# Blockchain Honor Degree Sem VII

# HBCC701 : Blockchain Development

## Module - 2 : Blockchain Programming (8 Hours)

Instructor : Mrs. Lifna C S



# Topics to be covered

- Types of Blockchain Programming
  - Solidity,
  - GoLang,
  - Vyper,
  - Java,
  - Simplicity,
  - Rholang,
- Comparative study of different blockchain programming languages,
- Decentralized file system-IPFS.

***Self-learning Topics:*** Emerging blockchain programming languages

# Types of Blockchain Programming


Courtesy : Biz4Solutions

# Types of Blockchain Programming

1. **Solidity:**
   - Specifically designed for writing smart contracts on the **Ethereum** platform.
2. **GoLang (Go):**
   - A **general-purpose programming language**
   - used for building a variety of applications, including blockchain systems.
3. **Vyper:**
   - For writing **Ethereum smart contracts**, known for its focus on security and simplicity.
4. **Java:**
   - A **widely-used programming language** that can be employed for blockchain development,
   - particularly in platforms like **Corda.**
5. **Simplicity:**
   - A blockchain programming language designed to improve the **security and efficiency** of smart contracts,
   - primarily for **Bitcoin's scripting language**.
6. **Rholang:**
   - The language used by the **RChain platform** for writing smart contracts,
   - designed to **provide concurrency and scalability.**

# Types of Blockchain Programming - (1) Solidity Features

- **Smart Contracts** :
  - Primarily used for writing smart contracts on the Ethereum blockchain,
  - enabling the **creation of decentralized applications (DApps)**.
- **Syntax Similarities** :
  - Syntax of Solidity is **influenced by languages like JavaScript, Python, and C++,**
  - making it relatively familiar to programmers with experience in these languages.
- **Data Types:**
  - Solidity supports various data types,
  - **uint, int, address (Ethereum addresses), bool , string, bytes, and more.**
- **State Variables:**
  - store data on the blockchain.
  - They can be **persistent across function calls and transactions**,
  - **maintaining the state of a smart contract.**
- **Solidity functions:**
  - Building blocks of smart contracts.
  - They can be **public, private, internal, or external**,
  - they can **return values or trigger state changes.**

Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Features

- **Modifiers:**
  - used to **change the behavior of functions**,
  - often **enforcing conditions before a function is executed**.
- **Events:**
  - used to **log specific occurrences within a smart contract**.
  - They **provide a way for DApps to listen and respond to important changes**.
- **Inheritance:**
  - helps to create more **modular and reusable smart contracts by inheriting properties and functions from other contracts.**
- **Visibility:**
  - **Functions and state variables** have different visibility levels,
  - such as **public, internal, private, and external,**
  - which **determine who can access them.**
- **Gas Considerations:**
  - Smart contracts on Ethereum **require gas to execute**.
  - Solidity **code should be written with efficiency in mind to minimize gas costs**.

Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Features

- **Security Concerns:**
  - Solidity **code should be carefully audited to avoid vulnerabilities**
  - Such as reentrancy attacks, integer overflows, and other common pitfalls in blockchain programming.
- **Version Pragmas:**
  - Solidity code **starts with a version pragma statement**,
  - indicating the **compiler version** that should be **used to compile the code**.
- **Testing and Deployment:**
  - Solidity code is typically **tested using frameworks like Truffle**
  - deployed **using tools like Remix or command-line interfaces**.
- **Ecosystem:**
  - Solidity has a **growing ecosystem of tools, libraries, and frameworks**
  - aid in **development, testing, and deployment** of Ethereum smart contracts.
- **Documentation:**
  - Ethereum's official Solidity documentation provides **comprehensive resources, examples, and best practices for writing secure and efficient smart contracts**.

Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Use cases

1. **Decentralized Finance (DeFi)**:
   - DeFi protocols leverage Solidity to create decentralized financial products and services, such as lending and borrowing platforms, decentralized exchanges (DEXs), stablecoins, yield farming, and automated market makers (AMMs).
2. **Token Creation:**
   - Solidity is used to create and manage various types of tokens, including fungible (ERC-20) and non-fungible (ERC-721 and ERC-1155) tokens.
   - These tokens enable digital representation of assets, collectibles, and ownership rights.
3. **Initial Coin Offerings (ICOs) and Security Token Offerings (STOs):**
   - Solidity is employed to design and implement ICO and STO smart contracts, allowing projects to raise funds by issuing tokens to investors.
4. **Decentralized Applications (DApps):**
   - Solidity is crucial for building the backend logic of DApps, ranging from social networks and online marketplaces to gaming platforms and decentralized governance systems.

Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Use cases

5. **Decentralized Autonomous Organizations (DAOs):**
   - DAOs use Solidity to create governance mechanisms and rules that allow token holders to participate in decision-making and influence the direction of a project.
6. **Supply Chain Management:**
   - Solidity is utilized to create smart contracts that track and verify the provenance and authenticity of products throughout the supply chain.
7. **Non-Fungible Token (NFT) Marketplaces:**
   - NFT marketplaces are built using Solidity to enable the buying, selling, and trading of unique digital assets, such as digital art, virtual real estate, and in-game items.
8. **Escrow and Multi-Signature Wallets:**
   - Solidity is used to create secure and programmable wallets that allow multiple parties to control and manage funds, enabling more complex financial arrangements.

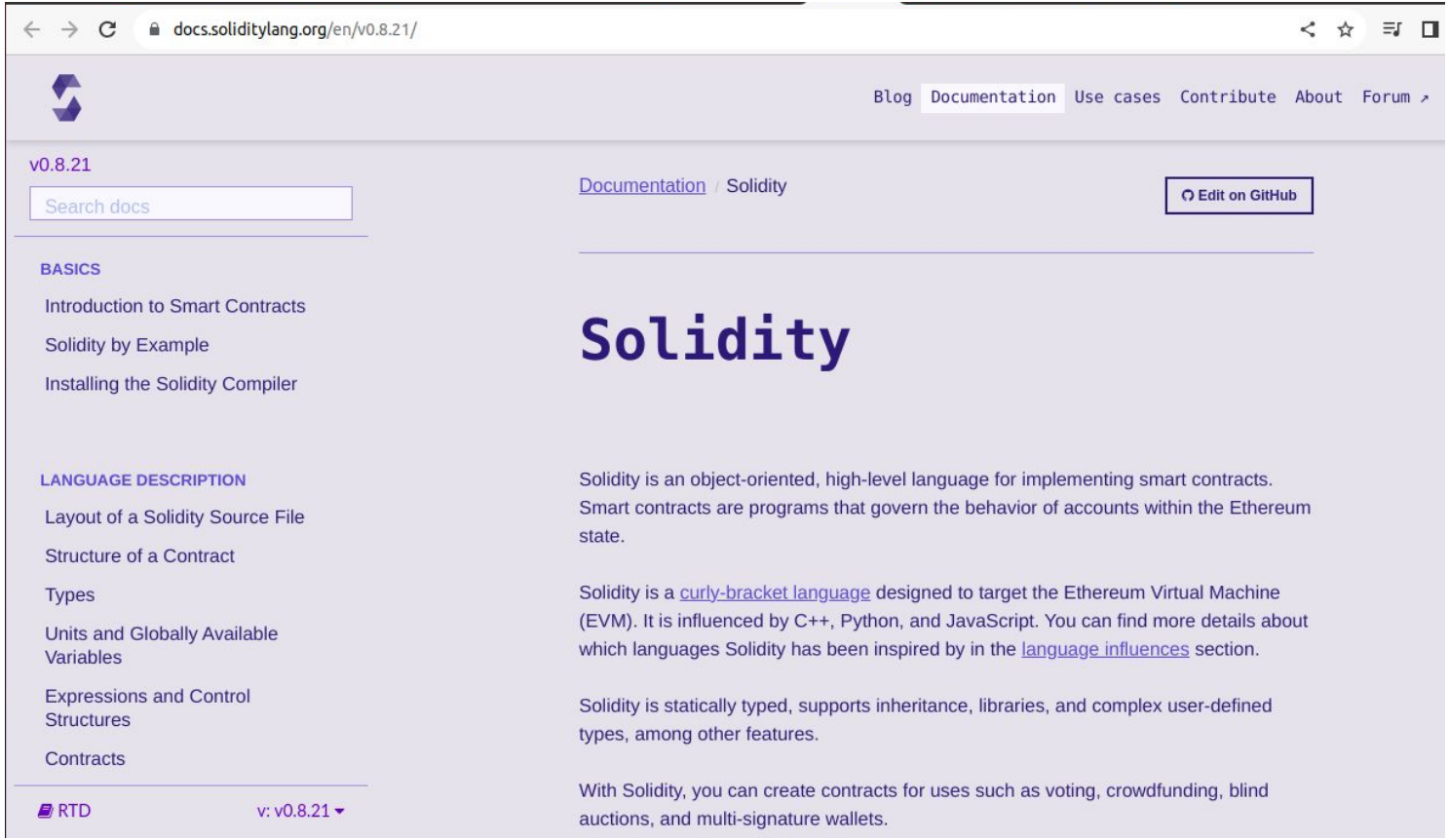
Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Use cases

9. **Decentralized Identity (DID):**
   - DID systems use Solidity to build self-sovereign identity solutions that provide users with control over their personal information while interacting with various services.
10. **Gambling and Betting Platforms:**
   - Solidity is employed to create smart contracts for decentralized gambling platforms, where players can participate in games and bets transparently and securely.
11. **Oracles and Data Feeds:**
   - Oracles use Solidity to fetch external data and provide it to smart contracts, enabling them to interact with real-world information.
12. **Real Estate and Land Registry:**
   - Solidity is used to create smart contracts that facilitate property transactions and maintain transparent records of ownership and transfers.

