

฿assie€oin



฿assie€oin

{{Cryptocurrencies uitgelegd}}

Gemaakt door Bastiaan van der Plaat uit H5A



Inhoudsopgave

Voorwoord	5
Deelvragen	5
Hoofdstuk 1	6
Hoofdstuk 1½	8
Het opzetten van de programmeer omgeving	8
Verschillende waarden	9
Numbers and Math	9
Strings	10
Booleans	12
Typeof	14
Echte code schrijven	15
Het maken van een programmeer bestand	15
Comments	16
Variabelen	17
Arrays of te wel Lijsten	19
Objecten of te wel een soort variabele met variabelen erin	20
Als dit dan dat if statements	21
For loops dingen vaker doen als een keer	22
Functies of te wel uw code recyclen en zo mooi ordenen	23
Hoofdstuk 2	25
Hashing	25
De blockchain	26
Een proof-of-work systeem	29
Het peer-to-peer netwerk	32
Nog de complete code	35
Hoofdstuk 3	38
Transacties	38
De webserver met transacties	41
Transacties beveiligen met een sleutel	42
De versleutelde webserver	47
De complete code van onze cryptocurrency	47
Hoofdstuk 4	52

BasissieCoin

Geschiedenis van de Bitcoin	52
Gevaren van het investeren in de cryptovaluta.....	53
Hoofdstuk 5	54
Bitcoin (BTC).....	55
Ethereum (ETH).....	55
Bitcoin Cash (BCH).....	56
Ripple (XRP).....	56
Litecoin (LTC).....	56
Tether (USDT).....	57
Monero (XMR)	57
Hoofdstuk 6	59
Bij de overheid	59
In de zorg.....	59
In de criminaliteit.....	60
Hoofdstuk 7	61
De conclusie	61
De toekomst.....	61
Bronnen.....	63
Sites die ik heb gebruikt.....	63
Boeken die ik heb gelezen	63
Logboek.....	63

Zouden cryptovaluta's in de toekomst normaal geld kunnen vervangen als ons primaire betaalmiddel, hoe werkt het en hoe maak u uw eigen cryptovaluta?

Hoofdvraag

Voorwoord

Toen ik aan dit Profiel Werk stuk begon had ik nog niet zoveel vragen over cryptovaluta's maar hoe meer ik er over te weten kwam, hoe meer ik er over wilde gaan leren en jullie ook wilde gaan vertellen. Ik besloot later zelf mijn eigen demo cryptocurrency te maken en als ik dat deed moest ik jullie ook de basis van het programmeren leren zo groei dit PWS door en door en uit eindelijk heb ik dit gemaakt. Hieronder staan mijn deelvragen in logische volgorde zoals ze ook in de hoofdstukken staan. Veel plezier met lezen en misschien maakt u zelf in de toekomst ook wel uw eigen cryptocurrency.

Deelvragen

- Hoe werkt programmeren?
- Wat is de blockchain?
- Wat zijn cryptovaluta?
- Hoe maak u uw een cryptocurrency?
- Hoe is de cryptocurrency ontstaan?
- Welke munten zijn tegenwoordig in omloop?
- Wat zijn andere toepassingen van de blockchain?

Hoofdstuk 1

> **Introductie | Wat is geld het toch waard?** <

U merkt het niet elke dag want het is een stille ontwikkeling. Het geld, uw geld wat is het toch waard? U zou denken het is de waarde die op het biljet staat of de waarde die op mijn bankrekening staat. Maar nee, dat is het niet, want het is niks waard, niks. Geld heeft waarde door het collectieve vertrouwen van de maatschappij in de valuta. Die door de overheid wordt uitgegeven. Dus eigenlijk vertrouwen in de overheid. Maar dat vertrouwen neemt steeds meer af, de politiek wordt extremer en mensen wennen aan mannen die racisme als de oplossing zien. Zo ouderwets. Het vertrouwen in de valuta's van vandaag is sinds de 2007 verandert. Mensen vertrouwen de valuta niet meer, mensen vertrouwen de banken niet meer. De banken wat hebben die met valuta te maken zou u denken? Alles want banken zijn de instellingen die de valuta uitgeven zij bepalen de prijs, de rente.

Dat zit zo, u zou denken dat geldwaarde heeft en dat was ook zo vroeger in de klassieke oudheid was een munt evenveel waard als het metaal waarvan de munt gemaakt is. Dit was zeer lang gebruikelijk tot dat het papieren geld in omloop kwam in het oude China. De biljetten hadden waarde omdat de keizer dat bevolen had. De keizer was de machtigste man in het hele rijk en hoewel niemand hem persoonlijk kende vreesde iedereen hem, zo had het biljet waarde. Later gingen westerse landen ook zo'n systeem gebruiken, alleen wij in het westen hadden geen grote leiders. Wij waren jonge verlichte democratieën, dus wij dachten het geld met goud en andere waarde vollen dingen te kunnen dekken. Een lange tijd was ieder biljet zo veel waard als er een beetje goud was. Zo was in direct op een rare manier geld gedekt. Maar deze methode was al snel niet houdbaar en het werd niet zo lang geleden afgeschaft. Nu konden overheden zoveel geld drukken als ze zelf wilde dus schoot de inflatie ook om hoog.

Tegenwoordig is de situatie anders de meeste mensen en bedrijven krijgen het geld van banken door geld te lenen. De banken vraagt hier over rente, dus het geld is het geld waard plus de rente. De rente wordt terugbetaald aan de banken en die leent het weer verder uit dus het geld wordt steeds minder waard.

Ook is veel geld tegenwoordig gewoon virtueel en staat het op uw bank app dit is helemaal raar. Want de bank heeft nu alle macht over jou en over de economie. Als we zo door gaan met het vitaliseren van het geld eindigt de wereld in een hele naar plek waar een groepje mensen alle macht hebben over de economie en over de levens van bijna alle mensen op deze aarde, later daar meer over.

Mensen hebben geen vertrouwen meer in de economie dus snakken ze naar ander betaalmiddelen. Maar ze willen wel het gemak van online virtualisatie houden. Daarop kan cryptovaluta's een uitkomst bieden. In dit werkstuk zal ik u precies uitleggen hoe u kan programmeren. Hoe u een blockchain kan schrijven. Hoe u op die blockchain een cryptovaluta kan bouwen. En nog veel meer informatie over alle andere cryptovaluta die nu rondslingeren. Verder zal ik mijn blik werpen op de toekomst want die is roze kleurig. Als we de kant van cryptovaluta's opgaan.

Hoofdstuk 1³/₄

> JavaScript een simpele taal apart <

Het opzetten van de programmeer omgeving

Voordat u mijn profiel werkstuk verder gaat lezen moet u eerst de basis van het programmeren weten. In dit gehele werkstuk gebruik ik de programmeer taal JavaScript. Dit is een simpel taaltje wat u zo onder de knie hebt.

Voor deze korte JavaScript tutorial ga ik er van uit dat u Windows 10 gebruikt. Gebruikt u Linux dan kan u waarschijnlijk dit hoofdstuk overslaan.

Eerst hebben we een programma nodig wat onze JavaScript code kan draaien. Dit programma wordt een *runtime* of *interpreteer* genoemd. Wij gaan de Node.js runtime gebruiken, dit is een gratis en open source JavaScript runtime.

Ga eerst naar nodejs.org dit is de website waar u de Node.js runtime kan downloaden. Klik vervolgens op de *current* download knop. Als het goed is download dit een msi (Microsoft Installer) bestand. Start het bestand en installeer de runtime, herstart daarna uw computer.

Klik dan op het zoek of Cortana icoontje naast het Windows startknopje op uw beeldscherm. Vervolgens typt u de letters *cmd* en drukt u op enter. Als het goed is verscheidt er dan een zwart venster genaamd de Opdrachtenprompt.

Type nu het commando *node* in, dit commando start de Node.js repl op.

Repl staat voor *Read-eval-print loop*, vertaald naar het Nederlands is dat: vraag om een regel code (read), voer die dan uit (eval), print de uitkomst (print) en begin overnieuw (loop). U bent nu klaar met het opzetten van uw programmeer omgeving. En u kan nu gaan beginnen met het typen van uw eerste regels code.

Verschillende waardes

Numbers and Math

In JavaScript zijn er verschillende soorten values of waardes we beginnen met nummers want programmeren gaat meestal daarom.

Steeds typt u een regel code en drukt u op enter dan krijgt u iets terug.

Om nummers te gebruiken typt u die gewoon in:

```
> 10
```

```
10
```

U kan ook komma getallen gebruiken alleen schrijf u die wel met de Engelse notatie met een punt:

```
> 1.5
```

```
1.5
```

Let op dat u bij grote getallen geen punten gaat plaatsen om de drie getallen:

```
> 1.000.000
```

SyntaxError

U krijgt dan een syntax error dit betekend dat de interpreter uw code niet begrijpt en dus ook niet kan uitvoeren

Ook kan u in getallen de wiskundige notatie gebruiken via de e:

```
> 4.2e3
```

```
4200
```

Zoals u kan zien worden getallen in JavaScript gewoon volledig uitgeschreven.

Om sommen te maken gebruikt u de notatie number + number net zoals bij wiskunde, dit wordt ook wel een operatie genoemd en de rekenregels in JavaScript zijn hetzelfde als bij wiskunde:

```
> 3 + 4
```

```
7
```

U kunt de spaties ook weg laten maar dan wordt het wel onoverzichtelijk:

```
> 3+4
```

```
7
```

Min sommen gaan via dezelfde weg:

```
> 4 - 6
```

```
-2
```

Keer gaat via het sterretje * teken:

```
> -4 * 3
```

```
-12
```

Delen gaat via het slash / teken:

```
> 12.4 / 3
```

```
4.1333333333333334
```

De rest van delen of modulo krijgt u via het procent % teken:

```
> 10 % 3
```

```
1
```

U kan ook de haakjes gebruiken net zoals bij wiskunde:

```
> (4 + 4) * 5.3
```

```
42.4
```

Opdracht: Vul zelf nog wat sommen in en kom erachter dat JavaScript eigenlijk een soort rekenmachine is.

Strings

Een ander waarde is een stukje tekst ook wel een string genoemd.

U maakt een string door tekst tussen de haakjes (links van de enter) te plaatsen:

```
> 'Hallo!'
```

```
'Hallo!'
```

U kan geen enters of andere haakjes in de string zetten want dan denk de interpreter dat de string abrupt is geëindigd:

```
> 'Hallo!
```

SyntaxError

U moet die karakters *escapen* dat betekent dat u een backslash \ ervoor zet om zo de interpreter te helpen. Een enter of nieuwe regel wordt dan bijvoorbeeld \n en een ander haakje wordt bijvoorbeeld \' en zo voort:

```
> 'Hij zei:\n\'He!\''
```

```
'Hij zei:\n\'He!\''
```

U kan geen wiskundige operaties uitvoeren op strings maar je kan wel twee strings aan elkaar plakken met de plus:

```
> 'Er was eens een ' + 'mens'
```

```
'Er was eens een mens'
```

U kan ook een nummer aan een string vast plakken let wel op dat als u ook nog een wiskundige operatie wilt doen dat u die dan wel in de haakjes doet:

```
> 'Er waren eens ' + (1 + 2 + 3) + ' biggen...'
```

```
'Er waren eens 6 biggen...'
```

U kan ook nog de lengte van een string krijgen via de .length die return nummer zodat u er ook nog verder mee kan rekenen:

```
> 'Er was eens een mens'.length + 5
```

```
25
```

Opdracht: Maak een paar stringen die een som uit reken bijvoorbeeld:

```
> '4 * 7 = ' + (4 * 7)
```

```
4 * 7 = 28
```

Booleans

Naast numbers en strings zijn er ook nog booleans dit is een simpele value want een boolean kan maar twee waardes hebben namelijk: *true* of *false*.

