FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Teoría de la Programación

75.24

Solidity

1er cuat 2018

Nombre	**Padrón**
Romero, Alejandro	83361
Savulsky, Sebastián Alejandro	93081

Índice

Índice **1**

Orígen **2**

Utilidad del Lenguaje **3**

Características básicas del lenguaje **4**

Paradigma, Compilado/Interpretado 4

Tipado 4

Control de flujo de datos 4

Parámetros de entrada y de salida 4

Tipos de Datos Abstractos (TDAs) 5

Booleanos: 5

Enteros 5

Dirección 5

Miembros de dirección 6

Números de punto fijo 6

Bytes-Arrays de tamaño fijo 6

Bytes-Arrays de tamaño variable 6

Literales de dirección 6

Enums 7

Tipos de funciones 7

Características avanzadas del lenguaje: **9**

Errores 9

Concurrencia 9

Manejo de Memoria 9

Ejemplos **10**

Estadísticas **11**

Estadísticas - CNBC **12**

Comparación con lenguajes similares **14**

Casos de estudio **14**

ICOs - Initial Coin Offering 14

CryptoKitties 15

EtherDelta 15

Orígen

Antes de hablar de Solidity, resulta necesario hacer un breve comentario sobre bitcoin y las blockchains en general.
Blockchain (cadena de bloques) es una tecnología revolucionaria sobre la cual se basan todas las criptomonedas existentes.
La misma permite a la vez: resistencia a la censura, descentralización, ausencia de confianza en terceros, globalidad y neutralidad.
La explicación de la misma excede el fin de este trabajo pero basta con saber que uno de sus fines más populares es el de transferencia de monedas entre individuos sin intermediarios.
Bitcoin surgió hace más de 10 años y fue la primera en utilizar el concepto de blockchain.
Por el año 2014, uno de sus desarrolladores, Vitalik Buterin, propuso una mejora a bitcoin que no fue aceptada. La misma consistía en la creación de lo que hoy en día se conoce como smart contracts. Una forma de ejecutar código dentro de la blockchain.

Esta negativa lo llevó a crear su propia moneda, conocida como Ethereum.
Ethereum es una plataforma open source, descentralizada que permite la creación de acuerdos de contratos inteligentes (smart contracts) entre pares, basada en el modelo blockchain.
Cualquier desarrollador puede crear y publicar aplicaciones distribuidas que realicen contratos inteligentes.

Con este escenario es que surge Solidity. Un lenguaje orientado a Contratos (smart contracts), de alto nivel, cuya sintaxis es similar a la de JavaScript y está diseñada para orientarse a la Máquina Virtual Ethereum (EVM).

Utilidad del Lenguaje

Cómo fue mencionado anteriormente. La utilidad principal del lenguaje es la creación de Smart Contracts que son ejecutados bajo la criptomoneda ethereum.
Un contrato inteligente es un programa informático que ejecuta acuerdos establecidos entre dos o más partes haciendo que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas.
La aparición de los contratos inteligentes se debe a la posibilidad que brinda blockchain de permitir que las personas, por sí mismas, hagan cumplir los contratos en el mundo real sin necesidad de un intermediario físico, es decir, sin necesidad de un juez o un árbitro.

La utilización de Solidity y la creación de smart contracts en sí, resulta buena para todo programa que requiera generar algo inmutable, seguro y con una base de datos descentralizada transparente.
Más adelante veremos ejemplos de utilización de solidity para crear smart contracts. Entre ellos está la creación de ICOs (Initial Coin Offering para juntar dinero), o la creación de mercados de intercambio de monedas sin intermediarios.

Solidity permite escribir información en la cadena de bloques ethereum. Esto lo vuelve susceptible a ataques que pueden saturar la red.
Para solucionar este problema, se impuso la necesidad de que cada operación computacional creada con solidity, tenga un costo monetario (denominado “Gas”, una subdivisión de “Ether” que es la moneda de ethereum).
Esto lo vuelve muy costoso si alguien quisiera abusar de la red creando bucles infinitos.
Es por este motivo que no es recomendable utilizar el lenguaje para crear programas que requieren mucho gasto computacional, ya que la ejecución del mismo se volvería muy costosa.

