Dylan Vo

Dr. Huffman, Dr. Omojokun

CS 8803

14 July 2025

Code Analysis Project

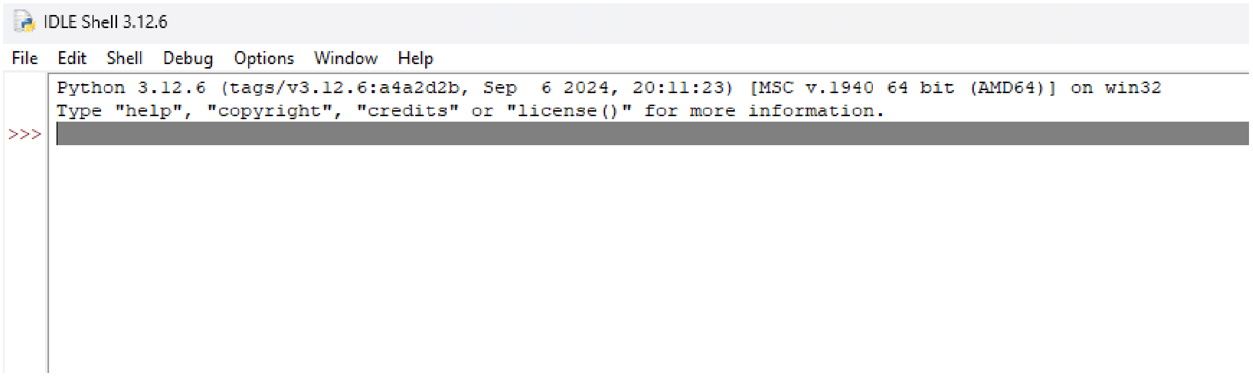
Introduction

In the subject of computer law, a common occurrence is the infringement of software-based intellectual property. When there are claims that one company infringes on another company's intellectual property, third parties must get involved to fully analyze the extent of any violations of these properties. In this analysis, I will be comparing the IDLE and Eric Python integrated development environments (IDEs) from a computer law perspective to understand if the developers of Python and IDLE reserve any rights to make claims on the Eric IDE. Both IDEs will be investigated, particularly in their debugging functionality, based on the following: user interface, software design, algorithms, application programming interfaces (APIs), and source code.