Dus als u zegt true of false heeft u al een boolean gemaakt:

```
> true
```

```
true
```

```
> false
```

```
false
```

U kan een boolean ook omdraaien met het uitroepteken:

```
> !true
```

```
false
```

```
> !false
```

```
true
```

Maar een boolean kan u meer als u bijvoorbeeld wilt weten of twee dingen gelijk zijn kan u een vergelijking doen en die geef dan true of false terug.

Deze operator vergelijkt of 10 gelijk is aan 10, of 10 gelijk is aan 9, of man gelijk is aan mens en of true gelijk is aan true:

```
> 10 == 10
```

```
true
```

```
> 10 == 9
```

```
false
```

```
> 'man' == 'mens'
```

```
false
```

```
> true == true
```

```
true
```

U kan ook kijken of iets niet gelijk is dan gebruik u dit:

```
> 10 != 10
```

```
false
```

```
> 'man' != 'mens'
```

```
true
```

U kan ook kijken of iets groter is als iets anders:

```
> 10 > 9
```

```
true
```

```
> 10 > 10
```

```
false
```

U kan ook kijken of iets groter is of gelijk als iets anders:

```
> 10 >= 10
```

```
true
```

U kan ook kijken of iets kleiner is als iets anders:

```
> 8 < 9
```

```
true
```

```
> 8 < 8
```

```
false
```

U kan ook kijken of iets kleiner is of gelijk als iets anders:

```
> 8 <= 8
```

```
true
```

U kan ook booleans koppelen met de *en* operator als er een false is dan is het false als alles true is dan geeft het true:

```
> true && true && true
```

```
true
```

```
> false && true && true
```

false

U kan ook booleans koppelen met de *of* operator als er een true is dan is het true:

```
> false || false || false
```

false

```
> true || false || false
```

true

Maar u weet dat de operators hier boven ook een boolean afgeven dus u kan ook schrijven:

```
> 4 == 3 || 4 == 2
```

false

```
> 4 != 4 && 3 != 3
```

true

Opdracht: Dat was een hele kluif, puzzel lekker ver met logica en maak nog wat vergelijkingen.

Typeof

Als kers op de taart heeft u ook nog de *typeof* operator deze operator is erg simpel want het geeft een string terug met de waarde van het item wat u er in stopt:

```
> typeof 3
```

'number'

```
> typeof 'Hoi'
```

'string'

```
> typeof (4 == 4)
```

'boolean'

Echte code schrijven

Het maken van een programmeer bestand

We hebben net de hele tijd code geschreven in de repl, maar we gaan voortaan alle code in een bestand schrijven. Dit heeft als voordeel dat u meerdere lijnen kan schrijven en dat u zo een compleet programma kan maken. Omdat te doen moet u via de file manager van Windows een nieuw tekstbestand aanmaken en de extensie (het gedeelte achter de laatste punt) veranderen naar `.js`, dit houdt in dat het bestand JavaScript code bevat.

Ik raad ook sterk aan dat u een betere editor voor Windows download, want de standaard editor van Windows genaamd Notepad is erg slecht en geeft geen code kleuring en autocompletion. Ik raad sterk aan dat u Notepad++ of Programmer's Notepad te downloaden, u kunt die downloaden op: <https://notepad-plus-plus.org/download/> en <http://www.pnotepad.org/download/>. Het maak mij niet uit welke editor u gebruikt ze zijn namelijk erg hetzelfde maar op bepaalde diepgaande punten net iets anders. Download het installatie bestand en draai het installatie programma. Start daarna uw net geïnstalleerde editor op en sleep uw `.js` bestand in de editor als het goed is ziet u dan een cursor en een veld waar u codes kan gaan typen.

Dan opent u het JavaScript bestand met u editor en kan u codes gaan in typen. Het eerste wat u in uw bestand moet schrijven is de lijn die hier onderstaat. Zoals u kan zien zal ik voortaan alle code in deze plaatjes laten zien zodat ze de kleur stijlen blijven houden en het ziet er ook nog mooi uit:

```
console.log('Hello World!');
```

Als u het bestand heeft gesaved moet u nu de terminal naar de map verzetten waar u het bestand heeft neergezet dit doet u zo. Klik in de Windows File Explorer op de folders namen balk die bovenin staat als het goed is komt er dan een path string te staan die begint met uw schijf letter waarschijnlijk `C:\` en dan het pat of the path zoals programmeurs het noemen naar uw folder. De path is een lijst van alle folders waar in het bestand zit tot het bestand. Eigenlijk best wel logisch. Nu

opent u een nieuwe Command Prompt en type **cd** en kopieert u hier u path dan krijgt u iets zoals **cd C:\Users\bplaat\Documents\Projects\bassiecoin**. Als u dat hebt gedaan is als het goed is uw cmd naar de goed path gegaan en dat ziet u ook aan de linker kant van de prompt. Nu wilt u het bestand uitvoeren en dat doet u door **node** {naam van het bestand met .js} in te typen. Mijn bestand heet bijvoorbeeld script.js dus ik draai het bestand met **node script.js**. Als het goed is moet er nu *Hello World!* op het scherm komen te staan. Gefeliciteerd u hebt uw eerste scriptje gemaakt.

We gaan nu het script wat aanpassen zodat het twee regels code kan uit printen. Zoals u kan zien is console.log een functie, dit is een stukje code wat u uitvoert met argumenten dit leg ik later wel uit. Maar wat u wel begrijpt is dat het een string als argument gebruikt en u weet wat dat voor type nu is als het goed is.

```
console.log(1 + 2 * 3);  
console.log('Lalalala' + 438);
```

Voer nu deze code in. Zoals u kan zien kan console.log ook numbers uitprinten en kan het ook string plakken aan. Nu begrijpt u ook wat de repl deed want dat voerde altijd de code uit met de console.log! Eigenlijk best wel simpel.

Opdracht: Probeer nog wat dingen met de console.log functie.

Comments

Comments zijn hele makkelijk maar superbelangrijke dingen om te leren een comment of in het Nederlands bijschrift is een regel of meerdere regels code om uw code uitleggen of te verduidelijken dit is erg belangrijk want zo zorgt u er voor dat andere mensen ook begrijpen wat u aan het doen bent als ze jouw code lezen. Een single line comment of enkele regel comment maakt u door twee slashjes neer te zetten alles daarachter is een comment tot aan het einde van de regel. Een multi lijn comment of meerder regels comment maakt u door een slash en een sterretje en het andersom om hem weer af tesluiten. Zie hier onder.


```
// Een enkele regel comment

/*
Een multi regel comment
*/
```

Zoals u kan zien is dit niet zo erg ingewikkeld maar het is wel erg belangrijk want door uw code te commenten is het duidelijker en veel gemakkelijker te begrijpen ook als u bijvoorbeeld een oud project van een paar jaren geleden weer oppakt. U heeft daar waarschijnlijk niet zoveel last van maar ik merk dat bij me zelf nu wel. Want ik wilde bijvoorbeeld een wat verouderde website weer levend maken maar vroeger plaatste ik bijna nooit comments dus ik moest de code weer helemaal opnieuw gaan lezen en begrijpen wat ik toen had verzonnen.

Variabelen

Variabelen misschien kent u zal al van bijvoorbeeld Wiskunde. In JavaScript zijn variabelen een soort geheugenplaatsen. Bekijk het voorbeeld eens hieronder en probeer uit te vinden wat `console.log` uit gaat printen. Oja, zoals u kan zien kan `console.log` ook meerdere argumenten aan met een komma dit zal ik later nog wel uitleggen maar het komt neer dat `console.log` er een spatie tussen zal zetten en het een andere kleur zal geven als het een number zou zijn.

```
var a = 4;
var b = a * 5;
a = 3
console.log(a, b);
```

Er komt 3 en 20 uit. Zoals u misschien wel had geraden. Maar variabelen kunnen van alles en nog wat zijn. Ze kunnen ook strings zijn of booleans. Als u een variabele voor het eerst wilt beschrijven moet u altijd het `var` statement gebruiken. Dit zegt de v8 engine in nodejs dat u dit woord wilt gebruiken voor een variabele.

Zonder dit statement `var` werkt de code ook wel maar dit word sterk afgeraden want dan moet javascript gaan raden wat u wilt gaan doen en dit werkt meestal wel maar kan soms rare gevolgen hebben. Variabelen kunnen ook strings zijn en andere woorden bevatten maar een variabele naam moet altijd wel beginnen met een letter en of een underscore (`_`).

```
var name = 'Bastiaan';
name = name = ' van der Plaat';
console.log(name);

var 3a = 'mag niet!';
```

Zoals u kan zien werken de bovenste drie regels prima. Maar de laatste regel niet goed, u kan ook zien dat de kleuring een beetje vercracked is en dat komt omdat mijn photocode programma het ook niet begrijpt OMDAT HET FOUT IS. U kan nooit een variabele beginnen met een iets anders als een letter of een underscore.

Er is ook nog een short cut voor bepaalde dingen die u vaak doet. Zonder deze dingen zou u erg veel code bloat moeten schrijven. Veel mensen zeggen al dat u bij een taal zoals JavaScript erg veel moet schrijven om de computer te vertellen wat u wilt maar ik vind dat wel fijn omdat het dan alles wat duidelijker maakt en het geen turbo taaltje is met veel dingen om te leren maar ik dwaal af.\

U kan `+=` gebruiken om een variabele met zichzelf en iets anders op te tellen. Dit doet u erg vaak. Dit werkt trouwens voor elke operator dus ook voor min en voor keer enz. Zie het code voorbeeldje voor meer uitleg hier onder.

```
var nummertje = 1;
nummertje = nummertje + 4;
nummertje += 4;
nummertje = nummertje * 8;
nummertje *= 8;
```

Ook is er nog de increment en de decrement deze zijn:

```
var nummertje = 1;
nummertje++;
nummertje--;
```

Deze verhogen een variabele met een of verlagen het met een dit gebruikt u vaak in een loop. Ja ik weet dat deze uitleg erg kort is maar als u meer van JavaScript variabelen is dan moet u naar deze site gaan: <https://eloquentjavascript.net/>. Ik moet het wel zo beknopt houden anders gaat mijn PWS alleen maar hierover daarom zal ik over de volgende dingen wat sneller heen gaan waarschijnlijk veel te snel ga dus zo ie zo naar die site en lees hem minstens een keertje door. Hij is trouwens wel in het Engels, maar alle programmeer dingen op het internet zijn in het Engels. Behalve dit natuurlijk 😊.

Arrays of te wel Lijsten

In het programmeren werkt u vaak met lijsten. In lijsten slaat u informatie dus andere typen op in een ongeordende manier. U kan alles in een array stoppen zelfs andere arrays. U maak een array redelijk simpel met de stompe haakjes []:

```
var cool = [ 'BassieBas', 'rapt', 'geweldig!' ];
console.log(cool, cool[0]);
```

Deze code maak een simpele variabele aan met daar in een array met strings. In de volgende lijst log ik de complete array u kan zien dat node js dat mooi print en ik print ook alleen de string *BassieBAS*. Dit doe ik door alleen het eerste item van de array te selecteerden dit doet u door aan de variabele de haakjes toe te voegen en daar in de positie of index door te geven. De positie is altijd nul gebaseerd dat is vrij standaard in computerland. Dit betekent dat 0 het eerste item is en 1 het tweede enzovoort. Dit klinkt onlogisch maar voor mij is het omgekeerd als iemand juist bij 1 begint met tellen vind ik dat vreemd omdat ik nul gebaseerd zo bekend vindt. U kan ook nog het aantal items van een array vinden door de `.length` item uit te lezen en u kan items toevoegen met de `push` functie, zie hieronder:

```
var cool = [ 'BassieBas', 'rapt', 'geweldig!' ];
cool.push('En', 'nog', 'wat!');
console.log(cool, cool.length);
```

Zoals u kan zien zijn arrays erg handig want u kan hiermee erg gemakkelijk informatie op gooien via push en ook de lengte krijgen via length. Bij het kopje loops zal ik jullie ook nog leren om door een lijst heen te lopen via *for* en dan wordt het nog veel handiger, maar dat komt later...

