 같이 입력을 해주면 시퀀스의 값을 3개로 나누어서 언팩킹을 해줍니다. a=0, rest=[1,2,3], b=4 로 대입이 되면서 언팩킹이 됩니다.

5) PEP 0274 내용입니다. 리스트 내장처럼 사전 내장이 추가되었습니다. 쓰는 방식은 아래와 같이 사용하면 됩니다.

```
{k: v for k, v in stuff}
```

위 방식은 dict(stuff)와 같은 기능을 하는데, 위 방식이 더 유용하다고 PEP 내용에서 입증 되었다고 하네요.

6) set 자료형의 표현이 바뀌었습니다. 예전 버전에서는 set(['a', 'b'])와 같이 표현이 되었으나 3버전부터는 {'a', 'b'} 와 같이 표현이 됩니다. 다만, 주의 할 점은 {} 방식은 사전을 의미합니다. 빈 set 형을 사용하려면 set()으로 사용해야 합니다. 또, set 내장이 지원 됩니다. 즉, {x for x in stuff}와 같이 사용할 수 있습니다. set(stuff)와 같은 기능이지만 더 유연하다고 합니다.

7) 8진법 표현 형식이 바뀌었습니다. 0o720와 같이 사용되며 0720의 표현은 사라졌습니다. 이 표현은 2.6에서 이미 적용되었습니다.

8) 2진법 형식이 생겼습니다. 0b1010과 같이 사용되며 이에 상응하는 내장 함수로 bin() 이 있습니다. 이 표현 역시 2.6에서 이미 적용 되었습니다.

9) 문자열에서 이미 말했 듯이 바이트 형식이 소개되었습니다. b'...' 나 B'...'로 표현이 되며, 이에 상응하는 함수로 bytes() 가 생겼습니다.

〈 바뀐 문법 〉

1) PEP 3109, PEP 3134 내용입니다. raise 문법이 새롭게 바뀌었습니다. 자세한 것은 아래에 설명 되어 있습니다.

2) as 와 with 가 새로운 예약어가 되었습니다.

3) True 와 False, None 이 예약어가 되었습니다.

4) PEP 3110. 예외 처리 문법에서 except exc, var 형식이 except exc as var 형태로 바뀌었습니다. 이 내용도 아래에 있습니다.

5) PEP 3115. 메타 클래스 문법이 바뀌었습니다.

```
class C:
   __metaclass__ = M
   ...
```

대신에

```
class C(metaclass = M) :
   ...
```

형식으로 바뀌었습니다. 따라서 더 이상 __metaclass__ 변수는 지원하지 않습니다.

6) 리스트 내장 문법에서 [... for var in item1, item2, ...] 이 더 이상 지원되지 않습니다. 대신에 [... for var in (item1, item2, ...)] 를 사용하면 됩니다. 그리고 리스트 내장은 다른 의미(semantics)를 가집니다. 리스트 구조 내부의 발생자 표현 문법과 더 가깝습니다. 또, surrounding scope 내의 loop control 변수가 더 이상 누수 되지 않습니다.

7) 생략(ellipsis) 구문이 점 표현 방식(...)으로 어디서든 사용될 수 있습니다. 예전 버전까지는 슬라이싱에서만 사용이 가능했습니다. 그리고 반드시 ... 써야 하며, . . . 방식은 안 됩니다.(예전까지는 됐습니다.)

< 제거된 문법 >
1) PEP 3113. 튜플 인자 언패킹 문법이 제거 되었습니다. 예를 들어서 def foo(a, (b, c)) : ... 를 더 이상 쓸 수 없고, 대신에 def foo(a, b_c) : b, c = b_c ... 와 같이 써야 합니다. 람다 함수에서도 마찬가지 입니다.

2) backtick 기호(`)가 사라졌습니다. 대신에 repr 을 사용해야 합니다.

3) <> 기호가 사라졌습니다. 대신에 != 사용해야 합니다.