User Interface

IDLE

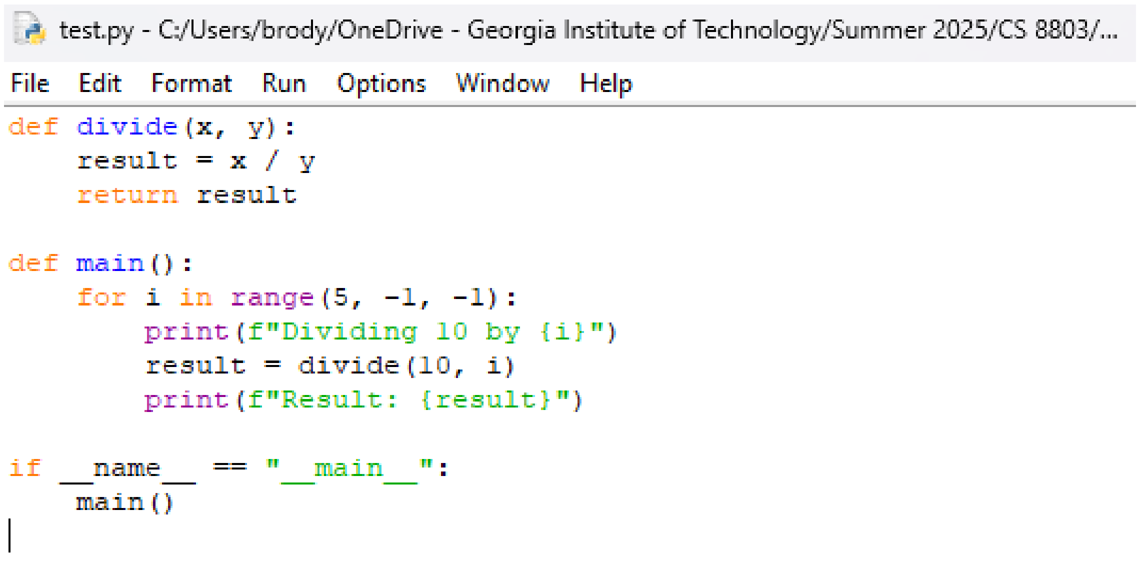
IDLE is Integrated Development and Learning Environment for Python. It comes bundled with Python and is intended to be a basic environment for users to run Python code. As shown in Figure 1, IDLE features a basic UI consisting of a menu bar and an interface for shell commands.



```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

Figure 1: IDLE shell UI

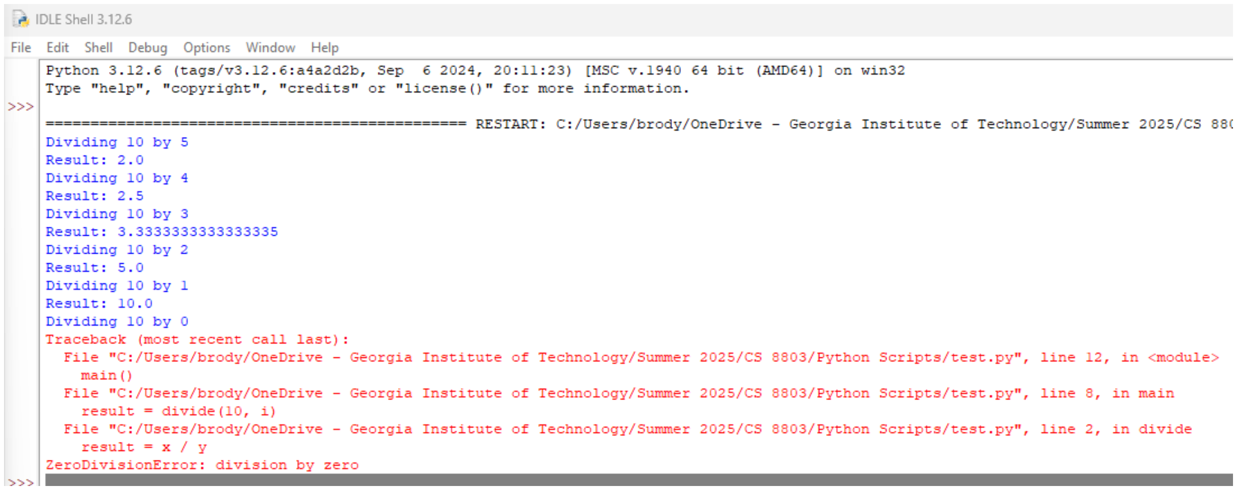
To write code, users must navigate to the menu bar and click File > New File to create a new file or File > Open to open an existing Python file. Figure 2 shows a snippet of the IDLE code editor with a test file "test.py" that will be used throughout this analysis for understanding the debugging features of both IDEs.


test.py - C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2025/CS 8803/...
File Edit Format Run Options Window Help

```python
def divide(x, y):
    result = x / y
    return result

def main():
    for i in range(5, -1, -1):
        print(f"Dividing 10 by {i}")
        result = divide(10, i)
        print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

