

# 자료구조 & 문자열, 리스트 함수

---

Seolyoung Jeong, Ph.D.

경북대학교 IT대학 컴퓨터학부

# 리스트

# 리스트 생성 및 추가

## ◆ aa = [0,0,0,0]

- 정수형 숫자 4개를 가지는 리스트 aa  
(이미 몇 개로 이루어지는지 알고 있을 때)

## ◆ 비어있는 리스트(aa = [ ])를 만들고, 필요한 만큼 추가

- ‘**리스트이름.append(값)**’ 함수로 리스트에 값 하나씩 추가
- 순차적으로 쌓여감 ([0] 자리부터~)

```
# 빈 리스트
aa = []


# 리스트 하나씩 추가
aa.append(1)    # append(추가할 값)
aa.append(2)
aa.append(3)
aa.append(4)

# 리스트 출력
print(aa)
```

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

# 리스트 생성 및 추가

- ◆ 0~99까지 숫자 리스트를 만들고, 합계 계산
  - 리스트 aa, 카운트변수 i, 합계 변수 sum

```
aa = []  
sum = 0  
  
## 리스트 생성 전 길이  
print("생성 전 길이 => %d" % len(aa))  
  
for i in range(0, 100, 1): # i+1 (0~99)  
      
    sum += aa[i]  
  
## 리스트 추가 후 길이  
print("추가 후 길이 => %d" % len(aa))  
  
print("합계 => %d" % sum)
```

# 역순 리스트 생성 및 추가

```
aa = []      # 원본 리스트
bb = []      # 역순 리스트
num = 0      # 리스트 개수

## 리스트 aa <- 0~199 사이의 짝수
for i in range(100):
    aa.append(i)

## 리스트 개수
num = len(aa)

## 리스트 bb <- aa 역순 (aa[num] ~ aa[0])
for i in range(100):
    bb.append(aa[i])

## 결과 출력
print("aa[0]: %d ~ aa[99]: %d" % (aa[0], aa[99]))
print("bb[0]: %d ~ bb[99]: %d" % (bb[0], bb[99]))
```

```
>>>
aa[0]: 0 ~ aa[99]: 198
bb[0]: 198 ~ bb[99]: 0
>>>
```

# 리스트 생성과 초기화

- ◆ 리스트를 구성하는 원소들은 정수, 실수, 문자, 변수 등
- ◆ 하나의 리스트에 여러가지 자료형 추가 가능

```
>>> var1 = 1
>>> lst1 = [1,2]
>>> lst2 = [var1, 17, 175.3, True, lst1]
>>> lst2
[1, 17, 175.3, True, [1, 2]]
```

# 리스트 생성과 초기화

## ◆ 문자를 이용하여 리스트 생성

```
>>> ss = "abcde"
>>> list(ss)
['a', 'b', 'c', 'd', 'e']
>>>
```

## ◆ range를 사용하여 리스트 생성

- 파이선의 세 가지 기본 시퀀스형 : list, tuple, range
- 숫자의 불변 시퀀스. for 루프에서 특정 횟수만큼 반복하는데 흔히 사용

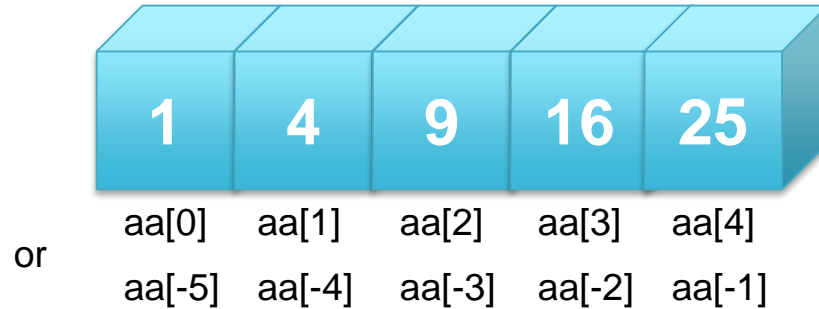
```
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> list(range(0,10,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

```
class range(stop)
class range(start, stop[, step])
```

- 파이썬 2.7 버전의 경우 range(10)의 결과는 리스트 [0,1,2,3,4,5,6,7,8,9]

# 리스트 값 사용

- ◆ 리스트의 **인덱스 값**을 이용하여 출력 및 사용 가능

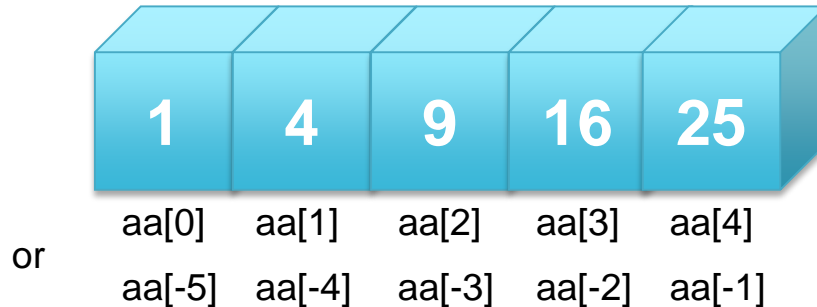


```
>>> aa = [1,4,9,16,25]
>>> aa
[1, 4, 9, 16, 25]
>>> aa[0]
1
>>> aa[-1]
25
>>> aa[-2]
16
>>>
>>> |
```



# 리스트 값 사용

- ◆ 리스트이름[시작번호:끝번호+1]'로 지정하면 리스트의 모든 값이 나옴



- ◆ 기호 **:(콜론)** 을 이용하여 **특정 구간** 지정 가능 (slice)

```
>>> aa[0:3]  
[1, 4, 9]  
>>> aa[2:4]  
[9, 16]
```

- ◆ 끝 인덱스는 출력에 포함되지 않음.  
그러므로, 출력하고 싶은 **끝번호+1**로 지정해주어야 함

# 리스트 값 사용

## ◆ 콜론의 앞이나 뒤 숫자 생략도 가능



or

aa[0]	aa[1]	aa[2]	aa[3]	aa[4]
aa[-5]	aa[-4]	aa[-3]	aa[-2]	aa[-1]

```
>>> aa = [1,4,9,16,25]
>>> aa[:2]
[1, 4]
>>> aa[2:]
[9, 16, 25]
>>> aa[-2:]
[16, 25]
>>> aa[:-2]
[1, 4, 9]
>>> aa[:]
[1, 4, 9, 16, 25]
>>>
```

# 리스트 값 사용

- ◆ 리스트 범위에서 step 값을 주어 그 값만큼 건너뛰어 가져옴
- ◆ list [시작:끝:스텝]

```
>>> aa = [1,4,9,16,25]
>>> aa[0:5:2]
[1, 9, 25]
>>>
>>>
>>> aa[:5:2]
[1, 9, 25]
>>> aa[0::2]
[1, 9, 25]
>>> aa[::2]
[1, 9, 25]
>>> aa[::]
[1, 4, 9, 16, 25]
```

- ◆ 음수로 감소

```
>>>
>>> aa[5:1:-1]
[25, 16, 9]
>>> aa[::-1]
[25, 16, 9, 4, 1]
>>>
```

# 리스트 값 연산

## ◆ 리스트끼리 더하기, 곱하기 연산도 가능

```
>>>  
>>>  
>>>  
>>> aa = [1,2,3]  
>>> bb = [4,5,6]  
>>> aa+bb  
[1, 2, 3, 4, 5, 6]  
>>> aa*3  
[1, 2, 3, 1, 2, 3, 1, 2, 3]  
>>>  
>>>
```

# 리스트 값 추가

## ◆ append(값) 함수

```
>>>  
>>> aa  
[1, 4, 9, 16, 25]  
>>> aa.append(36)  
>>> aa  
[1, 4, 9, 16, 25, 36]  
>>>
```

## ◆ insert(위치, 값) 함수

- 해당 위치 인덱스에 값 삽입

```
>>>  
>>> aa  
[1, 4, 9, 16, 25, 36]  
>>> aa.insert(2, 4.5)  
>>> aa  
[1, 4, 4.5, 9, 16, 25, 36]  
>>>
```

# 리스트 값 변경

## ◆ 한 개 값 변경

