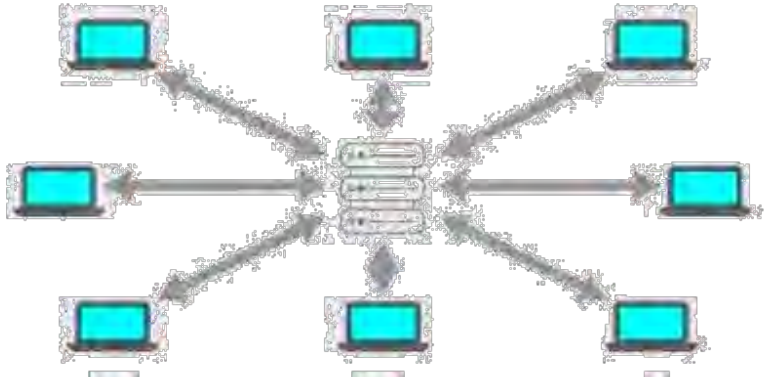
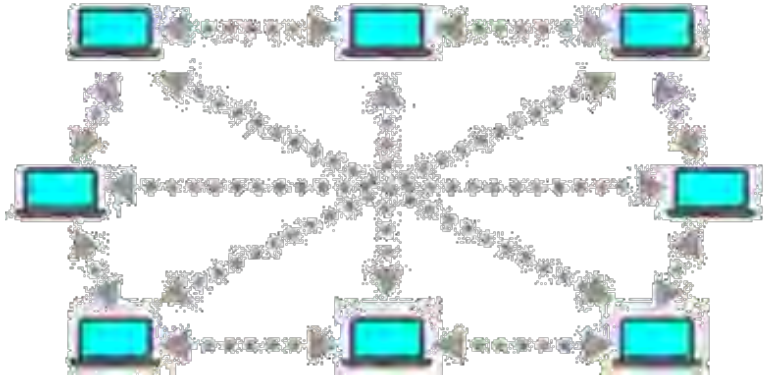
Децентрализованные приложения и смарт-контракты

Web3 и децентрализованные приложения

- **Web 1.0** – технологии, использовавшиеся в сети Интернет до начала 2000-х годов. Сеансовый доступ (обмен электронной почтой, файлами, размещение и получение информации в виде простых веб-страниц, форумов, объявлений на электронных досках).
- **Web 2.0** – господство технологии клиент-сервер, появление рынка обработки и анализа цифровых данных и компаний-бигтехов, обладающих большой долей этого рынка (напр., в 2023 году Google контролировала порядка 91,6% мирового рынка поисковых систем).
- **Web3** – децентрализованная обработка информации на базе блокчейн-технологий и распределённых приложений.



Web 2.0



Web 3.0

Децентрализованные приложения

Децентрализованное приложение (DApp) – приложение, исполнение и хранение информации в котором не привязано к конкретным вычислительным мощностям, а производится в рамках некоторой одноранговой сети, независимо от её конфигурации.

Top-10 блокчейн-платформ по числу DApp

Платформа	Базовая криптовалюта	Кол-во DApp, шт.	Кол-во аккаунтов в DApp, млн шт.
BNB Chain	BNB	5 261	1,95
Ethereum	ETH	4 524	0,58
Polygon	POL	2 042	1,82
TRON	TRX	1 378	0,006
EOS	EOS	583	0,076
Avalanche	AVAX	559	0,116
Arbitrum	ARB	456	0,811
Fantom	FTM	412	0,061
WAX	WAX	286	0,32
Solana	SOL	241	2,13

Децентрализованные приложения

Свойства:

- **открытый исходный код**. Так как приложение выполняется на компьютерах всех участников сети, для обеспечения доверия к нему любая заинтересованная сторона должна иметь возможность убедиться в отсутствии вредоносного кода и скрытых функций, позволяющих какой-либо группе лиц использовать приложение в своих интересах;

- **функционирование на основе консенсуса**. Решения о развитии приложения и изменении его функциональности принимаются сообществом на основе консенсуса. Ни одна заинтересованная сторона, включая разработчиков, не имеет единоличного контроля над приложением;

- **криптографическая защита**. Для обеспечения безопасности данных пользователей информация децентрализованного приложения защищена криптографическими механизмами и хранится в блокчейне, поддерживаемом участниками сообщества;

- **токенизация доступа**. Доступ к функциям децентрализованного приложения можно получить с помощью токена. Приложения могут поддерживать базовую криптовалюту либо генерировать собственные токены.

