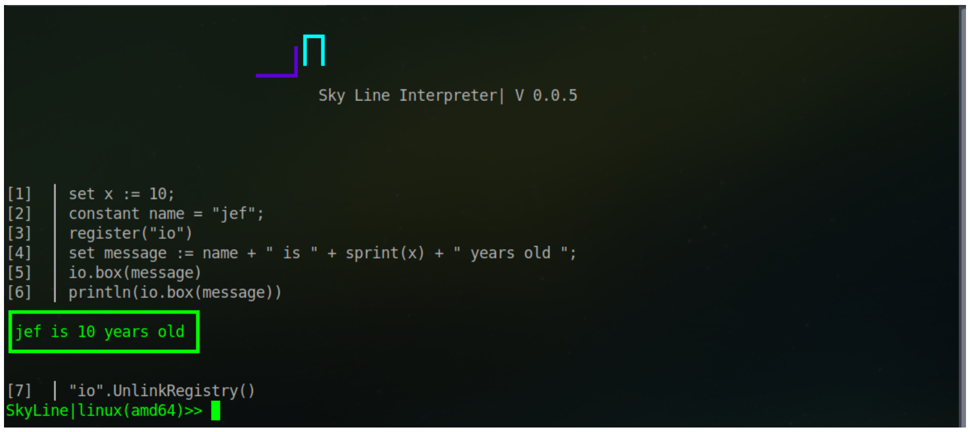


```
Sky Line Interpreter| V 0.0.5

set x := 10;
constant name = "jef";
register("io")
set message := name + " is " + sprint(x) + " years old ";
io.box(message)
println(io.box(message))

jef is 10 years old

"io".UnlinkRegistry()
SkyLine|linux(amd64)>>
```

Section 1 SkyLine - Benefits and basic intro to SkyLine

This document is a technical document for the SkyLine programming language and belongs to the SkyPenguin-Solutions development team and open-sourced community. If you have questions, concerns, or any other suspicions regarding this document please directly contact any admin within the official SkyPenguin discord server. For quite some time, the SkyLine programming language was a rebirth as previously explained of the Radical Processing Core (RPC) programming language. SkyLine then shortly became a living reality and is being worked on day in and day out. Because SkyLine became a public and open-sourced programming language, it is only appropriate that the development team discuss the learning issues with SkyLine, technical information about the language, a background for the language, and the issues SkyLine plans to address. The developers of the SkyLine programming language would like to note that before starting this document, this doc was written at approximately (7:13 PM Thursday, April 27, 2023 [EST]) and is a document for the current version in testing. Without further more to say, below you will find a section titled `An informal setting` which defines the rule sets, organization, description, and goals behind this document. This document will also include community arguments against the language with possible solutions to the language using prime example from other languages as well. These arguments are FORMAL COMMUNITY ARGUMENTS AND STATEMENTS. Arguments that kept place were not in any form of anger but were rather formal and professional arguments to give advice or questions about the language. This document contains a ton of information but the most beneficial information to you as a reader would be the arguments which may better answer your questions.

Note: The developers of the SkyLine programming language would like you to know that the language is in testing, its idea is going to be formed and designed throughout this document and the language will not be fully ready to show off its true side until the language is pushed into beta testing which will come when version 0.1.0 comes out. This language has an official repository, article, page, and discord dedicated to it as a prime part of SkyPenguin solutions and will be

constantly updated so the general items within this document will also change throughout the versions of this document.

Section [1] - Choosing SkyLine | A Road to Pros, cons, and Understanding

Choosing SkyLine as a programming language can be a significant decision for developers and cybersecurity experts. While it may not have the same level of documentation and third-party library support as languages like Python, it offers several unique advantages. One of the most significant benefits of SkyLine is its self-reliance, which can reduce the potential security risks of relying on third-party libraries. Additionally, SkyLine is designed specifically for cyber security, forensics, and mathematics, which can make it more effective for experts who work in these areas. Being open source and having an easy-to-manage code base also provides several advantages, including increased collaboration, transparency, and reliability. While it may take some time to learn and adapt to a new language, choosing SkyLine can be a valuable road to exploring its pros, and cons, and gaining a deeper understanding of programming languages and their impact on cyber security. In this section we will be going deep into different modules which talk about different reasons why SkyLine's idea might be better to form and why it can be a good workaround for most standard programming. This section will be broken up into modules that will discuss again specific topics like the idea, the problem skyline is trying to address, how it plans to fix this, the current testing state, the modification and flexibility skyline has, and why it might be a better choice for future automation.

Section[1] - Module (1) | SkyLine's prime idea

SkyLine as a language is attempting to solve the problem of dependency on third-party libraries in the field of cyber security. Many programming languages, including popular ones like Python, rely heavily on external libraries to perform tasks related to cyber security, forensics, and mathematics. However, this dependency can pose potential security risks, as vulnerabilities in these libraries can be exploited by attackers. SkyLine aims to be a self-reliant language that is built from scratch and does not depend on external libraries or frameworks to automate cyber security or mathematical-related tasks. This approach is aimed at reducing the risk of security vulnerabilities in code and giving developers more control over their code, especially in scenarios that require customization. Additionally, SkyLine seeks to provide cyber security learners with a language that can help them understand why programming languages exist and how they are useful in the cyber security world. You may already be wondering why third-party libraries are a problem. SkyLine is not trying to point out that using a verified or even random third-party library is bad but rather what goes into MOST third-party libraries. Let us take a look at the state of most third-party libraries today, not really targeting a library but rather just looking at the state. If you take a look at third-party libraries developed by third-party organizations and smaller companies and even teams you will often notice that the third-party libraries themselves require third-party libraries that also require third-party libraries. This can cause a LARGE amount of bloat in a program because of how much might not be used and might even conflict with specific states and actions that the program executes. An argument posted by an UnknownContributor to the statements of the projects mentioned that

`I think you don't need to reinvent the wheel on anything that is proven. I appreciate that there is no dependence on 3rd party but using a verified 3rd party is a no-brainer.`
This is a very valid argument, however, a formal rebuttal was made to this statement which is as follows.

