

Pytest와 함께 코드 테스트

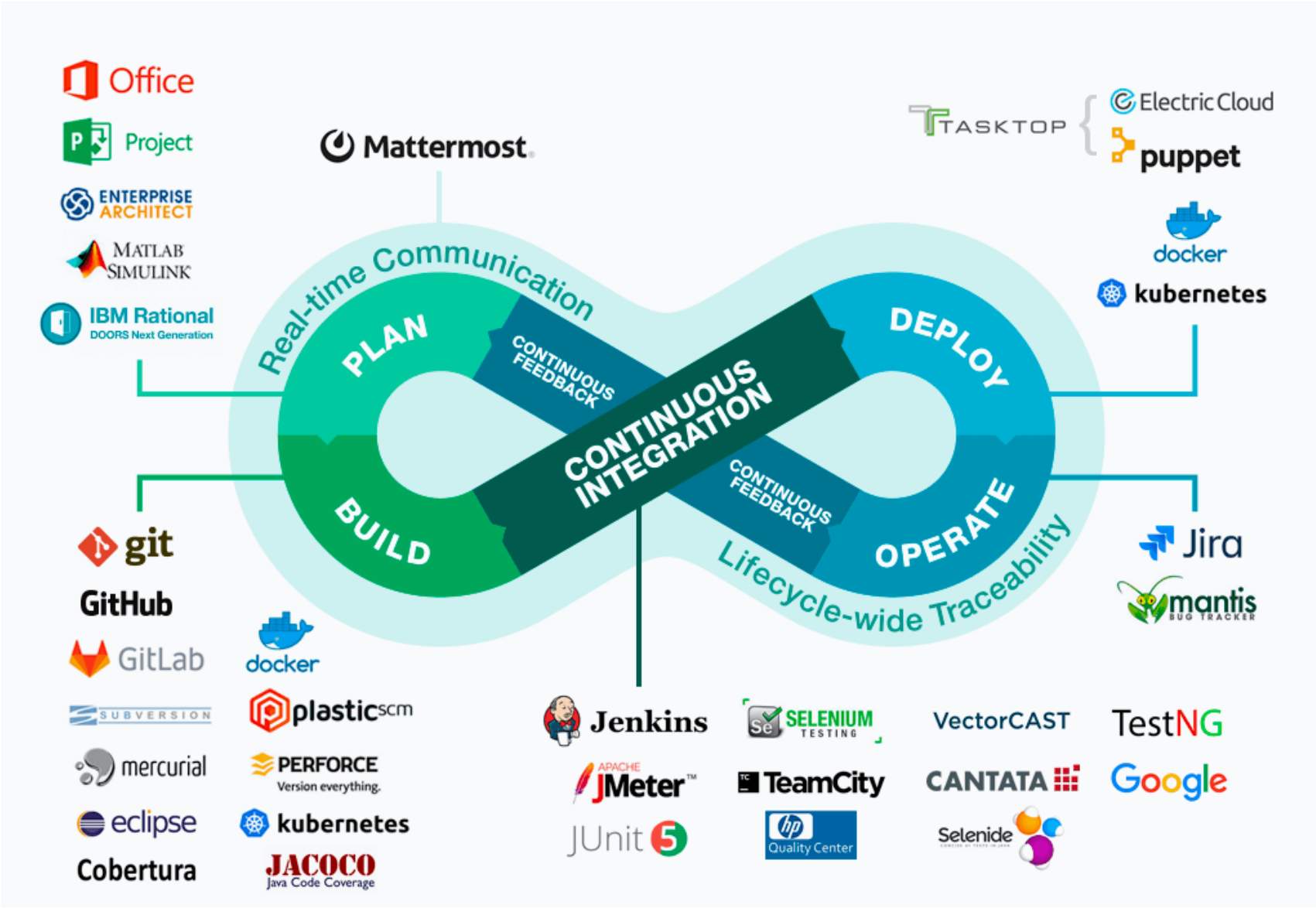
-실습-

컴퓨터공학부
천세진

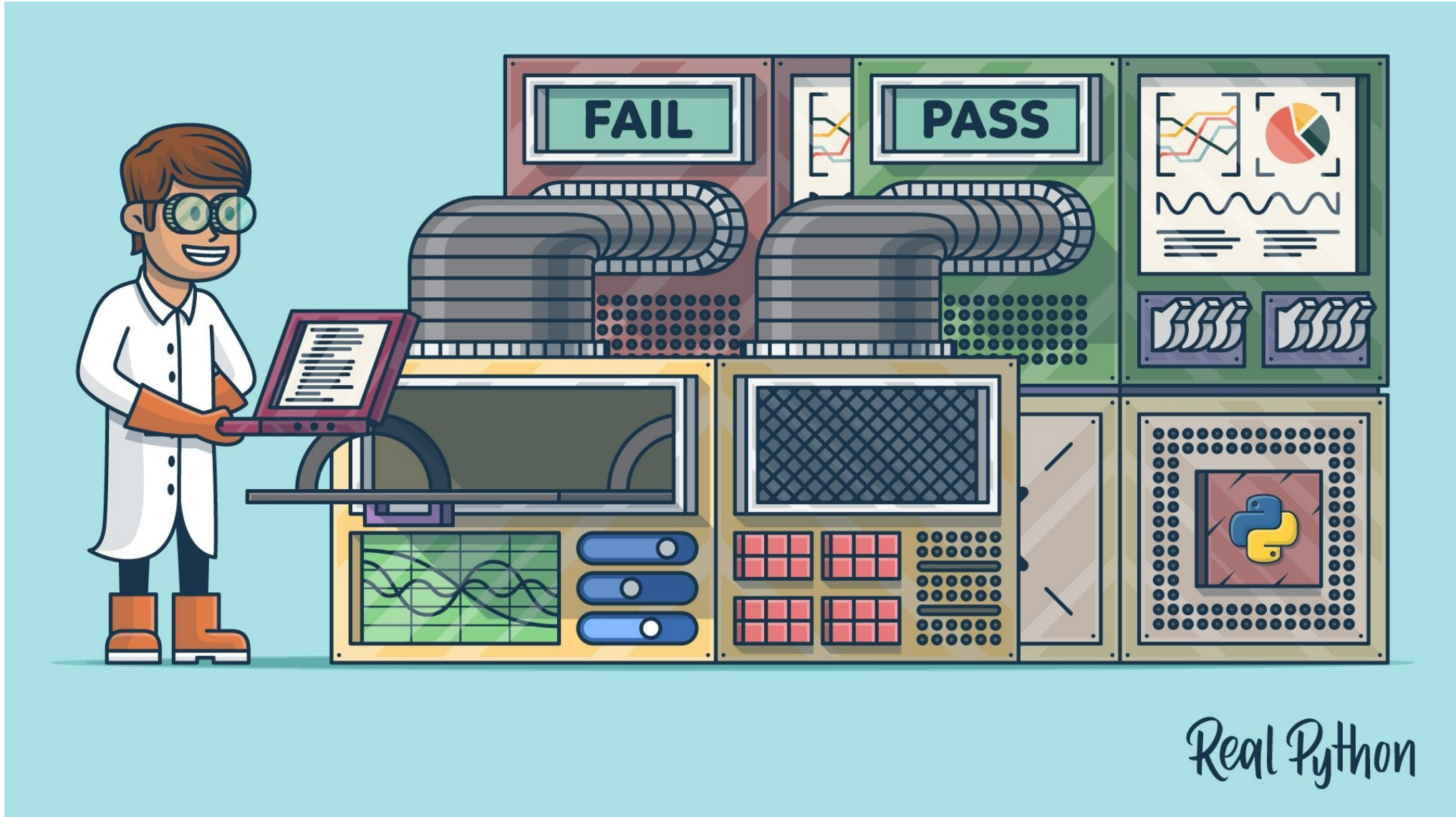
이번 실습의 목적

- 내가 만든 프로그램에 대해..
 - 남아있는 결함을 발견할 수 있다
 - 요구명세서를 만족하는지 확인할 수 있다
 - 사용자 및 비즈니스 요구 충족을 확인할 수 있다
 - 결함을 예방할 수 있다

CI/CD (Continuous Integration / Development)



Testing



<https://realpython.com/python-testing/>

Testing의 주목적

- 간단하고 확장가능한 테스트케이스(Test cases)
 - UI, 데이터베이스, APIs
- 주로 API를 위한 테스트 코드 작성을 위해 사용되었음
- 단위 테스트(Unit test)부터 복잡한 기능 테스트까지 지원

- 추가적인 목적
 - 품질 수준에 대한 자신감 획득과 정보 제공함
 - 비즈니스 리스크를 감소시키는 정보에 근거한 조언함
 - 개발 프로세스 점검, 이슈 제기함
 - 논리적 설계의 구현을 검증함



Generating Prime Number

dark

light

sublime

vim

emacs

C ▾

Description

Suppose that you want to count the number of cases where a prime number is generated when adding 3 numbers among the given numbers. Given an array `nums` containing numbers as the parameter, write a function `solution` to return the number of cases where a prime number is generated when adding 3 different numbers among `nums`.

Constraints

- Length of `nums` is between 3 and 50.
- Each element of `nums` is a natural number between 1 and 1,000, and there are no duplicate numbers.

Examples

nums	result
[1,2,3,4]	1
[1,2,7,6,4]	4

Example #1

7 can be generated when using [1,2,4].

Example #2

7 can be generated when using [1,2,4].

11 can be generated when using [1,4,6].

13 can be generated when using [2,4,7].

