

I. 강좌 소개 및 운영에 관한 사항

i. 프로그래밍 환경

'Python'이라는 언어로 컴퓨터를 제어하기 위해서는 컴퓨터에 'Python' 언어를 해석해주는 interpreter가 필요하다. Python interpreter는 다양한데, Anaconda를 많이 사용하는 편이다.

또한, Python은 내부함수와 추가적인 툴의 활용성과 편의성이 높아 강력하다고 평가되는 언어이다. 다만, 이런 함수나 툴은 사용자가 일일이 찾고 사용하는 것은 수고가 드는 일이다. 따라서 이러한 개발을 하는 개발자들이 함수나 툴을 찾고 사용하는 것을 편하게 만들어 주기 위해 서 만들어진 것이 통합개발환경(IDE)이다. 다양한 Python 통합개발환경(IDE) 중에서 가장 많이 사용되는 것이 Pycharm이다.

- Anaconda (<https://www.anaconda.com/distribution/>): python 설치.
- Pycharm (<https://www.jetbrains.com/pycharm/>): 일반적인 IDE(통합개발환경).
- Online Practice Site.

ii. 과목에서 다룰 내용

- 컴퓨터 공학 및 프로그래밍에 대한 기초
- Python (python 3) 언어의 기초적인 문법
 - ▷ number (int, float), string, list 등 기본적인 자료형과 연산
 - ▷ 함수(function)
 - ▷ 문자열 조작
 - ▷ 입출력
 - ▷ class 등 OOP (Objective-Oriented Programming) 기본 개념
- 문제해결을 위한 간단한 알고리즘에 대한 이해 및 복잡도

(complexity)

- 수업과 연계된 실습, 과제, 프로젝트 등을 통한 실제 코딩

iii. 강사 소개

- 이름: 신동훈 교수
- Email: dshin@dgist.ac.kr
- Office Hour: 수요일 15:00~16:30, E7-L13
- Office Hour는 여러가지 사정으로 바뀔 수 있으니, 이 곳에서 항상 사전메일하시기 바랍니다.
- 이메일 이용 시, 제목에 [SE213 x분반]을 포함할 것 (실습 분반)
- 이메일 이용 시, Python에 관한 내용과 실습 출석 등에 관한 부분은 실습 담당 교수/조교에게 보낼 것.
- 이메일 이용 시, 학점 등 평가에 대한 부분은 실습 담당 교수와 신동훈 교수에게 보낼 것.

iv. 교재 / 읽을 거리

1. 주 교재

Python 교재는 매우 많으니, 그 중에 한 권을 정해서 보면 됩니다. 아래 참고 교재에 온라인으로 접근 가능한 책들이 많이 있으니, 그 중에 한 권을 정해서 보는 것도 좋습니다.

<http://docs.python.org>

2. 참고 교재

▷ How to think like a computer scientist:

<http://www.openbookproject.net/thinkcs/python/english2e/>

▷ Python for Everybody, Exploring data in python 3 (정보교육을 위

한 파일): <http://www.pythonguides.com/book.php>(영어),
https://www.pythonguides.com/translations/KO/book_009_ko.pdf(한국어
번역)

- ▷ CS for all: <https://www.cs.hmc.edu/csforall/>
- ▷ Byte of python:
<https://www.gitbook.com/download/pdf/book/swaroopch/byte-of-python>(python 3 기준 영어본), http://byteofpython-korean.sourceforge.net/byte_of_python.pdf(Python 2 기준 한글본)
- ▷ 처음 시작하는 python, 빌 루바노빅 지음, 최길우 옮김, 한빛미디어, 2015년 12월 10일 출간
- ▷ 파이썬 완벽 가이드, 데이비드 M. 비즐리 지음, 송인철-송현제 옮김, 인사이트, 2012년 4월 6일 출간

3. Further readings on python or programming

- ▷ 어떻게 프로그래밍을 공부할 것인가:
<https://paper.dropbox.com/doc/UFXkagqwYBWfpDmigP0WE>
- ▷ Python style guide: (Official document)<https://www.python.org/dev/peps/pep-0008/>, (한국어 번역)
<http://kenial.tistory.com/902>, (주요 내용 한국어 번역 및 해설)
<https://spoga.github.io/2012/08/03/about-python-coding-convention.html>, (Google python style guide)
<https://google.github.io/styleguide/pyguide.html>, (참고)
<http://docs.python-guide.org/en/latest/writing/style/>

4. Python 프로그래밍할 때, 도움을 받을 수 있는 사이트

- ▶ References (사전처럼 언어에 필요한 내용을 찾아볼 수 있는 문서)
 - ▷ The python language reference: 공식적인 python 언어의 reference
- ▶ python 코드의 실행을 시각화 하여 보여주는 사이트
 - ▷ Python tutor

- ▶ Tutorials (python을 배울 수 있는 사이트)
 - ▷ codeacademy: 기초적인 python 코드를 온라인 상에서 따라하면서 배울 수 있는 사이트
 - ▷ PySchools python quick reference guide: 다른 언어를 알고 있는 사람이 python을 빠르게 배울 때 도움이 되는 사이트

5. 문제풀이

- ▷ 17 Coding Challenges to Help You Train Your Brain:
<http://codecondo.com/coding-challenges/>

II. Python Programming Basics: Variables, Operators, Standard Input/Output

i . Python 소개와 기초 문법

1. Python의 기본 자료형 (built-in Data types)

자료형은 데이터의 유형을 지정한다.

▶ 부울(bool) 자료형 (True와 False 2개의 유형만 존재한다.)

Bool 자료형	코드표현	숫자표현
자료유형 1	True	1
자료유형 2	False	0

▶ 정수(int) 자료형

int 자료형	숫자표현
자료예시 1	1
자료예시 2	2
자료예시 3	0
자료예시 4	-1
자료예시 5	-2
자료예시 6	100000

▶ 실수(float) 자료형

float 자료형	숫자표현
자료예시 1	3.14
자료예시 2	1.0
자료예시 3	6.02e23 (=6.02*10 ²³)

▶ 복소수(complex) 자료형

complex 자료형	숫자표현
자료예시 1	1+2j
자료예시 2	j
자료예시 3	1+4j

▶ 문자열(str) 자료형

str 자료형	코드표현
자료예시 1	'Hello, world'
자료예시 2	"Hello, world"
자료예시 3	'1'
자료예시 4	'1.0'

2. 연산자

- ▶ 연산자는 데이터/값들에 대한 연산을 정의한다.
 - ▷ 자료형에 따라 연산자가 정의된다. 예를 들면 int 자료형과 int 자료형을 더하는 연산자와 float 자료형과 float 자료형을 더하는 연산자는 같은 +를 쓰지만, 컴퓨터는 int 자료형과 int 자료형을 더하는 +를 int 덧셈용 연산자라고 인식하고, float 자료형과 float 자료형을 더하는 +를 float 덧셈용 연산자라고 인식한다.
 - ▷ 위의 예시에서 int 덧셈용 연산자가 +이고, float 자료형 덧셈용 연산자가 +인 것처럼, 같은 연산자가 자료형에 따라 다른 동작을 하기도 한다.
 - ▷ 연산자는 하나 혹은 그 이상의 피연산자(연산 되어지는 것, operand)를 가질 수 있다. 예를 들어 – (-1)과 같은 경우에 맨 앞의 -는 (-1)이라는 1개의 피연산자를 가지고 있는 상황인 것이고, 1+2와 같은 경우의 +는 1과 2라는 2개의 피연산자를 가지고 있는 상황인 것이다.

▶ 연산자의 종류

- ▷ 수에 대한 사칙연산에 사용되는 연산자

연산자	설명	예시	예시의 결과
+	더하기	5+8	13
-	빼기	90-10	80
*	곱하기	4*7	28
/	부동소수점 나누기 (일반적인 나누기)	7/2	3.5
//	정수 나누기 (나눈 값에서 정수만 가져 오기)	7//2	3
%	나머지	7%2	1
**	지수	3**4	81

▷ 연산자 우선 순위(operator precedence)

Operator	Description	우선순위
lambda	Lambda expression	LOW
if – else	Conditional expression	
or	Boolean OR	
and	Boolean AND	
not x	Boolean NOT	
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, incl. membership tests and identity tests	
	Bitwise OR	
^	Bitwise XOR	
&	Bitwise AND	
<<, >>	Shifts	
+, -	Addition and subtraction	
*, @, /, //, %	Multiplication, matrix multiplication, division, remainder	
+x, -x, ~x	Positive, negative, bitwise NOT	
**	Exponentiation	
await x	Await expression	
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference	
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or tuple display, list display, dictionary display, set display	HIGH

ex) $2^{**}\max(3-2, 0)-1 \Rightarrow 2^{**}\max(1, 0)-1 \Rightarrow 2^{**}1-1 \Rightarrow 2-1$
 $\Rightarrow 1$

3. Expression

하나의 값을 계산(evaluate)하기 위한 값, 연산자, 변수, 함수 요소들의 조합을 Expression이라고 부른다.

ex1) 42

ex2) 42+23

ex3) max(42, 23)

4. 변수

A. 변수가 필요한 이유

- ▶ 동일한 값을 여러 번 사용할 필요가 있다.
- ▶ 중간 결과를 나타내기 위해서 사용한다.
- ▶ 수학에서 사용하는 변수와 유사하다.

B. 정의

- ▶ 어떠한 값(value) 혹은 객체(object)에 대한 이름(alias)을 변수라고 한다.
- ▶ 엄밀히 말하면, 값(value) 혹은 객체(object)를 저장할 수 있는 이름이 붙어있는 메모리 상의 공간을 변수라고 한다.
- ▶ 변수에 값 혹은 객체를 대입(assignment)한 이후, 대입된 값 혹은 객체 대신에 변수의 이름을 사용할 수 있다.

C. 변수 이름의 규칙

▶ 규칙

- ▷ 변수 이름의 첫 글자는 반드시 알파벳 문자 혹은 밑줄(_)이여야 한다.
- ▷ 변수 이름의 나머지 글자는 문자, 밑줄, 숫자 모두 가능하다.
- ▷ 컴퓨터는 변수의 대/소문자를 구별해서 다르게 인식한다. 예를 들면, pi, Pi, PI의 3가지 변수가 있을 때, 컴퓨터는 이 변수들을 서로 다른 값으로 인식한다.
- ▷ python keyword(python의 내부함수 및 연산자들, 예약어라고 부름)는 변수 이름으로 설정할 수 없다.

▶ 예시

- ▷ 좋은 예: l, name_2_3
- ▷ 나쁜 예: 2things, two words, my-name, >a1b2_3

▶ 일반적으로 python에서의 변수이름은 할당되는 값들이 어떤 값들인지 잘 나타낼 수 있는 이름으로 설정한다. 또한 변수 이름은 모두 소문자로 작성하고, 단어 사이에서는 밑줄을 사용 한다.

D. 변수 대입(assignment)

▶ `variable_name = expression`의 꼴을 변수 대입문이라 정의한다.

▷ 의미: 오른쪽 `expression`의 결과값을 왼쪽 변수(할당된 저장소 공간)에 집어 넣으라(대입하라)는 명령이다.

▷ 주의: 할당할 변수의 이름은 항상 왼쪽에 위치하여야 하고, 할당될 변수 혹은 `expression`은 오른쪽에 위치하여야 한다.

▶ 예제

Programming Script	Terminal
#변수 greeting을 정의하고, 문자열 대입 <code>greeting = 'Hello, world!'</code> <code>print(greeting)</code>	Hello, world!
#변수 n을 정의하고, 값을 대입 <code>n = 42</code> <code>print(n)</code>	42
#변수 pi를 정의하고, 값을 대입 <code>pi = 3.1415926534</code> <code>print(pi)</code>	3.1415926534
#변수 r을 정의하고, 값을 대입 <code>r = 3</code> <code>print(pi*r**2)</code>	28.2743338806
#이미 만들어진 변수 r의 값을 변경 <code>r = 5</code> <code>print(pi*r**2)</code>	78.539816335

E. 변수 축약형

▶ `variable #= expression`의 꼴을 변수 축약형이라 정의한다.

▷ 의미: variable = variable # expression의 축약형태로, 변수의 값을 이용해 expression과 #연산을 하고,
다시 그 값을 변수에 대입하는 대입연산자를 변수 축약형이라고 한다.

▷ 이 변수 축약형이 실행되면, 가장 먼저 expression이 계산되고, 그 뒤에 변수와 expression이 계산된다.

▷ #의 자리에 들어갈 수 있는 연산자는 +, -, *, /, //, % 등이다.

▶ 사용 예시

`var += 3` ⇒ `var = var + 3`

`t *= 2 + 3` ⇒ `t = t * (2+3)`

F. 문자열과 변수

▶ 따옴표(' 또는 ")로 둘러싸여 있는 문자들을 문자열이라고 한다.

▶ 어떠한 값 또는 객체를 나타내는 이름을 문자열이라고 한다.

▶ 예제

`n = 42` ⇒ `n`이라는 변수에 42 할당하기.

`var1 = n` ⇒ `var1`이라는 변수에 `n`이라는 변수의 값 (=42) 할당하기.

`var2 = 'n'` ⇒ `var2`이라는 변수에 'n'이라는 문자열 할당하기.

`var3 = n + 1` ⇒ `var3`라는 변수에 `n`이라는 변수의 값 (=42)에 1을 더한 값 (=43) 할당하기.

`var4 = 'n' + 1` ⇒ Error 발생(문자열과 숫자를 더하는 연산자는 정의되어 있지 않다.)

5. Python에서의 함수

- ▶ 재사용이 가능한, 특정한 작업(계산, 입출력 등)을 수행하는 명령어의 집합을 함수라고 한다.
- ▶ 수학에서 인자와 동일한 역할을 하는 인자(argument)를 가질 수 있다.
- ▶ 수학에서 함수값과 동일한 역할인 반환값(return value)을 가질 수 있다.

6. 표준 출력(standard output): print()

- ▶ 컴퓨터의 메모리에 저장된 값을 출력매체를 통해 보여주도록 하는 함수를 표준 출력함수라고 하며, print()의 꼴로 정의한다.
- ▶ print()의 역할
 - ▷ 인자에 있는 내용을 화면에 출력하고 줄을 바꾼다.
 - ▷ 인자가 여러 개 있는 경우, 여러 인자 사이에 ,를 찍어 입력해주세요 하며 이 경우 ,가 빈 칸(Space 1칸)으로 변경되어 출력된다.
 - ▷ 문자열의 빈 칸(Space 1칸)이 아닌 그 외의 빈 칸(Space 1칸)은 특별한 의미를 가지지 않으며, 출력되지도 않는다.
- ▶ 예제: print()를 이용하여 계산기처럼 사용

Programming Script	Terminal
print(3) #정수	3
print(3.0) #실수	3.0
print(2 + 3) #expression인 2+3을 먼저 계산하고, 나온 값을 print()한다.	5
print(2 - 3)	-1
print(2 * 3)	6
print(2 / 3)	0.6666666666666666
print(2 // 3) #정수형으로 나눗셈	0
print(2 ** 3) #2*2*2	8
print('Hello, World!') #문자열	Hello, World!
print("Hello, World!") #문자열	Hello, World!
print(1, 2, 3) #여러 개의 인자	1 2 3
	8 8

```
print(2 * 2 * 2, 2 ** 3)
```

7. 표준 입력(standard input): input()

- ▶ 컴퓨터의 메모리에 입력매체를 통해 값을 저장하도록 하는 함수를 표준 입력 함수라고 하며, `input()`의 꼴로 정의한다.
- ▶ '프로그램으로 들어가는 데이터 스트림(일반적으로 키보드 입력)'라고 표현하기도 한다.
- ▶ `input()`의 역할
 - ▷ 일반적으로 키보드로 입력받은 내용을 문자열로 반환하는 역할을 한다.
 - ▷ 보통 키보드로 입력받은 내용을 문자열로 반환하여 변수로서 메모리에 저장하고 사용하기 위해서 `variable_name = input()`의 꼴로 사용되는데, `input()`의 꼴로 단독으로 쓰면 입력받은 값이 문자열로 반환된 후 저장되지 않고 바로 사라지기 때문이다.
 - ▷ 함수() 안에 인자(변수 혹은 값이나 문자열)를 집어넣을 경우, 인자의 내용을 화면에 출력한다.

▶ 예제: python에서 키보드 입력

Programming Script	Terminal
<pre>name1 = input() print(name1)</pre>	홍길동 홍길동
<pre>name2 = input('당신의 이름은? ') print('당신은', name2, '이군요.)')</pre>	당신의 이름은? 홍길동 당신은 홍길동 이군요.
<pre>name3 = input('당신의 이름은? ') age = input(name3 + '의 나이는? ') print(name3, '의 나이는', age, '세 입니다.)')</pre>	당신의 이름은? 홍길동 홍길동의 나이는? 20 홍길동 의 나이는 20 세입니다.

III. 형변환, 조건문, 모듈 사용의 기초

i . Computational Thinking (컴퓨터적 사고, 계산적 사고)

1. 분해 (Decomposition)

큰 문제를 해결 가능한 작은 문제로 나누는 것, 경우에 따라 여러 번 반복.

2. 패턴인식 (Pattern recognition)

(작게 나뉘어진) 주어진 문제에서 공통된 요소(패턴)을 찾음.

3. 추상화 (Abstraction)

문제 해결에 있어 중요한 공통된 부분만을 놔두고, 다른 구체적인 사항들을 없앰.

4. 알고리즘 (Algorithms)

어떤 입력에도 원하는 결과를 얻을 수 있도록 잘 정의된 연산 단계를 만든다. ex) 덧셈, 뺄셈, 곱셈, 나눗셈

ii. 형변환(type conversion)

1. 형변환 (type conversion)

▶ 한 자료형(data type)에서 다른 자료형으로 변경하는 것을 형변환(type conversion)이라 부른다.

▶ input()으로 입력 받은 값은 문자열로 반환되기 때문에 받은 값을 사용하여 원하는 연산(예: 사칙연산)을 수행하기 위해서는 형변환이 필요하다.

2. 형변환 함수: 바뀔 자료형의 이름을 함수처럼 사용한다.

A. int()

▶ () 안의 인자를 정수형으로 형변환 시키는 함수이다.

▶ 예시

▷ integer=int(3.14)로 입력하면, integer에 3이 저장된다.

▷ integer=int('abc')로 입력하면, 오류가 발생한다.

▷ integer=int('3.7')로 입력하면, integer에 3이 저장된다.

B. float()

▶ () 안의 인자를 실수형으로 형변환 시키는 함수이다.

▶ 예시

▷ real=float(3)로 입력하면, real에 3.0이 저장된다.

▷ real=float('abc')로 입력하면, 오류가 발생한다.

▷ real=float('3.7')로 입력하면, real에 3.7이 저장된다.

C. 형변환이 가능하지 않은 경우에는 오류가 발생한다. 예를 들면, 문자열 'abc'는 정수 혹은 실수로 변환되지 않는다.

3. 자료형 확인: type()

A. 자료형 확인 함수

▶ 주어진 객체, 변수 등의 클래스를 확인하는 함수이다.

B. 사용방법

▶ 일반적으로 자료형을 확인하고자 하는 객체(object)를 인자로서 전달하는 type(object)의 형태의 함수를 사용한다.

C. 사용예시

Programming Script	Terminal
--------------------	----------

<pre> print(type('홍길동')) print(type(3.1)) print(type(True)) # 변수에 저장된 자료형 확인 var = 10 print(type(var)) </pre>	<class 'str'> <class 'float'> <class 'bool'> <class 'int'>
--	--

Programming Script	Terminal
<pre> name = input('너의 이름은? ') print('당신은', name, '로군요') pi = 3.1415926534 #키보드로부터 값을 입력받음. r = input('Enter a radius') r = float(r) # 연산을 위해 형변환 print(pi*r**2) # input()의 반환값을 float()의 인자로 사용 r = float(input('Enter a radius: ')) print(pi*r**2) </pre>	너의 이름은? 미츠하 당신은 미츠하 로군요 Enter a radius: 3 28.2743338806 Enter a radius: 3 28.2743338806

iii. 제어 흐름 (control flow)

1. 명령문 (statement)

- ▶ 실행의 단위이다.
- ▶ 보통 한 줄의 코드로 나타내어진다.
- ▶ 명령문은 한 줄씩 차례로 실행된다.

2. 제어 흐름 (control flow)은 명령문들이 실행되는 순서/방법을 지시한다.

- ▶ 조건문: 조건에 따라 다른 명령문들을 실행한다.
- ▶ 반복문: 하나 혹은 그 이상의 명령문들을 반복해서 실행한다.
- ▶ 함수: 동일한 명령문들을 프로그램의 여러 곳에서 실행한다.

iv. 조건문: if, else, elif

1. 조건문과 불리언(boolean)식

- ▶ 조건에 따라 다른 명령문을 실행해야 되는 경우가 있다.
- ▶ 많은 프로그래밍 언어에서 조건문은 불리언(boolean)식으로 표현해야 한다.
 - ▷ 불리언(boolean)식: True 혹은 False를 계산하는 식(Expression)
- ▶ 다음과 같은 값들은 False로 인식된다.
 - ▷ false로 정의된 상수: None, False
 - ▷ 수를 나타내는 자료형에 0이 저장된 경우: 0, 0.0, 0j, Decimal(0), Fraction(0, 1)
 - ▷ 원소를 저장하고 있지 않은 자료구조: ""(내용이 없는 문자열), ()(내용이 없는 튜플), [](내용이 없는 리스트), set(), range(0)

- ▶ True or False로 인식될 수 있는 값에 무엇이 있는지는 다음 사이트를 통해 확인해볼 수 있다.
<https://docs.python.org/3/library/stdtypes.html?highlight=boolean#truth-value-testing>

2. 불리언(boolean)식(expression) 조건문의 표현

- ▶ 비교 연산자와 불리언 연산자를 통해 조건문을 표현하면, 불리언식이 된다.
- ▶ 비교 연산자의 종류

연산자	설명	예	결과
<	Less than	5 < 3	False
		3 < 5	True
>	Greater than	5 > 3	True
		3 > 5	False
<=	Less than or	3 <= 6	True

	equal to		
\geq	Greater than or equal to	$4 \geq 3$	True
$=$	Equal to	$2 == 2$ $'str' == 'str'$	True False
\neq	Not equal to	$'str' != 'stR'$	True

▶ 불리언(Boolean) 연산자의 종류

x	y	x and y	x or y	not x
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	

▷ and 연산: 두 인자가 모두 True여야 True를 반환하고, 그 이외에는 False를 반환하기 때문에, 첫 번째 인자가 True인 경우에만, 두 번째 인자를 계산한다.

▷ or 연산: 두 인자가 모두 False여야 False를 반환하고, 그 이외에는 True를 반환하기 때문에, 첫 번째 인자가 False인 경우에만, 두 번째 인자를 계산한다.

▷ not 연산: not 연산자는 불리언(boolen) 연산자 이외의 연산자보다 우선 순위가 낮다. 따라서 not a==b는 not (a==b)의 순서로 계산된다.

또한, 이 때문에 a==not b로 적으면 문법오류가 발생한다. (a==not) b로 읽혀지기 때문이다. a==(not b)를 연산할 의도였으면, a==(not b)로 적어주어야 오류가 발생하지 않는다.

3. 조건문: if, else, elif

Python Code	설명
<code>if condition1: statement1_1 statement1_2</code>	▶ if: 조건이 참일 때, 코드블럭(code block: if안에 있는 명령문들)을 실행하는 함수이다. (조건문을 사용할 때, 1번만 사용할 수 있다.)
<code>elif condition2: statement2_1 statement2_2</code>	▶ elif: 추가적인 조건과 그 조건이 참인 경우 해당하는 조건의 코드블럭을 실행하는 함수이다. 0개 이상 사용 가능하다. (즉, 생략 가능하고, 개수의 제한이 없다.)
<code>elif condition3: statement3_1 statement3_2</code>	
<code>else:</code>	▶ else: 만족하는 조건이 없을 때, else안에

<p><code>else_statement</code></p>	<p>있는 명령문들(코드블럭)을 실행하는 함수이다. (1개만 쓸 수 있으며, 생략가능하다.)</p> <p>▶ 왼쪽 Python Code의 전체적인 의미: 위에서부터 조건을 검사하여 가장 먼저 참이 되는 조건에 '해당하는 코드블럭만을 실행'한다. 만족하는 조건이 없을 때는 <code>else</code> 이후의 코드블럭을 실행한다.</p> <p>▶ 왼쪽의 Python Code에서 statement 앞에 빈칸이 4개 있는 이유는 Code Block을 구분해주기 위한 표시이기 때문이다.</p>
------------------------------------	---

4. 조건문 활용 예시

Python Code Script	Terminal
<pre># 조건문 사용1 if 3<5: print('3 is less than 5') if 5<3: print('5 is less than 3') n=3 if n<5: print('n is less than 5') if n=10 if n<5: print('n is less than 5')</pre>	<pre>3 is less than 5 n is less than 5</pre>
<pre># 조건문 사용2 (else절) if 3<5: print('3 is less than 5') else: print('5 is less than 3') n=3 if n<5: print('n is less than 5') else: print('n is NOT less than 5')</pre>	<pre>3 is less than 5 n is less than 5</pre>
<pre># 조건문 사용3 (elif & else 절) number = 5 if number < 0: print('A negative number') elif number == 0: print('Zero')</pre>	<pre>A single digit number</pre>

<pre> elif number < 10: print('A single digit number') else: print('A number with two or more digits') </pre>	
<pre> # 조건문 사용4 number = 5 if number < 0: print('number is a negative number') print('That means number is less than 0.') if number < 0: print('number is a negative number') print('It means number is less than 0.') </pre>	It means number is less than 0.
<pre> # 조건문 사용5 (중첩된 if문) number = 5 if number < 10: if number % 2 == 0: print('A single digit even number') else: print('A single digit odd number') else: if number % 2 == 0: print('An even number with two or more digits') else: print('An odd number with two or more digits') </pre>	A single digit odd number

v. 코드블럭(Code block)

1. 코드블럭이란?

'하나의 단위로 실행되는' '하나 이상의 명령문들'을 말한다.

2. python에서는 빈 칸으로 코드블럭을 표시한다.

▶ 탭 문자 혹은 빈 칸으로 들여쓰기를 만들어 코드블럭을 표시하는데, 공통된 코드블럭 내에서는 같은 정도의 들여쓰기를 사용해야 한다.

▶ 만약 statement1은 4개 빈칸으로 들여쓰기하고, statement2는 2개 빈칸으로 들여쓰기하면, 컴퓨터는 statement1까지만 코

드블럭으로서 인식한다.

- ▶ 들여쓰기는 4개의 빈 칸을 사용하는 것을 강력하게 권장한다. 빈 줄이 있을 경우 그 줄은 무시된다.

3. 예시

```
if condition:  
    statement1  
    statement2
```

Python Code Script	Terminal
# 조건문 사용1 if 3<5: print('3 is less than 5') if 5<3: print('5 is less than 3') n=3 if n<5: print('n is less than 5') if n=10 if n<5: print('n is less than 5')	3 is less than 5 n is less than 5
# 조건문 사용2 (else절) if 3<5: print('3 is less than 5') else: print('5 is less than 3') n=3 if n<5: print('n is less than 5') else: print('n is NOT less than 5')	3 is less than 5 n is less than 5
# 조건문 사용3 (elif & else 절) number = 5 if number < 0: print('A negative number') elif number == 0: print('Zero') elif number < 10: print('A single digit number')	A single digit number

	else: print('A number with two or more digits')	
# 조건문 사용4 number = 5 if number < 0: print('number is a negative number.') print('That means number is less than 0.') if number < 0: print('number is a negative number') print('It means number is less than 0.')	It means number is less than 0.	
# 조건문 사용5 (중첩된 if문) number = 5 if number < 10: if number % 2 == 0: print('A single digit even number') else: print('A single digit odd number') else: if number % 2 == 0: print('An even number with two or more digits') else: print('An odd number with two or more digits')	A single digit odd number	

vi. 모듈 사용: math, random

1. Python에서 모듈/패키지 소개

▶ Python에는 수천~수만 종류의 패키지가 존재한다. 따라서 이미 만들어진 모듈/패키지를 사용하면 간단하게 프로그램을 작성할 수 있다.

▶ 모듈 (module): 재사용하고자 하는 변수나 함수의 정의문들을 파일로 저장한 것으로, 다른 프로그램이나 쉘 환경에서 호출하여 사용할 수 있다.

▶ 패키지 (package): 보통 계층구조를 가지는 여러 모듈들을 모아 놓은 것을 패키지라고 한다.

- ▷ 전체 리스트: <https://pypi.org/>
- ▷ Python Standard Library:
<https://docs.python.org/3/library/>
- ▷ 유용한 패키지 리스트:
<https://wiki.python.org/moin/UsefulModules>

2. 모듈을 사용하기 위해 알아야 하는 개념: 네임스페이스(namespace)

- ▶ 네임스페이스(namespace): '명칭공간'으로 번역되기도 하며, 변수, 함수 등이 정의되어 있는 공간을 지칭하는 표현이다.
- ▷ 프로그램이 실행될 때, 컴퓨터는 전역 네임스페이스(global namespace)를 만들어 프로그램이 사용할 수 있도록 할당해준다.
- ▷ 프로그램에서 새로운 변수가 정의되면, 컴퓨터는 네임스페이스(namespace)에 변수의 값을 저장할 공간을 만든다.
- ▷ 프로그램의 실행 중 함수, 모듈을 불러오게 되면, 컴퓨터는 전역 네임스페이스(global namespace)와는 별개로 그 함수, 모듈만을 위한 독립적인 네임스페이스(이 네임스페이스의 이름에는 그 함수 혹은 모듈의 이름이 붙여진다.)를 만들어 그 함수, 모듈이 사용할 수 있도록 할당해준다.

3. 모듈/패키지 사용하기.

A. 모듈을 읽어들이는 방법

- ▶ import module_name의 명령어를 통해 프로그램에 모듈을 읽어 들일 수 있다.
- ▶ 보통 이미 만들어진 모듈(module)들은 .py의 확장자명을 가지고 있을 텐데, 이때 불러들일 모듈(module)의 파일명 ~~.py에서 확장자명 .py를 뺀 ~~가 module_name이 된다.

B. 모듈 사용방법

▶ xxx가 변수이름인 경우, module_name.xxx와 같이 명령어를 적으면, 모듈에 속한 변수 중 이름이 xxx인 변수를 가져와 global namespace에 module_name.xxx로 저장한다.

▶ xxx가 함수이름인 경우, module_name.xxx와 같이 명령어를 적으면, 모듈에 속한 함수 중 이름이 xxx인 함수를 가져와 실행한다.

▶ 사용예시

▷ module_name.function_name()

▷ module_name.variable_name

▶ 예제: math module의 사용

Python Code Script	Terminal
# using math	3.141592653589793
import math	2.718281828459045
	2.718281828459045
# constants	0.0
print(math.pi)	0.0
print(math.e)	1.0
	0.5403023058681398
# power and logarithmic functions	0.8414709848078965
x =1.0	
print(math.exp(x)) # return e**x	
print(math.log(x)) # return natural log(x)	
print(math.log10(x)) # return log(x) with base 10	
print(math.sqrt(x)) # return square root(x)	
print(math.cos(x)) # return cos(x)	
print(math.sin(x)) # return sin(x)	

▶ math.py module을 사용하는 방법에 관한 설명은 <https://docs.python.org/3/library/math.html>에 자세히 설명되어 있다.

IV. 함수(function)

i . 함수

1. 함수 소개

A. 함수란?

▶ 함수(function or subroutine/procedure)란?

- ▷ 특정 값을 입력 받아 명령문들의 집합을 실행하여 결과값을 출력하는 역할을 하는 부분(코드)이다.
- ▷ 특정한 기능을 하는 명령문들의 집합이다.

▶ 함수를 사용하는 이유

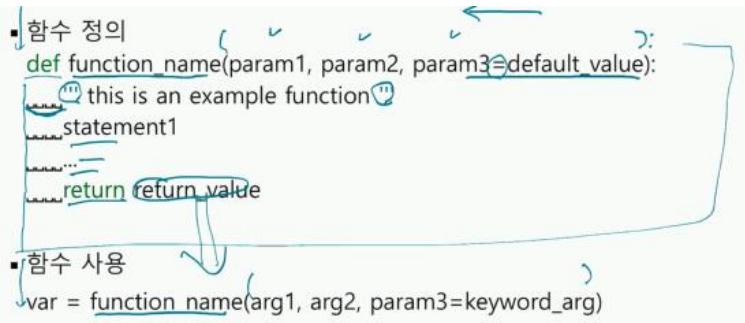
- ▷ 공통된 부분의 코드를 재사용할 수 있도록 하기 위해서 사용한다.
- ▷ 코드의 가독성을 높이기 위해 사용한다.
- ▷ 반복과 오류를 줄이기 위해 사용한다.
- ▷ 추상화(프로그래밍 코드에 추상적 역할을 할당하는 것)를 제공하기 위해 사용한다.

2. 함수의 정의와 인자 전달 방법

A. 함수의 정의와 사용

▶ 함수의 정의

Python programming script	설명
def function_name(parameters): statement1 statement2 ... return return_value	<ul style="list-style-type: none">▶ def ~: 함수명과 매개변수 (parameter, 함수 내부에서 사용되는 입력변수의 이름) 부분이다.▶ statement1...: 함수가 호출될 때마다 실행되는 코드블럭이다. (들여쓰기로 코드블럭 부분을 구분하게 된다.)▶ return ~: 반환값 (return value)를 돌려주기 위해 return문을 사용한다.



▶ 함수의 사용

Python programming script	설명
function_name(arguments)	<ul style="list-style-type: none"> ▶ 이렇게 함수 실행구문이 있으면, 컴퓨터는 글로벌네임스페이스에서 이게 무엇인지 찾는 작업을 하고, 이것이 함수라는 것을 알게 되면 위치를 찾고, 이미 함수가 정의된 그 위치로 가서 함수를 실행하게 된다. ▶ 함수 실행구문의 전달인자 (arguments)는 함수를 호출할 때 함수에게 전달하는 값을 말한다.

B. 함수 이름에 대한 규칙

▶ 변수명과 동일한 규칙이 적용된다. 즉, 알파벳(문자), 숫자, _로 이름을 정해야 하며, 알파벳 혹은 _로 이름이 시작해야 한다.

▶ python의 예약어(내장 변수 등으로 정의되어 있는 문구)들은 함수이름으로 사용할 수 없다. 그 예시로 True, False, None, in, is, and, or, not, nonlocal, global, if, elif, else, for, while, break, continue, def, return, lambda, yield, class, import, from, as, with, finally, try, exceptraise, assert, pass 등이 있다.

▶ python의 예약어에 무엇이 있는지 더 자세하게 알고 싶다면, docs.python.org에 접속해보도록 한다.

▶ python 내장 함수의 이름을 사용하는 것은 가능하지만, 오류가 빈번하게 발생할 수 있어 매우 권장하지 않는 방법이다.₩

C. python의 함수의 종류

▶ 내장 함수 (Built-in functions)

- ▷ python이 기본적으로 제공하는 함수들을 말한다.
- ▷ print(), input(), max(), min() 등이 있으며, 3.7 버전 기준 69개의 내장함수가 있다.
- ▷ python의 내장함수에 무엇이 있는지 더 자세하게 알고 싶다면, docs.python.org의 functions 항목을 참고하도록 한다.

▶ 라이브러리 함수 (library functions)

- ▷ 모듈 및 라이브러리를 사용할 때, 모듈 및 라이브러리에 정의되어 있는 함수들을 말한다.
- ▷ math 패키지의 exp(), log()가 그 예시가 된다.

▶ 사용자 정의 함수 (user-defined functions)

- ▷ 사용자가 직접 정의한 함수들을 말한다.

D. 함수를 정의하는 위치

- ▶ 함수를 부르기 전에 먼저 정의되어야 한다. 일반적으로는 파일 앞부분에 함수를 정의한다.
- ▶ 규모가 큰 프로그램을 짜게 될 경우에는, 논리적 구조에 따라 여러 파일에 코드를 나누고, class 혹은 module 등을 사용하여 파일들을 이어준다.

E. 예제

▶ 예제1: 함수의 정의와 호출

Python Script	Terminal
def say_hello(): print('Hello, World!') say_hello() print('Hi, python!') say_hello()	Hello, World! Hi, python! Hello, World!

▶ 예제2: 함수의 정의와 호출: 매개변수/인자와 반환값이 있는 경우

Python Script	Terminal
<pre>def circle_area(r): pi = 3.141592 area = r*r*pi return area circle_2 = circle_are(2) print(circle_2) circle_5 = circle_are(5) print(circle_5)</pre>	12.566368 78.5398

▶ 예제3: 위치를 이용한 인자 전달

Python Script	Terminal
<pre># 함수에 매개변수를 2개 정의하여, 정의 순서에 따른 값 2개를 전달하는 방법 def power(base, exponent): return base ** exponent # 전달 인자의 개수를 사전에 지정하지 않고 tuple을 이용하여 동적(개수에 상관없이 들어가는 값을 tuple로 엮는 것)으로 전달하는 방법 def power2(*args): base, exponent = args #함수 내부에 tuple을 정의함. return base ** exponent result = power(2, 2) print(power(2, 10)) print(power2(2, 10))</pre>	1024 1024

F. 매개변수의 기본값 지정

▶ 함수를 정의할 때, 매개변수의 기본값(default value)을 지정 할 수 있다.

