2022년 전기 졸업과제 중간보고서

과제물 파이썬 코드의 상대적 품질 평가 시스템 개발

Developing Quality Evalution System for Assignment Python Codes

분	과	B	
팀	명	CopycaTcher	
지 도 교 수			조 환 규
팀	장	201724475	박 인 오
팀	원	201724528	이 범 수
팀	원	201741161	윤 석 현

목차

1. 연구 동기 및 사전 조사

1. 1. 연구 동기

학생들은 프로그래밍 과제 수행 중 코드의 크기를 줄이기 위하여 변수명 또는 클래스명을 'a'와 같이 억지로 짧게 작성하고, 문제를 단순하게 해결하기 위해 조건문과 중첩되는 반복문을 남용하는 방법을 사용한다. 이 방식을 '우아하지 않다'고 한다. 우아하지 않게 작성된 코드는 가독성이 좋지 않게 된다. 가독성이 좋지 않은 코드를 작성하는 습관을 방치 할 경우 차후에 협업을 하거나 유지보수 작업을 수행 할 때, 우아한 코드보다 많은 소통이 필요하고 이해하는 데 시간을 더 크게 소모하게 된다.

위 문단에서 언급한 문제를 판단하는 방법으로, 사람이 직접 보고 판단하는 방법과 기존에 개발된 도구를 활용하는 방법이 있다. 하지만 사람이 직접 보고 판단하는 방법은 인력과 시간이 소모된다.

기존에 개발된 도구들은 단일 소스코드에 대한 절대적인 코드 분석만을 실시한다. 특정 목적을 위해 작성된 여러 코드들의 상대적인 점수화 방법은 존재하지 않았다. 때문에 과제와 같이 특정 목표를 위하여 작성된 코드들에 대해서는 평가하기 힘들다는 문제점이 존재한다. 본 과제에서는 이를 해결하는 도구를 만들고자 한다.

개발 될 도구는 기존에 존재하는 도구들과 달리, 특정한 목적을 위해 작성된 여러 코드들에 대한 분석을 실시하고 이들을 상대적으로 평가한다. 단일 목적을 대상으로 하기에, 목적이 다른 코드를 동시에 입력 코드군으로 사용하지 않는다고 가정한다. 상대적으로 평가하기에 수치들은 정규화된다. 그 결과에 따라 우아한 코드와 일반적인 코드 우아하지 않은 코드로 분류하고, 사용자가 분류 결과를 납득할 수 있어야 한다.

1. 2. 관련 연구 및 사용되는 모듈

- Recommending Refactoring Solutions Based on Traceability and Code Metrics[1]

소스 코드 디자인과 추적 가능성을 개선하는 대체 리팩토링 방법을 선택한다. 추적 가능성과 소스 코드 디자인의 품질을 정량화하기 위해 엔트로피 기반과 고전적인 커플링, 응집 metric을 활용한다(To this end, we select among alternative refactoring solutions according to how they improve the traceability as well as source code design. To quantify the quality of traceability and source code design we leverage the use of entropy-based and traditional coupling and cohesion metrics respectively).

Code Metric을 분석한 후 Code Refactoring에 대한 추천 Solution을 제공함으로써 우아한 코드에 좀 더 다가가는 것으로 보이지만 이러한 Code Metric은 Refactoring

가이드라인에 기준한 절대적 분석이며, 협업을 통한 소프트웨어 개발과 같은 상황에서는 옳은 방법이지만, Coding Test와 같이 다수의 사람이 특정한 한 가지 목표를 보고 소스코드들을 작성하는 상황에서는 이러한 분석을 통해서 우아함을 측정하는 것은 옳다고 보기 어렵다고 판단했다.

즉, 상황에 따라서는 절대평가가 아닌 상대평가를 통해 코드의 우아함을 측정 할 필요가 있다는 것이다.

- flake8[2]

PEP8 스타일 가이드 위반을 탐지하는 모듈이다. 설치된 Python 버전에 맞추어 파싱한다. 옵션에 따라 특정 스타일 가이드를 무시하거나, 위반한 가이드 그룹만 선택하여 출력할 수 있다.

- Radon[3]

단일 코드의 metric을 계산하는 모듈이다. 해당하는 metric들로는 raw metric과 cyclomatic complexity, Halstead metrics, maintainability index가 있다.

이름	대상언어	유/무료	개발 목적	주요 방법	측정값	개발자
flake8	Python	무료	PEP8 스타일 가이드 위반 탐지	Parse Tree	양의 정수	PyCQA
Radon	Python	무료	단일 소스코드 metric 계산	Abstract Syntax Tree	[0-100] 실수	rubik

표 1. 관련 연구 목록

- PLY[4]

Python Lex-Yacc의 약자이며, 순수하게 Python으로 작성된 구문 분석 도구이다. C 언어에 존재하는 Lex와 Yacc을 Python으로 이식한 것이며, 원본과 동일하게 LALR parsing 기술을 사용하고 다양한 디버깅 및 오류탐지 도구가 존재한다.

기존 Python에서 제공하는 Tokenizer의 경우 토큰의 종류가 적기 때문에 더 정확한 구문 분석을 위해 직접 Token의 종류를 설정 할 수 있는 PLY를 사용하게 되었다.