Objecten of te wel een soort variabel met variabelen erin

In het programmeren heeft u soms ook situatie dat u variabelen wilt groeperen. Om zo bijvoorbeeld uw code wat duidelijker te maken of omdat u verschillende variabelen in een array wilt stoppen. Dat kan met een object, een object is eigenlijk variabelen in een variabel. Het werkt erg simpel u maakt een nieuw object door de accolades te gebruiken {}:

```
var person = { name: 'Bastiaan', age: 16 };
console.log(person.name + ' is ' + person.age + ' years old!');
```

Zoals u kan zien moet u de variabelen instellen met de dubbele punt niet met een is teken. Dit is erg belangrijk anders krijgt u een SyntaxError. Ook moet u een komma plaatsen tussen de verschillende items. Als u dan een item wilt krijgen of wilt aanpassen moet u de variabel plus een punt en dan de item naam. Maar soms wilt u een dynamisch naam gebruiken als selector in een object dat kan u dan ook een object item krijgen als string of als nummer dat doet u zo:

```
var person = { name: 'Bastiaan', age: 16 };
console.log('Hello ' + person['name']);
```

Zoals u kan zien gebruikt u zo dezelfde syntax als bij arrays. Dit is ook zo gemaakt omdat een array eigenlijk een uitbreiding is over een object. Wat ik hiermee bedoel te zeggen is dat een array eigenlijk een soort object is. Maar u moet bij een array niet items gaan veranderen want dan kunnen rare dingen gaan gebeuren. Want dan gaat u het prototype van de Array variabel aanpassen. Als dit iets te ingewikkeld klinkt dat is het ook wel en ik ga dat niet in dit korte intro hoofdstukje uitleggen als u meer wilt weten raad ik nog een keer een hele goede site aan genaamd: <https://eloquentjavascript.net/>. Daar wordt alles uitgebreider en ook beter uitgelegd. Nu heb ik alle data types en variabelen gehad en ga ik verder met control flow en functies.

Als dit dan dat if statements

In het programmeren wilt u vaak checken of een bepaalde variabele een bepaalde waarde heeft en of het aan andere bepaalde checks voldoet. Als dat dan zo is dan moet u een bepaalde code blok uitvoeren. Dit doen we in JavaScript met het if statement. In een if statement stopt u altijd een boolean meestal met een soort logische vergelijking en als het true is dan voert hij de code uit en als het false is dan voert hij de code niet uit.

```
var nummertje = 3;
if (nummertje > 2) {
  console.log('Nummertje is groter als 2');
}
```

Zoals u kan zien vergelijkt deze code de variabele nummertje en als het groter is dan 2 dan print hij de tekst. U ziet dat code blokken ook worden gemaakt door de accolades. Net zoals bij het maken van objecten. U moet deze dingen wel uit elkaar houden want het zijn hele andere dingen. U hebt ook nog *else if* en *else* met deze dingen kan u uw if statement langer maken zodat u op meer dingen kan checken en als het niet goed is iets anders doen (*else*, alles is logisch Engels).

```

var nummertje = 3;
if (nummertje > 2) {
  console.log('Nummertje is groter als 2');
} else if (nummertje < 2) {
  console.log('Nummertje is kleiner als 1');
} else {
  console.log('Nummertje is waarschijnlijk 2');
}

```

Deze code is wat langer maar hij checkt nummertje en op andere dingen. U kan natuurlijk ook op meerdere dingen checken via de && zie types.

Opdracht: Maak nog wat if statements die ook wat ingewikkelder zijn 😊.

For loops dingen vaker doen als een keer

U hebt zeker wel een keer gehoord dat computers heel erg goed zijn in dingen vaker doen als een keer en dat is waar. U kan een loop maken dat is een stuk code die de computer vaker dan een keer uitvoert. Een simpele loop maakt u door de *for* loop te gebruiken, een *for* loop bestaat uit drie gedeelte een initialisatie gedeelte in dit gedeelte maakt u de variabelen uit die u gebruikt als teller. Het controle gedeelte als dit false wordt dan stop de loop met draaien. En het increment gedeelte dit maak de variabele hoger zodat u een bepaald keer kan draaien. Hieronder heb ik een simpele loop gemaakt die item 0 tot 9 zegt:

```

for (var i = 0; i < 10; i++) {
  console.log('Item ' + i);
}

```

Kijk eigenlijk is een loop maken best wel simpel. U kan net zoals bij if statements het allemaal uitbreiden u kan ook loops in loops plaatsen maakt niet uit. U moet er wel goed voorzorgen dat de variabelen en de constanten in de loop wel goed zijn anders krijgt u een oneindige loop. Dit is een vaak voorkomend probleem bij beginners en professioneels, mij overkomt het ook nog soms. De computer denkt dan dat het jouw code eeuwig moet uitvoeren maar dat wil u niet, als het goed is

loop dan uw computer niet vast want node.js is gesandbox, dit betekent dat node.js uw computer niet kan locken in een bepaalde state. Als het toch gebeurt moet u [CTRL + C] indrukken als het goed is killt dan uw terminal node. Het ligt er een beetje aan welke terminal u gebruikt maar in cmd, git bash en powershell moet dit zeker wel werken. U kan ook over een array heen lopen dit is een vaak voorkomend fenomeen en niet zo erg lastig u doe het zo als de code hieronder:

```
var names = ['Jan', 'Piet', 'Kaas'];
for (var i = 0; i < names.length; i++) {
  console.log('Hey ' + names[i]);
}
```

Dit is best wel simpel en ook wel logisch als u het zo bekijkt. Want u vraagt de lengte van de array dit zet u als max en u roept console.log als u een nieuw item hebt opgehaald. Er zijn nog meer dingen mogelijk maar om dit kort te houden zal ik die dingen niet uitleggen.

Opdracht: oefen nog wat met loops, arrays en ifs koppel ze aan elkaar en maak zo ook ingewikkeldere scripts.

Functies of te wel uw code recyclen en zo mooi ordenen

U hebt misschien nu wel vaker code gekopieerd en iets aangepast dat wordt vaak gedaan maar het is mooier en overzichtelijker en ook nog sneller om code die u vaak gebruikt maar steeds iets anders in een functie te stoppen. In een functie kan u argumenten stoppen en die krijgt u dan als argumenten in uw code blok beschikbaar. Zoals u kan zien is eigenlijk console.log ook een functie maar dan van node.

```
function add(a, b) {
  return a + b;
}
console.log(add(4, 8));
```

Het return statement is een bijzonder statement want alles wat u daar in stop (maximaal maar een waarde) wordt teruggestuurd naar waar de functie wordt aangeroepen. Het uitvoeren van de functie wordt dan ook gestopt. Want JavaScript kan maximaal maar een ding tegelijk doen. Met deze methode kan u uw code dus opsplitsen in individuele lossen brokjes code die een ding doen. Dat is het mooie van programmeren vind ik ook want als u alleen maar de argumenten van de functie gebruikt en geen ander globale variabelen dat zijn variabelen die buiten de functie zijn aangegeven. Kan u een functie zo over kopiëren naar een ander programmeer project van dezelfde taal omdat die functie dan clean is. Dat was het dan een hele korte introductie voor JavaScript zodat u dit PWS een beetje begrijpt. Wil u meer weten dan raad ik erg sterk aan om <https://eloquentjavascript.net/> te gaan lezen want het is veel completer.

Hoofdstuk 2

> De blockchain wat is het en hoe werkt het? <

Hashing

Voordat ik ga beginnen met het uitleggen van de blockchain moet u het principe van hash functies begrijpen. Een hash functie werkt erg simpel. In de functie stopt u een bepaalde string en de functie retournt dan een erg groot getal uniek aan de string die u erin stopt.

Ik ga voor mijn blockchain de SHA-256 hash functie gebruiken, deze functie wordt ook gebruikt door de Bitcoin dus geloof mij maar hij is goed en schaalbaar. De SHA-256 functie maakt voor een string een uniek 256-bits nummer. Dat is een nummer met het bereik van 0 tot $(2^{256} - 1)$, best een groot nummer dus.

U maakt een SHA-256 hash via de `createHash` functie in de crypto bibliotheek van nodejs. Eerst moet u deze bibliotheek inladen en dan maakt u een nieuw hash object aan, updated u dat hash object en format u die dan in het heximale stelsel. Dit is een veelgebruikt stelsel in de computerwereld en is op 16 gebaseerd in plaats van op 10 als u het decimale stelsel gewend bent. Het gebruikt dan normale 0 tot te met 9 cijfers en dan daarna nog de A tot te met de F. Zo ja kan dus cijfers omzetten in beide systemen. Op deze site staat ook een omreken tooltje: <https://www.binaryhexconverter.com/decimal-to-hex-converter> dat u kan gebruiken voor het omreken naar beide systemen.

```
var crypto = require('crypto');
var hash = crypto.createHash('sha256').update("Hallo Bastiaan!").digest('hex');
console.log(hash);
```

Dit print vervolgens “2d70cc64b5b0da8ad2831bd451666548410717c3789...db826b39d09ff9418b030” uit. Dit is dus het unieke nummer gekoppeld aan de tekst Hallo Bastiaan! Via deze methode kunnen we dus snel en gemakkelijk een uniek nummer aan een bepaalde tekst koppelen en dat gaan we dan weer gebruiken in onze blockchain in de volgende paragraaf.

De blockchain

Met de kennis van hash functies gaan we nu de blockchain bouwen. De blockchain werkt in principe eigenlijk erg simpel want het is niks meer dan een array die gehasht is. Dat wil zeggen dat ieder item of block in de array de hash van zichzelf bevat en dat in die hash de hash van het vorig item zit. Dit heeft het nut dat als u een item in de blockchain wilt aanpassen alle hashjes die daar na komen niet meer kloppen. Zo kunnen we dus checken of er niet met de blockchain is gerommeld. Wel geeft dit als nadeel dat de blockchain statisch is wat er eenmaal in zit kan nooit meer worden veranderd. Zie het plaatje hieronder voor meer info:



Oké nu we de technische kant weten gaan we het namaken met code. Maak een nieuw JavaScript bestand aan. En als eerste regel laat u de crypto bibliotheek.

```
var crypto = require('crypto');
```

Vervolgens maken we een functie waar mee we de hash van een block object berekenen dit doen we door alle items van een block de index, de vorige hash, de tijd waarop de block is gemaakt en de block data in een SHA-256 hash te stoppen.

```
function hashBlock (block) {
  return crypto.createHash('sha256').update(
    block.index + block.previousHash +
    block.timestamp + block.data).digest('hex');
}
```

Daarna maken we ons genesis block aan, hashen we die en beginnen we de blockchain array met genesisblock als eerste block.

```
var genesisBlock = { index: 0, previousHash: '',  
  timestamp: Date.now(), data: 'Genesis Block' };  
genesisBlock.hash = hashBlock(genesisBlock);  
var blockchain = [ genesisBlock ];
```

Het genesis of Genesis (van het eerste Bijbel boek) is een speciaal block want het heeft namelijk geen previoushash item want het is het eerste block. Daarom moeten we dit block altijd hand matig aan maken zodat dat goed gaat.

