

Prime Factors

CONTENTS

목차

Chapter1

TDD 훈련 가이드

Chapter2

Prime Factors 소개

Chapter3

Prime Factors 프로젝트 준비

Chapter3

Prime Factors TDD 개발

Chapter1

TDD 훈련 가이드

TDD 훈련 가이드 1

- **한번에 전부 구현하지 않는다.**

작고 점진적인 발전을 추구한다.

테스트 통과를 위한 간단한 코드를 구현한다.

- **입력과 출력이 명확한 작은 테스트 부터 개발한다.**

작은 개발로 시작하여, 점차 확장해 가며 개발을 완성해간다.

- **테스트 코드도 리팩토링을 하는 것이 좋다.**

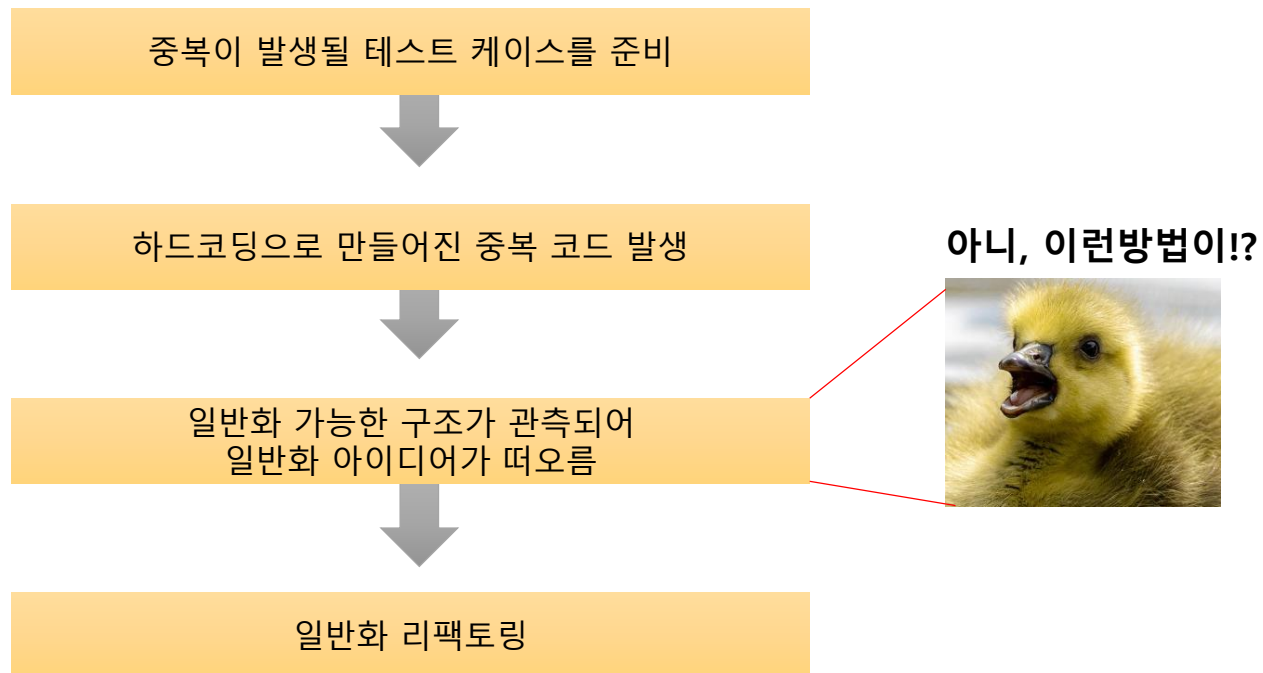
- Unit Test 코드는 관리되어야 하기 때문이다.

켄트백의 삼각측량 (Triangulation)

두 개 이상의 테스트 케이스, 그리고 중복 코드를 발생시킨다.

그 중복 속에서 일반화 가능한 구조가 관측된다.

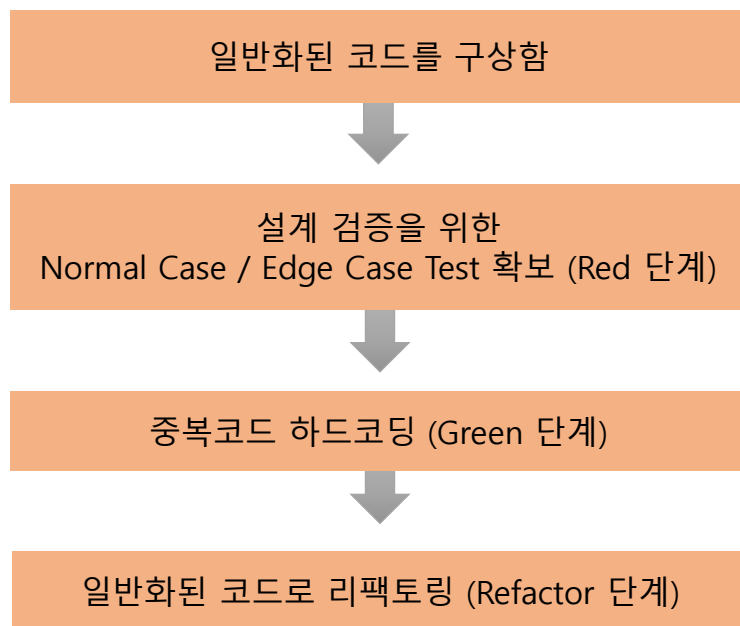
중복 코드를 근거로 코드를 점진적으로 안전하게 일반화하는 방식이다.



TDD의 훈련단계에서 지켜야 할 켄트백의 **baby Step**

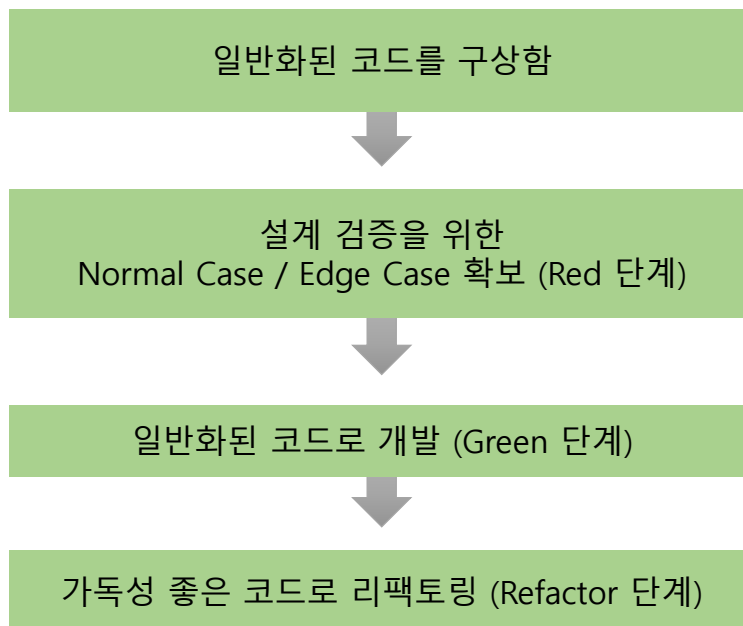
1. 아주 작은 테스트를 작성한다.
2. 통과를 위해 최소한의 코드를 작성한다.
3. 중복을 제거한다.

작게 시작해야, 자신만의 보폭을 코드 복잡성에 따라 조율할 수 있다.



