파이썬을 데이터분석
- 파이썬 기초

강사 : KAIST 김동훈

Ⅰ. 파이썬 기초

- 파이썬 이란
- 자료형
- 제어문
- 함수
- 평균 구하기

Ⅰ. 파이썬 기초 - 파이썬 이란?

1. Anaconda 설치
https://www.anaconda.com

설치방법 동영상 참조: https://youtu.be/fce61TnAUMs

2. print("Hello World!")

3. Python 이란?
- 파이썬(Python)은 1990년 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 개발한 인터프리터 언어이다. 귀도는 파이썬이라는 이름을 자신이 좋아하는 코미디 쇼인 "몬티 파이썬의 날아다니는 서커스(Monty Python's Flying Circus)"에서 따왔다고 한다.

- 파이썬의 사전적 의미는 '고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀'을 뜻하며, 아폴로 신이 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있다.

자료출처 : https://wikidocs.net/4307

Ⅰ. 파이썬 기초 - 파이썬 이란?

4. Python 의 특징

1) 파이썬은 인간다운 언어이다
   ㄴ 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어이다.
2) 파이썬은 **문법이 쉬워** 빠르게 배울 수 있다
   ㄴ 배우기 쉬운 언어, 활용하기 쉬운 언어
3) 파이썬은 무료이지만 강력하다
   ㄴ 파이썬과 C는 찰떡궁합
4) 파이썬은 간결하다
   ㄴ 간결함의 철학은 파이썬 문법 그대로 적용
5) 파이썬은 프로그래밍을 즐기게 해준다
6) 파이썬은 개발 속도가 빠르다

Ⅰ. 파이썬 기초 - 파이썬 이란?

5. 프로그램이 사용되는 곳

1) 형태에 따른 분류
- 윈도우 어플리케이션, ex) PowerPoint, Excel, Photoshop
- 웹 어플리케이션, ex) Facebook, Google, Coupang, Youtube
- 모바일 어플리케이션, ex) iOS, Android, ...
- 시스템 프로그래밍, ex) 리눅스 커널

2) 기능에 따른 분류
- 생산성 : 스프레드시트, 문서작업, 삽화, PDF, ...
- Social Media : 인스타, 채팅, 동영상, ...
- 과학, 수학 : 우주선 발사, 통계, 공학용 소프트웨어, ...
- 경제 : 주식 거래 및 차트, 금융, 비트코인, ...
- 놀이 : 게임, VR, ...
- 지식 : Wikipedia, NamuWiki, TED, Coursera, ...
- 경영 : MES, ERP, CRM, PLM, YMS, ...
- 병원 : MRI, CT, 진단 및 처방

Ⅰ. 파이썬 기초 - 파이썬 이란?

6. 프로그래밍 언어들
- 사람이 접근하기 쉬운가의 여부

저급 언어 (Low-Level) : 기계어, 어셈블리어
ㄴ 하드웨어 디바이스 드라이버, 일반 프로그램의 기능 최적화

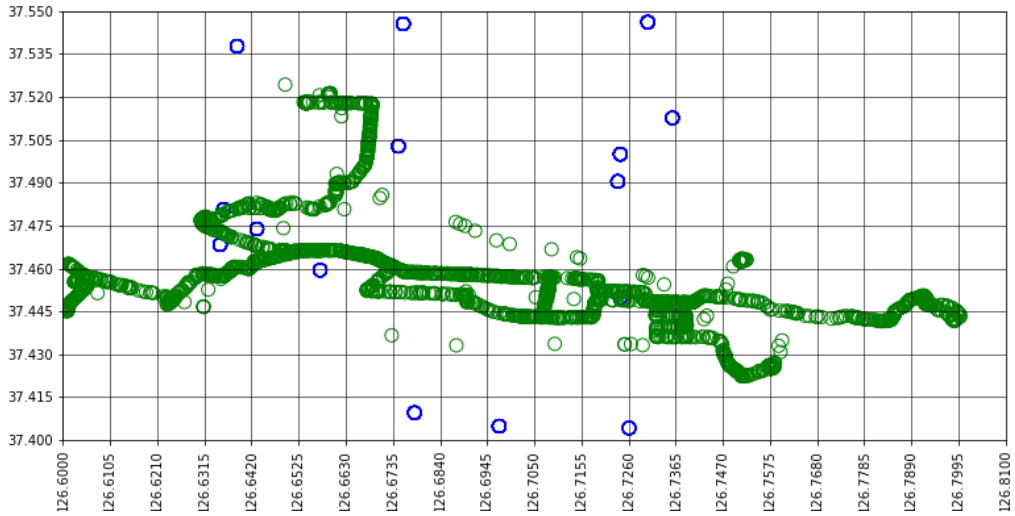
고급 언어 (High-Level) : C, C++, C#, Java, JavaScript, Python, Perl, ...
ㄴ 고급언어를 컴퓨터 언어인 기계어로 변환하기 위해 컴파일러와 인터프리터 등이 있다.

