

# 소프트웨어와 문제해결

Dr. Young-Woo Kwon

# 수업 개요

- 자료구조
  - 배열, 리스트, 스택, 큐, 사전
- 탐색(찾기)
  - 선형 탐색
  - 이진 탐색
  - 해시 탐색
- 실습

# 자료구조

- 자료 구조(data structure)
  - 사람이 사물을 정리하는 것과 같이 프로그램에서도 자료를 여러 가지 구조에 따라 정리하는 것
  - 일상생활에서의 정리:
    - 하루에 해야 할 일들을 순차적으로 수첩에 기록하기
    - 책상에 책을 쌓아 놓는 것
    - 상점에서 물건을 구입하기 위해 줄을 서는 것

# 자료 구조

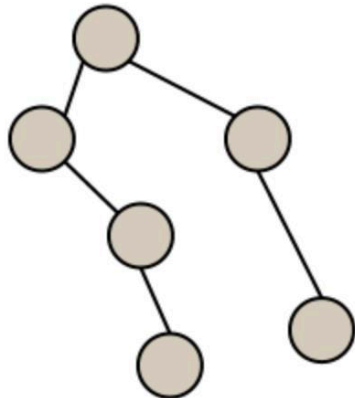
- 스택



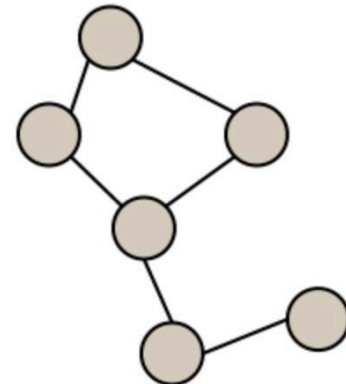
- 큐



- 트리



- 그래프



# 자료구조

일상생활 예	자료구조
물건을 쌓아놓는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색 구조
지도	그래프
조직도	트리

# 배열

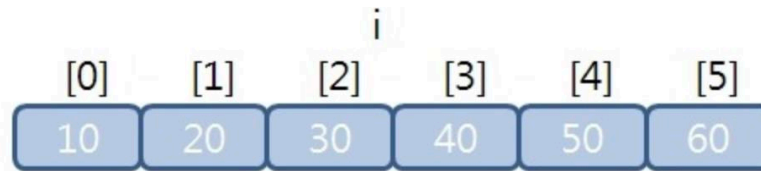
- 같은 종류의 모임
- 기초적이고 가장 중요한 자료 구조
- 일상생활의 예
  - 학년, 반
  - 아파트 동, 호수



# 배열

- 1차원 배열

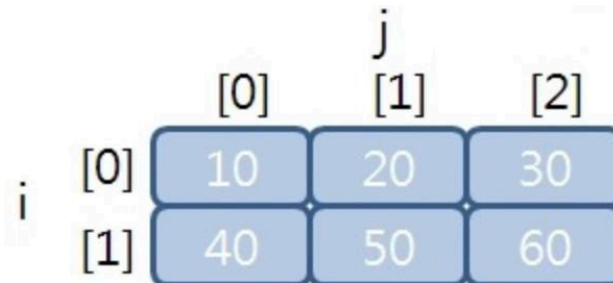
- 하나의 열을 사용하여 선형적으로 요소에 접근



1차원 배열 구조 표현

- 2차원 배열

- 여러 개의 행과 열을 사용하여 배열 요소에 접근



2차원 배열 구조 표현

# 리스트

- 순서가 있는 항목들의 모임, 즉 “목록”
- 리스트(list)의 예:
  - 숫자들:  $\{1, 2, 3, 4, 5, \dots, 100\}$
  - 요일들:  $\{\text{일요일}, \text{월요일}, \dots, \text{토요일}\}$
  - 매 월:  $\{1\text{월}, 2\text{월}, 3\text{월}, \dots, 12\text{월}\}$
  - 카드 한 벌의 값:  $\{\text{Ace}, 2, 3, \dots, \text{King}\}$



# 리스트

- 리스트(list)의 예: 최대 흥행 영화 리스트

순위	영화 제목	인덱스
1	아바타(Avatar)	0
2	타이타닉(Titanic)	1
3	스타워즈(Star Wars: The Force Awakens)	2
4	주라기 월드(Jurassic World)	3
5	어벤저스(Marvel's The Avengers)	4

# 리스트

- 인덱스(index)
  - 리스트의 항목을 식별하기 위해 사용하는 숫자
    - '0'부터 시작한다
- movie\_list에 영화를 저장하는 의사 코드

2011

2011년

```
movie_list[0] ← "아바타"  
movie_list[1] ← "타이타닉"  
movie_list[2] ← "스타워즈"  
movie_list[3] ← "주라기 월드"  
movie_list[4] ← "어벤저스"
```

의사코드 명칭 → 명칭 추천

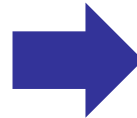
# 리스트

- 리스트를 초기화 하려면 다음의 문장을 사용

```
movie_list = {"아바타", "타이타닉", "스타워즈", "주라기 월드", "어벤저스"}
```

- 리스트 movie\_list에 저장된 영화를 꺼내어 출력하는 의사 코드 사용

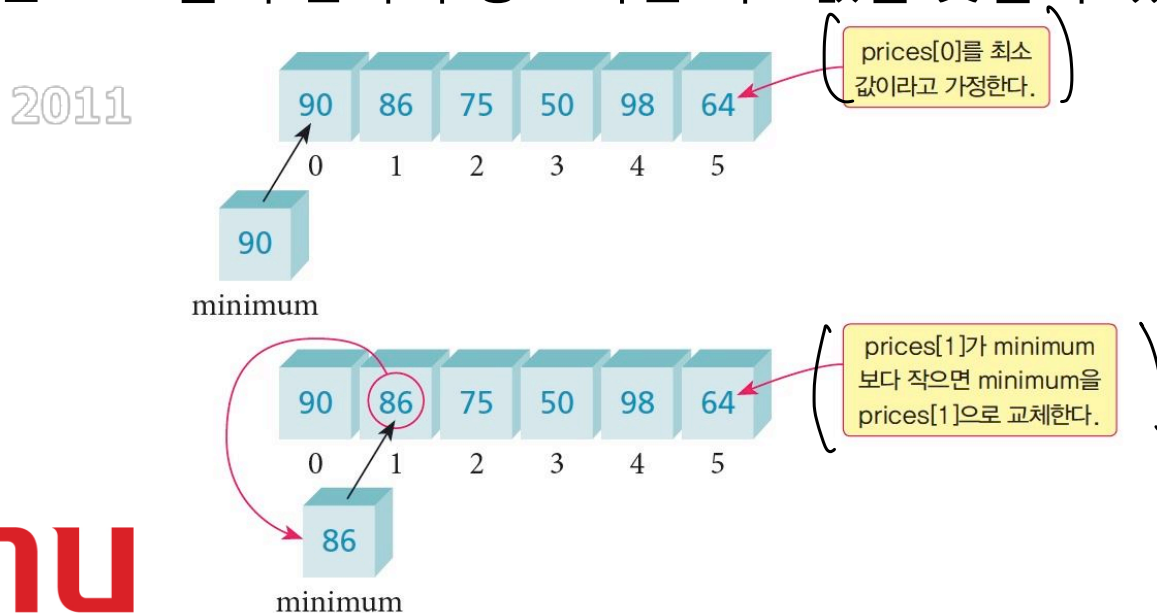
```
print movie_list[0]  
print movie_list[1]  
print movie_list[2]  
print movie_list[3]  
print movie_list[4]
```



```
i ← 0  
while i < 5  
    print movie_list[i]  
    i ← i + 1  
endwhile
```

# 리스트 사용 예

- 리스트에 저장된 값들의 최소값 구하기
  - 리스트의 첫 번째 요소를 최소값으로 가정
  - 리스트의 두 번째 요소부터 마지막 요소까지 이 최소값과 비교
  - 만약 어떤 요소가 현재의 최소값보다 작다면 이것을 새로운 최소값으로 변경
  - 모든 요소들의 검사가 종료되면 최소값을 찾을 수 있다



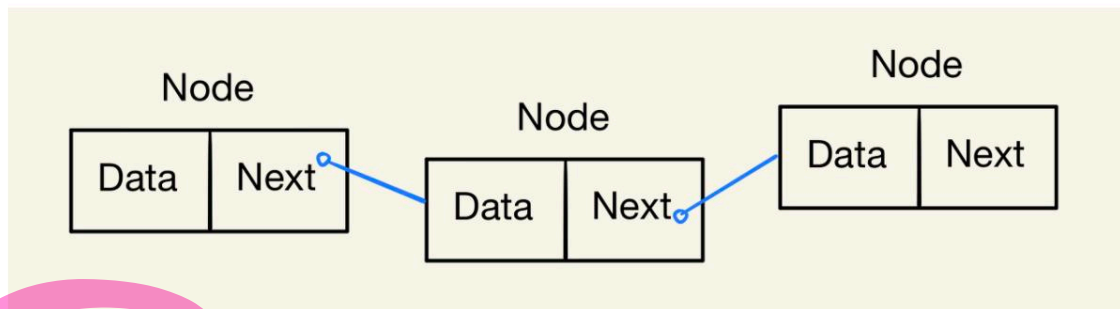
# 리스트

- 특정값 탐색: 가장 흔한 예->인터넷에서 정보검색
  - 예) 리스트에 숫자들이 저장되어 있고, 이 중에서 하나의 숫자를 찾을 때...