고급 개발자의 TDD 실무 스타일

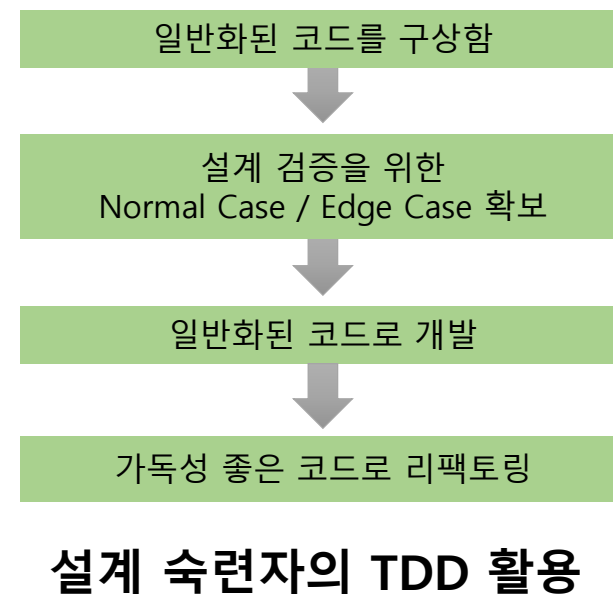
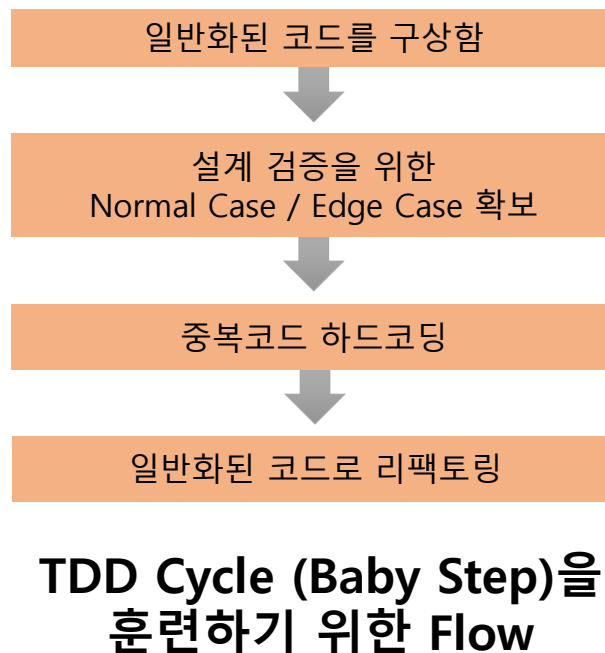
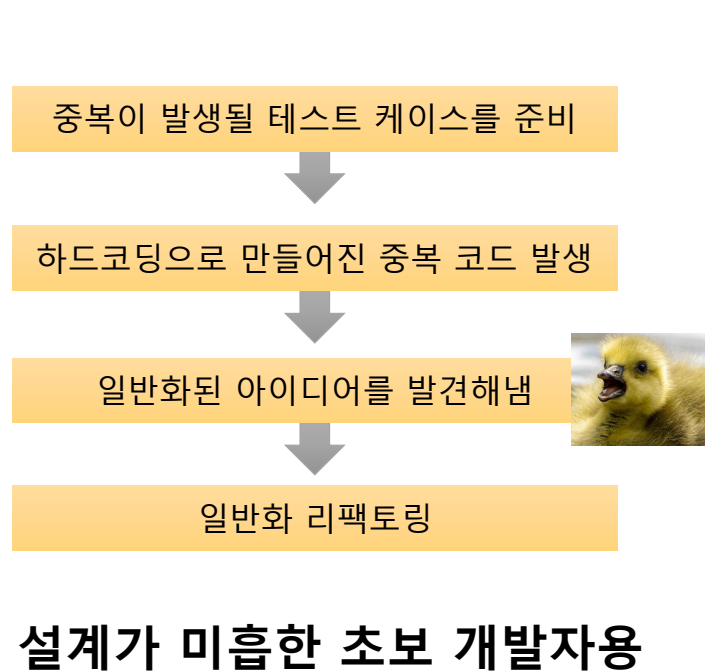
Red 단계에서는 설계가 명확하고,
테스트는 설계에 대한 검증 수단과, 리팩토링을 위한 준비로 쓰인다.
중복없는 코드를 바로 작성하면서, 가독성 있는 코드로 리팩토링 한다.



TDD 훈련 가이드 5

실습과정

- 1 단계 : 켄트백의 피보나치수열 KATA
- 2 단계 : Baby Step 훈련용 <-- Prime Factors KATA
- 3 단계 : 실무 스타일



Chapter2

Prime Factors 소개

Prime Factors 소개

소인수 분해

양의 정수를 소수들의 곱으로 표현

예시

수를 넣으면, 소인수 분해 된 배열을 반환한다.

2->[2]

3->[3]

4->[2,2]

6->[2,3]

8->[2,2,2]

9->[3,3]

12->[2,2,3]

14->[2,7]

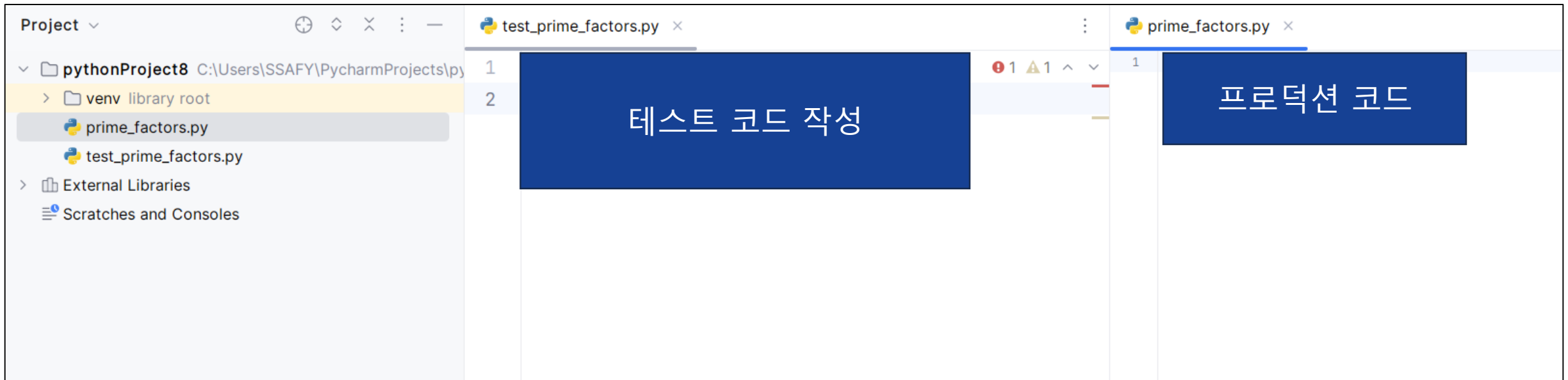
Chapter3

Prime Factors 프로젝트 준비

프로젝트, Repository 준비 + 실습을 위한 간단한 Git 복습

Unittest 기본 세팅

- test_prime_factors.py / prime_factors.py파일 생성
오타 조심!
창 배치



TestCase 추가

첫 번째 테스트케이스 추가

```
test_prime_factors.py x
1 ▶ def test_prime_factor_of_1():
2   ⚡ pass
```

기본 코드 작성

- unittest가 잘 동작됨을 확인한다.

```
test_prime_factors.py x
```

```
1 def test_prime_factor_of_1():
2     prime_factor = PrimeFactor()
3     assert 1 == 1
```

```
prime_factors.py x
```

```
1 class PrimeFactor:
2     pass
```

Test Results

- ✓ test_prime_factors
 - ✓ PrimeFactorTest
 - ✓ test_prime_factor_of_1

Git에 Push 하기

- 여기까지 작업 후, 첫 Commit 후, Push 진행

- ## Github Repository 생성하기

- Repo. name : prime-factors-번호
- Description : 이름 입력 (알아볼 수 있도록)
- private** 선택
- README.md 파일 **생성 안함**

- ## 생성 후 설정

- Setting > Collaborator : 강사 / 팀원 추가하기

github.com/new

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

Create a new repository

A repository contains all project files, including the revision history. Already have a project? [Import a repository.](#)

Owner * / Repository name *

Reviewer-XX / **PrimeFactors-12345**

Great repository names are short and memorable. Need inspiration? How about [ficti](#)

Description (optional)

이름

☐ Public
Any logged in user can see this repository. You choose who can commit.

