IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

파이썬 (Python)

파이썬 (Python) 이란?

파이썬(Python)은 1990년 암스테르담의 *귀도 반 로섬(Guido Van Rossum)*이 개발한 인터프리터 언어이다. 파이썬은 배우기 쉽고 강력하며 효율적인 자료 구조들과 객체 지향 프로그래밍에 대해 간단하고 효과적인 접근법을 제공한다. 우아한 문법과 동적 타이핑(typing)은 인터프리터의 특징들과 더불어 대부분의 플랫폼과 다양한 문제 영역에서 스크립트 작성 및 빠른 응용 프로그램 개발에 이상적인 환경을 제공한다. 파이썬의 인터프리터와 풍부한 표준 라이브러리는 소스나 바이너리 형태로 파이썬 공식 사이트(https://www.python.org/) 에서 무료로 제공되고 있으며 자유롭게 배포할 수 있다. 제 3자들이 무료로 제공하는 확장 모듈, 프로그램, 도구, 문서들의 배포판이나 링크를 포함하고 있다. 파이썬 인터프리터는 C나 C++(또는 C에서 호출 가능한 다른 언어들)로 구현된 새 함수나 자료 구조를 쉽게 추가할 수 있으며 고객화 가능한 응용 프로그램을 위한 확장 언어로도 적합한 프로그래밍 언어이다.

파이썬의 특징

파이썬의 가지고 있는 특징은 다음과 같다.

1. 인터프리터

앞에서 설명했듯 파이썬은 인터프리터 언어이다. 인터프리터란 프로그래밍 언어로 작성된 소스 코드를 바로 실행할 수 있는 프로그램 혹은 환경을 의미한다. 원시 코드를 기계어로 해석해주는 컴파일러(complier)와는 다르지만 현대에 들어서 컴파일러와 인터프리터를 같은 개념으로 바라보는 경우가 늘었다. 파이썬은 인터프리터 언어이므로 다른 프로그래밍 언어(C/C++, Java 등)들과 비교하면 속도가 느린 편이다. C/C++과는 약 5배~80배 정도 차이가 나며 자바와는 5배~20배 정도 차이가 난다. 다만 다른 프로그래밍 언어들과 생산성을 비교하면 굉장히 뛰어난 편이며 파이썬으로 만든 파이썬(PyPy)같은 경우 성능 향상이 되어 이전과 비교하면 훨씬 빨라졌고 프로그래밍을 대화식으로 할 수 있기 때문에 파이썬이 교육용으로도 많이 추천되고 인기가 많은 비결이라고 할 수 있다.

2. 생산성

파이썬은 간결한 문법과 풍부한 라이브러리 제공 등으로 개발하는데 있어 생산성이 뛰어나다. 파이썬의 인터프리터는 REPL(Read Eval Print Loop)이다. REPL이란 사용자가 명령어를 입력하면 시스템이 그에 해당하는 결과를 반환하는 환경을 말한다. 이러한 대화식 환경을 제공함으로써 소스 코드를 작성하고 별도의 컴파일 작업 없이 곧바로 결과를 알 수 있어 생산성이 뛰어나다는 강점을 가지고 있다.

3. 멀티 패러다임(Multi-paradigm) 지원

파이썬은 대표적으로 함수형(Functional) 패러다임과 객체 지향(Object-Oriented) 패러다임 두 가지 모두를 지원하고 있다. 따라서 개발하는데 있어 프로그래머의 취향대로, 다양한 방식으로, 다양한 관점으로 생각하며 프로그래밍을 할 수 있으며 이를 응용한 수많은 기법과 디자인 패턴을 접목시킬 수 있는 특징을 가지고 있다.

4. Glue Language

파이썬은 Glue Language라는 특징 때문에 다른 언어들과 잘 호환된다. C언어로 구현한 CPython(c.f: cython), Java로 구현한 Jython, 심지어 파이썬으로 파이썬을 구현한 PyPy 등 파이썬은 대체 가능한, 특정 플랫폼에 맞춰진 구현체를 가지고 있다는 특징이 있다.