1: **Third-party libraries can become a pain: ** When working with third-party libraries in the form of a bigger code base, bigger framework, professional environment, etc it can be annoying to constantly configure or build a program to constantly change the environment just for one tiny third party library. This way, SkyLine solved this issue with SLC a configuration engine and configuration language to help auto-tweak and auto-configure environments across multiple operating systems

2: **Third-party libraries often contain chains: ** When you look at most frameworks and the backend source code of most third-party libraries like someone's fork of gopacket you notice that those libraries also use libraries and those other libraries rely on TONS of third-party libraries. It is a chain that will never end. By the time you write A 20-line program for sending packets your binary file is thousands and thousands of lines long just because of a chain of third-party libraries being used to create a third-party external library.

3: **Environment and Operating System Support: ** When working with third-party libraries not only is configuring an entire code base to handle and support and rely (even on a verified) third-party library, operating system support is a pain in the ass sometimes. When it comes to something like go packet (google's packet-capturing library for Golang). It is a fantastic library and NOONE can argue with a Google engineer or team of Google developers especially hundreds of them and when everything is done using internal syscalls and socket calls. But when this library tries to operate on Linux it becomes a pain in the ass and will crash or fault without manually loading DLLs into your programs environment first

4: ****Third-party syntactic differences: **** When working with third-party libraries in one massive framework or one project it can not only be a pain to manage but you have to re-learn a whole new syntax each time you want to use a third-party library and that is a TON to re-learn. Because SkyLine plans to automate most things cyber security, you only truly have to learn it once as a language rather than learning it each time you want to use a feature or library.

This argument can be backed with a representation of a third-party library often used by CTF players or offensive cyber security developers or experts who rely on the Python programming language. This third-party library is known as PwnTools and like any or most third-party libraries, it also relies on third-party libraries which back the point of chains. PwnTools in itself uses

Pycryptodome, python-ptrace, capstone, ropgadget, pyelftools, pwntools-binutils, pyserial

All to execute the same task and to automate just a few more lines of code when the authors could have also automated much more in terms of itself. Not to mention as talked about before it still takes tweaking and installing and upgrading existing packages to do so. Mainly the idea SkyLine as a language tries to solve is the issue of using chained third-party libraries to execute a specific task even if it is a little one. This issue also exists in many other programming languages that do not have a rich standard library. **With this goal in mind, SkyLine also has an end goal to educate upcoming cybersecurity experts on how important it is to study and learn about code to make utilities or things yourself. Not everything (as the developers can admit) should be written by someone and deployed into production, however studying and reading into other people's code and building something self-reliant as SkyLine is in itself is HEAVILY encouraged. The language not only hopes to automate thousands and thousands of lines of library code while being self-reliant but also educate cyber security pioneers to better their future as developers!**

Section[1] - Module (2) | What makes SkyLine's goal different?

The point of SkyLine is to teach upcoming security experts a whole new realm of ideas and how to better themselves while also providing a solution to a problem like constantly configuring environments to install third-party libraries. However, the real deal is why and how will SkyLine's idea truly help and inspire people to be better developers or how will it even be better than most current popular languages out there? Well, truth being said, SkyLine can not even in the slightest chance do that right now as its proof of concept is VERY minimal. However, SkyLine's goal is not to be the next popular programming language to hit the market, its end goal is to not manage systems, develop web applications or be the next industry titan but rather silently explain and kill off specific sets and arguments which promote the use of third party libraries. Doing so will hopefully encourage people to make their libraries instead of relying on third parties which mostly always involve chains. Now sure, you can say the same thing against a programming language and make the argument that " a programming language can also be vulnerable to many attacks, it can also be taken down, ripped apart, abused and can also have flaws ". That is a very formal and amazing argument to form but can easily be argued. The following points argue on the side of third-party libraries which may help you better play out the logical.

ARGUMENT 1: While it is true that relying purely on third-party software or libraries can post other security vulnerabilities and related issues, third-party libraries can also provide a significant benefit in terms of code reuse and productivity. Not to mention code and most libraries are open sourced which means the community can also work with those libraries and make improvements.

This is where SkyLine comes into play. SkyLine is not telling you to eliminate all third-party libraries but rather telling you that programming languages are more likely to be kept up as they have bigger sites of communities, can be used more, and are more likely to be tested by the community of developers and experts using the language and its libraries or code bases. Programming languages are a completely different topic despite them doing the same thing as third-party libraries. Let's look at Golang versus a library for golang called Goquery. Goquery is a very popular and well used library in golang for better and more formal parsing of URLs, however, it also relies on other third-party libraries like its custom libraries or other third-party packages made separate from the language of Go but made by the Go developers like net/html, text/transform, text/encoding and unicode/norm. Go query in itself is a very handy library but it is so small that it is quite easy to recreate yourself using Golang's standard library. Programming languages themselves (most of the time) do not rely on third-party libraries which is the major difference between Go as a programming language and a third-party library like Goquery. Goquery can automate things easily but so can Go due to its strong standard library. The point that SkyLine is trying to make is that everything can be made without installing 9 third-party libraries and overly bloating your system or flooding your system with packages that are out of date or broken or waiting to be updated or flooded with bugs etc. Again, you can easily make the argument that programming languages have the same issues and they do but the major difference is most likely if a programming language gets out there it is developed by a team and foundation and constantly kept up and never really archived or ever really becoming legacy as its updates stay with the current side of things.

Section[1] - Module (3) | What makes SkyLine helpful to security experts?

There are many prime aspects as to what would make SkyLine in the further future helpful to cyber security experts based on its prime idea and concept which is in the process of being tested and experimented with.

