

# 알고리즘분석 실습 자료

## 2주차

Photo by [Piotr Guzik](#) on [Unsplash](#)



- 파이썬
  - ✓ 리스트, 함수, set, dictionary
  - ✓ 그래프
- 실습프로그램
  - ✓ 순차검색
  - ✓ 배열의 수 더하기
  - ✓ 교환정렬
  - ✓ 행렬곱셈
  - ✓ 이분검색
  - ✓ 피보나찌 수열 구하기



## 5. 리스트



- 여러 개의 데이터들을 저장하는 자료형
- 하나의 이름을 사용
- 이름과 위치를 나타내는 첨자(인덱스)를 결합하여 변수 표현
- 외형은 배열(array)의 형식이지만, 내부적으로는 연결리스트로 구현되어 있다.
- 다른 프로그램의 배열과 연결리스트의 기능을 복합적으로 가짐



- `a=[5,1, 3, 2, 4, 0]`

```
a=[5,1, 3, 2, 4, 0]
print (a)
print(a[0], a[2])
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
5	1	3	2	4	0

```
[5, 1, 3, 2, 4, 0]
5 3
>>>
```

- `a=['on', 'before', 'from', 'to']`

a[0]	a[1]	a[2]	a[3]
on	before	from	to

- `a=['on', 56, 'after', 123]`

a[0]	a[1]	a[2]	a[3]
on	56	after	123



a[0] a[1] a[2] a[3] a[4] a[5]

0	0	0	0	0	0
---	---	---	---	---	---

- a=[0] \*6 : 6개의 방을 0으로 설정
- a=[0,0,0,0,0,0] 과 동일

```
a=[0]*6  
print(a)
```

```
[0, 0, 0, 0, 0, 0]  
>>>
```

```
a=[0,0,0,0,0,0]  
print(a)
```

```
[0, 0, 0, 0, 0, 0]  
>>>
```



- 연산을 간단히 표시

하나의 변수명에 첨자만 달리하면서 데이터를 접근할 수 있다.

1년의 전체 날 수

```
a=[31,28,31,30,31,30,31,31,30,31,30,31]  
days=0  
for i in range(0,12):  
    days+=a[i]  
print(days)
```

```
365  
>>>
```

days += a[i]

=

days = days+a[i]

✓ +=, -=, \*=, /= 도 동일한 방식



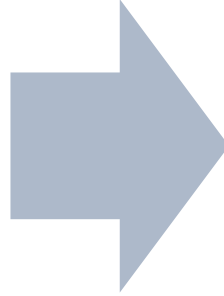


days += 2

days -= 2

days \*= 2

days /= 2



days = days + 2

days = days - 2

days = days \* 2

days = days / 2



리스트의 구성요소를 직접 반복문 for에 불러올 수 있다.

```
season=['spring', 'summer', 'fall', 'winter']  
for s in season:  
    print(s)
```

```
spring  
summer  
fall  
winter  
>>>
```

```
season=['spring','summer', 'fall', 'winter']  
for k in range(0,4):  
    print(season[k])
```





```
data1=['A','B','C','D']  
data2=['A','D','E','F']  
  
for s in data2 :  
    if s not in data1:  
        print(s)
```

```
E  
F  
>>>
```



- 리스트에서 지원하는 함수를 이용하여 다양한 기능을 쉽게 구현할 수 있다.
- 사용형식 - 리스트명.함수명(파라미터) 또는 함수명(리스트명)

- a=[5,1, 3, 2, 4, 0]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
5	1	3	2	4	0

a[1]을 변경



```
>>> a=[5, 1, 3, 2, 4, 0]
```

```
>>> a[1]=100
```

```
>>> a
```

```
[5, 100, 3, 2, 4, 0]
```

정렬



```
>>> a.sort()
```

```
>>> a
```

```
[0, 2, 3, 4, 5, 100]
```

추가



```
>>> a.append(9)
```

```
>>> a
```

```
[0, 2, 3, 4, 5, 100, 9]
```

삭제



```
>>> a.remove(9)
```

```
>>> a
```

```
[0, 2, 3, 4, 5, 100]
```

```
>>>
```



- a=[5,1, 3, 2, 4, 0]

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
5	1	3	2	4	0

2의 개수 ----->

역으로 나열 ----->

제일 우측 값 ----->

인덱스1에 있는 데이터 제거 ----->

```
>>> a=[5,1,3,2,4,0]
>>> a.count(2)
1
>>> a.reverse()
>>> a
[0, 4, 2, 3, 1, 5]
>>> a.pop()
5
>>> a
[0, 4, 2, 3, 1]
>>> a.pop(1)
4
>>> a
[0, 2, 3, 1]
```



- 자료구조에 연결하여 함수를 사용하는 것을 객체지향 (object-oriented)이라고 한다.
- 정의된 기능들은 해당 자료구조에서만 사용



## 다양한 데이터형에 사용할 수 있는 함수

- `a=[5,1, 3, 2, 4, 0]`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>	<code>a[5]</code>
5	1	3	2	4	0

리스트 내의 데이터 개수

최대

최소

