



# Python Programming for Science and Machine Learning

서울대학교 자연과학대학 물리천문학부

2018 가을학기

전산물리 (884.310) & 응용전산물리 (3342.618)

# 2 주차 강의노트 내용

## Python

- Introduction to Python
- Python data types & containers
- Python conditional statements

## 공지

- Exercise: 01 차 실습과제 ( 예습 ) [9/10( 월 ) - 9/14( 금 ) 24:00, 제출]
- Homework: 01 차 정규과제 [9/14( 금 ) - 9/21( 금 ) 24:00, 제출]
- 2 주차 (~9/13) 까지 연습반 확정하여 연습반 조교에게 개인별 과제 업로드 링크를 받아 실습과제 제출하기 (1 차 강의노트 참조)

# Python

- 귀도 반 로섬 (Guido van Rossum) 이 개발 (1991 년 발표 )
- 인터프리터식 다목적 고급 프로그래밍 언어 :
  - 개발자의 생산성과 코드의 가독성을 높이는 데 중점을 둔 프로그래밍 언어
  - 다양한 프로그래밍 패러다임 - 객체 지향 (class), 명령형 (def), 함수형 (lambda) 등의 프로그래밍을 지원
  - 자료형 (type) 을 동적으로 결정 : C/C++ 과 같은 자료형의 선언이 필요없음 , 실행 시간에 자료형을 검사 . 자동 메모리 관리
  - 인터프리터 (interpreter) 식 언어 : 인터프리터를 사용하여 소스코드를 한 줄씩 바로 실행하는 프로그래밍 언어 . 소스 코드를 기계어로 번역하는 컴파일러와 대비 .
- 방대한 표준 라이브러리와 패키지들이 존재
- 오픈소스
- Guido van Rossum wrote on the origin of Python :  
*...In December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).*



# Python 의 장점 : 효율성 , 유연성 , 그리고 다양성

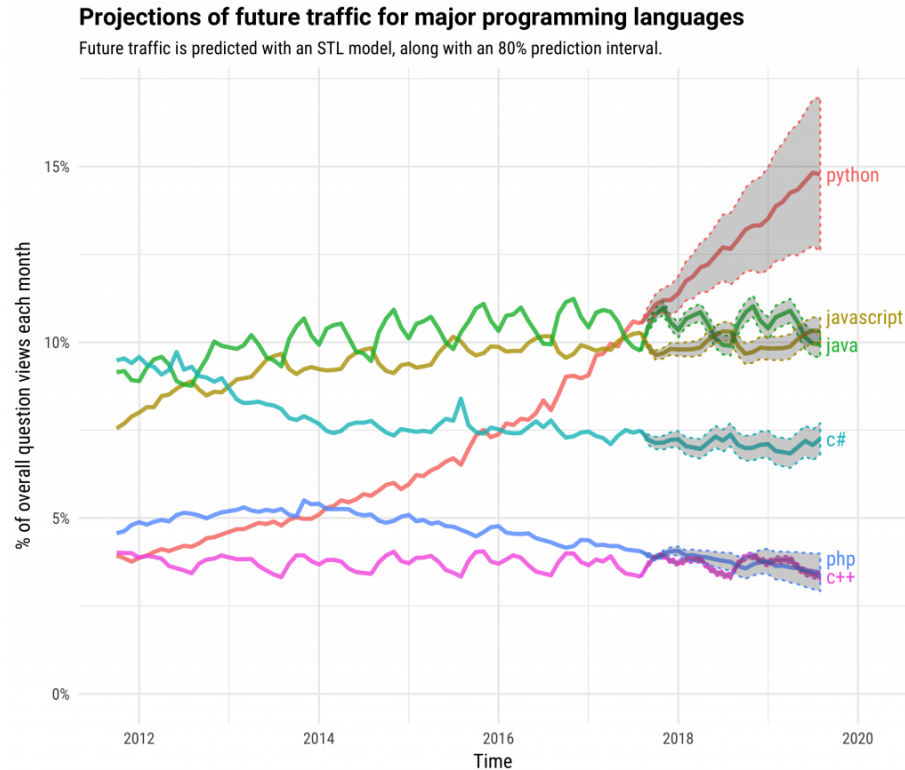
- 개발자의 생산성과 코드의 가독성을 높이는 데 중점을 둔 프로그래밍 언어

*“Life is too short, You need Python.”*

- 다른 언어들이 수많은 방법으로 하나의 기능을 구현할 수 있다면 , 파이썬은 가장 좋은 방법 한 가지만 활용하는 것을 선호
  - 프로그래밍 본래의 목적에 집중할 수 있도록 도와줌 .
  - 아이디어의 빠른 구현 및 테스트에 용이
- 유연성 : 자료형을 미리 선언할 필요가 없음 . 자동 메모리 관리 .
- 우수한 가독성 (Good readability):
  - **들여쓰기 : 줄의 시작을 맞추지 않으면 실행되지 않는다!**
- 방대한 라이브러리 : 파이썬 표준 라이브러리와 약 13 만개의 패키지 저장소

# 가파르게 수요가 늘고 있는 파이썬

- 일반적으로 웹이나 데스크톱 개발자 및 시스템 관리자들이 많이 사용해왔지만, 최근에는 데이터 과학자와 머신러닝 엔지니어를 포함한 공학과 과학 분야에도 점점 더 그 쓰임새를 넓혀가고 있다.