- **Self-Reliance:** The biggest thing on this list will have to be the self-reliance factor. SkyLine is built from scratch and does not depend on any third-party libraries or frameworks which means that the code written in SkyLine is all self-contained and is less likely to be impacted by external dependencies because of the self-reliance it holds. For instance, let's take the example of the PwnTools library in Python which is commonly used by cyber security experts for reverse engineering and exploit development. As previously discussed, PwnTools is an amazing library and powerful library and even has a pretty large codebase but that also means that it can be difficult to audit for security vulnerabilities. Furthermore, if a vulnerability is discovered in PwnTools or any of the third-party libraries PwnTools relies on that also means that it would impact a large number of applications and servers or even CTFs that would use that library. In contrast, SkyLine's self-reliant approach reduces the surface area of the code that needs to be audited for security vulnerabilities. Additionally, like PwnTools if a vulnerability is discovered in SkyLine, it would only impact applications that depend on it but not affect a large number of libraries that depend on it like it would In the case of PwnTools.

- **Modernistic and stealthy approach:** SkyLine's design philosophy is to take a modernistic and stealthy approach to cyber security tasks. This means that the language is designed to be efficient and effective in scenarios like penetration testing, where the goal is to identify and exploit vulnerabilities in a target system. For example, SkyLine includes features like easy configuration and image injection, and forensics utilities, which can be useful for conducting analytics on a program or even your payloads and injectors. Additionally, SkyLine's syntax is designed to be concise and expressive, which can reduce the amount of boilerplate code that needs to be written, and make the code more readable.

- **A prime goal and focus on one topic:** Most programming languages that were designed were designed for a reason but in the end became a general purpose programming language and became used for a multitude of tasks. General-purpose programming languages are very helpful to the world of programming, but it also takes the focus away from so many possibilities and topics that can aid the language. SkyLine aims to focus on two very important topics, cyber-security, and mathematics. With a high focus on these two topics and a high focus on the backend development, this can prevent SkyLine from going off the wrong track in later stages. This also means that the idea of SkyLine will continue to progress more and more as time goes on. As discussed before SkyLine is designed to be a highly specialized language for cyber security and mathematics. This means that the language is optimized for these types of tasks and includes features and functions that are specific to these domains. For example, SkyLine includes built-in functions for cryptographic operations, forensics operations, IoT operations, and file operations. It also plans to include functions for performing mathematical operations like automating search algorithms, rating algorithms, data sorting algorithms, and more. These features can make SkyLine a more efficient and effective tool for cybersecurity experts who need to perform these types of operations frequently.

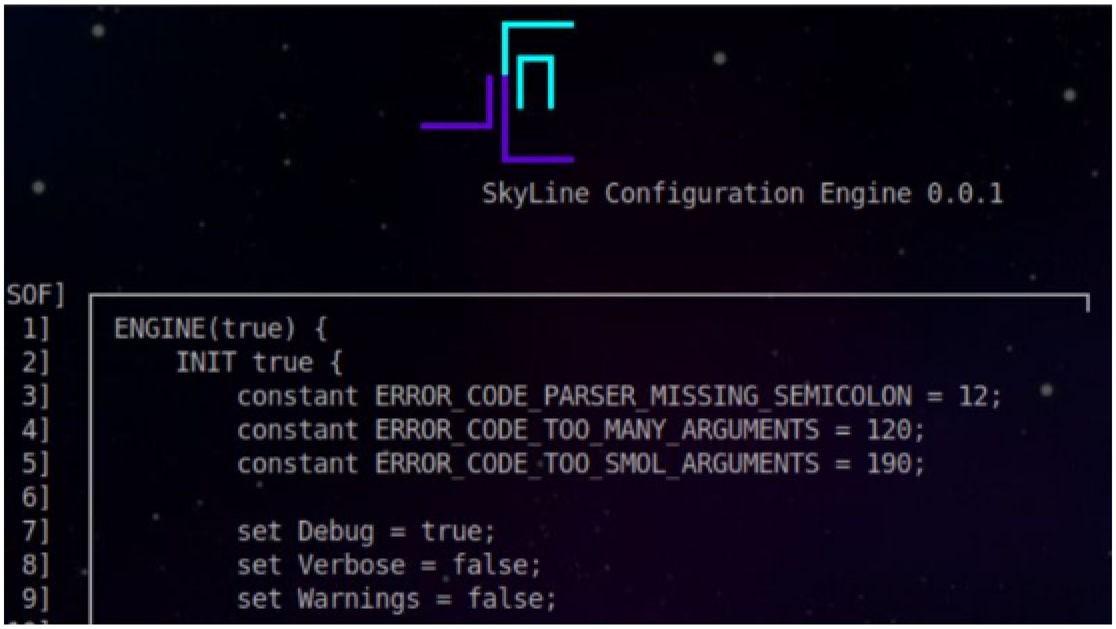
- **Education:** One of the second most prime aspects of SkyLine as a programming language is to inspire creativity and to educate the people that use the language. Currently, the backend is not the best representation for this but when the code and language become much more organized, the backend will be designed for people to understand. This will allow people to be able to scroll through the source files and instead of seeing a brick like the following shown below.

```
static asdl_seq *_loop0_37_rule(Parser *p);
static asdl_seq *_loop0_38_rule(Parser *p);
static asdl_seq *_loop0_39_rule(Parser *p);
static asdl_seq *_loop1_40_rule(Parser *p);
static asdl_seq *_loop0_41_rule(Parser *p);
static asdl_seq *_loop1_42_rule(Parser *p);
static asdl_seq *_loop1_43_rule(Parser *p);
static asdl_seq *_loop1_44_rule(Parser *p);
static asdl_seq *_loop0_45_rule(Parser *p);
static asdl_seq *_loop1_46_rule(Parser *p);
static asdl_seq *_loop0_47_rule(Parser *p);
static asdl_seq *_loop1_48_rule(Parser *p);
static asdl_seq *_loop0_49_rule(Parser *p);
static asdl_seq *_loop0_50_rule(Parser *p);
static asdl_seq *_loop1_51_rule(Parser *p);
static asdl_seq *_loop0_53_rule(Parser *p);
static asdl_seq *_gather_52_rule(Parser *p); // cpython(parser.c)
```

Instead, SkyLine will try to make the language as human-readable as possible to inspire people to create a much stronger code base and understand backend design and coordination while also being able to easily understand how programming languages work as a whole. SkyLine can be a valuable educational tool for security learners who are interested in learning how programming languages work and how they can be used in the cybersecurity world because of this. Not to mention SkyLine's design philosophy encourages developers to think critically about their code, which can be a valuable skill for future cybersecurity professionals. For example, SkyLine's self-reliant approach can teach learners about the potential security risks of relying on third-party libraries and the importance of auditing code for security vulnerabilities. Additionally, SkyLine's specialized focus on cyber security, forensics, and mathematics can expose learners to the types of problems that are common in these domains and help them develop specialized knowledge and skills.

