

## You

Write me a python code that solves the problem with such description: seven(times(five())) # must return 35  
 four(plus(nine())) # must return 13 eight(minus(three())) # must return 5 six(divided\_by(two())) # must return 3

## Copilot

Sure, here is a Python solution that uses classes and magic methods to solve the problem:

```
class Number(int):
    def __init__(self, num):
        super(Number, self).__init__()

    def plus(self, num):
        return Number(self + num)

    def minus(self, num):
        return Number(self - num)

    def times(self, num):
        return Number(self * num)

    def divided_by(self, num):
        return Number(self // num)

def zero(func=None): return Number(0) if not func else func(Number(0))
def one(func=None): return Number(1) if not func else func(Number(1))
def two(func=None): return Number(2) if not func else func(Number(2))
def three(func=None): return Number(3) if not func else func(Number(3))
def four(func=None): return Number(4) if not func else func(Number(4))
def five(func=None): return Number(5) if not func else func(Number(5))
def six(func=None): return Number(6) if not func else func(Number(6))
def seven(func=None): return Number(7) if not func else func(Number(7))
def eight(func=None): return Number(8) if not func else func(Number(8))
def nine(func=None): return Number(9) if not func else func(Number(9))
def ten(func=None): return Number(10) if not func else func(Number(10))

def plus(num): return lambda x: x.plus(num)
def minus(num): return lambda x: x.minus(num)
def times(num): return lambda x: x.times(num)
def divided_by(num): return lambda x: x.divided_by(num)
```

You can use this code like this:

```
print(seven(times(five()))) # returns 35
print(four(plus(nine()))) # returns 13
print(eight(minus(three()))) # returns 5
print(six(divided_by(two()))) # returns 3
```