A primer on the Ethereum Blockchain and Smart Contracts using Python and Serpent

Stefano Fioravanzo
CS Master Student @ UniTn
Junior Researcher @ Fondazione Bruno Kessler

stefano.fioravanzo@gmail.com
@sfioravanzo

PyCon Nove

THEORY

- BITCOIN INTRO
- IMMUTABILITY
- CONSENSUS
- LIMITATIONS
- ETHEREUM
- EVM - SMART CONTRACTS

- TESTING & DEPLOY
- RUNNING A PRIVATE CHAIN
- WRITING SMART CONTRACTS
- PYTHON EHT ECOSYSTEM

PRACTICE

ORIGINS

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto (2008)

"A lot of people automatically dismiss e-currency as a lost cause because of all the companies that failed since the 1990's. ***I hope it's obvious it was only the centrally controlled nature of those systems that doomed them****. I think this is the first time we're trying a decentralized, non-trust-based system."*
- Satoshi Nakamoto

DISTRIBUTED (CONSENSUS)

●

OPEN ACCESS

●

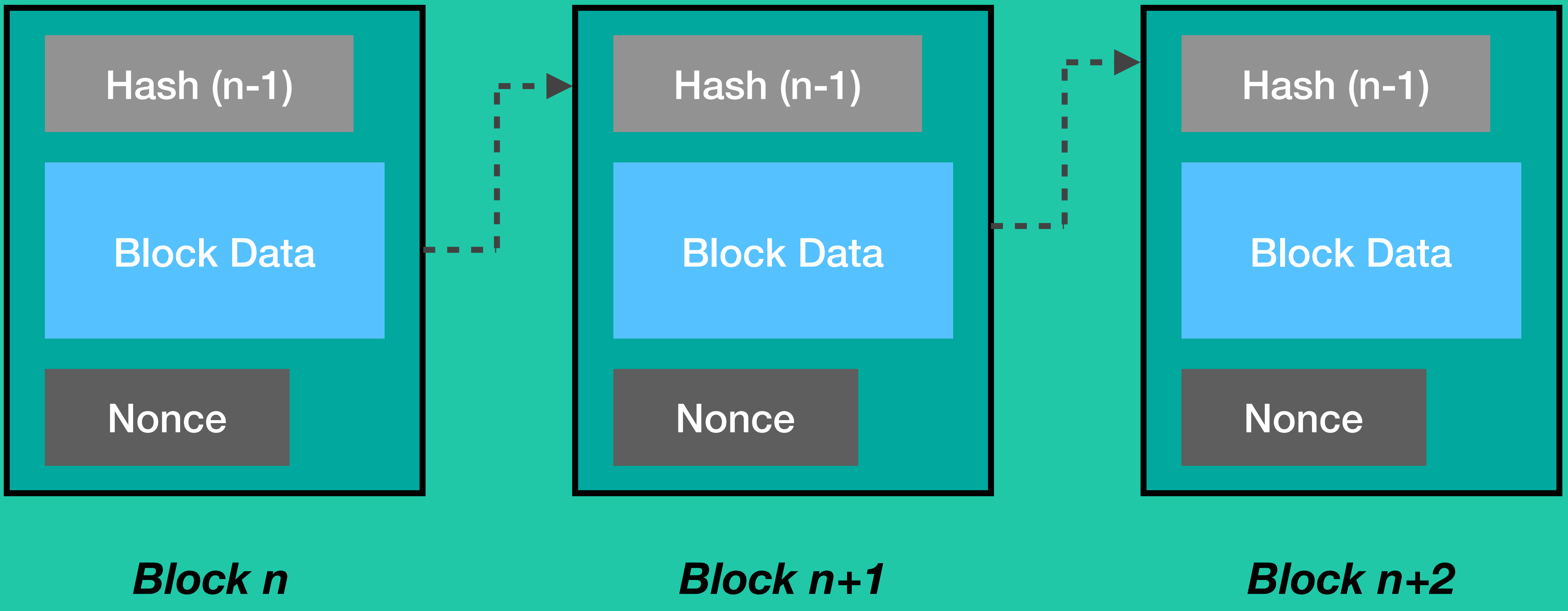
IMMUTABILITY

●

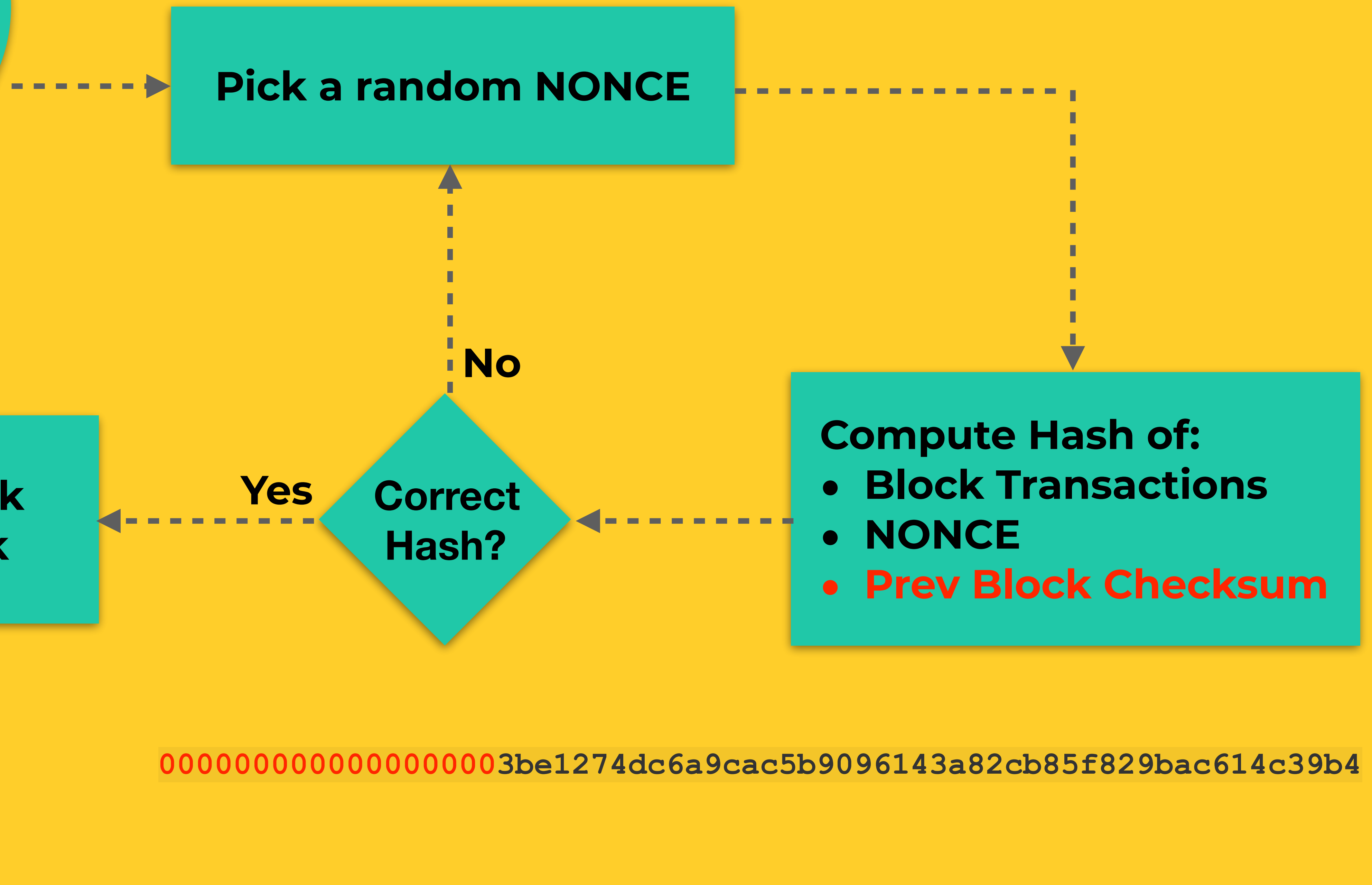
PROVENANCE

IMMUTABILITY · CONSENSUS

1. Hash functions: unpredictable and non-invertible
2. Blocks include a list of transactions
3. Block header includes hash of previous block
4. Solve hard computation puzzle: PoW



Mining



```
0000000000000000003be1274dc6a9cac5b9096143a82cb85f829bac614c39b4
```

MINING REWARD

Mining is incentivized with the inclusion of a reward in the new mined block

This is how new coins are created

BITCOIN ISSUES

Lack of Turing-Compl.

- **No Loops** Allowed
- **Cumbersome programming** of smart contracts

Value-blindness

- Cannot **finely** control withdrawals
- Need **hacky operations** to achieve complex transaction logics

Lack of state

- UTXO can either be **spent or unspent**
- Lack of a **“contract state”**

Blockchain-blindness

- UTXO are **blind** to BC data
- This **limits** possible applications

ETHEREUM

The Ethereum Whitepaper

Vitalik Buterin (2012)

*When I came up with Ethereum, my first first thought was, 'Okay, this thing is **too good to be true**.' As it turned out, the core Ethereum **idea was good** - fundamentally, **completely sound***
- Vitalik Buterin