To sum this module up, SkyLine's self-reliance, modernistic and stealthy approach, specialized focus, and educational potential can make it a compelling alternative to Python for cybersecurity experts. While Python has many powerful libraries and applications, its dependence on third-party libraries can create potential security risks and reduce the control that developers have over their code. SkyLine's self-reliant approach can reduce these risks and provide developers with more control over their code, while its specialized focus can make it more efficient and effective for cyber security tasks.

Section[1] - Module (4) | Why might SkyLine be an alternative to something like Python?



```
ENGINE(true) {
    INIT true {
        constant ERROR_CODE_PARSER_MISSING_SEMICOLON = 12;
        constant ERROR_CODE_TOO_MANY_ARGUMENTS = 120;
        constant ERROR_CODE_TOO_SMOL_ARGUMENTS = 190;

        set Debug = true;
        set Verbose = false;
        set Warnings = false;
```

PROTECTION NOTICE BEFORE SECTOR STARTS -

As was stated beforehand, we are comparing a programming language that fits best to SkyLine and the organization associated with SkyLine is in no way, shape, or form trying to say in itself it is better than one of the most popular programming languages but rather a way to compare and contrast its idea to one of the more popular languages of the century. It is important to note that when comparing two programming languages, it is essential to approach the topic objectively and without any bias. Skyline is a programming language that focuses on the areas of cyber security, forensics, and mathematics. It aims to provide a specialized tool that can be used for these specific purposes, and therefore cannot be directly compared to a general-purpose language such as Python. The purpose of this document is to provide a comparison between SkyLine and Python in terms of their respective strengths and weaknesses, particularly in the areas of security, efficiency, specialization, and control. This comparison is intended to assist professionals in determining which language is better suited for their specific needs. It is also important to note that the authors, programmers, organizations, sponsors, developers, companies, owners, founders, writers, educators, and communities associated with both languages are valued contributors to the field of computer science. This document is not intended to discredit their efforts or diminish their contributions in any way. SkyLine is an open-sourced program, meaning that anyone can access the source code and make modifications or improvements to the language. This open-source nature allows for greater collaboration and community involvement in the development process. Additionally, the support for plugins in SkyLine allows for the creation of custom functions that can be integrated directly into the language's environment, further expanding its capabilities. It is important to approach the comparison of programming languages professionally and objectively. While SkyLine and Python have different strengths and weaknesses, both have their place in the world of computer science. The purpose of this document is to highlight the specific areas where SkyLine excels, not to discredit or diminish the contributions of Python and its associated communities. SkyLine does not plan or even have the idea of copying, mocking, or participating in activities that can harm the language’s reputation but rather want to improve. THIS NOTICE WAS TO ADMIT TO THE RESPECTIVE GOALS OF THIS DOCUMENT AND THAT THE COMPARISONS MADE IN THIS DOCUMENT SHOULD NOT BE HELD AS A ‘dig’ AT PYTHON AND SHOULD BE COUNTED AS A CONTRIBUTION TO THE RESPECTED SETS.

Without further to say or note, let's move on.

SkyLine as talked about many times already has a reason for its existence, to better understand the world around itself and to address problems in that world. To do that, we must provide some reason as to why SkyLine might in the future become a better alternative to various scripting/general-purpose programming languages. In this next list, you will see some good pointers as to why SkyLine might be considered.

- **Security**: As talked about before in SkyLine’s prime and core concept, one of the major concerns was security for relying on third-party frameworks or libraries. When developers choose to use these third-party libraries, they are essentially trusting that the code within those libraries is secure and free of vulnerabilities. This can also be said about SkyLine but unlike third-party libraries, SkyLine’s community and developers strive to put out the most technical, informal, and educational articles, documents, and announcements regarding the security state of the language and its problems (why this document exists). However, this is not always the case. When we talked about PwnTools using a library called **“pycryptodome”** there was a reason that was brought up, simply to back the point of third-party libraries. In 2020, a vulnerability was discovered in a popular Python library called **"pycryptodome"**. This vulnerability could have allowed attackers to execute arbitrary code. Additionally, third-party libraries can become outdated or unsupported, leaving vulnerabilities unpatched. With SkyLine, developers have more control over their code and can be sure that their code is secure since they are not relying on third-party libraries and also keep up with the frequent releases and updates as well as documents and research that will be put into the language.