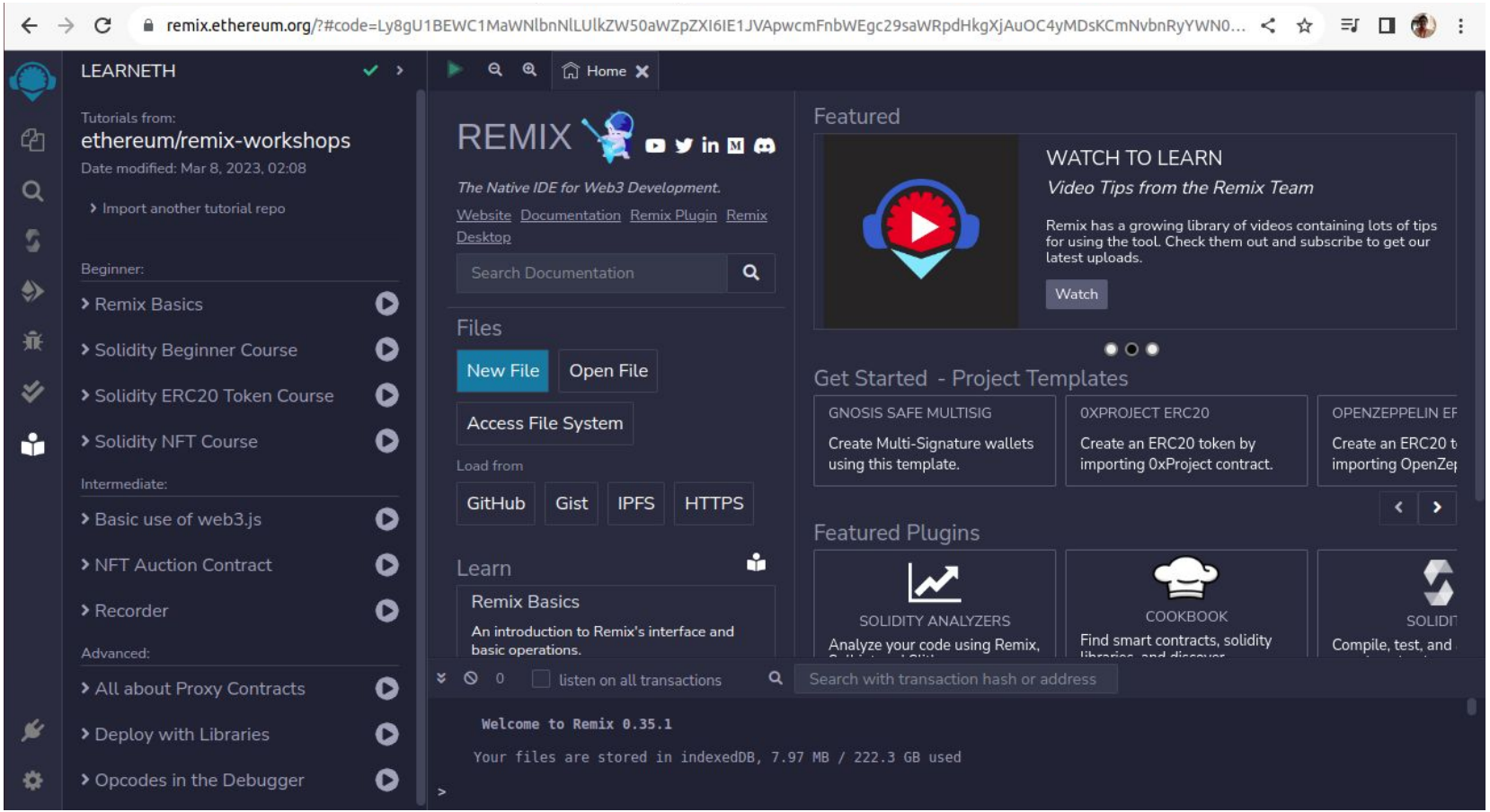
Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Learning



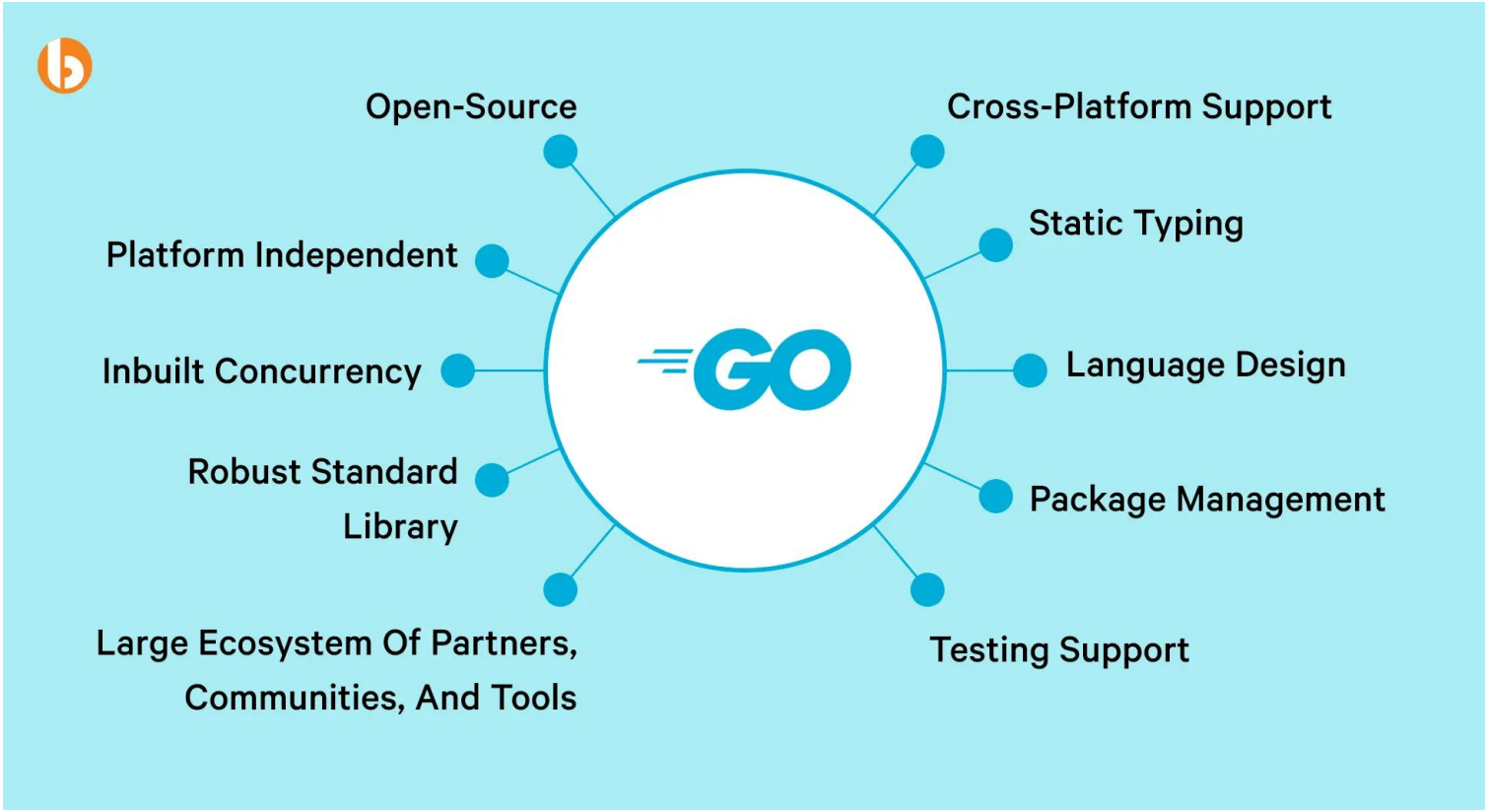

Courtesy : Solidity Doc

# Types of Blockchain Programming - (1) Solidity Learning




Courtesy : [Remix IDE - LearnETH - Solidity Course]

# Types of Blockchain Programming - (2) GoLang Features




Courtesy : Bacancy Technology

# Types of Blockchain Programming - (2) GoLang Features

1. **Efficiency and Performance:**
   - GoLang is known for its efficiency and high-performance characteristics.
   - Blockchain applications often involve handling a large number of transactions and computations.
   - Go's **fast compilation and execution speed** **contribute to creating performant blockchain systems.**
2. **Concurrency Support:**
   - Concurrency is crucial in blockchain systems where **multiple transactions and processes need to be handled simultaneously.**
   - Go's **goroutines and channels provide a powerful concurrency model**, making it easier to develop applications that handle parallel tasks efficiently.
3. **Memory Management**:
   - GoLang incorporates **automatic memory management through garbage collection.**
   - This helps **prevent memory leaks and reduces the chances of vulnerabilities** due to incorrect memory handling, which is particularly important for secure blockchain applications.
4. **Simple and Clean Syntax:**
   - Go's syntax is designed to be **simple and readable, which can accelerate development and contribute to code maintainability.**
   - Complex blockchain applications benefit from code that is **easy to understand and modify.**

Courtesy : GoLang

# Types of Blockchain Programming - (2) GoLang Features

5. **Strongly Typed Language**:
   - Being a **statically typed language**, Go performs **type checking at compile-time, catching errors early in the development process.**
   - This **reduces the risk of runtime errors and enhances code reliability.**
6. **Standard Library:**
   - Go comes with a comprehensive standard library that includes networking, file I/O, cryptography, and more.
   - This **library simplifies the development process by providing built-in functions and modules for common tasks.**
7. **Cross-Platform Compatibility:**
   - Go **can be compiled to run on different operating systems and architectures**, allowing for the creation of cross-platform blockchain applications and tools.
8. **Community and Resources:**
   - Go has a **thriving community of developers**, and it's often used in open-source projects.
   - The availability of resources, libraries, and frameworks in the Go ecosystem can significantly accelerate blockchain development.

Courtesy : GoLang

# Types of Blockchain Programming - (2) GoLang Features

9. **Security Focus**:
   - GoLang is designed with security in mind.
   - It includes features like array bounds checking and runtime checks to **prevent common security vulnerabilities, which is essential for secure blockchain applications.**
10. **Blockchain-Specific Libraries**:
   - Over time, the Go community has developed libraries and tools specific to blockchain development.
   - These libraries **facilitate interactions with blockchain networks, smart contracts, and cryptographic operations.**
11. **Compatibility with Existing Protocols**:
   - GoLang's ability **to work with low-level network protocols makes it suitable for building blockchain nodes and participating in blockchain networks.**
12. **Lightweight Binaries**:
   - Go **compiles to statically linked binaries**, which means that you can package your application along with its dependencies into a single executable, simplifying deployment.

Courtesy : GoLang

# Types of Blockchain Programming - (2) GoLang Use cases


Courtesy : GoLang Uses cases

# Types of Blockchain Programming - (2) GoLang Use cases

1. **Blockchain Node Development:**
   - GoLang is often used to develop blockchain nodes, the software that validates transactions, maintains the network, and synchronizes with the blockchain.
   - Notably, both **Ethereum's Geth** and **Binance Smart Chain's node** software are written in Go.
2. **Smart Contract Development:**
   - While Ethereum's primary language for smart contracts is Solidity, there are libraries and tools that allow you to write and interact with smart contracts using Go.
   - Go's concurrency support can be particularly useful in managing multiple transactions simultaneously.
3. **Decentralized Applications (DApps):**
   - GoLang can be used to build the backend logic of decentralized applications.
   - DApps often require efficient and concurrent handling of user interactions, data management, and interactions with the blockchain.
4. **Blockchain Infrastructure:**
   - Many blockchain projects use GoLang to build the infrastructure components of their networks, including consensus algorithms, network protocols, and cryptographic operations.

Courtesy : GoLang Uses cases

# Types of Blockchain Programming - (2) GoLang Use cases

5. **Command-Line Tools:**
   - Go's simplicity and fast compilation make it a good choice for building command-line tools that interact with blockchain networks.
   - These tools can range from wallet management to transaction monitoring.
6. **Microservices for Blockchains:**
   - GoLang's lightweight concurrency primitives are well-suited for building microservices that interact with various aspects of blockchain networks, including transactions, data retrieval, and monitoring.
7. **Blockchain Analytics and Monitoring:**
   - GoLang can be used to create tools for monitoring blockchain networks, analyzing transaction data, and providing insights into network health and performance.
8. **Smart Contract Testing:**
   - GoLang can be used to write tests and simulations for smart contracts, ensuring their functionality and security before deployment.

