

파이썬 프로그래밍 기초

목차

1. 파이썬 소개
2. 자료형
3. 제어문
4. 함수
5. 클래스
6. 모듈

파이썬 소개

프로그래밍 언어의 이해

- 프로그래밍 언어의 개념

- 프로그래밍 언어(programming language) : '인간이 원하는 것을 컴퓨터로 실행시키기 위해 사용하는, 컴퓨터가 이해할 수 있는 언어'이다.



[알파고와 이세돌의 바둑 대결(출처: Becoming Human)]

파이썬 소개

- **Python**

- 1990년 암스테르담의 귀도 반 로섬(Guido van Rossum)이 발표
- 인터프리터 방식 (한줄씩 소스 코드를 해석해서 그때 그때 실행해 결과를 바로 확인 가능)
- 귀도가 즐겨보던 영국의 6인조 코미디 그룹인 '몬티 파이썬'에서 유래
- 파이썬의 사전적 유래 - 그리스 신화에 등장했던 뱀의 이름이 파이썬. 파이썬 아이콘 모양이 뱀의 모양인 이유
- 초보자가 쉽게 배울 수 있는 프로그래밍 언어



파이썬 소개

- **파이썬의 장점**

- 비전공자도 쉽게 배울 수 있음
- 다양한 분야에서 활용할 수 있음
- 대부분의 운영체제에서 동일하게 사용됨

- **파이썬의 단점**

- c언어에 비해 실행속도가 많이 느림
- 최근 컴퓨팅 파워가 좋아져서 많은 연산이 필요한 경우가 아니라면 큰 차이를 느낄 수 없음

파이썬 소개

- **통합 개발 환경(IDE)**

- 프로그래밍을 할 수 있는 환경

- **텍스트 에디터**

- 프로그래밍 언어로 이루어진 코드를 작성
- 코드 실행기를 이용하여 텍스트 에디터가 작성한 코드를 실행

파이썬 소개

- **파이썬으로 할 수 있는 일**

- 시스템 유틸리티 제작
- GUI 프로그래밍
- C/C++와의 결합
- 웹 프로그래밍
- 수치 연산 프로그래밍
- 데이터베이스 프로그래밍
- 데이터 분석, 사물 인터넷

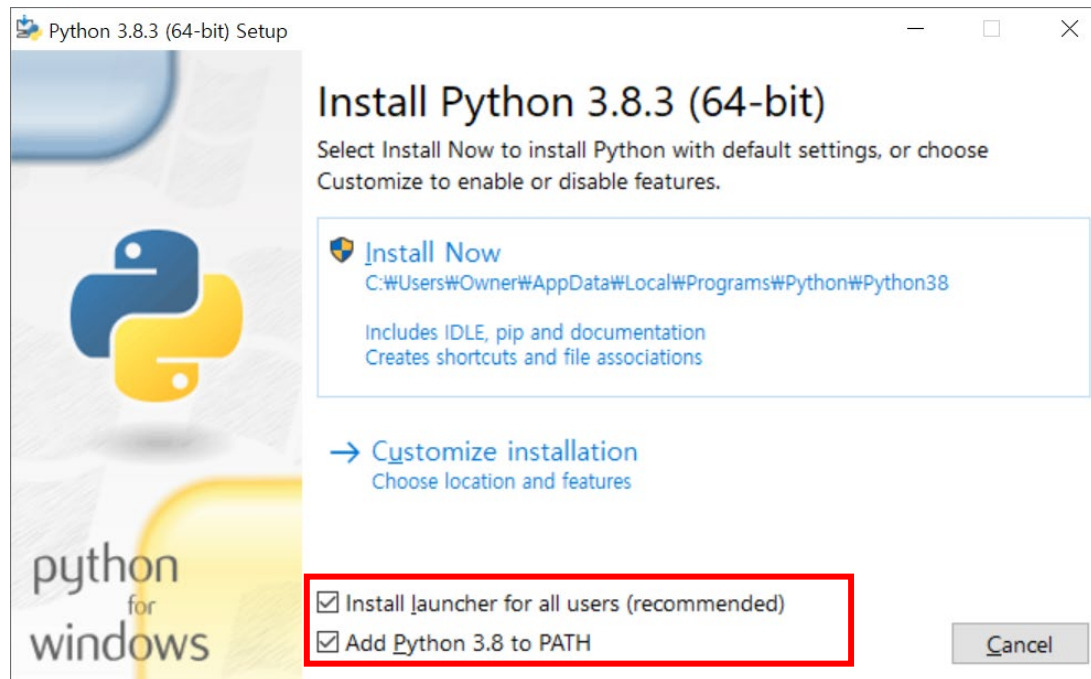
- **파이썬으로 할 수 없는 일**

- 시스템과 밀접한 프로그래밍 영역
- 모바일 프로그래밍

파이썬 설치 및 개발환경 구성

- 파이썬 설치

- 파이썬 공식 홈페이지(<https://www.python.org>)에서 파이썬 다운로드



파이썬 설치 및 개발환경 구성

- **인터프리터 (interpreter)**

- 파이썬으로 작성된 코드를 실행해주는 프로그램

- **파이썬 인터랙티브 셸**

- 파이썬 코드를 한 줄씩 입력하며 실행결과 볼 수 있는 도구

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32  
bit (Intel)] on win32 Type "help", "copyright", "credits" or "license"  
for more information.  
>>>
```

파이썬 설치 및 개발환경 구성


- **Anaconda 설치**

- 아나콘다 공식 홈페이지(<https://www.python.org>)에서 파이썬 다운로드
- <https://docs.anaconda.com/anaconda/install/>



Data science
technology for
human sensemaking.

A movement that brings together millions of data science practitioners, data-driven enterprises, and the open source community.



Get Started

파이썬 설치 및 개발환경 구성

- Anaconda 사용법

- introspection 인트로스펙션
- ? : 객체나 함수일때 정의되어 있는 문서를 출력해준다.
- ?? : 가능한경우 함수의 소스 코드도 보여준다.

```
a = [1,2,3,4]
```

```
a?
```

```
Type:          list
String form:    [1, 2, 3, 4]
Length:         4
Docstring:
Built-in mutable sequence.
```

```
If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

파이썬 설치 및 개발환경 구성

- Anaconda 사용법
 - introspection

```
def func_1(a,b):  
    return a+b
```

```
func_1??
```

Signature: func_1(a, b)

Docstring: <no docstring>

Source:

```
def func_1(a,b):  
    return a+b
```

File: c:\users\apollo\onedrive -\data\datascience\<ipython-input-19-41e494ecc00d>

Type: function

파이썬 설치 및 개발환경 구성

- **Anaconda 사용법**

- **매직 명령어**

- 파이썬 자체에는 존재하지 않는 '매직'명령어라고 하는 여러 가지 특수한 명령어를 포함하고 있다.
 - 명령어 앞에 % 기호를 붙여 사용한다.
 - %magic : 모든 매직함수에 대한 상세 도움말 출력
 - %time *statement* : statement의 단일 실행 시간을 출력
 - %timeit *statement* : statement를 여러 차례 실행한 후 평균 실행 시간을 출력, 매우 짧은 시간 안에 끝나는 코드를 측정할때 유용하다.
 - %pwd : 현재 위치를 나타낸다.
 - %matplotlib : 콘솔세션에 영향받지 않고 띄울수 있게 해준다. (통합되어서 작성하지 않아도 된다.)

파이썬 설치 및 개발환경 구성

- Anaconda 사용법

- 매직 명령어

```
%timeit [i for i in range(10)]
```

```
708 ns ± 1.56 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
%pwd
```

```
'C:\\Users\\apollo\\OneDrive -\\data\\datasience'
```

```
%magic
```

```
IPython's 'magic' functions  
=====
```

The magic function system provides a series of functions which allow you to control the behavior of IPython itself, plus a lot of system-type features. There are two kinds of magics, line-oriented and cell-oriented.

Line magics are prefixed with the % character and work much like OS command-line calls: they get as an argument the rest of the line, where arguments are passed without parentheses or quotes. For example, this will time the given statement::

파이썬 설치 및 개발환경 구성

- Anaconda 사용법

- 입출력 변수

- 함수의 실행 결과를 바로 반영할수 있도록 해준다.

```
2**20
```

```
1048576
```

```
_
```

```
1048576
```

```
_ +1
```

```
1048577
```


파이썬 설치 및 개발환경 구성

• Anaconda 사용법

▪ 입출력 변수

- 함수의 실행 결과를 바로 반영할수 있도록 해준다.

```
!dir
```

```
c 드라이브의 볼륨에는 이름이 없습니다.  
볼륨 일련 번호: 980D-3F52
```

```
C:\Users\apollo\OneDrive -\data\datascience 디렉터리
```

```
2020-08-27 오후 04:12 <DIR>      .  
2020-08-27 오후 04:12 <DIR>      ..  
2020-08-27 오후 03:52 <DIR>      .ipynb_checkpoints  
2020-08-27 오전 08:55          82,273 4_numpy.ipynb  
2020-08-19 오후 05:50 <DIR>      matplotlib  
2020-08-26 오전 11:40          586 savez_array.npz  
2020-08-26 오전 11:38          168 save_array.npy  
2020-08-27 오후 04:12      138,856 Untitled.ipynb  
                4개 파일          221,883 바이트  
                4개 디렉터리 159,855,824,896 바이트 남음
```

```
!ipconfig
```

```
Windows IP 구성
```

```
미더넷 어댑터 미더넷:
```

```
연결별 DNS 접미사. . . . . :  
링크-로컬 IPv6 주소 . . . . . : fe80::e933:14ae:cef2:1f5b%14  
IPv4 주소 . . . . . : 192.168.0.4  
서브넷 마스크 . . . . . : 255.255.255.0  
기본 게이트웨이 . . . . . : 192.168.0.1
```

파이썬 설치 및 개발환경 구성

- Anaconda 사용법
 - 운영체제 명령어
 - !: 명령어를 실행할수 있다.
 - %env : 시스템 환경변수 출력

파이썬 기본 사항

- **표현식 (expression)**
 - 값(숫자 수식, 문자열 등)을 만들어내는 간단한 코드
- **문장 (statement)**
 - 표현식이 하나 이상 모일 경우
- **프로그램 (program)**
 - 문장이 모여서 형성
- **키워드 (keyword)**
 - 프로그래밍 언어에서 미리 예약해 놓은 이름
 - 식별자 이름을 지정시 똑같이 사용할 수 없음

파이썬 기본 사항

- **식별자 (identifier)**

- 프로그래밍 언어에서 이름 붙일 때 사용하는 단어
- 변수 또는 함수 이름 등으로 사용함
- 키워드는 사용할 수 없음
- 언더 스코어('_') 외의 특수문자는 사용할 수 없음
- 숫자로 시작할 수 없음
- 공백은 사용할 수 없음
- 의미 있는 단어를 지정하는 것을 권장함

파이썬 기본 사항

- 연산자

- 연산을 수행하기 위한 문자

```
>>> 1 + 1
2
>>> 3 - 1
2
```

- 리터럴 (literal)

- 변수나 상수에 저장되는 값(Data) 자체

```
1
1234
"Hello World"
```

파이썬 기본 사항

- **print() 함수**

- 데이터를 출력하는 함수

- 하나만 출력

```
>>> print("Hello Python!")  
Hello Python!
```

- 여러 개 출력

```
>>> print(1, 2, "Hello Python!")  
1 2 Hello Python!
```

- 줄 바꿈(개행)

```
>>> print()
```

파이썬 기본 사항

- 주석 (comment)

- 프로그램 코드 실행시 영향 주지 않는 내용으로 설명을 위해 사용함
- # 기호를 주석으로 처리하고자 하는 부분 앞에 붙임(한 행 주석처리)
- 소스코드 파일에서 여러 줄을 주석 처리시 `""" 주석 """` 혹은 `''' 주석 '''`

```
"""
```

```
여러 줄을 주석 처리시  
따옴표 3개로 내용을 감싸주세요.
```

