Automated Comment and Notebook Generation for Python Code

Mayukh Nair[1,+], Sanchit Bansal[1,+], Vinayak Agarwal[1,+], Anjali Goyal[1], and Ashish Sureka[1,*]

[1]Department of Computer Science, Ashoka University, India
[*]Corresponding author: Ashish Sureka (ashish.sureka@ashoka.edu.in)
[+]these authors contributed equally to this work

ABSTRACT

Documenting code is highly integral to ensuring that people reading the code can fully understand the processes taking place and extend the code properly, thus making it crucial for educational as well as archival purposes. A unique approach to documentation of code files, usually a manual and cumbersome task, has been proposed here by means of a series of programs which perform an analysis of written Python code and automatically generates comments for some logical operations of the code, as well as compiling a Jupyter / Interactive Python (IPython) Notebook from which formatted text and code can be exported to a variety for formats. This approach to documentation has a variety of applications which includes a simpler and elegant method to document code for amateur programmers and educators, as well as simplify creating Jupyter Notebooks from less feature rich development environments and text editors.

Introduction

Software professionals agree that documentation of existing code is highly crucial to the development process, however this is impeded by the complexity of documentation tools which make it difficult to maintain code. A survey conducted by Forward et al. on tools and technologies for software documentation reported that developers preferred automated documentation tools, as well as the simplicity that word processors provide[1]. Most documentation tools require a great deal of manual work and learning which does not give the developer a self-motivated incentive to present their code in elegantly documented formats.

Python is a programming language that is easy to learn and has a simple but effective approach to object-oriented programming[2], and is widely preferred for multiple applications. There are multiple documentation tools available for code written in Python, such as Sphinx[3], however they suffer from the before-mentioned issue of being complicated and cumbersome to implement. Such tools deter first-time amateur programmers as well as educators from creating detailed documentation for the code they write.

Python, like many other programming languages, offers in-built support for breaking down syntactically correct program code into an abstract tree representation of the same, known as an Abstract Syntax Tree (AST). AST allows programmatic access to the structure of written syntactic code and can be utilized to access underlying objects defined in code, their values and operations performed, which can be used to generate comments on values held and processes performed.

Jupyter[4] is a language-agnostic command shell that offers an interactive interface and an interface akin to a notebook which can be formatted, host images and other interactive content, and be presented in a variety of file formats in an elegant manner. Code can be viewed, run and modified live in Jupyter notebooks, thus making them an ideal tool for both code writing as well as documentation.

A set of tools, written on the Python programming language and powered by libraries written for the language, allow the line-by-line analysis of code and automatically creates comments that can indicate the value or range that variables in code can inherit. A simplified commenting process to enter the author's text and notes which supports Markdown[5] for formatting plain text allows the code to be converted into Jupyter notebooks, which can then be exported to a variety of file formats. This is presented in an easy to use web interface which converts code in plain text into auto-commented and richly formatted code.

Methods

Approaches

A primary goal for auto generation of code was to return the runtime values of the variables declared in the python code. We tested several methods to solve the issue:

- **Memory** location of Variable : Aim was to use the function **id()** of python to obtain the memory address of each variable and access that memory after program execution.However there were several drawbacks to this approach. First, memory could have been allotted to something else once program execution was over. Second, we would not be able to track changes to the value of the variable.

- **Abstract Syntax Tree** (AST) : The *ast* module helps Python applications to process trees of the Python abstract syntax grammar. Previous work on ast had been used to visualize the program structure only. Parsing the python code to create a tree made it difficult to obtain flow of code based on tree traversal. External tracking of changes to variable values had to be implemented via dictionary. AST does not evaluates program expressions. Recursiveness of expressions - eg. x = x+(y/2) was tackled by using functions and recursive programming. Complex object referencing had to be implemented to traverse the tree.

Code structure

The code is divided into four distinctive programs, built on the Python programming language, each of which perform a distinct task. All of these programs connect to each other sequentially to form as the part of a web application which does not require any user setup.

- **app.py**: This is the code that powers the graphical front-end for the web application, and uses the Flask and Django libraries for interactive web applications on a Python-powered back-end, and consists of Python code which controls a web interface written in HTML5, CSS and JavaScript. The interface offers a text field for the user to paste their written code in, and choose a format which they want their documented code in.

- **comment.py**: This code converts the Python code submitted through the front-end into an abstract syntax tree (AST), which is then parsed by traversing the nodes and accessing the values and attributes for each object and expression. A series of techniques to access the AST, as mentioned in the Approaches section, and tracking the initial and changed values of variables is used to generate a set of comments. This is output as a dictionary with the keys as the line on which a comment is autogenerated, and the comment itself as the value.

- **autocomment.py**: The dictionary provided by comment.py is inserted into the original code here on a line-by-line basis.

