파이썬 클린 코드

(CLEAN CODE IN PYTHON)

Chapter1_ 코드 포매팅과 도구

클린코드

기술부채를 최소화, 가독성과 유지보수성 그리고 타인의 이해도를 높이는 효과적인 코드의 작성방법

- 클린 코드는 포매팅 이상의 훨씬 중요한것을 의미
- 표준 포매팅을 유지하는 것이 유지보수성의 핵심 유의사항이다
- **파이썬이 제공하는 기능을 사용하여 자체 문서화된 코드를 작성하는 방법**
- **코드의 레이아웃을 일정하게 유지하여 팀 멤버들이 문제의 본질을 해결하는데 초점을 맞출 수 있도록 도구를 설정하는 방법**

Docstring

: 소스 코드에 포함된 문서

```
def is_digit(user_input_number):
    '''
    Input:
      - user_input_number : 문자열 값
    Output:
      - user_input_number가 정수로 변환 가능할 경우는 True,
        그렇지 않을 경우는 False
    Examples:
      >>> import baseball_game as bg
      >>> bg.is_digit("551")
      True
      >>> bg.is_digit("103943")
      True
      >>> bg.is_digit("472")
      True
      >>> bg.is_digit("1032.203")
      False
      >>> bg.is_digit("abc")
      False
    '''
    # ===Modify codes below=============
    # 조건에 따라 변환되어야 할 결과를 result 변수에 할당
    result = None

    # ==================================
    return result
```

```
class Test(object):
    """두 개의 int 값을 입력받아 다양한 연산을 할 수 있도록 하는 클래스.

    :param int a: a 값
    :param int b: b 값
    """

    def __init__(self, a, b):
        self._a = a
        self._b = b

    def is_same(self):
        """미리 입력받은 a와 b값이 같은지 확인하여 결과를 반환합니다.

        :return: boolean True or False에 대한 결과, a와 b가 값으면 True, 다르면 False

        예제:
            다음과 같이 사용하세요:

            >>> Test(1, 2).is_same()
            False

        """
        return self._a == self._b

    def plus(self):
        """미리 입력받은 a와 b값을 더한 결과를 반환합니다.

        :returns: int a + b에 대한 결과

        예제:
            다음과 같이 사용하세요:

            >>> Test(2, 1).plus()
            3


        """
        return self._a + self._b
```

Docstring

: 소스 코드에 포함된 문서

```
def is_digit(user_input_number):
    '''
    Input:
      - user_input_number : 문자열 값
    Output:
      - user_input_number가 정수로 변환 가능할 경우는 True,
        그렇지 않을 경우는 False
    Examples:
      >>> import baseball_game as bg
      >>> bg.is_digit("551")
      True
      >>> bg.is_digit("103943")
      True
      >>> bg.is_digit("472")
      True
      >>> bg.is_digit("1032.203")
      False
      >>> bg.is_digit("abc")
      False
    '''
    # ===Modify codes below=============
    # 조건에 따라 변환되어야 할 결과를 result 변수에 할당
    result = None

    # ==================================
    return result
```

Docstring을 포함하는것이 좋은이유?