```
>>> aa = [1,5,9,16,25]
>>> aa[1] = 4
>>> aa
[1, 4, 9, 16, 25]
```

## ◆ 두 개 값 변경

```
>>> aa = [1,5,10,16,25]
>>> aa[1:3] = [4, 9]
>>> aa
[1, 4, 9, 16, 25]
```

- aa[1], aa[2] 의 값 4, 9 → 400, 401로 변경 원할 시 (aa[1:3]=[400,401])
- (오류) aa[1:2] = [400, 401]로 수행 (401은 세 번째 위치에 새로 추가)
- aa[1:2] → aa[1], 즉 aa[1] 자리에 새로운 리스트 [400, 401] 추가

```
>>> aa = [1,4,9,16,25]
>>> aa[1:2] = [400, 401]
>>> aa
[1, 400, 401, 9, 16, 25]
```

# 슬라이스 값에 따른 차이

```
>>> aa = [1,2,3,4,5]
>>> aa[1:3] = [2.5, 3.5]
>>> aa
[1, 2.5, 3.5, 4, 5]
>>>
>>> aa = [1,2,3,4,5]
>>> aa[1:2] = [2.5, 3.5]
>>> aa
[1, 2.5, 3.5, 3, 4, 5]
>>>
>>> aa = [1,2,3,4,5]
>>> aa[1:1] = [2.5, 3.5]
>>> aa
[1, 2.5, 3.5, 2, 3, 4, 5]
>>>
```

```
>>>
>>> aa = [1,2,3,4,5]
>>> aa
[1, 2, 3, 4, 5]
>>> aa[1:3]
[2, 3]
>>> aa[1:2]
[2]
>>> aa[1:1]
[]
>>>
```

# 리스트 값 삭제

- ◆ **del()** 함수를 사용하여 aa[1] 한개 값을 삭제하는 방법

```
>>> aa = [1,4,9,16,25]
>>> del (aa[1])
>>> aa
[1, 9, 16, 25]
```

- ◆ 여러 개의 항목을 삭제하려면 ‘aa[시작:끝+1]=[ ]’ 문장으로 설정

```
>>> aa = [1,4,9,16,25]
>>> aa[1:4] = []
>>> aa
[1, 25]
```



# 리스트 원소 갯수

## ◆ len() 함수 사용

```
>>> aa = [1,4,9,16,25]
>>> len(aa)
5
>>> aa.append(36)
>>> aa
[1, 4, 9, 16, 25, 36]
>>> len(aa)
6
```

# 리스트 접근 예 (참고)

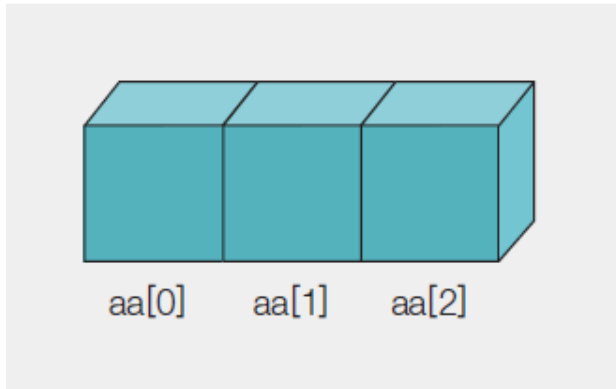
```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> len(letters)
7
>>>
>>> #replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>>
>>> #now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> len(letters)
4
>>>
>>> #clear the list
>>> letters[:] = []
>>> letters
[]
>>> len(letters)
0
>>> |
```

## 2차원 리스트

# 2차원 리스트

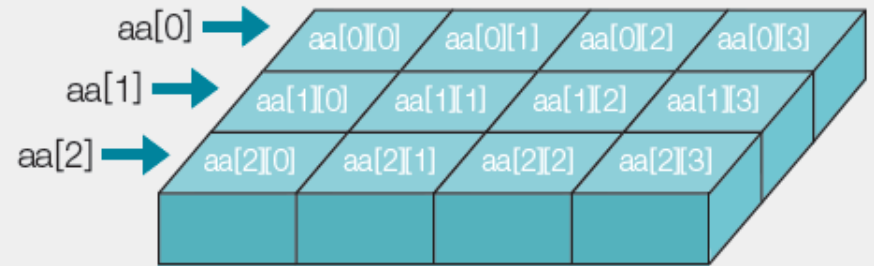
- ◆ 1차원 리스트를 여러 개 연결한 것, 두 개의 첨자([ ][ ])를 사용하는 리스트

- ◆ 1차원 리스트



```
aa = [10, 20, 30]
```

- ◆ 2차원 리스트



전체 리스트 이름 : aa

```
aa = [ [ 1, 2, 3, 4] ,  
       [ 5, 6, 7, 8] ,  
       [ 9, 10, 11, 12] ]
```

# 2차원 리스트

```
>>> a = ['a', 'b', 'c']
>>> b = [1, 2, 3]
>>> x = [a, b]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>>
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
>>> x[1]
[1, 2, 3]
>>> x[1][1]
2
```

# 2차원 리스트

- ◆ 리스트(list\_2)에 리스트(list\_1) 삽입
- ◆ **list\_2.append(list\_1)**

```
list_1 = []  
list_2 = []  
  
list_1 = [1,1,1]  
list_2.append(list_1)  
  
list_1 = [2,2,2]  
list_2.append(list_1)  
  
list_1 = [3,3,3]  
list_2.append(list_1)  
  
print("list 2: ", list_2)
```

```
>>>  
list 2:  [[1, 1, 1], [2, 2, 2], [3, 3, 3]]  
>>>
```

# 2차원 리스트 생성 및 출력 예

- ◆ 1씩 더해가며 4개의 숫자로 3개의 1차원 리스트 생성
  - `list_1(0)`: [1, 2, 3, 4]
  - `list_1(1)`: [5, 6, 7, 8]
  - `list_1(2)`: [9, 10, 11, 12]
- ◆ 1차원 리스트로 2차원 리스트 생성
  - `list_2`: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

---

튜플



# 리스트 vs. 튜플

- ◆ 리스트는 대괄호[ ]로 생성하고 튜플은 괄호( )로 생성
- ◆ 튜플은 값을 수정할 수 없으며 읽기만 가능
- ◆ 읽기 전용의 자료를 저장할 때 사용  
→ 의도치 않게 원소의 값이 변경되는 일을 사전에 막을 수 있음

```
>>> week1 = ('SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')
>>> week2 = 'sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat'
>>> week1
('SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')
>>> week2
('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')
```

- ◆ 생성 시 괄호( ) 생략 가능

# 리스트 vs. 튜플

- ◆ 튜플은 읽기 전용이므로 다음 코드는 모두 오류 발생

```
>>> week  
('SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')  
>>> week[1] = 'mon'  
Traceback (most recent call last):  
  File "<pyshell#300>", line 1, in <module>  
    week[1] = 'mon'  
TypeError: 'tuple' object does not support item assignment
```

```
>>> week.append('test')  
Traceback (most recent call last):  
  File "<pyshell#305>", line 1, in <module>  
    week.append('test')  
AttributeError: 'tuple' object has no attribute 'append'
```

```
>>> del(week[0])  
Traceback (most recent call last):  
  File "<pyshell#309>", line 1, in <module>  
    del(week[0])  
TypeError: 'tuple' object doesn't support item deletion
```

# 리스트 vs. 튜플

- ◆ 튜플 자체(전체)는 `del()` 함수로 지울 수 있음

```
>>> week
('SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')
>>> del(week)
>>> week
Traceback (most recent call last):
  File "<pyshell#333>", line 1, in <module>
    week
NameError: name 'week' is not defined
```

# 튜플 사용

- ◆ 리스트와 같이 인덱스 번호를 사용하여 원소에 접근 가능
- ◆ 리스트에서 수행하던 연산 사용 가능

```
>>> week = ('SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')
>>> day1 = week[0]
>>> day2 = week[5]
>>> print(day1, day2)
SUN FRI
```

- ◆ 튜플 범위에 접근 '콜론(시작번호:끝번호+1)'