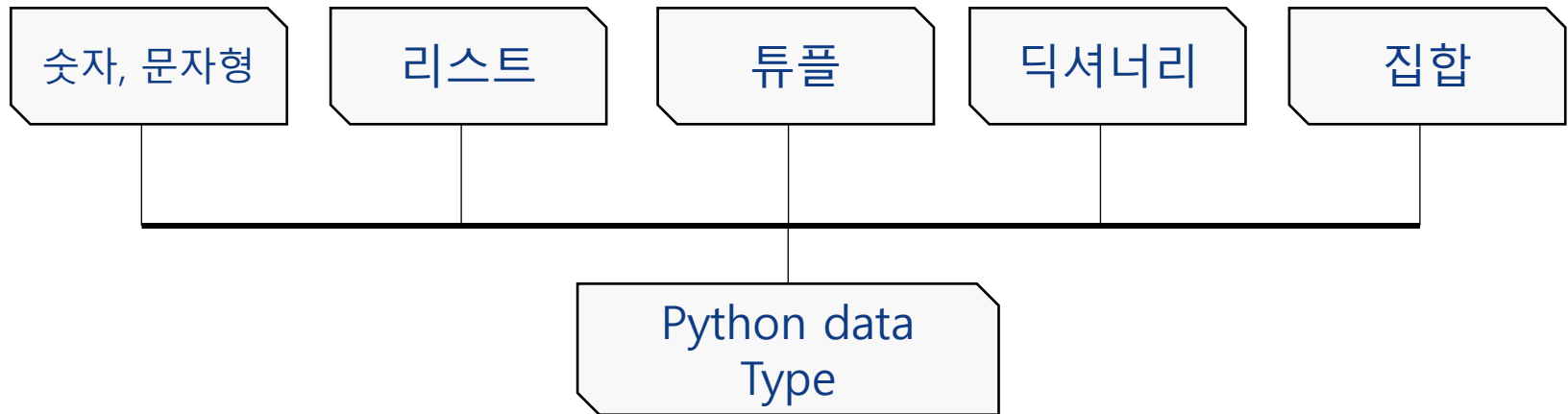
```
"""
```

자료형

자료형

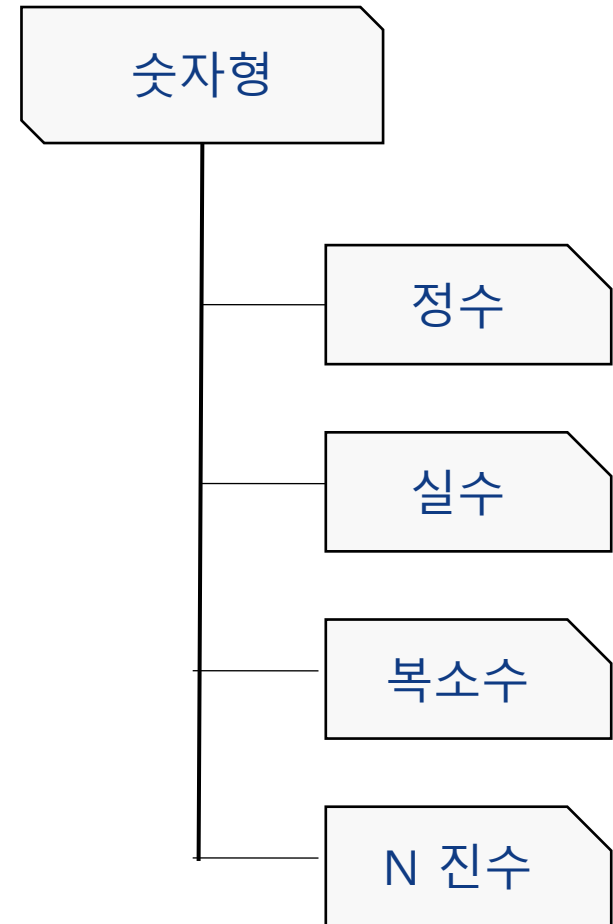
- 자료형 (data type)

- 자료형은 프로그램에서 자료를 호출, 가공, 저장 하기 위한 기본 단위이다.
- 자료형은 언어마다 다르며, 적절한 자료형을 사용하는 것이 성능과 효율 면에서 중요한 문제이다.
- 따라서, python 에서 다루고 있는 자료형을 알아야만 더 고급 프로그래밍 기법들에 대한 공부를 진행할 수 있다.



자료형

- 숫자형 자료는 숫자 형태로 이루어진 자료들을 의미한다.
- 파이썬에서는 복소수도 제공한다.
- 또한 정수와 실수 사이의 사칙 연산 역시 정상적으로 가능 하다. (캐스팅 불필요)
- 진법의 경우 개별 진법별로 표기하는 방법이 다르므로 숙지해야 한다.



자료형 - 숫자형

- 정수형(int)

- 일반적으로 사용하는 정수형 자료. 양의 정수, 음의 정수, 0
 - Ex) num1 = 1 / num2 = -1 / num3 = 0

- 실수형(float)

- 실수형은 소수점 숫자를 말함
- 일반적으로 사용 하는 표기 방식과 E,e를 사용한 부동소수점 표기법을 사용하기도 한다.
 - Ex) fpn1 = 3.141592 / fpn2 = 1.42e-2

자료형 - 숫자형

- 복소수(complex)

- 실수와 허수를 합친 형태로 만든 허수
- 허수 부분은 j로 표시(j는 -1의 제곱근)
- 복소수를 다루는 많은 함수와 메소드가 있음

복소수.real	복소수의 실수 부분 리턴	$a=1+2j$; a.real ⇒ 1.0을 리턴
복소수.imag	복소수의 허수 부분 리턴	$a=1+2j$; a.imag ⇒ 2.0을 리턴
복소수.conjugate()	복소수의 켄레 복소수 리턴	$a=1+2j$; a.conjugate() ⇒ $1-2j$ 를 리턴
abs(복소수)	복소수의 절대값을 리턴	$a=1+2j$; abs(a) ⇒ abs(a)

자료형 - 숫자형

- N 진수

- 파이썬은 2/8/16 진수를 기본적으로 표기 할 수 있음
- 또한 진법을 다루기 위한 다양한 메소드가 있음

2진수	0b로 표기	0b10 -> 2	bin()을 이용	bin(2) -> 0b10
8진수	0o로 표기	0o10 -> 8	oct()를 이용	oct(8) -> 0o10 or 010
16진수	0x로 표기	0x10 -> 16	hex()를 이용	hex(16) -> 0x10

자료형 - 숫자형

- **int**

```
>>> 10, 0x10, 0o10, 0b10  
(10, 16, 8, 2)
```

- **float**

```
>>> type(3.14), type(314e-2)  
(<class 'float'>, <class 'float'>)
```

- **complex**

```
>>> x=3-4j  
>>> type(x), x.imag, x.real, x.conjugate()  
(<class 'complex'>, -4.0, 3.0, (3+4j))
```

자료형 - 숫자형

• 연산자

▪ 산술 연산자

연산자	설명	사용 예시
+	덧셈 연산자	num1 + num2
-	뺄셈 연산자	num1 - num2
*	곱셈 연산자	num1 * num2
/	나눗셈 연산자	num1 / num2
%	나머지 연산자	num1 % num2
**	제곱 연산자	num1 ** num2
//	몫 연산자	num1 // num2

▪ 비교 연산자

연산자	설명	사용 예시
==	같다	(num1 == num2)
!=	같지 않다	(num1 != num2)
>	크다	(num1 > num2)
>=	크거나 같다	(num1 >= num2)
<	작다	(num1 < num2)
<=	작거나 같다	(num1 <= num2)

자료형 - 문자열

- 문자열 (string)

- 문자, 단어 등으로 구성된 문자들의 집합
- 큰 따옴표("") 또는 작은 따옴표('')로 묶어서 표현함
- \n, \t, \b 등의 특수 표기문자가 있음(이스케이프 문자)
- 문자열끼리 덧셈, 문자와 숫자의 곱셈이 가능함
- 문자열의 인덱싱, 슬라이싱이 가능함

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

```
>>> print('Hello World!')  
Hello World!
```

```
>>> print("""줄바꿈도  
... 그대로 적용됩니다""")  
줄바꿈도  
그대로 적용됩니다
```


자료형 - 문자열

- 문자열 안에 따옴표 넣기

```
>>> print('"Hello World!"라고 말한다.')  
Syntax Error: invalid syntax
```

- 문법 오류(Syntax Error)가 발생하여 실행 불가함

- 다음과 같이 시도해보자.

```
>>> print('"Hello World!"라고 말한다.')  
"Hello World!"라고 말한다.
```

```
>>> print("'Hello World!'라고 말한다.")  
'Hello World!'라고 말한다.
```

자료형 - 문자열

- **Escape 문자**

- \ 기호로 시작하는 특수한 문자로 문자열 내부에서 특수한 기능을 수행함

Escape 문자	설명
\n	개행(줄바꿈)
\t	탭
\r	캐리지 리턴
\0	널(Null)
\\	문자 \'\'
\'	단일 인용부호(')
\"	이중 인용부호(")

자료형 - 문자열

- 문자열 덧셈 연산

- 서로 다른 문자열을 합치는 연산

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
Python is fun!
```

- 문자열 곱셈 연산

- 문자열을 지정한 횟수만큼 반복하는 연산

```
>>> print("=" * 50)
>>> print("My Program")
>>> print("=" * 50)
=====
My Program
=====
```

자료형 - 문자열

- 인덱싱과 슬라이싱
- 인덱싱 : []
 - 문자 선택 연산자
 - 문자열 내부의 문자 하나를 선택함
 - 대괄호 안에 선택할 문자의 위치를 지정
 - 인덱스 (index)
 - 문자열 내부를 접근하기 위한 위치 값
 - zero index : 인덱스 숫자를 0부터 시작함
 - one index : 인덱스 숫자를 1부터 시작함

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

자료형 - 문자열

- 인덱싱

```
>>> a = "Life is too short, You need Python"
>>> a[3]
'e'
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
>>> a[-0]
'L'
>>> a[-2]
'o'
>>> a[-5]
'y'
```

자료형 - 문자열

- 인덱싱과 슬라이싱
- 슬라이싱 : [:]
 - 문자열 범위 선택 연산자
 - 문자열의 특정 범위를 선택
 - 대괄호 안에 범위 구분 위치를 콜론으로 구분
 - 파이썬에서는 마지막 숫자를 포함하지 않음

p	y	t	h	o	n
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

자료형 - 문자열

- 슬라이싱

```
>>> a = "Life is too short, You need Python"
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
>>> a[0:4]
'Life'
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
>>> a[:17]
'Life is too short'
>>> a[:]
'Life is too short, You need Python'
>>> a[19:-7]
'You need'
```

자료형 - 문자열

- 문자열 포매팅(Formatting)

- 문자열 안에 어떤 값을 동적으로 삽입하여 표현하는 방법

- 숫자 변수 대입

```
>>> number = 3
>>> print ("I eat %d apples." % number)
I eat 3 apples.
```

- 두 개 이상의 변수 대입

```
>>> number = 10
>>> day = "three"
>>> print ("I eat %d apples. so I was sick for %s days." % (number, day))
I eat 10 apples. so I was sick for three days.
```


자료형 - 문자열

- 문자열 포매팅(Formatting)

- 문자열 포맷 코드

포맷 코드	설명
%s	문자열 (string)
%c	문자 한 개 (Character)
%d	정수 (Integer)
%f	부동소수 (Floating-point)
%o	8 진수
%x	16 진수
%%	Literal % (문자 '%' 표시)

자료형 - 문자열

- 문자열 포매팅(Formatting)

- 정렬과 공백

```
>>> print("%10s" % "hi")
      hi
```

- 소수점 표현

```
>>> print("%0.4f" % 3.42134234)
3.4213
>>> print("%10.4f" % 3.42134234)
      3.4213
```

- format 함수 이용

```
print(" { }    { }    { } ".format( 11 , 22 , 33 ))
      11      22      33
```

– 넣고자 하는 데이터들을 format 함수 안에 순차적으로 작성

자료형 - 문자열

- 문자열 포매팅(Formatting)

- f-string

```
>>> test = 'hello'
>>> f'{}world'
hello world
```

자료형 - 문자열

- 문자열 함수

- 문자 갯수 세기(count)

