경동고등학교 프로그래밍 특강

- Python을 이용한 데이터분석 기초 -

2022 년 12 월

숭실대학교 IT대학 전자정보공학부 IT융합전공

이름: 이준우

목차

경동고등학교 프로그래밍 특강

- Python을 이용한 데이터분석 기초 -

숭실대학교 IT대학 전자정보공학부 IT융합전공

이준우

I. 서론

1.1 Purpose

경동고등학교 선배의 특강을 통해 컴퓨터공학 및 소프트웨어 공학에서 배우는 전공과목의 기본을 알아본다. Python 오픈소스 라이브러리를 활용하여 데이터 획득 및 처리, 분석, 시각화하는 방법과 인공지능의 머신 러닝에 대한 기초적인 내용을 학습한다. 이를 위해 데이터 사이언스 관련 라이브러리인 Matplotlib, Numpy, Pandas, Seaborn, Scikit-Learn 등의 오픈소스 라이브러리를 사용하는 방법을 학습하고 이를 통해 자신의 진로 탐색과 학생부 종합전형에 도움을 주는 것을 목표로 한다. 이 특강을 진행한 후 프로그래밍 언어의 활용성에 대해 생각하고, 공부하며, 더 발전시켜 학생부 종합전형에서 좋은 결과를 얻길 기원하는 마음에 진행한다.

II. 데이터 분석 기초

2.1 Python Basic Grammar

2.1.1 if ~ else, elif

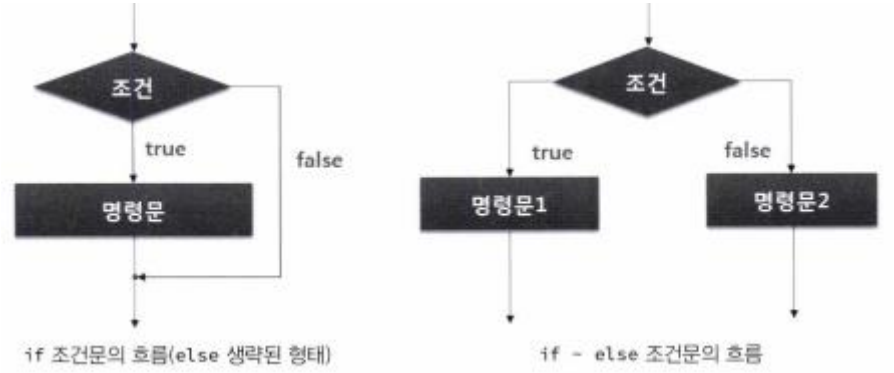
조건문이 없는 프로그램의 경우 흐름을 제어할 수 없기 때문에 항상 도일한 방식으로만 실행되는 단조로운 프로그램으로 기능이 제한될 수밖에 없다. 반면 조건문을 사용하면 프로그램이 사용자의 입력 및 상황에 맞게 제어될 수 있으므로 지능적인 프로그램을 제작할 수 있다.

Python의 조건문을 아래 3개의 문법으로 구분하여 설명하고자 한다.

조건문	설명
if	조건이 맞는 경우 실행할 명령을 정의
if … else …	조건에 맞는 경우와 맞지 않는 경우에 대하여 다른 명령을 정의
if…elif…elif...else	여러 가지 조건에 따라 다른 명령을 정의

if … else …조건문 활용

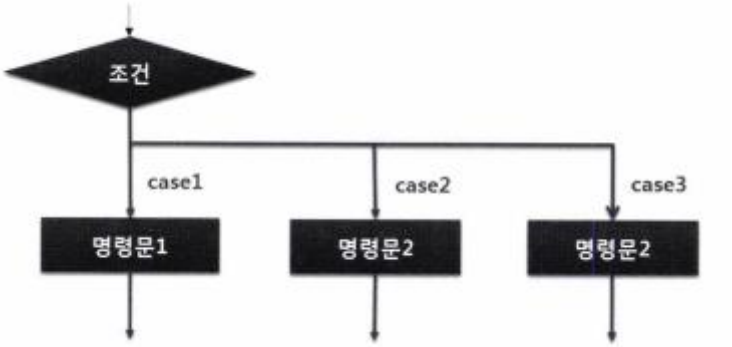
if ~ else 조건문을 사용하여 주어진 조건이 참(true)인 경우와 거짓(false)인 경우에 따라 명령문을 다르게 정의하여 실행할 수 있다. else부분은 생략될 수 있으므로 조건문이 참인 경우에만 특정 명령이 실행되도록 할 수도 있다.



if 조건문의 흐름(else 생략된 형태) if - else 조건문의 흐름

if ~ elif ~ else 조건문 활용

if ~ else 조건문의 경우 흐름을 2가지 방향으로만 지정할 수밖에 없는 한계가 있다. 그러나 실제적으로는 2가지 이상의 다양한 경우에 대하여 흐름을 제어해주어야 할 필요가 있다.



비교 연산자

두 개의 값 사이의 크기를 비교하여 참, 거짓을 판별해주는 연산자를 비교연산자라고 한다. 비교 연산자는 관계 연산자라고 부르기도 한다.

비교 연산자	사용 방식	의미
==	x == y	x, y의 값이 같은가?
!=	x != y	x, y의 값이 같지 않은가?
>	x > y	x값이 y값보다 큰가?
>=	x >= y	x값이 y값보다 크거나 같은가?
<	x < y	x값이 y값보다 작은가?
<=	x <= y	x값이 y값보다 작거나 같은가?

==연산자는 두 값이 같은가(equal)를 점검하는 연산자이다. 우리가 익숙하게 사용하는 = 연산자는 대입의 기능을 하는 연산자이다.

논리연산자

논리(logic)이란 참, 거짓이 분명한 명제를 기초로 추론하는 과정을 의미한다. 논리연산자는 이러한 논리의 추론 과정에서 기본적으로 사용되는 연산자이다. 조건을 판단하여 참, 거짓의 결과를 얻는 연산자를 논리 연산자라고 한다.