- React[5]

HTML을 컴포넌트 단위로 나눠 작성할 수 있어서 재사용성이 뛰어나고 가상의 DOM을 조작하기 때문에 DOM을 직접 조작하는 jQuery에 비해 빠르다.

- React-table[6]
유사도를 표로 나타내는데 사용하는 라이브러리이다. 정렬과 검색기능을 지원하고 tr, td 태그를 직접 작성해야 하는 작업을 mapping을 활용해 편리하다.

- Chart.js[7]
우아함의 정도를 그래프로 나타내는데 사용되는 라이브러리이다. 다양한 형태의 차트를 지원한다. 본 과제에서는 scatterChart를 사용한다.

- Express[8]
Node.js의 웹 프레임워크로 서버를 만드는데 Javascript로 통일하는 것이 좋을 것이라고 생각하여 Node.js의 표준 웹 서버 프레임워크로 불리는 Express를 사용하게 되었다.

1. 3. 개발하는 도구의 목표

개발하는 도구는 입력받은 코드들을 분석하여 elegant와 plain, ugly 코드로 분류하고, 사용자가 분류된 결과를 알기 쉽도록 하는 것이 목표이다. 입력받는 코드들은 모두 하나의 특정 목표를 위해 작성된 코드라고 가정하며, 우아함의 등급은 절대적 점수가 아닌 상대적인 점수로 부여한다. 도구의 실행 결과를 사용자가 납득할 수 있도록 하기 위해, 실행하기 전에 육안으로 확인하여 elegant와 plain, ugly로 분류하여 실행한다. 그 결과에 따라 각 판단 기준에 대한 가중치를 조정한다.

분류 기준은 코드 블록의 깊이와 반복문의 깊이, 반복문의 수, 함수당 호출 수를 고려한다. 코드 블록의 깊이와 내부 반복문의 깊이, 반복문의 수에 따라 가중치를 다르게 하여 감점형으로 계산한다. 실행 결과에 따라 가중치를 조정한다. 수치들은 과제 목표와 같이, 특정 목적에 따라 수치들의 추세가 바뀌므로, 정규화시켜 분포를 나타낸다. 또한 코드 metric 도구를 이용하여 계산된 수치도 고려한다. 기존 도구에 의해 계산된 수치들은 단일 코드를 분석하므로, 그 결과를 정규화한다.

PLY는 build를 수행할 때 코드 내부에서 위에 존재하는 함수에 선언된 토큰, 아래에 존재하는 함수에 선언된 토큰, 변수로 선언된 토큰 순서로 우선순위가 부여된다. 우선순위로는 복소수와 같이 복잡한 식을 가진 토큰들이 높은 우선순위를 갖도록 규칙을 정의하였다. 그 후 test, test_blank 함수를 실행하면 위의 우선순위에 따라 토큰이 분석된다.

2. 요구조건 및 제약 사항 분석에 대한 수정사항

2. 1. 기존 요구조건 및 수정사항

2. 1. 1. Python 소스코드의 Tokenize

Python 소스코드의 유사도와 우아함을 확인하기 위해서는 우선적으로 해당 소스코드의 구조를 파악할 수 있는 Tokenize 과정이 필요하다. 2022년 7월 중순 기준으로 Python 버전 3.10.5에서는 Python의 기본 Library로 tokenize.py가 존재하지만, Token의 종류가 적다. 그렇기 때문에 세부적인 Tokenize를 위해 Token의 종류를 개발자가 정할 수 있는 Lex를 이용하여 Tokenize를 진행하려 하였다. 그러나, 소스코드 유사도 검사와 우아함 측정이 모두 Python을 통해 작성되고, 기존 Lex가 더 이상 사후지원되지 않아 팀원과의 연계를 위해 PLY(Python Lex-Yacc)를 이용하는 것으로 요구조건을 일부 변경하였다.

기존 요구조건	Python 소스코드를 입력받아 해당 소스코드를 Tokenize한다. - Lex를 이용하여 해당 작업을 수행 한다. - 입력받는 소스코드는 .py 파일의 형태로 입력받는다. - Tokenize 이후 출력은 결과 값을 포함한 .txt 파일로 출력한다.
수정 요구조건	- PLY를 이용하여 해당 작업을 수행한다. - 입력받는 소스코드는 Python String(Raw String)의 형태로 입력받는다. - Tokenize 이후 출력은 PLY의 Token Object로 출력한다. - 위의 모든 기능을 Class 내부에 구현하여 다른 작업에서도 자유롭게 사용할 수 있도록 만든다.

표 2. Python 소스코드의 Tokenize의 요구조건

2. 1. 2. 소스코드의 유사도 검사

해당 과제에서 소스코드의 유사도를 확인하기 위해서 입력받은 소스코드들을 Tokenize하여 Python 소스코드의 구조를 파악한 뒤에, 특정 알고리즘을 이용해 유사도를 수치화하는 방법을 거친다. 이 흐름은 바뀌지 않았으나, Python 소스코드 Tokenize 프로그램의 입출력이 바뀌었고, 서비스를 위한 웹사이트를 작업하는 팀원이 유사성 검사 후 출력을 JSON 형식을 요구함에 따라 입출력에 대한 요구사항을 수정하

였다.

