



BlockchainPro



TOOPL00X

Ethereum: 30 Porad i Ciekawostek

DLA DEVELOPERÓW SOLIDITY

Prawa autorskie: Michał Załęcki, BlockchainPro

Zabronione jest kopiowanie, przetwarzanie i rozpowszechnianie bez pisemnej zgody autora lub wydawcy.

DTP: Jan Mickiewicz

Wydanie I
Wrocław, 2018/08/08



BlockchainPro



TOOPL00X

Spis Treści

#SPIS_TREŚCI#SP

SPIS TREŚCI	5
WSTĘP	7
ZAŁOŻENIA	9
PORADY	12
#1 Jakie znaczenie ma wielkość liter w adresie?	13
#2 Jaka jest różnica między adresem, a kluczem publicznym w Ethereum?	14
#3 Jaka jest różnica między SHA-3, a Keccak256?	15
#4 Jaka jest różnica między domyślną widocznością funkcji oraz zmiennej?	16
#5 Jak wiele transakcji mieści blok w Ethereum?	17
#6 Jak odczytać wartości zwróconą z funkcji podczas transakcji?	18
#7 Jak stwierdzić czy dany smart contract wspiera dane ABI?	19
#8 Jak wysłać transakcje za darmo na mainnet?	20
#9 Jak podróżować w czasie na Ethereum?	21
#10 Jak zamienić string na bytes i bytes na string korzystając z Web3?	22
#11 Jak połączyć się z Infura poprzez WebSockets?	23
#12 Jak wymusić transfer etherów do smart contractu?	24
#13 Jak zwrócić wiele wartości z funkcji?	25
#14 Jak modyfikator może wywołać funkcję wiele razy?	26
#15 Jak prosto zamienić wartości dwóch zmiennych?	27
#16 Jak ustawić niestandardowe wiadomości o błędach?	28
#17 Jak obliczyć function selector i przetestować wyjątek?	29
#18 Jak wykorzystać eventy jako tani storage?	30
#19 Jak wykorzystać gas dostępny w fallback function?	31
#20 Jak można logować eventy?	32
#21 Jak debugować testy z wykorzystaniem eventów?	33
#22 Jak użyć parametrów domyślnych?	34
#23 Jak połączyć dwa stringi?	35
#24 Jak zwrócić strukturę (ang. struct)?	36
#25 Jak zwrócić strukturę (ang. struct) z wykorzystaniem ABIEncoderV2?	37
#26 Jak pokolorować składnię Solidity na GitHub?	38
#27 Jak zweryfikować składnię kodu Solidity?	39
#28 Jak przeanalizować pokrycie testami kodu Solidity?	40
#29 Jak twoje funkcje mogą przysłonić te wbudowane?	41
#30 Jak działa podnoszone zmiennych (ang. hoisting)?	42
MATERIAŁY DODATKOWE	44
PODSUMOWANIE	46



Wstep

#WSTĘP #WSTĘP

Nazywam się Michał Załęcki. Pracuję w Tooploox jako Senior Software Engineer gdzie zajmuję się rozwiązaniami opartymi o Ethereum oraz Hyperledger.

Mam nadzieję, że bez względu na Twoje doświadczenie z Ethereum znajdziesz w tym krótkim poradniku przynajmniej kilka wskazówek, o których jeszcze nie słyszałeś lub nie słyszałaś. Część z tych porad, jak i wiele innych została lub zostanie opublikowana na facebooku BlockchainPro, zachęcam do polubienia i obserwowania!

Na końcu znajdziesz listę materiałów dodatkowych, z którymi zapoznanie się uważam za świetną inwestycję czasu.

Polska wersja tego ebooka dostępna jest za darmo dzięki BlockchainPro.pl gdzie prowadzę szkolenia dla developerów zainteresowanych wytwarzaniem zdecentralizowanych aplikacji opartych o Ethereum blockchain.

Michał Załęcki

A large, light gray, stylized letter 'S' watermark is centered on the page, spanning most of its height and width. It has a modern, rounded, and slightly irregular design.