17 can be generated when using [4,6,7].

solution.c

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #include <stdlib.h>
4
5 // nums_len은 배열 nums의 길이입니다.
6 int solution(int nums[], size_t nums_len) {
7     int answer = -1;
8     return answer;
9 }
```

Result

Result of [Run Test] or [Submit] will be displayed here



PyTest (CI/CD Tools)



- 매우 간단한 Syntax
- 병렬 테스트 가능
- 특정 테스트와 테스트의 서브셋을 실행가능
- 자동으로 테스트 검출



Contents

- Install pytest
- Your First pytest
- Assertions in pytest
- Run subsets of the entire test
- pytest Fixtures
- Parametrized Tests
- Writing test results
- pytest Framework Testing an API

Install PyTest

vs. Unittest

pytest-vs-unittest

Comparison between pytest and unittest test frameworks

##Comparison Table

Feature	Pytest	Unittest	Winner
Installation	Third Party	Built in	Unittest
Basic Infra	Can be only a function	Inheritance	Pytest
Basic Assertion	Builtin assert	TestCase instance methods	Pytest
Flat is better than nested	Function (1 level)	Method (2 level)	Pytest
Can run each other test	Can run unittest tests	Can't pytest test	Pytest
Test Result on console	Error Highlight, code snippet	Only line error, no highlight	Pytest
Multi param test	Yes, parametrize, keep flat	Yes, sub-test, increase nesting	Pytest
Test setup	fixture: module, session, function	Template Method: setup, tearDown	Pytest
Name Refactoring	poor, because of name conventions	rich, regular object orientation	Unittest
Running Failed Tests	built in (--lf, --ff)	your own =,(Pytest
Marks	built in	your own =,(Pytest

<https://github.com/renzon/pytest-vs-unittest>



Pytest 설치, 도움말

■ !pip install pytest (7.1.1)

1 !pip install pytest

```
Requirement already satisfied: pytest in /usr/local/lib/python3.7/dist-packages (3.6.4)
Requirement already satisfied: atomicwrites>=1.0 in /usr/local/lib/python3.7/dist-packages (from pytest) (1.4.0)
Requirement already satisfied: more-itertools>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from pytest) (8.12.0)
Requirement already satisfied: pluggy<0.8,>=0.5 in /usr/local/lib/python3.7/dist-packages (from pytest) (0.7.1)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-packages (from pytest) (21.4.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (from pytest) (1.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from pytest) (57.4.0)
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from pytest) (1.11.0)
```

1 !pytest -h

```
--log-cli-date-format=LOG_CLI_DATE_FORMAT
                        log date format as used by the logging module.
--log-file=LOG_FILE    path to a file when logging will be written to.
--log-file-level=LOG_FILE_LEVEL
                        log file logging level.
--log-file-format=LOG_FILE_FORMAT
                        log format as used by the logging module.
--log-file-date-format=LOG_FILE_DATE_FORMAT
                        log date format as used by the logging module.

typeguard:
  --typeguard-packages=TYPEGUARD_PACKAGES
                        comma separated name list of packages and modules to
                        instrument for type checking
```

Your First PyTest

test_sample.py 생성

test_sample.py ✕

```
1 import pytest
2 def test_file1_method1():
3     x=5
4     y=6
5     assert x+1 == y,"test failed"
6     assert x == y,"test failed"
7 def test_file1_method2():
8     x=5
9     y=6
10    assert x+1 == y,"test failed"
```

Run py.test



1 !py.test

```
===== test session starts =====  
platform linux -- Python 3.7.13, pytest-3.6.4, py-1.11.0, pluggy-0.7.1  
rootdir: /content, inifile:  
plugins: typeguard-2.7.1  
collected 2 items
```

```
test_sample.py F. [100%]
```

```
===== FAILURES =====
```

```
----- test_file1_method1 -----
```

```
def test_file1_method1():  
    x=5  
    y=6  
    assert x+1 == y, "test failed"  
>    assert x == y, "test failed"  
E   AssertionError: test failed  
E   assert 5 == 6
```

```
test_sample.py:6: AssertionError
```

```
===== 1 failed, 1 passed in 0.05 seconds =====
```

Assertions in PyTest

PyTest Assertion

- True 혹은 False 상태를 리턴에 대한 체크
 - 테스트에서 assertion 이 실패하면, 메소드 실행이 중지되고
 - 메소드 내 남아있는 코드들은 실행되지 않음
 - 그리고 다음 테스트 메소드들을 실행됨

fail

1

```
1 assert "hello" == "Hai"
2 assert 4==4
3 assert True
4 assert False
```



Run subsets of the entire test

어떻게 PyTest는 Test 파일과 메소드를 식별하는가?

- Test에 대한 Prefix로 탐지한다
- Test 파일은 다음과 같은 접두어/접미어

- test_
- _test

- Test 메소드는 test 키워드로 시작한 것만 인정

```
def test_file1_method1():  
def testfile1_method1():  
def file1_method1():  
def file1_test_method1():
```

3,4

test_login.py
login_test.py
testlogin.py
logintest.py

하나의 파일 혹은 여러 개 테스트 파일을 수행하는 방법

- 여러 개 실행하기 (하위 폴더까지 실행한다)

- `py.test`

- 하나의 파일만 수행하기 위해서는

- `py.test test_sample1.py`

전체 테스트의 하위 집합만 실행하는 방법

- Substring matching과 함께 이름의 테스트 그룹
- Marker에 의한 테스트 그룹

```
test_sample1.py
@pytest.mark.set1
def test_file1_method1():

@pytest.mark.set2
def test_file1_method2():
```

```
test_sample2.py
@pytest.mark.set1
def test_file1_method1():