5. 다양한 라이브러리 제공

언어적 특징과는 별개로 파이썬은 플랫폼에 상관없이 다양한 종류의 수많은 표준 라이브러리(standard library)를 기본으로 탑재되어 있다. 또한 다양한 종류의 수많은 오픈소스 라이브러리들이 존재하며 공식 사이트(https://pypi.org/)가 존재한다. 이러한 라이브러리들은 pip를 통해 손쉽게 받을 수 있다.

6. General Purpose

import antigravity 를 통해 나오는 재미난 만화에서 파이썬의 철학을 볼 수 있으며, 파이썬은 모바일 지원에 대한 약점이 존재한다. (참고: https://en.wikipedia.org/wiki/List_of_Python_software)

In [1]:
```
import antigravity
```

프로그래밍의 구성 요소

프로그래밍

프로그래밍이란 문법(키워드,식,문)을 이용해서 값을 입력받고 계산/변환하여 출력하는 흐름을 만드는 일이다.

[The Zen of Python (PEP 20)]

대화형 셀에 다음 코드를 입력해본다.

```
import this
```

실행해보면 파이썬의 철학을 엿볼 수 있다.

프로그래밍 언어의 문법 • 생각을 표현해내는 도구인 동시에, 생각이 구체화되는 틀 • 언어가 지향하고자 하는 철학에 따라 고안하였다. 문법에 대한 올바른 이해는 프로그래밍을 위한 필수적인 과정.

BDFL(Benevolent Dictator For Life!) 자비로운 종신 독재자 : Guido van Rossum, 파이썬의 창시자 https://en.wikipedia.org/wiki/Benevolent_dictator_for_life
https://legacy.python.org/doc/essays/