```
>>> a=[5,1,3,2,4,0]
>>> len(a)
6
>>> max(a)
5
>>> min(a)
0
```

함수명(변수명)



## 리스트의 함수를 이용한 프로그램 예

- 리스트의 가운데 값 출력하기

```
정렬 -----> a=[4,1,3,9,10]  
a.sort()  
리스트의 크기 -----> n=len(a)  
리스트의 가운데 위치 -----> n=int(n/2)  
리스트의 가운데 값 출력 -----> print(a.pop(n))
```

```
4  
>>>
```

`int(n/2)`:  $n/2$ 의 값을 정수값으로 만든다.



- `[m:n]` : 인덱스가 `m`인 데이터부터 인덱스가 `n-1`인 데이터까지를 표현
- `m`이 없을 경우는 처음부터, `n`이 없을 경우는 끝까지

```
a=[4,1,5,9,10]
h=3
leftHalf=a[:h]
rightHalf=a[h:]
print(leftHalf)
print(rightHalf)
```

```
[4, 1, 5]
[9, 10]
>>>
```

```
a=['a','b','c','d','e']
print(a[1:3])
print(a[:3])
print(a[3:])
```

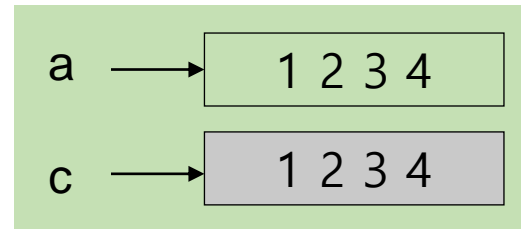
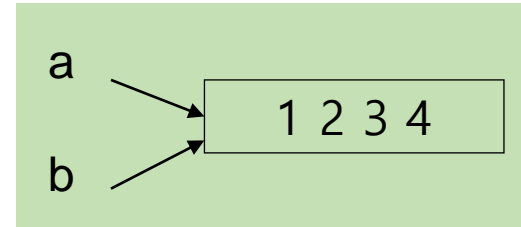
```
['b', 'c']
['a', 'b', 'c']
['d', 'e']
>>>
```





## 리스트는 mutable(값을 바꿀 수 있다)

- `b=a` 는 리스트 `a`를 `b`라고도 나타낼 수 있다.
- `c=a[:]` 는 리스트 `a`를 리스트 `c`에 복사한다.



```
a=[1,2,3,4]
b=a
c=a[:]
print(a,b,c)
a.append(5)
print(a,b,c)
```

a	b	c
[1, 2, 3, 4]	[1, 2, 3, 4]	[1, 2, 3, 4]
[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]	[1, 2, 3, 4]

변한다

변하지 않는다



## 행렬의 표현

A=

1	2	3
4	5	6

```
>>> A= [[1,2,3], [4,5,6]]
>>> A
[[1, 2, 3], [4, 5, 6]]
첫 행 -----> >>> A[0]
[1, 2, 3]
둘째 행 -----> >>> A[1]
[4, 5, 6]
행의 개수 -----> >>> len(A)
2
열의 개수 -----> >>> len(A[0])
3
A를 출력 -----> >>> for r in A:
                        print(r)

[1, 2, 3]
[4, 5, 6]
>>>
```



## 행열의 덧셈

A

1	2	3
4	5	6

B

5	2	3
1	3	1

```
A=[[1,2,3],[4,5,6]]
B=[[5,2,3],[1,3,1]]
hap=[[0,0,0],[0,0,0]]

for i in range(len(A)):
    for j in range(len(A[0])):
        hap[i][j]=A[i][j]+B[i][j]
for m in hap:
    print(m)
print(hap)
```

A의 행의 개수

A의 열의 개수

```
[6, 4, 6]
[5, 8, 7]
[[6, 4, 6], [5, 8, 7]]
>>>
```



## 행열의 곱셈

A

1	2	3
4	5	6

B

1	2
3	4
5	6

```
A=[[1,2,3],[4,5,6]]
B=[[1,2],[3,4],[5,6]]
mul = [[0,0],[0,0]]
```

```
for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            mul[i][j]+=A[i][k]*B[k][j]
for m in mul:
    print(m)
```

A의 행의 개수

B의 열의 개수

B의 행의 개수

```
[22, 28]
[49, 64]
>>>
```



## 전치행열 구하기

A

1	2	3
4	5	6

$A^T = \text{Transpose}(A) =$

1	4
2	5
3	6

```
A=[[1,2,3],[4,5,6]]
B=[[0,0],[0,0],[0,0]]

for i in range(len(A)):
    for j in range(len(A[0])):
        B[j][i]=A[i][j]
for g in B:
    print(g)
print(B)
```

```
[1, 4]
[2, 5]
[3, 6]
[[1, 4], [2, 5], [3, 6]]
>>>
```



## 6. 함수(function)

- 반복적으로 나타나는 기능을 독립적으로 구현
- 프로그램이 단순화 되고 가독성 증가

함수정의

```
def add(a):  
    a = a+1  
    return a
```

함수사용

-----> print(add(a))

함수사용

-----> print(add(a))

