2021-08-12 - SW 테스트케이스 설계

코딩은 결정, 피드백의 연속

- 이 함수는 이런 기능을 하도록 짜야겠군 - 결정
- 그런데 내가 이 함수는 잘 만든 걸까 - 피드백
- 그럼 함수를 완성하고 여기에 피드백을 받을 테스트 케이스를 만들어야겠다

코드 먼저 ⇒ 테스트 나중

- 현재 시간이 AM, PM인지 판단하여 리턴하는 함수를 만드는 경우

```
public string GetAMorPM(){

  var now = DateTime.now;
  if(now.Hour < 12){
    return "AM"
  }
  else {
    return "PM"
  }
}
```

- 필요한 테스트 케이스 수는 2개
- DateTime.now를 직접 사용했기 때문에 ⇒ 테스트가 어려워져 버렸음
  - 코드의 수정이 불가피!

테스트를 어렵게 만드는 것은?

- 불확실성 : 임의의 값, 임의의 시간 등이 필요한 경우
  - 전역 변수, API 서버로부터 받는 값 등
- 부수 작업 :
  - DB에 기록을 하는 것
  - 메일 발송
  - 시스템 외부에 뭔가를 던지지만 리턴값이 없는....
- 켄트 백
  - 결정과 피드백 사이의 간격이 커져 버린다
    - 코드 작성 시에는 왜 몰랐을까?
    - 좁히려면, 너무 많은 비용이 든다.

테스트 케이스 먼저 → 코드 나중

- Case 1 : 오전 시간인 10:00 경우에는 → AM이 출력
- Case 2 : 오후 시간인 15:20 경우에는 → PM이 출력
- 여기에 맞춰 코드를 작성하면 Datetime 라이브러리는 자연스레 개입이 되지 않는다.

로그인 함수를 TDD로 만들어 보자

- 어떤 경우들이 존재할까?
- 결정 테이블

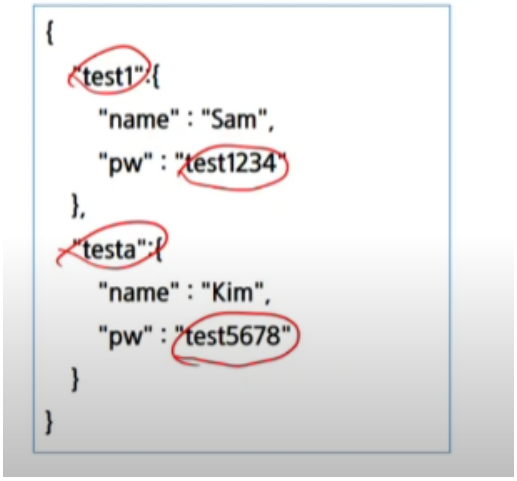
결정 테이블(Decision Table)

		1	2	3	4	5	6
조건	존재ID입력	Y	N	Y	N	N	Y
	ID미입력	N	Y	N	N	N	N
	일치PW입력	Y	N	N	N	N	N
	PW입력 안함	N	N	N	N	Y	Y
결과		정상적으로 로그인 됨	'아이디를 입력하세요' 라는 메시지 뜸	'패스워드가 일치하지 않습니다' 라는 메시지 뜸	'미등록 ID입니다' 라는 메시지가 뜸	'미등록 ID입니다' 라는 메시지가 뜸	'PW를 입력하세요' 라는 메시지가 뜸

User Data

- pytest라는 도구를 이용해서 보여주심

User.txt




```
import json

# Happy CASE를 제외한 오류를 정해둠

with open("User.txt", "r") as f:
  json_data = json.load(f)

# 일단은 무조건 fail이 떨어지는 케이스를 작성

def Login(ID, PW):
```

```
## test.py

import main
import pytest

def test_case0():
  # ID는 존재, PW는 일치
  assert main.Login("test1", "test1234") == True

  pass

def test_case1():
  # ID는 공란
  assert main.Login("", "test1234") == main.FIRST_CAES_RESULT

  pass

def test_case2():
  # ID는 존재, PW는 불일치
  assert main.Login("test1", "1234") == main.SECOND_CAES_RESULT

  pass

def test_case3():
  # ID는 미존재, PW는 공란
  assert main.Login("testtest", "") == main.THIRD_CAES_RESULT

  pass

def test_case4():
  # ID는 미존재, PW는 값 있음
  assert main.Login("testtest", "1234") == main.FOURTH_CAES_RESULT
```

```
  pass

def test_case5():
  # ID는 존재, PW는 공란
  assert main.Login("test1", "") == main.FIFTH_CAES_RESULT

  pass
```

- main에서 돌려보면 ⇒ 다 fail이 떨어지게 됨
- 하나하나 바꾸어주는 것이 성공의 과정

```
import json

# Happy CASE를 제외한 오류를 정해둠

FIRST_CASE_RESULT = 2 # ID가 공란
SECOND_CASE_RESULT = 3 # ID가 공란
THIRD_CASE_RESULT = 4 # ID가 공란
FOURTH_CASE_RESULT = 5 # ID가 공란
FIFTH_CASE_RESULT = 6 # ID가 공란

with open("User.txt", "r") as f:
  json_data = json.load(f)

# 일단은 무조건 fail이 떨어지는 케이스를 작성

def Login(ID, PW):
  # For True Case
  if(len(ID.strip()) > 0 and len(PW.strip())>0):
    try:
        if(json_data[ID]['pw'] == PW):
          return True;
    except(KeyError):
        pass
  # For First Case
  if(len(ID.strip()) == 0):
    return FIRST_CASE_RESULT
  # For Second Case
  if(len(ID.strip()) > 0 and len(PW.strip()) > 0):
    try:
        if(json_data[ID]['pw'] == PW):
          return SECOND_CASE_RESULT;
    except(KeyError):
        pass
  # For Third Case
  if(len(ID.strip()) > 0 and len(PW.strip()) == 0):
    try:
        if not (ID in json_data):
          return THIRD_CASE_RESULT;
    except(KeyError):
        pass
  # For Fourth Case
  if(len(ID.strip()) > 0 and len(PW.strip()) > 0):
```

```
    try:
        if not (ID in json_data):
          return FOURTH_CASE_RESULT;
    except(KeyError):
        pass
  # For FIFTH Case
  if(len(ID.strip()) > 0 and len(PW.strip()) == 0):
    try:
        if (ID in json_data):
          return FIFTH_CASE_RESULT;
    except(KeyError):
        pass
  return False
```

1. 오류 상황을 구현하고 코드를 채움
    a. 에러를 조기에 발견할 수 있는 부분
2. 계속 실행, 구현을 반복해서 해결한다.

TDD의 장,단점

- 장점
    - 디자인 패턴을 의식하지 않고 어느 정도 이에 따른 코드 결과물을 만들 수 있다.
    - 테스트 코드 고민이 줄어든다(결정과 피드백의 거리가 좁아짐)
- 단점
    - 테스트 케이스의 퀄리티에 따라 결과물이 달라진다
    - 발상의 전환에 시간이 걸린다
    - 잘 모르면 테스트에 구멍이 생기고 코드가 효율이 떨어짐

- 그러면 프론트에서는 어떻게 활용해야 하는가?
    - 이것을 TDD로 하기 어렵다는 부분이 존재
    - 프론트에서 테스트 결과물은 화면보다는 데이터로 승부를 해야 함
        - 데이터 위주로 하는 것이 나음!

- UI부분은 TDD가 약한 부분이 있음