# 연결 리스트

- 연결 리스트
  - 자료들이 연결된 것



- 끝말 잇기

• 기차 → 차고 → 고장 → 장난감 → 감꽃 → 꽃잎



# 연결 리스트

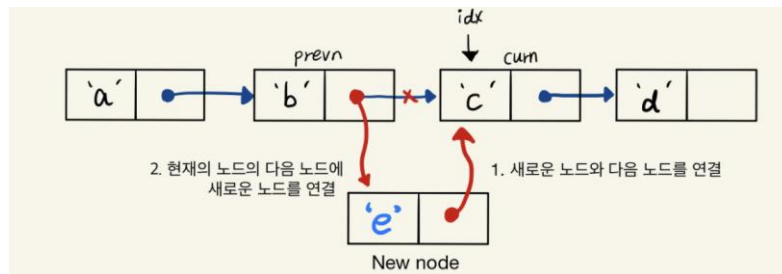


## • 장점

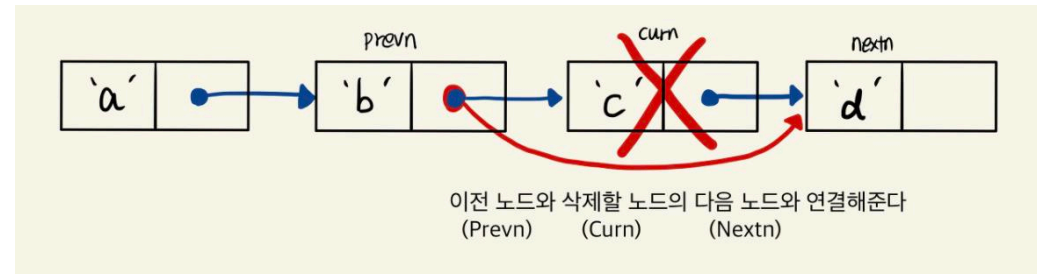
- 길이를 동적으로 조절 가능
- 데이터의 삽입과 삭제가 쉬움

## • 단점

- 임의의 요소에 바로 접근할 수 없음 → 리스트는 가변한데 말이지!
- 추가 공간 필요
- 리스트의 뒤에서부터 탐색하기 어려움



삽입

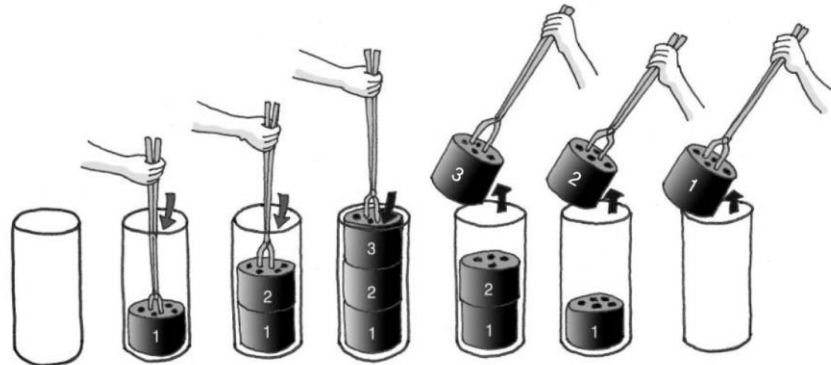


삭제



# 스택

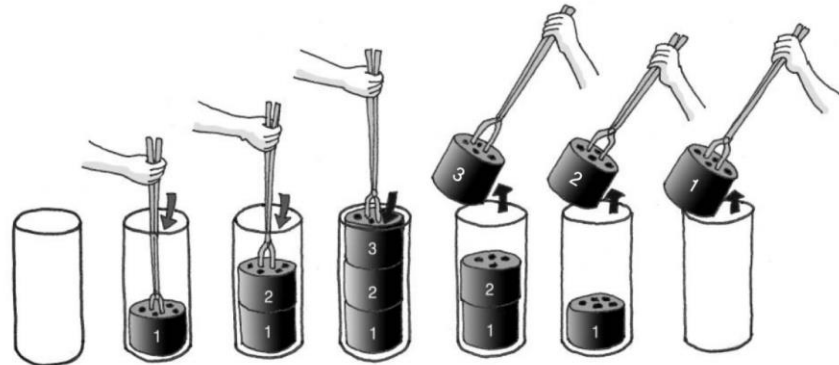
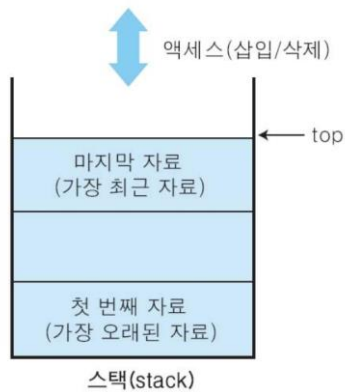
- 자료를 차곡차곡 쌓아 올린 형태의 자료 구조
  - 예: 연탄 아궁이





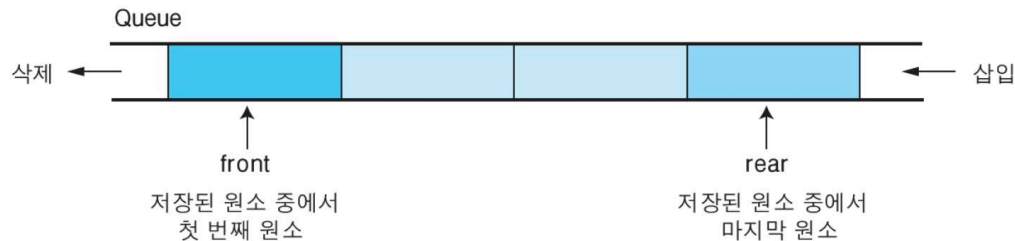
# 스택

- 자료를 차곡차곡 쌓아 올린 형태의 자료 구조로 자료의 삽입/삭제가 스택의 한 위치에서만 일어남
  - 예: 연탄 아궁이



# 큐

- 자료의 삽입과 삭제가 서로 다른 위치로  
제한된 자료구조



# 사전 (Dictionary) → 자료구조의 원형

- 사전 (**dictionary**)은 속성/값 (property/value)의 쌍으로 이루어진 자료구조
- 파이썬에서는 키/값 (key/value)으로 이루어짐

City	Temperature
“Daegu, S Korea”	17

The diagram illustrates the structure of a dictionary. It shows the code `temps = { "City" : "Daegu, S Korea", "Temperature" : 17 }`. Red annotations are used to identify parts of the dictionary:

- Two red curly braces above the code group each pair into a "key-value pair".
- A red arrow points from the word "key" below to the string `"City"`.
- A red arrow points from the word "value" below to the string `"Daegu, S Korea"`.



# 사전 (Dictionary)

```
temps = { " City" : "Daegu, S Korea" , "Temperature" : 17 }
```

- 추출

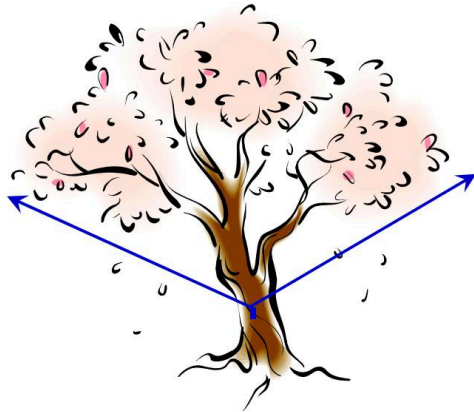
```
cityTemp = temps["Temperature"]  
cityName = temps["City"]
```

- 갱신

```
temps["Temperature"] = 20
```

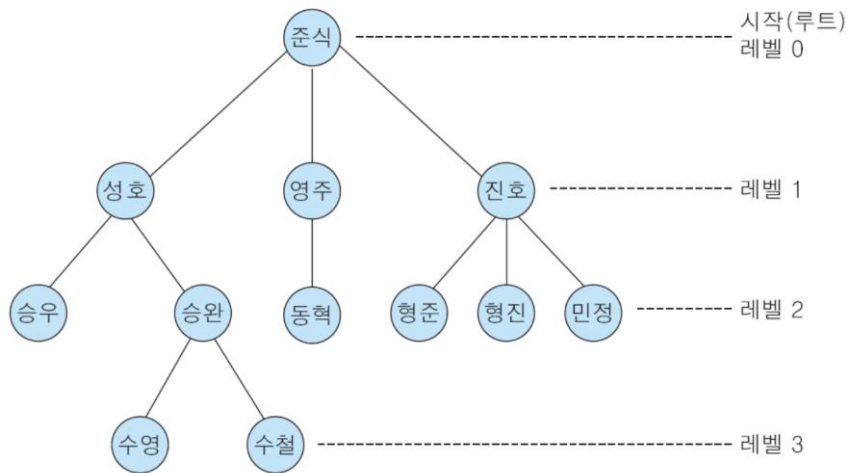
# 트리

- 원소들 간에 1:N 관계를 가지는 비선형 자료구조
- 원소들 간에 계층관계를 가지는 계층형 자료구조
- 상위 원소에서 하위 원소로 내려가면서 확장되는 나무 모양의 구조