Области применения

Децентрализованные финансы (DeFi) – совершение финансовых операций без посредников (выдача займов, децентрализованные биржи).

Обеспечивают контроль над своими средствами и характеризуются значительно более слабым регулированием, нежели в сфере традиционных финансов.

Децентрализованные автономные организации (DAO) – форма управления организациями, при которой принятие решений участниками и координация их действий осуществляется автоматически на основе смарт-контрактов, без какого-либо центрального органа управления.

Компьютерные игры – в отличие от традиционных компьютерных игр, предоставляют игрокам полный контроль над внутриигровыми активами и позволяют монетизировать их в реальной жизни.

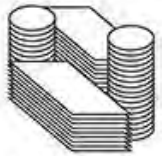
Развлечения и творчество – позволяют участникам взаимодействовать непосредственно друг с другом, не прибегая к услугам посредников (стриминговая платформа Audius, социальная сеть Steemit и т.д.)

Смарт-контракт

Смарт-контракт – компьютерный протокол транзакций, предназначенный для автоматизированного исполнения условий контрактов (Ник Сабо, 1994 г.)

Ник Сабо

Смарт-контракт vs. обычный контракт

СМАРТ-КОНТРАКТЫ	ОБЫЧНЫЕ КОНТРАКТЫ
 Виртуальный документ	 Бумажная версия документов
 Хранится в блокчейне	 Основана на праве
 Компьютерный язык	 Юридический язык
 Безопасно и без посредников	 Посредники, риск обмана
 Криптовалюты	 Обычные деньги

Смарт-контракт: пример

1. Создаем независимое хранилище, куда каждый может вносить данные, финансы, но не может удалить или забрать.
2. Антон кладет в это хранилище деньги за аренду квартиры.
3. Даша кладет туда код от двери своей квартиры.
4. Антону высылается код, Даше – подтверждение аренды на выбранные даты.
5. Если Антон приезжает и вводит код, то Даше перечисляется сумма.
6. Если код не подходит, то Антону возвращается сумма и контракт аннулируется.
7. Если Антон не приезжает, то Даше перечисляется сумма неустойки, а Антону возвращается остаток.
8. По окончании срока аренды контракт считается исполненным.

Реализация смарт-контрактов в Ethereum

Пользователи (кошельки)

- Управляются приватными ключами
- Инициируют транзакции
- Обладают балансом
- Управляются владельцем кошелька

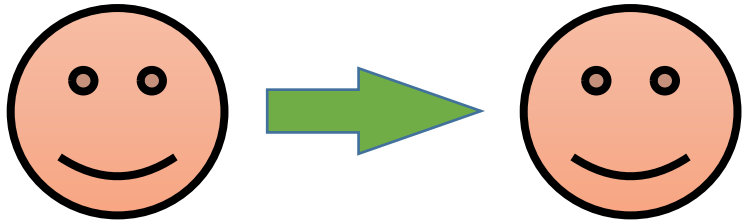
Контракты

- Управляются собственным кодом
- Реагируют на события выполнением функций
- Обладают балансом
- Имеют хранилище данных