Założenia

#ZAŁOŻENIA#ZAŁOŻENIA#ZAŁOŻENIA

W niniejszym ebooku przyjęto następujące założenia:

#1

Przykłady w Solidity zostały opracowane w oparciu o wersję 0.4.24

#2


Przykłady z Web3 zostały opracowane w oparciu o wersję 1.0.0-beta.34 lub 0.20.1 w przypadku użycia geth console.

#3

Przykłady w Python zostały opracowane w oparciu o wersję 3.6.4

-10%
WSZYSTKIE
KURSY*
na blockchainpro.pl
z kodem:
EBOOK

*Promocja obowiązuje na wszystkie szkolenia indywidualne do 2018/12/31.

A large, light gray, stylized letter 'S' is centered in the background of the page. It has a modern, rounded, and slightly irregular design, resembling a calligraphic or hand-drawn style.

Porady

#PORADY#PORADY

W tej części ebooka znajdziesz 30 porad i ciekawostek z zakresu programowania smart contractów na Ethereum.

Nie ma jednego źródła, z którego pochodzą, a ich poziom jest mocno zróżnicowany. Mam nadzieję, że doświadczeni w temacie znajdą przynajmniej kilka nowych informacji, a początkujący inspirację do zgłębienia swojej wiedzy.

Poza moimi doświadczeniami z Solidity bardzo duży wpływ na treść miały tematy, które przewinęły się przez społeczność redditu na r/ethereum i r/ethdev. Serdecznie polecam dołączenie do tych subredditów szczególnie jeżeli jesteś zainteresowany lub zainteresowana tematem programowania na Ethereum, ale też ogólnie rozwojem technologii blockchain.

Miłej zabawy!

#1#SOLIDITY#1#S

Porada #1 Jakie znaczenie ma wielkość liter w adresie?

Jak zawsze, to zależy. Bez względu na wielkość liter dany 40 literowy hash oznacza 20 bajtowy adres. Specyfikacja EIP-55 określa jednak w jaki sposób można wykorzystać różną wielkość liter do sprawdzenia poprawności adresu.

```
> web3.utils.isAddress("0xf17f52151EbeF6C7334FAD080c5704D77216b732")  
true  
> web3.utils.isAddress("0xF17f52151EbeF6C7334FAD080c5704D77216b732")  
false
```

#2#ETHEREUM#2

Porada #2

Jaka jest różnica między adresem, a kluczem publicznym w Ethereum?

Określanie adresu mianem klucza publicznego jest błędnym skrótem myślowym. Adres możemy jednak obliczyć z klucza publicznego ECDSA. Adres to 20 ostatnich bajtów (40 znaków w hex) hasha Keccak z 64-bajowego klucza publicznego.

```
# python
import sha3
key = bytearray.fromhex("<64 bytes public key>")
keccak = sha3.keccak_256()
keccak.update(key)
keccak.hexdigest()[24:]
# address: 0xf785c18e3c596a66119128f51053616b19865ebd
```

#3#SOLIDITY#3#S

Porada #3

Jaka jest różnica między SHA-3, a Keccak256?

W Solidity, funkcja sha3 jest aliasem funkcji keccak256 i korzysta z Keccak (zwycięskiego algorytmu w konkursie NIST na funkcję hashującą SHA-3). W innych językach/bibliotekach sha3 będzie implementować standard SHA-3 FIPS 202. Względem Keccak, NIST zmodyfikował jedynie tzw. padding. Wynik wywołania obu funkcji jest inny.

W kolejnej wersji Solidity aliasy sha3 jak i suicide nie będą dozwolone.

```
keccak256("The quick brown fox jumps over the lazy dog");  
// 0x4d741b6f1eb29cb2a9b9911c82f56fa8d73b04959d3d9d222895df6c0b28aa15
```

```
const { sha3_256, keccak_256 } = require("js-sha3");  
sha3_256("The quick brown fox jumps over the lazy dog");  
// 69070dda01975c8c120c3aada1b282394e7f032fa9cf32f4cb2259a0897dfc04  
keccak_256("The quick brown fox jumps over the lazy dog");  
// 4d741b6f1eb29cb2a9b9911c82f56fa8d73b04959d3d9d222895df6c0b28aa15
```

#4#SOLIDITY#4#

Porada #4

Jaka jest różnica między domyślną widocznością funkcji oraz zmiennej?