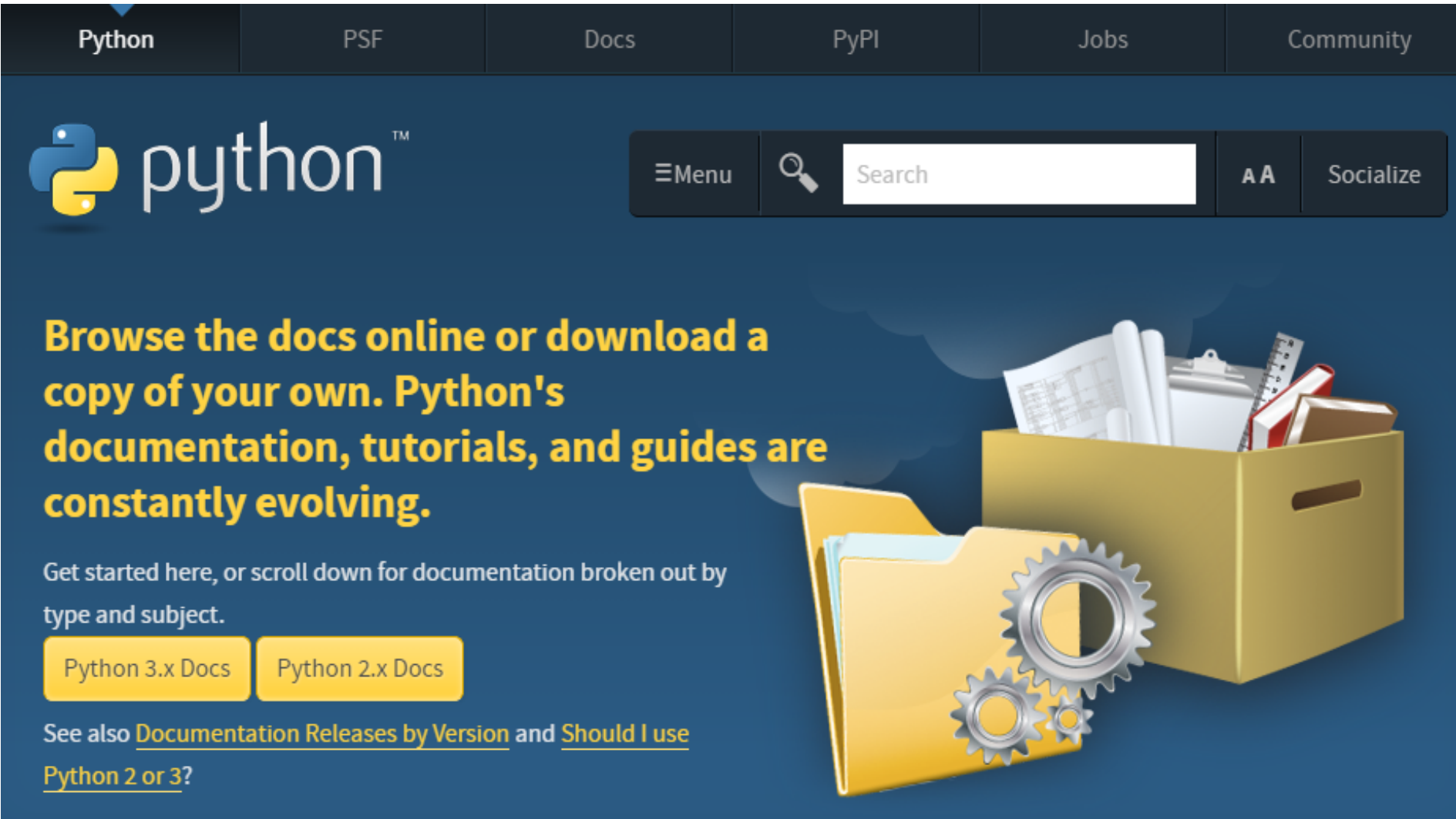


<https://stackoverflow.blog/2017/09/06/incredible-growth-python/>

- 파이썬으로 할 수 있는 수많은 작업들:
  - awesome-python (<https://github.com/vinta/awesome-python>):

# Python Software Foundation

- 공식 웹사이트 (<https://python.org>)
  - python2.7 공식 메뉴얼 (<https://docs.python.org/2.7>)
  - python3.6 공식 매뉴얼 (<https://docs.python.org/3.6>)



The image shows the Python Software Foundation website homepage. At the top, there is a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this is the Python logo and a search bar with a magnifying glass icon. To the right of the search bar are links for 'AA' and 'Socialize'. The main content area features a large yellow text block that reads: 'Browse the docs online or download a copy of your own. Python's documentation, tutorials, and guides are constantly evolving.' Below this text, there are two yellow buttons: 'Python 3.x Docs' and 'Python 2.x Docs'. At the bottom, there is a link to 'See also Documentation Releases by Version and Should I use Python 2 or 3?'. On the right side of the page, there is a 3D illustration of a yellow folder and a yellow box filled with papers, with several interlocking gears in the foreground.

