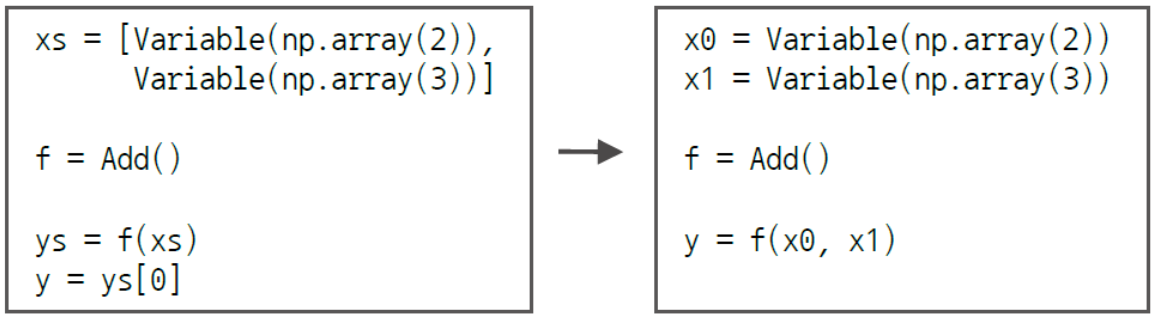
제2 고지 : 자연스러운 코드로

STEP 12 : 가변길이 인수 (개선 편)

12.1 첫 번째 개선 : 함수를 사용하기 쉽게

그림 12-1 현재의 코드(왼쪽)와 개선 후의 코드(오른쪽)



현재의 `Add` 클래스는 인수를 리스트에 모아서 받고 결과는 튜플에 반환하는데, 이것보단 **오른쪽 그림처럼 인수와 결과를 직접 주고 받는 편이 훨씬 자연스럽다.**

1. 함수를 정의할 때 인수에 `*` 를 붙이면 호출할 때 넘긴 인수들을 `*` 를 붙인 인수 하나로 모아서 받을 수 있다.
2. `outputs` 에 원소가 하나 뿐이면 리스트가 아닌 그 원소만을 반환한다.

In [ ]:

```python
import numpy as np


def as_array(x):

    """
    0차원 ndarray / ndarray가 아닌 경우
    """
    if np.isscalar(x):
        return np.array(x)
    return x


class Variable:
    def __init__(self, data: np.ndarray) -> None:
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError(f"{type(data)}은(는) 지원하지 않습니다.")
        self.data = data
        self.grad = None  # gradient
        self.creator = None  # creator

    def set_creator(self, func) -> None:
        self.creator = func

    def backward(self):
        """
        자동 역전파 (반복)
        """
        if self.grad is None:
            self.grad = np.ones_like(self.data)
```

```
        funcs = [self.creator]
        while funcs:
            f = funcs.pop()  # 1. 함수를 가져온다
            x, y = f.input, f.output  # 2. 함수의 입력 / 출력을 가져온다
            x.grad = f.backward(y.grad)  # 3. 역전파를 계산한다

            if x.creator is not None:
                funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다.


class Function:
    """
    Function Base Class

    """

    def __call__(self, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ys = self.forward(xs)
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, xs):
        x0, x1 = xs
        y = x0 + x1
        return (y,)
```

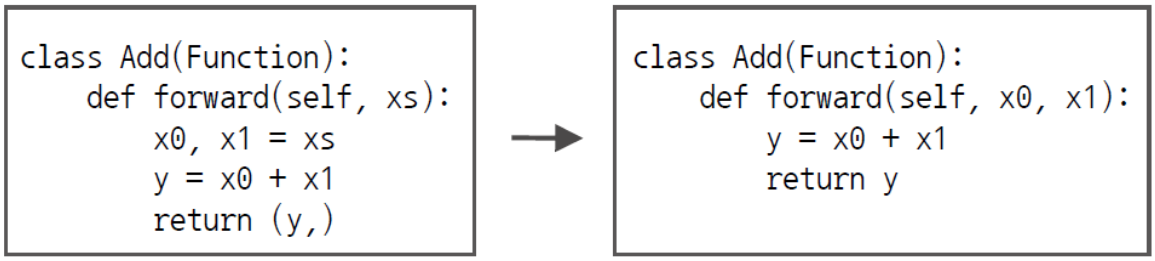
In [ ]:

```
x0 = Variable(np.array(2))
x1= Variable(np.array(3))
f = Add()
y= f(x0,x1)
print(y.data)
```

```
5
```