```
>>> a = "hobby"  
>>> a.count('b')  
2
```

- 위치 알려주기1(find)

```
>>> a = "Python is best choice"  
>>> a.find('b')  
10
```

- 위치 알려주기2(index)

```
>>> a = "Life is too short"  
>>> a.index('t')  
8
```

- 문자열 삽입 (join)

```
>>> a = ","  
>>> a.join('abcd')  
'a,b,c,d'
```

자료형 - 문자열

- 문자열 함수

- 소문자->대문자(upper)

```
>>> a = "hi"  
>>> a.upper()  
'HI'
```

- 대문자 -> 소문자 (lower)

```
>>> a = "HI"  
>>> a.lower()  
'hi'
```

- 왼쪽 공백 지우기 (lstrip)

```
>>> a = "          hi"  
>>> a.lstrip()  
'hi'
```

- 오른쪽 공백 지우기 (rstrip)

```
>>> a= "hi          "  
>>> a.rstrip()  
'hi'
```

자료형 - 문자열

- 문자열 함수

- 양쪽 공백 지우기 (strip)

```
>>> a = "          hi          "  
>>> a.strip()  
'hi'
```

- 문자열 바꾸기 (replace)

```
>>> a = "Life is too short"  
>>> a.replace("Life", "Your leg")  
'Your leg is too short'
```

- 문자열 나누기 (split)

```
>>> a = "Life is too short"  
>>> a.split()  
['Life', 'is', 'too', 'short']  
  
>>> a = "a:b:c:d"  
>>> a.split(':')  
['a', 'b', 'c', 'd']
```

자료형 - 리스트

- **리스트 (List)**

- 여러 개의 값을 담을 수 있는 변수
- 자료들을 모아서 사용할 수 있도록 함
- 대괄호 안에 자료들을 넣어 선언함

```
>>> a = []  
>>> b = [1, 2, 3]  
>>> c = ['Life', 'is', 'too', 'short']  
>>> d = [1, 2, 'Life', 'is']  
>>> e = [1, 2, ['Life', 'is']]
```

- **요소 (Element)**

- 리스트의 대괄호 안에 넣는 자료

자료형 - 리스트

- List의 인덱싱

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
>>> a[0]
1
>>> a[0] + a[2]
4
>>> a[-1]
3
>>> a = [1, 2, 3, ['a', 'b', 'c']]
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
```


자료형 - 리스트

- List의 인덱싱

```
>>> a[-1][0]
'a'
>>> a[-1][1]
'b'
>>> a[-1][2]
'c'
>>> a = [1, 2, ['a', 'b', ['Life', 'is']]]
>>> a[2][2][0]
'Life'
>>> a[2][1]
'b'
```

자료형 - 리스트

- List의 슬라이싱

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2, ]
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> c = a[2:]
>>> b
[1, 2]
>>> c
[3, 4, 5]
>>> a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
>>> a[2:5]
[3, ['a', 'b', 'c'], 4]
>>> a[3][:2]
['a', 'b']
```

자료형 - 리스트

- List 합치기

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

- List 반복

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

자료형 - 리스트

• List 요소 수정

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
>>> a[1:2]
[2]
>>> a[1:2] = ['a', 'b', 'c']
>>> a
[1, 'a', 'b', 'c', 4]
>>> a[0] = ['a1', 'a2', 'a3']
>>> a
[['a1', 'a21', 'a3'], 'a', 'b', 'c', 4]
```

- 참고 : 여기서 `a[1] = ['a', 'b', 'c']`와는 전혀 다른 결과값을 갖게 되므로 주의 하도록 하자.
- `a[1] = ['a', 'b', 'c']`는 리스트 `a`의 두 번째 요소를 `['a', 'b', 'c']`로 바꾼다는 의미이고 `a[1:2]`는 `a[1]`에서 `a[2]`사이의 리스트를 `['a', 'b', 'c']`로 바꾼다는 말이다. 따라서 `a[1] = ['a', 'b', 'c']`처럼 하면 위와는 달리 리스트 `a`가 `[1, ['a', 'b', 'c'], 4]`라는 값으로 변하게 된다.

자료형 - 리스트

- List 요소 삭제

```
>>> a
[['a1', 'a2', 'a3'], 'a', 'b', 'c', 4]
>>> a[1:3] = []
>>> a
[['a1', 'a21', 'a3'], 'c', 4]
>>> del a[1]
>>> a
[['a1', 'a21', 'a3'], 4]
```

자료형 - 리스트

- List 관련 함수들

- 리스트에 요소 추가 (append) : 리스트의 맨 마지막에 x를 추가시키는 함수

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> a.append([5,6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

- 리스트 정렬(sort) : 리스트의 요소를 순서대로 정렬하여 정렬된 값을 돌려준다.

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
>>> a = ['abc', '123', 'you need python']
>>> a.sort()
>>> a
['123', 'abc', 'you need python']
```

자료형 - 리스트

- List 관련 함수들

- 리스트 뒤집기(reverse) : 리스트를 역순으로 뒤집어주는 것으로 리스트를 그대로 거꾸로 뒤집는 일을 역할을 함.

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

- 위치 반환 (index) : index(x) 함수는 리스트에 x라는 값이 있으면 그 위치를 돌려준다.

```
>>> a = [1,2,3]
>>> a.index(3)
2
>>> a.index(1)
0
>>> a.index(0)
Traceback (innermost last):
File "", line 1, in ?
a.index(0)
ValueError: list.index(x): x not in list
```

자료형 - 리스트

- List 관련 함수들

- 리스트에 요소 삽입 (insert) : insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수

```
>>> a = [1,2,3]
>>> a.insert(0, 4)
[4, 1, 2, 3]
>>> a.insert(3, 5)
[4, 1, 2, 5, 3]
```

- 리스트 요소 제거 (remove) : remove(x)는 첫번째 나오는 x 를 삭제하는 함수

```
>>> a = [1,2,3,1,2,3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2]
```


자료형 - 리스트

- List 관련 함수들

- 리스트 요소 끄집어내기(pop) : 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제

```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
>>> a = [1,2,3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

– 인덱스 지정시 해당 인덱스의 요소 제거

- 갯수세기(count) : count(x)는 리스트 중에서 x가 몇 개 있는지를 조사하여 그 갯수를 돌려주는 함수

```
>>> a = [1,2,3,1]
>>> a.count(1)
2
```

자료형 - 리스트

- List 관련 함수들

- 리스트 확장(extend) : extend(x)에서 x에는 리스트만 올 수 있다.
원래의 a 리스트에 x 리스트를 더하게 된다.

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
```

- a.extend([4,5])는 a + [4,5]와 동일

자료형 - 튜플

- 튜플 (Tuple)

- 여러 값을 함께 모을 수 있는 자료형으로 튜플은 (과)으로 둘러쌘
- 리스트와 유사하나 한 번 만든 값을 변경하고 싶지 않은 경우에 사용함(읽기 전용)
- 실제 프로그램에서는 값이 변경되는 형태의 변수가 훨씬 많기 때문에 평균적으로 튜플 보다는 리스트를 더 많이 사용함
- 리스트 관련 함수의 일부(내용을 확인하는 함수)는 튜플에서도 사용가능함

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

- 리스트와 모습에서의 차이점
 - t2 = (1,)처럼 단지 1개의 요소만을 가질 때는 뒤에 콤마(,)를 반드시 붙여야 함
 - t4 = 1, 2, 3처럼 괄호()를 생략해도 무방함

자료형 - 튜플

- 튜플의 요소 변경 여부

- 튜플의 요소값은 한 번 정하면 지우거나 변경할 수 없다.

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
Traceback (innermost last):
File "", line 1, in ?del t1[0]
TypeError: object doesn't support item deletion

>>> t1[0] = 'c'
Traceback (innermost last):
File "", line 1, in ?t1[0] = 'c'
TypeError: object doesn't support item assignment
```

자료형 - 튜플

- 튜플 인덱싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

- 튜플 슬라이싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

자료형 - 튜플

- 더하기

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

- 곱하기(반복)

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

- 길이 구하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```

자료형 - 딕셔너리

- 딕셔너리 (Dictionary)

- 키(Key)와 값(Value)의 쌍으로 구성된 자료형
 - 키는 변경할 수 없는 자료형만 사용 가능함
 - 값은 자료형과 무관함
- {키1: 값1, 키2: 값2, ...}
 - 키와 값을 콜론(:)으로 묶음
 - 중괄호와 콤마 사용

```
>>> dic = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
```

```
>>> a = {1: 'hi'}
```

```
>>> a = { 'a': [1,2,3]}
```

〈딕셔너리 dic의 정보〉

key	value
name	pey
phone	01199993323
birth	1118

자료형 - 딕셔너리

- 딕셔너리 추가

- 대괄호 사용
- 딕셔너리[추가할_키] = 추가할_값
 - 대괄호 안에 키 넣고 = 사용해 값 저장

```
>>> a = {1:'a'}
>>> a[2] = 'b'
>>> a
{1:'a', 2:'b'}
>>> a['name'] = 'pey'
>>> a
{1:'a', 2:'b', 'name':'pey'}
>>> a[3] = [1,2,3]
>>> a
{1:'a', 2:'b', 'name':'pey', 3:[1,2,3]}
```


자료형 - 딕셔너리

- 딕셔너리 삭제

- `del a[key]`하면 그에 해당하는 key:value 쌍을 삭제함

```
>>> del a[1]
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

자료형 - 딕셔너리

- Key 이용하여 Value 얻기

```
>>> a = {1:'a', 2:'b'}  
>>> a[1]  
'a'  
>>> a[2]  
'b'
```

- a[1] 이 의미하는 것은 리스트나 튜플의 a[1] 과는 아주 다름
- [] 안의 숫자 1은 몇번째 요소를 뜻하는 것이 아니라 Key에 해당하는 1을 나타냄.
- 딕셔너리는 리스트나 터플의 인덱싱 방법이란 것이 존재하지 않음.
- a[1]은 딕셔너리 {2:'b'}에서 Key가 1인것의 Value인 'a'를 돌려주게 됨.

자료형 - 딕셔너리

- Key 리스트 만들기(keys)

- 리턴값으로 리스트가 필요한 경우에는 "list(a.keys())"를 사용
- dict_keys 객체는 for문에서 사용 가능
- dict_keys 객체는 리스트를 사용하는 것과 차이가 없지만, 리스트 고유의 함수인 append, insert, pop, remove, sort 등의 함수를 수행 불가능

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
>>> list(a.keys())
['name', 'phone', 'birth']
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

자료형 - 딕셔너리

- **Value리스트 만들기(values)**

```
>>> a.values()  
dict_values(['pey', '0119993323', '1118'])
```

- Value만 얻고 싶다면 a.values()처럼 values 함수를 사용
- values 함수를 호출하면 dict_values 객체가 리턴되는데, dict_values 객체 역시 dict_keys 객체와 마찬가지로 리스트를 사용하는 것과 동일하게 사용

- **Key, Value 쌍 얻기(items)**

```
>>> a.items()  
[('birth', '1118'), ('name', 'pey'), ('phone', '011993323')]
```

- items 함수는 key와 value의 쌍을 튜플로 묶은 값을 리스트로 돌려준다.

자료형 - 딕셔너리

- 딕셔너리 모두 지우기(clear)

```
>>> a.clear()  
>>> a  
{}
```

- clear() 함수는 딕셔너리 안의 모든 요소를 삭제
- 빈 리스트를 [], 빈 튜플을 ()로 표현하는 것과 마찬가지로 빈 딕셔너리도 {}로 표현

자료형 - 딕셔너리

- Key로 Value얻기 (get)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}  
>>> a.get('name')  
'pey'  
>>> a.get('phone')  
'0119993323'
```

- get(x) 함수는 x 라는 key에 대응되는 value를 리턴
- a.get('name')은 a['name']을 사용했을 때와 동일한 결과값을 리턴

자료형 - 딕셔너리

- Key로 Value얻기 (get)

```
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

- a['nokey']처럼 a 딕셔너리에 없는 키로 값을 가져오려고 할 경우 a['nokey']는 Key 오류를 발생시키고 a.get('nokey')는 None을 리턴한다는 차이가 있음 (None = False)
- 딕셔너리 안에 찾으려는 key 값이 없을 경우 미리 정해 둔 디폴트 값을 대신 가져 오게 하고 싶을 때에는 get(x, '디폴트 값')을 사용하면 편리함

```
>>> a.get('foo', 'bar')
'bar'
```

- a 딕셔너리에는 'foo'에 해당하는 값이 없음으로 디폴트 값인 'bar'를 리턴

자료형 - 딕셔너리

- 해당 Key가 있는지 조사 (in)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}  
>>> 'name' in a  
True  
>>> 'email' in a  
False
```

- in 함수는 딕셔너리 a에 x라는 key가 존재 여부의 참, 거짓을 판단하여 key가 존재하면 True를 key가 존재하지 않는다면 False를 반환함

자료형 - 집합

- **집합 (Set)**

- 집합은 파이썬 2.3부터 지원되기 시작한 자료형으로, 집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

- s1 처럼 set()의 괄호 안에 리스트를 입력하여 만들거나 s2 처럼 문자열을 입력하여 만들 수도 있다.
- s2를 보면 "Hello"라는 문자열로 set 자료형을 만들었는데 생성된 자료형에는 l 문자가 하나 빠져 있고 순서도 뒤죽박죽이다. 그 이유는 set에는 다음과 같은 2가지 큰 특징이 있기 때문이다.