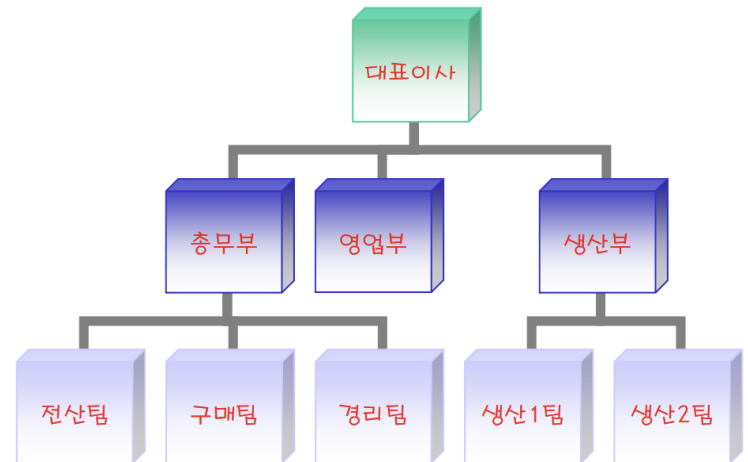


# 트리

- 트리의 예: 가계도, 조직도



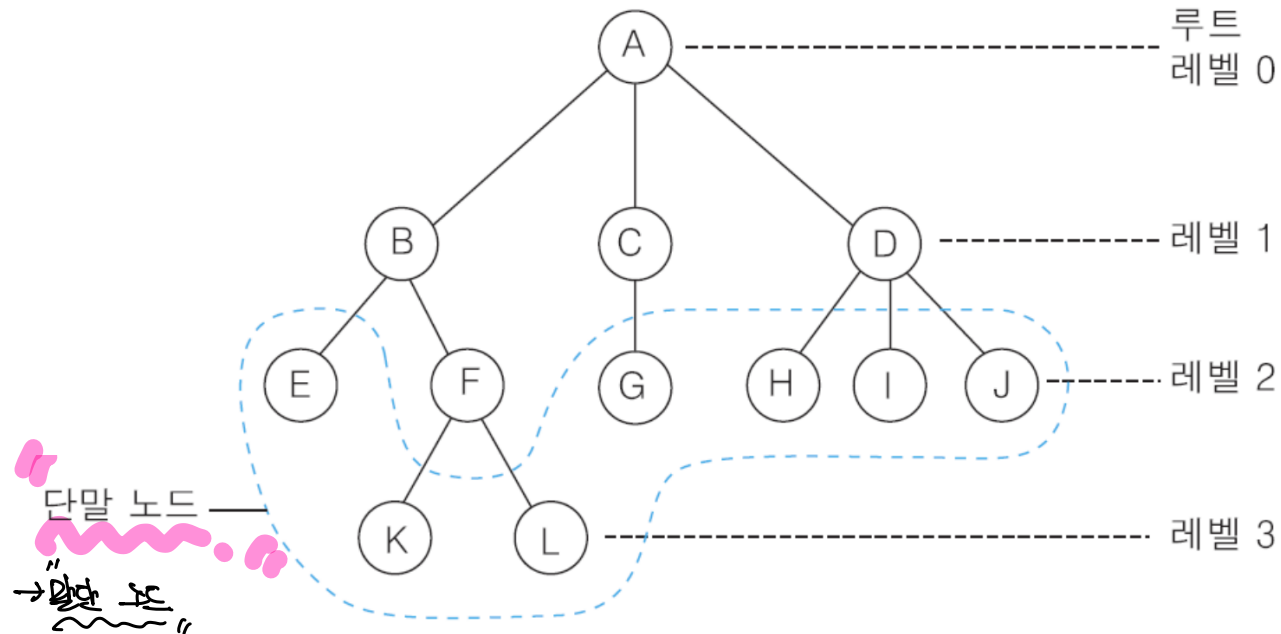
→ 연산하는 과정



# 트리

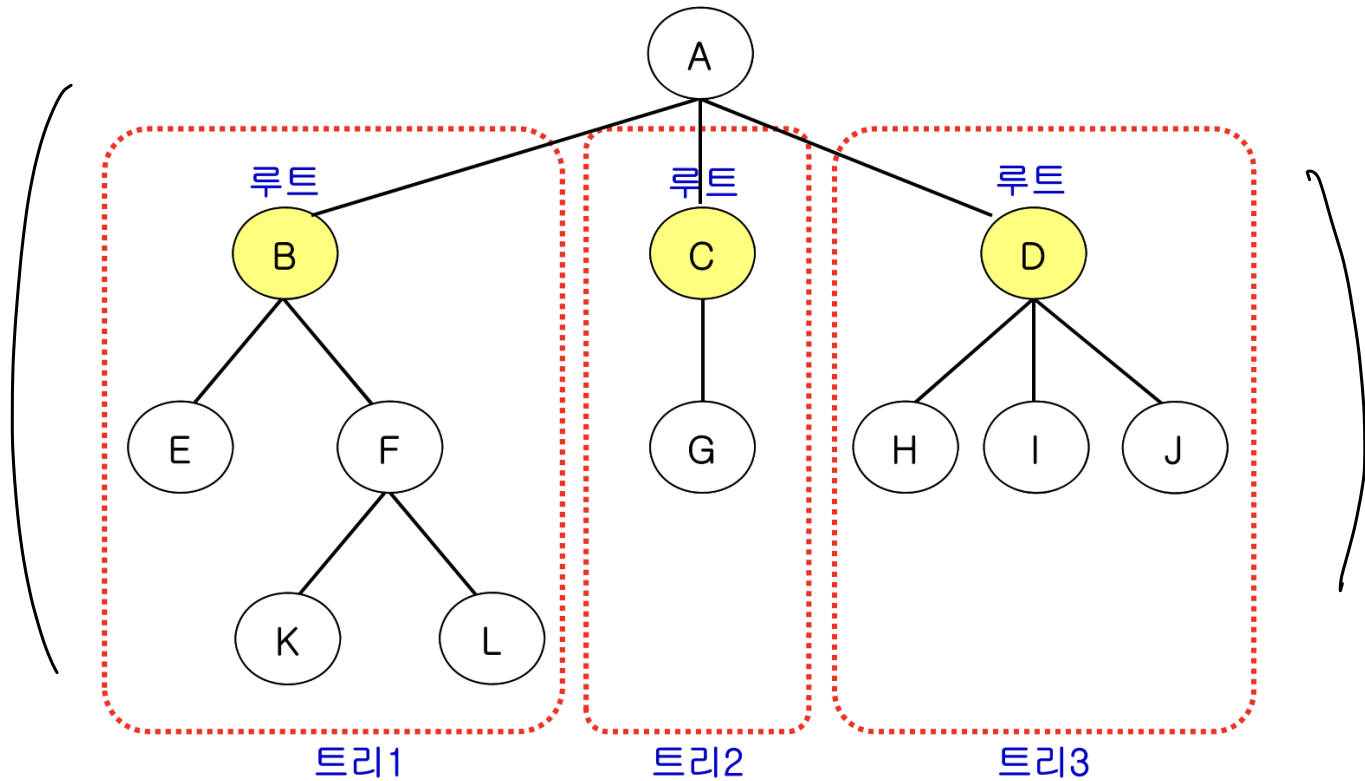
- 트리의 용어

— . . .



