

# 파이썬 복습!

프로그래밍2 2주차  
데이터사이언스  
곽찬희

# 1.4 시간 복잡도 분석: 순환 알고리즘

- 순환 알고리즘이란?
- 순환이 더 빠른 예도 있다: 거듭제곱 구하기
- 순환이 훨씬 느린 경우가 많다: 피보나치 수열의 계산
- 복잡한 문제를 쉽게 해결할 수 있다: 하노이의 탑



# 시간 복잡도 분석: 순환 알고리즘

- 순환 알고리즘

- ✓ 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- ✓ 정의자체가 순환적으로 되어 있는 경우에 적합

- ✓ 팩토리얼 구하기

$$n! = \begin{cases} 1 & n=1 \\ n*(n-1)! & n>1 \end{cases}$$

- ✓ 피보나치 수열

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-2) + fib(n-1) & \text{otherwise} \end{cases}$$

- ✓ 이항 계수, 하노이의 탑, 이진 탐색, ...



# 팩토리얼 구하기

- 순환적인 함수 호출 순서

- ✓ `factorial(3) = 3 * factorial(2)`
- ✓ `= 3 * 2 * factorial(1)`
- ✓ `= 3 * 2 * 1`
- ✓ `= 3 * 2`
- ✓ `= 6`

$$n! = \begin{cases} 1 & n=1 \\ n*(n-1)! & n>1 \end{cases}$$

n=3

```
def factorial(n) :  
    if n == 1 : return 1  
    else : return n * factorial(n - 1)
```

⑤ 6반환

①

n=2

```
def factorial(n) :  
    if n == 1 : return 1  
    else : return n * factorial(n - 1)
```

④ 2반환

②

n=1

```
def factorial(n) :  
    if n == 1 : return 1  
    else : return n * factorial(n - 1)
```

③ 1반환



# 팩토리얼: 순환과 반복

- $n$ 의 팩토리얼 구하기

순환 구조
$n! = n * (n-1)!$

↔

반복 구조
$n! = n * (n-1) * (n-2) * \dots * 1$

- 순환(recursion):  $O(n)$ 
  - ✓ 순환적인 문제에서는 자연스러운 방법
  - ✓ 함수 호출의 오버헤드
- 반복(iteration):  $O(n)$ 
  - ✓ for나 while문 이용. 수행속도가 빠름.
  - ✓ 순환적인 문제에서는 프로그램 작성이 어려울 수도 있음.
- 대부분의 순환은 반복으로 바꾸어 작성할 수 있음



# 순환이 더 빠른 예: 거듭제곱 계산

- 방법 1: 반복 구조

```
def power_iter(x, n) :           # 반복으로  $x^n$ 을 구하는 함수
    result = 1.0
    for i in range(n):           # 루브: n번 반복
        result = result * x
    return result
```

✓ 내부 반복문 :  $O(n)$



# 순환적인 거듭제곱 함수

- 방법 2: 순환 구조

*power(x, n)*

```
if n = 0
  then return 1;
else if n이 짝수
  then return power(x2, n/2);
else if n이 홀수
  then return x*power(x2, (n-1)/2);
```

$$\begin{aligned} \text{power}(x, n) &= \text{power}(x^2, n / 2) \\ &= (x^2)^{n/2} \\ &= x^{2(n/2)} \\ &= x^n \end{aligned}$$

$$\begin{aligned} \text{power}(x, n) &= x \cdot \text{power}(x^2, (n-1) / 2) \\ &= x \cdot (x^2)^{(n-1)/2} \\ &= x \cdot x^{n-1} \\ &= x^n \end{aligned}$$

```
def power(x, n) :
```

```
    if n == 0 : return 1
```

```
    elif (n % 2) == 0 :
```

```
        return power(x*x, n//2)
```

```
    else :
```

```
        return x * power(x*x, (n-1)//2)
```

# n이 짝수

# 정수의 나눗셈 (2.3절 참조)

# n이 홀수



# 복잡도 분석

- 순환적인 방법의 시간 복잡도

- ✓  $n$ 이 2의 제곱이라면 문제의 크기가 절반씩 줄어든다.

$$2^n \rightarrow 2^{n-1} \rightarrow \dots 2^2 \rightarrow 2^1 \rightarrow 2^0$$

- 시간 복잡도

- ✓ 순환적인 함수:  $O(\log_2 n)$

- ✓ 반복적인 함수:  $O(n)$



# 순환이 느린 예: 피보나치 수열

- 순환 호출을 사용하면 비효율적인 예
- 피보나치 수열: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

$$fib(n) \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-2) + fib(n-1) & otherwise \end{cases}$$

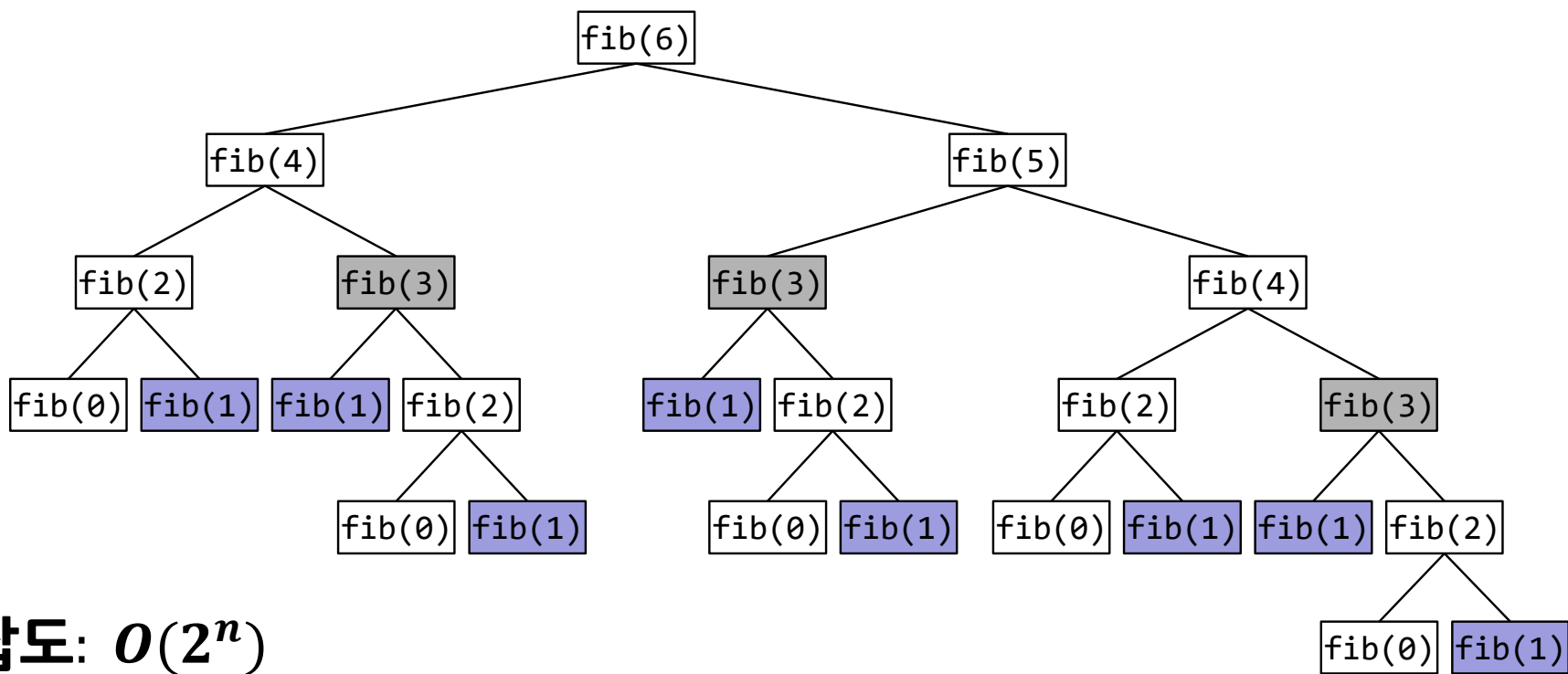
- 순환적인 구현

```
def fib(n) :                                # 순환으로 구현한 피보나치 수열
    if n == 0 : return 0                    # 종료조건
    elif n == 1 : return 1                  # 종료조건
    else :
        return fib(n - 1) + fib(n - 2)     # 순환호출
```