비교 연산자	사용 방식	의미
and	조건1 and 조건2	조건1, 조건2가 모두 만족하면 참(true)이다.
or	조건1 or 조건2	조건1, 조건2 둘 중 하나라도 만족하면 참(true)이다
not	not 조건	조건이 참(true)이면 거짓(false)이 되고, 거짓이면 참이 된다.

if 문법 살펴보기

if 문장을 사용하면 특정 조건이 맞을 경우에만 특정 명령이 실행되도록 할 수 있다. 아래 박스에 if 조건문을 사용하는 방식을 표현하였다.

```
if 조건문 :
    명령문1
```

1. **콜론(:) 사용하기**: if 키워드 다음에 조건문이 위치하며 조건문 옆에 콜론(:)을 꼭 입력해주어야 한다. 처음 Python을 배울 때 콜론을 빠뜨리는 경우가 많으므로 주의하자.
2. **조건문 괄호 사용 방식**: Python에서는 조건문에 여는 괄호 '('와 닫는 괄호 ')'는 일반적으로 사용하지 않는다. 물론 괄호를 사용해도 오류가 발생하지는 않지만 꼭 사용해야 하는 것은 아니다. C언어나 Java에서는 ()를 꼭 사용해야 하는 것과 차이가 있다.
3. **들여쓰기 주의하기**: 조건문이 만족할 경우 실행될 문장(명령문)은 들여쓰기가 되어 있어야 한다. 들여쓰기는 "if 조건문:"을 입력한 후 Enter를 누르면 자동으로 들여쓰기가 되는데 그 부분에 명령문을 작성하면 된다. 들여쓰기를 직접 지정할 경우 Tab 키를 사용하면 된다.

```
score = 80
if score >= 90:
    print("장학금 대상자입니다")
print("수고하셨습니다")
```

위 소스코드를 입력하면 어떤 결과를 얻게 될까? 조건이 만족하지 않으므로(거짓이므로) if에 종속된 문장이 실행되지 않는다. 즉 "수고하셨습니다"라는 문장만 화면에 출력될 것이다.

코드를 수정하여 점수를 입력 받은 후, 입력한 점수에 따라서 장학금을 판단하도록 수정해보자. input함수를 이용하여 텍스트를 입력 받은 후 int함수로 자료형을 정수로 변환하였다.

```
score = int(input("점수를 입력하세요: "))
if score >= 90:
    print("장학금 대상자입니다")
print("수고하셨습니다")
```

점수를 입력하세요: 95 장학금 대상자입니다 수고하셨습니다	점수를 입력하세요: 80 수고하셨습니다

if ~ else ~ 문법 살펴보기

조건문이 참(true)인 경우와 거짓(false)인 경우에 다른 명령을 수행하도록 할 때 if ~ else 문법을 사용한다.

```
if 조건문 :
    명령문1
```

1. **if와 else 문장은 같은 레벨이어야 한다**: 즉 들여쓰기 수준이 일치되어야 한다. 만약 들여쓰기가 일치되어 있지 않으면 에러가 발생한다.
2. **콜론 입력하기**: if 문장 및 else 문장 끝에는 콜론을 꼭 입력해 주어야한다.
3. **여러 문장 포함하기**: if 문 혹은 else 문에 여러 문장이 포함되도록 하려면 아래와 같이 들여쓰기를 일치하여 표시하여야 한다. 아래 박스에서는 2개의 문장을 예시로 표현하였지만, 그 이상의 개수로 확장될 수 있다. 종속되는 문장들은 들여쓰기가 모두 정확하게 일치하여야 함을 주의하자.

```
if 조건문 :
    명령문-참1
    명령문-참2
else :
    명령문-거짓1
    명령문-거짓2
```

```
score = int(input("점수가 몇 점 입니까?:  "))
if score >= 90:
    print("장학금 대상자입니다")
    print("축하합니다")
else:
     print("장학금 대상자가 아닙니다.")
     print("다음 학기를 노려봅시다")
print("수고하셨습니다")
```

실습1 짝수 홀수 판별하는 프로그램

어떤 정수를 입력 받아 짝수와 홀수를 판별하는 문제이다. 짝수, 홀수는 어떻게 판별할 수 있을까? 짝수는 2로 나누어 떨어지는 수이며, 홀수는 2로 나누어 떨어지지 않는 수(나머지가 1인 수)임을 고려하여 해결할 수 있을 것이다.

- 짝수는 2로 나누어 떨어지는 양의 정수이다.
- 홀수는 2로 나누어 떨어지지 않는 양의 정수이다.

아래와 같은 결과가 표시되도록 프로그램을 만들어보자.

===짝수 홀수 판별 프로그램=== 정수를 입력하세요: 7 정수 7을 입력했군요 당신이 입력한 수는 홀수입니다.	===짝수 홀수 판별 프로그램=== 정수를 입력하세요: -10 판별할 수 없는 수를 입력하셨습니다. 양의 정수만 짝수 홀수 판별 가능합니다.
7을 입력한 경우	-10을 입력한 경우

```
print("===짝수 홀수 판별 프로그램===")
n = int(input("정수를 입력하세요: ")
print("정수 %d를 입력했군요" %n)


if n%2 == 0 :
   print("당신이 입력한 수는 짝수입니다.")
else :
   print("당신이 입력한 수는 홀수입니다.")
```

위의 코드를 확장하여 예시 결과가 나타나는 프로그램을 만들어보자.

if ~ elif ~ else 조건문 이해하기

여러 가지 조건을 지정하고 각 조건에 맞는 명령을 수행하게 하려면 if ~ elif ~ else 문법을 사용하는 것이 적합하다. 매우 빈번하게 사용되는 문법이므로 잘 배워두도록 하자.