Vervolgens hebben we een for loop die tien nieuwe items toevoegd aan de blockchain met de data "Block #" waarbij de hashtack is verandert door het nummer van de block. We maken een nieuw block aan door een nieuw object te maken met de index van het vorige block plus een en de hash van de vorige block. In de timestamp voegen we Date.now() toe deze functie geeft de huidige tijd in milliseconden terug dat is erg nauwkeurig en dus perfect. Vervolgens hashen we het nieuw gemaakten block en voegen we hem toe aan de blockchain.

```
for (var i = 0; i < 10; i++) {  
  var previousBlock = blockchain[blockchain.length - 1];  
  var newBlock = { index: previousBlock.index + 1,  
    previousHash: previousBlock.hash,  
    timestamp: Date.now(), data: 'Block ' + i };  
  newBlock.hash = hashBlock(newBlock);  
  blockchain.push(newBlock);  
}
```

Daar na printen we de blockchain om hem te kunnen bekijken in onze terminal.

```
console.log(blockchain);
```

Draai nu het script in node.js en als het goed is ziet u dat er een array komt met allemaal gekoppelde hashjes. Gefeliciteerd u hebt uw eerste werkende blockchain geschreven! Voeg nog wat blokken toe en kijk hoe de blockchain langer wordt.

We hebben nu wel een blockchain maar we hebben nog geen manier om te controleren of de chain wel klopt. Daarom maken we een nieuwe functie aan genaamd `validBlock` die kijkt of twee blokken we kloppen en zo een boolean teruggeeft. Met deze functie gaan we vervolgens alle blokken valideren.

```
function validBlock (newBlock, previousBlock) {
  return typeof newBlock.index == 'number' &&
    typeof newBlock.hash == 'string' &&
    typeof newBlock.previousHash == 'string' &&
    typeof newBlock.timestamp == 'number' &&
    typeof newBlock.data == 'string' &&
    newBlock.index == previousBlock.index + 1 &&
    newBlock.previousHash == previousBlock.hash &&
    hashBlock(newBlock) == newBlock.hash;
}
```

De functie kijkt eerst of alle items van het block object wel goed zijn, hij checkt vervolgens of de index wel de opvolgende is van het vorige block. Ook of de hash van het vorige block wel klopt met de hash van het vorige block. Als laatste checkt hij dat de hash van het nieuwe block wel klopt door hem opnieuw te hashen.

We maken ook nog een functie `validBlockchain` waarmee we de complete blockchain kunnen controleren. Deze functie is erg simpel hij loop over de complete blockchain heen en als een item niet klopt dan return hij false. Alles alle items wel kloppen dan return hij true.

```
function validBlockChain (blockchain) {  
  for (var i = 1; i < blockchain.length; i++) {  
    if (!validBlock(blockchain[i], blockchain[i - 1])) {  
      return false;  
    }  
  }  
  return true;  
}
```

En als laatste roepen we de functie met de blockchain en als het goed is print dit true. Dat betekent dat de blockchain goed is en dat alles dus klopt.

```
console.log(validBlockChain(blockchain));
```

Nu hebben we een manier gemaakt om onze blockchain te valideren en zo kunnen we dus ook de blockchain van andere nodes controleren maar over dat straks meer.

Een proof-of-work systeem

We hebben nu wel een werkende blockchain maar hij is niet beveiligd want het hashen van een blok is zo snel dat iemand iets kan aanpassen aan de blockchain en alle volgende blokken ook opnieuw kan hashen want het hashen van een blok gaat toch zo snel. Ook kan iedereen op elk moment nieuwe blokken genereren en zo de server overvloeden of spammen met allemaal nutteloze blokken. Om dit probleem op te lossen gaan we gebruik maken van een proof-of-work systeem. Dit systeem zorgt ervoor dat het erg veel berekeningen dus tijd en energie kost om een blok uit te rekenen. Dit geeft als voordeel dat het onmogelijk is om snel nieuwe blokken uit te rekenen en dus ook om de chain opnieuw te genereren. Maar heeft wel als nadeel dat het heel erg veel stroom kost om nieuwe blokken aan te maken dit systeem zorgt er ook voor dat de Bitcoin zo veel geld kost om operationeel te houden want iedereen is de heel tijd bezig nieuwe blokken aan het hashen om zo de goeie hash te vinden een soort schatten jacht dus.

Om een proof-of-work systeem toe te voegen aan onze block chain voegen we de nonce nummer toe aan ieder block. Dus de hashBlock functie ziet er zo uit:

```
function hashBlock (block) {  
  return crypto.createHash('sha256').update(  
    block.index + block.previousHash +  
    block.timestamp + block.data +  
    block.nonce).digest('hex');  
}
```

Vervolgens maken we een nieuwe functie genaamd mineBlock deze functie gaat zoeken naar de goeie hash. De moeilijkheid dus hoelang het duurt word opgeslagen in de globale variabele difficulty. De while is een soort for loop maar dan zonder de initialisatie en de increment gedeelte want die zit in het while block. Deze code zoekt steeds de hash en als het niet goed is verhoogd hij de nonce met een. En hash hij het block opnieuw zo komen we uiteindelijk bij de goede hash uit. Door er steeds overnieuw er op in te gaan.

```
function mineBlock (block) {  
  block.hash = hashBlock(block);  
  while (block.hash.substring(0, difficulty) !=  
    '0'.repeat(difficulty)) {  
    block.nonce++;  
    block.hash = hashBlock(block);  
  }  
}
```

We zetten de difficulty standaard op drie en we passen de code van het genesisBlock iets aan zodat het de nieuwe mineBlock functie gaat gebruiken.

BasieCoin

```
var difficulty = 3;
var genesisBlock = { index: 0, previousHash: '',
  timestamp: Date.now(), data: 'Genesis Block', nonce: 0 };
mineBlock(genesisBlock);
var blockchain = [ genesisBlock ];
```

Vervolgens passen we de for loop ook wat aan zodat hij de nieuwe mineBlock functie gaat gebruiken. Dit is dus nu de manier om nieuwe blocken aan te maken.

```
for (var i = 0; i < 10; i++) {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(), data: 'Block ' + i, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);
}

console.log(blockchain);
```

Ook de validBlock functie passen we iets aan want hij moet nu ook gaan letten dat de nonce item een nummer is een dat dat gecontroleerd wordt.

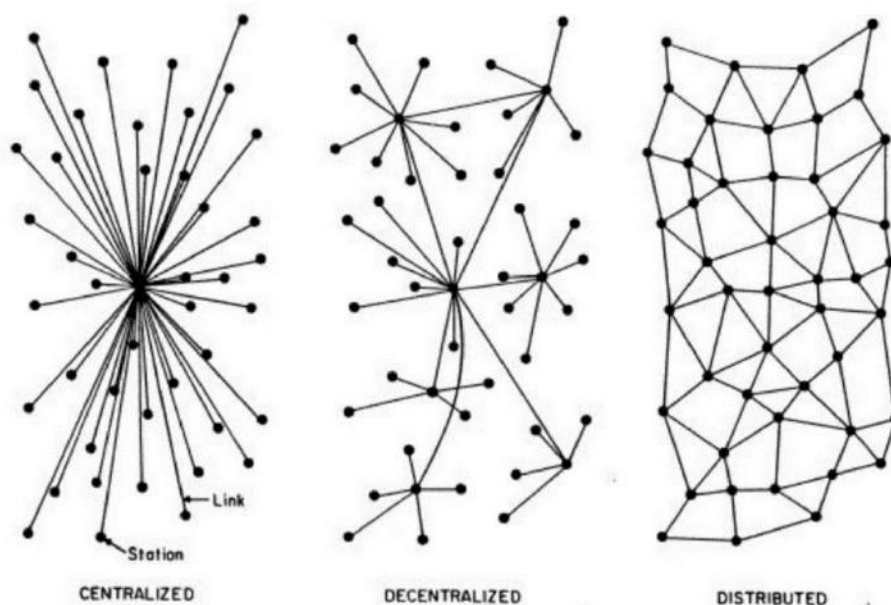
```
function validBlock (newBlock, previousBlock) {
  return typeof newBlock.index == 'number' &&
    typeof newBlock.hash == 'string' &&
    typeof newBlock.previousHash == 'string' &&
    typeof newBlock.timestamp == 'number' &&
    typeof newBlock.data == 'string' &&
    typeof newBlock.nonce == 'number' &&
    newBlock.index == previousBlock.index + 1 &&
    newBlock.previousHash == previousBlock.hash &&
    hashBlock(newBlock) == newBlock.hash;
}
```

De rest van de code kan hetzelfde blijven. Als het goed is draait het programma nu veel trager want het kost even tijd om voor alle blocks de goeie hash te vinden dit noemen ze ook wel mijnen in het crypto jargon.

Nu hebben we een compleet proof-of-work systeem ingebouwd in onze blockchain. In het echt is de difficulty bij veel blockchains wel veel groter dan die van ons maar anders duurt het echt te lang om een block te mijnen. Dus probeer wat waardes uit en zoek de goede die bij u past.

Het peer-to-peer netwerk

We hebben nu een blockchain gemaakt maar hij werkt alleen op onze eigen computer en we kunnen niet gemakkelijk nieuwe blokken toevoegen. De meeste blockchains lossen dit probleem op door meerdere servers te maken die met elkaar te communiceren. Er zijn verschillende manieren om deze servers te laten communiceren. U hebt de gecentraliseerde manier, de decentrale manier en de gedistribueerde manier. Geen enkele blockchain is gecentraliseerd want als de hoofdservers niet werkt dan werken alle andere servers ook niet en dat wilt u juist voorkomen met de blockchain. Decentraal is erg populair maar dat is nog steeds niet perfect want als een paar hoofdservers uitvallen dan licht nog steeds het netwerk plat. Gedistribueerd zou dan de oplossing moeten zijn maar dit is dan wel de lastigste manier om te bouwen, zie hieronder in het plaatje voor meer info.



Voor de gemakkelijkerheid maak ik in dit PWS de server niet peer-to-peer wat betekent dat de servers niet met elkaar kunnen praten. Dit heeft wel als voordeel dat ik de implementatie van onze server het simpelst mogelijk kan houden.

Voeg de volgende stukjes code onder uw andere regels code toe. Eerst laden we de http en de url bibliotheek. Deze twee bibliotheken geven ons de mogelijkheid om een webserver te starten waarmee mensen onze blockchain kunnen bezoeken en ook om urls te kunnen parsen. Urls zijn het complete internet namen met http ervoor en GET-variabelen. Vervolgens maken we een nieuwe http-server aan. In deze functie stoppen we dan een andere functie die elke keer als er een request naar de server gedaan wordt aangeroepen wordt. De betekend dat elke keer als u een pagina laadt de pagina gemaakt wordt door deze functie.

```
var http = require('http');
var url = require('url');

var server = http.createServer(function (request, response) {
  var url_parts = url.parse(request.url, true),
      pathname = url_parts.pathname,
      query = url_parts.query;
```

In de functie parsen we ook nog de url van de request en halen we de pathname dat is het dik gedrukte in de volgende url: <http://localhost:8080/bestand?q=das> er uit en ook nog de GET variabelen in de query variabele, dit is het gedeelte na het vraagteken.