Características básicas del lenguaje

Paradigma, Compilado/Interpretado

Solidity es un lenguaje similar en sintaxis a javascript, con soporte de creación de objetos. Difiere ampliamente del resto de los lenguajes en que está pensado para ser ejecutado sólo dentro de la Ethereum Virtual Machine.
Su código es guardado y ejecutado dentro de la cadena de bloques ethereum. Por las características de una blockchain, la misma es guardada en varios nodos (esto es lo que vuelve a una cadena de bloques descentralizada). Es por este motivo que un programa creado en solidity es ejecutado en cada uno de los nodos. A diferencia de otros lenguajes donde el programa puede correr solo una vez en un servidor o localmente.

Por tener esta particularidad, para desarrollar en solidity debemos instalarnos una red privada de ethereum local o utilizar la pública de testing.
Existen varios entornos de desarrollo para Solidity, pero el oficial se conoce como Remix.
https://remix.ethereum.org

Solidity es Turing Complete. La Ethereum Virtual Machine (EVM) es considerada una Máquina de Turing Universal, término que se refiere al software que es lo suficientemente hábil como para ejecutar cualquier código definido por el desarrollador.

Tipado

Solidity al contrario que javascript, es fuertemente tipado, esto quiere decir que si una variable se usa para almacenar un tipo de datos, dicha variable no podrá usarse para almacenar un tipo de datos diferente al primeramente almacenado.

Control de flujo de datos

Solidity acepta todas las mismas estructuras de control de javascript, a excepción de switch y goto.
Por lo que tendremos a nuestra disposición:`if`, `else`, `while`,`do`, `for`, `break`, `continue`, `return`,`? :`

Parámetros de entrada y de salida

Al igual que en Javascript, las funciones obtienen parámetros como entrada.
En sus parámetros de salida es donde existe una diferencia con javascript. En solidity es posible devolver un número arbitrario de parámetros de salida.

```
ej)
function arithmetics(uint _a, uint _b)
        public
        pure
        returns (uint o_sum, uint o_product)
    {
        o_sum = _a + _b;
        o_product = _a * _b;
    }
```

Tipos de Datos Abstractos (TDAs) [1]

Booleanos:

`bool`: The possible values are constants `true` and `false`.

Operadores:

- `!` (logical negation)
- `&&` (logical conjunction, “and”)
- `||` (logical disjunction, “or”)
- `==` (equality)
- `!=` (inequality)

Enteros

`int` / `uint` (`uint8` to `uint256`)

Dirección

`address`: Contiene valor de 20 bytes (el tamaño de una dirección de Ethereum)

[1] http://solidity.readthedocs.io/en/v0.4.24/types.html

Miembros de dirección

- `balance` and `transfer`
- `Send`
- `call`, `callcode` and `delegatecall`

Números de punto fijo

`fixed` / `ufixed`
`ufixedMxN` and `fixedMxN`
`M representa el número de bits tomados por el tipo y N representa cuántos puntos decimales están disponibles`

Bytes-Arrays de tamaño fijo

`bytes1`, `bytes2`, `bytes3`, …, `bytes32`. `byte` es un alias para `bytes1`.

Bytes-Arrays de tamaño variable

`bytes`:
Dynamically-sized byte array, No es un tipo!

`string`:
Dynamically-sized UTF-8-encoded string, No es un tipo!

Literales de dirección

`0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF` son de tipo `address`
Los literales hexadecimales que tienen entre 39 y 41 dígitos de longitud y no superan el checksum producen una advertencia y se tratan como literales de números racionales regulares.

Rational and Integer Literals
Enteros de 0 a 9, fraccionarios con punto del estilo `1.`, `.1` y `1.3`.
Notación científica es soportada también; ej: `2e10`, `-2e10`, `2e-10`, `2.5e1`.
String Literals
Se pueden declarar con comillas simples y dobles; ej: `"foo"` or `'bar'`

Hexadecimal Literals
Se usa la palabra reservada hex para su uso, por ej: `hex"001122FF"`

Enums

Ej:

```
enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }
    ActionChoices choice;
    ActionChoices constant defaultChoice = ActionChoices.GoStraight;
```

Tipos de funciones

Las funciones llevan las siguientes variables:

`function (<parameter types>) {internal|external} [pure|constant|view|payable] [returns (<return types>)]`

Tipos referenciales

Todos los tipos complejos poseen el atributo "data location" que puede tomar los valores:

(`storage` (donde se almacena state) / `memory` (no persistente) / calldata
Ubicación de datos forzados:

- parámetros (no retorno) de funciones externas: calldata
- variables de estado: almacenamiento

Ubicación de datos predeterminada:

- parámetros (también retorno) de funciones: memoria
- todas las demás variables locales: almacenamiento

Arrays

(`length` y `push`)

Alocación de memoria:

```
Ej.  uint[] memory a = new uint[](7);
  bytes memory b = new bytes(len);
```

Literales de Arrays/ Arrays Inline

```
g([uint(1), 2, 3]);
```

Structs; Ej:

```
Ej.    struct Funder {
        address addr;
        uint amount;
    }
```

Mappings

`mapping(_KeyType => _ValueType)`

Operadores con LValues

`-=`, `*=`, `/=`, `%=`, `|=`, `&=` and `^=`
Delete

`delete a` rese comporta como una asignación a `a`, para enteros esto es equivalente a `a = 0.`

Conversiones entre tipos elementales:

Implìcitas

`uint8` es convertible a `uint16` y `int128` a `int256`, pero `int8` no es convertible a `uint256`

Explícitas; Ej:

```
int8 y = -3;
uint x = uint(y);
```

Si se convierte a un tipo menor, se cortan los bits de mayor orden:

```
uint32 a = 0x12345678;
uint16 b = uint16(a); // b va a será 0x5678
```

Deducción de tipos; Ej:

```
uint24 x = 0x123;
var y = x;
```

Características avanzadas del lenguaje:

Errores

Solidity utiliza excepciones que revertirán el estado para manejar errores.
Las excepciones desharán todos los cambios realizados al estado en la llamada actual (y todas sus sub-llamadas).

`assert(bool condition)`:
Se arroja si la condición no se cumple - para ser utilizado para errores internos.

`require(bool condition)`:
Se arroja si la condición no se cumple - para ser utilizado para errores en entradas o componentes externos.

`revert()`:
cancelar la ejecución y revertir los cambios de estado

Concurrencia

La EVM (Ethereum Virtual Machine) no soporta concurrencia.
Por las características de la cadena de bloques, el código de los smart contracts corre en cada uno de los nodos existentes.
Esto resulta bueno desde una perspectiva de tener un sistema distribuido descentralizado, pero imposibilita la existencia de concurrencia.

Manejo de Memoria

Como el código de un smart contract existe por siempre dentro de ethreum, el uso de la memoria resulta sumamente costoso.
Es por este motivo que se creó el concepto de GAS. Una medida monetaria (en ethers) que es utilizada cuando se genera gasto computacional.

El uso de memoria temporal, resulta mucho menos costoso que el de almacenamiento.

Ejemplos

Desde una perspectiva de empresas que usan Ethereum:

- Contratos (b2b)

Compañías de contratos b2b:[2]
Bacancy Technology
Izettle
Elliptic
Clearmatics
Coinfabrik
10clouds
Applicature
Sodio

- Permisos de activos
- Acuerdos de accionistas
- Mercados de predicción
- Sistemas de votación
- Registros de dominio

Para la financiación entre pares:

- Crowdfunding
- Derivados
- Cobertura
- Seguro

Dapps centrados en el consumidor:

- Fideicomisos
- Tienda de activos personales
- Propiedad inteligente
- Intercambios financieros
- Guardando cuentas
- Testamentos
- Propiedad intelectual

Empresas que directamente se están construyendo sobre Ethereum:

- IoT (Internet of Things) que implementa un sistema de bloqueo generalista descentralizado.
- FMS (sistema de fabricación flexible) que ofrece una gran automatización de fábrica para pequeños comercios.
- Sistema de reputación back-end basado en etiquetas para contenido en línea.
- También IBM / Samsung (Internet of Things) es una bifurcación de una versión temprana de Ethereum Proof of Concept.