1. elif 문장은 개수에 제한 없이 확장될 수 있다.
2. else 문은 꼭 나와야 하는 것은 아니다. 필요 없는 경우 생략될 수 있다.

```
if 조건문1 :
   명령문1
elif 조건문2 :
   명령문2
elif 조건문3 :
   명령문3
…
else :
   명령문 n
```

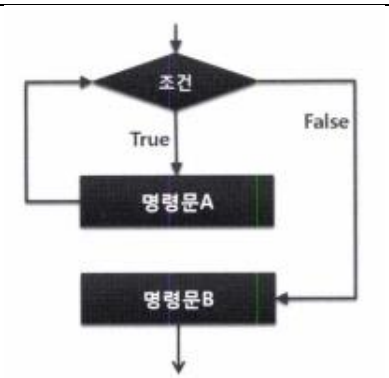
```
score = int(input("시험점수를 입력하세요:  "))
if score>=90 :
   print("시험을 아주 잘 봤군요.")
elif score>=80 :
   print("시험을 잘 봤군요.")
elif score>=70 :
   print("다음에는 잘 봐요…")
else :
   print("완전히 망했군요")
```

2.1.2 while, for

while 반복문 문법

while 반복문의 문법은 아래와 같다. 아래는 조건문이 참(true)인 동안 명령문 A가 계속 실행되다가 조건문이 거짓(false)이 되면 while 반복문을 빠져나와 명령문 B를 실행한다.

```
while 조건문 :
    명령문 A
명령문 B
```



아래와 같이 Count 1, Count 2, ···, Count 10의 형태로 1부터 10까지 세는 프로그램을 작성해 보자.

```
n = 1
while n <= 10:
   print("Count", n)       # print("Count %d" %n)으로 표현 가능
   n += 1          # n = n+1
print("종료합니다")
```

이와 같이 반복문을 사용할 경우 아래의 3가지 내용에 대한 설정 과정이 필요하다.

- [시작값]: 몇부터 시작할 것인가?
- [증가값]: 몇만큼씩 값이 변화될 것인가?
- [종료값]: 반복문이 언제 종료될 것인가?

for 반복문 문법

for 반복문의 문법은 아래와 같다.

```
for 변수 in range(시작값, 종료값, 증가값) :
    명령문1
명령문2
```

for 다음에 반복에 사용될 변수 이름을 지정한다. 변수 이름은 자유롭게 지정할 수 있다. t,n,I 등 원하는 이름을 지정하면 된다. 단, 이미 사용되지 않은 새로운 이름을 지정해야 한다.

in range 다음에 소괄호를 사용한다. 괄호 안에 쉼표를 사용하여 3개의 파라미터(매개변수)를 입력한다. 첫 번째는 시작값, 두 번째는 종료값, 세 번째는 증가값이다. 지정한 변수가 시작값에서 시작하여 증가값만큼 변화되다가 종료값을 만다거나 그 값을 넘어서면 반복문이 종료된다.

시작값을 생략하면 0부터 시작으로 간주한다. 증가값을 생략하면 1씩 증가하는 것으로 간주한다.

```
for n in range(1, 10, 1) :
   print("count %d" %n)
print("종료합니다.")
```

```
for n in range(10) :
   print("count %d" %n)
print("종료합니다.")
```

위 코드는 동일한 결과를 도출한다.

2.1.3 list

지금까지는 메모리에 어떤 값을 입력할 때 일반적으로 변수를 사용하였다. 변수는 어떤 한 개의 값을 메모리에 저장할 때 사용한다. 그런데 만약 한 개의 값이 아니라 50개의 값 혹은 그 이상의 값을 입력하려면 어떻게 해야 할까? 저장하는 변수를 여러 개를 만들어야 하므로 비효율적인 코드가 된다. 이런 상황에서 리스트를 만들어서 여러 데이터를 저장한다. 앞으로 프로그래밍을 배워갈수록, 그리고 더 복잡한 문제를 해결할수록 단일변수보다는 리스트의 형태를 많이 사용하게 될 것이다.

비어 있는 리스트 만들기

비어 있는 리스트, 즉 0개의 항목이 들어있는 리스트는 아래와 같이 대괄호를 열고, 닫아서 생성할 수 있다. 빈 리스트의 경우 출력하면 비어 있는 대괄호로 출력된다.

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

빈 리스트를 만드는 다른 방식은 아래와 같이 list() 함수를 사용하는 것이다. list함수에 파라미터를 부여하지 않으면 빈 리스트가 만들어진다.

```
a = list()              # a = []와 동일한 의미
print(a)
```

여려 값들이 들어있는 리스트 만들기

여러 개의 값들이 들어있는 리스트를 만들어보자. 우선 리스트는 여는 대괄호와 닫는 대괄호를 사용하여 지정해야 한다. 또한 여러 개의 값을 입력할 경우 대괄호 안의 각 항목들은 쉼표로 구분하여 나열해야 한다.

```
a = [10, 20, 30]
print(a)
```

여러 개의 값이 들어가는 리스트를 list 함수를 명시하여 사용하여 생성할 수도 있다. List 함수의 파라미터는 값들을 쉼표로 구분하여 대괄호로 묶는 형식이다. 바로 앞의 방식은 정확하게는 list함수를 생략한 것이라고 보면 된다.

```
a = list([10, 20, 30])
print(a)
```

리스트에 다양한 자료 넣기

앞의 예제에서는 리스트에 정수 값들을 저장하는 것을 확인하였다. 그러나 리스트에는 정수 값 이외에도 실수, 문자열 등 다양한 값들이 저장될 수 있다.

```
a = [10, 20, 30]
b = [1.1, 1.2, 1.3]
c = [“kim”, “park”, “choi”, “lee”]
```

또한 아래와 같이 특별하게 보이지만 리스트 안에 원소로 리스트가 들어갈 수도 있다. 아래에서 b와 c는 동일한 값을 갖는다. 단지 b는 한 줄로 입력했을 뿐이고, c는 보기 좋게 여러 줄로 입력한 것이다. C의 입력 형태를 보면 알겠지만 리스트 안에 리스트를 넣는 형태로 수학에서의 행렬 구조를 표현할 수 있다.

```
a = [ [1, 2, 3], [4, 5, 6, 7], [8, 9] ]
b = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
c = [ [1, 2, 3],
      [4, 5, 6],
      [7, 8, 9] ]
```

하나의 리스트에 동일한 형태(데이터 타입, 자료형)의 값들을 갖는 것이 일반적이지만, 꼭 그럴 필요는 없다. 리스트는 서로 다른 데이터 타입의 항목들을 포함할 수 있다.

