LECTURE 1

Getting Started with Python

오늘의 목표

- Python 언어와 철학을 이해한다.
- Python을 설치하고 실행할 수 있다
- Python 기본 이해
  - Type
  - Control Flow(if구문, for loop 등)
  - 연산자(Operator)
- 간단한 함수를 작성할 수 있다

ABOUT PYTHON

- 1980년대에 Guido van Rossum이 개발
  - Only became popular in the last decade or so.
- Python 2.x이 현재 대부분 사용되고 있으나, 미래의 Python은 Python 3.x임
- 인터프리터(Interpreted, very-high-level programming language)
- 다양한 프로그래밍 패러다임 지원
  - OOP, functional, procedural, logic, structured, etc.
- 일반적인 목적을 위한 기본라이브러리 풍부
  - 수치 모듈, 암호 모듈, OS 연동, 네트워크 모듈, GUI 지원 등

PHILOSOPHY

From *The Zen of Python* (https://www.python.org/dev/peps/pep-0020/, 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!

PHILOSOPHY

아름다움이 추함보다 좋다.
명시가 암시보다 좋다.
단순함이 복잡함보다 좋다.
복잡함이 꼬인 것보다 좋다.
수평이 계층보다 좋다.
여유로운 것이 밀집한 것보다 좋다.
가독성은 중요하다.
특별한 경우라는 것은 규칙을 어겨야 할 정도로 특별한 것이 아니다.
허나 실용성은 순수성에 우선한다.
오류 앞에서 절대 침묵하지 말지어다.
명시적으로 오류를 감추려는 의도가 아니라면.
모호함을 앞에 두고, 이를 유추하겠다는 유혹을 버려라.
어떤 일에든 명확한 - 바람직하며 유일한 - 방법이 존재한다.
비록 그대가 우둔하여 그 방법이 처음에는 명확해 보이지 않을지라도.
지금 하는게 아예 안하는 것보다 낫다.
아예 안하는 것이 지금 *당장*보다 나을 때도 있지만.
구현 결과를 설명하기 어렵다면, 그 아이디어는 나쁘다.
구현 결과를 설명하기 쉽다면, 그 아이디어는 좋은 아이디어일 수 있다.
네임스페이스는 대박 좋은 아이디어다 -- 마구 남용해라!

NOTABLE FEATURES

- 배우기 쉽다 (Easy to learn)
- 빠른 개발 (Supports quick development)
- 다양한 플랫폼 지원 (Cross-platform)
- 오픈 소스 (Open Source)
- 확장성 (Extensible)
- 기본 라이브러리와 활동적인 커뮤니티(Large standard library and active community)
- 다양한 응용에 활용 (Useful for a wide variety of applications)

GETTING STARTED

Python 설치 필요

- (추천1) 가상머신 활용 (Ubuntu 14.04 + Python)
- (추천2) Python을 local machine에 설치 (Anaconda Python 설치 추천)
- (추천3) 온라인 환경 이용
- * kaggle.com에 등록 후 kernel 활용
- * https://colab.research.google.com 에서 구글 계정 로그인 후 이용

GETTING STARTED

- 다양한 Editor가 존재
  - Jupyter Notebook, SublimeText, vim, emacs, Notepad++, PyCharm, Eclipse, etc.

INTERPRETER

- Python 인터프리터는 파이썬 코드를 bytecode로 변환하고, Python VM을 통해서 해당 bytecode를 실행 (Java와 유사)

- 두 개의 모드를 지원(Normal모드와 Interactive 모드)
  - Normal 모드: 전체.py 파일을 인터프리터를 통해 실행
  - Interactive 모드: 명령어 하나씩 문장단위로 수행(REPL: read-eval-print loop)

INTERPRETER: NORMAL MODE

helloworld.py 프로그램 짜기

```
print "Hello, World!"
```

터미널에서:

```
$ python helloworld.py
Hello, World!
```

Note: 파이썬 2.x와 파이썬 3.x에서
print함수에 차이가 있음 (괄호 유무).
향후를 위해 3.x버전 print 구문 사용 권장

```
from __future__ import print_function

print("Hello, World!")
```

INTERPRETER: NORMAL MODE

파이썬 파일임을 나타내는 첫줄 삽입 (helloworld.py)

```
#!/usr/bin/env python
print("Hello, World!")
```