# 트리

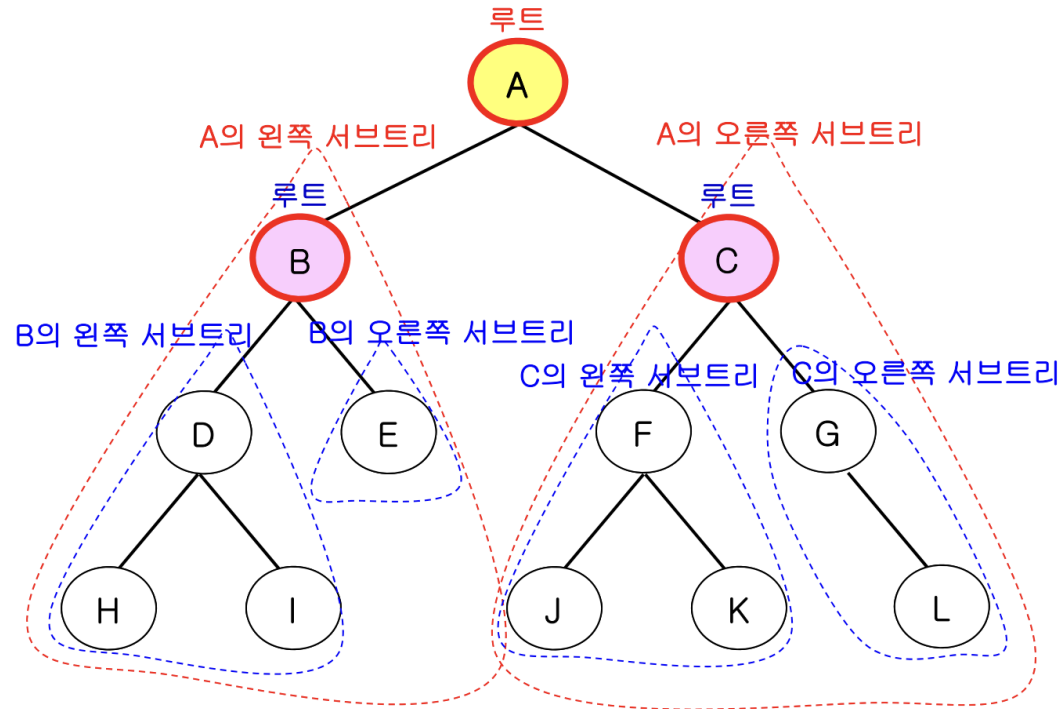
- Forest: 서브트리의 집합





# 트리

- 이진 트리 (Binary tree): 모든 노드는 왼쪽 자식과 오른쪽 자식 노드만을 가짐

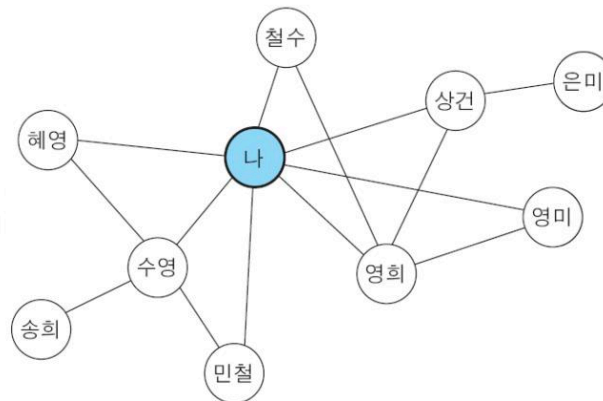
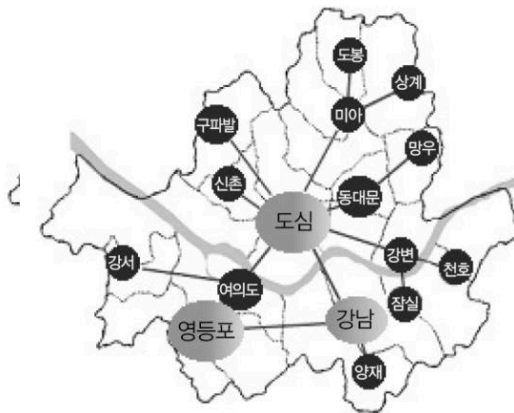


# 그래프

- 선형 자료구조나 트리 자료구조로 표현하기 어려운 N:M 관계를 표현하기 위한 자료구조
- 그래프의 예

→ BFS, DFS

이렇게 할때 편하겠군 ..



# 탐색

# 선형 탐색

- 숫자들의 리스트가 있다고 가정
- 이 숫자들은 정렬되어 있을 수도 있으며 아닐 수도 있음
- 선형 탐색은 리스트에서 순차적으로 비교하면서 숫자를 찾는 방법

→ 순차적으로 검색

3	5	2	1	0	9	7	8	6	4
---	---	---	---	---	---	---	---	---	---

# 이진 탐색 (Binary Search)

- 숫자들의 리스트가 있다고 가정
- 이 숫자들은 <sup>→ 선형 탐색의 대안점</sup> 크기 순으로 정렬되어 있음
- 이진 탐색은 리스트를 반으로 나누어 가운데 숫자와 비교하며 원하는 숫자를 찾는 알고리즘

→ 32 회로 끝네!

$O(\log N)$

# 이진 탐색 (Binary Search)

1. 만약 탐색값이 중앙값보다 크면 우리가 찾으려는 값은 리스트의 후반부에 있을 것이다.
2. 따라서 리스트의 전반부는 탐색의 범위에서 제외할 수 있다.
3. 이러한 기법을 남아 있는 숫자들에 반복적으로 적용한다.

# 이진 탐색 (Binary Search)

- 숫자의 리스트에서 '30'을 찾기:
  - 리스트의 중앙에 있는 값을 탐색값과 비교한다
  - 만약 일치하면 탐색 값을 찾은 것이므로 성공
  - 만약 탐색 값이 중앙값보다 작으면 우리가 찾고자 하는 값은 리스트의 전반부에 있을 것
  - 따라서 리스트의 후반부는 탐색의 범위에서 제외할 수 있다

# 이진 탐색 (Binary Search)

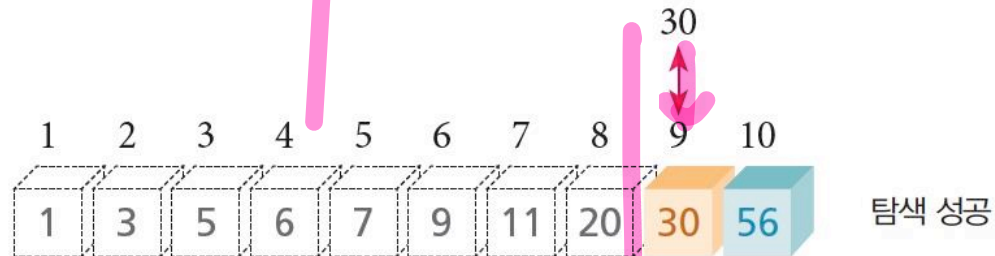
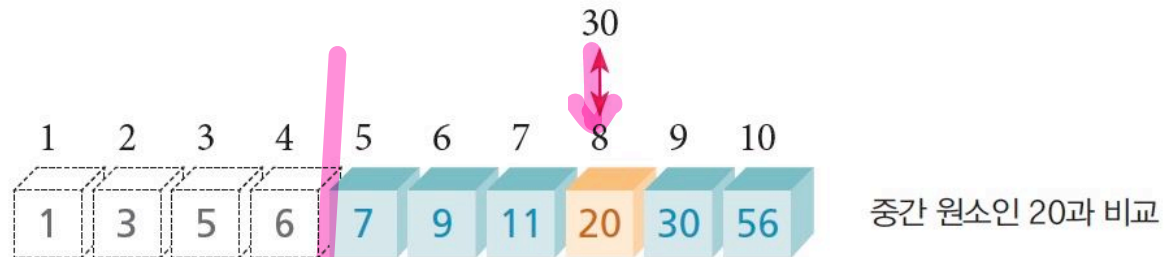
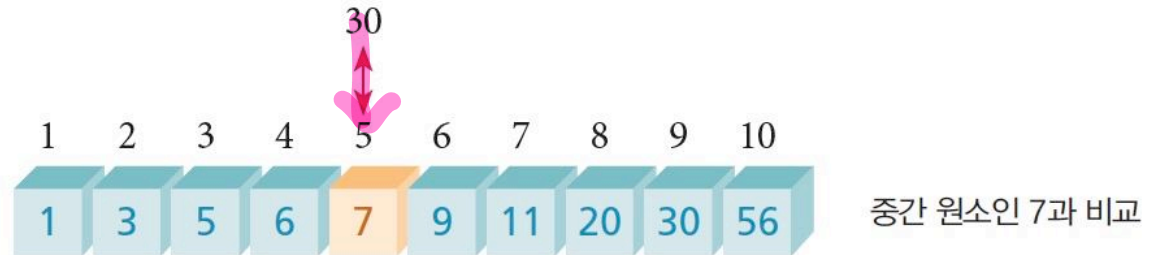


그림 3.5 이진 탐색의 예



# 해시 탐색 (Hash Search)

- 숫자들의 리스트가 있다고 가정
- 이 숫자들은 그룹으로 나누어져 저장되어 있음
- 해시 탐색은 해당하는 숫자가 속한 그룹을 순차적으로 비교하여 원하는 숫자를 찾는 알고리즘이다