☒ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

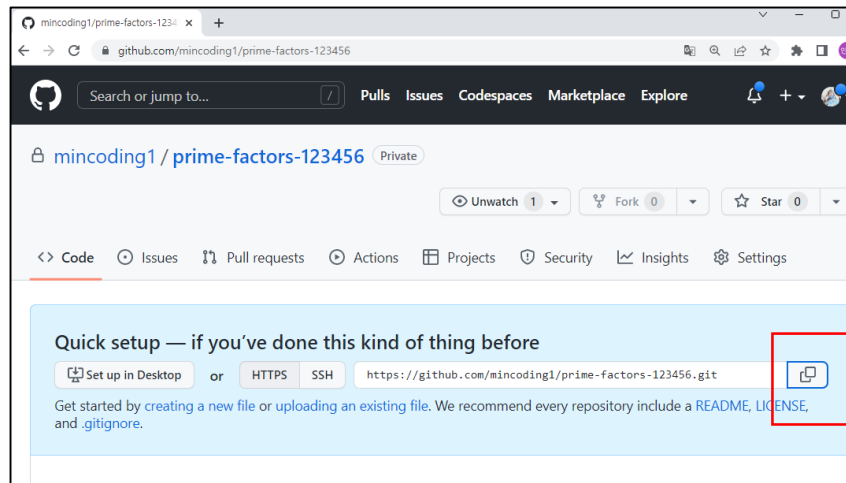
Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None

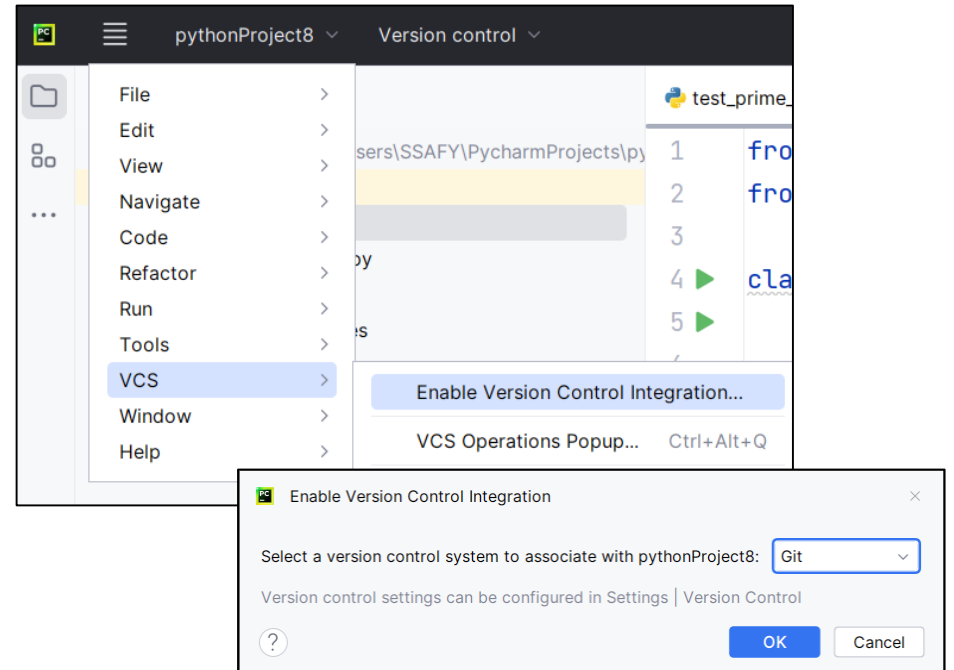
Choose a license

현재까지 내용 Push를 위한 준비

- Git GUI 도구를 이용하여 Push 진행



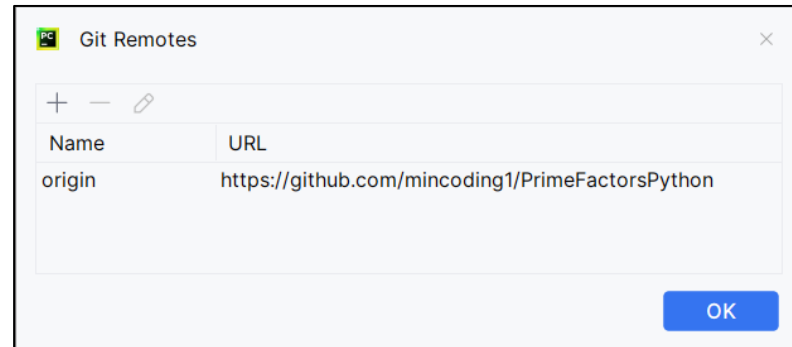
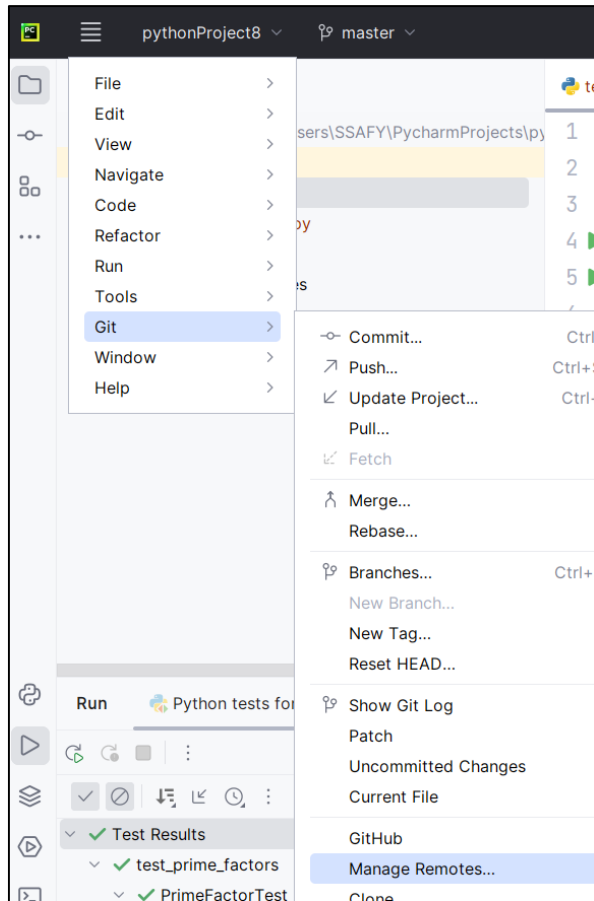
Github에서
Repo 생성 / **Git** 주소 복사하기