W Solidity, funkcje domyślnie są publiczne, czyli są częścią ABI i mogą zostać wywołane również przez inne kontrakty lub JSON-RPC. Zmienne stanu kontraktu są domyślnie wewnętrzne (ang. internal), nie mogą zostać wywołane zewnątrz i są dostępne dla dziedziczących kontraktów.

W kolejnej wersji Solidity każda funkcja powinna mieć specyfikator określający jej widoczność.

```
contract Foo {  
    string name = "BlockchainPro.pl"; // string internal  
    function getName() view returns (string) { // function public  
        return name;  
    }  
}
```


#5#ETHEREUM#5

Porada #5 Jak wiele transakcji mieści blok w Ethereum?

Pytanie jest podchwytliwe. W Ethereum ilość transakcji w bloku nie jest ograniczona poprzez fizyczny rozmiar bloku jak w przypadku Bitcoin (1MB), a poprzez limit gazu jaki transakcje zużyły. Obecnie na głównej sieci limit gazu wynosi ok. 8000000.

```
> web3.eth.getBlock("5807101").gasLimit
8007778
> web3.eth.getBlockTransactionCount("5807101")
109
```

#6#WEB3#FUNCTION

Porada #6

Jak odczytać wartości zwróconą z funkcji podczas transakcji?

Korzystając z Web3 w łatwy sposób możesz odczytać wartości zwracane przez funkcje kontraktu, które nie modyfikują stanu (view i pure). Wartości zwracane przez funkcje, które modyfikują stan mogą być odczytane tylko przez inny kontrakt. Jest to jeden z powodów, dlaczego warto również testować kod smart contractów w samym Solidity.

```
import "truffle/Assert.sol";

bool result = timedVoting.vote(ADDR);
Assert.equal(result, true, "vote doesn't return true");
```

#7 #SMART_CONTRACTS

Porada #7

Jak stwierdzić czy dany smart contract wspiera dane ABI?

Standard ERC-165 określa zasady implementacji widoku `supportsInterface`, który zwraca `true` jeżeli funkcja o przekazanej sygnaturze jest wspierana. Dzięki temu możemy np. stwierdzić czy dany token powinniśmy obsłużyć jako ERC-20 lub ERC-721.

```
interface ERC165 {  
    // Query if a contract implements an interface  
    // return `true` if the contract implements `interfaceID`  
    function supportsInterface(  
        bytes4 interfaceID  
    ) external view returns (bool);  
}
```

#8#MAINNET#SO

Porada #8 Jak wysłać transakcje za darmo na mainnet?

Ganache CLI, implementacja Ethereum używana do developmentu i testowania smart contractów pozwala na fork innego klienta oraz daje dostęp do 10 adresów po 100 ETH każdy. Transakcje wykonane w ten sposób nie pojawią się na mainnet, ale możesz korzystać z aktualnego stanu głównej sieci do wykonania “darmowych” testów penetracyjnych.

```
$ npm install -g ganache-cli
$ ganache-cli -f https://mainnet.infura.io/

# in a different tab

$ geth attach http://127.0.0.1:8545
> crowdsale.sendTransaction({ value: 10**18, from: web3.eth.accounts[0],
to: "0xCrowdsale" })
```

#9#ETHEREUM#9

Porada #9 Jak podróżować w czasie na Ethereum?