▷ 기본값(default value)을 지정하는 방법은 매개변수 이름 뒤에 = default_value의 형태로 기본값(default value)을 덧붙이는 것이다.

- ▷ 이렇게 매개변수의 기본값을 지정하면, 함수를 호출할 때, 인자를 넘겨주지 않은 경우에는 기본값을 사용하게 된다.
- ▶ 다만, 함수를 정의할 때, 기본값이 있는 매개변수들은 반드시 기본값이 없는 매개변수들 뒤에서 값을 받을 수 있도록 정해주어야 한다.
- ▶ 기본값이 있는 매개변수들을 여러 개 정의하고 인자를 매개변수의 개수만큼 주지 않으면, 앞쪽에 있는 기본값이 있는 매개변수는 인자를 받아 사용하고 뒤쪽에 있는 기본값이 있는 매개변수에 인자를 넘겨주지는 않는다. 그래서 뒤쪽에 있는 인자 를 못받은 매개변수들은 기본값을 사용하게 된다.

▶ 예제

Python Script	Terminal
def add1(value1, value2=1, value3=2): return value1 + value2 + value3	1089
print(add1(42, 1024, 23)) print(add1(42, 1024))	1068 45
print(add1(42))	

G. 키워드를 이용한 인자전달

- ▶ 그런데, 컴퓨터가 매개변수를 읽는 순서를 고려하여 함수를 정의하고 함수를 부를 때는 중간의 인자를 비우고 앞, 뒤의 인자를 준다는 것은 불가능하다. 그래서 사용하는 것이 매개변수 이름(키워드)을 사용해서 직접 값을 할당해주는 것이다.
- ▶ 이렇게 매개변수 이름(키워드)을 사용해서 직접 값을 할당해 주면 순서에 상관없이 값을 할당해줄 수 있다.
- ▶ 이 장점에 따라 디폴트 매개변수가 여러 개 있을 때, 그 중 일부 매개변수에만 인자를 넘길 때 주로 사용하는 방법이다.

▶ 예제

Python Script	Terminal
def power(base, exponent): return base**exponent	1024
print(power(2, 10))	1024
print(power(base=2, exponent=10))	1024
print(power(exponent=10, base=2))	

H. 하나의 키워드로 많은 인자 전달

- ▶ 또한, 인자가 몇 개 들어올지는 모르지만, 한 개의 키워드로 받고, 함수를 처리하고 싶다면, 함수를 정의할 때, 키워드 앞에 *를 사용한다.

- ▶ 예제: print 함수의 정의문

```
print(*objects, sep='', end='\n', file=sys.stdout, flush=False)
```

I. print() 함수의 실행문 전체구조 분석

- ▶ print() 함수 실행문: 인자들을 출력할 때 인자들 사이에 sep, 가장 마지막에 end를 출력하도록 하는 구조로 이루어져 있다.

- ▶ print(*objects, sep='', end='\n', file=sys.stdout, flush=False)

▶ 사용 예시

Python Script	Terminal
print(1, 2, 3)	1 2 3
print(4, 5)	4 5
print(1, 2, 3, end="")	1 2 34 5
print(4, 5)	123
print(1, 2, 3, sep="")	1hey2hey3
print(1, 2, 3, sep='hey')	
print() #blank line	After one blank line
print('After one blank line')	
print('\n') #two blank line	
print('After two blank lines')	After two blank lines

- ▶ print함수의 사용에 대한 예시나 실행문에 대한 분석은 <https://docs.python.org/3/library/functions.html#print>에 더 자세히 설명되어 있으므로 참고하도록 한다.

- ▶ 이처럼, python에서 다른 사람이 만들어 놓은 함수를 사용할 때는 함수에 어떤 인자를 주어야 하는지, 줄 수 있는지를 미리 확인하고 사용해야 한다.

3. 네임스페이스와 지역 변수

A. 네임스페이스(namespace): 변수, 함수 등이 정의되어 있는 공간을 말한다.

B. Python에서의 네임스페이스의 종류 및 생성/소멸

▶ 전역 네임스페이스 (global namespace): 프로그램이 실행될 때 생성되어 내장 함수와 전역 변수(함수 외부에서 정의된 변수)를 저장한다.

▶ 지역 네임스페이스 (local namespace): 함수가 호출될 때 생성되어 함수가 받는 매개 변수와 함수 안에서 정의된 변수(지역변수)를 저장한다. 그리고 함수의 실행이 끝나면 소멸된다.

C. 변수의 범위(scope)

▶ 변수의 범위(scope): python에서 네임 스페이스, 즉 변수 혹은 함수가 접근 가능한 영역을 변수의 범위(scope)라고 부른다.

▶ 변수 혹은 함수가 사용될 때, 컴퓨터는

▷ 우선 지역 네임스페이스에서 해당하는 변수 혹은 함수를 찾는다.

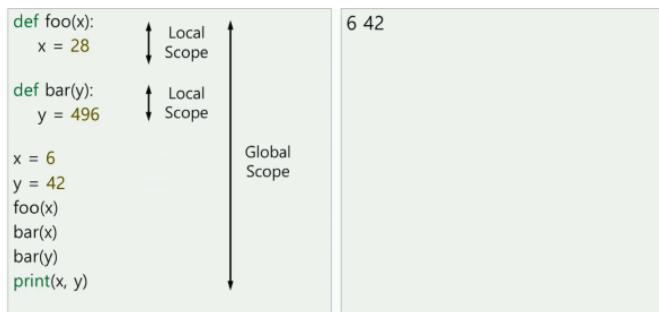
▷ 지역 네임스페이스에 그 변수 혹은 함수가 없으면, 그 다음으로 전역 네임스페이스에서 그 변수 혹은 함수를 찾는다.

▷ 전역 네임스페이스에서도 그 변수 혹은 함수를 찾을 수 없다면 NameError가 발생한다.

▶ 예제: 매개변수의 이름

Python Script	Terminal
def square(x): return x*x def square(y): return y*y value=square(3)	

▶ 예제: 지역 변수와 지역 변수의 범위(Scope)



▷ $x=28$ 은 $\text{foo}(x)$ 라는 함수 안에서만 영향을 주고, $y=496$ 은 $\text{bar}(y)$ 라는 함수 안에서만 영향을 준다.

▷ 이게 무슨말인가 하면 기본적으로 지역 네임스페이스는 함수가 끝나면 제거되기 때문에, $\text{foo}(x)$ 실행이 끝나면 $x=28$ 이 제거되고, $\text{bar}(y)$ 가 끝나면 $y=496$ 이 제거된다. 따라서 함수 밖에서 x 와 y 값을 출력하게 되면, 6과 42가 각각 출력된다는 것이다.

4. return문(return statement)과 반환값

A. 함수 내부에서 명령문 중 리턴문을 수행하면, 반환값을 돌려주고 함수의 실행을 종료하는 작업이 이루어진다.

- ▶ python이 지원하는 모든 객체가 반환값이 될 수 있다.
- ▶ 1개 이상의 값을 반환할 때는 tuple 자료형을 사용하여 반환하도록 한다.

- B. return문이 없거나 return문 뒤에 값을 넣어주지 않아 반환값이 없는 함수가 정의되어 있고, 이 함수가 실행되었을 때는 컴퓨터는 반환값이 없는 것을 나타내기 위해 None을 반환한다. 이 때, None은 문자열이 아닌, python 내부의 특수한 값이다.

C. 예제: return문

```
def age_group(age):
    if age <= 12:
        return 'childhood'
    elif age <= 18:
        return 'adolescent'
    elif age <= 65:
        return 'adult'
    else:
        return 'aged'

# the following line will not be executed
print('The last line of age_group()')

print(age_group(42))
```

adult

이 예제를 실행하면 print('The last line of age_group()')은 무슨일이 있어도 실행되지 않는다. 그 이유는 무슨 일이 있어도 조건문에서 리턴문을 만나 리턴값을 반환하고 함수를 종료하기 때문이다.

D. 예제: return문 with tuple

```
def divider(a, b):
    return a//b, a%b

print(divider(10, 3))

q, r = divider(10, 3)
print(q, r)
```

(3, 1)
3 1

E. 예제: 반환값이 없는 경우

<pre>def foo(): pass def bar(): return print(foo()) print(bar())</pre>	None None
--	--------------

▶ 코드블럭이 시행되면 무조건 코드블럭을 1개라도 채워주어야 하는데, 딱히 의미없는 코드블럭으로 두고 싶으면 pass명령문을 쓰도록 한다.

▶ pass 명령문은 아무 일도 하지 않고, 보통 코드블록이 필요한 경우에 채우는 용도로 사용한다. 같은 이유로 placeholder를 사용하기도 한다.

5. 함수가 호출될 때, 수행되는 작업 요약

- A. 인자가 있는 경우, 인자의 값을 먼저 계산한다.
- B. 함수를 위한 독립적인 지역 네임스페이스를 생성한다.
- C. 매개변수가 있는 경우,
 - 1. 새로 만들어진 네임스페이스에 매개변수를 생성한다.
 - 2. 호출될 때 제공된 인자 혹은 디폴트 인자를 매개변수에 대입한다.
- D. 함수 내부의 명령문을 실행한다.
 - 1. 함수 내부에서는 무의미한 명령문도(ex. pass, placeholder) 실행될 수 있다.
- E. 함수의 코드블럭의 끝 혹은 return문이 있는 경우 함수의 실행을 종료한다.
- F. 반환값 혹은 None을 함수를 호출한 곳으로 반환한다.
 - 1. 반환된 값은 호출한 곳, 즉 함수가 표현식에 사용된 곳에서 함수값(함수의 결과값)으로 사용된다.

G. 지역 네임스페이스를 소멸시킨다.

6. lambda 함수의 정의와 사용

A. lambda 함수

- ▶ 익명으로 함수를 정의하고 싶을 때 `lambda` 키워드를 이용하여 정의하는 함수이다.
- ▶ 한 줄로 표시되는 함수이다.
- ▶ 비교적 간단한 기능의 함수를 정의할 때 유용하며, 다른 함수의 인자로 함수를 넘겨줄 때 유용하다.
- ▶ `filter`, `map` 등의 함수와 자주 함께 사용된다.

B. lambda 함수 표현식

- ▶ **lambda parameters: expression**
- ▶ 예제:

```
def square(x):
    x_squared = x * x
    return x_squared

print(square(10))
```

위의 `square`와 같은 간단한 함수가 필요할 때, `lambda` 함수를 사용한다.

`lambda`함수를 사용하면, 다음과 같이 정리할 수 있다.

```
square = lambda x: x * x
print(square(10))
```

그런데 이 `lambda`함수의 장점은 같은 기능을 하는 함수를 다음과 같이 더 간단히 정리할 수 있다는 것이다.

```
print((lambda x: x*x)(10))
```

V. 자료구조와 반복문 1: list

i. 주석 (comment)

1. 주석의 필요성

▶ 주석은 다른 사람이나 스스로가 코드를 이해할 수 있도록 (코드의 가독성을 높이기 위해서) 코드에 대한 설명을 해놓은 것이다.

▶ 함수가 정의된 후 코드블럭의 첫줄에 주석이 적혀있는 경우 함수의 역할을 설명하는 주석이라는 것이 약속되어 있다.

▪ [참고] 함수 설명문

```
def function_name(parameters):
    """this is an example function"""
    statements
    ...
    return return_value
```

```
>>> help(print)
Help on built-in function print in module builtins:
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
        file: a file-like object (stream); defaults to the current sys.stdout.
        sep: string inserted between values, default a space.
        end: string appended after the last value, default a newline.
        flush: whether to forcibly flush the stream.
>>>
```

2. 주석을 넣는 방법

▶ #을 이용하면, #부터 그 줄의 끝까지는 주석으로 처리된다.

▶ 여러 줄의 주석을 다는 경우

▷ 모든 줄에 #을 표시한다.

▷ 따옴표 3개를 사용하여(" "), 여러줄에 걸친 문자열을 사용한다.

3. 주석을 넣을 때의 주의점

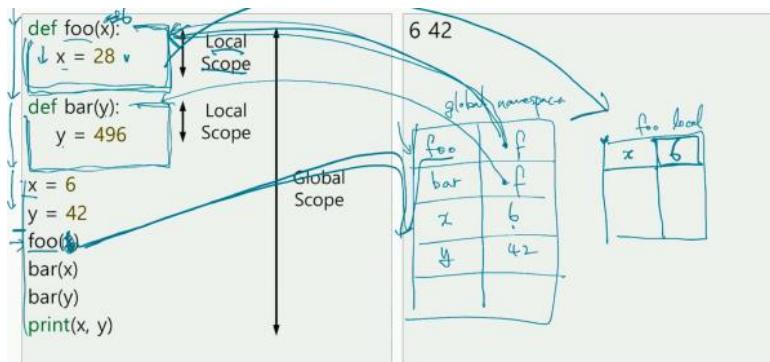
▶ 주석은 코드에 대해 설명하는 것이므로, 코드와 일관되도록 적어주어야 한다.

▶ 주석을 잘못 넣으면, 오히려 코드를 이해하는 것이 더 어려워진다.

▶ 코드블럭 안에 주석을 넣을 때는 주석도 코드블럭 안에 있음을 표시하기 위해 띄어쓰기 4칸을 한 후에 #을 쓰거나 ""을 써서 주석을 처리하도록 한다.

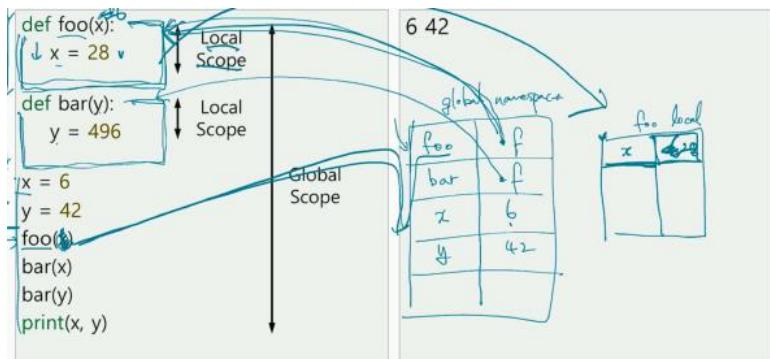
ii. 전역, 지역 변수

1. Remind



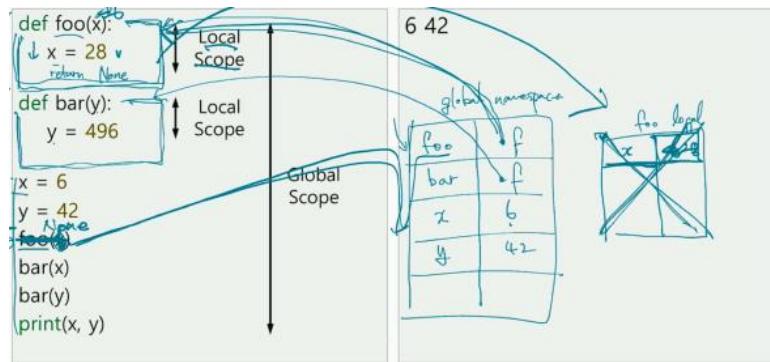
▶ 프로그램의 시작과 동시에 Global name Space가 형성되어 foo는 함수명이라는 것, bar은 함수명이라는 것, x는 6이라는 것, y는 42라는 것을 기록함.

▶ foo 함수가 실행되면, foo local namespace를 만들어 x=6, y=496이라는 Global namespace에 있는 것들을 기록함.



▶ 그런데, foo 안에서 x가 있는 것을 보고 foo local namespace에서 x라고 이름붙은 변수를 찾음. 있으면, x=28로

다시 정의하고, 없으면, Global namespace에서 x를 찾음.



- ▶ foo 함수가 끝나면, return 표시가 없다면 return None으로 인식하여 함수 실행문 자리에 None을 출력하고, foo namespace를 삭제함.

2. 함수에서 전역 변수를 설정하는 방법

```
var = 40
def fn():
    global var
```

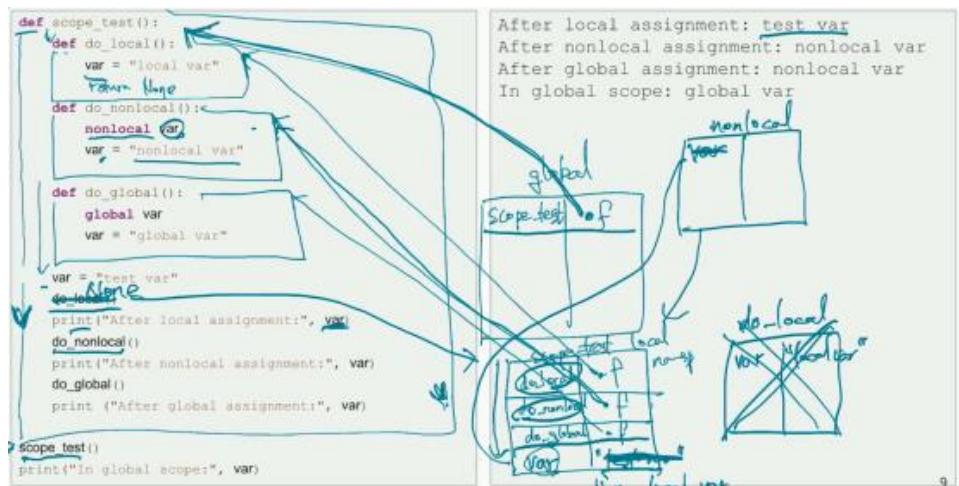
- ▶ 위와 같이 함수 안에서 global var라고 쓰면, 전역변수 namespace에 가서 var라고 적힌 변수를 직접 사용(수정or 계산)한다.

3. 중첩 함수에서 지역 변수를 설정하는 방법

```
def outer_fn():
    var = 40
    def inner_fn():
        nonlocal var
```

- ▶ 위와 같이 함수 안의 함수에서 nonlocal var라고 쓰면, nonlocal var를 쓴 함수의 부모 함수들 중에서 var라는 변수가 정의된 함수의 local name space에 가서 var라고 적힌 변수를 직접 사용(수정 or 계산)한다.

4. 예제: 지역 변수와 변수의 범위(scope)



iii. 자료구조

1. 자료구조: 자료를 구조화하고 저장하는 방식을 자료구조라고 한다.

- ▶ 하나의 변수에 하나 이상의 값 혹은 객체를 저장하기 위하여 주로 사용된다.
- ▶ Python은 특성이 다른 여러 형태의 자료구조를 제공한다.
 - ▷ 리스트(list)
 - ▷ 튜플(tuple)
 - ▷ 사전(dict)
 - ▷ 집합(set)

2. 시퀀스(sequence)

- ▶ 시퀀스는 python의 자료구조의 종류의 하나이다.
 - ▷ 변경가능한 시퀀스(mutable): list
 - ▷ 변경 불가능한 시퀀스(immutable): tuple, str

▶ 시퀀스의 공통적인 특징

- ▷ 유한한 순서가 있는 집합으로, 음이 아닌 정수로 그 순서(index)를 나타낸다.

- ▷ 슬라이싱(slicing, 시퀀스의 부분집합)이 제공된다.
- ▷ 유사한 형태의 여러 연산자 혹은 함수가 제공된다.

3. 리스트(list)

A. 정의: mutable sequences, typically used to store collections of homogeneous items

- ▶ **mutable:** 원소의 추가/삭제 혹은 값이 변경될 수 있음.
- ▶ 원소의 자료형: 임의의 자료형의 원소를 저장할 수 있음.

B. 인덱스(index): 음이 아닌 정수로 원소의 순서를 나타낸다.

C. 표기법: []를 사용하고 원소는 ,로 구분한다.

```
# defining lists
t1 = [42, 1024, 23] # a list with 3 elements
print(t1)

t2 = [] # empty list
print(t2)
```

[42, 1024, 23]
[]

- ▶ 빈 리스트도 안에 데이터만 없을 뿐 리스트 자료형이 맞다.

D. list의 원소 접근

- ▶ 리스트의 원소는 0부터 시작하는 인덱스로 접근할 수 있다.
- ▶ 음수가 사용되면, 가장 마지막 원소부터 역순으로 접근 한다.

```

t1 = [42, 1024, 23]
print(t1[0])
print(t1[1])
t1[2] = 7
print(t1[2])
print(t1)
print(t1[-1])
print(t1[-2])
print(t1[-3])

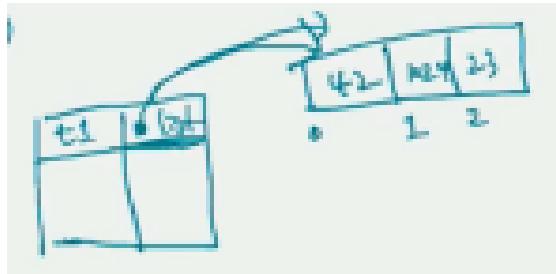
```

```

42
1024
7
[42, 1024, 7]
7
1024
42

```

- ▶ 리스트는 namespace에서 list로써 저장되어 있으며, 그 list가 불려지면, 그 list를 저장하고 있는 다른 메모리 공간에 가서 list를 불러온다. list가 저장된 다른 메모리 공간에 list는 인덱스가 0부터 오름차순으로 데이터에 붙어서 정렬되어 있다.



E. list의 여러 원소들 접근: 슬라이싱(slicing)

▶ name_of_list[start: end]

- ▷ name_of_list의 [start: end]에 위치한 원소들을 가지는 리스트를 생성한다. start번째 인덱스를 가진 데이터부터 end번째 인덱스를 가진 데이터 전까지 가져온다.
- ▷ start: 생략되면 첫 원소부터 가져온다.
- ▷ end: 생략되면 마지막 원소까지 가져온다.
- ▷ start와 end 모두 생략되면, 해당하는 리스트 전체를 다 가져온다.

```

t = [42, 1024, 23, 6, 28, 496]
print(t)
print(t[1:4])
print(t[3:])
print(t[:2])
print(t[:]) # copy of a list*

```

```

[42, 1024, 23, 6, 28, 496]
[1024, 23, 6]
[6, 28, 496]
[42, 1024]
[42, 1024, 23, 6, 28, 496]

```

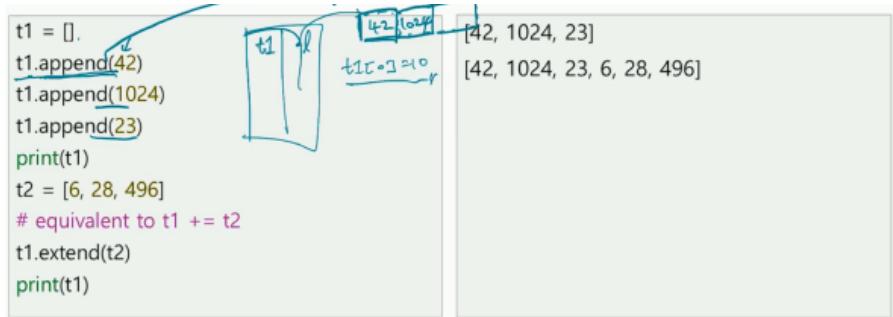
F. list에 대한 연산(이 연산 중 대부분은 다른 시퀀스(str, tuple)에도 적용된다.)

<pre>t1 = [42, 1024, 23] print(1024 in t1) print(7 in t1) print(1024 not in t1) print(len(t1)) # the number of elements print(min(t1)) # the minimum value print(max(t1)) # the maximum value t2 = [6, 28, 496] t3 = t1 + t2 # concatenation print(t3) t4 = t2 * 2 # repetition print(t4)</pre>	<pre>True False False 3 23 1024 [42, 1024, 23, 6, 28, 496] [6, 28, 496, 6, 28, 496]</pre>
---	---

- ▶ in: in 좌측의 데이터가 in 우측의 시퀀스 안에 있는가를 연산하는 불연산자. 있으면 True 없으면 False를 뱉는다.
- ▶ not in: not in 좌측의 데이터가 not in 우측의 시퀀스 안에 없는가를 연산하는 불연산자. 없으면 True, 있으면 False를 뱉는다.
- ▶ len(): ()안의 시퀀스의 데이터의 개수를 출력한다.
- ▶ min(): ()안의 시퀀스의 제일 작은 데이터를 출력한다.
- ▶ max(): ()안의 시퀀스의 가장 큰 데이터를 출력한다.
- ▶ +: +왼쪽의 시퀀스 뒤에 +오른쪽의 시퀀스를 합친다.
- ▶ *: *왼쪽의 시퀀스를 *왼쪽의 시퀀스 뒤에 *오른쪽의 숫자만큼 더한다.

G. list에 원소 추가하기

- ▶ append(): list에 ()에 있는 원소를 추가한다.
- ▶ extend(): .extend()함수 앞의 list에 ()안의 list의 모든 원소를 추가한다.



iv. 반복문

1. 반복문

- ▶ 코드 중 일정부분을 반복시키고 싶을 때 사용하는 구문.

2. while문

- ▶ 어떤 조건이 만족되는 동안 반복하도록 하는 함수이다.
- ▶ 주로 반복횟수를 모를 때 사용한다.
 - ▷ 예를 들어 조건에 맞거나 맞지 않을 때까지 사용자 /파일/네트워크를 입력하도록 할 때 사용한다.

▶ while문의 제어흐름

Python Code	설명
<pre>while condition: statement1 statement2 else: statement3 statement4</pre>	<ul style="list-style-type: none"> ▶ Condition이 True일 때, while에 해당하는 코드블럭의 명령어들을 1번 수행하고 다시 돌아와서 condition을 계산한다. ▶ condition이 True인 경우 while에 해당하는 코드블럭의 명령어들을 한번 더 수행하고 다시 돌아와서 condition을 계산한다. ▶ else절 (else-clause): Condition이 False일 때, else절 아래의 코드블럭을 1번 수행하고, while문을 마친다. 생략가능한 코드 절이다.

- ▶ 예제: while

```

counter = 0
while counter < 3:
    print(counter)
    counter += 1

# similar to the following
counter = 0
print(counter)
counter += 1 # counter = 1
print(counter)
counter += 1 # counter = 2
print(counter)
counter += 1 # counter = 3

```

```

0
1
2

```

▷ 예제의 코드 중 `counter+=1`은 `counter+1=counter`를 줄여 쓴 표현이다.

3. for문

▶ 스퀀스의 모든 원소에 대해서 반복을 수행하도록 하는 함수이다.

▶ 주로 얼마나 반복을 해야되는지 알 경우에 사용한다.

▷ 예를 들면, 스퀀스에 있는 모든 원소들에 대해 작업을 수행할 때 사용하거나 n번 반복할 때 사용된다.

▶ while문과 for문을 모두 쓸 수 있는 경우에는 for가 선호되는 경우가 많다. for문은 반복을 몇번 할 것인지 직관적으로 알 수 있기 때문이다.

▶ for문의 제어 흐름

Python Code	설명
<pre> for variable in sequence: statement1 statement2 else: statement3 statement4 </pre>	<ul style="list-style-type: none"> ▶ Sequence에 있는 원소들을 순서대로 variable에 대입한 후, for문 아래의 코드블럭의 명령어들을 수행한다. ▶ for문 아래 코드블럭에서 Sequence에 있는 원소 각각에 대해 수행할 연산을 정의 할 수 있다. ▶ Sequence에 저장된 모든 원소들에 대해 연산을 수행한 후, else 이후의 코드블럭을 수행하고, for문을 종료한다. ▶ else절은 생략 가능하다.

▶ 예제: for

```
# a simple for loop
for value in [42, 1024, 23]:
    print(value)

# similar to the following
value = 42
print(value)
value = 1024
print(value)
value = 23
print(value)
```

42
1024
23

4. range() 함수

▶ Range() 함수

▷ range(stop): immutable sequence(불변의 시퀀스) |
($0 \leq i < stop$) 즉, 0부터 stop에 해당하는 숫자까지 i값
만큼의 간격으로 나누어 한줄로 저장하는 함수. 0은 무
조건 포함하고, 0에 i를 더해가면서 stop보다 작은 숫
자이면 0 뒤에 이어서 저장하도록 하는 함수이다. 따
라서 stop 값은 포함되지 않는다. 보통 i는 1이다.

▷ range(start, stop, step): immutable sequence |
($start \leq i < stop, i += step$) 즉, 0부터 stop에 해당하는 숫
자까지 step값만큼의 간격으로 나누어 한줄로 저장하
는 함수. 0은 무조건 리스트에 포함하고, 0에 step 값을
더해가면서 stop보다 작은 숫자이면 0 뒤에 이어서 저
장하도록 하는 함수이다. 따라서 stop 값은 리스트에
포함되지 않는다.

```
# range() example
print(range(5))
print(list(range(5)))
print(list(range(1, 4)))
print(list(range(1, 4, 2)))
print(list(range(1, 5, 2)))

print(list(range(1, -5, -2)))
print(list(range(1, 5, -2)))
```

range(0, 5)
[0, 1, 2, 3, 4]
[1, 2, 3]
[1, 3]
[1, 3]
[1, -1, -3]
[]

▶ 반복문의 sequence로 활용되는 range()함수.

▷ 특정 회수를 반복하거나 특정 정수의 범위에 대한 반복을 하기 위해 사용한다.

▷ range()함수는 리스트를 반환한다고 생각하는 것은 엄밀히 말하면 틀린 것이지만, 프로그램 실행을 이해하는 것에는 도움이 됨.

```
# using range() with for loop
for index in range(5):
    print(index)

for index in range(3, 10, 3):
    print(index, end=' ')
```

0	
1	
2	
3	
4	
3 6 9	

▶ 예제: for loop with range() and len()

▷ 리스트의 인덱스를 이용한 반복을 하기 위해서 주 사용되는 구문이다.

▷ 리스트의 각 원소의 값을 변경하는 경우 주로 사용된다.

```
t = [42, 1024, 23]
for i in range(len(t)):
    print('t[' + str(i) + '] = ' + str(t[i]), sep=' ')
    t[i] = t[i] * 2

print(t)
```

t[0] = 42	
t[1] = 1024	
t[2] = 23	
[84, 2048, 46]	

v. 함수를 호출 할 때, 인자로 리스트를 전달하는 방법

```

def display(sequence):
    for v in sequence:
        print(v, end=' ')
    print()

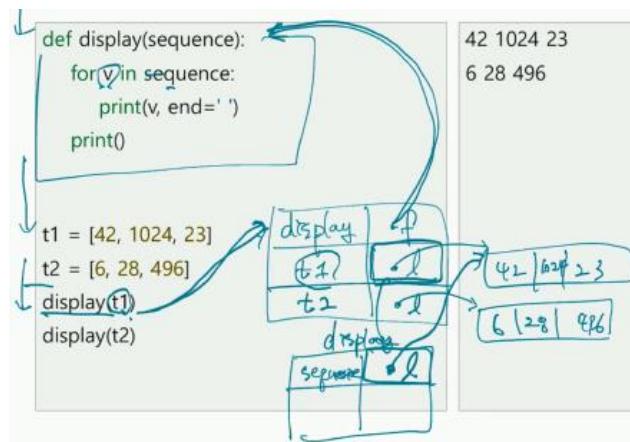
```

```

t1 = [42, 1024, 23]
t2 = [6, 28, 496]
display(t1)
display(t2)

```

42 1024 23
6 28 496



27

vi. 함수의 반환값으로 리스트를 사용하는 방법

```

def divider (nums):
    result = (nums[0]/nums[1],nums[0]%nums[1])
    return list (result)

```

```

div = [10, 3]
result = divider(div)
print("quotient:", result[0])
print("remainder:", result[1])

```

quotient: 3
remainder: 1

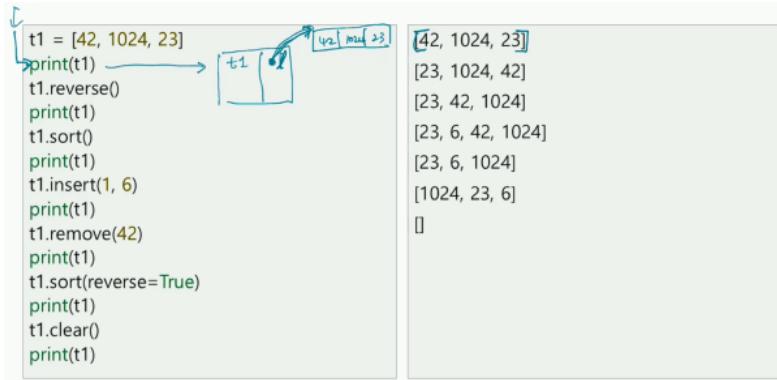
VI. 자료구조와 반복문 2: 문자열 처리와 list, str, tuple

i. 자료구조

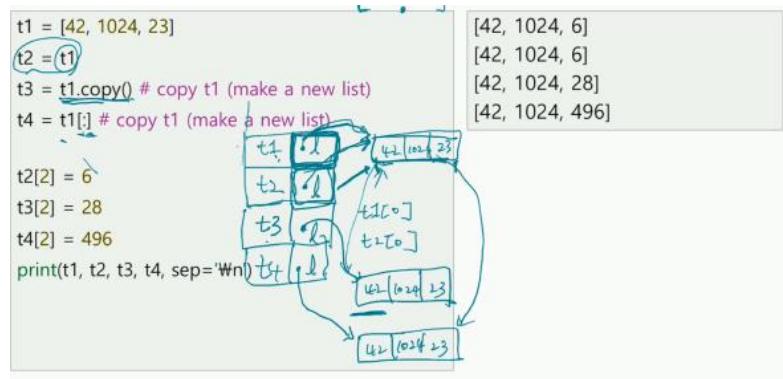
1. 리스트(list): 추가내용

A. 리스트가 지원하는 함수 (일부)

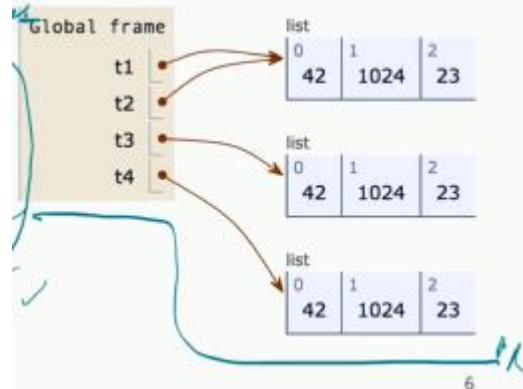
- ▶ `list.insert(i, x)` : 주어진 `i`라는 인덱스 위치에 `x`라는 항목을 삽입하고 본래 `i`번째 이후의 인덱스 위치의 항목을 `i+1`번째 이후로 미루는 함수이다. 그래서 `list.insert(0, x)`는 리스트의 제일 첫 자리에 `x`를 삽입하는 함수이고, `list.insert(len(list), x)`는 `list.append(x)`와 같은 작업을 하는 함수이다.
- ▶ `list.remove(x)` : 리스트에서 값이 `x`와 같은 항목을 인덱스 0번부터 오름차순으로 검색하여 나오는 첫번째 `x` 항목을 삭제하고 그 다음 인덱스들을 한 칸씩 앞으로 이동시키는 함수이다. 그런 항목이 없으면 `Value Error`를 일으킨다.
- ▶ `list.clear()` : `list`의 모든 항목을 삭제해서 빈 리스트를 만드는 함수이다. `del.list[:]와 같은 작업을 하는 함수이다.`
- ▶ `list.sort(key=None, reverse=False)` : 리스트의 항목들을 알파벳 오름차순 혹은 숫자 오름차순으로 정렬하는 함수이다. `reverse`를 `True`로 바꾸면 내림차순으로 정렬하게 된다.
- ▶ `list.reverse()` : 리스트의 요소들의 순서를 뒤집고 `None`를 `return`하는 함수이다. 다시말해 0번 리스트가 -1번 리스트로, 1번 리스트가 -2번 리스트로해서 요소들의 순서를 역으로 바꿔주는 함수이다.
- ▶ `list.copy()` : 리스트의 얕은 사본(shallow copy, 껍데기만 복사한 것)을 돌려준다. `list[:]와 같은 작업을 하는 함수이다. (얕은 사본, shallow copy의 반대말은 깊은 사본, deep copy로 내부까지 전부 복사한 것을 말한다.)`
- ▶ 추가적인 내용:
<https://docs.python.org/3/tutorial/datastructures.html>
- ▶ 예시: 리스트의 함수들(추가)



▶ 예시: 리스트의 복사, copy() 함수



▷ 이때, 이 예제가 실행될 때, namespace의 상태는 다음과 같다.



B. List Slicing (리스트의 부분을 잘라서 복사해서 가져오는 방법)

▶ list[i:j] : list의 i번 인덱스부터 j번 인덱스 전까지 잘라서 복사해서 가져온다.

▶ list[i:j:k] : list의 i번 인덱스부터 k의 step으로 띄운 인덱스 번째에 해당하는 데이터만 복사해서 j번 인덱스 전까지 작업한 리스트를 만들어서 가져온다.

<pre> lst = [1, 2, 3, 4, 5, 6] print(lst[:]) print(lst[0:6:2]) print(lst[1::2]) print(lst[-3:]) print(lst[-3:-6]) print(lst[-3:-6:-1]) </pre>	[1, 2, 3, 4, 5, 6] [1, 3, 5] [2, 4, 6] [4, 5, 6] [] [4, 3, 2]
---	--

C. List가 지원하는 함수 (추가)

- ▶ list.index(x[, i[, j]]) : i번 인덱스부터 j번 인덱스 사이에서 x를 검색하여 첫번째로 나오는 x라는 항목의 인덱스를 리턴하는 함수이다.
- ▶ list.count(x) : list에서 x라는 항목의 개수를 리턴하는 함수이다.

