| **Number** | **Version** | **Title** | **Credits** | **Assessment** |
| --- | --- | --- | --- | --- |
| AS91906 | 1 | Use complex programming techniques to develop a computer program | 6 | Internal |

|  |
| --- |
| **Submission Checklist (Achieved)** |

To meet the achievement level criteria it is important that your code meet the minimum expectations of the assignment. Before submitting, please complete the checklist below to ensure you have not missed anything significant in your submission.

|  |  |
| --- | --- |
| Checklist | Done? Y/N |
| My program uses variables storing at least two types of data (e.g. numeric, text, Boolean) | Y |
| My program uses sequence, selection (IF) and iteration (LOOP) control structures | Y |
| My program uses Input from a user, sensors or another external source & produced output | Yish |
| My program uses two or more complex programming techniques. | Y |
| My code is set out clearly using suitable whitespace | Y |
| I have included comments to document and explain what the code is doing | Y |
| I have prepared a video recording of my program working and will submit it with this document. | Y |
| I have completed the appropriate testing table(s) to show how effective my program is. | Yish |
| I have copied my completed code into the space provided in this document | Never |
| I have prepared a video recording of my program working and will submit it with this document. | Y |

|  |
| --- |
| **About Your Project** |

In the space below, tell me anything about your program that I may need to know.

Include elements such as…

* What features did you add beyond the basic brief?
* Were there any changes to the brief that were agreed by your teacher?
* What environment is needed for your code to run?
* Etc.

In short, explain anything beyond hitting run button for a standard python environment and expecting the program to act as the example program provided.

|  |
| --- |
| Need python >= 3.12 (match statements, @override) (tested on 3.12.3)  Program acts as a python package (module). It is not published on PyPI, so cannot be installed as easily as other packages. README.md contains further install instructions  To use debug graph, need  pip install networkx  pip install matplotlib  pip install pyqt5  pip install scipy  To use test suite’s HTML output, need  pip install pywin32  These are automatically installed by pip if the package is installed with the [debuggraphviewer] or [copy-html] options respectively. (See README.md, Building Manually section)  Program is a regular expression engine, constructing a deterministic finite automaton (a kind of state-machine) from a given regular expression. The DFA can efficiently (O(n) time) search strings.  A collection of examples are provided in `examples/`, which can be simply run once the required libraries are installed  To use as library, just import (how u do this will depend on where ur code is relative to the project), and then use the Regex constructor to compile a regex from a pattern, like `Regex(r"\w+(?:\.\w+)\*@\w+(\.\w+)+")` for a simple email regex. If this is stored (say, in a variable called rx), then it can be used as such: `rx.test(“my.email@gmail.com”)`, which should return True |

|  |
| --- |
| **1. Achieved Level Documentation** |

# 1.1 - 1.6

Evidenced within the code

# 1.7 Complex Tools

In the table below identify the complex tools you have used, why they have been used and where the evidence of their use can be found.

|  |  |  |
| --- | --- | --- |
| Complex Tool | Why was it used | Where is it used |
| *Student created classes/objects* | *To organise code and represent different constructs that make up the DFA structure (States (is a node) and Edges (are connections))* | *Everywhere* |
| GUI | To be able to actually see what is going on while debugging because otherwise debugging *frustrating* (i.e. I was crying while friends who saw it were like “ooo pretty picture”) bugs such as this:  (it was a simple regex it shouldn’t have exploded into that many states)  - would be entirely impossible as it would be impossible to keep track of all the states and edges in your head | debug\_graph\_viewer.py  rant:  turns out that theres a point when the code is complex enough that it does its own thing instead of what u were picturing  doesn’t help that its not deterministic because sets are unordered, and iteration order is thus random  the same result should always be achieved in the end, but the way it gets there is different each time so gl finding bugs :)  rant over |
| External libraries | To draw the gui because I wasn’t going to draw my own guis manually and do stuff such as picking a good layout to position the states in to make the graph more comprehendible; I didn’t have enough time to fix bugs AND make GUIs | debug\_graph\_viewer.py  uses networkx, matplotlib, scipy, pyqt5 |

|  |
| --- |
| **2.Merit Level Documentation** |

# 2.1 – Variable Names & Comments

Evidenced within code

# 2.2 - Conventions Used

Each programming language has a set of conventions that should be followed. For Python the current conventions are documented in the [PEP 8](https://www.python.org/dev/peps/pep-0008/) style guide and contain guidance such as:

* Variable and function names should all be lower case with words separated by underscores.
* Lines of code should not be longer than 79 characters
* Lines of comments should not be longer than 72 characters
* Functions should always contain a docstring

Other languages such as [C#](https://docs.godotengine.org/en/3.1/getting_started/scripting/c_sharp/c_sharp_style_guide.html) and [JavaScript](https://www.w3schools.com/js/js_conventions.asp) also have their own style guides.

In the table below identify the conventions you have followed and provide evidence that they have been applied.

|  |  |
| --- | --- |
| Selected Programming Language | Style Guide Used |
| Python | PEP(8 and others) |
| Evidence that Conventions have been applied | |
| *e.g.1 - A screen shot of the output from an online checking tool like* [**https://pep8plus.com/**](https://pep8plus.com/)*and / or* [**https://www.codewof.co.nz/style/**](https://www.codewof.co.nz/style/) **OR a not-online checker because im not crazy**  *e.g. 2 - A description of all of the conventions you have taken with examples. IM SORRY DID YOU SAY ALL????* | |
| Pylint Output:    All runs: [Pylint · Workflow runs · wntiv-main/regex (github.com)](https://github.com/wntiv-main/regex/actions/workflows/pylint.yml)  Note that many warnings have been disabled in the code (# pylint: disable=error-type), BUT ONLY where disabling those warnings is deemed to be valid. E.g. protected member use is often disabled for the internal library code, which should be able to access a value, but the end user shouldn’t (hence the member being protected). Some functions have many arguments, conditions, blocks, exit points (return), etc. Pylint flags “design” issues for these functions, but in cases where the function is deemed to be necessarily that large, these warnings have been disabled. This is common with very complex functions, often which cannot be refactored due to many nested loops which must pass a lot of state between them. (Trust me, I have considered this)  Conventions I have followed:  Indentation – 4 spaces (even tho tabs better - prove me wrong)  Example: my entire codebase  <- look spaces!!  Continuation lines either align to previous bracket like so:    (src/regex/regex\_factory.py, line 592)  OR are indented by one, relative to the parent line as such:    (src/regex/regex\_factory.py, line 597)  Line length – lines should be no more than 79 chars  e.g.    (src/regex/regex\_factory.py, line 592)    (src/regex/regex\_factory.py, line 597)  This makes code more readable, especially on narrow viewports (e.g. splitscreen, diff editor)  [Should a Line Break Before or After a Binary Operator?](https://peps.python.org/pep-0008/#should-a-line-break-before-or-after-a-binary-operator)  TLDR; before. E.g.    (src/regex/regex\_factory.py, line 541)  Blank lines:  Top-level class and functions surrounded with two blank lines  Methods with a signle blank line, etc.  e.g.    (src/regex/regexutil.py)  [Imports](https://peps.python.org/pep-0008/#imports)  e.g.    (tests/regex\_tests.py)  [Module Level Dunder Names](https://peps.python.org/pep-0008/#module-level-dunder-names)    (src/regex/\_\_init\_\_.py) (\_\_version\_\_ attr dynamically used for package deployment version)  [PEP 257 – Docstring Conventions | peps.python.org](https://peps.python.org/pep-0257/)  Everywhere  e.g.    Note: I have often skipped docstrings for \_\_init\_\_ when there are little/no arguments and/or the class docstring explains it well enough  No trailing whitespace  Example: well u cant see it, can you…?  In all seriousness, pylint warns about trailing whitespace, ergo, see above; no warning -> no trailing whitespace  Ok ive read enough PEP8 you get the gist  Oh yeah naming conventions:  PascalCase for classes  snake\_case elsewhere  begin with \_ for protected members  ALL\_CAPS for const  Example: | |