자료형 - 집합

- **집합 (Set) 자료형의 특징**

- 중복을 허용하지 않는다.
- 순서가 없다(Unordered)
 - 리스트나 튜플은 순서가 있기(ordered) 때문에 인덱싱을 통해 자료형의 값을 얻을 수 있지만 set 자료형은 순서가 없기(unordered) 때문에 인덱싱으로 값을 얻을 수 없다.
 - ※ 중복을 허용하지 않는 set의 특징은 자료형의 중복을 제거하기 위한 필터 역할로 사용
- set 자료형에 저장된 값을 인덱싱으로 접근하려면 리스트함수(list())나 튜플함수 (tuple())로 변환한 후 가능

자료형 - 집합

- 교집합, 합집합, 차집합

```
>>> s1 = set([1, 2, 3, 4, 5, 6])  
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 교집합

```
>>> s1 & s2  
{4, 5, 6}  
>>> s1.intersection(s2)  
{4, 5, 6}
```

- 합집합

```
>>> s1 | s2  
{1, 2, 3, 4, 5, 6, 7, 8, 9}  
>>> s1.union(s2)  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

자료형 - 집합

- 교집합, 합집합, 차집합

```
>>> s1 = set([1, 2, 3, 4, 5, 6])  
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 차집합

```
>>> s1 - s2  
{1, 2, 3}  
>>> s2 - s1  
{8, 9, 7}  
>>> s1.difference(s2)  
{1, 2, 3}  
>>> s2.difference(s1)  
{8, 9, 7}
```

자료형 - 집합

- 집합 자료형 관련 함수

- 값 1개 추가하기(add)

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

- 값 여러 개 추가하기(update)

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

- 특정 값 제거하기(remove)

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

자료형 - 부울

- **부울 (Boolean)**

- 참과 거짓을 나타내는 자료형으로, 가능한 값은 True와 False 뿐이다.
 - ※ True나 False는 파이썬의 예약어로 true, false와 같이 사용하지 말고 첫 문자를 항상 대문자로 사용
- 주로 부울은 부울 값들 간의 논리연산이나, 수치 간의 비교연산의 결과로 사용
- 비교연산자 : '크다(>)', '작다(<)', '같다(==)', '다르다(!=)', '크거나 같다(>=)', '작거나 같다(<=)'
- 논리연산자 : 'and(&)', 'or(|)', 'not'
 - and : 두 값이 모두 참이어야만 참을 반환
 - or : 둘 중 하나의 값만 참이면 참을 반환
 - not : 반대의 값을 반환
- 논리연산자는 비교연산자와 함께 제어문의 조건에서 주로 사용

제어문

제어문 - if 문

- if

- 프로그래밍에서 조건을 판단하여 해당 조건에 맞는 상황을 수행하는 데 쓰이는 것이 바로 if 문
- "돈이 있으면 택시를 타고 가고 돈이 없으면 걸어간다."

```
>>> money = True
>>> if money :
...     print("택시를 타고 가라")
... else:
...     print("걸어 가라")
...
택시를 타고 가라
```

- money에 입력된 True는 참
- if문 다음의 문장이 수행되어 '택시를 타고 가라'가 출력

제어문 - if 문

- if 문의 기본 구조

```
if <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    ...  
else:  
    <수행할 문장A>  
    <수행할 문장B>  
    ...
```

- 조건문을 테스트해서 참이면 if문 바로 다음의 문장(if 블록)들을 수행
- 조건문이 거짓이면 else문 다음 의 문장(else 블록)들을 수행
- else문은 if문 없이 독립적으로 사용 불가

제어문 - if 문

- **pass**

- 조건문을 판단하고 참 거짓에 따라 행동을 정의 할 때 아무런 일도 하지 않게끔 설정을 하고 싶을 때 사용
- "주머니에 돈이 있으면 가만히 있고 주머니에 돈이 없으면 카드를 꺼내라"

```
>>> pocket = ['paper', 'money', 'cellphone']
>>> if 'money' in pocket:
...     pass
... else:
...     print("카드를 꺼내라")
...
```

- pocket이라는 리스트 안에 money라는 문자열이 있기 때문에 if문 다음 문장인 pass가 수행
- 아무런 결과값도 보여 주지 않음

제어문 - if 문

- 다양한 조건을 판단하는 elif

- if-elif-else의 구조

```
if <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    ...  
elif <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    ...  
elif <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    ...  
else:  
    <수행할 문장1>  
    <수행할 문장2>  
    ...
```

제어문 - if 문

- 다양한 조건을 판단하는 elif

- if와 else만으로는 다양한 조건을 판단하기 어려운 경우가 발생
- "주머니에 돈이 있으면 택시를 타고, 주머니에 돈은 없지만 카드가 있으면 택시를 타고, 돈도 없고 카드도 없으면 걸어 가라"
- if 와 else 만으로 구성

```
>>> pocket = ['paper', 'handphone']
>>> card = True
>>> if 'money' in pocket:      # => 첫번째 판단
...     print(" 택시를 타다가라 ")
... else:
...     if card:               # => 두번째 판단 (들여쓰기 실행 필수)
...         print("택시를 타다가라")
...     else:
...         print("걸어가라")
...
... 택시를 타다가라
>>>
```

제어문 - if 문

- 다양한 조건을 판단하는 elif

- elif 로 구성

```
>>> pocket = ['paper', 'cellphone']
>>> card = True
>>> if 'money' in pocket:           # => 첫번째 판단
...     print("택시를 타고가라")
... elif card:                     # => 두번째 판단(들여쓰기 미실행)
...     print("택시를 타고가라")
... else:
...     print("걸어가라")
...
택시를 타고가라
```

제어문 - if 문

- if 문을 한 줄로 작성

- 기존 구조

```
>>> if 'money' in pocket :  
...     pass  
... else:  
...     print("카드를 꺼내라")  
...
```

- 한줄 구조

```
>>> pocket = ['paper', 'money', 'cellphone']  
>>> if 'money' in pocket : pass  
... else : print("카드를 꺼내라")  
...
```

제어문 - while 문

- **while**

- 반복해서 문장을 수행해야 할 경우 while문을 사용

- while 문의 기본 구조

```
while <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    <수행할 문장3>  
...
```

제어문 - while 문

- **while**

- "열 번 찍어 안 넘어 가는 나무 없다"

```
>>> treeHit = 0
>>> while treeHit < 10 :
...     treeHit = treeHit +1
...     print ("나무를 %d번 찍었습니다." % treeHit)
...     if treeHit == 10:
...         print ("나무 넘어갑니다.")
```

```
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```


제어문 - while 문

- while 문 탈출하기

- break

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print ("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee -1
...     print ("남은 커피의 양은 %d 입니다." % coffee)
...     if not coffee:
...         print ("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
... 
```

제어문 - while 문

- while문 처음으로 되돌아가기

- continue

```
>>> a = 0
>>> while a < 10:
...     a = a+1
...     if a % 2 == 0 :
...         continue
...     print (a)
...
1
3
5
7
9
```

제어문 - for 문

- for
 - for 문의 기본 구조

```
for 변수 in 리스트(또는 튜플, 문자열):  
    <수행할 문장1>  
    <수행할 문장2>  
    ...
```
 - 리스트의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입해서 <수행할 문장1>, <수행할 문장2>...을 수행

제어문 - for 문

- for
 - 전형적인 for문

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list :
...     print(i)
...
one
two
three
```
 - ['one', 'two', 'three']라는 리스트의 첫 번째 요소인 'one'이 먼저 i 변수에 대입된 후 print(i)라는 문장을 수행
 - 다음에 'two'라는 두 번째 요소가 i 변수에 대입된 후 print(i) 문장을 수행하고 리스트의 마지막 요소까지 이것을 반복

제어문 - for 문

- for

- 다양한 for문의 사용

```
>>> a = [(1,2), (3,4), (5,6)]
>>> for (first, last) in a:
...     print(first + last)
...
3
7
11
```

- a 리스트의 요소값이 튜플이기 때문에 각각의 요소들이 자동으로 (first, last)라는 변수에 대입
- 이 예는 이전에 살펴보았던 튜플을 이용한 변수값 대입 방법과 매우 비슷한 경우임

```
>>> (first, last) = (1, 2)
```

제어문 - for 문

- **for**
 - for문의 응용
 - "총 5명의 학생이 시험을 보았는데 시험점수가 60점이 넘으면 합격이고 그렇지 않으면 불합격이다. 합격인지 불합격인지에 대한 결과를 보여준다."
 - # marks1.py

```
marks = [90, 25, 67, 45, 80]
number = 0
for mark in marks :
    number = number +1
    if mark >= 60 :
        print ("%d번 학생은 합격입니다." % number)
    else :
        print ("%d번 학생은 불합격입니다." % number)
```

제어문 - for 문

- **for**
 - **continue**
 - continue를 for문에서도 사용 가능
 - for문 안의 문장을 수행하는 도중에 continue문을 만나면 for문의 처음으로 돌아가게 된다.
 - “총 5명의 학생이 시험을 보았는데 시험점수가 60점이 넘으면 합격이고 그렇지 않으면 불합격이다. 합격일 때만 결과를 보여준다.”
 - # marks2.py

```
marks = [90, 25, 67, 45, 80]
number = 0
for mark in marks :
    number = number +1
    if mark < 60 :
        continue
    print ("%d번 학생 축하합니다. 합격입니다. " % number)
```

제어문 - for 문

- **for**

- **range**

- 숫자 리스트를 자동으로 만들어 주는 함수

```
>>> a = range(10)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> a = range(1, 11)
>>> list(a)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```


제어문 - for 문

- **for**

- range

- #marks3.py

```
marks = [90, 25, 67, 45, 80]
for number in range(len(marks)) :
    if marks[number] < 60 :
        continue
    print ("%d번 학생 축하합니다. 합격입니다." % (number+1))
```

- len함수 : 리스트의 요소 개수를 돌려주는 함수
 - len(marks) => 5
 - range(len(marks)) => range(5)

제어문 - for 문

- for

- range

```
>>> for i in range(2,10):      # ①번 for문
...     for j in range(1, 10): # ②번 for문
...         print( i * j , end=" ")
...     print('')
...
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
...
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

- print(i * j , end=" ")와 같이 입력 인수 end를 넣어 준 이유는 해당 결과값을 출력할 때 다음줄로 넘기지 않고 그 줄에 계속해서 출력하기 위함임
- print("")는 2단, 3단 등을 구분하기 위해 두 번째 for문이 끝나면 결과값을 다음 줄부터 출력하게 해주는 문장

제어문 - for 문

- **for**
 - 리스트 안에 for문 포함하기

```
>>> a = [1,2,3,4]
>>> result = []
>>> for num in a:
...     result . append(num*3)
...
>>> print(result)
[3, 6, 9, 12]
```

```
>>> a = [1,2,3,4]
>>> result = [num * 3 for num in a]
>>> print(result)
[3, 6, 9, 12]
```

제어문 - for 문

- **for**

- 리스트 안에 for문 포함하기

```
>>> a = [1,2,3,4]
>>> result = [num * 3 for num in a if num % 2 == 0]
>>> print(result)
[6, 12]
```

- [1, 2, 3, 4] 중에서 짝수인 2와 4에만 3을 곱하여 담고 싶다면 리스트 내포 안에 "if 조건문"을 사용 가능

```
[표현식 for 항목 in 반복가능객체 if 조건문]
```

- for문을 2개 이상 사용하는 것도 가능

```
[표현식 for 항목1 in 반복가능객체1 if 조건문1
      for 항목2 in 반복가능객체2 if 조건문2
      ...
      for 항목n in 반복가능객체n if 조건문n]
```

함수

함수

• 함수의 구조

```
def 함수명(매개변수):  
    <수행할 문장1>  
    <수행할 문장2>  
    ...
```

- 함수의 선언은 def로 시작하고 콜론(:)으로 끝냄
- def는 함수를 만들 때 사용하는 예약어
- 함수명은 함수를 만드는 사람이 임의로 만들 수 있음
- 함수명 뒤 괄호 안의 매개변수는 이 함수에 입력으로 전달되는 값을 받는 변수
- 함수의 시작과 끝은 코드의 들여쓰기로 구분
- 시작과 끝을 명시해 줄 필요가 없다.