```
>>> week[1:3]
('MON', 'TUE')
>>> week[1:]
('MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT')
>>> week[:3]
('SUN', 'MON', 'TUE')
```

# 튜플 사용

## ◆ 더하기 및 곱하기 연산

```
>>> tt1 = (1,2,3)
>>> tt2 = (4,5,6)
>>> tt1 + tt2
(1, 2, 3, 4, 5, 6)
>>> tt1*3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

# 튜플과 리스트 상호 변환

- ◆ 원본 자체가 변경되는 것이 아니라 반환값이 변경.  
즉, 반환값을 다른 리스트나 튜플에 저장함으로써 활용 가능
- ◆ 사용방법 : `list(tuple_object)`, `tuple(list_object)`

```
>>> ll = [1,2,3]
>>> tt = (4,5,6)
>>> print(type(ll), type(tt))
<class 'list'> <class 'tuple'>
>>> ll_to_tt = tuple(ll)
>>> tt_to_ll = list(tt)
>>> print(type(ll_to_tt), type(tt_to_ll))
<class 'tuple'> <class 'list'>
>>>
>>> ll
[1, 2, 3]
>>> tt
(4, 5, 6)
>>> ll_to_tt
(1, 2, 3)
>>> tt_to_ll
[4, 5, 6]
```

# 일괄적으로 원소값 대입

- ◆ 여러 원소 값들을 일괄적으로 대입할 수 있음

```
>>> tt = (1,2,3,4,5)
>>> (n1, n2, n3, n4, n5) = tt
>>> n3
3
>>> (a1, a2) = tt
Traceback (most recent call last):
  File "<pyshell#494>", line 1, in <module>
    (a1, a2) = tt
ValueError: too many values to unpack (expected 2)
>>>
>>> b1, b2, b3, b4, b5 = tt
>>> b4
4
```

- ◆ 튜플 원소 개수 구하기 : `len(튜플)`
- ◆ 문자열 단어 수 구하기 : `len(문자열)`
  - 문자열은 변경이 불가능한 자료형 (튜플과 유사)

# zip() 함수

- ◆ 동일한 개수로 이루어진 자료형을 튜플로 묶어줌

```
>>>  
>>>  
>>> lst1 = [1,2,3,4,5]  
>>> lst2 = list("abcde")  
>>> lst2  
['a', 'b', 'c', 'd', 'e']  
>>>  
>>> list(zip(lst1, lst2))  
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]  
>>>
```

```
>>>  
>>> tpl1 = (6,7,8,9,10)  
>>> tpl2 = tuple("fghij")  
>>> tpl2  
('f', 'g', 'h', 'i', 'j')  
>>>  
>>> list(zip(tpl1, tpl2))  
[(6, 'f'), (7, 'g'), (8, 'h'), (9, 'i'), (10, 'j')]  
>>>
```



# 시퀀스 자료형(리스트, 튜플, range, 문자열)

## ◆ 특정 값이 있는지 확인 'in'

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> 30 in a
True
>>> 100 in a
False
>>>
>>> a = range(0, 100, 10)
>>> 30 in a
True
>>> 100 in a
False
```

## ◆ 특정 값이 없는지 확인 'not in'

```
>>> 100 not in a
True
>>> 30 not in a
False
```

## ◆ 문자열 확인

```
>>>
>>> "gg" in "eggs"
True
```

# for() 문 ~ in ~ 사용

```
>>> a = [1,2,3,4,5]
>>> lst = [i for i in a]
>>> lst
[1, 2, 3, 4, 5]
>>>
>>> lst = [i for i in a if i%2]
>>> lst
[1, 3, 5]
>>>
>>>
>>> bb = [(1,2), (3,4), (5,6)]
>>> for (i, j) in bb:
    print(i+j)

3
7
11
```

# enumerate() 함수

```
a = [38, 21, 53, 62, 19]
cnt = 1

for val in a:
    print("%d번째 : %d" % (cnt, val))
    cnt += 1
```

## ◆ enumerate(리스트) : 인덱스 번호 함께 추출

```
a = [38, 21, 53, 62, 19]

for cnt, val in enumerate(a):
    print("%d번째 : %d" % (cnt+1, val))
```

## ◆ enumerate(리스트, start=숫자) : 인덱스 시작 번호

```
a = [38, 21, 53, 62, 19]

for cnt, val in enumerate(a, start=1):
    print("%d번째 : %d" % (cnt, val))
```

# 문자열

# 문자열 역시 인덱스 번호를 가짐

```
>>>
>>> aa = [1, 2, 3, 4, 5]
>>> aa[0]
1
>>> aa[1:3]
[2, 3]
>>> aa[3:]
[4, 5]
>>>
>>>
>>> ss = "hello"
>>> ss[0]
'h'
>>> ss[1:3]
'el'
>>> ss[3:]
'lo'
>>>
```

Ln: 58 Col: 4

# 문자열의 연산

- ◆ 문자열 더하기(+)
- ◆ 문자열 곱하기 (\*)
- ◆ 문자열 길이 구하기 : len(문자열)

```
>>>
>>>
>>> ss1 = "hello "
>>> ss2 = "world"
>>>
>>> ss = ss1 + ss2
>>> ss
'hello world'
>>>
>>> ss = ss1*3
>>> ss
'hello hello hello '
>>>
>>> len(ss1)
6
>>>
```

Ln: 58 Col: 4

# 수정 불가능한 자료형

- ◆ 튜플과 유사하게 문자열 역시 수정 불가능한 자료형임

```
>>>  
>>> ss = "HELLO"  
>>> ss[1] = "e"  
Traceback (most recent call last):  
  File "<pyshell#3>", line 1, in <module>  
    ss[1] = "e"  
TypeError: 'str' object does not support item assignment  
>>>
```

# for문을 이용한 문자열 생성

## ◆ 새로운 문자열 (ss2="XXXXX")



## ◆ 문자 “HELLO” → 새로운 문자 “XXXXX” 생성

- len( ) 함수를 이용하여 문자열 개수 받아오기
- 문자열 개수만큼 for문 수행하며 새로운 문자 생성

```
ss1 = "HELLO"  
ss2 = ""  
  
ss_len = len(ss1)  
  
for i in range(0, ss_len, 1):  
    ss2 += "X"  
  
print(ss2)
```

## ◆ 문자열은 수정 불가능한 자료형이므로, 기존의 문자 수정 불가능 → 문자 새로 생성



# 문자열 함수

# 대/소문자 변환

- ◆ `upper( )` : 소문자 → 대문자
- ◆ `lower( )` : 대문자 → 소문자
- ◆ `swapcase( )` : 대소문자를 상호 변환
- ◆ `title( )` : 각 단어의 제일 앞 글자만 대문자로 변환

```
>>>  
>>> ss = "Python Programming is fun."  
>>> ss.upper()  
'PYTHON PROGRAMMING IS FUN.'  
>>>  
>>> ss.lower()  
'python programming is fun.'  
>>>  
>>> ss.swapcase()  
'pYTHON pROGRAMMING IS FUN.'  
>>>  
>>> ss.title()  
'Python Programming Is Fun.'  
>>>  
>>> |
```

# 문자열 찾기

- ◆ `count(찾을 문자열)` : 찾을 문자열이 몇 개 있는지 갯수를 셈
- ◆ `find(찾을 문자열)` : 찾을 문자열이 몇 번째 위치하는지 찾음  
`find('찾을 문자열', 시작위치)` 함수 : 시작위치부터 문자열을 찾음
- ◆ `rfind(찾을 문자열)` : 오른쪽부터 찾음
- ◆ `index(찾을 문자열)` : `find()` 함수와 동일한 용도, 찾을 문자열이 없다면 오류가 발생
- ◆ `startswith(문자열)` : 문자열로 시작하면 `True`를, 그렇지 않으면 `False`를 반환
- ◆ `endswith(문자열)` : 문자열로 끝나면 `True`를 반환

```
>>> ss = "Python Programming is fun."
>>> ss.count("is")
1
>>> ss.find("is")
19
>>> ss.find("are")
-1
>>> ss.index("are")
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    ss.index("are")
ValueError: substring not found
```

# 문자열 함수

- ◆ `strip()` : 좌우 공백 제거
- ◆ `lstrip()` : 왼쪽 공백을 제거
- ◆ `rstrip()` : 오른쪽 공백을 제거
  - 단, 문자열中间的 공백은 제거되지 않음