In [2]:
```
import this
```

```
The Zen of Python, by Tim Peters
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

`import this` 를 실행했을 때 나오는 장문을 번역하면 다음과 같다.

- 아름다운 것이 보기 싫은 것보다 좋다.
- 명시적인 것이 암묵적인 것보다 좋다.
- 간단한 것이 복합적인 것보다 좋다.
- 복합적인 것이 복잡한 것보다 좋다.
- 수평한 것이 중첩된 것보다 좋다.
- 희소한 것이 밀집된 것보다 좋다.
- 가독성이 중요하다.
- 규칙을 무시할 만큼 특별한 경우는 없다.
- 하지만 실용성이 순수함보다 우선한다.
- 에러가 조용히 넘어가서는 안된다.
- 명시적으로 조용히 만든 경우는 제외한다.
- 모호함을 만났을 때 추측의 유혹을 거부해라.
- 하나의 — 가급적 딱 하나의 — 확실한 방법이 있어야 한다.
- 하지만 네덜란드 사람(귀도)이 아니라면 처음에는 그 방법이 명확하게 보이지 않을 수 있다.
- 지금 하는 것이 안하는 것보다 좋다.
- 하지만 안하는 것이 이따금 지금 당장 하는 것보다 좋을 때가 있다.
- 설명하기 어려운 구현이라면 좋은 아이디어가 아니다.
- 설명하기 쉬운 구현이라면 좋은 아이디어다.
- 네임스페이스는 아주 좋으므로 더 많이 사용하자!

문법 (키워드, 식, 문)

Keywords = Reserved Words (3.6 기준)

import keyword 를 통해 파이썬에 정의되어 있는 키워드를 확인할 수 있다.

keyword로 정의되어있는 것은 identifier로 사용하지 않는다.

In [3]:
```
import keyword
keyword.kwlist
```

Out[3]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

Expression vs Statement (표현식 vs 구문)

Expression vs Statement (표현식 vs 구문)

표현식(expression) = 평가식 = 식

- 값
- 값들과 연산자를 함께 사용해서 표현한 것
- 이후 '평가'되면서 하나의 특정한 결과값으로 축약
  - 수
    - 1 + 1
      - 1 + 1 이라는 표현식은 평가될 때 2라는 값으로 계산되어 축약
      - 0과 같이 값 리터럴로 값을 표현해놓은 것
    - 문자열
      - 'hello world'
      - "hello" + ", world"
    - 함수
      - lambda
- 궁극적으로 '평가'되며, 평가된다는 것은 결국 하나의 값으로 수렴한다는 의미
  - *python에서는 기본적으로 left-to-right로 평가*

구문(statement) = 문

- 예약어(reserved word, keyword)와 표현식을 결합한 패턴
- 컴퓨터가 수행해야 하는 하나의 단일 작업(instruction)을 명시.
- 할당 (대입, assigning statement)
  - python에서는 보통 '바인딩(binding)'이라는 표현을 씀, 어떤 값에 이름을 붙이는 작업.
- 선언(정의, declaration)
  - 재사용이 가능한 독립적인 단위를 정의. 별도의 선언 문법과 그 내용을 기술하는 블럭 혹은 블럭들로 구성.
    - Ex) python에서는 함수나 클래스를 정의
  - 블럭
    - 여러 구문이 순서대로 나열된 덩어리
    - 블럭은 여러 줄의 구문으로 구성되며, 블럭 내에서 구문은 위에서 아래로 쓰여진 순서대로 실행.
    - 블럭은 분기문에서 조건에 따라 수행되어야 할 작업이나, 반복문에서 반복적으로 수행해야 하는 일련의 작업을 나타낼 때 사용하며, 클래스나 함수를 정의할 때에도 쓰임.
- 조건(분기) : 조건에 따라 수행할 작업을 나눌 때 사용.
  - Ex) if 문
- 반복문 : 특정한 작업을 반복수행할때 사용.
  - Ex) for 문 및 while 문
- 예외처리

Type

- 값에 대한 type이 중요함
- 값에 따라 서로 다른 기술적인 체계가 필요
  - 지원하는 연산 및 기능이 다르기 때문
- 컴퓨터에서는 이진수를 사용해서 값을 표현하고 관리
  - 정확하게 표현하지 못할 수가 있음
- 숫자형 (numeric)
  - 산술 연산을 적용할수 있는 값
  - 정수 : 0, 1, -1 과 같이 소수점 이하 자리가 없는 수. 수학에서의 정수 개념과 동일 (int)
  - 부동소수 : 0.1, 0.5 와 같이 소수점 아래로 숫자가 있는 수 (float)
  - 복소수 : Python에서 기본적으로 지원
- 문자, 문자열
  - 숫자 "1", "a", "A" 와 같이 하나의 낱자를 문자라 하며, 이러한 문자들이 1개 이상있는 단어/문장와 같은 텍스트
  - Python에서는 낱자와 문자열 사이에 구분이 없이 기본적으로 str 타입을 적용
    - byte, bytearry
- 불리언 (boolean)
