Python 통합 개발 환경 구축

정유진*, 이태형**, 김의태***
한국외국어대학교 컴퓨터공학과
e-mail: chungyj@hufs.ac.kr
lth1722@hanmail.net
mulmy0729@hanmail.net

Python Development Tool

Yoojin Chung, Tae-Hyung Lee, Eui-Tae Kim
Dept of Computer Engineering, Hankuk University of Foreign Studies

요 약

Python은 통합 개발 환경이 구축되어 있지 않아 Python을 이용해 개발하는 경우 대부분 Vi 혹은 Emacs 같은 텍스트 편집기를 이용해 코딩을 하고 다시 터미널 혹은 콘솔로 나와 일일이 디버깅을 해야만 한다. 게다가 import 한 모듈의 API에 대해서 알고 싶으면 Python Interpreter에서 다시 봐야 하는 등 상당한 불편함이 있다. 이에 본 논문에서는 한 화면에서 편집, 컴파일, 수행결과 보기, 디버거 등을 수행할 수 있는 Python 통합 개발 환경을 구축하였다.

키워드 : Python, 통합 개발 환경

I. 서론

파이썬(Python)은 1991년 발표된 범용의 고급 프로그래밍 언어로, 플랫폼 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typed) 대화형 언어이다. 파이썬은 다양한 곳에서 사용될 수 있는데, 웹 어플피케이션(Web application)의 스크립트 언어(scripting language)로 사용되기도 하고 특히, 정보 보안 산업에서 널리 사용된다. 또한, 3D 애니메이션과 게임에서도 널리 사용된다.

하지만, Python은 통합 개발 환경이 구축되어 있지 않아 Python을 이용해 개발하는 경우 대부분 Vi 혹은 Emacs 같은 텍스트 편집기를 이용해 코딩을 하고 다시 터미널 혹은 콘솔로 나와 일일이 디버깅을 해야만 한다. 게다가 import 한 모듈의 API에 대해서 알고 싶으면 Python Interpreter에서 다시 봐야 하는 등 상당한 불편함이 있다. 이에 본 논문에서는 Visual C++ 등에서 제공하는 것처럼 한 화면에서 편집, 컴파일, 수행결과 보기, 디버거 등을 수행할 수 있는 Python 통합 개발 환경을 구축하였다. 앞으로 본 논문에서 구축한 Python 통합 개발 툴을 Visual Python 으로 부르겠다.

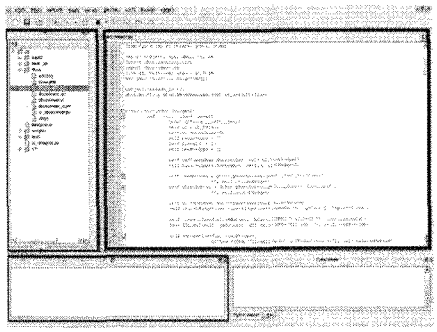
본 논문의 구성은 다음과 같다. 2장에서는 개발 환경을, 3장에서는 Visual Python의 구성요소를 각각 설명하고 4장에서는 결론을 기술한다.

II. 개발 환경

개발에 사용한 GUI는 PyQ [1]로 이는 Trolltech 사에서 개발한 Qt의 Python 바인딩 버전이다. 모든 개발은 Ubuntu 환경[3]에서 이루어졌고 Qt Designer[4]를 이용하여 UI를 작성하고 작성된 UI 파일을 pyuic 명령어를 이용해 Python 파일로 변환하였다. Qt는 모든 모듈과 함수들의 매뉴얼인 QT Assistant 를 제공한다. 또한, Qscintilla[6]라는 텍스트 편집 컴포넌트가 제공되는데, 이는 GUI 라이브러리도 제공한다.