```
>>>
>>>
>>> ss = "  H  E  L  L  O  "
>>>
>>> ss.strip()
'H  E  L  L  O'
>>>
>>> ss.rstrip()
'  H  E  L  L  O'
>>>
>>> ss.lstrip()
'H  E  L  L  O  '
>>>
```

# 문자열 함수

## ◆ strip(문자열) : 지정 문자열 삭제

```
>>>
>>> ss = "---h--e--l--l--o-----"
>>>
>>> ss.strip('-')
'h--e--l--l--o'
>>>
>>> ss = "<<< h < e < l > l > o >>>"
>>>
>>> ss.strip('<>')
'h < e < l > l > o '
>>>
>>> ss.lstrip('<>')
'h < e < l > l > o >>>'
>>>
>>> ss.rstrip('<>')
'<<< h < e < l > l > o '
>>>
```

# 문자열 함수 활용

## ◆ 중간 공백까지 제거하는 프로그램

```
in_str = "    Python    Programming    "  
out_str = ""  
  
for i in range(0, len(in_str)) :  
    if in_str[i] != " "  
        out_str += in_str[i]  
  
print("입력 문자열 : " + '[' + in_str + ']')  
print("공백 제거 => " + '[' + out_str + ']')
```

```
입력 문자열 : [    Python    Programming    ]  
공백 제거 => [PythonProgramming]
```

# 문자열 변경

- ◆ `replace ('기존 문자열', '새 문자열' )` : 문자열 변경

```
>>> ss = "파이썬 Programming"
>>> ss.replace("파이썬", "Python")
'Python Programming'
>>> ss.replace("바이썬", "Python")
'파이썬 Programming'
>>>
```

# 내가 만드는 문자열 함수

- ◆ 문자열을 입력 받아 's' 를 '\$' 로 변경하여 출력

```
ss = input("문자열 입력 : ")

print("출력 문자열 => ", end="")

for i in range(len(ss)):
    if ss[i] == 's':
        print("$", end="")
    else:
        print(ss[i], end="")
```

```
>>>
문자열 입력 : The sky is so nice
출력 문자열 => The $ky i$ $o nice
>>>
```



# 문자열 정렬 및 채우기

- ◆ `center(숫자)` : 숫자만큼 전체 자릿수를 잡은 후, 문자열을 가운데 배치
- ◆ `ljust(숫자)` : 숫자만큼 전체 자릿수를 잡은 후, 왼쪽에 붙여 출력
- ◆ `rjust(숫자)` : 숫자만큼 전체 자릿수를 잡은 후, 오른쪽에 붙여 출력
- ◆ `zfill(숫자)` : 오른쪽으로 붙여 쓰고 왼쪽 빈 공간은 0으로 채움

```
>>>
>>> ss = "Python"
>>>
>>> ss.center(10)
' Python '
>>> ss.center(10, '-')
'--Python--'
>>> ss.ljust(10)
'Python   '
>>> ss.rjust(10)
'    Python'
>>> ss.zfill(10)
'0000Python'
>>>
```

# 문자열 검사

- ◆ `isdigit()` : 전체가 숫자로만 구성되어 있는가?
- ◆ `isalpha()` : 전체가 글자(한글/영어)로만 구성되어 있는가?
- ◆ `isalnum()` : 전체가 글자와 숫자가 섞여서 구성되어 있는가?
- ◆ `islower()` : 전체가 소문자로만 구성되어 있는가?
- ◆ `isupper()` : 전체가 대문자로만 구성되어 있는가?
- ◆ `isspace()` : 전체가 공백문자로만 구성되어 있는가?

```
>>> "1234".isdigit()
True
>>> "abcd".isalpha()
True
>>> "abc123".isalnum()
True
>>> "abcd".isupper()
False
>>> "ABCD".islower()
False
>>> " ".isspace()
True
```

# 문자열 분리

- ◆ `split()` : 문자열을 공백으로 분리해서 리스트를 반환
- ◆ `split(지정 문자열)` : 문자열을 '지정 문자열'로 분리해서 리스트를 반환
- ◆ `splitlines()` : 행 단위로 분리

```
>>> ss = "Python Programming is very fun."
>>> ss.split()
['Python', 'Programming', 'is', 'very', 'fun.']
>>>
>>> ss = "one:two:three"
>>> ss.split(':')
['one', 'two', 'three']
>>>
>>> ss = "하나\n둘\n셋"
>>> ss.splitlines()
['하나', '둘', '셋']
```

참고) 문자열 리스트 연결 : '구분자 문자열'.join(리스트)

```
>>> ' '.join(['a', 'b', 'c'])
'a b c'
>>> '-'.join(['a', 'b', 'c'])
'a-b-c'
```

# 입력값 분리

## ◆ input() 함수로 입력받은 문자열 분리

- 입력받은 문자열 값을 '공백'을 기준으로 분리
- 변수에 차례대로 저장

```
a, b = input("문자열 두 개를 입력하세요: ").split()

print(a)
print(b)
```

```
문자열 두 개를 입력하세요: hello python
hello
python
>>>
```

# 두 숫자의 합 구하기

- ◆ 숫자 두 개를 입력받아서 두 숫자의 합 구하기

```
a, b = input("숫자 두 개를 입력하세요: ").split()  
  
print(a + b)
```

```
숫자 두 개를 입력하세요: 100 200  
100200  
>>> |
```

- ◆ input에서 입력받은 값은 문자열
- ◆ 문자열은 분리해도 문자열
- ◆ 문자열 결합 '+'

# 두 숫자의 합 구하기

## ◆ 변수 a, b를 정수로 변환

```
a, b = input("숫자 두 개를 입력하세요: ").split()

a = int(a)
b = int(b)

print(a + b)
```

```
숫자 두 개를 입력하세요: 100 200
300
>>> |
```

# map을 사용하여 정수로 변환

◆ 문자열 → 분리 → 정수 변환 (계속 반복)

◆ map (함수, 반복데이터)

- 반복 데이터의 각 요소에 대해 함수 수행된 결과를 묶어서 반환
- 결과값 데이터형은 리스트 아님
- 즉, 리스트로 변환 필요

```
>>>  
>>> list(map(int, "12345"))  
[1, 2, 3, 4, 5]  
>>>  
>>>
```

# map을 사용하여 정수로 변환

- ◆ `map()` : `int()` 정수변환 함수, 문자열
- ◆ 문자열: `input().split()`

```
a, b = map(int, input("숫자 두 개를 입력하세요: ").split())  
print(a + b)
```

```
숫자 두 개를 입력하세요: 100 200  
300  
>>>
```



# 입력받은 값을 특정문자를 기준으로 분리하기

- ◆ 공백이 아닌 다른 문자로 분리
- ◆ 예) 콤마(,)로 분리

```
a, b = map(int, input("숫자 두 개를 입력하세요: ").split(','))  
print(a + b)
```

```
숫자 두 개를 입력하세요: 100,200  
300  
>>> |
```

# (참고) print 출력 파라미터

- ◆ 기본: ,(콤마)로 구분된 여러 개 문자들을 공백으로 합쳐서 출력

```
>>> print(1, 2, 3)
1 2 3
>>> print("hello", "world")
hello world
>>>
```

- ◆ ‘sep’ 파라미터

- 값 사이에 공백이 아닌 다른 문자로 합쳐서 출력

```
>>> print(1, 2, 3, sep=', ')
1, 2, 3
>>> print(4, 5, 6, sep=',')
4,5,6
>>> print('hello', 'world', sep='')
helloworld
>>> print(1024, 768, sep='x')
1024x768
```

# print 출력 파라미터

## ◆ 줄바꿈(new line) 활용

```
>>> print(1, 2, 3)
1 2 3
>>> print(1, 2, 3, sep='\n')
1
2
3
>>> print("1\n2\n3")
1
2
3
>>>
```

# print 출력 파라미터

## ◆ 'end' 파라미터

- print는 기본적으로 출력하는 값 끝에 '\n'을 붙임
- 출력하는 값 끝에 특정 문자 추가