  - 참/거짓을 뜻하는 대수값. 보통 컴퓨터는 0을 거짓, 0이 아닌 것을 참으로 구분
  - True 와 False 의 두 멤버만 존재 (bool)
  - **Python에서는 숫자형의 일부**
- Compound = Container = Collection
  - 기본적인 데이터 타입을 조합하여, 여러 개의 값을 하나의 단위로 묶어서 다루는 데이터 타입
  - 논리적으로 이들은 데이터 타입인 동시에 데이터의 구조(흔히 말하는 자료 구조)의 한 종류. 보통 다른 데이터들을 원 소로 하는 집합처럼 생각되는 타입들
  - Sequence
    - list : 순서가 있는 원소들의 묶음
    - tuple : 순서가 있는 원소들의 묶음. 리스트와 혼동하기 쉬운데 단순히 하나 이상의 값을 묶어서 하나로 취급하는 용도로 사용
    - range
  - Lookup
    - mapping
      - dict : 그룹내의 고유한 이름인 키와 그 키에 대응하는 값으로 이루어지는 키값 쌍(key-value pair)들의 집합.
      - set : 순서가 없는 고유한 원소들의 집합.
  - None
    - 존재하지 않음을 표현하기 위해서 "아무것도 아닌 것"을 나타내는 값
    - 어떤 값이 없는 상태를 가리킬만한 표현이 마땅히 없기 때문에 "아무것도 없다"는 것으로 약속해놓은 어떤 값을 하나 만들어 놓은 것
    - None 이라고 대문자로 시작하도록 쓰며, 실제 출력해보아도 아무것도 출력되지 않음.
    - 값이 없지만 False 나 0 과는 다르기 때문에 어떤 값으로 초기화하기 어려운 경우에 쓰기도 함

연산

- 산술연산
  - 계산기

- 비교연산
  - 동등 및 대소를 비교. 참고로 '대소'비교는 '전후'비교가 사실은 정확한 표현
  - 비교 연산은 숫자값 뿐만 아니라 문자열 등에 대해서도 적용할 수 있음.
- 비트연산
- 멤버십 연산
  - 특정한 집합에 어떤 멤버가 속해있는지를 판단하는 것으로 비교연산에 기반을 둠
  - is, is not : 값의 크기가 아닌 값 자체의 정체성(identity)이 완전히 동일한지를 검사
  - in, not in : 멤버십 연산. 어떠한 집합 내에 원소가 포함되는지를 검사 ('a' in 'apple')
- 논리연산
  - 비교 연산의 결과는 보통 참/거짓. 이러한 불리언값은 다음의 연산을 적용. 참고로 불리언외의 타입의 값도 논리연산을 적용

연산자 우선순위

	Operator	Description
lowest precedence	or	Boolean OR
	and	Boolean AND
	not	Boolean NOT
	in, not in	membership
	==, !=, <, <=, >, >=, is, is not	comparisons, identity
	|	bitwise OR
	^	bitwise XOR
	&	bitwise AND
	<<, >>	bit shifts
	+, -	addition, subtraction
	*, /, //, %	multiplication, division, floor division, modulo
highest precedence	+x, -x, ~x	unary positive, unary negation, bitwise negation
	**	exponentiation

PEMDAS :

Parentheses - **E**xponentiation - **M**ultiplication - **D**ivision - **A**ddition - **S**ubtraction

Python 문법의 특징

- 규칙(키워드 등)의 수가 적고 대부분의 규칙이 일관된 맥락을 가지고 있음
  - Python 3 : 35 (True, False)
  - Python 2 : 31
  - C++ : 62
  - Java : 53
  - Visual Basic : >120

다른 프로그래밍 언어에 비해 키워드 수가 적다는 것을 알 수 있다.

- case sensitive
- space sensitive (Indentation / line oriented)
  - Indentation used to define block structure – Multiple statements can occur at same level of indentation
  - Implicit continuation across unclosed bracketing ((...), [...], {...})
  - Forced continuation after \ at end of line
  - possible to combine multiple statements (Compound statements) on a single line
    - semi-colon(;) is a statement separator
    - strongly discouraged
- https://docs.python.org/ko/3/index.html

주석 (Comments)

- 주석