# 순환적인 피보나치의 비효율성

- 같은 항이 중복해서 계산됨!  
✓  $n$ 이 커지면 더욱 심각



- 시간 복잡도:  $O(2^n)$



# 반복적인 피보나치 수열

```
def fib_iter(n) :                                # 반복으로 구현한 피보나치 수열
    if (n < 2): return n

    last = 0
    current = 1
    for i in range(2, n+1) :                      # 반복 루프
        tmp = current
        current += last
        last = tmp
    return current
```

- 시간 복잡도:  $O(n)$



# 하노이 탑 문제



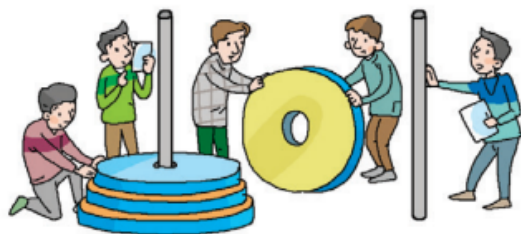
A

B

C



64개의 원판을 모두 C로 옮겨야 합니다. 이동 횟수는 최소로 해야 하고요.



A

B

C



소중한 것이니 반드시 한 번에 하나씩만 옮길 수 있어요.

A

B

C



A

B

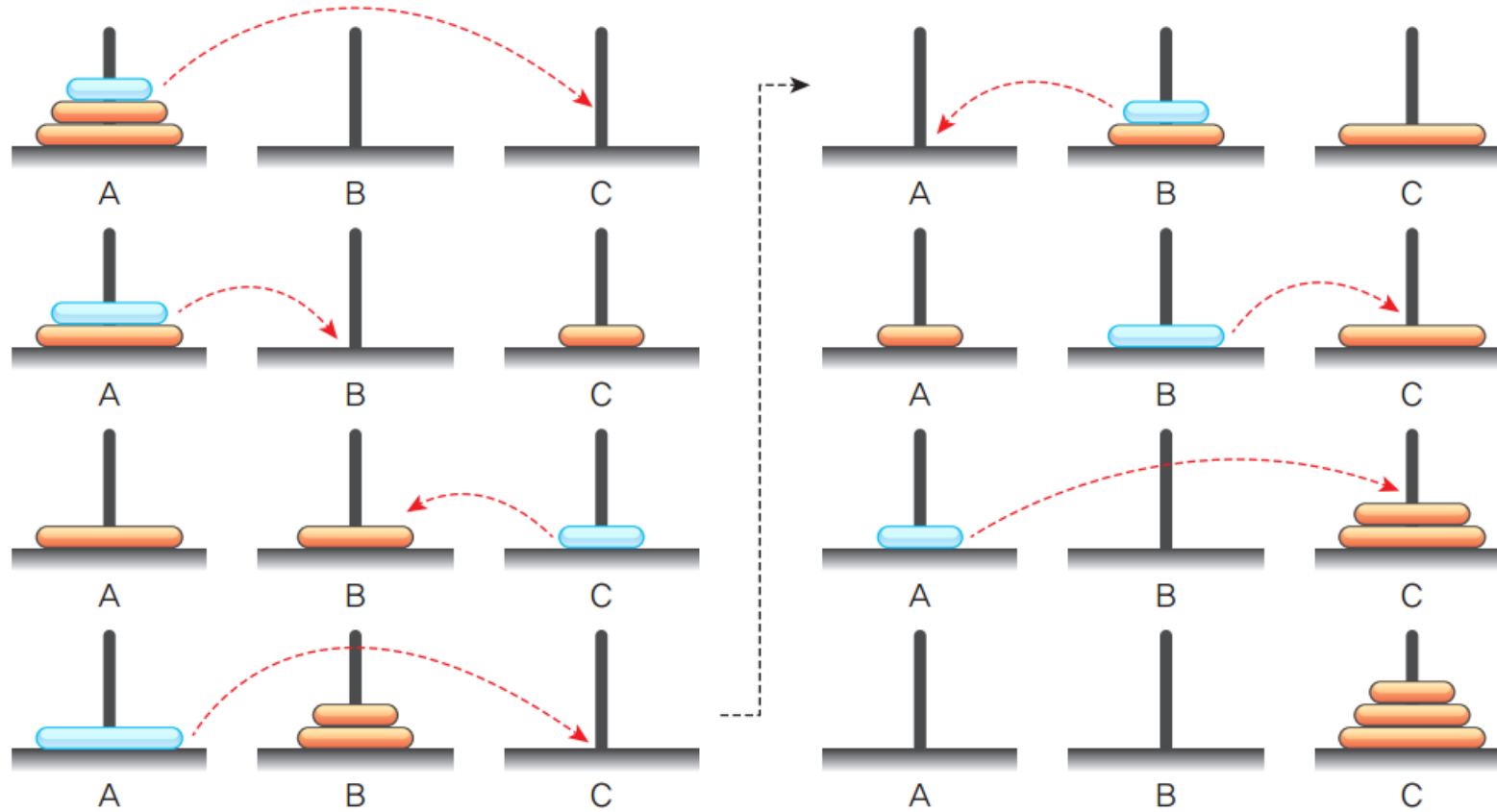
C



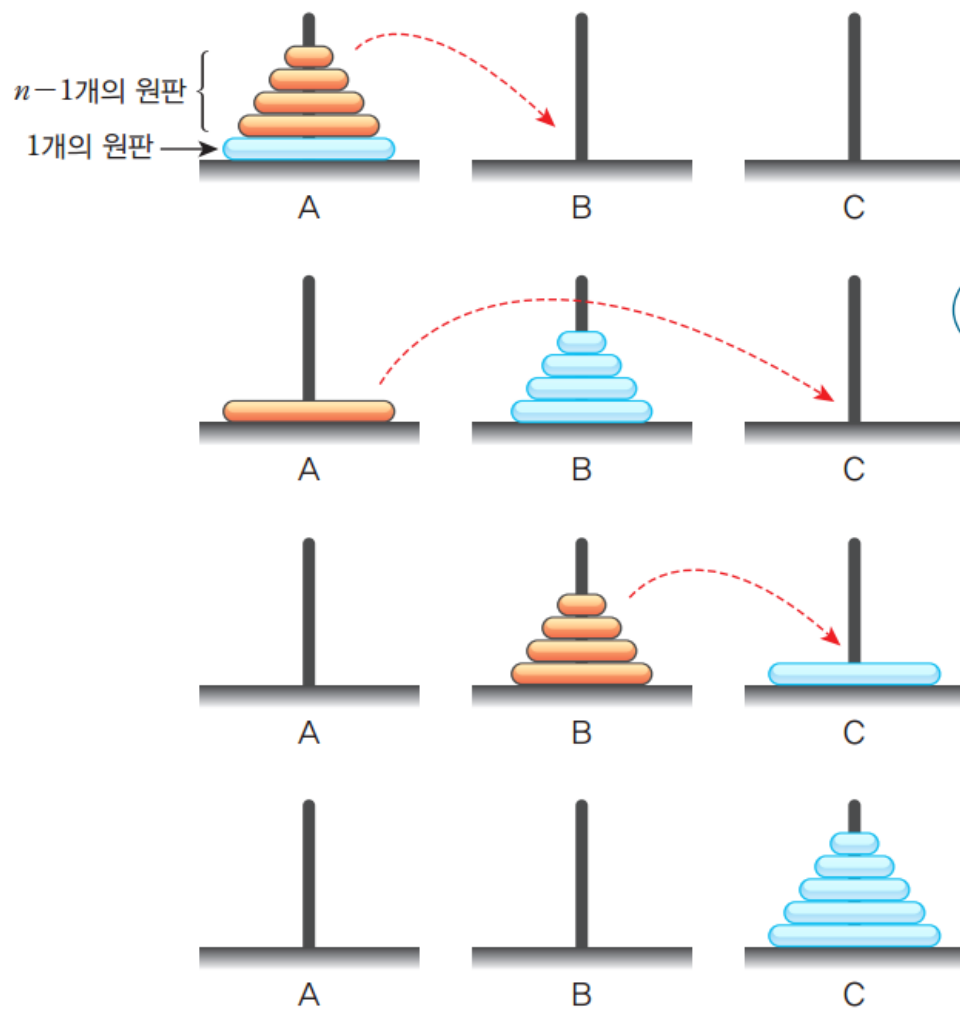
작은 판 위에 큰판이 올라가면 절대 안되요.

B를 임시 막대로 사용하면 됩니다.

# $n=3$ 인 경우의 해답



# 일반적인 경우에는?



먼저  
 $n-1$ 개를 C를 이용해서  
B로 옮기고

A에 남은  
하나는 쉽게 C로  
옮길 수 있고

B에 있는  
 $n-1$ 개를 A를 이용해서  
C로 옮기면... 끝.

아무리 많아도  
문제 없겠는데...



# 구현

- 어떻게  $n-1$ 개의 원판을 A에서 B로, 또 B에서 C로 이동하는가?
  - ✓ 순환을 이용

```
def hanoi_tower(n, fr, tmp, to) :           # Hanoi Tower 순환 함수

    if (n == 1) :                           # 종료 조건
        print("원판 1: %s --> %s" % (fr, to)) # 가장 작은 원판을 옮김
    else :
        hanoi_tower(n - 1, fr, to, tmp)     # n-1개를 to를 이용해 tmp로
        print("원판 %d: %s --> %s" % (n,fr,to)) # 하나의 원판을 옮김
        hanoi_tower(n - 1, tmp, fr, to)     # n-1개를 fr을 이용해 to로
```

```
hanoi_tower(4, 'A', 'B', 'C')              # 4개의 원판이 있는 경우
```



# 하노이탑(n=3) 실행 결과

```
C:\WINDOWS\system32\cmd.exe
원판 1: A --> B
원판 2: A --> C
원판 1: B --> C
원판 3: A --> B
원판 1: C --> A
원판 2: C --> B
원판 1: A --> B
원판 4: A --> C
원판 1: B --> C
원판 2: B --> A
원판 1: C --> A
원판 3: B --> C
원판 1: A --> B
원판 2: A --> C
원판 1: B --> C
```

원판의 이동(예 1번 원판을 A에서 B로 이동한다.)