```
print(1)
print(2)
print(3)

print(1, end="")
print(2, end="")
print(3)
```

# 리스트 함수

# 리스트 조작 함수

함수	설명	사용법
append()	리스트 제일 뒤에 항목을 추가한다.	리스트이름.append(값)
pop()	리스트 제일 뒤의 항목을 빼내고, 빼낸 항목은 삭제한다.	리스트이름.pop()
sort()	리스트의 항목을 정렬한다.	리스트이름.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트이름.reverse()
index()	지정한 값을 찾아서 그 위치를 반환한다.	리스트이름.index(찾을 값)
insert()	지정된 위치에 값을 삽입한다.	리스트이름.insert(위치, 값)
remove()	리스트에서 지정한 값을 제거한다. 단 지정한 값이 여러 개일 경우 첫 번째 값만 지운다.	리스트이름.remove(지울 값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 동일한 기능을 한다.	리스트이름.extend(리스트)
count()	리스트에서 찾을 값의 개수를 센다.	리스트이름.count(찾을 값)
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트이름[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트이름)

# 리스트 조작 예

```
>>> myList = [30, 10, 20]
>>> print("현재리스트: %s"%myList)
현재리스트: [30, 10, 20]
>>>
>>> myList.append(40)
>>> print(myList)
[30, 10, 20, 40]
>>>
>>> myList.pop()
40
>>> print(myList)
[30, 10, 20]
>>>
>>> myList.sort()
>>> print(myList)
[10, 20, 30]
>>>
>>> myList.reverse()
>>> print(myList)
[30, 20, 10]
>>>
>>> myList.index(20)
1
>>>
>>> myList.insert(2, 222)
>>> print(myList)
[30, 20, 222, 10]
>>>
>>> myList.remove(222)
>>> print(myList)
[30, 20, 10]
>>>
>>> myList.extend([77, 88, 77])
>>> print(myList)
[30, 20, 10, 77, 88, 77]
>>>
>>> myList.count(77)
2
>>>
```

# 정렬 기능 `sort()` vs. `sorted()`

## ◆ `리스트.sort()`

- 리스트 자체를 정렬

## ◆ `sorted(리스트)`

- 정렬된 결과만 반환

```
myList = [30, 10, 20]

newList = sorted(myList)

print("myList : %s" % myList)
print("newList : %s" % newList)
```

```
myList : [30, 10, 20]
newList : [10, 20, 30]
```



# 가장 작은 수 구하기

- ◆ 먼저, 리스트 a의 첫번째 요소 a[0]를 변수 smallest에 저장
- ◆ 반복문에서 리스트 요소 모두 반복하면서
  - i가 smallest보다 작으면 smallest에 i를 할당

```
a = [38, 21, 53, 62, 19]
smallest = a[0]

for i in a:
    if i < smallest:
        smallest = i

print("minimum value: %d" % smallest)
```

```
minimum value: 19
>>>
>>>
```

# 가장 작은 수 구하기

## ◆ 방법2) 정렬 후 첫번째 요소 가져오기

```
a = [38, 21, 53, 62, 19]
a.sort()
print("minimum value: %d" % a[0])
```

함수 사용 방법 다름~  
주의할 것!

## ◆ 방법3) min() 함수 사용하기

```
>>>
>>> a = [38, 21, 53, 62, 19]
>>> min(a)
19
>>>
```

# 가장 큰 수 구하기

## ◆ 방법1)

```
a = [38, 21, 53, 62, 19]
biggest = a[0]

for i in a:
    if i > biggest:
        biggest = i

print("maximum value: %d" % biggest)
```

```
maximum value: 62
>>>
```

## ◆ 방법2)

```
a = [38, 21, 53, 62, 19]
a.sort(reverse=True)

print("maximum value: %d" % a[0])
```

## ◆ 방법3)

```
>>>
>>> a = [38, 21, 53, 62, 19]
>>> max(a)
62
>>>
```

# 그 외 자료구조

# 자료구조(Data Structure)

- ◆ 데이터의 저장, 검색 등을 효율적으로 관리하는 방법
- ◆ 메모리 사용량 및 실행시간 등을 최소화
- ◆ 파이썬에서 기본적으로 제공하는 자료구조
  - 리스트(List) → 활용 스택(Stack), 큐(Queue)
  - 튜플(Tuple)
  - 딕셔너리(Dictionary)
  - 세트(Set)

# 딕셔너리

# 딕셔너리

- ◆ 사전과 비슷한 방법 사용
- ◆ 사전 : 원하는 단어를 찾기 위해 단어의 알파벳을 순서대로 찾고, 그에 맞는 의미의 값을 찾는 방식
- ◆ 딕셔너리 : 단어의 알파벳인 **키**와 단어의 뜻인 **값**을 하나의 **쌍으로 사용**
- ◆ **중괄호 { }** 로 묶어서 사용
- ◆ 각 원소는 **키-값**을 하나의 쌍으로 구성
- ◆ 인덱스를 사용하지 않고, 키 값을 사용하여 각 원소 구분

딕셔너리변수 = { 키1:값1 , 키2:값2, 키3:값3, ... }

```
>>> dic1 = { 1:'a', 2:'b', 3:'c' }
>>> dic1
{1: 'a', 2: 'b', 3: 'c'}
>>>
>>> dic2 = { 'n1':10, 'n2':20, 'n3':30 }
>>> dic2
{'n1': 10, 'n3': 30, 'n2': 20}
```

# 딕셔너리 생성 및 추가

## ◆ 예시) 학생정보 (학과, 학번, 이름)

```
>>> stu = { '학과': '컴퓨터학부', '학번': 1000, '이름': '김아름' }  
>>> stu  
{ '학과': '컴퓨터학부', '이름': '김아름', '학번': 1000 }
```

## ◆ 생성한 학생정보에 연락처 추가

```
>>> stu['연락처'] = '010-1111-2222'  
>>> stu  
{ '연락처': '010-1111-2222', '학과': '컴퓨터학부', '이름': '김아름', '학번': 1000 }
```



# 딕셔너리 생성

## ◆ dict() 함수

- 키와 값을 연결할 때
- 리스트, 튜플, 딕셔너리로 딕셔너리 만들 때

## ◆ 방법1) 키=값 형식으로 생성

```
>>>  
>>> man1 = dict(height=170, weight=65, sight=1.5, age=17)  
>>> man1  
{'height': 170, 'weight': 65, 'sight': 1.5, 'age': 17}  
>>>
```

- 주의) 키에 따옴표(“” 또는 “”) 사용 안함

## ◆ 방법2) zip() 함수로 키 리스트+값 리스트 묶어서 생성

```
>>>  
>>> man2 = dict(zip(['height', 'weight', 'sight', 'age'], [170, 65, 1.5, 17]))  
>>> man2  
{'height': 170, 'weight': 65, 'sight': 1.5, 'age': 17}  
>>>
```

# 딕셔너리 생성

## ◆ 방법3) 리스트 안에 (키, 값) 형식 튜플 나열

```
>>>  
>>> man3 = dict([('height',170), ('weight',65), ('sight',1.5), ('age',17)])  
>>> man3  
{'height': 170, 'weight': 65, 'sight': 1.5, 'age': 17}  
>>>
```

## ◆ 방법4) 중괄호{ }로 딕셔너리 생성

```
>>>  
>>> man4 = dict({'height':170, 'weight':65, 'sight':1.5, 'age':17})  
>>> man4  
{'height': 170, 'weight': 65, 'sight': 1.5, 'age': 17}  
>>>
```

# 딕셔너리 사용

- ◆ 이미 있는 키를 사용하면 쌍이 새로 추가되는 것이 아니라 기존의 값이 변경

```
>>> stu['학과'] = '통계학과'
>>> stu
{'연락처': '010-1111-2222', '학과': '통계학과', '이름': '김아름', '학번': 1000}
```

- ◆ ‘del(딕셔너리이름[키])’ 함수를 사용하여 삭제

```
>>> del(stu['연락처'])
>>> stu
{'학과': '통계학과', '이름': '김아름', '학번': 1000}
```

# 딕셔너리 사용

## ◆ 키를 이용하여 값에 접근

```
>>> stu['학번']  
1000  
>>> stu['이름']  
'김아름'  
>>> stu['학과']  
'통계학과'
```

## ◆ '딕셔너리이름.get(키)' 함수

