Solidity notları Part 1;

Contractlar : Bir sözleşme contract sözcüğüyle başlar ve onu tanımlar:

Pragma: Pragma, kullandığımız solidity dilinin hangi compiler veriyonunu kullandığını belitir.

Örnek Kod:

pragma solidity >=0.5.0 <0.6.0;

contract ZombieFactory {

//start here

}

State değerler ve Integerlar;

State değerler contractın hafızasında kalıcı olarak tutulan değerlerdir.

Unsigned Integer: genellikle uint kullanılan tek yönlü integerlardır. 0 dan küçük değerler almaları imkansızdır. Uint ve uint256 aynı şeyi temsil eder ikisi de 256 bitliktir.

Örnek Kod:

uint dnaDigits = 16;

Math Operasyonları:

{}

Structlar : Kompleks data tiplerini yaratmak için kullanılan yapıdır. Çoklu featurelar ekleyebiliriz.

Örnek Kod:

struct Zombie {

string name;

uint dna;

}

Arrayler : Solidityde 2 tip array vardır. Fixed ve Dynamic arrayler:

Fixed Array : Büyüklüğünü önceden belirlediğimiz arraylerdir.

Dynamic Array : Arraye veri eklendikçe büyüyen bir yapısı olan bir şeydir.

(Structlardan oluşan bir array yapmak istersek yazılacak kod : StructIsmı [] arrayIsmı ;)

Örnek Kod:

// Array with a fixed length of 2 elements:

uint[2] fixedArray;

// another fixed Array, can contain 5 strings:

string[5] stringArray;

// a dynamic Array - has no fixed size, can keep growing:

uint[] dynamicArray;

Fonksiyon Deklarasyonları : function fonksiyonIsmi(//variables) şeklinde bir yazılışı vardır.

Fonksiyon değerlerini girerken referans tiplerini(arrayler,structlar,mappingler,stringler gibi) memory’e declare ederiz.

Global değerlerden ayırmak için genelde fonksiyon içi değerlere underscore(\_) ile başlarız.

Örnek Kod:

function createZombie (string memory \_name, uint \_dna) public {

// start here

}

Arraylere eleman ekleme ve elemanı structırdan yaratma:

Örnek kodla hemen açıklayalım : Person Satoshi = Person (172,”Satoshi”);

Struct arraye ekleme : people.push(Satoshi);

Push methodu arrayin sonuna eleman ekleme yapar.

Private / Public fonksiyonlar:

Public fonksiyonlarda herhangi birisi ya da herhangi bir kontrat o fonksiyonu çağırabilir. Bu tür fonksiyonlar saldırıya açıktırlar.

Private olunca sadece o kontratlardaki fonksiyonlar onlara ulaşabilirler.

Private fonksiyonlar genellikler başında underscore(\_) taşırlar.

Return fonksiyonlar : Decalaresi şöyle

function \_generateRandomDna(string memory \_str) private view returns (uint) {

// start here

}

(Burada aynı zamanda view kullanımı da var)

View fonksiyonlar read\_only fonksiyonlardır. Call ettikten sonra state değerleri değiştiremezsiniz. State variable değiştiren, event emit eden, başka kontratlar yaratan, kendini imha(selfdestruct) method olan, vall ile ether transfer eden, view ya da pure çağırmayan fonksiyonları call eden, low level call kullanan view fonksiyonlar compiler hatası gönderir.

Pure fonksiyonlar : read ya da modify yapamaz. Parametre verilen ya da local olan değerleri kullanarak sadece döndürebilir. State variable readleyen, address eve ya balance’a ulaşan, herhangi bir global degree block ya da msgye ulaşan, pure olmayan fonksiyonları çağırmada bu fonksiyonlar compiler hatası verir.

Eventler: Front-end ile iletişime geçmeyi sağlayan bir fonksiyondur.

event IntegersAdded(uint x, uint y, uint result); // event oluşturulur.

function add(uint \_x, uint \_y) public returns (uint) {

uint result = \_x + \_y;

// bu fonksiyonun kullanıldığını göstermek için emitlenir.

emit IntegersAdded(\_x, \_y, result);

return result;

}

Js de şöyle bir fonksiyonla etkileşir:

YourContract.IntegersAdded(function(error, result) {

// do something with result

})

Ek not: push methodu arrayin uzunluğunu geri döndürür.