Chapter 8: Smart Contracts and Vyper.

Vyper is an experimental, contract-oriented programming language for the Ethereum Virtual Machine that strives to provide superior auditability, by making it easier for developers to produce intelligible code.

A recent study analyzed nearly one million deployed Ethereum smart contracts and found that many of these contracts contained serious vulnerabilities.

Vulnerabilities are introduced into smart contracts via code. It may be strongly argued that these and other vulnerabilities are not intentionally introduced, but regardless, undesirable smart contract code evidently results in the unexpected loss of funds for Ethereum users, and this is not ideal.

Vyper is designed to make it possible to write secure code.

One of the ways in which Vyper tries to make unsafe code harder to write is by deliberately omitting some of Solidity's features. It is important for those considering developing smart contracts in Vyper to understand what features Vyper does not have, and why. Therefore, in this section, we will explore those features and provide justification for why they have been omitted.

Vyper has no:
- Modifiers.
- Inheritance or Polymorphism.
- Inline Assembly.
- Function Overloading.
- Implicit variable typecasting.

Choosing explicit over implicit typecasting means that the developer is responsible for performing all casts. While this approach does produce more verbose code, it also improves the safety and auditability of smart contracts.

Vyper also makes use of Decorators:
@private, @public, @constant and @payable and any one can be used before the start of a function.

Moreover, Vyper's variables and functions must be ordered.

One transaction from mid-April 2018 shows the malicious transfer of over 57,896,044,618,658,100,000,000,000,000,000,000,000,000,000,000,000,000,000,000 BEC tokens. This transaction was the result of an integer overflow issue in BeautyChain's ERC20 token contract.

Vyper is a powerful and interesting new contract-oriented programming language. Its design is biased toward "correctness," at the expense of some flexibility.