III. Visual Python 의 구성요소

Visual Python 은 MDI (Multiple Document Interface) 방식으로 구성했으며 메인 윈도우 프레임(메인 윈도우 프레임) 속에 크게 리소스(리소스) 부분, 편집기(Editor)부분, 출력(Output) 부분 등의 3가지 요소가 플러그인 형식으로 들어간다. 〈그림 1〉은 Visual Python의 화면 구성을 보여주는데 왼쪽 위 화면이 리소스 부분이고 오른쪽 위가 편집기 부분, 왼쪽 아래 화면이 출력부분, 오른쪽 아래 부분이 디버거 부분이다. 아래에서 메인 윈도우 프레임과 3가지 요소를 하나씩 설명하겠다.



〈그림 1〉 Visual Pyhton의 화면 구성

III-1. 메인 윈도우 프레임

메인 윈도우 프레임은 Qt Designer를 이용하여 제작했으며 나머지 3가지 요소는 Qt에서 제공하는 QDockWidget [5] 형식으로 만들었다. 이것은 메인 윈도우 프레임 형식인 QMainWindow의 안쪽에 붙을 수 있는 기능을 제공하며 위에서 보이는 것과 같이 아래, 위, 왼쪽, 오른쪽에 붙을 수 있다.

3가지 요소를 플러그인 방식으로 구성한 방법은 pluginmanager.py 파일에서 구현하였다. 메인 윈도우 프레임이 인터프리트 되면서 Pluginmanager의 인스턴스를 한번 발생시키고 Pluginmanager의 load라는 함수를 실행시킨다.

III-2 리소스(Resource)

리소스는 폴더를 열고 그 폴더의 하위 폴더와 파일들을 조사하여 트리 구조로 보여주는 일을 하는데 이는 caldirec 라는 함수가 수행한다.

caldirec 함수는 먼저 QtCore.QDir 에서 제공하는 entryInfoList 메소드[2]를 사용해 해당 디렉토리 안쪽의 디렉토리 및 파일을 list 형식으로 저장한다. 하나 하나씩 파일인지 검사하고 파일일 경우 뒤의 확장자 py 혹은 pyw 이라면 리소스의 트리 구조에 넣고 QtGui.QFileIconProvider 에서 제공하는 아이콘으로 앞쪽 아이콘을 표현한다.

디렉토리일 경우 기본적으로 ".", ".." 이런 요소들을 걸러내고 트리구조에 넣은 후 동일하게 아이콘을 표시하고 다시 하위 디렉토리를 찾기 위해 findSubdirec 함수를 호출한다. 그 이외에는 파일인 경우와 하는 일이 동일하다.

리소스가 수행하는 역할 중 또 하나는 트리 구조의 파일을 더블클릭 했을 경우 편집기 에게 넘겨주는 역할이다. 이것은 QAbstractItemView 의 activated 라는 시그널을 이용해 구현했다. activated 시그널은 해당 아이템을 클릭 혹은 더블클릭 또는 엔터(enter)를 누르면 이 시그널이 발생한다. 시그널이 발생되면 편집기의 open 함수를 아이템의 절대 경로와 함께 호출한다.

III-3 편집기

편집기는 기본적으로 Qt의 Tab Widget을 상속받는다. 또한 Qsintilla를 상속받게 된다. 그래서 그 두 객체의 함수를 기본으로 사용하게 되며 사용을 하는 방법은 SendMessage를 사용하여 Qscintilla의 필요한 함수들을 사용할 수 있다.

Font와 Size 같은 경우 pyQt 의 QFont와 QSize를 사용하여 각각의 글자 폰트(Font)와 글자 크기(Size)를 구성한 뒤 QScintilla 의 함수들을 이용하여 편집기에 장착한다. 또한 그밖의 유저 함수들과 각각의 필요한 시그널 들을 connect 함수를 이용하여 연결시켜 줌으로써 편집기를 구현하는데 있어 기본적인 것들을 초기화 시켜준다.

Scintilla는 소스 코드 하일라이팅, 코드 자동 완성, 소스 코드 폴딩, 콜 팁 등의 기능을 가지고 있다. 또한 Margin이라는 영역을 두어 행 번호와 브레이크 포인트(break point) 표시, 소스 코드 폴딩 마크를 표시할 수 있다.