C언어나 Java를 공부할 때 배열(array)을 배우게 된다. 보통 배열에서는 하나의 데이터 타입의 값들만 입력될 수 있는데 리스트는 그러한 제한이 없다는 것을 기억해 두자. Python은 C를 기반으로 제작되었음. C언어의 포인터 기능을 이중으로 사용하는 이중 포인터 구조를 통해 Python은 리스트안에 동일한 데이터 타입만 넣을 필요가 없어짐.

range함수 활용하기

만약 1부터 100까지 100개의 항목을 리스트로 만들려면 어떻게 할까? 이렇게 특정 패턴으로 연속된 많은 수를 일일이 다 기록해 주는 것은 매우 불편하고 비효율적인 방법이다. 이러한 경우 range 함수를 이용하는 것이 적합하다. range 함수는 for문을 배울 때 이미 접한 내용이다. range는 연속된 수열을 만들어 주는 함수이다. range함수는 기본적으로 시작값, 종료값, 증가값의 3개의 파라미터를 사용하여 연속된 값을 만들어주는 함수이다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = list(range(1,11,1))                #range(시작값, 종료값, 증가값)
print(a)
print(b)
```

위 코드를 실행하면 a와 b가 동일하게 출력되는 것을 알 수 있다.

리스트의 길이 확인하기

리스트의 길이를 확인할 경우 내장 함수인 len 함수를 사용하면 된다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = list(range(1,11,1))                #range(시작값, 종료값, 증가값)
len(a)
len(b)
```

인덱스로 항목 활용하기

리스트에 들어있는 항목들은 리스트 이름[위치 번호] 형식으로 항목의 값을 사용할 수 있다. 위치 번호는 보통 인덱스라고 부르며 항목의 위치를 의미하는 정수이다.

```
a = [10, 20, 30]
print(a[0])
print(a[1])
print(a[2])
```

위치 번호(index)는 0부터 시작함을 주의해야한다.

리스트 인덱스 사용하기

다시 한번 리스트에 대해서 정리해보자. 인덱스 번호는 0부터 시작한다. 따라서 리스트에 n개의 항목이 들어가 있을 경우 인덱스는 0 부터 시작하며 n-1까지 사용할 수 있다.

인덱스의 값을 사용하여 특정 항목의 값을 출력하거나 사용할 수 있을 뿐 아니라 해당 위치의 값을 변경시켜줄 수도 있다.

```
a = list(range(10,31,10))
sum = 0
ave = 0
for i in range(len(a))
sum += a[i]
ave = sum/len(a)
print(ave)
```

```
a = [10, 20, 30]
sum = a[0] + a[1] + a[2]
ave = sum/3
print(ave)
```

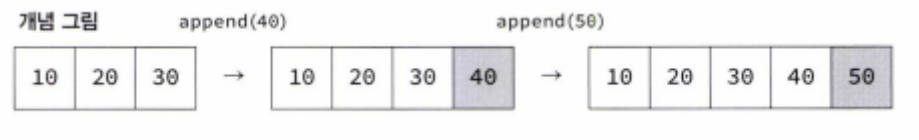
아래와 같이 특정 항목의 값을 변경시켜 줄 수도 있다.

```
a = [10, 20, 30]
a[0] = 100
a[1] = 200
a[2] = a[0] + a[1]
print(a)                    #[100, 200, 300] 출력
```

리스트 맨 뒤에 항목 추가하기

리스트의 맨 뒤에 항목(값)을 입력할 경우 append함수를 사용한다.

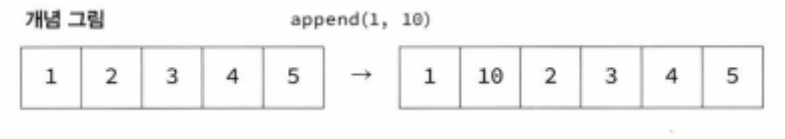
```
a = [10, 20, 30]
a.append(40)
a.append(50)
print(a)                    #[10, 20, 30, 40, 50] 출력
```



리스트 특정 위치에 항목 추가하기

리스트에서 제공하는 insert함수를 사용하여 리스트의 특정 위치에 값을 삽입할 수 있다. 이 때 기존의 값이 지워지는 것이 아니라 뒤로 미뤄지는 방식으로 삽입된다.

```
a = [1, 2, 3, 4, 5]
print(a)
a.insert(1, 10)
print(a)                    #[1, 10, 2, 3, 4, 5] 출력
```



리스트에서 제공하는 함수들은 다양하다. 추가적인 함수는 스스로 공부해보자.

함수명	기능	명령
append	리스트의 맨 뒤에 새로운 항목 추가	a.append(10)
insert	리스트의 특정 위치에 항목 삽입	a.insert(2,10)
extend	원래의 리스트에 다른 리스트 연결	a.extend([4,5,6])
count	리스트에 포함된 특정 값의 항목의 개수를 리턴	a.count(2)
remove	리스트에서 입력된 갑과 같은 항목을 모두 지운다	a.remove(2)
index	특정 값의 리스트에서의 위치, 즉 인덱스를 알려준다	a.index(3)
sort	리스트의 값들을 오름차순으로 정렬한다	a.sort()

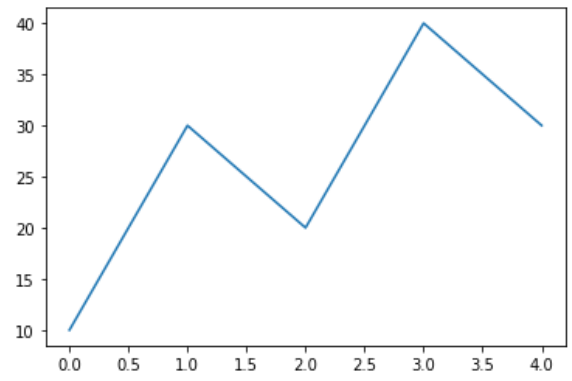
2.2 Matplotlib with Numpy & Pandas

2.2.1 Matplotlib 라이브러리 기초

Matplotlib는 Python 데이터 시각화의 대표적 라이브러리이다.

