VYPER

Onboarding the next generation of web3 developers

benny

Developer Advocate - Vyper

ETH Riyadh 2024

What's Vyper?

- A Pythonic smart contract programming language for the EVM
- Designed for simplicity, safety and efficiency
- Used by major protocols across the Ethereum ecosystem


Curve


Frax


Lido


Perpetual Protocol


Velodrome

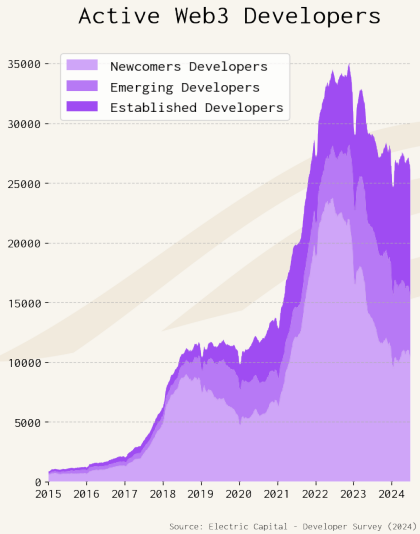

Yearn

Ripe Finance

What's Vyper?

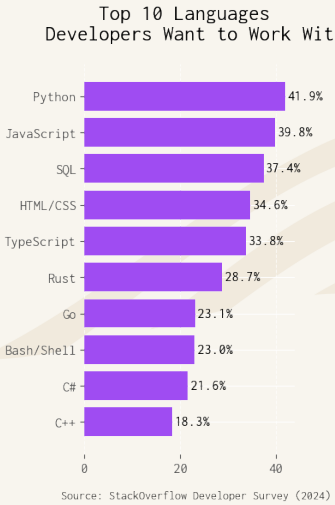
- Vyper is currently the second most popular smart contract language in the Ethereum ecosystem, after Solidity
- Vyper is positioning itself to be the first choice for all new web3 developers:
  - Vyper is easy, safe and efficient by default
  - You don’t need to be an EVM wizard to write good, optimized contracts

Web3 Developer Landscape



- There are less than 30,000 active developers accross the industry, less than in many individual tech companies
- New developers are the largest share of all Web3 developers
- As the industry continues to grow, we will see influxes of new developers - both new graduates and experienced engineers

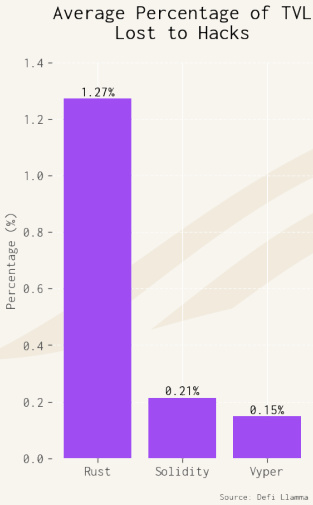
Vyper is Beginner Friendly



- **Vyper's syntax is very similar to Python**, a language that most developers are not only familiar with but **want** to work with

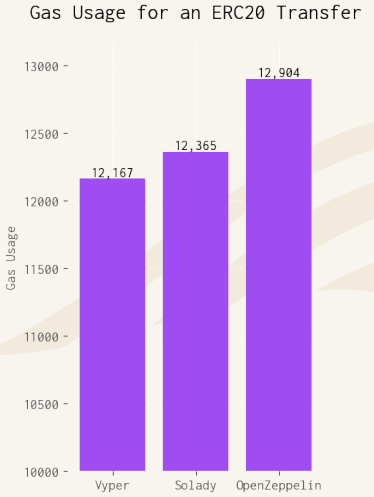
```
@external
def hello_year() -> uint256:
  print("Hello, World!")
  return 2024
```

Vyper is Safe by Default



- **No footguns**: Safe arithmetic, no operator or function overloading, no modifiers
- **Modularity over inheritance**: Inheritance is hard to read, hard to audit and has led to multiple exploits
- **Fixed gas limits**: Finite length loops and no recursive calls prevent gas limit attacks.
- **No assembly**: Low level attempts at optimization are another common source of vulnerabilities

Vyper is Optimal by Default



- Vyper contracts have **lower gas usage** and up to **50% lower bytecode size** compared to Solidity
- Optimizations are done directly by the compiler. There is no need for tricks and hacks to get optimal efficiency.
- This makes for contracts that are **easier to write and easier to read**

Vyper is Optimal by Default

```
assembly {
 let from_ := shl(96, from)
 mstore(0x0c, or(from_, _BALANCE_SLOT_SEED))
 let fromBalanceSlot := keccak256(0x0c, 0x20)
 let fromBalance := sload(fromBalanceSlot)
 if gt(amount, fromBalance) {
  mstore(0x00, 0xf4d678b8) // `InsufficientBalance()`.
  revert(0x1c, 0x04)
 }
 sstore(fromBalanceSlot, sub(fromBalance, amount))
 mstore(0x00, to)
 let toBalanceSlot := keccak256(0x0c, 0x20)
 sstore(toBalanceSlot, add(sload(toBalanceSlot), amount))
 mstore(0x20, amount)
 ...
}
```

Solady

```
self._before_token_transfer(owner, to, amount)
owner_balanceOf: uint256 = self.balanceOf[owner]
assert (owner_balanceOf >= amount,
"erc20: transfer amount exceeds balance")
self.balanceOf[owner] = unsafe_sub(owner_balanceOf, amount)
self.balanceOf[to] = unsafe_add(self.balanceOf[to], amount)
log IERC20.Transfer(sender=owner, receiver=to, value=amount)
self._after_token_transfer(owner, to, amount)
...
```

Snekmate

Vyper is Optimal by Default

```
@external
def get_six() -> uint256:
  a: uint256 = 3
  b: uint256 = 2
  return a * b
```

Equivalent in Vyper

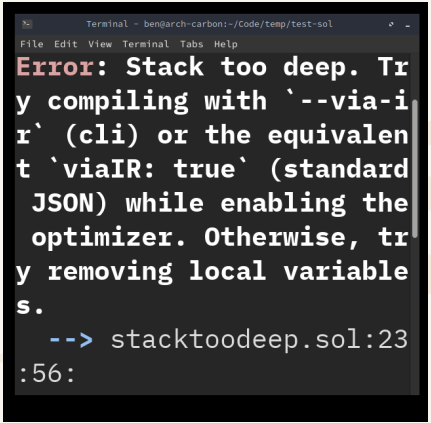
```
@external
def get_six() -> uint256:
  return 6
```

```
function get_six()
public pure returns (uint256) {
  uint256 a = 3;
  uint256 b = 2;
  return a * b;
}
```

Different in Solidity

```
function get_six()
public pure returns (uint256) {
  return 6;
}
```

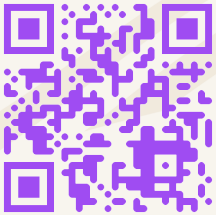
Vyper is Readable



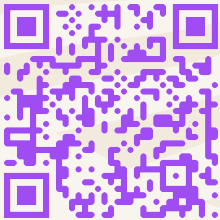
- **No stack too deep errors** forcing rewrites that impede readability
- **Faster, easier and cheaper** to audit

Vyper is for Everyone

- Vyper is easy to use, easy to set up: `pip install vyper`
- You don't have to start with custodial contracts: util contracts for analytics, zaps for DeFi, simple on-chain games are all great to get started.



Telegram Channel



Discord Server

Cyfrin Course