Python PSF Docs PyPI Jobs Community

python™

Menu Search AA Socialize

**Browse the docs online or download a copy of your own. Python's documentation, tutorials, and guides are constantly evolving.**

Get started here, or scroll down for documentation broken out by type and subject.

Python 3.x Docs Python 2.x Docs

See also [Documentation Releases by Version](#) and [Should I use Python 2 or 3?](#)

# Python2 vs 3

- Python2 가 아직 더 많이 쓰임 .
- 하지만 2 는 더 이상 큰 개선이 진행되지 않으며 , 2 로 작성한 코드는 3 에서 진행되지 않는 경우도 종종 생긴다 .
- 파이썬에 입문하는 경우 파이썬 3 최신 버전의 사용을 권장 . 2 로 작성된 코드는 그대로 안정적으로 사용하되 천천히 업그레이드 .
- 2 와 3 코드 변환 (2to3, <https://docs.python.org/2/library/2to3.html>)

Here is a sample Python 2.x source file, `example.py`:

```
def greet(name):  
    print "Hello, {0}!".format(name)  
print "What's your name?"  
name = raw_input()  
greet(name)
```

\$ 2to3 example.py

```
def greet(name):  
    print("Hello, {0}!".format(name))  
print("What's your name?")  
name = input()  
greet(name)
```

# 유용한 Python 관련 사이트

- 파이썬 재단 페이지 (<https://python.org>)
- 파이썬 공식 매뉴얼 (<https://docs.python.org/3.6>)
- 파이썬 표준 라이브러리 (<https://docs.python.org/3/library/index.html>)
- 파이썬 패키지 인덱스 (<https://pypi.python.org/pypi>)
- 공학과 과학 패키지 (SciPy ecosystem : <https://www.scipy.org>)
  - NumPy (<http://www.numpy.org/>)
  - Matplotlib (<https://matplotlib.org/>)
  - SymPy (<http://www.sympy.org/>)
  - pandas (<https://pandas.pydata.org/>)
  - scikit-learn (<http://scikit-learn.org/stable/>)
  - IPython (<https://ipython.org/>) & Jupyter (<https://jupyter.org/>)
  - Cython (<http://cython.org/>)
- 파이썬 튜토리얼 :
  - 점프 투 파이썬 (<https://wikidocs.net/book/1>)
  - 튜토리얼포인트 (<https://www.tutorialspoint.com/python/index.htm>)



# Python 환경 설정

- Version check : `$ python --version`
- Which python? : `$ which python`
- 파이썬 환경변수 설정 (`~/.bashrc`)
- [https://www.tutorialspoint.com/python/python\\_environment.htm](https://www.tutorialspoint.com/python/python_environment.htm)

# 파이썬 프로그램 ( 스크립트 ) 을 이루는 주요 구성요소

## 1. 인코딩 선언 :

파이썬 3 에서의 문자열은 , 기본적으로 유니코드 (character set) 하의 문자열로 인식되어 내부적으로 처리되며 , 이 유니코드 문자열을 파일로 저장하거나 전송할때 , 바이트 문자열로의 인코딩이 필요하다 . 프로그램의 서두 ( 혹은 입출력 함수의 인자로써 ) 에 인코딩을 명시할 수 있으며 , 명시가 없을시에는 UTF-8 이 기본 인코딩으로 사용된다 .

## 2. 주석 :

파이썬에서 주석은 한 줄짜리는 줄 앞에 # 를 붙여서 정의 . 여러 줄일경우 ( 곱따옴표 3 개 ) """ 혹은 ( 홑따옴표 3 개 ) ''' 로 주석의 내용을 앞뒤로 감싸면 된다 .