@pytest.mark.set1
def test_file1_method2():
```



옵션1) Substring matching과 함께 이름의 테스트 그룹

- `py.test -k <expression> -v`
 - `-k <expression>` is used to represent the substring to match
 - `-v` increases the verbosity

■ 함수명이 method1것만 실행 시키시오

```
===== test session starts =====
platform linux -- Python 3.7.13, pytest-3.6.4, py-1.11.0, pluggy-0.7.1 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content, inifile:
plugins: typeguard-2.7.1
collected 4 items / 2 deselected

test_sample1.py::test_file1_method1 FAILED [ 50%]
test_sample2.py::test_file2_method1 FAILED [100%]
```

Marker에 의한 테스트 그룹

- `@pytest.mark` -> 마커(Marker)
- `py.test -m <name>`
 - `-m <name>` mentions the marker name

- 마커의 이름이 set2것만 실행시키시오

```
===== test session starts =====
platform linux -- Python 3.7.13, pytest-3.6.4, py-1.11.0, pluggy-0.7.1
rootdir: /content, inifile:
plugins: typeguard-2.7.1
collected 4 items / 3 deselected

test_sample1.py . [100%]

===== 1 passed, 3 deselected in 0.01 seconds =====
```

PyTest Fixtures

- 매 테스트 메소드 전에 일부코드를 반복적으로 실행하고 싶을때,
 - Fixtures: 반복되는 고정된 코드
 - 데이터베이스 연결 등
- @pytest.fixture

Pytest Fixtures

```
1 import pytest
2 @pytest.fixture
3 def supply_AA_BB_CC():
4     aa=25
5     bb =35
6     cc=45
7     return [aa,bb,cc]
8
9 def test_comparewithAA(supply_AA_BB_CC):
10     zz=35
11     assert supply_AA_BB_CC[0]==zz,"aa and zz comparison failed"
12
13 def test_comparewithBB(supply_AA_BB_CC):
14     zz=35
15     assert supply_AA_BB_CC[1]==zz,"bb and zz comparison failed"
16
17 def test_comparewithCC(supply_AA_BB_CC):
18     zz=35
19     assert supply_AA_BB_CC[2]==zz,"cc and zz comparison failed"
```



```
1 | !py.test test_fixture.py
```

```
===== test session starts =====
platform linux -- Python 3.7.13, pytest-7.1.2, pluggy-1.0.0
rootdir: /content
plugins: forked-1.4.0, xdist-2.5.0, typeguard-2.7.1
collected 3 items

test_fixture.py F.F [100%]

===== FAILURES =====
_____ test_comparewithAA _____

supply_AA_BB_CC = [25, 35, 45]

    def test_comparewithAA(supply_AA_BB_CC):
        zz=35
>       assert supply_AA_BB_CC[0]==zz, "aa and zz comparison failed"
E       AssertionError: aa and zz comparison failed
E       assert 25 == 35

test_fixture.py:11: AssertionError
_____ test_comparewithCC _____

supply_AA_BB_CC = [25, 35, 45]

    def test_comparewithCC(supply_AA_BB_CC):
        zz=35
>       assert supply_AA_BB_CC[2]==zz, "cc and zz comparison failed"
E       AssertionError: cc and zz comparison failed
E       assert 45 == 35

test_fixture.py:19: AssertionError
===== short test summary info =====
FAILED test_fixture.py::test_comparewithAA - AssertionError: aa and zz compar...
FAILED test_fixture.py::test_comparewithCC - AssertionError: cc and zz compar...
===== 2 failed, 1 passed in 0.08s =====
```

여러 test파일에 같은 Fixture를 적용하기

```
1 import pytest
2 @pytest.fixture
3 def supply_AA_BB_CC():
4     aa=25
5     bb =35
6     cc=45
7     return [aa,bb,cc]
```

```
1 import pytest
2 def test_comparewithAA(supply_AA_BB_CC):
3     zz=35
4     assert supply_AA_BB_CC[0]==zz,"aa and zz comparison failed"
5
6 def test_comparewithBB(supply_AA_BB_CC):
7     zz=35
8     assert supply_AA_BB_CC[1]==zz,"bb and zz comparison failed"
9
10 def test_comparewithCC(supply_AA_BB_CC):
11     zz=35
12     assert supply_AA_BB_CC[2]==zz,"cc and zz comparison failed"
```

```
1 import pytest
2 def test_comparewithAA(supply_AA_BB_CC):
3     zz=35
4     assert supply_AA_BB_CC[0]==zz,"aa and zz comparison failed"
5
6 def test_comparewithBB(supply_AA_BB_CC):
7     zz=35
8     assert supply_AA_BB_CC[1]==zz,"bb and zz comparison failed"
9
10 def test_comparewithCC(supply_AA_BB_CC):
11     zz=35
12     assert supply_AA_BB_CC[2]==zz,"cc and zz comparison failed"
```