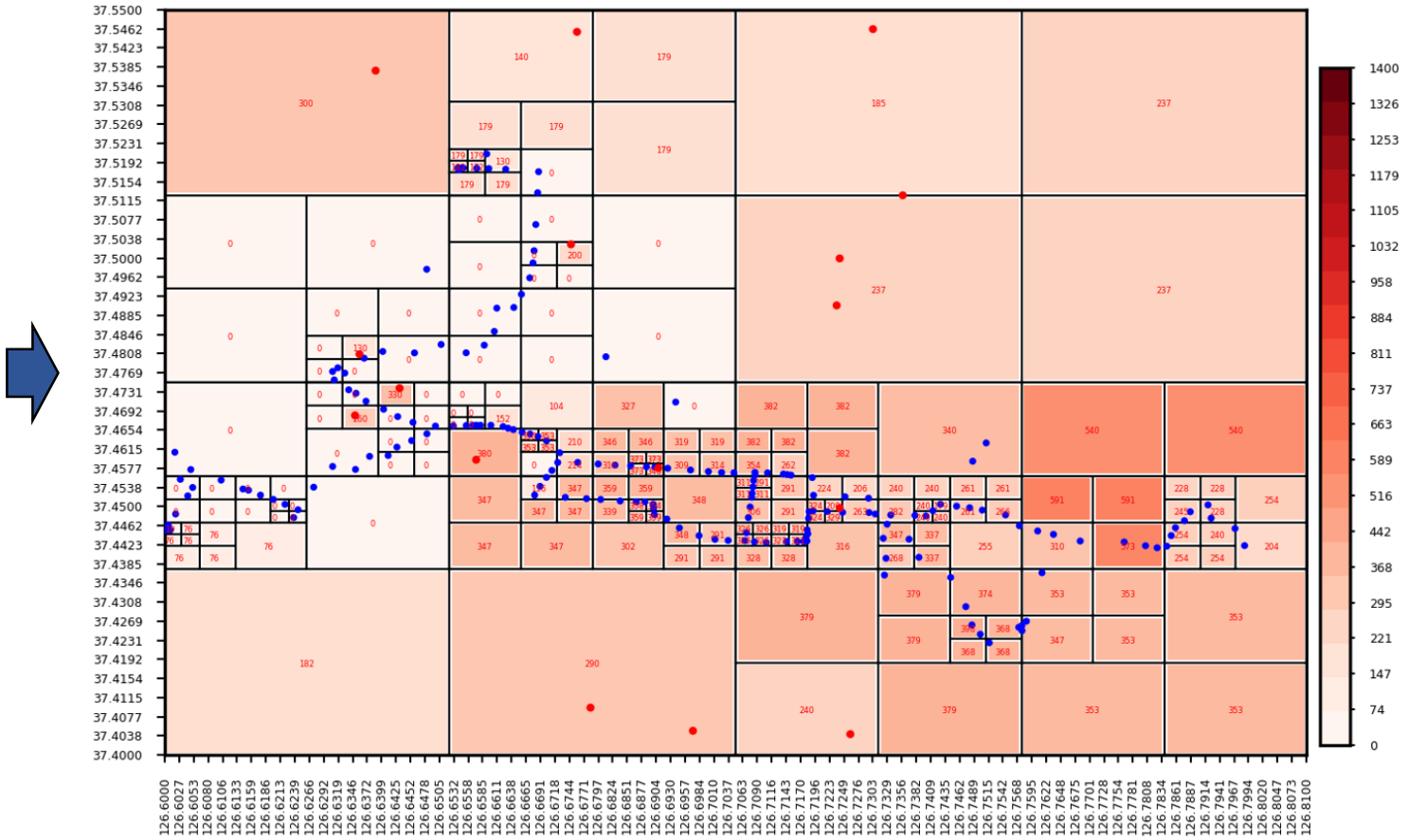
Ⅰ. 파이썬 기초 - 파이썬 이란?

7. 프로그래밍 예시

1) 김연아 타자 프로그램 (C#), https://youtu.be/4WaVEtnEdD4

2) 미세먼지 예측(Python),





Ⅰ. 파이썬 기초 - 파이썬 이란?

7. 프로그래밍 예시

3) TexasHoldem(Python), https://youtu.be/3aDdwXxidBU

Ⅰ. 파이썬 기초 - 평균 구하기

철수의 기말고사 성적

국어	86
영어	70
수학	90

단계

1) (국어 + 영어 + 수학) /3

2) avg 변수에 할당

3) 국어, 영어, 수학도 변수에 담기

4) calculate_average 함수 사용

Ⅰ. 파이썬 기초 - 자료형

1. 숫자형 (Numeric)

1) 정수형 (Integer) : 123, 0, -345

① 8진수 (Octal) : 0o34, 0o25
② 16진수 (Hexadecimal) : 0x2A, 0xFF

※ 컴퓨터는 0과 1로만 이루어져 있다.
010010 … 01111 …

Ⅰ. 파이썬 기초 - 자료형

1. 숫자형 (Numeric)

2) 실수형 (Float) : 123.45, -1234.5, 3.4e10 (지수 표현방식)

3) 숫자형을 활용하기 위한 연산자

- 사칙연산 : +, -, *, /
- **** (power of)**, ex) 2 ** 3 = 2^3 = 8
- **% (나머지)**, ex) 7%2 = 1 (Modulus)
- // (몫), eX) 7 // 2 = 3