## 3. 필요한 라이브러리 등의 import

## 4. 함수 / 클래스 정의 (def/class)

- 들여쓰기 : 일반적으로 공백문자 4 개를 사용한다 .
- ltab 간격 정의 : (Linux / macOS) find & edit ~/.vimrc & add “set tabstop=4”
- 다음줄에 이어서 쓸 때 : ‘\’ ( 역슬래시 ) 를 사용

## 5. 이후 실행 스크립트를 나열

`if __name__ == '__main__':` ‘ 이 파일이 메인 실행파일로 호출되었을 때 실행하라 ’ 는 뜻

# 파이썬 프로그램 ( 스크립트 ) 작성 규칙 예 (\*)

‘main.py’ (\$vi main.py => save & exit (esc, :, wq, enter) )

```
1 1 # -*- coding: utf-8 -*-
2
3 """
4 Add comments here
5 """
6
7 # import of math module
8 import math
9
10 # definition of a function/method
11 def func(x):
12     """
13     Add comments of this function here. 1-tab indentation.
14     """
15
16     # 1-tab indent scripting
17     y = math.cos(x)**2 + math.sin(x)**2
18
19     return y
20
21 # Meaning of the "if __name__ == '__main__':" statement
22 # = RUN the scripts below the "if __name__ == '__main__':" statement,
23 # IF this script (main.py) is executed as a main program
24 # (simply when we run it with '$python3 main.py' in the command prompt).
25
26 if __name__ == '__main__':
27
28     x = 1.23456
29     print ("cos(%f)^2 + sin(%f)^2 = %f"%(x,x,func(x)))
30
31
```

\$python3 main.py

# First “Hello,World!” of Python

- Using interactive interpreter prompt (interactive Python shell)

```
wscho@ubuntu:~/PPSML $ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello, World!")
Hello, World!
>>> exit()
```

- Using direct interpreter in command prompt

```
wscho@ubuntu:~/PPSML $ vi hello_world.py
wscho@ubuntu:~/PPSML $ cat hello_world.py

print ("Hello, World!")

wscho@ubuntu:~/PPSML $ python3 hello_world.py
Hello, World!
```

# 파이썬 데이터 자료형

## Data Types of Python

# 내장 데이터 타입 (Built-in data types)

data type	definition
정수 (int)	정수값 (integer) 을 저장하는 데이터 타입 . Python2 에서는 (int,long), Python3 에서는 (int) 만 존재
부동 소수 (float)	double-precision 의 부동 소수 타입 .
복소수 (complex)	각각이 float 타입인 실수부와 허수부를 갖는 복소수
부울 (Boolean)	True or False 만을 값으로 갖는 부울
문자열 (string)	문자열 ( 불변형 )
리스트 (list)	임의의 데이터 타입을 구성 요소로 갖는 , 순서가 있는 데이터 집합 ( 가변형 )
튜플 (tuple)	임의의 데이터 타입을 구성 요소로 갖는 , 순서가 있는 데이터 집합 ( 불변형 )
바이트 (bytes)	일련의 바이트를 나타내는 데이터 타입 ( 불변형 )
바이트 배열 (bytearray)	일련의 바이트를 나타내는 데이터 타입 ( 가변형 )
집합 (set)	구성 요소간에 순서가 없는 데이터의 집합 ( 가변형 )
딕셔너리 (dictionary)	구성 요소간에 순서가 없는 데이터 키 (key) & 값 (value) 쌍의 집합

# 변수의 자료형 & type 함수 (\*)

- 파이썬은 자료형이 동적인 언어로서, 변수 (variable) 는 굳이 자료형을 선언할 필요가 없지만, 값 자체는 자료형이 있으며, 변수는 생명 주기 동안 가지는 값의 자료형을 바꿀 수도 있다.
- 변수가 가진 값의 자료형을 알아보는 방법 : type 함수 이용

```
>>> a=3.14
>>> b=4
>>> c=1+2j
>>> d=complex(a,b)
>>> e='cat'
>>> print ("type of a, b, c, d, e : \n %s\n %s\n %s\n %s\n %s"%(type(a)
,type(b),type(c),type(d),type(e)))
type of a, b, c, d, e :
<class 'float'>
<class 'int'>
<class 'complex'>
<class 'complex'>
<class 'str'>
```

# 변수명 짓기

- 변수명은 문자, 숫자, 밑줄을 포함하여 지을 수 있으며, 다만 숫자는 처음에 나올 수 없다.
- 파이썬의 변수명은 대소문자를 구분한다.
- 다음의 예약어는 변수명으로 사용할 수 없다.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield



# 수치 자료형 (Numerical Types) – 1. 정수형 (\*)

## int: 정수형

### 1. 정수앞에

- '0b' 를 붙이면 2 진수 ,
- '0o' 을 붙이면 8 진수 ,
- '0x' 를 붙이면 16 진수로 인식

```
>>> 0b100
4
>>> 0o100
64
>>> 0x100
256
```

### 2. 10 진수 (x) 입력을 받아서 원하는 진수로 변환하는 함수 ( 출력은 문자열형 (string))

- bin(x): x 를 2 진수로
- oct(x): x 를 8 진수로
- hex(x): x 를 16 진수로

```
>>> bin(4)
'0b100'
>>> oct(64)
'0100'
>>> hex(256)
'0x100'
```