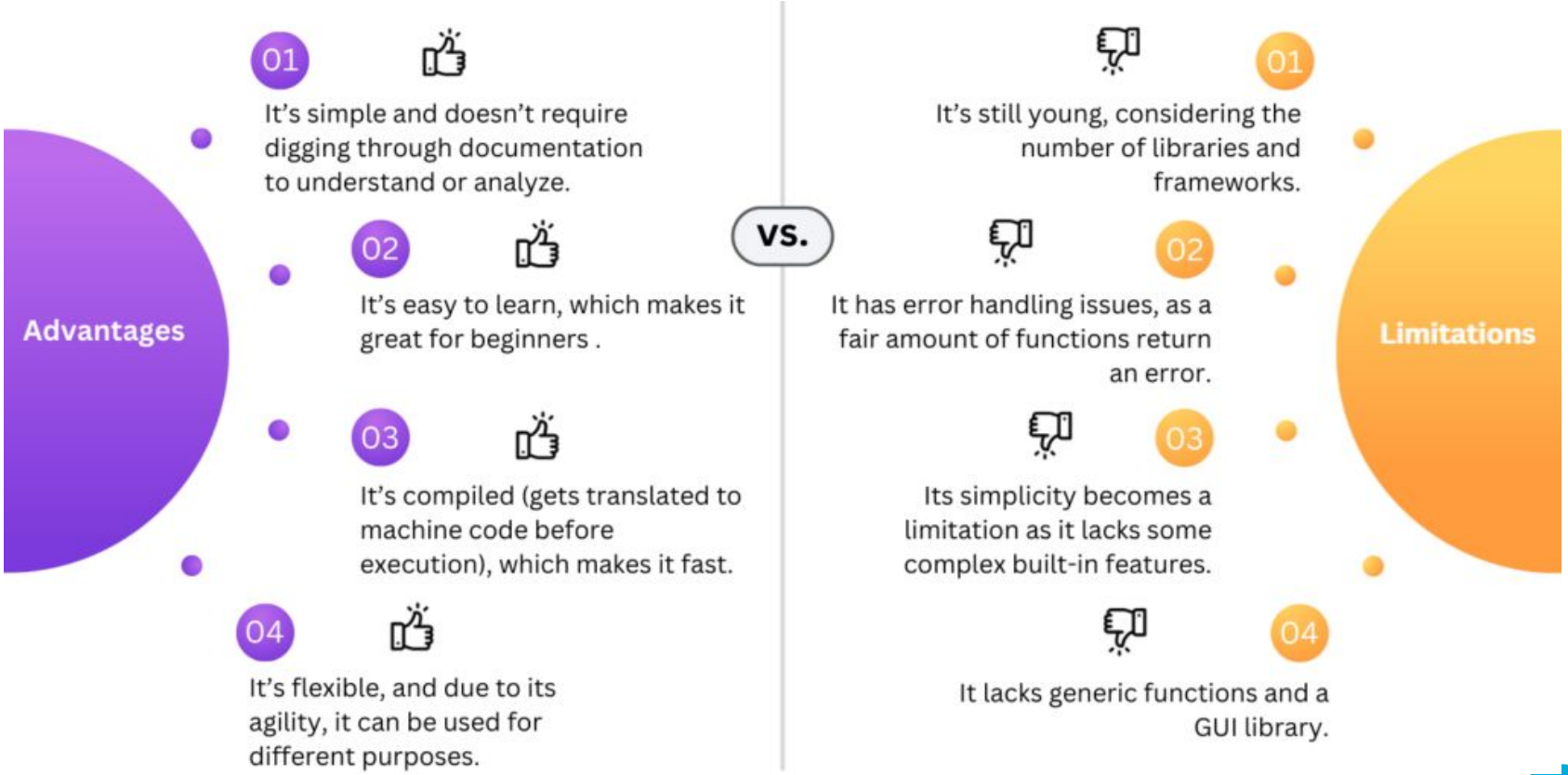
Courtesy : GoLang Uses cases

# Types of Blockchain Programming - (2) GoLang Use cases

9. **Blockchain Middleware:**
   - Middleware components that connect applications to blockchain networks can be developed using Go.
   - These components facilitate communication, data processing, and integration between applications and blockchains.
10. **Cryptocurrency Exchanges:**
   - GoLang can be used **to develop backend systems** for cryptocurrency exchanges, enabling trading, order matching, and wallet management.
11. **Cryptocurrency Wallets:**
   - Building cryptocurrency wallets with GoLang allows efficient interaction with blockchain networks for transactions, balance checks, and data retrieval.
12. **Interoperability Solutions:**
   - GoLang can be used to build interoperability solutions that facilitate communication and data exchange between different blockchain networks or between blockchains and external systems.

Courtesy : GoLang Uses cases

# Types of Blockchain Programming - (2) GoLang Pros & Cons

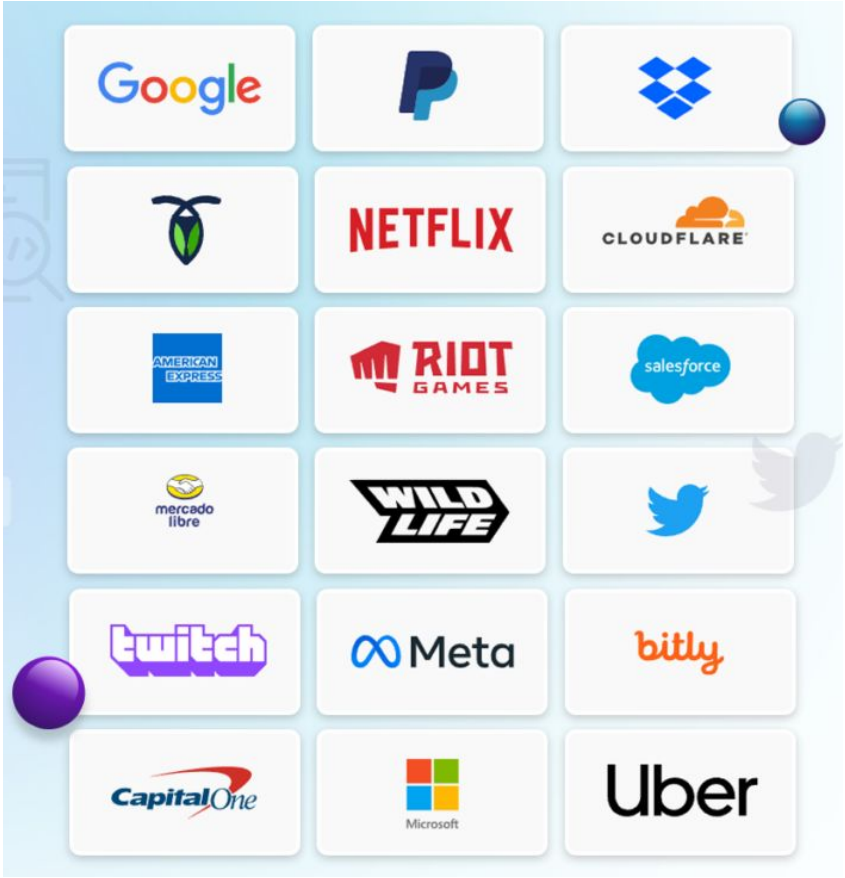



Courtesy : Kilzos - Go Lang

# Types of Blockchain Programming - (2) GoLang Web Frameworks

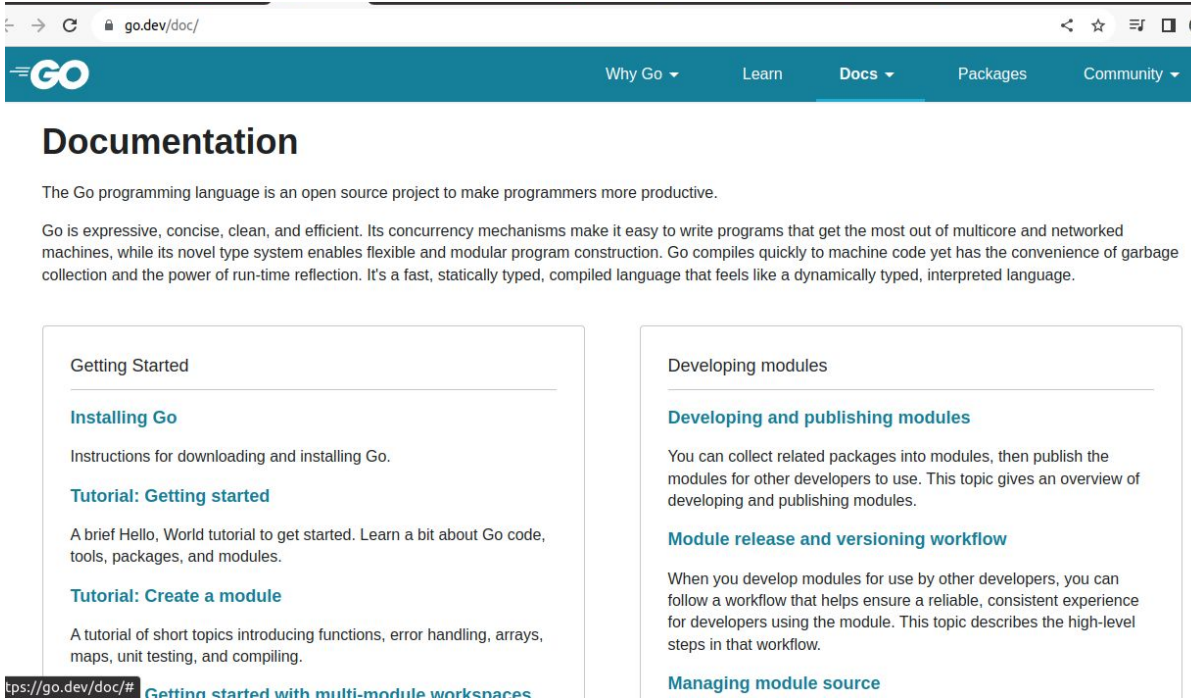

Courtesy : Kilzos - Go Lang

# Types of Blockchain Programming - (2) GoLang Market Stats




Courtesy : Kilzos - Go Lang

# Types of Blockchain Programming - (2) GoLang Learning







Courtesy :

GoLang Documentation

GoLang by Examples

# Types of Blockchain Programming - (3) Vyper Features

1. **Simplicity and Readability:**
   - Vyper's syntax is **intentionally simpler and more readable** than Solidity's.
   - This design choice **aims to reduce the potential for errors** and make it easier for developers to understand and reason about the code.
2. **Security-Oriented Design:**
   - Vyper is designed to help developers **avoid common pitfalls and vulnerabilities** that can arise in smart contract development.
   - Its syntax discourages complex and potentially risky features in favor of straightforward and safer alternatives.
3. **Limited Functionality:**
   - Vyper intentionally **omits certain features that could introduce complexity and security risks**.
   - For example, Vyper **doesn't support recursive function calls, inline assembly, and low-level memory management**, which can lead to unexpected behavior and vulnerabilities.
4. **Explicitness:**
   - Vyper **emphasizes explicitness in coding, making it clear how variables and functions interact.**
   - This can help in **reducing ambiguity and potential issues.**
5. **Type Safety:**
   - **strongly typed**, meaning that variable types are explicitly defined and checked at compile time.
   - This helps **prevent common type-related errors** that can lead to vulnerabilities.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Features

6. **Gas Efficiency:**
   - **encourages code that can be more easily optimized** by the Ethereum Virtual Machine (EVM), leading to potentially lower gas costs for executing contracts.
7. **Formal Verification:**
   - Vyper's simplicity makes it **more amenable to formal verification techniques,** which **can provide mathematical proofs of correctness for smart contracts.**
   - This can enhance the security of contracts by providing stronger guarantees about their behavior.
8. **Compatibility with Ethereum:**
   - Vyper compiles to Ethereum Virtual Machine (EVM) **bytecode,** making it **compatible with the Ethereum network.**
   - This means that contracts written in Vyper **can be deployed and executed on Ethereum** just like contracts written in Solidity.
9. **Ecosystem and Community**:
   - While Solidity is more widely used and has a larger ecosystem, Vyper has gained traction as a **security-focused alternative**.
   - The Vyper community continues to grow, and developers interested in enhanced security often explore using Vyper for their contracts.
10. **Learning Curve**:
    - Vyper's simplicity can be an advantage for security but may also mean a learning curve for developers who are more familiar with Solidity or other programming languages.
    - However, the l**earning curve is often outweighed by the potential security benefits.**

Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Use cases

1. **Security-Critical Contracts:**
   - Vyper's design places a strong emphasis on security and readability.
   - It is particularly suitable for writing contracts that **handle sensitive or valuable assets,** where security vulnerabilities could have significant consequences.
2. **Financial Instruments:**
   - Vyper can be used to create smart contracts for various financial instruments within the decentralized finance (DeFi) space, such as lending and borrowing platforms, stablecoins, decentralized exchanges (DEXs), and yield farming protocols.
3. **Token Standards:**
   - Vyper is suitable for creating tokens, including both fungible and non-fungible tokens (NFTs).
   - Vyper's readability and security features can help in preventing issues often associated with token standards.
4. **Governance and Voting Systems:**
   - Vyper can be used to develop smart contracts for decentralized autonomous organizations (DAOs) and other governance mechanisms, allowing token holders to participate in decision-making processes.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Use cases

5. **Escrow and Multi-Signature Wallets:**
   - Contracts that manage funds with certain conditions or require multiple parties to sign off can be developed using Vyper, ensuring that the logic is secure and transparent.
6. **Supply Chain and Provenance:**
   - Vyper can be employed to build contracts that track the origin and history of products throughout the supply chain, enhancing transparency and authenticity.
7. **Decentralized Identity (DID):**
   - Vyper is suitable for creating self-sovereign identity solutions where users have control over their personal data while interacting with various services.
8. **Lotteries and Games:**
   - Vyper can be used to create secure and transparent smart contracts for decentralized games, lotteries, and other interactive applications.

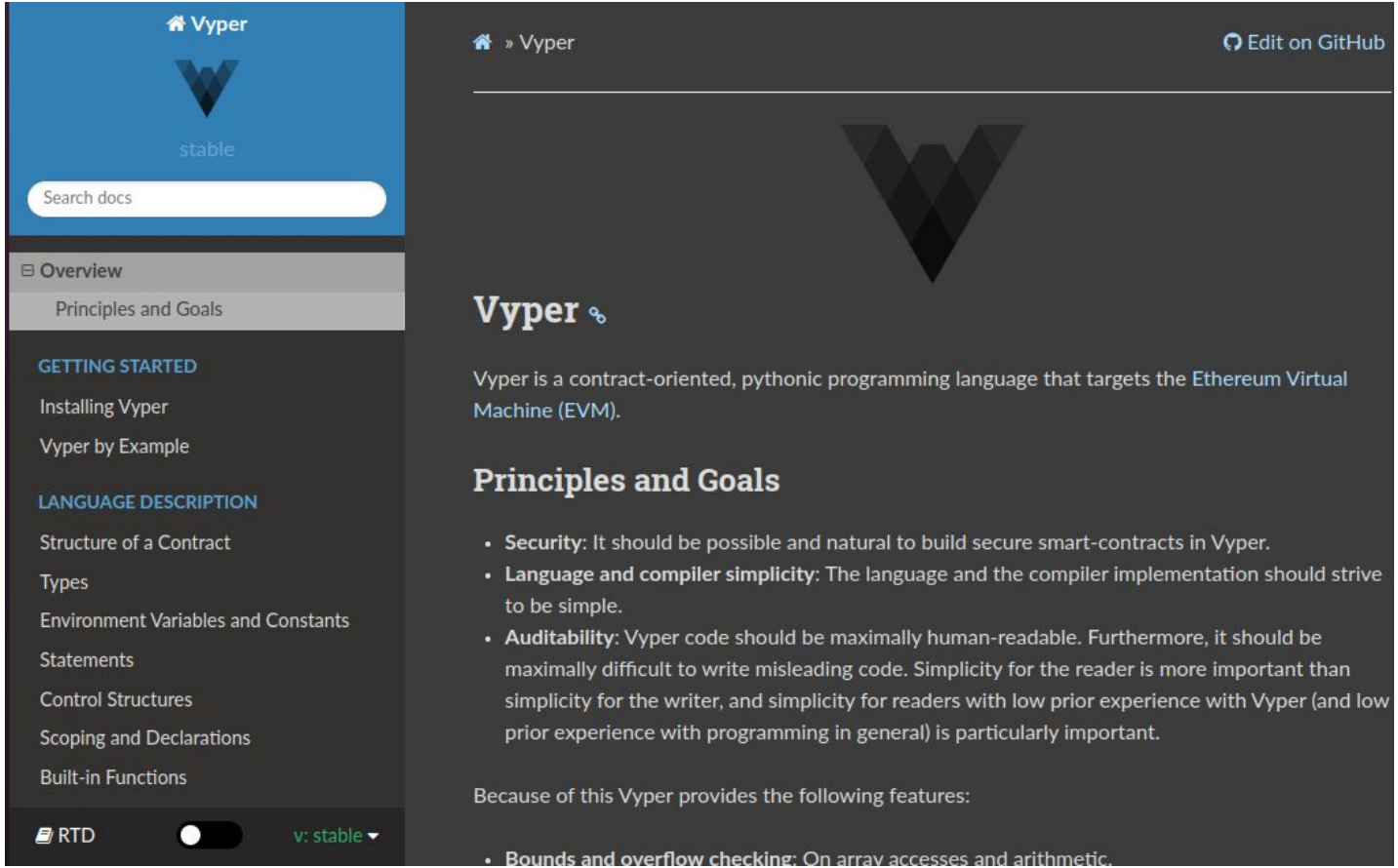
Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Use cases

9. **Real Estate Transactions:**
    - Contracts that facilitate real estate transactions, property ownership, and transfers can be developed using Vyper, ensuring the accuracy and transparency of records.
10. **Environmental Initiatives:**
    - Vyper can be applied to create contracts that support carbon credit trading, where emission reductions are tokenized and traded on the blockchain.
11. **Micropayments and Donations:**
    - Contracts that handle microtransactions and donations can be developed using Vyper, with an emphasis on security and efficiency.
12. **Custom Logic Contracts:**
    - Vyper is suitable for any scenario where custom logic is needed in a smart contract, provided that the contract's requirements align with Vyper's design philosophy.

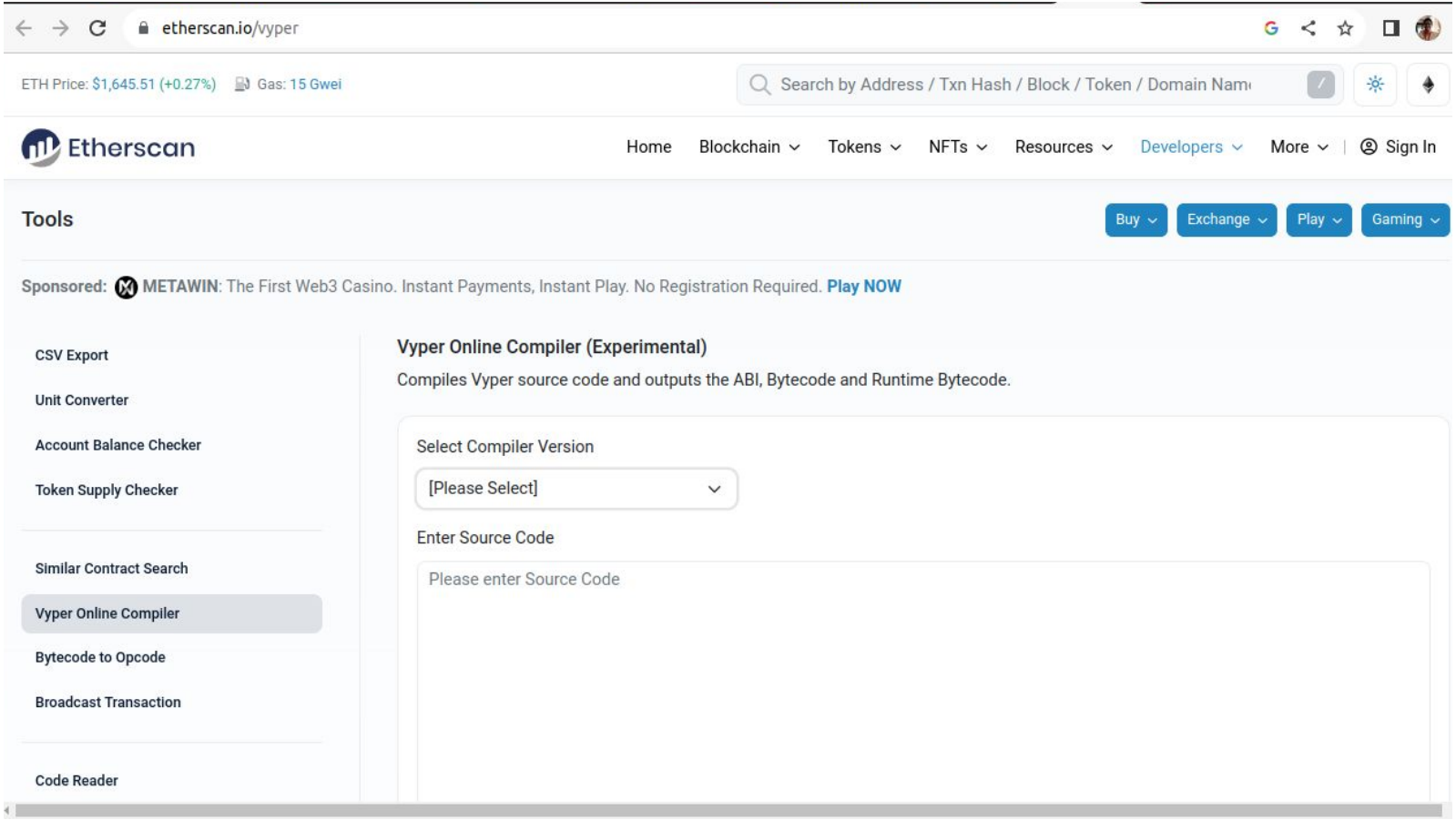
Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Learning