버전관리시스템으로 Git을 사용하겠다고
지정해주기

원격 Repository 지정

- Origin 으로 등록해두기

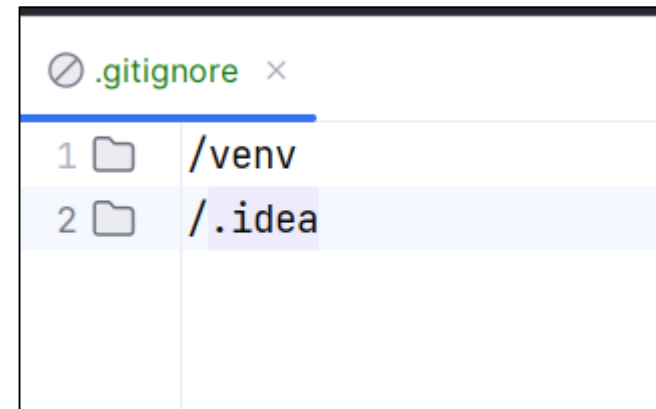
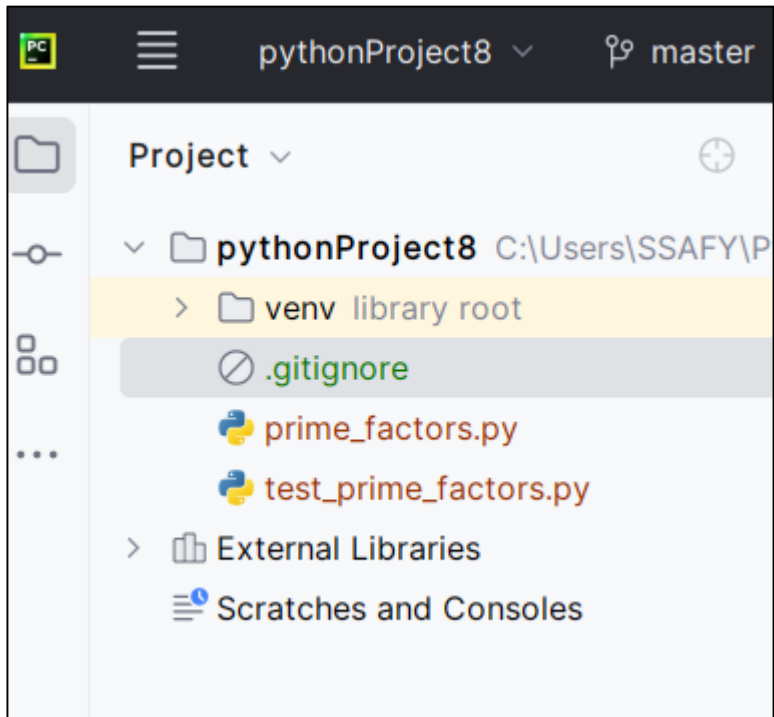


+ 버튼 클릭 후, 복사한 Git 주소 기입
Remote 이름은 "origin" 으로 하자.

OK 를 눌러 창을 닫음

.gitignore 파일 만들기

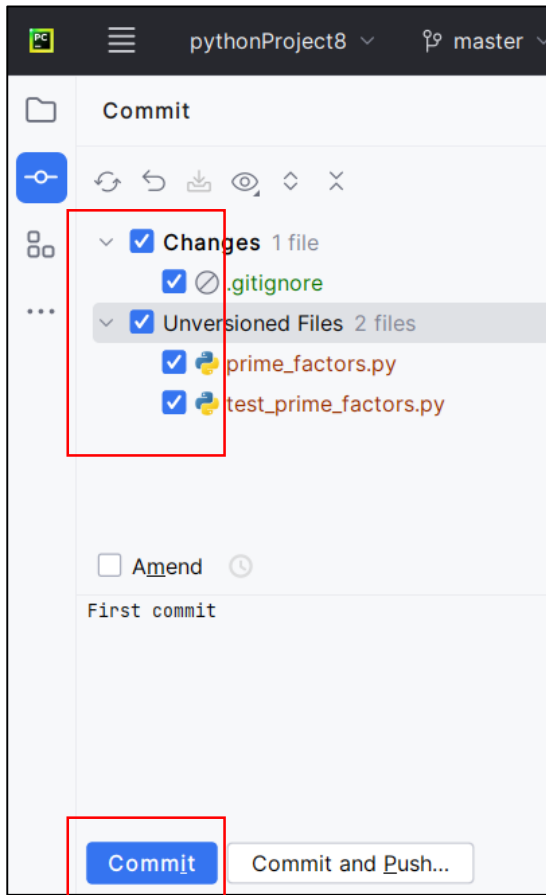
- 필요한 파일만 push 할 수 있도록 지정



venv, .idea 폴더 내용은 모두 Commit 하지 않는다.

Commit 하기

- Local Repo.로 Commit 하기

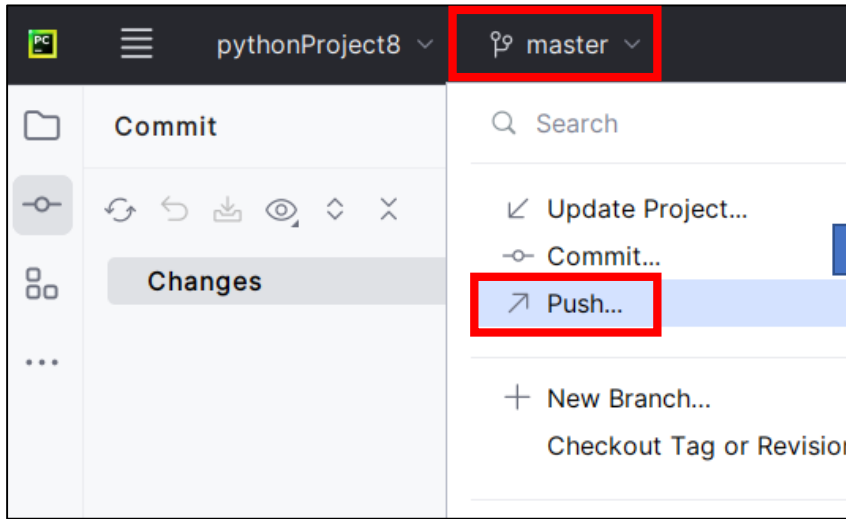


하단, Git 탭을 누르고
[Log] 탭을 한번 더 누르면
Commit 이력을 확인할 수 있음

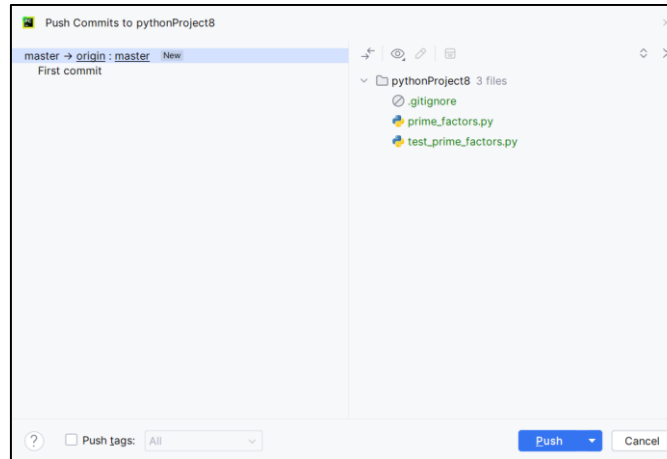
변경된 파일 모두 선택하고
"First commit" 이라는 Commit 메시지 입력 후 Commit 클릭

