초짜를 위한 파이썬

by Pupp3tM, a.k.a. David Borowitz
2001. 11.11 한글판 johnsonj

목차

소 개

파이썬이란 무엇인가**?**

파이썬은, 간단히 말해서, 스크립트 언어입니다. 기능상으로는 펄과 비슷하지만, 내가 아는**?**그렇게 알려져 있지 않습니다. 이 점을 해결하기 위한 것이 이 지침서의 목적입니다. 대단히 고-수준의 언어로서 복잡한 작업을 수행할

수 있을 뿐만 아니라, 놀랍도록 배우기 쉽습니다. 많은 다른 부가 모듈(addons) 도 사용가능하여 MP3로부터 (유닉스의) 윈도우 도구까지 모든 것을 제어할 수 있습니다. 내가 파이썬에서 발견한 것은 프로그램하기에 정말 순수하게 *재미 있*는 언어로서, 대단히 직관적이고 다양한 용도에 적합하다는 것입니다.

파이썬은 유닉스로부터 윈도우까지, 거의 모든 플랫폼에서 실행될 수 있습니다. 파이썬을 얻으려면, 우선 Python.org에 가셔서, 내려 받으세요!

이 지침서는 프로그래밍 경험은 없으나 적어도 어느정도는 아는 사람들을 대상으로 합니다. 나는 C 와 Perl같은 언어에 대한 수많은 참조서를 작성하였습니다, 이해하면 좋겠지만, 그냥 넘어가도 크게 잃을 것은 없습니다.

그 웃긴 이름은 도데체 무엇인가, 이렇게 묻겠지요? 일종의 육식 파충류인가? 아닙니다, 독자 여러분. 파이썬의 창조자는 열성적인 몬티 파이썬(Monty Python) 팬이었고 현재도 그렇습니다. 그 이름은 BBC 쇼인 "Monty Python's Flying Circus,"에서 왔으며 공식적인 문서에서는, "문서에 몬티 파이썬 풍자극을 언급하는 것은 허용될 뿐만 아니라, 격려된다"라고 말하고 있습니다. 그럼에도 불구하고, 몬티 파이썬 영화의 많은 부분을 보지 못한 것이 유감입니다. 그래서 약간 인용하여 볼 것입니다 :)

자 물러 앉아서, 콜라 한잔을 들고, 파이썬을 내려 받으세요, 그리고 배울 준비를 하세요 :)

최상위로

인터프리터에 관하여

파이썬은 스크립트, 즉, 인터프리터 언어입니다. 컴파일러를 가지는 C와는 다르게, 우리는 파이썬에서...(드럼 두두두 짠)...파이썬 인터프리터를 가집니다. 파이썬을 정확하게, 유닉스 쉘 혹은 도스 박스에 설치하였으면, 'python'이라고 타자하세요 (주의: 내가 타자하라고 하면서 인용부호에 집어 넣을 때 그 인용부호는 타자하지 마세요. 이미 그렇게 이해하셨으리라 확신하지만, 그래도...). 만약 펄 혹은 다른 어떤 인터프리터 언어를 이전에 사용해 보셨다면, 무언가를 깨달았을 것입니다: 프롬프트가 있습니다 (세개의 귀여운 ">>>" 문자들)! 표준입력(stdin)을 기다리지도 않으며 "Usage: python <script-file>"이라고 말하지도 않습니다! 바로 그것입니다, 파이썬은 *상호대화* 모드를 가지고 있습니다. 이 모드에서, 어떠한 파이썬 명령어도 타자할 수 있으며 그것은 약간은 다르긴 하지만, 스크립트로부터 타이프한 것과 똑 같이 작동합니다. 대단히 중요한 것은, 만약 (할당을 제외하고) 일종의 값을 반환하는 일종의 변수 혹은 어떤 것을 타자한다면, 자동적으로 그 결과를 출력할 것입니다. 무언가를 시도해 보기에 (그리고 배우기에) 깔끔합니다! 이제 프롬프트에서 '1 + 1'을 타이프하고 가만히 계세요. 무엇이 보입니까? 2! 아직은 그렇게 강력한 언어는 아닙니다:

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

모든 종류의 변수나 혹은 수학 함수를 가지고 이렇게 할 수 있으며 (파트3b를 참조) 어떤 값을 얻을 수 있습니다. 문자열과의 작업도 마찬가지입니다:

```
>>> "elp! I'm being oppressed"
"elp! I'm being oppressed"
```

이 가짜 몬티 파이썬 인용록을 주목하세요 :). 어쨌든, 이것이 그 인터프리터에 대한 모든 것입니다. 이제 우리는 사물의 핵심으로 들어갑니다. 만약 어느 곳에서나, 간단한 스크립트가 있고 새로이 파일을 만들고 싶지 않다면, 편안하게 그것을 그 인터프리터 안으로 타자해 넣으세요 (혹은 더 좋은 것은 복사해서 붙이는 것이겠지요), 똑 같이 작동할 것입니다. 잠까만요, 이점을 살펴봅시다 :). 만약 내가 어떤 것을 스트립트안으로 넣는다면, 그것은 거의 스

크립트로만 효과를 얻을 수 있다는 뜻이며 상호대화 모드로는 작동하지 않는다는 것을 뜻합니다. 유감입니다만.

잠깐! 떠나기전에 주의사항 한가지. 인터프리터를 끝내려면, 'quit'을 타자하는 것과 같은 어떤 것을 할 수는 없습니다. 어떤 이유로 (나도 모릅니다), 그것은 작동하지 않습니다. Ctrl+D를 눌러서 종료하여야 합니다 (적어도 유닉스에서는 그렇습니다만, 윈도우에서는 테스트해보지 않았습니다). 어쨌든 'quit'이라고 타자하면, 다음과 같이 'Ctrl-D를 사용하여 종료하세요(Use Ctrl-D (i.e. EOF) to exit).'와 같은 말을 하여 줄 것입니다. 그러면 그것을 따르세요.

최상위로

첫 번째 프로그램

Hello, World!

좋습니다, 만약 프로그래밍 지침서를 이전에 읽어본 적이 있다면 화면에 "Hello, World!"와 같은 종류의 어떤 것을 출력하는 프로그램을 모든 언어들이 가지고 있다는 것을 보았을 겁니다. 불행하게도, 이 지침서도 다르지 않습니다. 좋습니다, 자 이제 시작해 봅시다:

```
#!/usr/bin/python
#위의 라인은 유닉스에서만 필요합니다; 위 라인을 여러분의 파이썬 경로로 바꾸세요

hello = "Hello, World!"
print hello
```

바로 이것입니다! 이것을 텍스트 파일에 넣고서, hello.py로 이름짓고, 실행해보시면, 우리는 행복감에 빠집니다. 유닉스에서 'chmod +x hello.py'를 수행하여 그것을 실행할 필요가 있습니다, 그렇지 않으면 단지 'python hello.py'라고 타자하세요. 윈도우에서, 내가 확신하기로는 .py 파일은 자동적으로 파이썬 인터프리터와 연결됩니다.

최상위로

설명

"와," 아마 여러분은, "쉽군."이라고 말하겠지요. 실제로 그렇습니다. 그렇지만, 무슨일이 일어날지 이제 겨우 멀고 먼 실마리의 한 쪽을 잡은 것일 뿐입니다. 라인 단위로 그것을 이해해 보겠습니다.

```
#!/usr/bin/python
```

이 라인은, 읽어 보신데로, 유닉스 사용자에게만 필요합니다. 이것은 여러분의 쉘에게 파이썬 실행파일이 어디에 있는지 말하여 주는데 그리하여 그 스크립트를 실행할 수 있습니다.

```
#위 라인은 유닉스에서만 필요합니다; 여러분의 파이썬 경로로 바꾸세요
```

정말 쓸모없는 라인이야, 라고 생각하실지 모르지만, 이것은 중요한 점을 예시하여 주고 있습니다: 주석! 모든 훌

륭한 프로그래머는 여러분의 코드에 주석이 필요하다고 생각합니다, 그래서 #을 입력하면 그 라인 이후로 모든 것은 주석이고 인터프리터는 무시할 것입니다.

```
hello = "Hello, World!"
```

이 라인이 진짜로 필요한 것은 아닙니다, 그러나 오직 하나의 라인만 가지는 Hello, World! 스크립트는 약간 어색해 보이므로, 어쨌든간에 이 라인을 추가하였습니다. hello는 문자열 변수입니다, 나중에 그것에 관하여 논의할 것입니다, 그리고 = 연산자는 "Hello, World!"를 그의 값으로 할당합니다. 이전에 프로그램해 본 경험이 있다면 이것을 분명히 보았으리라 확신합니다.

```
print hello
```

이것은 마지막 라인입니다, 예상하신대로, hello의 내용을 출력합니다. 만약 perl을 하시는 소년 (혹은 소녀 :)라면, 하나의 변수를 출력할 때 그 변수의 이름에 인용부호를 붙일 수가 없다는 것을 이해하실 필요가 있습니다. 그러나 나중에 보게 되듯이, 그것은 너무나 편리합니다.