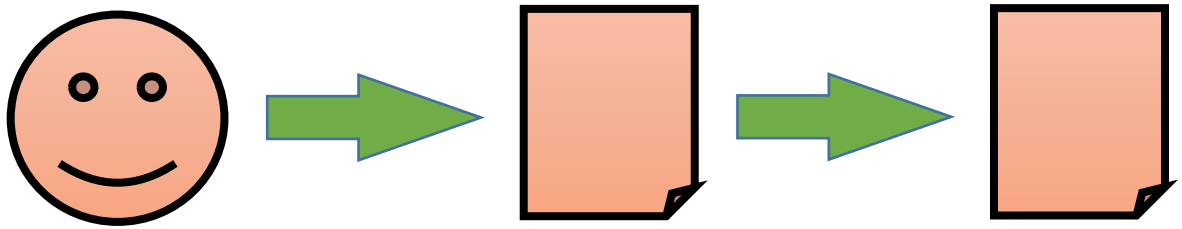
Хранилище смарт-контракта

Адрес контракта

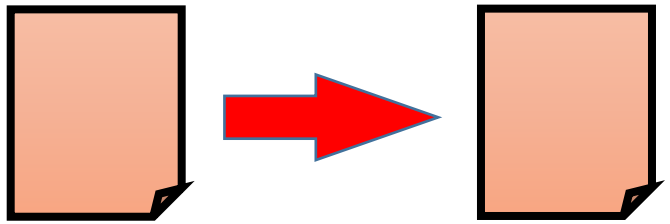
Смарт-контракт vs. пользователь



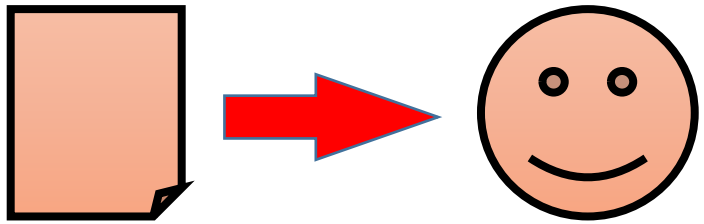
Обычный перевод



Пользователь вызвал функцию, она вызвала функцию другого контракта



Контракт не может самостоятельно инициировать вызов функций



Контракт не может самостоятельно переводить средства пользователям

Solidity

Solidity – высокоуровневый,контрактно-ориентированный, компилируемый, **полный по Тьюрингу** язык программирования.

Используется **исключительно** для разработки смарт-контрактов для платформы **Ethereum**.

Основан на языках **JavaScript, C++.**

Существуют **сторонние библиотеки** функций, например OpenZeppelin, Truffle.

Отличается **частыми релизами** версии компилятора, высоким развитием оптимального малоресурсного функционала.

Ограничен в типах данных и методах, в том числе **не поддерживает** операции с **вещественными** числами.