- **autodocgen.py**: The commented code is now analyzed line by line. Python-recognized comments (represented by a hash symbol at the start) which contain an asterisk at the beginning of the line are interpreted as plain text, and any Markdown formatting within it is parsed. Other lines are treated as lines of code, and additional code detects code blocks and formats them into cells, the basic unit of a Jupyter notebook. This is then used to generate a Jupyter notebook and export it to formats using the *nbformat* and *nbconvert* libraries.

Results

The various programs are able to work in sequence and generate automatically commented and richly documented versions of the code input by the user. *comments.py* generates a variety of comments which indicate variable declarations, as well as computed results of variable values and mathematical operations. A sample of the comments generated are as shown below.

```
x=8 # Value of variable x was 8
y= x**8 # Value of variable y was 16777216
x = y - (x**5) # Value of variable x was 16744448
print (y+(x%6)) # Value printed = 16777218
if x==y:
    print (x)
else:
    x=x+3 # Value of variable x was 16744451
```

The commented code is then formatted to appear in a Jupyter notebook, and can be exported to one of six formats: PDF, HTML, LaTeX, Script, Slides, or an .ipynb notebook.

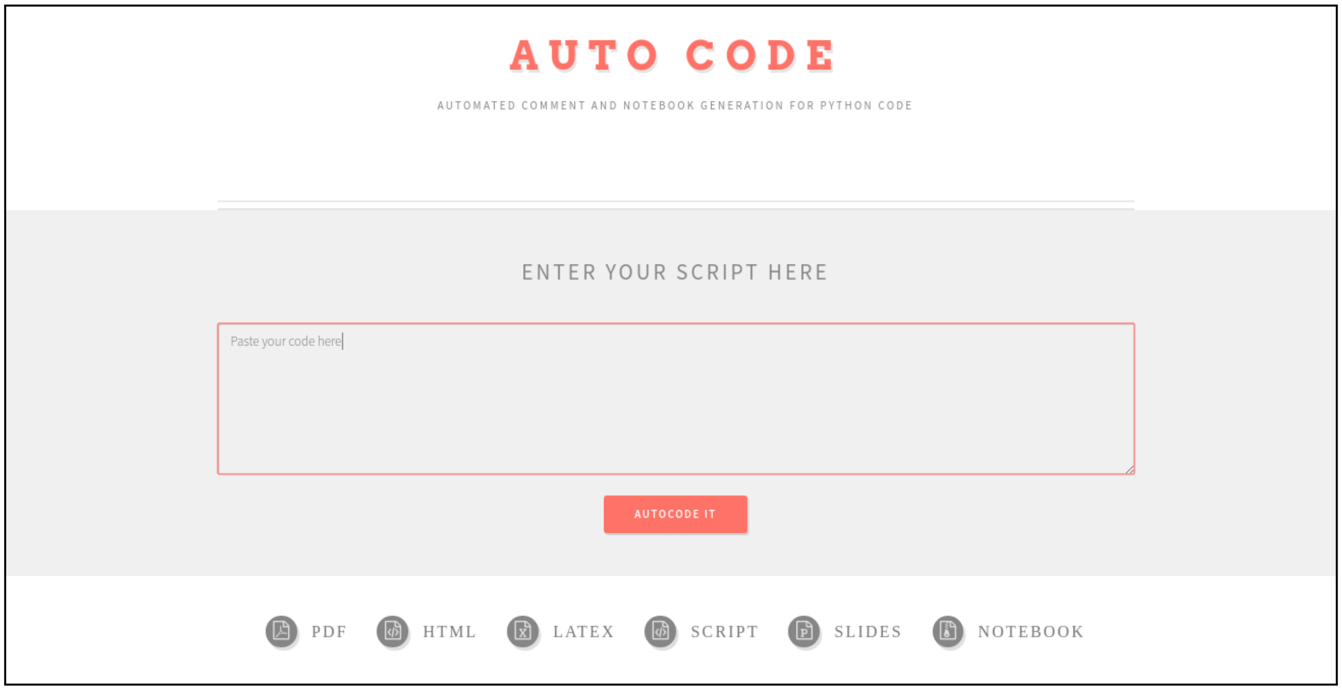
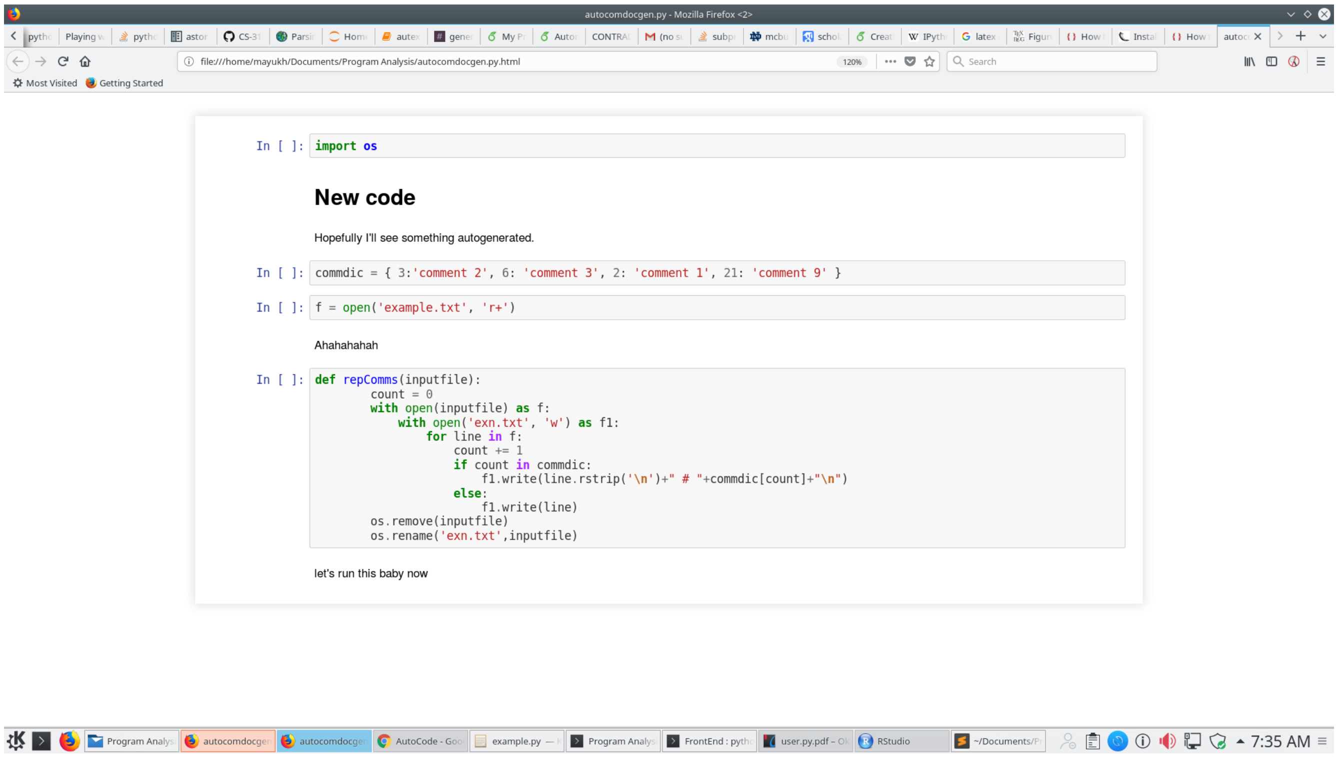