```
===== test session starts =====
platform linux -- Python 3.7.13, pytest-7.1.2, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: forked-1.4.0, xdist-2.5.0, typeguard-2.7.1
collected 10 items / 4 deselected / 6 selected

test_basic_fixture.py::test_comparewithAA FAILED [ 16%]
test_basic_fixture.py::test_comparewithBB PASSED [ 33%]
test_basic_fixture.py::test_comparewithCC FAILED [ 50%]
test_basic_fixture2.py::test_comparewithAA FAILED [ 66%]
test_basic_fixture2.py::test_comparewithBB PASSED [ 83%]
test_basic_fixture2.py::test_comparewithCC FAILED [100%]
```

Parameterized Test

Test를 parameterize하는 목적

- Arguments(입/출력)의 집합에 대한 테스트
- @pytest.mark.parametrize

```
1 import pytest
2 @pytest.mark.parametrize("input1, input2, output", [(5,5,10), (3,5,12)])
3 def test_add(input1, input2, output):
4     assert input1+input2 == output, "failed"
```

```

===== test session starts =====
platform linux -- Python 3.7.13, pytest-7.1.2, pluggy-1.0.0
rootdir: /content
plugins: forked-1.4.0, xdist-2.5.0, typeguard-2.7.1
collected 2 items

test_addition.py .F [100%]

===== FAILURES =====
----- test_add[3-5-12] -----

input1 = 3, input2 = 5, output = 12

    @pytest.mark.parametrize("input1, input2, output",[(5,5,10),(3,5,12)])
    def test_add(input1, input2, output):
>         assert input1+input2 == output,"failed"
E       AssertionError: failed
E       assert (3 + 5) == 12