좋습니다! 우리의 첫 번째 프로그램입니다! 이제 더 진행하기 전에 한 두개의 주의사항을 주겠습니다. 먼저, 모든 라인의 마지막에는 아무것도 없다는 것을 아실 수 있습니다, 펄이나 C 에서와 같은 ; 이 전혀 없읍니다. 그것은 파이썬의 편리한 사양중의 하나입니다. 나는 그것이 더욱 직관적이라고 생각하는데 왜냐하면 블록은 들여쓰기로 결정되고, 모든 서술문은 자신만의 라인을 가지기 때문입니다...혹시 그것에 대해 동의하지 않을지는 모르겠으나 반드시 익숙해질 필요가 있습니다.

최상위로

변수와 수학

변수: 문자열, 숫자, 등등.

파이썬에서는, 맛볼만한 다른 어떤 언어와 마찬가지로, 수 많은 데이타 형이 있습니다, 주로, (임의의 정밀도를 가지는) 숫자, 문자열, 리스트, 그리고 터플이 있습니다. 터플은 리스트와 비슷한, 더욱 "진보된" 변수 형입니다, 그래서 더 이상 깊이 들어가지 않겠습니다만, 간접적인 파이썬(Intermediate Python)섹션에서 약간 다루어 볼 것입니다.

숫자. 뭐라고 할 수 있을까요? 숫자는 숫자입니다. 거기에 할당하려면, Hello World!에서 문자열에 했던 것과 똑같이 하는데, 단지 변수의 이름, = 연산자, 그리고 원하시는 것의 순서로 배정하세요:

```
>>> a = 5
>>>
```

변수 이름에 관하여 한 말씀: 파이썬이 정확하게 얼마나 긴 변수를 허용하는지는 확신할 수 없습니다(내가 알아야 하나요?), 그러나 일반적으로 20자 언저리 이상의 문자를 가져서는 안됩니다. 다른 언어 처럼, 변수 이름은 모두 알파벳 문자와 밑줄문자 (_)여야 합니다, 그러나 어떤 언어와는 다르게 밑줄문자로 *시작할* 수 있습니다.

어쨌든, 문자열에 관해서는, 문자열은 분명히, 문자들로 이루어진 문자열을 저장합니다. 이전에 들어보지 못한 것은 아무것도 없습니다. 문자열에 대해서, 여러분은 인용부호의 형태를 사용할 수 있습니다 ("" 또는 ''), 그러나 어떤 종류인지는 확인하세요. 만약, 예를 들어 어포스트로피를 문자열안쪽에서 사용하고자 한다면, 무엇인가를 확인해야 할 필요가 있습니다. 그 문자열이 "" 인용부호를 가지면, 그것에 관해 걱정할 필요는 없습니다, 그저 어포스트

로피를 붙이세요. 그렇지만, 만약 '' 인용부호 안에 있다면, 그 어포스트로피를 탈출시킬 필요가 있을 것입니다. 이것은 인터프리터가 어포스트로피를, 단지 인용부호인 것처럼, 문자열의 마지막으로 생각하지 않기 때문입니다. 어포스트로피를 탈출시키려면, 단지 \ (역사선)을 그 인용부호 앞에 놓으세요. 한번 해보세요! 훌륭합니다:

```
>>> "I'm Dave"
"I'm Dave"
>>> 'Python\'s my favorite language'
"Python's my favorite language"
>>> "\"I can't think of anything else to say!\" he said"
'"I can\'t think of anything else to say!" he said'
>>> 'That's too bad'
      File "<stdin>", line 1
    'That's too bad'
          ^
SyntaxError: invalid syntax
```

마지막 두 개의 예제를 주목하세요. 첫 번째 예제는 "" 인용부호의 탈출 예를 보여 주고 있으며, 또한 다음과 같은 예를 보여 주고 있습니다: 두 가지 종류 모두의 인용부호를 문자열에 가지고 있으면, 파이썬은 하나를 기본값으로 하고 다른 하나를 탈출시킬 것입니다. 걱정하지 마세요; 만약 이것을 스크린에 출력한다면, 출력에 \ 문자는 보여주지 않을 것입니다. 마지막 예제에서, 인용부호를 탈출시키지 않았을 때 일어나는 일이 보입니다. 파이썬은 멋지게 에러를 처리하는 방법을 가진다는 것을 눈치채셨을 겁니다, 그러나 거기에 더 이상 깊이 들어가지는 않겠습니다.

탈출에 관하여 말하자면, 여기에 탈출시킬 수 있는 문자들이 더 있습니다 (만약 'n'이라면 '\n'을 사용하여 그것을 탈출시키세요):

- n - newline을 출력한다, 즉. 새로운 라인을 시작한다.
- t - 수평 탭
- b - 역사선. 마지막으로 타자된 문자를 삭제한다
- a - 시스템 비프음
- \ - 역사선을 출력한다
- r - 캐리지 리턴을 출력한다. 여러분이 무엇을 하고 있는지 이해하지 못한다면 newline을 사용하세요.

물론 더 많은 것이 있겠지만, 여러분은 그것을 그렇게 자주 사용하지는 않을 것입니다.

문자열에 대해서는, 그것들을 + 연산자로 연결할 수 도 (함께 묶을 수도) 있습니다. 만약 두 개의 기호 문자열을 가지고 있다면 (즉. 문자열을 반환하는 함수가 아니라면), 심지어는 연산자조차 필요없습니다:

```
>>> 'holy' + 'grail'
'holygrail'
>>> 'BS' 'RF'
BSRF
```

다음은 리스트입니다. 리스트는 어떤 언어에서는 배열이라 불리지만, 거의 같은 것입니다 (음, 완전하게는 아니지만...). 리스트는 한 다발의 값들로 각괄호 [] 안에 함께 그룹지어집니다. 리스트는 같은 리스트안에 임의의 개수의 변수들과, 심지어는 문자열과 숫자까지도 담을 수 있습니다, (**역주 : 그러나 터플 혹은 내포된 리스트는 담을 수 없습니다???). 리스트를 할당하는 것은 간단합니다:

```
>>> a = [1, "thin", "mint", 0]
>>> a
[1, 'thin', 'mint', 0]
```

리스트는 이름으로 간단하게 접근될 수 있습니다, 그러나 그 이름 바로 뒤에 각괄호를 두고 그 안에 지표 숫자를 배정함으로써, 리스트에 있는 변수들에 지표로 접근할 수 있습니다. 지표 숫자는 0 에서 시작한다는 것을 주목하세요:

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

또한 음의 지표 숫자를 사용해서 그 리스트의 뒤에서부터 시작하여 셀수도 있습니다:

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

지금으로써는 이것이 지표에 관한 모든 것입니다, 그러나 간접적인 문자열 섹션에서 우리는 더 재미있는 것들을 배우게 될텐데, 조각썰기 지표로 리스트에서 하나이상의 요소에 접근하는 법을 배웁니다. 또한 지표-호출에 대한 지식을 문자열에 자신있게 사용할 수 있습니다. 문자열에 있는 문자들은 리스트에 있는 요소들처럼 행동합니다:

```
>>> a = "BSRF"
>>> a[1]
S
```

그렇지만 주의 하세요: 리스트에서 개별적인 요소를 대체할 수 있지만, 문자열에서는 그렇지 않습니다:

```
>>> a = [1, "thin", "mint", 0]
>>> b = "BSRF"
>>> a[3] = 4
>>> a
[1, 'thin', 'mint', 4]
>>> b[1] = "M"
Traceback (innermost last):
  File "<stdin>Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment", line 1, in ?
```

이게 모두입니다! 적어도 데이타 형에 관한한...