ETHEREUM EVM

●

IMPROVED PoW

●

GHOST PROTOCOL

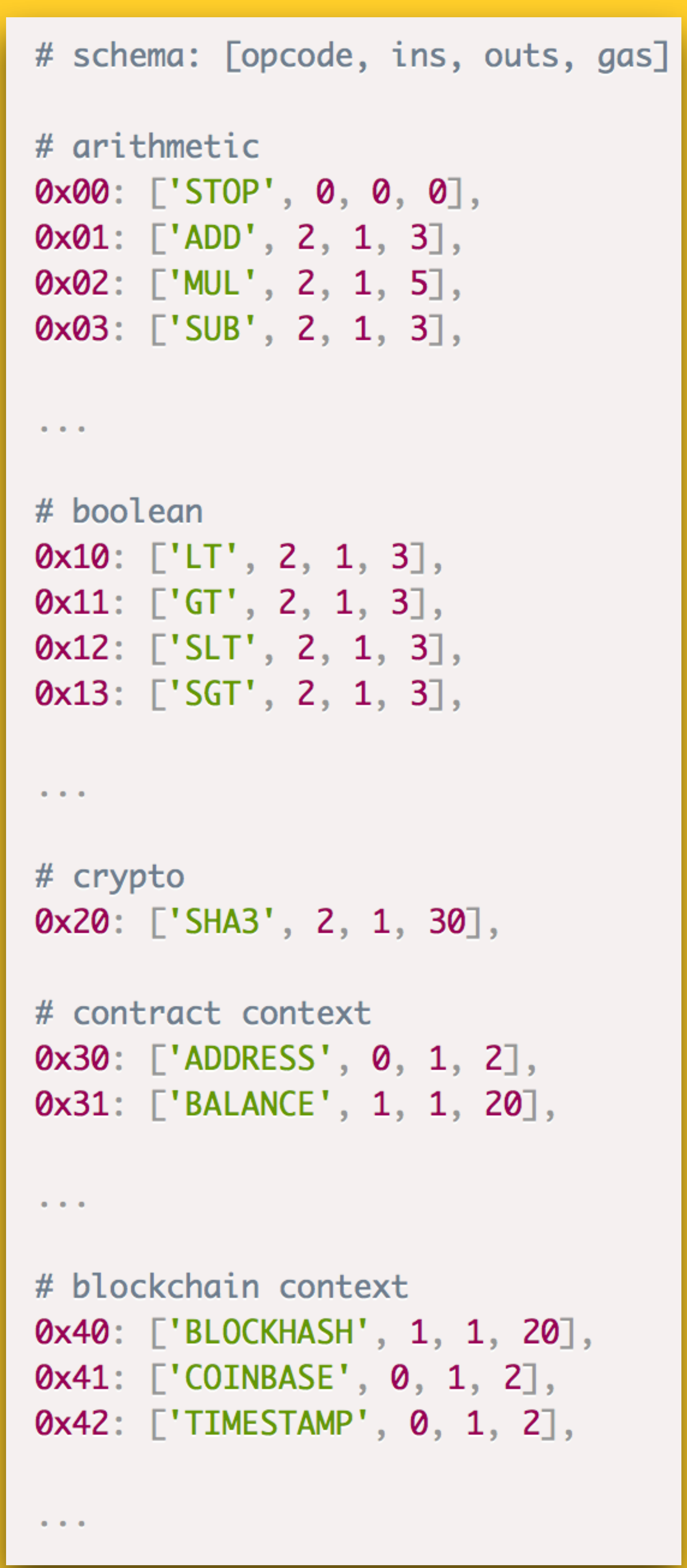
●

FASTER CONFRIMATIONS

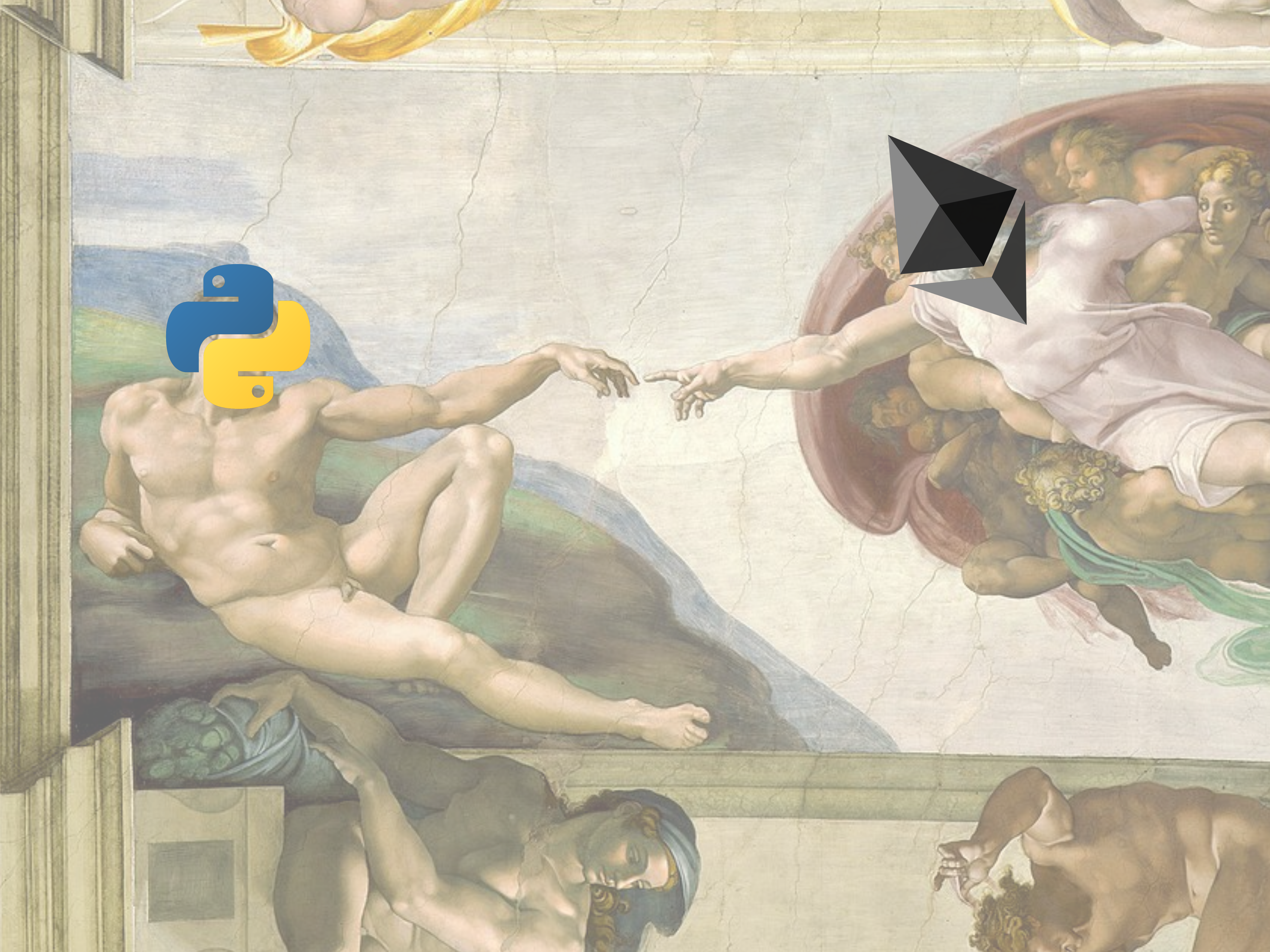
EVM

- Program execution is **sandboxed**
- **Stack-based** virtual machine
- Program execution completely **deterministic**
- **Ephemeral memory** byte-array
- **Persistent** storage tree
- **Prevents** DoS Attacks - **GAS**

OVER 100 OPCODES



```
# schema: [opcode, ins, outs, gas]

# arithmetic
0x00: ['STOP', 0, 0, 0],
0x01: ['ADD', 2, 1, 3],
0x02: ['MUL', 2, 1, 5],
0x03: ['SUB', 2, 1, 3],

...

# boolean
0x10: ['LT', 2, 1, 3],
0x11: ['GT', 2, 1, 3],
0x12: ['SLT', 2, 1, 3],
0x13: ['SGT', 2, 1, 3],

...

# crypto
0x20: ['SHA3', 2, 1, 30],

# contract context
0x30: ['ADDRESS', 0, 1, 2],
0x31: ['BALANCE', 1, 1, 20],

...

# blockchain context
0x40: ['BLOCKHASH', 1, 1, 20],
0x41: ['COINBASE', 0, 1, 2],
0x42: ['TIMESTAMP', 0, 1, 2],

...
```



ETHEREUM PYTHON TOOLS

1. Ethereum node

- **go-ethereum** (Geth)
- Parity - Rust
- cpp-ethereum - C++
- pyethapp (Python)
- …

2. RPC Library

Connect to an Eth node :

- Web3.js
- **Web3.py**
- Web3j
- …

3. SC Language

- **Solidity** (JS Like)
- **Serpent** (Python Like)
- **LLL** (List Like)
- **Vyper** (?)