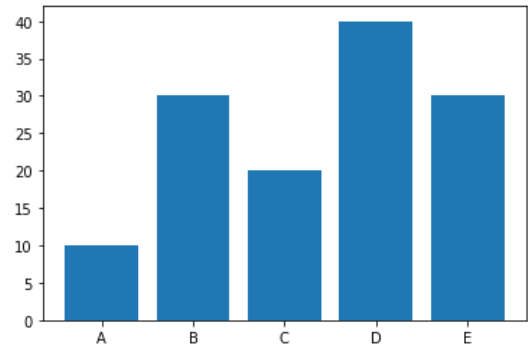
간단한 Plot 차트 그리기

```
import matplotlib.pyplot as plt       #matplotlib라이브러리 호출하여 plt로 정의
values = [10, 30, 20, 40, 30]         #리스트로 데이터 생성
plt.plot(values)       #values값을 y값으로 하여 차트 생성, x값은 0부터 차례대로 생성
plt.show()       #생성된 차트 표시
```



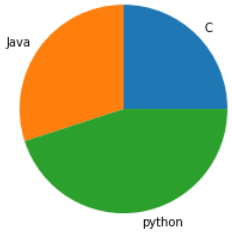
간단한 bar차트 그리기

```
import matplotlib.pyplot as plt


items = [‘A’, ‘B’, ‘C’, ‘D’, ‘E’]

values = [10, 20, 30, 40, 50]


plt.bar(items, values)              #bar형 차트 생성, plt.bar(x축 데이터, y축 데이터)

plt.show()
```



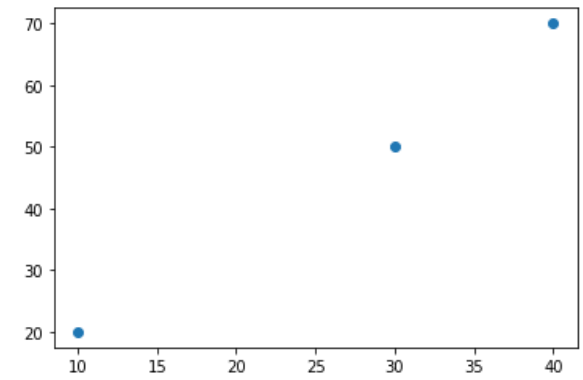
간단한 pi 차트 그리기

```
import matplotlib.pyplot as plt


lang = [‘C’, ‘Java’, ‘Python’]

count = [25, 30, 45]


plt.pie(count, labels = lang)   #pi형 차트 생성, plt.pie(비율, labels = 이름 붙일 데이터)

plt.show()
```



간단한 Scatter 차트 그리기

```
import matplotlib.pyplot as plt


x = [10, 30, 40]

y = [20, 50, 70]


plt.scatter(x,y)        #scatter형 차트 생성, plt.scatter(x축 데이터, y축 데이터)

plt.show()
```



차트를 표현하는 기본 방식 10단계

시각화를 위해 Matplotlib라이브러리 임포트

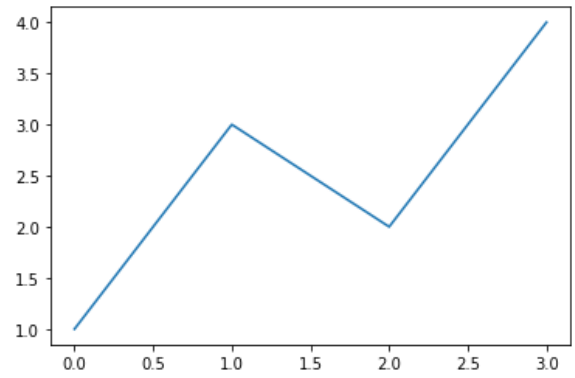
```
import matplotlib.pyplot as plt
```

리스트 자료형 형태로 데이터 생성

```
import matplotlib.pyplot as plt

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

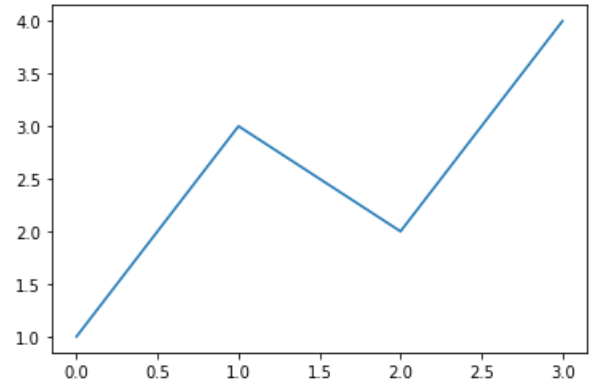
차트 생성

```
import matplotlib.pyplot as plt

data = [1,3,2,4]

plt.plot(data)  #plot 함수의 파라미터로 값이 존재하는 리스트 변수 입력(y축 데이터)

# 참조 x축 범위는 자동으로 생성(0부터 시작하여 1씩 증가)
```



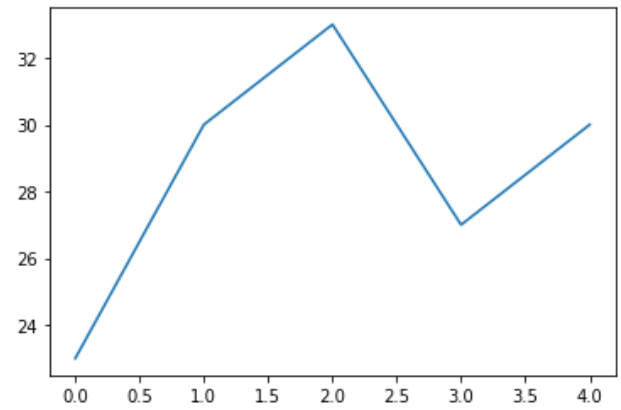
그래프 생성

```
import matplotlib.pyplot as plt

data = [1,3,2,4]

plt.plot(data)  #plot 함수의 파라미터로 값이 존재하는 리스트 변수 입력(y축 데이터)

plt.show()

# 참조 show()로 그래프를 호출하는 것이 원칙
```



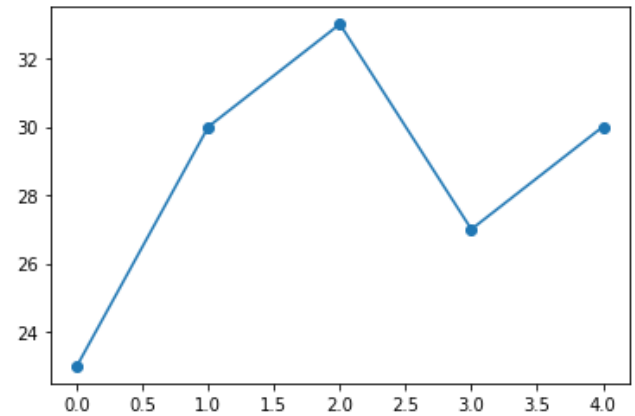
값을 변경해보기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data)

plt.show()
```