최상위로

수학과 연산자

파이썬에는 기본적인 수학 연산자들이 있습니다: +, -, *, /, **, 그리고 %. 학교에 다녀 보셨다면 +, -, *, 그리고 / 등등 이런것이 무엇을 뜻하는지를 아시겠지요. **와 %는 빠졌습니다. ** 는 지수승 연산자입니다:

```
>>> 2 ** 6
64
```

%는 표준적인 나머지 연산자입니다. 나머지? 네. 잘 모르시겠다면, % 연산자는 두 번째 숫자로 첫 번째 숫자를

나누어서 그 나머지를 돌려 줍니다. 살펴 보세요:

```
>>> 4 % 2
0
>>> 10 % 3
1
>>> 21 % 6
3
```

이러한 것들을 시도해 보는 중에 느끼셨을 테지만 만약, 예를 들어, 1을 3으로 나눈다면, 0을 얻는다는 것을 깨닫으셨을 겁니다. 이것은 파이썬이 정수와 작업할 때 항상 수치를 내림하기 때문입니다. 부동소수점 수와 작업하려면, 단지 . (소수점)을 표현식의 어딘가에 넣기만 하세요:

```
>>> #나쁩니다
... 1/3
0
>>> #좋습니다
... 1./3
0.333333333333
>>> #역시 좋습니다
... 1/3.
0.333333333333
```

기본적인 수학 연산자 이외에도, 파이썬에는 비교 연산자도 있습니다 (무엇을 예상하고 계신가요?). 더 많은 섹션에서 우리는 문맥에 맞게 그것들을 사용하는 법을 배울 것입니다, 그러나 어쨌든 여기에 그것들을 보입니다:

- a == b a가 b와 동등한지를 점검한다 (a와 b 단지 수치뿐만 아니라, 어떤 형도 될 수 있습니다)
- a > b a가 b를 초과하는지 점검한다 (위의 괄호를 보세요)
- a < b a가 b보다 미만인지 점검한다 (등등.)
- a >= b a가 b이상인지 점검한다
- a <= b a가 b이하인지 점검한다
- a != b a가 b와 동등하지 않은지 점검한다

아마도 여러분은 문자열과 리스트 같은 것들이 비교될 수 있다는 점만 빼고는 이것으로부터 그렇게 유용한 것을 얻지는 못할 겁니다. 사실, 리스트 (또는 문자열)은 심지어 비교되지 위해서 같은 크기일 필요조차 없습니다:

```
>>> #반환값 1은 참을 의미하고, 0 은 거짓을 의미합니다
... [1, 2, 3, 4] > [1, 2, 3]
1
>>> [1, 2, 4] > [1, 2, 3]
1
>>> [1, 2, 3] < [1, 3, 2]
1
>>> [1, 2, -1] > [1, 2]
1
```

어떻게 리스트 비교가 수행되는 지를 주목하세요. 첫 번째 값이 비교가 됩니다. 만약 동등하지 않다면, 다음 값이 비교됩니다. 만약 두 개 모두 동등하다면, 다음 값이 비교가 됩니다. 이런식으로 하나의 리스트에 있는 값이 다른 리스트 안에 있는 값과 같지 않을 때까지 계속됩니다. (만약 모든 것이 같다면 그 리스트는 동등합니다). 세 번째 예제를 보세요: 첫 번째 리스트의 3번째 값(지표2인, 구성원)(3)이 두 번째 리스트의 3번째 구성원(2)보다 큼에

도 불구하고, 두 번째 리스트는 첫 번째 보다 여전히 큰데 이유는 인터프리터가 그렇게 멀리까지 가지 않기 때문입니다; 두 번째 리스트에 있는 두 번째 숫자(3)가 첫 번째 리스트의 두 번째 숫자(2)보다 크다는 것을 알고, 멈추기 때문입니다 (혼동된다고요? 좋습니다). 만약 인터프리터가 null 값을 가지면 (즉. 리스트에 항목이 남아 있지 않으면), 더 긴 리스트가 더 큽니다 (위의 4번째 예제를 보세요).

마지막으로 연산자에 관한 것입니다: 논리 연산자. and, or, not (펄과는 다르게, xor은 없습니다). and 는 만약 참이라면 마지막으로 만나는 값을 반환하고, 거짓이라면, 0을 반환합니다. or 도 마찬가지입니다, 그러나 그 인수들 중 오직 하나라도 non-zero (또는 문자열에 대해서는 non-null)라면 참입니다. 마지막으로, not은 한개의 인수가 0(null)이라면 1을 반환하고 그 인수가 0이 아니라면 0을 반환합니. 다음을 살펴보세요:

```
>>> 1 and 4
4
>>> 2 or 0
2
>>> 'monty' and 'python'
'python'
>>> not 0
1
>>> not ''
1
```

최상위로

입력/출력

정보를 출력하기

우리의 Hello, World! 예제에서, 이미 출력을 인쇄하는 방법 하나를, 즉 (말-그대로의) print 서술문을 보았습니다. print는 펄에서 그런것과 마찬가지로 많은 일을 합니다, 화면에 문자열 또는 모든 종류의 변수 혹은 서술문을 출력해 냅니다. print로 입력하는 모든 것은 자동적으로 newline, "\n",이 추가됩니다.

```
>>> print "She's a witch!"
She's a witch!
>>> print 5 + 3
8
>>> a = "Burn her!"
>>> print a
Burn her!
>>> #이것이 무엇을 하는 것인지 모른다고 해서 걱정하지 마세요; 이것은 간적접인 파이썬 섹션에서 다룹니다
... print "Yeah, b" + a[1:]
Yeah, burn her!
```

새로운 라인이 추가되지 않기를 원한다면, 그런 경우가 자주 있으실 텐데, 간단하게 ','를 print 서술문이 있는 그 라인의 마지막에 추가하세요. 주목할 것은 이것은 오직 비-상호대화 모드에서만 작동합니다, 그래서 여러분은 새로운 file.py 같은 것을 만들 필요가 있습니다. 시험삼아 이것을 시도해 보세요:

```
#!/usr/bin/python
```

```
a = 5
print "The value of a is",
print a,
print "!"
```

출력:

```
The value of a is 5 !
```

print 서술문에 (비-문자열) 변수들을 삽입하는 방법은 여러가지가 더 있습니다. 컴마를 사용하실 수 있는데, 그 변수 주위에 공백하나를 배정하여 줄 것입니다. 마지막으로 가장 중요한 방법은 변수를 문자열로 먼저 변환하는 것인데, 역방향 인용부호(``) 안에 그것을 놓음으로써 간단하게 수행됩니다. 그리고나서 이것을 문자열에, 마치 다른 문자열 변수 혹은 값에 대하여 하는 것처럼, 추가하기만 하세요 (위를 참조). 이 예제들을 점검하여 보세요:

```
>>> a=5
>>> #숫자 좌우의 공백에 주목하세요
... print "A duck costs $", a, "."
A duck costs $ 5 .
>>> #같은 일을 하지만, 다른 방식으로 입력합니다
... print "A duck costs $" + `a` + "."
A duck costs $5.
```

stderr (또는 말이 난 김에 stdout과 stdin)가 무엇인지 아는 분이라면, 네 그렇습니다, 파이썬에서는 직접 그것에 접근할 수 있습니다. 그렇지만 이것은 약간 더 진보된 테크닉을 사용합니다, 그래서 여기에 무슨일이 일어나는지 실제로 이해하려면 파일 입출력(File I/O) 그리고 모듈(Modules)를 읽어 보셔야 합니다. 어쨌든, 이 스크립트를 시험해 보세요:

```
#!/usr/bin/python

import sys #모듈에 대하여 읽어 보시라고 말하지 않았나요?
sys.stderr.write( "This is some error text\n" ) #여기에서 \n이 필요합니다
#왜냐하면 직접적으로 파일 기술자에 써넣고 있기 때문입니다 (파일 입/출력을 읽어 보세요)
```

실행하세요, 다소간 정상적으로 보이는데, 그저 다음과 같이 "This is some error text"를 출력합니다. 이제 그것을 다음과 같이 "python test.py > /dev/null" (유닉스) 또는 "test.py > nul" (도스/윈) 시도해 보세요. 이게 무엇인가요? 그것을 /dev/null 또는 null, 즉 출력을 버리는 쓰레기통에 방향전환하였음에도 불구하고 출력은 여전히 되는군요. print가 하는 일인, 표준출력(stdout)에 쓰는 대신에, 여기에서 여러분은 표준에러(stderr)에 썼습니다, 표준에러는 (비록 같은 화면에 나타나기는 하지만) stdout과 별개이며 프로그램이 마주치게 될 모든 에러를 여러분에게 알려줍니다.