Now, from the terminal:

```
$ ./helloworld.py
Hello, World!
```

INTERPRETER: INTERACTIVE MODE

interactive 모드에서의 예

Some options:
-c : executes single command.
-O: use basic optimizations.
-d: debugging info. More
can be found here.

```
$ python
>>> print "Hello, World!"
Hello, World!
>>> hellostring = "Hello, World!"
>>> hellostring
'Hello, World!'
>>> 2*5
10
>>> 2*hellostring
'Hello, World!Hello, World!'
>>> for i in range(0,3):
...     print "Hello, World!"
...
Hello, World!
Hello, World!
Hello, World!
>>> exit()
$
```

SOME FUNDAMENTALS

- Indentation이 python에서는 매우 중요함. 다른 언어에서는 괄호({} or ())를 이용하지만, Python에서는 indentation을 이용하여 코드 블록(block)을 나눔

- 주석 처리(comment)
  - 한 줄 주석은 #.
  - 여러 줄 주석은 시작과 끝에 세번의 ‘ (single quote)를 사용

```
# here’s a comment
for i in range(0,3):
    print i
def myfunc():
    """here’s a comment about
    the myfunc function"""
    print "I'm in a function!"
```

PYTHON TYPING

- Python은 강하고(strongly), 변화가능한(dynamically) type language.
- 강한 Type 체크
  - Python이 명확히 static하게 type check를 수행하지는 않지만, type이 다른 두 연산자의 연산 금지
  - 다른 타입의 연산을 위해서는 명시적인 변환 필요
  - 예: 2 + “four” ← 계산되지 않음!
- 변화가능한 타입(Dynamic Typing)
  - 모든 type은 runtime시에 check 수행
  - 즉, 사용되기 전에 해당 변수가 어떤 타입인지 선언 불필요

NUMERIC TYPES

기본 type은 int, long, float and complex가 있음

- constructors로는 int(), long(), float(), and complex() 사용

- 모든 숫자 타입은 (complex는 제외) 일반적인 연산(예: +, -, *, /)을 지원 (a list is available here).
- 다른 숫자 타입의 수치연산과 비교를 지원함

NUMERIC TYPES

- 수(Numeric)
  - **int**: C의 long int와 동일 (2.x), 3.x에서는 unlimited
  - **float**: C에서의 double과 동일
  - **long**: 2.x에서 unlimited이고, 3.x에서는 더 이상 사용 안함
  - **complex**: 복소수

- Supported operations include constructors (i.e. int(3)), arithmetic, negation, modulus, absolute value, exponentiation, etc.

```
$ python
>>> 3 + 2
5
>>> 18 % 5
3
>>> abs(-7)
7
>>> float(9)
9.0
>>> int(5.3)
5
>>> complex(1,2)
(1+2j)
>>> 2 ** 8
256
```

SEQUENCE DATA TYPES

파이썬은 7가지의 sequence 타입이 있다.
strings, Unicode strings,
lists, tuples,
bytearrays, buffers, xrange objects.

모든 데이터 타입은 array형식을 지원함 (서로 다른 제약이 존재)

가장 일반적인 시퀀스 데이터 타입은 strings, lists, and tuples임.

xrange 데이터 타입은 enumeration-controlled loops에 많이 사용

SEQUENCE TYPES: STRINGS

스트링은 single-quote(‘) 또는 double-quote (“) 사용

스트링은 불변 (Strings are immutable)

정말 많은 built-in string 함수 지원 (listed here).

```
mystring = "Hi, I'm a string!"
```

SEQUENCE TYPES: STRINGS

Escape sequences such as '\t', '\n', etc.

스트링 앞에 'r'을 붙이면 Escape sequence와 무관하게 그대로 raw value 리턴

스트링 format 연산자는 '%'로 C와 유사. 스트링 포맷은 here 참조

두 개의 String을 같이 쓰면, 자동적으로 두 개의 string을 concate 시켜줌

```
print "\tHello,\n"
print r"\tWorld!\n"
print "Python is " "so cool."
```

```
$ python ex.py
        Hello,
\tWorld!\n
Python is so cool.
```

SEQUENCE TYPES: LISTS

Lists는 다양한 타입을 하나로 처리하는데 매우 유용

Lists는 [] 로 선언하거나, construtor를 이용하여 선언가능

Lists는 변경 가능(mutable)하고 다양한 연산을 지원 (stack ops, queue ops, sort)