기존 요구조건	Tokenize된 Python 소스코드들을 입력받아 각 소스코드간의 유사도를 수치화한다. - Lex를 통해 Tokenize된 소스코드는 .txt 파일들로 각각 입력받는다. - 구조적 유사성을 수치화 할 수 있는 알고리즘을 이용해 유사성의 정도를 수치화한다. - 수치화된 정보를 웹사이트에 보낸다.
수정 요구조건	- PLY를 통해 구현된 Tokenizer를 소스코드 유사성 확인 프로그램에 Import 한다. - Import한 Tokenizer를 이용해 입력받은 Python 소스코드들을 Tokenize한다. - Tokenizer의 출력 값인 Token Object에서 type값을 추출하여 Token type 이루어진 List를 생성한다. - Token type list로 전환된 각각의 소스코드들에 대해 Local Alignment Algorithm을 적용시켜 유사성의 정도를 수치화 시킨다. - 특정 소스코드에 대한 유사성의 수치화 값들을 JSON 형식으로 전환한 뒤에 결과 값을 출력하기 위한 웹 사이트로 송신한다.

표 3. 소스코드의 유사도 검사의 요구조건

2. 1. 3. 소스코드의 우아함의 척도

초기에 계획하였던 우아함의 척도로는 변수명이 사전에 등록된 단어인지와, 변수의 타입 변경 여부가 있었다. 변수명이 사전에 등록된 단어인지 파악하기 위해서는 토큰화를 거친 후 타입에 따라 판별해야 하는데, 토큰 출력 형식이 변경되어 변수명 검사 알고리즘은 일부 수정 및 보완할 필요가 있어 일정이 연기되었다. 또한, 변수의 타입 변경 여부는 동적 검사 시에만 가능하기에 계획에서 제거되었다. 기존에 계획되지 않은 다른 척도들이 추가되었다. 추가된 척도로는, 순환 복잡도(cyclomatic complexity, CC)와 Halstead volume, Halstead difficulty, Halstead effort, 유지관리 인덱스(maintainability index, MI)가 있다. 우아함 검사 후 결과를 JSON을 요구함에 따라 출력에 대한 요구사항을 수정하였다.

기존 요구조건	- 우아함의 척도로 변수명을 검사한다. - 변수의 타입이 변경되는지 확인한다.
수정 요구조건	- 우아함의 척도로 변수명을 검사한다. - 순환 복잡도(cyclomatic complexity)를 검사한다. - Halstead metric(Halstead volume, difficulty, effort)을 검사한다. - 유지 관리 인덱스(maintainability index)를 검사한다. - 우아함 검사의 결과를 JSON 형식으로 한다.

표 4. 소스코드의 우아함의 척도의 요구조건

2. 2. 기존 제약 사항 및 수정사항

2. 2. 1. 기존 제약 사항

제약 사항	① 소스코드의 유사도와 우아함을 측정할 테스트용 코드가 부족할 수 있다. ② 개발된 도구가 입력받는 코드는 정상 동작을 보장하지 않는다. ③ 짧은 길이의 코드 또는 구현이 간단한 코드인 경우 유사도가 높을 수 있다.
해결 방안	① 임의로 코드를 작성하여 사용한다. 임의로 작성하는 코드들은 고의로 일부를 copy하여 작성한 것과, 전혀 다르게 작성한 것들로 구성한다. ② Python Interpreter를 거쳐 코드의 정상 동작 여부를 판단한 후 도구를 사용한다. ③ 수치화된 유사도만을 출력하지 않고, 평균과 표준편차를 구하여 점수를 정규화한다.

표 5. 기존 제약 사항과 해결 방안

2. 2. 2. 수정/추가 제약 사항

제약 사항	① 적은 사람이 임의로 코드를 작성하여 만드는 소스코드들로는 테스트에 한계가 존재한다. 더 많은 사람이 특정 목표(ex. 과제)를 위해 작성한 소스코드들이 필요하다.

	② 유사도 검사와 우아함 검사의 결과 형식을 통일하여 웹 사이트에 표시하기 용이하도록 한다.
해결 방안	① 지도교수님께서 강의하시는 컴퓨터프로그래밍입문 수업의 과제로 제출된 학생들의 소스코드들을, 지도교수님의 허가 하에 유사도 검사와 우아함 측정 테스트에 사용한다. ② 유사도 검사와 우아함 검사의 결과 형식을 JSON으로 통일한다.

표 6. 수정 및 추가된 제약 사항과 해결 방안

3. 설계 상세화 및 변경 내역

3. 1. Python 소스코드의 Tokenize

3. 1. 1. reserved word 및 built in function Token 설정

Python 3.10.5 기준 reserved word, built in function을 reserved dictionary를 이용해 정규표현식을 이용하지 않고 Tokenize에 대한 정보를 설정한다. **(부록, 코드 1)**

3. 1. 2. [3. 1. 1.]에서 정의한 token들을 제외한 나머지 token들을 정의

앞서 reserved dictionary로 따로 정의한 token들을 제외한 정규표현식으로 구현할 token들을 tokens list에 정의한다. **(부록, 코드 2)**