$$\begin{aligned}T(n) &= 2T(n-1) + 1 \\&= 2(2T(n-2) + 1) + 1 \\&= 2(2(2T(n-3) + 1) + 1) + 1 \\&= 2^{n-1}T(1) + \dots \\&= 2^{n-1} + \dots \\&= O(2^n)\end{aligned}$$





## 2. 파이썬 복습

# 학습목표

---

- 이 책에서 사용되는 파이썬 문법들을 이해한다.
- C와 같은 기존 언어와 파이썬 코드의 차이를 이해한다.
- 파이썬에서 지원하는 다양한 컬렉션 자료형을 활용할 수 있다.



# 2.1절 ~ 2.5절 파이썬 기초

- 2.1 파이썬 이란?
- 2.2 자료형, 리터럴과 변수
- 2.3 파이썬의 연산
- 2.4 함수 호출과 입출력 함수
- 2.5 제어 구조와 반복



# 2.1 파이썬이란?

- AI와 빅데이터의 부상과 함께 최근 각광받고 있는 언어

PYPL PopularitY of Programming Language

Worldwide, Sept 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	31.56 %	+2.9 %
2		Java	16.4 %	-3.1 %
3		Javascript	8.38 %	+0.3 %
4		C#	6.5 %	-0.8 %
5		PHP	5.85 %	-0.5 %

- ✓ 1991년에 귀도 반 로섬(Guido van Rossum)
- ✓ 문법이 쉬움. 코드를 보면 직관적으로 알 수 있는 부분이 많음
- ✓ 인터프리터 방식 ↔ 컴파일 방식
- ✓ 스크립트 모드 지원
- ✓ 통합개발환경: IDLE, 주피터 노트북, 파이참, 비주얼 스튜디오



## 2.2 자료형, 리터럴과 변수

- 파이썬 예약어(keyword)

and	as	assert	break	class	continue	def
del	elif	else	except	finally	for	from
global	if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try	while
with	yield	False	None	True		



## 2.2 자료형, 리터럴과 변수

### • 리터럴과 자료형

분류	내장 자료형	리터럴		
수치	정수(int)	10      -30      0xfffe      073		
	실수(float)	3.14      -0.45      123.032E-13		
	복소수(complex)	complex(1,2)      1+2j      4+5j		
	부울(bool)	True      False		
시퀀스	문자열(str)	'game'      "over"      "C"		
	리스트(list)	[]      [0, 1, 2, 3]      [0, 'hello', 3.14]		
	튜플(tuple)	(0, 1, 2, 3)      ('hello', 'world', 'game')		
매핑	딕셔너리(dict)	{ 3.14 : "phi", 4.5 : "score" }		
집합	집합(set, frozenset)	{ 1, 2, 3 }      {'one', 'two', 'three' }		



## 2.2 자료형, 리터럴과 변수

### • 변수 이해하기



- 파이썬에서는 모든 자료가 클래스로부터 만들어진 객체이다.
- 변수는 다른 객체를 참조하는 참조자 또는 포인터의 역할을 한다.