- **Efficiency**: SkyLine has quite an interesting design for an interpreted language and despite being in the testing phase it has a pretty decent design plan. Like the engine, SkyLine will plan to use algorithms and regex to pick up and detect specific tokens during lexical analysis which will make the language faster on top of the idea of a byte code compiler tied to a VM known as SVM (SkyVM). This virtual machine will also make the language much more faster and efficient at executing and parsing specific units or blocks of code. By eliminating the need for third-party libraries, SkyLine can reduce the amount of code and processing required to execute a given task. This can result in faster and more efficient execution times, which can be critical in time-sensitive operations. Additionally, since SkyLine is specifically designed for cybersecurity-related tasks, it may be optimized for certain operations that may not be as efficient in Python. This design also ensures that SkyLine will keep up with modern code standards and modern developer opinions. Not necessarily taking everything from the community but making a good side and leaning towards the community.
- **Specialization and non-hybrid focus:** As talked about in the prime and core concept and idea module, SkyLine is only planning to focus on two major topics which include cyber security and mathematics. But what exactly makes this so special and a static focus? Because SkyLine carries this design, it allows it to offer a much more unique advantage over a general-purpose programming language like Python. This is again due to the factor that SkyLine will have only a prime end goal and prime focus on these topics allowing for a much more efficient backend and effective set of code and libraries as well as better tools and capabilities for performing specialized cybersecurity-related tasks. A good example is SkyLine's current support for forensics as its forensics module is one of the most worked-on modules right now. This carries functions, code sets, variables, and even demos that can be particularly useful for cybersecurity-related education or in cybersecurity-related applications such as frameworks and other various sets of tools. In addition to that statement, SkyLine's specialized focus can lead to a more streamlined and targeted approach to solving specific problems that are found within the future or even currently as more libraries and frameworks are developed. One of these main use cases for SkyLine is that because SkyLine is made directly for one or two specific topics and has a deep focus on this, it is best to use a language that would be directed towards that topic rather than a general-purpose programming language. For example, R-Script is commonly used for data analytics and mathematics because of the standard functions and calculitic functions R has built into it. You would not use a general-purpose programming language like JavaScript for something like calc or data analytics, you would rather use R

because R was built for that topic. Because SkyLine is built for cybersecurity-related topics, this can result in much faster development times, write times, improved code quality, and more robust and reliable solutions. The idea and the core concept of SkyLine's standard library to be directed towards branches of cyber security like cryptography, digital forensics, web security, binary and app security, and mathematics mixed with Golang's fantastic library and the amazingly rich standard library will make SkyLine truly great for those topics. Overall, SkyLine's specialization in cyber security, forensics, and mathematics can make it a valuable tool for experts working in these areas. By offering more specialized tools and capabilities, SkyLine can lead to more efficient and effective code, faster development times, and more robust and reliable solutions in the field of cyber security.

- **Native Interaction with Go:** One of the prime advantages of SkyLine is that it is developed using the Go programming language. The go programming language is a widely used, fast, compiled, statically typed programming language that has one of the richest standard libraries out there. Golang allows the idea of using plugins with their standard library called 'plugins'. This allows developers to compile programs as ***.SO*** (Shared Object) files and plug them directly into a given project. Due to this amazing feature, SkyLine allows you to interact directly with its backend to create plugins for your project and in future versions can even use the SLC-E (SkyLine Configuration Engine) to better load, customize and interact with modules in Go. The core benefit to this is allowing golang to work with SkyLine in your projects, so if you ever needed to write a project but needed to go for a specific reason then you can use SkyLine as a prime language and plug go files that interact or are interlaced with each other directly into the backend of your project. This makes it much easier to fit and tailor towards the developer's projects and also make sure that specific functions can be run and called locally within SkyLine. Because of this, and because Go also has support for CGO you would also be able to tie Go, C, and SkyLine into your projects with just the right configuration. Not only does this pose a benefit to your project and the flexibility but it also can represent how well SkyLine can be integrated into other projects.

- **Control:** The level of control that SkyLine offers to developers is a key advantage in specific situations where customization is absolutely needed. Since

SkyLine does not rely on any third-party packages, heavy functions, hardware-reliant or operating system-specific modules even self-reliant modules that are hardware-reliant, developers have complete control over their code, the environments its being run through and even the state which allows them to tailor it to specific situations and scenarios. In contrast to SkyLine, Python relies heavily on third-party libraries even to make a simple GET request you need to download a third-party library which can sometimes if not most times limit the control that developers have over their own code forcing them to either use the library of spend the next year of their project writing their own libraries and classes and modules just for that specific need of customization. While these libraries and frameworks can make development faster and more efficient they also introduce potential vulnerabilities and dependencies that developers can not always manage plus the amount of environment configuration it takes to do such things or to install such libraries or packages. Even then, developers still have to constantly and carefully keep up with their project and the dependencies which may not work the same across multiple operating systems and architectures due to the developers of the dependencies. Additionally, using third-party code and libraries can sometimes limit the ability to customize and fine-tune the code to meet specific needs or specifics to your current project. The advantage of having control over the code in SkyLine is that developers are not restricted by the limitations of pre-existing libraries or frameworks. Because of the backend, developers can create custom algorithms and tools that are specifically tailored to the needs of the project. This is especially important in the field of cyber security where a one-size-fits-all approach is not always appropriate, especially on the targets. Now, here is where a formal drawback and fair argument to this paragraph. One main and primary issue right now is that SkyLine is still in a testing phase as explained a hundred times already, this means that it has a much smaller community than Python and does not nearly have the same technical sides or even similarities in terms of performance due to the nature of the language and the size of its development teams. This also means that there are fewer standard libraries and tools available for developers to use in their projects. However, this can be seen as an advantage for the time being, as it encourages developers to create their own custom code and tools which can result in efficient and effective solutions. Summing up this point, the level of control that SkyLine offsets to developers is a key advantage in certain situations, especially those where customization is necessary. While Python and other languages have a vast range of pre-existing libraries and frameworks available, the level of control that they offer is limited to some extent by those dependencies. In contrast, SkyLine offers a level of flexibility and adaptability that is unmatched by other languages, allowing developers to create tailored solutions to cyber security challenges.

- **Open Sourced: T**his is not really a direct argument against Python or most other languages because those languages are also themselves open sourced but it is a good point to make. Here is the issue with open-sourced development when it comes to SkyLine. One thing SkyLine has planned is to work with both defensive and offensive security experts in the cyber security field. However, having an open-sourced framework especially a programming language that has a backend filled with signatures and different payloads is just a gateway to being blocked. SkyLine will remain open source but in later versions has plans to implement much more heavily restricted closed source protection features however a majority of the language including 99.9% of the standard libraries will be open sourced, and 100% will remain open sourced UNTIL it becomes a problem. However, despite this tiny annoying side of things below you will find some pointers that talk about the benefits of SkyLine being an open-sourced programming language.

    - **Community-driven development:** Open source allows for a community-driven approach to development, where developers from around the world can contribute their knowledge and expertise to improve the language. This means that the language can be developed and improved much faster than if it were being developed by a small team of developers working in isolation.