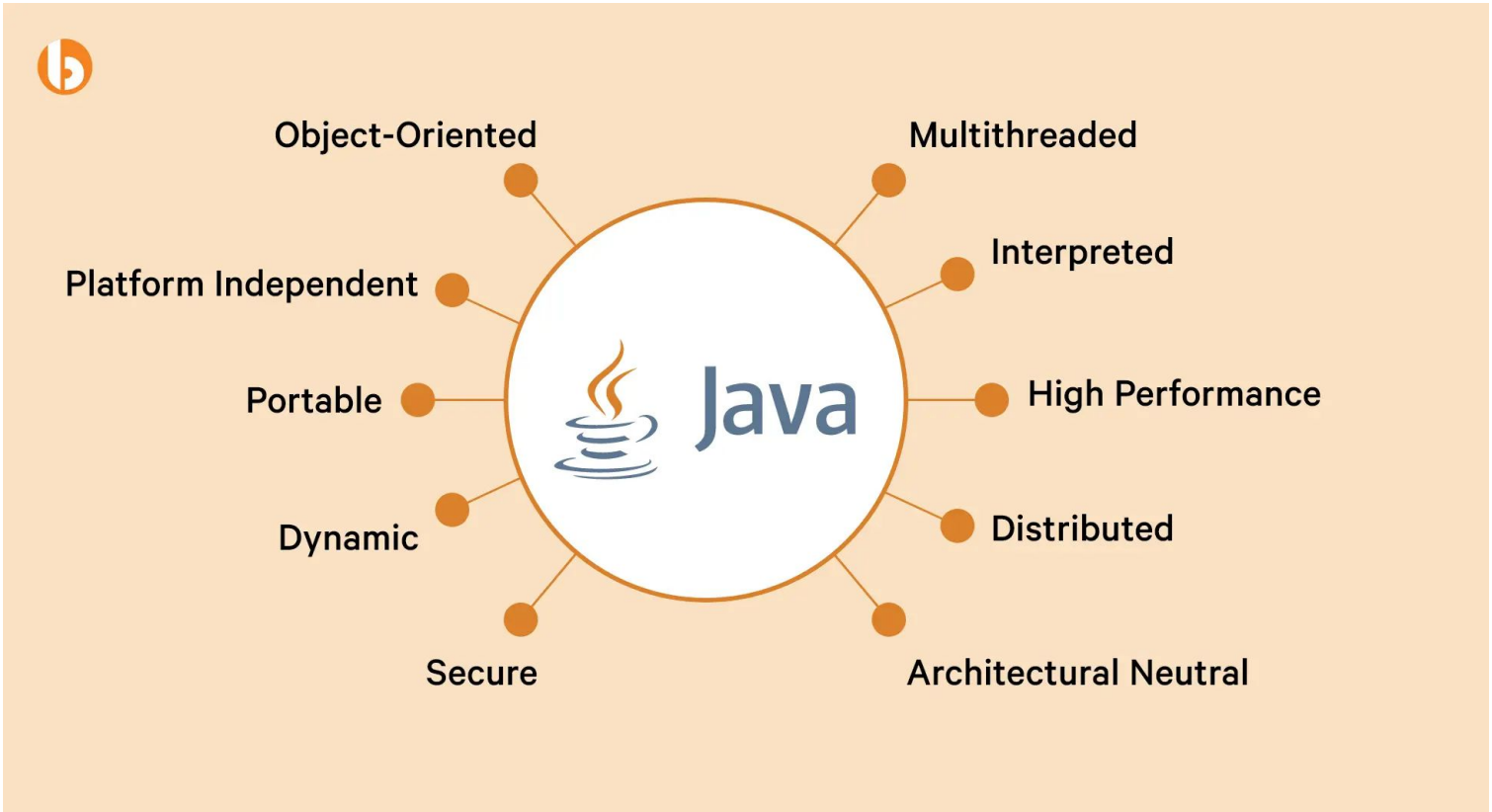

Courtesy : Vyper Doc

# Types of Blockchain Programming - (3) Vyper Learning




Courtesy : Vyper Online Editor

# Types of Blockchain Programming - (4) Java Features




Courtesy : Bacancy Technology

# Types of Blockchain Programming - (4) Java Features

- Java is a versatile programming language with a **wide range of applications, including blockchain development.**

1. **Hyperledger Fabric:**
   - Is a **permissioned blockchain framework** that allows you to create **private and consortium blockchains for enterprise use cases.**
   - Java is one of the supported programming languages for writing chaincode (smart contracts) in Hyperledger Fabric.
   - **Chaincode can be written in Java using the Fabric Java SDK.**
2. **Corda:**
   - Is a blockchain platform **designed for businesses and financial institutions.**
   - It allows for the development of applications that **require privacy and a degree of trust among participants**.
   - Corda smart contracts can be **written in Java.**
3. **Development of Blockchain Libraries:**
   - Java can be used to **develop libraries and tools that facilitate** blockchain development.
   - For instance, you can create **Java libraries for interacting with blockchain networks, handling transactions, and managing cryptographic operations**.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (4) Java Features

4. **Interoperability Solutions:**
   - Java's **portability and cross-platform compatibility** make it a good choice for developing interoperability solutions.
   - These solutions **enable communication and data exchange** between different blockchain networks or between blockchains and external systems.
5. **Consensus Algorithm Implementations:**
   - For developers with advanced blockchain expertise, Java can be used to **implement custom consensus algorithms or modify existing ones.**
   - This is relevant when developing private blockchains or experimenting with new consensus mechanisms.
6. **Blockchain Middleware:**
   - **Middleware components that connect applications to blockchain networks can be developed**.
   - These components **facilitate communication, data processing, and integration between business applications and blockchain networks.**

Courtesy : Vyper Doc

# Types of Blockchain Programming - (4) Java Features

7. **Enterprise Blockchain Solutions:**
   - For enterprises building blockchain solutions with **complex business logic and integration requirements**, Java's extensive ecosystem and object-oriented nature can be advantageous.
8. **Integration with Existing Systems:**
   - Java's popularity in enterprise software development makes it a **strong candidate for integrating blockchain functionality into existing enterprise systems and workflows.**
9. **Learning Resources:**
   - Java has a rich array of **learning resources, tutorials, and documentation available,** making it accessible for developers who are already familiar with the language.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (4) Java Use cases

1. **Enterprise Blockchain Solutions:**
   - Java is often used to develop enterprise-grade blockchain solutions that require integration with existing systems and complex business logic. It's well-suited for building private and consortium blockchains tailored to specific business needs.
2. **Hyperledger Fabric Network Components:**
   - Hyperledger Fabric, an enterprise blockchain framework, supports Java for developing chaincode (smart contracts) and network components. Java is used to write chaincode logic that is executed within the Fabric network.
3. **Corda Distributed Applications:**
   - Corda, another enterprise blockchain platform, allows developers to create distributed applications (CorDapps) in Java. Corda's focus on privacy and interoperability makes Java a suitable choice for building solutions that require secure data sharing.
4. **Interoperability Solutions:**
   - Java's portability and cross-platform compatibility make it useful for building interoperability solutions that connect different blockchain networks or facilitate communication between blockchains and external systems.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (4) Java Use cases

5. **Consensus Algorithm Implementations:**
   - For developers with deep blockchain expertise, Java can be used to implement custom consensus algorithms or modify existing ones. This is relevant when developing private blockchains or experimenting with new consensus mechanisms.
6. **Blockchain Middleware:**
   - Java can be used to build middleware components that connect applications to blockchain networks, facilitating data exchange, transaction processing, and integration with business processes.
7. **Cryptocurrency Exchanges and Trading Platforms:**
   - Java's extensive libraries and networking capabilities make it a viable choice for building the backend systems of cryptocurrency exchanges, enabling trading, order matching, and wallet management.
8. **Data Analytics and Reporting:**
   - Java can be employed to create tools for analyzing and reporting on blockchain data, helping organizations gain insights into transaction history, network health, and other metrics.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (4) Java Use cases

9. **Supply Chain and Logistics:**
   - Java can be used to build blockchain solutions that track and verify the movement of goods in supply chains, enhancing transparency and accountability.
10. **Digital Identity Solutions:**
   - Java can be applied to create blockchain-based digital identity solutions that provide users with control over their personal data while interacting with various services.
11. **Healthcare and Medical Records:**
   - Java can be used to build secure and interoperable solutions for managing and sharing medical records on blockchain networks, ensuring privacy and data integrity.
12. **Regulatory Compliance and Auditing:**
   - Java can be employed to develop blockchain applications that facilitate regulatory compliance, auditing, and reporting by providing transparent and tamper-resistant records.

Courtesy : Vyper Doc

# Types of Blockchain Programming - (5) Simplicity Features

1. **Minimalistic Syntax:**
   - The language's syntax would be minimal and straightforward, making it easy for developers to read and write code without unnecessary complexities.
2. **Limited Abstraction:**
   - The language might focus on providing a limited set of well-defined and clear abstractions, allowing developers to express blockchain logic concisely without excessive layers of abstraction.
3. **Built-in Security Features:**
   - Simplicity could incorporate security features by default, reducing the potential for vulnerabilities and making it easier for developers to write secure smart contracts.
4. **Reduced Complexity:**
   - The language might avoid advanced language features and concepts that can introduce complexity. This can help developers avoid mistakes and misunderstandings in their code.
5. **Explicit Typing:**
   - Simplicity might use explicit typing to enhance code clarity and prevent type-related errors, which is especially important in smart contract development.

Courtesy :

# Types of Blockchain Programming - (5) Simplicity Features

6. **Focus on Determinism:**
   - The language could emphasize deterministic behavior, ensuring that code execution produces the same results across different nodes in a blockchain network.
7. **Ease of Verification:**
   - A simple language could enable easier formal verification of smart contracts, allowing for mathematical proofs of correctness and enhancing security.
8. **Gas Efficiency:**
   - Simplicity could encourage the development of code that's gas-efficient, reducing the cost of executing transactions on the blockchain.
9. **Resource Efficiency:**
   - The language could promote efficient use of memory and other resources, preventing potential bottlenecks and optimizing performance.
10. **Documentation and Education:**
    - A simple language would likely come with comprehensive documentation, tutorials, and resources to help developers learn and adopt the language quickly.
11. **Community and Ecosystem:**
    - An active and growing community would be crucial for the success of the language, fostering collaboration, sharing of best practices, and the development of libraries and tools.

Courtesy :

# Types of Blockchain Programming - (5) Simplicity Use cases

1. **Smart Contract Development:**
   - Designing a programming language with simplicity in mind can be beneficial for writing secure and auditable smart contracts on blockchain platforms. Simplicity in the language's syntax and semantics can reduce the potential for bugs, vulnerabilities, and misunderstandings in the contract code.
2. **Education and Learning:**
   - A programming language that emphasizes simplicity can be an excellent tool for teaching programming concepts to beginners. Such a language can help newcomers focus on core concepts without being overwhelmed by complex syntax and advanced features.
3. **Rapid Prototyping:**
   - A simple programming language can enable developers to quickly prototype ideas and concepts without getting bogged down in intricate language features. This can be useful for exploring new technologies or experimenting with blockchain concepts.
4. **Code Maintainability:**
   - Simple and clear code is easier to maintain, refactor, and modify over time. A programming language designed for simplicity can lead to more maintainable and less error-prone code in the long run.

Courtesy :

# Types of Blockchain Programming - (5) Simplicity Use cases

5. **Open Source Contributions:** A simple programming language can encourage more developers to contribute to open-source projects. Contributors from various skill levels can more easily understand and modify code written in a language that prioritizes simplicity.
6. **Cross-Disciplinary Collaboration**: Simplicity in a programming language can facilitate collaboration between individuals from different disciplines, such as domain experts and developers. Clear and straightforward code can bridge communication gaps and lead to more effective collaboration.
7. **Embedded Systems:** Simple programming languages can be suitable for writing code for embedded systems, where resource constraints and real-time requirements necessitate efficient and focused programming.
8. **Scripting and Automation:** A simple language can be useful for writing scripts and automation tasks, where the primary goal is to achieve specific outcomes without the need for complex language constructs.
9. **Legacy System Integration:** A simple programming language can be used to create interfaces and integration points for legacy systems, enabling them to interact with modern technologies.
10. **Domain-Specific Languages (DSLs):** Creating a simple DSL tailored to a specific industry or domain can make it easier for non-technical experts to express complex concepts and logic effectively.