Push 하기

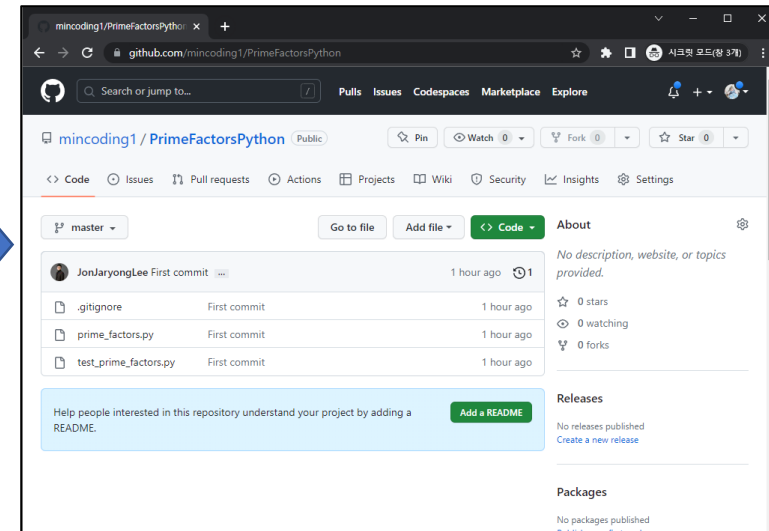
- Push 하여 결과 확인하기



Push 선택



Push 누르기



Github 에서 결과확인하기

실습 과정 구성

- 본 실습 과정은 총 7개의 TDD Cycle Step 이 있다.
- 각 Step 은 다음과 같이 나누어져 있다.

1. Red
2. Green
3. Refactor

프로젝트 제출 방법

- **Commit 타이밍**

1. Red 단계에서는 Commit 하지 않는다.
2. Green 단계 후 Unit Test Pass 시 **Commit** 을 한다.
3. Refactor 단계 마다 Unit Test 를 수행하며, Pass 시 **Commit**을 한다.

- **Commit 메시지 헤더 Format**

- Red 단계 : Commit 하지 않는다. (동작되는 코드만 Commit 해야 하기 때문)
- Green 단계 : **[feature] page 번호**
- Refactor 단계 : **[refactoring] page 번호**

- **Push 하여 최종 제출하기**

7 단계 까지 완료 이후, Push 1회만 진행

Chapter4

Prime Factors TDD 개발

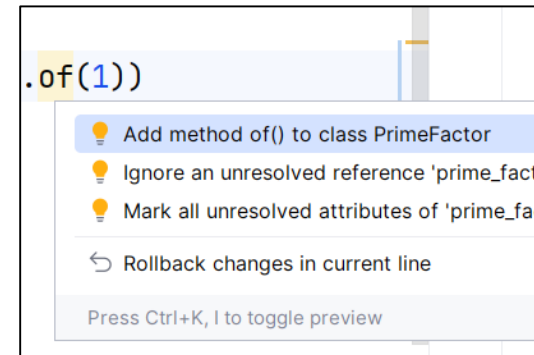
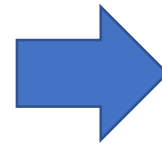
assertEqual문을 완성

그리고 of 함수에 Alt + Enter를 눌러 메서드 자동생성

```
def test_prime_factor_of_1():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(1) == []
```

Test

of(1)의 소인수 분해 결과는
[] 빈 배열이 나오는 것을 기대함

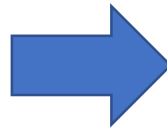


of 라는 메서드를 자동생성하기 위해,
of 에 커서를 두고, Alt + Enter 누르기

- List 리턴하는 형태로 바꿈
빌드는 되지만, Unit Test에서 Fail이 나도록 함

Production

```
class PrimeFactor:  
    def of(self, param):  
        pass
```



```
class PrimeFactor:  
    def of(self, param) -> []:  
        return None
```

- 기대값 : Fail 발생 (앞으로는 Run 단축키를 눌러 테스트를 진행할 것)
- Red를 눈으로 확인했으니, Green 단계 진행

The image shows a code editor with two panes: 'Test' and 'Production'.

Test Pane (test_prime_factors.py):

```
def test_prime_factor_of_1():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(1) == []
```

Production Pane (prime_factors.py):

```
1 class PrimeFactor:  
2     def of(self, param) -> []:  
3         return None  
4  
5  
6
```

Test Results Window:

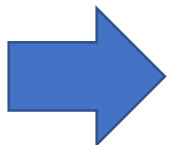
Test Results	5 m
test_prime_factors	5 m
PrimeFactorTest	5 m
test_prime_factor_of_1	5 m

빈 배열을 기대했지만,
null 값이 리턴되어 Fail 발생

- Unit Test에 통과될 수준만큼만 작성

```
class PrimeFactor:
    def of(self, param) -> []:
        return None
```

Production



```
class PrimeFactor:
    def of(self, param) -> []:
        factors = []
        return factors
```

Production

✓ Test Results
✓ test_prime_factors
✓ PrimeFactorTest
✓ test_prime_factor_of_1

어떤 수를 넣던,
무조건 [] 빈 배열이 나오는 것으로 변경

따로 Clean 코드로 만들 내용이 없으므로, 생략함.

이렇게 TDD한 Cycle을 완료함

(코드 수정한 내용이 없으므로, Commit 안함)

prime_factors.py ×

Production

```
1 class PrimeFactor:
2     def of(self, param) -> []:
3         factors = []
4         return factors
5
```

Clean 코드로 변경할 내용이 없음

- Red – '2'를 소인수분해 하는 테스트케이스 작성

```
def test_prime_factor_of_1():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(1) == []  
  
def test_prime_factor_of_2():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(2) == [2]
```

testPrimefactorOf1 코드를 복사 붙여넣기 하여 수정 진행
코드 입력 후 Red를 눈으로 확인

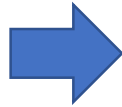
Test Results	4 ms	[] != [2]
test_prime_factors	4 ms	Expected :[2]
PrimeFactorTest	4 ms	Actual :[]
test_prime_factor_of_1	1 ms	<Click to see difference>
test_prime_factor_of_2	3 ms	Traceback (most recent call l

결과가 [2] 가 나와야 하는데,
실제 값은 빈 배열이 나왔으므로 Fail

- Pass가 될 정도로만 구현하기
이름 바꾸기 단축키 : Shift + F6
꼭 암기할 것. (더 이상 교재에 이름 바꾸기 단축키 안내 안함)

```
class PrimeFactor:
    def of(self, param) -> []:
        factors = []
        if param == 2:
            factors.append(2)
        return factors
```

Production



```
class PrimeFactor:
    def of(self, param) -> []:
        factors = []
        if param == 2:
            factors.append(2)
        return factors
```