```
a=3
```

```
print(add(a))
```

```
a=5
```

```
print(add(a))
```

```
4  
6  
>>>
```

✓ 함수사용 이전에 함수가 정의되어야 한다.



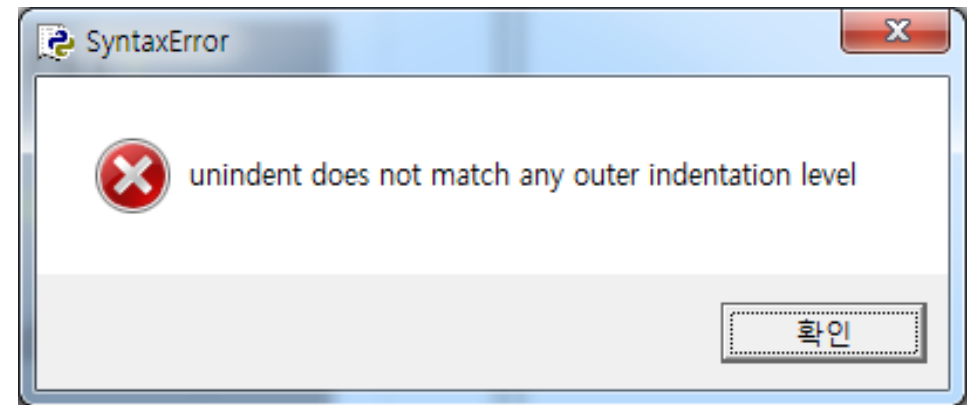
- 함수 내부의 들여쓰기가 일치하여야 한다.

```
def add(a):  
    a = a+1  
    return a  
a=3  
print(add(a))  
a=5  
print(add(a))
```

O

```
def add(a):  
    a = a+1  
    return a  
a=3  
print(add(a))  
a=5  
print(add(a))
```

X





- 함수 내부에도 명령문에 따른 적절한 들여쓰기를 사용하여야 한다.

```
def add(a):  
    if a>1:  
        a = a+1  
    return a  
a=3  
print(add(a))  
a=5  
print(add(a))
```

O



- 주 프로그램의 변수명과 함수내의 변수명

- ✓ 서로 다를 수 있다.
- ✓ 매개변수의 위치에 의한 매핑

```
def add(k):  
    k= k+1  
    return k  
a=3  
print(add(a))
```

```
4  
>>>
```



- 함수 내부에서 변수의 값이 바뀌더라도 외부의 변수값은 바뀌지 않는다.

```
def add(a):  
    a= a+1  
    print("내부",a)  
    return a  
  
a=3  
print(add(a))  
print("외부",a)
```

```
내부 4  
4  
외부 3  
>>>
```



- 함수 내부에 두 개 이상의 변수를 전달할 수 있다.

```
def sum_of_squares(x,y):  
    c = x*x+y*y  
    return c  
a=3  
b=4  
print(sum_of_squares(a,b))
```

```
25  
>>>
```



- 함수 내부에 네 개의 변수를 전달한 예

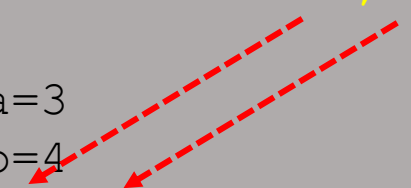
```
def average(a,b,c,d):  
    return (a+b+c+d)/4  
w=3  
x=5  
y=7  
z=10  
print(average(w,x,y,z))
```

```
6.25  
>>>
```



- 함수의 결과로 외부에 두 개 이상의 값을 전달할 수 있다.

```
def comp(x, y):  
    sum = x+y  
    mul = x*y  
    return sum, mul  
  
a=3  
b=4  
c, d = comp(a, b)  
print('Sum is', c)  
print('Multiplication is ', d)
```

A diagram with two red dashed arrows. One arrow starts from the 'sum' variable in the 'return' statement of the 'comp' function and points to the variable 'c' in the assignment 'c, d = comp(a, b)'. The other arrow starts from the 'mul' variable in the 'return' statement and points to the variable 'd' in the same assignment.

```
Sum is 7  
Multiplication is 12  
>>>
```



- 함수는 또 다른 함수를 호출할 수 있다.

주프로그램 → numInput → numComp

```
def numComp(a,b,c):  
    a=int(a)  
    b=int(b)  
    c=int(c)  
    return a*a + b*b +c*c  
  
def numInput ( ):  
    a, b, c = input("A = "), input("B = "),input("C = ")  
    print("제공 합 = ", numComp(a,b,c))  
  
numInput()
```

```
A = 4  
B = 5  
C = 6  
제공 합 = 77  
>>>
```

주프로그램



호출

numInput



호출

numComp





## 피보나치 수열

- $\text{fib}(n)$ 을 정의하는데,  $\text{fib}(n-1)$ 과  $\text{fib}(n-2)$ 를 사용한다
  - ✓ Fibonacci numbers
  - ✓  $\text{fib}(0)=0$ ,  $\text{fib}(1)=1$ ,  $\text{fib}(2)=\text{fib}(1)+\text{fib}(0)=1$
  - ✓  $\text{fib}(3)=\text{fib}(2)+\text{fib}(1)=1+1=2$
  - ✓ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, .....

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2), n \geq 2$$

## ✓ 재귀(recursion) 가능

- 함수가 자신을 호출하는 것
- 피보나치 수열 생성 함수를 구현