→ PYTHON은 동적 타이핑이기 때문

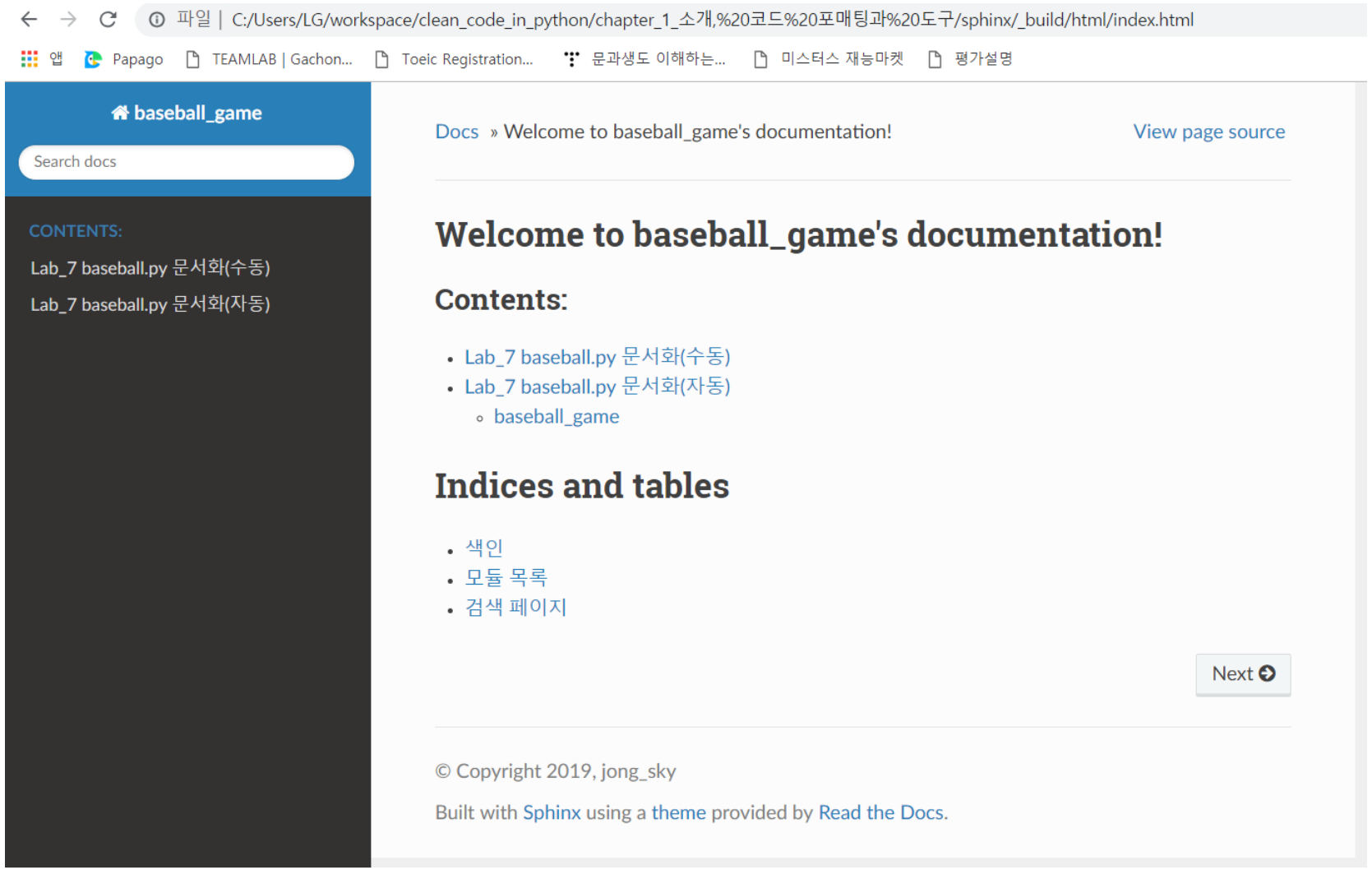
(또는 프로젝트 표준에 따라 필수일 수도 있음)

파이썬은 파라미터의 타입을 체크하거나 강요하지 않는다.

함수의 이름과 파라미터의 이름이 충분히 설명적이라면 좋다.
그러나 그렇지 않은 경우에는 docstring이 도움이 될 것이다.

Sphinx(스핑크스)

: 프로젝트 문서화를 위한 기본 골격을 만들어주는 모듈



Sphinx(스핑크스)