Het volgende stukje code moet u eronder plakken. Deze code check of de pathname "/" is. Als dat zo is ze hij het content-type van de http-headers op application/json en stuurt hij de complete blockchain terug. Dit komt erop neer dat als u naar de homepage van onze webserver gaat hij de complete blockchain aan u terug geeft. Dit doet hij in JSON dat is een speciale manier van het weergeven van objecten en arrays in de vorm die erg lijkt op JavaScript.

```
if (pathname == '/') {
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify(blockchain));
}
```

De volgende regels code maken dat als u naar /validate gaat hij check of de blockchain wel klopt en als dat zo is zegt hij dat validated: true is.

```
else if (pathname == '/validate') {
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'validated': validBlockChain(blockchain) }));
}
```

Het volgende wat langere stukje maak de /mineBlock pagina aan. Deze pagina roep u aan met de GET variabel data. Deze data gebruikt hij om een nieuw block aan te maken in de blockchain. Als het goed is, is deze pagina ook wat trager omdat hij eerst een nieuw block moeten mijnen.

```
else if (pathname == '/mineBlock') {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(), data: query.data, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}
```

De volgende pagina is de code van alle overige pagina's. Als u op zo'n onzin pagina komt dan zegt hij niet gevonden in het Engels. De 404 http-error code zegt tegen de browser dat de pagina eigenlijk niet bestaat.

```
else {
  response.writeHead(404, { 'Content-Type': 'text/html' });
  response.end('<h1>404 Not Found!</h1>');
}
```

De laatste regels sluiten de lange functie af en laten de webserver luisteren op localhost:8080, ook zegt hij dat nog als de server opgestart is.

```
});  
server.listen(8080, '127.0.0.1', function () {  
  console.log('Server is listening on: http://localhost:8080/');  
});
```

Als het goed is heeft u nu een web interface gemaakt met uw blockchain als u codeblokken wat onoverzichtelijk vond dan kan u naar de volgende pagina gaan want daar staan nog een keer de complete code van hoofdstuk 2.

U moet nu het script draaien en in uw webbrowser naar <http://localhost:8080/> gaan als het goed is ziet u nu uw blockchain! Als u een block wilt toevoegen moet u naar <http://localhost:8080/mineBlock?data=Nieuw%20Block> gaan. En als u nog wilt checken of dat blockchain wel valide is dan moet u naar <http://localhost:8080/validate> gaan.

Via deze web interface kunt u nu zonder de terminal te gebruiken of daar het script aan te passen uw blockchain aanpassen. Weet wel dat als u de server weer uit wilt doen via [CTRL+C] dat dan alle ingevoerde data weer weg gaat. Want hij onthoudt de blockchain alleen maar in het geheugen.

In het volgende hoofdstuk gaan we de blockchain uitbreiden om er zo een cryptocurrency van te maken, maar u kan deze basis blockchain voor alle soorten data gaan gebruiken.

[Nog de complete code](#)

De complete code staat op de volgende pagina's.

BassieCoin

```
var crypto = require('crypto');

function hashBlock (block) {
  return crypto.createHash('sha256').update(
    block.index + block.previousHash +
    block.timestamp + block.data +
    block.nonce).digest('hex');
}

function mineBlock (block) {
  block.hash = hashBlock(block);
  while (block.hash.substring(0, difficulty) !=
    '0'.repeat(difficulty)) {
    block.nonce++;
    block.hash = hashBlock(block);
  }
}

var difficulty = 3;
var genesisBlock = { index: 0, previousHash: '',
  timestamp: Date.now(), data: 'Genesis Block', nonce: 0 };
mineBlock(genesisBlock);
var blockchain = [ genesisBlock ];

for (var i = 0; i < 10; i++) {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(), data: 'Block ' + i, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);
}

console.log(blockchain);

function validBlock (newBlock, previousBlock) {
  return typeof newBlock.index == 'number' &&
    typeof newBlock.hash == 'string' &&
    typeof newBlock.previousHash == 'string' &&
    typeof newBlock.timestamp == 'number' &&
    typeof newBlock.data == 'string' &&
    typeof newBlock.nonce == 'number' &&
    newBlock.index == previousBlock.index + 1 &&
    newBlock.previousHash == previousBlock.hash &&
    hashBlock(newBlock) == newBlock.hash;
}
```

BassieCoin

```
console.log(validBlockChain(blockchain));

var http = require('http');
var url = require('url');

var server = http.createServer(function (request, response) {
  var url_parts = url.parse(request.url, true),
      pathname = url_parts.pathname,
      query = url_parts.query;

  if (pathname == '/') {
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify(blockchain));
  }

  else if (pathname == '/validate') {
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify({ 'validated': validBlockChain(blockchain) }));
  }

  else if (pathname == '/mineBlock') {
    var previousBlock = blockchain[blockchain.length - 1];
    var newBlock = { index: previousBlock.index + 1,
                    previousHash: previousBlock.hash,
                    timestamp: Date.now(), data: query.data, nonce: 0 };
    mineBlock(newBlock);
    blockchain.push(newBlock);
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify({ 'success': true }));
  }

  else {
    response.writeHead(404, { 'Content-Type': 'text/html' });
    response.end('<h1>404 Not Found!</h1>');
  }
});

server.listen(8080, '127.0.0.1', function () {
  console.log('Server is listening on: http://localhost:8080/');
});
```

Hoofdstuk 3

> We hebben een blockchain hoe nu verder? <

Transacties

In het vorige hoofdstuk hebben we een werkende blockchain gemaakt. We gaan in dit hoofdstuk daarop verder bouwen. Een cryptocurrency gebruikt ook een blockchain maar in plaats van data zit er een lijst van transacties in elk block. In transacties zijn gewoon objecten met het adres van de zender, het adres van de ontvanger en het aantal coins wat er moet over worden verzonden.

We willen eerst onze *hashBlock* functie aanpassen zodat het de transacties in de hash mee stopt. Ik heb het data veld verwijderd want die dient nu nergens meer voor. Ook willen we net zoals bij onze webserver de transactie array omzetten naar JSON zodat het een string word en we het dus ook kunnen hashen.

```
function hashBlock (block) {  
  return crypto.createHash('sha256').update(  
    block.index + block.previousHash +  
    block.timestamp + JSON.stringify(block.transactions) +  
    block.nonce).digest('hex');  
}
```

Vervolgens voegen we de functie *minePendingTransactions* toe. Deze functies maakt een nieuw block aan en voegt in dat block alle wachtende transacties van dat moment. De mijn functie blijft trouwens gewoon hetzelfde. Als deze functie een nieuw block heeft gemijnd leegt hij de wachtende transactie want deze zijn verwerkt in de blokchain en maakt hij een nieuwe transactie aan. Deze transactie heeft ook wel de mining reward. Het mijnen van een block kost namelijk erg veel energie en als bedankje krijgt de mijner van een block bij de volgende block een klein aantal coins. Dit is de enigste manier hoe nieuwe munten in het netwerk kunnen komen en alle grote cryptocurrency werken via deze manier. Bij de Bitcoin krijgt u bijvoorbeeld 10,5 Bitcoin als u een blok mijnt.

```
function minePendingTransactions (miningRewardAddress) {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(),
    transactions: pendingTransactions, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);
  pendingTransactions = [
    { from: 'mining-reward-address',
      to: miningRewardAddress,
      amount: MINING_REWARD }
  ];
}
```

Verder maken we onder de difficulty variabel de pendingTransactions array en een constante (dat is een variabel die nooit verandert) waarin de mining reward staat. Als u wilt dat de mining reward hoger is bij uw cryptocurrency kan u dit verhogen of verlagen.

```
var difficulty = 3;
var pendingTransactions = [];
var MINING_REWARD = 100;

var genesisBlock = { index: 0, previousHash: '',
  timestamp: Date.now(), transactions: [], nonce: 0 };
mineBlock(genesisBlock);
var blockchain = [ genesisBlock ];
```

Vervolgens passen we de code aan die wat random blokken gegenereerd om onze blockchain wat te vullen. We mijnen nu altijd eerst een block en geven de mining reward aan het adres *bastiaan-address* vervolgens maken we een nieuwe transactie aan van Bastiaan naar Jan van 50 coins. Op dit moment is er geen controle over of een bepaald adres wel de goede hoeveelheid coins heeft om wel een transactie te maken maar zo iets kunnen we later altijd nog maken.

₿BassieCoin

```
for (var i = 0; i < 10; i++) {
  minePendingTransactions('bastiaan-address');
  pendingTransactions.push({ from: 'bastiaan-address',
    to: 'jan-address', amount: 50 });
}

console.log(blockchain);
```

En als laatste printen we nog de blockchain. Ook maken we onder de validatie code nog een nieuwe functie aan waarmee we kunnen kijken hoeveel coins een bepaald adres heeft. Dit berekenen we door over alle blocks en alle transacties heen te lopen en dan als hij wat weg geeft het er aftrekt en als het er bij komt wat bij doen. Deze functie retournt dan een nummer met het aantal coins.

```
function getBalanceOfAddress (address) {
  var balance = 0;
  for (var i = 0; i < blockchain.length; i++) {
    for (var j = 0; j < blockchain[i].transactions.length; j++) {
      if (blockchain[i].transactions[j].from == address) {
        balance -= blockchain[i].transactions[j].amount;
      }
      if (blockchain[i].transactions[j].to == address){
        balance += blockchain[i].transactions[j].amount;
      }
    }
  }
  return balance;
}
```

En als laats printen we nog de balans uit van Bastiaan en van Jan. Als het goed is moet hier bij 450 coins uit komen.

```
console.log('Bastiaan: ' + getBalanceOfAddress('bastiaan-address'));
console.log('Jan: ' + getBalanceOfAddress('jan-address'));
```


Gefeliciteerd, u hebt van uw blockchain nu uw eerste cryptocurrency gemaakt. We moeten nog veel doen maar de basis ligt er en die is hetzelfde als bijvoorbeeld de Bitcoin. Onze cryptocurrency kan trouwens ook komma getallen aan. Dus kan u ook kleine hoeveelheden geld over maken naar elkaar.

De webserver met transacties

We gaan nu ook de webserver code iets aanpassen zodat deze ook met de nieuwe cryptocurrency kan gaan werken. De code blijft redelijk hetzelfde alleen moet u eerst een nieuwe pagina toevoegen genaamd *addTransaction*. Met deze pagina kunnen we zo via de webbrowser nieuwe transacties toevoegen.

```
else if (pathname == '/addTransaction') {
  pendingTransactions.push({ from: query.from, to: query.to,
    amount: parseInt(query.amount) });
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}
```

Vervolgens passen we de mijn code ook wat aan zodat het onze mijn block functie gebruikt. Ook moeten we als we deze pagina aan willen roepen in de GET-variabel address het reword address toevoegen.

```
else if (pathname == '/mineBlock') {
  minePendingTransactions(query.address);
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}
```

Als laatste maken we een nieuwe pagina aan waarmee we kunnen checken hoeveel coins iemand heeft in onze cryptocurrency. Deze pagina roept ook gewoon de balans functie op en gebruikt als input de GET-variabel address.

```

else if (pathname == '/getBalanceOfAddress') {
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'balance':
    getBalanceOfAddress(query.address) }));
}

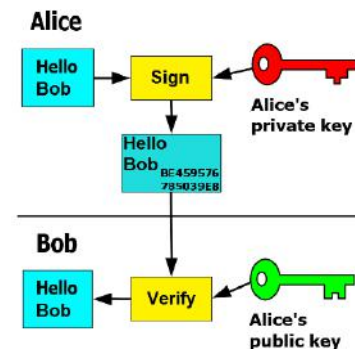
```

Als u de volledige code wilt checken moet u even naar het einde van dit hoofdstuk gaan. En als u alles goed het aangepast. Dan moet u via node.js het programma weer opnieuw opstarten. Als u dan naar <http://localhost:8080/> gaat ziet u de blockchain die we hebben gemaakt. U kan nu ook nieuwe transacties toevoegen door naar <http://localhost:8080/addTransaction?from=bastiaan-address&to=jan-address&amount=50> te gaan. De adressen kan u natuurlijk aanpassen en zo ook de hoeveelheid die u wilt sturen. Als u wat transacties hebt toegevoegd kan u een block mijnen door naar <http://localhost:8080/mineBlock?address=bastiaan-address> te gaan. Als het goed is duurt deze pagina wat langer om te laden. Omdat hij nog een block moet mijnen. Daarna als u succes terug krijgt moet u terug gaan naar home en ziet u als u naar beneden scrolt uw nieuw gemaakte block met daarin de transacties die u had toegevoegd.