### 3. Python3 에는 정수 (int) 의 범위를 넘어서는 Python2 의 long type 이 모두 int 로 처리된다.

# 수치 자료형 (Numerical Types) – 2. 실수형 (\*)

## float : ( 부동소숫점 ) 실수형

- 부동소숫점 숫자는 ( 내부적으로 ) 다음과 같이 나타내어진다
  - $x = \pm m2^e$
  - $x$  는 숫자,  $m$  은 가수,  $e$  는 지수
- -1022 보다 작거나 1023 보다 큰 지수는 표현할 수 없다
- 언더플로우 (underflow) : 산술 연산 결과가 표현 한계, 약  $2^{-1022} \sim 10^{-308}$  보다 작을때
  - denormal number 를 사용하면 조금더 작은 수도 가능하다 ( 약  $2^{-1074}$  )
- 오버플로우 (overflow) : 산술 연산 결과가  $2^{1023} \sim 10^{308}$  보다 클때

```
>>> a = 2.0**1023
>>> a
8.98846567431158e+307
>>> a = 2.0**1024
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result out of range')
>>> b = 2.**(-1023)
>>> b
1.1125369292536007e-308
>>> b = 2.**(-1075)
>>> b
0.0
```

# 수치 자료형 (Numerical Types) – 3. 복소수형 (\*)

## complex: 복소수형

- 파이썬은 기본적으로 복소수를 지원한다. 허수부분은 문자 j로 나타내며, 실수와 허수부분은 float 형으로 저장된다.

```
>>> c = 1+2j
>>> print c
(1+2j)
>>> print c.real
1.0
>>> print c.imag
2.0
>>> print abs(c)
2.2360679775
```

- 복소수의 일반적인 산술 연산도 지원된다 (cmath 모듈 사용)

```
>>> import cmath
>>> d=complex(0.0,-cmath.pi)
>>> d
-3.141592653589793j
>>> cmath.exp(d)
(-1-1.2246467991473532e-16j)
```

# 수치 자료형의 연산

Operation	Result	Full documentation
$x + y$	sum of $x$ and $y$	
$x - y$	difference of $x$ and $y$	
$x * y$	product of $x$ and $y$	
$x / y$	quotient of $x$ and $y$ ( 나눈 몫 )	
$x // y$	floored quotient of $x$ and $y$ ( 나눈 몫의 정수값 )	
$x \% y$	remainder of $x / y$ ( 나머지 )	
$-x$	$x$ negated	
$+x$	$x$ unchanged	
<code>abs(x)</code>	absolute value or magnitude of $x$	<a href="#"><code>abs()</code></a>
<code>int(x)</code>	$x$ converted to integer	<a href="#"><code>int()</code></a>
<code>float(x)</code>	$x$ converted to floating point	<a href="#"><code>float()</code></a>
<code>complex(re,im)</code>	a complex number with real part $re$ , imaginary part $im$ . $im$ defaults to zero.	<a href="#"><code>complex()</code></a>
<code>c.conjugate()</code>	conjugate of the complex number $c$	
<code>divmod(x, y)</code>	the pair $(x // y, x \% y)$	<a href="#"><code>divmod()</code></a>
<code>pow(x, y)</code>	$x$ to the power $y$	<a href="#"><code>pow()</code></a>
$x ** y$	$x$ to the power $y$	

# 문자열 (string) (\*)

- 파이썬은 ASCII 문자열과 유니코드 문자열을 지원한다.
- 파이썬 3 는 모두 유니코드로 처리

## Python3

- 유니코드문자열은 ‘...’ 나 “...”처럼 좌우를 감싸고, 세쌍의 앞뒤 큰따옴표 """...""" 는 여러줄의 문자열을 감싼다.
- (Python2) 유니코드 문자열은 u 로 시작 ( a = u'this is ... ' )
- 인코딩을 사용하면 유니코드 문자열을 바이트 문자열로 변환할 수 있다 ( 예를 들면 utf8 encoding 을 사용 )

```
>>> a = 'this is an unicode string in Python3'
>>> a
'this is an unicode string in Python3'
>>> type(a)
<class 'str'>
>>> isinstance(a, str)
True
>>> isinstance(a, bytes)
False
```

→ 됨

```
>>> b = a.encode('utf-8')
>>> b
b'this is an unicode string in Python3'
>>> type(b)
<class 'bytes'>
>>> isinstance(b, str)
False
>>> isinstance(b, bytes)
True
```

# 문자열 (string) (\*)

- 파이썬 문자열의 특징은 리스트와 비슷하게 반복문을 사용할 수 있는 객체이며, 연산자도 지원한다.

```
>>> for i in 'Python':  
...     print i  
...  
P  
y  
t  
h  
o  
n
```

```
>>> a = 'Python'  
>>> a[0]  
'P'  
>>> a[5]  
'n'  
>>> b = ' is fun ! '  
>>> c = a + b  
>>> c  
'Python is fun ! '  
>>> 3*c  
'Python is fun ! Python is fun ! Python is fun ! '
```

