

국민대학교
소프트웨어 융합대학
소프트웨어 프로젝트 II

조원 : 20181588 김용호, 20181599 김진명
Python Code Checker
(Python Coding Style Check Program)

소프트웨어 상세 설계서

목차

1. 서론
 - 1.1 개요
 - 1.2 개발 목적
 - 1.3 구현 단계
2. 모듈의 구현 방식
3. 인터페이스 설계
4. 코드 구현과 알고리즘

1. 서론

1.1 개요

본 소프트웨어 상세 설계서는 국민대학교 소프트웨어 융합대학 소프트웨어 프로젝트II 의 Adeventure Design을 수행하기 위해 파이썬 코딩 스타일 가이드라인을 참조하여 개발할 Python Coding Style Checker 프로그램의 상세 설계서 입니다. 본 문서에서는 프로그램의 개발 목적과 구현 단계, 그리고 프로그램 모듈들의 구현 방식, 알고리즘, 구현 전략을 작성하였고 이 문서를 토대로 개발할 것 입니다.

1.2 개발 목적

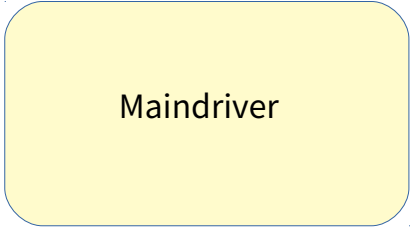
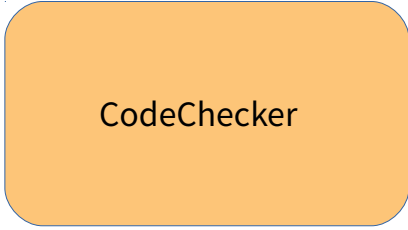
코드 규칙(Coding Convention)이란 같은 코드를 작성하고 유지 보수하는 개발자들 사이에 규정된, 코드가 지켜야 할 규약입니다. 예를 들어, 띄어 쓰기를 한다 거나 불필요한 괄호를 제거하는 등이 있습니다. 개인마다 코드를 작성하는 스타일은 다르지만, 코드 리뷰(Code Review)를 할 때 작성된 코드가 매우 길거나, 공백이 없어 읽기가 어렵다면 거부감을 느낄 수 있습니다. 이를 방지하고자, Python Coding Style Guideline에 작성되어있는 코드 규칙을 참고하여, 사용자의 코드를 코딩 규칙을 지켜서 자동으로 고쳐 주는 프로그램을 개발하고자 합니다. 이 프로그램을 사용하는 것은 소프트웨어의 품질을 향상, 보장하기 위한 코드 리뷰 활동에 가장 중요하고 기초적인 시작일 것 입니다.

1.3. 구현 단계



2. 모듈의 구현 방식

프로그램에서 주요한 역할을 하는 모듈 Maindriver와 Maindriver에서 보내는 파이썬 코드를 Python Coding Style Guideline을 지켜서 코드를 고친 후, Maindriver에 리턴하는 모듈 Codechecker 이 두 가지 모듈을 구현할 것 입니다.

	
GUI	Maindriver에서 코드를 받음
파일 읽기와 저장	받은 코드를 코딩 규칙에 따라수정
코드를 Codechecker모듈에 보냄	고친 코드를 Maindriver에 보냄