4. Testing & Deploy

- `pyethereum.test`
- Populus framework

ETHEREUM PYTHON TOOLS

1. Ethereum node

- **go-ethereum** (Geth)
- Parity - Rust
- cpp-ethereum - C++
- pyethapp (Python)
- …

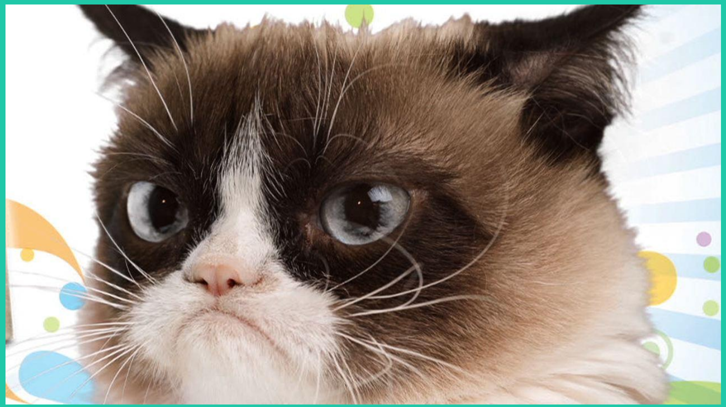
2. RPC Library

Connect to an Eth node :

- Web3.js
- **Web3.py**
- Web3j
- …

3. SC Language

-
-
-
-



Where's Python??

4. Testing & Deploy

- `pyethereum.test`
- Populus framework

SERPENT

Designed to be **very similar to Python**, despite some differences:

- No list comprehensions, dictionaries or other advanced features
- No classes, just contract functions
- Persistent storage with `data` keyword
- Call other contracts with `extern`
- Short string represented as integers
- Serpent numbers wrap around 2^256
- Access to bc and message state (`tx`, `msg`, `block`)

CROWDFUNDING EXAMPLE IN SERPENT

```
data recipient
data goal
data deadline
data contrib_total
data contrib_count
data contribs[](sender, value)    # define infinite array

def create_campaign(recipient:str, goal, timelimit):
    # campaign already exists
    if self.recipient:
        return("Already initialized")
    self.recipient = recipient
    self.goal = goal
    self.deadline = block.timestamp + timelimit
    return self.recipient

def contribute():
    # Update contribution total
    total_contributed = self.contrib_total + msg.value
    self.contrib_total = total_contributed

    # Record new contribution
    sub_index = self.contrib_count
    self.contribs[sub_index].sender = msg.sender
    self.contribs[sub_index].value = msg.value
    self.contrib_count = sub_index + 1

    # refund if expired or goal reached
    self.refund()
    return self.contrib_total
```

```
def refund():
    # If expired, refund all contributors
    if block.timestamp > self.deadline or
        total_contributed >= self.goal::
        i = 0
        c = self.contrib_count
        while i < c:
            send(self.contribs[i].sender,
                 self.contribs[i].value)
            i += 1
        self.clear()
    return(2)

def progress_report():
    return(self.contrib_total)

def clear():
    if self == msg.sender:
        self.recipient = 0
        self.goal = 0
        self.deadline = 0
        c = self.contrib_count
        self.contrib_count = 0
        self.contrib_total = 0
        i = 0
        while i < c:
            self.contribs[i].sender = 0
            self.contribs[i].value = 0
            i += 1
```