<pre> lst = [42, 1024, 23, 6, 23, 496] print(lst.count(23)) print(lst.count(3)) print(lst.index(23)) print(lst.index(23,3)) print(lst.index(3)) print(lst.index(23,3,4)) </pre>	2 2 0 4 ValueError: 3 is not in list
---	--

D. sorted() 내장 함수와 list.sort() 리스트 함수

- ▶ list.sort() : list.sort(key=None, reverse=False)의 형태로 사용되는 함수이다. list안의 요소들을 오름차순으로 정리하고 None을 리턴하는 함수이다. 즉, list 자체가 바뀌는 함수이다.
- ▶ sorted(list) : list안의 요소들이 오름차순으로 정리된 리스트를 리턴하는 함수이다. 즉, list 자체는 바뀌지 않고, 오름차순으로 정렬된 새로운 리스트가 생기는 함수이다.

<pre> t1 = [42, 1024, 23, 6, 28, 496] t1.sort() print(t1) t2 = [42, 1024, 23, 6, 28, 496] t3 = sorted(t2) print(t2) # 변경되지 않음 print(t3) </pre>	[6, 23, 28, 42, 496, 1024] [42, 1024, 23, 6, 28, 496] [6, 23, 28, 42, 496, 1024]
--	--

E. 예시: 함수 인자로 리스트를 사용하는 법 1

```

def double(arg):
    for i in range(len(arg)):
        arg[i] = arg[i] * 2
    return arg

```

[84, 2048, 6]
[84, 2048, 6]
[168, 4096, 28]

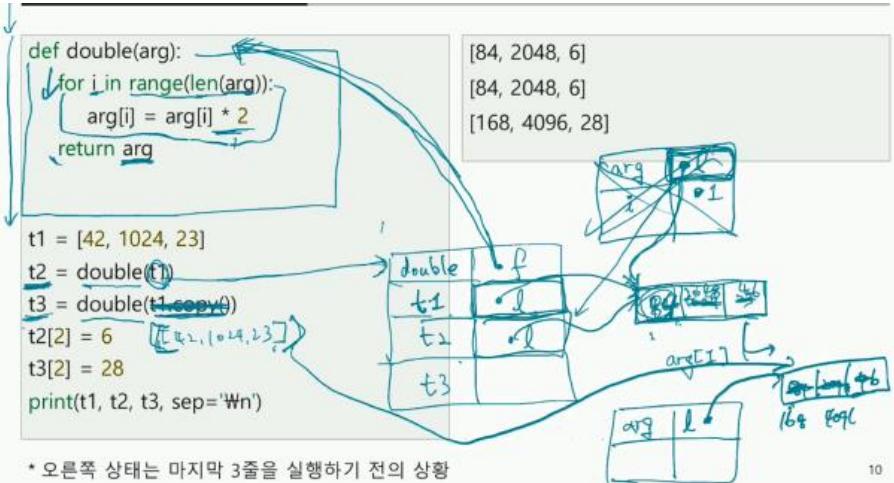
```

t1 = [42, 1024, 23]
t2 = double(t1)
t3 = double(t1.copy())
t2[2] = 6
t3[2] = 28
print(t1, t2, t3, sep='\n')

```

* 오른쪽 상태는 마지막 3줄을 실행하기 전의 상황

10



* 오른쪽 상태는 마지막 3줄을 실행하기 전의 상황

10

F. 예시: 함수 인자로 리스트를 사용하는 법 2

```

def double(arg):
    arg = arg.copy()
    for i in range(len(arg)):
        arg[i] = arg[i] * 2
    return arg

```

[42, 1024, 23]
[84, 2048, 46]

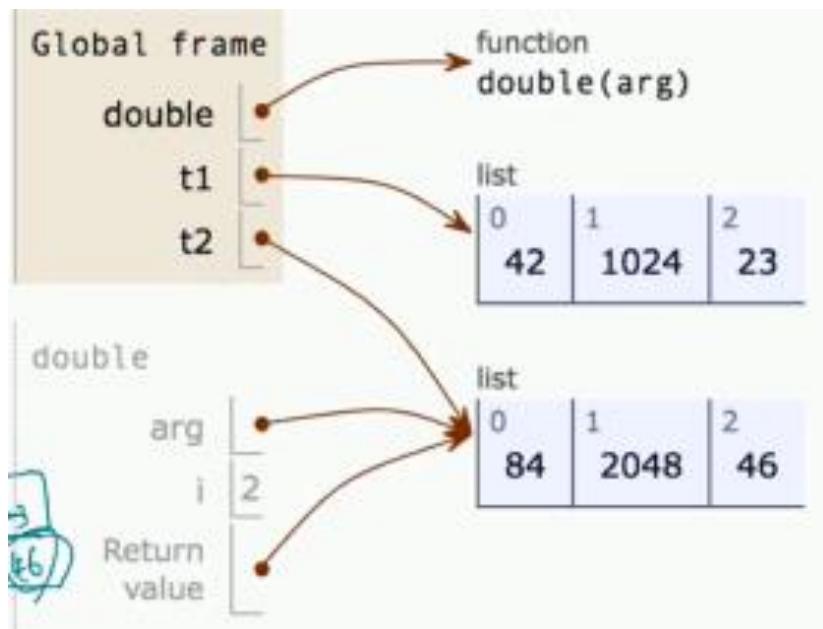
```

t1 = [42, 1024, 23]
t2 = double(t1)
print(t1)
print(t2)

```

* 오른쪽 상태는 마지막 2줄을 실행하기 전의 상황

11



2. 스트링(str)

A. 문자열을 나타내는 방법을 스트링(str)이라고 한다.

▶ 'abc' 와 같이 써도 되고, "abc"와 같이 써도 된다. 의미상으로는 동일하지만 한 프로그램 내에서는 일관성 있게 사용하는 것을 권장 한다.

▶ 여러줄에 걸친 문자열을 나타내고 싶다면, ""abc def""나 """abc def"""와 같이 따옴표를 3번 중첩하여 사용한다. 의미상으로는 ""나 """나 동일하지만, ""는 주석에서 사용되는 경우가 더 많기 때문에 """ 을 사용하는 것을 추천한다.

▶ 사실 여러줄에 걸친 문자열을 표현할 때는 ""abc def""나 """abc def"""보다는 'abc\ndef'가 더 선호되는 편이다. 이때, \n은 줄바꿈 예약어이다.

B. 이스케이프 시퀀스(escape sequence): ""보다는 '로 한 줄 안에 문자열을 쓰는 것을 선호하게 되어, 나오게 된 문자열 표현 예약어

Name	Symbol	Meaning
Newline	\n	Moves cursor to next line

Carriage return	\r	Moves cursor to beginning of line
Horizontal tab	\t	Prints a horizontal tab
Hex number	\x(number)	Traslates into char represented by hex number(컴퓨터에게 16진수를 인식시킬 때 쓴다.)
Single quote	'	Prints a single quote
Double quote	"	Prints a double quote
Backslash	\	Prints a backslash(컴퓨터에게 문자로서 \기호를 사용했다는 것을 인식시키기 위해 쓴다.)
Octal number \(number)	Octal number \(number)	Translates into char represented by octal(컴퓨터에게 8진수를 인식시킬 때 쓴다.)

- C. str에 대한 연산 (str의 문자 하나가 리스트의 인덱스 한 개에 할당되어 있다고 생각하면 된다. 다만, 리스트의 연산에서 원소를 바꾸는 연산이나 원소를 추가하는 연산은 str에서는 지원하지 않는다.)

<pre>s1 = 'Hello, world!' print(s1[0]) print(s1[1]) print(s1[1:4]) print('o' in s1) print('h' in s1) print(len(s1)) s2 = 'python' s3 = s1 + s2 print(s3) s4 = s2 * 2 print(s4)</pre>	H e ll True False 13 Hello, world!python pythonpython
--	--

- ▶ 여기서 독특한게 'x' in str인데, 이것은 해당 str에 'x'라는 문자가 있는지 연산하는 것으로 있으면, True, 없으면 False를 리턴하는 불연산이다.

- ▶ len(str)의 경우 단순 띄어쓰기나 , 같은 문자도 str의 원소로서 취급하여 총 문자의 갯수를 세는 연산이다.

3. 튜플(tuple)

A. 튜플(tuple)

- ▶ 정의: immutable sequences(수정 불가능한 시퀀스)이다. typically used to store collections of homogeneous items.

▷ immutable: 원소의 추가/삭제, 값의 변경이 불가능하다.

▷ 원소의 자료형: 임의의 자료형의 원소를 저장할 수 있다.

- ▶ 인덱스(index): 음이 아닌 정수로 원소의 순서를 나타낸다.

- ▶ 표기법: ()을 사용하고, 원소는 ,로 구분하여 정의한다.

▷ 정의할 때, ()는 생략 가능하다. 또한, 원소가 하나일 때는 Expression(수식)과 구분하기 위해서 ,를 뒤에 추가하여야 한다.

- ▶ 리스트와 유사한 점

▷ 인덱싱: 튜플도 인덱싱을 할 때는 []을 이용한다.

▷ 튜플은 리스트와 비슷한 연산 혹은 함수를 사용한다. 다만 원소의 변경은 없기 때문에, append, insert 등의 함수는 사용하지 못한다.

B. tuple에 대한 연산

<pre>t1 = (42, 1024, 6) t2 = (42,) # tuple with 1 element print(t1[1]) print(t1[1:3]) print(len(t1)) print(min(t1)) print(max(t1)) t2 = t1 + t1 t3 = t1 * 3 print(t2) print(t3) t1[2] = 3 # error t2.append(0) # error</pre>	<table border="1" style="margin-left: 20px; border-collapse: collapse;"> <tr><td style="padding: 5px;">t1</td><td style="padding: 5px;">t</td></tr> <tr><td style="padding: 5px;">t1</td><td style="padding: 5px;">t</td></tr> <tr><td style="padding: 5px;">t1</td><td style="padding: 5px;">t</td></tr> </table>	t1	t	t1	t	t1	t	1024 (1024, 6) 3 6 1024 (42, 1024, 6, 42, 1024, 6) (42, 1024, 6, 42, 1024, 6, 42, 1024, 6)
t1	t							
t1	t							
t1	t							

- ▶ 리스트와 마찬가지로 Slicing은 tuple[i:j]와 같이 사용하며, 리스트의 연산과 마찬가지로 i번 인덱스부터 j번 인덱스 전까지만 복제하여 새 튜플을 만들어 리턴하는 함수이다.

4. 튜플 언팩킹

- ▶ 튜플에 있는 값들을 변수에 대입하는 것을 튜플 언팩킹이라 한다.
 - ▷ 참고: 튜플을 정의하는 것을 튜플 패킹(tuple packing)이라 한다.)
 - ▷ 튜플 패킹/언팩킹 때 ()는 생략 가능함.
- ▶ 튜플 언팩킹 때, 원소의 개수와 변수가 일치해야 함.
- ▶ 참고: 다른 시퀀스(list, str, range 등)에서도 언팩킹이 지원됨.

```
my_tuple = (42, 1024, 23) # tuple packing
first, second, third = my_tuple # tuple unpacking
print('first = ', first, ', second = ', second, ', third = ', third, sep='')
```

first = 42, second = 1024, third = 23

* 가변 길이의 원소를 언팩킹하는 방법이 python3부터 지원됨

17

- ▶ 예시: 튜플/리스트/스트링 언팩킹

```
my_tuple = 42, 1024, 23
first, second, third = my_tuple
print(my_tuple)
print('first = ', first, ', second = ', second, ', third = ', third,
sep='')
print(type(my_tuple))
my_list = [42, 1024, 23]
(a, b, c) = my_list
print('a = ', a, ', b = ', b, ', c = ', c, sep='')
string = 'Wow'
(a, b, c) = string
print('a = ', a, ', b = ', b, ', c = ', c, sep='')
```

```
(42, 1024, 23)
first = 42, second = 1024,
third = 23
<class 'tuple'>
a = 42, b = 1024, c = 23
a = W, b = o, c = w
```

- ▷ 언팩킹 때, 원소의 개수와 변수가 일치해야 한다는 것에 주의해야 한다!!

- ▶ 예시: 변수의 값 교환(swap)

```

"""Swap two value
1. make a tuple (var2, var1) with values of var2 and var1
2. unpack the tuple to var1 and var2
"""

var1 = 42
var2 = 1024
var1, var2 = var2, var1
print(var1, var2)

```

1024 42

▶ 예시: 함수의 반환값으로 튜플 사용

```

def divide(dividend, divisor):
    """Return (quotient, remainder)"""
    quotient = dividend // divisor
    remainder = dividend % divisor
    return quotient, remainder # return a tuple

q, r = divide(13, 3) # tuple unpacking
print('quotient = ', q, ', remainder = ', r, sep='')

quotient = 4, remainder = 1

```

5. list, tuple, str의 비교

- A. list: **mutable sequences**
- B. tuple: **immutable sequences**
- C. str: **immutable sequences of Unicode code points**

▶ immutable: 변경 불가능

▶ Unicode: 한중일 등 세계 각국 문자를 포함하는 문자표현 방식
(python 3부터 적용됨.)

- D. list와 str은 sequence로 유사한 연산들이 정의되어 있음.

6. 리스트의 원소로 리스트/튜플 사용

- A. 리스트는 원소로 python의 어떠한 데이터형도 가질 수 있다. 기본자료

형 뿐만 아니라 리스트, 튜플 등을 원소로 가질 수도 있는 것이다.

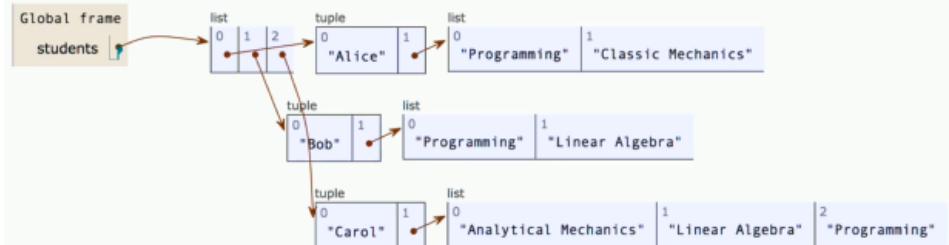
```
students = [  
    ('Alice', ['Programming', 'Classic Mechanics']),  
    ('Bob', ['Programming', 'Linear Algebra']),  
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming']))  
print(students[2])  
print(students[2][1])  
print(students[2][1][2])
```

```
('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])  
['Analytical Mechanics', 'Linear Algebra', 'Programming']  
Programming
```

▶ len(students) = 3인 것에 주의한다!!

B. 리스트의 원소로 리스트/튜플을 사용한 A의 예제의 경우 메모리 상태는 다음과 같다.

```
students = [  
    ('Alice', ['Programming', 'Classic Mechanics']),  
    ('Bob', ['Programming', 'Linear Algebra']),  
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming']))
```



7. 예제1: 리스트의 원소가 리스트/튜플일 때 반복문

```
students = [  
    ('Alice', ['Programming', 'Classic Mechanics']),  
    ('Bob', ['Programming', 'Linear Algebra']),  
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming']))  
for student in students:  
    print('Name :', student[0])  
    print('Courses :', student[1])
```

student = ('Alice', ['Programming', 'Classic Mechanics'])
tuple ('Alice', ['Programming', 'Classic Mechanics'])

```
Name : Alice  
Courses : ['Programming', 'Classic Mechanics']  
Name : Bob  
Courses : ['Programming', 'Linear Algebra']  
Name : Carol  
Courses : ['Analytical Mechanics', 'Linear Algebra', 'Programming']
```

8. 예제 2: 리스트의 원소가 리스트/튜플일 때 반복문

```
students = [
    ('Alice', ['Programming', 'Classic Mechanics']),
    ('Bob', ['Programming', 'Linear Algebra']),
    ('Carol', ['Analytical Mechanics', 'Linear Algebra', 'Programming'])
]

for student_name, courses in students:
    print('Name :', student_name)
    print('Courses :', courses)
```

Handwritten annotations:

- Above the first list item, 'Alice' is circled.
- Below the first list item, 'student_name' is underlined.
- Below the second list item, 'courses' is underlined.
- Below the third list item, 'student_name' is underlined.
- Below the third list item, 'courses' is underlined.
- Handwritten arrows point from 'student_name' and 'courses' to their respective underlined parts.
- Handwritten arrows point from 'student_name' and 'courses' to the corresponding variables in the loop body.
- Handwritten text 's_n, cou' is written near the variables.
- Handwritten text 'Alice, C' is written near the list items.

```
Name : Alice
Courses : ['Programming', 'Classic Mechanics']
Name : Bob
Courses : ['Programming', 'Linear Algebra']
Name : Carol
Courses : ['Analytical Mechanics', 'Linear Algebra', 'Programming']
```

9. 가변 언패킹(unpacking)

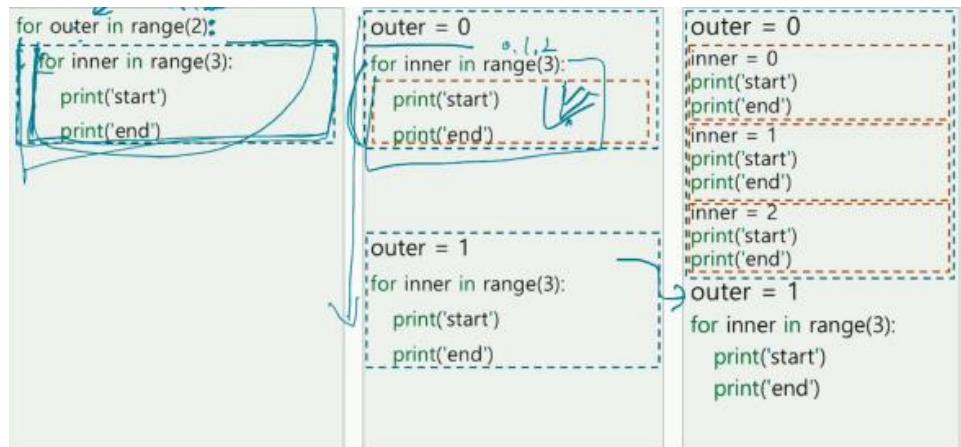
- ▶ 튜플/리스트에 있는 값을 각각 변수에 대입하는 것을 언패킹이라 한다.
- ▶ 앞에서도 말했듯, 언패킹 때는 반드시 원소의 개수와 변수의 개수가 일치하여야 한다.
 - ▷ 단, python3 부터는 *를 붙인 변수명으로 가변 길이 원소 언패킹이 가능해졌다.

```
my_list = [1, 2, 3, 4]
first, second, third, fourth = my_list # unpacking
first, rest = my_list
first, second, *rest = my_list
rest, last = my_list
first, *rest, last = my_list
```

- ▷ 다만, 가변 길이가 2개 이상이 되면, 값을 몇 개까지 변수에 저장해야 할지 모르기 때문에, 오류가 난다. 위의 예시에서 *first, rest, *last와 같은 언패킹은 하지 못한다는 것이다.

iii. 반복문 추가 설명

1. 반복문 중첩-nested for loops



2. else 절이 있는 for문

- ▶ for/while문의 else절은 for/while문의 반복이 끝난 뒤 실행된다.
 - ▷ for에 있는 else절의 경우: sequence의 모든 값을 loop variable에 대입한 후 실행된다.
 - ▷ while에 있는 else절의 경우: 조건이 False가 된 이후 실행된다.

```
t = [42, 1024, 23]
for v in t:
    print(v)
else:
    print('end of for loop')
```

42 1024 23	end of for loop
------------------	-----------------

- ▶ 그런데, else절에 있던지 없던지, while문, for문이 끝나면, 그 뒤의 코드는 실행되기에 else절을 쓸 필요가 있는가라는 의문이 있을 수 있다.

3. 제어 흐름: break, continue, pass(반복문의 else절의 필요성을 느끼게 해주는 제어 흐름 조절 예약어)

- A. **break:** 가장 안쪽의 for 혹은 while문의 실행을 끝내는 명령

- ▶ break문을 이용해 반복문의 실행을 끝내면 else절이 실행되지 않는다. (else절도 for 혹은 while문의 일부이기 때문)

- ▶ 예제

```
number = 7
for i in range(2, number):
    if number % i == 0:
        break
    else:
        print(number, 'is a prime number.')
```

7 is a prime number.

▷ 위 예제는 number를 받아 number를 2부터 그 number전까지의 수로 나누어 만약, 나눠서 나머지가 0이 되는 수가 있으면(소수가 아닌 수), 반복문을 break하도록 하고, 없으면(소수인 수), else문을 실행하도록 하는 알고리즘이다.

B. continue:

loop의 코드블럭을 돌다가 이 명령을 만나면, 남은 코드블럭을 실행하지 않고, 바로 다음 반복을 수행한다.

- ▶ while에서 continue를 만날 경우: continue 다음 코드블럭을 무시하고 조건을 검사하는 곳으로 가서 True이면 while문 내부의 코드블럭의 반복을 처음부터 재개한다.
- ▶ for에서 continue를 만날 경우: continue 다음의 코드블럭을 실행하지 않고, sequence의 다음 값을 loop variable에 대입하고 for문 내부의 코드블럭을 다시 실행한다.

- ▶ 예제

```
for num in range(2, 10):
    if num % 2 == 0:
        print('Found an even number', num)
        continue
    print('Found a number', num)
```

Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9

▷ 위 예제는 2와 같거나 크고 10보다 작은 num이라는 변수

를 2로 나눠서 2로 나눠지면 even number를 출력하고 다음 반복을 실행하고, 나눠지지 않으면, a number를 출력하고 다음 반복을 실행하는 알고리즘이다.

C. **pass**: 어떤 연산도 수행하지 않고, 주로 문법적으로 필요한 경우 사용되는 명령이다. (같은 기능을 하는 명령으로 placeholder가 있다.)

▶ 예제

```
# pass is typically used as a placeholder
def foo():
    pass

for i in range(10):
    pass

while False:
    pass
```

4. 반복문 사용의 주의점 – 무한반복

A. for 반복문에서 무한반복이 되는 예시

```
numbers = [1, 2, 3]
for number in numbers:
    print(number)
    if number % 2 == 0:
        numbers.append(number)

print(numbers)
```

```
1
2
3
2
2
```

▷ 위 예시는 for문 안에서 for문을 돌도록 하는 리스트에 계속 원소를 보충해줘서 무한 반복에 빠지게 되는 예제이다. 이렇게 for문에서도 무한 반복에 빠지게 될 수 있으므로, 반복문 사용 시에는 항상 무한 반복에 주의하도록 한다.

▷ 위 예시를 무한 반복에 빠지지 않도록 하여 동작하는 방법은 다음과 같다.

```
numbers = [1, 2, 3]
for number in numbers[:]:
    print(number)
    if number % 2 == 0:
        numbers.append(number)

print(numbers)
```

```
1
2
3
[1, 2, 3]
```

▷ 무한 반복에 빠지지 않도록 새롭게 처리한 것은 for문에 number를 그대로 넣어주지 않고, 초기 number를 복사한 새로운 리스트를 넣어주도록 한 것이다. 이렇게 하면, number의 원소는 늘어나도 for문을 돌아가게 하는 리스트의 원소는 늘어나지 않기 때문에 무한 반복에 빠지지 않는다.

▷ 애초에 가변해서 쓰는 값을 반복문을 돌기 위한 재료로 사용하는 것은 올바른 방법이 아니다.

VII. 문자열 함수, 문자열 처리, 파일입출력

i. 리스트 추가설명

1. List comprehension

▶ for문, if문을 내포하여 리스트 생성에 활용하는 것을 List comprehension이라고 한다.

▷ 예시: 0에서 9까지 숫자의 제곱이 저장된 리스트 생성

```
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
squares = []
for x in range(10):
    squares.append(x**2)
```

```
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
squares = [x**2 for x in range(10)]
```

좌측의 구문을 List comprehension을 사용하면, 오른쪽 구문이 된다.

▷ 예시: 0에서 9까지 숫자 중 짝수의 제곱이 저장된 리스트 생성

```
# [0, 4, 16, 36, 64]
squares = []
for x in range(10):
    if x % 2 == 0:
        squares.append(x**2)
```

```
squares = [x**2 for x in range(10) if x%2==0]
```

좌측의 구문을 List comprehension을 사용하면, 오른쪽 구문이 된다.

▷ 예시: 리스트 중첩 내포 (행렬의 transpose 구하기)

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]
tp = [[row[i] for row in matrix] for i in range(4)]
[1, 5, 9],
[2, 6, 10],
[3, 7, 11],
[4, 8, 12]
```

```
transposed = []
for i in range(4):
    transposed.append([row[i] for row in matrix])
for i in range(4):
    transposed_row = []
    for row in matrix:
        transposed_row.append(row[i])
    transposed.append(transposed_row)
```

우측의 구문을 List comprehension을 사용하면, 왼쪽 구문이 된다.

ii. 문자열 처리

1. 문자열 관련 함수

A. 문자열 함수(일부)

- ▶ 문자열 함수의 용법: str.function(arguments)와 같은 형식으로 쓴다.
 - ▷ 나누기/붙이기: split(), splitlines(), join()
 - ▷ 문자열 검색/치환: find(), rfind(), startswith(), endswith(), count(), replace(), index(), rindex()
 - ▷ 서식화: rjust(), ljust(), center(), format()
 - ▷ 대소문자: title(), capitalize(), lower(), upper()
 - ▷ 문자 확인: isalnum(), isdecimal(), isalpha()
 - ▷ 공백문자 제거: strip(), rstrip(), lstrip()
- ▶ str에 위치할 수 있는 것들
 - ▷ 문자열
 - ▷ 문자열을 가르키는 변수
 - ▷ 문자열을 반환하는 함수
- ▶ 반환값: 대다수의 함수들은 문자열을 반환
 - ▷ 예외: split(), splitlines()는 리스트를 반환한다.
- ▶ 추가적인 내용은
<https://docs.python.org/3.7/library/stdtypes.html#string-methods>

B. split(): 구분자를 기준으로 분리된 문자열의 리스트를 반환

- ▶ str.split(sep=None, maxsplit=-1)
 - ▷ 문자열이 들어왔을 때, 단어로 쪼개서 리스트로 반환하는 함수이다. 문자열을 쪼갤 때는 함수에서 sep 변수로 받은 값에 해당하는 문자를 기준으로 쪼개게 된다. 만약 sep이 주어지지 않으면, whitespace(문자열의 공백 및 , 를 포함한 파이썬에서 정해둔 다양한 문자열의 요소)를 기준으로 쪼개게 된다.
 - ▷ maxsplit이 주어지면, 쪼개는 횟수를 maxsplit의 값만큼 진행 한다. 예를 들어 maxsplit=1이면, 한 문장을 한번만 쪼갠다. 그

결과 두개의 문자열이 리턴된다. (즉, split으로 리턴되는 리스트의 원소의 개수는 maxsplit+1개이다.)

▷ maxsplit이 주어지지 않거나 -1이면, split의 횟수에 제한을 주지 않는 것이다. 그 결과 문자열을 처음부터 끝까지 다 돌면서 해당하는 기준문자에 대해 문자열을 쪼갠다.

▷ sep이 주어지면, 그 sep을 기준으로 문자열을 쪼갠다. 예를 들어 '1,,2'.split(',')이라 쓰면, ['1', '', '2']가 리턴된다.

▷ sep으로 다중 문자열을 주는 것도 가능하다. 예를 들어 '1<>2<>3'.split('<>')이라 쓰면, ['1', '2', '3']가 리턴된다.

▷ str이 빈 문자열 ""인 경우 split함수는 []을 리턴한다.

C. splitlines() : 개행문자로 분리된 문자열의 리스트를 반환

▶ str.splitlines([keepends])

▷ 이 함수는 line boundaries(\n: 줄 바꾸기)에서 문자열을 쪼개서 line들을 만들고, 이 line들을 리스트에 넣어 반환하는 함수이다.

▷ 함수의 parameter인 keepends는 Boolean값으로, True 또는 False를 입력하도록 한다. 그런데 이 parameter는 입력해도 되고, 입력하지 않아도 된다. 그 표시로 [keepends]라고 쓴 것이다. keepends를 입력하지 않거나 False를 입력하면, 리스트 안의 원소들(line들)은 \n을 기준으로 쪼개진게 들어가되 \n은 없어지고, keepends를 True로 입력하면, 리스트 안의 원소들 (line들)은 \n을 기준으로 쪼개진게 들어가되 각 lines의 끝에 \n이 온전히 보존된 채로 들어가게 된다.

▷ 참고: split('\'\n')과 유사하나, 마지막 빈 줄을 남기지 않는다는 점에서 차이가 있다.

D. 예시: split(), splitlines() 함수

```

quotès = ["First, solve the problem. Then, write the code. - John Johnson",
          "Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley",
          "Computers are good at following instructions, but not at reading your mind. - Donald Knuth",
          "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods"]

list_quote = quotes.splitlines()
print(list_quote)
for quote in list_quote:
    sentence, author = quote.split(' - ')
    print("'" + sentence + "' by '" + author)
    # or print(f"'{sentence}' by {author}'")

```

- ▶ 위의 예시는 quotes에 3중 따옴표를 사용한 문자열을 집어넣고 있다. 3중 따옴표를 사용하면, 여러줄에 걸친 문자열도 입력할 수 있다. (3중 따옴표가 주석의 의미로만 사용되는 것이 아니다.)
- ▶ 컴퓨터가 여러줄의 문자열을 읽을 때는 줄이 바뀔 때, \n이 있는 것으로 이해하고 읽는다.
- ▶ 이때, quotes.splitlines()를 사용하면, \n을 기준으로 문자열이 쪼개지고, list로 반환된다. 즉 여기에서는 함수의 결과로 ['First, solve the problem. Then, write the code. – John Johnson', 'Without requirements or design, programming is the art of adding bugs to an empty text file. – Louis Srygley', 'Computers ar good at following instructions, but not at reading your mind – Donald Knuth', 'Always code as if the guy who ends up maintaing your code will be a violent psychopath who knows where you live. – John Woods']가 반환된다.
- ▶ 그리고 for문을 보면 list_quote의 원소들을 각각 quote라 칭해서 quote.split()이 그 원소 하나 하나를 반칸-빈칸을 기준으로 쪼갠다. list_quote의 0번 원소를 기준으로 보자면, ['First, solve the problem. Then write the code', 'John Johnson']이 나오게 되는 것이다.
- ▶ 그 이후 쪼개서 나온 2개의 원소 중 앞의 것을 sentence에 넣고, 뒤의 것을 author에 넣는다.
- ▶ 그 이후 "'" + sentence + "' by' + author를 출력한다. 즉, "와 sentence와 " by와 author를 합쳐서 출력한다.
- ▶ 따라서 실행결과 다음과 같은 것을 볼 수 있다.

['First, solve the problem. Then, write the code. - John Johnson', 'Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley', 'Computers are good at following instructions, but not at reading your mind. - Donald Knuth', 'Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods']
 "First, solve the problem. Then, write the code." by John Johnson
 "Without requirements or design, programming is the art of adding bugs to an empty text file." by Louis Srygley
 "Computers are good at following instructions, but not at reading your mind." by Donald Knuth
 "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live." by John Woods

E. join(): 문자열들을 합친 문자열을 반환

▶ str.join(iterable)

- ▷ for문에서 원소를 순차적으로 한번씩 빼는 것이 가능한 자료구조(iterable 자료구조)를 input으로 받는 함수이다.
- ▷ iterable 자료구조의 원소들 사이에 str을 넣고 전체적으로 한 문자열로 만들어 리턴하는 함수이다.

▷ 예시: join()

<pre>t = ['apple', 'pear', 'banana'] s1 = '.'.join(t) s2 = ','.join(t) print(s1) print(s2)</pre>	apple.pear.banana apple, pear, banana
--	--

F. strip(): 앞뒤로 지정된 문자열을 제거

▶ str.strip([chars])

- ▷ input parameter인 [chars]는 생략가능하다.
- ▷ [chars]를 주면, str을 복사하고 문장 맨 앞에서부터 chars에 해당하지 않는 문자가 나올때까지 문자들을 제거하여 나간다. 또한, 맨 뒤에서부터 chars에 해당하지 않는 문자가 나올 때까지 chars를 제거하여 리턴한다. [chars]를 주지 않거나 None을 주면, str을 복사하고, str의 맨 앞에서부터 공백(빈칸, 띄어쓰기, tab, 줄바꿈)이 없을 때까지 공백을 제거하여나가고, 맨 뒤에서부터 공백이 없을 때까지 공백을 제거하여 리턴한다.
- ▷ 참고: 파일/네트워크에서 한 줄을 읽은 후, 공백을 제거하기

위해 주로 사용한다.

G. `rstrip()`, `lstrip()` : 앞 혹은 뒤의 지정된 문자열 제거

▶ `str.rstrip([chars])`

▷ [chars]를 주면, str을 복사하고 문장 맨 뒤에서부터 chars에 해당하지 않는 문자가 나올 때까지 chars를 제거하여 리턴한다. [chars]를 주지 않거나 None을 주면, str을 복사하고, str의 맨 뒤에서부터 공백(빈칸, 띄어쓰기, tab, 줄바꿈)이 없을 때까지 공백을 제거하여 리턴한다.

▶ `str.lstrip([chars])`

▷ [chars]를 주면, str을 복사하고 문장 맨 앞에서부터 chars에 해당하지 않는 문자가 나올 때까지 문자들을 제거하여 리턴한다. [chars]를 주지 않거나 None을 주면, str을 복사하고, str의 맨 앞에서부터 공백(빈칸, 띄어쓰기, tab, 줄바꿈)이 없을 때까지 공백을 제거하여 리턴한다.

H. 예제: `strip()`, `rstrip()`, `lstrip()`

```
print(' spacious '.strip())
print('www.example.com'.strip('cmowz.'))
comment_string = '#..... Section 3.2.1 Issue #32 .....'
print(comment_string.strip('.#! '))
print(' spacious '.rstrip())
print('mississippi'.rstrip('ipz'))
print(' spacious '.lstrip())
print('www.example.com'.lstrip('cmowz.'))
```

```
spacious
example
Section 3.2.1 Issue #32
    spacious
mississ
spacious
example.com
```

I. 문자열 검색/치환 함수: `str.find()`

▶ str을 앞에서부터 확인하여 ()안에 해당하는 문자가 쓰인 첫번째 인덱스를 리턴해준다. 만약, ()안에 해당하는 문자가 str에 없으면 -1을 리턴해준다.

"ab123ab".find("ab")	0
"ab123ab".find("ba")	-1
"ab123".find("b1")	1
"ab123ab".find("b")	1

J. 문자열 검색/치환 함수: str.rfind()

- ▶ str을 뒤에서부터 확인하여 ()안에 해당하는 문자가 쓰인 첫번째 인덱스를 리턴해준다. 만약 ()안에 해당하는 문자가 str에 없으면 -1을 리턴해준다.

```
"ab123ab".rfind("b") || 6
```

K. 문자열 검색/치환 함수: str.count()

- ▶ str을 확인하여 ()안에 해당하는 문자가 몇 번 쓰였는지 개수를 세서 리턴해준다.

```
"ab123ab".count("b") || 2
```

L. 문자열 검색/치환 함수: str.index()

- ▶ str을 앞에서부터 확인하여 ()안에 해당하는 문자가 쓰인 첫번째 인덱스를 리턴해준다. 만약 ()안의 문자가 str안에 없으면, error를 발생시킨다.

```
"ab123ab".index("b") || 1
```

M. 문자열 검색/치환 함수: str.rindex()

- ▶ str을 뒤에서부터 확인하여 ()안에 해당하는 문자가 쓰인 첫번째 인덱스를 리턴해준다. 만약 ()안의 문자가 str안에 없으면, error를 발생시킨다.

```
"ab123ab".rindex("b") || 6
```

N. 문자열 검색/치환 함수: str.startswith()

▶ str의 맨 앞을 확인해서 ()안의 문자로 시작하는지의 여부를 확인하는 함수이다. ()안의 문자로 시작하면 True를, ()안의 문자로 시작하지 않으면, False를 리턴한다.

"ab123ab".startswith("ba")	False
"ab123ab".startswith("ab")	True

O. 문자열 검색/치환 함수: str.endswith()

▶ str의 맨 뒤를 확인해서 ()안의 문자 순서로 끝나는지의 여부를 확인하는 함수이다. ()안의 문자 순서로 끝나면 True를, ()안의 문자 순서로 끝나지 않으면, False를 리턴한다.

"ab123ab".endswith("ab")	True
"ab123ab".endswith("ba")	False

P. 문자열 검색/치환 함수: str.replace()

▶ str을 확인해서 ()안에서 첫번째 자리에 해당하는 문자를 찾아 ()안에서 두번째 자리에 해당하는 문자로 교체하는 함수이다.

"ab123ab".replace("a", 'bb')	bbb123bbb
------------------------------	-----------

Q. 대소문자 변환: str.title()

▶ str에 입력된 문자열에서 문장이 시작할 때의 첫번째 문자를 대문자로 바꿔준다.

str = "abc Hello World!! ABC" str.title() str.capitalize() str.lower() str.upper()	Abc Hello World!! Abc Abc hello world!! abc abc hello world!! abc ABC HELLO WORLD!! ABC
--	--

R. 대소문자 변환: str.capitalize()

▶ str에 입력된 문자열 자체에 대해 문자열의 첫번째 문자를 대문자로 바꿔준다.