  - 해시 문자(#)로 시작하고 줄의 끝까지 이어진다.
    - 주석은 줄의 처음에서 시작할 수도 있고, 공백이나 코드 뒤에 나올 수도 있다.
    - 하지만 문자열 리터럴 안에는 들어갈 수 없다. 문자열 리터럴 안에 등장하는 해시 문자는 주석이 아니라 해시 문자일 뿐이다.
    - 주석은 코드의 의미를 정확히 전달하기 위한 것이고, 파이썬이 해석하지 않는 만큼, 예를 입력할 때는 생략해도 된다.
- #
  - Line-based, i.e., but independent of indentation (but advisable to do so)
  - There are no multi-line comments Ø ‘’’ ‘’’
- #!
  - Shell-script friendly, i.e., #! as first line
  - https://en.wikipedia.org/wiki/Shebang_(Unix)
- # -- *coding: <encoding-name />* --
  - https://docs.python.org/3/reference/lexical_analysis.html#encoding-declarations

Coding Style

대부분 언어는 서로 다른 스타일로 작성될 (또는 더 간략하게, 포맷될) 수 있음. 어떤 것들은 다른 것들보다 더 읽기 쉬움. 다른 사람들이 코드를 읽기 쉽게 만드는 것은 항상 좋은 생각이고, 훌륭한 코딩 스타일을 도 입하는 것은 그렇게 하는 데 큰 도움을 줌.

- style guide
  - Offers guidance on accepted convention, and when to follow and when to break with convention
  - mostly on layout and naming
- But there is also programming guidance
- PEP 8 conformance can be checked automatically
  - E.g., http://pep8online.com/, pep8, flake8
- Pythonic style
  - 다른 언어와 다른 특색
  - 간단 명료, 풍부한 표현력
  - Encouraged style is idiomatically more functional than procedural

참고: https://www.python.org/dev/peps/pep-0008/#naming-conventions

- Camel Case
  - Second and subsequent words are capitalized, to make word boundaries easier to see. (Presumably, it struck someone at some point that the capital letters strewn throughout the variable name vaguely resemble camel humps.)
    - Example: numberOfCollegeGraduates
- Pascal Case = CapWords
  - Identical to Camel Case, except the first word is also capitalized.
    - Example: NumberOfCollegeGraduates
    - Camel Case와 혼용해서 사용하기도 함
- Snake Case
  - Words are separated by underscores.
    - Example: number_of_college_graduates

PEP8

- 들여 쓰기에 4-스페이스를 사용하고, 탭을 사용하지 않는다.
  - 4개의 스페이스는 작은 들여쓰기 (더 많은 중첩 도를 허락한다) 와 큰 들여쓰기 (읽기 쉽다) 사이의 좋은 절충이다. 탭은 혼란을 일으키고, 없애는 것이 최선이다.
- 79자를 넘지 않도록 줄 바꿈을 한다.
  - 이는 작은 화면을 가진 사용자를 돕고 큰 화면에서는 여러 코드 파일들을 나란히 볼 수 있게 한다.
- 함수, 클래스, 함수 내의 큰 코드 블록 사이에 빈 줄을 넣어 분리한다.
  - 가능하면 주석은 별도의 줄로 넣는다.
  - 독스트링(Doc-string)을 사용한다.
- 연산자들 주변과 콤마 뒤에 스페이스를 넣고, 괄호 바로 안쪽에는 스페이스를 넣지 않는다.
  - a = f(1, 2) + g(3,4).
- 클래스와 함수들에 일관성 있는 이름을 붙인다;
  - 관례는 클래스의 경우 CamelCase, 함수와 메서드의 경우 lower_case_with_underscores 로 작성한다. 첫 번째 메소드 인자의 이름으로는 항상 self를 사용한다.
- 여러분의 코드를 국제적인 환경에서 사용하려고 한다면 특별한 인코딩을 사용하지 않는다.
- 마찬가지로, 다른 언어를 사용하는 사람이 코드를 읽거나 유지할 약간의 가능성만 있더라도 식별자에 ASCII 이외의 문자를 사용하지 않는다.

Assignment (할당)