Figure 2: IDLE code editor

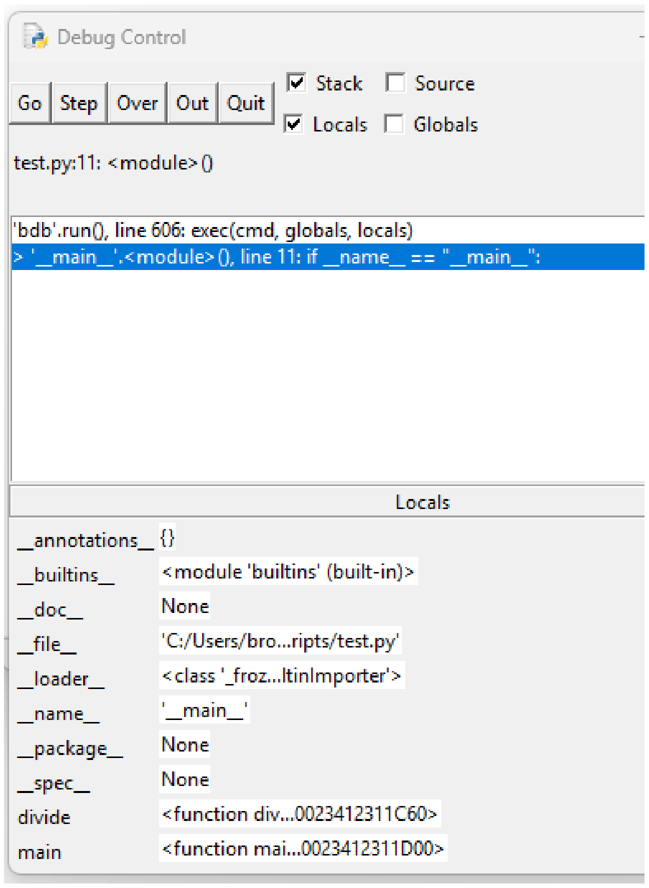
In the IDLE code editor, there is a menu bar and a space for users to modify their code. If the user wants to run their code, they will need to navigate to the menu bar in the code editor and click Run > Run Module. Figure 3 shows the execution of test.py in IDLE.



```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============================================== RESTART: C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2025/CS 88
Dividing 10 by 5
Result: 2.0
Dividing 10 by 4
Result: 2.5
Dividing 10 by 3
Result: 3.3333333333333335
Dividing 10 by 2
Result: 5.0
Dividing 10 by 1
Result: 10.0
Dividing 10 by 0
Traceback (most recent call last):
  File "C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2025/CS 8803/Python Scripts/test.py", line 12, in <module>
    main()
  File "C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2025/CS 8803/Python Scripts/test.py", line 8, in main
    result = divide(10, i)
  File "C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2025/CS 8803/Python Scripts/test.py", line 2, in divide
    result = x / y
ZeroDivisionError: division by zero
>>>
```

Figure 3: IDLE execution of test.py

The test.py script is a simple loop that divides 10 by 5 and prints the result. It then decrements the denominator by 1 and keeps doing this division until a division by zero error occurs.



[✓ Stack] [ ] Source
[✓ Locals] [ ] Globals

```
test.py:11: <module>()
```

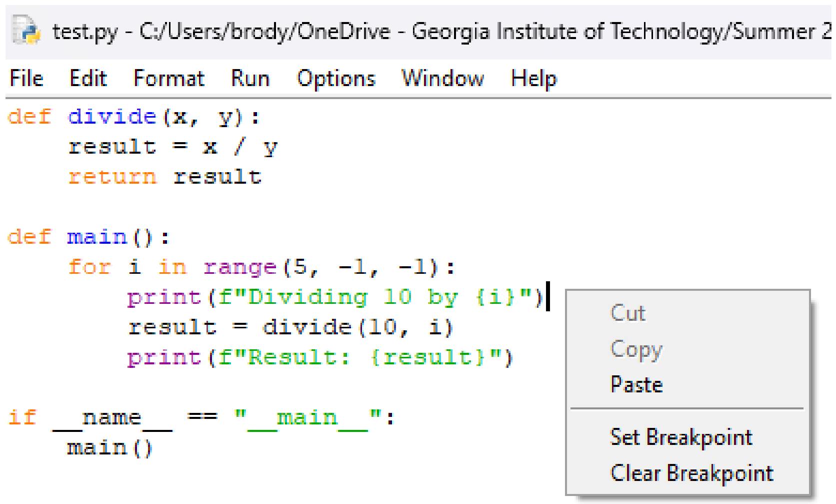
```
'bdb'.run(), line 606: exec(cmd, globals, locals)
> '__main__'.<module>(), line 11: if __name__ == "__main__":
```

