Python Comments

PEP 20: Readability counts

참조: https://peps.python.org/pep-0008/#comments

> "One of Guido's key insights is that code is read much more often than it is written."
>
> "Guido(Guido Van Rossum, Python 창시자)는 코드는 보통 쓰여지는 것 보다 읽히는 경우가 더 많다고 지적한다."

Python언어의 창시자인 Guido가 지적하는것과 같이 하나의 코드 베이스는 쓰여지는 시간보다 여러 개발자에 의하여 읽히는 경우가 더 많다.

즉, 좋은 코드는 그 코드의 성능 또는 기능을 떠나서 나 뿐만 아니라 다른 여러 사람이 함께 읽고, 필요에 따라서는 수정하며 공동 개발할 수 있는 가독성이 좋은 코드이다. 하지만 이렇게 가독성이 좋은 코드를 작성하기란 성능이 좋은 알고리즘을 개발하는 것 만큼이나 어렵다. 이번 챕터에서는 comments(주석) 처리를 통하여 조금이나마 코드에 가독성을 더해보자.

이번 챕터에서 진행하는 모든 내용은 가장 상단에 있는 `Style Guide for Python Code`를 참조하여 작성하였다. 시간적 여유가 있을 때 해당 문서를 한번 쯤 읽어보는 것을 추천한다.

Comments

코드의 주석은 항상 코드와 함께 최신화 되어야 한다. 코드가 수정되면 해당 코드 블록에 함께 작성된 주석도 업데이트 되어야한다.

주석의 모든 문장은 시작과 끝이 분명해야하며, 누가 읽든지 분명하게 내용을 전달해야 한다. 주석은 코드를 이해함에 있어 도움을 주기위하여 작성하며, 주석으로 인하여 이해가 더 어려워지는 불필요한 주석은 좋지않다.

Block Comment

블록 코멘트는 보통 `#` 으로 시작되는 주석을 의미하며, 블록 코멘트 하단에 위치한 코드 블록에 대한 설명을 담는다.

블록 코멘트 주석은 코드 블록과 동일한 들여쓰기 깊이(Indentation Level)을 유지한다. `#`으로 시작하며 `#`이후 1칸의 공백을 작성한다.

```
# This block comment applies to the code block below.
# This "list comprehenshion" code creates a list of integers ra
my_list = [x for x in range(11)]
```

```
def add_two(a, b):
    # block code should always have the same level of indentati
    result = a + b
    return result
```

Inline Comment

인라인 코멘트는 코드와 동일한 줄에서 작성되는 주석을 의미한다.

인라인 코멘트는 코드를 읽을 때 방해가 될 수 있으므로 최대한 절제해야하며, 명확한 코드에 대하여 이중적으로 주석을 작성할 필요는 없다.

인라인 코멘트는 코드의 우측에, 코드에서 부터 최소 2개의 공백을 작성한 후, `#` 으로 시작한다.

```
x = x + 1  # Adding 1 to the variable x  (X)
```

```
x = x + 1  # compensate for border  (O)
```

위 예제는 인라인 코멘트에 대한 예제이다.

첫번째 에제와 같이 명확한 코드를 구태어 주석으로 한번 더 설명해줄 필요는 없다. 이와같은 활용은 코드의 가독성을 오히려 저하시킬 수 있다.

두번째 예제의 경우, 주석으로 해당 코드가 어떤 목적으로 작성되었는지, 어떤 동작을 하는지에 대한 "설명"을 한다. 이러한 인라인 코멘트 주석은 때때로 유용하며 코드를 읽는데 도움을 줄 수 있다.

> 💡 위 예시와 같이 변수명을 임의적으로 불분명하게 작성하는 경우 코드만 보고 해당 코드의 역할을 한번에 알아차리기는 어렵다. 따라서 그 역할 및 기능에 대한 inline comment는 코드를 이해하는데에 도움을 줄 수 있다. 하지만 만약 변수명이 명확하게 작성되었다면 inline comment 마저 필요없을 수 있다.
>
> Inline comment는 최소한으로 절제하며 사용하는것이 좋으므로 inline comment를 작성하고자 할 때, inline comment가 꼭 필요로 한지, 코드 자체를 더 분명하게 작성할 수 있는 방법이 있을지 한번쯤은 고민해보자.

Documentation Strings

Documentation Strings (Docstrings)는 ***https://peps.python.org/pep-0257/*** 에 기재되며 공식화 되었다.