Górnicy mogą manipulować czasem bloku, ale chodzi o coś innego. Ganache, implementacja Ethereum używana do developmentu i testowania smart contractów implementuje dodatkowe metody i jedną z nich jest `evm_increaseTime`, która pozwala na przesunięcie czasu o pewną ilość sekund. Jest to szczególnie przydatne do przetestowania kontraktu, którego działanie zależy od czasu bloku.

```
geth attach http://127.0.0.1:8545
> web3.eth.getBlock(web3.eth.blockNumber).timestamp
1530485084
> web3.currentProvider.send({ jsonrpc: "2.0", method: "evm_increaseTime",
params: [10000] })
> web3.eth.sendTransaction({ from: web3.eth.accounts[0] })
> web3.eth.getBlock(web3.eth.blockNumber).timestamp
1530495128
```

#10#STRING2BYTES

Porada #10

Jak zamienić string na bytes i bytes na string korzystając z Web3?

Web3 posiada całą masę funkcji przydatnych podczas tworzenia front-endów zdecentralizowanych aplikacji, jak implementacja biblioteki BigNumber, sha3, randomHex i wiele innych.

```
> web3.utils.fromUtf8("Michał")  
'0x4d69636861c582'  
  
> web3.utils.toUtf8("0x4d69636861c582")  
'Michał'
```

#11 #WEBSOCKET

Porada #11 Jak połączyć się z Infura poprzez WebSockets?

Infura znana jest z tego, że dostarcza publiczne endpointy pozwalające podłączyć się do głównej i testowych sieci Ethereum oraz IPFS. Chociaż na obecną chwilę nie jest to opisane w dokumentacji, istnieje możliwość połączenia się z wykorzystaniem WebSockets by zapewnić wsparcie dla subskrypcji Web3 1.x.

```
const Web3 = require("web3");  
const web3 = new Web3("wss://mainnet.infura.io/ws");  
// or new Web3("wss://mainnet.infura.io/_ws");  
const Emitter = new web3.eth.Contract(abi, "0xf94..688");  
Emitter.events.MyEvent().on("data", console.log.bind(console));
```

#12#SMART_CONTRACTS

Porada #12 Jak wymusić transfer etherów do smart contractu?

Z reguły, jeżeli kontrakt nie implementuje fallback function oznaczonej słowem kluczowym payable to nie można wysłać do niego środków standardową transakcją. Jednym ze sposobów wymuszenia takiego transferu jest przekazanie adresu kontraktu jako parametr dla wywołania funkcji selfdestruct. Po co ktoś miałby to zrobić? Źle zaimplementowany kontrakt, który sprawdza czy balans jest równy sumie zdeponowanych środków, może być podatny na atak DoS.

Inne sposoby wymuszenia transferu to ustawienie adresu jako odbiorcy coinbase transaction albo wysłanie środków na adres zanim kontrakt zostanie stworzony.

```
contract ForceTransfer is Ownable {
    constructor() public payable {}
    function kill(address _addr) public onlyOwner {
        selfdestruct(_addr);
    }
}
```


#13#VALUE#FUNC

Porada #13 Jak zwrócić wiele wartości z funkcji?

Istnieją dwa sposoby na zwrócenie wartości z funkcji. Pierwszy to użycie `return`, a drugi to przypisanie wartości do nazwanych output parameters jak w przypadku funkcji `getBid2`, która w tym przykładzie korzysta z destrukcji do przypisania wartości `addr` i `value`.

```
contract Auction {
    address bidder;
    uint256 bid;

    function getBid() public view returns (address, uint256) {
        return (bidder, bid);
    }

    function getBid2() public view returns (address addr, uint256 value) {
        (addr, value) = getBid();
    }
}
```

#14 #STRING2BYTES

Porada #14 Jak modyfikator może wywołać funkcję wiele razy?

W Solidity, modyfikatory używane są do tego by sprawdzić warunki początkowe czy trochę rzadziej, by sprawdzić warunki końcowe lub zmodyfikować stan. Niecodziennym przykładem użycia jest wywołanie modyfikowanej funkcji więcej niż raz. Dziwne, ale działa.

```
contract Counter {  
    uint256 public count;  
  
    modifier doubleCount() {  
        _; _;  
    }  
  
    function click() public doubleCount { count++; }  
}
```

#15 #SWAP #15 #SV

Porada #15 Jak prosto zamienić wartości dwóch zmiennych?