```
>>> stu.get('학과')  
'통계학과'  
>>> stu.get('이름')  
'김아름'  
>>> stu.get('학번')  
1000
```

# 딕셔너리 사용

- ◆ 딕셔너리이름.get(키) - 키가 없을 때 아무것도 반환하지 않음

```
>>> stu['주소']
Traceback (most recent call last):
  File "<pyshell#641>", line 1, in <module>
    stu['주소']
KeyError: '주소'
>>> stu.get('주소')
>>>
```

- ◆ ‘딕셔너리이름.keys( )’ 함수 - 딕셔너리의 모든 키 반환

```
>>> stu.keys()
dict_keys(['학번', '학과', '이름'])
```

- ◆ ‘list(딕셔너리이름.keys( ))’ 함수 - 앞에 dict\_keys 를 빼줌 (리스트 형태)

```
>>> list(stu.keys())
['학번', '학과', '이름']
```

# 딕셔너리 사용

- ◆ ‘딕셔너리이름.values( )’ 함수 - 딕셔너리의 모든 값 반환

```
>>> stu.values()  
dict_values([1000, '통계학과', '김아름'])
```

- ◆ ‘딕셔너리이름.items( )’ 함수 - 튜플 형태 (키,값)

```
>>> stu.items()  
dict_items([('학번', 1000), ('학과', '통계학과'), ('이름', '김아름')])
```

- ◆ 키 in 딕셔너리 : 키가 있으면 True, 없으면 False 반환

```
>>> '이름' in stu  
True  
>>> '주소' in stu  
False
```

# 딕셔너리 사용

## ◆ for문을 활용하여 딕셔너리의 모든 값 출력

```
singer = {}  
  
singer['이름'] = '아이오아이'  
singer['구성원수'] = 11  
singer['데뷔'] = '프로듀스101'  
singer['대표곡'] = '픽미픽미픽미업'  
  
for k in singer.keys():  
    print("%s : %s" % (k, singer[k]))
```

강재원

```
데뷔 : 프로듀스101  
대표곡 : 픽미픽미픽미업  
이름 : 아이오아이  
구성원수 : 11
```

# 세트 (Set)

- ◆ 중복이 없는 비순서적인 집합

- ◆ set 객체 생성

- set 객체 이름 = set(반복 가능한 객체: 리스트, 문자열, range)

```
>>>  
>>> a = set('apple')  
>>> a  
{ 'p', 'l', 'a', 'e' }  
>>>  
>>> b = set(range(5))  
>>> b  
{ 0, 1, 2, 3, 4 }  
>>>  
>>> c = set([1,2,3,4,5])  
>>> c  
{ 1, 2, 3, 4, 5 }  
>>>
```



# 세트 (Set) 사용

## ◆ 수학적 연산 사용 가능 :

```
>>>
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>>
>>> a
{'d', 'a', 'b', 'r', 'c'}
>>>
>>> b
{'l', 'a', 'c', 'z', 'm'}
>>>
>>> a-b
{'d', 'r', 'b'}
>>> a|b
{'m', 'd', 'r', 'a', 'l', 'b', 'c', 'z'}
>>> a&b
{'a', 'c'}
>>> a^b
{'b', 'd', 'r', 'l', 'z', 'm'}
>>>
```

# 세트 (Set) 사용

## ◆ 집합연산과 세트 함수

```
>>>
>>> a = {1,2,3,4}
>>> b = {3,4,5,6}
>>> a | b
{1, 2, 3, 4, 5, 6}
>>> set.union(a,b)
{1, 2, 3, 4, 5, 6}
>>>
>>> a & b
{3, 4}
>>> set.intersection(a,b)
{3, 4}
>>>
>>> a - b
{1, 2}
>>> set.difference(a,b)
{1, 2}
>>>
>>> a ^ b
{1, 2, 5, 6}
>>> set.symmetric_difference(a,b)
{1, 2, 5, 6}
>>>
```

# 세트 (Set) 사용

## ◆ 부분집합, 상위집합 확인

```
>>>  
>>> a = {1,2,3}  
>>> b = {1,2,3,4,5}  
>>>  
>>> a < b  
True  
>>> a.issubset(b)  
True  
>>> b.issuperset(a)  
True  
>>>
```

# 세트 (Set) 사용

## ◆ 여러 자료형 저장 가능하지만, 중복은 허용 안함

- list는 못 넣음. tuple은 저장 가능

```
>>>
>>> s1 = {1, 1, 2, "2", "text", "text", 3}
>>> s1
{1, 2, 3, 'text', '2'}
>>>
>>> s2 = {1, 2, [1,2,3]}
Traceback (most recent call last):
  File "<pyshell#278>", line 1, in <module>
    s2 = {1, 2, [1,2,3]}
TypeError: unhashable type: 'list'
>>>
>>> s3 = {1, 2, (1,2,3)}
>>> s3
{1, 2, (1, 2, 3)}
>>>
```

## ◆ Set에는 순서가 없기 때문에 index 사용 못함

- 값의 유무 확인 가능

```
>>> s1 = {1, 1, 2, "2", "text", "text", 3}
>>> 1 in s1
True
```

# 세트 (Set) 사용

## ◆ 세트 값 추가 및 제거

- 세트.add(값)
- 세트.update(세트)
- 세트.remove(값) : [주의] 값이 없으면 에러 발생
- 세트.discard(값) : 요소가 없으면 그냥 넘어감 (에러 발생 안함)

```
>>> s1.add(6)
>>> s1
{1, 2, 3, 6, 'text', '2'}
>>> s1.update({7,8,9})
>>> s1
{1, 2, 3, 6, 7, 8, 9, 'text', '2'}
>>>
>>> s1.remove(6)
>>> s1
{1, 2, 3, 7, 8, 9, 'text', '2'}
>>> s1.remove(4)
Traceback (most recent call last):
  File "<pyshell#401>", line 1, in <module>
    s1.remove(4)
KeyError: 4
>>> s1.discard(4)
>>>
```

# frozenset

## ◆ 내용을 변경할 수 없는 세트

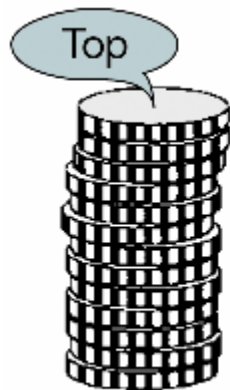
```
>>> a = frozenset(range(10))
>>> a
frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9})
>>> a.add(10)
Traceback (most recent call last):
  File "<pyshell#317>", line 1, in <module>
    a.add(10)
AttributeError: 'frozenset' object has no attribute 'add'
>>>
>>> a |= 10
Traceback (most recent call last):
  File "<pyshell#319>", line 1, in <module>
    a |= 10
TypeError: unsupported operand type(s) for |=: 'frozenset' and 'int'
```

## ◆ 세트 안에 세트 넣을 때 사용 가능