Courtesy :

# Types of Blockchain Programming - (6) Rholang Features

- Rholang is a programming language designed for the **RChain blockchain platform.**
- RChain is a blockchain that aims **to address scalability and performance issues** that have been observed in earlier blockchain networks.
- Rholang is specifically tailored to take advantage of **RChain's unique architecture and concurrency model.**
- Here are some key aspects of Rholang:

1. **Concurrent Execution:** Rholang is built around a calculus known as the π-calculus, which provides a foundation for expressing concurrent processes. This concurrency model aligns well with the RChain platform's focus on scalability and parallelism.
2. **Process Calculus:** Rholang's syntax and semantics are based on the π-calculus, a formal model for describing concurrent computations. This calculus enables precise descriptions of concurrent behavior in a decentralized setting.
3. **Channel-Based Communication:** In Rholang, communication between processes happens through channels. This allows for secure and precise communication between different parts of a distributed application.

# Types of Blockchain Programming - (6) Rholang Features

4. **Smart Contracts:** Rholang is primarily used for writing smart contracts on the RChain platform. Smart contracts written in Rholang can interact with each other in a concurrent and decentralized manner.
5. **Deterministic Execution:** Rholang's design emphasizes deterministic execution, meaning that given the same inputs, the behavior of a contract is predictable and consistent across all nodes in the network.
6. **Formal Verification:** The mathematical foundations of Rholang, derived from the π-calculus, make it amenable to formal verification techniques. This can enhance the security and reliability of smart contracts written in Rholang.
7. **Resource Management:** Rholang introduces the concept of "persistent" and "ephemeral" data, where persistent data is stored on-chain while ephemeral data exists temporarily for the duration of a contract's execution.

Courtesy :

# Types of Blockchain Programming - (6) Rholang Features

8. **Correct-by-Construction:** Rholang's design encourages correct contract behavior by construction, helping developers avoid common pitfalls and errors.
9. **Rev Addresses:** Rholang introduces the concept of "rev addresses" as a way to manage resources and accounts on the RChain platform.
10. **Reflective Architecture:** Rholang allows smart contracts to introspect and manipulate the contract space, enhancing flexibility and programmability.
11. **Resource Calculus:** Rholang includes a resource calculus, which helps ensure that resources are used in a controlled and accountable manner.

# Types of Blockchain Programming - (6) Rholang Use cases

1. **Decentralized Applications (DApps):** Rholang can be used to develop decentralized applications (DApps) on the RChain platform. DApps can cover a wide range of domains, from finance and supply chain to gaming and identity management.
2. **Scalable Smart Contracts:** Rholang's concurrency model aligns well with RChain's focus on scalability and parallelism. It can be used to create smart contracts that execute efficiently and can handle a large number of transactions concurrently.
3. **Data Storage and Retrieval:** Rholang can be used to develop contracts that store and retrieve data on the RChain blockchain. Its concurrency capabilities can enhance the speed and efficiency of data access and manipulation.
4. **Decentralized Finance (DeFi)**: Similar to other blockchain platforms, Rholang can be used to build DeFi protocols on RChain. These may include lending platforms, decentralized exchanges, automated market makers, and more.

Courtesy :

# Types of Blockchain Programming - (6) Rholang Use cases

5. **Tokenization:** Rholang can be used to tokenize various assets on the RChain blockchain, creating fungible and non-fungible tokens (NFTs) that represent ownership of assets such as real estate, intellectual property, and collectibles.
6. **Decentralized Identity (DID):** Rholang can play a role in developing self-sovereign identity solutions on the RChain platform, allowing users to control their digital identities and personal data securely.
7. **Governance Mechanisms:** Rholang can be used to create smart contracts that enable decentralized governance and decision-making within the RChain ecosystem, allowing token holders to participate in shaping the platform's future.
8. **Content Sharing and Intellectual Property:** Rholang can facilitate the creation of platforms for content sharing, licensing, and management, where creators and consumers can interact directly with transparent and secure protocols.

Courtesy :

# Types of Blockchain Programming - (6) Rholang Use cases

9. **Supply Chain and Logistics:** Rholang can be applied to build blockchain-based solutions for tracking and verifying the movement of goods in supply chains, improving transparency and traceability.
10. **Energy and Sustainability:** Rholang can be used to develop blockchain applications related to energy trading, carbon credits, and sustainable resource management.
11. **Decentralized Autonomous Organizations (DAOs):** Rholang can be used to create DAOs that operate autonomously based on predefined rules and logic, enabling decentralized decision-making and resource allocation.

Courtesy :

# Comparative study of different blockchain programming languages

| Parameters | Vyper | Solidity |
|---|---|---|
| **Language Design** | Simple, security-focused, minimalist. | Feature-rich, syntax similar to JavaScript. |
| **Performance** | Faster, more efficient. | Slower. |
| **Security** | Designed with security in mind, stack-only arithmetic, no loops, and no inheritance. | More prone to security vulnerabilities. |
| **Community** | Smaller, less active. | Larger, more active. |
| **Language Features** | Simple, limited feature set. | Advanced, more feature-rich. |
| **Contract Interaction** | Limited | Advanced, ability to call and send transactions to other contracts. |

Courtesy : Geeks For Geeks



# Comparative study of different blockchain programming languages

| Parameters | Vyper | Solidity |
|---|---|---|
| **Development Speed** | Faster development time due to simpler syntax. | Slower development time due to more advanced features. |
| **Compilation Time** | Faster compilation time due to simpler syntax. | Slower compilation time due to more complex features. |
| **Error Detection** | Early error detection due to simpler syntax. | Late error detection due to more complex features. |
| **Contract Size** | Smaller contract size due to limited feature set. | Larger contract size due to more complex features. |
| **Contract Readability** | Vyper is easy to read and understand due to its simple syntax. | Solidity is more difficult to read and understand due to more complex features. |

Courtesy : Geeks For Geeks



# Comparative study of different blockchain programming languages

| | Solidity | Vyper |
|---|---|---|
| Ease of syntax | Easy to learn for C++ and JavaScript programmers. | Easy to learn for Python programmers. |
| Flexibility of learning | Access to multiple developer resources and tools. | Limited amount of learning materials. |
| Size of arrays and strings | Dynamic sizing of strings and arrays. | Limited size of strings and arrays |
| Community support | Extensive community. | Newly developing community. |
| Definition of contract and error handling | Need for licensing detail to define contracts and compile contracts to identify errors. | Only requires Vyper version for defining contracts along with immediate error alerts. |
| Variable definition | Solidity variable definition is complicated and requires semi-colons. | A variable definition is straightforward in Vyper. |
| Creating auction | Complicated process for create auctions with the need for defining errors. | Vyper can define errors using asserts and use external decorators. |
| Specifying functions | Difficult process with the need for defining the function alongside payable and if statements. | Vyper uses asserts for specifying functions along with the use of an external decorator. |
| Withdrawal | Trouble writing 'if' statements. | No additional scripting requirements for withdrawal. |
| Contract ending | Write an 'if' statement and verify the ending variable value being set to 'True.' | Use asserts for verifying the end of the smart contract deadline. |

Courtesy : 101 Blockchains

# Comparative study of different blockchain programming languages

| | Solidity | Rust |
|---|---|---|
| Definition | Object-oriented, statically-typed high-level programming language for smart contract development. | Low-level, multi-paradigm programming language focused on high performance and memory safety. |
| Supported Blockchains | Ethereum and EVM-compatible blockchains such as Polygon, Optimism, Polkadot, Avalanche and Arbitrum. | Solana, Near, Sui and Aptos. |
| Features | • In-built conditional statements<br>• In-built data types<br>• Secure coding environment | • Move semantics<br>• Pattern matching<br>• Cost-effective abstraction<br>• Memory safety<br>• Type Interface<br>• Threads without data races |
| Type of Language | High Level | Low Level |
| Programming Paradigm | Object-Oriented Programming | Multi-paradigm programming with support for functional, dynamic, imperative and object-oriented programming. |
| Advantages | • Easier learning.<br>• Massive collection of developer tools. | • Memory safety.<br>• Better speed and performance. |
| Disadvantages | • Integer overflows.<br>• Challenges in static analysis. | • Slower compilation.<br>• Difficulty in learning. |

Courtesy : 101 Blockchains

# Comparative study of different blockchain programming languages

| Vyper | Characteristics | Solidity |
|---|---|---|
| It is in the early stages and may offer poor support. | Support | Support for solidity is widely available. |
| • An integer overflow causes the program to revert.<br>• It has built-in asserts in Ether transfer functions. | Security | • Integer underflow and overflow<br>• Unchecked call return value |
| Vyper does not offer modifiers for enhancing readability and detecting malicious codes. | Features | Solidity supports modifiers, which can be delegated in another code location. This allows performing checks before executing contracts and beyond their execution |
| Vyper does not offer inheritance, resulting in codes being audited conveniently as fewer files require an audit. | Class Inheritance | Solidity offers multiple inheritances requiring readers to learn precedence rules to comprehend conflict resolutions. |
| By eliminating function overloading, it is more convenient to understand which when functions are being called. | Function overloading | Function overloading in Solidity allows harmful codes to get ignored. |
| Vyper is not a complete substitute for Solidity, as several functionalities offered by Solidity are specifically restricted when using Vyper for coding. | Flexibility | Solidity offers smart contract flexibility, which enhances its range of applications. |
| The resources and guides available for Vyper are consequently fewer. Hence finding community support for addressing difficulties can be challenging. | Developer Community | Solidity is a lot more popular in the blockchain community platform. |

Courtesy : Blockchain Council

# Comparative study of different blockchain programming languages

| CRITERIA | SOLIDITY | MOVE | CLARITY |
|---|---|---|---|
| BLOCKCHAIN PLATFORM | ETHEREUM | DIEM | STACKS 2.0 |
| NEED FOR COMPILATION | The compilation is mandatory. The compiler available in Solidity is solc. | The compilation is necessary. The compiler available in Move is Move IR. | The compilation is not required in this case as it is an interpreted language. |
| SUPPORTED DATA TYPES | Booleans<br>Address<br>Strings<br>Integers<br>Functions<br>Byte arrays<br>Enums | Booleans<br>256-bit addresses<br>References<br>Unsigned 64-bit Integers<br>Fixed-size byte array | Integers<br>Booleans<br>Principal<br>Response<br>Buffer<br>Tuple<br>Optional<br>List |
| COMPLEX TYPES | Structs<br>Fixed-size and dynamic-size arrays<br>Mappings | Structs<br>Resources | Mappings |
| TURING COMPLETENESS | Turing complete | Turing complete | Turing incomplete |
| VULNERABILITY TO REENTRANCY ATTACKS | Vulnerability in event of vulnerability in code | No signs of vulnerability | No signs of vulnerability |
| DYNAMIC DISPATCH | Support for dynamic dispatch | No support for dynamic dispatch | No support for dynamic dispatch |
| FLEXIBILITY OF USE | Highest flexibility | Medium-level flexibility | Least flexibility |

Courtesy : 101 Blockchains

# Comparative study of different blockchain programming languages

| | SOLIDITY | clarity |
|---|---|---|
| Blockchain | Ethereum | Stacks *(a programming layer for Bitcoin)* |
| Year created | 2014 | 2020 |
| Compiled | Yes | No |
| Turing completeness | Turing complete | Decidable (Turing incomplete) |
| Supported types | Signed and unsigned integers, Booleans, Addresses, Enums, Bytes | Booleans, Buffer, Integers, List, Optional, Principal, Response, Tuple |
| Gas estimation | Fundamentally an estimation, given the undecidable nature of the function call graph | Precise gas estimation; complete static analysis of the call graph so you get exact cost before execution |
| Language design philosophy | Imperative programming style, like C++, JavaScript, Python | Functional programming style, like LISP and Scala |
| Development environment | Remix, Truffle, Hardhat | VS Code, Clarinet, ClarityTools |
| Reentrancy allowed | Yes | No |