Locals

__annotations__	{}
__builtins__	<module 'builtins' (built-in)>
__doc__	None
__file__	'C:/Users/bro...ripts/test.py'
__loader__	<class '_froz...ltinImporter'>
__name__	'__main__'
__package__	None
__spec__	None
divide	<function div...0023412311C60>
main	<function mai...0023412311D00>

Figure 4: IDLE debugger

Figure 4 shows the IDLE debugger control window when debugging test.py. The debugger has options for Go, Step, Over, Out, and Quit. It also has a "Locals" panel that displays the variables that available within the scope of the current code block.


test.py - C:/Users/brody/OneDrive - Georgia Institute of Technology/Summer 2
File
Edit
Format
Run
Options
Window
Help
Cut
Copy
Paste
Set Breakpoint
Clear Breakpoint

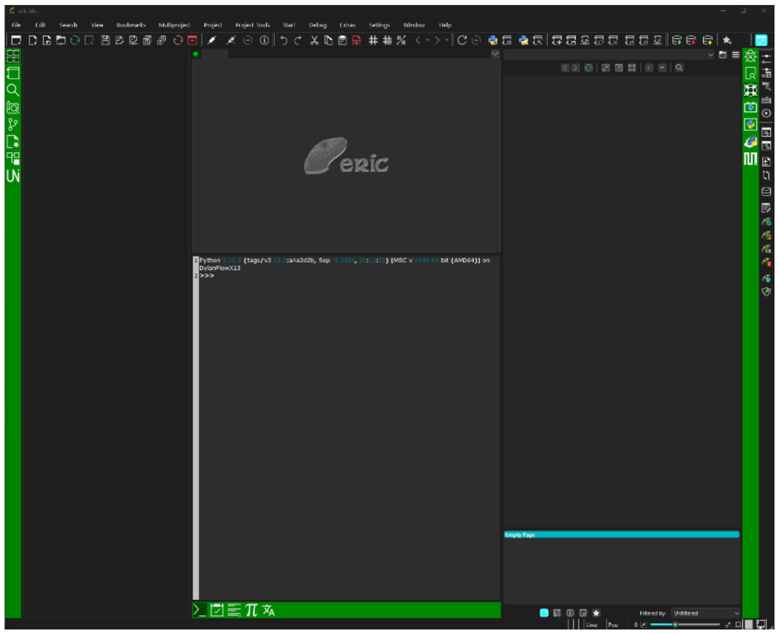
```
def divide(x, y):
    result = x / y
    return result

def main():
    for i in range(5, -1, -1):
        print(f"Dividing 10 by {i}")
        result = divide(10, i)
        print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

Figure 5: IDLE debugger breakpoint options

Figure 5 shows the options for setting and clearing breakpoints in the code editor. To access these options, the user must right-click on the line of code that they'd like to set or clear a breakpoint at and choose the option they want to use.

Eric IDE

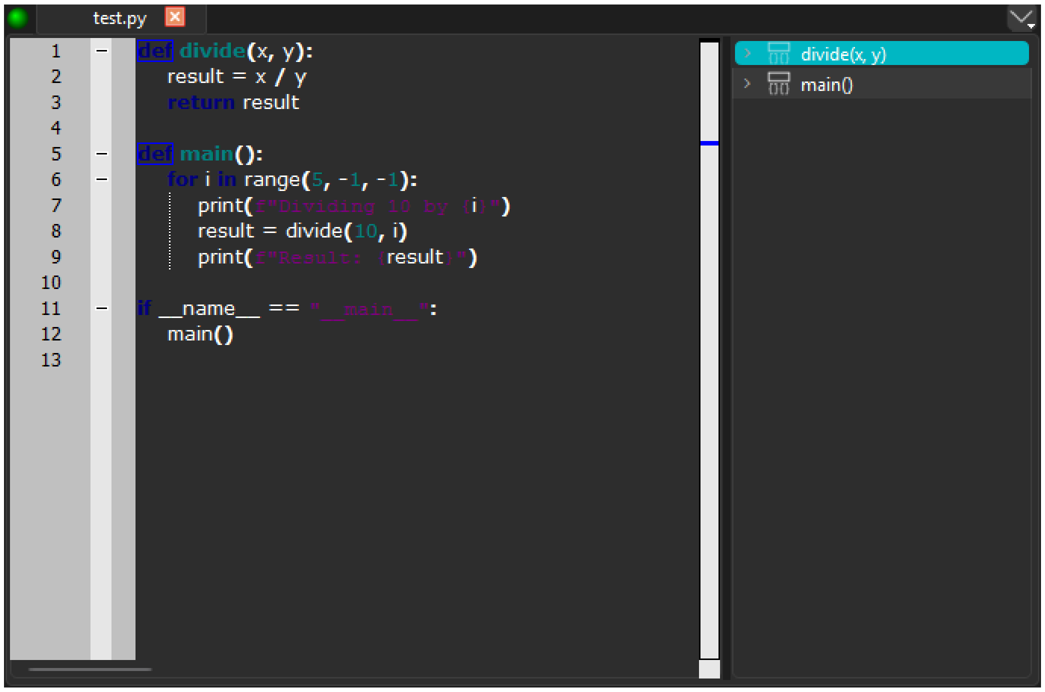
Eric IDE is a full-featured Python IDE written in Python and PyQt. It contains many advanced tools when compared to IDLE. Contrary to IDLE, Eric IDE must be downloaded separately from Python. Figure 6 shows the main UI of Eric IDE, including a menu bar, shell tab, side bars with advanced features, buttons for switching views, and much more.