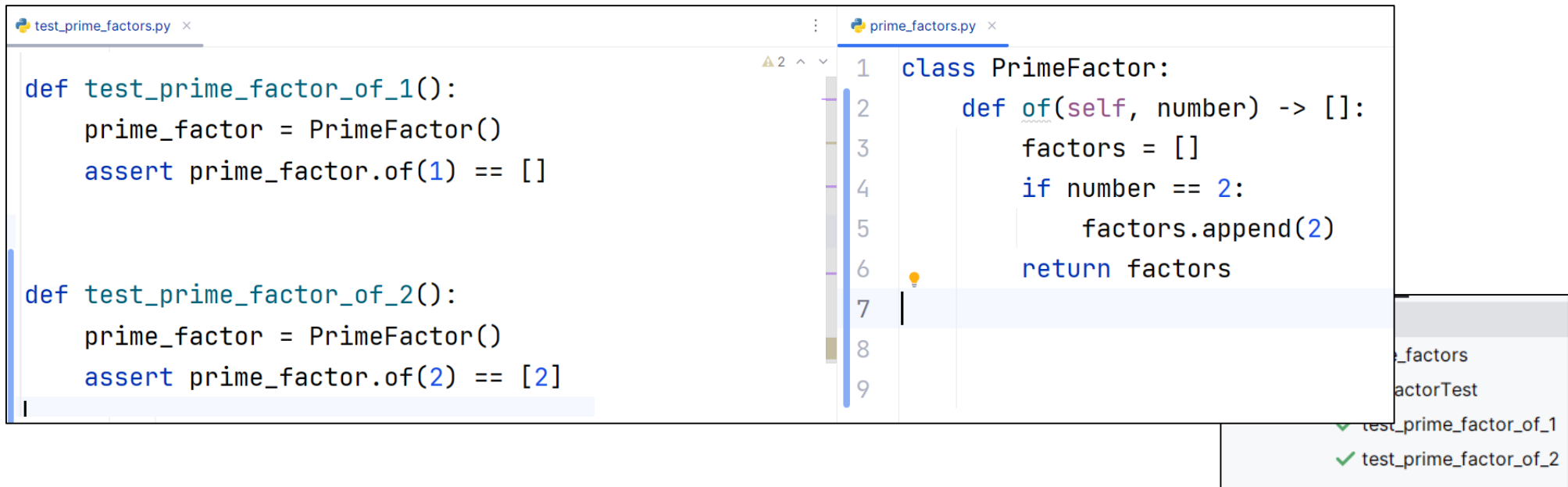


```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number == 2:
            factors.append(2)
        return factors
```

변수명 바꾸기 (이름바꾸기)
단축키 : Shift + F6 누르고
원하는 이름 기입하기

변수명을 number로 변경

Green을 보았고 Refactoring 차례지만 따로 할 내용이 아직 없으므로 다음 사이클로 넘어간다.



The screenshot shows a code editor with two files open: `test_prime_factors.py` and `prime_factors.py`. The left pane shows the test file with two test functions, `test_prime_factor_of_1()` and `test_prime_factor_of_2()`, both using `PrimeFactor()` and `assert` to verify the `of` method. The right pane shows the `PrimeFactor` class with an `of` method that returns a list of factors. A light blue selection bar highlights line 7 in the `prime_factors.py` file. A small dropdown menu is visible on the right side of the editor, showing a list of files including `_factors`, `factorTest`, `test_prime_factor_of_1`, and `test_prime_factor_of_2`. The status bar at the bottom right shows two green checkmarks, indicating that all tests passed.

```
test_prime_factors.py x
def test_prime_factor_of_1():
    prime_factor = PrimeFactor()
    assert prime_factor.of(1) == []

def test_prime_factor_of_2():
    prime_factor = PrimeFactor()
    assert prime_factor.of(2) == [2]

prime_factors.py x
1 class PrimeFactor:
2     def of(self, number) -> []:
3         factors = []
4         if number == 2:
5             factors.append(2)
6         return factors
7
8
9
```

- _factors
- factorTest
- test_prime_factor_of_1
- test_prime_factor_of_2

- Red – '3'을 소인수분해 하는 테스트케이스 작성

Test

```
def test_prime_factor_of_1():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(1) == []  
  
def test_prime_factor_of_2():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(2) == [2]  
  
def test_prime_factor_of_3():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(3) == [3]
```

소인수 3을 넣었을 때 [3] 이 나와야 함.

- Fail을 눈으로 확인하였으니, 이제 Green 단계를 진행함

✓ × Test Results	3 ms	[] != [3]
✓ × test_prime_factors	3 ms	Expected :[3]
✓ × PrimeFactorTest	3 ms	Actual :[]
✓ test_prime_factor_of_1	0 ms	<Click to see diff
✓ test_prime_factor_of_2	0 ms	Traceback (most recent call last):
× test_prime_factor_of_3	3 ms	File "C:\Users\...\test_prime_factor.py", line 10, in test_prime_factor_of_3

Fail을 눈으로 확인함

- Green – 테스트케이스 성공

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number == 2:
            factors.append(2)
            elif number == 3:
                factors.append(3)
        return factors
```

코드를 추가하고,
Green을 눈으로 확인

✓ Test Results

✓ test_prime_factors

✓ PrimeFactorTest

✓ test_prime_factor_of_1

✓ test_prime_factor_of_2

✓ test_prime_factor_of_3

- Production Code를 Refactoring 함

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number == 2:
            factors.append(2)
        elif number == 3:
            factors.append(3)
        return factors
```

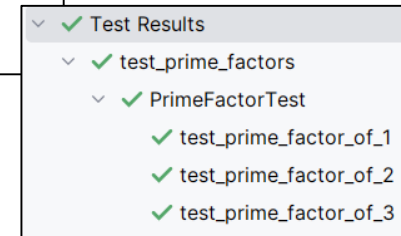
Production

하드코딩 일부를 number 변수로 변경



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number == 2:
            factors.append(number)
        elif number == 3:
            factors.append(number)
        return factors
```

리팩토링 후 반드시 Test로 결과 확인

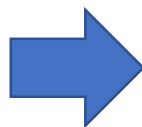


- Refactoring – generalize

조금 더 범용적인 코드로 리팩토링함
여기까지 TDD한 Cycle 완료

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number == 2:
            factors.append(number)
        elif number == 3:
            factors.append(number)
        return factors
```



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            factors.append(number)
        return factors
```

리팩토링 후 반드시 Test로 결과 확인

Test Results	
test_prime_factors	
PrimeFactorTest	
test_prime_factor_of_1	✓
test_prime_factor_of_2	✓
test_prime_factor_of_3	✓

- Red – '4'를 소인수분해 하는 테스트케이스 작성

Test

```
def test_prime_factor_of_4():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(4) == [2, 2]
```

4를 소인수 분해하면, [2, 2] 가 나와야하는
테스트 코드 추가하기

Test Results
test_prime_factors
PrimeFactorTest
test_prime_factor_of_1
test_prime_factor_of_2
test_prime_factor_of_3
test_prime_factor_of_4

4일 때, 소인수 분해가 되도록 코드 추가

GREEN

- Green – 테스트케이스 성공

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            factors.append(number)
        return factors
```

기존 소스코드

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                factors.append(2)
                factors.append(2)
            else:
                factors.append(number)
        return factors
```

코드 추가하기

4인 경우, [2, 2]가 되고,
그 외는 기존처럼 [number]가 되도록 함

```
✓ Test Results
  ✓ test_prime_factors
    ✓ PrimeFactorTest
      ✓ test_prime_factor_of_1
      ✓ test_prime_factor_of_2
      ✓ test_prime_factor_of_3
      ✓ test_prime_factor_of_4
```

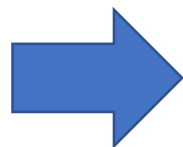
Green을 눈으로 확인

- Refactoring – generalize

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                factors.append(2)
                factors.append(2)
            else:
                factors.append(number)
        return factors
```

Production

기존 소스코드



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                if number % 2 == 0:
                    factors.append(2)
                    number //= 2
                if number % 2 == 0:
                    factors.append(2)
                    number //= 2
            else:
                factors.append(number)
        return factors
```

조금 더 범용적인 코드가 되도록
준비작업

✓ Test Results	
✓ test_prime_factors	
✓ PrimeFactorTest	
✓ test_prime_factor_of_1	
✓ test_prime_factor_of_2	
✓ test_prime_factor_of_3	
✓ test_prime_factor_of_4	

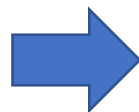
- Refactoring - if 반복구문 리팩토링
여기까지 하나의 TDD Cycle을 마무리

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                if number % 2 == 0:
                    factors.append(2)
                    number //= 2
                if number % 2 == 0:
                    factors.append(2)
                    number //= 2
            else:
                factors.append(number)

        return factors
```

