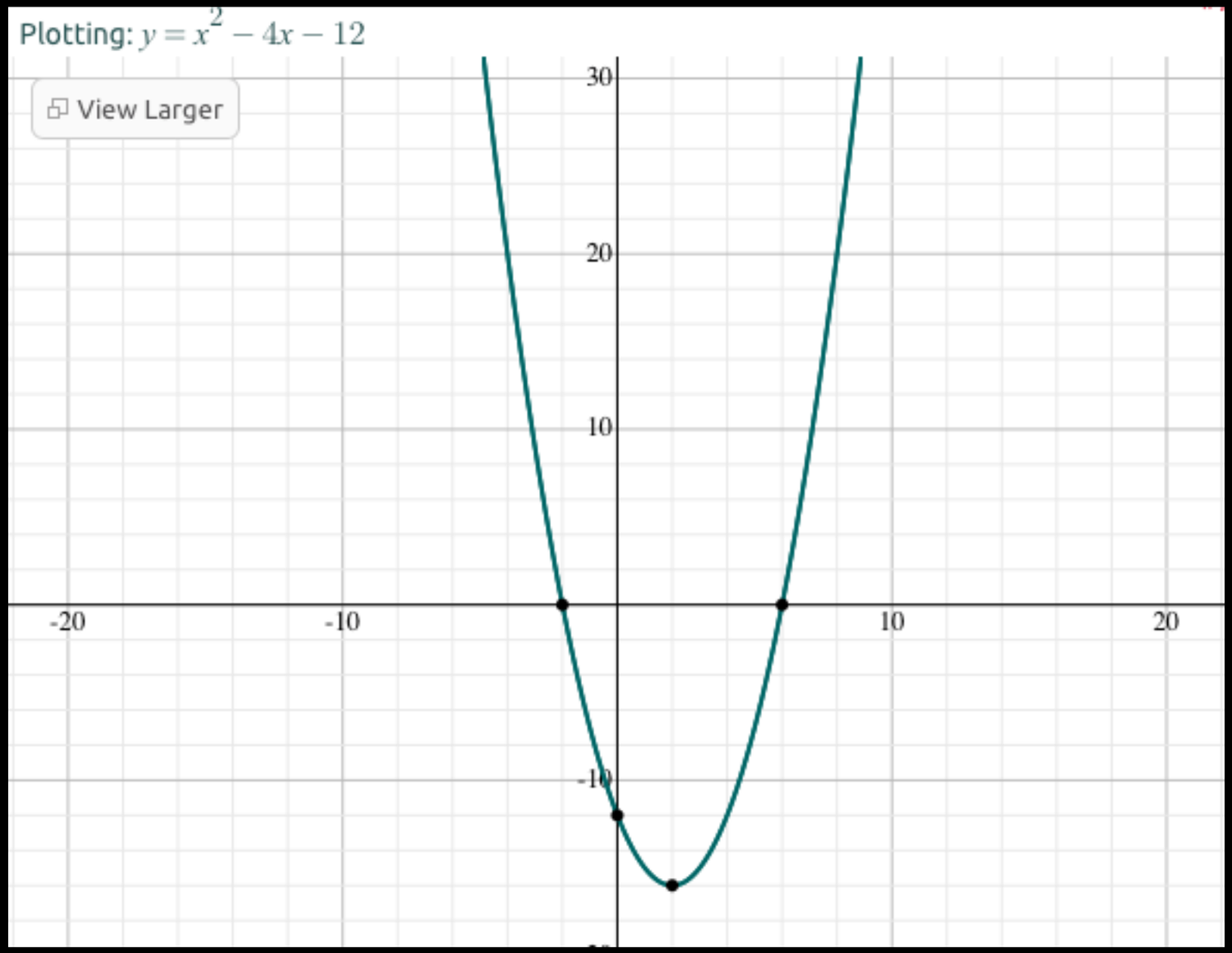
함수

클래스

함수



f(x) = y = x^2 - 4x - 12

파이썬에서 사용하는 함수는? 이런게 아니죠 ㅎㅎ

함수 형태

```
In [87]: def 함수이름(매개변수):
    ...:     (실행문)
    ...:     (실행문)
    ...:     (실행문)
    ...:     return (내보낼값)
```

함수는 자주사용하는 코드를 한 곳에 모아놓은 것을 말합니다.

그리고 함수 바깥에서 전달받은 값이 저장되는 변수를 매개변수 라고 부릅니다.

바깥에서 함수를 호출할때 전달하는 값은 인수 라고 부릅니다.

```
add(10, 20)     # 10과 20이 인수
```

```
In [99]: def sum (a, b):
    ...:     return a+b
    ...:

In [100]: c=sum(1,2)

In [101]: print(c)
3

In [102]:
```

```
In [107]: def python():
     ...:     print("hello python")
     ...:

In [108]: python()
hello python

In [109]:
```

함수의 매개변수가 없어도 가능

return이 없어도 가능

```
In [119]: def mul(*args):
     ...:     sum=1
     ...:     for i in args:
     ...:         sum=sum*i
     ...:     return sum
     ...:

In [120]: mul(1,2,3,4,5,6,7)
Out[120]: 5040

In [121]:
```

다음과 같이 매개변수에 * 표시를 하면 원하는
만큼 유동적으로 파라미터를 받을 수 있습니다!

*args가아닌다른 문자를 써도 이상없지만,
*args를관례적으로 자주 사용합니다
부를때는 아규먼트(argument)라고 합니다.

```
In [124]: def get_kwarg(**kwarg):
     ...:     print(kwarg)
     ...:

In [125]: get_kwarg(a=1, name='python', sport='baseball')
{'a': 1, 'name': 'python', 'sport': 'baseball'}

In [126]:
```

파라미터에 **표시를 하게되면,
keyword가 있는 형태의 파라미터를 받을 수 있습니다.