## 2.2 자료형, 리터럴과 변수

- 변수(variable)

```
number = 132          # 변수 생성 및 사용 문장
number = number + 8    # 변수 사용 문장
pi = 3.14              # float 변수
comp = 1 + 2j          # complex 변수
isValid = True         # bool 변수
msg = 'game over !!!'  # str 변수
A = [0, 1, 1, 2, 3, 5, 8, 13 ] # list 변수
```

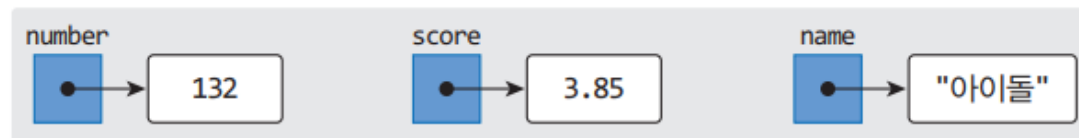




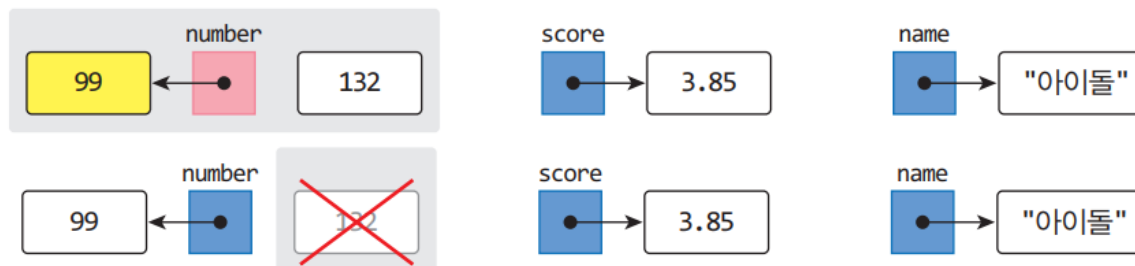
## 2.2 자료형, 리터럴과 변수

- 변수의 동작 정확히 이해하기

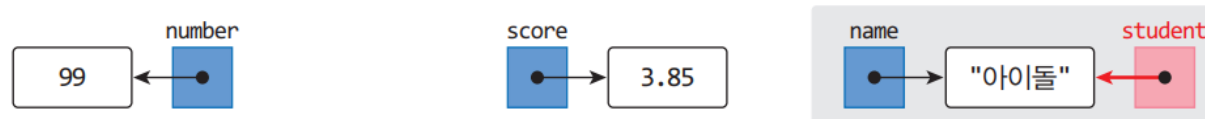
```
number = 132  
score = 3.85  
name = "아이돌"
```



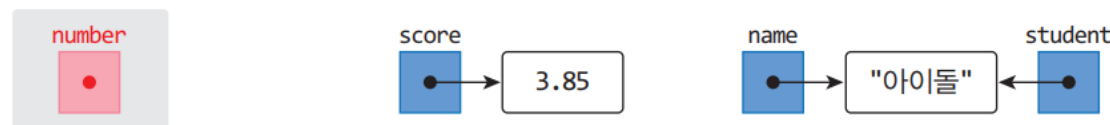
```
number = 99
```



```
student = name
```



```
number = None
```



## 2.3 파이썬의 연산

- 나눗셈 연산자 변경
  - ✓ 연산자 / : 실수의 나눗셈 (결과가 실수)
  - ✓ 연산자 // : 정수 연산(floor division)
- 이항 연산자 \*\* 추가
- 단항 연산자 ++, -- 제공 없음
  - ✓ x++가 아니라 x += 1
- 관계 연산자 >, <, >=, <=, ==, !=
- 불리언 연산자: ||, &&, ! → or, and, not
- in과 not in 연산자

```
'a' in 'banana'           # True  
'seed' in 'banana'       # False
```

```
A = [0, 1, 1, 2, 3, 5, 8, 13 ]  
if 3 in A :  
while 4 in A :
```



## 2.4 함수 호출과 입출력 함수

- 함수 호출

```
y = sum ( a, b )           # 함수 sum 호출. 인수 2개
w = x * z + func( a, b, c ) # 함수 func 호출. 인수 3개
print( "game over" )       # 화면출력함수 print 호출.
```

- 키보드 입력함수: input()

```
name = input("당신의 이름을 입력하세요.") # 문자열
hobby = input("취미가 무엇입니까?")      # 문자열

age = int( input("나이가 몇 살입니까?"))  # 입력 문자열을 정수로 변환
score = float( input("평균 학점이 얼마입니까?")) # 입력 문자열을 실수로 변환
```



## 2.4 함수 호출과 입출력 함수

- 화면 출력함수: print()

```
print( "game over" )
```

```
print("game", end=" ")  
print("over")
```

\\ Backslash (\)

\n Newline

\ " Double quote (")

\ Single quote (')

\t Horizontal tab



# 2.5 제어 구조와 반복

- 분기(branching): if, else, elif

```
if value % 2 == 0 :  
    print("짝수입니다.")
```

```
if score >= 90 :  
    print("합격입니다.")  
    print("장학금 대상입니다.")
```

```
if score >= 90 :  
    print("합격입니다.")  
    print("장학금 대상입니다.")
```

```
if value % 2 == 0 : print("짝수입니다.")      # 블록을 한 줄로 처리  
else : print("홀수입니다.")                  # 블록을 한 줄로 처리
```

```
if score >= 90 :  
    grade = "A"  
else :  
    if score >= 80 :  
        grade = "B"  
    else :  
        if score >= 70 :  
            grade = "C"  
        else :  
            if score >= 60 :  
                grade = "D"  
            else:  
                grade = "F"
```

```
if score >= 90 :  
    grade = "A"  
elif score >= 80 :  
    grade = "B"  
elif score >= 70 :  
    grade = "C"  
elif score >= 60 :  
    grade = "D"  
else :  
    grade = "F"
```



## • 반복(looping)

```
dan = int(input("구구단 단 입력: "))
for n in range( 2, 10, 1 ) :
    print( "%2d x %2d = "% (dan, n), dan*n )
```

```
dan = int(input("구구단 단 입력: "))
n = 2
while n < 10 :
    print( "%2d x %2d = "% (dan, n), dan*n )
    n += 1
```

## • Range()

```
for n in range( 5 ) :           # n: 0, 1, 2, 3, 4
```

```
for n in range( 2, 10 ) :       # n: 2, 3, ..., 9
```

```
for n in range( 10, 3, -2 ) :    # n: 10, 8, 6, 4
```

```
for item in [12, 33, 52, 26, 99] : # 리스트의 모든 항목에 대해 반복
```

```
    print( "값 =", item)         # 12, 33, 52, 26, 99 출력
```

```
for c in "Game Over !" :        # 문자열의 각 문자에 대해
```

```
    print( "값 =", c)
```

```
myDict = { 'A':12, 'B':33, 'C':52, 'D':26, 'E':99 }
```

```
mySet = set([12, 33, 52, 26, 99])
```

```
for e in mySet :
```

```
    print( "값 =", e)
```

```
for e in myDict :
```

```
    print( "키 =", e)
```

```
    print( "값 =", myDict[e])
```

## 2.6 컬렉션 자료형

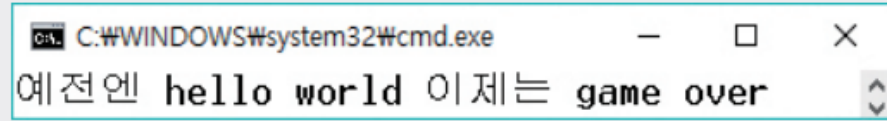
- 문자열(str)
- 리스트(list)
- 튜플(tuple)
- 딕셔너리(dict)
- 집합(set)



# 문자열(str)

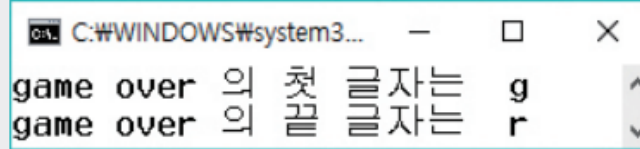
- 문자열

```
msg = 'game over'  
hi = "hello world"  
sum = "예전엔 " + hi + " 이제는 " + msg  
print(sum)
```