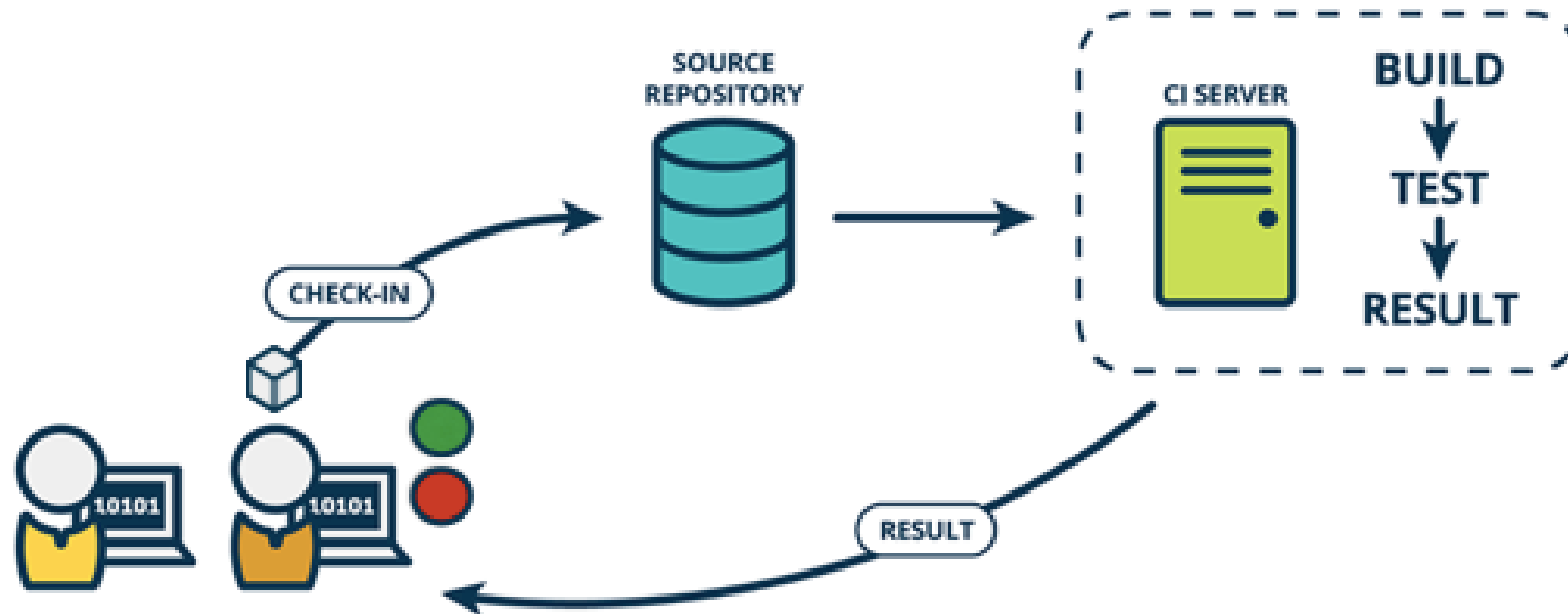
test_addition.py:4: AssertionError
===== short test summary info =====
FAILED test_addition.py::test_add[3-5-12] - AssertionError: failed
===== 1 failed, 1 passed in 0.07s =====