함수

- 함수의 정의

```
def sum(a, b):  
    return a + b
```

- "이 함수의 이름(함수명)은 sum이고 입력으로 2개의 값을 받으며 결과값은 2개의 입력값을 더한 값이다."
- return 은 함수의 결과값을 돌려주는 명령어

함수

- 매개변수와 인수

- 매개변수는 함수에 입력으로 전달된 값을 받는 변수
- 인수는 함수를 호출할 때 전달하는 입력값

```
def sum(a, b) : # a, b는 매개변수  
    return a+b
```

```
print(sum(3, 4)) # 3, 4는 인수
```


함수

- 입력값과 결과값에 따른 함수의 형태

- 입력값 ---> 함수 ----> 리턴값

- 일반적인 함수

- 입력값이 있고 결과값이 있는 함수가 일반적인 함수

```
def 함수이름(매개변수):  
    <수행할 문장>  
    ...  
    return 결과값
```

```
def sum(a, b):           # 입력값 0  
    result = a + b  
    return result       # 결과값 0
```

- 결과값을 받을 변수 = 함수명(입력 인수 1, 입력 인수 2, ...)

함수

- 입력값과 결과값에 따른 함수의 형태

- 입력값이 없는 함수

- 입력값이 없고 결과값이 있는 함수

```
>>> def say( ):          # 입력값 x
...     return 'Hi'      # 결과값 0

>>> a = say()
>>> print(a)
Hi
```

- 결과값을 받을 변수 = 함수명()

함수

- 입력값과 결과값에 따른 함수의 형태

- 결과값이 없는 함수

- 입력값은 있고 결과값이 없는 함수

```
>>> def sum(a, b):  
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))  
...  
>>> sum(3, 4)  
3, 4의 합은 7입니다.  
>>> a = sum(3, 4)  
>>> print(a)  
None
```

- 함수명(입력 인수1, 입력 인수2, ...)

함수

- 입력값과 결과값에 따른 함수의 형태

- 입력값도 결과값도 없는 함수

```
>>> def say():  
...     print('Hi')  
...  
>>>  
>>> say()  
Hi
```

– 함수명()

함수

- 매개변수 지정하여 호출

```
>>> def sum(a, b):  
...     return a+b  
...  
  
>>> sum(a=3, b=7) # a에 3, b에 7을 전달  
10  
>>> sum(b=5, a=3) # b에 5, a에 3을 전달  
8
```

함수

- ***매개변수**

```
def 함수이름(*매개변수):  
    <수행할 문장>  
    ...
```

- 매개변수명 앞에 *을 붙이면 입력값들을 전부 모아서 튜플로 만들어 줌

함수

- ****매개변수**

```
def 함수이름(**매개변수):  
    <수행할 문장>  
    ...
```

- 매개변수명 앞에 **을 붙이면 입력값들을 전부 모아서 딕셔너리로 만들어 줌

```
>>> func(a=1)  
{'a': 1}  
>>> func(name='foo', age=3)  
{'age': 3, 'name': 'foo'}
```

함수

- return 값

```
>>> def sum_and_mul(a,b):  
...     return a+b, a*b  
...  
>>> result = sum_and_mul(3,4)  
>>> result  
(7,12)  
>>> sum, mul = sum_and_mul(3, 4)  
>>> sum  
7  
>>> mul  
12
```

- sum_and_mul 함수의 결과값 a+b와 a*b는 튜플값 하나인 (a+b, a*b)로 돌려준다.

함수

- return 값

```
>>> def sum_and_mul(a,b):  
...     return a+b  
...     return a*b  
...  
>>> result = sum_and_mul(2, 3)  
>>> print(result)  
5
```

- `sum_and_mul(2, 3)`의 결과값은 5 하나뿐이다. 두 번째 `return`문인 `return a*b`는 실행되지 않았다는 뜻

함수

- **return 의 영향**

- 어떤 특별한 상황이 되면 함수를 빠져나가고자 할 때는 return을 단독으로 써서 함수를 즉시 빠져나갈 수 있다.

```
>>> def say_nick(nick):  
...     if nick == "바보":  
...         return  
...     print("나의 별명은 %s 입니다." % nick)  
...  
>>> say_nick('야호')  
나의 별명은 야호입니다.  
>>> say_nick('바보')  
>>>
```

함수

- 매개변수에 초깃값 미리 설정하기

```
def say_myself(name, old, man=True):  
    print("나의 이름은 %s 입니다." % name)  
    print("나이는 %d살입니다." % old)  
    if man:  
        print("남자입니다.")  
    else:  
        print("여자입니다.")
```

```
>>> say_myself("홍길동", 27) # or say_myself("홍길동", 27, True)  
나의 이름은 홍길동입니다.  
나이는 27살입니다.  
남자입니다.
```

```
>>> say_myself("김영미", 27, False)  
나의 이름은 김영미입니다.  
나이는 27살입니다.  
여자입니다.
```

함수

- 매개변수에 초깃값 미리 설정하기

```
def say_myself(name, old, man=True):  
    print("나의 이름은 %s 입니다." % name)  
    print("나이는 %d살입니다." % old)  
    if man:  
        print("남자입니다.")  
    else:  
        print("여자입니다.")
```

```
>>> say_myself("홍길동", 27)  
SyntaxError: non-default argument follows default argument
```

- 초깃값을 설정해 놓은 매개변수 뒤에 초깃값을 설정해 놓지 않은 매개변수는 사용할 수 없음

함수

- 함수 안에서 함수 밖의 변수를 변경하는 방법

- 1. return 이용

```
a = 1
def vartest(a):
    a = a + 1
    return a

a = vartest(a)
print(a)
```

- 2. global 이용

```
a = 1
def vartest():
    global a # 함수 안에서 함수 밖의 a 변수를 직접 사용하겠다는 뜻
    a = a + 1

vartest()
print(a)
```

– 함수는 독립적으로 존재하는 것이 좋기 때문에 global 은 사용하지 않는 편이 좋다.

함수

- **lambda**

- 간단한 함수를 쉽게 구현할 수 있는 익명함수
- 함수를 매개변수로 전달하는 코드를 더 효율적으로 작성 가능함

lambda 매개변수1, 매개변수2, ... : 매개변수를 이용한 표현식

```
>>> sum = lambda a, b: a+b
>>> sum(3,4)
7
```

```
>>> def sum(a, b):
...     return a+b
...
>>>
```

함수

- **lambda**

```
>>> myList = [lambda a,b:a+b, lambda a,b:a*b]
>>> myList
[at 0x811eb2c>, at 0x811eb64>]
>>>
>>> myList[0]
at 0x811eb2c>
>>>
>>> myList[0](3,4)
7
>>>
>>> myList[1](3,4)
12
```

클래스

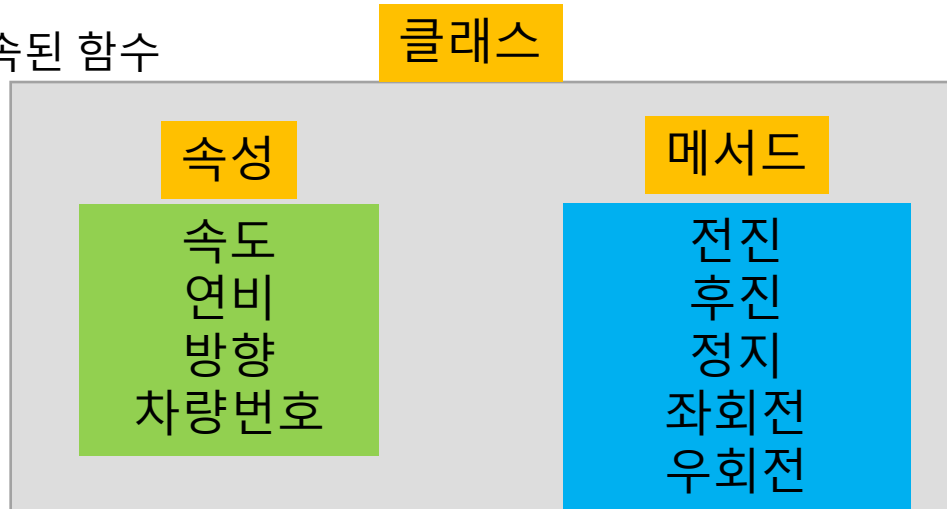
클래스

- **객체 지향 프로그래밍 (Object Oriented Programming)**
 - 객체를 우선으로 생각해서 프로그래밍하는 것
 - 클래스 기반의 객체 지향 프로그래밍 언어는 클래스를 기반으로 객체 만들고, 그러한 객체를 우선으로 생각하여 프로그래밍함
 - 로직을 상태(state)와 행위(behavior)로 이루어진 객체로 만드는 프로그래밍 기법
- **객체 (object)**
 - 여러 가지 속성 가질 수 있는 모든 대상
- **클래스 (class)**
 - 객체에 포함할 변수와 함수를 미리 정의한 것. 객체의 설계도에 해당.
- **인스턴스 (instance)**
 - 생성자 사용하여 이러한 클래스 기반으로 만들어진 객체

클래스

• 클래스란?

- 클래스는 객체(object)를 표현하기 위한 문법
- 객체지향의 가장 기본적 개념
- 관련된 속성과 동작을 하나의 범주로 묶어 실세계의 사물을 흉내냄
- 멤버
 - 클래스 구성하는 변수와 함수
- 메서드
 - 클래스에 소속된 함수



클래스

- 클래스를 사용하는 이유

- 계산기 생성 조건
 - 더하기 계산기 생성
 - 계산기는 이전에 계산한 결과값을 기억

```
result = 0

def add(num):
    global result
    result = num + result
    return result

print(add(3))
print(add(4))
```

클래스

- 클래스를 사용하는 이유

- 계산기 생성 조건
 - 더하기 계산기 생성
 - 계산기는 이전에 계산한 결과값을 기억
 - 2대의 계산기가 필요한 상황이 발생
- 각 계산기는 각각의 결과값을 유지해야 하기 때문에 위와 같이 add 함수 하나만으로는 결과값을 따로 유지할 수 불가능
- 함수 하나로는 위에 조건을 만족하는 것이 힘들기 때문에 2개의 함수를 각각 따로 만들어 함.