4) exec 키워드가 사라졌습니다. 대신 함수로 남아 있게 됩니다. 또한, exec()는 더 이상 stream 인자를 받지 않습니다. 예를 들어서 print 1 이 저장되어 있는 파이썬 파일을 변수 f 에 열었을 때, exec f 를 쓰게 되면 print 1 을 바로 읽고 실행이 되었으나 더 이상 이런 것이 불가능 하고, 대신에 exec(f.read()) 와 같이 사용해야 합니다.

5) 정수와 문자열에서 l, L, u, U 가 더 이상 지원되지 않습니다.

6) from (module) import * 문법은 모듈 레벨에서만 실행이 됩니다. 더 이상 함수 내에서는 불가능 합니다. 다만, from (module) import (name) 은 사용이 가능합니다.

7) 상대적 임포트 문법으로 from .[module] import name 만 가능합니다. 점(.) 으로 시작되지 않는 임포트 구문은 절대적 임포트로 해석됩니다.(2.5에서 상대적 임포트가 나왔을 때 점 없이도 상대적 임포트가 가능 했었습니다.)

8) Classic Class 가 사라졌습니다.

7. Changes Already Present In Python 2.6

이 부분의 내용들은 파이썬 2.6에도 이미 적용된 내용들입니다.

1) PEP 343 : The 'with' statement
with 구문이 추가되었습니다. 더 이상 __future__ 에서 임포트 되지 않습니다. 자세한 사항은 파이썬 2.5에 기술된 내용과 같으므로 http://bluekyu.textcube.com/108 를 참고 해주세요.

2) PEP 366 : Explicit Relative Imports From a Main Module
파이썬을 -m 옵션을 추고 실행하면 스크립트로 모듈을 실행 한다고 합니다. 그런데 파이썬 패키지 내에 있는 모듈을 실행 시키면 상대적 임포트가 제대로 작동을 안한다고 합니다. 이 부분을 해결하기 위해 모듈에 __package__ 속성을 추가 했습니다. 이 속성이 나타날 때, 상대적 임포트는 __name__ 속성 대신에 이 속성의 값에 대해서 상대적이게 됩니다.

3) PEP 370 : Per-user site-packages Directory
파이썬에서 개인 유저마다 site-packages 디렉토리를 가질 수 있도록 하였습니다. 그 전에는 site-packages 가 전체에서 쓸 수 있도록 되어 있었는데, 개인 마다도 쓸 수 있도록 소개가 되었습니다. site-packages 는 표준 라이브러리가 아닌 3rd 라이브러리들이 설치되는 경로입니다. 우분투나 쿠분투에는 site-packages 가 아니라 dist-packages 로 되어 있습니다. (이것 때문에 몇몇 프로그램에서는 오류가 발생한다는 얘기도 있네요. 그리고 site.py 를 보니까 파일 내부에 데비안 계열의 경우 dist-packages 에 저장이 된다고 쓰여져 있습니다.)

디렉토리는 플랫폼에 따라서 경로가 다르게 나타납니다. Unix 와 Mac 에서는 ~/.local/ 이고, 윈도우즈에서는 %APPDATA%/Python 입니다. 이 내부에 unix, mac 은 lib/python2.6/site-packages 과 같이, 윈도우에서는 Python26/site-packages 와 같이 나타납니다. 그리고 이 경로를 원하지 않을 경우 PYTHONUSERBASE 환경변수를 통해서 변경을 할 수 있고, 파이썬을 실행 할 때 -s 옵션을 주면 사용을 하지 않게 됩니다. (-S 는 개인 유저 경로가 아닌 전체 경로에 있는 site-packages 를 사용하지 않을 때 사용합니다.)

마지막으로 site.py 를 수정해서 site-packages 의 경로를 수정할 수도 있습니다. 자세한 사용 방법은 site.py를 보시면 됩니다.