※ Python 3.x 부터 '정수 나누기(/) 정수' 연산은 실수가 됨. Python 2.x 에서는 Floor 연산 적용

Ⅰ. 파이썬 기초 - 자료형

2. 시퀀스(Sequence) : 순서있는 컬렉션

1) 문자열 (String)

- "hello", 'hello',
  "You are genius",
  "영희가 학교에 간다. ₩n 철수도 간다."
- escape characters :
  ₩n : 줄 바꿈
  ₩t : 탭(tab)

2) List : 파이썬에서 가장 많이 쓰이는 자료구조

["Jeff", "Tom", "Steve" , "James",]
[1, 2, 3, 4, 5]
[1, "Jeff", 75.5,]

3) Tuple : 순서쌍 형태

(10, 20, 30, 40, 50)
("Hi", "Hello", "You", "Are", "a", "Boy")
(70, "Computer", "Banana", 66.5)

Ⅰ. 파이썬 기초 - 자료형

3. 불린 (Boolean, 참/거짓)

- True / False

4. 기타

1) Set : 집합 형태의 Collections

2) Dictionary : Key, Value 형태의 Collections

5. type() : 어떤 자료형인지 확인하고 싶을 때 사용

ex)

```
type(100)
type("Hello")
type(75.5)
type( [ 65.5, 405, 302, 133.5 ] )
```

Ⅰ. 파이썬 기초 - 제어문

1. IF, ELSE 문

1) if, else

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

2) 조건문이란?
- 참(True)와 거짓(False)를 판단하는 문장

Ⅰ. 파이썬 기초 - 제어문

1. IF, ELSE 문

3) 비교연산자

비교연산자	설명
x < y	x가 y보다 작다
x > y	x가 y보다 크다
x == y	x와 y가 같다
x != y	x와 y가 같지 않다
x >= y	x가 y보다 크거나 같다
x <= y	x가 y보다 작거나 같다

4) and, or, not

연산자	설명
x or y	x와 y둘중에 하나만 참이어도 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

Ⅰ. 파이썬 기초 - 제어문

1. IF, ELSE 문

5) elif 문

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

Ⅰ. 파이썬 기초 - 제어문

2. while 문

- 반복해서 수행해야 할 경우 사용 (Loop)

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

ex) 1 에서 10까지 숫자를 출력해 보자

무한루프 (Infinite Loop) 조심!

Ⅰ. 파이썬 기초 - 제어문

3. for 문

- 반복문 (Loop)
- **파이썬에서 가장 많이 사용하는 대표적인 반복문**

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
    ...
```

ex)

```
a_list = [10, 20, 30, 40, 50]

for x in a_list:
    print(x)
```

Ⅰ. 파이썬 기초 - 제어문

for 문의 2가지 형태

① range 구문으로 데이터를 생성해서 순환

```
sum = 0

for i in range(1,11):
    sum = sum + i
```

② list 등 여러 원소를 가지는 변수를 순환

```
aList = [70, 80, 85, 88, 75, 92, 89]
sum = 0

for i in aList:
    sum = sum + i
```

Ⅰ. 파이썬 기초 - 제어문

함수 안에서 사용 예)

```
def calc_height(hList):
    sum = 0
    max = 0

    for i in hList:
        sum = sum + i
        if i>max:
            max = i

    avg = sum/len(hList)
    return avg, max


hList = [ 180, 185, 190, 192, 193, 170, 172, 173, ]
avg, max = calc_height(hList)
print("주성이 반 키 Stat: Avg.:{} Max:{}".format(avg, max))
```

Ⅰ. 파이썬 기초 - 제어문2

1. **for 문과 함께 자주 사용하는 range 함수**

- 기본구문
  range(시작 숫자, 끝 숫자, 간격)

※ 끝 숫자는 포함하지 않는다.

- ex)
  range(10) : 0부터 9까지의 range 객체 -> 0,1,2,3,4,5,6,7,8,9
  range(1, 11) : 1부터 10까지의 range 객체 -> 1,2,3,4,5,6,7,8,9,10
  range(1,11,1) : 1부터 10까지의 range 객체 (interval = 1) -> 1,2,3,4,5,6,7,8,9,10
  range(1,11,2) : 1부터 10까지의 range 객체 (interval = 2) -> 1,3,5,7,9

Ⅰ. 파이썬 기초 - 제어문2

1. **for 문과 함께 자주 사용하는 range 함수**

for 문과 range 의 결합, 예) 1~10까지의 숫자 더하기

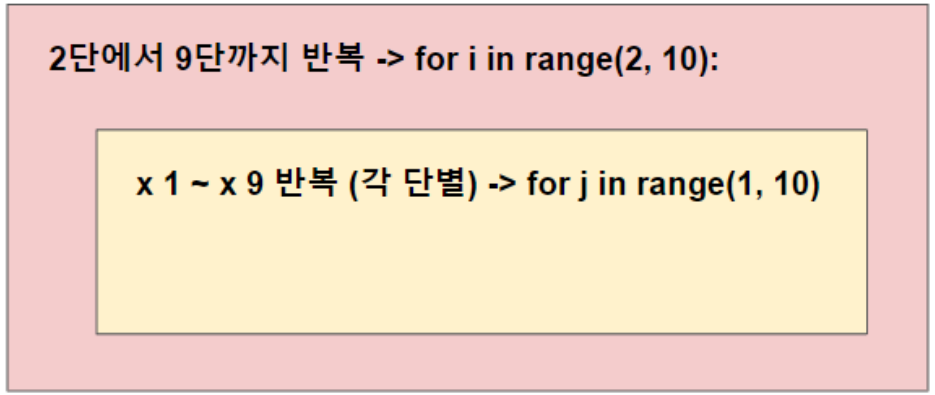
```
sum = 0
for i in range(1, 11):
    sum = sum + i