3. 인터페이스 설계

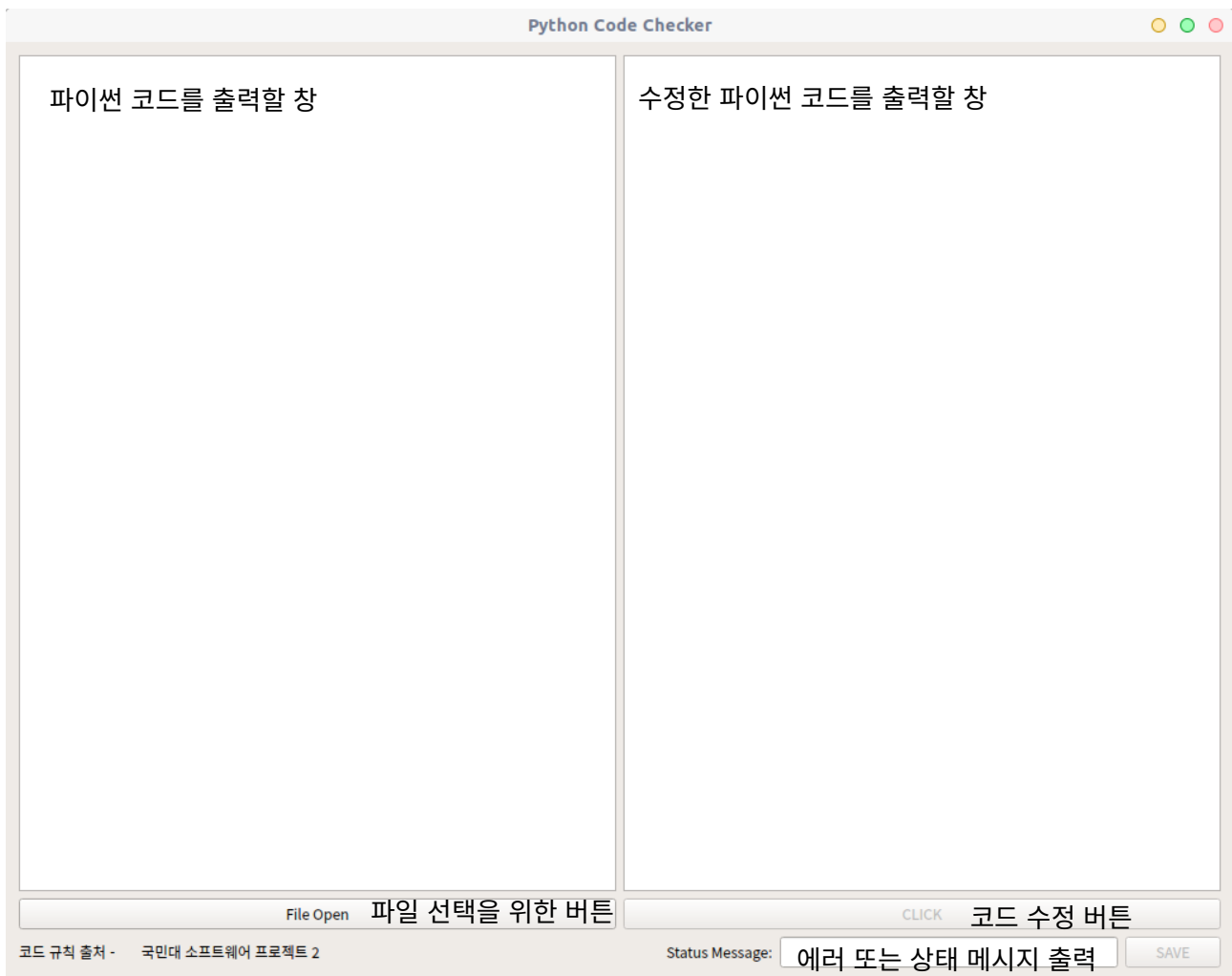


사진1 : Codechecker 인터페이스 샘플

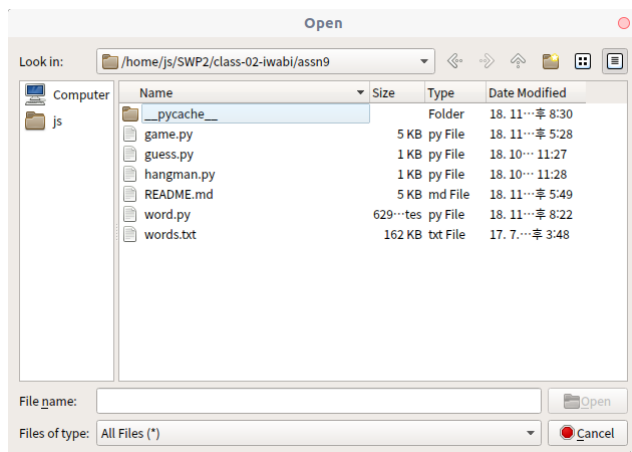


사진2: File Open 버튼 클릭 시 파일 선택 창



4. 코드 구현과 알고리즘

4-1 Maindriver.py

1. UI 구현

```
def initUI(self):
```

```
    # 1 line UI
    self.fileLineEdit = QTextEdit()
    self.fileLineEdit.setReadOnly(True)
    self.fileLineEdit.setFixedSize(500, 700)
    self.resultLineEdit = QTextEdit()
    self.resultLineEdit.setReadOnly(True)
    self.resultLineEdit.setFixedSize(500, 700)
    hbox = QHBoxLayout()
    hbox.addWidget(self.fileLineEdit)
    hbox.addWidget(self.resultLineEdit)

    # 2 line UI
    self.openbutton = QPushButton("File Open")
    self.clickbutton = QPushButton("CLICK")
    self.clickbutton.setEnabled(False)
    hbox1 = QHBoxLayout()
    hbox1.addWidget(self.openbutton)
    hbox1.addWidget(self.clickbutton)
```

```
    # 3 line UI
    self.taglabel = QLabel("코드 규칙 출처 - \t 국민대 소프트웨어 프로젝트 2")
    self.statuslabel = QLabel("Status Message:")
    self.saveButton = QPushButton("SAVE")

    self.saveButton.setEnabled(False)
    self.statusLineEdit = QLineEdit()
    self.statusLineEdit.setReadOnly(True)
    self.statusLineEdit.setAlignment(Qt.AlignCenter)
    hbox2 = QHBoxLayout()
    hbox2.addWidget(self.taglabel)
    hbox2.addStretch()
    hbox2.addWidget(self.statuslabel)
    hbox2.addWidget(self.statusLineEdit)
    hbox2.addWidget(self.saveButton)
    # 세로 배치
    vbox = QVBoxLayout()
    vbox.addLayout(hbox)
    vbox.addLayout(hbox1)
    vbox.addLayout(hbox2)
    self.setLayout(vbox)

    # 이벤트 핸들러
    self.openbutton.clicked.connect(self.openbuttonClicked)
    self.clickbutton.clicked.connect(self.clickbuttonClicked)
    self.saveButton.clicked.connect(self.saveButtonClicked)
```

- 첫 번째 라인에는 QTextEdit 2개를 가로 500픽셀, 세로 700픽셀로 읽기 모드로 설정합니다.
- 두 번째 라인에는 openbutton과 clickbutton을 생성하여 배치합니다.
- 세 번째 라인에는 tagLabel과 statusLabel, saveButton을 생성하고 배치합니다.
- openbutton, clickbutton, saveButton은 클릭 시 각각 openbuttonClicked, clickbuttonClicked, saveButtonClicked 함수를 호출하도록 하였습니다.

2. open버튼 클릭

```
def openbuttonClicked(self):
    self.resultLineEdit.clear() #파일 다시 불러오면 결과창 초기
    self.saveButton.setEnabled(False)
    self.string = []