: 프로젝트 문서화를 위한 기본 골격을 만들어주는 모듈

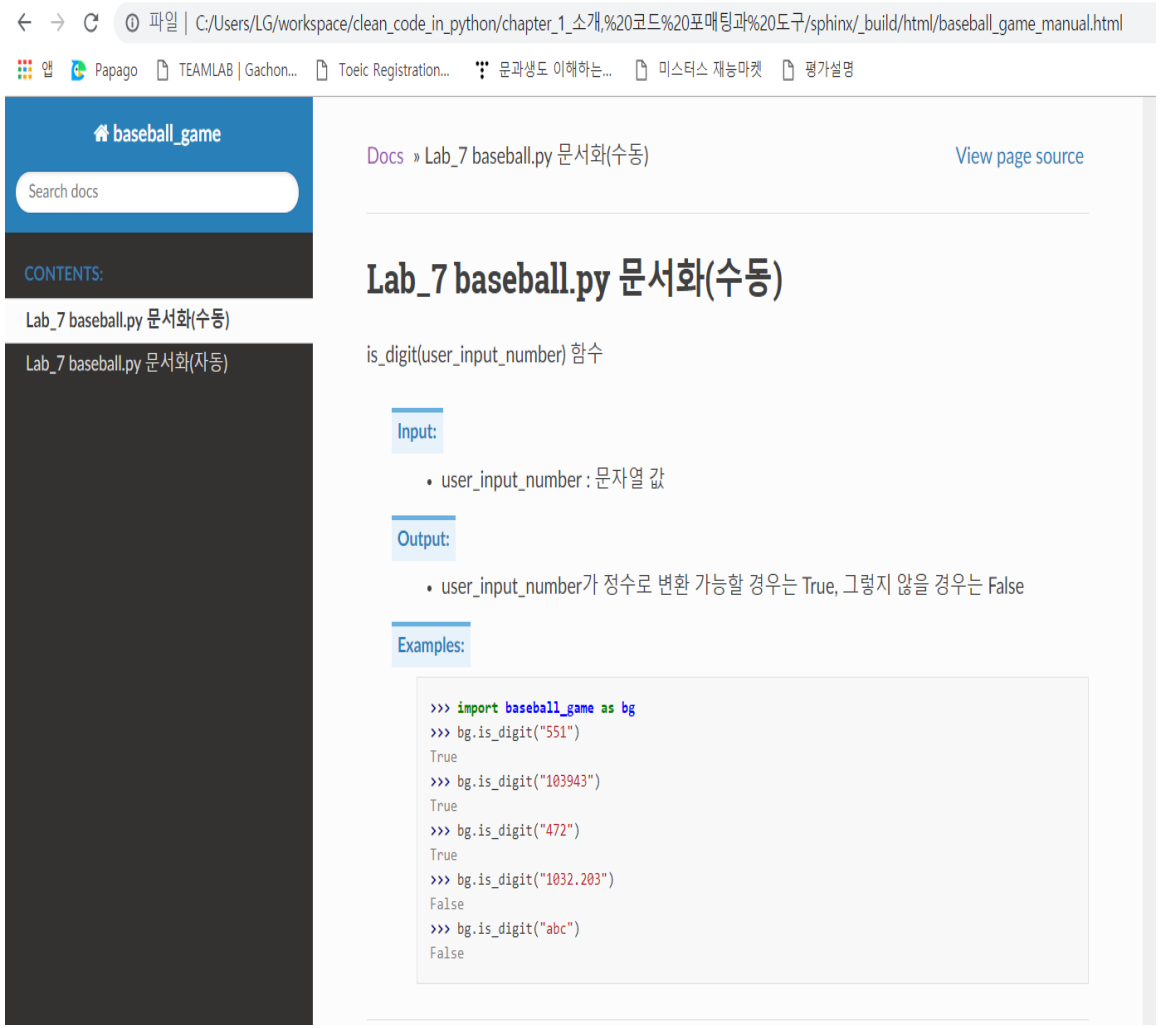
```
.. baseball documentation master file, created by
   sphinx-quickstart on Thu May  2 22:24:18 2019.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Lab_7 baseball.py 문서화(수동)
==================================


is_digit(user_input_number) 함수

    Input:
      - user_input_number : 문자열 값
    Output:
      - user_input_number가 정수로 변환 가능할 경우는 True,
        그렇지 않을 경우는 False
    Examples:
      >>> import baseball_game as bg
      >>> bg.is_digit("551")
      True
      >>> bg.is_digit("103943")
      True
      >>> bg.is_digit("472")
      True
      >>> bg.is_digit("1032.203")
      False
      >>> bg.is_digit("abc")
      False

==================================
```