이러한 Scintilla의 기능을 사용하여 초기화된 편집기에 구현한 주요 기능은 다음과 같다.

가. 폰트

QFont 함수의 정의는 〈그림 2〉와 같다. Qfont를 상속받아 알맞은 값으로 설정한 후 QScintilla의 member Fuction인 setfont로 설정하여 준다.

QFont::QFont ()

Constructs a font object that uses the application's default font.

See also QApplication::setFont() and QApplication::font().

〈그림 2〉 QFont 함수 정의

나. 인덱스 (Index)

편집기는 기본적으로 PyQt의 tab widget을 상속 받는다. tab widget의 indexOf 함수를 사용하여 파일의 인덱스 만큼을 margine bar에 표시하게 된다.

다. Import된 함수 자동완성

period(.)이 눌러졌을 경우 현재의 line과 index 포지션

을 얻은 후 그 값으로 인해 text를 얻어오게 된다. 얻어진 텍스트는 're' 모듈의 함수를 사용하여 파싱한 후에 그 모듈에 알맞은 함수들을 리스트에 넣어 뿌려주게 된다.

라. Backspace 눌렀을 시 자동 완성

동작원리를 예를 들어 설명하면 만약 'abcd'라는 함수를 쓰다가 'd'를 backspace 키를 이용하여 지웠을 경우 현재의 위치와 단어의 시작위치를 근거로 하여 남은 단어 'abc'를 의 값을 얻어온 후 얻어진 값으로 파싱하여 모듈의 함수네임들을 불러와 리스트에 나타내어 준다.

마. Breakpoint

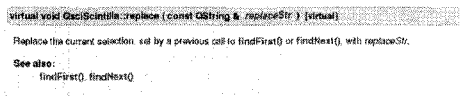
마우스가 더블클릭을 하였을 경우 이벤트를 발생시켜 준다. 클릭된 곳의 margine 값을 구해서 만약 정해놓은 구역의 margine이 클릭되었을 경우 그곳에서 setBreakpoint 함수를 호출하여 마치 그곳에 breakpoint를 걸어놓은 동일한 효과를 준다.

바. Find

검색창인 FindDialog 에서 얻어진 검색어를 편집기의 find 함수에 인자로 넘겨준다. QScintilla에서 제공되는 FindFirst라는 함수를 사용하여 단어를 찾고 함수가 실행된 후 바로 FindNext를 사용하여 다음 단어를 찾게되는 과정을 소스가 끝날 때 까지 반복하게 된다.

사. replace

Replace 함수의 정의는 〈그림 3〉과 같다.



〈그림 3〉 Replace 함수 정의

검색창에 Replace에서 얻어진 검색어와 대체될 단어를 얻어온 후 편집기의 replace 함수에 두 단어를 인자로 넘겨주게 된다. QScintilla에서 제공되는 repalce라는 함수를 사용하여 검색단어를 찾은 후에 changed SIGNAL을 일으켜 대체 단어로 바꿔준 후 FindNext 함수를 사용하여 다음 위치를 찾아 문서가 끝날 때 까지 반복하게 된다.

아. Folding / unfolding

Folding / unfolding를 제공하기 위해서는 QScintilla에서 제공되는 foldAll 이란 함수를 사용한다. 이 함수 같은 경우 한한 실행되었을 때 만약 unfolding 상태이면 folding 상태로 folding 상태이면 unfolding 상태로 바꿔 주는 함수이다.

Folding / unfolding이 표시되어 있는 위치를 찾은 후 fold level을 조사하여 Base 값과 HeaderFlag를 or 연산하여 index의 foldlevel을 검사한후 조건에 맞다면 folding 시켜주거나 unfolding 시켜주게 된다.

III-4. Output

Output 에서는 프로그램을 실행한 후 출력 텍스트 혹은 에러 메지지를 보여주는 것으로 QTextEdit 를 기반으로 만들어졌다.

디버그는 Python -u 옵션을 이용해 버퍼링 모드를 해제하고 프로그램을 실행하면 해당 프로그램의 Standard Output이 그대로 나오게 된다.