Transacties beveiligen met een sleutel

De cryptocurrency die we nu hebben werkt eigenlijk wel prima. Maar er is een groot probleem want iedereen kan transactie voor iedereen maken. Stelt u eens een wereld voor waar iedereen elkaars geld kan uitgeven dat werkt natuurlijk niet. Zo is het nauw ook met onze cryptocurrency we moeten een manier verzinnen waarmee alleen u een transactie kan maken en niemand anders.

Dat is mogelijk met public-cryptografie. Werkt een beetje het zelfde als het hashen van teksten. Iedere gebruiker van de cryptocurrency heeft twee sleutels een privé sleutel en een publieke sleutel. Met de privé sleutel genereert u per transactie een handtekening over de hash van de transactie. En met de publieke



sleutel kan iedereen verifiëren of dat wel klopt. De publieke sleutel is ook gewoon het standaard adres waar iedereen uw Bitcoins naar toe stuurt. Zo kan alleen jij nieuwe transacties maken voor jouw adres. Dit heeft natuurlijk wel het nadeel dat als u uw privé sleutel kwijtraakt en of bijvoorbeeld gestolen wordt. U alles kwijt bent want dan kan niemand of iedereen weer transactie maken voor jouw. Bij de Bitcoin is bijvoorbeeld berekend dat ongeveer 20% van alle Bitcoins verloren is gegaan omdat de sleutels kwijt zijn geraakt.

Dit verhaaltje kunnen we natuurlijk ook met code maken. Eerst laden we de o zo handige crypto bibliotheek. Vervolgens genereren we een nieuwe keypair dat zijn een publieke en een privé sleutel die aan elkaar gekoppeld zijn. Vervolgens hebben we een bericht. En voor de bericht maken we een signature met de privé sleutel en die handtekening verifiëren we dan met de publieke sleutel.

```
var crypto = require('crypto');

var keys = crypto.generateKeyPairSync('rsa', {
  modulusLength: 2048,
  publicKeyEncoding: { type: 'spki', format: 'pem' },
  privateKeyEncoding: { type: 'pkcs8', format: 'pem' }
});
console.log(keys.publicKey, keys.privateKey);

var message = 'Hallo Bastiaan!';
var signature = crypto.createSign("sha256").update(message)
  .sign(keys.privateKey, 'hex');
console.log(message, signature);
console.log(crypto.createVerify('sha256').update(message)
  .verify(keys.publicKey, signature, 'hex'));
```

Het eerste we moeten doen als we onze transacties willen beveiligen is een paar kleine maar erg handige functies aanmaken. De eerste heet *hashTransaction*, zo als u waarschijnlijk al denk heeft deze functie een transactie en return hij de hash. We moeten onze transacties hashen omdat we de hash gaan signeren. U zou denken dat we net zo goed het complete object kunnen signen, maar het signen is een erg sloom proces en de hash is toch uniek aan de transactie dus hierdoor gaat het gehele proces een stuk sneller.

BassieCoin

```
function hashTransaction (transaction) {
  return crypto.createHash('sha256').update(
    transaction.from + transaction.to +
    transaction.amount + transaction.publicKey).digest('hex');
}
```

De volgende functie die we maken is de *signTransaction* functie deze functie heeft de transactie nodig en een privé sleutel met deze sleutel maakt hij dan een handtekening / signature en dit ze we als item aan de transactie vast.

```
function signTransaction (transaction, privateKey) {
  transaction.hash = hashTransaction(transaction);
  transaction.signature = crypto.createSign("sha256")
    .update(transaction.hash).sign(privateKey, 'hex');
}
```

Vervolgens passen we de *minePendingTransactions* functie wat aan zodat de mining reward voortaan ook een hash en ook de andere essentiële items bevat.

```
function minePendingTransactions (miningRewardAddress) {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(),
    transactions: pendingTransactions, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);

  var miningRewardTransaction = { from: 'mining-reward-address',
    to: miningRewardAddress, amount: MINING_REWARD,
    publicKey: '', hash: '', signature: '' };
  miningRewardTransaction.hash = hashTransaction(miningRewardTransaction);
  pendingTransactions = [ miningRewardTransaction ];
}
```

Na dat passen we de code die onze blockchain vult wat verder aan zodat hij voortaan onze transacties ook sign met de publieke en privé sleutel / key. Ook maken we aan het begin nog een sleutel paar aan en die noemen we *bastiaanKeys* want het zijn de sleutels die gekoppeld zijn aan dat bepaalde adres.

```
var bastiaanKeys = crypto.generateKeyPairSync('rsa', {
  modulusLength: 2048,
  publicKeyEncoding: { type: 'spki', format: 'pem' },
  privateKeyEncoding: { type: 'pkcs8', format: 'pem' }
});

for (var i = 0; i < 10; i++) {
  minePendingTransactions('bastiaan-address');
  var transaction = { from: 'bastiaan-address',
    to: 'jan-address', amount: 50 };
  transaction.publicKey = bastiaanKeys.publicKey;
  signTransaction(transaction, bastiaanKeys.privateKey);
  pendingTransactions.push(transaction);
}
```

Ook voegen we nog een functie toe die een transactie kan valideren. Hij kijkt eerst of het een mining reward is en als dat zo is dan hoeft hij het niet te valideren want we maken die zelf en we weten dus dat die goed zijn. Verder controleert hij of de items wel goed zitten, of de hash wel klopt en of de signature wel past bij de mee gegeven publieke sleutel.

```
function validTransaction (transaction) {
  if (transaction.from == 'mining-reward-address') return true;
  return typeof transaction.from == 'string' &&
    typeof transaction.to == 'string' &&
    typeof transaction.amount == 'number' &&
    typeof transaction.publicKey == 'string' &&
    typeof transaction.hash == 'string' &&
    typeof transaction.signature == 'string' &&
    transaction.hash == hashTransaction(transaction) &&
    crypto.createVerify('sha256').update(transaction.hash)
      .verify(transaction.publicKey, transaction.signature, 'hex')
}
}
```

Als laats passen we de *validBlock* functie aan zodat hij over al zijn transacties heen loopt en die stuk voor stuk controleert met de net gemaakte functie *validTransaction*. Zo kunnen we dus de complete blockchain controleren en dus checken of er niet mee geprutst is.

```
function validBlock (newBlock, previousBlock) {
  if (typeof newBlock.index == 'number' &&
      typeof newBlock.hash == 'string' &&
      typeof newBlock.previousHash == 'string' &&
      typeof newBlock.timestamp == 'number' &&
      newBlock.transactions.constructor == Array &&
      typeof newBlock.nonce == 'number' &&
      newBlock.index == previousBlock.index + 1 &&
      newBlock.previousHash == previousBlock.hash &&
      hashBlock(newBlock) == newBlock.hash) {

    for (var i = 0; i < newBlock.transactions; i++) {
      if (!validTransaction(
        newBlock.transactions[i])) {
        return false;
      }
    }
    return true;
  }
  return false;
}
```

Dat was het dan, we hebben nu onze cryptocurrency beveiligd en alleen jij kan nog een bepaalde transactie maken.

De versleutelde webserver

We moeten alleen nog een klein stukje van onze server code aanpassen zodat het ook transacties kan signeren via de web interface. Alleen de pagina */addTransaction* moet worden aangepast de rest blijft allemaal hetzelfde. We vragen nu ook de public en private sleutel in de GET-request zodat mensen ook nieuwe transacties kunnen maken op die manier.

```
else if (pathname == '/addTransaction') {
  var newTransaction = { from: query.from, to: query.to,
    amount: parseInt(query.amount), publicKey: query.publicKey };
  signTransaction(newTransaction, query.privateKey);
  pendingTransactions.push(newTransaction);
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}
```

We zijn nu klaar met het maken van onze demo cryptocurrency. Ik hoop dat u er een hoop van hebt geleerd en zo ook in aanraking bent gekomen met het programmeren. De rest van dit Profiel Werkstuk gaat vooral over de cryptocurrency als een gamechanger in de economie en de voor en nadelen ervan.

De complete code van onze cryptocurrency

De complete code staat op de volgende pagina's.

BassieCoin

```
var crypto = require('crypto');

function hashBlock (block) {
  return crypto.createHash('sha256').update(
    block.index + block.previousHash +
    block.timestamp + JSON.stringify(block.transactions) +
    block.nonce).digest('hex');
}

function mineBlock (block) {
  block.hash = hashBlock(block);
  while (block.hash.substring(0, difficulty) !==
    '0'.repeat(difficulty)) {
    block.nonce++;
    block.hash = hashBlock(block);
  }
}

function hashTransaction (transaction) {
  return crypto.createHash('sha256').update(
    transaction.from + transaction.to +
    transaction.amount + transaction.publicKey).digest('hex');
}

function signTransaction (transaction, privateKey) {
  transaction.hash = hashTransaction(transaction);
  transaction.signature = crypto.createSign("sha256")
    .update(transaction.hash).sign(privateKey, 'hex');
}

function minePendingTransactions (miningRewardAddress) {
  var previousBlock = blockchain[blockchain.length - 1];
  var newBlock = { index: previousBlock.index + 1,
    previousHash: previousBlock.hash,
    timestamp: Date.now(),
    transactions: pendingTransactions, nonce: 0 };
  mineBlock(newBlock);
  blockchain.push(newBlock);

  var miningRewardTransaction = { from: 'mining-reward-address',
    to: miningRewardAddress, amount: MINING_REWARD,
    publicKey: '', hash: '', signature: '' };
  miningRewardTransaction.hash = hashTransaction(miningRewardTransaction);
  pendingTransactions = [ miningRewardTransaction ];
}

var difficulty = 3;
var pendingTransactions = [];
var MINING_REWARD = 100;

var genesisBlock = { index: 0, previousHash: '',
  timestamp: Date.now(), transactions: [], nonce: 0 };
mineBlock(genesisBlock);
var blockchain = [ genesisBlock ];
```


BassieCoin

```
var bastiaanKeys = crypto.generateKeyPairSync('rsa', {
  modulusLength: 2048,
  publicKeyEncoding: { type: 'spki', format: 'pem' },
  privateKeyEncoding: { type: 'pkcs8', format: 'pem' }
});

for (var i = 0; i < 10; i++) {
  minePendingTransactions('bastiaan-address');
  var transaction = { from: 'bastiaan-address',
    to: 'jan-address', amount: 50 };
  transaction.publicKey = bastiaanKeys.publicKey;
  signTransaction(transaction, bastiaanKeys.privateKey);
  pendingTransactions.push(transaction);
}

console.log(blockchain);

function validTransaction (transaction) {
  if (transaction.from == 'mining-reward-address') return true;
  return typeof transaction.from == 'string' &&
    typeof transaction.to == 'string' &&
    typeof transaction.amount == 'number' &&
    typeof transaction.publicKey == 'string' &&
    typeof transaction.hash == 'string' &&
    typeof transaction.signature == 'string' &&
    transaction.hash == hashTransaction(transaction) &&
    crypto.createVerify('sha256').update(transaction.hash)
      .verify(transaction.publicKey, transaction.signature, 'hex')
}

function validBlock (newBlock, previousBlock) {
  if (typeof newBlock.index == 'number' &&
    typeof newBlock.hash == 'string' &&
    typeof newBlock.previousHash == 'string' &&
    typeof newBlock.timestamp == 'number' &&
    newBlock.transactions.constructor == Array &&
    typeof newBlock.nonce == 'number' &&
    newBlock.index == previousBlock.index + 1 &&
    newBlock.previousHash == previousBlock.hash &&
    hashBlock(newBlock) == newBlock.hash) {

    for (var i = 0; i < newBlock.transactions; i++) {
      if (!validTransaction(newBlock.transactions[i])) {
        return false;
      }
    }
    return true;
  }
  return false;
}
```