    - **Faster problem-solving:** With an open-source code base, the community can easily identify and report bugs, security issues, and other problems that arise. This allows developers to address issues much faster than if they had to wait for users to report them. In addition, with a larger community of developers working on the language, there are more people available to help solve problems and suggest solutions. It also helps developers of the project better map out prime issues with the language and even helps the developers better understand what the community prefers in their own style to better work with SkyLine and to make it much more efficient for cybersecurity-related tasks.

    - **Greater transparency and trust:** With an open-source code base, users can see exactly how the language works, and can even modify it to suit their specific needs. This increases transparency and helps to build trust among users, as they have more control over the language and can ensure that it is secure and reliable. When working in fields like mathematics and cyber security it is absolutely imperative that developers also have an outstanding level of trust with the community to provide the most accurate and successful results. When it comes to mixing a programming language with these two prime fields, that trust now needs to be quite perfect and the understanding and fluency need to be direct with the community in order to provide and support the language's idea and what it plans to change.

- **Flexibility and customization:** With an easy-to-manage code base, developers can easily modify and customize the language to suit their specific needs. This allows them to tailor the language to specific use cases, making it more efficient and effective. Not to mention given how simplistic the language plans to be even in the further future and how it plans to educate people, it will make it much easier to fork, edit, or make your own version. In this case, SkyLine will be much easier to understand especially in later versions.

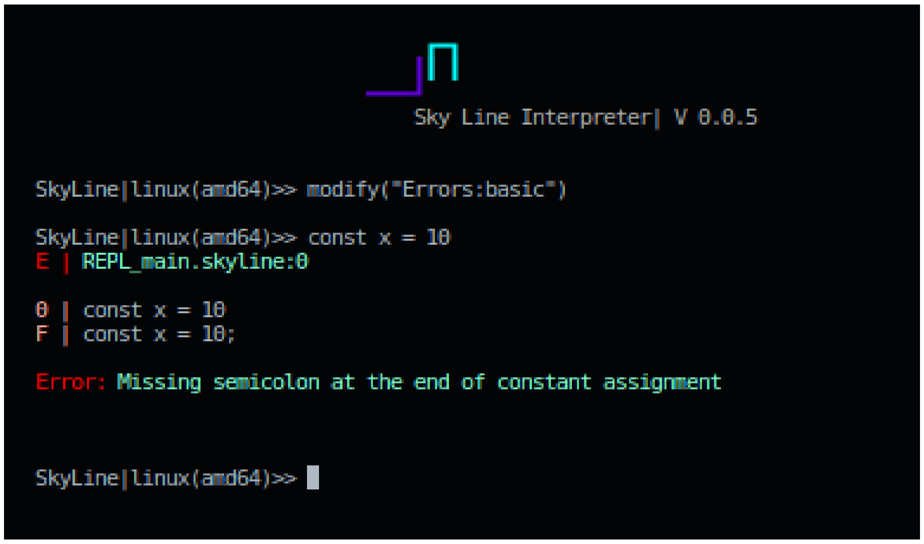
- **Faster adoption:** When a language is an open source, potential users can see the code base and how it works. This can help to increase trust and transparency and may lead to faster adoption of the language. In cybersecurity-related topics, this will also help with how well-documented the code will be especially linking to other resources and prime examples and people that helped write those examples. For example, in the forensics utilities library, there is a well-documented explanation as to how you can use SkyLine to inject images with malicious code and manage to exploit vulnerabilities using that image.

Concluding this section, while Python is a very widely used and versatile programming language, SkyLine provides several advantages in the field of cyber security. The language's focus on security, efficiency, specialization, and control makes it a strong choice for developers who require a self-reliant and secure programming language. It is also important to note this again SkyLine and Python are two completely different languages but they can still be compared in some aspects such as modularity, customization, frontend work, backend work, and much more along that side of things. However, keep in mind per the notice and note before this section, SkyLine, and Python will be used for totally different aspects of the language.

Section 2 SkyLine - Downfalls and Current States of SkyLine

This would not be a proper document if I did not argue or fight against SkyLine as a programming language. I have found that being honest with the issues in SkyLine will brand it better and give people a good reason to deny the use case right now. Again, this programming language has a good set idea and a good range of execution but there are MANY issues still existing in the language as it is still being experimented with and is not a programming language that will be dedicated to just skipping and ignoring the issues. So this section otherwise known as Section 2 will talk about the current state of the project, where it is right now, the current version, issues, problems, glitches, bugs, or current existing problems that stop SkyLine from fulfilling its end goal.

Section[2] - Module (1) | SkyLine's Primary Issues



```
                              Sky Line Interpreter| V 0.0.5

SkyLine|linux(amd64)>> modify("Errors:basic")

SkyLine|linux(amd64)>> const x = 10
E | REPL_main.skyline:0

0 | const x = 10
F | const x = 10;

Error: Missing semicolon at the end of constant assignment


SkyLine|linux(amd64)>>
```

SkyLine's error system has plans to become one of the most verbose error systems alive inside of the interpreted community while also being the most customizable, modern, stealthy, and different error systems out there. However, with the current state of SkyLine not being organized and not being properly worked on (which will be changed VERY soon) the error system has many issues. Below I have mapped out a good example of the issues residing within the error system.