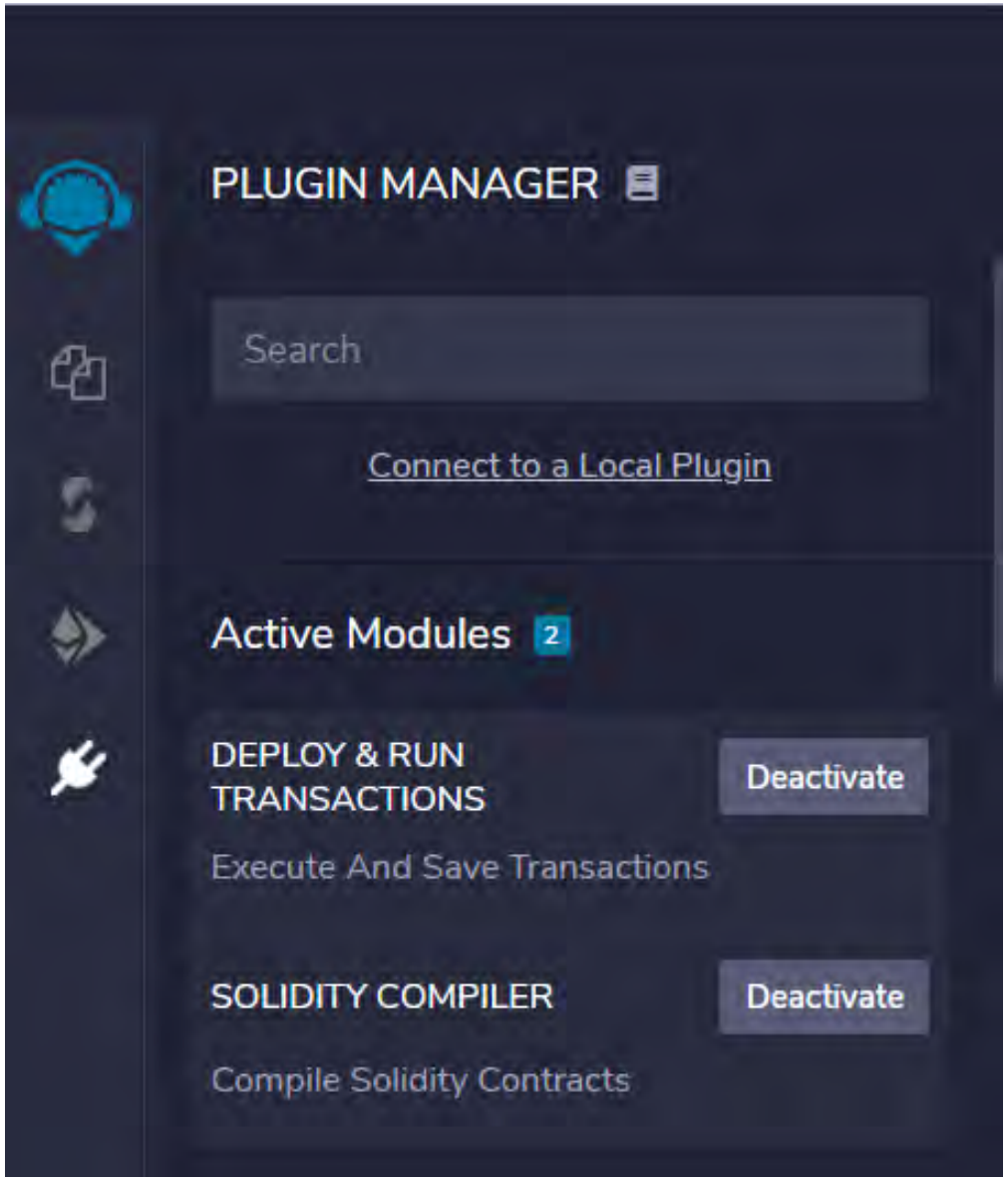
RemixIDE

RemixIDE – среда разработки, отладки и тестирования смарт-контрактов на языке Solidity.

Основные модули, используемые при работе:

- Solidity Compiller – компиляция смарт-контракта
- Deploy&Run Transactions – публикация контракта в сети Ethereum и проведение транзакций