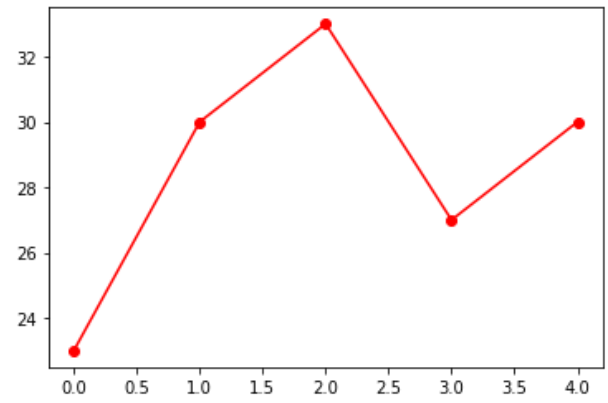
점(marker) 표시하기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data, marker = 'o')

plt.show()
```



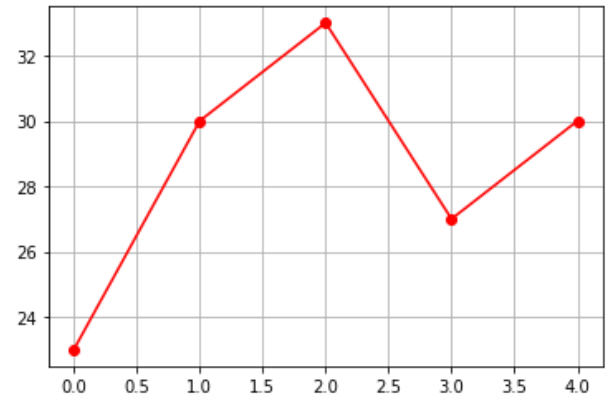
색상 변경하기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data, marker = 'o', color = 'red')

plt.show()
```



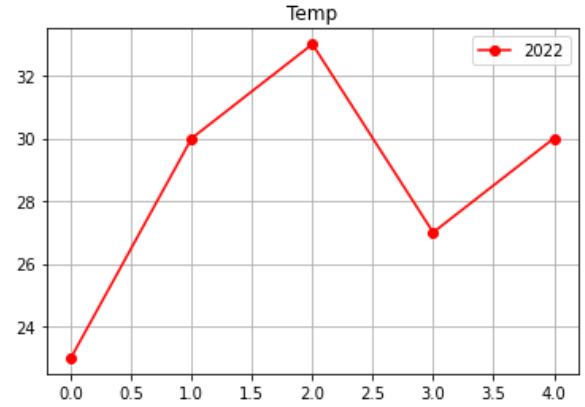
격자(grid) 표시하기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data, marker = 'o', color = 'red')

plt.grid()      #격자를 표시하는 함수, plt.grid(True)로 하여도 무관

plt.show()
```



제목 표시하기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data, marker = 'o', color = 'red', label = '2022')

plt.grid()      #격자를 표시하는 함수, plt.grid(True)로 하여도 무관

plt.title("Temp")    #제목을 표시하는 함수

plt.legend()      #plt.plot의 label을 표시하는 함수

plt.show()
```



파일로 저장하기

```
import matplotlib.pyplot as plt

data = [23, 30, 33, 27, 30]

plt.plot(data, marker = 'o', color = 'red', label = '2022')

plt.grid()      #격자를 표시하는 함수, plt.grid(True)로 하여도 무관

plt.title("Temp")    #제목을 표시하는 함수

plt.legend()      #plt.plot의 label을 표시하는 함수

plt.savefig('기후변화.png')  #차트를 이미지 파일로 저장, plt.savefig('파일이름.확장자')

plt.show()
```

Matplotlib 실습

```
import matplotlib.pyplot as plt
plt.plot([23, 30, 33, 27, 30])   #plot함수의 파라미터로 리스트를 작성하여도 사용가능
plt.show()
```

```
import matplotlib.pyplot as plt

부서A판매량 = [10, 30, 20, 40]
부서B판매량 = [20, 35, 27, 32]

# 색상의 경우 r,g,b형태 외 red, yellow, olive 등 색상 이름으로 지정 가능
# 참조 x축 데이터를 따로 넣어주지 않으면 범위는 자동으로 생성(0부터 시작하여 1씩 증가)

plt.plot(부서A판매량, marker = 'o', color = 'r', markersize = 10, lable = 'A')
plt.plot(부서B판매량, marker = '*', color = 'b', markersize = 10, lable = 'B')

plt.tile("Sales Volume Comparison")    #제목을 표시하는 함수
plt.grid()      #격자를 표시하는 함수

plt.xlim(0,5)    #x축 범위
plt.ylim(0,50)    #y축 범위

plt.xlabel('Worker')   #x축 이름
plt.ylabel('Result')    #y축 이름
plt.legend() #plt.plot의 label을 표시하는 함수
plt.show()
```

```
import matplotlib.pyplot as plt

부서A판매량 = [10, 30, 20, 40]
부서B판매량 = [20, 35, 27, 32]
사원번호 = ['101', '102', '103', '104']

plt.plot(사원번호, 부서A판매량, marker = 'o', color = 'r', markersize = 10, lable = 'A')

plt.tile("Sales Volume Comparison")    #제목을 표시하는 함수
plt.grid()      #격자를 표시하는 함수

plt.xlim(0,5)    #x축 범위
plt.ylim(0,50)    #y축 범위

plt.xlabel('Worker')   #x축 이름
plt.ylabel('Result')    #y축 이름
plt.legend() #plt.plot의 label을 표시하는 함수

plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

temp1994=[39.6,37.9,37.9,34.9,33.9,35.3,35.9,35.3,32.5,36.8]
temp2018=[29.3,33.9,32.5,32.4,35.6,35.9,33.6,33.8,37.0,29.3]

# plt.plot(리스트 이름, marker='모양', color= '색', markersize = 크기, label = '이름 )
plt.plot(temp1994, marker='o', color= 'b', markersize = 5, label = '1994 ')
plt.plot(temp2018, marker='o', color = 'orange', markersize = 5, label = '2018')
```

```
plt.title('Temp')


plt.xlabel('Day') #x축 이름
plt.ylabel('Temp') #y축 이름


plt.ylim(25,41)
plt.xlim(0,9)


plt.grid()


plt.legend()


#x축 값의 표시단위 xticks함수, np.arnage(0,10,1)는 0부터 9까지 1씩 증가하며 값 생성
plt.xticks(np.arange(0,10,1))


#y축 값의 표시단위 yticks함수, np.arnage(25,42,2)는 25부터 41까지 2씩 증가하며 값 생성
plt.yticks(np.arange(25,42,2))


plt.show()
```

2.2.2 Pandas

Pandas란 데이터 분석을 위한 Python 라이브러리이다. 대용량의 데이터 처리를 지원한다.