BassieCoin

```
function validBlockChain (blockchain) {
  for (var i = 1; i < blockchain.length; i++) {
    if (!validBlock(blockchain[i], blockchain[i - 1])) {
      return false;
    }
  }
  return true;
}

console.log(validBlockChain(blockchain));

function getBalanceOfAddress (address) {
  var balance = 0;
  for (var i = 0; i < blockchain.length; i++) {
    for (var j = 0; j < blockchain[i].transactions.length; j++) {
      if (blockchain[i].transactions[j].from == address) {
        balance -= blockchain[i].transactions[j].amount;
      }
      if (blockchain[i].transactions[j].to == address){
        balance += blockchain[i].transactions[j].amount;
      }
    }
  }
  return balance;
}

console.log('Bastiaan: ' + getBalanceOfAddress('bastiaan-address'));
console.log('Jan: ' + getBalanceOfAddress('jan-address'));

var http = require('http');
var url = require('url');

var server = http.createServer(function (request, response) {
  var url_parts = url.parse(request.url, true),
      pathname = url_parts.pathname,
      query = url_parts.query;

  if (pathname == '/') {
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify(blockchain));
  }

  else if (pathname == '/validate') {
    response.writeHead(200, { 'Content-Type': 'application/json' });
    response.end(JSON.stringify({ 'validated': validBlockChain(blockchain) }));
  }
});
```

BassieCoin

```
else if (pathname == '/addTransaction') {
  var newTransaction = { from: query.from, to: query.to,
    amount: parseInt(query.amount), publicKey: query.publicKey };
  signTransaction(newTransaction, query.privateKey);
  pendingTransactions.push(newTransaction);
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}

else if (pathname == '/mineBlock') {
  minePendingTransactions(query.address);
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'success': true }));
}

else if (pathname == '/getBalanceOfAddress') {
  response.writeHead(200, { 'Content-Type': 'application/json' });
  response.end(JSON.stringify({ 'balance':
    getBalanceOfAddress(query.address) }));
}

else {
  response.writeHead(404, { 'Content-Type': 'text/html' });
  response.end('<h1>404 Not Found!</h1>');
}
});
server.listen(8080, '127.0.0.1', function () {
  console.log('Server is listening on: http://localhost:8080/');
});
```

Hoofdstuk 4

> Wie heeft het nauw verzonnen? <

Geschiedenis van de Bitcoin

Er was eens een mens met het pseudoniem Satoshi Nakamoto, hij had een idee, een groot idee, een idee waarmee hij de wereld kon veranderen van binnenuit. Hij wou het complete monetaire systeem omgooi. Overheden en banken hebben te veel macht en ze misbruiken die macht ook. Daardoor gebeuren er dingen zoals de huizen crisis. En dat schaadt de uiteindelijk werkende consument. Hij zag dat die macht ook steeds groter werd door de virtualisatie van het geld. Weinig mensen tegenwoordig betalen nog contant. Dat kon anders, mensen moeten geld kunnen overmaken zonder tussen personen zoals banken of overheden.

Er zijn al verschillende oplossingen voor dit probleem bedacht maar ze hadden allemaal een valkuil: het double spending problem. Dit probleem is dat een virtuele munt meerdere keren kan worden uitgegeven. U kan dit oplossen door een centrale server waarmee iedereen verbindingen maakt. Maar dit geeft die een centrale server alle macht. En is ook wat de banken vandaag de dag gebruik van maken. Een decentraal netwerk is beter. Want als een server sluit blijft het netwerk gewoon nog werken. Maar bij een decentraal netwerk kan een munt meerdere keren worden uitgegeven want niet alle servers slaan de transactie te gelijk op. Daarom schreef Satoshi een document en plaatste hij die op het internet waarmee hij de blockchain uitlegde. Door die blockchain wordt het probleem opgelost en kan het ook decentraal gedraaid worden.

Hij plaatste de code voor zijn coin bitcoin genaamd op internet en al gauw gingen mensen mee werken aan het project. Zo ontstond de bitcoin die vandaag de dag nog steeds gebruikt wordt. Snel na het op internet zetten van de code gingen andere mensen ook hun eigen coins maken en zo ontstonden de coins die vandaag de dag nog steeds worden gebruikt. In het volgende hoofdstuk ga ik wat dieper in op verschillende coins die nu in omloop zijn.











Gevaren van het investeren in de cryptovaluta

U hebt het vast wel een keer in het nieuws gehoord dat er veel gevaren zijn bij het investeren in cryptovaluta zoals de Bitcoin of de Monero. Dat komt omdat de prijzen die u voor het omruilen van cryptovaluta naar echte valuta zeer erg fluctueren. Daarmee valt erg veel geld mee te verdienen. Maar daar mee valt ook veel geld mee te verliezen. Er zijn mensen die alles hebben verkocht om bitcoins mee te kopen in de hoop dat het meer waard zou worden. En die toen alles kwijt zijn geraakt omdat het een paar maanden geleden allemaal is geklapt. Op het hoogste punt was de bitcoin wel 20.000 dollar per stuk waard. Maar nu schommelt het een beetje tussen de twee en de zes duizend. Wel is bitcoin de enigste cryptovaluta die zo veel waard is de meeste zijn namelijk veel minder waard daar ga ik het over hebben in het volgende hoofdstuk.

Hoofdstuk 5

> Welke cryptocurrencies zijn er nog meer? <

Zoals u misschien weet zijn er veel verschillende cryptocurrencies in omloop. Duizenden coins worden nu elke dag gebruikt maar een handje vol van die coins is echt populair geworden. Hier onder staat een tabel van de tien beroemdste cryptocurrencies die op het moment van schrijven de grootste marktkapitalisatie hebben. Waarschijnlijk is deze lijst weer heel snel verandert, en dus totaal verandert als u dit leest op een later tijdstip. De tabel bevat de prijs per coin het aantal beschikbare coins en de mark kapitalisatie van de coin dat is de prijs keer het aantal beschikbare coins, daar komen meestal best hoge bedragen uit. Maar dat is niks vergeleken met normaal geld want er zijn triljarden euro's en dollars in omloop.

#	Naam	Prijs	Aantal beschikbaar	Marktkapitalisatie
1	 Bitcoin	€ 5,546.10	17,252,037 BTC	€ 95,681,494,563
2	 Ethereum	€ 196.69	101,785,726 ETH	€ 20,020,243,431
3	 XRP	€ 0.25	39,650,153,121 XRP	€ 9,856,517,104
4	 Bitcoin Cash	€ 432.99	17,333,400 BCH	€ 7,505,237,449
5	 EOS	€ 4.33	906,245,118 EOS	€ 3,926,940,596
6	 Stellar	€ 0.17	18,773,730,637 XLM	€ 3,211,145,373
7	 Litecoin	€ 47.90	58,169,503 LTC	€ 2,786,181,839
8	 Tether	€ 0.86	2,756,421,736 USDT	€ 2,375,650,627
9	 Cardano	€ 0.07	25,927,070,538 ADA	€ 1,882,535,109
10	 Monero	€ 95.57	16,382,247 XMR	€ 1,565,618,834

De tien meest waardevolle en dus meest gebruikte cryptocurrencies op 6 september 2018 om 10 uur 's ochtends (bron cryptolist.ml)

De site die ik heb gebruikt (cryptolist.ml) heb ik zelf gemaakt u kan hem zelf ook gebruiken. De site heb ik het heel simpel gemaakt met een lange tabel zonder reclame en andere trackers, zodat u een schoon beeld hebt van de situatie in cryptoland. Verder gaak ik hieronder nog wat extra informatie over de meest gebruikte coins in de wereld vertellen.

Bitcoin (BTC)

De bitcoin was de eerste en de meest bekende cryptocurrency in de wereld. Als u iemand op straat vraagt of ze hebben gehoord van de bitcoin dan is de kans groot dat ze ja zeggen. De bitcoin is ontstaan nadat Satoshi Nakamoto een bestand had verspreid genaamd "Bitcoin: A Peer-to-Peer Electronic Cash System". En dit korte bestand stond zijn idee van een decentraal netwerk met een vast protocol genaamd een blockchain. In die blockchain kunnen dan transacties worden gezet die nooit meer kunnen veranderen. Zo loste hij het double spending probleem op. Dat probleem betekend dat u een munt niet twee kan uitgeven die kan niet door het proof-of-work systeem. In januari 2009 begon het netwerk toen Satoshi het genesis block mijnde. De bitcoin is op dit moment de meest gebruikte cryptocurrency van de wereld. Hij was in het verleden wel meer waard dan dit. De bitcoin wordt wel geplaagd door wat verouderde design keuzes. Zo is de block-rate om de tien minuten. Dat is redelijk traag vergeleken met andere coins. Ook zijn de transactiekosten veel hoger dan andere coins. En kost het gehele netwerk gigantisch veel stroom doordat het gebruik maakt van het proof-of-work systeem. Maar dat maakt voor de meeste mensen niet uit want deze coin is het meeste waard. En word nog steeds veel gebruikt in de criminaliteit.

Ethereum (ETH)

Ethereum is de tweede grootste currency op dit moment. Alleen ethereum is niet een standaard currency zoals wij die hebben gemaakt. Nee, ethereum is een netwerk met veel meer mogelijkheden. Want het ethereum maakt gebruik van een scripting virtual machine in de blockchain. Dit betekent dat u stukjes code kan toevoegen aan uw transactie en dat die dan in het netwerk worden uitgevoerd dit is een interessant concept maar maakt het netwerk wel veel ingewikkelder.

Ik zelf ben eerder voorstander van een minimalistische blockchain met een simpel protocol en andere applicaties die daar los van staan om er mee te communiceren. Maar het ethereum model biedt wel meer mogelijk heden zo kunt u eigenlijk alle bankzaken in het netwerk zetten. En ingewikkelde contracten zoals leningen, hypotheek en andere donaties naar goeden doelen. Deze transactie zet u dan met een stukje code in de blockchain en die wordt dan uitgevoerd. De currency die over dit netwerk wordt gehandeld heet ether.

Bitcoin Cash (BCH)

Bitcoin cash is een hard fork van het bitcoin netwerk. Een hard fork is een kopie van de blockchain die verder een ander leven draagt. Veel Chinese miners waren het oneens met bitcoin omdat de block grote (hoeveel transacties in een block gaan) niet zo groot was. Dus waren de transactie kosten hoger. Zij hebben dus collectief ervoor gekozen om te breken met bitcoin en hun eigen afsplitsing te maken. In het begin liep Bitcoin Cash niet zo goed. Maar sinds hebben veel miners weer vertrouwen in de afsplitsing en zit hij nu weer in de top 10.

Ripple (XRP)

Ripple is een andere cryptocurrency. Het is bijzonder want de coin wordt gemaakt door een bedrijf genaamd Ripple. Het netwerk kan voortbestaan zonder het bedrijf. Maar waarschijnlijk als Ripple omvalt valt de coin mee met het bedrijf. Maar Ripple is een erg stabiel bedrijf die was ontstaan doordat de stichters een manier wouden vinden waarmee men snel geld kon sturen over de wereld en met lage transactiekosten. Uiteindelijk zagen zij de kracht van de blockchain met bitcoin en hebben ze hun eigen versie ervan gemaakt. De source code van het netwerk is open source en staat ook op Github. Zoals u boven kon zien in de tabel is de coin veel minder waard als de andere maar er zijn wel erg veel in omgang. Ik vind ik mooi want als u geld overmaakt heeft u niet zo'n lang komma getal als bij bitcoin of monero.