```
str = "abc Hello World!! ABC"  
str.title()  
str.capitalize()  
str.lower()  
str.upper()
```

```
Abc Hello World!! Abc  
Abc hello world!! abc  
abc hello world!! abc  
ABC HELLO WORLD!! ABC
```

S. 대소문자 변환: str.lower()

- ▶ 대상 문자열의 문자를 전부 다 소문자로 변경한다.

```
str = "abc Hello World!! ABC"  
str.title()  
str.capitalize()  
str.lower()  
str.upper()
```

```
Abc Hello World!! Abc  
Abc hello world!! abc  
abc hello world!! abc  
ABC HELLO WORLD!! ABC
```

T. 대소문자 변환: str.upper()

- ▶ 대상 문자열의 문자를 전부 다 대문자로 변경한다.

```
str = "abc Hello World!! ABC"  
str.title()  
str.capitalize()  
str.lower()  
str.upper()
```

```
Abc Hello World!! Abc  
Abc hello world!! abc  
abc hello world!! abc  
ABC HELLO WORLD!! ABC
```

U. 문자 확인: str.isalpha()

- ▶ str이 알파벳으로만 이루어져 있는지 검사하는 함수. 알파벳으로만 이루어져 있으면 True를, 숫자나 특수기호도 포함된 문자열이면 False를 리턴한다.

```
"ab123".isalpha()
```

```
False
```

V. 문자 확인: str.isalnum()

- ▶ str이 알파벳과 숫자로 이루어져 있는지 검사하는 함수. 알파벳과 숫자로만 이루어져 있으면, True를, 특수기호도 포함된 문자열이면 False를 리턴한다.

```
"ab123".isalnum()
```

True

W. 문자 확인: str.isnumeric()

- ▶ str이 숫자로만 이루어져 있는지 검사하는 함수. 숫자로만 이루어져 있으면 True를, 알파벳이나 특수기호도 포함된 문자열이면 False를 리턴한다.

```
"ab123".isnumeric()  
"123".isnumeric()
```

False
True

2. 문자열 서식화

A. python에서의 문자열 서식화

- ▶ 서식문자열에 따라, (변수에 저장된) 값의 자리수, 표현방식을 서식화하는 것을 말한다.
- ▶ 서식화된 '문자열'을 반환한다는 것은 변수에 대입하거나, printf() 함수를 이용하여 출력한다는 것을 의미한다.
 - ▷ 참고: C언어의 printf() 함수가 아닌, sprintf() 함수에 해당한다.

B. python에서 지원하는 문자열 서식화 방식

- ▶ Printf-style string formatting (all python versions)
 - ▷ C언어의 printf() 함수와 유사한 서식문자열을 지원한다.
- ▶ format() function with format string syntax (python3, python 2.6+)
 - ▷ 서식문자열의 표현이 더 풍부해졌다.
- ▶ Literal string interpolation (python 3.6+)
 - ▷ 서식문자열에 수식 혹은 변수의 이름을 적을 수 있는 기능을 추가하였다.

- ▶ Template string (python 2+): printf-style과 유사하다.

C. 문자열 서식화 방법 1: Printf-style string formatting (%operator) 소개

- ▶ 사용형태: format % values

▷ format: 서식 문자열로, 다음 형태의 변환명시자(conversion specifier)를 포함한다.

- %로 시작하고, 변환형을 나타내는 하나의 문자로 끝나는 변환명시자.

- mapping key, conversion flag, 최소 문자 폭, 정확도를 명시하는 변환명시자.

▷ values: 다음 중 하나의 형식을 사용한다.

- 한 개의 값(서식 문자열이 하나의 값을 요구할 때).
- 서식문자열이 요구하는 아이템 수와 같은 수의 값이 있는 튜플.
- 한 개의 사전(dict)과 같은 매핑 객체(mapping object)

- ▶ 기능: 서식문자열에 따라 values에 주어진 값들을 원하는 형식의 ‘문자열’로 변환한다.

- ▶ 단점: 튜플, 사전, 객체 등 복잡한 자료형의 값을 출력할 때 제한점이 많다.

- ▶ %operator는 string formatting operator 또는 interpolation operator라고도 불린다.

- ▶ 참고: C언어의 printf() 혹은 sprintf() 함수에서 서식문자열과 유사하다.

D. 문자열 서식화 방법 1: 변환 명시자(Conversion specifier) 형식

- ▶ %로 시작해서, 변환형을 나타내는 하나의 문자로 끝나야 한다. 즉, %(optional)(conversionoperator)의 형식으로 표현되어야 한다.

- ▶ (Optional)자리에는 mapping key를 쓸 수 있다.

- ()안에 문자들로 주어진다.

- 맵핑 객체 형태의 values에서 키가 ()안의 문자들인 값으로 대체된다.

▶ (Optional)자리에 conversion flag (일부)를 쓸 수도 있다.

- '0': 수의 경우, 남는 공간이 0으로 채워진다.
- '-': 변환된 문자열이 왼쪽에 정렬된다.
- conversion flag를 지정하지 않으면 문자열이 우측에 정렬된다.

▶ (Optional)자리에 최소 문자 폭: 출력되는 값의 최소 문자의 수를 지정하는 정수를 쓸 수도 있다.

▶ (Optional)자리에 정확도: '.'와 숫자로, 소수 부분의 자릿수를 명시할 수도 있다.

▶ (Optional)자리에 length modifier를 쓰는 언어도 있으나, python에서는 사용되지 않는다.

Conversion	Meaning
'd' or 'i'	value를 문자열 안에 정수로 변환해서 집어넣을 때 사용.
'o'	value를 문자열 안에 8진수로 변환해서 집어넣을 때 사용.
'u'	Obsolete type – it is identical to 'd'.
'x' or 'X'	value를 문자열 안에 16진수로 변환해서 집어넣을 때 사용한다. (lowercase('x') or uppercase('X'))
'e' or 'E'	value를 문자열 안에 실수로 변환해서 집어넣되, 매우 작거나 매우 커서 지수승으로 표기해야 할 때 사용. (lowercase('e') or uppercase('E').)
'f' or 'F'	value를 문자열 안에 실수로 변환해서 집어넣을 때 사용.
'g' or 'G'	value에 정수가 들어오면 정수로, 매우 커거나 작아서 지수승으로 표기해야 하면 지수승으로, 실수로 들어오면 실수로 변환해서 문자열 안에 집어넣어준다.
'c'	value를 한 개의 문자로 변환하여 문자열 안에 집어넣을 때 사용. (숫자 한 개나 문자 한 개에 대해 사용한다.)

'r', 's', 'a'	value를 문자열로 변환하여 문자열 안에 집어넣을 때 사용한다. 이렇게 쓰는 것은 각각 repr(), str(), ascii()와 같은 기능을 한다.
'%'	문자열 안에 %를 쓰고 싶으면 %%와 같이 사용하면 된다.

E. 문자열 서식화 방법 1 예시: printf-style string formatting

<pre>var = 42 string = 'Answer' pi = 3.141592653589793 s1 = 'The %s is %.d.' % (string, var) print(s1) print('0123456789' * 2) print('%10d' % var) print('%010d' % var) print('%-10d' % var) print('%.f' % pi) print('%.2f' % pi) print('.10f' % pi) print('%.10.2f' % pi) print('%g, %g, %g' % (var, pi, 10000000000)) print('{%(name)s: %(average)g}' % {'name': 'Alice', 'average': 99.9})</pre>	The Answer is 42. 01234567890123456789 _____42 0000000042 42_____ 3.141593 3.14 3.1415926536 _____3.14 42.3.14159.1e+10 Alice: 99.9
---	---

F. 문자열 서식화 방법 2: format() 함수를 이용한 문자열 서식화

▶ 사용형태: string.format(arguments)

- string: 서식 문자열
- arguments: 서식 문자열에 의해 변환될 인자들

▶ 서식 문자열에 '{fieldname:format_spec}'의 형태가 주어지면 인자의 값의 서식을 변경한다.

- fieldname: 어떤 인자의 값을 서식화할지를 지정
- format_spec: 어떠한 형태로 서식화할지를 지정

☞ format_spec의 형식:

[[fill][align][sign][0][width].[precision]][type]

▶ 참고: 서식화를 지정하는 서식 명시자(formatting specifier)는 printf 형태의 변환 명시자와 유사하지만, 그 기능이 확장되었고 사

용이 조금 더 편리하도록 변경되었다.

G. 문자열 서식화 방법 2 예시: `format()` 함수

```
d = 42
f = 3.14
s = 'apple'
print('{0} {1} {2}'.format(d, f, s))
print('{2} {0} {1}'.format(d, f, s))
print('{0} is {0}'.format(s))
print('{d} {f} {s}'.format(d=6, f=1.618,
s='pineapple'))
print('{0:d} {0:f} {0:g}'.format(42))
```

```
42 3.14 apple
apple 42 3.14
apple is apple
6 1.618 pineapple
42 42.000000 42
```

```
print('0123456789' * 2)
print('{:10d}'.format(42))
print('{:>10d}'.format(42))
print('{:<10d}'.format(42))
print('{:^10d}'.format(42))
print('{:10.2f}'.format(3.1415926535))
print('{:1d}'.format(42))
print('{:5.5f}'.format(3.1415926535))
```

```
01234567890123456789
        42
        42
42        42
        42
        3.14
42
3.14159
```

H. 문자열 서식화 방법 3: Formatted string literal을 이용한 문자열 서식화

▶ f-string (f"str"의 형식으로 사용한다.)

▷ 문자열 앞에 f를 이용하여 표시한다. (python에서 문자열 앞에 한 글자를 추가하여, 특수한 문자열을 나타내는 규칙을 사용하는 것이다.)

▷ 서식 표현은 `format()` 함수와 유사하다.

▷ '{ }' 안에 계산이 가능한 표현(expression)이 들어간다.

▷ f-string이 실행될 때, '{ }'안의 표현이 연산(evaluate)된다.

I. 문자열 서식화 방법 3 예시: Formatted string literal

pi = 3.14159265 width = 10 precision = 2 print('0123456789' * 3) print(f'{pi}') print(f'{pi:10.2f}') print(f'{pi:{width}.{precision}f}') print(f'{max(width, precision)}') t = [42, 1024, 23] for i in range(len(t)): print(f'{i}: {t[i]}') * 빈칸은 „로 표시함	012345678901234567890123456789 3.141592653.143.14 10 0..42 1..1024 2..23
---	---

26

iii. 파일 입출력 (실행결과를 파일로 만들어 저장하는 방법)

1. 파일의 종류

- ▶ 파일이란?
 - ▷ 문자, 숫자 등으로 이루어진 정보의 집합체이다.
 - ▷ HDD, SSD, USB drive, 클라우드 등의 저장장치에 저장된다.
 - ▷ 파일이 저장되어 있는 공간(파일 경로 혹은 path)과 이름(파일명)으로 구분된다.
- ▶ 텍스트 파일: 사람들이 인지할 수 있는 여러 줄의 문자들로 이루어진 파일
 - ▷ 문자인코딩(예: ASCII, UTF-8, CP-949)에 따라 표현할 수 있는 문자가 달라진다.
 - ▷ 참고: 파이썬 소스 파일(.py), html 파일 (.html) 등도 텍스트 파일의 일종이다.
- ▶ 바이너리 파일: 텍스트 파일이 아닌 파일
 - ▷ 아래와 한글 문서파일 (.hwp), 워드파일 (.doc), 이미지파일 (.jpg, .png) 등

2. 텍스트 파일 입출력

▶ 파일 입출력 전후로 파일을 열고, 닫는 단계가 필요하다.(컴퓨터로 문서를 편집하기 위해서 파일을 여는 것과 유사하다.)

▷ 파일 열기(open)

▷ 파일 읽기 혹은 쓰기 (read or write)

▷ 파일 닫기 (close): with를 사용한 경우 생략한다.

▶ python에서 제일 간단한 텍스트 파일 입출력 방법

▷ 파일입력

- 방법1: read() 함수를 이용하는 방법 ↪ 파일 전체를 하나의 문자열로 읽는다.

- 방법2: split(), splitlines() 함수를 이용하는 방법 ↪ 한 줄씩 처리한다.

▷ 파일 출력

- print()함수를 이용하는 방법 ↪ 화면에 인쇄하는 것처럼 파일을 출력하는 것이 가능하다.

3. 텍스트 파일 읽기

▶ 코드 설명

▷ 아래의 코드는 코드파일과 같은 주소에 있는 example.txt라는 이름의 파일의 내용을 변수 file_contents에 저장하는 알고리즘을 수행한다.

- 일반적인 방법

```
fileobj = open('example.txt', 'rt')
```

```
file_content = fileobj.read()
```

```
fileobj.close()
```

- with를 사용하는 방법 (close()를 호출할 필요가 없다.)

```
with open('example.txt', 'rt') as fileobj:
```

```
    file_content = fileobj.read( )
```

4. 텍스트 파일 쓰기

▶ 코드 설명

▷ 아래의 코드는 코드파일과 같은 주소에 있는 example.txt라는 이름의 파일에 문자열 something을 쓰는 알고리즘을 수행한다. 코드파일과 같은 주소에 example.txt가 없으면, example.txt 파일을 자동으로 만들어서 알고리즘을 수행한다.

• 일반적인 방법

```
fileobj = open('example.txt', 'wt')
```

```
print('something', file=fileobj)
```

```
fileobj.close( )
```

• with를 사용하는 방법 (close())를 호출할 필요가 없다.)

```
with open('example.txt', 'wt') as fileobj:
```

```
    print('something', file=fileobj)
```

5. 예시: 파일입출력 함수

```
quotes = "First, solve the problem. Then, write the code. - John Johnson  
Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley  
Computers are good at following instructions, but not at reading your mind. - Donald Knuth  
Always code as if the guy who ends up maintaining your code will be a violent psychopath who  
knows where you live. - John Woods  
with open('quotes.txt', 'wt') as f:  
    print(quotes, file=f)  
with open('quotes.txt', 'rt') as f:  
    file_contents = f.read()  
    print(file_contents)
```

예시: 파일입출력 함수 (실행 결과)

First, solve the problem. Then, write the code. - John Johnson
Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley
Computers are good at following instructions, but not at reading your mind. - Donald Knuth
Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods

6. 읽을 거리

- ▶ python 문자열 자료형과 함수: <https://wikidocs.net/13>
- ▶ python 문자열 서식화
 - ▷ PyFormat Using % and .format() for great good!: <https://pyformat.info/>
 - ▷ The 4 Major Ways to Do String Formatting in Python: <https://dbader.org/blog/python-string-formatting>
- ▶ References
 - ▷ printf-style string formatting:
<https://docs.python.org/3/library/stdtypes.html?highlight=printf#old-string-formatting>
 - ▷ Format string syntax:
<https://docs.python.org/3/library/string.html#formatstrings>
 - ▷ Literal string interpolation:
<https://www.python.org/dev/peps/pep-0498/>

VIII. 집합(set), 사전(dict), 바이너리 파일 입출력

i. 파일 입출력 (바이너리 파일)

1. 파일

▶ 파일

- ▷ 문자, 숫자 등으로 이루어진 정보의 집합체이다.
- ▷ HDD, SSD, USB drive, 클라우드 등의 저장장치에 저장된다.
- ▷ 파일이 저장되어 있는 공간(파일 경로 혹은 path)과 이름(파일명)으로 구분된다.
- ▶ 텍스트 파일: 사람들이 인지할 수 있는 여러 줄의 문자들로 이루어진 파일이다.
 - ▷ 문자 인코딩(ex. ASCII, UTF-8, CP-949)에 따라 표현할 수 있는 문자가 달라진다.
 - ▷ 참고: 파이썬 소스 파일(.py), html 파일 (.html) 등도 텍스트 파일의 일종이다.
- ▶ 바이너리 파일: 텍스트 파일이 아닌 파일을 말한다. (1byte를 기본 단위로 사용한다.)
 - ▷ ex. 아래아 한글 문서파일 (.hwp), 워드파일 (.doc), 이미지파일 (.jpg, .png) 등이 있다.

2. 바이너리 파일 입출력

- ▶ 파일 입출력 전후로 파일을 열고, 닫는 단계가 필요하다. (컴퓨터로 문서를 편집하기 위해서 파일을 여는 것과 유사하다.)
 - ▷ 파일 열기 (open)
 - ▷ 파일 읽기 혹은 쓰기 (read or write)
 - ▷ 파일 닫기 (close): with을 사용한 경우 생략한다.
- ▶ python에서 제일 간단한 바이너리 파일 입출력 방법 (입출력 모드 "b") : 텍스트 파일은 그냥 print 같은 함수를 썼지만, 바이너리 파일은

byte 단위로 관리를 해주어야 한다.

▷ 파일 입력

- `read()` 함수 이용

▷ 파일 출력

- `write()` 함수 이용(일반적인 `print` 함수보다 byte 단위의 관리에 적합한 함수이다.)

- 인자로 `b "a b c"`을 쓰면, a, b, c 각각에 1byte를 할당하여 저장한다. 이것을 byte string이라 한다.

▶ 파일 입출력 모드: `r(read mode)` / `w(write mode)`: 파일을 처음 만들고 나서 입력을 위해 사용하는 입출력 모드) / `a append mode`: 이미 값이 들어있는 파일을 열어서 추가로 데이터를 입력하기 위해 사용하는 입출력 모드) / `r+(read mode로 파일을 열어서 read와 write를 둘 모두 사용하고 싶을 때, r+t의 형태로 모드를 입력해준다./ 마찬가지로 a+t의 형태는 append 모드로 파일을 열어서 read와 write를 모두 사용하는 모드를 말한다. 이때, t는 text mode를 말한다.)`

3. 예제: 바이너리 파일 읽기/쓰기

```
f = open('workfile', 'rb+')
print(f.write(b'0123456789abcdef') )
f.seek(5) # Go to the 6th byte in the file
print(f.read(1) )
f.seek(-3, 2) # Go to the 3rd byte before the end
print(f.read(1))
print(f.read())
f.close()
```

```
16
b'5 '
b'd'
b'ef'
```

▶ `f = open('workfile', 'rb+')`은 workfile이라는 이름의 파일을 binary(byte) 형식 read mode를 기본으로 여는데, read mode와 write mode를 모두 사용할 수 있게 여는 작업을 f라는 변수로 저장한다는 것이다.

▶ workfile을 byte 파일로 인식하고, 열었기 때문에, 파일에 무언가를 입력하고 싶다면, `f.write(b'string')`을 `print()` 안에 인자로 넣어서 주어야

한다. 이렇게 입력하면, string의 문자 1개 당 1byte가 할당되어 저장된다.

▶ byte 파일의 특징인데, 이렇게 byte 파일을 열면(read mode든, write mode든) 전에 입력이 끝난 지점에서 다시 동작을 시작한다. 다시말해, `print(f.write(b'0123456789abcdef'))`라 입력하면, byte file에 `0123456789abcdef`까지 입력하고, f 바로 뒤에 커서(파일 포인터)가 가있는 상태가 된다. 그러면 여기서 `read()`를 실행하려고 하면, 뒤에 아무것도 없어서 아무것도 출력하지 않고 종료된다. 그래서 출력하고 싶은게 있으면, 그 자리 바로 전으로 커서(파일 포인터)를 옮겨서 `read()`를 실행하여야 한다.

▶ 그 커서(파일 포인터)를 옮기는 방법은 `seek()` 함수를 사용하는 방법이다. 위 예제처럼 `f.seek(5)`를 실행하면, f에 따라 바이너리(byte) 파일을 열고, f 뒤에 있던 커서를 파일에서 5번째 byte 뒤(즉, 6번째 byte 자리)로 커서가 옮겨진다. 따라서 이 직후 `read(1)`를 실행하면, 6번째 byte부터 `read`로 받은 인자의 숫자만큼 읽어온다. 여기서는 1을 `read`에 주었으므로, 6번째 byte 1개만 읽어온다.

▶ 이 `seek`는 인자를 주는 또 다른 방식이 있다. 일반적으로 `seek(i)`라 쓰면, 파일 시작하자 마자 있는 첫번째 byte를 1이라 할 때, i번째 byte 바로 뒤로 커서가 이동한다. 그런데, `seek(i, 1)`이라 쓰면, 현재 위치를 1이라 할 때, i번째 byte 바로 뒤로 커서가 이동한다. 또, `seek(-i, 2)`이라 쓰면, 파일 제일 마지막에 있는 byte를 1이라 할 때, 뒤에서부터 앞으로 오름차순으로 세어볼 때, i번째 byte로 커서가 이동한다. i를 음수로 주면 현재 커서의 위치를 1이라 볼 때 파일 앞 방향으로 i번째 데이터의 위치로, i를 양수로 주면 현재 커서의 위치를 1이라 볼 때 파일 뒤 방향으로 i+1번째 데이터 위치로 이동하는 것이다.

▶ `read()`도 인자를 주는 방식이 있다. 일반적으로 `read()`라고 써서 실행하면, 파일에서 커서가 위치한 자리의 byte부터 파일이 끝날 때까지 모두 읽어온다. 그런데 `read(i)`라 쓰면, 파일에서 커서가 위치한 자리의 byte를 포함해서 i개(i byte)만큼의 데이터를 읽어온다. 이때, 불러온 데 이터가 문자이면, python에서 'alphabet'의 형태가 아니라 binary 문자라는 의미에서 b'phabet'의 형태로 저장 및 표현하게 된다.

▶ 이렇게 `open()` 함수를 써서 파일에 데이터를 쓰거나 읽었으면, 마지막에선 꼭 `close()` 함수를 써준다. 위에서처럼 f라는 객체로 여는 작업을 지정했으면, `f.close()`라고 써준다.

ii. 자료구조

1. 집합(set)

A. Python에서의 집합

- ▶ 순서가 없고 중복이 없는 원소들을 가지고 있는 자료형 (수학에서의 집합과 유사하다.)

▶ 불변하는(immutable) 자료형만 원소로 가질 수 있다. ex. 부울, 정수, 실수, 튜플, 문자열 등이 여기에 해당한다.

- ▶ 일반적으로 문자열 혹은 정수가 원소로 많이 사용된다.

B. 정의하는 방법

- ▶ {}를 이용하고, 원소들을 ,로 구분하여 정의한다.
- ▶ 원소들을 포함하는 리스트, 튜플 등을 set() 함수의 인자로 넣으면, 자료형이 set으로 형변환 된다.
- ▶ 단, 빈 집합(empty set)을 정의하고 싶을 때는 원소 없이 set() 함수만 사용하도록 한다. {}는 빈 집합이 아니라 빈 사전으로 인식된다.

C. 출력하는 방법

- ▶ {}를 사용해서 출력한다.

D. 예제 1: 집합의 생성

```

s1 = {42, 1024, 23}
print(s1)
s2 = set() # empty set
print(s2)
basket = ('apple', 'orange', 'apple', 'pear', 'orange',
'banana')
print(basket)
str_set = set('abc')
print(str_set)
t1 = (42, 1024, 23)
tuple_set = set(t1)
print(tuple_set)
l1 = [42, 1024, 23, 1024, 23, 1]
list_set = set(l1)
print(list_set)
print(list(list_set), sorted(list(list_set)))

```

```

{1024, 42, 23}
set()
{'pear', 'banana', 'apple', 'orange'}
{'c', 'a', 'b'}
{1024, 42, 23}
{1024, 1, 42, 23}
[1024, 1, 42, 23] [1, 23, 42,
1024]

```

9

- ▶ set() 함수 안에 문자열을 인자로 넣으면, 문자열이 문자로 분리되어서 각각의 문자가 집합의 원소가 된다.

E. 예제 2: 집합의 추가/삭제

```

s1 = set() # empty set
print(s1)
s1.add(6)
print(s1)
s1.add(23)
print(s1)
s1.add(6)
print(s1)
s1.update([6, 1024, 4096])
print(s1)
s1.remove(6) # KeyError if 6 not in s1
print(s1)
print(42 in s1)
print(len(s1)) # number of elements in s1
s1.clear()
print(s1)

```

```

set()
{6}
{6, 23}
{6, 23}
{1024, 4096, 6, 23}
{1024, 4096, 23}
False
3
set()

```

10

- ▶ set.add(value)는 set에 value를 추가하는 함수이다.
- ▶ set.update(sequence)는 set에 sequence 자료형 안에 있는 원소를 모두 한꺼번에 집어넣는 함수이다.
- ▶ set.remove(value)는 set에 있는 value를 제거하는 함수이다. 다만, 지목한 value가 없으면, KeyError를 발생시킨다.
- ▶ value in set은 value가 set에 있는지 불연산 하는 예약어이다. 있으면 True, 없으면 False를 출력한다.
- ▶ len(set)은 set의 원소의 개수를 반환하는 함수이다.
- ▶ set.clear()는 set의 원소를 모두 제거하는 함수이다.

F. 예제 3: 집합의 연산자 1

```
s1 = {6, 23}
s2 = {42, 1024, 23}
print(s1 | s2) # union
print(s1.union(s2))
print(s1 - s2) # difference
print(s1.difference(s2))
print(s2 - s1) # difference
print(s2.difference(s1))
print(s1 & s2) # intersection
print(s1.intersection(s2))
print(s1 ^ s2) # symmetric difference
print(s1.symmetric_difference(s2))
```

{1024, 6, 23, 42}
{1024, 6, 23, 42}
{6}
{6}
{1024, 42}
{1024, 42}
{23}
{23}
{1024, 42, 6}
{1024, 42, 6}

- ▶ `s1 | s2`는 `s1`과 `s2`를 합치는 합집합 연산이다.
- ▶ `s1.union(s2)`는 `s1`과 `s2`를 합치는 함수이다.
- ▶ `s1 - s2`는 `s1`에서 `s2`를 빼는 차집합 연산이다.
- ▶ `s1.difference(s2)`는 `s1`에서 `s2`를 빼는 함수이다.
- ▶ `s2 - s1`는 `s2`에서 `s1`을 빼는 차집합 연산이다.
- ▶ `s2.difference(s1)`는 `s2`에서 `s1`을 빼는 함수이다.
- ▶ `s1 & s2`는 `s1`과 `s2`의 교집합을 구하는 연산이다.
- ▶ `s1.intersection(s2)`는 `s1`과 `s2`의 교집합을 구하는 함수이다.
- ▶ `s1 ^ s2`는 `s1`과 `s2`의 대칭 차집합(합집합-교집합)을 구하는 연산이다.
- ▶ `s1.symmetric_difference(s2)`는 `s1`과 `s2`의 대칭 차집합(합집합-교집합)을 구하는 함수이다.

G. 예제 4: 집합의 연산자 2

s1 = {6, 23}	
s2 = {42, 1024, 23}	
print(s1.isdisjoint(s2)) # disjoint	False
print(s1 == s2)	False
print(s1 <= s2) # subset	False
print(s1.issubset(s2))	False
s1.remove(6)	True
print(s1 <= s2)	True
print(s1 < s2)	False
print(s1 >= s2) # superset	False
print(s1.issuperset(s2))	False
print(s1 > s2)	False
s3 = (s1 s2).copy()	False
print(s3)	{1024, 42, 23}

- ▶ set1.isdisjoint(set2)는 set1과 set2에 교집합이 없는지 불 연산하는 함수이다. 없으면 True, 있으면 False를 출력한다.
- ▶ set1 == set2는 set1과 set2가 같은지 불 연산하는 것이다. 같으면 True, 다르면 False를 출력한다.
- ▶ set1 <= set2는 set1이 set2과 같거나 set2의 부분집합인지 불 연산하는 것이다. 부분집합이면 True, 아니면 False를 출력한다.
- ▶ set1.issubset(set2)는 set1이 set2의 부분집합인지 불 연산하는 함수이다. 부분집합이면 True, 아니면 False를 출력한다.
- ▶ set1.remove(value)는 set1에서 value라는 원소를 제거하는 함수이다.
- ▶ set1 < set2는 set1이 set2의 부분집합인지 불 연산하는 것이다. 부분집합이면 True, 아니면 False를 출력한다.
- ▶ set1 >= set2는 set1이 set2과 같거나 set2를 부분집합으로 가지는 지 불 연산하는 것이다. 부분집합으로 가지면 True, 아니면 False를 출력한다.
- ▶ set1.issuperset(set2)는 set1이 set2를 부분집합으로 가지는 지 불 연산하는 함수이다. 부분집합으로 가지면 True, 아니면 False를 출력한다.
- ▶ set1 > set2는 set1이 set2를 부분집합으로 가지는 지 불 연산하는 것이다. 부분집합으로 가지면 True, 아니면 False를 출력한다.
- ▶ set1.copy()는 set1을 복사한 새 집합을 만드는 함수이다.

H. 예제 5: Set comprehension (집합 내포)

```
s = {x for x in 'abracadabra' if x not in 'abc'}  
print(s)
```

```
{'d', 'r'}
```

2. 사전(dict)

A. 리스트, 튜플의 특징

- ▶ 여러 아이템을 하나의 변수 이름 아래 저장하는 자료형이다.
- ▶ 숫자를 이용한 인덱스로 아이템에 접근이 가능하다.

B. 사전(dict)이란?

- ▶ '키(key)'와 '값(value)'의 쌍들로 구성된 mutable mapping(상호 연결) 구조 자료형이다.
- ▶ 키(key): 서로 다른 value에 같은 key값을 할당하는 것은 불가능하다. 또한 이 때문에, key는 불변하는 타입(immutable type, 부울, 정수, 실수, 튜플, 문자열)을 사용하여 정의해주어야 한다. 일반적으로는 문자열 혹은 정수가 많이 사용된다.
- ▶ 값(value): Python의 어떠한 데이터 형태(리스트, 튜플, 사전 등도 포함)이든 값으로 사용할 수 있다. 서로 다른 key에 같은 value를 할당하는 것은 가능하다.

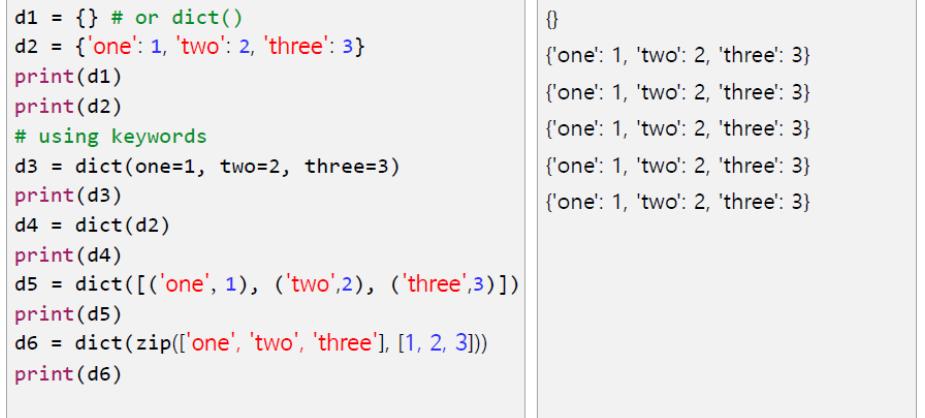
C. 참고: 다른 언어에서는 associative memory, associative array, hash, hashmap 등으로 불린다.

D. 예제 1: 사전의 정의와 사용



▶ 딕셔너리(사전) 자료형에서 원하는 value를 가져오고 싶으면, key 값을 이용하여 가져온다. 이때, 사용법은 list[index] 처럼 dictionary[key]의 형태로 쓰는 것이다.

E. 예제 2: 사전 생성하기



▶ zip(list1, list2)는 list1과 list2에서 같은 인덱스를 가진 value끼리 묶어 튜플을 만들어서 리턴하는 함수이다.

F. 예제 3: 사전으로 형변환하는 방법: dict() 함수를 사용.

▶ 사전을 만들 때, 두 개의 원소를 가진 시퀀스를 사전으로 형변환하는 방법이 가장 많이 사용된다. 각 시퀀스의 첫 번째 항목은 키(key), 두 번째 항목을 값(value)으로 하여 사전으로 변환된다.

▶ key1:value1일 때, dictionary[key1] = value2 를 입력하여 해당하는 key의 value를 바꾸는 것도 가능하다. 다만 key를 바꾸는 것은 불가능하다.

```

t1 = [['answer', 42], ['pi', 3.14], ['e', 2.718]]
d1 = dict(t1)
print(d1)
t2 = ('answer', 42), ('pi', 3.14), ('e', 2.718))
d2 = dict(t2)
print(d2)

```

```

{'answer': 42, 'pi': 3.14, 'e': 2.718}
{'answer': 42, 'pi': 3.14, 'e': 2.718}

```

G. 예제 4: 항목을 추가하는 방법

▶ 새로운 키/값을 이용하여 원소를 추가할 수 있다.

▷ 리스트와 다르게 초기화되지 않은(미리 정의되지 않은) 키(리스트의 인덱스에 해당)을 사용할 수 있다.

▶ dictionary[key1] = value2를 입력했는데, 해당 dictionary에 key1이 없으면, 그 dictionary에 key1을 만들고 value2를 넣어서 새 원소를 만들어준다.

▶ dictionary.update(key = value)를 통한 값 변경 및 항목 추가도 가능하다. dictionary에 해당 key가 있으면 value를 입력한 value로 바꿔주고, 해당 key가 없으면, 해당 key를 만들고 해당 value를 할당해준다.

▶ 참고: 값을 읽을 때(print 함수 사용과 같은 경우)는 미리 정의된 키만 사용할 수 있다. 정의되지 않은 키를 사용하면 KeyError가 발생한다.

```

numbers = {'pi': 3.14, 'e': 2.718}
numbers['golden_ratio'] = 1.618 # add a new item
numbers.update(pi = 3.1416) # {'pi':3.1416}
print(numbers)
print(numbers['answer']) # KeyError: 'answer'

```

```

{'pi': 3.1416, 'e': 2.718, 'golden_ratio': 1.618}

```

H. 예제 5: 항목을 삭제하는 방법: del, clear()

▶ del dictionary[key] 함수를 사용하는 방법: dictionary의 원소 중 함수에 입력된 key에 해당하는 원소와 키를 한꺼번에 지운다. 그런데, dictionary에 함수에 입력된 key가 없으면 KeyError를 일으킨다.

▶ dictionary.clear() 함수를 사용하는 방법: 사전에 있는 모든 원소

를 삭제한다.

```
numbers = {'answer': 42, 'kilo': 1024, 'birthday': 23}
print(numbers)
del numbers['kilo']
print(numbers)
numbers.clear()
print(numbers)

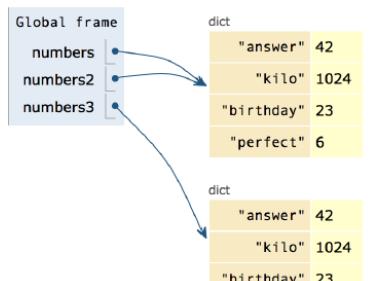
{'answer': 42, 'kilo': 1024, 'birthday': 23}
{'answer': 42, 'birthday': 23}
{}
```

I. 예제 6: 사전을 복사하는 방법: copy() 함수

▶ 주의할 점!: sequence 자료형을 복사하고 싶을 때, sequence1 = sequence2의 형태로 쓰면 안된다. 이건 sequence2의 주소를 sequence1에 복사하는 것이기 때문에, 실질적으로는 sequence1이나 sequence2나 같은 데이터를 지목한다. 즉, 데이터가 복사된 것은 아니다. 따라서 데이터까지 모두 복사해서 새로운 sequence1을 만들고 싶다면, sequence1 = sequence2.copy()라고 쓰는 것이 맞다.

```
numbers = {'answer': 42, 'kilo': 1024, 'birthday': 23}
numbers2 = numbers
numbers3 = numbers.copy()
numbers2['perfect'] = 6
print(numbers)
print(numbers2)
print(numbers3)

{'answer': 42, 'kilo': 1024, 'birthday': 23, 'perfect': 6}
{'answer': 42, 'kilo': 1024, 'birthday': 23, 'perfect': 6}
{'answer': 42, 'kilo': 1024, 'birthday': 23}
```



* 오른쪽은 4행까지 실행한 후의 상태

20

J. 예제 7: 사전 안의 항목을 얻는 방법: get() 함수

▶ dictionary.get(key[, default])라는 함수를 써서 dictionary 안의 key에 해당하는 value를 리턴받을 수 있다. 해당하는 key값이 dictionary안에 없으면, default를 리턴한다. 그런데 함수에서 [, default]로 쓰인 것처럼 default를 넣는 것은 선택사항인데, dictionary에 해당하는 key가 없고 default가 주어지지 않았다면,

None을 리턴한다. 따라서 이 함수는 KeyError를 발생시키지 않는다. 근데, KeyError를 발생시키지 않아서 프로그램이 멈출일은 없지만, 결과물만 보고서는 해당 key가 default인지 진짜 value인지 헷갈리는 경우도 있기 때문에 주의해서 사용해야 한다.

```
numbers = {'answer': 42, 'kilo': 1024, 'birthday': 23}
numbers['perfect'] = 6 # new item is added
#print(numbers['prime']) # KeyError: 'perfect'
print(numbers.get('answer', 0))
print(numbers.get('prime', 0))
```

42
0

K. 예제 8: 키 멤버십 테스트 작업: in, not in 예약어

▶ 특정한 키가 사전에 속해 있는지 확인하는 작업을 키 멤버십 테스트 작업이라 한다. 이 작업은 key in dictionary와 같이 쓰이면, 키가 있는지 없는지 확인해서 있으면 True를, 없으면 False를 리턴한다. 반대로 key not in dictionary와 같이 쓰이면, 키가 있는지 없는지 확인해서 있으면 False를, 없으면 True를 리턴한다.