Figure 1. Screenshot of the user interface, powered by *app.py*.


autocomdocgen.py - Mozilla Firefox <2>
file:///home/mayukh/Documents/Program Analysis/autocomdocgen.py.html
Most Visited
Getting Started
7:35 AM

```
In [ ]: import os
```

New code

Hopefully I'll see something autogenerated.

```
In [ ]: commdic = { 3:'comment 2', 6: 'comment 3', 2: 'comment 1', 21: 'comment 9' }
```

```
In [ ]: f = open('example.txt', 'r+')
```

Ahahahahah

```
In [ ]: def repComms(inputfile):
                count = 0
                with open(inputfile) as f:
                    with open('exn.txt', 'w') as f1:
                        for line in f:
                            count += 1
                            if count in commdic:
                                f1.write(line.rstrip('\n')+" # "+commdic[count]+"\n")
                            else:
                                f1.write(line)
                os.remove(inputfile)
                os.rename('exn.txt',inputfile)
```

let's run this baby now

Figure 2. Auto-generated HTML output of formatted and generated notebook.

References

1. Forward, A. & Lethbridge, T. C. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, DocEng '02, 26–33 (ACM, New York, NY, USA, 2002). DOI 10.1145/585058.585065.

2. Van Rossum, G. & Drake, F. L. *An introduction to Python* (Network Theory Ltd. Bristol, 2003).

3. Brandl, G. *et al.* Sphinx: Python documentation generator. *URL: http://sphinx. pocoo. org/index. html (13.8. 2012)* (2009).

4. Kluyver, T. *et al.* Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, 87–90 (2016).

5. Gruber, J. Markdown language (2004).

Author contributions statement

M.N., S.B. and V.A conceived the project idea, M.N. was responsible for *autocomment.py* and *autodocgen.py*, S.B. was responsible for app.py and the user interface and V.A. was responsible for comment.py and integrating the different programs into a sequential processing system. A.G. and A.S. supervised the project and offered feedback and guidance towards the same. M.N, S.B. and V.A. reviewed the manuscript.

Additional information

The program is parsed to create a tree. Every node is hence traversed using a for loop. For every node, we check if its an assignment/expression/if-else statement.