https://remix.ethereum.org/

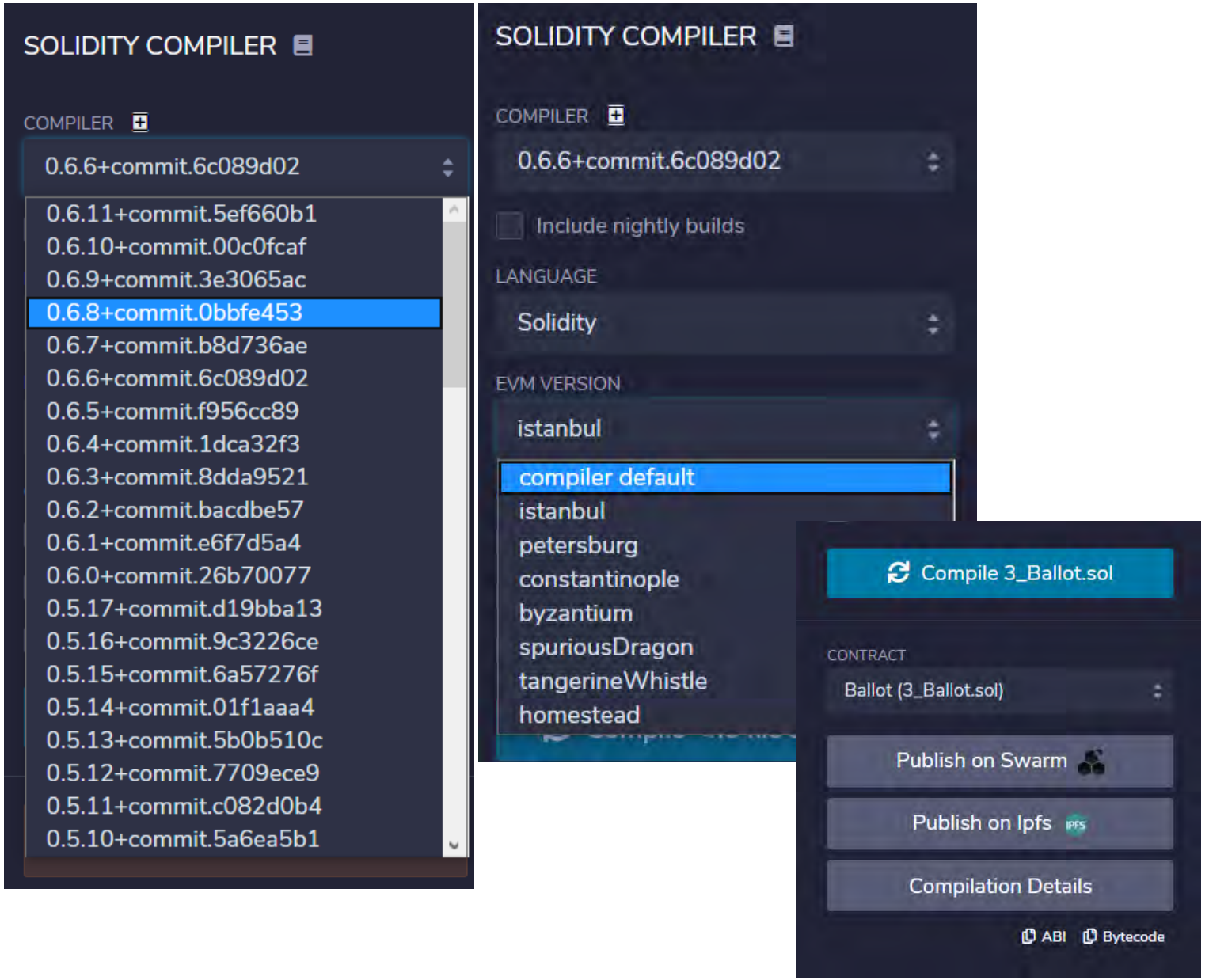


RemixIDE

SolidityCompiller – плагин предоставляющий выбор версии языка, выбор хардфорка.

Также после успешной компиляции контракта на вкладке будут доступны:

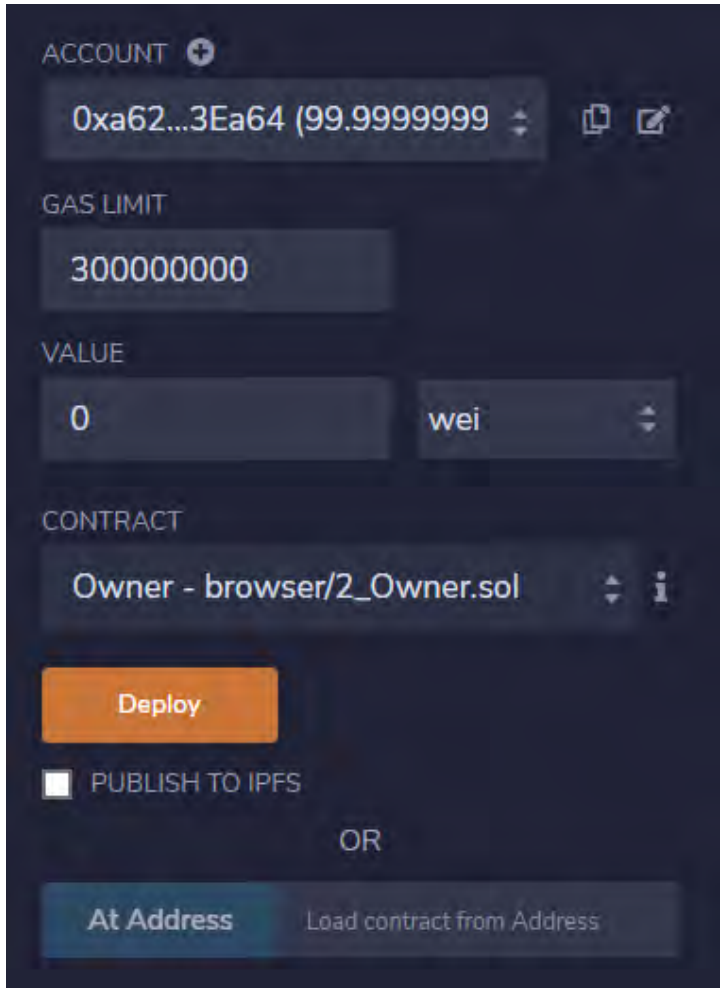
- ABI – представление контракта в формате json с описанием входных и выходных параметров каждой функции
- Bytecode – те же данные в бинарном виде
- прочие данные компиляции