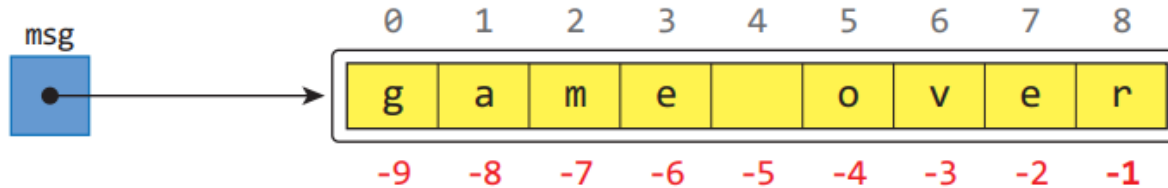


C:\WINDOWS\system32\cmd.exe  
예전엔 hello world 이제는 game over

```
print(msg, '의 첫 글자는 ', msg[0])  
print(msg, '의 끝 글자는 ', msg[-1])
```



C:\WINDOWS\system32\cmd.exe  
game over 의 첫 글자는 g  
game over 의 끝 글자는 r



```
hobby = "테니스"  
age = 21  
score = 4.5  
msg1 = " 당신의 학점은 %4.1f입니다" % score  
msg2 = " 취미=%s, 나이=%d, 학점=%f" % (hobby, age, score)
```





# 리스트(list)

- 스마트한 배열

```
big3 = [ ]  
lotto = [23, 34, 11, 42, 9]  
big4 = [ '제이플라', '도티', '대도서관', '보람튜브' ]  
print("lotto[1] = ", lotto[1])  
big4[2] = '블랙핑크'
```

메소드	설명	big3.append("알라딘") big3.append("엘사") big3.append("안나")
s.append(item)	항목 item을 리스트 s의 맨 뒤에 추가한다.	
s.extend(lst)	리스트 lst를 s에 추가한다.	
s.count(item)	리스트에서 항목 item의 개수를 세고 그 개수를 반환한다.	
s.index(item,[시작],[종료])	리스트에서 항목 item을 찾아 가장 작은 인덱스를 반환한다. 탐색의 시작 위치와 종료 위치를 지정할 수도 있다.	
s.insert(pos, item)	pos 위치에 항목 item을 삽입한다.	
s.pop(pos)	pos 위치의 항목을 s에서 꺼내고 반환한다.	
s.remove(item)	항목 item을 s에서 제거한다.	
s.reverse()	리스트 항목의 순서를 뒤집는다.	
s.sort([key], [reverse])	항목을 정렬한다.	

# 튜플(tuple)

- 리스트(list)와 동일하지만 크기나 값을 변경할 수 없음

```
t = ( 0, 3, 7 )           # 항목이 3개인 튜플
a = ( 2 )                 # 항목이 1개인 튜플
b = ( 'game', 1, 3.14, 2019 ) # 항목이 4개인 복합 튜플

print (" 취미=%s, 나이=%d, 학점=%f" % (hobby, age, score)) # 튜플
```

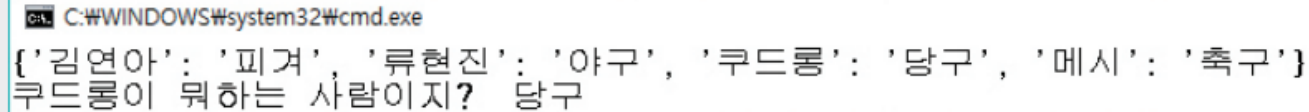
- 메모리는 효율적 (3장)



# 딕셔너리(dict)

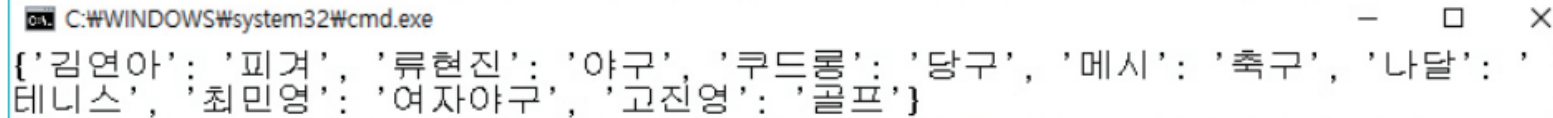
- 키(key)와 관련된 값(value)로 이루어진 항목(entry)들의 집합

```
map = { '김연아':'피겨', '류현진':'야구', '쿠드롱':'당구', '메시':'축구' }  
print(map)  
print('쿠드롱이 뭐하는 사람이지? ', map['쿠드롱'])
```



C:\WINDOWS\system32\cmd.exe  
{'김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구'}  
쿠드롱이 뭐하는 사람이지? 당구

```
map['나달'] = '테니스' # 맵에 항목 추가  
map.update({'최민영':'여자야구', '고진영':'골프'}) # 여러 항목 추가  
print(map)
```



C:\WINDOWS\system32\cmd.exe  
{'김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구', '나달': '테니스', '최민영': '여자야구', '고진영': '골프'}

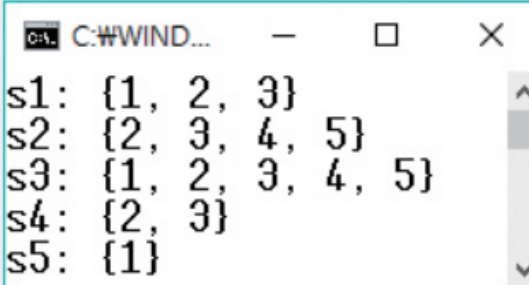


# 집합(set)

- set과 frozenset(내용을 변경할 수 없는 set)

```
s1 = { 1,2,3 }           # 집합 객체
s2 = { 2,3,4,5 }         # 집합 객체
s3 = s1.union(s2)         # 합집합
s4 = s1.intersection(s2)  # 교집합
s5 = s1 - s2              # 차집합

print("s1:", s1)
print("s2:", s2)
print("s3:", s3)
print("s4:", s4)
print("s5:", s5)
```



```
s1: {1, 2, 3}
s2: {2, 3, 4, 5}
s3: {1, 2, 3, 4, 5}
s4: {2, 3}
s5: {1}
```

```
s5 = { 3.14 }           # 원소가 하나인 집합
map = { 3.14 : 'Phi' }  # 엔트리가 하나인 딕셔너리
```