print(sum)
```

Ⅰ. 파이썬 기초 - 제어문2

2. 중첩 Loop (Nested Loop) 를 사용한 구구단 만들기

- 중첩 Loop 란?
ㄴ Loop 안에 Loop 가 반복되는 구조

- 구구단 만들기



- 코드1

```
for i in range(2, 10):
    for j in range(1, 10):
        print(i, " x ", j, " = ", i*j)
```

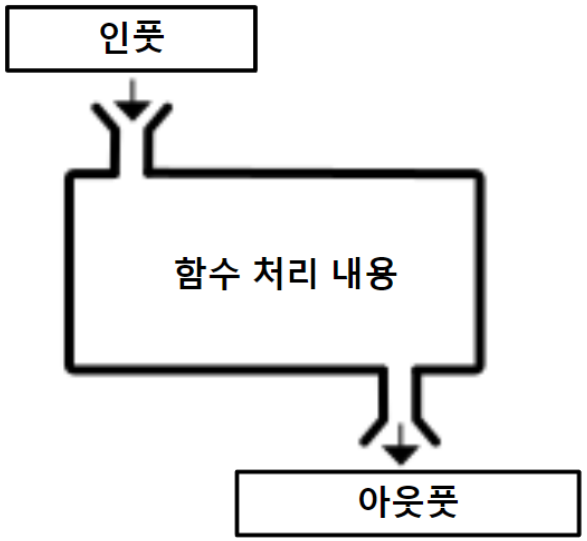
- 코드2 각 단은 같은 줄(행)에 출력하고,
단이 바뀌면 다음줄에 출력하도록 바꾸자.

- 코드3 문자열의 format 함수를 써서 꾸며보자

Ⅰ. 파이썬 기초 - 함수

1. 함수

- 인풋이 들어와 아웃풋을 출력해 내는 구조
- 인풋, 함수 처리 내용, 아웃풋(return) 으로 구성



- 인풋 : '인자' 혹은 '인수', 영어로는 'arguments'
- 아웃풋 : 리턴 (Return)
- 함수 처리 내용 : 인풋을 활용하여 아웃풋을 만들어 내는 과정

Ⅰ. 파이썬 기초 - 함수

함수의 예시)

```
def calculate_avg(score_list):
    if len(score_list) < 1:
        print("no avg result")
        return -1
    sum = 0
    for score in score_list:
        sum = sum + score
    avg = sum / len(score_list)
    return avg

a_score_list = [ 75, 85, 90, 100, 95 ]
avg = calculate_avg(a_score_list)
print("평균 성적: ", avg)
```

Ⅰ. 파이썬 기초 - 함수

3. 함수 변수 Scope
- 함수의 인자는 함수 내부에서만 사용된다.

ex)

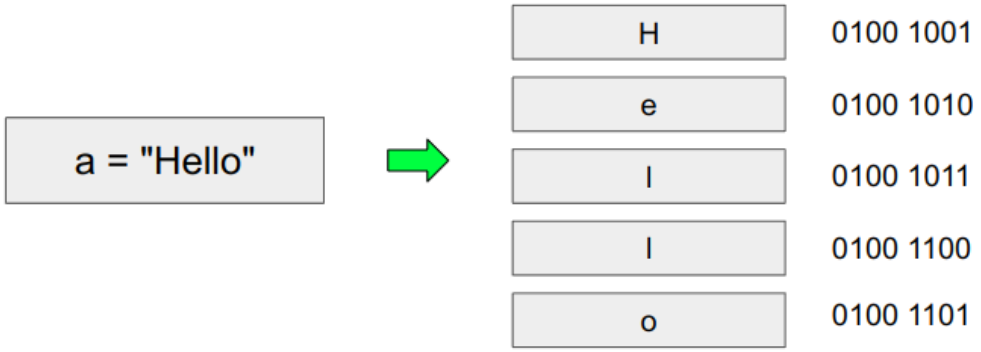
```
def a_func(a,b,c):
    a = 10
    b = 20
    c = 30
    print(a, b, c)

a = 25
b = 35
c = 45

a_func(a,b,c)
print(a,b,c)
```

Ⅱ. 문자열과 콘솔 입출력

- 문자열의 이해
- 콘솔 입출력

Ⅱ. 문자열과 콘솔 입출력

1. 문자열은 문자들의 배열 (array of characters)



2. 문자열의 인덱싱

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> a = "Hello"
>>> print(a[0], a[4])
H o
>>> print(a[-1], a[-5])
o H
```