4) PEP 371 : The multiprocessing Package
multiprocessing 패키지가 추가 되었습니다. 패키지 이름 그대로 새로운 프로세스를 만들 수 있습니다. threading 모듈과 유사하다고 합니다. 자세한 것은 모듈을 직접 참고하세요.

5) PEP 3101 : Advanced String Formatting
문자열 포맷이 향상되었습니다. str.format() 형식으로 사용이 가능합니다만 예전의 % 기호도 현재 지원하고 있지만 3.2 부터는 없애는 것으로 되어 있습니다. 그리고 2.6과 달리 3 에서는 문자열이 str 타입이므로 str 타입만 지원하고 bytes 타입은 지원하지 않습니다. 자세한 사항은 아래 글에 다가

정리를 해두었습니다.

- http://bluekyu.textcube.com/136

6) PEP 3105 : print As a Function
위에서 자세히 설명한 부분입니다.

7) PEP 3110 : Exception-Handling Changes
위에서 짧게 언급했고, 아래에서 더 자세히 소개합니다.

8) PEP 3112 : Byte Literals
위에서 언급한 내용이고, 문서 상에는 C 와 어떻게 관련되어 있는지 나와 있습니다. 자세한 것은 직접 참조해주세요.

9) PEP 3116 : New I/O Library
io 모듈이 파일 입출력의 새로운 표준이 되었습니다. sys.stdin, sys.stdout, sys.stderr 는 io.TextIOBase 의 인스턴스입니다. 내장 함수인 open() 도 io.open() 처럼 사용 되고 추가적인 키워드도 붙었습니다. 그리고 유효하지 않은 mode 인자일 경우 IOError 가 아니라 ValueError 를 발생시킵니다. 더 자세한 것은 직접 참조해주세요.

10) PEP 3118 : Revised Buffer Protocol
buffer() 내장 함수가 없어지고, 유사한 기능으로 memorryview() 가 내장으로 제공됩니다. 자세한 것은 직접 참조.

11) PEP 3119 : Abstract Base Classes
객체 지향과 관련된 내용인 것 같은데, 잘 모르겠네요. 자세한 것은 직접 참조.

12) PEP 3127 : Integer Literal Support and Syntax
정수와 관련해서 위에서 언급한 내용입니다.

13) PEP 3129 : Class Decorators
클래스에서도 장식자를 메소드 장식자처럼 사용할 수 있게 되었습니다.

14) PEP 3141 : A Type Hierarchy for Numbers
수에 대한 체계가 변경 되었습니다. 위에서 언급하지 못한 ABC 가 새롭게 바뀌면서 이 부분도 바뀐 것 같습니다. 가장 일반적인 ABC 는 Number 입니다. Complex 는 Number 의 서브클래스 입니다. Real 은 Complex 의 서브클래스 입니다. Rational 은 Real 의 서브클래스 입니다. Integer 는 Rational 의 서브클래스 입니다. 그리고 분수와 관련된 모듈인 fractions 모듈이 추가 되었습니다. 더 자세한 것은 직접 참조하세요.

8. Library Changes

라이브러리들이 많이 바뀌었습니다. 이 부분에 대해서는 너무 많아서 필요한 몇 개만 적겠습니다. 나머지는 PEP 3108을 참조해주세요.

1) md5 모듈과 sha 모듈이 hashlib 모듈로 대체 되었습니다.

2) cPickle 모듈 이름이 _pickle 로 바뀌었습니다.

3) StringIO 와 cStringIO 의 클래스가 io 모듈에 추가되었습니다.

4) 여러 그룹들이 재그룹화 되었습니다.