```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on
DylanFlowX13
>>>
```

Figure 6: Eric IDE main UI

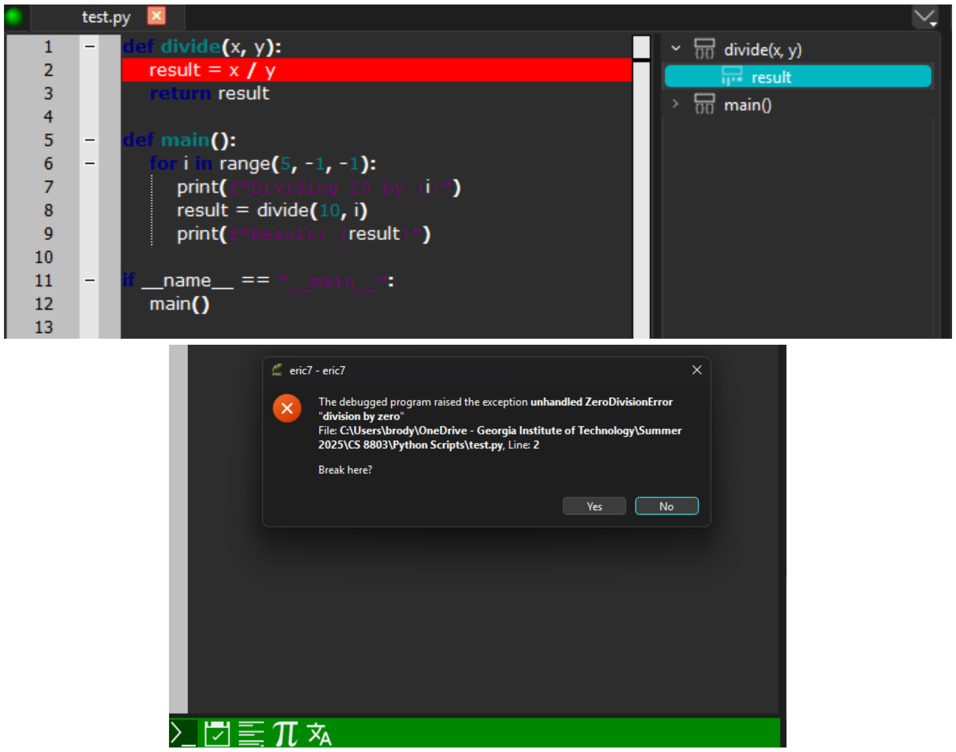
Like IDLE, Eric IDE offers a basic code editor. There is also an extra panel that shows the function in the current code and any variables defined within them. Figure 7 shows these details.