```
bts_position = {
    '랩몬스터': '리더, 메인래퍼',
    '진': '서브보컬',
    '슈가': '리드래퍼',
    '제이홉': '서브래퍼, 메인댄서',
    '지민': '리드보컬, 리드댄서',
    '뷔': '서브보컬',
    '정국': '메인보컬, 서브래퍼, 리드댄서'}
print('진' in bts_position)
print('슈가' not in bts_position)
print('서브보컬' in bts_position)
```

True
False
False

L. 예제 9: 사전(dict)에 사용할 수 있는 함수들: keys(), values(), items()

- ▶ dictionary.keys() : dictionary를 구성하는 key들을 반환하는 함수이다.
- ▶ dictionary.values() : dictionary를 구성하는 value들을 반환하는 함수이다.
- ▶ dictionary.items() : dictionary를 구성하는 key와 value짝을 튜플

로 감싸서 원소 짹 튜플들을 반환하는 함수이다.

▶ 이 함수들로 리턴받은 값은 list도 dict도 tuple도 아니기에 필요 한 경우, list() 함수를 이용해서 list로 형변환해서 사용해야 한다.

```
numbers = {'pi': 3.14, 'e': 2.718, 'gr': 1.618}
print(numbers.keys())
print(numbers.values())
print(numbers.items())
print(list(numbers.items())[2])
```

```
dict_keys(['pi', 'e', 'gr'])
dict_values([3.14, 2.718, 1.618])
dict_items([('pi', 3.14), ('e', 2.718), ('gr', 1.618)])
('gr', 1.618)
```

M. 예제 10: keys(), values(), items()를 for문과 같이 사용하는 방법

```
bts_position = {
    '랩몬스터': '리더, 메인래퍼',
    '진': '서브보컬',
    '슈가': '리드래퍼',
    '제이홉': '서브래퍼, 메인댄서',
    '지민': '리드보컬, 리드댄서',
    '뷔': '서브보컬',
    '정국': '메인보컬, 서브래퍼, 리드댄서'}
for nickname in bts_position.keys():
    print(nickname)
    for nickname, position in
        bts_position.items():
            print(nickname, ':', position)
```

랩몬스터	
진	
슈가	
제이홉	
지민	
뷔	
정국	
랩몬스터 : 리더, 메인래퍼	
진 : 서브보컬	
슈가 : 리드래퍼	
제이홉 : 서브래퍼, 메인댄서	
지민 : 리드보컬, 리드댄서	
뷔 : 서브보컬	
정국 : 메인보컬, 서브래퍼, 리드댄서	

N. 예제 11: Dict comprehension (사전 내포)

```
d = {x: x**2 for x in (2, 4, 6)}
print(d)
```

```
{2: 4, 4: 16, 6: 36}
```

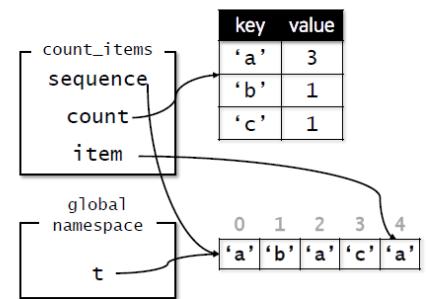
3. Sequence 자료형

A. **Sequence 자료형**: list, tuple, string, dictionary와 같이 Element들에 인덱스를 부여하여 줄줄이 나열하여 저장하는 구조의 자료형을 Sequence 자료형이라고 한다.

B. Sequence의 아이템 수를 세서 Dict로 리턴하는 방법 (직접 세는 방법)

```
01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         if item in count:
05:             count[item] += 1
06:         else:
07:             count[item] = 1
08:     return count
09:
10: t = ['a', 'b', 'a', 'c', 'a']
11: print(count_items(t))
```

{'a': 3, 'b': 1, 'c': 1}



C. Sequence의 아이템 수를 세서 Dict로 리턴하는 방법 (get() 함수를 이용하는 방법)

```
01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         count[item] = count.get(item, 0) + 1
05:     return count
06:
07: t = ['a', 'b', 'a', 'c', 'a']
08: print(count_items(t))
```

{'a': 3, 'b': 1, 'c': 1}

D. Sequence 자료형에 속하는 자료형들과 집합(집합은 Sequence 자료형이 아니다.)의 특징 비교.

리스트(list)	튜플(tuple)	사전(dict)	집합(set)
정의 t = [1, 2, 3]	t = (1, 2, 3) t = 1, 2, 3	t = {'1': 3, '2': 4, '3': -1}	t = {1, 2, 3}
인덱스 방식	정수 인덱스	정수 인덱스	(immutable인) 키 예: 정수, 문자열, 튜플
인덱스 예시	t[2]	t[2]	t['2']
변경 가능	가능	불가능	가능

IX. 객체지향 프로그래밍 소개

i. 객체지향 프로그래밍 (Object-Oriented Programming)

1. 객체지향 프로그래밍 소개

A. 객체지향 프로그래밍이란?

▶ 절차지향 프로그래밍 (Procedural Programming)

- ▷ 프로시저 (Procedure: 프로그램 언어에 따라 서브루틴, 함수, 메소드 등으로도 불림) 호출을 사용하는 프로그래밍 패러다임을 절차지향 프로그래밍이라고 한다.
- ▷ 모듈화를 통해 작업하고자 하는 독립적인 범위(scope)를 만들고, 인자와 반환값을 통해 프로시저와 데이터를 교환한다.
- ▷ 어떠한 '작업'을 수행하는 것에 초점을 맞추어 프로그램을 개발하는 것이다.

▶ 객체지향 프로그래밍 (Objected-Oriented Programming, OOP)

- ▷ 절차지향 프로그래밍이 프로시저만 분리하여 프로그래밍을 하는 것이었기에 자료와는 분리되어 있다. 따라서 한가지 예로 add라는 함수가 있다면 이게 문자열을 연산하는 건지, 숫자를 연산하는건지, 또 다른 어떤 자료형을 연산하는 건지 알 수가 없다. 이 한계점 때문에 자료형과 그 자료형을 연산하는 연산(함수)을 함께(객체로) 정리하고자 하는 패러다임이 나왔다.
 - ▷ 데이터와 프로시저를 결합한 '객체'라는 개념에 기반한 프로그래밍 패러다임을 객체지향 프로그래밍이라고 한다.
 - ▷ 작업 대상이 되는 '자료'(Data)와 '처리방법'을 동시에 설계
- ▶ 참고: 객체지향과 다른 '함수형 프로그래밍'도 있다.

B. 왜 객체지향 프로그래밍을 사용하는가?

- ▶ 더 높은 수준의 추상화(Abstract)를 제공하기 위한 방향으로 프로그래밍 언어가 진화하였다. (Machine language -> Assembly language -> Procedural languages C or Fortran -> Object-oriented languages C++ or Java, python -> Declarative or functional(함수 중심적 프로그래밍-수학적으로 증명하기에 용이할 것이다.) languages

Lisp, Haskell, ML, prolog, Scala)

- ▶ 즉, 어떤 실체가 있거나 없는 개념을 무엇이든지 구현할 수 있도록 하는 방향으로 프로그래밍 언어가 진화하였다.

- ▶ 목적

- ▷ 유지보수의 용이성: 데이터와 그 데이터에 대한 독립적인 함수가 같이 정리되어 있어서 구성이 쉽다. 따라서 유지보수가 용이하다.
- ▷ 코드의 재사용성이 향상된다.: 함수 뿐만이 아니라 자료구조도 함께 모듈화했으므로 코드, 디자인 방법 등을 재사용할 수 있다.
- ▷ 확장성 제공: 코드에 대한 요구사항이 변경되었을 때, 기능의 추가/변경이 용이하다.

C. 객체지향 프로그래밍을 지원하는 언어

- ▶ 객체 지향을 지원하는 언어 (지원 정도에는 차이가 있다.)

- ▷ C++: C언어에 객체지향 개념을 지원한다.

- ▷ Java

- ▷ python

- ▷ scala (다중 패러다임 언어: 함수형 & 객체지향 언어)

- ▷ ruby

- ▷ C#

- ▷ Objective-C

- ▶ 현재 널리 사용되는 대부분의 언어에서 객체지향 프로그래밍의 개념을 지원하고 있다.

2. 파이썬과 객체지향 프로그래밍

A. Python의 객체지향 프로그래밍 지원

- ▶ python은 객체지향을 염두에 두고 설계된 언어이다.
 - ▷ 간단한 문법으로 대부분의 객체지향 개념을 지원한다.
 - ▷ 엄격한 문법으로 객체지향을 지원하는 C++, Java와는 차이가 있다.
- ▶ python에서는 모든 것이 객체로 구현되어 있다. 따라서 기본자료형, 자료구조, 함수가 모두 객체지향적으로 구현되어 있어서 한 자료에 대해 그 자료만을 위한 연산이 짹으로 존재한다.
- ▶ 클래스(class) vs 인스턴스(instance) 혹은 객체(object)
 - ▷ 클래스: 어떤 객체를 집어넣을 때, 컴퓨터가 그것을 인식할 수 있도록 정의한 것으로 자료형이라 부르기도 한다. 쉽게 말해, 다양한 책상다리의 구조를 컴퓨터가 바로 인식하는 것은 불가능하므로, 다리와 책상 그리고 재질 등을 컴퓨터에 미리 알려주고 어떤 책상다리의 구조가 입력되었을 때, 이를 인식하도록 만드는 작업을 해야 한다. 이때 미리 알려주는 작업이 클래스를 만드는 작업인 것이다. 어떤 객체의 형태(type)를 정의한 것이 class인 것이다.

ex. int, float

 - ▷ 인스턴스 (instance)/ 객체 (object): 특정 클래스에 값이 주어진 실체

ex. 42, 1024, 3.14

B. Class (자료형을 나타내기 위한 설계도)의 구성요소

- ▶ 클래스의 속성 (attributes)
 - ▷ 데이터 속성 (data attributes): Class를 만든다는 것이 자동차 설계도를 만드는 것이라 할 때, 차문, 유리, 차체 등에 해당한다. Class의 변수를 말한다.
 - member variable, field, attributes, properties, characteristics
 - ▷ 메소드(methods): Class를 자동차 설계도를 만드는 것이라 할 때, 데이터 속성을 가지고 할 수 있는 기능들을 정의한 것

을 의미한다. 예를 들면, 엑셀을 봐야 연료를 분사시킬수 있다 는 것이나 버튼을 눌러 창문을 내릴 수 있다는 것이 이에 해당 한다. Class의 데이터 속성을 받아 처리하는 함수를 말한다.

- member functions, methods, responsibilities

▶ 클래스 & 인스턴스 변수 (class & instance variable)

▷ 클래스 변수: Class를 만드는 것이 자동차 설계도를 만드는 것이라 할 때, 자동차의 상표와 같이 Class(자동차 설계도)를 보고 만든 모든 자동차(인스턴스 or 객체)들에 공유되는 변수를 말한다. 즉, Class 내에 정의되어 있기 때문에 그 Class를 통해 만든 인스턴스(or 객체)들이 공통적으로 가지고 있는 변수를 말한다. 따라서, Class를 만들 때, 이 Class를 사용하는 모든 인스턴스가 공통적으로 가져야 할 변수를 만들어야 한다면 클래스 변수로 선언한다.

▷ 인스턴스 변수: Class를 만드는 것이 자동차 설계도를 만드는 것이라 할 때, 자동차의 번호판처럼 Class(자동차 설계도)를 보고 만든 모든 자동차(인스턴스 or 객체)들의 서로 다른 변수를 말한다. 즉, 클래스의 인스턴스 별로 독립적인 변수이다. 따라서, Class를 만들 때, 인스턴스 변수의 자리는 만들어주되, 인스턴스 변수값 할당은 하지 않는다. 인스턴스 변수는 인스턴스에 나타내져 있어야 한다.

▶ 클래스 & 인스턴스 함수 (class & instance function)

▷ 클래스 함수: Class 내에 정의되어 있는 함수로, Class 내부의 클래스 변수를 확인하거나 수정할 수 있는 함수이다. Class 내에 정의되어 있는 함수이기에 이 Class를 공유하는 모든 인스턴스는 당연히 이 함수를 공유(사용가능하다는 것)한다. 당연히 Class 내에 없는 인스턴스 변수에는 접근 불가능하다.

▷ 인스턴스 함수: Class 내에 정의되어 있는 함수로, 같은 Class를 공유하는 모든 인스턴스들 각각에서 자신의 인스턴스 변수를 접근하는 기능을 제공하는 함수이다. 첫 번째 인자의 이름은 어떤 것으로 해도 되지만, 관례적으로 self(인스턴스 자신을 의미함.)를 사용한다.

C. 자료형과 식별자를 반환하는 함수: type(), id()

- ▶ type(): 어떠한 값 혹은 변수에 저장된 값의 자료형을 반환하는 함수이다.
- ▶ id(): 객체의 식별자(객체들을 구별하기 위한 unique한 값이며, 값 자체는 중요하지 않다.)를 반환하는 함수이다. (다른 언어에서는 보통 메모리 주소를 반환하지만, 파이썬은 그렇지 않으므로 신경쓰지 말자.)
- ▶ 같은 Class를 사용하는 서로 다른 인스턴스에 type함수를 쓰면, 같은 값이 나오고 id함수를 쓰면 다른 값이 나온다. 따라서 이름이 다른 인스턴스 2개가 주어졌을 때, 2개가 같은 것을 가리키고 있는지 확인하려면, id 함수를 써보면 된다. 2개가 같은 값이면 같은 인스턴스를 가리키는 2개의 변수명인 것이고, 다른 값이면 서로 다른 인스턴스를 가리키는 2개의 변수명인 것이다.

```
d1 = 42
t1 = [42, 1024, 23]
t2 = t1
t3 = t1.copy()
print(type(d1))
print(type(t1))
print(type(t2))
print(type(t3))
print(id(t1), id(t2), id(t3))
```

```
<class 'int'>
<class 'list'>
<class 'list'>
<class 'list'>
4554987528 4554987528 4552072968
```

D. 식별자 혹은 객체의 값의 동일 여부를 확인하는 예약어: is, is not,

`==, !=`

- ▶ is, is not: 동일한 객체(인스턴스)를 가르키는지 확인하는 예약어이다. 즉, 식별자가 동일한지 판단하는 연산이다.
- ▶ `==, !=`: 객체가 저장하고 있는 내용이 동일한지를 판단하는 연산이다.
- ▶ (Advanced) 사용자가 만든 클래스의 경우,
 - ▷ `==` 연산자 계산을 하려면, Class 내부에 인자로 넘겨진 객체의 내용과 현재 객체 내용의 동일 여부를 검사해서 True 혹은 False를 반환하는 특수한 함수(`__eq__()`)를 정의해주어야 한다.

▷ Class에 이 함수를 만들지 않았을 경우에는 is 연산자의 결과가 == 연산자의 결과로 사용된다.

t1 = [42, 1024, 23] t2 = t1 t3 = t1.copy() print(t1 == t2, t1 is t2) print(t1 == t3, t1 is t3)	True True True False
--	-------------------------

3. Class 정의

A. Class를 간단히 정의하면 '데이터 + 연산(함수)의 좀 더 강한 결합'이다.

B. 클래스(새로운 자료형)의 정의 문법

▶ 클래스(새로운 자료형) 정의 문법

`class ClassName:`

`<statement_1>`

`<statement_2>`

...

`<statement_n>`

▶ 클래스 안에는 임의의 명령문을 사용할 수 있으나, 일반적으로 함수의 정의를 사용한다.

▶ 클래스 이름에 관한 규칙

▷ 변수 및 함수의 이름의 규칙과 동일하다. 즉, 알파벳, 숫자, _로 구성해야하며, 숫자로 시작할 수는 없다.

▷ 일반적으로 사용자가 만드는 클래스는 대문자로 시작하는 이름을 지어준다. (함수의 이름이 보통 소문자로 시작하기 때문에 둘을 구분지어주기 위한 방법이다.)

C. 클래스가 제공하는 연산: 속성 참조와 객체화

▶ 속성 참조 (attribute reference): 클래스 내에 있는 클래스 변수/함수 및 인스턴스 변수/함수를 접근하는 방법을 말한다.

▷ 클래스 변수/함수

- `ClassName.variable_name`
- `ClassName.function_name()`

▷ 인스턴스 변수/함수

- `instance_name.variable_name`
- `instance_name.function_name()`

▶ 객체화 (instantiation): 클래스 파일을 만들고, 다른 파일에서 이 클래스 파일을 부르면(import) 이 클래스는 객체화된다. 그러면 이 클래스를 부른 파일은, 클래스 이름을 함수의 형태로 사용하여, 클래스를 객체로서 사용할 수 있게 된다.

- `ClassName()`

4. Class 변수/함수의 사용

A. 클래스 변수/함수의 정의와 사용

▶ 클래스가 선언되면, 클래스의 정보(어떤 변수, 함수를 가지는지)가 전역 네임스페이스에 저장된다.

▶ 클래스 변수/함수를 사용하는 방법: 변수/함수 이름 앞에 클래스 이름을 같이 써야 한다. 심지어는 클래스의 메소드 내에서도 사용할 때에 앞에 클래스 이름을 같이 써주어야 한다. 예시는 다음과 같다.

▷ 클래스 변수: `ClassName.variable`

▷ 클래스 함수: `ClassName.function`



5. 인스턴스 변수/함수의 사용

A. 클래스의 인스턴스화: 세부적인 사항

▶ 클래스를 인스턴스화 할 때, python이 하는 작업은 다음과 같다.

(1) 인스턴스를 저장할 공간(일종의 네임스페이스)를 만든다.

(2) `__init__(self, ...)` 함수를 호출한다.

① 이때, 첫 번째 인자인 `self`는 (1)에서 만든 공간을 가리킨다.

② `self` 이외의 다른 인자들은 함수 호출과 같은 규칙으로 `__init__()`의 매개변수로 전달된다.

(3) 새로 만들어진 공간(즉 `self`가 가르키는 공간)이 반환값처럼 반환된다.

▶ `__init__()` 함수의 특성

▷ 이름은 바꿀 수 없다. 무조건 써야 한다.

▷ 반환값을 가질 수 없다.

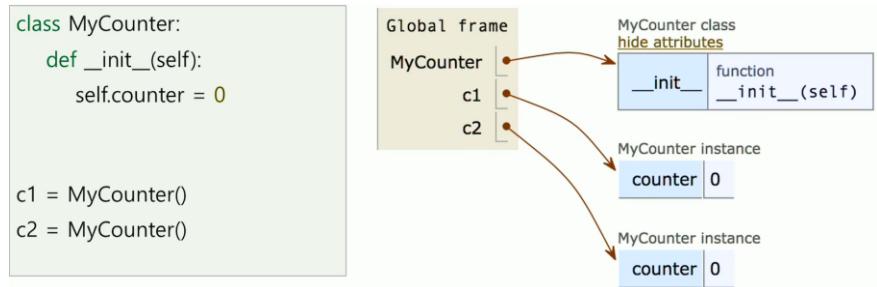
▷ 참조값을 전달받는 첫 번째 매개 변수의 이름은 관례적으로 `self`를 사용한다.

B. 예시: 클래스의 인스턴스화

▶ 인스턴스 변수: 인스턴스마다 독립적으로 가지고 있는 변수들을 말한다.

▷ 이 인스턴스 변수는 클래스를 인스턴스화할 때 만들어진 공간에 저장된다.

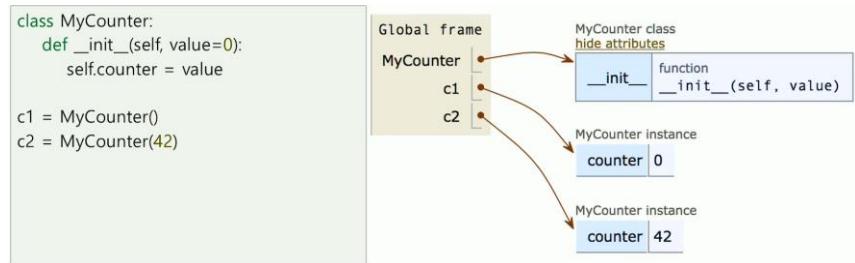
▷ 클래스를 만들 때, `__init__()` 혹은 다른 인스턴스 함수에서 `self`를 변수로 받도록 정의하면, 클래스의 인스턴스화가 완료된 후 그 함수를 사용해서 실제로 이 인스턴스 변수에 접근하는 것이 가능해진다.



▷ 위 예제는 MyCounter라는 부를 때마다 Counting을 해주는 객체를 설계해서 c1이라는 인스턴스와 c2라는 인스턴스 두 개를 실제로 만들어낸 것이다.

C. __init__() 함수에서 인자 사용하기

- ▶ 다른 함수와 마찬가지로 매개변수, 디폴트 인자 등을 가질 수 있다.
- ▶ 참고: 첫 번째 인자인 self는 python이 자동으로 채워주기 때문에, 코드를 짤 때, 굳이 self라는 인자를 제공할 필요는 없다.



D. 인스턴스 함수/변수의 사용

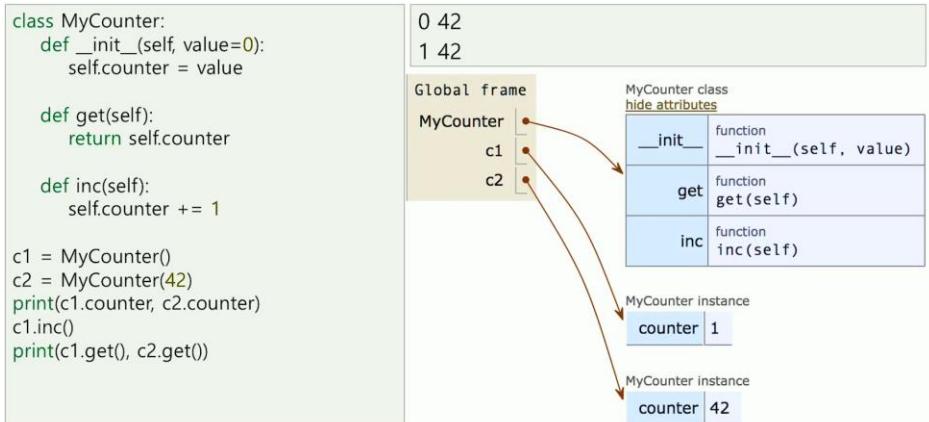
- ▶ 인스턴스 함수/변수의 사용: 객체(인스턴스) 뒤에 '.'을 찍고, 함수/변수 이름을 이어서 쓰면 인스턴스 함수/변수에 접근할 수 있다. 예시는 다음과 같다.

- ▷ instance_variable.function()
- ▷ instance_variable.variable
- ▶ 인스턴스 함수의 특징
 - ▷ 첫 번째 인자 (관례적으로 self라는 이름을 사용)로 객체의 참조값이 전달된다. (함수를 호출했을 때, python이 자동으로 전달하므로, 함수 호출문에 self를 쓸 필요는 없다.)

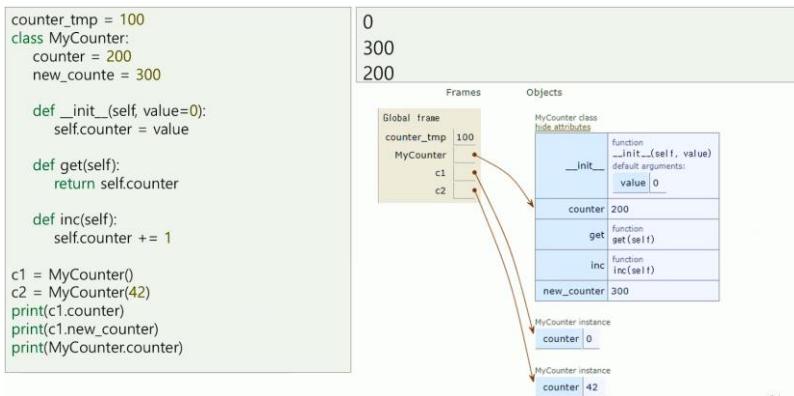
▷ 인스턴스 함수의 내부에서 아래와 같은 방법으로 자기 자신을 대상으로 하는 또 다른 인스턴스 변수 혹은 인스턴스 함수의 사용이 가능하다.

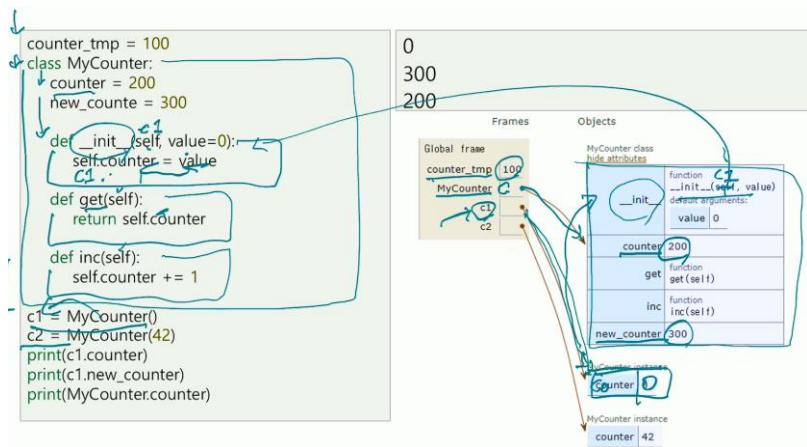
- self.function()
- self.variable

E. 예시: 인스턴스 변수/함수



F. Class와 Namespace 사이의 관계



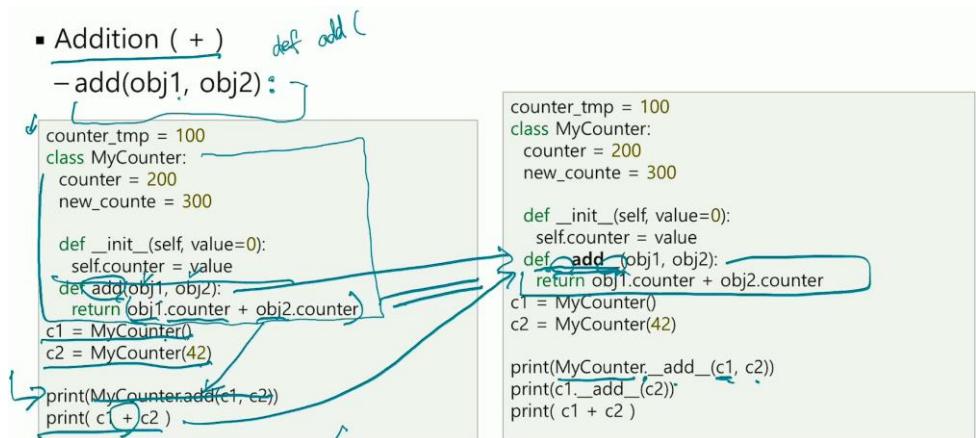


▶ c1.counter와 같이 c1의 counter라는 변수를 물어보면, 컴퓨터는 먼저 c1에 가서 counter라는 변수(인스턴스 변수)가 있는지 찾고 있으면, 그 counter값을 리턴하며, 없으면 class가 정의된 곳으로 가서 counter변수(클래스 변수)가 있는지 찾고 있으면 counter값을 리턴한다. 여기에도 없으면, Error를 표시한다.

▶ 이때, self를 변수로 받는 함수들은 인스턴스 메소드(함수)이고, self라는 변수 없이 동작하는 함수들은 클래스 메소드(함수)이다.

▶ python tutor를 사용해서 코드가 어떻게 작동하는지 잘 살펴보도록 한다. : <http://www.pythontutor.com/visualize.html#mode=edit>

G. Class에 직접 Operation 연산자를 써서 연산을 진행하도록 하는 방법 (Binary Arithmetic Operations)



▶ 왼쪽에서는 c1 객체 안의 counter값, c2 객체 안의 counter값을 더하는 add 함수(Class 함수이다.)를 정의했다. 그래서 마지막 줄 바로 전 줄에 보면, 이 함수를 쓰기 위해서 MyCounter.add(c1, c2)와 같은 함수 호출이 있다.

출문을 작성하였다.

▶ 그런데, Mycounter의 인스턴스화(객체화)에 따라 생성된 객체 내에는 어짜피 변수가 counter 밖에 없다. 그러면 그냥 $c1 + c2$ 로 쓰면, $c1$ 객체 안의 counter값, $c2$ 객체 안의 counter값을 더하는 작업이 진행되도록 만들고 싶다고 생각이 든다. 그렇다고 왼쪽 코드 마지막 줄처럼 $c1 + c2$ 만 추가해서는 연산이 되지 않는다.

▶ 이때, 사용되는 것이 `_add_()` 함수이다. 오른쪽 코드처럼 `_add_(obj1 + obj2)`라는 함수를 정의하게 되면, `MyCounter._add_(c1, c2)`로 써도 $c1$ 객체 안의 counter값, $c2$ 객체 안의 counter값을 더하는 작업이 일어나며, `c1._add_(c2)`라고 써도 $c1$ 객체 안의 counter값, $c2$ 객체 안의 counter값을 더하는 작업이 일어난다. 또한, $c1 + c2$ 라고 써도 $c1$ 객체 안의 counter값, $c2$ 객체 안의 counter값을 더하는 작업이 일어난다. python의 interpreter가 `+`를 `_add_()` 함수로 변환하여 받아들이기 때문이다.

▶ 이 `_add_()` 외에도 python의 interpreter에는 특수 기호로 표현된 연산자들을 의미하는 함수들이 정의되어 있다. 보통 그런 함수들은 `_function_()`와 같은 형식으로 정의되어 있으므로, 내가 만든 Class에서 그와 같은 기능을 제공해주고 싶다면 그런 함수들을 찾아 정의해주어야 한다. 이 내용은

<https://docs.python.org/3.7/reference/datamodel.html?highlight=add#object.add>에 나와있다.

H. 사용하는 이름 목록을 반환하는 함수: `dir()`

```
counter_tmp = 100
class MyCounter:
    counter = 200
    new_counte = 300

    def __init__(self, value=0):
        self.counter = value
        self.counter1 = value

    def get(self):
        return self.counter

    def inc(self):
        self.counter += 1

c1 = MyCounter()
print(dir(MyCounter))
print(dir(c1))
print(c1.__dict__)
```

['_class', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init__.subklass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__return__', '__setattribute__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'counter', 'get', 'inc', 'new_counter']

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init__.subklass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__return__', '__setattribute__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'counter', 'counter1', 'get', 'inc', 'new_counter']

{'counter': 0, 'counter1': 0}

Frames

Global frame
counter_tmp 100
MyCounter

Objects

MyCounter class
 Attributes
 Function
 __init__(self, value)
 Value 0

 counter 200

 Function
 get (self)

 Function
 inc (self)

 new_counter 300

MyCounter instance
 counter 0

 counter1 0

23

▶ `dir()` 함수: `dir(object)`의 형식으로 사용하는 함수로, object를 대상으

로 사용할 수 있는 함수들의 이름과 object에서 호출하거나 할당할 수 있는 변수들의 이름을 리스트로 만들어 반환하는 함수이다.

- ▶ dict(object)에서 object로 class name을 집어 넣으면, 그 class의 class 함수, class 변수, 인스턴스 함수가 나온다.
- ▶ dict(object)에서 object로 instance name을 집어 넣으면, 그 instance의 class 함수, class 변수, 인스턴스 함수, 인스턴스 변수가 나온다.
- ▶ 남이 만들어 놓은 객체를 내가 사용하고자 할 때, 어떻게 사용할 수 있는지 전혀 모르므로 그 객체를 대상으로 할 수 있는 작업들을 모두 찾아보아야 한다. 이때, 이 작업들의 이름을 리스트로 만들어 리턴해주는 dir() 함수를 사용하는 것이다.
- ▶ 그런데, dir() 함수를 사용하면 class에 정의된 것보다 더 많은 함수, 변수들이 나오는데, class를 정의하면 거기에 정의한 함수, 변수에 python이 기본적으로 지원하는 함수들, 변수들이 추가되기 때문이다.

6. 클래스 디자인 예시

A. 수업 관리 클래스(객체) 만들기.

- ▶ 구현되어야 하는 변수
 - ▷ 학생 (instance variable)
 - ▷ 과목 명 (class variable)
- ▶ 구현되어야 하는 함수
 - ▷ 수강생 출력 (class function)
 - ▷ 특정 학생 점수 출력 (instance function)
 - ▷ 평균 출력 (class function)
 - ▷ 정렬 (class function)

- ▶ 그런데, 이때, 학생도 클래스(객체)화 하는 것이 가능하다.

- ▷ 학생이라는 클래스에서 구현되어야 하는 변수

- 이름 (instance variable)

- 학번 (instance variable)
- 과제점수 (instance variable)
- 중간고사 점수 (instance variable)
- 기말고사 점수 (instance variable)
- 반영비율 (class variable)

▷ 학생이라는 클래스에서 구현되어야 하는 함수

- 학점 계산
- 비교 연산 등등

▷ 학생이라는 객체를 클래스로 구현한 모습

```
class Student:
    rate = (40, 30, 30)
    def __init__(self, name, id,
                 homework=0, mid=0, final=0):
        self.name = name
        self.id = id
        self.homework = homework
        self.mid = mid
        self.final = final

    def __repr__(self):
        return self.name

    def __eq__(self, obj):
        if self.id == obj.id:
            return True
        else:
            return False
```

$a = \text{Student}()$

- 이때, __eq__ 함수를 Student Class를 기반으로 하는 두 instance의 student id가 같은지를 확인하는 연산이라고 정의하였기 때문에 `student1 == student2`라고 하면 컴퓨터는 두 instance의 student id가 같은지 boolean 연산 한다.
- 만약, __eq__ 함수를 따로 정의해주지 않는다면, `student1 == student2`를 컴퓨터가 `student1 is student2`로 고쳐서 읽어서 `student1`이 가리키는 메모리 상의 공간과

student2가 가리키는 메모리 상의 공간이 같은지 확인하는 Boolean 연산을 진행한다. 즉 두 instance(namespace)가 가리키는 공간이 같은 것인지 확인하는 Boolean 연산을 진행한다.

▶ 수업 관리 클래스(객체)를 구현한 모습

```
class ClassManager:  
    """수업관리 class"""  
  
    def __init__(self):  
        pass  
  
    def sort(self):  
        pass  
  
    def get_grade(self, name):  
        pass  
  
    def print_all_students(self):  
        pass
```

7. 그 외에 객체지향 프로그래밍에서 중요한 내용(2학기에 열리는 객체지향 프로그래밍 과목에서 다루게 됨.)

A. 객체지향 프로그래밍의 여러 기법

- ▶ 은닉화(encapsulation)
- ▶ 상속(inheritance)
- ▶ 다형성(polymorphism)
- ▶ 추상화(abstraction)

B. 왜 객체지향을 써야 하는가에 대한 고찰

C. 디자인 패턴

D. 다양한 예제

E. 등등

8. 읽을 거리

A. 클래스에 대한 추가적인 설명

▶ <http://www.flowdas.com/thinkpython/15-classes-and-objects/>

B. Advanced: (python을 이용한) 객체지향 프로그래밍에 대한 소개

▶ 파이썬 – OOP (한국어)

▶ python3 – OOP (영어)

X. 모듈

i. 모듈

1. 모듈의 사용법

A. 모듈/패키지 사용하기

▶ 모듈을 읽어들이는 방법

▷ 일반적으로 `import module_name`을 파일 맨 앞에 써서 파일이 해당 `module_name`의 함수 및 변수를 사용한다는 것을 알린다. 이렇게 쓰면, 컴퓨터는 `global namespace`에 `module_name`이라는 변수를 만들고, 이 변수에 해당파일이 저장된 공간(`module_name`이라는 `namespace`)으로 가는 주소를 할당한다.

▷ 여기서 `module_name`은 확장자 `.py`를 뺀 파일 이름을 말한다.

▶ 모듈(모듈 내의 변수 및 함수) 사용방법: `module_name.xxx`와 같이 사용하면 모듈에 속한 함수 혹은 변수 이름을 뜻한다. 사용예시는 다음과 같다.

▷ `module_name.function_name()`

▷ `module_name.variable_name`

B. 예제: `math`

<pre># using math import math # constants print(math.pi) print(math.e) # power and logarithmic functions x = 1.0 print(math.exp(x)) # return e ** x print(math.log(x)) # return natural log(x) print(math.log10(x)) # return log(x) with base 10 print(math.sqrt(x)) # return square root(x) # trigonometric functions print(math.cos(x)) # return cos(x) print(math.sin(x)) # return sin(x)</pre>	<pre>3.141592653589793 2.718281828459045 2.718281828459045 0.0 0.0 1.0 0.5403023058681398 0.8414709848078965</pre>
--	--

C. `import` 하는 여러가지 방법 1(모듈에서 원하는 값들만 읽어들이는 방법)

- ▶ 일반적인 방법 (모듈의 namespace를 만들어 거기에 모듈의 내용을 전부 import하는 방법)

```
import math  
print(math.pi, math.e)
```

- ▶ 특정 객체(변수, 함수, 클래스 등)를 그 이름 그대로 import 하는 방법

```
from math import pi, e  
print(pi, e)
```

- ▶ 모든 객체(*표시가 모든을 의미한다.)를 그 이름 그대로 import하는 방법 (사용자가 정의한 변수, 함수, 클래스와 혼동이 일어나기 쉽기 때문에 권장하지 않는 방법이다.)