Litecoin (LTC)

Litecoin is een andere coin gemaakt door Charlie Lee een voormalige Google werknemer. Hij begon de coin met verschillende stuken bitcoin code. Maar die zijn nu grotendeels herschreven. In november 2013 bereikte de coin een markt-kapitalisatie van 1 miljard dollar. Veel mensen sind die datum gingen Litecoin mijnen omdat verschillende media de suggestie hadden gewekt dat het een alternatief van bitcoin zou kunnen zijn sinds dien is de coin alleen maar verder gegroeid en een volwassen coin geworden. De blockchain werkt grotendeels

hetzelfde als de blockchain op een paar verschillen zo gebruikt het een andere hash functie en meer coins die gemijnd kunnen worden.

Tether (USDT)

Tether is een aparte coin met een erg interessante geschiedenis. De coin begon bij het bedrijf Tether Limited. Het is een gesloten coin want de coins worden vrijgegeven door het bedrijf zelf. En het is een stabiele coin want de makers beloofde dat 1 tether altijd 1 dollar waar zou zijn. Vandaar ook de USD in de beurs afkorting. Veel crypto handelaren vonden dit erg handig want ze konden nu hun coins snel omrekenen met tether en zo op hun account laten staan. Dit scheelde het omrekenen naar echte valuta waar kosten aan verbonden zitten. Dit werd gedaan door best veel brokers. En de coin was dus ook erg gewoeld. Alleen het was een bubbel die uiteindelijk klapte en de coin werd minder waard dan 1 dollar nu staat hij op zo'n 86 cent maar dat is nog steeds 14% minder als was beloofd. En veel handelaren hebben dus ook hun vertrouwen in deze coin opgegeven.

Monero (XMR)

Monero heeft een mooi plekje in mijn hart want de coin is gemaakt om 100% procent anoniem te zijn. Ik vind dat dit een doel moet zijn van elke coin. Bij de bitcoin bijvoorbeeld is het best makkelijk te tracken wie geld naar wie stuurt. Dit is vooral handig voor de overheid die criminaliteit tegen wilt gaan. Onlinesites zoals Silk Road deden al hun online transacties over bitcoin, in het volgende hoofdstuk ga ik hier nog wat dieper op in. Monero daarentegen is volledig anoniem en steeds meer mensen ook criminelen maken daar dan ook gebruik van. Ook maakt Monero gebruik van een andere hash algoritme die ervoor gemaakt is dat het beter werkt als u het draait op uw processor dan met een videokaart. Dit is voor een mooi gebaar want iedereen kan dus de coin mijnen en niet alleen als u speciale hardware daarvoor koopt. De Monero is ook veel gebruikt bij crypto mijning malware. Dit is als een virus jouw computer laat mijnen voor iemand anders. Er zijn ook scripts die u zo op uw site kan zetten die uw computer van uw bezoekers voor u laat mijnen. Ik zelf heb ook verschillende sites gemaakt waarop die scripts staan. Ga maar eens naar: bokogames.ml en bassiegames.ml. Laat de pagina open of speel een stom flash spelletje. Open uw

Taakbeheer, op Windows rechter muisklik op menubalk onderin en dan taakbeheer. Ga dan naar meer details en dan Prestaties. U ziet dat uw processor 100% aan het mijnen is sluit nu het tabblad af en u ziet het weer omlaag gaan. Veel mensen vinden dit een virus maar het is 100% veilig want de code draait gewoon in de website zelf. Er zijn ook echte virussen die deze code op de achtergrond van uw computer draaien ook als u uw browser sluit dit is wel een virus. Maar gelukkig draaien de meeste computers tegenwoordig Windows 10 en daar zit de virusscanner genaamd Windows Defender in en die scant voor dit soort schadelijk gebruik van uw computer.

Hoofdstuk 6

> Wat zijn andere toepassingen <

De blockchain is een hele goede en globale techniek die overal kan toe voor worden gepast. Het lost eigenlijk het probleem van vertrouwen op die nu erg heerst op het internet. Want hoe kan u bewijzen dat u bent wie u zegt dat u bent. In de vorm van accounts maar die kunnen gekraakt worden of in de vorm van een soort blockchain die decentraal is en waar iedereen mee aan kan werken.

Bij de overheid

Voorals in publieke instellingen en ruimtes zie ik daarvoor een enorme grote markt want de overheid moet voor iedereen open zijn en informatie delen met iedereen zoals elke democratie hoort te doen. Maar dan moeten er wel keuzes worden gemaakt we moeten kiezen of we het gaan omarmen of gaan verstoppen want de blockchain geeft de overheid ook minder macht. En u weet wat politici daarvan vinden ze willen hun macht behouden en ze haten vernieuwing daar komt nog bij dat ze meestal weinig van moderne technologieën afweten. Dus ja er moet een keuze komen.

In de zorg

Ook kan de zorg een goede optie hebben in het decentraal opslaan maar toch beveiligde vorm van patiëntendossier zo kan de integriteit gewaarborgd blijven maar kunnen patiënten ook thuis hun informatie in zien en bewerken. Eigenlijk is er voor alle technologieën wel een manier met een decentraal blockchain te bedenken want de technologie is zo wijd alles is mogelijk en in de toekomst zullen we nog wel meer dingen zien denk ik. Cryptovaluta kwamen eerst maar wat er morgen komt. Zo was het ook met het internet niemand kon bedenken dat Facebook of YouTube ooit mogelijk waren zo is het ook met dit.

In de criminaliteit

De cryptocurrencies worden vooral gebruikt voor goede doeleinden maar zoals alles goeds wordt het ook gebruikt in de criminaliteit. Het wordt zelfs zo veel gebruikt dat ik wel kunt zeggen dat zonder de criminaliteit de bitcoin nooit zo veel euro waard was geworden. Het wordt vaak gebruikt als betaalmiddel op zwarte markten op het dark web. Hiermee kunnen klanten anoniem dingen kopen en betalen aan andere anonieme aanbieders.

Een van die grootste site was Silk Road deze site is niet te vinden op het normale internet maar op het dark web. Dit is het verborgen internet wat u alleen kan bereiken via Tor. Dit is trouwers heel erg makkelijk u hoeft alleen maar de Tor browser te downloaden op: <https://www.torproject.org/download/download> dan moet u als u het hebt geïnstalleerd naar: <http://6khxwj7viwe5xjm.onion/> gaan en dan zit u op een node van DreamMarket dit is nu op het moment van schrijven (6-12-2018) een van de beroemdste markets op het internet. Weet wel dat markets komen en gaan op het dark web dus waarschijnlijk is de site wel down over een paar maanden. Dus als u gemakkelijk drugs wilt kopen of iets anders illegaals u vindt, vind het op die markets en u betaald met een bepaalde cryptovaluta meestal Monero tegenwoordig omdat het 100% privé is. Zoals u ziet is Tor Browser gewoon de laatste versie van Firefox maar dan zo getweakt dat het tor netwerk gebruikt wat ervoor zorgt dat u anoniem op het internet kan serveren.

Hoofdstuk 7

> Conclusie en de toekomst <

De conclusie

Ik heb jullie in deze PWS verteld hoe u uw eigen cryptocurrency kan maken en ook hoe u een beetje kan programmeren in JavaScript verder. Heb ik nog wat verteld over de geschiedenis, de gevaren van het investeren op dit moment, de verschillende coins die nu in omloop zijn en ook nog andere toepassingen. Alles wat gezegd kan worden tegen mensen met wat minder kennis over dit onderwerp heb ik wel bijgebracht.

Zoals jullie misschien wel hadden kunnen lezen ben ik best wel een voorstander want de blockchain technologie en de kracht die het programmeurs zoals ik geeft. Ook vind ik het idee van virtueel crypto grafisch geld erg cool en ook handig. Maar zoals het er nu uit ziet denk ik dat het niet erg snel aangaat slaan. Lees het volgende kopje de toekomst als u nog wat meer wilt lezen over de toekomst.

De toekomst

De toekomst is roze kleurig voor de cryptocurrencies denk ik. Want de technologie erachter kan voor alles worden gebruikt voor blockchain kan voor alles worden gebruikt.

Zoals u misschien kan denken geloof ik in de cryptovaluta. Het is de toekomst en hoe dan ook zal dit de wereld overnemen. Maar voor bedrijven zoals Visa en Mastercard is dit een ramp want ze verdienen miljarden met pintransacties. Want u denkt er misschien niet bij na maar elke keer als u pint krijgen die bedrijven wat geld van transactiekosten. En als de maatschappij cryptovaluta gaan gebruiken dan vallen die bedrijven weg en dat willen ze kosten wat kost verkomen.

Maar ik moet wel er nog bij vertellen dat mensen verandering haten. Het liefste willen ze allemaal dat alles nooit verandert en dus hetzelfde blijft. Dus ze zijn net na een paar jaar gewend aan het pinsysteem en het moet weer anders met cryptocurrencies. Veel mensen zullen het niet begrijpen en zich eraf van keren.

Ook heet cryptocurrencies zoals de Bitcoin al een heel erg slecht imago en mensen ze het op het eerste gezicht niet vertrouwen.

Mensen veranderen niet snel maar ze kunnen wel snel veranderen als er een ramp gebeurt. Daarom voorspel ik dat de cryptovaluta niet snel erg beroemd zal worden maar tot er een grote financiële ramp komt. Iets gelijks aan de huizen crisis van 2007 en 2008. Verandering moet u door de strot duwen. Als zoiets gebeurt van zullen de mensen zich snel afwenden van de gebruikelijk banken met hun pin zullen ze zich volledig opnemen in het cryptovaluta land. En de prijzen van die valuta zullen dan ook gigantisch stijgen.

Maar tot dat moment komt. Zullen ze wel een nish dingetje zijn die programmeurs zoals ik wel cool vinden maar dat verder niemand gebruikt. De cryptovaluta's ze zijn de toekomst maar te vroeg op aarde gekomen. Ooit wordt het alles nu is het eigenlijk niks.

BassieCoin

Bronnen

Sites die ik heb gebruikt

- https://en.wikipedia.org/wiki/Cryptographic_hash_function
- <https://en.wikipedia.org/wiki/Bitcoin>
- <https://en.wikipedia.org/wiki/Ethereum>
- https://en.wikipedia.org/wiki/Bitcoin_Cash
- [https://en.wikipedia.org/wiki/Tether_\(cryptocurrency\)](https://en.wikipedia.org/wiki/Tether_(cryptocurrency))
- https://en.wikipedia.org/wiki/Public-key_cryptography
- [https://en.wikipedia.org/wiki/Ripple_\(payment_protocol\)](https://en.wikipedia.org/wiki/Ripple_(payment_protocol))
- https://en.wikipedia.org/wiki/Dark_web
- [https://en.wikipedia.org/wiki/Tor_\(anonymity_network\)](https://en.wikipedia.org/wiki/Tor_(anonymity_network))
- [https://en.wikipedia.org/wiki/Monero_\(cryptocurrency\)](https://en.wikipedia.org/wiki/Monero_(cryptocurrency))
- <https://en.wikipedia.org/wiki/JavaScript>
- <https://en.wikipedia.org/wiki/Peer-to-peer>
- <https://www.nu.nl/internet/3641364/bitcoin-gevaren-en-alternatieven.html>

Boeken die ik heb gelezen

- <https://driestarcollege-zeta.auralibrary.nl/ajdetailsx.aspx?DOCSTART=E0069>

Logboek

Datum	Beschrijving
12-7-2018	Begin van het idee bedenken en de eerste stapjes
6-9-2018	Boek geleend en gelezen
7-10-2018	Hoofdstuk 4, 5 afgemaakt
21-11-2018	Verder gewerkt aan het programmeren van de BassieCoin
6-12-2018	Hoofdstuk 1 en 6 afgemaakt
7-12-2018	Hoofdstuk 2 groot en deels en 7 afgemaakt
10-12-2018	Dingen afronden in verschillende hoofdstukken
11-12-2018	Hoofdstuk 2 afgemaakt
12-12-2018	Hoofdstuk 3 afgemaakt