기존 소스코드



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
            else:
                factors.append(number)

        return factors
```

조금 더 범용적이도록
소스코드 리팩토링

Test Results	
✓	test_prime_factors
✓	PrimeFactorTest
✓	test_prime_factor_of_1
✓	test_prime_factor_of_2
✓	test_prime_factor_of_3
✓	test_prime_factor_of_4

- Red – '6'을 소인수분해 하는 테스트케이스 작성

Test

```
def test_prime_factor_of_6():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(6) == [2, 3]
```

6에 대한 소인수 분해 값인 [2, 3] 기대값 테스트

Test Results
test_prime_factors
PrimeFactorTest
test_prime_factor_of_1
test_prime_factor_of_2
test_prime_factor_of_3
test_prime_factor_of_4
test_prime_factor_of_6

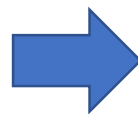
- Green – 테스트케이스 성공

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
            else:
                factors.append(number)

        return factors
```

Production

기존 소스코드



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
            elif number == 6:
                factors.append(2)
                factors.append(3)
            else:
                factors.append(number)

        return factors
```

6일때 [2, 3]이 나오도록 코드 추가

```
✓ Test Results
  ✓ test_prime_factors
    ✓ PrimeFactorTest
      ✓ test_prime_factor_of_1
      ✓ test_prime_factor_of_2
      ✓ test_prime_factor_of_3
      ✓ test_prime_factor_of_4
      ✓ test_prime_factor_of_6
```

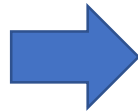
- Refactoring – generalize

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
            elif number == 6:
                factors.append(2)
                factors.append(3)
            else:
                factors.append(number)

        return factors
```

Production

기존 소스코드



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            if number == 4:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
            elif number == 6:
                while number % 2 == 0:
                    factors.append(2)
                    number //= 2
                while number % 3 == 0:
                    factors.append(3)
                    number //= 3
            else:
                factors.append(number)

        return factors
```

```
✓ Test Results
  ✓ test_prime_factors
    ✓ PrimeFactorTest
      ✓ test_prime_factor_of_1
      ✓ test_prime_factor_of_2
      ✓ test_prime_factor_of_3
      ✓ test_prime_factor_of_4
      ✓ test_prime_factor_of_6
```

조금 더 범용적인 코드가 되도록 준비작업

리팩토링 – Extract local variable (변수 추출하기)

Refactor

- Refactoring – extract local variable

Refactor 메뉴 단축키 : **Ctrl + Alt + Shift + T** (암기 할 것!)

prime_factors.py x

Production

```
1 class PrimeFactor:
2     def of(self, number) -> []:
3         factors = []
4         if number > 1:
5             if number == 4:
6                 while number % 2 == 0 :
7                     factors.append(2)
8                     number //= 2
9             elif number == 6:
10                while number % 2 == 0 :
11                    factors.append(2)
12                    number //= 2
```

Refactor This Selection

- 1 Change Signature...
- 2 Introduce Variable...
- 3 Introduce Constant...
- 4 Introduce Field...

Multiple occurrences found
Replace this occurrence only
Replace all 6 occurrences

6개 모두 변경하기

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4:
                while number % divisor == 0 :
                    factors.append(divisor)
                    number //= divisor
            elif number == 6:
                while number % divisor == 0 :
                    factors.append(divisor)
                    number //= divisor
            while number % 3 == 0:
                factors.append(3)
                number //= 3
        else:
            factors.append(number)

        return factors
```

Test Results

- ✓ test_prime_factors
- ✓ PrimeFactorTest
 - ✓ test_prime_factor_of_1
 - ✓ test_prime_factor_of_2
 - ✓ test_prime_factor_of_3
 - ✓ test_prime_factor_of_4
 - ✓ test_prime_factor_of_6

숫자 2 대신, 변수 값으로 이름을 붙여줌

변수 추출하기 / 변수 도입하기
(extract local variable / Introduce Variable)

객체 이름을 divisor 로 변경 후
리팩토링 잘 되었는지 테스트 진행

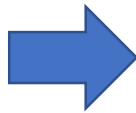
- 숫자 3 대신 divisor 값으로 변경

Production

```
elif number == 6:
    while number % divisor == 0:
        factors.append(divisor)
        number //= divisor
    while number % 3 == 0:
        factors.append(3)
        number //= 3
else:
    factors.append(number)

return factors
```

기존 소스코드



```
elif number == 6:
    while number % divisor == 0:
        factors.append(divisor)
        number //= divisor
    divisor += 1
    while number % divisor == 0:
        factors.append(divisor)
        number //= divisor
    divisor += 1
else:
    factors.append(number)

return factors
```

divisor + 1 로 해당 코드 수정
그리고 테스트

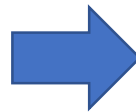
Test Results	
✓	test_prime_factors
✓	PrimeFactorTest
✓	test_prime_factor_of_1
✓	test_prime_factor_of_2
✓	test_prime_factor_of_3
✓	test_prime_factor_of_4
✓	test_prime_factor_of_6

- Refactoring - 비슷한 중복을 완전한 중복으로 변환
이렇게 하나의 TDD Cycle 완료

Production

```
elif number == 6:  
    while number % divisor == 0 :  
        factors.append(divisor)  
        number //= divisor  
        divisor += 1  
    while number % divisor == 0:  
        factors.append(divisor)  
        number //= divisor  
        divisor += 1  
else:  
    factors.append(number)  
  
return factors
```

기존 소스코드



```
elif number == 6:  
    divisor = 2  
    while number > 1 :  
        while number % divisor == 0 :  
            factors.append(divisor)  
            number //= divisor  
            divisor += 1  
    else:  
        factors.append(number)  
  
return factors
```

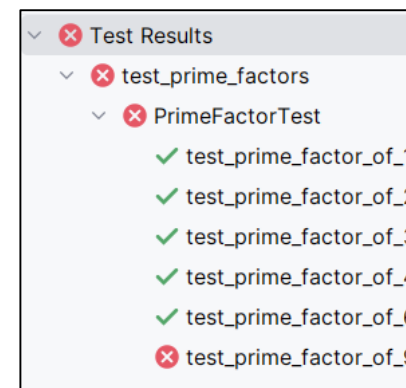
divisor / number 순서 유의하여
코드 작성하기

Test Results	
✓	test_prime_factors
✓	PrimeFactorTest
✓	test_prime_factor_of_1
✓	test_prime_factor_of_2
✓	test_prime_factor_of_3
✓	test_prime_factor_of_4
✓	test_prime_factor_of_6