```
def fib(n):  
    if n==0:  
        return 0  
    elif n==1:  
        return 1  
    else:  
        return fib(n-1) + fib(n-2)  
  
n=6  
print(fib(n))
```

```
8  
>>>
```

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

n	0	1	2	3	4	5	6	7
fib(n)	0	1	1	2	3	5	8	13

# n!을 재귀를 이용하여 구현

```
def fact(n):  
    if n==1:  
        return 1  
    else:  
        return n * fact(n-1)  
  
n=4  
print(fact(n))
```

```
24  
>>>
```

$$\text{fact}(n) = n \times \text{fact}(n-1)$$

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1$$

(n-1)!

- 함수 내부에 리스트를 전달하여 사용할 수 있다.

```
def days(a):  
    d = 0  
    for i in range(0,12):  
        if i%2 == 0:  
            d +=a[i]  
    return d  
a=[31,28,31,30,31,30,31,31,30,31,30,31]  
print(days(a))
```

```
184  
>>>
```

홀수 달의 날수의 총합

## 7. 출력형식      형식이 있는 output: 1개 이상의 값 또는 변수를 정해진 형식으로 출력

출력할 변수 또는 값

```
a=1.2  
b=345  
print(f'A is {a}, B is {b}')
```

```
A is 1.2, B is 345  
>>>
```

```
age=20  
name='Han'  
print(f'My name is {name} and my age is {age}.')
```

```
My name is Han and my age is 20.  
>>>
```

```
a=2  
print(f'{a} {a*a} {a*a*a}')
```

```
2 4 8  
>>>
```

```

a=1.2
b=345
print(f'A is {a:3.1f}, B is {b:3d}')
print(f'A is {a:7.3f}, B is {b:5d}'))

```

```

A is 1.2, B is 345
A is 1.200, B is 345
>>>

```



```

>>> a=123.4567
>>> print(f'{a:10.3e}')
1.235e+02
>>> print(f'{a:10.4e}')
1.2346e+02
>>> print(f'{a:13.4E}')
1.2346E+02
>>> print(f'{a:10.1f}')
123.5
>>> print(f'{a:11.5f}')
123.45670

```

10.3e



소수점 아래 3자리

10.4e



10.4E

대문자 E

- 1.235e+02는  $1.235 \times 10^2$ 을 표시한 것
- $1.235 \times 10^2$ 은  $1.234567 \times 10^2$  가 반올림된 것



```

sum=0
d=1.5
while d < 3.5:
    print(d)
    sum = sum + d
    d = d+0.1
print("합=",sum)

```

```

1.5
1.6
1.700000000000000002
1.800000000000000003
...
...
...
3.300000000000000016
3.400000000000000017
합= 49.0000000000000014
>>>

```

## 출력 형식을 정의한 후

```

sum=0
d=1.5
while d < 3.5:
    print(f'{d:6.2f}')
    sum = sum + d
    d = d+0.1
print(f'합= {sum:6.2f}')

```

```

1.50
1.60
1.70
1.80
...
...
3.30
3.40
합= 49.00
>>>

```

```
name='han'
print(f'My name is {name: ^10}.')
print(f'My name is {name: <10}.')
print(f'My name is {name: >10}.')
```

```
a=123
print(f'a={a:10} ')
print(f'a={a:>10} ')
print(f'a={a:^10} ')
print(f'a={a:<10} ')
```

```
My name is      han      .
My name is han          .
My name is              han.
>>>
```

```
a=          123
a=          123
a=    123
a=123
>>>
```

## Program 수행 시간 확인

```
import time
N=50000000
d=[]

a=range(N)
startTime= time.time()

for i in a:
    d.append(2*i)

endTime = time.time()
print(endTime -startTime)
```

# Set and Dictionary

# Set

- 중복된 원소를 제거
- {} 로 표시
- empty set은 set()로 정의
- integer, float, tuple, string을 원소로 가질 수 있다.

```
a = {1, 3, 3, 5, 7}  
print(a)
```



```
{1, 3, 5, 7}
```

```
a = set( )  
print(a)  
a.add(9)  
print(a)
```



```
set()  
{9}
```

```
a = {1, 2.3, "ABC", (5, 6, "D")}  
print(a)
```



```
{2.3, 1, 'ABC', (5, 6, 'D')}
```

- mutable(변할 수 있는) element인 list, dictionary를 원소로 가질 수 없다.

```
a = {[1,2,3]}  
print(a)
```

리스트 [1,2,3]을 원소로 한다.



```
TypeError: unhashable type: 'list'
```

```
a = set([1,2,3])  
print(a)
```

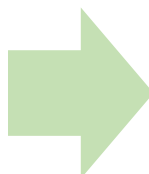
리스트 [1,2,3]이 갖고 있는 값들을 원소로 한다.



```
{1, 2, 3}
```

- set 선언 a={ }은 dictionary를 선언한다.

```
a = { }  
print(type(a))
```



```
<class 'dict'>
```

```
a=set( )  
print(type(a))
```



```
<class 'set'>
```

- `set_name.add( )`는 하나의 원소를 추가한다.