    faddress = QFileDialog.getOpenFileName(self)
    self.fname = faddress[0].split('/')[-1]
    try:
        f = open(self.fname, 'r')
    except:
        self.taglabel.setText("Error = FileNotFoundError")
        self.statusLineEdit.setText("해당 파일이 같은 디렉토리에 있는지 확인해주세요.")
    else:
        data = f.read()
        self.taglabel.setText("코드 규칙 출처 - \t 국민대 소프트웨어 프로젝트 2")
        self.fileLineEdit.setText(data)
        self.statusLineEdit.setText('open = ' + faddress[0])
        self.clickbutton.setEnabled(True)
```

- open 버튼 클릭 시 파일을
하나 선택해서 filelineEdit

창에 불러오도록 하였습니다.

- 대신 파일을 불러올 때 파일은 이 파이썬 프로그램과 같은 디렉토리에 있어야 하기 때문에 try ~ except
예외처리를 해줍니다.

3.click

```
def clickbuttonClicked(self):
    f = open(self.fname, 'r')
    for line in f:
        for character in line:
            if ';' in line:
                line = self.check.semiclone(line)
            elif character in self.check.operator_Character:
                line = self.check.operator_check(line)
            elif character in self.check.special_Character:
                line = self.check.special_check(line)
            elif character in self.check.bracket_Character:
                line = self.check.bracket_check(line)

        if line.strip().find(self.check.import_Character[0]) == 0 or line.strip().find(
            self.check.import_Character[1]) == 0:
            line = self.check.import_check(line)
        elif line.strip().find(self.check.class_Character) == 0:
            line = self.check.class_check(line)
        elif line.strip().find(self.check.def_Character) == 0:
            line = self.check.def_check(line)
        elif line.strip().find(self.check.return_Character) == 0:
            line = line.replace('return(', 'return (')
            line = self.check.return_check(line)
        self.string.append(line)

    for i in range(len(self.string)):
        if '\n' not in self.string[i]:
            self.string[i] += '\n'
    f.close()
    self.saveButton.setEnabled(True)
    self.clickbutton.setEnabled(False)
    self.resultLineEdit.setText(''.join(self.string)) #string리스트를 문자열로 변환 후 표시
    self.statusLineEdit.setText("저장하려면 SAVE를 누르세요.")
```

- click 버튼을 클릭 시 파일을 읽고 코딩 규칙에 맞게 변환하여 resultlineEdit창에 불러오도록 합니다.