Ⅱ. 문자열과 콘솔 입출력

3. 문자열 슬라이싱

```
>>> a = "Hi! I am Donghun Kim. Nice to meet you."
>>> print(a[0:6])
Hi! I
>>> print(a[-9:])
meet you.
```

4. 문자열 연산

'a' + 2 → **ERROR!**

```
>>> a = "Orange"
>>> b = "Banana"
>>>
>>> a+b
'OrangeBanana'
>>>
>>> a*2
'OrangeOrange'
>>>
>>> "nana" in b
True
```

Ⅱ. 문자열과 콘솔 입출력

5. 문자열 함수

함수명	기능
len()	**문자열의 문자 개수를 반환**
upper()	**대문자로 변환**
lower()	**소문자로 변환**
title()	각 단어의 앞글자만 대문자로 변환
capitalize()	첫 문자를 대문자로 변환
count('찾을 문자열')	'찾을 문자열'이 몇 개 들어 있는지 개수 반환
find('찾을 문자열')	'찾을 문자열'이 왼쪽 끝부터 시작하여 몇 번째에 있는지 반환
replace('찾을 문자열', '바꿀 문자열', 바꿀 개수)	'찾을 문자열' 을 '바꿀 문자열' 로 변환 '바꿀 개수' 는 옵션, 없을 경우 문자열 전체에서 변환
startswith('찾을 문자열')	'찾을 문자열'로 시작하는지 여부 반환
endswith('찾을 문자열')	'찾을 문자열'로 끝나는지 여부 반환
strip()	좌우 공백 삭제
rstrip()	오른쪽 공백 삭제
lstrip()	왼쪽 공백 삭제
split()	문자열을 공백이나 다른 문자로 나누어 리스트로 반환
isdigit()	문자열이 숫자인지 여부 반환

Ⅱ. 문자열과 콘솔 입출력

6. Escape 문자

- 자체로 입력이 어려운 문자들을 표현하기 위해서 Backslash (₩) 를 사용하여 표현

특수문자	기능
\ Enter	다음 줄과 연속임을 표현
₩₩	₩ 문자 자체
₩'	' 문자
₩"	" 문자
\b	백스페이스
\n	줄 바꾸기
\t	Tab 키
\e	Esc 키

Ⅱ. 문자열과 콘솔 입출력

1. input() : 화면 입력

ex) result = input()

result = result + 100

무슨 일이 일어날까? 어떻게 해결할 수 있나?

2. print() : 화면 출력

Q1) 끝 날 때 줄 바꿈을 없애려면 ?
print(end="")

Q2) print 안에 변수와 문자열을 섞어서 출력하고 싶어요.
print("a: ", a," , b : ", b)
old formatting
new formatting

Ⅱ. 문자열과 콘솔 입출력

2. 변수의 자료형에 따른 서식

서식	설명
%s	문자열(string)
%c	문자 1개(character)
%d	정수(integer)
%f	실수(floating-point)
%o	8진수
%x	16진수
%%	문자 % 자체

Ⅱ. 문자열과 콘솔 입출력

3. New Formatting : 신규 방식 (추천)

1) {} 는 format bracket
```
print("{} {}".format(a,b))
```

2) {0}, {1}, {2}, ... 는 formatting 할 인자의 순번
```
print("{0} {1}".format(a,b))
print("{1} {0}".format(a,b))
```

3) 정수형 formatting : {:d}
```
print("{0:d} {1:d}".format(a,b))
```

4) 정수형 자릿수 formatting : {:xd}
```
print("{0:2d} {1:3d} {3:4d}".format(2, 2**2, 2**3))
```

Ⅱ. 문자열과 콘솔 입출력

3. New Formatting : 신규 방식 (추천)

5) 실수형 formatting : {:f}
```
print("{0:f} {1:f}".format(a,b))
```

6) 실수형 정수자리와 소수자리 지정 formatting : {:x.xf}
```
pi = 3.141592653
print("{0:6.2f}".format(pi))
```

6) 인자의 이름을 넣을 수도 있다.
```
print('This {food} is {adjective}.'.format(food='spam',
adjective='absolutely horrible'))
```

Ⅱ. 문자열과 콘솔 입출력

2. New Formatting another : 문자열 앞에 'f' 를 넣는 방식

print(f'text something {*variable*}')

예)
```
>>> a = 30, b = 60
>>> print(f'a={a}, b={b}')
```

Ⅲ. 자료구조

- 리스트
- 튜플
- 딕셔너리

Ⅲ. 자료구조 - 리스트

1) Basics

- 파이썬의 대표 자료형으로 여러 type 의 자료를 원소가 가짐
- 수정 가능 (mutable), 추가/삭제/변경이 가능
- 파이썬에서 가장 많이 널리 쓰이는 자료구조
- 순서를 가짐 => 원소들의 순서가 다르면 다른 list 임.

```
>>> a = [ 1, 2, "bar", 45.6 ]
```

ex)

```
a = [ ]
b = [1, 2, 3, 4, 5]
c = ['Life', 'is', 'too', 'short']
d = [1, 2, 'Life', 'is']
e = [1, 2, ['Life', 'is'] ]
```

