# Python을 하기 위한 당신의 욕구 자극하기

* 1. 컴퓨터로 일을 하다보면, 결국 자동화 하기 위한 방법을 찾게됨.   
     (큰 텍스트 파일 정리, 이미지 정렬 등)
  2. 전문적인 언어(C++, java 등의 언어)는 프로그램을 만들기까지 너무나도 지루함.   
     (컴파일, 테스트 등)
  3. 파이썬은 어떤 OS에서든 더 간단하고 빠르게 작업을 완수 할 수 있게 해줌. (모듈화, 재사용, 각종 툴킷 기능 지원)
  4. 파이썬은 interpreted 언어이므로 컴파일, 링킹 과정이 필요가 없다! 즉각적으로 실행 결과를 테스트할 수 있음.
  5. 파이썬의 프로그램은 작고 읽기 쉽게 작성됨.   
     파이썬으로 작성된 프로그램들은 C++, java등의 프로그램보다 코드 크기가 작다.
     + 복잡한 연산을 쉽게 표현 가능.
     + {} 가 필요 없음 (indent만 필요함)
     + 변수 선언할 필요 없음.
  6. 파이썬은 확장 가능함. (C로 작성된 모듈을 쉽게 호출함)  
     만약 속도가 중요한 프로그램이거나 파이썬으로 사용할 수 없는 바이너리 형식은 C로 만든 모듈을 호출 할 수 있음!

**※ 아무튼 파이썬은 BBC쇼의 어떤 프로에서 따온 것으로 파충류랑 전혀 연관이 없음.**

# 파이썬 해석기 사용

* 1. 해석기 호출! (터미널에 아래 명령어 타이핑, python 3.8 기준)

|  |
| --- |
| python3.8 |

end-of-file 키를 입력하면 종료됨 (Ctrl+D-based on Unix, Ctrl+Z-based on Windows)  
아니면 quit(), exit() 등의 함수 호출!

해석기를 그냥 실행하면 실시간으로 명령을 주고 받으며 종료할 때까지 실행된다.

해석기 파라미터로 -c를 쓰면, 명령만 실행하고 python을 종료시킨다.

-m 파라미터를 쓰면 특정 파일, 모듈을 로드한다.

-I 파라미터를 추가하면, 실행 후 계속 실시간 명령을 주고받을 수 있음.  
**※ 파라미터 docs** [**https://docs.python.org/3/using/cmdline.html#using-on-general**](https://docs.python.org/3/using/cmdline.html#using-on-general)

* + 1. Argument Passing (인자 전달)

스크립트 이름과 몇가지 인자가 추가된 후에 sys 모듈의 argv 변수로 변환이 됨.  
sys모듈의 argv 멤버를 확인하면 어떤 Argument를 입력했는지 확인 가능

|  |
| --- |
| import sys  print(sys.argv[0])  print(len(sys.argv)) |

* + 1. Interactive Mode (상호작용 모드)

tty? : Terminal Control Functions? 아마 python 터미널 인터페이스를 뜻하는 듯?  
일반적으로 python을 실행하면 나오는 대화형 터미널을 뜻해 보인다.  
보통 >>> 사인은 핵심 프롬프트  
… 사인은 두번째 프롬프트

해석기는 환영 메시지로 버전 정보와 저작권에 대한 정보를 알린다.

Continuation lines들은 multi-line construct에서 요구된다. (if나 for문 등의 statement)

* 1. 해석기와 구동 환경
     1. Encoding

기본적으로 모두 UTF-8로 대함. UTF-8 인코딩은 대부분의 언어들을 표현할 수 있는 인코딩이다.

혹시나 다른 인코딩을 사용한다면, 아래와 같은 주석을 첫줄에 표시.

|  |
| --- |
| *# -\*- coding: cp1252 -\*-* |

# 파이썬 소개

* 1. 파이썬 계산기
     1. Numbers

해석기는 간단한 계산기로도 행동한다. 식은 굉장히 직관적.  
+, -, \*, /, 그리고 (, ) 등등.

정수는 int  
분수를 가지는 것은 float

/ 연산자는 항상 float를 반환한다.  
만약 나눈 후 int를 반환 받고 싶다면 // 연산자를 사용!

\*\*의 경우 power 연산자 (제곱)

interactive mode의 경우에는 \_라는 이름의 변수에 마지막으로 출력한 값이 저장됨.  
(이 변수는 무조건 read-only로 대해야함

int float외에도 Decimal과 Fraction 타입을 지원한다.  
python은 complex number(허수?)에 대한 개념을 가지고 있으며 5j 등 suffix 사용.

* + 1. Strings

String liternal로 single quote, double quote로 감싸는 것 둘다 사용 가능함.  
\ 문자는 escape quote로 사용

\' \t \" \n 등

""", ''' 으로 3개를 연속으로 사용한 경우는 \n 없이 end-of-line을 입력할 수 있게함.

\* 연산자로 string을 여러 번 반복할 수 있음.

+ 연산자로 string 붙이기 가능.