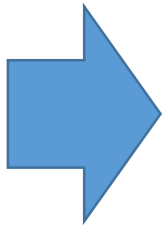


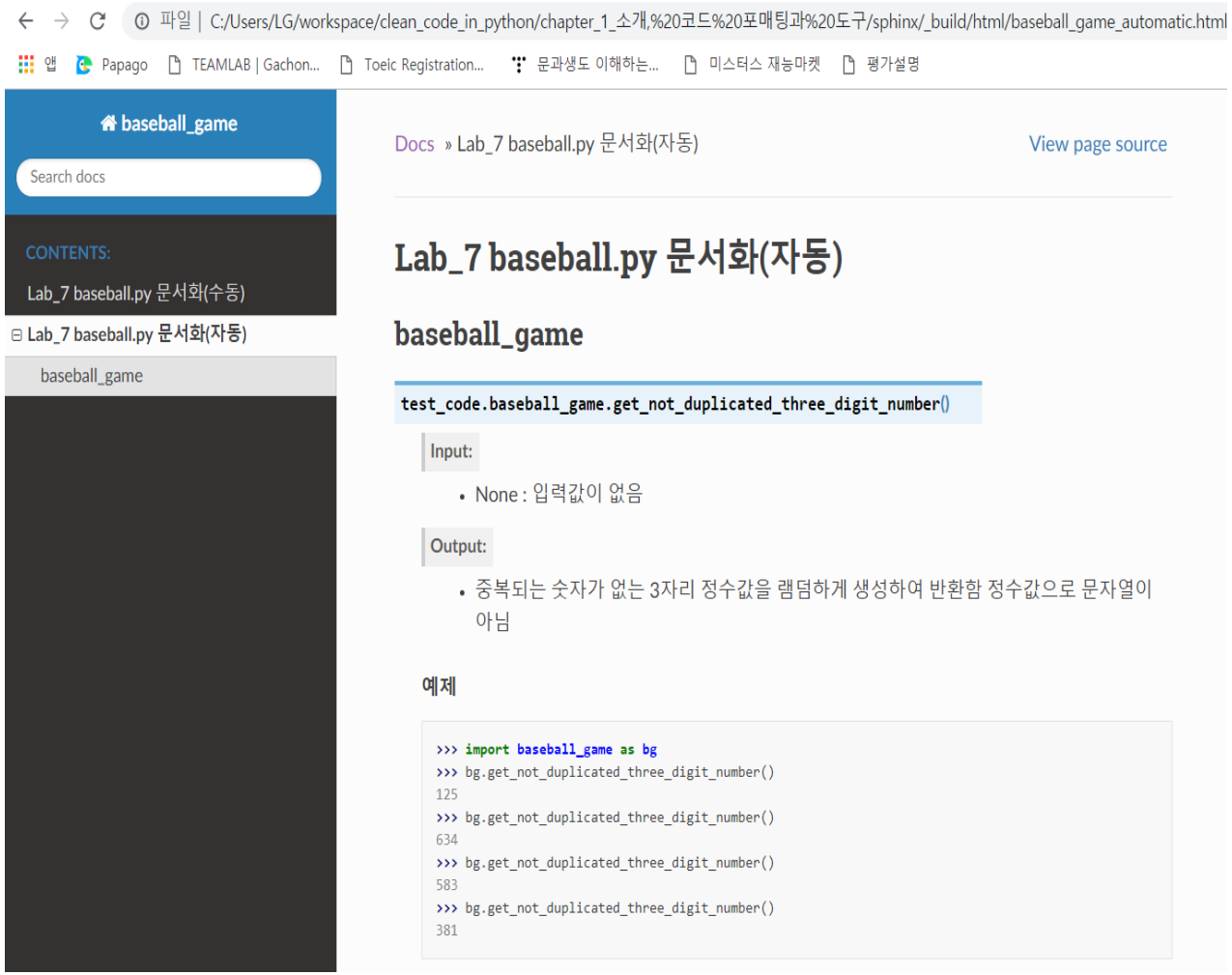
Sphinx(스핑크스)

: 프로젝트 문서화를 위한 기본 골격을 만들어주는 모듈 (autodoc 익스텐션)

```
Lab_7 baseball.py 문서화(자동)
==================================

.. automodule:: test_code.baseball_game
    :members:
```





