첫번째로 pylint는 파이썬에서 버그와 스타일 문제를 찾기 위한 도구입니다. 코드의 오타 또는 변수 할당 전에 변수 사용한 것과 같은 문제를 잡아냅니다.  
첫번째 사진은 Pylint에 내장되어있는 모든 경고 목록을 불러오는 코드입니다. 두번째 사진은 뒤에 특정 경고문의 코드를 붙임으로써 원하는 특정메시지를 불러오는 코드입니다.  
위 사진과 같이 코드를 입력하고 파일에 대해 pylint를 검사하면 작성한 python 코드에서 문제가 되는 부분을 메시지로 출력합니다.

Import는. 개별 클래스나 함수가 아닌 한 모듈에서 다른 모듈로 코드를 공유합니다.  
x가 모듈이고 y가 모듈 x에 정의된 객체라면 …  
import를 사용할 때 동일한 모듈에 있다고 모듈을 생략하고 적으면 안됩니다.

Package는 모듈의 전체 이름을 사용해서 갖고 옵니다. Package는 모듈의 이름이 충돌되거나 import가 잘못되는 문제를 막을 수 있습니다. 하지만 패키지 계층 구제를 복제해야합니다.

예외처리는 오류 또는 예외 조건을 처리하기 위해서 코드블록의 정상적인 제어 흐름을 벗어나는 것을 말합니다. 이로 인해 제어 흐름의 복잡성을 낮출 수 있고 특정 조건이 발생했을 때 여러 프레임을 통과할 수 있지만 제어 흐름 혼동시킬 수 있는 단점을 갖고 있습니다.  
예외를 사용하려면 먼저 함수 형식으로 사용해야합니다. 또한 내장된 예외 클래스를 사용해야합니다. except문을 사용할 때 try/except: 문을 사용하는 걸 권하지만 그 안의 코드를 최대한 적게 사용하는 걸 권합니다. 만약 try문의 예외를 확인하고 싶다면, finally를 사용하세요. As는 예외를 캡쳐합니다.

전역변수는 모듈 또는 클래스 속성에서 선언된 변수입니다. 모듈의 처음 부분에 수행되기 때문에 모듈의 동작 변경할 수 있습니다. 전역변수는 변수지만 모듈 레벨의 상수가 허용됩니다.

중첩된 지역함수나 클래스는 지역변수를 끝낼 때 사용합니다. 먼저 클래스는 메서드, 함수, 클래스 내에서 정의할 수 있습니다. 함수는 메서드, 함수 내에서 정의할 수 있습니다. 단점으로는 중첩된 클래스나 지역 클래스의 인스턴스는 선택할 수 없습니다. 중첩이 되면 외부 함수를 더 길게 만듭니다.  
따라서 로컬값을 초과하는 경우, 중첩을 피하세요. 또 클래스나 함수를 중첩할 때 모듈 앞에 언더바를 쓰세요.

컴프리헨션과 제너레이터는 리스트, 딕셔너리, 쎗과 같은 컨테이너 타입과 이터레이터를 사용하기 때문에 간결하고 효율적입니다. 따라서 컴프리헨션은 명확하고 단순합니다. 제너레이터는 리스트 작성을 안하기 때문에 효율적입니다. 하지만 코드가 복잡하면 읽기 힘들어집니다.  
따라서 아래 두 식처럼 한 줄에 모든 식을 작성하는 것을 추천합니다.

컨테이너 타입은 초기 이터레이터와 테스트 연산자를 정의합니다. 추가 메서드를 호출하지 않아 단순하고 효율적이고 모든 타입에서 지원하기 때문에 포괄적입니다. 하지만 메서드 이름만으로 객체 타입을 판별할 수 없다는 단점이 있습니다. 아래 사진처럼 리스트로 반환되는 방법보다 아래 사진처럼 반환되는 걸 선호하세요. 되도록이면 왼쪽 사진처럼 표현하고 오른쪽처럼 사용하지 마십시오.

제너레이터 함수가 실행되면 이터레이터, yeild가 반환된다. Yield 후에 다음 값이 필요할 때까지 함수가 중지된다. 제너레이터를 사용하면 코드가 단순해지고 적은 메모리를 차지한다는 장점이 있습니다. 제너레이터 함수에서 반환을 사용할 때는 리턴보다 일드를 사용하세요.

람다 함수는 함수 내에서 익명함수를 정의한다. 편리하지만 지역함수보다 읽기 능력이나 디버깅 능력이 떨어진다. 또한 스택 추적의 이해도도 떨어진다. 표현력 또한 저하된다. 코드가 한줄일 때는 람다함수를 사용하는 것을 권장하지만 코드가 60줄처럼 길어질 때는 중첩함수의 사용을 권장한다. 연산할 때는 람다함수보다 연산자 모듈의 함수를 사용하는 것이 더 낫다.

3항 연산자라고도 불리는 조건식은 if문보다 더 짧은 구문을 제공한다. 따라서 간단한 식에 주로 사용된다. 만약 식이 길어지면 if문 전체를 사용한다. true-expression, if-expression, else-expression을 포함한 모든건 한줄에 있다.  
첫 사진처럼 식이 간단할 때는 조건식을 사용하지만 두번째 사진처럼 너무 길어질 경우에는 if-else문 전체를 적는다.

초기 인수값은 함수 파라미터에 변수값을 직접 지정하는 것을 말한다. 예외가 드물게 발생하는데 함수를 정의하지 않아도 수행 가능하다. 파이썬은 오버로드를 지원하지 않는데 초기 인수값이 이를 도와준다. 하지만 모듈을 불러올 때 한번만 초기 인수값을 측정하므로 리스트나 딕셔너리처럼 변경이 가능한 객체에서는 문제가 된다.  
왼쪽 사진처럼 초기 값을 지정할 수 있다. 오른쪽 사진처럼 time함수는 변경이 가능하기 때문에 저렇게 선언할 수 없다.

단순한 접근자 또는 세터 메서드의 데이터에 접근하거나 데이터를 설정할 때 사용되는게 속성이다. 속성은 Get/set 메서드를 제거하기 때문에 가독성이 향상되고 파이썬의 특성인 클래스의 인터페이스를 유지한다. 하지만 object를 파이썬 2에서 상속해야되고 연산자 오버로딩 같은 부작용을 숨겨야한다. 또한 서브 클래스에 혼란을 줄 수 있다.  
속성이 재정의가 되지 않은 경우, 상속은 명백하지 않기 때문에 서브 클래스에서 재정의된 메서드가 속성에 호출되도록 접근 메서드를 간접적으로 호출해야한다. 따라서 어세서나 세터메서드의 데이터 접근하거나 지정하기 위해 @프로퍼티라는 코드를 사용합니다.  
다음 사진을 보면 @property를 통해 값을 가져옵니다.

0, none, 괄호 안의 비어잇는 것들은 fale로 처리할 수 있습니다. 단, 문자열 0은 true입니다. 트루 폴스를 사용하면 오류는 줄고 속도가 높아질 수 있습니다.  
아래 코드를 보면 불린을 확인할때는 연산자 = 을 사용해서 거짓을 판별하기보단 not을 사용해야합니다. 다른 값을 비교할 때는 None을 사용하세요. 문자열, 리스트, 튜플 같은 시퀀스의 공백을 표현할 때는 len 함수를 써서 길이를 비교하는 방법을 이용하세요. 정수값과 정수 0을 비교할 수 있습니다.