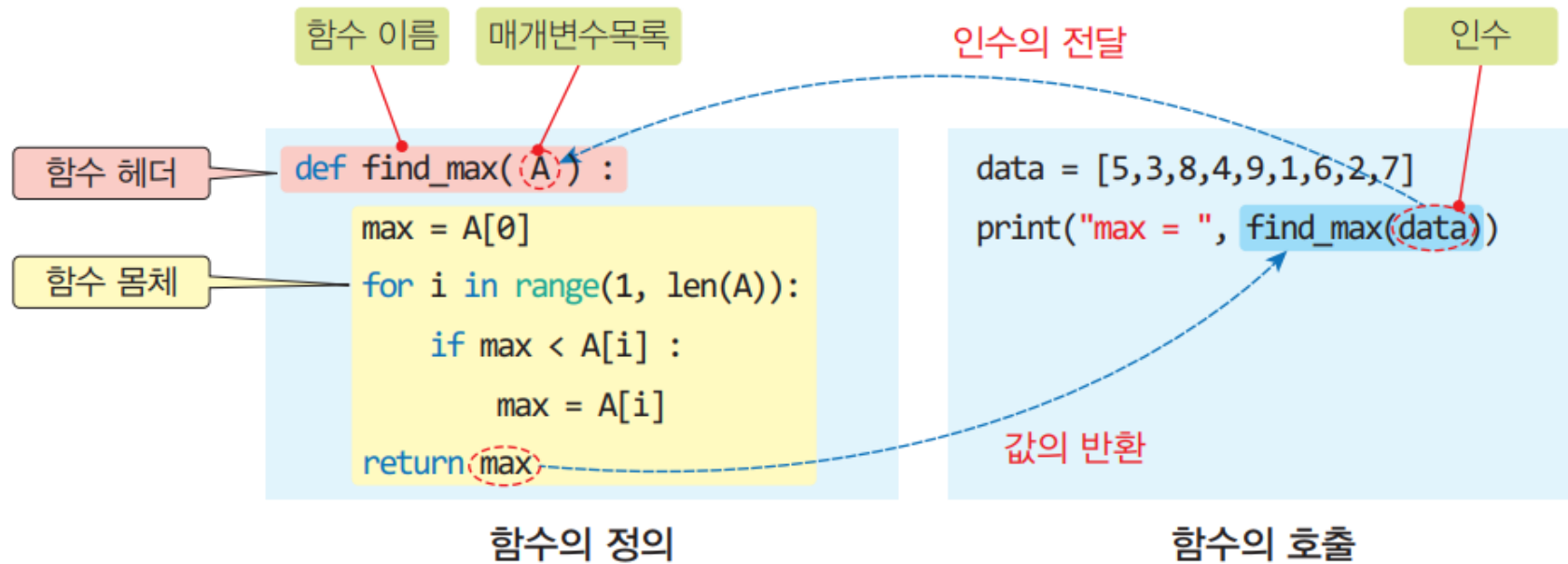
# 2.7절 ~ 2.9절 함수, 변수, 모듈

- 2.7 사용자 정의 함수
- 2.8 변수의 범위
- 2.9 모듈과 이름 공간(namespace)



## 2.7 사용자 정의 함수

- 파이썬 내장 함수: `type()`, `len()`, `ord()` 등
- 사용자 정의 함수



# 여러 개의 값의 반환할 수 있다

- 여러 개의 값의 반환

```
def find_min_max(A) :                                # 최댓값과 최솟값을 동시에 찾아 반환
    min = A[0]
    max = A[0]
    for i in range(1, len(A)) :                      # i : 1 ~ len(A)-1
        if max < A[i] : max = A[i]                  # 최댓값 갱신
        if min > A[i] : min = A[i]                  # 최솟값 갱신
    return min, max                                   # 최솟값과 최댓값을 반환
```

```
data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]
x, y = find_min_max(data)                            # 최솟값과 최댓값을 반환받음
print("(min,max) = ", (x,y))                         # x와 y를 튜플로 만들어 출력
```



# 디폴트 인수와 키워드 인수

- 디폴트 인수

```
def sum_range(begin, end, step=1) :           # 매개변수 step이 기본 값을 가짐
    sum = 0
    for n in range(begin, end, step) :
        sum += n
    return sum
```

```
print("sum = ", sum_range(1, 10))             # step은 디폴트 값(1)으로 처리됨
```

```
print("sum = ", sum_range(1, 10, 2))          # 정상 호출. step은 2
```

- 키워드 인수

```
print("sum = ", sum_range(step=3, begin=1, end=10)) # 키워드 인수 사용
```

```
print("game ", end=" ")                       # 라인피드가 발생하지 않음(키워드 인수 사용)
```





## 2.8 변수의 범위

- **내장 범위(built-in scope)**
  - ✓ 언어의 일부로 정의된 변수와 리터럴들
  - ✓ 프로그램의 어디에서나 사용할 수 있다.
- **전역 범위(global scope)**
  - ✓ 소스 파일의 맨 꼭대기 레벨(함수나 클래스 밖)에서 생성
  - ✓ 프로그램의 어디에서나 사용할 수 있다.
- **지역 범위(local scope)**
  - ✓ 함수나 클래스의 멤버함수(메소드) 안에서 생성
  - ✓ 그 안에서만 사용. 함수의 매개변수들도 지역범위
- **인스턴스 범위(instance scope)**
  - ✓ 클래스의 데이터 멤버로 생성된 변수
  - ✓ 클래스 내의 다른 함수들에서 사용할 수 있다.



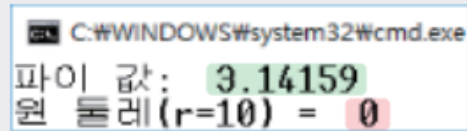
# 전역 변수

```
def calc_perimeter(radius) :  
    # global perimeter  
    print("파이 값: ", pi)  
    perimeter = 2*pi * radius
```

# 파이값 출력 --> OK  
# 원 둘레 계산 --> 이상?

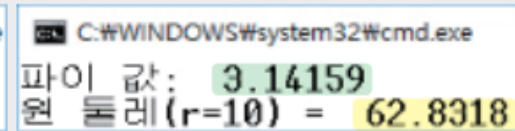
```
pi = 3.14159  
perimeter = 0  
calc_perimeter(10)  
print("원둘레(r=10) = ", perimeter)
```

# 전역변수 pi : 파이값의 근사치  
# 전역변수 perimeter : 원 둘레



C:\WINDOWS\system32\cmd.exe  
파이 값: 3.14159  
원 둘레(r=10) = 0

이상한 결과?



C:\WINDOWS\system32\cmd.exe  
파이 값: 3.14159  
원 둘레(r=10) = 62.8318

원하는 결과



## 2.9 모듈과 이름 공간(namespace)

```
# 파일명: min_max.py
def find_min_max(A) :
    ...
    return min, max
```

# 다음 함수가 min\_max.py에 저장되어 있음  
# 최댓값과 최솟값을 동시에 찾아 반환  
# 2.7장의 find\_min\_max()함수 코드와 동일  
# 최솟값과 최댓값을 반환

```
# 파일명: sum.py
def sum_range(begin, end, step=1) :
    ...
    return sum
```

# 다음 함수가 sum.py에 저장되어 있음  
# 매개변수 step이 기본 값을 가짐  
# 2.7장의 sum\_range()함수 코드와 동일

```
# 파일명: my_job.py
import min_max
import sum

data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]
print("(min,max) = ", min_max.find_min_max(data))
print("sum = ", sum.sum_range(1, 10))
```

# min\_max 모듈을 사용함  
# sum.py 모듈을 사용함  
# min\_max 모듈의 함수 사용  
# sum 모듈의 함수 사용



```
from math import pow, sqrt      # math 모듈에서 pow, sqrt 식별자를 사용
result = pow(2,10)              # pow 식별자를 바로 사용 가능
dist = sqrt(1000)              # sqrt 식별자를 바로 사용 가능
```

# 파일명: my\_job.py

```
from min_max import *          # min_max 모듈의 모든 식별자 사용 가능
from sum import *              # sum 모듈의 모든 식별자 사용 가능
```

```
data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]
print("(min,max) = ", find_min_max(data))    # 바로 사용
print("sum = ", sum_range(1, 10))           # 바로 사용
```