Ⅲ. 자료구조 - 리스트

2) List Indexing
- 인덱스(index) 를 이용하여 접근가능

3) Built-in functions and operators
- **in** and **not in**
- + and *
- len(), min(), max()

4) List methods
- **append**() : 맨 뒤에 새로운 요소 삽입
- **index**() : 요소 값이 나타나는 첫 번째 인덱스를 반환
- **sort**() : List 를 정렬 함 (자기 자신을 변형 시킴)
- **len()** : list 의 길이 (원소의 개수) 를 반환하는 함수

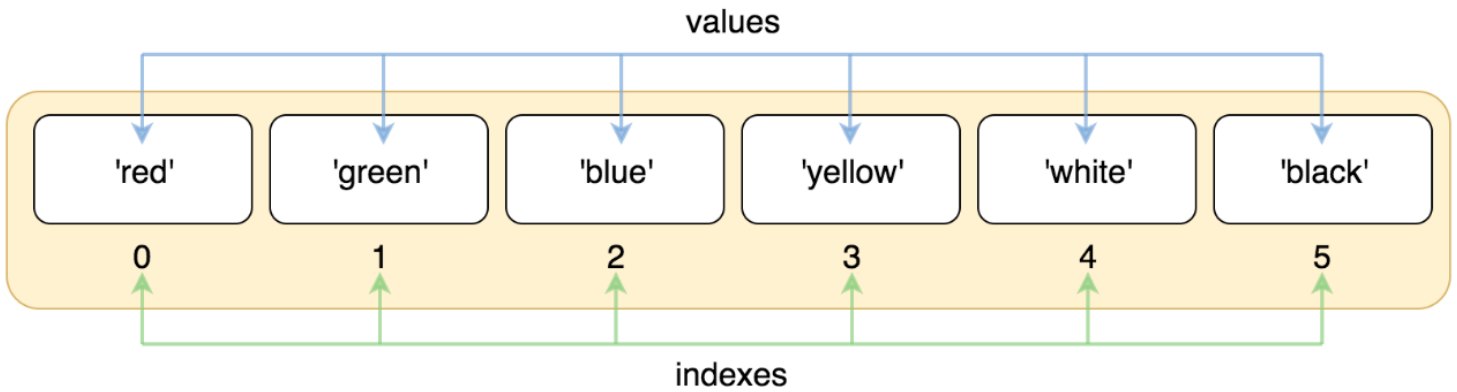
ex)
```
a = [1,2,3,4,5]
print(len(a))
```

Ⅲ. 자료구조 - 리스트

1. List 인덱싱

- List 는 각 요소마다 숫자 인덱스가 부여되고, 0 부터 시작한다.

```
>>> colors = ['red', 'green', 'blue', 'yellow', 'white', 'black']
```



1) 정방향 인덱싱

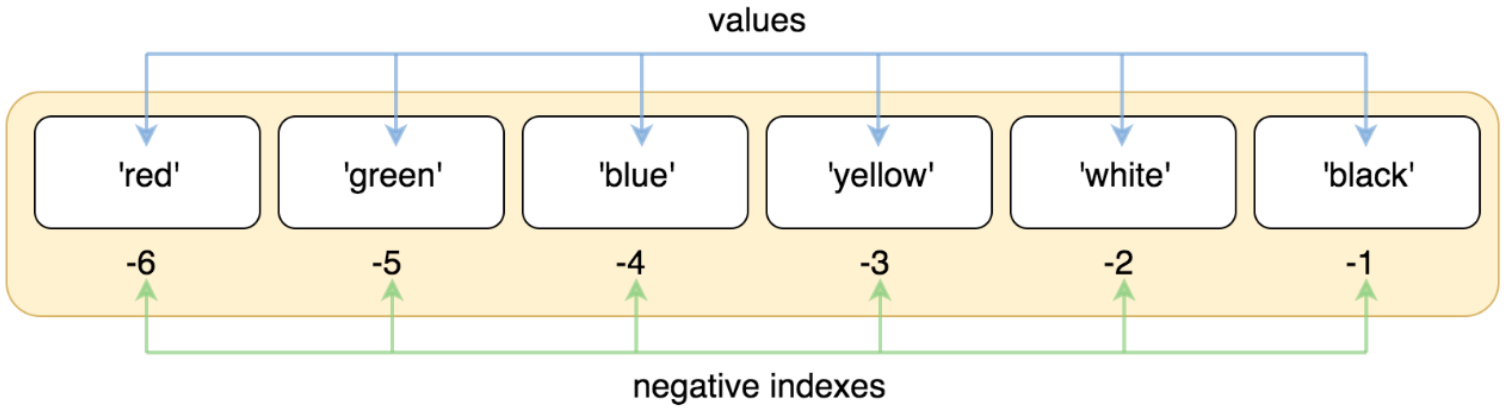
```
>>> colors[0]
'red'
>>> colors[5]
'black'
```

참고자료: https://bit.ly/3awoM3r

Ⅲ. 자료구조 - 리스트

2) 역방향 인덱싱

```
>>> colors = ['red', 'green', 'blue', 'yellow', 'white', 'black']
>>> colors[-1]
'black'
>>> colors[-2]
'white'
```