- 다양한 방법으로 변수를 문자열에 넣을 수 있다 (str(): 문자열로 형변환)

```
>>> print 'my number is ' + str(3)  
my number is 3  
>>> print 'my number is %s' % (3)  
my number is 3  
>>> print 'my number is %(number)s' % dict(number=3)  
my number is 3
```

- 문자열 slicing 을 해보자 a[0:1], a[1:4], a[:2], a[-2:] ( 문자열 / 리스트 인덱스 익히기 )

# 리스트 (list) (\*)

파이썬의 리스트는 값의 나열 (array) 이다. 순서가 존재하며, 여러 타입의 값을 담을 수 있다. 문자열과 마찬가지로 0 부터 시작하는 인덱스가 있으며, 역시 슬라이싱도 가능하다.

```
>>> animals = ['cat', 'dog', 'tiger']
>>> animals
['cat', 'dog', 'tiger']
>>> type(animals)
<type 'list'>
```

기존 리스트에 값을 추가할 때

- append() 메소드 : 맨 뒤에 추가 ( 추가되는 컨테이너를 본 리스트의 원소로 추가 )
- extend() : 맨 뒤에 추가 ( 본 리스트의 연장 )
- insert() : 원하는 위치에
- + 연산자 이용 : 맨 뒤에

```
>>> animals = ['cat', 'dog', 'tiger']
>>> animals
['cat', 'dog', 'tiger']
>>> type(animals)
<type 'list'>
>>> animals.append('eagle')
>>> animals
['cat', 'dog', 'tiger', 'eagle']
>>> animals.insert(1,'frog')
>>> animals
['cat', 'frog', 'dog', 'tiger', 'eagle']
>>> animals.extend(['crow','hornet'])
>>> animals
['cat', 'frog', 'dog', 'tiger', 'eagle', 'crow', 'hornet']
>>> animals += ['bee']
>>> animals
['cat', 'frog', 'dog', 'tiger', 'eagle', 'crow', 'hornet', 'bee']
```

# 리스트 (list) (\*)

- 값의 위치 (index) 반환 : index()
- 리스트 값들의 개수 출력 : count()
- 원하는 위치의 값 출력 & 사라짐 : pop() ( 인자 없을 시 맨 뒤의 값 )
- 해당 값의 제거 : remove() ( 앞에서부터 )
- 정렬 : sort() & reverse()

```
>>> animals += ['cat']
>>> animals.index('cat')
0
>>> animals.index('cat',1)
8
>>> animals.count('cat')
2
```

```
>>> animals.pop()
'cat'
>>> animals.pop()
'bee'
>>> animals.pop(1)
'frog'
>>> animals
['cat', 'dog', 'tiger', 'eagle', 'crow', 'hornet']
```

```
>>> animals.remove('cat')
>>> animals
['dog', 'tiger', 'eagle', 'crow', 'hornet']
>>> animals.sort()
>>> animals
['crow', 'dog', 'eagle', 'hornet', 'tiger']
>>> animals.reverse()
>>> animals
['tiger', 'hornet', 'eagle', 'dog', 'crow']
```



# 리스트 (list) & 튜플 (tuple) (\*)

- 리스트 ( 리스트 내장 , list com prehension → 3 주차 ) 의 활용

```
>>> a = [1,2,3,4,5]
>>> b = [x * 3 for x in a if x % 2 == 0]
>>> print b
[6, 12]
```

- array 와 리스트의 차이점은 , array 의 경우 배열의 모든 항목이 같은 자료형이어야 한다는 것이다 . ( 대신 속도가 빠름 )
- 튜플 ( 소괄호 '(' ')' 안에 나열 ) 은 리스트와 유사하지만 , 크기와 항목값을 변경할 수 없다 .

```
>>> a = [1,2,3,4,5]
>>> a[0] = 100
>>> a
[100, 2, 3, 4, 5]
```

```
>>> tup = (1,2,3,4,5)
>>> tup[0] = 100
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- 리스트값의 순환

```
>>> a = [True, 'dog', 10]
>>> for i in a:
...     print(i, type(i))
...
True <class 'bool'>
dog <class 'str'>
10 <class 'int'>
```

# 사전 (Dictionary) (\*)

- 파이썬에서 사전은 해시 테이블로서, 키 객체와 연결된 값 객체가 쌍으로 저장된 테이블이다. 순차적이지 않고, 키를 통해 검색되고 저장된다.