3. 1. 3. 정규표현식을 이용해 token 선별 규칙을 구현

tokens list에 정의된 token들을 t_TOKEN 형태의 변수나 함수로 정의하고 내부에 정규표현식을 입력하여 해당 token을 선별하는 규칙을 구현한다. 특히, 함수로 구현된 token 규칙들은 token적용에 대한 우선순위를 구현할 수 있다. 예를들어 3.1이라는 실수는 INTEGER, DOT, INTEGER라는 token으로 표현할 수도 있지만, FLOATNUM 이라는 token으로 표현할 수도 있다. 이를 결정하는 것은 INTEGER token과 FLOATNUM token 사이의 우선순위를 어떻게 결정하냐에 따라 달라지며, 해당 프로그램에서는 FLOATNUM token의 우선순위를 더 높게 설정해 3.1이라는 실수가 입력될 경우 FLOATNUM이라는 token이 출력된다.
(부록, 코드 3)

3. 1. 4. Lex build와 tokenize 확인을 위한 test 함수구현

위의 모든 과정을 거친 후 Lex를 build하여 tokenize 준비를 끝낸다. 그리고 test와 test_blank 함수를 이용해 Python 소스코드를 String으로 입력받아 Token Object 형태로 출력한다. test_blank 함수의 경우 test 함수와 다르게 들여쓰기를 파악하기 위해 추가적인 작업을 수행한다. **(부록, 코드 4)**

3. 2. 소스코드의 유사성 검사

Python 소스코드 Tokenizer 개발을 우선하였기 때문에 유사도 검사에 대한 상세화를 연기했다. 일정에 맞추어 업무를 재분담하여 개발한다.

3. 3. 소스코드의 우아함의 척도
3. 3. 1. 반복문과 호출 탐색

코드 블록의 깊이와 반복문의 깊이, 반복문의 수, 함수당 호출 수를 각각 계산하고, 평균을 구한다. 코드 블록의 깊이와 내부 반복문의 깊이, 반복문의 수에 따라 가중치를 다르게 하여 감점형으로 계산한다. 실행 결과에 따라 가중치를 조정한다. 수치들은 과제 목표와 같이, 목적에 따라 수치들의 추세가 바뀌므로 정규화하여 분포를 나타낸다.

3. 3. 2. 변수명과 사전의 단어 비교

클래스를 인스턴스화할 때, 단어 사전으로 사용할 단어를 받고, PEP8 스타일 가이드 기준을 설정한다. PEP8 스타일 가이드 기준은 웹에서 사용자가 권장 기준을 선택하거나, 별도로 선택할 수 있도록 한다. **(부록, 코드 5)**

변수명 및 클래스명이 자주 쓰이는 단어인지 확인한다. 이를 위해 토큰화 과정이 요구된다. 토큰들 중에서 VARIABLE로 판단한 토큰에 대해 수행한다. 또한, ‘aaa’와 같이 의미없는 문자열의 반복을 탐지한다. 두 기준을 모두 검사하여 하나라도 만족하지 못하면 위반한 것으로 판단한다. **(부록, 코드 6)**

3. 3. 3. PEP8 스타일 가이드 위반

작성된 코드가 위반한 PEP8 스타일 가이드를 찾는다. 오픈소스 라이브러리 flake8을 이용한다. 현재 위반 사항 중 '파일 끝의 라인 없음'과 같은 일부 가이드는 무시(ignore 옵션)하고 있으나 가중치를 낮게 하거나 사용자가 선택할 수 있도록 한다. **(부록, 코드 7)**

3. 3. 4. 순환 복잡도(cyclomatic complexity)

순환 복잡도(이하 CC)는 코드 블록의 decision 횟수에 비례한다. CC에 영향을 주는 decision으로는 if와 elif, for, while, except, with, assert, 여러 comprehension, bool 연산자가 있다. else와 finally는 판단하지 않고 종속되기에 영향을 주지 않는다. CC의 수치가 낮을수록 코드의 안정성이 높다.

$$CC = decisions + 1$$

식 1. cyclomatic complexity

코드에서 계산된 CC의 리스트를 만든다. 그 합과 최대 CC, 평균 CC를 이용하여 복잡도를 계산한다. **(부록, 코드 8)**

3. 3. 5. Halstead metrics

Halstead metrics는 코드의 연산자와 피연산자의 수에 따라 복잡도를 결정한다. [3. 3. 4.]의 CC는 decision에만 의존한다는 점을 보완할 수 있기에 척도로 채택했다. η_1은 고유한 연산자의 수, η_2는 고유한 피연산자의 수, N_1은 총 연산자의 수, N_2는 총 피연산자의 수라고 할 때, Halstead volume과 difficulty, effort는 다음과 같이 계산된다. 각 수치들은 목표에 따라 다르므로 코드들에 대해 정규화할 필요가 있다. **(부록, 코드 9)**

$$volume\ V = (N_1 + N_2)\log_2(\eta_1 + \eta_2)$$

$$difficulty\,D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$$

$$effort\,E = D \times V = V/L$$

식 2. Halstead metrics

3. 3. 6. 유지 관리 인덱스(maintainability index)