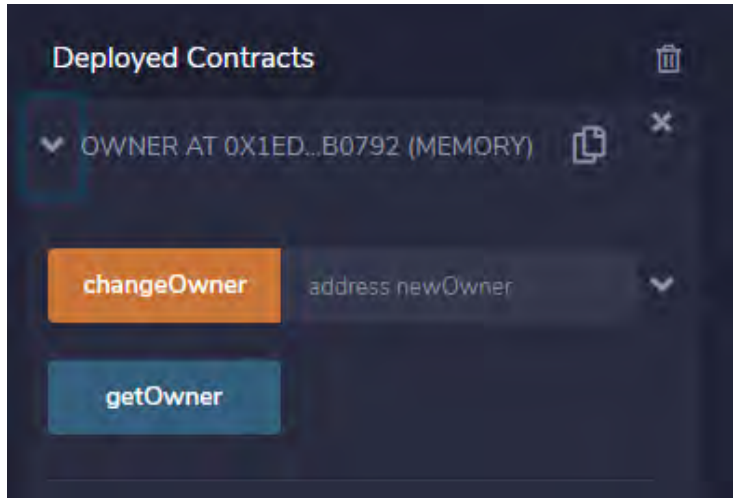


RemixIDE

Deploy&Run Transactions – плагин позволяет опубликовать контракт в сети и отладить его. Для разработчика доступны:

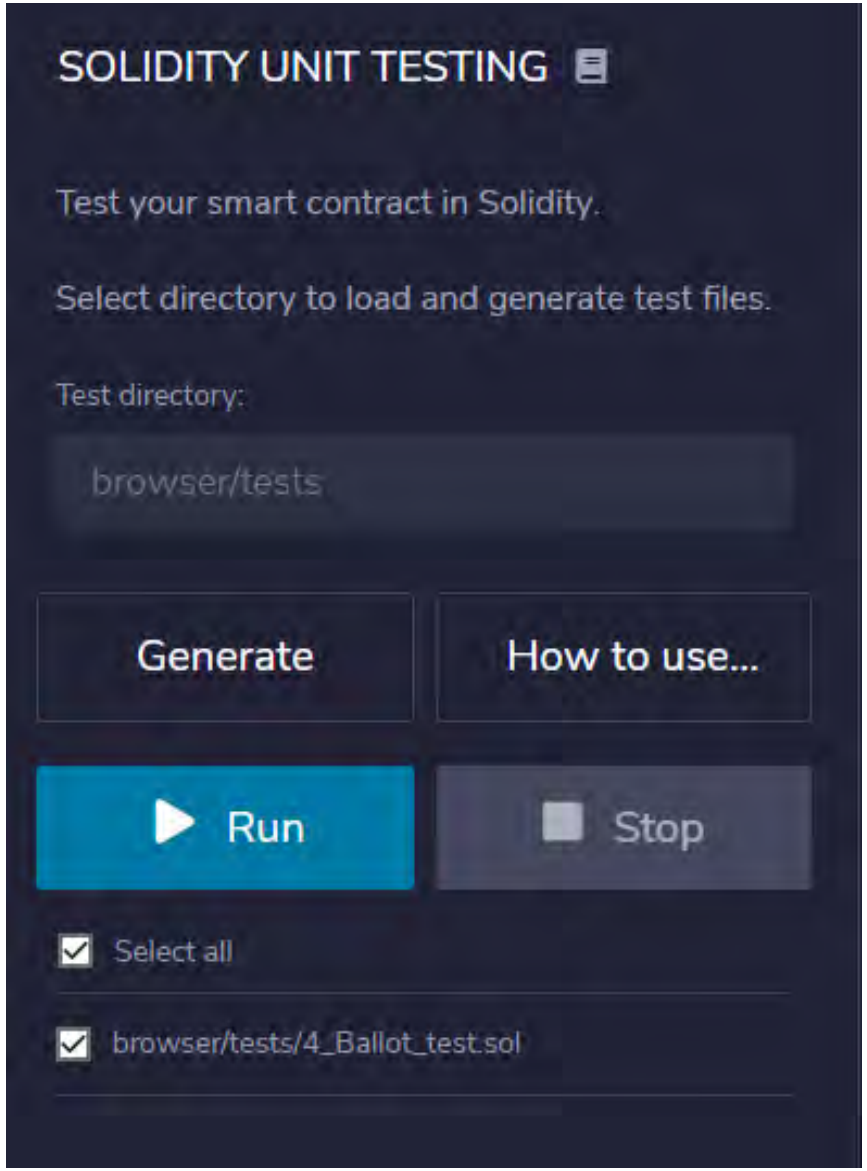
- Выбор пользователя – автора транзакции
- Ограничение газа для транзакции
- Количество средств, которые будут переданы в кошелек контракта вместе с выполнением функции
- Способ размещения контракта в сети – с генерацией нового адреса или с использованием конкретного
- Веб-интерфейс для работы с функциями контракта





RemixIDE

Solidity Unit Testing – плагин, который предоставляет возможность разрабатывать тесты и проводить unit-тестирование для функций разработанного контракта.