```
a= {1,2,3}  
print(a)  
a.add(4)  
print(a)
```



```
{1, 2, 3}  
{1, 2, 3, 4}
```

```
a.add(5,6)
```



```
TypeError: add() takes exactly one  
argument (2 given)
```

- `set_name.update( )`: 괄호 내에는 tuple, list, string 또는 set.

```
a= {1,2,3}  
a.update([4,5])  
print(a)
```



```
{1, 2, 3, 4, 5}
```

```
a= {1,2,3}  
a.update((4,5))  
print(a)
```



```
{1, 2, 3, 4, 5}
```

```
a= {(1,2), (3,4)}  
print(a)  
a.update((5,6))  
print(a)
```



```
{(1, 2), (3, 4)}  
{(1, 2), (3, 4), 5, 6}
```

```
a= {(1,2), (3,4)}  
print(a)  
a.add((5,6))  
print(a)
```



```
{ (1, 2), (3, 4) }  
{ (1, 2), (3, 4), (5, 6) }
```



- `set_name.discard( )` 또는 `set_name.remove( )`: 괄호 내에는 원소

```
a={1,2,3}
a.discard(1)
print(a)
a.remove(2)
print(a)
```



```
{2, 3}
{3}
```

- `discard`는 set 내에 원소가 없어도 가능. `remove`는 원소가 없을 경우 error

```
a={1,2,3}
a.discard(4)
print(a)
a.remove(4)
print(a)
```



```
{1, 2, 3}
. . . . .
a.remove(4)
KeyError: 4}
```

- 여러 개의 원소를 제거

```
a={1,2,3,4,5}
b=set( )
for x in a:
    if x not in {3,5}:
        b.add(x)
print(b)

a={1,2,3,4,5}
a={x for x in a if x not in {3,5}}
print(a)
```



```
{1, 2, 4}
{1, 2, 4}
```

# set operation

- union

```
a={1,2,3}  
b={3,4,5}  
c = a|b  
print(c)  
  
print(a.union(b))
```



```
{1, 2, 3, 4, 5}  
{1, 2, 3, 4, 5}
```

- intersection

```
a={1,2,3}  
b={3,4,5}  
print(a & b)  
  
print(a.intersection(b))
```



```
{3}  
{3}
```

- difference

```
a={1,2,3}
b={3,4,5}
print(a - b)
print(a.difference(b))
```



```
{1, 2}
{1, 2}
```

- symmetric difference

```
a={1,2,3}
b={3,4,5}
print(a ^ b)
print(a.symmetric_difference(b))
```



```
{1, 2, 4, 5}
{1, 2, 4, 5}
```

- member

```
a={1,2,3}  
print(2 in a)  
print(5 in a)
```



```
True  
False
```

## Dictionary

- { 'key' : 'value' }
- {} 로 표시
- key는 immutable{변할 수 없는} type을 가져야 한다.
- string, number, tuple 가능

```
a={ }  
print(a)
```

```
a = {1: 'car', 2: 'house'}  
print(a)
```



```
{ }  
{1: 'car', 2: 'house'}
```

- value로는 다양한 형태가 가능.

```
a={'kk':'car', 2:[5,7]}  
print(a)
```



```
{2: [5, 7], 'kk': 'car'}
```

- dict()는 dictionary로 만든다.

```
a={'college':'eni', 'dept':'ce'}  
print(a)  
  
a=dict({'college':'eni', 'dept':'ce'})  
print(a)  
  
a=dict([('college','eni'), ('dept','ce')])  
print(a)
```



```
{'dept': 'ce', 'college': 'eni'}  
{'dept': 'ce', 'college': 'eni'}  
{'dept': 'ce', 'college': 'eni'}
```

- key를 이용하여 value를 찾는 방법

```
a={'college':'eni', 'dept':'ce'}  
print(a['college'])  
print(a['dept'])  
print(a.get('dept'))
```



```
eni  
ce  
ce
```

- key의 value를 변경하는 방법

```
a={'college':'eni', 'id':1234}  
a['id']=4321  
print(a)
```



```
{'id': 4321, 'college': 'eni'}
```



- key를 추가하는 방법

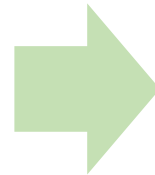
```
a={'college':'eni', 'id':1234}  
a['age']=23  
print(a)
```



```
{'college': 'eni', 'id': 1234, 'age': 23}
```

- key를 삭제하는 방법

```
a={'college':'eni', 'id':1234, 'age':23}  
del a['college']  
print(a)  
a.clear()  
print(a)
```



```
{'id': 1234, 'age': 23}  
{}
```

- membership 확인하는 방법

```
a={'college':'eni', 'id':1234, 'age':23}  
print('id' in a)  
print('home' in a)
```



```
True  
False
```

- iterating 방법

```
a={'college':'eni', 'id':1234, 'age':23}  
for i in a:  
    print(a[i])
```



```
eni  
1234  
23
```

# stack

```
class Stack(object):

    top=0
    data=None
    length=None

    def __init__(self, length):
        self.length=length
        self.data =self.length*[0]

    def isFull(self):
        return (self.top == self.length)

    def isEmpty(self):
        if self.top == 0:
            return True

    def push(self, name):
        if self.isFull( )==True:
            print("Stack is full")
            return
        else:
            self.data[self.top]=name
            self.top += 1
```