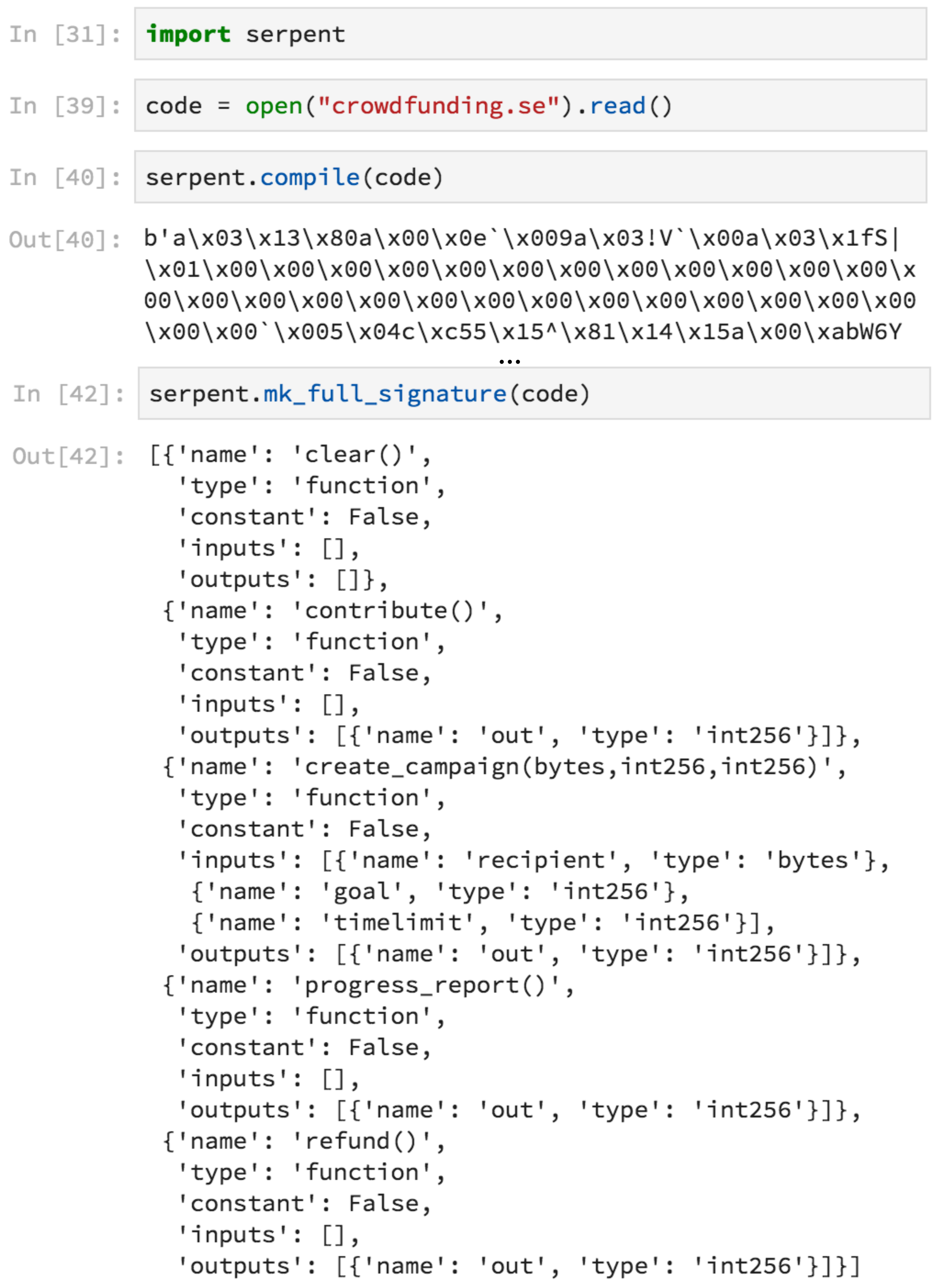
BYTECODE & ABI

Compiled code is a **binary representation of EVM opcodes**. Can easily decode it, but var and func names are **hashed**.

ABI

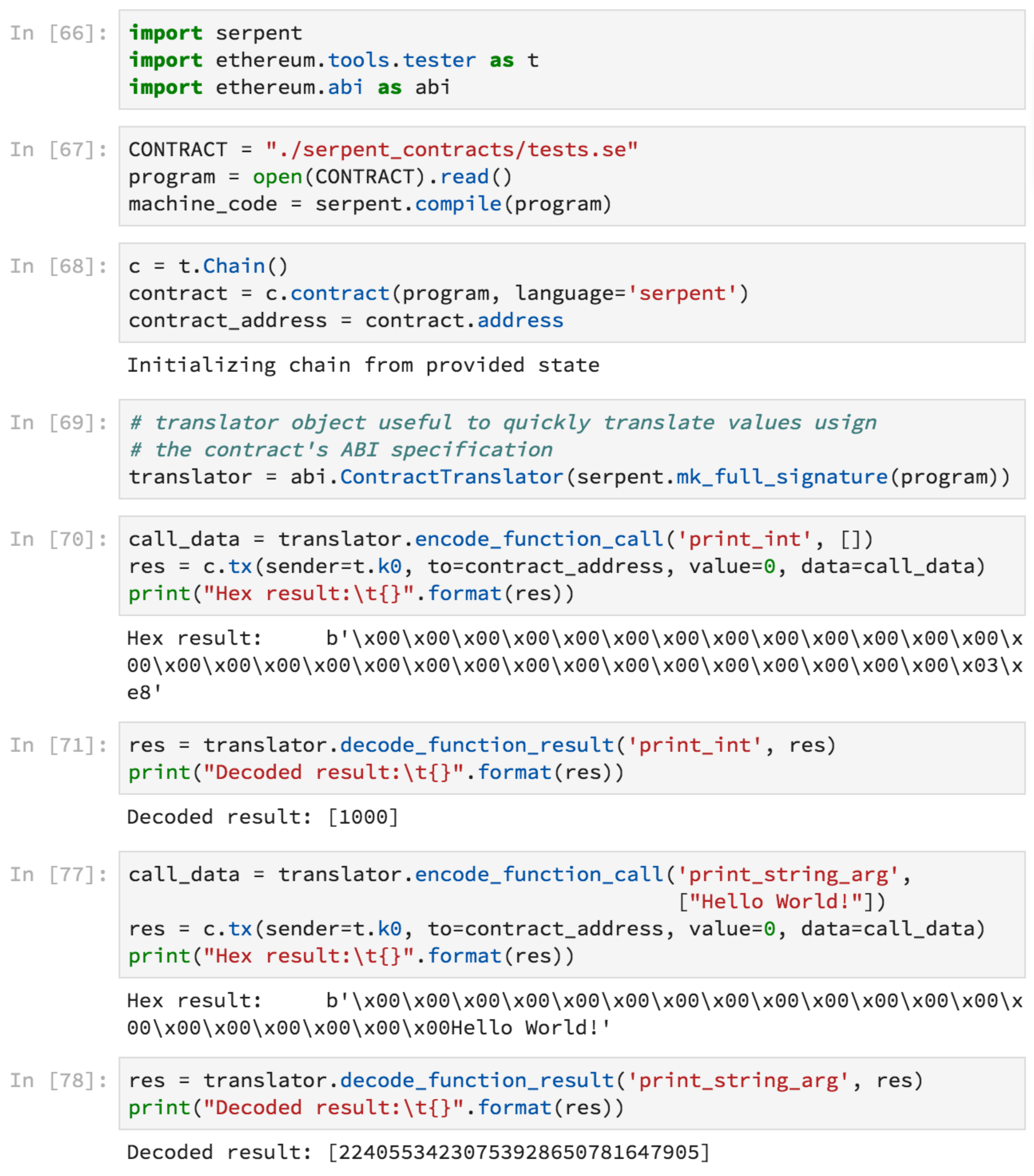
(Application Binary Interface)