```



Writing testing results

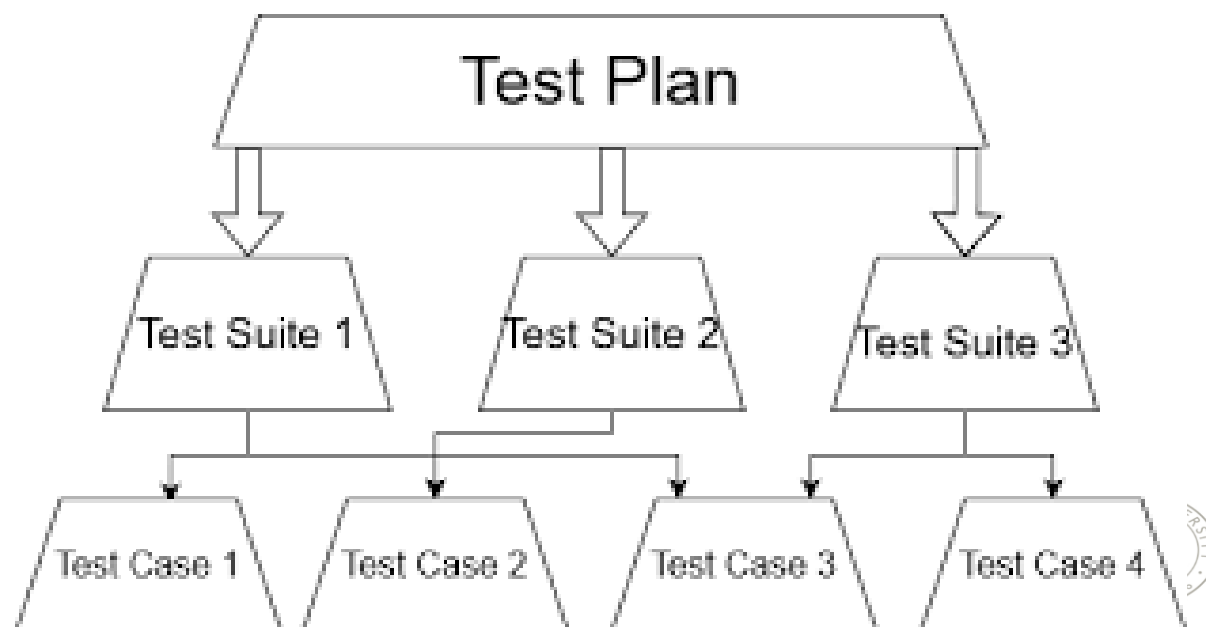
CONTINUOUS INTEGRATION



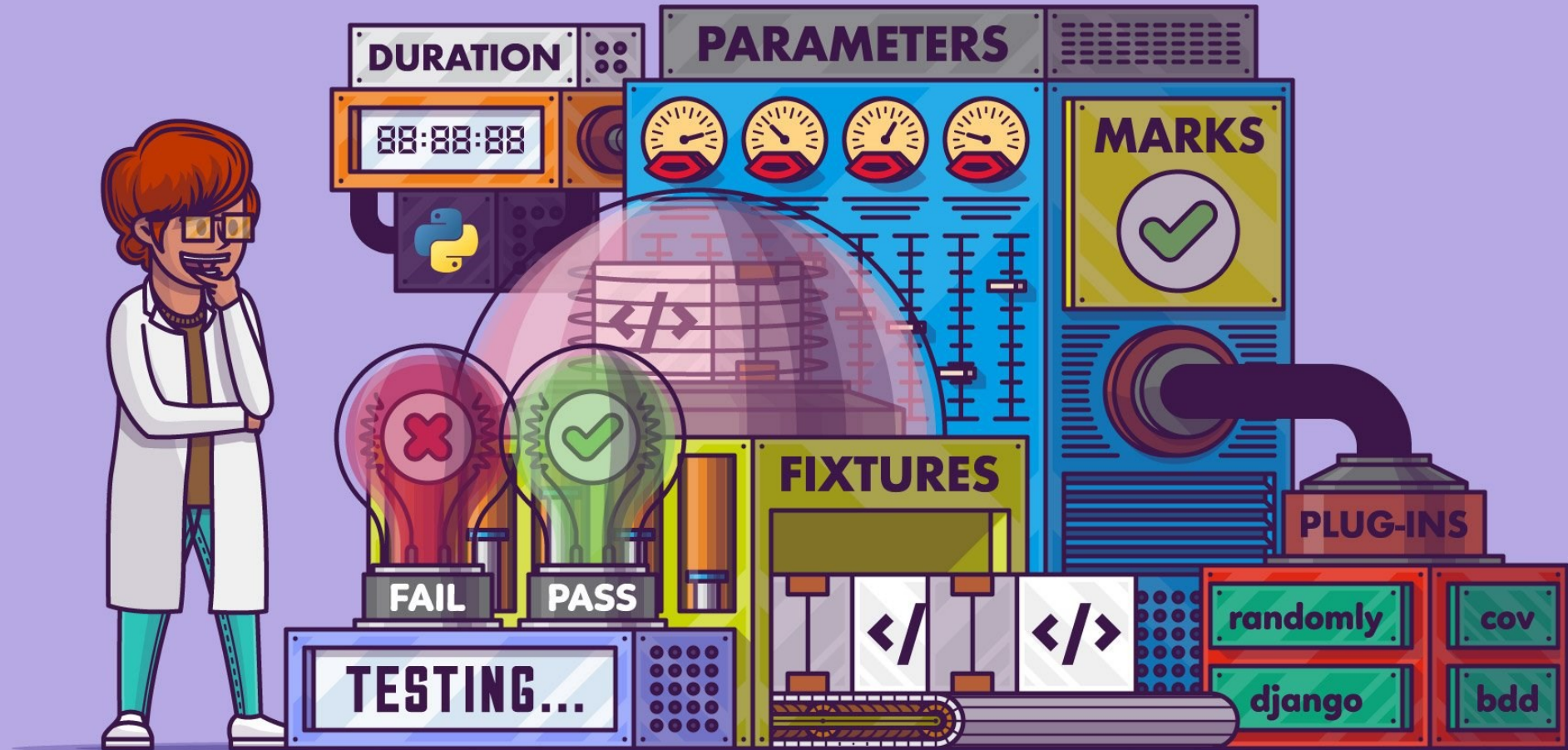
XML format 형태로 테스트의 결과를 출력

■ `py.test test_sample1.py -v --junitxml="result.xml"`