Lists는 내포(nestable)가능

lists of lists of lists…

```
mylist = [42, 'apple', u'unicode apple', 5234656]
print mylist
mylist[2] = 'banana'
print mylist
mylist [3] = [['item1', 'item2'], ['item3', 'item4']]
print mylist
mylist.sort()
print mylist
print mylist.pop()
mynewlist = [x for x in range(0,5)]
print mynewlist
```

```
[42, 'apple', u'unicode apple', 5234656]
[42, 'apple', 'banana', 5234656]
[42, 'apple', 'banana', [['item1', 'item2'], ['item3', 'item4']]]
[42, [['item1', 'item2'], ['item3', 'item4']], 'apple', 'banana']
banana
[0, 1, 2, 3, 4]
```

SEQUENCE DATA TYPES

- **list:** 다양한 타입 혼용 가능
- 아이템의 변경 가능

- **tuple:** list와 유사하지만, 내용에 대한 변경 불가
  괄호 ( )로 표시되고, 각 아이템은 comma(,)로 구분

```
$ python
>>> mylist = ["spam", "eggs", "toast"] # List of strings!
>>> "eggs" in mylist
True
>>> len(mylist)
3
>>> mynewlist = ["coffee", "tea"]
>>> mylist + mynewlist
['spam', 'eggs', 'toast', 'coffee', 'tea']
>>> mytuple = tuple(mynewlist)
>>> mytuple
('coffee', 'tea')
>>> mytuple.index("tea")
1
>>> mylonglist = ['spam', 'eggs', 'toast', 'coffee', 'tea', 'banana']
>>> mylonglist[2:4]
['toast', 'coffee']
```

COMMON SEQUENCE OPERATIONS

모든 sequence 데이터 타입(string, Unicode, list, tuple, …)은 다음 연산을 지원

Operation	Result
x in s	s에 아이템 x가 있으면 True, 아니면 False
x not in s	s에 아이템 x가 없으면 True, 아니면 False
s + t	s와 t를 붙임(concatenation)
s * n, n * s	n 개의 shallow copy를 붙임(concatenation)
s[i]	s의 i번째 아이템. (index 시작은 0 부터)
s[i:j]	s의 i번째부터 j번째까지의 아이템 (Slice)
s[i:j:k]	s의 i번째부터 j번째까지 k칸씩 띄어서
len(s)	s의 길이
min(s)	s에서 가장 작은 아이템
max(s)	s에서 가장 큰 아이템
s.index(x)	s에서 x가 처음 나온 index의 위치
s.count(x)	s에서 x 아이템의 개수

COMMON SEQUENCE OPERATIONS

변경가능한(Mutable) sequence
타입에서는 다음 연산 가능

Operation	Result
s[i] = x	s의 i번째 아이템에 x를 추가
s[i:j] = t	s의 i번째 부터 j번째 까지 아이템의 값을 t로 변경
del s[i:j]	s의 i번째부터 j번째 까지 아이템을 삭제(Same as s[i:j] = [])
s[i:j:k] = t	s[i:j:k] 아이템들의 값을 t로 변경
del s[i:j:k]	s[i:j:k] 아이템을 list에서 삭제
s.append(x)	리스트 s에 x를 추가

COMMON SEQUENCE OPERATIONS

변경가능한(Mutable) sequence
타입에서는 다음 연산 가능

s.extend(x)	**Same as s[len(s):len(s)] = x.**
s.count(x)	Return number of i's for which s[i] == x.
s.index(x[, i[, j]])	Return smallest k such that s[k] == x and i <= k < j.
s.insert(i, x)	i번째에 x를 삽입 (Same as s[i:i] = [x])
s.pop([i])	i번째 아이템을 삭제하고 retur함. default는 맨 마지막 아이템
s.remove(x)	x를 리스트 s에서 삭제함(Same as del s[s.index(x)])
s.reverse()	리스트 s를 Reverse
s.sort([cmp[, key[, reverse]]])	리스트 s를 정렬

실습

https://www.programiz.com/python-programming/list

기본 BUILT-IN DATA TYPES

- Set (셋)
  - **set:** 순서가 없는(unordered) 고유한(unique) 원소의 집합
  - **frozenset:** 변경 불가능한 set

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange']
>>> fruit = set(basket)
>>> fruit
set(['orange', 'pear', 'apple'])
>>> 'orange' in fruit
True
>>> 'crabgrass' in fruit
False
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b
set(['r', 'd', 'b'])
>>> a | b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
```

기본 BUILT-IN DATA TYPES