Courtesy : Hiro.so

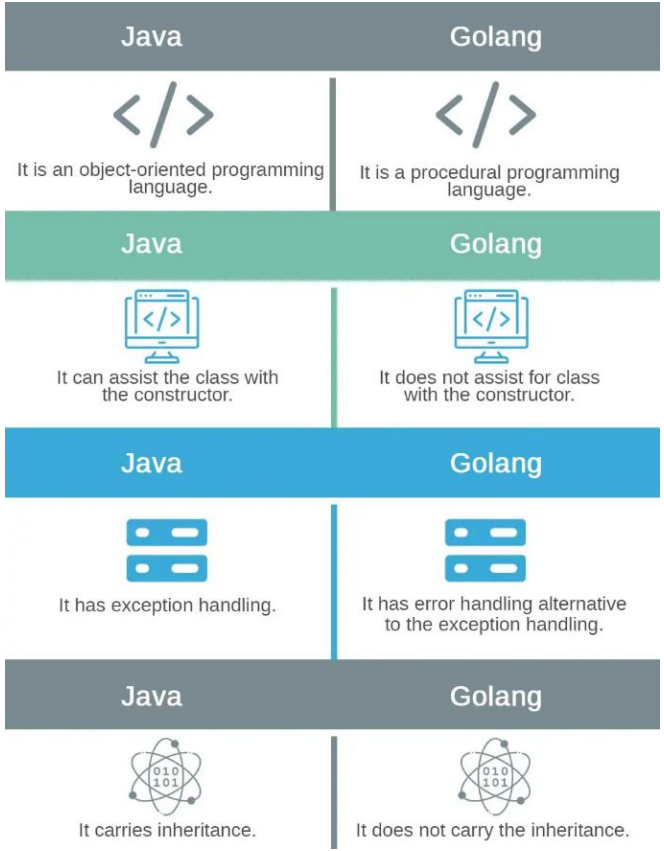
# Comparative study of different blockchain programming languages

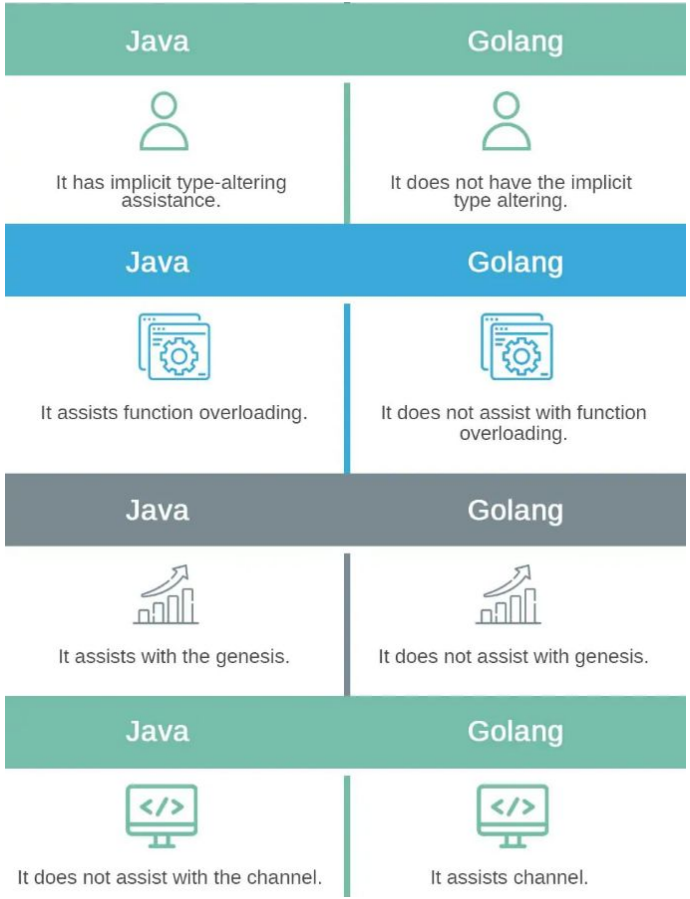
| Java | Golang |
|---|---|
| It is an object-oriented programming language. | It is a procedural programming language. |
| It can assist the class with the constructor. | It does not assist for class with the constructor. |
| It has exception handling. | It has an error-handling alternative to exception handling. |
| It carries inheritance. | It does not carry the inheritance. |
| It has implicit type-altering assistance. | It does not have the implicit type altering. |
| It assists function overloading. | It does not help with function overloading. |
| It helps with the genesis. | It does not help with genesis. |
| It does not help with the channel. | It assists the channel. |

Courtesy : Educba



# Comparative study of different blockchain programming languages




Courtesy : Educba

# Comparative study of different blockchain programming languages

| Parameters | Golang | Java |
|---|---|---|
| 1. Syntax | Golang has a simpler and more concise syntax with fewer keywords and punctuation marks. | Java has a more verbose syntax with more keywords and punctuation marks. |
| 2. Concurrency | Golang has built-in support for concurrency with goroutines and channels. | Java has concurrency features like threads and synchronized blocks. |
| 3. Compilation | Golang is a compiled language with fast compilation times. | Java is a compiled language that can also be interpreted through the use of a JVM. |
| 4. Garbage collection | Golang has a concurrent garbage collector which does not stop the program execution. | Java has a garbage collector that may pause the program execution to free memory. |
| 5. Error handling | Golang has a unique error-handling mechanism using explicit return values that can be easily checked. | Java has a try-catch-finally mechanism for handling exceptions. |
| 6. Type system | Golang has a static type system that supports type inference. | Java has a static type system that does not support type inference. |
| 7. Package management | Golang has a built-in package management system called "go modules". | Java uses external package management tools like Maven and Gradle. |
| 8. Performance | Golang is known for its high-performance and low-latency processing capabilities. | Java is relatively slower in terms of performance due to its virtual machine architecture. |
| 9. Community support | Golang has a growing and active community with a focus on simplicity and performance. | Java has a mature and large community with a focus on enterprise development. |
| 10. Use cases | Golang is often used for system-level programming, networking, and web development. | Java is often used for enterprise development, Android app development, and web development. |

Courtesy : [Turing - GoLang Vs Java]

# Comparative study of different blockchain programming languages

| Parameters | Golang | Java |
|---|---|---|
| **Type of Language** | Concurrent and Procedural Language | Object-Oriented Programming Language |
| **Error Detection and Handling** | Uses Errors instead of Exceptions Handling | Concept of Exceptional Handling used for Errors |
| **Speed of Coding** | More compact and unforgiving code, with less room for errors. | More stable and follows a familiar object-oriented approach. |
| **Performance** | Faster than Java | Fast |
| **Memory Usage** | It has pointers and no links for Garbage Collection | Virtual Machine along with a traditional Garbage Collector running from time to time manages the memory. |

Courtesy : Banacy Technology

# Comparative study of different blockchain programming languages

| Parameters | Golang | Java |
|---|---|---|
| **Application** | Designed for simplicity and scalability; therefore, exceptional multithreading is its important feature. | The JVM interacts with the hardware, making the coding efficient to work in any application. |
| **Feature-Rich** | Comparatively, fewer features | Extensive Features |
| **Popularity** | More Popular | Comparatively, Less Popular |
| **Community Support** | Comparatively, small but active | Huge Community along with Active community support |
| **Inheritance** | Not Supported | Supported |
| **Compactness** | More Compact Programs | Less Compact Programs |

Courtesy : Banacy Technology

# Comparative study of different blockchain programming languages

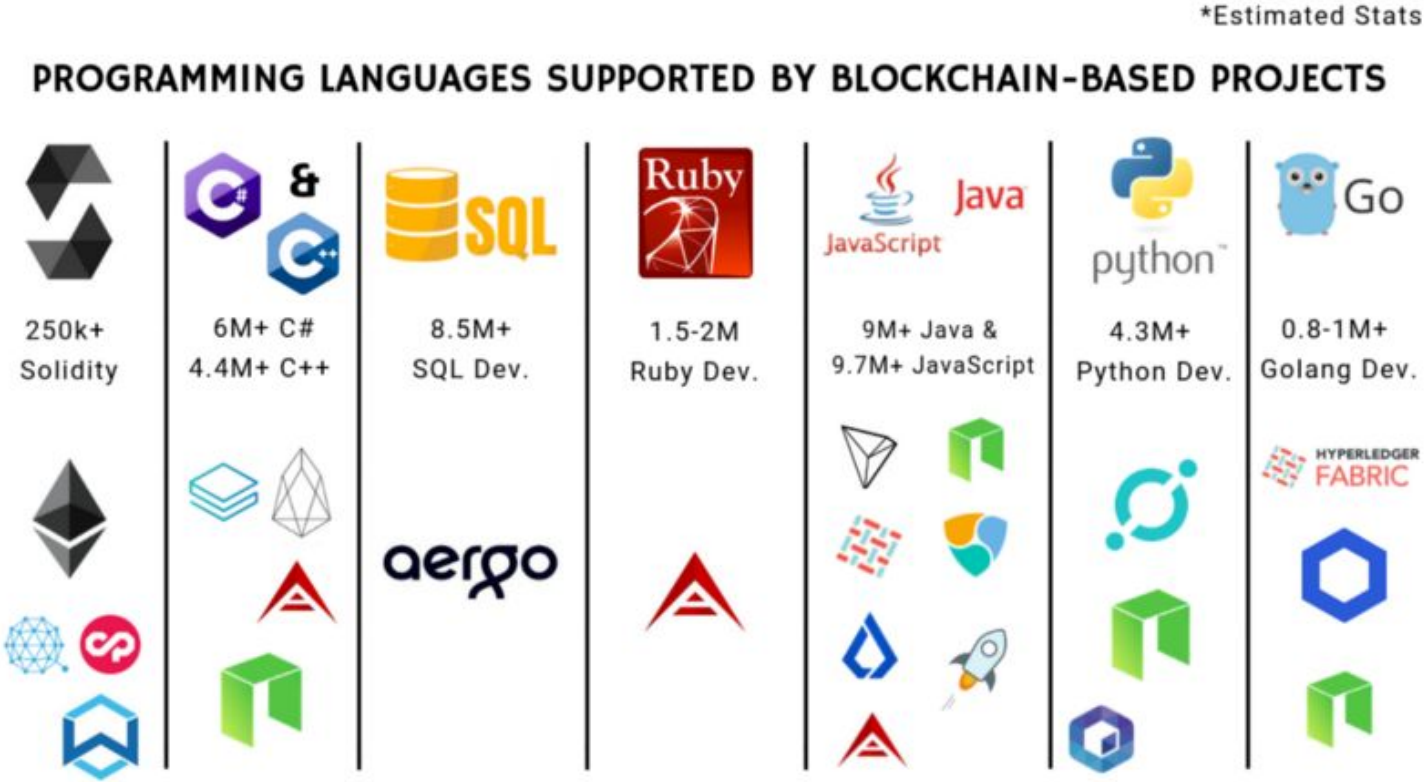
| Parameters | Golang | Java |
|---|---|---|
| **Goroutines** | Supported | Not Supported |
| **Generics** | Not Supported | Supported |
| **Threads** | More Economical | Less Economical |
| **Talent Pool** | Increasing by the day and might even surpass Java | Already very extensive and vast |
| **Learning Curve** | Simple | Comparatively Complex |
| **Security** | Checksum Database | No Pointers, Security Manager |
| **Development Cost** | Comparatively High | Comparatively Low |
| **Channel** | Not Supported | Supported |

Courtesy : Banacy Technology



# Types of Blockchain Programming

*Estimated Stats




Courtesy : Michael Drapper

# Decentralized file system-IPFS (Revelance)

- World Wide Web is a global network, i: centralized in terms of data storage.
- Servers in large server farms or cloud platforms, typically **owned by a single company.**
- Users must establish a HTTP / HTTPS connection to the specific server, which serves as the centralized point for data retrieval.