```
from math import*  
print(pi, e, tau)
```

D. import 하는 여러가지 방법 2(모듈의 이름을 줄여서 읽어들이는 방법)

- ▶ 모듈 이름을 간략하게 줄여서 가져오는 것이 가능함.: 단, 모듈 이름을 줄일때는 관례적인 경우를 따르는 것이 좋음.

▷ 예시1: import math as m

```
print(m.pi, m.e)
```

▷ 예시2: import numpy as np

▷ 예시3: import pandas as pd

- ▶ 그런데 이렇게 모듈의 이름을 줄여서 가져오면, 그 모듈을 사용할 때는 줄인 이름만 사용할 수 있다. 즉, math를 m이라는 줄인이름으로 import했다면, m.pi와 같은 방식으로만 사용할 수 있다는 것이다. math.pi라고 쓰면 Error가 난다. 그 이유는 import math as m이라 쓰면, 컴퓨터는 global namespace에 m이라는 변수를 만들고 이 변수에 math라는 모듈이 저장된 namespace로의 주소(pointer, reference)를 할당하기 때문이다.

2. 모듈을 만드는 방법과 import의 의미

- ▶ 어떤 파일이든 모듈로 사용하는 것이 가능하다.
 - ▷ .py를 제외한 파일 이름이 모듈 이름이 된다.
 - ▷ 파일 안에서 정의한 변수, 함수, 클래스를 import해서 사용하는 것이 가능하다.

- ▶ 모듈을 import하면 다음의 작업이 순서대로 시행된다.
 - ▷ 모듈을 위한 별도의 namespace가 생성된다.
 - ▷ 모듈을 위한 namespace에서 import 대상인 파일 프로그램을 한 줄씩 읽으면서 실행한다.
 - 이 과정에서 변수, 함수, 클래스 등이 있으면, namespace 안에 해당 내용이 생성된다.
 - 이 과정에서 화면 출력 등과 같은 입출력 명령을 만나면 그 명령을 그대로 실행한다.

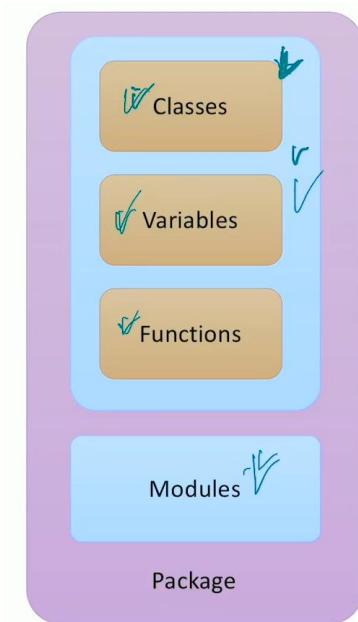
3. 예시: 모듈의 생성 및 사용

<pre>#vector.py def add(v1, v2): v = [] for i in range(len(v1)): v.append(v1[i] + v2[i]) return v</pre>	<pre>#main.py import vector t1 = [42, 1024, 23] t2 = [6, 28, 496] print(vector.add(t1, t2))</pre>
[48, 1052, 519]	

- ▶ 왼쪽과 같은 vector.py라는 파일을 만들어 두고, 오른쪽의 main.py라는 파일에서 vector.py를 import하는 예시이다.

4. 패키지(Package)란?

- ▶ 하나의 Object를 만들기 위해서 관련된 Classes, Variables, Functions를 모두 모아놓은 파일이 module이었다.
- ▶ 그런데 이런 module을 여러 개를 한꺼번에 사용해야 되는 경우가 있다. 예를 들면 수학 연산과 기하학과 벡터연산을 한꺼번에 하고 싶게 되는 경우 math 모듈, 기하학 모듈, 벡터 모듈을 각각 import해서 가져와야 하는 것이다.
- ▶ 그런데 import해야 하는 파일이 상당히 많아지면 번거로워지므로, 관련된 module들을 하나로 묶어 패키지를 만들고 패키지를 가져오는 것으로 그 module들의 내용을 모두 쓸 수 있게 하였다. 이것이 패키지이다.



5. 패키지를 만드는 방법

- ▶ 패키지를 사용하고자 하는 python 파일이 있는 디렉토리에 패키지 파일을 만들어야 해당 python 파일에서 그 패키지를 사용할 수 있다.
- ▶ 패키지 파일 안에 사용하고자 하는 모듈 파일들을 넣으면, python 파일에서 그 패키지 파일을 import하는 것으로써 그 모듈파일들을 사용할 수 있게 된다.
- ▶ 예전에는 패키지 파일 안에 사용하고자 하는 module 파일들을 다같이 집어넣고 __init__.py라는 파일을 하나 더 만들어주어야 패키지로서 사용할 수 있었다. 그런데, 요즈음에는 __init__.py를 안만들어도 패키지

파일로 사용할 수 있다.

- ▶ 패키지 파일을 python 파일에서 사용하는 방법은 newpkg라는 패키지 안의 module1을 쓰고 싶다면, import newpkg.module1이라 쓰는 것이다.
- ▶ newpkg 패키지 파일 안의 newminipkg 패키지 파일 안의 module1을 쓰고 싶다면, import newpkg.newminipkg.module1처럼 쓰면 된다. 즉, 패키지를 안에 패키지를 넣어서 만들어서 python 파일에서 접근하는 것도 가능하다.

6. 파이썬이 실행될 때 자동으로 만들어지는 전역변수: `_name_`

- ▶ `_name_`: 파이썬 프로그램이 실행될 때, 어떤 파일을 가져와서 실행하고 있는지를 나타내는 전역변수
 - ▷ 개별적인 python 파일을 실행할 때(예: elice, pycharm, shell 등에서 실행): '`_main_`'이라는 값이 `_name_`에 할당됨.
 - ▷ 한 파일이 모듈로 import되어 실행될 때: 모듈 함수가 실행될 때 '`_moduleName_`'이 `_name_`에 할당된다. 그 외에 개별적인 python 파일의 변수나 함수가 실행될 때는 '`_main_`'이라는 값이 `_name_`에 할당된 상태가 된다.
- ▶ 참고: 아래와 같이 모듈 함수 내에 if문을 작성해서 module을 import하지 않았을 때만 수행되는 테스트 코드 등을 넣는 것이 일반적임

```
if __name__ == '__main__':
    # example code or test code
    # ...
    .print()
```

- ▶ 예제: fibonacci.py 파일을 모듈로만 사용하는 것이 가능하도록 짠 코드

```

""" Provides Fibonacci related functions"""

def fibonacci_list(n):
    """Return Fibonacci series up to n"""
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if __name__ == '__main__':
    assert(fibonacci_list(10) == [1, 1, 2, 3, 5, 8])

```

12

- ▷ module로 사용되어 fibonacci.fibonacci.list(n)으로 사용되면 if문은 함수 정의문 바깥에 있으므로 실행되지 않는다. 정의문 안에 위치하더라도 모듈로 사용된거여서 __name__은 '__fibonacci__'가 되어 if문은 false가 되어 실행되지 않는다.
- ▷ 하지만, fibonacci.py를 직접 켜서 파일을 실행하면 if문까지 실행되어서 assert(fibonacci_list(10) == [1, 1, 2, 3, 5, 8])가 실행된다.
- ▷ 이때, assert()는 ()안의 값이 True일 때, error를 발생하는 함수인데, fibonacci.py를 직접 켜서 파일을 실행하면 fibonacci_list(10)는 항상 [1, 1, 2, 3, 5, 8])이 되므로 if문이 무조건 실행되어 코드가 Error가 난다.
- ▷ 이렇게 사용하는 것이 좋은 이유는 내가 module을 직접 만들 때, module내의 함수들이 제대로 동작하는지 테스트해야 하는데, 테스트한 결과가 이 module로 사용할 때도 실행되면 안되므로, 이렇게 쓰는 것이다. 이렇게 쓰면 모듈로 사용할 때는 테스트코드가 실행되지 않기 때문이다.

ii. 함수/클래스/모듈의 문서화

1. docstring

- ▶ docstring이란 클래스, 함수 등의 설명을 적은 것을 말한다.
 - ▷ 클래스나 함수의 선언문 바로 뒤에 여러줄에 걸친 문자열 ("""로 시작과 끝을 표시한 것)을 의미한다.
 - ▷ 참고: <https://www.python.org/dev/peps/pep-0257>

```

class MyCounter:
    """MyCounter is a simple implementation of a counter"""
    def __init__(self, value=0):
        self.counter = 0

    def inc(self):
        """Increase counter value by 1"""
        self.counter += 1

    def get(self):
        """Return the current counter value"""
        return self.counter

    print('Help on MyCounter')
    help(MyCounter)
    print('Help on MyCounter|inc()')
    help(MyCounter|inc)
    print('dir() on MyCounter')
    print(dir(MyCounter))

```

2. help()

- ▶ `help()` 함수는 클래스에 대한 설명을 출력하는 함수이다. (docstring 을 포함한 설명을 출력한다.)

```

Help on MyCounter
1 Help on class MyCounter in module __main__:
2
3 class MyCounter(builtins.object)
4 | MyCounter is a simple implementation of a counter
5 |
6 Methods defined here:
7
8 | __init__(self, value=0)
9 |     Initialize self. See help(type(self)) for accurate signature.
10 |
11 | get(self)
12 |     Return the current counter value
13 |
14 | inc(self)
15 |     Increase counter value by 1
16 |

```

```

17 | -----
18 | Data descriptors defined here:
19 |
20 | __dict__
21 |     dictionary for instance variables (if defined)
22 |
23 | __weakref__
24 |     list of weak references to the object (if defined)

```

```

Help on MyCounter|inc()
1 Help on function inc in module __main__:
2
3 inc(self)
4     Increase counter value by 1

```

3. dir()

- ▶ `dir()` 함수는 클래스의 속성(함수, 변수)을 모두 출력하는 함수이다.

```
dir() on MyCounter
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
 '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'get', 'inc']
```

iii. Python Standard Library: 파일에 기본적으로 내장되어 있는 모듈들

1. Generic Operating System Services (컴퓨터로부터 값을 받아오는 함수를 제공하는 모듈)

A. time – Time access and conversions

- ▶ time module이란?: 컴퓨터의 시스템에서 시간값을 읽어 가져오는 함수들을 제공하는 모듈이다.
- ▶ 사용 예시

```
import time
now = time.localtime()

s = "%04d-%02d-%02d %02d:%02d:%02d" % (now.tm_year,
now.tm_mon, now.tm_mday, now.tm_hour, now.tm_min, now.tm_sec)
print(s)

print(time.time())
```

2019-05-28 07:29:47
1588857441.7334049

- ▷ time.time() 함수: 코드를 실행할 때의 초를 불러온다.

2. Numeric and Mathematical Modules numbers – Numeric abstract base classes (수적인 연산을 하는 함수를 제공하는 모듈)

A. math – Mathematical functions

B. random – Generate pseudo-random numbers

- ▶ random module이란?: 숫자를 또는 문자 등을 랜덤으로 출력하는 함수들을 제공하는 모듈이다.

▶ 사용 예시

<pre>import random # using random module print(random.random()) # float, 0.0 <= x < 1.0 print(random.uniform(1, 10)) # float, 1.0 <= x < 10.0 print(random.randrange(10)) # int, 0 <= x < 10 print(random.randrange(1, 10)) # int, 1 <= x < 10 print(random.randint(1, 10)) # int, 1 <= x <= 10 # choice(): select one element in a given sequence print(random.choice('abc'))</pre>	0.3677067585658439 1.0474826989553674 0 1 3 b
---	--

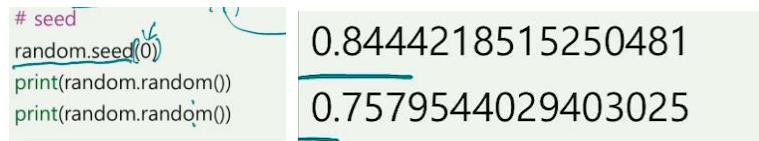
* 실행할 때마다 결과가 달라질 수 있음

19

- ▷ random.random() 함수: 0보다 같거나 크고 1보다는 작은 float 값을 랜덤으로 반환하는 함수이다.
- ▷ random.uniform(a, b) 함수: a보다 같거나 크고 b보다는 작은 float 값을 랜덤으로 반환하는 함수이다.
- ▷ random.randrange(b) 함수: 0보다 같거나 크고 b보다는 작은 int 값을 랜덤으로 반환하는 함수이다.
- ▷ random.randrange(a, b) 함수: a보다 같거나 크고 b보다는 작은 int 값을 랜덤으로 반환하는 함수이다.
- ▷ random.randint(a, b) 함수: a보다 같거나 크고 b보다는 작거나 같은 int 값을 랜덤으로 반환하는 함수이다.
- ▷ random.choice('string') 함수: string을 구성하는 문자 중에서 아무 문자나 랜덤으로 1개 뽑아서 반환하는 함수이다.
- ▶ 이 ramdom 모듈의 함수들은 마치 random으로 값을 내는 것 같지만 실제로 컴퓨터는 랜덤으로 값을 만드는 것이 불가능하다. 그래서 컴퓨터는 어떤 Operation을 통해 만든 값을 추출하는데 그 값이 마치 random인 것처럼 보이는 것이다. 대표적인 예시로 C언어는 random 함수가 시간값을 읽고 여기에 곱이나 차 등의 연산을 써서 반환값을 만드는 방식으로 동작해서 마치 랜덤값이 나오는 것처럼 보인다.
- ▶ 그런데, 이렇게 랜덤으로 출력되는 코드를 짜면 이 random

값의 변화에 따라 코드가 오류가 날 수도 있고 정상동작할 수도 있다. 그래서 random 코드를 사용하면 코드의 유지, 보수, 디버깅이 힘든데, 그래서 이 모듈은 random.seed(a) 함수를 제공한다.

▶ random.seed(a) 함수: 컴퓨터가 랜덤변수를 출력하는 연산의 기준은 a라는 숫자에 맞춰주는 것이다. 컴퓨터의 랜덤연산이 실은 규칙적인 계산이기 때문에 이렇게 맞춰주면 다음에 출력되는 숫자의 규칙이 직접적으로 보여서 다음에 출력되는 값을 유추할 수 있다. 그래서 랜덤변수를 사용한 코드는 이 함수를 사용해서 유지, 보수, 디버깅을 한다. 다음에 나올 값이 무엇인지 대강 알 수 있으므로, 그 값에 따라 오류를 찾기가 쉽기 때문이다.



```
# seed
random.seed(0)
print(random.random())
print(random.random())
```

0.8444218515250481
0.7579544029403025

3. Text Processing Services (Regular expression을 처리하는 함수를 제공하는 모듈)

A. re – Regular expression operations

B. 정규식(regular expression) 처리란?

▶ 어떤 웹의 텍스트를 긁어오거나 텍스트 파일의 텍스트를 긁어오거나 바이너리 파일의 텍스트를 긁어와서 데이터를 추출해보면, 추출이 어렵다는 것을 알 수 있다.

▶ 그 이유는 Contact: 가 쓰인 문장에서 Contact: 뒤에 쓰인 문장을 가져오라고 했는데, 어떤 파일에서는 Contact: 라 쓰여 있을지는 모르지만, _Contact: 또는 Contact :라고 쓰여있을 수도 있기 때문이다.

▶ 위처럼 Contact: 가 쓰인 문장에서 Contact: 뒤에 쓰인 문장을 가져오라고 쓰면, _Contact: 또는 Contact :라고 쓰여있는 곳 뒤에 있는 전화번호는 들고 오지 못하게 되는 것이다.

▶ 이제 이것을 해결하기 위해 쓰는 것이 정규식 처리 모듈이다.

▶ 정규식 처리 모듈은 문자의 패턴을 가지고 패턴에 맞는 문자열을 찾아내는 것을 도와준다. 예를 들면 전화번호는 숫자숫자숫자-숫자숫자숫자-숫자숫자숫자숫자 꼴의 패턴을 가진 문자열이므로 정규식 처리모듈을 써서 이 패턴을 정의하고 텍스트를 읽을 때 이 패턴에 맞는 문자열은 출력하라고 하면 앞에서 말한 문제가 발생하지 않고 전화번호들을 제대로 가져올 수 있게 되는 것이다.

▶ 한마디로 정규식 처리 모듈은 문자열 패턴을 정의하는 것을 도와주는 모듈이다.

C. 정규식(regular expression) 처리의 방법 (문자열 패턴의 정의 방법)

* Pattern

. : 줄바꿈 기호를 제외한 한글자와 매칭
[] : 대괄호 안의 문자와 한글자 매칭
예) [012], [0-9], [a-z], [0-9a-zA-Z]
[\t\n\r], [^a-z]

▶ 정규식 처리는 일단 re 모듈에서 지원하므로 re모듈을 import하고 시작한다.

▶ .은 문자열 패턴에서 한 글자를 의미한다.

▶ [012]라고 쓰면 그 자리에 있는 한 글자가 0또는 1 또는 2로 쓰여진 패턴을 가지고 있다는 뜻이다.

▶ [0-9] 라고 쓰면 그 자리에 있는 한 글자가 0부터9까지의 수 중 아무 숫자 하나로 쓰여진 패턴을 가지고 있다는 뜻이다.

▶ [0-9a-zA-Z] 라고 쓰면 그 자리에 있는 한 글자가 0부터9까지의 수나 a-z까지의 알파벳 혹은 A-Z까지의 알파벳 중 아무 글자 하나로 쓰여진 패턴을 가지고 있다는 뜻이다.

▶ [WtWnWr] 라고 쓰면 그 자리에 있는 한 글자가 tab이나 엔터키(줄바꿈), 스페이스바(띄어쓰기) 중 아무거나 하나로 쓰여진 패턴을 가지고 있다는 뜻이다.

▶ [^a-z] 라고 쓰면 그 자리에 있는 한 글자가 a-z까지의 알파벳으로는 절대 쓰이지 않은 패턴을 가지고 있다는 뜻이다.

▶ 그래서 이 정규식 처리 모듈을 가지고 전화번호를 패턴화하면,
'[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]' 가 된다.

▶ 근데 이렇게 같은 패턴이 반복되는 문자열이 있는 경우가 있는데 이 경우에는 다음과 같은 연산자를 써서 패턴 표현을 간단하게 줄일 수 있다.

{m,n} : m번 이상 n번 이하 반복

? : {0, 1}

* : {0, }

+ : {1, }

▷ {m,n}은 해당 문자패턴이 m번 이상 n번 이하 반복된다는 의미이다.

▷ {0, 1}은 ?로 바꿔서 표현할 수 있다.

▷ {0, }은 *로 바꿔서 표현할 수 있다.

▷ {1, }은 +로 바꿔서 표현할 수 있다.

▶ 같은 패턴이 반복되는 문자열 연산자를 쓰면
'[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9]' 게 썼던 전화번호 문자열 패턴이
'[0-9]{2,3}-[0-9]{3,4}-[0-9]{4}' 와 같이 간단해 진다.

D. re – Regular expression operations module의 사용예시

```

import re

st1 = "Affiliation: DGIST
ID: 201911000
Contacts : 054-785-6648"
st2 = "Affiliation: DGIST
ID: 202011999
Contacts : 054-785-6649
Mobile : 011.0000.1234"

# re.match(), re.search()
print(re.findall('20[12][0-9]11[0-9]{3}', st1))
print(re.findall('20[12][0-9]11[0-9]{3}', st2))
print(re.findall('0[0-9]{1,2}?([0-9]{3,4})?([0-9]{4})', st1))
print(re.findall('0[0-9]{1,2}.?([0-9]{3,4}).?([0-9]{4})', st2))

```

24

4. Internet Protocols and Supportwebbrowser – Convenient Web-browser controller (웹브라우저를 컨트롤하는 함수를 제공하는 모듈)

A. urllib – URL handling modules

B. URL을 처리하는 방법

- ▶ 웹브라우저를 컨트롤하는 함수를 제공하는 모듈을 사용하면 웹에서 데이터를 긁어와서 빅데이터 분석을하거나(크롤링이라 부른다.) 연구에 반영하는 등의 매우 유용한 활용을 하는 것이 가능해진다.
- ▶ 그래서 먼저 인터넷 환경에 대해 알아보고 웹브라우저를 컨트롤하는 함수를 제공하는 모듈에 대해 알아보도록 하자.
- ▶ 인터넷 환경은 URL이라는 주소를 가지고 들어가게 된다. 이 URL은 보통 <http://algo.dgist.ac.kr/login.html>과 같이 되어 있는데, 여기서 <http://>는 프로토콜을 의미하고, algo.dgist.ac.kr은 서버의 위치를, login.html은 서버에 있는 해당 웹페이지 파일을 의미한다.
- ▶ 그리고 서버의 위치는 algo.dgist.ac.kr로 쓰여져 있지만 사실은 10.180.2.80과 같은 ip 주소(컴퓨터에 내장된 네트워크 카드 번호)로 되어 있다. DNS가 중간에 매개해서 algo.dgist.ac.kr에 맞는 ip주소를 읽어 10.180.2.80로 접속하게 되는 것이다.
- ▶ 그리고 웹페이지는 웹페이지 파일로 만들어지는데, 이 파일 역시 python과 비슷하게 html이라는 mark up language 파일로 구성되어 있다. 웹 브라우저가

python 인터프리터 같이 코드를 읽어서 웹 화면으로 만들어 보여주는 방식으로 동작한다.



login.html

▶ 그러면 이제 이 html 파일을 읽어서 의미있는 데이터들을 추출하는 작업이 이제 크롤링 작업이 된다.

C. urllib 사용예시 (크롤링 작업으로의 사용예시)

```
from urllib.request import urlopen

with urlopen('http://10.180.2.80/test.html') as f:
    txt = f.read() # return byte string

print(txt)
```

Document

Date : 2020/5/5
날짜 : 비
미세먼지 : 좋음
Location: DGIST
Contact: 054-785-6648

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Document</title>
</head>
<body>
<pre>
Date : 2020/5/5
날짜 : 비
미세먼지 : 좋음
location: DGIST
Contact: 054-785-6648
</pre>
</body>
</html>
```

```
b'<!DOCTYPE html>\r\n<html\r\nlang="en">\r\n<head>\r\n<title>Document</title>\r\n</head>\r\n<body>\r\n<pre>\r\nDate : 2020/5/5\r\n날짜 : 비\r\n미세먼지 : 좋음\r\nlocation: DGIST\r\nContact: 054-785-6648\r\n</pre>\r\n</body>\r\n</html>\r\n'
```

▶ from urllib.request import urlopen: urllib이라는 패키지 파일의 request라는 모듈에서 urlopen이라는 함수를 가져오도록 하는 코드이다.

▶ urlopen('webpageURL')이라고 쓰면 해당 웹페이지의 html코드를 파이썬이 읽어서 byte string으로 가져

을 수 있게 된다.

▶ 그런데 html파일은 byte string(ASCII)이 아니라 utf-8형식을 따라 문자가 쓰여져 있다.

▶ 그래서 그냥 txt = f.read()라고 쓰면 알수없는 코드들이 나와서 txt = f.read()로 받고, txt.decode('utf-8')이라고 써서 binary string(byte string)을 utf-8 형식으로 바꿔 출력하도록 해야 해당 웹페이지가 어떤 html코드로 쓰여져 있는지 볼 수 있게 된다.

```
print(txt.decode('utf-8'))
```

```
<!DOCTYPE html>WrWn<html  
lang="en">WrWn <head>WrWn  
<title>Document</title>WrWn  
</head>WrWn <body>WrWn  
<pre>WrWnDate : 2020/5/5WrWn날씨 :  
비WrWn미세먼지 : 좋음WrWnlocation:  
DGISTWrWnContact: 054-785-6648WrWn  
</pre>WrWn  
</body>WrWn</html>WrWn
```

▶ 근데 이렇게 가져와도 알아보기 힘든건 마찬가지이다. 그래서 html코드의 정리방식으로 정리하기 위해서 다음 코드를 써주면 아래 사진처럼 html코드의 정리방식으로 python의 터미널에 출력되게 된다.

```
for sentence in(txt.decode('utf-8')).splitlines():  
    print(sentence)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Document</title>
</head>
<body>
<pre>
Date : 2020/5/5
날씨 : 비
미세먼지 : 좋음
location: DGIST
Contact: 054-785-6648
</pre>
</body>
</html>
```

- ▶ 이제 여기서 크롤링을 해보자. 예시로 다음과 같은 코드로 전화번호를 가져오는 크롤링을 해보자.

```
if sentence.startswith('Contact:'):
    phone = sentence[9:])
print(phone)
```

이 코드를 위 코드 마지막에 추가하면
054-785-6648 가 출력된다.

D. 정규식(regular expression) 처리

- ▶ 그런데 위처럼 urllib를 사용해서 어떤 웹의 텍스트를 긁어오거나 텍스트 파일의 텍스트를 긁어오거나 바이너리 파일의 텍스트를 긁어와서 데이터를 추출해보면, 추출이 어렵다는 것을 알 수 있다.
- ▶ 그 이유는 이전에 예시에서 Contact: 가 쓰인 문장에서 Contact: 뒤에 쓰인 문장을 가져오라고 했는데, 어떤 파일에서는 Contact: 라 쓰여 있을지는 모르지만, _Contact: 또는 Contact :라고 쓰여있을 수도 있기 때문이다.
- ▶ 위 코드처럼 Contact: 가 쓰인 문장에서 Contact: 뒤에 쓰인 문장을 가져오라고 쓰면, _Contact: 또는

Contact :라고 쓰여있는 곳 뒤에 있는 전화번호는 들고
오지 못하게 되는 것이다.

▶ 이제 이것을 해결하기 위해 쓰는 것이 정규식 처리
모듈이다.(이후는 정규식 처리 모듈 설명 참조)

5. 추가적인 내용은 <https://docs.python.org/3.7/library/index.html>을 참고하
기.

iv. 파이썬 외부에서 제공되고 있는 유명한 패키지 파일들

1. NumPy: Fundamental package for array computing with python. (사실 Python
에서의 Array는 효율적이지 못하게 구현되어있는거라서 실행속도가 매우 느리다.
그래서 이런 Array의 구조와 연산을 효율적이게 해서 빠른속도로 연산해주는 연
산들이 들어있는 패키지가 NumPy이다.)
2. SciPy: Scientific Library for python.. (과학의 수식 같은 것들을 지원하는 패키
지)
3. Pandas: Powerful data structures for data analysis, time series and statistics. (R
과 같은 통계프로그램처럼 많은 데이터를 받고 분석하고 통계하는 것을 빠르게
할 수 있는 연산들을 제공하는 패키지)
4. Matplotlib: Python plotting package. (그래프를 그리는 것을 도와주는 패키지)
5. Django: A high-level python web framework that encourages rapid
development and clean, pragmatic design. (Python 코드로 웹을 만들 수 있게
해주는 패키지)
6. PyQt: Python bindings for the Qt cross platform application toolkit. (웹페이지
에서 웹의 UI(웹 내의 계산기, 뒤로가기 버튼 등등)를 구현할 수 있게 해주는 패
키지)
7. TensorFlow(Keras): An open source machine learning framework for everyone.
(Python 코드로 머신러닝을 할 수 있게 해주는 패키지)
8. PyTorch: A deep learning package. (Python 코드로 딥러닝을 할 수 있게 해주
는 패키지-TensorFlow를 기반으로 만들어져있다.)
9. 그 외 내용은 <https://pypi.org/>를 참고하기.

X I. 예외처리, 재귀 알고리즘

※ 예외처리까지는 python의 syntax(문법)이고, 재귀 알고리즘부터 프로그래밍 알고리즘을 만드는 방법(알고리즘 방법론)을 배운다.

i . 예외(Error)처리

1. 예외의 종류(Error types)

▶ Syntax Error(문법 예외): Python 문법을 틀리게 썼을 때 나오는 Error이다. 이 Error는 컴퓨터가 기계어로 번역하지 못하는 부분에서 오류가 나는 것이므로 컴퓨터가 어디서 Error가 났는지 찾아서 Error 원인을 띄워준다. 따라서 다른 Error들 중에서 가장 디버깅하기 쉬운 Error이다.

▷ print "test" (python 2 버전에서는 쓸 수 있는 문법이지만 python 3버전에서는 쓸수 없는 문법이다.)

▷ var =

▷ var *=)+!@#

```
File "<stdin>", line 1
var *=)
^
SyntaxError: invalid syntax
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

▷ if True

```
print("True")
```

▶ Runtime Error: 문법적으로는 문제가 없어 python 코드들이 실행되었으나 실행중에 문제(Error in Runtime)가 생겨서 나오는 Error이다.

▷ Zero division Error :0에 관한 잘못된 연산으로 인해 생기는 Error

```
a=0
```

b=10

print(b/a) :문법적으로는 문제가 없으나 어떤 수를 0으로 나누는 것은 불가능하므로 zero division error가 생긴다.

▷ 1 + int(input()):문법적으로는 문제가 없으나 키보드로 받는 문자가 숫자가 아닌 알파벳일 때, int('a')와 같이 되어버려서 Error(Runtime Error)가 생긴다.

▶ Logical Error

▷ Hard to find : Logical Error는 문법적 오류도 없고, Runtime Error도 없어서 실제로는 실행이 잘 된다. 하지만 실행결과가 생각한 대로 나오지 않는 경우가 Logical Error이다. 따라서, 컴퓨터가 찾을 수 있는 Error가 아니여서 찾기가 어렵다.

▶ 그 외의 오류들

Python은 다양한 원인으로 발생하는 Error들을 Exception 칸에 도식화해서 분류 및 정리해놓고 있다. 각각의 Error는 하나의 객체로 봐도 된다.



6

2. Exception Handling (예외처리)

▶ 예외 처리 구문의 형식

▷ try:

#예외가 발생할 수 있는 명령문들의 코드 블록

except 예외이름:

#해당 예외 발생시, 처리하는 코드 블록

else:

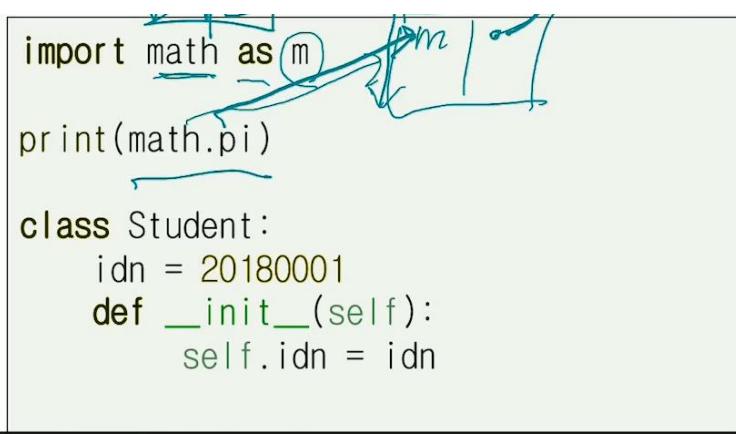
#예외가 발생하지 않을 때 수행되는 코드

finally:

#예외발생 여부와 관계없이 항상 수행되는 코드

3. Exception Handling(예외 처리) 예제: NameError

- ▶ NameError란 호출한 변수이름이 Global namespace에 없을 경우 발생하는 Error로 다음과 같은 상황에서 발생한다.



```
import math as m
print(m.pi)

class Student:
    idn = 20180001
    def __init__(self):
        self.idn = idn
```

```
>>> import math as m
>>> print(m.pi)
3.141592653589793
>>> print(math.pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
>>>
```

- ▶ 그런데, 이런 Error가 내가 쓴 코드가 아니라 모듈에서 발생하였다면 상당히 힘들어진다. 따라서 모듈을 구성할 때는 오류가 발생할 수 있음을 예상하고, 이에 대해 Handling code를 넣어놓는 것이 좋다.

- ▶ NameError Exception Handling Code 추가

```

import math as m

try:
    print(math.pi)
except:
    print("Exception")

class Student:
    idn = 20180001
    def __init__(self):
        self.idn = idn
try:
    st1 = Student()
except NameError:
    print("NameError")

```

▷ 이 코드를 실행하면, try를 실행하다가 오류가 나면 except의 코드블럭을 실행한다. 이렇게 처리해주면, 코드를 받아서 사용하기만 하는 사용자에게 부탁할 오류가 났을 때의 지침 같은 것을 except에 넣는 것도 가능하고, 프로그램의 오류 사항을 임의의 장소(로그)에 기록해두는 코드를 except에 넣는 것도 가능하다.

4. Exception Handling(예외 처리) 예제: IndexError

▶ IndexError란 정의된 Sequence 자료형에서 정의된 인덱스 갯수보다 더 큰 인덱스를 사용할 경우 발생하는 Error이다. 다음과 같은 예시 코드에서 발생하는 Error이다.

```

lst = list(range(10))
lst.remove(3)
print(lst[0])
print(lst[9])

```

▶ IndexError Exception Handling Code 추가

```
lst = list(range(10))
lst.remove(3)

try:
    print(lst[9])
except IndexError:
    print("index error")

try:
    print(lst[10])
except LookupError:
    print("index error")
```

▷ 이 코드를 실행하면, try를 실행하다가 오류가 나면 except의 코드블럭을 실행한다. 이렇게 처리해주면, 코드를 받아서 사용하기만 하는 사용자에게 부탁할 오류가 났을 때의 지침 같은 것을 except에 넣는 것도 가능하고, 프로그램의 오류 사항을 임의의 장소(로그)에 기록해두는 코드를 except에 넣는 것도 가능하다.

▷ 이 코드에서 except LookupError: 라고 있는데, 이 LookupError는 python의 Error(Exception) 분류 상 Exception의 하위 Error에 속하고, IndexError와 KeyError의 상위 Error이다. 따라서 이렇게 적으면, IndexError나 KeyError 발생 시에 except 구문이 실행되게 된다.

▷ 같은 원리로 except Exception:이라 적으면 모든 Error가 Exception의 하위 Error이기 때문에 모든 Error에 대해 except의 코드블럭이 실행되게 된다.

5. Exception Handling(예외 처리) 예제: ValueError

▶ ValueError란 연산의 진행시에 알맞지 않은 parameter를 집어넣은 경우 발생하는 Error이다. 다음 예제에서 lst.index(-1)이 바로 그 예시이다.

```
lst = list(range(10))
lst.remove(3)
```

A hand-drawn diagram of a list represented as a horizontal row of boxes. Inside the boxes are the numbers 1 through 9. Above the first box is the number 0. A bracket above the list indicates the indices from 0 to 9. An arrow points from the text 'lst.remove(3)' to the fourth box containing the number 4, which is crossed out with a red line.

```
print(lst.index(5))
print(lst.index(-1))
```

▷ lst.index(-1)에서 lst의 안의 value 중 -1이라는 값은 없으므로 ValueError가 발생한다.

▶ ValueError Exception Handling Code 추가

```
lst = list(range(10))
lst.remove(3)

try:
    print(lst.index(-1))
except IndexError:
    print("index error")

try:
    print(lst.index(-1))
except ValueError:
    print("value error")
```

▷ 이 코드를 실행하면, try를 실행하다가 오류가 나면 except의 코드블럭을 실행한다. 이렇게 처리해주면, 코드를 받아서 사용하기만 하는 사용자에게 부탁할 오류가 났을 때의 지침 같은 것을 except에 넣는 것도 가능하고, 프로그램의 오류 사항을 임의의 장소(로그)에 기록해두는 코드를 except에 넣는 것도 가능하다.

▷ 이 코드에서 lst.index(-1) 부분은 ValueError가 발생한다. 그

런데 이 try의 짹인 except에 변수로 IndexError를 주었다. 이렇게 되면, try에서 발생한 Error를 except에서 캐치하지 못하고, try에서 발생한 Error가 그대로 터미널에 출력되게 된다.

▷ 그 아래의 코드는 lst.index(-1) 부분에서 ValueError가 발생 한다. 그리고, 여기에서는 try의 짹인 except에 변수로 ValueError가 들어가 있어서 try에서 발생한 Error를 except에서 캐치하고 except의 코드블럭이 실행되게 된다.

6. Exception Handling(예외 처리) 예제: KeyError

▶ KeyError란 dictionary에서 key를 가지고 value를 호출하려는데, 해당 key가 dictionary에서 없는 경우 발생하는 Error이다. 다음 예제에서 count[item] += 1이 바로 그 예시이다.