클래스

- 클래스를 사용하는 이유

```
result1 = 0
result2 = 0

def add1(num):
    global result1
    result1 = num + result1
    return result1

def add2(num):
    global result2
    result2 = num + result2
    return result2

print(add1(3))
print(add1(4))
print(add2(3))
print(add2(7))
```

- 똑같은 일을 하는 add1과 add2 함수를 만들었고 각 함수에서 계산한 결과값을 유지하면서 저장하는 전역 변수 result1, result2가 필요

클래스

- 클래스를 사용하는 이유

```
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result = num + self.result
        return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.add(3))
print(cal1.add(4))
print(cal2.add(3))
print(cal2.add(7))
```

클래스

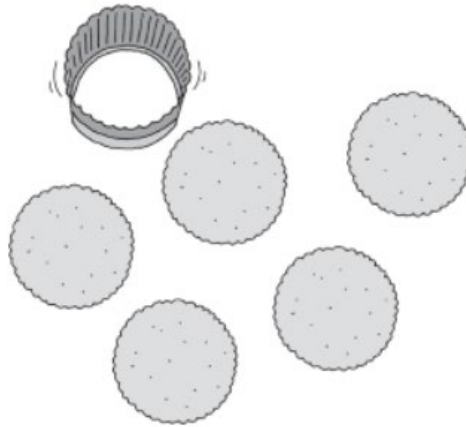
- 클래스를 사용하는 이유

- Calculator 클래스로 만든 별개의 계산기 cal1, cal2가 각각의 역할을 수행
- 계산기(cal1, cal2)의 결과값 역시 다른 계산기의 결과값과 상관없이 독립적인 값을 유지
- 클래스를 사용하면 계산기 대수가 늘어나더라도 객체를 생성만 하면 되기 때문에 함수를 사용하는 경우와 달리 매우 간단
- 빼기 기능을 더하려면 Calculator 클래스에 빼기 기능 함수를 추가

클래스

- 클래스와 객체

- 과자 틀 → 클래스 (class)
- 과자 틀에 의해서 만들어진 과자 → 객체 (object)
- 클래스로 만든 객체에는 객체마다 고유한 성격을 가짐
- 과자 틀로 만든 과자에 구멍을 뚫거나 조금 베어 먹더라도 다른 과자에는 아무 영향이 없는 것과 마찬가지로 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않음



클래스

- 클래스와 객체

- 클래스 생성

```
>>> class Cookie:  
>>>     pass
```

- 아무 기능도 갖고 있지 않은 클래스
- 이러한 클래스도 "과자 틀"로 "과자"를 만드는 것처럼 객체를 생성하는 기능이 있음
- 객체는 클래스로 만들며 1개의 클래스는 무수히 많은 객체를 만들어 낼 수 있음

```
>>> a = Cookie()  
>>> b = Cookie()
```

- Cookie()의 결과값을 돌려받은 a와 b가 바로 객체
- 함수를 사용해서 그 결과값을 돌려받는 모습과 비슷

클래스

- 클래스와 객체

- 객체와 인스턴스의 차이

- 클래스로 만든 객체를 인스턴스라고도 함

```
a = Cookie()
```

- a 는 객체
 - a 객체는 Cookie의 인스턴스
 - 즉 인스턴스라는 말은 특정 객체(a)가 어떤 클래스(Cookie)의 객체인지를 관계 위주로 설명할 때 사용

클래스

- 사칙연산 클래스 만들기

- 클래스 구조 생성

```
>>> class FourCal:  
...     pass  
...  
>>>
```

- pass는 아무것도 수행하지 않는 문법으로 임시로 코드를 작성할 때 주로 사용

클래스

- 사칙연산 클래스 만들기

- 객체에 숫자 지정할 수 있게 만들기

```
>>> class FourCal:  
...     def setdata(self, first, second):  
...         self.first = first  
...         self.second = second  
...  
>>>
```

- 앞에서 만든 FourCal 클래스에서 pass 문장을 삭제
 - setdata 함수를 생성

클래스

• 사칙연산 클래스 만들기

▪ 객체에 숫자 지정할 수 있게 만들기

– setdata 메서드

```
def setdata(self, first, second):    # ① 메서드의 매개변수
    self.first = first                # ② 메서드의 수행문
    self.second = second              # ② 메서드의 수행문
```

– ① setdata 메서드의 매개변수

- setdata 메서드는 매개변수로 self, first, second 3개 입력값을 받음
- 메서드의 첫 번째 매개변수 self는 특별한 의미를 가짐
- a 객체를 만들고 a 객체를 통해 setdata 메서드를 호출
- >>> a = FourCal()
- >>> a.setdata(4, 2)
- 객체를 통해 클래스의 메서드를 호출하려면 a.setdata(4, 2)와 같이 도트(.) 연산자를 사용

클래스

• 사칙연산 클래스 만들기

▪ 객체에 숫자 지정할 수 있게 만들기

– setdata 메서드

```
def setdata(self, first, second):    # ① 메서드의 매개변수
    self.first = first                # ② 메서드의 수행문
    self.second = second              # ② 메서드의 수행문
```

– ① setdata 메서드의 매개변수

- 파이썬 메서드의 첫 번째 매개변수 이름은 관례적으로 self를 사용
- 객체를 호출할 때 호출한 객체 자신이 전달되기 때문에 self를 사용한 것
- self 말고 다른 이름을 사용해도 상관없음
- 메서드의 첫 번째 매개변수 self를 명시적으로 구현하는 것은 파이썬만의 독특한 특징

클래스

• 사칙연산 클래스 만들기

▪ 객체에 숫자 지정할 수 있게 만들기

– setdata 메서드

```
def setdata(self, first, second):    # ① 메서드의 매개변수
    self.first = first                # ② 메서드의 수행문
    self.second = second              # ② 메서드의 수행문
```

– ① setdata 메서드의 매개변수

• 메소드 호출 방법

- >>> a = FourCal()
- >>> FourCal.setdata(a, 4, 2)
- 클래스 이름.메서드 형태로 호출할 때는 객체 a를 첫 번째 매개변수 self에 꼭 전달

- >>> a = FourCal()
- >>> a.setdata(4, 2)
- 객체.메서드 형태로 호출할 때는 self를 반드시 생략해서 호출

클래스

• 사칙연산 클래스 만들기

▪ 객체에 숫자 지정할 수 있게 만들기

– setdata 메서드

```
def setdata(self, first, second):    # ① 메서드의 매개변수
    self.first = first                # ② 메서드의 수행문
    self.second = second              # ② 메서드의 수행문
```

– ② setdata 메서드의 수행문

- a.setdata(4, 2)처럼 호출하면 setdata 메서드의 매개변수 first, second에는 각각 값 4와 2가 전달
- self.first = 4
- self.second = 2
- self는 전달된 객체 a이므로 다시 다음과 같이 해석
- a.first = 4
- a.second = 2
- a.first = 4 문장이 수행되면 a 객체에 객체변수 first가 생성되고 값 4가 저장
- a.second = 2 문장이 수행되면 a 객체에 객체변수 second가 생성되고 값 2가 저장

클래스

- 사칙연산 클래스 만들기

- 객체에 숫자 지정할 수 있게 만들기

- 객체 생성

```
>>> a = FourCal()  
>>> b = FourCal()
```

- a 객체의 객체변수 first를 생성

```
>>> a.setdata(4, 2)  
>>> print(a.first)
```

- b 객체의 객체변수 first를 생성

```
>>> b.setdata(3, 7)  
>>> print(b.first)
```

- a 객체의 first 값은 b 객체의 first 값에 영향받지 않고 원래 값을 유지하고 있음을 확인

- 클래스로 만든 객체의 객체변수는 다른 객체의 객체변수에 상관없이 독립적인 값을 유지 함

클래스

- 사칙연산 클래스 만들기

- 더하기 기능 만들기

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def add(self):
...         result = self.first + self.second
...         return result
... 
```

- 매개변수 입력

```
>>> a = FourCal()
>>> a.setdata(4, 2)
```

- add 메서드를 호출

```
>>> print(a.add())
```

클래스

- 사칙연산 클래스 만들기

- 곱하기, 빼기, 나누기 기능 만들기

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...
...     def add(self):
...         return self.first + self.second
...     def mul(self):
...         result = self.first * self.second
...     def sub(self):
...         result = self.first - self.second
...     def div(self):
...         result = self.first / self.second
...
...
>>>
```

클래스

- 생성자 (Constructor)

```
>>> a = FourCal()
>>> a.add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in add
AttributeError: 'FourCal' object has no attribute 'first'
```

- FourCal 클래스의 인스턴스 a에 setdata 메서드를 수행하지 않고 add 메서드를 수행하면 "AttributeError: 'FourCal' object has no attribute 'first'" 오류가 발생
- setdata 메서드를 수행해야 객체 a의 객체변수 first와 second가 생성되기 때문

클래스

- **생성자 (Constructor)**

- 객체에 초기값을 설정해야 할 필요가 있을 때는 setdata와 같은 메서드를 호출하여 초기값을 설정하기보다는 생성자를 구현하는 것이 안전한 방법
- 생성자(Constructor)란 객체가 생성될 때 자동으로 호출되는 메서드를 의미
- 파이썬 메서드 이름으로 `__init__`를 사용하면 이 메서드는 생성자가 됨
- `__init__` 메서드의 `init` 앞뒤로 붙은 `__`는 언더스코어(_) 두 개를 붙여 쓴 것

클래스

- 생성자 (Constructor)

```
def __init__(self):  
    self.first = first  
    self.second = second
```

- `__init__` 메서드는 `setdata` 메서드와 이름만 다르고 모든 게 동일
- 메서드 이름을 `__init__`으로 했기 때문에 생성자로 인식되어 객체가 생성되는 시점에 자동으로 호출되는 차이

클래스

- 생성자 (Constructor)

```
>>> a = FourCal()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: __init__() missing 2 required positional arguments:  
'first' and 'second'
```

- `a = FourCal()`을 수행할 때 생성자 `__init__`이 호출되어 위와 같은 오류가 발생
- 오류가 발생한 이유는 생성자의 매개변수 `first`와 `second`에 해당하는 값이 전달되지 않았기 때문

클래스

- 생성자 (Constructor)

```
>>> a = FourCal(4, 2)
>>>
```

- 위와 같이 수행하면 `__init__` 메서드의 매개변수에는 각각 오른쪽과 같은 값이 대입

매개변수	값
self	생성되는 객체
first	4
second	2

- `__init__` 메서드도 다른 메서드와 마찬가지로 첫 번째 매개변수 `self`에 생성되는 객체가 자동으로 전달된다는 점을 기억

클래스

- 클래스의 상속

- 상속

- 부모 클래스의 모든 속성(데이터, 메소드)를 자식 클래스로 물려줌
 - 클래스의 공통된 속성을 부모 클래스에 정의
 - 하위 클래스에서는 특화된 메소드와 데이터를 정의

- 장점

- 각 클래스마다 동일한 코드가 작성되는 것을 방지
 - 부모 클래스에 공통된 속성을 두어 코드의 유지보수가 용이
 - 각 개별 클래스에 특화된 기능을 공통된 인터페이스로 접근 가능

- 상속을 사용하여 우리가 만든 FourCal 클래스에 a^b 을 구할 수 있는 기능을 추가

- FourCal 클래스를 상속하는 MoreFourCal 클래스 생성

```
>>> class MoreFourCal(FourCal):  
...     pass  
...  
>>>
```

클래스

• 클래스의 상속

- 상속을 사용하여 우리가 만든 FourCal 클래스에 a^b 을 구할 수 있는 기능을 추가
- FourCal 클래스를 상속하는 MoreFourCal 클래스 생성
 - 클래스를 상속하기 위해서는 다음처럼 클래스 이름 뒤 괄호 안에 상속할 클래스 이름을 넣어주면 됨

```
class 클래스 이름(상속할 클래스 이름)
```

- MoreFourCal 클래스는 FourCal 클래스를 상속했으므로 FourCal 클래스의 모든 기능을 사용할 수 있음

```
>>> a = MoreFourCal(4, 2)
>>> a.add()
>>> a.mul()
>>> a.sub()
>>> a.div()
```

클래스

- 클래스의 상속

- 상속은 기존 클래스를 변경하지 않고 기능을 추가하거나 기존 기능을 변경하려고 할 때 사용
- a의 b제곱을 계산하는 MoreFourCal 클래스 생성

```
>>> class MoreFourCal(FourCal):  
...     def pow(self):  
...         result = self.first ** self.second  
...         return result  
...  
>>>
```

- pass 문장은 삭제하고 위와 같이 두 수의 거듭제곱을 구할 수 있는 pow 메서드를 추가

클래스

• 메서드 오버라이딩

```
>>> a = FourCal(4, 0)
>>> a.div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    result = self.first / self.second
ZeroDivisionError: division by zero
```

- FourCal 클래스의 객체 a에 4와 0 값을 설정하고 div 메서드를 호출하면 4를 0으로 나누려고 하기 때문에 위와 같은 ZeroDivisionError 오류가 발생
- 0으로 나눌 때 오류가 아닌 0을 돌려주도록 만들고 싶다면?