1. 자동화된 분석을 지원
2. 대용량의 데이터 처리 지원
3. 머신러닝, 시각화 등의 데이터사이언스 관련 라이브러리에서 사용

Pandas를 활용하기 위해서는 Pandas가 지원하는 DataFrame이라는 구조에 데이터를 입력해야 한다.

```
import pandas as pd
stdData = [['kim', 20, '서울시'], ['lee', 25, '경기도'], ['park', 22, '제주도'], ['choi', 19, '강원도']]
df = pd.DataFrame(stdData)          # DataFrame()함수; 리스트로 DataFrmae 생성하여 df변수에 저장
df
```

```
import pandas as pd

이름 = ['kim', 'lee', 'park', 'choi']
나이 = [20, 25, 22, 19]
주소 = ['서울시', '경기도', '제주도', '강원도']

df2 = pd.DataFrame({'name': 이름, 'age': 나이, 'addr': 주소})
df2
```

위 소스코드는 동일한 결과를 도출하는 것을 실습을 통해 알아볼 수 있다.

Pandas로 Excel파일 불러오기

```
from google.colab import files

myfile = files.upload()
```

```
import io

import pandas as pd

birthData = pd.read_excel('data.xlsx')

birthData
```

데이터 보기: head(), tail()

```
birthData.head()  #기본적으로 5개의 레코드(행) 표시, 변수이름.head()
```

```
birthData.tail()  #기본적으로 5개의 레코드(행) 표시, 변수이름.tail()
```

```
birthData.head(7)  #지정된 개수만큼 표시
```

```
birthData.tail(3)  #지정된 개수만큼 표시
```

info 함수로 데이터 파악하기

```
birthData.info()  #변수이름.info
```

describe 함수로 데이터 파악하기

```
birthData.describe() #변수 이름.describe
```

행과 열 일부 선택하기

Pandas로 Excel파일 불러오기

```
from google.colab import files

myfile = files.upload()
```

```
import io

import pandas as pd

birthData = pd.read_excel('data.xlsx')    #pd.read_excel('파일이름.확장자')

birthData
```

특정 레코드(행) 선택하기

```
birthData[0:3]   #변수이름[시작할 행:종료 행+1], 리스트에서 슬라이싱 하는 방식과 동일
```

특정 칼럼(열) 선택하기

```
birthData.연도   #방식1
```

```
birthData.['연도']      #방식2: 따옴표, 칼럼 이름에 띄어쓰기가 있는 경우 사용
```

```
birthData.[['연도', '출생아수']]      # 여러 개의 칼럼(열) 선택하기, 대괄호 2개 사용
```

특정 행과 열 선택하기

```
birthData.연도[0:5]     #birthData.['연도'][0:5]와 동일
```

데이터분석 실습

```
from google.colab import files

myfile = files.upload()
```

```
import io

import pandas as pd

df = pd.read_excel('data.xlsx')    #pd.read_excel('파일이름.확장자')
```

```
import matplotlib.pyplot as plt

plt.plot(df['연도'], df['출생아수']

plt.show
```

```
data1950 = df[0:9]
```

```
import matplotlib.pyplot as plt

plt.plot(data1950['출생아수'], marker = 'o', markersize = 7)

plt.grid()

plt.ylim(600_000, 1_200_000)

plt.show()
```

```
data = df['출생아수'][29:50]

plt.plot(data, marker = 'o', markersize = 7)

plt.grid()

plt.show()
```

조건에 맞는 데이터 선택하기: query 함수

```
birthData.query('1980<=연도<=2000')  # 1980부터 2000년까지 데이터 선택
```

```
birthData.query('연도>=2000 and 출생아수>=500000')  # 2000년 이후 50만명 이상 출생한 데이터
```

```
data2 = birthData.query('연도>=2000 and 출생아수>=500000')
plt.plot(data2['연도'], data2['출생아수'], marker = 'o', markersize = 7)
plt.grid()
plt.show()
```

2.2.3 Numpy

Numpy는 과학적 계산을 위한 파이썬 외부 라이브러리로 데이터 분석, 시각화, 머신러닝 등의 작업에 필수적이다. 벡터, 행렬 등의 자료구조 및 연산 지원 역할을 수행한다. 제공하는 함수로 통계함수(최대, 최소, 평균, 중간값, 분산, 표준편차 등), 수학 함수(삼각함수, 로그함수 등), 벡터 및 행렬연산(행렬의 곱, 역행렬, 전치행렬 등), 공학수학, 선형대수학 등이 있다.

Numpy를 통해 우리는 데이터를 생성하고, 많은 데이터를 쉽고 빠르게 처리하며, 복잡한 연산을 쉽게 수행할 수 있다.

Numpy를 사용하는 이유

Question 5명의 학생이 특강수업에 참여하고 있다. 이 수업은 중간고사 50% 기말고사 50%의 비율로 최종 점수가 계산된다. 중간고사와 기말고사를 평균한 결과를 도출하고 전체적으로 7점을 가산점을 적용한 값을 구하시오.