Korzystając z destructuring assignment możesz łatwo zamienić wartość dwóch zmiennych.

```
contract Swap {  
  uint8 public a = 0;  
  uint8 public b = 1;  
  
  function swap() public returns (uint256) {  
    (a, b) = (b, a);  
  }  
}
```

#16#ERRORS#16#

Porada #16 Jak ustawić niestandardowe wiadomości o błędach?

Istnieją trzy funkcje do wyrzucenia wyjątków: `assert`, `revert` i `require`. Dwa ostatnie, `revert` i `require` przyjmują odpowiednio jako pierwszy i drugi parametr opcjonalną wiadomość o błędzie. Używanie niestandardowych wiadomości o błędach może znacznie ułatwić debuggowanie.

```
contract Workshop {  
    function signUp(address _addr) public payable returns (uint256) {  
        require(_addr != address(0), "you forgot to specify the address");  
        require(msg.value == 0.5 ether, "ticket costs 0.5 ETH");  
        // ...  
    }  
}
```

#17 #FUNCTION_S

Porada #17 Jak obliczyć function selector i przetestować wyjątek?

By przetestować wyjątek w Solidity np. gdy używasz Solidity do testów jednostkowych smart contractów musisz użyć `address.call`, który zwraca `false` w przypadku wyjątku i `true` w przypadku braku wyjątku. Do użycia `call` musimy jednak policzyć selektor funkcji, czyli pierwsze 4 bajty hasha Keccak256 z sygnatury funkcji.

```
bytes4 signature = bytes4(keccak256("transferOwnership(address)"));
bool result = address(ownable).call(signature, address(this));
Assert.equal(result, false, "non-owner can call transfer ownership");
```

#18#STORAGE#18

Porada #18 Jak wykorzystać eventy jako tani storage?

Największą zaletą eventów jest ich bardzo niski koszt (8 gas/byte) w porównaniu do zapisania informacji w stanie kontraktu (20 000 gas). Eventy nie mogą być odczytane przez inne smart kontrakty ale mogą dostarczyć kluczowych informacji dla zdecentralizowanej aplikacji.

```
// event Donated(address indexed donator, uint256 value, string msg);  
  
const Donations = new web3.eth.Contract(abi, address);  
const [donator] = await web3.eth.getAccounts();  
const myDonations = await Donations.getPastEvents("Donated", { donator });
```

#19 #FALLBACK_F

Porada #19 Jak wykorzystać gas dostępny w fallback function?

Fallback function to metoda kontraktu bez nazwy, która wywoływana jest m. in. w przypadku standardowego transferu etherów. W przypadku transferu gas jest ograniczony do 2300 co nie wystarczy na zmianę stanu lub kolejny transfer. Wystarczy jednak na wyemitowanie eventu!

W kolejnej wersji Solidity fallback function powinna być oznaczona jako external.

```
event Donated(address indexed sender, uint value);

function () public payable {
    emit Donated(msg.sender, msg.value); // only 1166 gas
}
```

#20#EVENT#LOG

Porada #20 Jak można logować eventy?

Eventy są wysokopoziomową abstrakcją zaimplementowaną w oparciu o logi. Oznacza to, że możesz wykorzystać niskopoziomową funkcję `log1` do uzyskania tego samego efektu co emitowanie eventu. Funkcje `emit` i `emit2` mają ten sam efekt.

```
contract MyEvents {
    event MyEvent(address value);

    function emit() public {
        emit MyEvent(msg.sender);
    }

    function emit2() public {
        log1(bytes32(msg.sender), keccak256("MyEvent(address)"));
    }
}
```


#21 #DEBUG #TEST

Porada #21

Jak debugować testy z wykorzystaniem eventów?