```
def divide(x, y):
    result = x / y
    return result

def main():
    for i in range(5, -1, -1):
        print(f"Dividing 10 by {i}")
        result = divide(10, i)
        print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

Figure 7: Eric IDE code editor

I used the same test.py script used for testing IDLE on the Eric IDE. Figure 8 shows the results. Contrary to IDLE, Eric IDE displays a pop-up window detailing the error that occurred and highlights the line and variable in the code editor that caused the error. This is different from IDLE's approach that simply prints the error following the rest of the prints from the script.



```
def divide(x, y):
    result = x / y
    return result

def main():
    for i in range(5, -1, -1):
        print(f"Dividing 10 by {i}")
        result = divide(10, i)
        print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

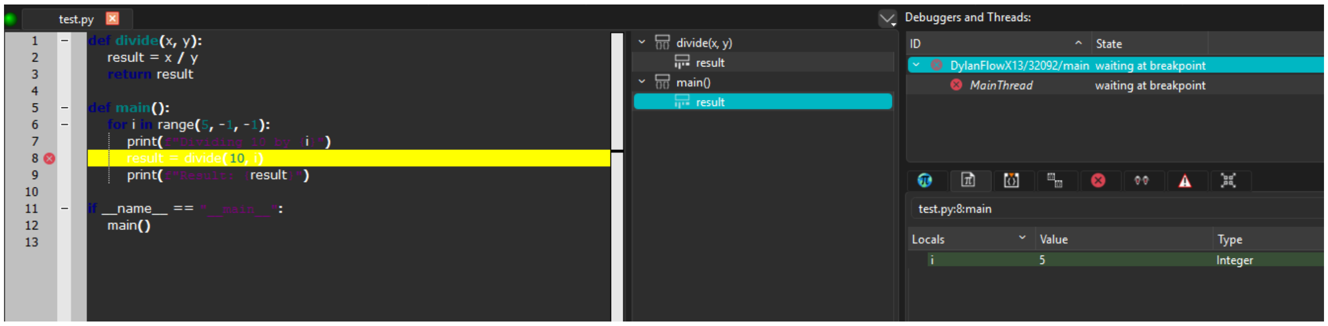
eric7 - eric7

The debugged program raised the exception **unhandled ZeroDivisionError "division by zero"**
File: **C:\Users\brody\OneDrive - Georgia Institute of Technology\Summer 2025\CS 8803\Python Scripts\test.py**, Line: **2**

Break here?

Figure 8: Eric IDE execution of test.py

Eric IDE's debugger includes a more intuitive interface with more features than IDLE. To debug, the user must first set breakpoints by clicking on the space to the right of the line number they'd like the code break at. Once this is done, a red "X" will appear, marking that the breakpoint has been set. To remove it, the user can click the X. To begin debugging, the user must go to the menu bar and click Start > Debug Script or press F5. Once debugging, the user can view information about the current values of the variables and breakpoints that the debugger is waiting for next.