최상위로

사용자와 상호대화하기

좋습니다, 인정하지요, 이 제목은 약간 광범위한 감이 있습니다. 이 섹션에서 나는 단지 사용자로부터 변수를 입력받는 방법만을 다룰 것이며, 키눌림 나포라든가 그러한 것은 다루지 않을 것입니다. 어쨌든, 파이썬에서 표준입력은 두 개의 간단한 함수로 완성됩니다: input과 raw_input. 사용하기에 더 쉽기 때문에, 먼저 raw_input을 다루

어 보겠습니다.

raw_input은 한개의 인수를 가집니다, 그 인수는 프롬프트로서 사용자가 무언가를 입력하기를 기다리는 동안에 보여질 것입니다. 그것은 타자된 문자열을 반환합니다, 그래서 할당을 할 때에 극히 유용합니다. 간단한 예제 하나:

```
>>> a = raw_input( "Type something: " )
Type something: Dave
>>> a
'Dave'
```

너무 고민하지 마세요. 나는 'Dave'를 타자해 넣었습니다, 그리고 raw_input은 그것을 변수 a에 할당 했습니다. "거참 훌륭하군 좋아," 이렇게 말하시겠지만, "그러나 문자열을 입력하고 싶은 것이 아니라면? 숫자나 리스틀 원한다면?" 자, 실제로는 약 세 개의 방법이 있습니다. 첫 번째로 가장 간단한 방법은 input 함수로서, 조금 더 내려가 보시면 보시게 될 것입니다. 다른 것은 string.atoi인데, 약간은 더 진보된 방법으로, 입력 처리에 약간의 유연성을 추가하여 줍니다, 그렇지만 그것은 오직 문자열을 정수로 변환할 뿐입니다. (더 자세한 내용은 다음에: Modules). 더 자세하게 들어가지는 않겠습니다, 그러나 여기에 간단한 atoi예제가 있습니다:

```
>>> from string import atoi
>>> a = "5"
>>> a
'5'
>>> a = atoi(a)
>>> a
5
>>> #파이썬 2.0 이상을 가지고 있다면, 그저 이렇게 하기만 하면 됩니다.
... a = "5"
>>> a = int(a)
>>> a
5
```

와우, a는 '5'로부터 5로 변환되었습니다. 이제 계속 가 봅시다. 내가 이전에 말한바와 같은, 모든 종류의 변수들을 입력하고자 한다면 input을 사용하고 싶을 것입니다. Input은 raw_input처럼 작동합니다, 파이썬이 그 변수들을 입력해 넣은 그대로 이해한다는 점만 빼고는 말이지요. 만약 그것이 탐탁지 않다면, 이런식으로 생각해 보세요: 진짜 숫자를 가졌다면, 그것은 숫자로 저장됩니다. 어떤 값들을 [] 각괄호안에 가졌다면, 그것은 리스트로 저장됩니다. 문자열을 가지기 위해서 인용부호 안에 어떤 것을 배정할 수도 있습니다. 확인해 보세요:

```
>>> a = input( "Enter something: " )
Enter something: 5
>>> a
5
>>> a = input( "Enter something: " )
Enter something: [3, 6, 'Hello']
>>> a
[3, 6, 'Hello']
>>> a = input( "Enter something: " )
Enter something: Hello
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<string>", line 0, in ?
NameError: There is no variable named 'Hello'
```

에고고! 무슨일이 일어났습니까? 자, input이 가지는 하나의 함정은 그것이 모든 것을 일종의 변수로 바라본다는 것입니다; input은 Hello가 한 변수이 이름이라고 생각했습니다, 그래서 Hello가 없다는 것을 발견하자 뒤틀려 버렸습니다. 이것을 해결하는 방법은, 예외 나포와 같은, 여러 방법들이 있지만, 그것은 이 지침서의 범위를 벗어납니다. 지금으로써는, 여러분이 원하는 변수의 형을 입력할 것이라고 사용자를 믿어야할 필요가 있을 것입니다 (압니다, 나도 압니다, 여러분은 보통 사용자를 믿지 않으시겠지요. 그러나...). 앞으로, 앞으로 나갑시다.

최상위로

프로그램 제어

만약 ...라면(What If...)

만약 이전에, 베이직에서 플래시까지 그 어떤 것으로, 프로그램 해본 *경험이* 있으시다면, if 블록을 보셨을 겁니다. 이런, 나는 그것을 설명하기 위하여 한 섹션 전체를 할애해야 하는 이유를 모르겠습니다, 왜냐하면 내가 할 모든 것은 여러분에게 그 구조를 보여주는 것 뿐이기 때문입니다, 맞지요? 좋습니다, 이전에 사용해 보신 비교 연산자들이 기억나시나요? 좋습니다. 이 예제 코드를 한 번 살펴 봅시다:

```
>>> name = raw_input( "What's your name? " )
What's your name? David
>>> if name < "Dave":
...     print "Your name comes before mine." #여기에 있는 것은 탭입니다
... elif name > "Dave":
...     print "Your name comes after mine."
... else:
...     print "Hey, we have the same name!"
...
Your name comes after mine.
```

별거 아닌 예제입니다, 나도 압니다, 그러나 훑어보아야 할 요점이 있습니다. 연구해 보세요. 이제 여기에서 여러분은 알고 있던 다른 언어와는 다른 점을 눈치 채셨을 것입니다 (그렇지 않다면, 내가 뭐라고 말해야 하나). 우리 역시, 그 점을 차근 차근히 살펴볼 것입니다. 먼저, if 가 보입니다, 그리고나서 조건 서술문이 보입니다. 조건 서술문은 이전에 우리가 보았던, 하나의 표현식 그리고 하나의 연산자 그리고 또 다른 표현식의 형태와 같습니다. 그냥 단순히 표현식만을 가질 수도 있습니다, 그리고 그 표현식이 non-zero라고 평가되면 if는 그것을 참이라고 판단할 것입니다. 그러나 어쨌든, 다음에는 콜론이 옵니다, 그리고 나서 새로운 라인이 나옵니다. 여러분은 이 newline이 필요합니다. 다음의 라인 혹은 둘 셋 그렇지 않으면 많은 라인에서는, 그 코드는 들여쓰기 되어 있습니다. 한개의 공백, 네개의 공백, 여덟개의 공백, 모두 상관없이, 단지 들여쓰기 되어 있기만 하면 됩니다. 코드가 일반적으로 얼마나 복잡하느냐에 따라서, 이것은 편리할 수 있는데, 코드란 아름다운 모습이거나 아니면 지긋지긋한 코드 블록이 됩니다. 개인적으로 나는 그것을 좋아합니다. 다음 라인은 비밀 스럽게-보이는 서술문 elif입니다. 보자마자 들여쓰기 되어 있지 않다는 것을 알 수 있습니다; 그것은 같은 코드 블록에 있지 않습니다. elif는 life를 거꾸로 한 것이 아니라, else if를 줄여 놓은 것입니다. 만약 첫 번째 조건이 참이 아니라면, 다음 조건을 점검합니다. 다음 블록으로, 갑니다. 마지막으로 else가 옵니다: 이 후방지원 서술문은, 위의 조건들 모두가 만족되지 못하면, (원하시는 만큼 놓을 수 있으며, 마치 C-스타일의 switch 서술문을 대체하는 기능을 제공합니다) 만나게 됩니다.

한마디로 말해서, 말한 그대로입니다.

최상위로

영원히...(For Ever...)

와우, 가장 볼품없는 섹션 제목을 단것은 아닌가요? 어쨌든, 다음의 표준 회돌이로 나아갑니다: for 회돌이. 파이썬의 for 회돌이는, 펄이나 PHP의 foreach 회돌이를 보시지 않았다면, 아마도 여러분이 보신 것과는 약간 다를 것입니다. 파이썬에서 for 회돌이는 문자열이나 리스트의 각 항목을 순회하면서 그것을 특정한, 지정된 변수에 할당합니다. 구문은 직관적이며 심지어는 인간의 언어와 유사하기까지 합니다. 살펴보고 배워보세요:

```
>>> for letter in "Brian": #블록은 위의 방식과 똑 같이 취급됩니다
...     print letter
...     #이것은 단지 "Brian"의 각 문자를 각각의 라인에 출력합니다
...
B
r
i
a
n
```

대단히 간단합니다. 그러나 이 괴이한 for 회돌이가 부족하다고 말씀하실지도 모르겠는데 왜냐하면 그것은 예를 들어, 1에서 10까지의 숫자를 모두 순회하지 않고서 그것들을 i 에, *다른* 언어들처럼, 집어 넣고 있기 때문입니다. 그것은 언제나 어디서나 유용한 range() 함수로 쉽게 극복됩니다. range()는, for와 함께 수행되면 완벽하게, 지정된 범위의 숫자를 담은 리스트를 반환할 것입니다 (자, 파이썬 창조자들은 그러한 이름들을 어디에 사용하는 것일까요?). 여러분은 인수중의 한 숫자를 다음과 같이 지정할 수 있습니다:

```
>>> #이것은 0에서 시작하여, 여섯개의 숫자를 담은 리스트를 반환합니다.
... range(6) #이것은 여러분이 넣은 그 인수가 실제로는 그 리스트에 존재하지 않는 다는 것을 뜻합니다.
[0, 1, 2, 3, 4, 5]
>>> #6에서 10-1까지, 또는 6-9 까지의 리스트.
... range(6,10) #10은, 위와 같이, 실제로는 나타나지 않는다는 것을 기억하세요.
[6, 7, 8, 9]
>>> range(6,10,2) #위와 동일, 단지 2 만큼씩 증가.
[6, 8]
>>> range(10,6,-1) #음의 방향으로 증가도 역시 작동합니다. 마지막 숫자, 6은, 여전히 출현하지 않습니다.
[10, 9, 8, 7]
```

이러한 것들은 있는 그대로 대단히 자기-설명적입니다. 또 다른 작은 꼼수는, 만약 지표 숫자로, 리스트에 있는 요소들을 순회하기를 원한다면, len() 함수를 (이것은 그 리스트의 길이를 반환합니다) range() 함수와 함께 사용하는 것입니다:

```
>>> l = ["I'm", "not", "dead"]
>>> for i in range(len(l)):
...     print "Item", i, "of l is "+l[i]
...
Item 0 of l is I'm
Item 1 of l is not
Item 2 of l is dead
```

자, 이것은 for 회돌이에 싸 넣고 있습니다. 기억하세요: 만약 여러분이 실제로 foreach의 구현을 해결할 수 없다면, range()는 여러분의 친구입니다.