Eventy sprawdzą się świetnie również, gdy starasz się debugować kod np. podczas testów. W przypadku błędu Truffle wypisze w konsoli wszystkie eventy, które zostały wyemitowane, podczas gdy test był uruchomiony.

```
event Debug(uint256 value);
event Debug(string value);

function test() public {
    emit Debug(10);
    emit Debug("Hello, World");
    Assert.equal(a, b, "a doesn't equal b");
}
```

#22#SOLIDITY#22

Porada #22

Jak użyć parametrów domyślnych?

W Solidity, funkcja identyfikowana jest po swojej sygnaturze, która wyliczana jest również z typów przyjmowanych parametrów. Dzięki temu możesz przeciążyć funkcję i np. zaimplementować domyślne wartości.

```
contract Donations {  
    event Donated(uint256 value, address donator, string name);  
  
    function donate(string _name) public payable {  
        emit Donated(msg.value, msg.sender, _name);  
    }  
  
    function donate() public payable {  
        donate("Anonymous");  
    }  
}
```

#23 #STRING2STR

Porada #23 Jak połączyć dwa stringi?

Solidity nie posiada wbudowanych funkcji, które umożliwiałyby łatwe wykonywanie operacji na ciągach znaków. Jedną z bardziej udanych bibliotek jest solidity-stringutils.

```
import "./strings.sol";

contract HelloWorld {
    using strings for *;

    function greetings(string _name) public pure returns (string) {
        return "Hello, ".toSlice().concat(_name.toSlice())
    }
}
```

#24#STRUCT#24#

Porada #24 Jak zwrócić strukturę (ang. *struct*)?

Funkcje mogą zwracać typy definiowane przez użytkownika tylko gdy są dostępne wewnętrznie (ang. *internal*). Jeżeli chcesz zwrócić strukturę z funkcji dostępnej zewnętrznie możesz rozbić ją na kilka parametrów wyjściowych.

```
struct Vote {  
    address voter;  
    uint8 votedFor;  
}  
  
Vote[] internal votes;  
  
function getVote(uint256 _n) internal view returns (Vote) {  
    return votes[_n];  
}  
  
function getVote2(uint256 _n) public view returns (address, uint8) {  
    Vote memory vote = getVote(_n);  
    return (vote.voter, vote.votedFor);  
}
```

#25#STRUCT#25#

Porada #25

Jak zwrócić strukturę (ang. *struct*) z wykorzystaniem ABIEncoderV2?

ABIEncoderV2 jest **eksperymentalną** funkcjonalnością Solidity, która wprowadza wsparcie dla przesyłania dynamicznych i zagnieżdżonych wartości między smart contractami i zdecentralizowanymi aplikacjami. Ethers.js, świetna alternatywa dla Web3.js, zaimplementowała już wsparcie dla ABI v2.

```
pragma experimental ABIEncoderV2;

struct Vote {
    address voter;
    uint256 votedFor;
}

function getVote(uint256 _n) public view returns (Vote) {
    return votes[_n];
}
```

#26#SOLIDITY#26

Porada #26 Jak pokolorować składnię Solidity na GitHub?

Na obecną chwilę, GitHub nie koloruje kodu w plikach *.sol. Jeżeli chcesz poprawić czytelność pull requestów w twoim repozytorium dodaj do repozytorium plik .gitattributes o następującej treści.

```
*.sol linguist-language=Solidity
```

#27#SOLIDITY#27

Porada #27 Jak zweryfikować składnię kodu Solidity?

Solium jest narzędziem do statycznej analizy kodu smart contractów, który poza formatowaniem kodu potrafi też wskazać potencjalne błędy bezpieczeństwa. Alternatywą dla Solium jest Solhint.

```
# Run solium on the current directory
solium -d .
# Run solium on Token.sol file
solium -f ./Token.sol
# Disable entire next line
// solium-disable-next-line
# Disable specific rule on the current line
// solium-disable-line security/no-low-level-calls
```

#28#SOLIDITY#28

Porada #28

Jak przeanalizować pokrycie testami kodu Solidity?

solidity-coverage to narzędzie do analizy pokrycia kodu testami, które bez potrzeby konfiguracji działa z Truffle, popularnym frameworkiem do projektów w Solidity. Opcjonalna konfiguracja pozwala na zmianę ustawień testrpc czy pominięcie wybranych plików.

```
./node_modules/.bin/solidity-coverage
```

Alternatywą dla solidity-coverage jest sol-cov stworzony przez developerów z projektu Ox.