```
>>> a = frozenset(range(10))
>>> b = frozenset(range(3))
>>> c = frozenset({a,b})
>>> c
frozenset({frozenset({0, 1, 2, 3, 4, 5, 6, 7, 8, 9}), frozenset({0, 1, 2})})
```

# 리스트를 활용한 스택 (Stack)

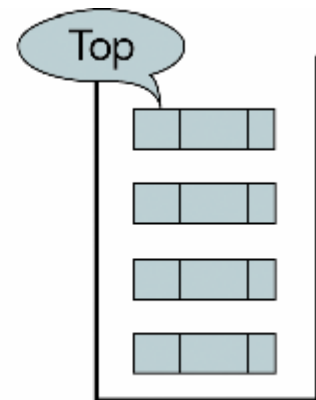
- ◆ 나중에 넣은 데이터를 먼저 꺼내도록 설계된 구조
- ◆ Last-In First-out (LIFO)



Stack of coins

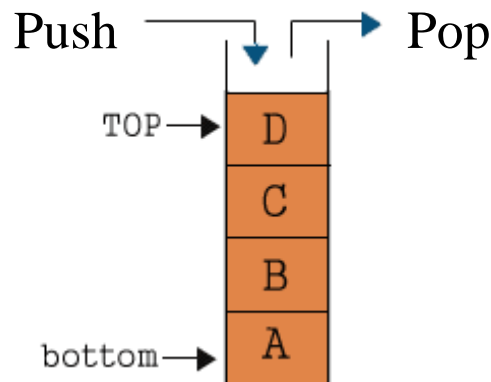


Stack of books



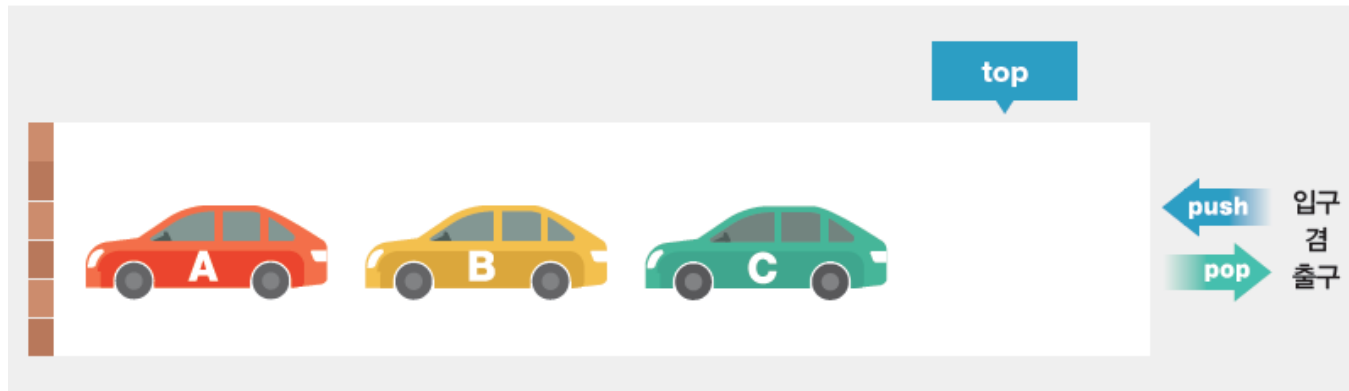
Computer stack

- ◆ **Push (데이터) : 데이터 입력**
  - 파이썬 리스트함수 중 `append()` 사용
- ◆ **Pop ( ) : 데이터 출력**
  - 출력된 데이터는 리스트에서 제거됨



# 한쪽이 막힌 주차장 프로그램

- ◆ 스택은 한쪽 끝이 막힌 자료구조로, 가장 먼저 들어간 것이 가장 나중에 나옴.
- ◆ 자동차가 들어간 순서는  $A \rightarrow B \rightarrow C$ 지만,  
나오는 순서는  $C \rightarrow B \rightarrow A$



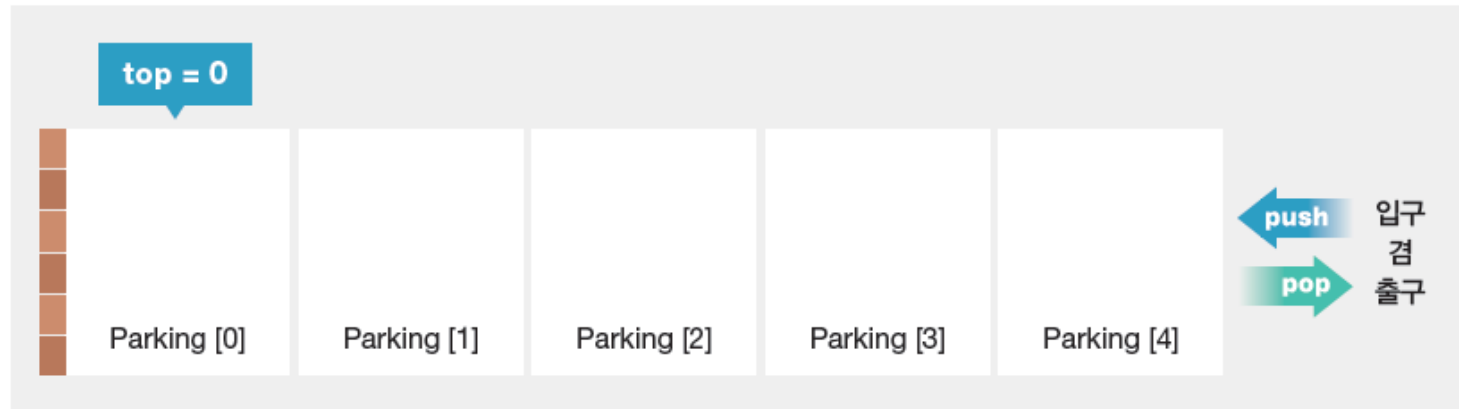
- ◆ LIFO(Last In First Out) 구조라고도 함.
- ◆ 비어있는 위치를 **top**이라 칭함. 만약 자동차 C가 빠져나가면 top은 현재 스택 안에 들어있는 데이터 중 가장 마지막 데이터의 다음 위치를 가리킴.
- ◆ 데이터를 넣는 것을 **푸시(Push)**, 빼는 것을 **팝(Pop)**이라고 함



- ◆ 자동차 5대가 들어갈 수 있고, 한쪽이 막힌 주차장

```
parking=[ "", "", "", "", "" ]  
top=0
```

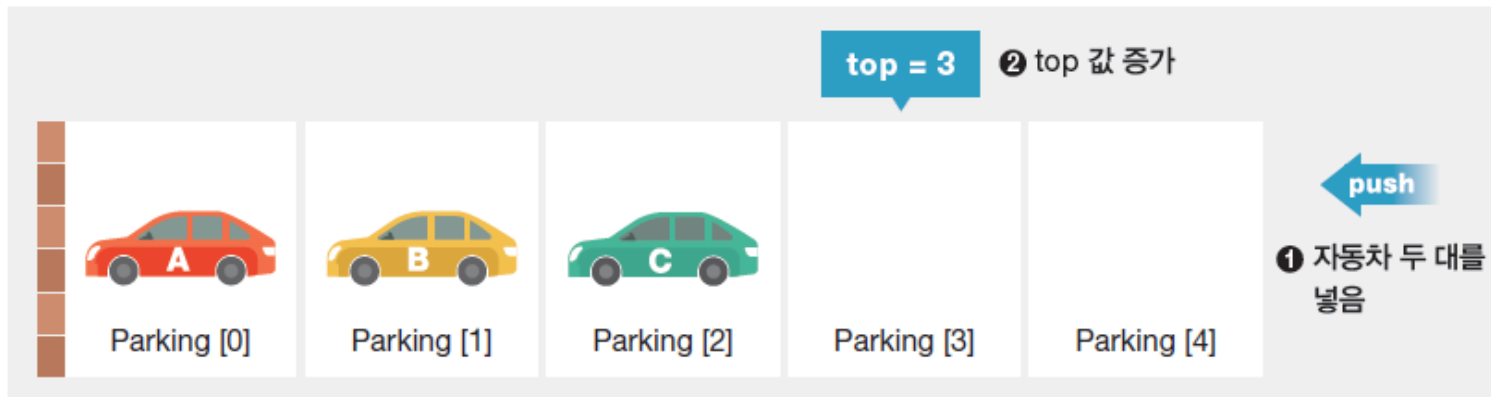
- ◆ 비어 있는 주차장



- ◆ 주차장에 자동차 한 대 넣기



## ◆ 주차장에 자동차 두대 추가로 넣기



## ◆ 자동차 한 대 빼기



# 코드

## 변수 선언 부분

```
parking = [ ]  
carName, outCar = "A", ""  
top = 0  
select = 0
```

## 메인(main) 코드 부분

```
while (select != 3) :  
    select = int(input("<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : "))  
  
    if (select == 1) :    # 메뉴 <1> 자동차 넣기  
        if( top >= 5 ) :    # 최대 5대 들어갈 수 있음  
            print("주차장이 꽉 차서 못들어감")  
        else :  
            parking.append(carName)  
            print(" %s 자동차 들어감. 주차장 상태 ==> %s " % (parking[top], parking))  
            top += 1  
            carName = chr(ord(carName)+1)  
        elif (select == 2) :    # 메뉴 <2> 자동차 빼기  
            if( top <= 0 ) :  
                print("빠져나갈 자동차가 없음")  
            else :  
                outCar = parking.pop()  
                print(" %s 자동차 나감. 주차장 상태 ==> %s " % (outCar, parking))  
                top -= 1  
                carName = chr(ord(carName)-1)  
        elif (select == 3) :    # 메뉴 <3> 끝  
            break;  
        else :  
            print("잘못 입력했네요. 다시 입력하세요.")  
  