```
>>> a = {'k1':1, 'k2':2}
>>> print a['k1']
1
>>> print a['k2']
2
>>> 'k1' in a
True
>>> 'k2' in a
True
>>> 'k3' in a
False
```

- has\_key(), keys(), values(), items(), update() ...
- del 을 사용하여 요소 제거 .
- 사전 키와 값의 순환 (→ 3 주차 'loops')

```
>>> d = {"orange":10 , "apple":20 , "melon":30}
>>> for key, value in d.items():
...     print (key,value)
...
orange 10
apple 20
melon 30
```

파이썬 조건문

Conditional Statements of Python

# 조건문의 기본 형식 : if .. elif .. else (\*)

- 조건에 따라서 여러가지 명령을 수행한다. 'if' 로 시작하며 '(' 안에 조건식을 넣고 ':' 으로 마무리하여, 이 조건식이 만족될 때의 명령이 그 다음줄에 명시될 것을 알린다. **다음줄의 명령구문은 하위 개념임으로 들여쓰기로 시작한다.**
- 'if' 이후의 다른 조건은 'elif (조건식):' 의 형식으로, 그 하위 명령을 똑같은 방식으로 들여쓰기로 작성. 여기서 'elif' 는 'else if' 의 뜻이다.
- 'if' 이후의 다른 조건의 경우, 구체적인 조건식이 필요없다면, ( 즉, 명시된 if와 elif 들의 조건에 해당하지 않는 다른 모든 경우에 해당하는 명령을 작성하려면 ), 조건식없이 'else:' 로 시작하여 그 밑에 들여쓰기로 명령을 작성한다.

## <Bool 변수로 조건문 작성 >

```
hungry = True # Bool type
sleepy = False # Bool type

if ( hungry == True ) :
    print ( " I am hungry " )
elif ( hungry == False ) :
    print ( " I am not hungry " )

if ( sleepy == True ) :
    print ( " I am sleepy " )
else:
    print ( " I am not sleepy " )
```

## <Bool 변수로 조합된 논리, 비트연산 조건문 작성 >

```
# mixed (Boolean and bitwise operators)
# (not a&b...) = not (a&b...)
# (not a&b... and/or c) = (not (a&b..)) and/or (c)
# (~ a&b...) = (~a)&b...
# => '~' is a unary (bitwise) operator (단항연산자)
hungry = True
sleepy = False

if ( hungry and sleepy ):
    print ( " Then, have a meal and take a nap." )
elif ( (not hungry) & sleepy ):
    print ( " Then, take a nap." )
elif ( hungry & ~sleepy ):
    print ( " Then, have a meal." )
else:
    print ( " Then, let's study." )
```

# 조건문의 기본 형식 : if .. elif .. else (\*)

<Bool 변수가 조합된 논리연산 조건문 작성 >

```
# using Boolean operators
if ( hungry and sleepy ):
    print (" Then, have a meal and take a nap.")
elif ( (not hungry) and sleepy ):
    print (" Then, take a nap.")
elif ( hungry and (not sleepy) ):
    print (" Then, have a meal.")
else:
    print (" Then, let's study.")
```

<Bool 변수가 조합된 비트연산 (bitwise operation) 조건문 작성 >

```
# using bitwise operators
if ( hungry & sleepy ):
    print (" Then, have a meal and take a nap.")
elif ( ~hungry & sleepy ):
    print (" Then, take a nap.")
elif ( hungry & ~sleepy ):
    print (" Then, have a meal.")
else:
    print (" Then, let's study.")
```

# 조건문을 위한 조건식 만들기 1 (= Bool 타입의 값 만들기) (\*)

- 조건식의 참 거짓 판단 : 자료형의 bool 판단값과 같다, True 는 참, False 는 거짓을 나타냄.
  - bool(x) : x 는 True or False 를 판별하고 싶은 입력자료형 / 조건식
- 또한 정수 0, 실수 0.0, 시퀀스 계열의 (), [], {}, 빈문자열 "", 아무것도 없음을 나타내는 None 은 언제나 False 로 판단됨 . 반면이 그 이외의 값이 할당된 변수의 경우 항상 참으로 판별됨

```
# Bool 타입의 참 거짓
>>>bool(True)
True
>>>bool(False)
False

# 숫자 계열의 참 거짓
>>>bool(12)
True
>>>bool(0)
False
>>>bool(12.0)
True
>>>bool(0.0)
False
```

```
# 문자열의 참 거짓
>>>bool('cat')
True
>>>bool('')
False

# 시퀀스 계열의 참 거짓
>>>bool(['cat',1,2])
True
>>>bool(())
False
>>>bool({})
False

# None타입의 참 거짓
>>>bool(None)
False
```

# 조건식 만들기 2 (\*)

## Bool 변수로 논리연산 (and, or, xor) 조건문 작성 예

- 'and' ( 혹은 ' & ' 로도 표현가능 ) 는 두 Bool 변수 모두가 참이면 참 : True if (T,T),
- 'or' ( '|' 로도 표현 ) 은 둘 중 하나 이상이 참이면 참 : True if (T,T) or (T,F) or (F,T)
- 'xor, exclusive or' ( '^ ' 로 표현 ) 은 두 Bool 변수가 다를때 참 : True if (T,F) or (F,T)
- 논리적 조건들의 판단의 순서 : 'and', 'or' 인 경우 반드시! 왼쪽부터 오른쪽으로 평가가 이루어지나 ' & ', '|' 의 경우에는 다 평가를 한 후 진행한다.

'&' 를 쓸 경우 : 10/a 까지 계산하여 에러