0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

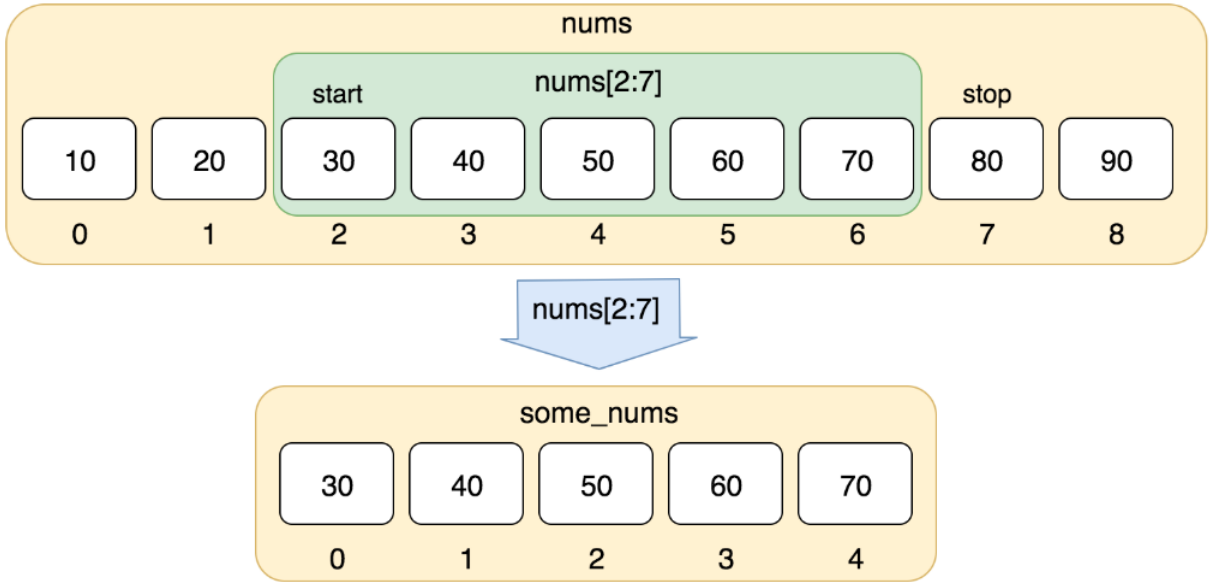
참고자료: https://bit.ly/3awoM3r

Ⅲ. 자료구조 - 리스트

2. List 슬라이싱

- List 의 일부분을 도려내어 List 형태로 반환한다.
- List[시작 숫자:끝 숫자] 형태이며, 끝 숫자는 포함하지 않음

>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]



Ⅲ. 자료구조 - 리스트

1) 기본적인 슬라이싱

```
>>> nums[0:4]
[10, 20, 30, 40]

>>> nums[2:7]
[30, 40, 50, 60, 70]

>>> nums[:5] #0부터 시작한다는 의미
[10, 20, 30, 40, 50]

>>> nums[-3:] # -3 (= 뒤에서 부터 3번째) 부터 끝까지
[70, 80, 90]
```

Ⅲ. 자료구조 - 리스트

2) Step 을 활용한 슬라이싱

```
>>> nums[0:3]
>>> nums[0:3:2] #0부터 3까지 (3포함 안 함) 2칸씩 건너 뛰며
[10, 30]

>>> nums[1: :2] #1부터 끝까지 2칸씩 건너 뛰며
[20, 40, 60, 80]
```

3) 역방향 Step 활용한 슬라이싱

```
>>> nums[첫:끝:-1]
[90,80,70,60,50,40,30,20,10]

>>> nums[-1:-4:-2] # -1부터 -4까지 2칸씩 거꾸로
[90, 70]
```

Ⅲ. 자료구조 - 리스트

5) **리스트 순환 (Traversal)**

```
aList = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

sum = 0
for num in aList:
    sum += num

print(sum)
```

6) Convert List ↔ Tuple, Set
ex) list() ↔ tuple()

Ⅲ. 자료구조 - 튜플

1. Tuple 자료형
- 복수개의 원소로 구성 됨 (다른 type 의 원소를 가질 수 있음)
- iterable
- '()' 로 정의

```
>>> a = ( "orange", "banana", "apple", "mango" )
>>> b = ( 10, 20, 30, 40, 50, 60 )
>>> c = ( 23, "rr", 123.4 )
```

2. tuple 에서의 index 사용
- list 와 마찬가지로 tuple 의 원소를 indexing 가능

3. immutable data structure
- list 와 반대로 요소를 수정할 수 없음

참고자료 : https://realpython.com/python-lists-tuples/#defining-and-using-tuples

Ⅲ. 자료구조 - 튜플

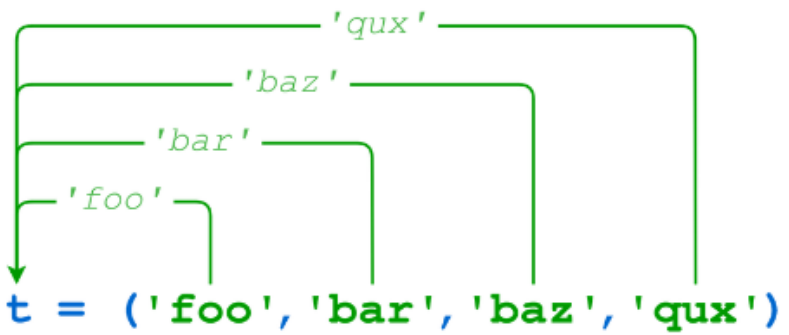
4. tuple 을 왜 사용하는가 ?