```
▼<testsuites>
  ▼<testsuite errors="0" failures="1" hostname="17104684703d" name="pytest" skipped="0"
    tests="2" time="0.037" timestamp="2022-05-06T14:50:42.009703">
      <testcase classname="test_addition" name="test_add[5-5-10]" time="0.001"/>
      ▼<testcase classname="test_addition" name="test_add[3-5-12]" time="0.001">
        <failure message="AssertionError: failed assert (3 + 5) == 12">input1 = 3, input2 = 5,
        output = 12 @pytest.mark.parametrize("input1, input2, output", [(5,5,10),(3,5,12)]) def
        test_add(input1, input2, output): > assert input1+input2 == output,"failed" E
        AssertionError: failed E assert (3 + 5) == 12 test_addition.py:4:
        AssertionError</failure>
      </testcase>
    </testsuite>
  </testsuites>
```



Summary



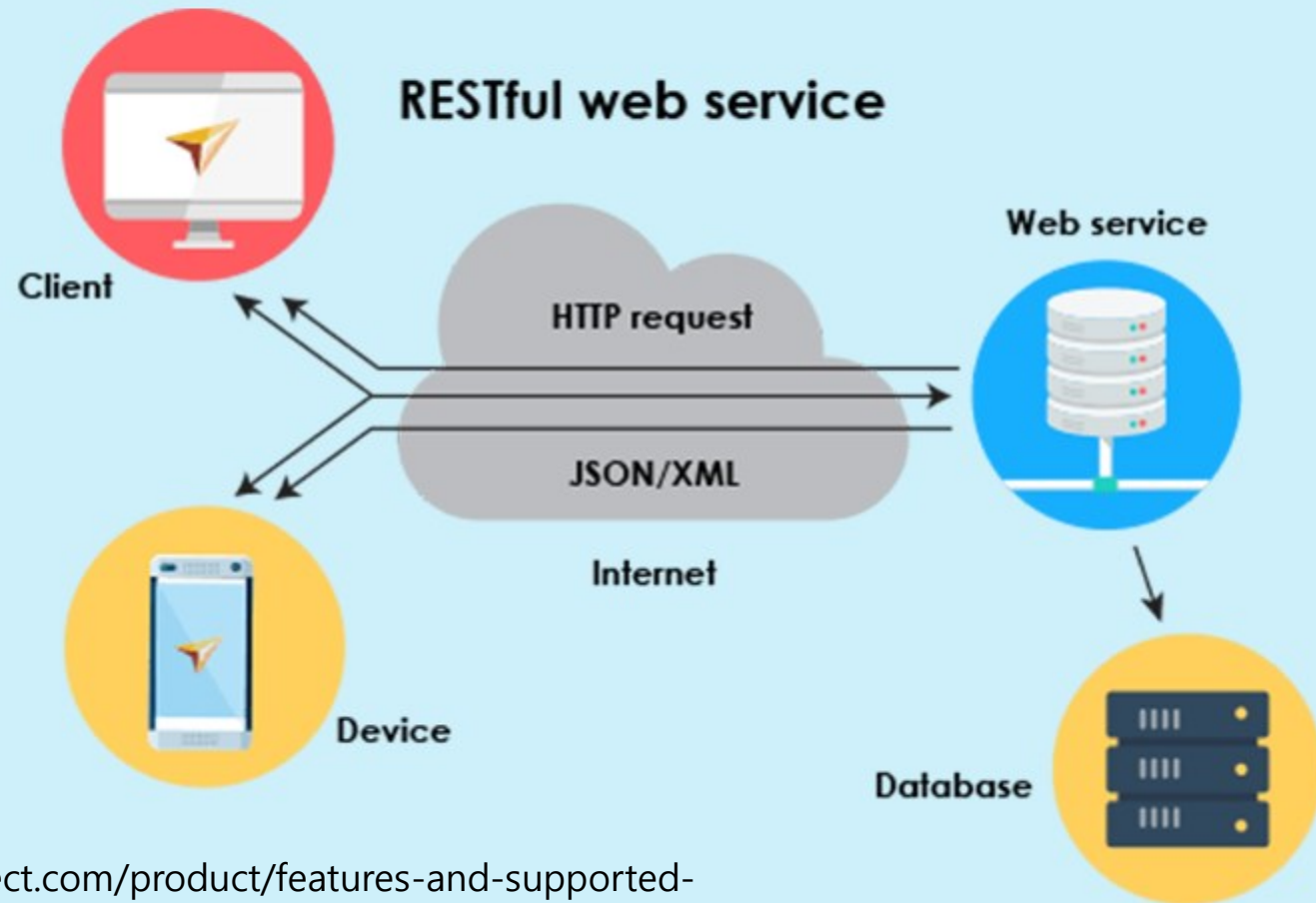
Real Python

<https://realpython.com/python-assert-statement/>


PyTest Framework Testing an API

API 테스트

Web Service & API Testing



Source: <https://www.testarchitect.com/product/features-and-supported-platforms/testarchitect-supported-platforms/web-service-and-api-testing>



Test your front-end against a real API

Fake data

No more tedious sample data creation, we've got it covered.

Real responses

Develop with real response codes. GET, POST, PUT & DELETE supported.

Always-on

24/7 **free** access in your development phases. Go nuts.

A hosted REST-API ready to respond to your AJAX requests.

Some tests

- 일부 사용자에 대한 List
 - 유효한 사용자 여부 체크
 - 유효하지 않은 사용자 여부 체크
- 사용자에 대한 로그인
 - 이메일과 비밀번호와 함께 유효한 로그인 시도
 - 비밀번호 없이 비유효한 로그인 시도
 - 이메일 없이 비유효한 로그인 시도

일부 사용자에게 대한 List (1)

- import pytest
- import requests `# request.get(url)`
- import json

■ 유효한 사용자 여부 체크

- `@pytest.mark.parametrize("userid, firstname",[(1,"George"),(2,"Janet")])`
- url
- resp
 - `resp.text`
 - `resp.status_code`
- jsonObj
 - `json.loads`

		Request	Response
		<code>/api/users/2</code>	200
GET	LIST USERS		
GET	SINGLE USER		
GET	SINGLE USER NOT FOUND		
GET	LIST <RESOURCE>		
GET	SINGLE <RESOURCE>		
GET	SINGLE <RESOURCE> NOT FOUND		