# 해시 탐색 (Hash Search)

$$4 = 36 \% 8$$

$$2 = 18 \% 8$$

$$0 = 72 \% 8$$

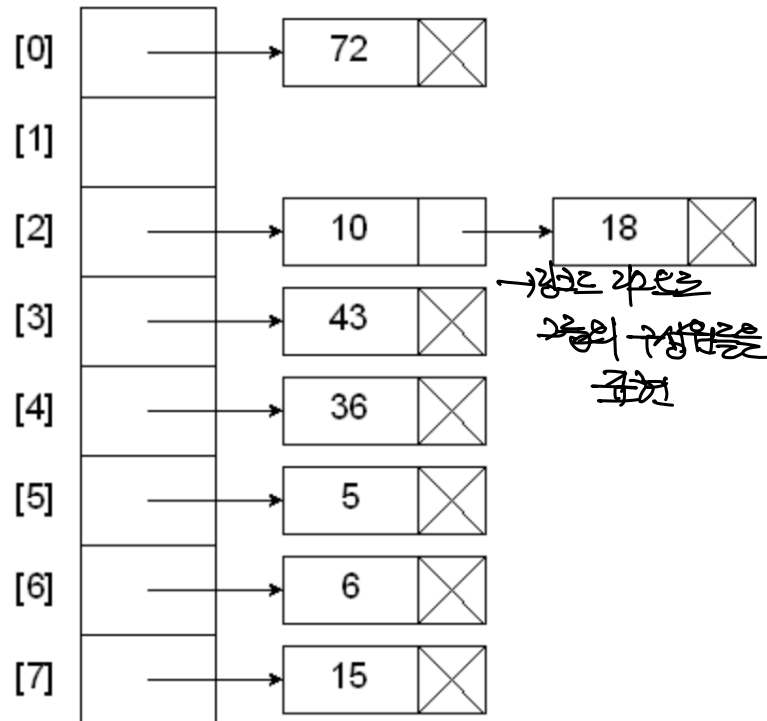
$$3 = 43 \% 8$$

$$6 = 6 \% 8$$

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

# 해시 탐색 (Hash Search)

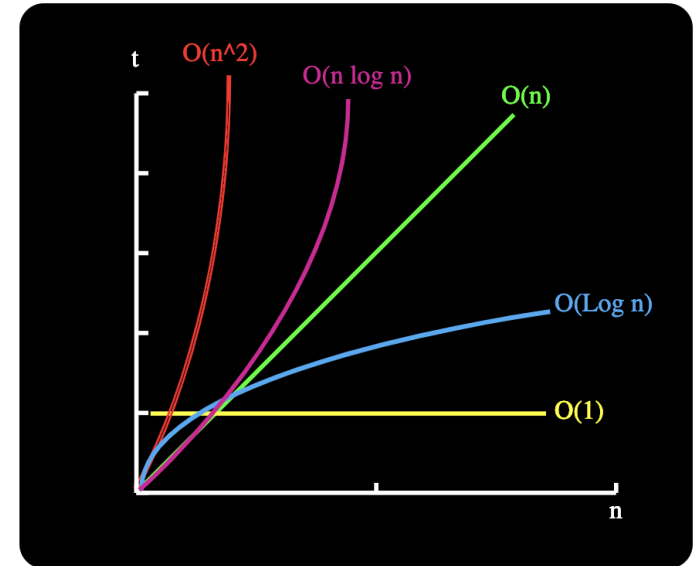
4 = 36 % 8  
2 = 18 % 8  
0 = 72 % 8  
3 = 43 % 8  
6 = 6 % 8  
2 = 10 % 8  
5 = 5 % 8  
7 = 15 % 8



# 알고리즘 평가

- 좋은 알고리즘이란?
  - 정확한 답?
  - 빠른 계산?
  - 적은 메모리 사용량?
  - 이해하기 쉬운 논리?

# 시간 복잡도 비교



$n$	$\log n$	$n$	$n \log n$	$n^2$	$2^n$	$n!$
10	0.003ns	0.01ns	0.033ns	0.1ns	1ns	3.65ms
20	0.004ns	0.02ns	0.086ns	0.4ns	1ms	77years
30	0.005ns	0.03ns	0.147ns	0.9ns	1sec	$8.4 \times 10^{15}$ yrs
40	0.005ns	0.04ns	0.213ns	1.6ns	18.3min	--
50	0.006ns	0.05ns	0.282ns	2.5ns	13days	--
100	0.07	0.1ns	0.644ns	0.10ns	$4 \times 10^{13}$ yrs	--
1,000	0.010ns	1.00ns	9.966ns	1ms	--	--
10,000	0.013ns	10ns	130ns	100ms	--	--
100,000	0.017ns	0.10ms	1.67ms	10sec	--	--
1'000,000	0.020ns	1ms	19.93ms	16.7min	--	--
10'000,000	0.023ns	0.01sec	0.23ms	1.16days	--	--
100'000,000	0.027ns	0.10sec	2.66sec	115.7days	--	--
1,000'000,000	0.030ns	1sec	29.90sec	31.7 years	--	--



$\log_n$   $s_1$   
 $n \log_n$   $s_2$   
 $\log_n$   $n_1$

# 문자 데이터 출력하기

## 1 문자 데이터 출력하기

프로그램 01 ... 영문 문자열과 한글 문자열 출력하기

## 2 숫자 데이터 출력하기

프로그램 02 ... 정수와 실수 출력하기

프로그램 03 ... 더하기, 빼기, 곱하기, 나누기 계산 결과 출력하기

## 3 그래픽 데이터 출력하기

프로그램 04 ... 거북이를 앞과 왼쪽으로 이동하면서 선 그리기

프로그램 05 ... 사각형 그리기

Thinking!

잠깐! Coding

Coding! Programming

# 문자 데이터 출력하기

- 문자열

- 큰따옴표(" ")나 작은따옴표(' ')로 감싼 문자의 집합



- 올바른 예

`"ABC" "abc" "123" "12.3" "@ # @" "파이썬" "Python"`

- 틀린 예

```
>>> x = "AB
SyntaxError: EOL while scanning string literal
>>> x = "AB'
SyntaxError: EOL while scanning string literal
>>> x = 'AB
SyntaxError: EOL while scanning string literal
>>> x = 'AB"
SyntaxError: EOL while scanning string literal
>>> x = ""AB""
SyntaxError: invalid syntax
>>> x = """"AB""""
SyntaxError: EOL while scanning string literal
```





# 문자 데이터 출력하기



TIP

## 문자열을 만드는 방법

1. 큰따옴표로 양쪽 둘러싸기

```
"ABC", "A's C"
```

2. 작은따옴표로 양쪽 둘러싸기

```
'ABC', 'A"BC'
```

3. 큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기

```
"""ABC"""
```

4. 작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기

```
'''ABC'''
```

# 문자 데이터 출력하기

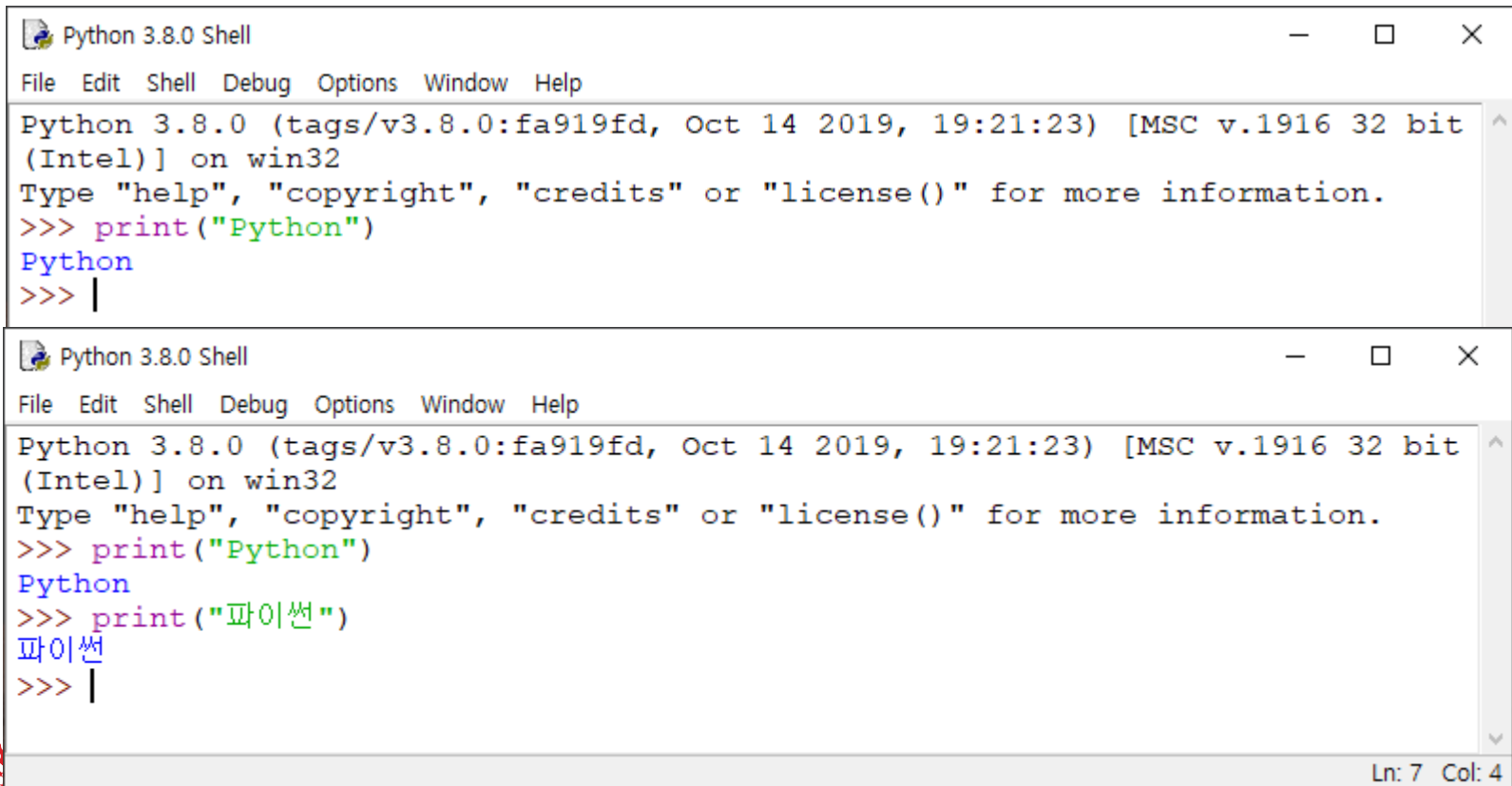
- 문자열 출력
  - print() 함수 인수에 출력하려는 문자열 값을 전달하여 실행

>>> print(      )      "Python"  
                  ↓                    문자열  
                  Python  
                  출력 결과

인수

# 문자 데이터 출력하기

- Python 영문 문자열과 파이썬 한글 문자열 출력



The image shows two screenshots of a Python 3.8.0 Shell window. The top screenshot shows the command `print("Python")` being executed, resulting in the output `Python`. The bottom screenshot shows the command `print("파이썬")` being executed, resulting in the output `파이썬`. Both screenshots show the standard Python 3.8.0 Shell interface with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a status bar at the bottom indicating line and column numbers.