< dbm 패키지 >

Current Name	Replacement Name
anydbm	dbm.__init__
dbhash	dbm.bsd
dbm	dbm.ndbm
dumbdm	dbm.dumb
gdbm	dbm.gnu
whichdb	dbm.__init__

< urllib 패키지 >

Current Name	Replacement Name
urllib2	urllib.request, urllib.error
urlparse	urllib.parse
urllib	urllib.parse, urllib.request, urllib.error
robotparser	urllib.robotparser

< tkinter 패키지 >

Current Name	Replacement Name
Dialog	tkinter.dialog
FileDialog	tkinter.filedialog
FixTk	tkinter._fix
ScrolledText	tkinter.scrolledtext
SimpleDialog	tkinter.simpledialog
Tix	tkinter.tix
Tkconstants	tkinter.constants
Tkdnd	tkinter.dnd
Tkinter	tkinter.__init__
tkColorChooser	tkinter.colorchooser
tkCommonDialog	tkinter.commondialog
tkFileDialog	tkinter.filedialog
tkFont	tkinter.font
tkMessageBox	tkinter.messagebox
tkSimpleDialog	tkinter.simpledialog
turtle	tkinter.turtle

다음 내용은 PEP 3108에 없는 내용입니다.

1) sets 모듈이 제거 되었습니다. 내장 함수인 set 을 쓰면 됩니다.

2) sys 모듈이 정리되었습니다. sys.exitfunc(), sys.exc_clear(), sys.exc_type, sys.exc_value, sys.exc_traceback 이 제거되었습니다.

3) array.array 타입이 정리되었습니다. read(), write() 메소드가 없어지고, fromfile(), tofile() 을 사용하면 됩니다. 그리고 c 타입 코드가 없어지고 b 나 u 타입 코드를 사용하면 됩니다.

4) operator 모듈이 정리되었습니다. sequenceIncludes() 와 isCallable() 이 없어 졌습니다.

5) thread 모듈이 정리되었습니다. acquire_lock() 과 release_lock()이 없어지고, acquire()와 release() 를 사용하면 됩니다.

6) random 모듈이 정리되었습니다. jumpahead() 가 제거 되었습니다.

7) new 모듈이 없어졌습니다.

8) tmpfile 모듈을 위해 os.tmpnam(), os.tempnam(), os.tmpfile() 이 제거 되었습니다.

9) tokenize 모듈이 바이트와 작동하도록 바뀌었습니다. 주 입력 지점(Main Entry Point)는 generate_tokens 대신 tokenize.tokenize()로 바뀌었습니다.

10) string.letters와 string.lowercase, string.uppercase 가 사라졌습니다. 대신에, string.ascii_letters 등을 사용하면 됩니다.

11) 모듈 __builtin__ 이름이 builtins 로 바뀌었습니다. global 이름 공간에서 볼 수 있는 __builtins__ 변수는 바뀌지 않았습니다. builtin 을 수정하려면 __builtins__ 가 아닌 builtins 를 사용하면 됩니다.

9. Changes To Exceptions

예외 처리 부분이 많이 바뀌었습니다.

1) PEP 0352
모든 예외는 BaseException 을 상속해야 합니다. 결과적으로 문자열 예외는 사라졌습니다.

2) 거의 모든 예외는 Exception 을 상속합니다. 즉, BaseException 은 SystemExit 나 KeyboardInterrup 와 같은 상위 레벨을 다루기 위한 예외 클래스의 기반입니다. 대부분의 예외들을 처리하기 위해서 제안하는 방식으로는 except Exception 을 사용하면 됩니다.

3) StandardError 는 제거 되었습니다.

4) 예외는 더 이상 시퀀스로 작동하지 않습니다. 대신에 args 속성을 사용하면 됩니다.
(열혈강의 파이썬 p. 481 의 예제에서 16번째 줄인 아래의 코드가 더 이상 작동이 안 된다는 의미입니다.)

```
print(a[0], a[0].__class__.__name__, id(a[0]))
```

5) PEP 3109
raise Exception, args 대신에 raise Exception(args) 를 사용해야 합니다. 추가적으로 더 이상 traceback 명시할 수 없습니다. 만약, 이것을 하려면 __traceback__ 속성에 직접 할당 할 수 있습니다.