```
{
  "data": {
    "id": 2,
    "email": "janet.weaver@reqres.in",
    "first_name": "Janet",
    "last_name": "Weaver",
    "avatar": "https://reqres.in/img/fac"
  },
  "support": {
    "url": "https://reqres.in/#support-h",
    "text": "To keep ReqRes free, contri"
  }
}
```


일부 사용자에게 대한 List (2)

- import pytest
- import requests # request.get(url)
- import json
- 유효하지 않은 사용자 여부 체크
 - url
 - resp
 - resp.text
 - resp.status_code
 - jsonObj
 - json.loads

GET	LIST USERS
GET	SINGLE USER
GET	SINGLE USER NOT FOUND

Request /api/users/23	Response 404
	{}

사용자에 대한 로그인 (1)

- import pytest
- import requests `# request.post(url, data=data)`
- import json
- 이메일과 패스워드와 함께 유효한 로그인 시도
 - `data = {'email':'test@test.com','password':'something'}`
 - url
 - resp
 - `resp.text`
 - `resp.status_code`
 - jsonObj
 - `json.loads`

POST	LOGIN - SUCCESSFUL
POST	LOGIN - UNSUCCESSFUL
GET	DELAYED RESPONSE

Request
`/api/login`

```
{  
  "email": "eve.holt@reqres.i  
  "password": "cityslicka"  
}
```

Response
200

```
{  
  "token": "QpwL5tke4Pnpja7X4"  
}
```

사용자에 대한 로그인 (2)

- import pytest
- import requests **# request.post(url, data=data)**
- import json
- **패스워드 없이 비유효한 로그인 시도**
 - data = {'email':'test@test.com'}
 - url
 - resp
 - resp.text
 - resp.status_code
 - jsonObj
 - json.loads

POST	LOGIN - SUCCESSFUL	Request /api/login	Response 400
POST	LOGIN - UNSUCCESSFUL	{ "email": "peter@klaven" }	{ "error": "Missing password" }
GET	DELAYED RESPONSE		

사용자에 대한 로그인 (2)

- import pytest
- import requests `# request.post(url, data=data)`
- import json
- 패스워드 없이 비유효한 로그인 시도
 - data = {}
 - url
 - resp
 - resp.text
 - resp.status_code
 - jsonObj
 - json.loads

POST	LOGIN - SUCCESSFUL	Request /api/login	Response 400
POST	LOGIN - UNSUCCESSFUL	<pre>{ "email": "peter@klaven" }</pre>	<pre>{ "error": "Missing password" }</pre>
GET	DELAYED RESPONSE		

- pytest에서 Assertions 사용법 확인
- 일부 테스트만 사용하는 방법
 - Substring match, Marker
- PyTest Fixtures
 - 여러 개의 test파일을 위한 고정된 변수
- Parametrized Tests
 - 파라미터화한 테스트, 예) 코딩테스트
- Output results
 - 다른 시스템에 전달하기 위한 테스트
- PyTest Framework for Testing an API

test

fxiure