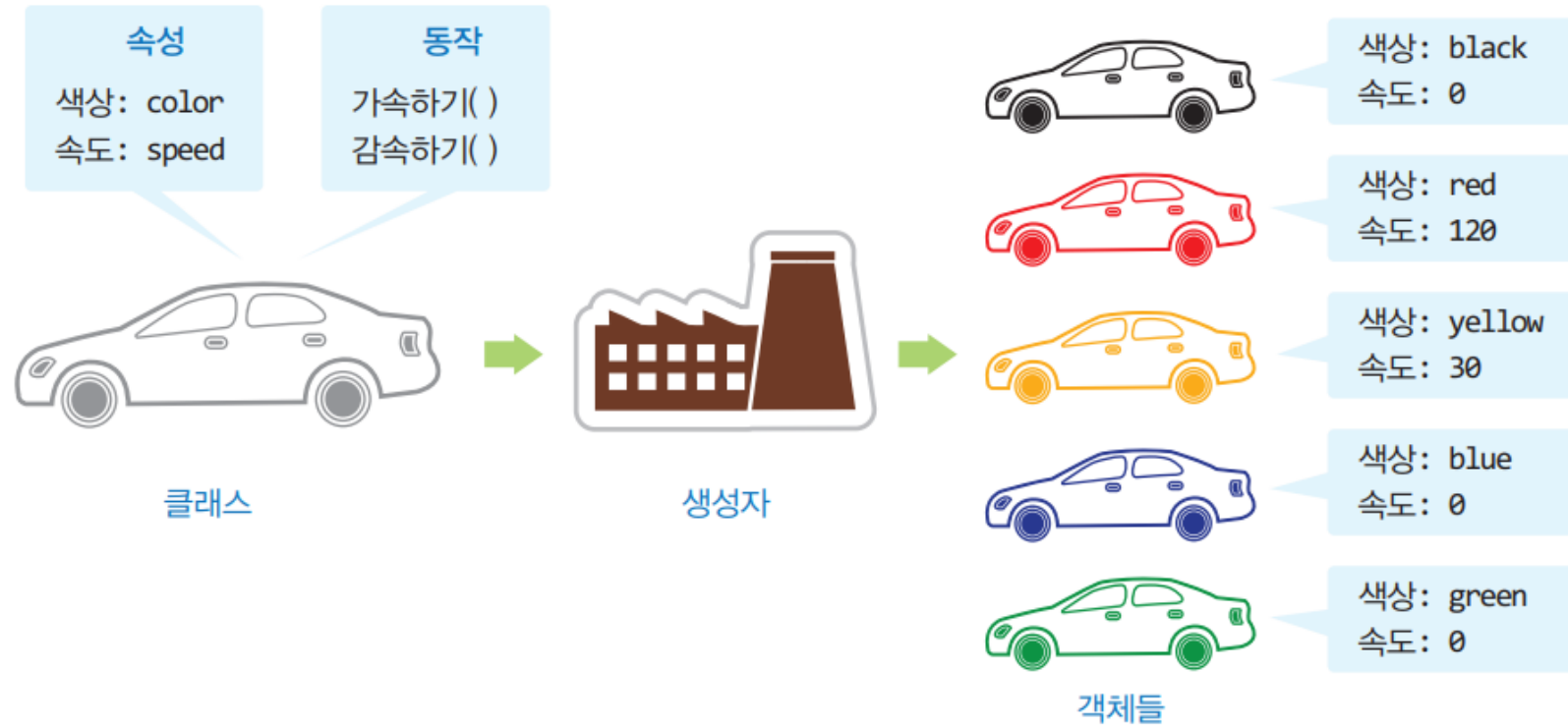


# 2.10절 ~ 2.12절 객체지향 프로그래밍

- 2.10 클래스
- 2.11 연산자 중복
- 2.12 상속



## 2.10 클래스



### • 클래스 정의

```
class Car :
```

```
# 자동차와 관련된 클래스 정의 블록이 이어짐. 들여쓰기에 유의할 것
```



# 클래스와 생성자

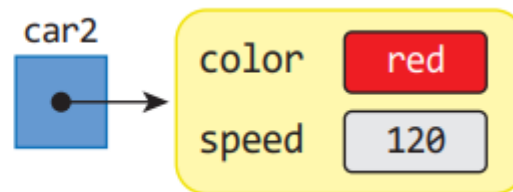
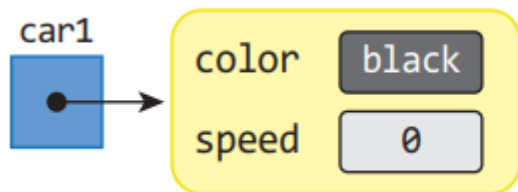
## • 생성자

```
class Car :  
    def __init__(self, color, speed = 0) :  
        self.color = color  
        self.speed = speed
```

# 생성자 함수  
# 데이터 멤버 color 정의 및 초기화  
# 데이터 멤버 speed 정의 및 초기화

```
car1 = Car('black', 0)  
car2 = Car('red', 120)  
car3 = Car('yellow', 30)  
car4 = Car('blue', 0)  
car5 = Car('green')
```

# 검정색, 속도 0  
# 빨간색, 속도 120  
# 노란색, 속도 30  
# 파랑색, 속도 0  
# 초록색, 속도 0 (디폴트 인수 사용)

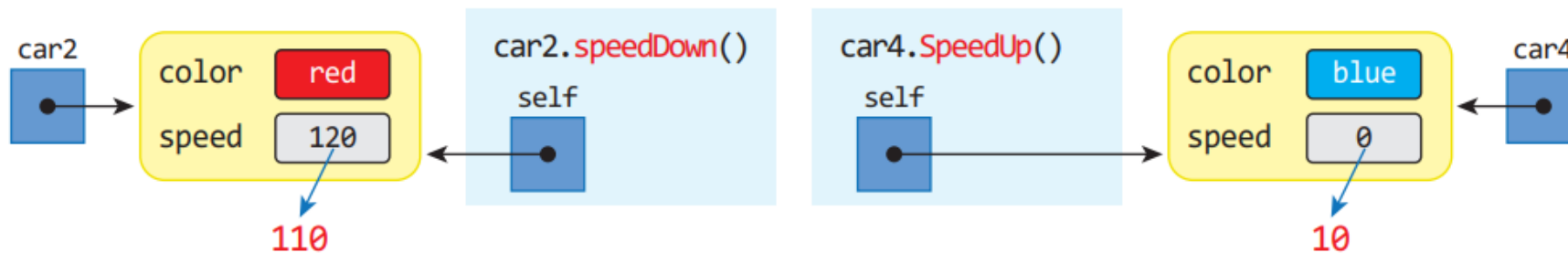


# 멤버함수 구현과 활용

- 멤버 함수

```
class Car :  
    # ...  
    def speedUp(self) : self.speed += 10      # 가속 동작: 속도 10 증가  
    def speedDown(self) : self.speed -= 10    # 감속 동작: 속도 10 감소
```

```
car2.speedDown()      # car2 감속: 속도 10 감소  
car4.speedUp()        # car4 가속: 속도 10 증가
```



```
car3.color = 'purple'      # car3의 색상을 purple로 변경  
car5.speed = 100          # car5의 속도를 100으로 변경
```



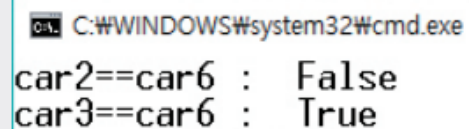


## 2.11 연산자 중복

- 비교 연산자 == 중복

```
def __eq__(self, carB) : return self.color == carB.color
```

```
print("car2==car6 : ", car2==car6)  
print("car3==car6 : ", car3==car6)
```