```
def pop(self):
    if self.isEmpty() == True:
        print("no data")
        return
    else:
        a = self.data[self.top-1]
        self.top -= 1
    return a
def printStack(self):
    if(self.isEmpty() == True):
        return
    for i in range(0, self.top):
        print(self.data[i])

print()
```

```
s = Stack(5)
s.push("A")
s.push("B")
s.printStack()
print(s.pop())
```

```
A
B

B
>>>
```

# queue

```
class Queue(object):

    front=0
    rear=0
    data=None
    length=None

    def __init__(self, length):
        self.length=length
        self.data =self.length*[0]

# 최대 n-1개의 데이터가 저장. 만일 rear=front를 full의 판단으로 사용한다면
# 꽉찬 것과 empty와 구분 불가
    def isFull(self):
        return ((self.rear+1) % self.length==self.front)

    def isEmpty(self):
        if self.front==self.rear:
            return True
```

```
def enqueue(self, name):
    if self.isFull( )==True:
        print("Queue is full")
        return
    else:
        self.data[self.rear]=name
        self.rear=(self.rear+1)%self.length

def dequeue(self):
    if self.isEmpty( )==True:
        print("no data")
    else:
        self.front=(self.front+1) % self.length

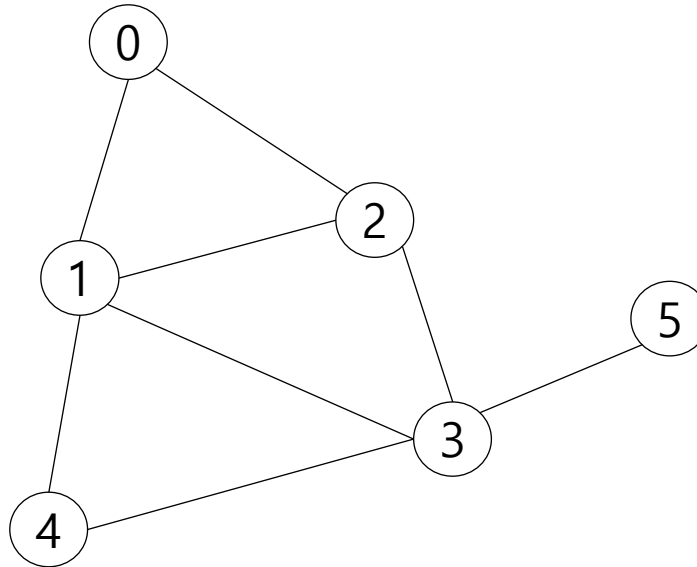
def printQ(self):
    if(self.isEmpty()==True):
        return
    if(self.rear<self.front):
        for i in range(self.front, self.length):
            print(self.data[i], end=" ")
        for i in range(0, self.rear):
            print(self.data[i], end=" ")
    else:
        for i in range(self.front, self.rear):
            print(self.data[i], end=" ")
    print()
```

```
q= Queue(5 )
q.enqueue("A")
q.printQ()
q.enqueue("B")
q.printQ()
q.enqueue("C")
q.enqueue("D")
q.printQ()
q.dequeue()
q.printQ()
q.dequeue()
q.printQ()
q.enqueue("E")
q.printQ()
q.enqueue("F")
q.printQ()
q.dequeue()
q.printQ()
```

```
A
A B
A B C D
B C D
C D
C D E
C D E F
D E F
>>>
```

# 그래프의 파이썬 표현

## 1. list of list

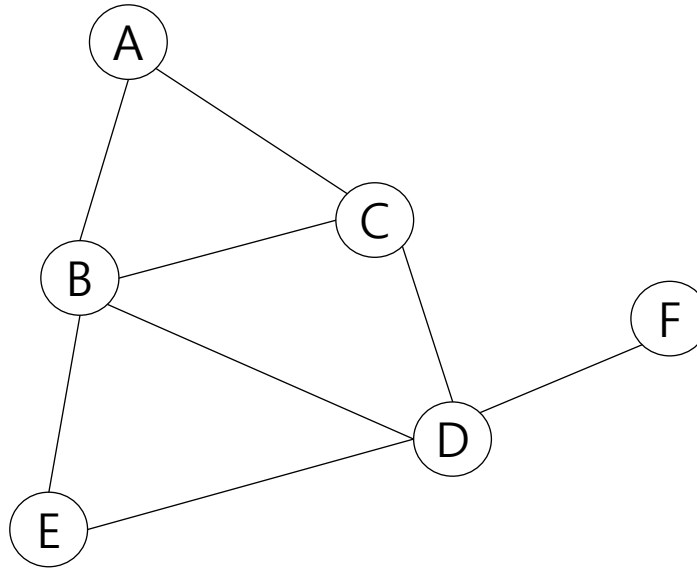