유지 관리 인덱스(이하 MI)는 코드가 얼마나 유지 보수하기 용이한지를 나타내는 척도이다. MI는 코드의 소스 라인 수(SLOC) L과 CC, Halstead volume V, 주석의 라인 수 C에 의해 계산된다. MI는 실험적인 인덱스로, 제공하는 단체에 따라 계산식이 다르다. **(부록, 코드 10)**

$$MI = \max[0, 100 \frac{171 - 5.2\ln V - 0.23CC - 16.2\ln L + 50\sin\sqrt{2.4rad(C)}}{171}]$$

식 3. maintainability index

4. 갱신된 과제 추진 계획

4. 1. 유사도 검사

7월 3주 ~ 7월 4주 : Python 소스코드 Tokenizer 완성
7월 4주 ~ 8월 1주 : Python 소스코드 Tokenizer 테스팅, 디버깅
8월 1주 ~ 8월 3주 : 소스코드 유사도 검사 알고리즘 구현, 테스트, 디버깅
8월 3주 ~ : 시스템 통합 작업 및 테스트, 디버깅

4. 2. 우아함 검사

7월 1주 ~ : 예제 코드 작성
7월 3주 ~ 8월 1주 : 우아함 검사 알고리즘 보완, 테스트, 디버깅
7월 4주 ~ 8월 3주 : 소스코드 유사도 검사 알고리즘 구현, 테스트, 디버깅
8월 3주 ~ : 시스템 통합 작업 및 테스트, 디버깅

4. 3. 서버-프론트

~ 7월 3주 : 서버-프론트 구현
7월 3주 ~ 7월 4주 : 라이브러리 적용 및 구현, 페이지 출력
7월 4주 : 프론트 디자인 완료
8월 1주 ~ 8월 3주 : JSON 결과 가공 및 페이지 출력
8월 3주 ~ : 시스템 통합 작업 및 테스트, 디버깅

5. 구성원별 진척도

각자 담당한 작업의 진척도는 표 6과 같다.

담당자	내용	상태
박 인 오	우아함 검사 알고리즘 작성	진행 중
	- 변수명 작성 기준 위반 판단	완료
	- PEP8 스타일 가이드 위반 판단	완료
	- 소스코드 metric 계산	진행 중
	- 우아함 검사 결과 JSON화	미완료
이 범 수	Python 코드 토큰화 알고리즘 작성	완료
	- 토큰 정의 및 우선순위 결정	완료
	- 토큰화 함수 구현	완료
이 범 수 박 인 오	소스코드 유사도 검사 알고리즘 구현	진행 중
	- 유사도 검사 결과 JSON화	미완료
윤 석 현	서버 구현	진행 중
	- Node.js 서버 구현	완료
	- react-table, chart.js 라이브러리 적용	진행 중
	프론트 구현	진행 중
	- 메인, 유사도 검사, 우아함 검사 화면	완료
	- JSON 결과 적용	미완료

표 7. 진척도 현황

6. 보고 시점까지의 과제 수행 내용 및 중간 결과

- 소스코드의 Tokenize 및 소스코드 유사도 검사

그림 1은 변수선언과 조건문, print 함수가 쓰이는 Python 소스코드를 test_blank 함수에 input으로 집어넣은 결과 값을 보여주고 있다. 변수와 대입, 조건문과 비교연산자, print 함수와 같은 내부함수를 정상적으로 인식, tonkenize 하고 있다.

```
>>> m = PythonLexer()
>>> m.build()
>>> m.test_blank(r"""a = 7
b = 5

if(a > 5) :
        print(f"a가 b보다 큼")
else :
        print("b가 a보다 큼") """)
LexToken(VARIABLE,'a',1,0)
LexToken(ASSIGNMENT,'=',1,2)
LexToken(INTEGER,'7',1,4)
LexToken(NEWLINE,'\n',1,5)
LexToken(VARIABLE,'b',2,6)
LexToken(ASSIGNMENT,'=',2,8)
LexToken(INTEGER,'5',2,10)
LexToken(NEWLINE,'\n\n',2,11)
LexToken(IF,'if',4,13)
LexToken(LPAREN,'(',4,15)
LexToken(VARIABLE,'a',4,16)
LexToken(GREATER,'>',4,18)
LexToken(INTEGER,'5',4,20)
LexToken(RPAREN,')',4,21)
LexToken(COLON,':',4,23)
LexToken(NEWLINE,'\n',4,24)
LexToken(BLANK,'\t',5,25)
LexToken(PRINT,'print',5,26)
LexToken(LPAREN,'(',5,31)
LexToken(STRING,'f"a가 b보다 큼"',5,32)
LexToken(RPAREN,')',5,43)
LexToken(NEWLINE,'\n',5,44)
LexToken(ELSE,'else',6,45)
LexToken(COLON,':',6,50)
LexToken(NEWLINE,'\n',6,51)
LexToken(BLANK,'\t',7,52)
LexToken(PRINT,'print',7,53)
LexToken(LPAREN,'(',7,58)
LexToken(STRING,'"b가 a보다 큼"',7,59)
LexToken(RPAREN,')',7,69)
>>>
```

그림 1. Python 소스코드 Tokenize 결과

- 소스코드의 PEP8 스타일 가이드 위반

그림 2는 입력으로 주어진 파일들에 대해 PEP8 스타일 가이드 위반 결과를 보여주고 있다. 빈 리스트는 위반한 내용이 없다는 뜻이며, 사전 설정된 ignore 옵션에 해당하는 위반은 표시되지 않도록 했다.

```
PEP8 violations: [[], [], [], ["2 E712 comparison to False should be
 'if cond is False:' or 'if not cond:'"], []]
PEP8 violation counts: [0, 0, 0, 1, 0]
```