```
a = 0.  
  
if a & 10./a:  
    print("10/a = %s"%(10./a))  
else:  
    print(" division is not defined.")
```

```
Traceback (most recent call last):  
  File "if_short_circuit_eval.py", line 4, in <module>  
    if a & 10/a:  
ZeroDivisionError: division by zero
```

'and' 를 쓸 경우 : a 까지만 계산하여 에러없음

```
a = 0.  
  
if a and 10./a:  
    print("10/a = %s"%(10./a))  
else:  
    print(" division is not defined.")  
  
division is not defined.
```

```
hungry = True # Bool type  
sleepy = False # Bool type  
  
# logical bool operation (and) : (hungry & sleepy)  
my_condition_and = (hungry and sleepy)  
# logical bool operation (or) : (hungry | sleepy)  
my_condition_or = (hungry or sleepy)  
# logical bool operation (xor, exclusive or) : (hungry ^ sleepy)  
my_condition_xor = (hungry ^ sleepy)  
  
if my_condition_and:  
    print (" Then, have a meal and take a nap")  
else:  
    print (" Then, choose one : napping, eating, studying")  
  
if my_condition_or:  
    print (" Then, have a meal OR take a nap ")  
else:  
    print (" Then, let's study ")  
  
if my_condition_xor:  
    print (" Then, have a meal OR take a nap ")  
else:  
    print (" Then, choose one : eating&napping, studying")
```

# 조건식 만들기 3

- 조건에 따른 Bool 변수값 만들기 연습

예) 현재 가진돈 (money), 커피값 (coffee), 그리고 졸린 여부 (sleepy=True/False) 을 입력받아 조건문 스크립트를 만들어보자

```
# 졸린지 물어보고 Bool변수 조건식 (sleepy_b) 정의
sleepy_input = input( '졸리니?(True/False): ' )
if sleepy_input == "True":
    sleepy_b = True
elif sleepy_input == "False":
    sleepy_b = False
else:
    print(" True or False 둘 중 하나만 입력해주세요.")
    exit(0)

"""
현재 주머니 사정과 커피 값 파악하고,
조건식 (buy_coffee) 정의
"""

money = float(input( '현재 가진 돈 (0 이상 실수): ' ))
coffee = float(input( '커피 한잔 값 (0 이상 실수): ' ))
bad_coffee = 100.0
"""파이썬은 수치 조건문에 다음과 같이 연속적인 조건 나열이 가능합니다."""
buy_coffee = bad_coffee < coffee < 0.2*money

print("sleepy_b = "+str(sleepy_b))
print("buy_coffee = "+str(buy_coffee))

if (sleepy_b and buy_coffee):
    print ("=> 졸리니 커피 한 잔 사 마셔야지 ...")
elif (sleepy_b and not buy_coffee):
    print ("=> 졸린데 커피가 너무 비싸군.")
elif (not sleepy_b and buy_coffee):
    print ("=> 졸리진 않은데, 심심하니 커피 한 잔 해야지.")
elif (not sleepy_b and not buy_coffee):
    print ("=> 졸리지도 않고, 커피도 비싸니 다음에.")
else:
    print("이건 무슨 경우지?")
```

```
wscho@ubuntu:~/PPSML $ python3 if_2.py
졸리니?(True/False): True
현재 가진 돈 (0 이상 실수): 10000
커피 한잔 값 (0 이상 실수): 1500
sleepy_b = True
buy_coffee = True
=> 졸리니 커피 한 잔 사 마셔야지 ...
```



# 1 차 (n01) 실습과제 (exercise)

## 실습목표 :

- 터미널에서 Interactive Python Shell 의 사용방법을 익힌다 .
- 터미널에서 IPython 사용방법을 익힌다 .
- Interactive Python Shell 에서 입력했던 간단한 명령어들을 파이썬 스크립트 파일 (.py) 로 종합하여 한번에 실행하는 방법을 익힌다 .
- Jupyter notebook 에서의 파이썬 스크립트 입력 및 실행 방법을 익힌다 .
- Jupyter notebook 에서 code 셀과 text(markdown) 셀을 조화롭게 사용하는 법을 익힌다 .

## 실습과제 :

- 본 ppsml\_note\_02 (2 주차 강의 ) 슬라이드에서 , 제목에 (\*) 가 표시된 항목들의 파이썬 스크립트 예제들을 , 자신만의 Jupyter notebook 으로 정리하여 미리 연습 및 테스트 ( 자유롭게 변형 가능 ) 해보고 노트북 파일 (.ipynb) 을 제출한다 .

(due 9/14( 금 ) 24:00, 파일이름형식 준수 )

# 1 차 (n01) 정규과제 (homework)

정규과제 :

(9/14( 금 ) - 강의중 소개 )

(due - 9/21( 금 ) 24:00, 파일이름형식 준수 )