- 중간고사 성적: [90, 80, 70, 60, 50]
- 기말고사 성적: [80, 70, 60, 50, 40]

Numpy 사용 안하는 경우	Numpy 사용

중간 = [90, 80, 70, 60, 50]
 	
기말 = [80, 70, 60, 50, 40]
 	
최종 = []
 	
n = len(중간)
 	
for i in range(n):
 	
  v = (중간[i] + 기말[i])/2
 	
  최종.append(v)
 	
for I in range(n)
 	
  최종[i] = 최종[i] + 7
 	
print(최종)
	중간 = [90, 80, 70, 60, 50]
	
	기말 = [80, 70, 60, 50, 40]
	
	mid = np.array(중간)
	
	fin = np.array(기말)
	
	score = (mid + fin)/2 + 7
	
	print(score)

Numpy Array 생성하기

리스트를 만든 후 Array로 변환

```
data1 = [1,2,3,4,5]
data1
```

```
arr1 = np.array(data1)
arr1
```

```
type(data1)
```

```
type(arr1)
```

수열과 그래프 표현

수열 생성

- **시작값, 종료값, 증가값을 고려한 Array 생성**
  - ✓ np.arange(시작값, 종료값, 증가값)
  - ✓ 실수 값 지원, list가 아니라 Array 형태로 생성
- **시작값, 종료값, 구분수를 통한 Array 생성**
  - ✓ np.linspace(시작값, 종료값, 구분수)
  - ✓ 여기서 구분수는 등급의 개수, 100으로 하면 100개의 구간으로 나누는 수열 생성

간단한 그래프 생각해보기

```
xlist = [0,1,2,3]
ylist = [1,3,5,7]


plt.plot(xlist,ylist)
plt.xlim(0.7)
plt.ylim(0.7)


plt.show()
```

```
x= np.arange(0,4,1)
y = 2*x+1
plt.plot(x,y)
plt.show()
```

```
xarr = np.arange(0, 3.14*2, 0.1)

ysin = np.sin(xarr)

ycos = np.cos(xarr)


plt.plot(xarr, ysin, label = 'sin')

plt.plot(xarr, ycos, label = 'cos')


plt.title('sin&cos graphs')

plt.xlabel('x value')

plt.ylabel('y value')


plt.legend()

plt.grid()


plt.show()
```

np.linspace 함수 이해하기

```
a = np.linspace(1, 10, 10)
print(a)
```

```
a = np.linspace(0, 25, 30)
print(a)
```

```
xarr = np.linspace(-np.pi, np.pi, 1000)

ysin = np.sin(xarr)

ycos = np.cos(xarr)



plt.plot(xarr, ysin, label = 'sin')

plt.plot(xarr, ycos, label = 'cos')



plt.title('sin&cos graphs')

plt.xlabel('x value')

plt.ylabel('y value')



plt.legend()

plt.grid(True)



plt.show()
```

Numpy 활용

```
scores = []


for i in range(0,5):
    val = int(input('%d번 성적입력:' %(i+1)))
    scores.append(val)


print('\n입력된 전체점수')
print(scores)
```

```
sumr = np.sum(scores) #합계
aver = np.mean(scores) #평균
varr = np.var(scores) #분산
stdr = np.std(scores) #표준편차


print(sumr)
print(aver)
print(varr)
print(stdr)
```

```
#정규분포에 따라 난수 생성 np.random.normal(평균, 표준편차, 생성 수)
data4 = np.random.normal(170, 10, 30)
data4
```

```
#초기값, 종료값 사이 난수 생성 np.random.randint(초기값, 종료값, 생성 수)
data5 = np.random.randint(1,7,100) =
print(data5)
```

2.3 Seaborn with Numpy & Pandas

2.3.1 Seaborn 라이브러리 기초

데이터 로딩, 구조 이해

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


df = sns.load_dataset('iris')
df
```

```
df.head()
```

```
df.describe()
```

```
df.info()
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


titanic = sns.load_dataset('titanic')
titanic
```

Relplot 차트

```
sns.set_style('whitegrid')


sns.relplot(data = titanic, x = 'age', y = 'fare', hue = 'sex')

plt.show()
```

```
sns.set_style('whitegrid')


sns.relplot(data = titanic, x = 'age', y = 'fare', kind = 'line', hue = 'sex')

plt.show()
```

Lineplot 차트

```
sns.lineplot(data = titanic, x = 'age', y = 'fare', color = 'black' )

plt.show()
```

Distplot 차트, 히스토그램

```
sns.distplot(titanic.fare.dropna()) #.dropna()는 빈 데이터 삭제

plt.show()
```

```
pilter = titanic.query('fare<=150')

sns.distplot(pilter.fare.dropna()) #.dropna()는 빈 데이터 삭제

plt.show()
```

Countplot 차트

```
sns.countplot(titanic.who)
plt.show()
```

```
sns.countplot(titanic.sex)
plt.show()
```

III. 데이터 분석 심화

3.1 Machine Learning with sklearn

Scikit-learn(sklearn)은 Python 프로그래밍 언어용 자유 소프트웨어 머신러닝 라이브러리이다. 다양한 분류, 회귀, 그리고 서포트 벡터 머신, 랜덤 포레스트, 그라디언트 부스팅을 포함한 클러스터링 알고리즘을 특징으로 하며, Python의 수치 및 과학 라이브러리 Numpy 및 Scipy와 함께 운용되도록 설계되었다.

3.1.1 결정트리 머신러닝 실습

자동차, 오토바이, 자전거 구별하는 결정트리 생성


```
from sklearn import tree #sklearn라이브러리 중 tree 호출

#입력데이터(x[0]=바퀴개수, x[1]=크기;0=작다 1=크다, x[2]=동력;1=엔진)

x = [[2,0,1],[2,0,0],[4,1,1],[4,0,1]]

y = [1,2,0,0] #class(종류), 자동차=0, 오토바이=1, 자전거=2

clf = tree.DecisionTreeClassifier() #결정트리 함수 호출하여 clf변수에 저장

clf = clf.fit(x,y) #결정트리 모델 생성
```

```
clf.predict([[2,1,1]]) #새로운 데이터에 대한 예측
```

바이러스 판정 결정트리 생성


```
from sklearn import tree #sklearn라이브러리 중 tree 호출

x = [[0,0],[1,1],[1,0]] #입력데이터(x[0]=증상1, x[1]=증상2, 데이터 = 1 -> 양성)

y = [0, 1, 0] #class(종류), 음성=0, 양성=1

virusclf = tree.DecisionTreeClassifier() #결정트리 함수 호출하여 virusclf변수에 저장

virusclf = virusclf.fit(x,y) #결정트리 모델 생성
```

```
virusclf.predict([[1,0]]) #새로운 데이터에 대한 예측
```

```
virusclf.predict([[0.7, 0.6]]) #새로운 데이터에 대한 예측
```

```
virusclf.predict([[0,1]]) #새로운 데이터에 대한 예측
```