클래스

• 메서드 오버라이딩

- FourCal 클래스를 상속하는 SafeFourCal 클래스 생성

```
>>> class SafeFourCal(FourCal):  
...     def div(self):  
...         if self.second == 0: # 나누는 값이 0인 경우 0을 리턴하도록 수정  
...             return 0  
...         else:  
...             return self.first / self.second  
...  
>>>
```

- SafeFourCal 클래스는 FourCal 클래스에 있는 div 메서드를 동일한 이름으로 다시 작성
- 부모 클래스(상속한 클래스)에 있는 메서드를 동일한 이름으로 다시 만드는 것을 메서드 오버라이딩(Overriding, 덮어쓰기)이라고 함
- 이렇게 메서드를 오버라이딩하면 부모클래스의 메서드 대신 오버라이딩한 메서드가 호출

모 들

모듈

- **모듈 (Module)**

- 함수, 변수, 클래스들을 모아놓고, 서로 다른 파이썬 프로그램에서 호출하여 사용할 수 있도록 설계된 파이썬 라이브러리
- 리눅스의 패키지 관리자처럼 관리할 수 있는 pip, easy_install 과 같은 모듈 관리자를 제공함으로써 필요한 모듈을 검색하고, 설치하는 것이 용이
- 모듈은 개별 파이썬 스크립트에서 임포트 시켜야 함
- 모듈의 장점
 - 코드 재사용성이 높아짐
 - 모듈만 수정하면 프로그램의 동작 로직 자체를 재설계 하는 효과
 - 메모리 절약 효과 (필요한 모듈만 import 가능)
 - 사용자가 필요에 따라 모듈을 만들고 파일로 저장해서 배포 가능

모듈

• 모듈 사용

▪ 모듈 생성

– # mod1.py

```
def add(a, b):  
    return a + b
```

```
def sub(a, b):  
    return a - b
```

– add와 sub 함수만 있는 파일 mod1.py를 생성

– 파이썬 확장자 .py로 만든 파이썬 파일은 모두 모듈

▪ 모듈 불러오기

– 명령 프롬프트 창을 열고 mod1.py를 저장한 디렉터리(C:\Users\user\python\mymod)로 이동 후 대화형 인터프리터를 실행

– 반드시 모듈을 저장한 디렉터리로 이동한 다음 진행해야 대화형 인터프리터에서 모듈을 읽을 수 있음

모듈

• 모듈 사용

▪ 모듈 불러오기

- import 모듈이름

```
import math                # 모듈 импорт  
math.sin()
```

- import 모듈이름 as 모듈_식별자

```
import math as mt          # 새로운 식별자로 모듈 импорт  
mt.sin()
```

- from 모듈이름 import 함수

```
from math import sin, pi   # 모듈에서 지정함수만 импорт(모듈이름 생략)  
sin()
```

- from 모듈이름 import *

```
from math import *         # 모듈에서 모든함수를 импорт(모듈이름 생략)  
sin()
```

모듈

• 모듈 사용

- if `__name__ == "__main__"`: 의 의미

– # mod1.py

```
def add(a, b):  
    return a+b  
  
def sub(a, b):  
    return a-b  
  
print(add(1, 4))  
print(sub(4, 2))
```

– add(1, 4)와 sub(4, 2)의 결과를 출력하는 다음 문장을 추가

```
C:\Users\user\python> python mod1.py  
5  
2
```

모듈

- 모듈 사용

- if `__name__ == "__main__"`: 의 의미

```
C:\Users\user\python> python
Type "help", "copyright", "credits" or "license" for more
information.
>>> import mod1
5
2
```

- import mod1을 수행하는 순간 mod1.py가 실행이 되어 결과값을 출력

모듈

• 모듈 사용

- if `__name__ == "__main__"`: 의 의미

– # mod1.py

```
def add(a, b):  
    return a+b
```

```
def sub(a, b):  
    return a-b
```

```
if __name__ == "__main__":  
    print(add(1, 4))  
    print(sub(4, 2))
```

- if `__name__ == "__main__"`을 사용하면 C:\Users\user\python>python mod1.py처럼 직접 이 파일을 실행했을 때는 `__name__ == "__main__"`이 참이 되어 if문 다음 문장이 수행
- 반대로 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 사용할 때는 `__name__ == "__main__"`이 거짓이 되어 if문 다음 문장이 수행되지 않음

모듈

- 모듈 사용

- 클래스나 변수 등을 포함한 모듈

- 모듈은 함수 뿐만 아니라 클래스나 변수 등을 포함 가능

- # mod2.py

```
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

def add(a, b):
    return a+b
```

- 이 파일은 원의 넓이를 계산하는 Math 클래스와 두 값을 더하는 add 함수 그리고 원주율 값에 해당되는 PI 변수처럼 클래스, 함수, 변수 등을 모두 포함

모듈

• 모듈 사용

▪ 클래스나 변수 등을 포함한 모듈

```
>>> import mod2  
>>> print(mod2.PI)  
3.141592
```

- 모듈이름.모듈변수 처럼 입력해서 모듈 파일에 있는 변수 값 사용 가능

```
>>> a = mod2.Math()  
>>> print(a.solv(2))  
12.566368
```

- 모듈에 있는 클래스를 사용하려면 "."(도트 연산자)로 클래스 이름 앞에 모듈 이름을 먼저 입력

```
>>> print(mod2.add(mod2.PI, 4.4))  
7.541592
```

- 모듈에 있는 함수 사용 가능

모듈

• 모듈 사용

- 모듈을 저장한 디렉터리로 이동하지 않고 모듈을 불러와서 사용하는 방법
 - 방법1 : `sys.path.append(모듈을 저장한 디렉터리)` 사용하기
 - 방법2 : `PYTHONPATH` 환경 변수 사용하기

```
C:\Users\user\python>
```

```
C:\Users\user\python> mkdir mymod
```

```
C:\Users\user\python> move mod2.py mymod
```

- `mod2.py` 파일을 `C:\Users\user\python\mymod`로 이동

모듈

• 모듈 사용

- 모듈을 저장한 디렉터리로 이동하지 않고 모듈을 불러와서 사용하는 방법

- sys.path.append(모듈을 저장한 디렉터리) 사용하기

```
C:\Users\user\python\mymod> python
>>> import sys
```

- sys 모듈은 파이썬을 설치할 때 함께 설치되는 라이브러리 모듈
- sys 모듈을 사용하면 파이썬 라이브러리가 설치되어 있는 디렉터리를 확인 가능

```
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python37.zip', 'c:\\Python37\\DLLs',
'c:\\Python37\\lib', 'c:\\Python37', 'c:\\Python37\\lib\\site-packages']
```

- sys.path는 파이썬 라이브러리가 설치되어 있는 디렉터리를 보여 줌
- 파이썬 모듈이 위 디렉터리에 들어 있다면 모듈이 저장된 디렉터리로 이동할 필요 없이 바로 불러서 사용 가능

모듈

• 모듈 사용

- 모듈을 저장한 디렉터리로 이동하지 않고 모듈을 불러와서 사용하는 방법

- sys.path.append(모듈을 저장한 디렉터리) 사용하기

```
>>> sys.path.append("C:/Users/user/python/mymod")
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python37.zip', 'c:\\Python37\\DLLs',
'c:\\Python37\\lib', 'c:\\Python37', 'c:\\Python37\\lib\\site-packages',
'C:/Users/user/python/mymod']
```

- sys.path.append를 사용해서 C:/Users/user/python/mymod 라는 디렉터리를 sys.path에 추가
- 다시 sys.path를 보면 가장 마지막 요소에 C:/Users/user/python/mymod 라고 추가된 것을 확인

모듈

• 모듈 사용

- 모듈을 저장한 디렉터리로 이동하지 않고 모듈을 불러와서 사용하는 방법

- PYTHONPATH 환경 변수 사용하기

```
C:\> set PYTHONPATH=C:/Users/user/python/mymod
C:\> python
>>> import mod2
>>> print(mod2.add(3,4))
7
```

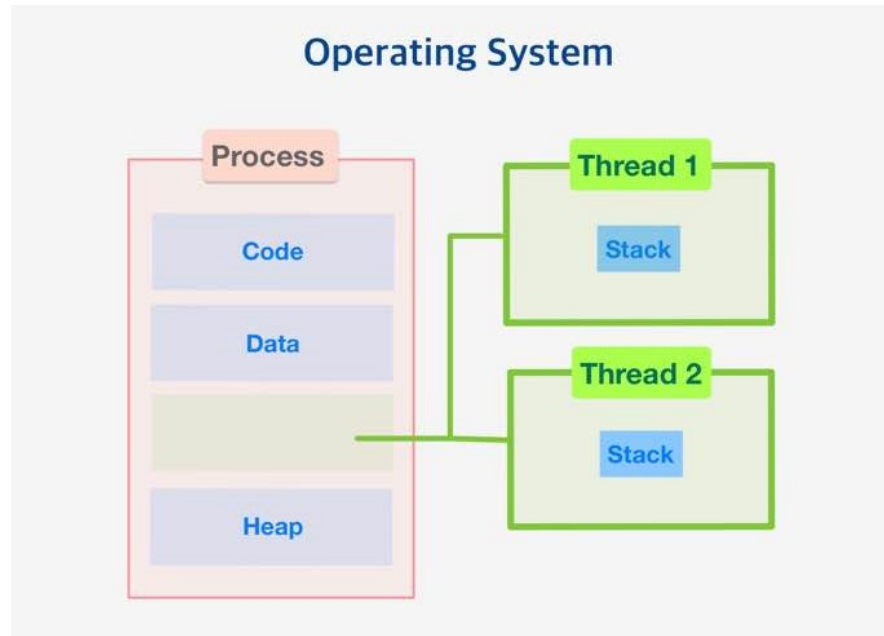
- set 명령어를 사용해 PYTHONPATH 환경 변수에 mod2.py 파일이 있는 C:\Users\user\python\mymod 디렉터를 설정
- 디렉터리 이동이나 별도의 모듈 추가 작업 없이 mod2 모듈을 불러와서 사용 가능

Multi Threading

모듈

• 멀티스레딩/멀티프로세싱

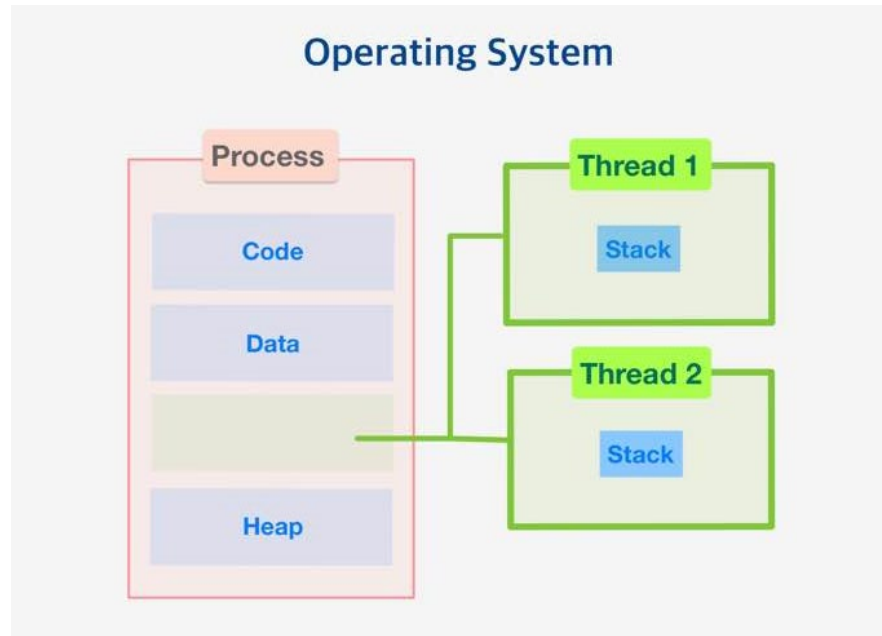
- 스레드(thread)는 프로세스가 할당 받은 자원을 이용하는 실행 단위
- 스레드는 프로세스 내에서 프로세스의 자원을 이용해서 실제로 작업을 수행
 - 하나의 프로세스 안에 있는 스레드들은 각각 독립적으로 스택을 가지고 실행되지만, 코드와 데이터는 공유한다