```
def divide(x, y):
    result = x / y
    return result

def main():
    for i in range(5, -1, -1):
        print(f"Dividing 10 by {i}")
        result = divide(10, i)
        print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

ID	State
DylanFlowX13/32092/main	waiting at breakpoint
MainThread	waiting at breakpoint

test.py:8:main

Locals	Value	Type
i	5	Integer

Figure 9: Eric IDE debugger

<u>Comparing the User Interfaces</u>

Feature	**IDLE**	**Eric**
Debugger launch	Menu bar > Debug > Debugger	Menu bar > Start > Debug Script

Debugger step-through options	Go, step, over, quit	Step, step over, step out, continue
Breakpoints	Right click on the line in code > set/clear breakpoint	Clickable breakpoints in code editor
Variable view	No variable panel	Panels for variables, call stack, watch panels
Layout	Minimal and simple	Multi-pane layout with dockable tabs and widgets

Table 1: User interface comparison

Table 1 was created to easily compare the features in the IDLE and Eric IDEs. Using this table, I have come up with a risk assessment of low when it comes to UI. While both IDEs support debugging features and the ability to create, modify, and run python code, their UIs different significantly. The presence of a debugging feature in both IDEs is not problematic from a legal standpoint. IDLE is designed to be very simple and have a workflow based on shell commands. Eric is more similar to modern professional IDEs with many panels and features for software development. There appear to be no copied symbols, icons, or interaction flows between the two UIs. A legal precedent that can be referred to is Apple v. Microsoft, in which Apple Computer, Inc sued Microsoft Corporation for using GUI elements similar to Apple's MacOS. The US Supreme Court denied Apple's request and ruled that the UI elements are not copyrightable under similar circumstances to this case.

Software Design

```
IDLE:

idlelib/

 — debugger.py

 — debugger_r.py

 — rpc.py


Eric:

eric6/

 — Debugger/

    ├── DebuggerInterface.py

    ├── DebugClient.py

    └── DebugServer.py
```

Figure 10: IDE file organization

Both IDLE and Eric use a client-server architecture for their debuggers. However, their implementations vary as IDLE uses a simpler approach while Eric has a more modular and layered design. This can be seen through the file structure shown in Figure 10. IDLE's debugger components are bundled with the main application, relying on a Remote Procedure Call (RPC) system to talk to the debugged subprocess. Within IDLE's source code in the idlelib directory, debugger.py and rpc.py contain the code implementing the debugger.

Eric's modular debugger is implemented as a separate subsystem from the rest of the main IDE. It has its own client-server architecture for communication, breakpoint management, and execution control. Since Eric is also built using the PyQt framework, it also utilizes Qt's signals and slots and dockable UI components. This architecture gives the IDE more flexibility for the developers and for users.

While both IDEs consist of a debugger operating with a client-server architecture, this is a standard design decision that provides debugger functionality and cannot be made more unique. Based on the source code and functionality testing, there is no clear evidence that the developers of Eric copied any parts of the structure of IDLE's software. From a legal perspective, the common design elements would be considered standard design practice, making the risk of infringement based on software design somewhere between low and moderate.

Algorithms

Both IDEs rely on Python's built-in functionality for tracing and debugging to manage execution flow. They both utilize standard debugging algorithms like stepping into and over functions, continuing execution, and exception halting. While these are common algorithms used, these are also features that are standard for any debugger (Algorithmic Program Debugging). They are also implemented using standard practices found in Python's documentation. For example, using Python's function sys.settrace(), both IDEs implement line-by-line execution by setting a trace function. This trace function is also used with other algorithms to implement other functionalities like exception handling.

IDLE's approach uses minimal abstraction, which makes the interface much simpler for the user. The debugger is tied to the shell and uses hooks to step through the code. On the other hand, Eric implements similar functionality using a more object-oriented structure. Eric has protocol handling between the IDE and the debugging server and provides the user with more feedback in the form of variable tracking and stack inspection.