2개 이상의 string literals 들은 자동으로 연결된다. '1' '2' -> '12'   
(이것은 특히 소스 코드에서 여러 줄로 나누어 입력하고 싶을 때 유용)  
**하지만 변수와는 같이 사용할 수 없음. (만약, 이 경우는 + 연산자 사용)**

String은 배열처럼 색인이 될 수 있다.

|  |
| --- |
| **>>>** word = 'Python'  **>>>** word[0] *# character in position 0*  'P'  **>>>** word[5] *# character in position 5*  'n'  **>>>** word[-1] *# last character*  'n'  **>>>** word[-2] *# second-last character*  'o'  **>>>** word[-6]  'P'  **>>>** word[0:2] *# characters from position 0 (included) to 2 (excluded)*  'Py'  **>>>** word[2:5] *# characters from position 2 (included) to 5 (excluded)*  'tho'  **>>>** word[:2] + word[2:]  'Python'  **>>>** word[:4] + word[4:]  'Python'  **>>>** word[:2] *# character from the beginning to position 2 (excluded)*  'Py'  **>>>** word[4:] *# characters from position 4 (included) to the end*  'on'  **>>>** word[-2:] *# characters from the second-last (included) to the end*  'on' |

|  |
| --- |
| +---+---+---+---+---+---+  | P | y | t | h | o | n |  +---+---+---+---+---+---+  0 1 2 3 4 5 6  -6 -5 -4 -3 -2 -1 |

String 타입은 불변타입이다. 즉, string 내부의 char를 변경할 수 없음.  
만약 어떤 글자를 바꾸자고자 한다면, 아래처럼 새로 공간을 할당하여 만들어야함.

|  |
| --- |
| **>>>** 'J' + word[1:]  'Jython'  **>>>** word[:2] + 'py'  'Pypy' |

내장된 len 함수는 string의 길이를 반환한다.

* + 1. List

다양한 데이터 타입을 묶기 위해 일반적으로 List라는 타입을 사용.

**>>>** squares = [1, 4, 9, 16, 25]

**>>>** squares

[1, 4, 9, 16, 25]

, 로 각 요소를 나누며 [, ]으로 감싼다.

list는 각 요소를 slicing 하는 indexing이 있다.

**>>>** squares[-3:] *# slicing returns a new list*

[9, 16, 25]

이 경우, -3에서부터 마지막 요소까지의 list를 새로 만들어 반환한다.  
이때 Shallow Copy로 list가 생성되니 참고. (Deep Copy와의 차이점?)

list는 + 연산자를 통해 2개의 그룹을 합성할 수 있다.

**>>>** cubes = [1, 8, 27, 65, 125] *# something's wrong here*

**>>>** 4 \*\* 3 *# the cube of 4 is 64, not 65!*

64

**>>>** cubes[3] = 64 *# replace the wrong value*

**>>>** cubes

[1, 8, 27, 64, 125]

list에 append 함수를 이용하여 새로운 아이템을 list의 끝에 추가할 수 있다.

slicing으로 할당하는 것도 허용한다. 이것은 요소 내용을 clear하거나 수정할 수 있다.

**>>>** letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

**>>>** letters

['a', 'b', 'c', 'd', 'e', 'f', 'g']

**>>>** *# replace some values*

**>>>** letters[2:5] = ['C', 'D', 'E']

**>>>** letters

['a', 'b', 'C', 'D', 'E', 'f', 'g']

**>>>** *# now remove them*

**>>>** letters[2:5] = []

**>>>** letters

['a', 'b', 'f', 'g']

**>>>** *# clear the list by replacing all the elements with an empty list*

**>>>** letters[:] = []

**>>>** letters

[]

내장된 len 함수 역시 list의 길이를 반환한다.

중첩된 list를 통해 다차원 배열을 구현할 수 있다.

* 1. 파이썬 코딩 첫 걸음

파이썬은 아래 예시처럼, 한번에 2개 이상의 변수를 한번에 초기화 또는 계산할 수 있다.  
(아래 예시 참고)

**>>>** *# Fibonacci series:*

**...** *# the sum of two elements defines the next*

**...** a, b = 0, 1

**>>> while** a < 10:

**...**  print(a)

**...**  a, b = b, a+b

- 첫 줄의 경우 다중 할당을 포함한다. 이것은 a, b 변수에 동시에 0, 1 값을 할당.

- 마지막 줄은 이것을 다시 사용하는데, 오른쪽의 식부터 처리되는 것을 보여준다. 즉, b, a +b가 먼저 수행 된 후 a, b에 할당된 다는 것이다.

* + 1. print 함수

이 함수는 주어진 argument의 값을 write한다.  
Multiple Argument를 이용하면 공백과 함께 문자열이 합쳐서 출력된다.  
이때 end나 구분자를 변경할 수도 있다.

**>>>** a, b = 0, 1

**>>> while** a < 1000:

**...**  print(a, end=',')

**...**  a, b = b, a+b

**...**

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,