STEP 11 가변길이 인수 (순전파 편)

- 가변길이 입출력을 처리할 수 있도록 확장
- 인수와 반환값의 타입을 리스트로 바꾸고 필요한 변수들을 이 리스트에 넣음

11.2 Add 클래스 구현

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

xs = [Variable(np.array(2)), Variable(np.array(3))]    # 리스트로 준비
f = Add()
ys = f(xs)    # ys 튜플
```

STEP 12. 가변길이 인수 (개선 편)

STEP 13. 가변길이 인수 (역전파 편)

13.1 가변길이 인수에 대응한 Add 클래스의 역전파

- 덧셈의 역전파 : 상류에서 흘러오는 미분값을 그대로 '흘려보내는' 것.

13.2 Variable 클래스 수정

- 여러 개의 변수에 대응할 수 있도록 수정

```
while funcs:
    f = funcs.pop()
    # 출력 변수인 outputs 에 담겨있는 미분값들을 리스트에 담음.
    gys = [output.grad for output in f.outputs]
    gxs = f.backward(*gys)      # 함수 f의 역전파 호출 (리스트 언팩)
    if not isinstance(gxs, tuple):      # gxs가 튜플이 아니라면 튜플로 변환
        gxs = (gxs, )

    # 역전파된 미분값을 Variable의 인스턴스 변수 grad에 저장.
    for x, gx in zip(f.inputs, gxs):
        x.grad = gx

        if x.creator is not None:
            funcs.append(x.creator)
```

13.3 Square 클래스 구현

- add, square 함수 실제 사용

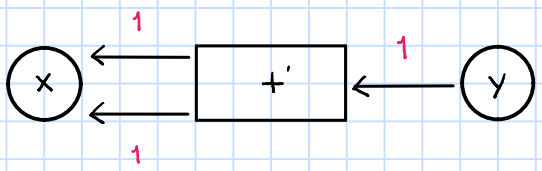
$z = x^2 + y^2$ 계산

```
x = Variable(np.array(2.0))
y = Variable(np.array(3.0))

z = add(square(x), square(y))
z = backward()

print(z.data)
print(x.grad)
print(y.grad)
```

STEP 14. 같은 변수 반복 사용



- Variable 클래스 코드에 반영

```
for x, gx in zip(f.inputs, gxs):
    if x.grad is None:
        x.grad = gx
    else:
        x.grad = x.grad + gx    # 전달된 미분값을 더해줌
```

14.3. 미분값 재설정

- 메모리를 절약하고자 인스턴스를 재사용하는 경우 잘못된 미분값을 들려줄 수 있음.

- Variable 클래스에 미분값을 초기화하는 cleargrad 메서드 추가

```
class Variable :
    def __init__(self, data):
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError('{}은(는) 지원하지 않습니다.'.format(type(data)))

        self.data = data
        self.grad = None    # 미분값, None 초기화 (실제 역전파 시 미분값 계산하여 대입)
        self.creator = None

    def set_creator(self, func):
        self.creator = func

    def backward(self):
        if self.grad is None:
            self.grad = np.ones_like(self.data)

        funcs = [self.creator]
        while funcs:
            f = funcs.pop()     # 맨 마지막 함수
            # 출력 변수인 outputs 에 담겨있는 미분값들을 리스트에 담음.
            gys = [output.grad for output in f.outputs]
            gxs = f.backward(*gys)      # 함수 f의 역전파 호출 (리스트 언팩)
            if not isinstance(gxs, tuple):      # gxs가 튜플이 아니라면 튜플로 변환
                gxs = (gxs, )

            # 역전파된 미분값을 Variable의 인스턴스 변수 grad에 저장.
            for x, gx in zip(f.inputs, gxs):
                if x.grad is None:
                    x.grad = gx
                else:
                    x.grad = x.grad + gx    # 전달된 미분값을 더해줌

                if x.creator is not None:
                    funcs.append(x.creator)

    def cleargrad(self):
        self.grad = None
```

```
class Function:
    def __call__(self, *inputs):
        xs = [x.data for x in inputs]
        ys = self.forward(*xs)          # 별표를 붙여 unpack
        if not isinstance(ys, tuple):   # 튜플이 아닌 경우 추가 지원
            ys = (ys,)
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)    # output에 creator를 설정
        self.inputs = inputs  # 입력변수를 기억(보관)
        self.outputs = outputs  # 출력 기억(보관)
        # 리스트의 원소가 하나라면 첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, x):
        raise NotImplementedError()

    def backward(self, gy):
        raise NotImplementedError()
```

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

    def backward(self, gy):
        return gy, gy
```

```
class Square(Function):
    def forward(self, x):
        y = x ** 2
        return y

    def backward(self, gy):
        x = self.inputs[0].data
        gx = 2 * x * gy
        return gx
```