```
$ python
>>> gradebook = dict()
>>> gradebook['Susan Student'] = 87.0
>>> gradebook
{'Susan Student': 87.0}
>>> gradebook['Peter Pupil'] = 94.0
>>> gradebook.keys()
['Peter Pupil', 'Susan Student']
>>> gradebook.values()
[94.0, 87.0]
>>> gradebook.has_key('Tina Tenderfoot')
False
>>> gradebook['Tina Tenderfoot'] = 99.9
>>> gradebook
{'Peter Pupil': 94.0, 'Susan Student': 87.0, 'Tina Tenderfoot': 99.9}
>>> gradebook['Tina Tenderfoot'] = [99.9, 95.7]
>>> gradebook
{'Peter Pupil': 94.0, 'Susan Student': 87.0, 'Tina Tenderfoot': [99.9, 95.7]}
```

- 매핑(Mapping)
  - **딕셔너리(dict)**: 해쉬 테이블(hash tables) key와 임의의 objects를 매핑

PYTHON DATA TYPES

지금까지 몇 가지 Python data types을 살펴보았음

특히, numeric, strings, lists에 익숙하여야 함

하지만, 이것만으로는 프로그램 개발시 충분하지 않다.

control flow를 살펴보자

CONTROL FLOW TOOLS

[While 구문]

```
while cond_expression:
    statements
```

조건문(cond_expression)을 만족하면, 이후 statements를 수행

* statements의 띄어쓰기(indentation) 주의

```
i = 1
while i < 4:
    print i
    i = i + 1
flag = True
while flag and i < 8:
    print flag, i
    i = i + 1
```

1
2
3
True 4
True 5
True 6
True 7

CONTROL FLOW TOOLS

[if 구문]

```
if cond_expression:
    statements
```

```
a = 1
b = 0
if a:
    print "a is true!"
if not b:
    print "b is false!"
if a and b:
    print "a and b are true!"
if a or b:
    print "a or b is true!"
```

a is true!
b is false!
a or b is true!

CONTROL FLOW TOOLS

[if else 구문]

```
if expression:
    statements
else:
    statements
```

C언어의 else if 구문을
python에서는 `elif` 사용

```
a = 1
b = 0
c = 2
if a > b:
    if a > c:
        print "a is greatest"
    else:
        print "c is greatest"
elif b > c:
    print "b is greatest"
else:
    print "c is greatest"
```

—

c is greatest

CONTROL FLOW TOOLS

[for 구문]

```
for var in sequence:
    statements
```

sequence의 아이템 각각을 var에
대입(assign)하여 statements를 수행하고,
sequence의 끝에 도달하면 loop를 빠져나옴

```
for letter in "aeiou":
    print "vowel: ", letter
for i in [1,2,3]:
    print i
for i in range(0,3):
    print i
```

---

vowel: a
vowel: e
vowel: i
vowel: o
vowel: u
1
2
3
0
1
2

CONTROL FLOW TOOLS

C의 for 구문에서와 다르게 python에서는 loop를 위해 sequence가 필요. 그래서, 일반적인 integer의 sequence 생성 함수 필요. range()와 xrange().

둘다 integer의 sequence를 생성하지만, range()는 list를, xrange()는 xrange object를 생성

range()는 고정된 리스트를 우선 생성하지만, xrange()의 경우에는 필요시에만 아이템을 생성함.

그래서, 아주 큰 loop에서는 (예: 1조의 값) 반드시 xrange()를 사용하여야 함

```
for i in xrange(0, 4):
    print i
for i in range(0,8,2):
    print i
for i in range(20,14,-2):
    print i
```

```
0
1
2
3
0
2
4
6
20
18
16
```

CONTROL FLOW TOOLS

[loop구조와 연관된 키워드]
break, continue, pass, and else.

- break – 현재 loop을 빠져나옴
- continue – loop의 시작으로 감
- else – loop이 끝나고 반드시 수행

```
for numin  range(10,20):
    if num%2 == 0:
        continue
    for i in range(3,num):
        if num%i == 0:
            break
    else:
        print num, 'is a prime number'
```