최상위로

While We...and Others

나는 이 제목에 관하여 아무것도 *언급하지* 않을 것입니다. 그러면, 언제나-대단히-흔한 마지막 회돌이로 나아가겠습니다: while.
while은 작은 코드 블록을 영원히 수행합니다, 그렇지 않으면 적어도 이미 정의된 어떤 조건을 만족할 때까지 수행합니다.

```
>>> while name != "Dave"
...     name = raw_input( "What's my name? " )
...
What's my name? David
What's my name? Monty
What's my name? Dave
>>>
```

찾는 값을 내가 주기전까지는 집요하게 내가 나가는 것을 거부하고 있는 것을 주목하세요. 물론, while 회돌이는 사용자 입력에 독립적일 필요는 없습니다. 여기에서 이 회돌이는 1000 이하의 모든 입방체를 계산할 것입니다:

```
>>> c = 0
>>> while c**3 <= 1000:
...     print `c`+"^3="+`c**3`
...     c = c + 1
...
0^3=0
1^3=1
2^3=8
3^3=27
4^3=64
5^3=125
6^3=216
7^3=343
8^3=512
9^3=729
10^3=1000
```

회돌이의 또 다른 몇몇 사양은 이른바 다음과 같습니다: break, continue, 그리고 else.
break는 완전하게 회돌이를 빠져 나와서, 모든 서술문들을 건너 뛸 것입니다 (else 절로 가는데, 나중에 살펴 볼것입니다). continue는 회돌이 안의 자신이 있는 곳에서 멈추고 다음 반복을 계속합니다. else 절은, 어떤 회돌이에도 사용가능한데 (단 if에만 안되지요), 그 회돌이가 종료한 후에, 즉. for 회돌이 안에서 더 이상 반복할 항목이 남아 있지 않을 때 실행될 코드를 지정합니다. 이 약간-변경된-그러나-볼품있는 예제를 보세요:

```
>>> for i in range(3):
...     name = raw_input( "What's my name? " )
...     if name != "Dave":
```

```
...             print "Wrong!"
...             continue
...     else:
...             print "Damn straight!"
...             break
... else:
...             print "Too many guesses! You're wrong!"
...
What's my name? David
Wrong!
What's my name? David
Wrong!
What's my name? David
Wrong!
Too many guesses! You're wrong!
>>> #다시 시도해 보세요, 그러나 나는 그 코드를 여기에 다시 타자하지는 않겠습니다.
What's my name? Dave
Damn straight!
```

자, 제어 구조에 대해서는 이 정도로 하지요...

최상위로

간접적인 파이썬

터플!

자, 이제 우리는 대단히-고무적인 터플 섹션에 있습니다. 터플은 다른 종류의 변수로서 리스트와 닮았지만 더욱 일반적이며 보다 더 광범위한 사양을 가집니다. 터플은 보통 활괄호()에 넣어집니다,그러나 때로는 심지어 할당하기 위해서 활괄호조차 필요하지 않습니다:

```
>>> t = (1, 2, 3) #와우, 우리의 첫 번째 터플입니다 :)
>>> t
(1, 2, 3)
>>> t + 1, 2, 3 #누가 괄호가 필요한가요? 음 대부분의 경우에 여러분은 다음과 같이 합니다...
>>> t
(1, 2, 3)
>>> t[0] #여전히 각괄호를 사용하여 지표를 얻습니다
1
>>> t = 1, #여러분은 후미에 컴마가 필요합니다, 그렇지 않은면 t는 단지 정수일 뿐입니다
(1)
```

훌륭하군, 이렇게 말씀하실지 모르겠으나, 이러한 "터플"이 구형 리스트보다 더 특별한 의미는 무엇인가? 자, 하나는, 터플은 다른 터플안에 내포될 수 있습니다:

```
>>> tu = tuple(range(1,4)) #tuple 함수의 사용법을 주목하세요
>>> ple = tuple(range(4,7))
```

```
>>> tuples = tu, ple
>>> tuples
((1, 2, 3), (4, 5, 6))
>>> print tuples[1][1] #이런식으로 접근하세요
5
```

이와 같이 터플을 내포시키는 것은 C 에서의 2-차원 배열 (배열의 배열)과 같은 것들을, 귀찮은 포인트 지정 없이, 대체하여 줄 수 있습니다. 단지, 편의를 위하여 tuple() 함수도 주목하세요. 이것은 모든 구형 리스트를 구형 터플로 변환합니다. 그의 반대인, list() 함수도 존재합니다. 또 다른 재미 있는 터플의 사양도 언급할 만합니다: 포장(packing)과 포장풀기(unpacking). 터플을 포장하는 것은 기본적으로 우리가 위에서 본 것들로, 두 개 이상의 변수들을 터플에 컴마로 배정하는 것입니다. 포장풀기도 유용합니다. 이것이 어떻게 작동하는지 살펴 보세요:

```
>>> n = tuple(range(1,6))
>>> a1, a2, a3, a4, a5 = n
>>> print a1, a2, a3, a4, a5
1 2 3 4 5
```

잘 이해가 안 가실 경우에는, 두 번째 라인이 하는 것을 보시면 터플 n의 각 요소를 취해서 그것을 a1에서 a5까지의 서로 다른 변수들에 할당하는 것을 볼 수 있습니다. 이런 포장 풀기는, 여러분이 이해 하고 있듯이, 리스트와도 작동합니다. 변수들의 리스트를 리스트 각괄호[] 안에 놓으세요. 어떻게 이 두 개의 예제가 같은 일을 하는 지 주목하세요:

```
>>> l = range(1,4)
>>> [l1, l2, l3] = l
>>> t1, t2, t3 = tuple(l)
>>> print l1, l2, l3, "\n", t1, t2, t3
1 2 3
1 2 3
```

일단의 터플을 포장풀기하고자 할 때, 리스트에 있는 변수의 개수가 반드시 그 터플의 길이와 일치해야 한다는 것을 기억하세요. 포장하기와 포장풀기는 하나 더 재미 있는 기회를 제공합니다, 즉 다중 할당이라고 합니다. 이러한 것들이 터플과는 전혀 상관없는 듯이 보일지도 모릅니다, 그러나 지금은 보세요, 나중에 설명하겠습니다:

```
>>> a, b = 0, 1
>>> a
0
>>> b
1
```

보세요, a와 b는 같은 라인에서 다른 값을 할당받습니다! 아주 유용하고, 대단히 간결합니다. 이제, 이곳은 *간접적인(Intermediate)* 섹션이므로, 순서대로 설명하겠습니다.(그렇지만 약간 간단합니다). 먼저 오른 쪽을 보세요: 값 0과 1은 "임시" 터플에 포장되어집니다. 그것을 임으로 t라고 부르겠지만 실제로는 이름을 가지는지 알지 못합니다. 자 이제 왼쪽을 보세요. a와 b는, 컴마로-분리되었으므로, 포장풀려진 터플을 받아 들일 것입니다. 이 터플은 t입니다. 정상적인 포장풀기와 마찬가지로, t의 첫 번째 요소는 a에 들어가고 두 번째는 b에 들어갑니다. 보세요! 여러분은 다중 할당을 한 것입니다.

최상위로

문자열과 조각썰기 지표화

거슬러 올라가 어디선가 내가 문자열을 a[1:]와 같이 지표화 한 곳을 기억하나요 ? 자, 그것은 조각썰기 지표화의 간단한 예제입니다. 조각썰기 지표화로 여러분은 문자열 (또는 리스트, 터플, 그러나 보통은 문자열에서 더욱 유용합니다)에서 한 개 이상의 요소를 취할 수 있으며, 실제로 그 전 조각을 (그러므로 그 이름을) 취할 수 있습니다. 단지 두 개의 숫자만 배정하세요, 시작지표와 끝지표를, 각괄호 안에 배정하세요. 살펴보세요:

```
>>> s = 'supercalifragalisticexpialidotious'
>>> s[0:5]
'super'
```

잠깐, 그것은 옳지 않습니다! s의 5번 요소는 'c'입니다, 'r'이 아니에요! 음, 나는 전에 거짓말을 했습니다. 두 번째 숫자는 여러분이 원하는 가장 큰 지표보다 큽니다. (range()를 기억하시나요? 바로 그와 같습니다.) 그러한 것들은 다루어 보셔야할 필요가 있을 것이라고, 생각합니다.