- Standard “API”
- Data encoded according to spec
- Names are hashed and request properly encoded



```
In [31]: import serpent

In [39]: code = open("crowdfunding.se").read()

In [40]: serpent.compile(code)

Out[40]: b'a\x03\x13\x80a\x00\x0e`\x009a\x03!V`\x00a\x03\x1fS|
         \x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
         00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
         \x00\x00`\x005\x04c\xc55\x15^\x81\x14\x15a\x00\xabW6Y
                                    ...
In [42]: serpent.mk_full_signature(code)

Out[42]: [{'name': 'clear()',
           'type': 'function',
           'constant': False,
           'inputs': [],
           'outputs': []},
          {'name': 'contribute()',
           'type': 'function',
           'constant': False,
           'inputs': [],
           'outputs': [{'name': 'out', 'type': 'int256'}]},
          {'name': 'create_campaign(bytes,int256,int256)',
           'type': 'function',
           'constant': False,
           'inputs': [{'name': 'recipient', 'type': 'bytes'},
            {'name': 'goal', 'type': 'int256'},
            {'name': 'timelimit', 'type': 'int256'}],
           'outputs': [{'name': 'out', 'type': 'int256'}]},
          {'name': 'progress_report()',
           'type': 'function',
           'constant': False,
           'inputs': [],
           'outputs': [{'name': 'out', 'type': 'int256'}]},
          {'name': 'refund()',
           'type': 'function',
           'constant': False,
           'inputs': [],
           'outputs': [{'name': 'out', 'type': 'int256'}]}]
```

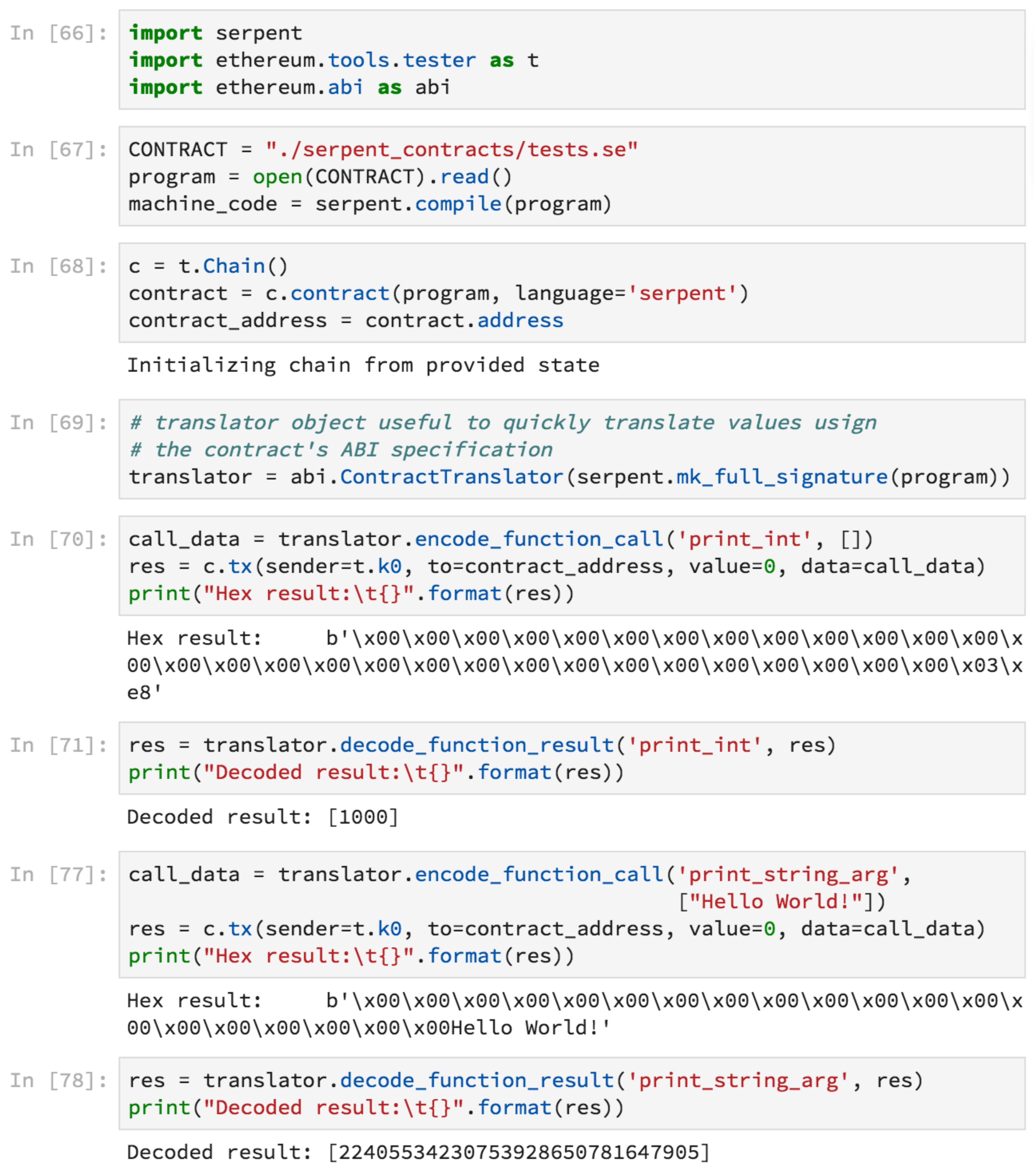
TESTING



```
In [66]: import serpent
         import ethereum.tools.tester as t
         import ethereum.abi as abi

In [67]: CONTRACT = "./serpent_contracts/tests.se"
         program = open(CONTRACT).read()
         machine_code = serpent.compile(program)

In [68]: c = t.Chain()
         contract = c.contract(program, language='serpent')
         contract_address = contract.address

         Initializing chain from provided state

In [69]: # translator object useful to quickly translate values usign
         # the contract's ABI specification
         translator = abi.ContractTranslator(serpent.mk_full_signature(program))

In [70]: call_data = translator.encode_function_call('print_int', [])
         res = c.tx(sender=t.k0, to=contract_address, value=0, data=call_data)
         print("Hex result:\t{}".format(res))

         Hex result:     b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
         00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x
         e8'

In [71]: res = translator.decode_function_result('print_int', res)
         print("Decoded result:\t{}".format(res))

         Decoded result: [1000]

In [77]: call_data = translator.encode_function_call('print_string_arg',
                                                     ["Hello World!"])
         res = c.tx(sender=t.k0, to=contract_address, value=0, data=call_data)
         print("Hex result:\t{}".format(res))

         Hex result:     b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
         00\x00\x00\x00\x00\x00\x00Hello World!'

In [78]: res = translator.decode_function_result('print_string_arg', res)
         print("Decoded result:\t{}".format(res))

         Decoded result: [22405534230753928650781647905]
```