6) PEP 3110
except Exception, variable 대신 except Exception as variable 을 사용해야 합니다. 만약, 두 가지 예외를 잡아내려면 튜플을 사용하면 됩니다. 그리고 except 블록이 사라질 때 variable 변수는 제거가 됩니다. 즉, 아래와 같은 코드는

```
except E as N :
   foo
```

다음과 같이 변환될 수 있습니다.

```
except E as N :
   try :
       foo
   finally :
       N = None
       del N
```

따라서 예외 처리 구문 이후에도 값을 계속 참조하기를 원한다면 예외 처리에서 변수의 이름을 다르게 지정해주어야 합니다.

7) PEP 3134
예외의 연쇄 처리 방식이 변경 되었습니다. 두 가지 방식이 있는데, 하나는 묵시적(implicit) 방식이고, 다른 하나는 명시적(explicit) 방식입니다. 묵시적 연쇄(2차 예외)는 except 나 finally 구문에서 발생하게 됩니다.

```
>>> try :
...     raise Exception('first')
... except :
...     raise Exception('second')
...
Traceback (most recent call last):
 File "<stdin>", line 2, in <module>
Exception: first

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
 File "<stdin>", line 4, in <module>
Exception: second
```

위와 같이 try 구문에서 에러가 발생 했을 때, except 구문에서 처리를 하게 되는데, 2버전의 경우 except 에서 에러가 발생하면 try 에서 발생한 에러는 무시가 됩니다. 그러나 3버전부터는 위와 같이 첫번째에서 에러가 났고, 처리 도중에 두번째 에러가 발생 했다는 것을 알려줍니다. 이렇게 첫번째에서 발생한 에러가 소멸하는 것을 방지하기 위해 첫번째 에러는 두번째 에러 클래스의 __context__ 속성에 저장이 됩니다.

다음으로 명시적 연쇄입니다. 명시적 예외는 raise EXCEPTION from CAUSE 의 구문에 의해서 발생이 되고, 이 구문은
exc = EXCEPTION
exc.__cause__ = CAUSE
raise exc
와 같은 의미를 가집니다. 즉, 명시적 예외가 발생하면 예외를 일으킨 CAUSE를 EXCEPTION 예외 클래스의 __cause__ 속성에 저장을 합니다. 예를 하나 들면

```
>>> try :
...     1/0
... except Exception as var :
...     raise Exception('1/0 is Error!') from var
...
Traceback (most recent call last):
 File "<stdin>", line 2, in <module>
ZeroDivisionError: int division or modulo by zero

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
 File "<stdin>", line 4, in <module>
Exception: 1/0 is Error!
```

위와 같이 try 에서 예외가 발생 했는데, 그 예외를 except 에서 잡아내서 명시적으로 var 때문에 예외가 일어 났다는 것을 알려주고 있습니다. 즉, 묵시적 연쇄는 의미 그대로 파이썬 내부에서 2차 에러를 발생 시킨 1차 에러의 원인을 알려주게 됩니다. 그러나 명시적 연쇄는 프로그래머가 강제적으로 2차 에러와 함께 1차 에러의 원인을 알려주게 끔 합니다. (참고적으로 IDLE 에서는 이러한 연쇄들이 나타나지 않습니다.)

8) PEP 3134
예외 객체의 traceback 이 __traceback__ 속성 내에 저장이 됩니다. 예전에는 traceback 의 객체를 얻기 위해서는 sys.exc_traceback 이나 sys.exc_info()[2] 를 사용해야 했지만 이제는 간단하게 __traceback__ 속성만 참조를 하면 됩니다. 사용 방법은 아래와 같이 쓰면 됩니다.

```
def do_logged(file, work):
   try:
       work()
   except Exception, exc:
       write_exception(file, exc)
       raise exc

from traceback import format_tb

def write_exception(file, exc):
   ...
   type = exc.__class__
   message = str(exc)
   lines = format_tb(exc.__traceback__)
   file.write(... type ... message ... lines ...)
   ...
```

9) 윈도우즈에서 확장 모듈을 로드하는 데 실패 했을 때, 몇 가지 예외 메시지가 향상되었습니다.