이제, 이전에 내가 주었던 그 예제는 어떻습니까? 거기에는 두 번째 지표가 없었습니다, 맞지요? 정확합니다. 물론 : 의 앞과 뒤에는 기본 지표가 있습니다. 콜론 앞은 기본 값이 0이고, 뒤에는 기본 값이 len(s)이거나,또는 그 문자열의 길이입니다:

```
>>> s = 'supercalifragalisticexpialidotious'
>>> s[:5]
'super'
>>> s[27:]
'dotious'
```

여러분은 s의 27번째 까지 계속 세어가서 dotious를 얻어야만 한다는 것이 불편하게 생각될 것입니다. 그저 마지막 일곱 글자만 원한다고 가정해 봅시다. 음의 지표 숫자도 역시 작동한다고 말한 것이 기억나나요? 여기, 조각썰기에도 그것은 잘 작동합니다. 단지 시작 지표가 끝 지표 앞에 오는 것만 확인하시면 됩니다 (내가 말한 바를 이해하시게 될 것입니다):

```
>>> s = 'supercalifragalisticexpialidotious'
>>> s[-7:]
'dotious'
>>> s[int(-.5*len(s)):len(s)-5] #지표는 임의의 정수 표현식이 될 수 있습니다
'ticexpialido'
>>> s[:] #콜론만 있는 것은 없는 것과 동일합니다
'supercalifragalisticexpialidotious'
>>> s[:10]+s[10:] #As a rule of thumb, s[:n]+s[n:]=s
'supercalifragalisticexpialidotious'
>>> s[-5:5]
''
```

그 지표가 일치되지 않을 때 어떤일이 일어날지 이해가 되십니까? 아무런 문자열도 없습니다. 이봐, 그건 에러보다는 좋습니다, 그러나 여전히 원한바는 아닙니다. 그러면, 왜 내가 이 전체 섹션을 조각썰기 지표화에 바치고 있는지 이유를 알아 볼까요? 음, 나는 그것이 정말 진짜로 깔끔하다고 생각합니다. 또한 그것 덕분에, 기이한 어떤 때는 기막힌 (유용한 것은 말할 것도 없고) 방식으로 문자열들을 이어 붙일 수 있습니다. 유용하지-않은 방식으로 이것을 살펴 보세요:

```
>>> #두개의 이름을 혼합합니다. 이것을 더욱 복잡하게 할 수도 있는데, 다음과 같이
... #첫 번째 이름에 있는 한 모음뒤에 두 번째 이름의 자음이 따라옵니다.
... name1 = raw_input( "Type a name: " )
```

```
Type a name: Dave
>>> name2 = raw_input( "Type another name: " )
Type another name: Guido
>>> print name1[:3]+name2[-3:]
Davido
```

물론, 여러분은 이어붙이기와 조각썰기에 대하여 보다 더 실용적인 용도를 찾을 것이라고 확신합니다, 그러나 보세요, 나는 먼저 여러분에게 그것이 어떻게 작동하는 지를 보여야 합니다, 그렇지 않나요?

최상위로

파일 입/출력

시스템에 있는 파일에 접근하는 방법이 없는 프로그래밍 언어를 어떻게 가지겠습니까? 나도 진짜로 모르겠으나, 파이썬은 그렇게 할 수 있으니 상관 없습니다. 자, 그러면 어떻게 하는 것인지 궁금할 것입니다. 기본적으로, 먼저 파일을 열 필요가 있고, 그것에 객체를 할당할 필요가 있습니다. 객체라고? 그러한 것들에 대한 논의한 바 없습니다, 그렇지요? 객체는 C에서의 구조체와 (더욱 적절하게는 클래스)와 비슷하며, 다른 어떤 언어에서는 레코드와 비슷합니다. 기본적으로 그것은 데이타 형이며 (그리고 어떤 경우에는 특히) 함수를 포함하여 다른 데이타 형이 조립되어 구성됩니다. 파일 핸들(열려질 파일을 지시하는 변수)은 , 예상대로, 파일 객체 형을 가집니다. 여러분은 open 함수를 사용하여 파일핸들을, 다음과 같이, 할당할 수 있습니다:

```
>>> f = open( "myfile.txt", "r" )
>>> f                                   #이것은 변할 것입니다
<open file 'myfile.txt', mode 'r' at 0x8122338>
```

여러분은 myfile.txt가 무엇인지 알고 계시리라 생각합니다 (힌트: 열려질 파일). "r" 부분은 파일을 개방하기 위한 모드입니다. 파일을 여는 모드로 파이썬은 그 파일로 부터 데이타를 읽어 들이려고 하는지, 그 파일에 데이타를 쓰려고 하는지 등등을 알 수 있습니다. 여기에 여러분이 사용할 수 있는 모드의 목록이 있습니다:

- r - Reading 읽기
- w - Writing 쓰기
- a - Appending 추가 (쓰기와 동일, 단지 새로운 데이타는 그 파일의 끝에 씌여진다)
- r+ - Reading and writing 읽기와 쓰기
- 인수 없음 - Reading 읽기

파이썬은 만약 여러분이 이러한 것들 말고 어떤 모드를 지정할 지라도 에러를 보여 주지는 않을 것이라는 것을 주목하세요. 그렇지만, 나중에 예상치 못한 일들을 만들 가능성이 있으니 주의하세요. 윈도우와 맥에서는 또한 이진 모드, "b"가 있는데, 중요하기는 하지만, 약간은 혼란스럽습니다. 이진 모드는 newlines을 포함하여 모든 문자들을 정상적인 문자로 취급합니다. 다른 모드로는, 곧 살펴 보겠지만, newlines에 의지하여 데이타를 자르거나 혹은 찾을 수 있습니다. 이러한 newline-사용 메쏘드의 첫 번째는 readline입니다. 한 파일 객체에 대한 readline 메쏘드에 접근하려면, 그 객체의 이름과 점 뒤에 'readline()'을 놓으세요. 이것은 그 파일에 있는 다음 라인을 담은 문자열을 반환합니다. 또한 readlines도 사용가능한데, 이것은 파일의 모든 라인을 하나의 리스트에 읽어 들입니다. readlines는 선택적인 인수를 취할 수 있는데 그 파일로 읽어 들일 최대한의 바이트 수를 지정할 수 있으며, 게다가 한 라인의 끝에 도달하기 위해 필요한 무엇이라도 지정할 수 있습니다. 하나 더 주의 사항: readline (또는 readlines)는 그 파일의 끝에 다다르면 빈 문자열 ''를 반환합니다. 마지막으로, 예제입니다:

```
>>> f.readlines() #우리가 가상하는 파일; 원하시는대로 만드세요
['This is the first line\012', 'This is the second line\012', "Hmm I'm running out
```

```
of creative things to say\012", 'Alright, this is the last line']
>>> f.seek(0) #이것을 파일의 처음으로 되돌릴 필요가 있습니다; 아래를 보세요
>>> f.readline()
'This is the first line\012'
```

잘 이해를 못하시겠다면, 모든 문자열의 마지막에 있는 \012는 단지 newline입니다. 파일을 읽을 때, 파이썬은 \n과 같은 간단한 \ 문자들을 16진 (아스키) 코드로 대체합니다. 이해하지 못하겠다면, 그저 \012는 실제로는 \n이다 라고 기억하세요. 파일에 대하여 더욱 중요한 세 개의(어쩌면 네개) 메쏘드가 있습니다. 첫째는, read와 seek입니다. read는 단지 전체 파일을 하나의 문자열에 읽어 들입니다, 혹은 여러분이 지정하였다면 최대 개수의 바이트를 읽어 들입니다. seek은 한개의 인수를 필요로 합니다, 그것은 그 파일에서의 오프세트로서 가야할 혹은 탐색할 주소입니다. 흠? 자, 파이썬은 그 오프세트에 작은 포인터를 두고서 그것을 움직이므로써 파일을 읽습니다. 파일로 부터 읽어 들일 때, 그 작은 포인터는 여러분이 읽는 뒤쪽 부분으로 움직입니다. 만약 다시 읽으면, 그 포인터가 떠났던 곳으로부터 시작할 것입니다. seek은 그 파일에서 지정된 점으로 그 포인터를 (앞으로) 혹은 뒤로 가게 합니다. 여기에 또 다른 예제가 있습니다, 위와 같은 가상적인 파일을 이용합니다:

```
>>> f.seek(12) #오프세트는 0에서 시작합니다, 그러므로 12는 'f'입니다
>>> f.read(10)
'first line'
>>> f.seek(0)
>>> f.read() #전체 파일. 아마도 거대한 파일에 대해서 이렇게 하시면 안됩니다.
"This is the first line\012This is the second line\012Hmm I'm running out
of creative things to say\012Alright, this is the last line"
```