#29#CONTRACTS

Porada #29

Jak twoje funkcje mogą przysłonić te wbudowane?

Upewnij się, że znasz kod kontraktów, po których dziedziczysz. Wywołanie `selfdestruct` niszczy kontrakt i wysyła jego fundusze na wskazany adres. W tym przykładzie `selfdestruct` został nadpisany przez rozszerzenie kontraktu `Straitjacket` i jego wywołanie nie będzie miało żadnego efektu. Kompilator generuje ostrzeżenie w przypadku przysłonięcia wbudowanej funkcji.

```
contract Straitjacket {
    function selfdestruct(address _to) internal {}
}

contract MyContract is Straitjacket {
    function destroy() public {
        selfdestruct(msg.sender);
    }
}
```

#30#HOISTING#3

Porada #30 Jak działa podnoszenie zmiennych (ang. hoisting)?

Hoisting czyli podnoszenie definicji zmiennej na początek funkcji znany z zachowania `var` w JavaScript występuje również w Solidity.

Uwaga! To zachowanie spowoduje `DeclarationError` w kolejnej wersji Solidity.

```
contract Hoisting {  
    function double(uint256 _n) public pure returns (uint256) {  
        result = 2 * _n;  
        uint256 result;  
        return result;  
    }  
}
```

A large, light gray, stylized letter 'S' watermark is centered on the page, spanning most of its width and height. It has a modern, rounded, and slightly irregular design.

Materiały dodatkowe

#MATERIAŁY_DOI

CryptoZombies

Interaktywny kurs Solidity i Web3 dla początkujących.

Ethernaut

Tekstowa gra przygodowa dla bardziej zaawansowanych, w której by przejść na kolejny poziom musisz rozwiązać określony problem z wykorzystaniem Solidity.

Solidity Security

Pokażna lista znanych wektorów ataków i popularnych antywzorców.

Ethereum Smart Contract Security Best Practices

Obszerny poradnik omawiający różne problem dotyczące bezpieczeństwa smart contractów oraz znane ataki.

Test-driven development with Solidity

Moje wprowadzenie do Solidity poprzez TDD.

Making Sense of Ethereum's Layer 2 Scaling Solutions

Wysokopoziomowe wprowadzenie do State Channels, Plasma i Truebit.



Podsumowanie

#PODSUMOWANIE

Jak już pewnie zdążyłeś lub zdążyłaś się zorientować, pisanie w Solidity nie jest pozbawione wyzwań i fakt, że bez sztuczek w postaci proxy pattern kontrakty są niemodyfikowalne nie ułatwia zadania programiście. Bez względu na język programowania, pisanie smart contractów jest ekscytującą ścieżką kariery w branży, która jeszcze tak naprawdę dopiero raczkuje.

Lista porad i ciekawostek cały czas rośnie i nowe wpisy pojawiają się regularnie w języku polskim na profilu BlockchainPro na Facebooku. Jeżeli masz dla mnie informację zwrotną na temat materiałów zawartych w ebooku napisz maila na michal@blockchainpro.pl

Jeżeli jesteś w pozycji osoby, która uczestniczy w procesie rekrutacji to ciekawym pomysłem może okazać się użycie przedstawionych tu informacji do poszukiwania nowych współpracowników.

Koniecznie daj mi znać, jak poszło!

Michał Załęcki



**PODZIEL
SIĘ EBOOKIEM
ZE ZNAJOMYMI!**

f in t





**OBSERWUJ NAS
W SOCIAL MEDIA!**

f @ M in t



BlockchainPro



TOOPL00X