**NOTE: This section is intended to demonstrate that you are aware of and have adhered to the conventions for your selected language. A screen shot alone does not demonstrate that you have been successful throughout your entire project so make sure that you thoroughly check each aspect of your work.**

|  |
| --- |
| **3.Excellence Level Documentation** |

In the table below explain how you have addressed each of the refined characteristics of the assessment and where this can be seen in your program.

|  |  |  |
| --- | --- | --- |
| Refinements | Explanation | Where is it seen |
| 3.1 The program code is efficiently and effectively organised | API code is packaged into one big module, internally separated into separate files for organisation | src/regex |
| 3.2 The program has features to help manage user input | Yeah if the user does smth wrong we raise an appropriate error (remember we are API so we want to alert API user with hard error if they e.g. input invalid regex) | Mostly in RegexBuilder.PatternParseError  In src/regex/regex\_factory.py |
| 3.3 The program has user friendly output | 1 - its an API it doesn’t really have output, it returns a regex object which can be used in sensible ways and is well documented (= user-friendly?)  2 – there is always the debug graph viewer if the user is confused | Debug output in src/regex/debug\_graph\_viewer.py  Api output in class Regex in src/regex/regex.py |
| 3.4 The program includes error capture features | No it has error *raising* features in case the user does smth weird we raise an appropriate error with a detailed error message | RegexBuilder.PatternParseError  In src/regex/regex\_factory.py |
| 3.5 The program report unexpected behaviour to the user | I just told u twice already | RegexBuilder.PatternParseError  In src/regex/regex\_factory.py  Again |

|  |
| --- |
| **4.Final Submission & Testing** |

# 4.1 - Your Video

Record a video showing your game in operation with valid inputs. If you wish to show any error capturing you have included in your program, record these in a separate video.

Be sure to take your time and give the viewer of this video and opportunity to fully see your game in action. Save your video recording in a suitable file format (e.g. wmv or mp4 – NOT an iSpring file)

|  |
| --- |
| Below tell us the name of this file and where it is stored. |
| project\_root/docs/\*.mp4 (multiple videos if I forgot about things)  if u cant find it, its in the same place on github, link is <https://github.com/wntiv-main/regex> |

# 4.2 – Demonstrate Authenticity

Evidenced by Teacher throughout project development & after submission

# 4.3 Valid “Expected” Input Testing (Achieved)

All unit test outputs over the development process can be seen at [Tests · Workflow runs · wntiv-main/regex (github.com)](https://github.com/wntiv-main/regex/actions/workflows/tests.yml)

Valid “Expected” tests are tests that show your program operates as expected if the inputs received are as you expect them to be. For each input test that your program works using valid inputs.

# 4.4 Boundary Input Testing (Merit)

Boundary tests are tests that show your program operates as expected if the inputs received are at the extreme ends of the allowed input range. It is also worth including the other side of each boundary to show that the boundaries are actually operating correctly. E.g. To test an input asking for a number between 1 and 10, apply 4 tests: 1 then 10, but also 0 and 11!

Works very well for *numerical* data, with *well-defined ordering*, however not so well for *categorical* data, such as the strings I am working with… so very limited boundaries to test.

# 4.5 Comprehensive Invalid Input Testing (Excellence)

**Comprehensive testing needs to show testing of all areas of the program.**

Invalid tests are tests that ensure the program operates as expected regardless of the inputs made by the user. This will be the biggest of all of the testing sections.

No, it wont be the biggest, because there is very little invalid input testing you can do when the only input is the regular expression pattern string that the end user passes to the API

# Your Code

Please copy the final code from your program into the space below.

|  |
| --- |
| Final Code |
| Yeah right like that will help  <https://github.com/wntiv-main/regex> if you want |