그림 2. PEP8 스타일 가이드 위반

- 소스코드의 cyclomatic complexity, Halstead metrics, maintainability index

그림 3은 입력으로 주어진 파일들에 대해 CC, halstead metrics, MI를 보여주고 있다.

```
code check:
depths: [[], [3, 1, 1, 2, 3, 3, 2, 2, 2, 3, 2, 2, 2, 2], [4], [3, 1, 1,
  1, 1, 3, 9], [1, 4, 2]]
max depths: [0, 3, 4, 9, 4]
halstead: [[0, 0, 0], [57.110323830864054, 2.0, 114.22064766172811],
 [15.509775004326936, 1.0, 15.509775004326936], [176.46653521143952,
 3.9473684210526314, 696.5784284662086], [4.754887502163469, 0.5,
 2.3774437510817346]]
Maintainability Index metrics: [100.0, 69.16084953634936, 82
 .7460580939387, 47.97947337442263, 93.49186663693075]
```

그림 3. CC, Halstead metrics, MI

- 프론트 화면 및 기능

그림 4는 메인 화면에 해당한다. 사용자는 파일들을 드래그 앤 드롭하여 업로드할 수 있다. 그림 5, 6은 각각 유사도 검사 결과와 우아함 검사 결과 화면에 해당한다. 서버로부터 계산된 결과를 페이지에 표시한다.