`pyethereum.tester` provides the means to test your smart contracts easily **without the need to start a private node**

TESTING



```
In [66]: import serpent
         import ethereum.tools.tester as t
         import ethereum.abi as abi

In [67]: CONTRACT = "./serpent_contracts/tests.se"
         program = open(CONTRACT).read()
         machine_code = serpent.compile(program)

In [68]: c = t.Chain()
         contract = c.contract(program, language='serpent')
         contract_address = contract.address

         Initializing chain from provided state

In [69]: # translator object useful to quickly translate values usign
         # the contract's ABI specification
         translator = abi.ContractTranslator(serpent.mk_full_signature(program))

In [70]: call_data = translator.encode_function_call('print_int', [])
         res = c.tx(sender=t.k0, to=contract_address, value=0, data=call_data)
         print("Hex result:\t{}".format(res))

         Hex result:     b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
         00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x
         e8'

In [71]: res = translator.decode_function_result('print_int', res)
         print("Decoded result:\t{}".format(res))

         Decoded result: [1000]

In [77]: call_data = translator.encode_function_call('print_string_arg',
                                                     ["Hello World!"])
         res = c.tx(sender=t.k0, to=contract_address, value=0, data=call_data)
         print("Hex result:\t{}".format(res))

         Hex result:     b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
         00\x00\x00\x00\x00\x00\x00Hello World!'

In [78]: res = translator.decode_function_result('print_string_arg', res)
         print("Decoded result:\t{}".format(res))

         Decoded result: [22405534230753928650781647905]
```

Other libraries built on top of `pyethereum.tester`

`eth-tester`

`ethereum-tester-client`
`eth-testrpc`
OLD!

Go Implementation of a **full Ethereum node**

- Wide range of net compatibility
- JSON-RPC endpoints
- Currently the **best** implementation of the Ethereum protocol

Geth

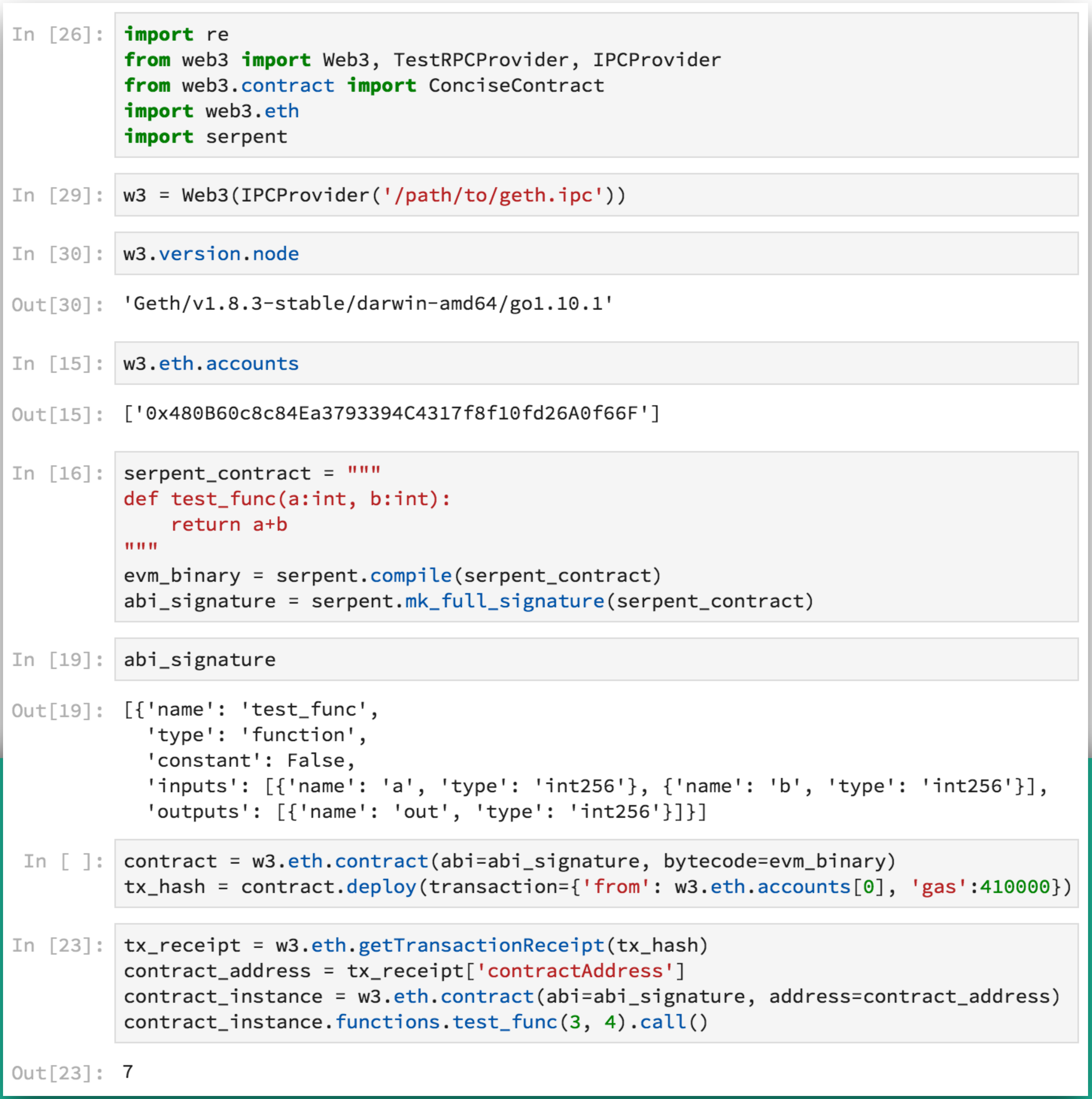
```
# start a full node - connected to main net
geth console

# start a full node - connected to PoW (Ropsten) net
geth --testnet

# start a full node - connceted to PoA (Rinkeby) net
geth --rinkeby

# setup single node private net
geth --dev --rpc --ipcpath ~/custom/path/geth.ipc --datadir ~/custom/path/mytestnet
```