- **Great Idea Horrid Execution:** One of the prime issues was the way the error system was designed. The error system had three different output formats but all followed the same general structure. All systems could be modified, all systems would use color code to better format and formulate output, all systems would use the native SkyCode system which was a set of error codes and organization for SkyLine and all would follow the same direct modification standard. Halfway into the development of SkyLine, the error system stopped being a prime priority due to something else popping up. The original idea was to use `modifiers` which will be talked about later, to modify the state of the error system. Upon further development and idea, the language became more and more unstable each time and became worse off every single time the modifier was added to the state. The error systems development then later froze and had to go back onto the drawing board and never got finished. The error system right now is all over the place, sometimes you will receive an error tree, other times you will receive one giant error system and one giant error code, a developer error, sometimes an inaccurate error, and sometimes an error system that was never modified but acts as if it was modified. This renders the language useless if you are not a developer and do not make your code top-level perfect. Now, it is in the plans that SkyLine will be rewritten after its 0.0.5 release and will hang on that release for a few months until SkyVM can be put into place, SkyBC, SkyCE can be put into place and other various systems such as the error systems, syntactic rule sets and more can be replaced and changed. The error system is one of the most vital systems in a programming language, it guides the programmer into learning more about the language and eventually mastering it. Due to these failures and due to the mass amount of issues with SkyLine's error system, in the current experimentation phase it is not an easy way in or an easy way out. Even for developers, it requires modification of the language before you can write projects or code. Right now, to you, the reader, it may be confusing as to why this document is even glorifying the language they immediately talk bad about the language. This is because the purpose of the document is to help people understand the concept of the language, and why it exists, answer any questions about its concepts, and discuss its current development states and the concepts it wants to bring to the world. In order to properly make a programming language, it is important to note out the cons and talk bad about the language and admit its current faults from a developers perspective especially if you have been working on it for so long and have had to rewrite it multiple times due to horrible design plans and no end goal. In the case of the error system, this is a top priority for the language as it is what makes the language go around.

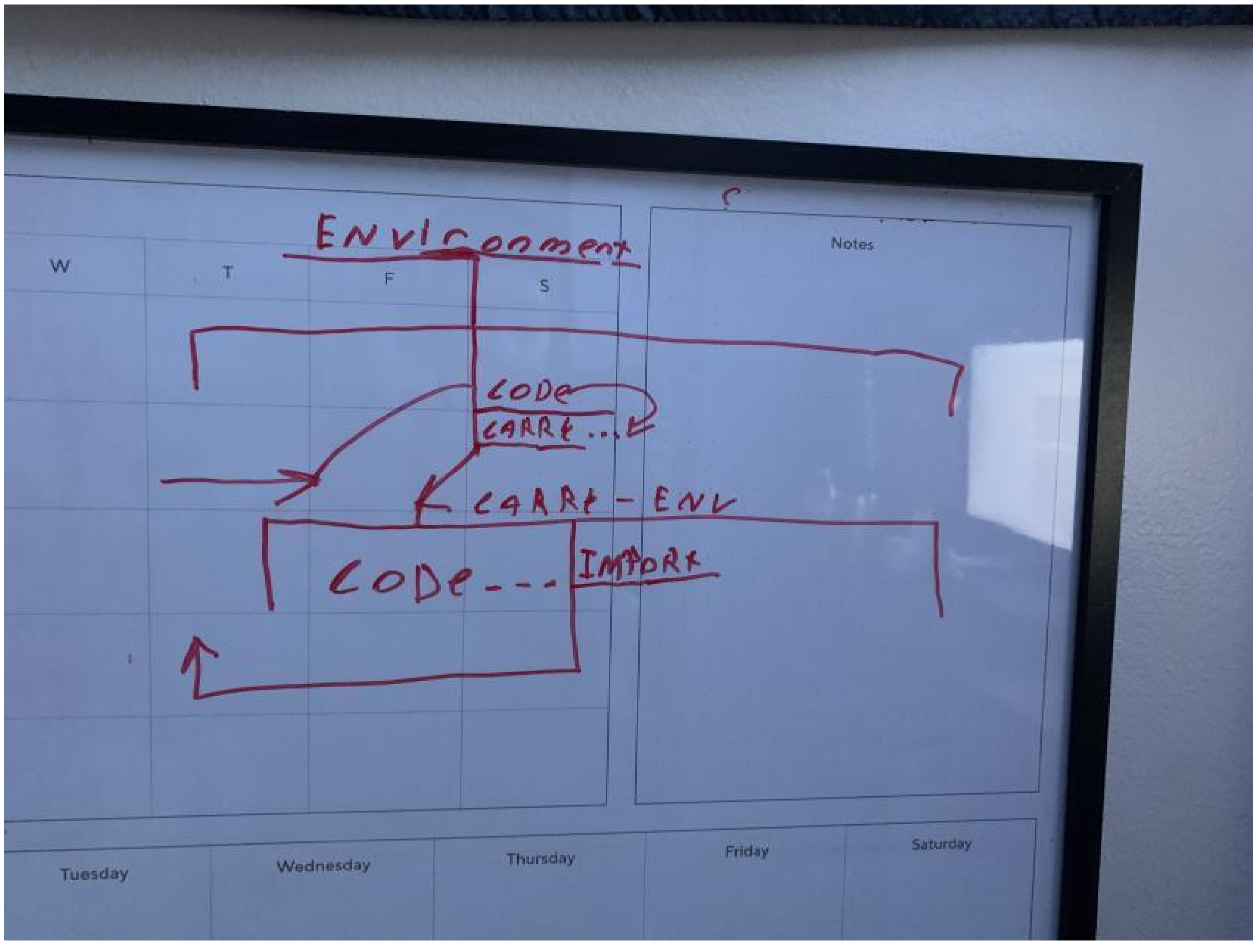
- **Horrible design:** The language is in a very weird hybrid syntactic state, you can define constant variables as const or constant, assign variables with let, set, cause, or allow, and also define switch as switch or sw and default as default or df and case as cs or case. Right now the syntax of the language, design of the language, design of the directories, design of the documents, design of the libraries, design of SkyVM, and the design of the engine are all one giant cluster of nonsense to anyone who has not actually made this language. SkyLine is supposed to make it easier for people to understand how languages work internally, but due to the confusion about what the language even is in terms of source code right now it kind of kills that whole side of it. Again, SkyLine is still in testing, each concept is individually worked around and worked with and isolated into a simulated environment within the language and isolated from everything else. The design to the language needs to be mapped out so much more so again once 0.0.5 is released the language will hang for quite a bit and will change a TON after that release. Once version 0.0.5 is released and the new version is ready it will be changed so much that it will quite literally skip straight to version 0.1.0 which will be the first primary testing stage. This means importing, file systems, file checking, symbol checking, symbol and engine checking, engine verification, customization, web servers, documentation, design documents, core concepts, core design, core libraries, standard functions, invokes, variables and the syntax will be fully designed by that version plus SkyVM will be fully implemented as well which will be quite fun and interesting to go down the road of.