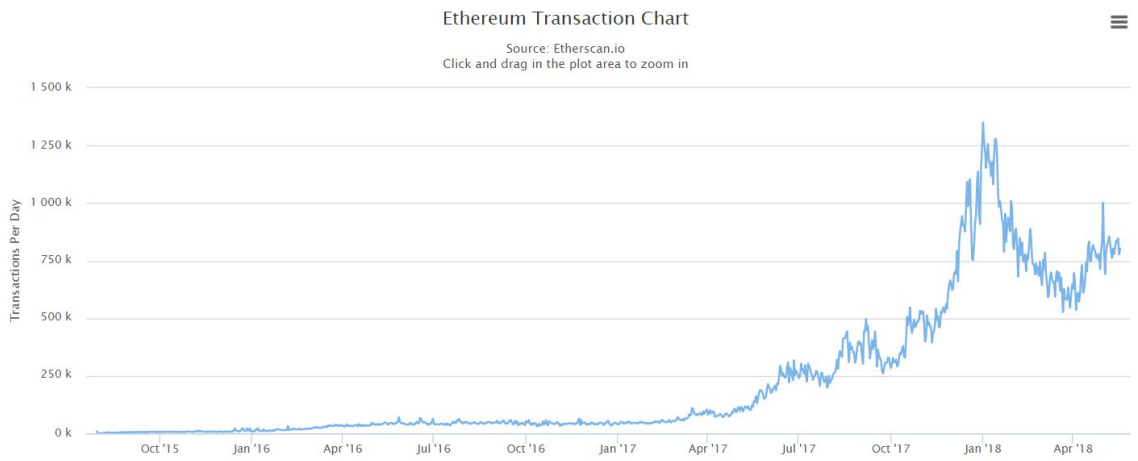
[2] Fuente: https://hackernoon.com/top-ethereum-smart-contract-development-company-8cfdd906cee

Estadísticas

A pesar de ser un lenguaje nuevo, la forma de programar smart contracts ya fue duplicada en otras monedas como NEO o Stratis.

Cantidad de Smart Contracts
Cantidad de smart contracts creados: 27114
Aquí se puede apreciar la cantidad de veces que fue utilizado Solidity en aplicaciones que ya se encuentran en producción.
https://etherscan.io/contractsVerified

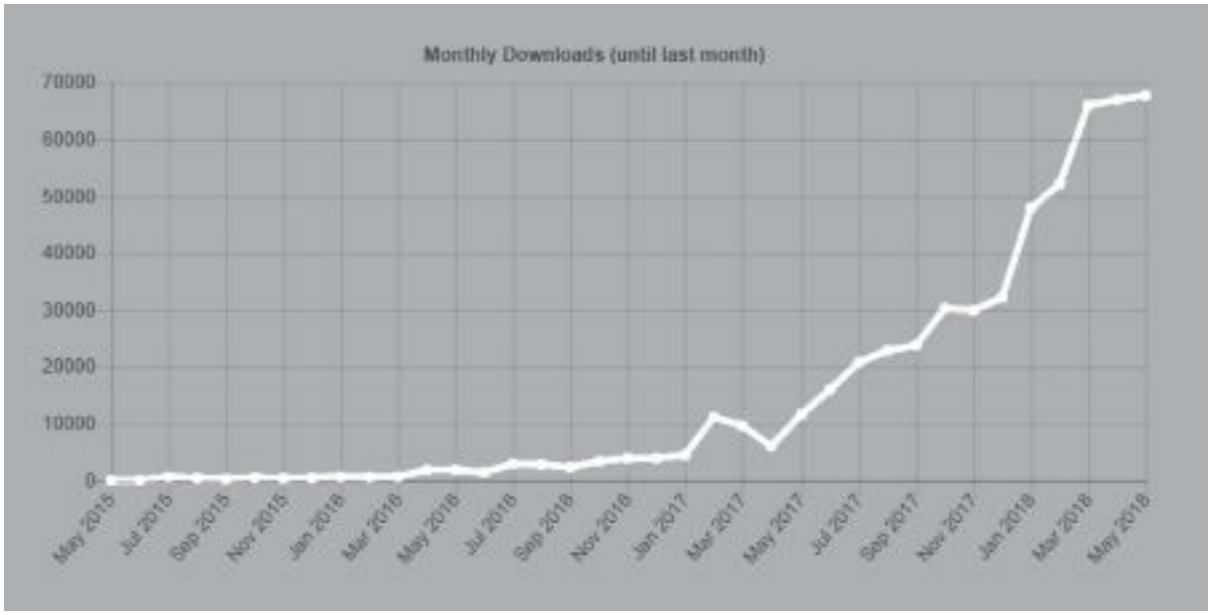
Ethereum transaction chart
Con este gráfico se puede ver la cantidad de interacciones con la cadena de bloques ethereum desde su creación.
A mayor número, mayor es el interés.



