PEP 8 파이썬 코딩 스타일

Python Enhancement Proposal 8 (PEP 8)은 파이썬 코딩 스타일에 대한 가이드를 제시하고 있다. PEP 8은 2001년 귀도 반 로썸에 의해 처음 제안되었으며, python.org 의 PEP 링크에 자세히 소개되어 있다. 파이썬 프로그래머들은 일반적으로 이러한 PEP 8 코딩 스타일에 따라 프로그래밍을 하고 있는데, 이러한 일관된 코딩 스타일을 적용하는 것은 자신의 코드를 명료하게 할 뿐만 아니라 특히 다른 개발자 혹은 커뮤니티간 코딩을 공유할 때 매우 효율적이다.

아래는 PEP 8 의 중요한 부분에 대한 요약이다.

카테고리	코딩 스타일	예제
코드 레이아웃	들여쓰기를 할 때 Tab 대신 공백(Space)을 사용한다. 특히 Python 3는 Tab과 공백을 혼용해서 사용하는 것을 허용하지 않는다.	
	문법적으로 들여쓰기를 할 때는 4개의 공백을 사용한다	
	각 라인은 79자 이하로 한다. 라인이 길어서 다음 라인으로 넘어갈 때는 원래 들여쓰기 자리에서 4개 공백을 더 들여쓴다	
	함수나 클래스는 2개의 공백 라인을 추가하여 구분한다. 메서드는 한 개의 공백 라인으로 구분한다	
	import는 (여러 모듈을 콤마로 연결하지 말고) 한 라인에 하나의 모듈을 import한다	No: import os, sys Yes: import os import sys
	컬렉션 인덱스나 함수 호출, 함수 파라미터 등에서 불필요한 공백을 넣지 않는다	No: spam(ham[1], { eggs: 2 }) bar = (0,) spam (1) Yes: spam(ham[1], {eggs: 2}) bar = (0,) spam(1)
	변수 할당시 할당자 앞뒤로 하나의 공백만 넣는다	No: i=i+1 Yes: i = i + 1
명명규칙	함수, 변수, Attribute는 소문자로 단어 간은 밑줄(_)을 사용하여 연결한다	예: total_numbers
	클래스는 단어 첫 문자마다 대문자를 써서 연결하는 CapWords 포맷으로 명명한다	예: CoreClass
	모듈명은 짧게 소문자로 사용하며 밑줄을 쓸 수 있다. 패키지명 역시 짧게 소문자를 사용하지만 밑줄은 사용하지 않는다.	예: serial_reader
	모듈 상수는 모두 대문자를 사용하고 단어마다 밑줄로 연결하는	예: MAX_COUNT = 100

	ALL_CAPS 포맷으로 명명한다	
	클래스의 public attribute는 밑줄로 시작하지 말아야 한다	예: name
	클래스의 protected instance attribute는 하나의 밑줄로 시작한다	예: _initialized
	클래스의 private instance attribute는 2개의 밑줄로 시작한다	예: __private_var
	인스턴스 메서드는 (객체 자신을 가리키기 위해) self 를 사용한다	예: def copy(self, other):
	클래스 메서드는 (클래스 자신을 가리키기 위해) cls 를 사용한다	예: def clone(cls, other):
문장과 표현식	if, for, while 블럭 문장을 한 라인으로 작성하지 말 것. 여러 라인에 걸쳐 사용하는 것이 더 명료함	No: if a <0: a = 0 Yes: if a <0: a = 0
	a는 b가 아니다를 표현할 때 a is not b 를 사용한다. not a is b 를 사용하지 말 것	No: if not a is b Yes: if a is not b
	값이 비어있는지 아닌지를 검사하기 위해 길이를 체크하는 방식을 사용하지 말 것. 대신 if mylist 와 같이 표현함	No: if len(mylist) == 0 Yes: if not mylist No: if len(mylist) >0 Yes: if mylist
	import 문은 항상 파일의 상단에 위치하며, 표준 라이브러리 모듈, 3rd Party 모듈, 그리고 자신의 모듈 순으로 import 한다	예: import os import numpy import mypkg
	모듈 import시 절대 경로를 사용할 것을 권장한다. 예를 들어, sibling 모듈이 현재 모듈과 같은 폴더에 있더라도 패키지명부터 절대 경로를 사용함. 단, 복잡한 패키지 경로를 갖는 경우 상대경로(.)를 사용할 수 있다.	No: import sibling Yes: import mypkg.sibling from mypkg import sibling from . import sibling # 상대경로 from .sibling import example