어노테이션(annotations)

: 주석(사전적의미), 함수 인자로 어떤 값이 와야 하는지 힌트를 주는 것

```
In [68]: def add_number(a:float,b:float)->float:
             c = a+b
             return c

In [69]: add_number.__annotations__
Out[69]: {'a': float, 'b': float, 'return': float}

In [70]: add_number("안녕","반가워")
Out[70]: '안녕반가워'
```

```
def data_from_response(response: dict)->dict:
    if response["status"] != 200:
        raise ValueError
    return {"data":response["payload"]}
```

1. response객체의 올바른 인스턴스는 어떤 형태일까?
2. 결과의 인스턴스는 어떤 형태일까?

→ 파라미터와 함수 반환 값의 예상 형태를 docstring으로 문서화

어노테이션(annotations)

: 주석(사전적의미), 함수 인자로 어떤 값이 와야 하는지 힌트를 주는 것

```
def data_from_response(response:dict) -> dict:
    """ response에 문제가 없다면 response의 payload를 반환

    - response 사전의 예제::
    {
        "status":200, #<int>
        "timestamp":"....", # 현재 시간의 ISO 포맷 문자열
        "payload":{....} # 반환하려는 사전 데이터
    }

    - 반환 사전 값의 예제::
    {"data":{..} }

    - 발생 가능한 예외:
    - HTTP status가 200이 아닌 경우 ValueError 발생
    """
    if response["status"] != 200:
        raise ValueError
    return {"data":response["payload"]}
```

```
def data_from_response(response: dict)->dict:
    if response["status"] != 200:
        raise ValueError
    return {"data":response["payload"]}
```

1. response객체의 올바른 인스턴스는 어떤 형태일까?
2. 결과의 인스턴스는 어떤 형태일까?

→ 파라미터와 함수 반환 값의 예상 형태를 docstring으로 문서화

PEP(Python Enhance Proposal)

: 파이썬을 개선하기 위한 개선 제안서 (PEP-8:파이썬언어의 컨벤션제안서)

Code lay-out

- 들여쓰기는 공백 4칸을 권장합니다.
- 한 줄은 최대 79자까지
- 최상위(top-level) 함수와 클래스 정의는 2줄씩 띄어 씁니다.
- 클래스 내의 메소드 정의는 1줄씩 띄어 씁니다.

Comments

- 코드와 모순되는 주석은 없느니만 못합니다. 항상 코드에 따라 갱신해야 합니다.
- 불필요한 주석은 달지 마세요.
- 한 줄 주석은 신중히 다세요.
- 문서화 문자열(Docstring)에 대한 컨벤션은 PEP 257을 참고하세요.

다양한 검사 도구

Mypy : 정적 타입 검사 도구(모든 파일을 분석하여 타입 불일치를 검사)

Pylint : 코드의 구조를 검사하는 도구 (PEP-8을 준수했는지 여부 검사)

Makefile : (리눅스개발환경에서)빌드를 자동화, 포매팅 검사, 코딩 컨벤션 검사 자동화 도구

Black : Black 고유 방식으로 자동 포매팅 도구 (ex) 문자열에는 항상 쌍따옴표사용) → 코드의 차이 최소화

Black

: 자동으로 포매팅해주는 도구

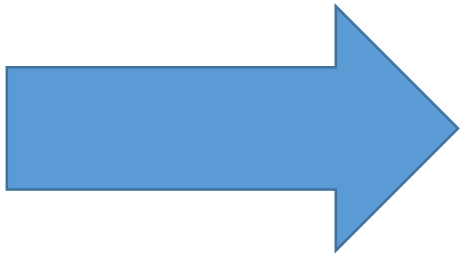
test_black.py

```
def my_function(name):
    """

    >>> my_function('black')
    'received black'
    """

    return 'received {0}'.format(name.title())
```

```
reformatted test_black.py
All done! ✨ 🍰 ✨
1 file reformatted.
```



test_black.py

```
def my_function(name):
    """

    >>> my_function('black')
    'received black'
    """

    return "received {0}".format(name.title())
```

PEP-8에 의하면 올바른 코드지만 Black의 문법을 따르지 않음