Docstring이란 module(모듈), function(함수), class(클래스), or method(매소드)의 가장 첫 라인에 쓰여지며, 해당 오브젝트에 대하여 정의 한다.

이렇게 정의된 Docstring은 해당 오브젝트의 `__doc__` 속성으로 저장된다.

Docstring은 한 문장 또는 여러 문장으로 정의될 수 있으며 앞, 뒤로 세개의 quotation mark (따옴표)를 사용하여 작성한다.

- **Single Line of docstring for a function**

```
def add_two_numbers(a, b):
    """ Add two numbers and return the result. """
    result = a + b
    return result


def add_numbers(*args):
    """ Return the sum of the elements from the arg list. """
    return sum(args)
```

더욱 더 자세한 내용을 내포할 시 여러 줄의 Docstring을 작성할 수 있다.

아래는 총 네가지의 Docstring 포멧에 대한 예제를 보여준다. 모든 내용을 이해할 필요는 없으며, 그냥 어떤식으로 동일한 내용을 다르게 표현했는지 정도만 확인해보자.

출처: https://www.datacamp.com/tutorial/docstrings-python

1. **NumPy/SciPy Docstring**

```
class Vehicles:
    """
    The Vehicles object contains lots of vehicles

    Parameters
    ----------
    arg : str
        The arg is used for ...
    *args
        The variable arguments are used for ...
    **kwargs
        The keyword arguments are used for ...

    Attributes
    ----------
    arg : str
```

```
        This is where we store arg,
    """
    def __init__(self, arg, *args, **kwargs):
        self.arg = arg

    def cars(self, distance, destination):
        """We can't travel distance in vehicles without fuels

        Parameters
        ----------
        distance : int
            The amount of distance traveled
        destination : bool
            Should the fuels refilled to cover the distance?

        Raises
        ------
        RuntimeError
            Out of fuel

        Returns
        -------
        cars
            A car mileage
        """
        pass
```

2. **Google docstrings**

```
class Vehicles:
    """
    The Vehicle object contains a lot of vehicles

    Args:
        arg (str): The arg is used for...
```

```
        *args: The variable arguments are used for...
        **kwargs: The keyword arguments are used for...

    Attributes:
        arg (str): This is where we store arg,
    """
    def __init__(self, arg, *args, **kwargs):
        self.arg = arg

    def cars(self, distance, destination):
        """We can't travel distance in vehicles without fuels

        Args:
            distance (int): The amount of distance traveled
            destination (bool): Should the fuels refilled to

        Raises:
            RuntimeError: Out of fuel

        Returns:
            cars: A car mileage
        """
        pass
```

3. **reStructuredText (Sphinx)**

```
class Vehicle:
    """
    The Vehicle object contains lots of vehicles

    :param arg: The arg is used for ...
    :type arg: str
    :param `*args`: The variable arguments are used for ...
    :param `**kwargs`: The keyword arguments are used for ..
    :ivar arg: This is where we store arg
```

```
    :vartype arg: str
    """



    def __init__(self, arg, *args, **kwargs):
        self.arg = arg

    def cars(self, distance, destination):
        """We can't travel a certain distance in vehicles wi

        :param distance: The amount of distance traveled
        :type distance: int
        :param destination: Should the fuels be refilled to
        :type destination: bool
        :raises: :class:`RuntimeError`: Out of fuel

        :returns: A Car mileage
        :rtype: Cars
        """
        pass
```

4. Epytext

```
class Vechicle:
    """
    The Vechicle object contains lots of bechicles

    @param arg: The arg is used for ...
    @type arg: str
    @param args: The variable arguments are used for ...
    @param kwargs: The keyword arguments are used for ...
    @ivar arg: This is where we store arg
    """

    def __init__(self, arg, *args, **kwargs):
```

```
        self.arg = arg

    def cars(self, distance, destination):
        """We can't travel a certain distance in vechicles wi

        @param distance: The amount of distance traveled
        @type distance: int
        @param destination: Should the fuels be refilled to c
        @type destination: bool
        @raise RuntimeError: Out of fuel

        @return: A Car mileage
        @rtype: Cars
        """
```

위 예제중 Pycharm에서 기본적으로 사용하는 Docstring 포멧은 3번, **reStructuredText (Sphinx)** 타입이며, 특별한 경우가 아니라면 해당 docstring 포멧을 가장 많이 접할 수 있다.