Комментарии NatSpec

`@title`	Заголовок, описывающий контракт, библиотеку, интерфейс или структуру данных
`@author`	Имя автора контракта, библиотеки, интерфейса или структуры данных
`@notice`	Информация для конечного пользователя о работе программы или её элемента
`@dev`	Информация для разработчиков о программе или её элементе
`@param`	Описание параметра функции
`@return`	Описание возвращаемого значения функции
`@inheritdoc`	Указание на базовый объект, из которого необходимо скопировать недостающие разделы документации
`@custom:<тег>`	Пользовательский тег, любая дополнительная информация, определяемая разработчиком

Типы данных Solidity

address, address payable	Ethereum-адрес, 20-байтная последовательность. Чтобы была возможность отправлять средства, должен иметь тип address payable
bool	Логический тип, принимает значения True или False
int (int8, int32, int128, int256)	Целое число. Может указываться размер в битах, по умолчанию используется 256
uint (uint8, uint32, uint128, uint256)	Беззнаковое целое число. Может указываться размер в битах, по умолчанию используется 256
fixed, fixedMxN ufixed, ufixedMxN	Знаковые и беззнаковые действительные числа с фиксированным количеством знаков после запятой. Значение M указывает размер типа в битах, оно должно быть кратно 8 и находиться в диапазоне от 8 до 256. Значение N – количество цифр после запятой, оно должно быть от 0 до 80. Типы ufixed и fixed соответствуют ufixed128x18 и fixed128x18.
string	Строка переменного размера в кодировке UTF-8

Типы данных Solidity

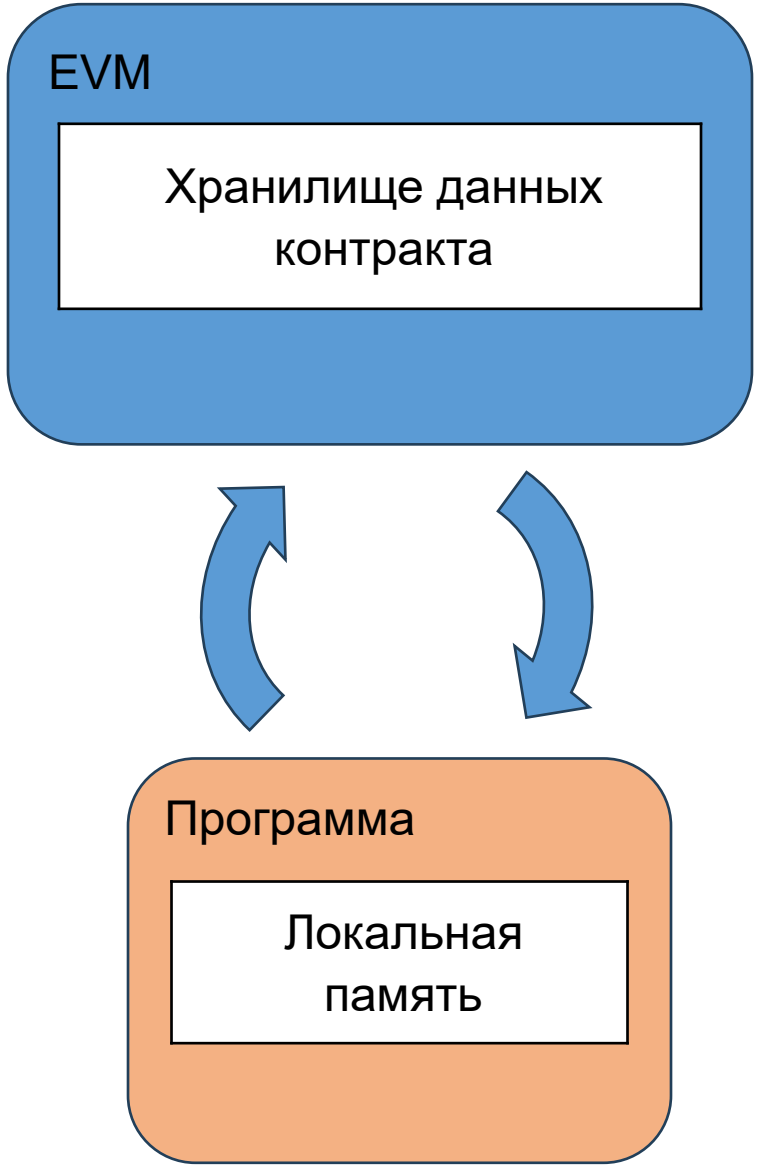
`bytes1, ..., bytes32`	Массив байтов фиксированного размера
`bytes`	Массив байтов переменного размера
`[]`	Массив данных. Тип хранящихся значений указывается перед скобками. Например, uint[5] – массив из 5 целых чисел. Если значение в скобках отсутствует, то считается, что массив имеет переменную длину.
`enum`	Перечисление – тип данных, принимающий конечное множество заданных значений. Задаётся перечислением всех возможных значений в фигурных скобках. Например: **enum Directions {Left, Right, Forth, Back}**
`struct`	Структура – составной тип данных, содержащий несколько значений. Задаётся указанием наименований и типов всех хранящихся значений данных. Например: **struct User {address addr; uint balance;}**
`mapping`	Сопоставление (мэппинг) – тип данных, хранящий взаимосвязи «ключ – значение». В качестве ключа может выступать элементарный тип данных, контракт или перечисление. Значением может являться любой тип данных. Например: **mapping(address => User) UserInfo;**