```
g = [[0,1,1,0,0,0] , [1,0,1,1,1,0],[1,1,0,1,0,0],  
      [0,1,1,0,1,1] , [0,1,0,1,0,0], [0,0,0,0,1,0]]  
print (g)
```

```
[[0, 1, 1, 0, 0, 0], [1, 0, 1, 1, 1, 0],  
 [1, 1, 0, 1, 0, 0], [0, 1, 1, 0, 1, 1],  
 [0, 1, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]]
```



## 2. dict of sets



```
g={'A': {'B','C'}, 'B':{'A','C','D','E'},  
   'C': {'A','B','D'}, 'D':{'C','B','E','F'},  
   'E': {'B','D'}, 'F':{'D'}}  
print(g)
```

```
{'A': {'C', 'B'}, 'B': {'C', 'E', 'D', 'A'},  
'C': {'B', 'D', 'A'}, 'D': {'C', 'F', 'B', 'E'},  
'E': {'B', 'D'}, 'F': {'D'}}  
{'C', 'B'}
```

Q: 그래프의 노드 개수?

### 1. list of list

```
print(len(g))  
print(len(g[0]))
```

### 2. dict of sets

```
print(len(g))  
print(len(g['A']))
```

Q: 그래프의 에지 개수는?

## 1. list of list

```
def n_of_edges(g):  
    e=0  
    for i in range(0, len(g)):  
        for j in range(i, len(g)):  
            e+=g[i][j]  
    return e  
print(n_of_edges(g))
```

## 2. dict of sets

```
def n_of_edges(g):  
    e=0  
    for i in g:  
        e+= len(g[i])  
    return e  
print(int(n_of_edges(g)/2))
```

Q: 노드 i와 j가 연결되었는가?

1. list of list

```
print(g[0][2])
```

0 or 1

2. dict of sets

```
print('C' in g['A'])
```

True or False

Q: 노드 i에 연결된 에지의 수?

### 1. list of list

```
def n_of_connected_edges(g,i):  
    e=0  
    for j in range (0,len(g)):  
        e+=g[i][j]  
    return e  
  
print(n_of_connected_edges(g,2))
```

2번 노드에 연결된 에지의 수

### 2. dict of sets

```
def n_of_connected_edges(g,i):  
    return len(g[i])  
print(n_of_connected_edges(g,'A'))
```

노드 'A'에 연결된 에지의 수

## Alg 1.1

## 순차검색 알고리즘

**문제:**  $n$ 개의 키로 구성된 배열  $S$ 에 키  $x$ 가 있는가?

**입력:** 양의 정수  $n$ , 1에서  $n$ 까지의 첨자를 가진 키의 배열  $S$ , 그리고  $x$

**출력:**  $S$ 안에  $x$ 의 위치를 가리키는  $\text{location}$ ( $S$ 안에  $x$ 가 없으면 0)

```
void seqsearch(int n,
               const keytype S[],
               keytype x,
               index& location) { // 출력
    location = 1;
    while (location <= n && S[location] != x)
        location++;
    if (location > n)
        location = 0;
}
```

✓ while-루프: 아직 검사할 항목이 있고,  $x$ 를 찾지 못했나?

✓ if-문: 모두 검사하였으나,  $x$ 를 찾지 못했나?

## [실습프로그램] 순차검색 알고리즘

```
def seqsearch(s,x):
```

순차검색 구현

```
s=[3,5,2,1,7,9]  
loc = seqsearch(s,4)  
print(loc)
```

-1

## Alg 1.2

## 배열의 수 더하기

**문제:**  $n$ 개의 수로 된 배열  $S$ 에 있는 모든 수를 더하라

**입력:** 양의 정수  $n$ , 수의 배열  $S$ (첨자는 1부터  $n$ )

**출력:**  $S$ 에 있는 수의 합,  $sum$

```
number sum(int n, const number S[ ]) {  
    index i;  
    number result;  
  
    result = 0;  
    for (i=1;i<=n;i++)  
        result = result+S[i];  
    return result;  
}
```



## [실습프로그램] 배열의 수 더하기

```
def sum1(s):  
    result=0  
    for a in s:
```

부분 구현

```
s=[3,5,2,1,7,9]  
answer = sum1(s)  
print(answer)
```

27

```
def sum2(s):  
    result=0  
    for i in range(len(s)):
```

부분 구현

```
s=[3,5,2,1,7,9]  
answer = sum2(s)  
print(answer)
```

27

## Alg 1.3

## 교환정렬

**문제:** 비내림차순(nondecreasing order)으로  $n$ 개의 키를 정렬하라

**입력:** 양의 정수  $n$ , 키의 배열  $S$ (첨자는 1부터  $n$ )

**출력:** 키가 비내림차순으로 정렬된 배열  $S$

```
void exchangesort(int n, keytype S[ ]) {  
    index i, j;  
  
    for (i=1; i<=n-1; i++)  
        for (j=i+1; j<=n; j++)  
            if(S[j] < S[i])  
                exchange S[i] and S[j]  
}
```

## [실습프로그램] 교환정렬