모듈

- Threading 모듈

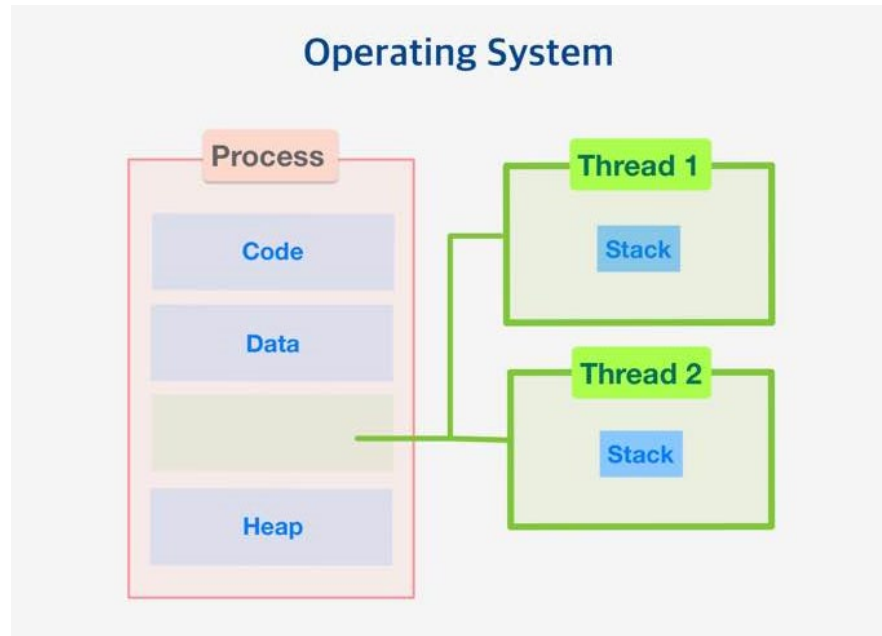
- 스레드는 프로세스 내에서 프로세스의 자원을 이용해서 실제로 작업을 수행
 - 하나의 프로세스 안에 있는 스레드들은 각각 독립적으로 스택을 가지고 실행되지만, 코드와 데이터는 공유한다



모듈

- Threading 모듈

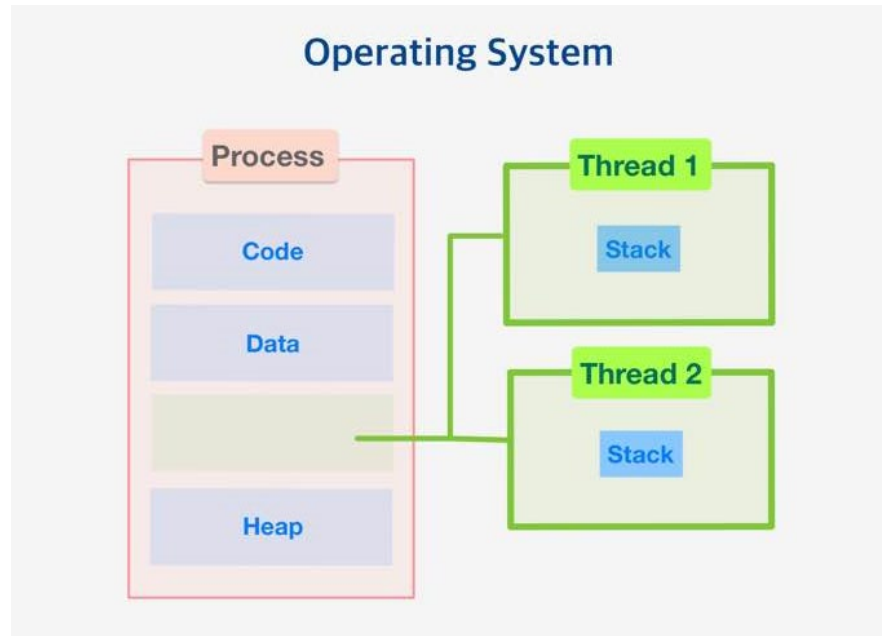
- 스레드는 프로세스 내에서 프로세스의 자원을 이용해서 실제로 작업을 수행
 - 하나의 프로세스 안에 있는 스레드들은 각각 독립적으로 스택을 가지고 실행되지만, 코드와 데이터는 공유한다



모듈

- Threading 모듈

- 스레드는 프로세스 내에서 프로세스의 자원을 이용해서 실제로 작업을 수행
 - 하나의 프로세스 안에 있는 스레드들은 각각 독립적으로 스택을 가지고 실행되지만, 코드와 데이터는 공유한다



socket

Socket

- **소켓이란?**

- 서로 떨어진 두 대의 컴퓨터 사이에서 TCP/IP 네트워크를 통해 상호 통신이 가능하도록 운영체제에서 해당 자원을 할당/처리 해주는 방식
- 컴퓨터간에 데이터를 주고받기 위한 프로세스 처리 방식
 - 서버 : 서비스를 제공하는 컴퓨터
 - 클라이언트 : 서비스를 요청하는 컴퓨터

Socket

- **서버 클라이언트 모델**

- 소켓은 TCP/IP 프로토콜의 프로그래머 인터페이스 이다. 소켓을 경유한 프로세스 간의 통신은 클라이언트/서버 모델에 기초한다.
- 서버 프로세스로 알려진 프로세스는 해당 컴퓨터에서 유일하게 할당된 번호의 소켓을 만든다.
- 클라이언트 프로세스는 해당 번호의 소켓을 통해서 서버 프로세스와 대화를 할수 있다.
- 연결에 성공하면 서버와 클라이언트 모두에게 각각 하나의 소켓 기술자가 반환되는데 이것으로 읽기와 쓰기를 한다.
- 소켓은 양방향 통신을 한다.

Socket

• 포트번호

- 물리적인 전송선은 하나지만 그것을 여러개의 응용 프로그램이 나누어 쓰기 위해서는 포트라는 것을 만들었다.
- 한 컴퓨터 내의(소켓을 사용하는) 모든 서버 프로세스는 별도의 포트 번호가 부여된 소켓을 가지고 있다. 이것은 TCP/IP가 지원하는 상위 계층의 응용 프로그램을 구분하기 위한 번호.
- 파이썬으로 서비스의 포트번호를 확인하려면 다음 함수를 사용한다.

```
import socket
socket.getservbyname('http','tcp')
socket.getservbyname('ftp','tcp')
```

Socket

- **socket 모듈을 이용해 소켓 객체를 생성하는 방법**
 - family : 소켓에서 사용할 주소 형식
 - type : 서버와 클라이언트 사이에서 사용하는 전송 유형 설정
 - protocols : 소켓 타입을 세부적으로 구분할때 사용(생략 가능)

```
s = socket.socket(family, type, protocols)
```

Socket

- 소켓의 종류

- family

- 소켓은 도메인(Domain)과 유형(Type)에 따라 분류할 수 있다.
- 도메인은 서버와 클라이언트의 소켓이 있는 장소를 가리킨다.
- AF는 Address Family 의 약자이다.
 - AF_INET : IPv4 소켓. : 클라이언트와 서버는 다른 기계에 있을수 있다. 주소 표현을 위해 (host,port) 튜플이 사용됨.
 - AF_INET6 : IPv6 소켓이다. : socket.getservbyname('http','tcp') 주소 표현을 위해 (host, port, flowinfo, scopeid) 튜플이 사용됨.
 - AF_UNIX :유닉스 도메인 소켓이다.
 - AF_TIPC : 리눅스 전용 프로토콜
 - AF_BLUETOOTH : 블루투스 프로토콜
 - AF_PACKET : 링크 수준 패킷

Socket

- 소켓의 종류

- Type

- 소켓 유형(Type)은 서버와 클라이언트 사이에 있을 수 있는 통신 유형이다.
- SOCK_STREAM과 SOCK_DGRAM 이 가장 일반적으로 사용된다.
 - SOCK_STREAM : TCP 통신 소켓 유형. 신뢰성 있는 스트림 방식의 소켓을 만든다. (TCP)일련 번호가 붙으며, 양방향 연결에 기초한 바이트 가변 길이의 스트림 데이터
 - SOCK_DGRAM : UDP 통신 소켓 유형. 데이터 그램 방식의 소켓을 만든다. 정보와 비슷한 비 연결, 비신뢰적인 고정 길이의 메시지를 사용
 - SOCK_RAW : 무가공 소켓
 - SOCK_RDM : 신뢰성 있는 데이터 그램
 - SOCK_SEQPACKET : 순서를 갖는 연결 모드로 레코드 전송에 사용

Socket

- **UDP 소켓 프로그래밍**

- UDP는 비연결형 프로토콜이므로 서버와 클라이언트에서 연결 요청과 연결 접수를 수행할 필요가 없다. 서버는 사용하려는 포트번호로만 소켓에 묶으면 되고, 클라이언트도 자신이 사용하려는 포트 번호를 소켓에 묶으면 된다.
- protocols : 소켓 타입을 세부적으로 구분할때 사용(생략 가능)
- 다음과 같은 순서로 소켓 생성
 1. socket() 객체 생성
 2. IP 주소 설정
 3. 포트 번호 설정
 4. bind()함수를 이용해 IP주소와 포트 번호 연동 (# 클라이언트에서는 bind()함수가 필요없다.)
 5. 소켓 서버 동작 구현
 6. close() 함수를 이용해 소켓 객체 종료

Socket

- **TCP 소켓 프로그래밍**

- 먼저 서버와 클라이언트가 TCP 소켓을 만들고 서로 연결한 다음 데이터를 송수신하고 소켓을 종료하는 TCP 절차에 대해 알아보자.
- 다음과 같은 순서로 소켓 생성
 1. socket() 객체 생성
 2. IP 주소 설정
 3. 포트 번호 설정
 4. bind() 함수를 이용해 IP 주소와 포트 번호 연동 (# 클라이언트에서는 bind() 함수가 필요없다.)
 - 5. listen() 함수를 통해 처리할 수 있는 연결 수를 설정**
 6. 소켓 서버 동작 구현
 7. close() 함수를 이용해 소켓 객체 종료

Socket

- 소켓을 이용한 다양한 프로그래밍
 - 채팅 프로그램 만들기
 - 포트스캐너 제작

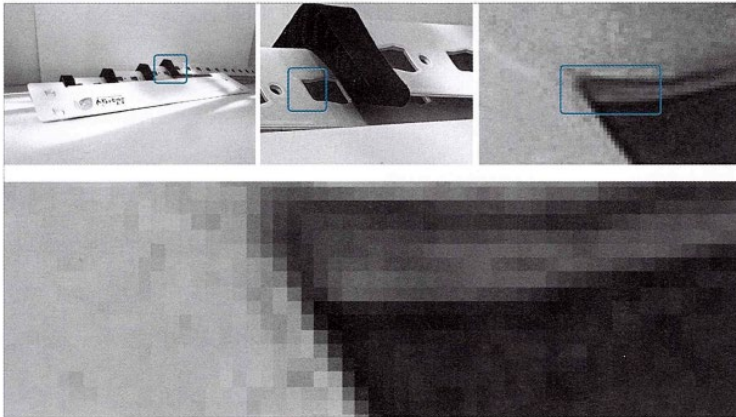
이미지 가공

Socket

- pillow

- pip install pillow

- 이미지의 구조



- RGB 구조로 이루어져 있다



Socket

- **PIL**

- `img = Image.open("경로")`
- `Image.new()`
- `object.resize`
- `object.crop`
- `object.transpose(Image.FLIP_LEFT_RIGHT)`
- `object.transpose(Image.FLIP_TOP_BOTTOM)`
- `object.filter(ImageFilter.BLUR)`
- `object.save()`

- <https://ddolcat.tistory.com/690>