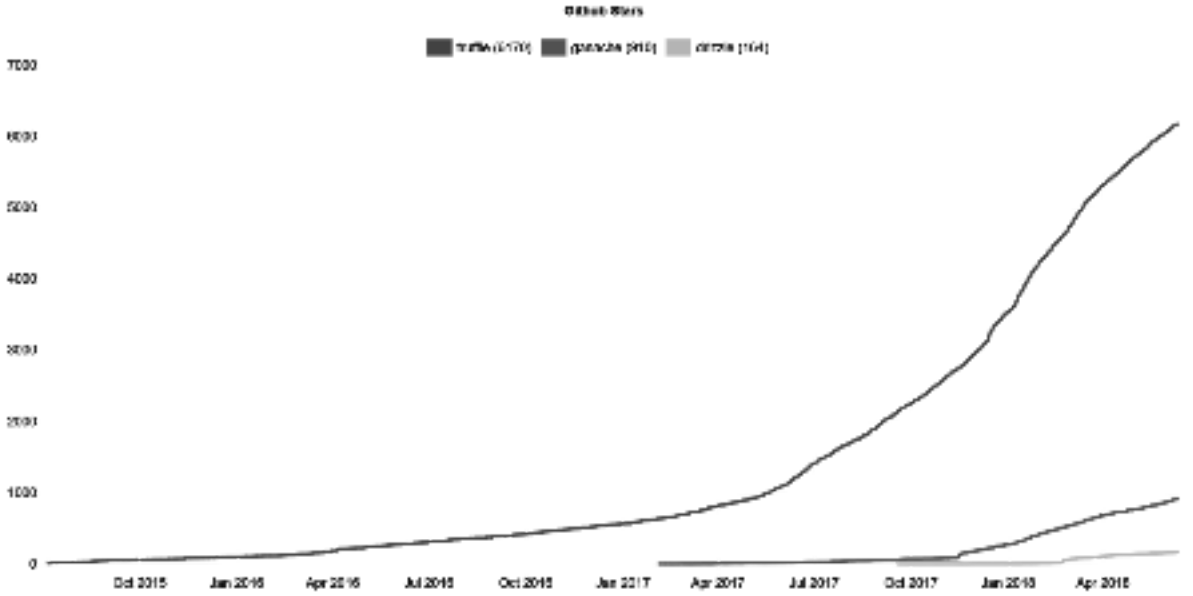
Estadísticas - CNBC

Según datos de Octubre de 2017 de la cadena de noticias estadounidense CNBC, Cerca de 35,000 desarrolladores y más de 500 start-ups desarrollan en esta plataforma.
Hay actualmente 70.000 Descargas mensuales del framework Truffle para desarrollo Ethereum.