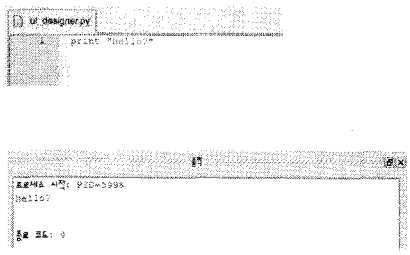
프로그램을 실행할 때 QProcess를 이용해 실행하고 프로그램의 Standard Output 의 발생여부를 확인할 수 있는 시그널인 readyReadStandardOutput, 그리고 Standard Error 의 발생여부를 확인할 수 있는 시그널인 readyRead StandardError를 이용하여 텍스트를 출력했다. 또한 프로그램을 실행할 때 startDetached 를 이용하여 Visual Python의 안에서 돌아가는 프로그램이 아닌 개별적으로 프로그램이 실행하게 했다.

정상적인 출력일 경우 다시 말해 standard output인 경우 그냥 default로 검은색을 사용했으며 standard error인 경우 빨간색을 사용해 출력했다. QTextEdit는 기본적으로 html을 지원하기 때문에 아래와 같이 표현해서 빨간색이 나오도록 했다. 그 예가 〈그림 4〉와 같다.

```
def __error(self, error):
    self.parent().activateDock()
    #print 'error', error
    self.__status = ''
    cause = self.tr('Error: %1').arg(self.tr('Unknown error'))
    if error==QtCore.QProcess.FailedToStart:
        cause = self.tr('Error: %1').arg(self.tr('Failed to start'))
    elif error==QtCore.QProcess.Crashed:
        cause = self.tr('Error: %1').arg(self.tr('Crashed'))
    elif error==QtCore.QProcess.Timedout:
        cause = self.tr('Error: %1').arg(self.tr('Timedout'))
    elif error==QtCore.QProcess.WriteError:
        cause = self.tr('Error: %1').arg(self.tr('Write error'))
    elif error==QtCore.QProcess.ReadError:
        cause = self.tr('Error: %1').arg(self.tr('Read error'))
    else:
        cause = error
    cause = '<font color="#ff0000">'+toHtml(cause)+'</font>'
    self.editor.append('<br>'+cause)
    remote = PluginManager.instance().lookup('remote')
    self.disconnect(remote.server(),
            QtCore.SIGNAL("agentConnected(PyQt_PyObject)"),
            self.__debuggerConnected)
```

〈그림 4〉 밑줄 부분이 html의 붉은색 효과

〈그림 5〉는 표준 출력을 output 프레임에 가져온 결과를 보여준다.



〈그림 5〉 Standard output을 가져온 결과

IV. 결론

본 논문에서는 한 화면에서 편집, 컴파일, 수행결과 보기, 디버거 등을 수행할 수 있는 Python 통합 개발 환경을 구축하였다. 이러한 환경의 구축은 저자가 아는 한 처음으로 만들어진 것이며, 앞으로 Python 사용자에게 편리한 개발 환경을 제공해 줄 수 있을 것으로 기대된다.

향후 부분 수행 등의 좀 더 다양한 기능의 추가와 튜닝을 통해 Python 통합 개발 환경을 개선해 나갈 것이다.

참고문헌

[1] http://www.riverbankcomputing.co.uk/
[2] Fredrik Lundh, Python Standard Library, 2001
[3] 이강성, 열혈강의 파이썬, 2005
[4] Python Homepage, http://www.python.org/doc/
[5] 파이썬 사용자 모임 http://www.python.or.kr
[6] QScintilla Class Library
http://www.riverbankcomputing.co.uk
/static/Docs/QScintilla2/annotated.html
[7] Developerworks와 Micro Software의 박준철 책임연구원님의 글
http://www.ibm.com/developerworks
/kr/library/opendw/20070529/
http://www.imaso.co.kr/?doc=bbs/
gnuboard_pdf.php&bo_table=article&
page=3&wr_id=647&publishdate=20030901