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Python")
Python
>>> |

Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("파이썬")
파이썬
>>> |
```

Ln: 7 Col: 4

# 연습 1: 문자 데이터 출력하기

- print()함수에서 + 연산자를 사용하여 문자열을 연결하기
- print()함수에서 “10” + “20”의 결과가 무엇인지 확인하기
- print()“10” +. 0의 결과는 무엇인지 확인하기
- print()함수에서 “파이썬” \* 10의 결과는 무엇인지 확인하기

# 숫자 데이터 출력하기

- 숫자

- 소수점의 유무에 따라 정수, 실수로 구분

1 0 -1 100 1234567890

1.2 -1.2 0.123456789



## TIP

### 정수의 최댓값/최솟값 확인

파이썬 3에서 정수의 최댓값은 `sys.maxsize`이며, 최솟값은 `-sys.maxsize - 1`이다.

```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
>>> import sys
>>> print(sys.maxsize)
2147483647
>>> print(-sys.maxsize - 1)
-2147483648
>>>
```

Ln: 12 Col: 4



# 숫자 데이터 출력하기

- 숫자 출력

- print() 함수 인수에 콤마로 구분된 값을 전달하여 실행

>>> print( 1, 0, -1, 100, 1234567890 )  
 정수들을 콤마(,)로 구분  
 인수  
 ↓  
 1 0 -1 100 1234567890  
 출력 결과

# 숫자 데이터 출력하기



프로그램

p02-02

정수와 실수 출력하기

다음 순서로 정수와 실수를 각각 출력해보자.

1

1 0 -1 100 1234567890

1.2 -1.2 0.123456789

```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
>>> print(1)
1
>>> print(1, 0, -1, 100, 1234567890)
1 0 -1 100 1234567890
>>> print(1.2, -1.2, 0.123456789)
1.2 -1.2 0.123456789
>>>
```

Ln: 20 Col: 4



KNU

# 숫자 데이터 출력하기

- 수식
  - 연산자, 피연산자로 구성
  - 산술연산자와 수식

연산	연산자	수식	결과
덧셈	+	$6 + 4$	10
뺄셈	-	$6 - 4$	2
곱셈	*	$6 * 4$	24
나눗셈	/	$6 / 4$	1.5



# 숫자 데이터 출력하기

- 수식의 계산 결과 출력
  - print() 함수 인수에 수식을 전달하면 계산 결과 출력

```
>>> print( 6 + 4, 6 - 4, 6 * 4, 6 / 4 )
```

인수                      수식

↓

```
10 2 24 1.5
```

출력 결과

# 숫자 데이터 출력하기

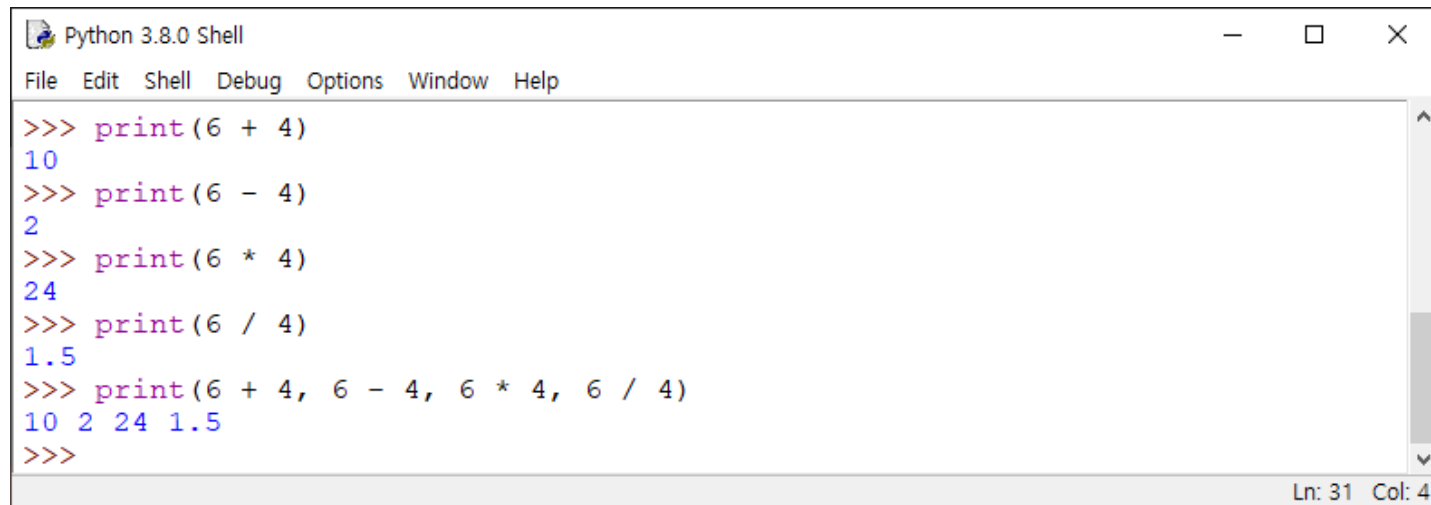
- 다음 순서로 숫자들을 덧셈, 뺄셈, 곱셈, 나눗셈을 하고 계산결과를 출력해보자.

–  $6 + 4$

–  $6 - 4$

–  $6 * 4$

–  $6 / 4$



```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
>>> print(6 + 4)
10
>>> print(6 - 4)
2
>>> print(6 * 4)
24
>>> print(6 / 4)
1.5
>>> print(6 + 4, 6 - 4, 6 * 4, 6 / 4)
10 2 24 1.5
>>>
```

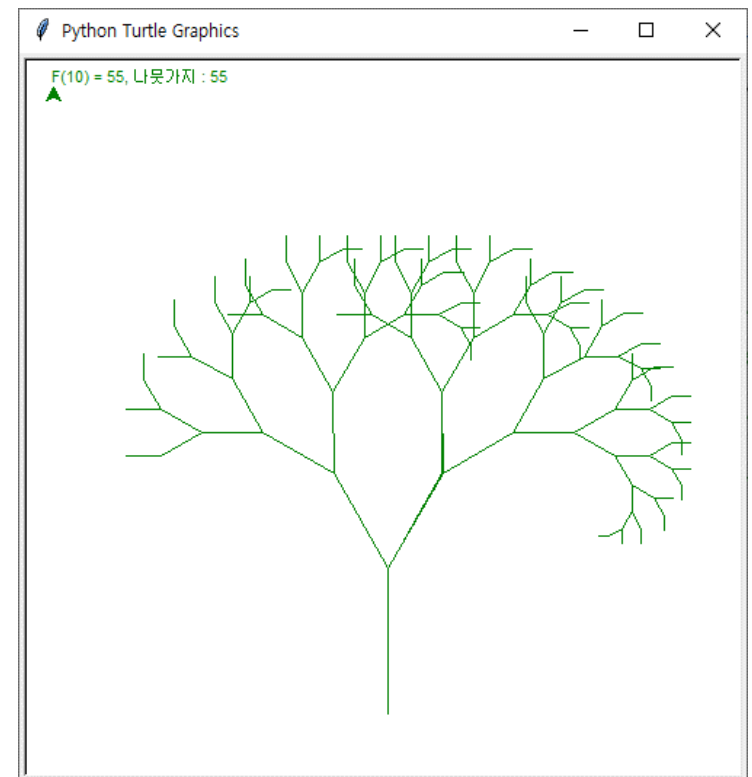
Ln: 31 Col: 4

# 연습 2: 숫자 데이터 출력하기

- $6//4$ 와  $6\%4$ 를 계산하여 출력결과를 비교하기
- $0/1$ ,  $0/0$ ,  $1/0$ 을 각각 계산하여 어느 계산에서 오류가 발생하는지 확인하고 이유를 생각해보기
- 20자리 이상의 큰 정수에 대해서 산술연산을 하였을 때 결과는 어떻게 되는가?
- 1 부터 5까지의 모든 정수를 더해서 출력하기
- 5, 10, 15의 평균 값을 구하기

# 그래픽 데이터 출력하기

- 터틀 그래픽 (turtle graphic)
  - 1966, 교육용 프로그래밍 언어인 Logo에서 처음 소개
  - 꼬리에 잉크가 묻은 거북이를 종이에 올려놓고 리모컨으로 조작하는 방식으로 동작
  - 화면에서 거북이를 이용하여 지나간 흔적으로 만들어지는 그림
  - 거북이가 펜을 가지고 있고 프로그래머가 명령을 이용하여 거북이를 움직이면 그림이 그려짐

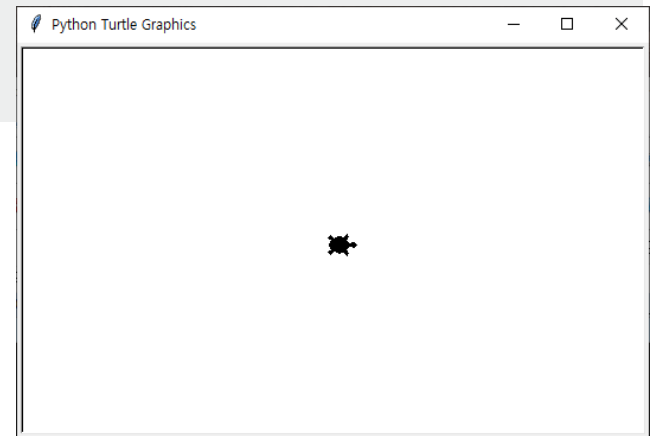








# 그래픽 데이터 출력하기

- 터틀 그래픽의 사용
  - import 예약어로 turtle 모듈을 불러와 사용
  - turtle.shape("turtle")에 의해 거북이가 캔버스에 나타남