- 프로그램 실행이 더 빠르다.
- 데이터가 수정되는 걸 원치 않을 때

5. tuple packing and unpacking

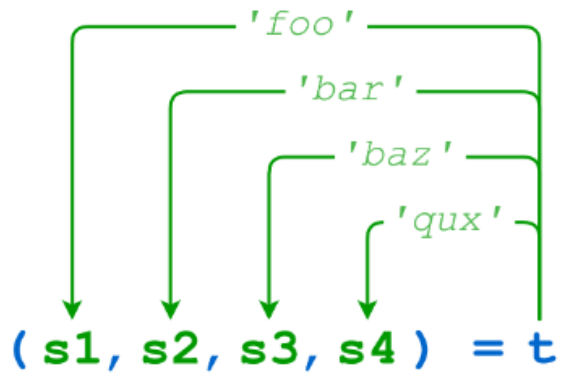
① packing

```
>>> t = ( "foo", "bar", "baz", "qux
```



② unpacking

```
>>> ( s1, s2, s3, s4 )  = t
>>> s1
>>> s2
>>> s3
>>> s4
```



Ⅲ. 자료구조 - 튜플

6. 함수의 packing and unpacking

- 함수의 return 값을 여러개 반환할 경우 tuple 객체로 반환 됨
- 대입연산자를 통해 unpacking 되어 할당 됨

ex)

```
def calc_basic_stat(records):
    sum = 0.
    for record in records:
        sum += record
    return sum, round(sum/len(records), 1), len(records)

sum, mean, count = calc_basic_stat(m_school_weights)
```

참고자료 : https://realpython.com/python-lists-tuples/#defining-and-using-tuples

Ⅲ. 자료구조 - 딕셔너리

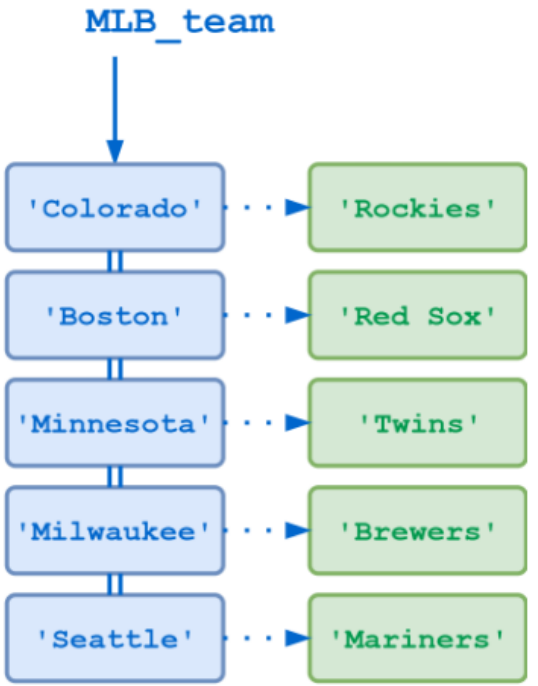
1. 기본사항

- Key 와 Value 로 구성
- Key 와 Value 에는 여러 type 이 사용될 수 있다.
- key 값은 반드시 **immutable type** 이어야 한다.

```
>>> dict = {[1,2,3]:100, [2,3]:200} ?
```

- 형식, a_dict = { **key1:val1**, **key2:val2**, … }

```
MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'
}
```



참고자료 : https://realpython.com/python-dicts/

Ⅲ. 자료구조 - 딕셔너리

2. 딕셔너리 정의하는 두 가지 방법

① 빈 dictionary 로 부터

```
a_dict = {}
a_dict["a"] = 10
a_dict["b"] = 20
a_dict["c"] = 30
```

- 새로운 <key, value> 를 넣고 싶을 때

```
>>> a_dict["d"] = 40
```

- 기존 key 값의 value 를 변경하고 싶을 때

```
>>> a_dict["a"] = 100
```

② 초기값을 포함하여 생성

```
a_dict = {"a":10, "b":20, "c":30}
```

참고자료 : https://realpython.com/python-dicts/

Ⅲ. 자료구조 - 딕셔너리

3) 딕셔너리 접근하기
- Key 값으로 indexing

```
a_dict = {}
a_dict['A'] = 100
a_dict['B'] = 200
a_dict['C'] = 300

print(a_dict['A'])
```

- keys(), values(), items() 로 순환 (Traversal)

```
for k, v in a_dict.items():
    print("key 값", k)
    print("value 값", v)
```

참고자료 : https://realpython.com/python-dicts/