Both IDEs implement the same standard debugger algorithms, but in different ways. They also implement algorithms that rely heavily on Python's runtime and limitations. These limitations and similarities are expected as they are both environments for the same coding language trying to create the same functionality. Based on these findings, there is no clear evidence showing that Eric may have copied algorithms directly from IDLE. A legal precedent that set the foundation for differentiating between ideas and expression was Baker V. Selden in 1879. As a result, the Corut ruled that copyrights shall protect expression of ideas and not the idea itself. The same ruling can be used in the context of algorithms as the implementation of algorithms represents a common system that must be used to perform the same function. Because of this, I will call the risk of infringement in algorithms low.

APIs

Both IDEs utilize standard Python APIs for debugging, including sys, traceback, and pdb. These APIs are commonly used by most Python-based debuggers. One difference, however, is that IDLE relies only on the standard library and does not allow internal debugging APIs to be used externally. In short, IDLE's debugger logic is embedded into IDLE's internal structure so it cannot be accessed outside of the IDE.

While Eric is also built using Python, it also implements PyQt for graphical interfaces. Eric's debugger APIs are built using PyQt's object-oriented structure. Compared to IDLE's APIs, Eric's APIs are more complex but allow for a greater separation between code for user controls and application logic.

Since both tools are based on publicly available and standard libraries, there is no clear indication of proprietary IDLE APIs being used by Eric. The APIs used by Eric are very different from those used by IDLEs, as Eric's are much more advanced. In terms of legal precedence, Google LLC v. Oracle America, Inc. in 2021 showcases a similar situation. In this case, Oracle claimed copyright infringement on Google's use of Java API declarations in the Android OS. The Court ruled that Google's use of Oracle's Java APIs were fair use because the reuse was transformative, limited and necessary, functional, and benefited the public. This precedence matches the occurrence of IDLE v. Eric because standard APIs are used for similar compatibility and functionality. The APIs used by both IDEs are publicly available Python APIs and all behaviors and functions used by them are functional, not creative. Due to these differences, the risk of infringement in APIs can be considered low.

Source Code

The source code for the IDEs shows clear differences in coding style, file structure, and complexity. While IDLE is written with a more procedural style minimizing abstraction, Eric is much more complex. IDLE uses simple control logic to manage execution flow and breakpoints. Eric uses an object-oriented approach and divides the code into classes and modules. The implementation of PyQt also requires more code for the graphical user interface, signal handling, and design of the interface. Lastly, there are additional features like variable watches and loggers that are not present in IDLE.

By manually comparing the source code for both IDEs, it is clear that there is no direct copying of IDLE's source code by Eric. The code structures are very different with variable names, method definitions and comments being unique to each IDE. In Computer Associate Int'l, Inc. v. Altai, Inc. in 1992, a legal precedent was formed distinguishing between literal copying and non-literal copying. The Court created the Abstraction-Filtration-Comparison test to determine what non-literal parts of source code are protected. Using the same test for this case, we can conclude that since IDLE and Eric use different coding styles, different file structures, function names, etc. there is no evidence of any infringement. Because there also appears to be no direct copying of code or software structure, the risk of infringement can be considered negligible to low.

Works Cited

“Algorithmic Program Debugging.” *Wikipedia*, Wikimedia Foundation, 29 June 2025, en.wikipedia.org/wiki/Algorithmic_program_debugging.

“Apple Computer, Inc. v. Microsoft Corp..” *Wikipedia*, Wikimedia Foundation, 9 May 2025, en.wikipedia.org/wiki/Apple_Computer,_Inc._v._Microsoft_Corp.

“Baker v. Selden, 101 U.S. 99 (1879).” *Justia Law*, supreme.justia.com/cases/federal/us/101/99/. Accessed 12 July 2025.

“Computer Associates International, Inc. v. Altai, Inc..” *Wikipedia*, Wikimedia Foundation, 13 Sept. 2023, en.wikipedia.org/wiki/Computer_Associates_International,_Inc._v._Altai,_Inc.

“Google LLC v. Oracle America, Inc..” *Wikipedia*, Wikimedia Foundation, 30 June 2025, en.wikipedia.org/wiki/Google_LLC_v._Oracle_America,_Inc.