**kwargs역시 다른 문자로도 사용 가능하지만
관례적으로 **kwargs를 사용합니다.

이친구는 아규먼트 앞에 k를 붙여서
키워드 아규먼트 라고 부릅니다.

```
In [129]: def use_arg_kwarg(*args,**kwargs):
     ...:     print(args)
     ...:     print(kwargs)
     ...:

In [130]: use_arg_kwarg(2,4,6,name='asdf',school='cbnu')
(2, 4, 6)
{'name': 'asdf', 'school': 'cbnu'}
```

*arg와 **kwarg 동시에 사용 가능

***args**

arguments의 약자
키워드가 없는 형태의 파라미터를 넘기기 위해 사용

****kwargs**

Key worded argument의 약자
키워드가 있는 형태의 파라미터를 넘기기 위해 사용

```
In [140]: def calculator(mode, *args):
     ...:     if mode==1:
     ...:         answer=0
     ...:         for i in args:
     ...:             answer+=i
     ...:
     ...:     elif mode==2:
     ...:         answer=1
     ...:         for i in args:
     ...:             answer*=i
     ...:     else:
     ...:         print("잘 못 된  메 뉴  선 택 입 니 다 ")
     ...:     return answer
     ...:

In [141]: calculator(1,1,2,3,4)
Out[141]: 10

In [142]: calculator(2,3,2,4)
Out[142]: 24
```

일반 변수와도 함께 사용 가능

간단하게 중간정리!

함수 : 자주사용하는 코드를 한곳에 모아놓은 것

매개변수 : 함수의 동작에 필요한 변수를 외부로부터
받아오는 변수

*args

리스트 형태로 외부에서
주는 만큼 다 받아준다.

*kargs

Args와 같은 기능이지만
딕셔너리 자료형으로 받는다.

사용자가 프로그램 내에서 변수를 정해줄 때도 있지만,
실시간 대화형식으로 변수를 입력받을 수도 있습니다.

```
In [144]: key=input()
hello python

In [145]: key
Out[145]: 'hello python'

In [146]:
```

다음과 같이 변수 = input()
매서드를 이용해서 사용자로부터 입력을 받을 수 있습니다

```
In [153]: key=input("마 한번 입력해봐라 : ")
마 한번 입력해봐라 : 든킨드나쓰

In [154]: key
Out[154]: '든킨드나쓰'

In [155]:
```

input 괄호 안에 질문 내용을 입력할 수 도 있습니다.

이를 이용해서 유연하게 사용자로부터
변수를 입력받을 수 도 있습니다!

클래스와 객체

Class and Object

클래스

클래스는 객체를 표현하기 위한 문법입니다.

객체

미리 정의된 클래스에 의해 생성되는 변수!!!

객체가 어떤 건지 많이들 어려워 하는데
간단하게 변수라고 생각하시면 쉽습니다

Class -> 붕어빵틀
Object -> 붕어빵

붕어빵과 붕어빵틀을 흔히 예시로 드는 것중 하나인데요

클래스라는 틀을 통해 객체를 계속해서 찍어낼수 있다.
라고 이해하시면 됩니다.

클래스 사용 방법

```
class 클래스이름:          # 클래스 만들기
    def 메서드(self):     # 메서드 만들기
        코드
```

클래스는 class에 클래스 이름을 지정하고

:(콜론)을 붙인 뒤 다음 줄부터

def로 메서드를 작성합니다.

메서드는 클래스 안에 들어있는 함수를 뜻합니다.

객체 사용 방법

```
인스턴스 = 클래스()      # 인스턴스(객체) 만들기
인스턴스.메서드()        # 인스턴스로 메서드 호출
```

클래스는 ()(괄호)를 붙인 뒤 변수에 할당하여 인스턴스(객체)를 만듭니다. 그리고 인스턴스 뒤에 .(점)을 붙여서 메서드를 호출합니다.

클래스(Class) 정의

```python
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()
```

객체(Object)
붕어빵

클래스(Class)
붕어빵 틀

객체는 클래스로부터 만들어 진다

꼭 기억하세요

생성자

객체가 생성 될때 실행되는 메소드

소멸자

객체가 메모리상에서 삭제될 때 실행되는 메소드
즉 프로그램이 종료될 때 실행

생성자 메소드

def __init__(self)

소멸자 메소드

def __del__(self)

아래 코드를 통해 직접 생성자와 소멸자의 실행 위치를 확인해 보세요

```
class BreadMold:
    a = "hello class"

    def __init__(self):
        print("생성자")

    def __del__(self):
        print("소멸자")

    def mymethod(self):
        print("클래스 내부변수는 : "+self.a)


bread = BreadMold()
bread.mymethod()
bread2 = BreadMold()
```

상속

Inheritance

상속 : 물려받다.

B클래스는 A클래스를 물려받다.

상속을 통해서 상위 클래스의 기능들을 물려 받을 수 있습니다

상위 클래스의 기능을
따로 구현하지 않고 **바로 사용할 수 있습니다**

```
class 기반클래스이름:
    코드

class 파생클래스이름(기반클래스이름):    # 기반 클래스를 상속받음
    코드
```

아래 코드를 통해 직접 상속의
기능을 확인해보세요

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def study(self):
        print('공부하기')

james = Student()
james.greeting()  # 기반 클래스 Person의 메서드 호출
james.study()     # 파생 클래스 Student에 추가한 study 메서드
```

더 자세히 설명해준 사이트

https://wikidocs.net/28