```
>>> import turtle  
>>> turtle.shape("turtle")
```

- [Python Turtle Graphics] 화면의 중앙(x:0, y:0)에 거북이가 나타남
- turtle.shape("turtle")에 의해 거북이 모양 변경



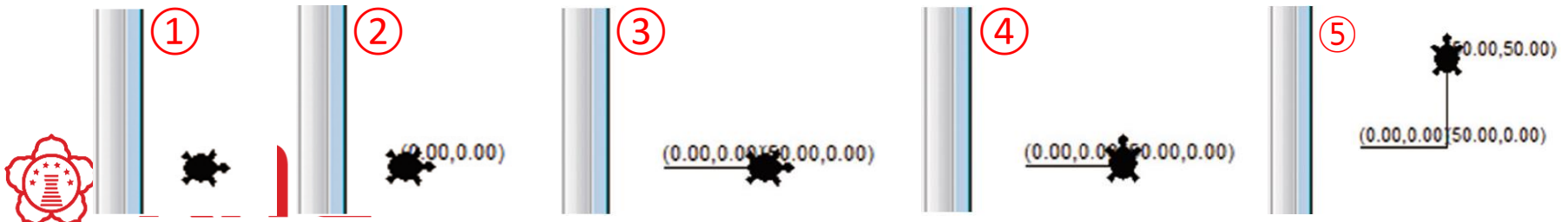
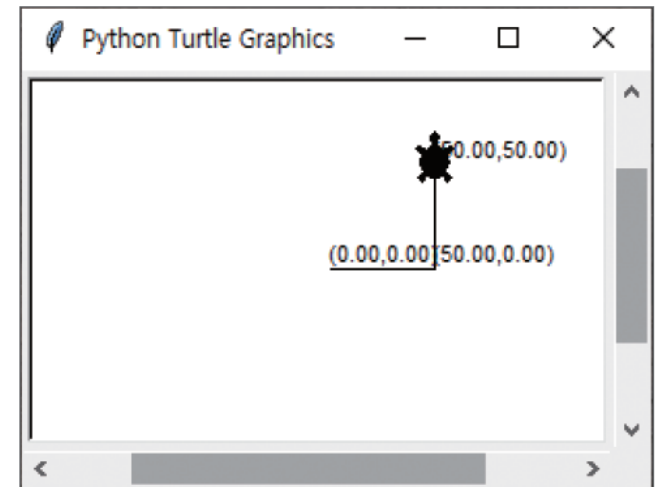
classic	arrow	turtle	circle	square	triangle
					



# 그래픽 데이터 출력하기

- 거북이를 앞으로 50 이동하고 왼쪽으로 90도 회전하고 50도 앞으로 이동하면서 선을 그린다. 현재 위치에서 거북이의 현재 위치를 출력한다

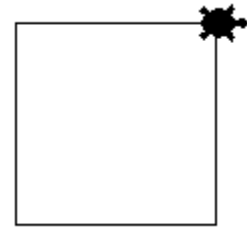
```
>>> import turtle
① >>> turtle.shape("turtle")
② >>> turtle.write(turtle.position())
③ >>> turtle.forward(50)
>>> turtle.write(turtle.position())
④ >>> turtle.left(90)
⑤ >>> turtle.forward(50)
>>> turtle.write(turtle.position())
```



# 그래픽 데이터 출력하기

- 거북이를 이용하여 한 변의 길이가 100인 사각형 그리기

```
>>> import turtle
>>> turtle.shape("turtle")
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
```



# 그래픽 데이터 출력하기

- 정사각형을 그리기 위하여 회전, 이동 과정이 동일하게 4회 반복되었음

```
>>> import turtle
>>> turtle.shape("turtle")
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>>
```

반복  
반복  
반복  
반복

- 위 코드를 반복문을 사용하여 간결하게 작성할 수 있음

```
>>> import turtle
>>> turtle.shape("turtle")
>>> for i in range(4):
    turtle.right(90)
    turtle.forward(100)
```



# 연습 3: 그래픽 데이터 출력하기

- 다음코드의 결과는 무엇인가?

```
>>> import turtle
>>> turtle.shape("turtle")
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
>>> turtle.right(90)
>>> turtle.forward(100)
```

# 수식과 연산자

- 수식(expression)
  - 피연산자들과 연산자의 조합으로 구성
  - 피연산자(operand) : 연산의 대상이 되는 것
  - 연산자(operator)
    - 어떤 연산을 나타내는 기호
    - 산술 연산자, 관계 연산자, 논리 연산자, 비트 연산자 등
  - 수식의 연산에 의해 결과 값이 생성됨

피연산자	연산자	피연산자	
6	+	2	----- 수식
	8		----- 수식의 결과값

# 사칙연산하기

- 사칙연산
  - 덧셈, 뺄셈, 곱셈, 나눗셈 연산자

연산	연산자	수식	결과
덧셈	+	$6 + 4$	10
뺄셈	-	$6 - 4$	2
곱셈	*	$6 * 4$	24
나눗셈	/	$6 / 4$	1.5

# 사칙연산하기

- 사용자로부터 두 개의 정수를 입력받아 각각 변수 x와 변수 y에 대입하고 print()함수를 이용하여 사칙연산 결과를 출력하기

```
>>> x = int(input("정수1 : "))
```

```
정수1 : 4
```

```
>>> y = int(input("정수2 : "))
```

```
정수2 : 2
```

```
>>> print(x + y)
```

```
6
```

```
>>> print(x - y)
```

```
2
```

```
>>> print(x * y)
```

```
8
```

```
>>> print(x / y)
```

```
2.0
```



# 연습 4: 사칙연산하기

- 변수 odd에 1부터 10까지 모든 홀수를 더하여 대입하고 변수 even에 1부터 10까지의 모든 짝수를 더하여 대입하자. 그리고 변수 even에서 odd의 값을 빼서 변수 diff에 대입하고 모든 변수의 값을 출력하자.
- 이름을 문자열로 입력받아 변수 name에 대입하고 출생연도를 정수로 입력받아 변수 year에 대입해보자. 그리고 나이를 계산하여 age 변수에 대입하자.

# 정수 나눗셈과 나머지 연산하기

- 실수 나눗셈
  - / 연산자에 의한 나눗셈 연산은 피연산자가 둘 다 정수라 하더라도 항상 실수 연산을 하여 결과값이 실수가 됨
- 정수 나눗셈과 나머지 계산
  - 나눗셈의 의한 정수 결과를 구할 경우 // 연산자를 사용, 연산의 결과는 나눗셈의 몫에 해당하는 결과임
  - 나눗셈의 나머지 값을 구할 경우에는 % 연산자를 사용

연산	연산자	수식	결과
나눗셈(실수)	/	6 / 4	1.5
나눗셈(정수)	//	6 // 4	1
나머지	%	6 % 4	2

$$1.5 \leftarrow 6 / 4$$

$$\begin{array}{r} 1 \\ 4 \overline{) 6} \\ \underline{-4} \\ 2 \end{array} \leftarrow 6 // 4$$

$$2 \leftarrow 6 \% 4$$

# 정수 나눗셈과 나머지 연산하기

- 정수를 입력받아 500원 동전 개수와 100원 동전 개수를 구해보자