4.save

```
def saveButtonClicked(self):  
    self.saveButton.setEnabled(False)  
    self.statusLineEdit.setText("저장되었습니다.")  
    string = self.resultLineEdit.toPlainText() #resultLineEdit의 텍스트를 저장  
    self.writefile(string)  
    self.showfile()
```

- save 버튼 클릭 시 writefile()함수와 showfile()함수 호출

5. write

```
def writefile(self, string):  
    f = open(self.fname, 'w')  
    for i in string:  
        f.write(i)  
    f.close()
```

- writefile은 파일을 쓰기 모드로 설정하여 파일을 코딩 규칙에 맞는 코드로 재 저장합니다.

6. show

```
def showfile(self):  
    f = open(self.fname, 'r')  
    output = f.read()  
    self.resultLineEdit.setText(output)  
    f.close()
```

- showfile은 코딩 규칙에 맞게 변환한 파일을 다시 읽어 resultLineEdit에 출력합니다.

4-2 Codechecker.py

1. 세미콜론

```
def clickbuttonClicked(self):
    f = open(self.fname, 'r')
    for line in f:
        for character in line:
            if ';' in line:
                line = self.check.semiclone(line)

def semiclone(self, line):
    line = line.rstrip()
    if line[-1] == ';': # 세미콜론제거
        line = line[:-1]
    try:
        if line.find(';') != -1:
            if line[line.find(';')+1].isalpha() or line[line.find(';')+1] == ' ':
                if line[line.find(';')+1] == ' ':
                    line = line[:line.find(';')] + '\n' + line[line.find(';') + 2:]
                else:
                    line = line[:line.find(';')] + '\n' + line[line.find(';') + 1:]
        except:
            return line
    return line
```

- 행의 마지막을 세미콜론 (semicolon)으로 끝내거나, 두 문장을 한 행에 표현하기 위하여 세미콜론으로 병치하지 않도록 처리합니다.

2 연산자 띄어쓰기

```
def clickbuttonClicked(self):
    f = open(self.fname, 'r')
    for line in f:
        for character in line:
            if ';' in line:
                line = self.check.semiclone(line)
            elif character in self.check.operator_Character:
                line = self.check.operator_check(line)

def operator_check(self, line): #연산자
    string = ""
    for i in line:
        if i not in self.operator_Character:
            string += i
        else:
            string += '@'

    for i in line:
        if i in self.operator_Character:
            while line.find(i) != -1:
                index = line.find(i)
                string = string[:index].rstrip() + " " + i + " " + string[index+1:].lstrip()
                line = line[:index].rstrip() + " " + '@' + " " + line[index + 1:].lstrip()

    try:
        for i in range(len(string)): #a<=b<=c
            if string[i] in self.operator_Character:
                if string[i+2] in self.operator_Character:
                    string = string[:i+1] + string[i+2:]
        return string
    except:
        return string
```

- 이항 연산자의 앞/뒤에는 각각 하나 씩의 공백을 두도록 처리해 줍니다.

3. ‘:’, ‘,’ 띄어쓰기

```
def clickbuttonClicked(self):
    f = open(self.fname, 'r')
    for line in f:
        for character in line:
            if ';' in line:
                line = self.check.semiclone(line)
            elif character in self.check.operator_Character:
                line = self.check.operator_check(line)
            elif character in self.check.special_Character:
                line = self.check.special_check(line)

def special_check(self, line): #special2
    string = ""
    for i in line:
        if i not in self.special_Character:
            string += i
        else:
            string += '@'

    for i in line:
        if i in self.special_Character:
            while line.find(i) != -1:
                index = line.find(i)
                string = string[:index].rstrip() + i + " " + string[index + 1:].lstrip()
                line = line[:index].rstrip() + '@' + " " + line[index + 1:].lstrip()
    return string
```


- 콤마, 세미콜론, 콜론의 앞에는 공백을 두지 않고, 뒤에는 하나의 공백을 두도록 처리해 줍니다.

4. 괄호(리스트, 튜플, 집합) 띄어쓰기

```
def clickbuttonClicked(self):
    f = open(self.fname, 'r')
    for line in f:
        for character in line:
            if ';' in line:
                line = self.check.semiclone(line)
            elif character in self.check.operator_Character:
                line = self.check.operator_check(line)
            elif character in self.check.special_Character:
                line = self.check.special_check(line)
            elif character in self.check.bracket_Character:
                line = self.check.bracket_check(line)

def bracket_check(self, line): #special3
    string = ""
    for i in line:
        if i not in self.bracket_Character:
            string += i
        else:
            string += '@'

    for i in line:
        if i in self.bracket_Character:
            while line.find(i) != -1:
                index = line.find(i)
                string = string[:index].rstrip() + i + string[index+1:].lstrip()
                line = line[:index].rstrip() + '@' + line[index + 1:].lstrip()

    return self.operator_check(self.special_check(string))
```

- 괄호, 대괄호, 중괄호의 안쪽에 불 필요한 공백을 두지 않도록 처리합니다.

