The Art of Readable Code

- 이름에 정보 담기 (네이밍을 잘하자!!)
  a. 변수,함수,혹은 클래스 등의 이름을 결정할 때는 항상 같은 원리가 적용된다.
  b. 이름에 정보를 담아내라!
- 특정한 단어 고르기
  a. 매우 구체적인 단어를 선택하여 '무의미한' 단어를 피해라
  b. ex) get(), getSize(), stop()
  c. 위 예시들은 좋지 않은 예이다. 무엇을 얻어야하는지 무엇을 멈춰야하는지 모른다.
- 보편적인 이름 피하기
  a. ex) tmp, retval
  b. retval의 경우 반환되는 값이라는 것은 알 수 있지만 무엇에 관한것인지 알 수가 없다.
- 루프 반복자
  a. i,j,k,iter,it 같은 이름은 보편적이지만 혼동을 초래할 것이다.
- 추상적인 이름보다 구체적인 이름을 선호하라
  a. ex) delay -> delay_secs
- 이름이 얼마나 길어야 하는가?
  a. 좁은 범위(scope)에서는 짧은 이름도 괜찮다.
  b. 큰 범위에서는 의미를 분명하게 만들기 위해 정보를 충분히 표현해야한다.
- 미학
  a. 좋은 소스는 보는 사람의 눈을 편하게 해야한다. -> 가독성
  b. 코드를 읽는 사람이 이미 친숙한 일관성있는 레이아웃을 사용하라 -> 실루엣을 동일해 보이도록 만들어라 -> 줄 바꿈, 정렬, 들여쓰기
  c. 서로 연관된 코드는 하나의 블록으로 묶어라 -> 코드를 문단으로 쪼개라
  d. 코드를 훑어보는데 걸리는 시간이 적을수록, 사람들은 코드를 더 쉽게 사용할 수 있다.
  e. 이러한 미학은 장점을 가져온다.
     - 중복된 코드를 없애서 코드를 더 간결하게 한다.
     - 테스트의 추가가 훨씬 쉬워졌다.
     - 코드의 구조자체를 개선시킨다.
- 주석
  - 주석을 읽는 것은 실제로 코드를 읽는 시간을 갉아 먹고, 화면을 차지한다. -> 꼭 필요한 경우에 달 수 있도록 하자
    a. 코드에서 빠르게 유추할 수 있는 내용은 주석으로 달지말라
    b. 나쁜 이름에 주석을 달지마라 -> 차라리 대신 이름을 고쳐라
    c. 생각을 기록하라
       - 좋은 주석은 단순히 자신의 생각을 기록하는 것 만으로도 탄생할 수 있다. 즉, 코딩할때 생각했던 중요한 생각을 기록하면 된다.
       - 코드의 의도를 명시하라
    d. 코드에 있는 결함을 설명하라

        - 코드는 지속적으로 진화하며, 그러는 과정에서 버그를 갖게 될 수 밖에 없다. 이러한 결함을 설명하는 것을 부끄러워할 필요는 없다.
    e. 코드(주석)을 읽는 사람의 입장이 되어라
        - 코드를 처음 읽는 외부인의 입장이 되어라
    f. 주석은 명확하게, 최대한 구체적이고 자세하게 작성해야 한다.반면 주석은 화면에서 추가적인 면적을 차지하고 읽는데 시간을 요구하므로 간결해야한다.
    g. 구체적인 용법을 설명해주는 입/출력 예를 사용하라
- 읽기 쉽게 흐름제어 만들기
    - 삼항연산자보다는 if/else를 이용하는게 좋다.
- 코딩테스트의 합격/탈락을 결정하는 원리
    - 중첩된 if문을 마구잡이로 사용하는 경우
- 한번에 하나씩
    - 함수는 오직 한가지 작업만 수행해야 한다.
    - 커다란 함수를 여러 작은 함수로 나누는 것은 좋다. 하지만 커다란 함수 안에 있는 코드를 재조직하여 그 안에 여러 개의 독자적인 논리적 영역으로 나눌 수 있다.
- 변수와 가독성
    - 변수의 수가 많을수록 기억하고 다루기 어려워진다. -> 중간결과, 임시변수를 꼭 필요한 경우에만 사용하도록 한다.
    - 변수의 범위를 좁혀라 -> 전역변수를 피하라: 이름중복, 의도하지 않은 변수의 변경, 실수
    - 변수값이 달라지는 곳이 많을수록 현재값을 추측하기 더 어려워진다. -> 변수값이 가능한 한번만 할당되도록 하라.
    - 일반적인 목적의 코드를 프로젝트의 특정(목적)코드에서 분리하라
- 상관없는 하위문제 추출하기
    - 커다란 문제를 작은 문제들로 쪼개어 분리, 해결한다. (divide&conquer)
- 생각을 코드로 만들기
    - 결국! 소스코드도 “쉬운 말”로 작성해야 합니다 -> 코드를 더 명확하게 만들도록 노력해야합니다.
    - 라이브러리가 제공하는 기능을 잘 활용한다.
    - 논리를 명확하게 설명하라
        - ex) if문 사용시 간결한 논리로 작성
- 코드의 분량 줄이기
    - 처음부터 오버스팩으로 개발할 필요는 없다. 필요한 기능의 구현에 집중하라
    - 코드베이스를 작게 유지하기
        - 최선의 방법은 프로젝트가 성장하더라도 ‘코드베이스를 최대한 작고 가볍게 유지’하는 것이다.
            - 일반적인 유틸리티를 많이 생성하여 중복된 코드를 제거한다.
            - 사용하지 않는 코드 혹은 필요 없는 기능을 제거하라
            - 프로젝트가 서로 분철된 하위 프로젝트로 구성되게 하라
            - 코드베이스의 무게를 항상 의식하여 가볍고 날렵하게 유지시켜라