10. Miscellaneous Other Changes

〈 Operators And Special Methods 〉
1) == 연산자가 NotImplemented 를 리턴하는 것을 제외하고, != 는 == 의 반대입니다.

2) 언바운드 메소드(unbound methods)의 개념이 사라졌습니다. 클래스 속성으로 메소드를 참조하게 되면 평범한 함수 객체를 얻게 됩니다.

3) __getslice__(), __setslice__(), __delslice__() 이 사라졌습니다. a[i:j] 문법은 a.__getitem__(slice(i, j)) 으로 전환 됩니다. 만약, 할당이나 삭제로 사용이 될 때는 __setitem__() 이나 __delitem__() 으로 바뀝니다.

4) PEP 3114. 반복자에서 사용되던 표준 next() 메소드의 이름이 __next__() 로 변경되었습니다. 즉, 다른 표준 메소드처럼 언더바를 사용합니다. 그리고 반복자를 호출하기 위해 a.next() 대신 내장함수인 next(a) 를 사용해서 반복자를 호출할 수 있습니다.

5) __oct__() 와 __hex__() 메소드가 제거되었습니다. 대신, oct() 와 hex() 메소드는 __index__() 메소드를 사용합니다.

6) __members__ 와 __methods__ 메소드가 제거되었습니다.

7) 함수 속성의 이름이던 func_X 가 __X__ 형식으로 변경되었습니다. 따라서, 이들의 이름이 함수 속성의 이름 공간에서 해제 되었습니다.

8) __nonzero__() 가 __bool__() 로 변경되었습니다.

〈 Builtins 〉
1) PEP 3135. 내장 함수인 super() 가 인자 없이 발생할 수 있게 되었습니다. 그리고 이것이 클래스 구문 내에 정의된 인스턴스 메소드 내에 있다고 했을 때, 올바른 클래스와 인스턴스가 자동적으로 선택되어집니다. 만약, 인자와 같이 쓰여지게 되면 예전의 super() 처럼 사용됩니다.

2) PEP 3111. raw_input() 이 input() 으로 이름이 변경되었습니다. 즉, input() 은 sys.stdin 으로부터 라인을 읽고 새 라인(newline)을 없앤 후 값을 리턴합니다. 그리고 입력이 이르게(prematurely) 종료되었다면 EOFError 를 발생합니다. 예전에 input() 함수를 사용하려면 eval(input()) 을 사용하면 됩니다.

3) __next__() 메소드를 호출하기 위해 내장 함수 next() 가 추가되었습니다.

4) intern() 이 sys.intern() 이 되었습니다.

5) apply() 가 제거 되어습니다. 즉, apply(f, args) 대신 f(*args) 로 사용하면 됩니다.

6) callable() 이 제거 되었습니다. callable(f) 대신 isinstance(f, collections.Callable) 를 사용하면 됩니다. operator.isCallable() 함수도 제거 되었습니다.

7) coerce() 가 제거되었습니다. classic 클래스가 사라졌기 때문에 이 함수는 더 이상 목적을 제공하지 않습니다.

8) execfile() 이 제거되었습니다. execfile(fn) 대신에 exec(open(fn).read()) 를 사용하면 됩니다.

9) file 타입이 제거 되었습니다. open() 을 사용하면 됩니다. io 모듈 내에 있는 open 이 리턴할 수 있는 몇 가지 다른 종료의 스트림(stream) 이 있습니다.

10) reduce() 가 제거되었습니다. functools.reduce() 를 사용하면 됩니다. (문서 상에는 "그런데 그것이 정말로 필요하다면, 99%로 명확한 for 루프가 더 읽기 좋다"고 합니다.)