HTTP drawbacks.

- It functions effectively only for small files due to its cost-effectiveness
- Challenges : transmitting petabyte-sized data sets, managing high-volume real-time media streams, and ensuring file permanence.
- **need for greater availability and accommodating larger files in a decentralized manner.**

**Solutions :** mirrror servers and content delivery networks, effectively distribute content from an "origin" server to consumers by deliberately **storing the content close to the end users.**

- **InterPlanetary File System (IPFS)**
  - **an innovative Web3 phenomenon**, represents a decentralized network implementation, representing a significant advancement in file storage and retrieval.
  - empowers users with **greater control and a more resilient internet experience.**

Courtesy : Cointelegraph

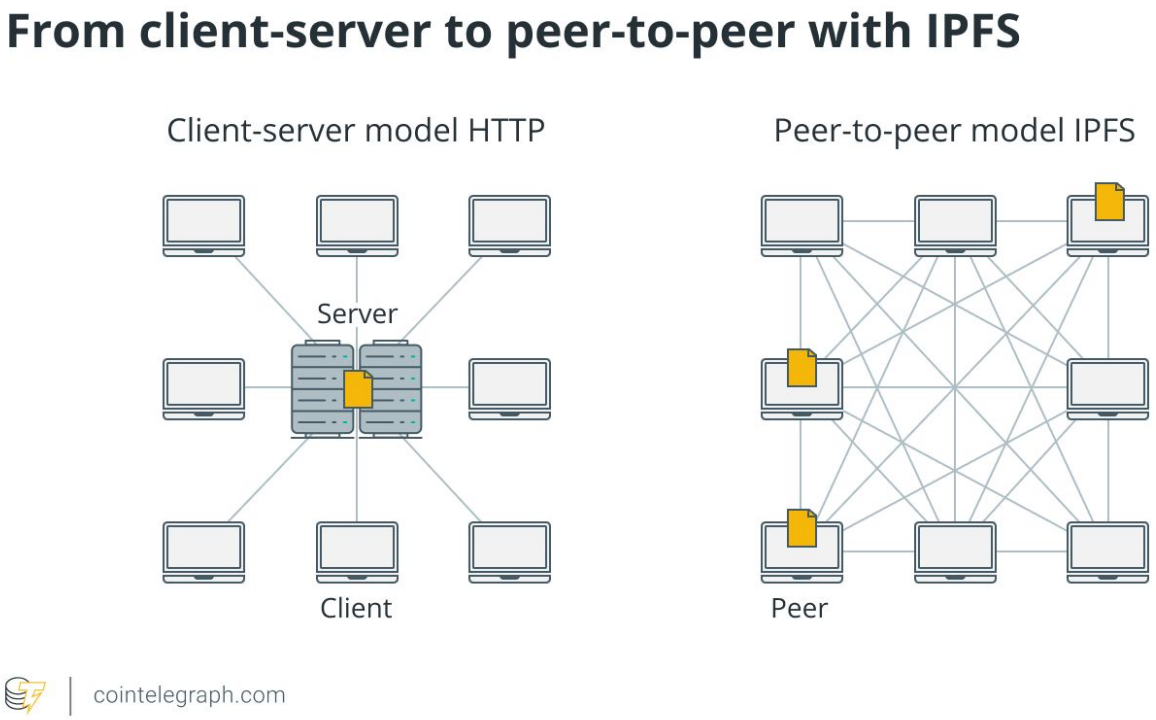
# Decentralized file system-IPFS

**What is IPFS, and how does it work?**

- the IPFS protocol is designed to establish a fully decentralized system capable of functioning across places as disconnected or as far apart as planets.
- IPFS was introduced in 2015 by computer engineer Juan Benet and maintained by the Protocol Labs team who also created Filecoin FIL $3.31 , a cryptocurrency and **cooperative digital storage and data retrieval method** based on blockchain technology.
- IPFS is a **peer-to-peer (P2P) distributed system for storing, accessing and sharing files, websites, applications and data. IPFS builds upon the decentralized environment and incorporates distributed and bandwidth-saving techniques from torrents.**

Courtesy : Cointelegraph

# Decentralized file system-IPFS




Courtesy : Cointelegraph

# Decentralized file system-IPFS

- The key distinction between the centralized and decentralized web lies in **how data is identified and retrieved**.
- In the centralized web, people depend on trusted entities to host their data and access it using location-based **Uniform Resource Locators (URLs).**
- IPFS network
  - **uses a content-addressed system,** where the content itself plays a key role in helping people to locate what they're looking for.
  - **each piece of content is identified** by a unique hash called **IPFS Content Identifier (CID)**.
    - This means that the content is stored and retrieved based on its hash rather than its location, making it much harder to censor or manipulate.

Courtesy : Cointelegraph

# Decentralized file system-IPFS

## Location addressing vs. content addressing

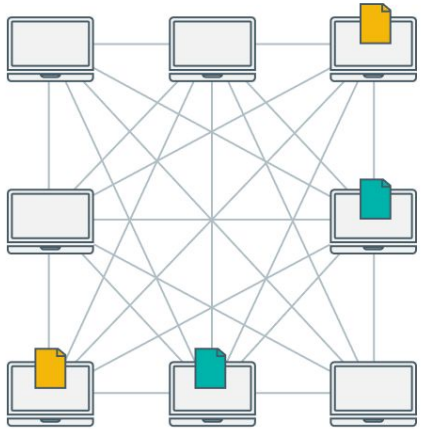
### Location addressing


https://www.nasa.gov/sites/default/files/thumbnails/image/ki67.jpg

**DNS:**

spotify.com → 35.186.224.25
google.com → 142.250.186.174
**nasa.gov** → **18.66.2.84**
backup-server.com → 172.67.182.225

### Content addressing with IPFS



**CID:**

ilysaclfdc;yzlic7tqcwlas7dqw6dkqfa
cjsfXQIsqqo92iawfsd76daq78ag9y

Different CID

Courtesy : Cointelegraph

# Decentralized file system-IPFS

The decentralized web of IPFS consists of

- interconnected computers called nodes that use **distributed hash tables (DHT),**
- a **decentralized storage system** that provides lookups and storage for the mapping of keys to values.
- In a DHT, each node is accountable for specific keys and mapped values and can effectively retrieve the corresponding value for a given key.

- IPFS nodes **store data and make it available to anyone who requests it.**
  - When a file or web page is requested, a copy of the file is cached on the requester's node.
  - As more people ask for the same data, additional cached copies are created.
  - Subsequent requests for the file can be fulfilled by any node or combination of nodes that have it. This way, the responsibility of delivering the data and fulfilling requests is shared among multiple locations, making it more efficient and accessible.

Courtesy : Cointelegraph

# Decentralized file system-IPFS

**Is IPFS a blockchain? -** IPFS and blockchain are both decentralized technologies,

- **IPFS**
  - focuses on creating a global, decentralized network for storing and sharing files.
  - It aims **to improve the efficiency and resilience of traditional web protocols** by allowing files to be stored in multiple locations, making them resistant to censorship and ensuring availability even if some nodes go offline.
  - provides a decentralized storage and distribution system
- **Blockchain**
  - serves primarily as a **decentralized ledger, recording transactions or data** in a transparent and tamper-proof manner.
  - relies on consensus mechanisms and cryptographic algorithms to ensure the integrity and security of the data stored on the chain.
  - often used for decentralized applications (DApps) and
  - involves cryptocurrencies, smart contracts and, for instance, decentralized finance (DeFi).
  - provides a decentralized and transparent way to record and verify transactions or data.

Courtesy : Cointelegraph

# Decentralized file system-IPFS

| Features | IPFS | Blockchain |
|---|---|---|
| **Primary function** | Distributed file system | Decentralized ledger |
| **Content addressing** | Yes | No (uses transaction hashes, block IDs) |
| **Data storage** | Files and hypermedia | Transactions, data or smart contracts |
| **Data retrieval** | Efficient based on content addressing | Sequential based on transaction history |
| **Immutability** | Files can be changed or updated | Data is immutable once added to the chain |
| **Use cases** | Decentralized websites, data storage, file sharing | Cryptocurrencies, DApps, supply chain, etc. |
| **Security** | Content is encrypted and distributed | Cryptographic algorithms ensure security |
| **Scalability** | Efficient distribution of files | Scalability challenges with increased data |

Courtesy : Cointelegraph

# Decentralized file system-IPFS

**What is IPFS used for?**

- delivering content globally, storing files securely and facilitating efficient file sharing.
- act as a **complementary file system for public blockchains and other P2P systems**.
- It has the potential **to enhance the scalability of DApps on platforms like Ethereum.**
- By integrating with Ethereum's smart contracts, IPFS can **provide secure and cost-effective storage capabilities within the crypto ecosystem,** improving Ethereum's overall performance.
- IPFS, **along with Filecoin, creates incentives for data storage.** This combination can play a significant role in the development and storage of the data of nonfungible tokens (NFTs).
- enhancing DApps scalability to revolutionizing NFTs by ensuring helpful data records.

Courtesy : Cointelegraph

# Decentralized file system-IPFS

**Is IPFS traceable?**

- Each IPFS node has a **public PeerID** that can be used to track the associated IP address over time by looking it up in the DHT.
- As a protocol for P2P data delivery and storage free for everyone, IPFS is a public network.
- Nodes involved in the network store data that is linked to globally consistent CIDs and broadcast their availability to other nodes through publicly accessible DHTs.
- While **IPFS traffic between nodes is encrypted**, the fundamental metadata that nodes disclose to the DHT, including their **unique node identifiers (PeerIDs) and the CIDs of data** they offer, is accessible to the public and can be tracked. Anyone can access these types of data on IPFS.

Courtesy : Cointelegraph

# Decentralized file system-IPFS

**What are the downsides of IPFS?**

- **relatively slow compared to traditional web protocols.**
- **Interoperability** is also a crucial factor for IPFS.
- **Incentivizing users to contribute their resources to the IPFS network** is another challenge. As IPFS relies on a P2P network for data distribution and storage, creating effective incentives and rewards mechanisms becomes imperative. Encouraging active participation and resource allocation from users will play a vital role in ensuring the sustainability and growth of the IPFS ecosystem.
- **Security** is an ongoing concern for any technology, including IPFS. While IPFS employs content addressing and encryption to safeguard data stored on the network, there may still be potential vulnerabilities and security risks that need to be addressed. Continuous efforts to enhance security measures and mitigate potential threats are essential to maintain the integrity and confidentiality of user data within the IPFS network.
- requiring **regular and systematic releases of new versions to incorporate upgrades.**

Courtesy : Cointelegraph