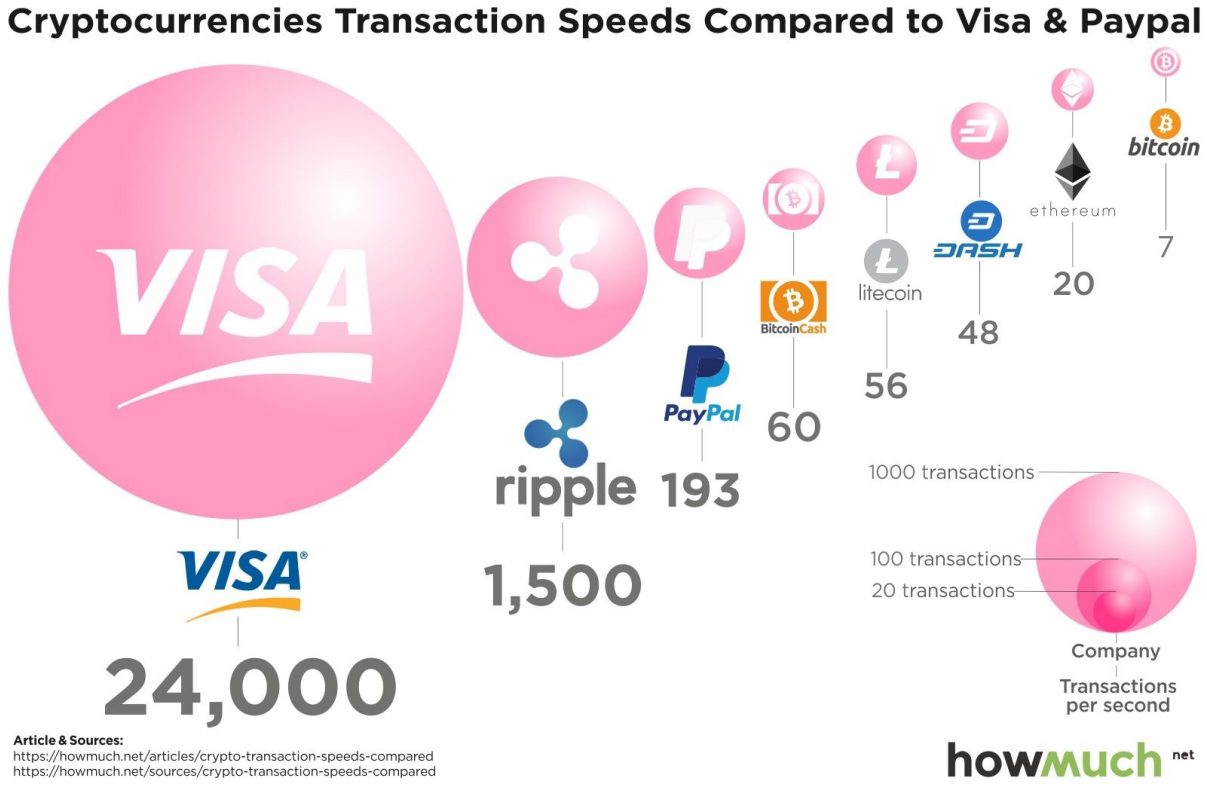
Estadísticas - Truffle (framework) - Descargas mensuales



Estadísticas - Github Stars - Truffle,Ganache,Drizzle



Transacciones/segundo vs Visa y PayPal



Comparación con lenguajes similares

En la actualidad existen varias criptomonedas que pretenden seguir el camino trazado por ethereum.
Entre ellas se encuentra Stratis o NEO. Dos monedas que permiten, al igual que ethereum, la creación de smart contracts.
El lenguaje elegido en ambas es C#.
La visión de ambas es la de eliminar la dificultad de aprender un idioma nuevo como lo es Solidity.
Nosotros consideramos que la complicación de aprender un lenguaje nuevo como Solidity, es despreciable en comparación a lo que se gana al tener un lenguaje creado exclusivamente con este fin. Con la seguridad y la conveniencia en crear smart contracts puestas como prioridad.

Casos de estudio

ICOs - Initial Coin Offering

ICO es un acrónimo que significa Initial Coin Offering, es decir, oferta inicial de moneda. El acrónimo ICO se parece bastante al de IPO, Initial Public Offering (que en castellano se denomina OPV, oferta pública de venta) término que se utiliza cuando una empresa sale a bolsa y quiere ofrecer las acciones a los posibles inversores a cambio de dinero.
En el caso de una ICO lo que se pretende financiar es el nacimiento una nueva criptomoneda, al estilo de Bitcoin o Ethereum. Se trata de *tokens* virtuales escasos, protegidos por criptografía, que tienen un valor debido a su escasez y a su demanda.
La forma de crear estos tokens es mediante un smart contract programado en Solidity

OpenZeppelin es un repositorio github que tiene código probado y establecido como seguro por la comunidad para crear ICOs
https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts

CryptoKitties

CryptoKitties fue uno de los primeros juegos desarrollados en solidity para correr bajo la blockchain ethereum.
El fin del mismo es comprar, vender y criar gatitos virtuales únicos. Su ADN se basa en la dirección de un token creado con este fin.

EtherDelta[3]

EtherDelta es un mercado de intercambio de tokens. Su código base es un smart contract creado en solidity que tiene todas las operaciones básicas de transferencia de dinero (en ethers), e intercambio de tokens entre comprador y vendedor.
Vale aclarar que EtherDelta y el resto de los desarrollos tiene su contraparte web. Un sitio desarrollado en cualquier lenguaje que puede interactuar con los contratos creados en solidity mediante el uso de una libreria javascripts llamada web3.js

[3] https://github.com/etherdelta