```
01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         count[item] += 1
05:     return count
06:
07: t = ['a', 'b', 'a', 'c', 'a']
08: print(count_items(t))
```

▶ KeyError Exception Handling Code 추가

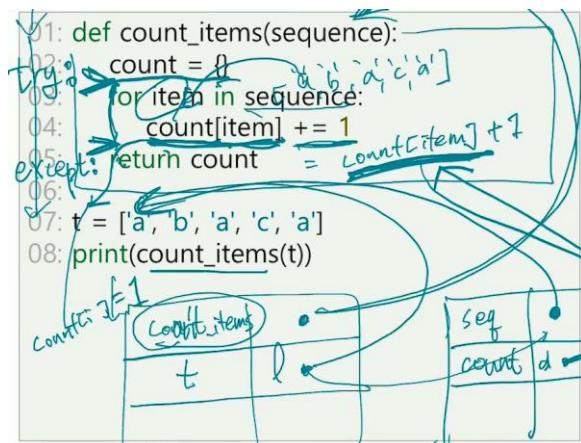
```

01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         count[item] = count.get(item, 0) +
1
05:     return count
06:
07: t = ['a', 'b', 'a', 'c', 'a']
08: print(count_items(t))

```

▷ Exception Handling 전에는 `count[item] += 1`을 `count[item] = count.get(item, 0) + 1`으로 바꾸어 `count`에 `item`이라는 key가 없는 경우 이 자리를 0으로 대신해 채우고 여기에 1을 더하도록 한 값을 `item`이라는 key의 짹 value로 dictionary에 넣도록 하는 것으로 `KeyError`가 발생하지 않도록 수정하는 것으로 `Error`가 발생하지 않게 하였다.

▷ 그런데 이렇게 `get`을 사용하면 `item`에 있던 것이 0이었는지, `key`를 찾을 수 없어서 0이 나왔는지 알수없는 사태가 발생한다는 것을 이전에도 강조하였다. 따라서 이때 `get` 대신 `try, except` 구문을 사용해서 `Error`를 알리고 대신 `get`을 사용하겠다는 통지를 주는 코드를 `except` 코드블럭 내에 집어넣으면 이러한 사태를 예방할 수 있다. 이렇게 사용하는 것이 더 올바른 방법이 아닐까 한다.



- 따라서, 경우에 따라서는 에러가 안나도록

틀어막은 코드보다는 에러가 발생하되 어떻게 알리고 처리하는지 알 수 있도록 try, except 를 사용하는 것이 더 좋을 수 있다.

7. Multiple Exception Handling(다중 예외 처리)

- ▶ try에 있는 코드블럭을 실행할 때, Error가 한가지가 아니고 여러가지가 발생할 수 있는 경우에는 어떻게 대처해야 할까?
- ▶ 이 경우 except Exception: 이라고 쓰면 어떤 에러가 발생하든 같은 except Exception:의 코드블럭을 실행하게 되기 때문에 어떤 Error가 발생한거고 어떻게 디버깅해야 할지 몰라서 난감하다.
- ▶ 이 경우 여러가지 Error가 발생할 수 있는 try의 코드블럭에 대해 여러가지의 except 구문을 사용하면 된다. 여러 개의 except구에 각각 다른 Error 변수를 넣어서 실행하면 각 Error에 대해 서로 다른 대처가 가능한 코드가 만들어진다.

```
base = 100
while True:
    print(base/int(input()))
```



```
base = 100
while True:
    # ValueError
    try:
        print(base/int(input()))
    except ValueError:
```



```
base = 100
while True:
    # ValueError
    try:
        print(base/int(input()))
    except ValueError:
        pass
    except ZeroDivisionError:
        pass
```



```
base = 100
while True:
    # ValueError
    try:
        print(base/int(input()))
    except (ValueError, ZeroDivisionError):
        pass
```

▷ 이렇게 사용하면 try에서 Error가 발생했을 때, try 바로 아래의 except 구문부터 차근차근 보면서 해당하는 Error를 다른 except구에 먼저 들어가 실행되게 된다..

8. Exception Handling 예제 – finally, else 구문 예제

- ▶ try에서 Error가 발생했을 때, 그 Error가 except에서 다뤄지는 Error이면 except를 실행한다. 그런데, try의 코드 전체가 Error 없이 끝났다면 실행되는 것이 else이다. 그리고 except가 실행되던 else가 실행되던 둘 다 실행되지 않던 항상 except와 else가 끝나거나 Error에 대처할 구문이 없는 경우 최종적으로 실행되는 것이 finally이다.

```

def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
divide(2, 1)
print()
divide(2, 0)
print()
divide("2", "1")

```

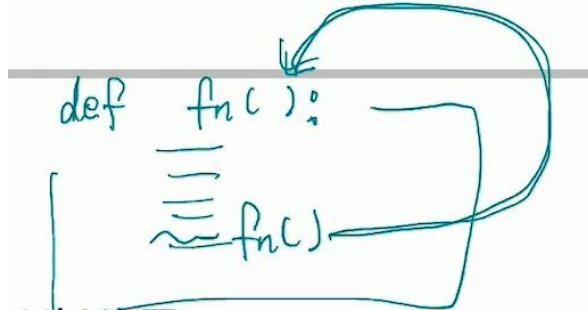
result is 2.0
executing finally clause
division by zero!
executing finally clause
executing finally clause
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s)
for /: 'str' and 'str'

ii. 재귀호출(recursion)

1. 재귀호출(recursion)이란?

▶ 정의

▷ 스스로를 호출하는 함수를 재귀호출이라고 부른다.



▷ 이러한 재귀호출 함수를 이용한 프로그래밍 방법을 재귀호출이라고 부르기도 한다.

▶ 사용하는 목적

▷ 큰 문제를 작은 문제로 나누어 해결할 때 주로 사용한다.

- 이러한 방법을 분할 정복법(divide-and-conquer)이라고 한다.

▷ 복잡한 문제로 비교적 간결하게 표현할 수 있다는 장점이 있다.

▶ 참고

▷ 수학에서 점화식(recurrence formula or recursion formula)에 해당한다.

- ▷ 영화 인셉션에서 꿈속의 꿈속의 꿈과 비슷한 개념이다.

2. 재귀함수의 구성

▶ 재귀함수의 구성요소

- ▷ Base case: 문제의 해결이 매우 간단한 경우 (재귀되는 부분)
- ▷ Recursive case: 작은/간단한 문제의 답을 이용하여, 더 큰/복잡한 문제의 답을 표시하는 경우 (재귀하여 풀고자 하는 전체 부분)
- ▷ 참고: 재귀함수를 사용하는 방법은 다양하나, 일반적으로 복잡한 문제를 recursive해서 점차 작은 문제로 만들어 base case로 만들어내는 구조로 사용하기 때문에, 함수의 앞에 base case, 뒤에 recursive case를 작성한다. base로 계산한 것을 recursive해서 원하는 값을 최종값을 얻어내기 위함이다.

▶ 데이터(정보)의 전달

- ▷ 문제의 크기/데이터를 인자를 사용하여 전달한다.
- ▷ 문제의 답은 반환값/변환가능한 자료구조(list, dict) 등을 이용하여 전달한다.
- ▶ 함수를 호출할 때마다, 그 함수의 매개변수와 지역변수를 위한 공간이 생성된다. → 각각의 함수를 호출할 때 만들어지는 매개변수와 지역 변수의 값들은 독립적이다.

3. 재귀호출의 예시: 팩토리얼

▶ 초기항 : $0! = 1$

▶ 점화관계식: $n! = n \cdot (n-1)! \text{ for } n = 1, 2, 3\dots$

```
def factorial(n):
    # base case
    if n == 0:
        return 1

    # recursive case
    return n * factorial(n - 1)

print(factorial(4))
```

24

▷ 0) recursive code의 알고리즘 진행을 설명하자면 다음과 같다.

```

factorial(3)
def factorial(n): # n = 3
    if n == 0:
        return 1
    return n * factorial(n - 1)

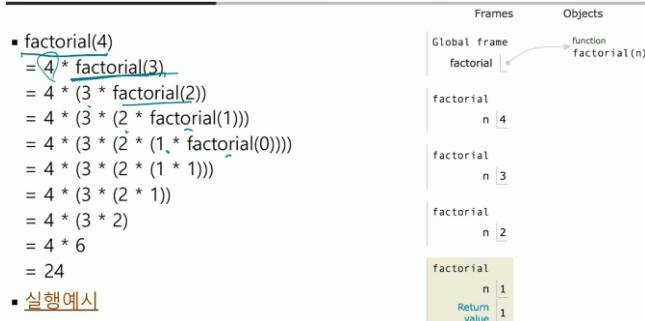
def factorial(n): # n = 2
    if n == 0:
        return 1
    return n * factorial(n - 1)

def factorial(n): # n = 1
    if n == 0:
        return 1
    return n * factorial(n - 1)

def factorial(n): # n = 0
    if n == 0:
        return 1
    return n * factorial(n - 1)

```

예시: 팩토리얼 (함수 호출 다이어그램)



4. 재귀호출의 예시: 피보나치 수열

▶ Base case: $F_n = 1$ for $n = 0, 1$

▶ Recursive case: $F_n = F_{n-1} + F_{n-2}$ for $n = 2, 3, 4 \dots$

```

def fibonacci(n):
    # base case
    if n == 0 or n == 1:
        return 1

    # recursive case
    return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(5))

```

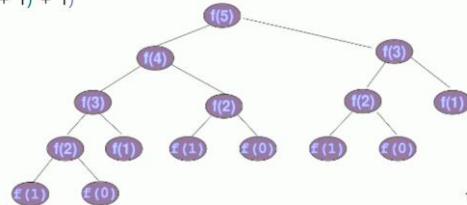
8

▷ 이 recursive code의 알고리즘 진행을 설명하자면 다음과 같다.

- fibonacci()를 f()로 줄여서 나타냈음

$$\begin{aligned}
 f(5) &= f(4) + f(3) \\
 &= (f(3) + f(2)) + (f(2) + f(1)) \\
 &= ((f(2) + f(1)) + (f(1) + f(0))) + ((f(1) + f(0)) + f(1)) \\
 &= (((f(1) + f(0)) + f(1)) + (f(1) + f(0))) + ((f(1) + f(0)) + f(1)) \\
 &= (((1+1)+1)+(1+1))+((1+1)+1) \\
 &= ((2+1)+2)+(2+1) \\
 &= (3+2)+3 \\
 &= 5+3 \\
 &= 8
 \end{aligned}$$

→ 같은 값이 여러 번 계산됨



19

5. 재귀호출의 문제점

- ▶ 피보나치 수열과 같은 경우 이미 했던 연산을 반복해서 또하고 또하고 또하는 알고리즘이 되어버린다. 그 결과 불필요한 연산시간이 늘어나서 실행시간이 늘어난다.
- ▶ recursion을 하면 함수 안에 함수를 반복해서 열기 때문에 namespace로만 메모리를 엄청나게 소비한다. 때문에 간단한 펙토리얼이나 피보나치를 할 경우에도 계산해야 하는 값이 매우 커지면 ex) 10000 namespace가 거의 10000개 가까이 생겨서 값 할당은 많이 하지 않았어도 namespace로 인해 memory full 현상이 생겨서 프로그램 에러가 뜬다.

6. 동적계획법(dynamic programming, memorization)

- ▶ 단순히 재귀호출을 사용하는 방법은 두 가지 문제를 야기했다. 첫 번째는 이미 했던 연산의 반복으로 인한 불필요한 연산시간 추가이고, 두 번째는 함수 안에 함수를 반복해서 열기 때문에 namespace로만 메모리를 엄청나게 소비한다는 것이다. 이것을 해결하기 위해 동적 계획법이라는 것이 등장하였다.
- ▶ 이 동적 계획법(dynamic programming, memorization)이란 계산된 값을 사전에 저장하여 같은 계산을 다시 해야 할 때 사전에서 그 값을 가지고 오는 방식으로 중복계산을 피하는 알고리즘이다.
- ▶ 동적 계획법을 피보나치 수열 예제에 적용하면 코드를 다음과 같이

쓸 수 있다.

```
def fibonacci_dyn(n, fib):
    # base case, using dynamic programming
    if n in fib:
        return fib[n]
    # recursive case
    fib[n] = fibonacci_dyn(n - 1, fib) + fibonacci_dyn(n - 2, fib)
    return fib[n]

fib = {0: 1, 1: 1} # Using
print(fibonacci_dyn(5, fib))
```

8

▶ 이렇게 동적계획법을 사용하면, 이미 한 연산을 반복하지 않아 연산 속도를 빠르게 만드는 것이 가능하며, 이미 한 연산을 반복하기 위해서 열리는 namespace도 없기에 메모리도 보다 적게 사용하면서 연산을 할 수 있게 된다.

7. Helper 함수를 이용한 동적계획법(dynamic programming, memorization)

▶ 그런데 위에서 사용한 동적계획법은 fib을 밖에서 정의해서 넣어주기 때문에 사용자가 함수를 쓸 때, 내부에서 이미 한 연산을 fib에 넣어 준다는 것을 인식하고 이해한 뒤 사용해야 한다.

▶ 때문에 사용자가 원하는 것인 '단지 n을 넣은 것 만으로 피보나치 수열의 n번째 숫자 출력'을 하기 위해서는 함수를 2개를 만들어 기본 fib를 만들어주는 함수, 피보나치 수열을 계산하는 함수의 역할을 부여 해주어야 한다. 이때, 피보나치 수열을 계산하는 함수가 fib를 만들어주는 함수 안에 위치하게 되기 때문에 피보나치 수열을 계산하는 함수는 helper함수가 된다.

▶ 이렇게 재귀함수의 결과를 그때그때 저장하기 위한 방법(동적계획법)으로 helper함수를 사용하는 방법도 있다.

```
def fibonacci_helper(n, fib):
    # base case
    if n in fib:
        return fib[n]
    # recursive case
    fib[n] = fibonacci_helper(n - 1, fib) + fibonacci_helper(n - 2, fib)
    return fib[n]

def fibonacci():
    fib = {0: 1, 1: 1}
    return fibonacci_helper(5, fib)

print(fibonacci())
```

8

8. Default값을 이용한 동적계획법(dynamic programming, memorization)

- ▶ Helper 함수를 이용한 동적계획법(dynamic programming, memorization)도 함수를 두번써서 코드를 작성하는 만큼 코드의 관리 시에 함수 두개를 동시에 관리해야 한다는 불편함이 따른다.
- ▶ 이때, 한 변수를 Default값으로 주고 이 변수가 Default값 일 때, 재귀 함수의 결과를 그때그때 저장하기 위한 초기 dictionary를 만들어주는 코드를 재귀함수 안에 넣어 helper함수와 본래 함수를 한 개로 만드는 방법도 있다.
- ▶ 하지만, 이렇게까지 해도 함수에 함수를 반복해서 여는 작업으로 인해 생기는 namespace의 막대한 공간차지는 해결되지 않는다. recursion의 근본적인 원리이자 문제이기 때문이다.

```

def fibonacci_dyn2(n, fib=None):
    print(f'Calling fibonacci_dyn2() with n={n}, fib={fib}') # track function call
    if not fib:
        fib = {0: 1, 1: 1} # initialize dict fib, when function is invoked
    # base case
    if n in fib:
        return fib[n]
    # recursive case
    fib[n] = fibonacci_dyn2(n - 1, fib) + fibonacci_dyn2(n - 2, fib)
    return fib[n]

print(fibonacci_dyn2(5))

```

9. 재귀(recursion) vs 반복(iteration)

- ▶ 재귀(recursion) 방법은 어떻게 해도 함수에 함수를 반복해서 여는 작업으로 인해 생기는 namespace의 막대한 공간차지는 해결되지 않았다. recursion의 근본적인 원리이자 문제이기 때문이다.
- ▶ 그리고, recursion에 중간값을 저장해서 불필요한 연산을 최소화하는 동적계획법을 사용하지 않으면 반복을 사용하는 것보다 훨씬 시간이 오래 걸리게 된다.
- ▶ 그런데, 근본적으로 재귀(recursion)으로 구현될 수 있는 모든 것은 반복(iteration)문으로 표현이 가능하다. 그리고 반복으로 구현되는 모든 것도 반복으로 바꿀 수 있다. 때문에 상황에 따라 어떤것을 사용해야 할지 분별해서 사용하는 것이 좋다.
- ▶ 일반적인 문제 접근 방법은

▷ 재귀의 경우, 큰 문제를 나누어서 작은 문제로 만들고 이 작은 문제를 풀어나가면서 결과적으로 큰 문제를 해결하는데 사용된다. 즉, Top-down 방식이다. (알고리즘 상으로 설명하자면 Base case를 만날때까지 함수는 계속 recursion된다. 그러다가 Base case를 만난 순간부터 열려있던 함수를 하나하나 종료하면서 결과적으로 recursion case를 해결한다.)

▷ 반복의 경우, 작은 문제를 해결하고 이 작은 문제를 큰 문제로 키우는데 사용된다. 즉, Bottom-up 방식이다.

▶ 각각의 경우 중간 결과 저장은 어떻게 해야하는가.

▷ 재귀 호출을 이용할 경우에는 중간결과가 함수가 호출될 때마다 만들어지는 namespace 혹은 반환값 등에 자동으로 저장된다.

▷ 반복문을 이용할 경우에는, 별도의 변수에 중간 결과를 저장해야 될 경우가 있다.

10. 재귀(recursion)와 반복(iteration)의 비교 예제: 팩토리얼

재귀

- $n!$ 을 $(n-1)!$ 을 이용하여 계산
- $(n-1)!$ 을 $(n-2)!$ 을 이용하여 계산
- ...

```
def factorial(n):  
    # base case  
    if n == 0:  
        return 1  
  
    # recursive case  
    return n * factorial(n - 1)
```

반복문

- $n!$ 을 $1!, 2!, 3!, \dots$ 순으로 계산
- ```
def factorial_loop(n):
 product = 1
 for i in range(1, n + 1):
 product *= i

 return product
```

## 11. 재귀(recursion)와 반복(iteration)의 비교 예제: 피보나치

### 재귀

```
def f_r (num):
 if num == 0 or num == 1:
 return 1
 return f_r(num-1) + f_r(num-2)

import time
start_t = time.time()
print(f_r(10))
print(time.time()-start_t)
```

### 반복문

```
def f_i (num):
 seq_list = []
 p1, p2 = 1, 0
 for i in range(num+1):
 seq_list.append(p1+p2)
 p1, p2 = p2, seq_list[i]
 return seq_list[num]

start_t = time.time()
print(f_i(10))
print(time.time()-start_t)
```

## X II. 검색 알고리즘(Search algorithm)과 정렬 알고리즘(Sort algorithm)

### i. 알고리즘이란

#### 1. 알고리즘에 대한 소개

##### ▶ 알고리즘

- ▷ 특정한 유형의 문제를 해결하기 위한 체계적인 방법
- ▷ 어떠한 형태의 입력에 대해서도 정확한 답을 찾는 방법
- ▷ 예시: 10진수의 사칙연산, 검색, 정렬,....

##### ▶ 알고리즘 복잡도: 알고리즘의 좋고 나쁨을 따지는 보편적인 기준.

- ▷ 시간 복잡도: 얼마나 빨리 실행되는지?
- ▷ 공간 복잡도: 얼마나 많은 추가 공간을 필요로 하나?

### ii. 검색(탐색) 알고리즘

#### 1. 검색(searching): 문제의 정의

▶ 리스트에서 찾고자 하는 원소가 어디에 있는지, 즉 인덱스를 찾는 것을 검색(searching)이라 한다.

▷ 예시:

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| 'a' | 'b' | 'a' | 'c' | 'a' |
|-----|-----|-----|-----|-----|

'a'의 인덱스는 0

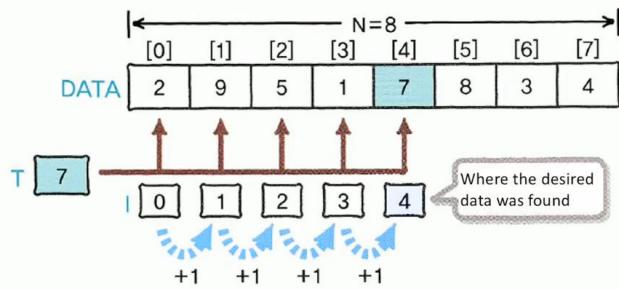
'c'의 인덱스는 3

#### 2. 순차 검색(sequential search)

▶ 원하는 값을 찾을 때까지, 리스트의 맨 앞에서부터 원소를 하나씩 빼서 살펴보는 알고리즘을 순차 검색(sequential search)라고 한다.

▶ 데이터의 구조를 모를 때 가장 간단하게 사용할 수 있는 방법의 검색 알고리즘이며, 매우 구현하기 쉽고 간단하다.

▷ 예시: Find item 7 in item list [2, 9, 5, 1, 7, 8, 3, 4]



▶ 구현된 모습 (range와 len 함수를 이용하여 구현됨.)

```
def sequential_search(sequence, item):
 for index in range(len(sequence)):
 if sequence[index] == item:
 return index
 return None
```

### 3. 이진 검색(binary search)

▶ 그런데, 순차 검색을 사용하면 데이터 개수가 많을 때, 검색 속도가 매우 느리다는 단점이 생긴다. 그래서 이진 검색(binary search)라는 알고리즘이 등장했다.

▶ 우선 이진 검색(binary search)을 사용하기 위해서는 검색할 리스트가 정렬되어 있어야 한다. 즉, 이진 검색(binary search)은 리스트가 정렬되어 있는 경우, 더 효율적인 검색을 하기 위한 방법인 것이다.

▶ 이진 검색의 알고리즘: 리스트가 (오름차순으로) 정렬되어 있을 때, 원하는 원소를 찾기 위해

▷ 처음에는 리스트 전체를 검색을 하는 범위로 설정한다.

▷ 검색을 하는 범위의 가운데 있는 수가 원하는 원소인지 비교한다.

- 검색을 하는 수가 맞으면, 검색을 완료한다.

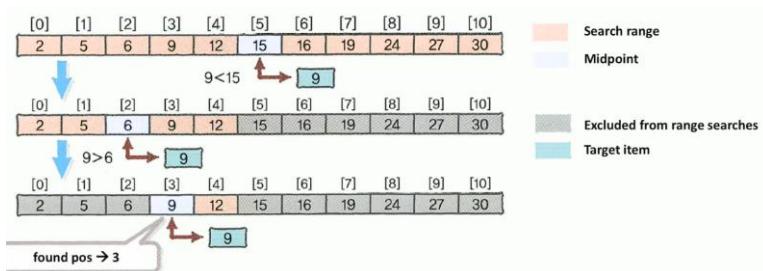
- 가운데 있는 수가 원하는 수보다 크면, 처음부터 가운데 하나 전까지를 검색하는 범위로 설정한 후 반복

한다.

- 가운데 있는 수가 원하는 수보다 작다면, 가운데 하나 다음부터 끝까지를 검색하는 범위로 설정한 후 반복한다.

▶ 이 이진 검색(Binary search) 알고리즘은 우리가 사전을 찾는 방법을 알고리즘화 한 것이다. 사전을 펼쳐서 단어의 순서보다 앞인지 뒤인지 계속 판별하는 것으로 최종적으로 단어를 찾아내는 것을 알고리즘화 한 것이다.

▶ 예시: Find the item 9 in sorted list [2, 5, 6, 9, 12, 15, 16, 19, 24, 27, 30]



▶ 구현된 모습: 간단한 Version.

```
def binary_search(sequence, item):
 first = 0
 last = len(sequence) - 1 # last is INCLUSIVE

 while first <= last:
 mid = (first + last) // 2
 if sequence[mid] == item:
 return mid

 if item < sequence[mid]:
 last = mid - 1
 else:
 first = mid + 1

 return None
```

### iii. 알고리즘 복잡도(Algorithm complexity)

#### 1. 알고리즘 복잡도(Algorithm complexity)

▶ 알고리즘 복잡도 분석(Algorithm complexity analysis)

▷ 특정한 알고리즘의 문제의 크기( $n$ )에 따른 실행시간(시간 복잡도)과 추가적으로 필요한 저장공간(공간 복잡도)을 분석하는 것을 알고리즘 복잡도 분석이라고 한다.

▷ Big-O notation을 이용하여 나타낸다. (주로 고차항만으로 표시한다.)

▷ 최선/평균/최악(best/average/worst case)의 3가지 경우를 모두 분석한다.

▶ 시간 복잡도 분석

▷ 비교, 대입, 기본연산이 단위시간(1)의 실행시간을 가진다고 가정한다.

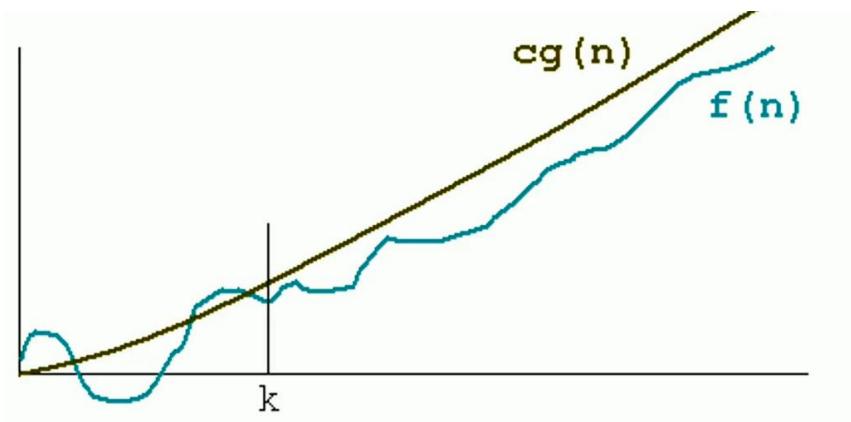
▷ 자료구조(리스트, 튜플, 문자열 등)에 대한 연산은 별도로 어떠한 시간 복잡도를 가지는지 분석해야 한다.

▶ 공간 복잡도

▷ 기본 자료형(int, float, 글자)이 단위 저장 공간을 가진다고 가정한다.

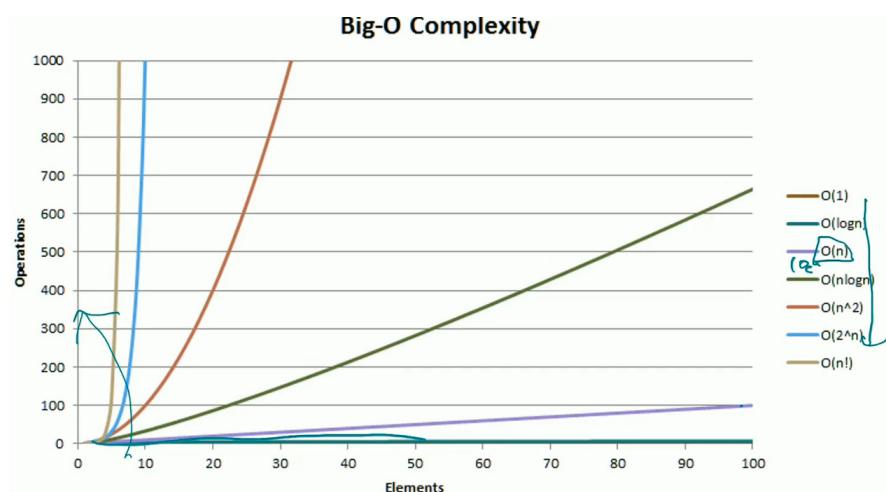
## 2. Big-O notation의 정의

▶ 모든  $n \geq k$ 에 대하여, 어떠한 상수  $c$ 와  $k$ 가  $0 \leq f(n) \leq cg(n)$ 을 만족 할 때,  $f(n) = O(g(n))$ 으로 표시한다. 이 때,  $c$ 와  $k$ 는  $n$ 과 독립적으로 고정된 상수여야 한다.



## 3. 알고리즘 복잡도에 따른 비교

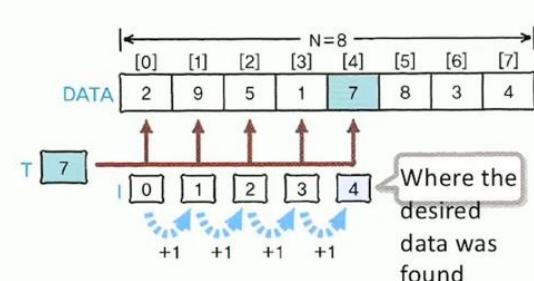
|         | n=1 | n=2 | n=4 | n=8   | n=16  | n=32       |
|---------|-----|-----|-----|-------|-------|------------|
| 1       | 1   | 1   | 1   | 1     | 1     | 1          |
| log n   | 0   | 1   | 2   | 3     | 4     | 5          |
| n       | 1   | 2   | 4   | 8     | 16    | 32         |
| n log n | 0   | 2   | 8   | 24    | 64    | 160        |
| $n^2$   | 1   | 4   | 16  | 64    | 256   | 1024       |
| $n^3$   | 1   | 8   | 64  | 512   | 4096  | 32768      |
| $2^n$   | 2   | 4   | 16  | 256   | 65536 | 4294967296 |
| n!      | 1   | 2   | 24  | 40320 | 20.9T | Don't ask! |



#### 4. 검색 알고리즘 시간 복잡도 분석

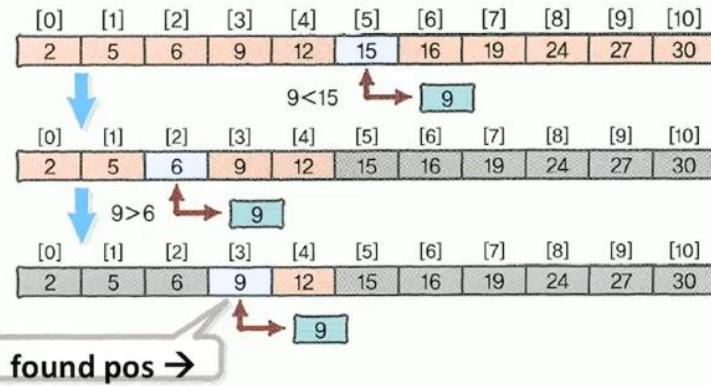
▶ 순차 검색: 처음부터 하나씩 원소를 검색하는 방법.

- Best case:  $O(1)$
- Worst case:  $O(n)$
- Average case:  $O\left(\frac{n}{2}\right) = O(n)$



▶ 이진 검색: 중앙값을 한 번씩 뽑아 찾는 범위를 반씩 줄여서 검색하는 방법.

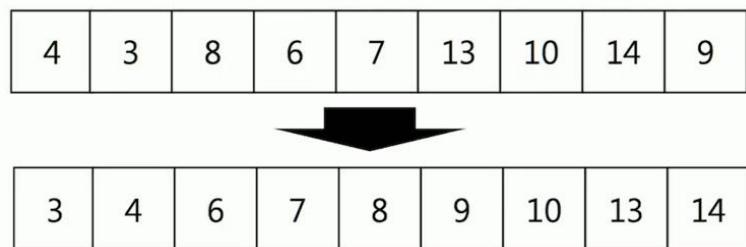
- Best case:  $O(1)$
- Worst case:  $O(\log n)$
- Average case:  $O(\log n)$



#### iv. 정렬 알고리즘

##### 1. 정렬(Sorting): 문제의 정의

- ▶ 정렬이란 아이템들을 일정한 순서에 따라 정렬하는 것을 말한다.
  - ▷ 주로 리스트에 저장되어 있는 아이템들을 정렬하는 것을 말 한다.
  - ▷ 정렬 알고리즘은 마무리 단계에 정렬된 아이템들을 원래 리스트(.sort()) 혹은 별도의 리스트(sorted())에 저장하도록 한다.
  - ▷ 정렬을 할 기준이 제공되어야 한다. (ex. 숫자 크기 순, 알파벳 순)



- ▶ 정렬 알고리즘의 고려 사항: 시간 복잡도 & 공간 복잡도
- ▶ 정렬 알고리즘을 사용하는 함수 예시: .sort(), sorted()

```
t = [4, 3, 8, 6, 7, 13, 10, 14, 9]
t2 = sorted(t)
print(t)
print(t2)
t.sort()
print(t)
```