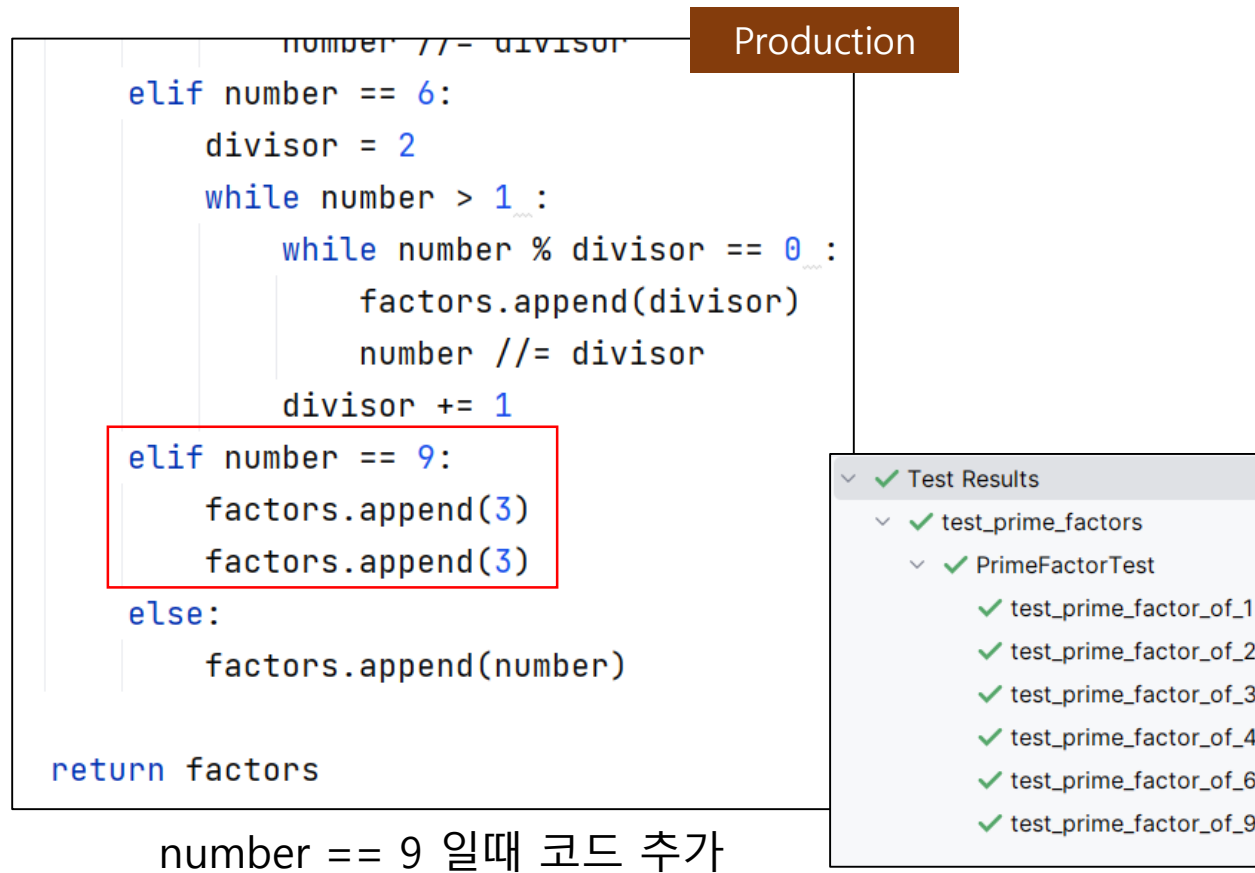
- Red – '9'를 소인수분해 하는 테스트케이스 작성

```
def test_prime_factor_of_9():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(9) == [3, 3]
```

새로운 테스트코드 추가
소인수분해 9일때 정답은 [3, 3]이어야함



- Green – 테스트케이스 성공



- Refactoring - 중복제거

```
number //= divisor
elif number == 6:
    divisor = 2
    while number > 1:
        while number % divisor == 0:
            factors.append(divisor)
            number //= divisor
        divisor += 1
elif number == 9:
    factors.append(3)
    factors.append(3)
else:
    factors.append(number)

return factors
```

Production

기존코드 (사각 영역, 소스코드 제거)



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4:
                while number % divisor == 0:
                    factors.append(divisor)
                    number //= divisor
            elif number == 6 or number == 9:
                divisor = 2
                while number > 1:
                    while number % divisor == 0:
                        factors.append(divisor)
                        number //= divisor
                    divisor += 1
            else:
                factors.append(number)

        return factors
```

코드 추가하기

Test Results	
test_prime_factors	
PrimeFactorTest	
test_prime_factor_of_1	✓
test_prime_factor_of_2	✓
test_prime_factor_of_3	✓
test_prime_factor_of_4	✓
test_prime_factor_of_6	✓
test_prime_factor_of_9	✓

- Refactoring - 중복제거

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4:
                while number % divisor == 0:
                    factors.append(divisor)
                    number //= divisor
            elif number == 6 or number == 9:
                divisor = 2
                while number > 1:
                    while number % divisor == 0:
                        factors.append(divisor)
                        number //= divisor
                    divisor += 1
            else:
                factors.append(number)

        return factors
```

기존코드 (사각 영역, 소스코드 제거)



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4 or number == 6 or number == 9:
                divisor = 2
                while number > 1:
                    while number % divisor == 0:
                        factors.append(divisor)
                        number //= divisor
                    divisor += 1
            else:
                factors.append(number)

        return factors
```

else if를 if로 변경하고,
number == 4 || 내용 추가.

Test Results	
✓	test_prime_factors
✓	PrimeFactorTest
✓	test_prime_factor_of_1
✓	test_prime_factor_of_2
✓	test_prime_factor_of_3
✓	test_prime_factor_of_4
✓	test_prime_factor_of_6
✓	test_prime_factor_of_9

- Red – '12'를 소인수분해 하는 테스트케이스 작성

```
def test_prime_factor_of_12():  
    prime_factor = PrimeFactor()  
    assert prime_factor.of(12) == [2, 2, 3]
```

12 소인수분해시
[2, 2, 3]이 나와야 한다는 테스트코드 추가.

Test Results

- test_prime_factors
 - PrimeFactorTest
 - test_prime_factor_of_1 ✓
 - test_prime_factor_of_12 ✗
 - test_prime_factor_of_2 ✓
 - test_prime_factor_of_3 ✓
 - test_prime_factor_of_4 ✓
 - test_prime_factor_of_6 ✓
 - test_prime_factor_of_9 ✓

- Green – 테스트케이스 성공

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4 or number == 6 or number == 9 or number == 12:
                divisor = 2
                while number > 1:
                    while number % divisor == 0:
                        factors.append(divisor)
                        number //= divisor
                    divisor += 1
            else:
                factors.append(number)

        return factors
```

Production

✓ Test Results

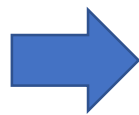
- ✓ test_prime_factors
 - ✓ PrimeFactorTest
 - ✓ test_prime_factor_of_1
 - ✓ test_prime_factor_of_12
 - ✓ test_prime_factor_of_2
 - ✓ test_prime_factor_of_3
 - ✓ test_prime_factor_of_4
 - ✓ test_prime_factor_of_6
 - ✓ test_prime_factor_of_9

number == 12 일때 코드 추가 후 Green 확인하기

- Refactoring – 중복코드 제거, generalize

Production

```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        if number > 1:
            divisor = 2
            if number == 4 or number == 6 or number == 9 or number == 12:
                divisor = 2
            while number > 1:
                while number % divisor == 0:
                    factors.append(divisor)
                    number //= divisor
                divisor += 1
            else:
                factors.append(number)
        return factors
```



```
class PrimeFactor:
    def of(self, number) -> []:
        factors = []
        divisor = 2
        while number > 1:
            while number % divisor == 0:
                factors.append(divisor)
                number //= divisor
            divisor += 1
        return factors
```

최종테스트

✓ Test Results	
✓ test_prime_factors	
✓ PrimeFactorTest	
✓ test_prime_factor_of_1	
✓ test_prime_factor_of_12	
✓ test_prime_factor_of_2	
✓ test_prime_factor_of_3	
✓ test_prime_factor_of_4	
✓ test_prime_factor_of_6	
✓ test_prime_factor_of_9	

기존코드 (사각 영역, 소스코드 제거)

감사합니다.