그리고, 마지막으로, write 함수가 있습니다, 그것은 (놀랍게도!) 여러분의 파일에 씁니다. 그저 괄호속에 쓰기를 원하는 문자열을 넣으세요. 그렇지만, 확실하게 파일이 모드 w, r+, 또는 a로 열려 있는지를 확인하세요! 또한, 새로운 파일을 생성하기를 원하신다면 반드시 모드를 w 또는 a로 사용해야지, r+ 로 사용하시면 안됩니다. 마지막 메쏘드는 close인데, 파일을 닫고 사용한 모든 메모리를 깨끗하게 풀어 줍니다. 파일작업이 끝나거나 혹은 떠나기 전에 모든 파일을 닫는 것은 좋은 프로그래밍 습관입니다. 이 섹션을 위한 마지막 예제입니다, 그러면 계속 가 볼까요:

```
>>> f = open( "mynewfile.txt", "w" )
>>> f.write( "This is some text" )
>>> f.close()
>>> f.read() #앗! 닫혀진 파일을 읽을 수가 없네요 (또는 그런 점이라면 읽기-전용 파일도 마찬가지입니다)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: I/O operation on closed file
>>> f = open( "mynewfile.txt" )
>>> f.read()
'This is some text'
```

최상위로

모듈

개관과 수입하기

우리가 파이썬의 광범위한 내장 모듈에 관하여 논의하기 전에 먼저, 모듈이란 무엇인지 그리고 어떻게 거기에 접근하는지를 검토하여 봅시다. 모듈은 스크립트로서 (혹은 컴파일된 이진파일) 핵심 언어인 파이썬으로부터 개별적으로 인터프리터 환경으로 적재될 수 있습니다. 모듈은 import 서술문으로 접근할 수 있습니다, 그 서술문은 다양한 방식으로 사용될 수 있습니다:

```
>>> import sys #이러한 예제에 대하여 우리는 내장 모듈 sys 를 사용할 것입니다
>>> sys.argv #객체에 접근하는 것과 마찬가지로 그 모듈의 이름에 접근합니다
['']
>>> from sys import argv #모듈의 이름을 호출하는 또 다른 방법입니다
>>> argv
['']
>>> from sys import * #sys 모듈에 있는 모든 것을 얻습니다
>>> stdout
<open file '<stdout>', mode 'w' at 0x80d34b0>
```

혼란스럽나요? 실제로는 그렇지 않습니다. 파고 들어가 봅시다. 첫 번째 라인은 모듈 sys를 (내장 모듈중의 하나) 수입합니다. 주목할 것은 다음 라인에서 우리는 sys의 구성원, argv (인터프리터로 건네지는 인수들)를, . (점)으로, 객체에 대하여 하였던 것처럼, 호출할 필요가 있다는 것입니다. 이것이 이른-바 sys 이름영역(namespace)를 사용하는 것입니다, 즉, 모든 sys의 구성원들은, 명시적으로, 그 이름 'sys'를 사용하여 접근될 수 있습니다. 다른 것으로는, 우리가 'from <module> import <members>'라고 배정하면, 우리는 전역 이름영역을 사용하는 것입니다. 그러면 그 메쏘드의 구성원에 다른 어떤 종류의 지역 변수 혹은 기본 함수들 처럼 접근할 수 있습니다. 만약 전체 모듈을 전역 이름영역에서 사용하고 싶으시면, 단지 다음과 같이 'from <module> import *' 하세요. 그렇게 어렵지 않지요, 그렇지요?

최상위로

내장 모듈

파이썬은, 다재다능한 언어로서, 방대한 모듈을 기본 배포본에 구축해 내장하였습니다. 내가 방대하다는 의미는: 나의 생각으로, 파이썬 2.0에 (무용지물이 된것들과 OS-종속적 모듈을 포함하여) 196개의 모듈이 사용가능하다는 것입니다. 내가 "내장된(builtin),"이라고 말한 의미는 그것들을 수입할 필요가 없다는 것을 뜻하는 것이 아니라, 단지 파이썬의 설치에 포함된다는 것입니다. 그래서, 여기에서 약 다섯 개의 선택을 나열하겠습니다, 그것들이 무엇을 하는 것인지 기본적인 설명과 함께, 약간의 구성요소들을 나열해 보겠습니다.

- **sys** - 기본적인 시스템/프로그램 함수
  - **argv** - 인터프리터에 건네지는 명령어 리스트
  - **stdout, stdin, stderr** - 기본적인 표준 출력, 표준 입력, 그리고 표준 에러. 이러한 것들은 정상적인 파일처럼 접근할 수 있습니다.
  - **exit** - 프로그램을 우아하게 종료하며, 종료 코드로 하나의 값을 접수합니다 (기본적으로 0=정상종료, non-0=에러)
  - **path** - 파이썬이 모듈을 수입하기 위해서 탐색할 때 찾아보는 경로
- **math** - 수학적인 함수와 상수. 여기에서 그 모든 것들을 나열하지는 않겠습니다, 그러나 math 모듈은 많은 함수들을 가지고 있어서 sin, acos, tanh, log, 등등이 있습니다. 여기에 상수들이 있습니다:
  - **pi** - 수학적인 상수 π. 원주율로서, 대략 3.1415926535897931 입니다.
  - **e** - 자연 대수(ln)의 밑수. 0->무한대인 모든 숫자 k에 대하여 1/k!의 합의 해를 얻어 보세요. 이상하게도, 계승 (그리고 무한대)는 math 모듈에 포함되어 있지 않습니다.
- **string** - 문자열을 다루기 위한 함수

- **printable** - 이른-바 출력가능한 문자들을 담은 문자열. 또한 숫자, 기호 (대문자와 소문자), 공백, 그리고 구두점도 사용가능합니다.
- **atoi, atof, atol** - 문자열을 각각 정수, 실수, 그리고 배정도 정수로 변환합니다. 파이썬 2.0에서는 내장 함수 int, float, 그리고 long을 사용해야 합니다.
- **find** - 두 번째 문자열 (인수)가 첫 번째 문자열에서 나타나는 제일 낮은 곳의 지표를 발견한다.
- **count** - 첫 번째 문자열에서 두 번째 문자열이 나타나는 빈도를 센다.
- **split** - 첫 번째 인수로 주어진 문자열을 공백 문자로 쪼갠다; 리스트를 반환한다. 두 번째 인수가 주어지면, 그 문자열은 공백대신에 그 인수에 따라서 쪼개진다.
- **strip** - 주어진 문자열에서 선두 그리고 후미의 공백을 제거한다.
- **replace** - 첫 번째 문자열에서 모든 두 번째 문자열의 출현을 세 번째 문자열로 대체한다. 음, 이런식으로 생각하세요: replace(str, "old", "new")는 str에서 모든 "old"의 출현을 "new"로 대체한다.

- **random**
  - **seed** - 무작위수 생성기에 세 개의 선택적인 인수를 가지고 씨값을 매긴다; 기본값은 현재 시간
  - **random** - 다음의 무작위 수를 0 과 1 사이의 부동소수점 수로 반환한다.
  - **randint** - 주어진 두 개의 정수 사이에 있는 무작위 수를 반환한다.
  - **uniform** - randint와 동일, 오직 부동소수점수와만 작동한다.
  - **choice** - 지정된 리스트 혹은 터플로부터 한 요소를 무작위로 선택한다.
- **time** - 지역 시간 과 임의 시간을 조작하고 획득하는 함수. 더 완전한 정보는 파이썬 라이브러리 문서를 보세요.
  - **time** - 인수가 없습니다, 시스템의 현재 시간을 ...어떤 시간 이후로, 초단위로 나타내는 긴 숫자를 반환합니다. 실제로는 일종의 해석기 없이는, 무작위 수 씨값을 매기는 것을 제외하고는, 유용하지 않음.
  - **localtime** - 일종의 해석기. 시간이 주어지면 (time 함수가 반환하는 것처럼), 그것은 다음과 같은 정보를 가지는 터플을 반환한다: year (four-digit), month (1-12), day (1-31), hour (0-23), minute (0-59), second (0-61 - 60 & 61은 윤초를 위한 것이며, 절대로 사용되지 않는다), weekday (0-6, 월요일은 0이다), day (1-366), 일광 절약 시간 (0 - no, 1 - yes, -1 - 시스템 기본값). 유용합니다 :)
  - **gmtime** - 위와 동일, 여러분 시스템의 지역 시간이 아니라, 그리니치 Mean Time이라는 점만 제외.
  - **asctime** - localtime 또는 gmtime이 주어지면, 다음 'Thu Mar 22 18:24:35 2001' 형태의 문자열을 반환한다.
- **re** - 정규 표현식 (regexp) 연산자. 여기에서 그것들을 다루지는 않겠습니다, 그러나 여러분 펄 팬들은 파이썬에도 존재한다는 것을 아실 것이라고 생각합니다. 파이썬의 regexp에는 명확하게 특이성이 있습니다, 그러므로 문서를 읽어 보세요...문서는 여러분의 친구입니다 :).