```
[4, 3, 8, 6, 7, 13, 10, 14, 9]
[3, 4, 6, 7, 8, 9, 10, 13, 14]
[3, 4, 6, 7, 8, 9, 10, 13, 14]
```

## 2. 정렬 알고리즘의 종류 (비교 기반 정렬과 비교 기반이 아닌 정렬 알고리즘으로 크게 나뉜다.)

- ▶ Insertion sort
- ▶ Selection sort
- ▶ Merge sort
- ▶ Quick sort
- ▶ Heap sort
- ▶ Bubble sort
- ▶ Shell sort
- ▶ Bucket sort
- ▶ Radix sort
- ▶ etc.....
- ▶ 참고: [https://en.wikipedia.org/wiki/Sorting\\_algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm)

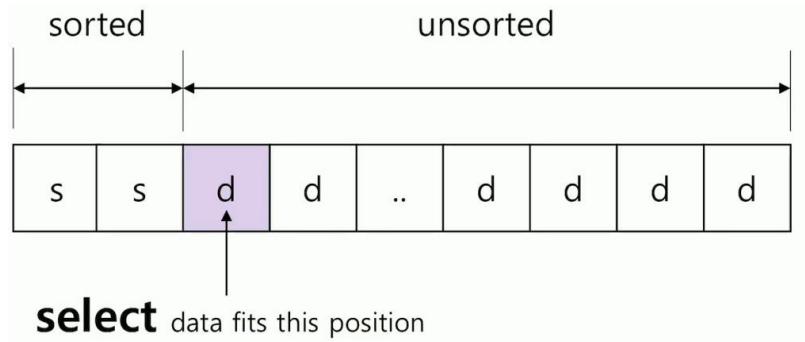
## 3. 선택 정렬(selection sort)

### A. 선택 정렬(selection sort)

- ▶ 모든 아이템들이 정렬이 될 때까지 다음 작업을 반복하도록 하는 알고리즘을 선택 정렬(selection sort)라고 한다.

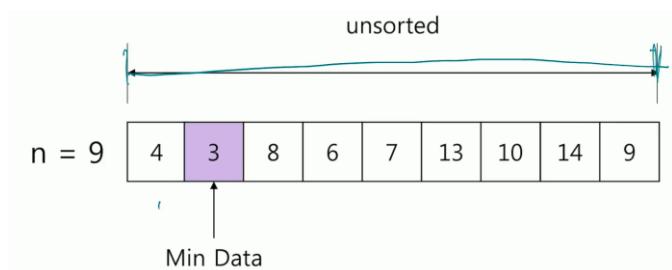
▷ 정렬이 되지 않은 구간에서 가장 작은 수를 찾는다.

▷ 정렬이 되지 않은 구간의 첫번째 숫자와 가장 작은 수의 위치를 바꾼다.

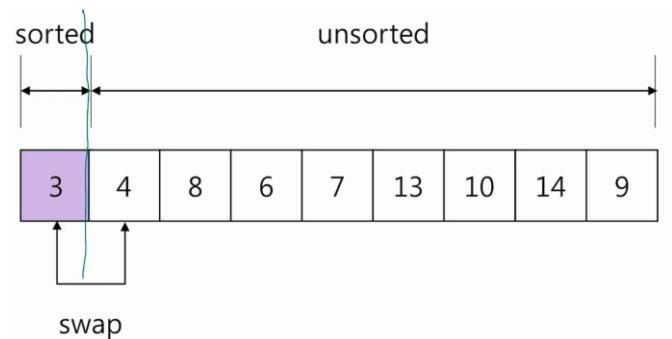


## B. 선택 정렬을 하는 과정 예시

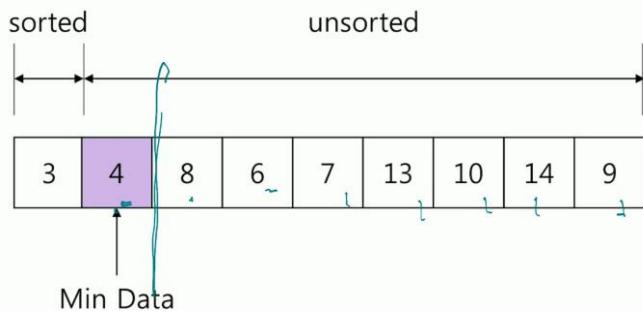
### ▶ 선택 정렬(1/17)



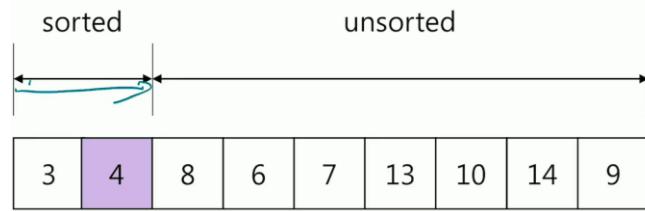
### ▶ 선택 정렬(2/17)



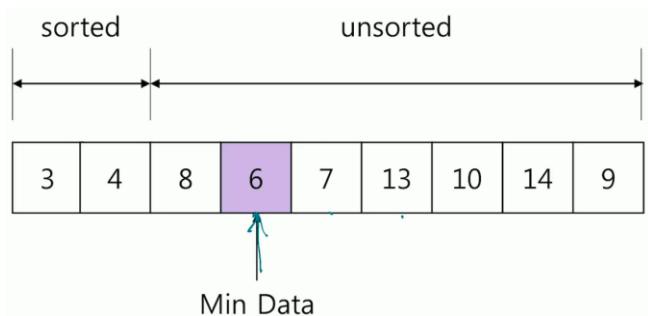
### ▶ 선택 정렬(3/17)



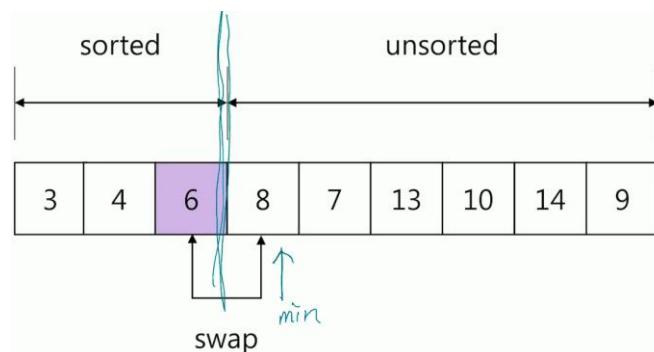
▶ 선택 정렬(4/17)



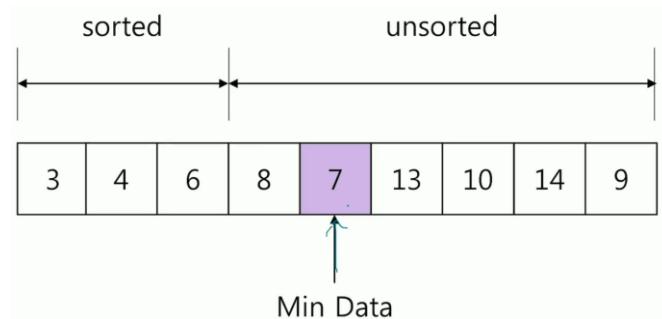
▶ 선택 정렬(5/17)



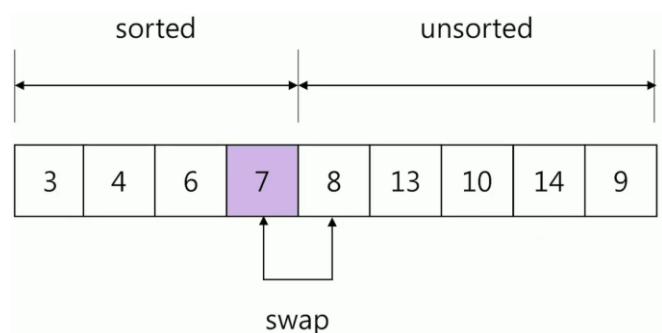
▶ 선택 정렬(6/17)



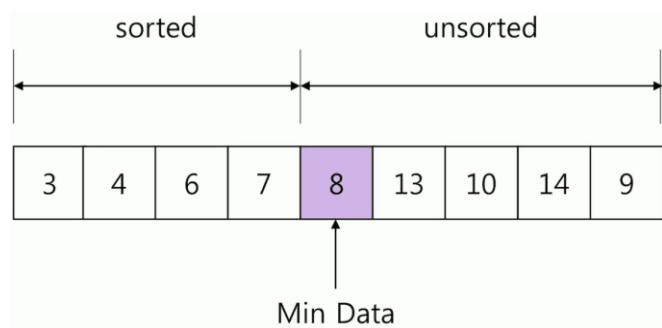
▶ 선택 정렬(7/17)



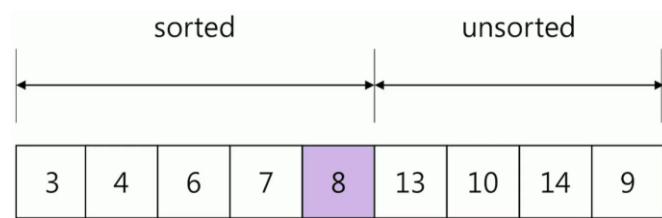
#### ▶ 선택 정렬(8/17)



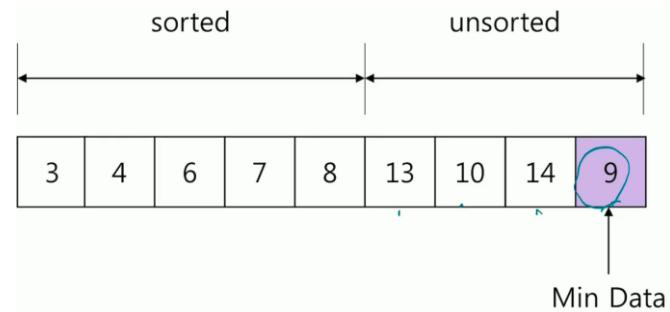
#### ▶ 선택 정렬(9/17)



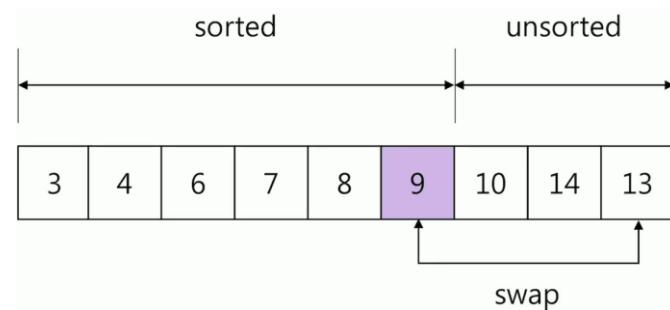
#### ▶ 선택 정렬(10/17)



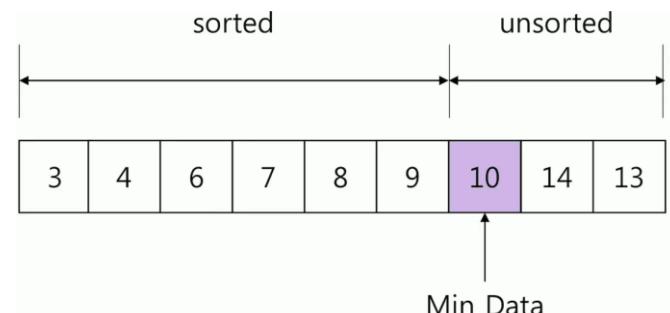
#### ▶ 선택 정렬(11/17)



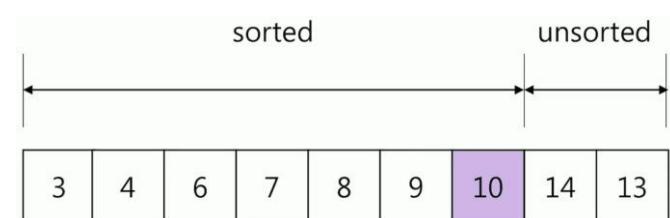
#### ▶ 선택 정렬(12/17)



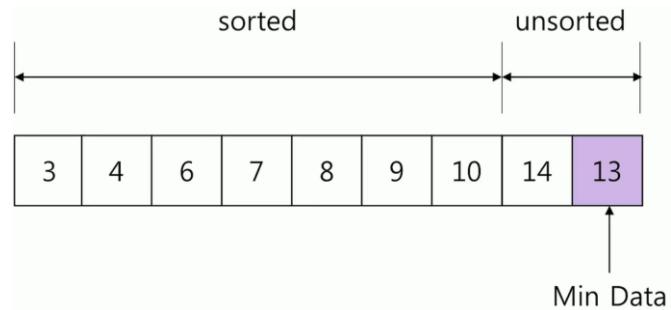
#### ▶ 선택 정렬(13/17)



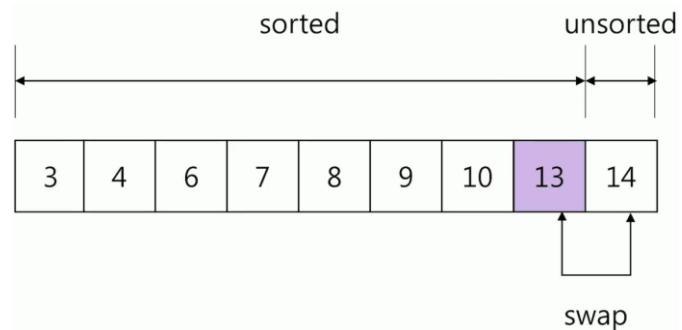
#### ▶ 선택 정렬(14/17)



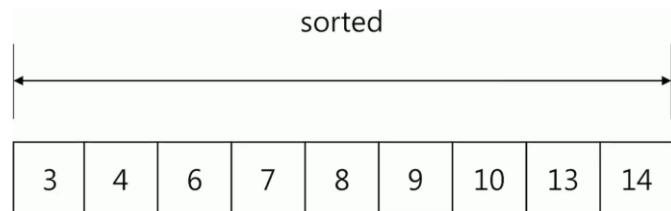
#### ▶ 선택 정렬(15/17)



#### ▶ 선택 정렬(16/17)



#### ▶ 선택 정렬(17/17)



▷ unsorted인 원소의 개수가 1개 이하이면 sorted되었다고 보고 함수를 종료한다.

▷ 이것을 직접 구현하기 위해서 필요한 객체는 현재 위치를 나타내는 포인터 i와 unsorted 중에서 가장 작은 값을 나타내는 포인터 min이다.

### C. 선택 정렬(selection sort)의 구현

▶ 이 알고리즘을 python 코드로 직접 구현하면 다음과 같다.

## selection\_sort()

```

def selection_sort(t):
 for i in range(len(t) - 1):
 # find a location with the minimum item
 min_location = i
 for j in range(i + 1, len(t)):
 if t[min_location] > t[j]:
 min_location = j
 t[i], t[min_location] = t[min_location], t[i] # swap items

t = [4, 3, 8, 6, 7, 13, 10, 14, 9]
selection_sort(t)
print(t)

```

[3, 4, 6, 7, 8, 9, 10, 13, 14]

### D. 선택 정렬의 복잡도 분석

- ▶ 선택 정렬은 리스트에 n개의 아이템이 있을 경우, 다음 단계를 n-1회 반복하는 알고리즘이다.
  - ▷ 정렬되지 않은 부분에서 가장 작은 수를 찾음.
  - ▷ 가장 작은 수와 정렬되지 않은 부분의 첫번째 수를 교환.
- ▶ 시간 복잡도
  - ▷ 비교횟수:  $(n - 1) + (n - 2) + \dots + 1 = \frac{(n-1)n}{2}$   
 $\rightarrow O(n^2)$
- ▶ 공간 복잡도
  - ▷ 교환을 위한 변수 1개  $\rightarrow O(1)$

.

## 4. 병합 정렬(merge sort)

### A. 병합 정렬(merge sort)

- ▶ 분할정복법(divide&conquer)에 의한 정렬 알고리즘이다. 다시 말해, 정렬해야 하는 데이터들을 받아 분할하고 분할한 것에 대해 각각 정렬한 뒤 합쳐서(merge) 전체적으로 정렬된 데이터들을 리턴한다.
- ▶ n개의 아이템이 있는 리스트를 정렬하기 위하여,
  - ▷ n=1: base case(리스트는 이미 정렬되어 있음)

▷  $n > 1$ : merge\_sort()와 merge() 함수를 이용해 정렬

- 왼쪽 절반을 merge\_sort()를 이용해 정렬
- 오른쪽 절반을 merge\_sort()를 이용해 정렬
- 정렬된 데이터를 merge()를 이용해 병합

### B. 병합(merging): merge() 함수에서 일어나는 일

▶ 병합 정렬(merge sort) 알고리즘의 과정 중에서 이미 정렬된 두 개의 리스트를 합쳐 하나의 정렬된 리스트로 만드는 과정.

▶ 두 리스트가 이미 각각 정렬된 상태이기 때문에 각 리스트에서 가장 작은값 찾기를 시전할 필요 없이 그냥 각 리스트 맨 앞의 데이터를 가져오면 각 리스트에서 가장 작은값이 된다. 이렇게 가져온 두가지 데이터를 비교해서 더 작은 값을 새 리스트에 먼저 넣고, 다시 각 리스트에서 맨 앞 데이터 가져오기 및 비교 후 추가를 반복하면 merging함과 동시에 sorting이 된다.

▪ 예

|    |   |    |    |    |
|----|---|----|----|----|
| X: | 3 | 10 | 23 | 54 |
|----|---|----|----|----|

|    |   |   |    |    |
|----|---|---|----|----|
| Y: | 1 | 5 | 25 | 75 |
|----|---|---|----|----|

Result:

|   |   |   |    |    |    |    |    |
|---|---|---|----|----|----|----|----|
| 1 | 3 | 5 | 10 | 23 | 25 | 54 | 75 |
|---|---|---|----|----|----|----|----|

### C. 병합(merging)을 하는 과정 예시

▶ 병합(1/9)

|    |   |    |    |    |
|----|---|----|----|----|
| X: | 3 | 10 | 23 | 54 |
|----|---|----|----|----|

|    |   |   |    |    |
|----|---|---|----|----|
| Y: | 1 | 5 | 25 | 75 |
|----|---|---|----|----|



Result:

|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|



▶ 병합(2/9)

|    |   |    |    |    |    |   |   |    |    |
|----|---|----|----|----|----|---|---|----|----|
| X: | 3 | 10 | 23 | 54 | Y: | 1 | 5 | 25 | 75 |
|    | ↑ |    |    |    |    | ↑ |   |    |    |

|         |   |  |  |  |  |  |  |  |
|---------|---|--|--|--|--|--|--|--|
| Result: | 1 |  |  |  |  |  |  |  |
|         | ↑ |  |  |  |  |  |  |  |

▶ 병합(3/9)

|    |   |    |    |    |    |   |   |    |    |
|----|---|----|----|----|----|---|---|----|----|
| X: | 3 | 10 | 23 | 54 | Y: | 1 | 5 | 25 | 75 |
|    | ↑ |    |    |    |    | ↑ |   |    |    |

|         |   |   |  |  |  |  |  |  |
|---------|---|---|--|--|--|--|--|--|
| Result: | 1 | 3 |  |  |  |  |  |  |
|         | ↑ |   |  |  |  |  |  |  |

▶ 병합(4/9)

|    |   |    |    |    |    |   |   |    |    |
|----|---|----|----|----|----|---|---|----|----|
| X: | 3 | 10 | 23 | 54 | Y: | 1 | 5 | 25 | 75 |
|    | ↑ |    |    |    |    | ↑ |   |    |    |

|         |   |   |   |  |  |  |  |  |
|---------|---|---|---|--|--|--|--|--|
| Result: | 1 | 3 | 5 |  |  |  |  |  |
|         | ↑ |   |   |  |  |  |  |  |

▶ 병합(5/9)

|    |   |    |    |    |    |   |   |    |    |
|----|---|----|----|----|----|---|---|----|----|
| X: | 3 | 10 | 23 | 54 | Y: | 1 | 5 | 25 | 75 |
|    | ↑ |    |    |    |    | ↑ |   |    |    |

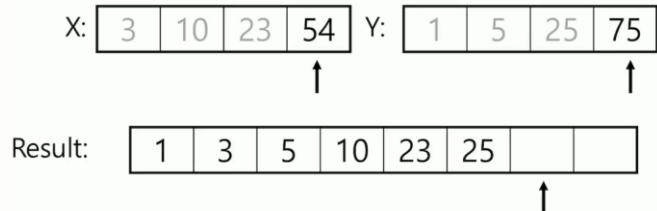
|         |   |   |   |    |  |  |  |  |
|---------|---|---|---|----|--|--|--|--|
| Result: | 1 | 3 | 5 | 10 |  |  |  |  |
|         | ↑ |   |   |    |  |  |  |  |

▶ 병합(6/9)

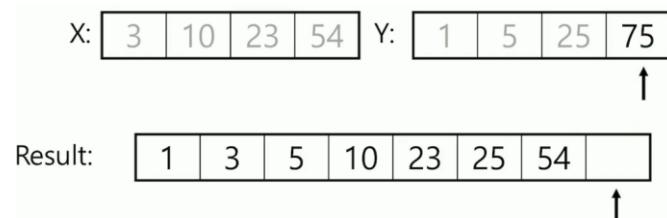
|    |   |    |    |    |    |   |   |    |    |
|----|---|----|----|----|----|---|---|----|----|
| X: | 3 | 10 | 23 | 54 | Y: | 1 | 5 | 25 | 75 |
|    | ↑ |    |    |    |    | ↑ |   |    |    |

|         |   |   |   |    |    |  |  |  |
|---------|---|---|---|----|----|--|--|--|
| Result: | 1 | 3 | 5 | 10 | 23 |  |  |  |
|         | ↑ |   |   |    |    |  |  |  |

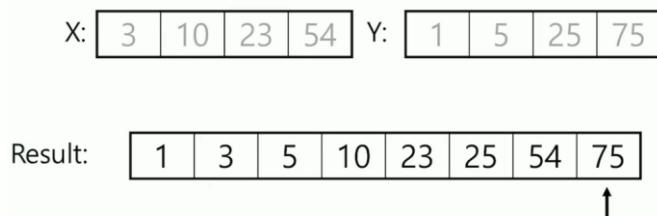
▶ 병합(7/9)



▶ 병합(8/9)

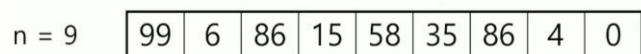


▶ 병합(9/9)

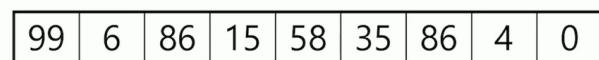


D. merge sort를 하는 예시: Divide를 한 후, merge를 진행하는 sorting 알고리즘. (Divide는 이전 길이의 1/2배인 두 리스트를 만드는 것을 반복하여 맨 처음 주어진 리스트를  $n=1$ 인 리스트까지 작게 만드는 과정이다.)

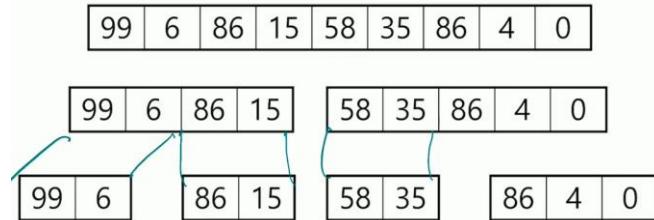
▶ merge sort-Divide(1/10)



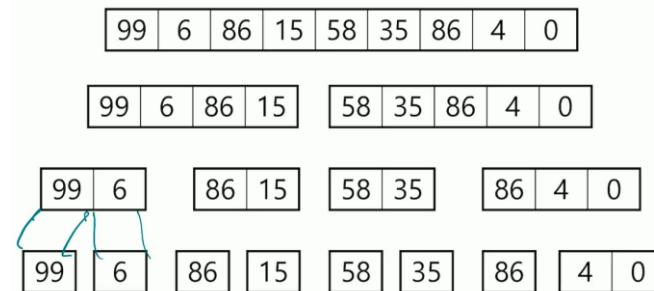
▶ merge sort-Divide (2/10)



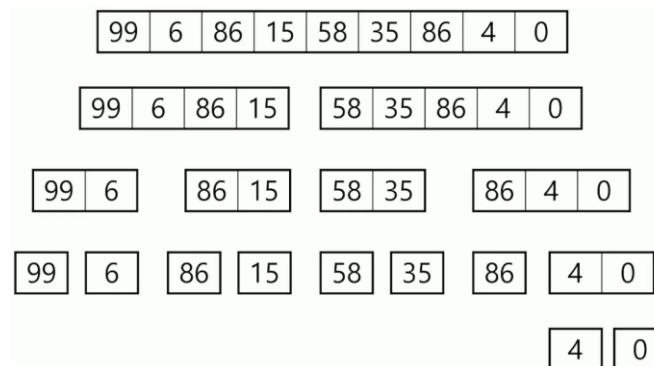
▶ merge sort-Divide (3/10)



▶ merge sort-Divide (4/10)

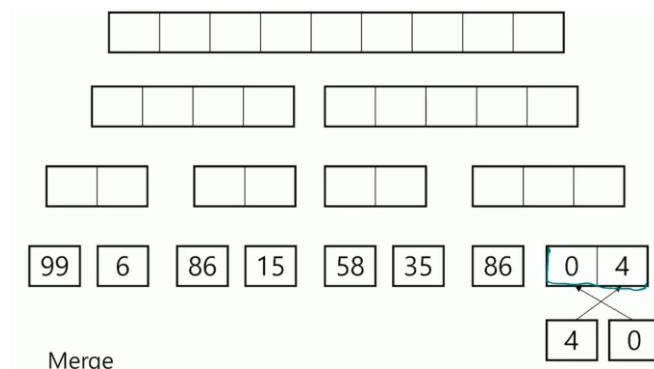


▶ merge sort-Divide (5/10)

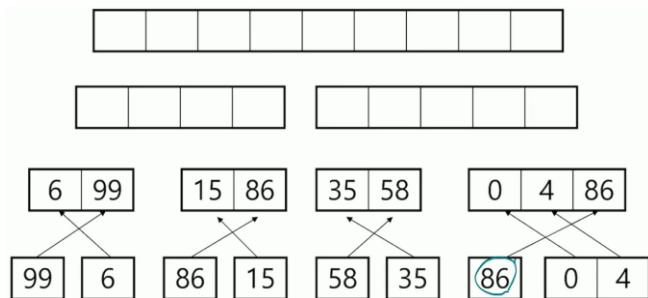


이제 모든 분할이 완료되었으므로, leaf(Divide)로 분리된 제일 마지막 리스트의 형태)부터 merge 알고리즘을 적용해서 merge sort를 해나간다.

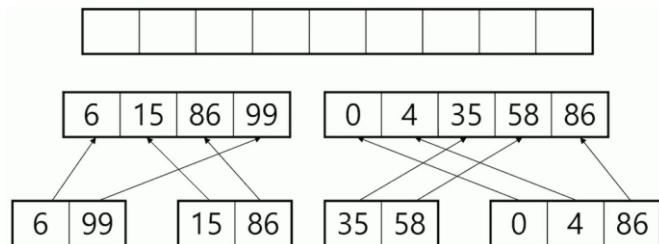
▶ merge sort-merge(6/10)



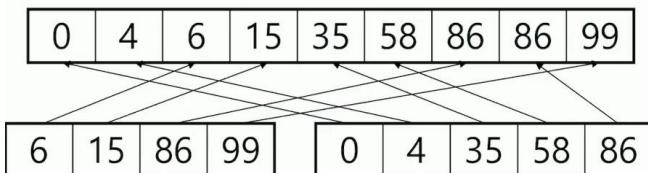
▶ merge sort-merge (7/10)



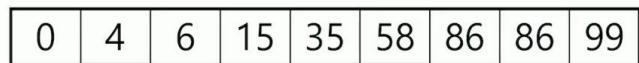
► merge sort-merge (8/10)



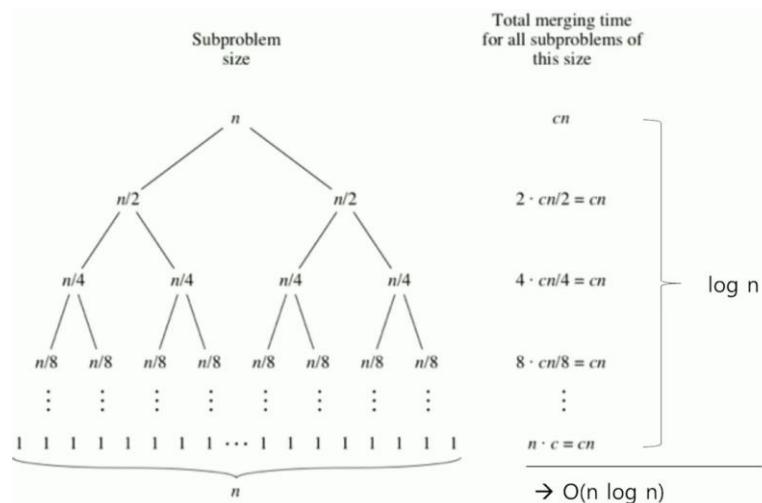
► merge sort-merge (9/10)



► merge sort-merge (10/10)



E. 병합 정렬 복잡도 분석: `merge_sort()`

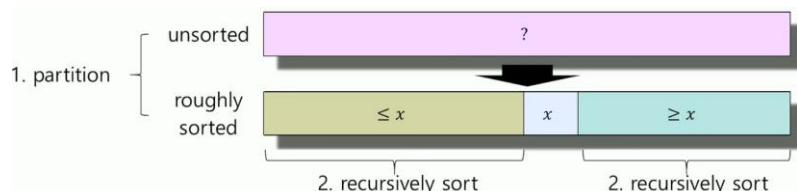


- ▶ Dividing은  $\log(n)$ 번 실행되고, merging은 각각의 step에서 단위시간(1)의 비교를  $n$ 번 사용하므로 merge\_sort는 시간복잡도  $O(n\log(n))$  time이 소요된다.
- ▶ 수학적으로  $O(n\log(n))$  time이 소요되는 sorting 알고리즘이 제일 빠른 알고리즘이 증명되었다. 따라서 이보다 빠른 sorting 알고리즘은 없다.

## 5. 쿠 정렬(quick sort)

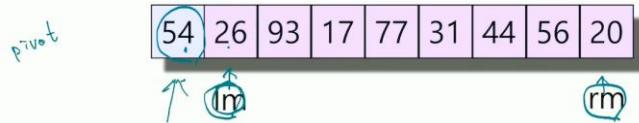
### A. 쿠 정렬(quick sort)

- ▶ 평균적으로 가장 빠른(시간복잡도는 안좋을지 몰라도 실제 실행시간이 가장 빠른) 정렬 알고리즘으로 가장 빈법하게 사용된다.
  - ▷ merge sort, heap sort에 비해 평균 2-3배 정도 빠르다.
  - ▶ 분할정복법을 이용한다. (일반적으로 재귀를 사용하여 구현된다.)
    - ▷ 순서1: partition 알고리즘: pivot value보다 큰 수와 작은 수의 영역으로 분할하고, pivot보다 큰 수는 큰 수의 영역으로, 작은 수는 작은 수의 영역으로 보낸다.
      - pivot value는 일반적으로 가장 앞, 가운데, 끝의 값 중 하나를 사용한다.
    - ▷ 순서2: 순서1이 완료되면 순서1에서 pivot이었던 데이터는 정렬 후에도 제자리를 유지하게 된다. 이제 두개로 분할된 영역에서 각각 또 pivot을 뽑아서 재귀적으로 정렬한다.



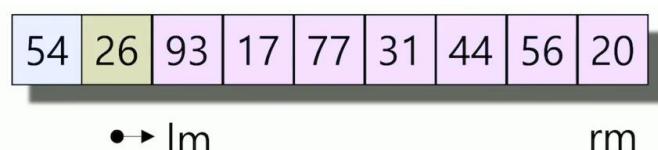
## B. Partition 알고리즘 예시

### ▶ Partition 방법(1/11)



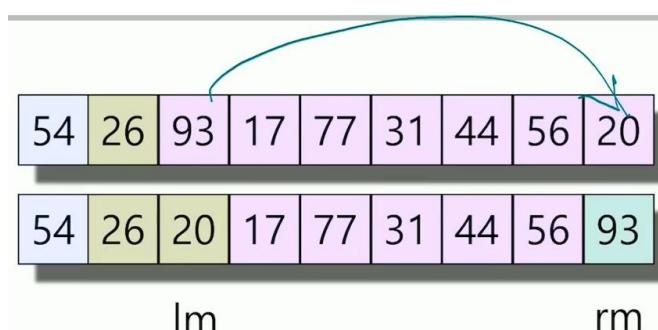
- ▷ 사진처럼 우선 리스트의 0번 데이터를 pivot으로 잡고 pivot 다음 데이터를 lm (left most 포인터), 리스트 가장 마지막 데이터를 rm (right most 포인터)로 가리키게 한다.
- ▷ 먼저 lm이 pivot보다 작은지 큰지 검사한다. lm이 pivot보다 작으면 옮기지 않고 제자리에 둔다.

### ▶ Partition 방법(2/11)



- ▷ lm의 위치를 한칸 우측으로 옮기고 이번에 lm이 가리키는 데이터를 다시 pivot과 비교한다.
- ▷ 이번에 lm이 가리키는 데이터 93은 pivot인 54보다 크다. 그러면 이제 lm은 놔두고 rm을 살펴본다. 현재 rm이 가리키는 데이터는 20으로 pivot보다 작다. 그러면 현재 rm이 가리키는 데이터는 앞으로 보내주어야 한다.

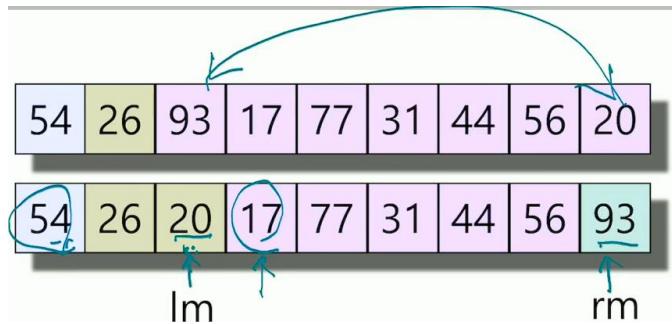
### ▶ Partition 방법(3/11)



- ▷ 앞으로 가야하는 rm와 뒤로 가야하는 lm의 데이터를 서로 swap(바꿔)해준다.

▷ swap되어 lm으로 온 데이터를 검사해서 pivot보다 작은지 확인한다. 여기서는 20이므로 맞으므로 그대로 둔다.

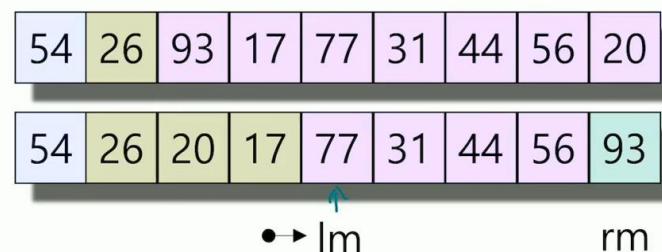
#### ▶ Partition 방법(4/11)



▷ lm의 위치를 한칸 우측으로 옮기고 이번에 lm이 가리키는 데이터를 다시 pivot과 비교한다.

▷ 17이므로 pivot보다 작다. 따라서 그대로 둔다.

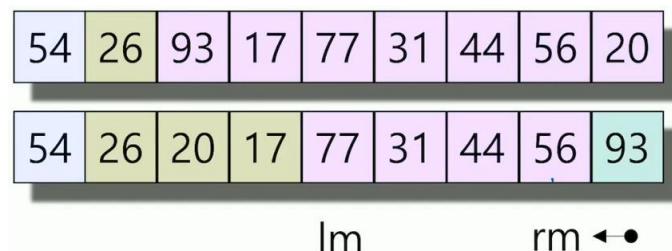
#### ▶ Partition 방법(5/11)



▷ lm의 위치를 한칸 우측으로 옮기고 이번에 lm이 가리키는 데이터를 다시 pivot과 비교한다.

▷ 이번에 lm이 가리키는 데이터 77은 pivot인 54보다 크다. 그러면 이제 lm은 놔두고 rm을 살펴본다. 현재 rm이 가리키는 데이터는 93으로 pivot보다 크다. 그러면 93은 그대로 둔다.

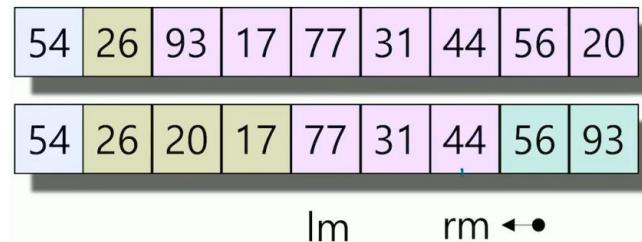
#### ▶ Partition 방법(6/11)



▷ rm의 위치를 한칸 좌측으로 옮기고 이번에 rm이 가리키는 데이터를 다시 pivot과 비교한다.

▷ 이번에 rm이 가리키는 데이터 56은 pivot인 54보다 크다. 그러면 56은 그대로 둔다.

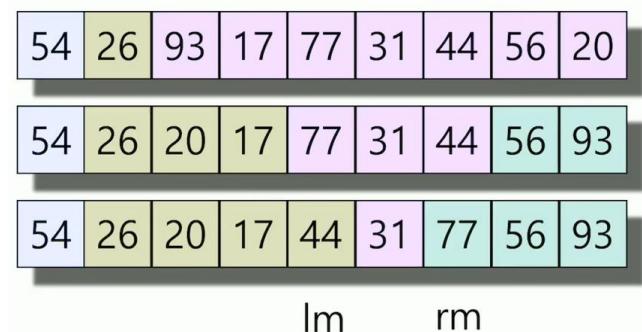
#### ▶ Partition 방법(7/11)



▷ rm의 위치를 한칸 좌측으로 옮기고 이번에 rm이 가리키는 데이터를 다시 pivot과 비교한다.

▷ 이번에 rm이 가리키는 데이터 44는 pivot인 54보다 작다. 그러면 현재 rm이 가리키는 데이터는 앞으로 보내주어야 한다.

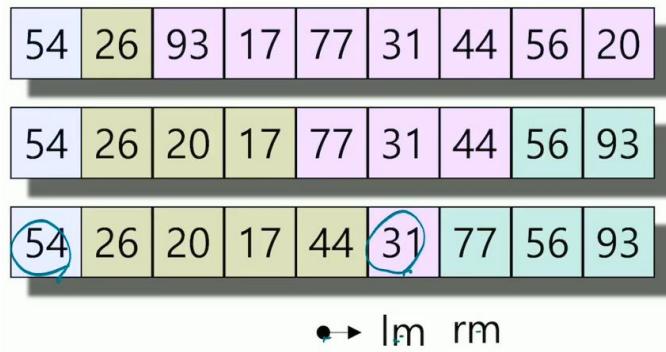
#### ▶ Partition 방법(8/11)



▷ 앞으로 가야하는 rm와 뒤로 가야하는 lm의 데이터를 서로 swap(바꿔)해준다.

▷ swap되어 lm으로 온 데이터를 검사해서 pivot보다 작은지 확인한다. 여기서는 44이므로 맞으므로 그대로 둔다.

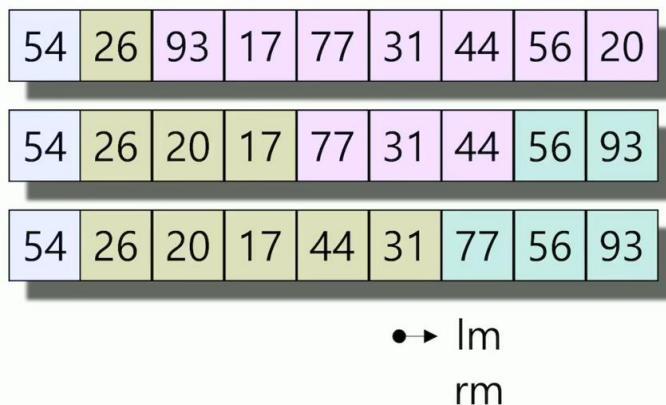
#### ▶ Partition 방법(9/11)



▷  $Im$ 의 위치를 한칸 우측으로 옮기고 이번에  $Im$ 이 가리키는 데이터를 다시 pivot과 비교한다.

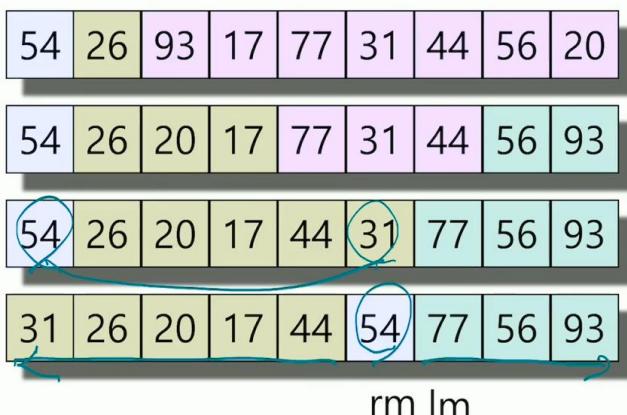
▷ 31이므로 pivot보다 작다. 따라서 그대로 둔다.

#### ▶ Partition 방법(10/11)



▷  $Im$ 의 위치를 한칸 우측으로 옮긴다. 그런데 옮기다가  $Im$ 과  $rm$ 이 같은 데이터를 가리키고 있게 되면,  $Im$ 을 옮기기 직전 데이터와 pivot데이터의 자리를 swap 해준다.

#### ▶ Partition 방법(11/11)



## 6. 정렬 알고리즘 비교

|        | Selection sort | Merge sort    | Quick sort                                |
|--------|----------------|---------------|-------------------------------------------|
| 시간 복잡도 | $O(n^2)$       | $O(n \log n)$ | Average: $O(n \log n)$<br>Worst: $O(n^2)$ |
| 공간 복잡도 | $O(1)$         | $O(n)$        | $O(1)$                                    |

- ▶ Quick sort는 pivot이 중앙값으로 잡힐 때(Best, Average), Merge sort처럼 작용해서  $O(n \log n)$  time이 걸리고, pivot이 양 side 끝값으로 잡힐 때(Worst), Selection sort처럼 작용해서  $O(n^2)$  time이 걸린다.

## 7. 읽을 거리

- 검색 알고리즘: [https://en.wikipedia.org/wiki/Search\\_algorithm](https://en.wikipedia.org/wiki/Search_algorithm)
- 알고리즘 복잡도 분석: [https://en.wikipedia.org/wiki/Analysis\\_of\\_algorithms](https://en.wikipedia.org/wiki/Analysis_of_algorithms)
- 정렬 알고리즘
  - [https://ko.wikipedia.org/wiki/정렬\\_알고리즘](https://ko.wikipedia.org/wiki/정렬_알고리즘)
  - <https://namu.wiki/w/%EB%8B%A8%EB%8A%A0%EB%8D%BC%ED%8A%8C>
- 알고리즘 실행 예시
  - 애니메이션: <http://www.sorting-algorithms.com>
  - 시각화: <http://sortvis.org/visualisations.html>
  - 청각화: <http://flowingdata.com/2010/09/01/what-different-sorting-algorithms-sound-like/>
  - 댄스화 (AlgoRhythmic):  
<https://www.youtube.com/channel/UCIqiLefbVHsOAXDAxQJH7Xw>
- 알고리즘 복잡도 분석
  - [https://en.wikipedia.org/wiki/Analysis\\_of\\_algorithms](https://en.wikipedia.org/wiki/Analysis_of_algorithms)

- Done -