Типы хранения переменных

`storage` – переменная хранится в глобальном состоянии EVM, поэтому она доступна при последующих вызовах функций контракта.

`memory` – значение переменной хранится в локальной памяти, выделяемой при вызове функций контракта. После завершения работы функции эта память очищается и соответствующее значение пропадает.

`calldata` – область памяти, в которой хранятся аргументы вызываемых функций. В отличие от других данных в памяти, значения переменных, указанных с calldata, не могут быть изменены.



Глобальные значения

Сообщение:

`msg.sender` – адрес создателя транзакции

`msg.value` – количество wei в транзакции

`msg.data` – данные сообщения

Транзакция:

`tx.origin` – отправитель первоначальной транзакции, инициировавшей вызов программы

`tx.gasprice` – цена газа транзакции

Блок:

`block.number` – номер текущего блока

`block.timestamp` или `now` – время создания текущего блока

`block.basefee` – базовая комиссия текущего блока

`block.gaslimit` – лимит газа текущего блока

Константы

`constant` – задаётся на этапе компиляции программы.

`immutable` – определяются на этапе публикации контракта в блокчейне

Денежные суммы:

1 `wei` = 1

1 `gwei` = 10^9 `wei`

1 `ether` = 10^{18} `wei`

Время:

1 `seconds` = 1

1 `minutes` = 60 `seconds`

1 `hours` = 60 `minutes`

1 `days` = 24 `hours`

1 `weeks` = 7 `days`

Объявление функции

function my_function**(**type var, …..**)** атрибуты функции ***returns*****(**type, …..**) {**

тело функции

}

Видимость функций

	Контракт	Дочерние контракты	Внешние вызовы
public			
private			
internal			
external			

Объявление функции

function my_function***(***type var, …..***)*** атрибуты функции ***returns(***type, …..***) {***

тело функции

}

Прочие атрибуты функций

view – чтение, функция возвращает данные из памяти, не изменяя их

pure – вычисления не требуют сведений из глобального состояния

payable – в ходе выполнения функции изменяются балансы

returns – описание типов возвращаемых значений

Алгоритмические конструкции в Solidity

`if (условие) {...;}` `else {...;}`	условный оператор
`for (инициализация; условие; приращение) {...;}`	цикл с параметром
`while (условие) {...;}`	цикл с предусловием
`do` `{...;} while (условие);`	цикл с постусловием

Структура контракта

```
pragma solidity (><=)0.7.0;
```
версия компилятора

```
contract NewContract {
```
объявляем контракт

```
    uint a;
    bool b;
    address Vova = 0xAc771378BB6c2b8878fbF75F80880cbdDefd1B1e;
```
глобальные переменные

```
    constructor {
        .........
    }
```
конструктор – функция, которая выполняется при публикации контракта

```
    function MyFunction {
        .........
    }
```
методы контракта – вызываются транзакциями пользователей или другими контрактами

```
}
```