print("현재 주차장에 %d 대가 있음" % top)  
print("프로그램을 종료합니다.")
```

# 스택 주차장 코드

## 1. 변수

- 주차장 스택 (parking 리스트)
- 자동차 이름 : A, B, C, ... 알파벳 대문자
- top (주차장 끝 번호, 스택 끝 번호)
- 3가지 선택 : <1> 넣기 <2> 빼기 <3> 끝

## 2. 메뉴가 3이 아닌 동안

1. 메뉴 입력 (정수로 변환)

2. <1>번 선택 경우

① 만약, 5대를 넘어서면 “주차장 꽉 찼음”

② 아니면, parking.append(자동차 이름)

<2>번 선택 경우

① 만약, 0보다 작은 경우(스택 비었음), “자동차 없음”

② 아니면, parking.pop()

<3>번 선택 경우

① 종료 (break)

나머지, 잘못 선택

## 3. 현재 주차장 대수 출력

## 4. 종료

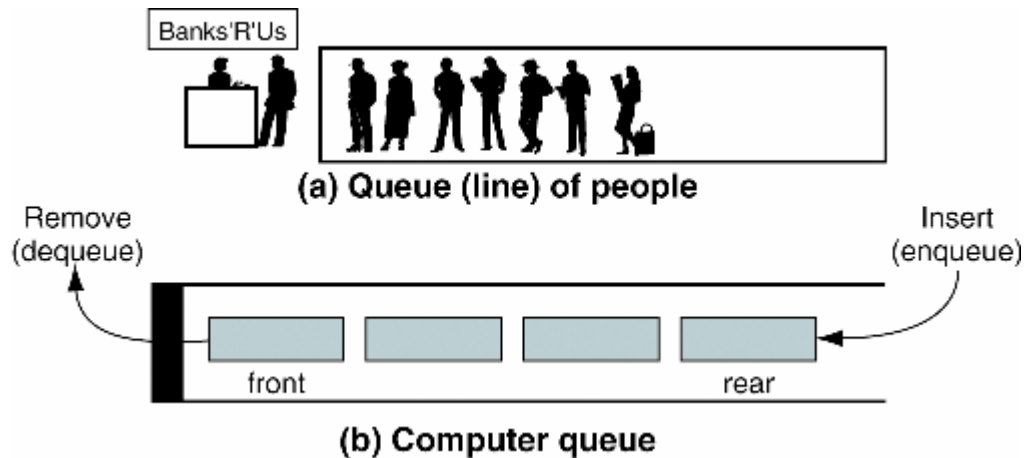
# 스택 주차장 코드

- ♦ ord(알파벳) 함수 : 문자('A') → 아스키코드 숫자(65) 로 변환
- ♦ chr(숫자) 함수 : 아스키코드 숫자(66) → 문자('B')로 변환

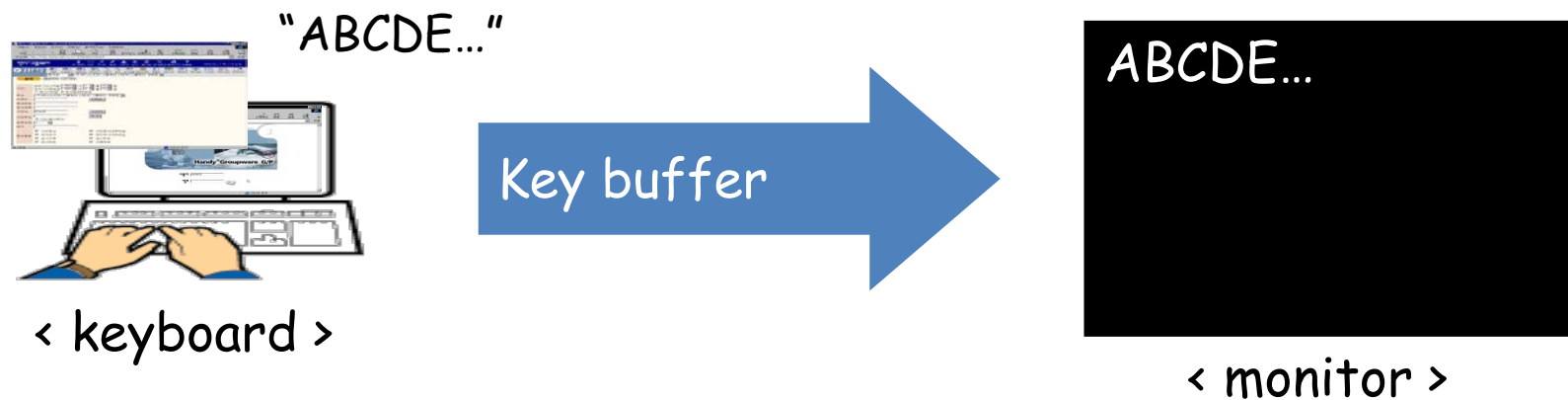
```
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
A 자동차 들어감. 주차장 상태 ==> ['A']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
B 자동차 들어감. 주차장 상태 ==> ['A', 'B']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
C 자동차 들어감. 주차장 상태 ==> ['A', 'B', 'C']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
D 자동차 들어감. 주차장 상태 ==> ['A', 'B', 'C', 'D']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
E 자동차 들어감. 주차장 상태 ==> ['A', 'B', 'C', 'D', 'E']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 2
E 자동차 나감. 주차장 상태 ==> ['A', 'B', 'C', 'D']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 2
D 자동차 나감. 주차장 상태 ==> ['A', 'B', 'C']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 1
D 자동차 들어감. 주차장 상태 ==> ['A', 'B', 'C', 'D']
<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : 3
현재 주차장에 4 대가 있음
프로그램을 종료합니다.
```

# 리스트를 활용한 큐 (Queue)

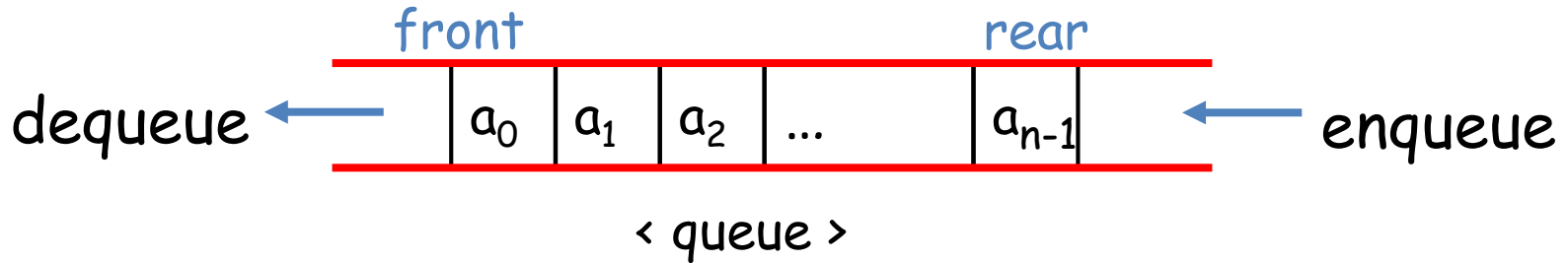
- ◆ 먼저 넣은 데이터를 먼저 꺼내도록 설계된 구조
- ◆ First-In First-out (FIFO)



- ◆ 키보드 사용



# 큐 (Queue)



- ◆ enqueue (데이터) : 데이터 입력
- ◆ dequeue ( ) : 데이터 출력
- ◆ 출력된 데이터는 리스트에서 제거
- ◆ 큐 in 파이썬
  - 데이터 입력 시 → `append( )` 함수 사용
  - 데이터 출력 시 → `pop(0)` 함수 사용
  - 내부적으로 인덱스 번호를 이동시켜 줘야 함
  - 주로 별도로 구현된 모듈 사용

**Any Questions...**  
**Just Ask!**