- **Syntactic Rules:** The rules of SkyLine are very ungoverned and do not really fully exist, sometimes the error system can not even catch an error. This is an obvious issue when working with the language because if you go to forget a semicolon sometimes the error system will tell you that a prefix parse function or infix parse function does not exist, when you are missing a semicolon on specific statements then the error system will snap into place. Another problem that was also figured out was that underscores despite being supported in the consumer functions do not actually work fully, the lexer or scanner will just freeze and forget how to handle them. In other cases I have found the entire file panicking, the program crashing due to empty or no statement calls, the program completely losing its mind for forgetting a simple token, standard library functions half relying on user input and others not, etc. This list can and will continue forever. This gives the developers another prime reason to rewrite the language and why it would be a good idea to continue to rewrite everything but this time rewrite it well and better, take it much more seriously as SkyLine now has an end goal- to educate and to provide. When the rules in the syntax are not even clear it is a good idea that the language is really not going well and that the issues within the language need to be solved before progressing to fulfill the idea of the language. The developers have also found that sometimes you can place a semicolon in places you do not need it and get an error, and vice versa for not placing it where it is required and either getting passed or getting a broken logic error in the parser or evaluator. This is why the step for the language is a bit of a pain and it is not yet even closely ready for the base design to be put into production. Of course, the language may be used to write a specific framework or to better understand data and information but right now it should not be used in anything production wise even for your own programs.

- **The standard library:** The point of SkyLine is to better work with a more structured standard library for cyber security professionals. Right now that standard library in itself is confusing and does not have a primary point to have the naming convention it does. Right now SkyLine needs to focus on its prime systems and then finally focus on the standard library last but it would be something worth noting and worth actually looking into once version 0.0.5 is released and put out there. The standard library NEEDS upgrading and the standard object call functions all need to be working perfectly and can not work the way they want to right now. The invoke ideas and functions are great but it might be best to take a debloating approach and not register 30-string object call functions.

- **Speed and Performance:** The language is not generally slow, in some operations it does 2 times better than Python in specific file operations as it has been tested to be a 0m0. 0.0.2s difference. The speed of the language sometimes varies on the style you are writing or using which is something that needs to be fixed. Instead of using a base switch on the lexer, the language should be using regular expressions and other sets of algorithms to better work with the tokens it is categorizing. This will make the language much more performant and will allow the language to be quite flexible and changeable in the further future especially if it is reading from a list. The SkyVM and SkyBCC or SkyLine Byte Code Compiler will also be considered to be added because if built correctly the byte code compiler and the virtual machine will increase performance quite well and very interestingly. This will also allow for specific syntactic rules to be caught so much faster and be much more accurate.

- **Incomplete functions:** The SkyLine programming language is still in its very beginning testing phase and has yet to leave it. There are multiple forms missing from the language as a whole such as boolean algebra, types such as [all, int32, int64, int8, float64, float128, complex128, byte, etc] not to mention symbols and operators are also missing and regex has yet to be implemented so have regex comparison operators and else if statements while also not allowing for the use case of Func definitions without becoming a closure. There is a TON of issues with the incompletion of SkyLine's syntax. When the language is rewritten allowing multiple types, multiple different forms of functions, and a better state of the language will be so much more beneficial. This means that the language will also undergo MAJOR changes. Of course, the base syntax will remain the same but it might be easier to also implement classes, modules, and type structures as well as something much better for managing systems and managing the systems there. There will also be a complete overhaul to the modifier statement corresponding to the modification engine. That is something I also wanted to note which will be discussed later but the modifier might be depreciated because the engine already does all of that for you and will just ruin the state of the language if you use modify() in a complex way. It is much better to import data via engine already so why not just eradicate it? There was also a concept in SkyLine known as **`carry|""|`** which would stop parsing current files and just instantly execute each and every function or variable that is called but not import the data into the environment. It would rather use the current state and just execute the file in a separate internal environment internal to the environment already created. Below is a whiteboard example of what carry will do when a carry symbol is detected.



- This whiteboard shows the basic environment and some code, under the code is a carry statement. If the carry statement exists then it will jump to creating a new environment known as SL-Carry-Env-SecondScope and will then parse and run its own code. If import is detected it will hop out of that environment to check if the data of the imported file exists in the previous environment by exporting the hash and then if not it will jump back and import its own data creating a internal environment to store that data in. This concept might be added back but became a problem as it created its own environment and could not be tied directly to the prime language's environment. However, the issue was not primarily that it was spawning its own environment but rather the factor that it never actually destroyed the environment until parsing the main file was done or the main project. This means if you use `carry` 14 times, the language will gradually decrease in performance and speed because of the amount of copied environments that are never destroyed.

To sum this module up these issues marked a whole new realm for the language and really show that it is not even ready to go on to prove its ideas. While the language is already in itself very very fast, it has some problems going around the backend development of the language mostly coming from null or nil values trying to be parsed. This is something that was heavily thought about and ended in a conclusion that said the developers should just up and re-design the entire language and actually do everything differently this time. Add a virtual machine, a byte code compiler, much more optimization, debug features, and a better error and self-healing code system. This also means that the language in the further future will have much more complexity added to the backend in hopes to make the language much more accurate than what it is right now. The current issues with the SkyLine programming language are exactly that, its error and syntactic rule systems. These systems are about the most vital to the language itself outside of prime components because they really tie the user experience together and can also better help developers debug code when necessary.

– DOCUMENT END

SkyLine is still in testing and this document was just a demo and related to the language itself and proving its idea as well as further information. This language along with the document will be constantly updated past this brick.

- 1:58 PM Sunday, April 30, 2023 (EST)