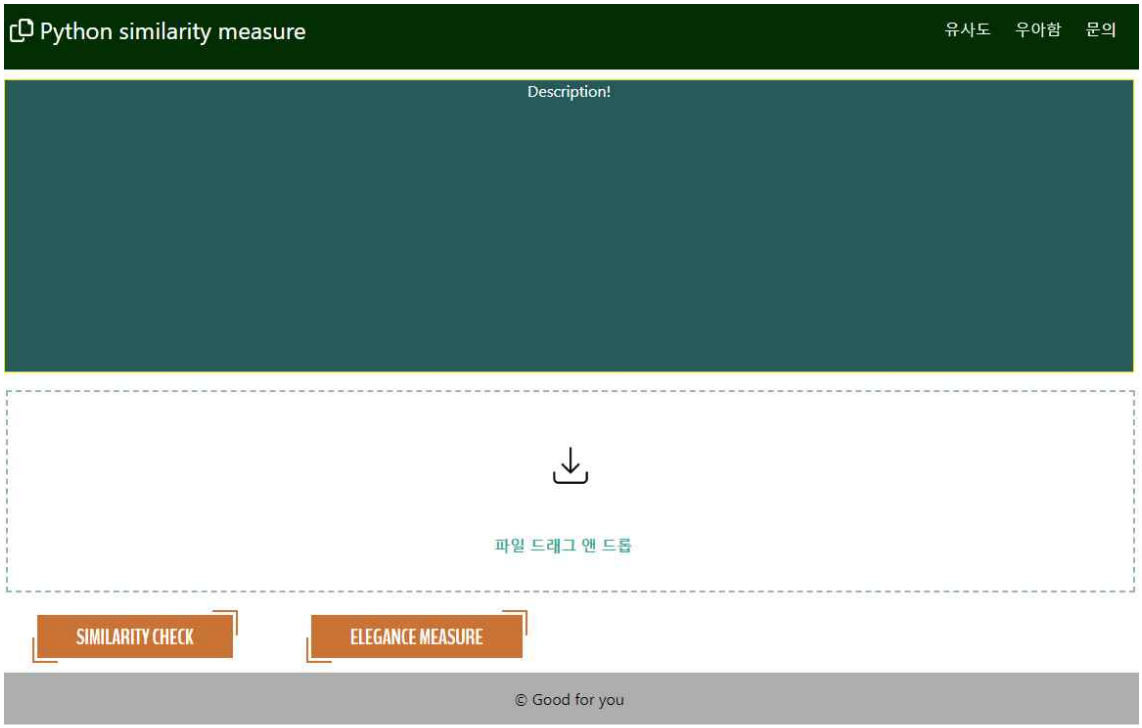

그림 4. 메인 화면

그림 5. 유사도 검사 결과 화면

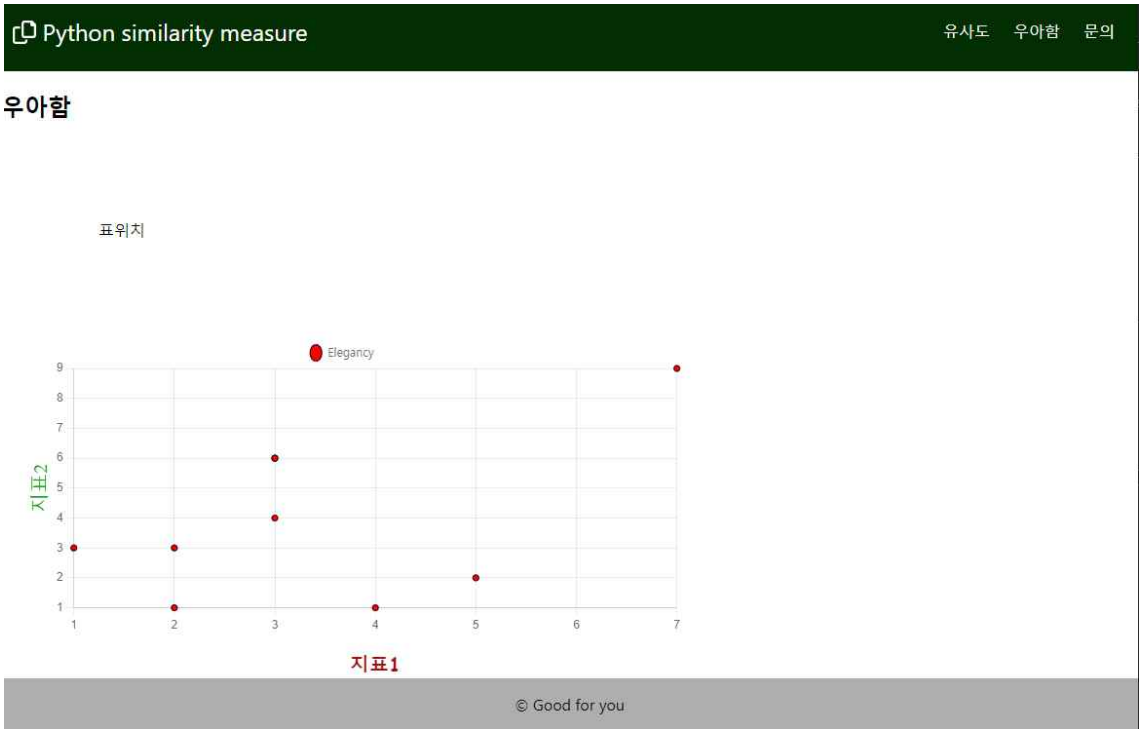

그림 6. 우아함 검사 결과 화면

7. 참고문헌

[1] A. S. Nyamawe, H. Liu, Z. Niu, W. Wang and N. Niu, "Recommending Refactoring Solutions Based on Traceability and Code Metrics," in IEEE Access, vol. 6, pp. 49460-49475, 2018, doi: 10.1109/ACCESS.2018.2868990.

[2] https://flake8.pycqa.org/en/latest/

[3] https://radon.readthedocs.io/en/latest/

[4] https://www.dabeaz.com/ply/

[5] https://ko.reactjs.org/docs/react-api.html

[6] https://tanstack.com/table/v8/docs/guide/introduction

[7] https://www.chartjs.org/docs/latest/

[8] https://expressjs.com/ko/4x/api.html

8. 부록

```
import ply.lex as lex

class PythonLexer(object) :
    reserved = {
        'False' : 'FALSE',      #reserved words
        'None' : 'NONE',
        'True' : 'TRUE',
        'and' : 'AND',
        'as' : 'AS',
        'assert' : 'ASSERT',
        'async' : 'ASYNC',
        'await' : 'AWAIT',
        'break' : 'BREAK',
        'class' : 'CLASS',
        'continue' : 'CONTINUE',
        'def' : 'DEF',
        'del' : 'DEL',
        'elif' : 'ELIF',
        'else' : 'ELSE',
        'except' : 'EXCEPT',
        'finally' : 'FINALLY',
        'for' : 'FOR',
        'from' : 'FROM',
        'global' : 'GLOBAL',
        'if' : 'IF',
        'import' : 'IMPORT',
        'in' : 'IN',
        'is' : 'IS',
        'lambda' : 'LAMBDA',
        'nonlocal' : 'NONLOCAL',
        'not' : 'NOT',
        'or' : 'OR',
        'pass' : 'PASS',
        'raise' : 'RAISE',
        'return' : 'RETURN',
        'try' : 'TRY',
        'while' : 'WHILE',
        'with' : 'WITH',
        'yield' : 'YIELD',
        'abs' : 'ABS',          #built in functions
```

코드 1. reserved dictionary

```
tokens = [
    'VARIABLE',
    'INTEGER',
    'FLOATNUM',
    'COMPLEXNUM',
    'STRING',
    'PLUS',
    'MINUS',
    'TIMES',
    'DIVIDE',
    'MODULUS',
    'FLOORDIVISION',
    'EXPONENT',
    'GREATER',
    'LESS',
    'GREATEREQUAL',
    'LESSEQUAL',
    'EQUAL',
    'NOTEQUAL',
    'LPAREN',
    'RPAREN',
    'LBRACE',
    'RBRACE',
    'LBRACKET',
    'RBRACKET',
    'COMMA',
    'DOT',
    'COLON',
    'ASSIGNMENT',
    'UNDERBAR',
    'TRIPLEDOT',
    'BLANK',
    'NEWLINE',
] + list(reserved.values())
```