```
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
```

PYTHON PROGRAM 예제

참조: Project Euler

피보나치(Fibonacci sequence) 수열은 이전의 두 값을 더해서 새로운 값을 생성한다. 처음 1과 2로 시작해서 처음 10개의 수열은 다음과 같다:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

400만이 넘지 않은 피보나치 수열 중 짝수 원소의 합을 구하는 프로그램을 작성하시오

A SOLUTION USING BASIC PYTHON

```
from __future__ import print_function

total = 0
f1, f2 = 1, 2
while f1 < 4000000:
    iff1 % 2==0:
        total = total + f1
    f1, f2 = f2, f1 + f2
print(total)
```

Python 3.x version의 print 사용

Python은 다수의 다중
리턴 가능
왼쪽 변수가 설정되기 전에
오른쪽의 값들은 fully evaluated

Output: 4613732

FUNCTIONS

[함수 선언]

```
def function_name(args):
    statements
```

def 키워드를 이용하여 선언
arguments를 받을 수 있음

reture을 통해 값을 리턴함(list 리턴 가능)

```
# Defining the function
def print_greeting():
    print "Hello!"
    print "How are you today?"

print_greeting() # Calling the function
```

Hello!
How are you today?

FUNCTIONS

함수에 파라미터 전달 방식은 call by reference임

당연하게도 변경가능한(mutable objects)만 함수에서 변경 가능

* string은 unmutable object임.

```
def hello_func(name, somelist):
    print "Hello,", name, "!\n"
    name = "Caitlin"
    mylist[0] = 3
    return 1, 2
myname = "Ben"
mylist = [1,2]
a,b = hello_func(myname, mylist)
print myname, mylist
print a, b
```

Hello, Ben !

Ben [3, 2]
1 2

FUNCTIONS

다음 코드의 출력값은?

```
def hello_func(names):
    for n in names:
        print "Hello, ", n, "!"
    names[0] = 'Susie’
    names[1] = 'Pete’
    names[2] = 'Will’
names = ['Susan', 'Peter', 'William']
hello_func(names)
print "The names are now ", names, ".\n"
```

FUNCTIONS

다음 코드의 출력값은?

```
def hello_func(names):
    for n in names:
        print "Hello,", n, "!"
    names[0] = 'Susie'
    names[1] = 'Pete'
    names[2] = 'Will'
names = ['Susan', 'Peter', 'William']
hello_func(names)
print "The names are now", names, "."
```

Hello, Susan !
Hello, Peter !
Hello, William !
The names are now [‘Susie’, ‘Pete’, ‘Will’] .

A SOLUTION WITH FUNCTIONS

Python 인터프리터는 특별한 환경 변수 설정이 가능함.

특정 파일의 모듈을 main함수로 실행하고 싶다면, 특정 __name__ variable을 이용하여
그 값을 " __main__ " 으로 설정하면 됨

```
from __future__ import print_function

def even_fib():
    total = 0
    f1, f2 = 1, 2
    while f1 < 4000000:
        if f1 % 2 == 0:
            total=total + f1
        f1, f2 = f2,f1  + f2
    return total

if __name__ == "__main__":
    print(even_fib())
```

INPUT

- raw_input()
  - 사용자 입력을 요구할 경우에 사용하며, string형식으로 입력받아 리턴
  - 만약 argument를 따로 주는 경우는, 프롬프트(prompt)로 사용됨
- input()
  - raw_input() 은 string data를 받을 때 활용하고, python expression 자체를 입력 받는 경우에는 input()을 사용
  - 주어진 expression의 값을 Return함.
  - Dangerous – don't use it : 사용하지 말것을 권장

```
>>> print(raw_input('What is your name? '))
What is your name? Caitlin
Caitlin
>>> print(input('Do some math: '))
Do some math: 2+2*5
12
```

Note: Python 3.x에서는 input()과 raw_input()는 동일하게 동작

A SOLUTION WITH INPUT

```
from __future__ import print_function

def even_fib(n):
    total = 0
    f1, f2 = 1, 2
    while f1 < n:
        if f1 % 2 == 0:
            total = total + f1
        f1, f2 = f2,f1  + f2
    return total

if __name__ == "__main__":
    limit = raw_input("Enter the max Fibonacci number: ")
    print(even_fib(int(limit)))
```

```
Enter the max Fibonacci number: 4000000
4613732
```

CODING STYLE

코딩 스타일 가이드 : PEP 8 here 참조

pylint (Python source code analyzer)를 통해 개인들의 좋은 코딩 스타일에 도움을 줌