```
s=[3,2,5,7,1,9,4,6,8]  
n = len(s)  
for i in range (0,n-1):
```

부분 구현

```
print(s)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Alg 1.4

## 행렬곱셈

문제: 두 개의  $n \times n$  행렬의 곱을 구하라

입력: 양의 정수  $n$ , 수의 2차원 배열  $A$ 와  $B$ . 여기서 이 행렬의 행과 열은 모두 1부터  $n$ 까지의 첨자를 갖는다

출력:  $A$ 와  $B$ 의 곱이 되는 수의 2차원 배열  $C$ . 이 행렬의 행과 열은 모두 1부터  $n$ 까지의 첨자를 갖는다

```
void matrixmult(int n, const number A[ ][ ],
                const number B[ ][ ],
                number C[ ][ ]) {
    index i, j, k;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++) {
            C[i][j]=0;
            for (k=1; k<=n; k++)
                C[i][j]=C[i][j]+A[i][k]*B[k][j];
        }
}
```

## [실습프로그램] 행렬곱셈

```
def matrix_multiplication(a,b):
```

구현

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

```
b=[ [4,1],[1,0]]
```

```
print(matrix_multiplication(a,b))
```

```
[[6, 1], [16, 3]]
```

```
void binsearch(int n,
               const keytype S[],
               keytype x,
               index& location) {    // 출력
    index low, high, mid;

    low = 1; high = n;
    location = 0;
    while (low <= high && location == 0) {
        mid = (low + high) / 2;    // 정수나눗셈
        if (x == S[mid])
            location = mid;
        else if (x < S[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
}
```

✓ while-루프: 아직 검사할 항목이 있고,  $x$ 를 찾지 못했나(location=0)?

# [실습프로그램] 이분검색 알고리즘

```
def bs(data,item, low, high):
```

구현

```
data=[1,3,5,6,7,9,10,14,17,19]  
n=10  
location=bs(data,17,0,n-1)  
print(location)
```

8

# $n$ 번째 피보나찌 수 구하기

피보나찌(Fibonacci) 수열의 정의

$$\begin{aligned}f_0 &= 0 \\f_1 &= 1 \\f_n &= f_{n-1} + f_{n-2} \quad \text{for } n \geq 2\end{aligned}$$

예: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, ...

$$f_n = \frac{[(1 + \sqrt{5})/2]^n - [(1 - \sqrt{5})/2]^n}{\sqrt{5}} \quad (\text{예제 B.9 in appendix B})$$



# 피보나찌 수 구하기(재귀적 방법)

- 문제:  $n$ 번째 피보나찌 수를 구하라.
- 입력: 양수  $n$
- 출력:  $n$  번째 피보나찌 수
- 알고리즘:

```
int fib1 (int n) {  
    if (n <= 1)  
        return n;  
    else  
        return fib1(n-1) + fib1(n-2);  
}
```

# 피보나찌 수 구하기 알고리즘 (반복적 방법)

```
int fib2 (int n) {  
    index i;  
    int f[0..n];  
    f[0] = 0;  
    if (n > 0) {  
        f[1] = 1;  
        for (i = 2; i <= n; i++)  
            f[i] = f[i-1] + f[i-2];  
    }  
    return f[n];  
}
```

# 두 피보나찌 알고리즘의 비교

$n$	$n+1$	$2^{n/2}$	반복(Alg 1.7)	재귀(Alg 1.6 하한)
40	41	1,048,576	41ns	1048 $\mu$ s
60	61	$1.1 \times 10^9$	61ns	1s
80	81	$1.1 \times 10^{12}$	81ns	18mins
100	101	$1.1 \times 10^{15}$	101ns	13days
120	121	$1.2 \times 10^{18}$	121ns	36years
160	161	$1.2 \times 10^{24}$	161ns	$3.8 \times 10^7$ years
200	201	$1.3 \times 10^{30}$	201ns	$4 \times 10^{13}$ years

Assume that 1 transaction takes 1 ns. (1 ns =  $10^{-9}$  second, 1  $\mu$ s =  $10^{-6}$  second)

## [실습프로그램] 피보나찌 수 구하기

```
def fib1(n):
```

재귀적 방법으로 구현

```
for i in range(0,10):  
    print( f'{i:2d} {fib1(i)}: 6d }')
```

```
def fib2(n):
```

반복적 방법으로 구현

```
for i in range(0,10):  
    print(f'{i:2d} {fib1(i)}: 6d }')
```

n	fib(n)
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34

## 시간 측정 방법: import time, time.time( ) 사용

```
import time
def fib1(n):
```

재귀적 방법

```
for i in range(30,36):
    stime = 현재 시간 확인
    fib1(i)
    print( f'{i:2d}{시간:10.5f}  ')
```

30	0.44500
31	0.69500
32	1.14800
33	1.83900
34	3.00100
35	4.88600

```
import time
```

```
def fib2(n):
```

반복적 방법 사용

```
print()
```

```
for i in range(30,36):
```

```
    stime = 현재 시간 확인
```

```
    fib2(i)
```

```
    print( f'{i:2d} {시간:10.5f}')
```

30	0.00000
31	0.00000
32	0.00000
33	0.00000
34	0.00000
35	0.00000

다양한 n의 값에 대한 fib1과 fib2의 수행시간을 확인해 보시오.

n	fib1	fib2
10		
20		
30		
50		
100		
200		