코드 2. tokens list

```
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_MODULUS = r'%'
t_FLOORDIVISION = r'//'
t_EXPONENT = r'\*\*'
t_GREATER = r'>'
t_LESS = r'<'
t_GREATEREQUAL = r'>='
t_LESSEQUAL = r'<='
t_EQUAL = r'=='
t_NOTEQUAL = r'!='
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_LBRACKET = r'\['
t_RBRACKET = r'\]'
t_COMMA = r','
t_DOT = r'\.'
t_COLON = r':'
t_ASSIGNMENT = r'='
t_UNDERBAR = r'_'
t_TRIPLEDOT = r'\.{3}'
t_BLANK = r'[ \t]'

t_ignore_COMMENT = r'\#.*'

def t_COMPLEXNUM(self, t):
    r'([+-]?(0|[1-9]\d*)\.\d+([eE][+-]?(0|[1-9]\d*))?)?[+-]?(0|[1-9]\d*)\.\d
    return t

def t_FLOATNUM(self, t):
    r'[+-]?(0|[1-9]\d*)\.\d+([eE][+-]?(0|[1-9]\d*))?'
    return t

def t_INTEGER(self, t):
    r'[+-]?(0|[1-9]\d*)'
```

코드 3. t_TOKEN 형태의 변수와 함수들

```
    def build(self, **kwargs):
        self.lexer = lex.lex(module=self, **kwargs)

    def test(self, data):
        self.lexer.input(data)
        while(True):
            tok = self.lexer.token()
            if(not tok):
                break
            print(tok)

    def test_blank(self, data):
        self.lexer.input(data)

        newline_flag = False

        while(True):
            tok = self.lexer.token()
            if(not tok):
                break
            if(tok.type == 'NEWLINE'):
                newline_flag = True
            if(tok.type == 'BLANK' and newline_flag == False):
                continue
            if(tok.type != 'NEWLINE' and tok.type != 'BLANK' and newline_flag ==
                newline_flag = False
            print(tok)
```

코드 4. build, test, test_blank 함수

```
import os
import re
import constants
from nltk.downloader import download as nltk_download
from nltk.corpus import words as nltk_words
from flake8.api import legacy as flake8
from radon import complexity, raw, metrics


class Elegance:
    def __init__(self) -> None:
        """
        Initializes dictionary and the style guide.
        """
        nltk_download('words')
        word_list = nltk_words.words()
        self.word_set = set(word_list)
        self.style_guide = None
        self.reset_style_guide()

    def reset_style_guide(self) -> None:
        """
        Resets the style guide.
        """
        self.style_guide = flake8.get_style_guide(ignore=['E121', 'E126', 'E127'
                                                          'E225', 'E226', 'E231'
                                                          'E261', 'E265', 'E302'
                                                          'E402', 'E501', 'E741'
                                                          'W293', 'W391', 'W503'
                                                          'B007', 'B950'],
                                                  max_line_length=200)
```

코드 5. Elegance class

```
    def check_name_dictionary_word(self, word: str) -> bool:
        """
        Checks if a word is correct variable name.
        :param word: str: variable name to check
        :returns: bool: True if the word is a correct variable name
        """
        assert len(word) > 0, 'length of word must be greater than 0'
        if word in self.word_set:
            if not any(regex.match(word) for regex in constants.REGEX_VAR_NAME_E
                return False
        else:
            return True

    def check_sequential_word(self, word: str) -> bool:
        """
        Checks if a word has triple consecutive characters.
        :param word: str: variable name to check
        :returns: bool: True if the word has triple consecutive characters
        """
        return True if re.search(r'(.)\1\1', word) else False
```

코드 6. 단어 검사

```
    def __get_flake8_report(self, file_path: str = None) -> flake8.Report:
        """
        Checks the elegance of a file.
        :param file_path: str: file path
        :returns: flake8.api.legacy.Report
        """
        self.reset_style_guide()
        if file_path is None:
            file_path = os.getcwd()
        return self.style_guide.input_file(file_path)

    def get_PEP8_metrics(self, files: list[str]) -> list[list[str]]:
        """
        Returns the PEP8 violations of a list of code files.
        :param files:
        :returns: list[list[str]]
        """
        assert len(files) > 0, 'length of files must be greater than 0'
        reports = []
        for file in files:
            report = self.__get_flake8_report(file)
            reports.append(report.get_statistics(''))
        return reports
```

코드 7. PEP8 위반 검사

```
    def get_cyclomatic_metrics(self, codes: list[str]) -> list[list[int]]:
        """
        Returns the cyclomatic complexities of a list of codes.
        :param codes: strings of raw code
        :returns: list[list[int]]
        """
        depths = []
        for code in codes:
            elements = complexity.cc_visit(code)
            depths.append([element.complexity for element in elements])
        return depths
```

코드 8. 순환 복잡도

```
    def get_halstead_metrics(self, codes: list[str]) -> list[list[float]]:
        """
        Returns the Halstead metrics of a list of codes.

        :param codes: strings of raw code
        :return: list[list[voulume, difficulty, effort]]
        """

        halstead = []
        for code in codes:
            element = metrics.h_visit(code)
            halstead.append([element.total.volume, element.total.difficulty, ele
        return halstead
```

코드 9. Halstead metrics

```
    def get_mi_metrics(self, codes: list[str]) -> list[float]:
        """
        Returns the MI metrics of a list of codes.
        :param codes: strings of raw code
        :returns: list[float]
        """
        return [metrics.mi_visit(code, True) for code in codes]
```

코드 10. 유지 관리 인덱스