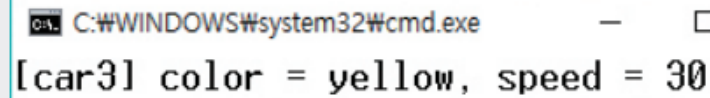


```
C:\WINDOWS\system32\cmd.exe  
car2==car6 : False  
car3==car6 : True
```

- 문자열로 변환 연산자: \_\_str\_\_

```
def __str__(self) :  
    return "color = %s, speed = %d" % (self.color, self.speed)
```

```
print("[car3]", car3)
```



```
C:\WINDOWS\system32\cmd.exe  
[car3] color = yellow, speed = 30
```



# 중복 가능한 연산자들

[표 2.1] 파이썬 클래스의 연산자 중복정의 표    단항 연산자    이항 연산자

Operation	Class Method	Operation	Class Method
str( obj ) len( obj ) item in obj y = obj[idx] obj[idx] = val	__str__( self ) __len__( self ) __contains__(self, item ) __getitem__(self, idx ) __setitem__(self,idx, val)	obj + rhs obj - rhs obj * rhs obj / rhs obj // rhs obj % rhs	__add__( self, rhs ) __sub__( self, rhs ) __mul__( self, rhs ) __truediv__( self, rhs ) __floordiv__( self, rhs ) __mod__( self, rhs )
obj == rhs obj < rhs obj <= rhs obj != rhs obj > rhs obj >= rhs	__eq__( self, rhs ) __lt__( self, rhs ) __le__( self, rhs ) __ne__( self, rhs ) __gt__( self, rhs ) __ge__( self, rhs )	obj ** rhs obj += rhs obj -= rhs obj *= rhs obj /= rhs obj //= rhs obj %= rhs obj **= rhs	__pow__( self, rhs ) __iadd__( self, rhs ) __isub__( self, rhs ) __imul__( self, rhs ) __itruediv__( self, rhs ) __ifloordiv__( self, rhs ) __imod__( self, rhs ) __ipow__( self, rhs )



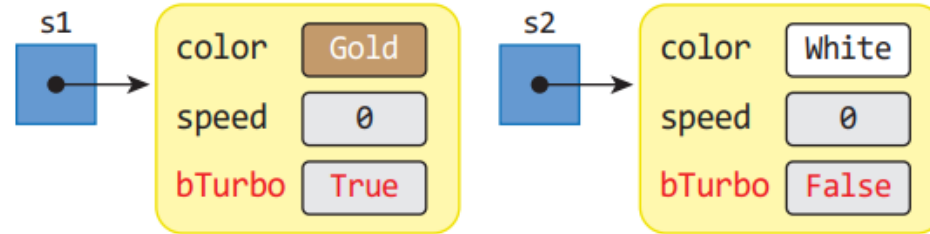
# 2.12 상속

## • 클래스의 상속

```
class SuperCar(Car) :  
    def __init__(self, color, speed = 0, bTurbo = True) :  
        super().__init__(color, speed)    # 부모(Car)클래스의 생성자 호출  
        self.bTurbo = bTurbo             # 터보모드를 위한 변수 생성 및 초기화
```

## • 객체의 생성

```
s1 = SuperCar("Gold", 0, True)  
s2 = SuperCar("White", 0, False)
```



```
s1.speedUp()  
s2.speedUp()  
print("슈퍼카1:", s1)  
print("슈퍼카2:", s2)
```

```
C:\WINDOWS\system32\cmd.exe  
슈퍼카1: color = Gold, speed = 10  
슈퍼카2: color = White, speed = 10
```



# 재정의(Overriding)

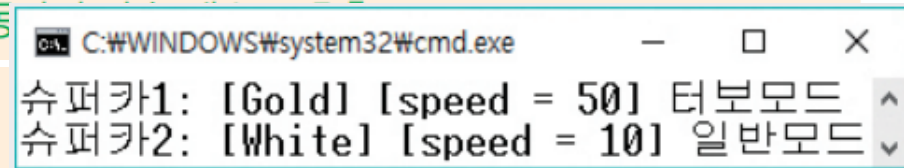
- 메소드의 재정의

```
def setTurbo(self, bTurbo = True) :    # 터보모드를 On/Off하는 메소드
    self.bTurbo = bTurbo
```

```
def speedUp(self) :                    # SuperCar의 SpeedUp. 메소드 재정의
    if self.bTurbo :                  # 터보 모드이면
        self.speed += 50              # 속도가 급속히(50) 증가됨
    else :                             # 터보 모드가 아니면
        super().speedUp()             # 일반 자동차
```

```
def __str__(self) :
    if self.bTurbo :
        return "[%s] [speed = %d] 터보모드" % (self.color, self.speed)
    else :
        return "[%s] [speed = %d] 일반모드" % (self.color, self.speed)
```

```
s1.speedUp()
s2.speedUp()
print("슈퍼카1:", s1)
print("슈퍼카2:", s2)
```



```
C:\WINDOWS\system32\cmd.exe
슈퍼카1: [Gold] [speed = 50] 터보모드
슈퍼카2: [White] [speed = 10] 일반모드
```



# 파이썬 정리

기능	내용	위치
파일 입출력	<ul style="list-style-type: none"> <li>라인 편집기에서 파일 입출력 기능</li> <li>소스파일의 괄호검사</li> </ul>	3.4절 4.3절
슬라이싱 기능	<ul style="list-style-type: none"> <li>스택 내용의 역순출력</li> <li>원형큐의 내용 출력 함수 display()</li> <li>병합정렬의 merge() 함수</li> </ul>	4.2절 5.2절 12.4절
클래스 상속	<ul style="list-style-type: none"> <li>원형큐 클래스를 상속한 원형덱 클래스의 구현</li> <li>이진탐색트리를 이용한 맵 클래스를 상속하여 AVL트리를 이용한 맵 클래스를 구현 (멤버함수 재정의 포함)</li> </ul>	5.5절 9.4절
연산자 중복	<ul style="list-style-type: none"> <li>집합클래스에서 원소의 비교</li> <li>맵의 응용에서 엔트리를 문자열로 변환</li> </ul>	7.3절 7.7절
딕셔너리와 집합 사용 예	<ul style="list-style-type: none"> <li>나의 단어장에서 딕셔너리를 사용한 구현</li> <li>딕셔너리와 집합을 이용한 그래프 표현</li> <li>그래프의 탐색, 신장트리, 연결성분 검사 문제에서 그래프 표현 방법으로 사용</li> </ul>	7.7절 10.2절 10장
람다 함수	<ul style="list-style-type: none"> <li>Kruskal의 최소비용 신장트리 알고리즘에서 간선 리스트를 가중치의 내림차순으로 정렬</li> </ul>	11.3절
모듈 사용	<ul style="list-style-type: none"> <li>time 모듈: 알고리즘 시간 측정</li> <li>queue 모듈: 큐의 응용 기수정렬</li> <li>heapq 모듈: 힙 트리</li> <li>collections 모듈: 그래프 너비우선탐색에서 deque 클래스를 큐로 사용</li> <li>random 모듈: 난수 발생(random.randint())</li> <li>sys 모듈: 정수 무한대(sys.maxsize) 사용</li> <li>copy 모듈: 깊은 복사(copy.deepcopy) 사용</li> </ul>	1.3절 5.3절 12.7절 8.5절 10.3절 12.7절 11.3절 11.4절