```
>>> x = int(input("금액 : "))
```

```
금액 : 750
```

```
>>> x500 = x // 500
```

```
>>> x100 = x % 500
```

```
>>> x100 = x100 // 100
```

```
>>> print("500원 :", x500, "100원 :", x100)
```

```
500원 : 1 100원 : 2
```

# 연습 5: 정수 나눗셈과 나머지 연산하기

- 50원 동전 개수와 10원 동전 개수도 함께 구할 수 있도록 프로그램 변경하기



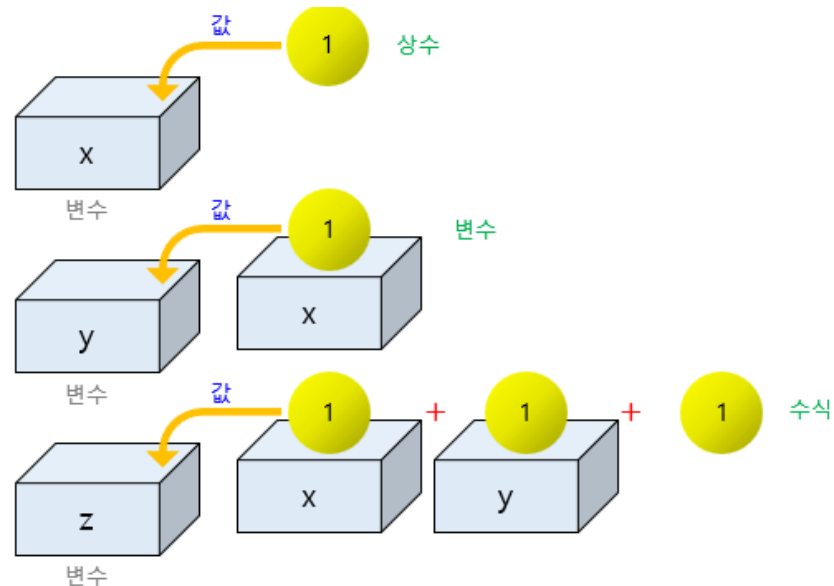
# 대입 연산자와 대입문

- 대입 연산자(assignment operator), = 연산자
  - 변수에 값을 대입할 때 사용
  - **배정 연산자, 할당 연산자**
  - 대입 연산자인 = 기호는 "같다"라는 의미가 아니고,  
변수에 "값을 저장"하는 의미임

# 대입 연산자와 대입문

- 대입문(assignment statement)
  - 대입 연산자가 사용된 문장
  - 배정문, 할당문
  - 대입문에서 대입 연산자의 왼쪽은 반드시 변수이고, 오른쪽은 변수, 상수를 포함한 어떠한 형태의 수식도 가능
  - 다음 대입문에서,  $x, y, z$  : 변수,  $1$  : 상수,  $x + y + 1$  : 수식

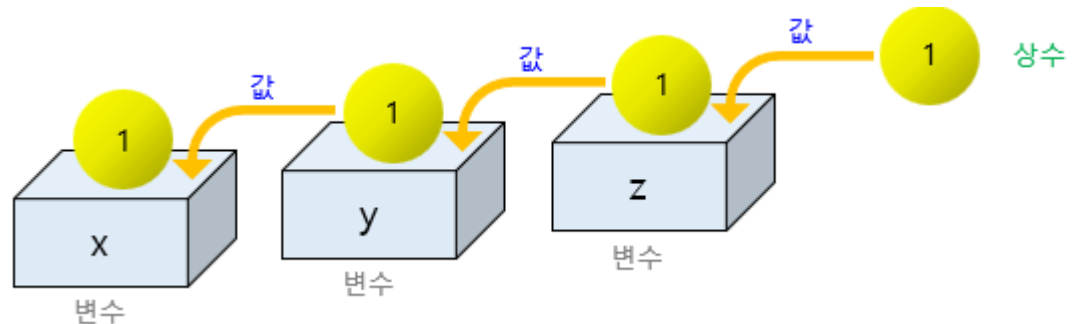
```
x = 1
y = x
z = x + y + 1
```



# 대입 연산자와 대입문

- 다중 대입문(multiple assignment statement)
  - 하나의 대입문에서 대입 연산자가 여러 개 사용될 수 있음
  - 여러 개의 변수에 동일한 값을 대입할 수 있음
  - 다음 대입문의 경우 변수 x, y, z에 1 값이 대입되며,  $z = 1$ ,  $y = z$ 의 값(1),  $x = y$ 의 값(1)의 순서로 대입됨

$x = y = z = 1$



# 대입 연산자와 대입문

- 다중 대입문(**multiple assignment statement**)
  - 형식1 :  $a = b = c = 1$ 
    - 여러 개의 변수에 같은 값을 순차적으로 대입
  - 형식2 :  $a, b = 1, 2$ 
    - = 양쪽에 여러 개의 변수, 여러 개의 수식을 한번에 기입
    - ,로 구분하며 양쪽의 변수 및 표현의 개수는 동일해야 함
    - 형식2의 방식을 사용하여 두 변수의 값을 서로 바꿀 수 있음

```
>>> x, y = 3, 4
>>> print(x, y)
3 4
>>> x, y = y, x
>>> print(x, y)
4 3
```

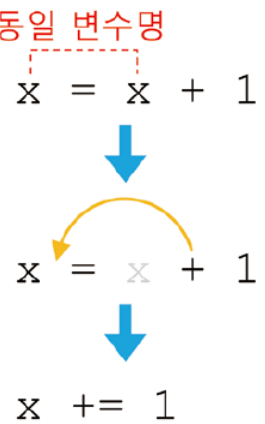
# 대입 연산자와 대입문

- 대입문에서의 오류
  - 대입 연산자의 왼쪽에 변수가 아닌 상수나 수식이 위치할 경우 오류 발생

```
>>> 1 = x
SyntaxError: can't assign to literal
>>> y + 1 = x
SyntaxError: can't assign to operator
```

# 증분 대입 연산자

- 증분 대입 연산자(compound assignment operator)
  - 다른 연산자와 대입 연산자를 결합시켜 놓은 연산자
  - $x = x + 1$ 과 같은 문장을 증분 대입 연산자를 사용하여  $x += 1$ 로 간략히 작성 가능
  - 일반적으로 '복합 대입 연산자'로 사용, 파이썬에서는 '증분(augmented) 대입 연산자'라는 용어로 사용되고 있음



복합 대입 연산자	문장	의미
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
//=	$x //= y$	$x = x // y$
%=	$x \% = y$	$x = x \% y$



# 연산자의 우선순위를 고려하여 연산하기

- 연산의 우선 순위
  - 수식에 2개 이상의 연산자가 사용될 때 어느 연산자를 먼저 평가하여 계산할지 결정해야 함
  - 다음 두 수식의 결과는 어떻게 계산될까?

$$a + b * c$$

$$x * y + z$$

$$\begin{array}{r} a + b * c \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{r} a + b * c \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{r} x * y + z \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{r} x * y + z \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$

- 괄호 사용시 계산의 순서를 보다 더 명확히 할 수 있음

$$a + (b * c)$$

$$(x * y) + z$$

$$\begin{array}{r} a + (b * c) \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{r} (x * y) + z \\ \hline \textcircled{1} \\ \hline \textcircled{2} \end{array}$$



# 연산자의 우선순위를 고려하여 연산하기

- 연산자 우선순위(operator precedence)
  - 산술연산의 경우 기본적으로 다음과 같은 수학적 관례를 따르고 있음
  - 첫 글자를 따서 PEMDAS로 기억
    - 괄호(Parentheses)는 가장 높은 우선순위를 가지며, 괄호 내의 식이 먼저 실행됨
    - 지수승(Exponentiation)은 다음으로 높은 우선순위를 가짐
    - 곱셈(Multiplication)과 나눗셈(Division)은 동일한 우선순위를 가짐
    - 덧셈(Addition)과 뺄셈(Subtraction)은 동일한 우선순위를 가짐
    - 같은 우선순위를 갖는 연산자는 왼쪽에서 오른쪽 순서로 실행됨





## 4.3 연산자의 우선순위를 고려하여 연산하기

- 연산자 우선순위(operator precedence)

순위	연산자	설명	순위	연산자	설명
1	**	지수 연산	8	< > <= >=	관계 연산(비교)
2	~ + -	비트 반전, +부호, -부호	9	== !=	관계 연산(동등)
3	* / // %	곱셈, 실수 나눗셈, 정수 나눗셈, 나머지	10	is, is not	아이덴티티 연산
4	+ -	덧셈, 뺄셈	11	in, not in	소속 연산
5	<< >>	왼쪽 비트 이동, 오른쪽 비트 이동	12	not	논리 부정
6	&	비트 AND	13	and, or	논리 AND, 논리 OR
7	^	비트 XOR, 비트 OR	14	= += -= *= /= //= %= **=	대입 연산



# 연산자의 우선순위를 고려하여 연산하기

- 연산자 우선순위의 변경
  - 괄호를 사용하여 연산자 우선순위 변경 가능

$a + b * c$     #  $a + (b * c)$   
 $x * y + z$     #  $(x * y) + z$

$(a + b) * c$   
 $x * (y + z)$

$$\begin{array}{c} a + \underbrace{b * c}_{\textcircled{1}} \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{c} \underbrace{x * y}_{\textcircled{1}} + z \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{c} \underbrace{(a + b)}_{\textcircled{1}} * c \\ \hline \textcircled{2} \end{array}$$

$$\begin{array}{c} x * \underbrace{(y + z)}_{\textcircled{1}} \\ \hline \textcircled{2} \end{array}$$

# 연산자의 우선순위를 고려하여 연산하기

- 두 정수를 입력받아 평균값 구하기

```
>>> x = int(input("값1 : "))
값1 : 10
>>> y = int(input("값2 : "))
값2 : 20

>>> z = (x + y) / 2
>>> print("평균 :", z)
평균 : 15.0
```

# 연습 6: 연산자의 우선순위를 고려하여 연산하기

- 10, 20, 30, 40, 50의 평균값을 구하여 변수 a에 대입하고 변수 a의 값을 출력하기
- $3 + 2 * 4 / 2$  수식의 계산결과를 변수 b에 대입해보자. 괄호를 사용하여 다시 표현하고 결과를 c에 대입하고 변수 b와 c의 값을 출력하자
- $((3 + 2) * 4) / 2$  수식의 계산 결과를 변수 d에 대입하고 변수 d의 값을 출력하자