최상위로

사용자-정의 모듈과 함수

물론, 여러분 자신의 모듈을 사용할 수 없다면 이러한 모듈 모두가 무슨 소용이 있겠습니까? 자, 사용자-정의 모듈은 .py 파일로서 파이썬의 모듈 디렉토리중의 하나에 있습니다. 그 모듈을 수입하면, 그 모듈의 실행코드, 모든 실행함수, 변수, 클래스, 등등이 수입되어 들어옵니다. 다른 모듈을 수입하는 것과 마찬가지로 어떤 것을 수입할지 지정할 수 있습니다. 자, 그러면, 이 정도면 훌륭하게 잘 설명되었습니다, 그러나 이러한 모듈의 대부분은 함수입니다. 그러면, 자신만의 함수를 만드는 법을 공부해 보실까요? 좋습니다.

좋습니다. 우리는 함수를 원합니다. 나는 수학을 좋아하므로, 우리는 math 모듈에 있는 log 함수를 이용하여 임의의 밑수를 가지는 간단한 로그를 연습해 볼 것입니다:

```
>>> def logb( b, n ):
...   from math import log
...   a = log(n)/log(b)
...   if a - int(a) == 0: #a는 정수입니다
...     return int(a)
...   else:
...     return a
...
```

얼마나 멋진가요. 고등학교 대수학을 공부하시지 않았을 때에만, 약간 더 설명이 필요할 뿐입니다 (그렇게 화내지 마세요...나는 이제 겨우 신입생입니다 :). 이 'def'란 놈은 무엇인가? 자, 그것은 함수의 이름과 그것이 취하는 인수일 뿐입니다. 이제 어떤 인수가 좋을 지는 여러분이 생각해 주셨으면 좋겠습니다. 그러면 반환값이 있습니다. 그것은...어...함수가 반환하는 것입니다 :). 실행은, break 서술문과 마찬가지로 return 서술문 뒤에서 중지합니다. 무언가를 반환하지만 않는다면 말이지요. 좋습니다. 이제, 말할 것이 다 떨어진 고로...여기에 예제 모듈을, 단지 연습용으로, 보여 드립니다, 이봐, 이거 지침서 맞아, 그렇지 않아? 시작합시다, 이것을 moremath.py에 넣으세요:

```
#!/usr/bin/python

def logb( b, n ): #Definition
  from math import log #Need this!
  a = log(n)/log(b) #Change of base formula
  if a - int(a) == 0: #a is an int
    return int(a) #Get rid of that decimal :)
  else: #a is a float
    return a #Just return it

#Ye Olde Golden Ratio <www.verbose.net>
phi = 1.6180339887498948
```

이제 인터프리터에서는:

```
>>> import moremath
>>> moremath.phi #부동소수점 수는 그렇게 정밀하지 않습니다, 여러분이 인지 하시다시피
1.6180339887499999
>>> from moremath import logb
>>> logb(2,1024)
10
```

와우! 모듈입니다! 다른 모듈과 똑 같은 모듈입니다! 그러나 내 스스로 여러분이 해 내셨습니다! 또는 여러분 스스로 내가 해 냈습니다....또는...짬깐...여러분은 여러분 스스로 해 내셨습니다. 이제 moremath.py를 저장해 놓은 디렉토리를 들여다 보시면, moremath.pyc 파일을 발견할 수 있습니다. 이것은'편집된(compiled)' 파일입니다, 그것은 실제로 컴파일 된 것은 아니고 (바이트 코드로서 -일종의 자바의 .class 파일과 비슷하여서), .py 파일 보다 적은 메모리 부담을 가집니다. 그렇지만, 내가 아는 한 .pyc 파일은 이식성이 없습니다, 그래서 만일 이 모듈을 자신을 위해서 사용하고자 한다면 .pyc 파일을 사용할 수 있으며 만약 배포하고자 한다면, 권고하건데 .py 파일을 유지하세요. 어떤 경우라도, 여전히 .py 파일을 가지고 있으면 확실히 더 쉽게 소스를 편집할 수 있습니다. 질문 있나요 :)?

최상위로

닫으면서

파이썬이 펄보다 나은 이유

인정합니다: 파이썬이 펄보다 더 좋은 것은 아닐 수도 있습니다. 적어도, 그것은 다릅니다. 많은 부분에서 파이썬이 더 *좋습니다*. 그리고 그것은 분명한 장점을 가집니다, 그리고 내가 살펴본, 혹은 나열해 본, 등등 그것이 바로 내가 생각한것입니다. 시작해 보겠습니다.

- 직관성. 골치아픈, 불명확한 코드 대신에, 파이썬이 사용하는 구문과 블록은 약간은, 정상적인 (영어) 구어와 들여쓰기와 유사합니다.
- 고-수준성. 리스트 조각썰기(slicing), 터플 포장하기(packing), 등등. 고-수준의 데이타 조작을 허용하여 주므로 포인터와 같은 복잡하고 골치아픈 모든 것들이 필요 없습니다...잠깐만요...그것은 C입니다...비록 효율적이긴 하지만, 때로는 ?erhacker인 나조차도 헷갈리게 하곤 합니다. 그러나 그것은 펄보다는 파이썬에서 더 쉽습니다.
- 더욱 통합된 객체-지향 (OO) 지원. 펄에서의 OO는 클래스와 괴이한 것등등과 같은 해쉬와 하는 작업에 연관되어 있습니다. 기초에서부터, 파이썬은 OO를 목표로 디자인 되었습니다...그렇지만 클래스에 관하여 많이 논의하지는 않았습니다. 죄송합니다.
- 더 훌륭한 C 통합. 물론, 이것은 여기에서 논쟁거리입니다, 그러나 나는 파이썬이 펄보다는 더 쉽게 C에 내장될 수 있다고 생각합니다...무어라고 말할 수 있을까요?
- 상호대화 모드. 별개의 파일로 스크립트를 배정할 필요가 없습니다. 제가 말하는 바를 이해하실 겁니다.
- 광범위한 데이타 구조. 펄은 배열, 해쉬, 그리고 스칼라를 가집니다. 파이썬은 스칼라(정수, 실수, 배정도 정수), 리스트, 사전 (내가 언급하지 않았습니다; 해쉬 혹은 연관 배열과 유사합니다), 터플을 가집니다. 그리고 파이썬은 확장성이 있으며, 그 문서에 따라서 진화합니다.
- range 함수. 사람들은 펄이 가진것과 같은, 총괄적인 for 회돌이가 없다고 불평합니다. range야말로 진짜 for 회돌이를 실현하는 것이며...완전히 다른 형태의 회돌이를 가질 필요가 없습니다. 충돌부담도 더 적습니다. 그리고 range를 다른 용도로도 사용할 수 있습니다.
- 몬티 파이썬. 'Nuff가 가라사대.

최상위로

참조서

- Python.org. 필-독.
- 현재의 파이썬 문서 당연히 읽어야함.
- Zope. 파이썬으로 작성된 대단한 웹서버.
- 프로그래머의 천국에 있는 파이썬 섹션. 이외의 훌륭한 연결점.
- Black Sun Research Facility. 네트워크, 프로그래밍, 크랙, 운영체제, 등등을 위한 엄청난 사이트. 이 지침서는 거기에서 발간되어야 합니다 :)
- Pythonline. 몬티 파이썬, 공식적인 사이트인지 아닌지, 확신 못합니다.

어떠한 비평, 질문, 열렬한 칭찬, 비방, 모욕편지, 등등 편안하게 나에게 편지하세요 또는 나에게 메시지를 BSRF의 메시지 보드에 남겨주세요. 회원으로서, 정기적으로 점검합니다. 모욕편지도 좋습니다, 그러나 죽인다는 위협은 사양합니다.

자 이제, 기나긴 여정끝에, 결말에 도달했습니다. 여러분이 무언가를 배웠기를 바라며, 너무 아드레날린이 분비되어 격렬하게 코딩하지 않기를 바랍니다. 이렇게 끝내기가 어색합니다, 그렇지만 이제...이만...여기서...끝내야겠습

니다.

최상위로