11) reload() 가 제거되었습니다. 대신, imp.reload() 를 사용하면 됩니다.

12) dict.has_key() 가 제거되었습니다. 대신, in 연산자를 사용하면 됩니다.

11. Build and C API Changes
문서 상에 이 부분이 시간 압박 때문에 매우 불완전하다고 합니다. 이 부분에 관해서는 직접 참고하시길 바랍니다.

이 것으로 파이썬 3의 새로운 점에 관해서 글을 마칩니다. 만약, ABC 나 메타클래스와 같이 본문에서 설명이 부족한 부분에 대해서는 아래 두 글을 참고해 주세요.

- http://www.ibm.com/developerworks/kr/library/l-python3-1/
- http://www.ibm.com/developerworks/kr/library/l-python3-2/

마 지막으로, 파이썬 3가 파이썬 2.5보다 속도가 10% 정도 떨어졌다고 합니다..... 가장 큰 원인이 작은 정수형(int) 이 사라지면서 발생한 문제라고 합니다. 나중에 많이 개선될 것이라고 생각합니다. ^^

99. 구 위키독스 방명록

파이썬 입문용으로 잘 보고 갑니다. 이제 다른 문서들 구경하러 가야 겠네요

-doolyes

좋은 문서 잘 봤습니다. TDD방법론... 정말 좋은 경험이었습니다. By. Mr. Lonley.

잘 보게 되네요. 이런 페이지 조차 Python으로 만들 수 있다는게 신기합니다. 하나씩 차근차근 공부해야 겠네요.

감사합니다.^^ - ItsMe

감사감사 ^*^ ㅎㅎ

3.0버전에 변화된부분에대한 설명과 예제를 좀 알려주시면 좋겟어요....

Pythone 강좌를 너무 쉽고 정확하게 설명해 주셔서, 너무 큰 도움이 됩니다.. ^^)
감사합니다.

흔적.

python과 님 덕분에 프로그래밍에 재미를 느낍니다.^^;
고맙습니다.

파이썬에 대해 속성으로 아주 재미있게 공부할 수 있게 되어서 정말 정말 기쁩니다. 진심으로 감사합니다. (예전에 책으로도 보았습니다만 책보다 위키독스가 더 활용도가 높네요 ^^;) 추가적인 부분으로 XML관련 부분과 C 프로그래머들이 생각하는 포인터 개념에서 파이썬을 활용할 수 있는 부분을 따로 정리해주시면 금방 이해하고 활용하는데 도움이 되지 않을까 합니다. 예를 들어, 변수 scope, call by reference, call by value 에 대한 파이썬에서의 차이점, 파이썬에서 제공하는 id() => 객체 ID 제공함수 , sys.getrefcount() 등을 활용한 내부 동작 설명 등이 좀 더 깊은 파이썬 프로그래밍을 하는데 많은 도움 될 듯 합니다. ^^ 앞으로도 더 많은 이야기들이 추가되고 발전될 것을 바라며 감사드립니다. -2009.04.01 (-_-; 핫..만우절이네요..이런..) 오진원.

파이썬에 대한 기본을 잡는데 정말 큰 도움이 되었습니다. 깊이 감사드립니다. 2009.04.12 by Robinson

3.0에 대한 강좌좀 부탁드려요

파이썬을 배우기에 너무 좋네요. 감사합니다. by Lak

파이썬의 매력을 가르쳐 주셔서 감사합니다. by simon

C언어에 꼴아박다가 포기하고 있던 참에 파이썬을 알게 되었습니다. 감사합니다. 그리고 3.0에 대한 정보도 가르쳐 주면 감사하겠네요. ㅎㅎ

덕분에 (프로그램이라고 부르기도 힘든 것들이지만) 간단한 것들을 만들 수 있게 되었습니다. 감사합니다. 요즘 공책에 하는 낙서는 파이썬 스크립트 :) 10.01.15 by 이규민