12.2 두 번째 개선 : 함수를 구현하기 쉽도록

그림 12-2 현재의 코드(왼쪽)와 개선 후의 코드(오른쪽)



현재 인수는 리스트로 전달되고 결과는 튜플을 반환하고 있다. 이보단, 입력도 변수를 직접 받고, 결과도 직접 변수를 돌려줄 수 있도록 개선한다.

1. 함수를 호출할 때 `*` 를 활용하여 **리스트 언팩(list unpack).**
2. `ys` 가 튜플이 아닌 경우 튜플로 변경.
3. `forward()` 에서는 출력 원소가 하나 뿐이라면 해당 원소를 직접 반환.

In [ ]:

```python
import numpy as np


def as_array(x):

    """
    0차원 ndarray / ndarray가 아닌 경우
    """
    if np.isscalar(x):
        return np.array(x)
    return x


class Variable:
    def __init__(self, data: np.ndarray) -> None:
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError(f"{type(data)}은(는) 지원하지 않습니다.")
        self.data = data
        self.grad = None  # gradient
        self.creator = None  # creator

    def set_creator(self, func) -> None:
        self.creator = func

    def backward(self):
        """
        자동 역전파 (반복)
        """
        if self.grad is None:
            self.grad = np.ones_like(self.data)
        funcs = [self.creator]
        while funcs:
            f = funcs.pop()  # 1. 함수를 가져온다
            x, y = f.input, f.output  # 2. 함수의 입력 / 출력을 가져온다
            x.grad = f.backward(y.grad)  # 3. 역전파를 계산한다

            if x.creator is not None:
                funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다.


class Function:
    """
```

```
    Function Base Class

    """

    def __call__(self, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ####################################
        ys = self.forward(*xs) # 1. 리스트 언팩
        if not isinstance(ys,tuple): # 2. 튜플이 아닌 경우 추가 지원
            ys = (ys,)
        ####################################
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, x0,x1):
        y = x0 + x1
        return y
```

12.3 add 함수 구현

In [ ]:

```
import numpy as np


def as_array(x):

    """
    0차원 ndarray / ndarray가 아닌 경우
    """
    if np.isscalar(x):
        return np.array(x)
    return x


class Variable:
    def __init__(self, data: np.ndarray) -> None:
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError(f"{type(data)}은(는) 지원하지 않습니다.")
        self.data = data
        self.grad = None  # gradient
        self.creator = None  # creator
```

```
    def set_creator(self, func) -> None:
        self.creator = func

    def backward(self):
        """
        자동 역전파 (반복)
        """
        if self.grad is None:
            self.grad = np.ones_like(self.data)
        funcs = [self.creator]
        while funcs:
            f = funcs.pop()  # 1. 함수를 가져온다
            x, y = f.input, f.output  # 2. 함수의 입력 / 출력을 가져온다
            x.grad = f.backward(y.grad)  # 3. 역전파를 계산한다

            if x.creator is not None:
                funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다.


class Function:
    """
    Function Base Class

    """

    def __call__(self, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ######################################
        ys = self.forward(*xs) # 1. 리스트 언팩
        if not isinstance(ys,tuple): # 2. 튜플이 아닌 경우 추가 지원
            ys = (ys,)
        ######################################
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, x0,x1):
        y = x0 + x1
        return y

def add(x0,x1):
    return Add()(x0,x1)
```

In [ ]:

```
x0, x1 = Variable(np.array(2)), Variable(np.array(3))
y = add(x0, x1)
print(y.data)
```

```
5
```