5. import 띄어쓰기

```
if line.strip().find(self.check.import_Character[0]) == 0 or line.strip().find(
    self.check.import_Character[1]) == 0:
    line = self.check.import_check(line)

def import_check(self, line): #special4
    line = self.special_check(line)
    tmp = line.split()
    string = ""
    string1 = ""
    if line[0] == '#': #틀 #import a, b [import , a, b] index = 0, for i in range(1): string = import ' '여쓰기 되있는 import
        return line
    if tmp[0] == self.import_Character[0]:
        index = tmp.index(self.import_Character[1])
        for i in range(index + 1):
            string = string + tmp[i] + ' '
        for i in range(index + 1, len(tmp)):
            string1 += string + tmp[i] + '\n'
            string1 = string1.replace(',', ', ')

    elif tmp[0] == self.import_Character[1]:
        string += tmp[0] + ' '
        if line.find('as') != -1:
            return line
        else:
            for i in range(1, len(tmp)):
                string1 += string + tmp[i] + '\n'
                string1 = string1.replace(',', ', ')

    return string1
```

- 서로 다른 모듈에 대한 import 문장은 개별 행을 차지하도록 처리합니다.

4-6 class

```
if line.strip().find(self.check.import_Character[0]) == 0 or line.strip().find(
    self.check.import_Character[1]) == 0:
    line = self.check.import_check(line)
elif line.strip().find(self.check.class_Character) == 0:
    line = self.check.class_check(line)

def class_check(self, line): #special5
    tmp = line.split()
    if '(' not in tmp[1] and ')' not in tmp[1]:
        index = line.find(':')
        string = line[:index] + '(object)' + line[index:]
    else:
        string = line

    return string
```

- 클래스가 다른 베이스 클래스를 상속하지 않는 경우, 명시적으로 object를 상속함으로 표현하도록 처리하였습니다.

4-7 def

```
if line.strip().find(self.check.import_Character[0]) == 0 or line.strip().find(
    self.check.import_Character[1]) == 0:
    line = self.check.import_check(line)
elif line.strip().find(self.check.class_Character) == 0:
    line = self.check.class_check(line)
elif line.strip().find(self.check.def_Character) == 0:
    line = self.check.def_check(line)

def def_check(self, line): #special6
    openindex = line.index('(')
    closeindex = line.index(')')
    string = line[openindex:closeindex + 1]
    if string != '':
        for i in string:
            if i in self.operator_Character:
                characterindex = string.index(i)
                string = string[:characterindex].rstrip() + i + string[characterindex + 1:].lstrip()
            else:
                continue
        line = line[:openindex] + string + line[closeindex + 1:]
    else:
        line = line

    return line
```

- 키워드 인자 또는 디폴트 인자 값을 표현할 때에는 = 의 앞과 뒤에 공백을 두지 않도록 처리 하였습니다.

4-8 return

```
if line.strip().find(self.check.import_Character[0]) == 0 or line.strip().find(
    self.check.import_Character[1]) == 0:
    line = self.check.import_check(line)
elif line.strip().find(self.check.class_Character) == 0:
    line = self.check.class_check(line)
elif line.strip().find(self.check.def_Character) == 0:
    line = self.check.def_check(line)
elif line.strip().find(self.check.return_Character) == 0:
    line = line.replace('return(', 'return (')
    line = self.check.return_check(line)

def return_check(self, line): #special7
    tmp = line.split()
    if tmp[1].find('(') == 0:
        firstindex = line.index('(')
        line = line[:firstindex] + line[firstindex + 1:-1]
    string = ""
    for i in line:
        if i not in self.operator_Character:
            string += i
        else:
            string += '@'

    for i in line:
        if i in self.operator_Character:
            while line.find(i) != -1:
                index = line.find(i)
                string = string[:index].rstrip() + i + string[index + 1:].lstrip()
                line = line[:index].rstrip() + '@' + line[index + 1:].lstrip()
    return string
```

- 불필요한 괄호를 가급적 사용하지 않도록 처리하였습니다.