If you really can not avoid using Python:

`pip install py-geth`

```python
from geth import LoggingMixin, DevGethProcess

# Use LoggingMixin to redirect geth process
# stdout and stderr output to ./logs
class MyLoggingGeth(LoggingMixin, DevGethProcess):
    pass
geth = MyLoggingGeth(chain_name="my_dev_chain",
                     base_dir="./my_dev_chain",
                     overrides=overrides)
# Start the process
geth.start()
```

Web3.py



```
In [26]: import re
         from web3 import Web3, TestRPCProvider, IPCProvider
         from web3.contract import ConciseContract
         import web3.eth
         import serpent

In [29]: w3 = Web3(IPCProvider('/path/to/geth.ipc'))

In [30]: w3.version.node

Out[30]: 'Geth/v1.8.3-stable/darwin-amd64/go1.10.1'

In [15]: w3.eth.accounts

Out[15]: ['0x480B60c8c84Ea3793394C4317f8f10fd26A0f66F']

In [16]: serpent_contract = """
         def test_func(a:int, b:int):
             return a+b
         """
         evm_binary = serpent.compile(serpent_contract)
         abi_signature = serpent.mk_full_signature(serpent_contract)

In [19]: abi_signature

Out[19]: [{'name': 'test_func',
           'type': 'function',
           'constant': False,
           'inputs': [{'name': 'a', 'type': 'int256'}, {'name': 'b', 'type': 'int256'}],
           'outputs': [{'name': 'out', 'type': 'int256'}]}]

 In [ ]: contract = w3.eth.contract(abi=abi_signature, bytecode=evm_binary)
         tx_hash = contract.deploy(transaction={'from': w3.eth.accounts[0], 'gas':410000})

In [23]: tx_receipt = w3.eth.getTransactionReceipt(tx_hash)
         contract_address = tx_receipt['contractAddress']
         contract_instance = w3.eth.contract(abi=abi_signature, address=contract_address)
         contract_instance.functions.test_func(3, 4).call()

Out[23]: 7
```

Geth output:

```
> Submitted contract creation
> Commit new mining work
> Successfully sealed new block
> 🔨 mined potential block
```

I missed to tell you one **important** thing…

I missed to tell you one **important** thing…

Being a low-level language, Serpent is **NOT RECOMMENDED** for building applications unless you really really know what you're doing. The creator recommends **Solidity** as a default choice, **LLL** if you want close-to-the-metal optimizations, or **Viper** if you like its features though it is **still experimental**.

https://blog.zeppelin.solutions/serpent-compiler-audit-3095d1257929

VYPER

- Designed to be the “successor” of Serpent
- Still in alpha stage (Beta soon?)
- Python compiler
- Less powerful *by design*
- Prevents writing *unsafe* and *misleading* code
- Not a replacement of other languages

There is no complete solution for a Python based SC language at the moment!

POPULUS

- The **only** Python framework for SC
- Helps in **Deploy** & **Test**
- Provides only **Solidity** support
- **Automatic** test module generation
- Tests are run against an **in-memory** Eth BC
- Automatically manages connection to Eth net
- **Programmatically** deploy SC to chain

We can expect Serpent (?) or Viper support in the near future

And now that you know what to do... **BE CAREFUL!**

- You **don’t need much** to start developing for the Ethereum blockchain
- It is extremely important to **know the underlying technology**
- Once you play around with the main net....**real money is at stake**!
- **Store** your Smart Contract’s ABI

And now that you know what to do... **BE CAREFUL!**

- You **don't need much** to start developing for the Ethereum blockchain
- It is extremely important to **know the underlying technology**
- Once you play around with the main net....**real money is at stake**!
- **Store** your Smart Contract's ABI

- Be aware of execution costs:
    - *Contract creation* **COSTS** money
    - *Code execution* **COSTS** money
    - *Data Storage* **COSTS** money
    - *Memory allocation* **COSTS** money

60+ tips: `http://populus.readthedocs.io/en/latest/gotchas.html`

Thank You!

Full tutorial at:
github.com/StefanoFioravanzo/ethereum-python-tutorial

stefano.fioravanzo@gmail.com
@sfioravanzo

PyCon Nove