```
# Simple assignment
parrot = 'Dead'

# Assignment of single value to multiple targets (Aliasing)
x = y = z = 0

# Assignment of multiple values to multiple targets
lhs, rhs = lhs, rhs

# Augmented assignment
counter += 1

# Global assignment from within a function
world = 'Hello'
def farewell():
    global world
    world = 'Goodbye'
```

- augmented assignments are statements not expressions
- Cannot be chained and can only have a single target
- no ++ or --

Arithmetic	Bitwise
+=	&=
-=	|=

Arithmetic	Bitwise
/=	>>=
%=	<<=
//=	
**=	

변수 / 식별자

- 일반적 의미
  - 아직 알려지지 않거나 어느 정도까지만 알려져 있는 양이나 정보에 대한 상징적인 이름
    - 대수학 : 수식에 따라서 변하는 값
    - 상수 : 변하지 않는 값
- 프로그래밍에서의 변수
  - 값을 기억해 두고 필요할 때 활용할 수 있음
    - 중간 계산값을 저장하거나 누적 등
- Python
  - 값(객체)을 저장하는 메모리 상의 공간을 가르키는(object reference) 이름
    - A variable is a name for an object within a scope
    - None is a reference to nothing
  - Python은 모든 것이 객체이므로 변수보다는 식별자로 언급. 변수로 통용해서 사용하기도 함
    - 변수, 상수, 함수, 사용자 정의 타입 등에서 다른 것들과 구분하기 위해서 사용되는 변수의 이름, 상수의 이름, 함수의 이 름, 사용자 정의 타입의 이름 등 '이름'을 일반화 해서 지칭하는 용어
    - Python에는 이름있는 상수 개념 없음
  - 변수의 경우, 선언 및 할당이 동시에 이루어져야 함
    - A variable is created in the current scope on first assignment to its name
    - It's a runtime error if an unbound variable is referenced

Identifiers (식별자)

- case-sensitive names
  - E.g., functions, classes and variables
- Some identifier classes have reserved meanings

Class	Example	Meaning
_*	_load_config	Not imported by wildcard module imports — a convention to indicate privacy in general
__*__	__init__	System-defined names, such as special metho d names and system variables
__*	__cached_value	Class-private names, so typically used to name private attributes

Mutability

- 객체 (Objects)
  - 데이터(data)와 행위를 추상화한 것(abstraction)
  - Python 프로그램의 모든 데이터는 객체나 객체 간의 관계로 표현
    - (폰 노이만(Von Neumann)의 "프로그램 내장식 컴퓨터(stored program computer)" 모델을 따르며, 그 관점에서 코드 역시 객체로 표현
  - 모든 객체는 아이덴티티(identity), 형(type), 값(value)을 가짐
    - 아이덴티티
      - 한 번 만들어진 후에는 변경되지 않음
      - 메모리상에서의 객체의 주소로 생각
      - `is` 연산자는 두 객체의 아이덴티티를 비교
      - `id()` 함수는 아이덴티티를 정수로 표현한 값을 반환
    - 형
      - 객체가 지원하는 연산들을 정의 (예를 들어, "길이를 갖고 있나?")
      - 그 형의 객체들이 가질 수 있는 가능한 값들을 정의
      - `type()` 함수는 객체의 형(이것 역시 객체다)을 돌려줌
      - 아이덴티티와 마찬가지로, 객체의 형 (type) 역시 변경되지 않음
        - 어떤 제한된 조건으로, 어떤 경우에 객체의 형을 변경하는 것이 가능
- 값을 변경할 수 있는 객체들을 **가변 (mutable)**이라 함
- 만들어진 후에 값을 변경할 수 없는 객체들을 **불변 (immutable)**이라 함
  - 가변 객체에 대한 참조를 저장하고 있는 불변 컨테이너의 값은 가변 객체의 값이 변할 때 변경된다고 볼 수도 있음; 하지만 저장하고 있는 객체들의 집합이 바뀔 수 없으므로 컨테이너는 여전히 불변이라고 여겨짐. 따라서 불변성은 엄밀하게는 변경 불가능한 값을 갖는 것과는 다름.
  - 객체의 가변성(mutability)은 그것의 형에 의해 결정
  - 제자리(inplace)에서 멤버를 추가, 삭제 또는 재배치하고 특정 항목을 반환하지 않는 메서드는 컬렉션 인 스턴스 자체를 반환하지 않고 `None` 을 반환
- 형은 거의 모든 측면에서 객체가 동작하는 방법에 영향을 줌.
  - 불변형의 경우, 새 값을 만드는 연산은 실제로는 이미 존재하는 객체 중에서 같은 형과 값을 갖는 것을 돌려줄 수 있음. 반면에 가변 객체에서는 이런 것이 허용되지 않음.
    - `a = 1; b = 1` 후에, a 와 b 는 값 1을 갖는 같은 객체일 수도 있고, 아닐 수도 있음.
    - `c = []; d = []` 후에, c 와 d 는 두 개의 서로 다르고, 독립적이고, 새로 만들어진 빈 리스트임이 보장
    - `c = d = []` 는 같은 객체를 c 와 d 에 대입