Lecture 15 - Creating a 1st contract

Setting up a new project

```
$ mkdir payfor2
$ cd payfor2
$ truffle init
```

Scaffold commands to start with

```
$ truffle create contract PayFor
$ truffle create test PayFor
```

You can now build the empty contract and test it.

In a 2nd window start ganache-cli

```
$ ganache-cli
```

Edit truffle-config.js - to connect to ganache. Around line 74

```
74:    development: {
75:      host: "127.0.0.1",
76:      port: 8545,
77:      network_id: "*",
78:    },
```

add a "migration" file in the ./migrations/ directory create `2_initial_migrations.js` with:

```
const PayFor = artifacts.require("PayFor");

module.exports = function(deployer) {
  deployer.deploy(PayFor);
};
```

Now Compile and load the contract:

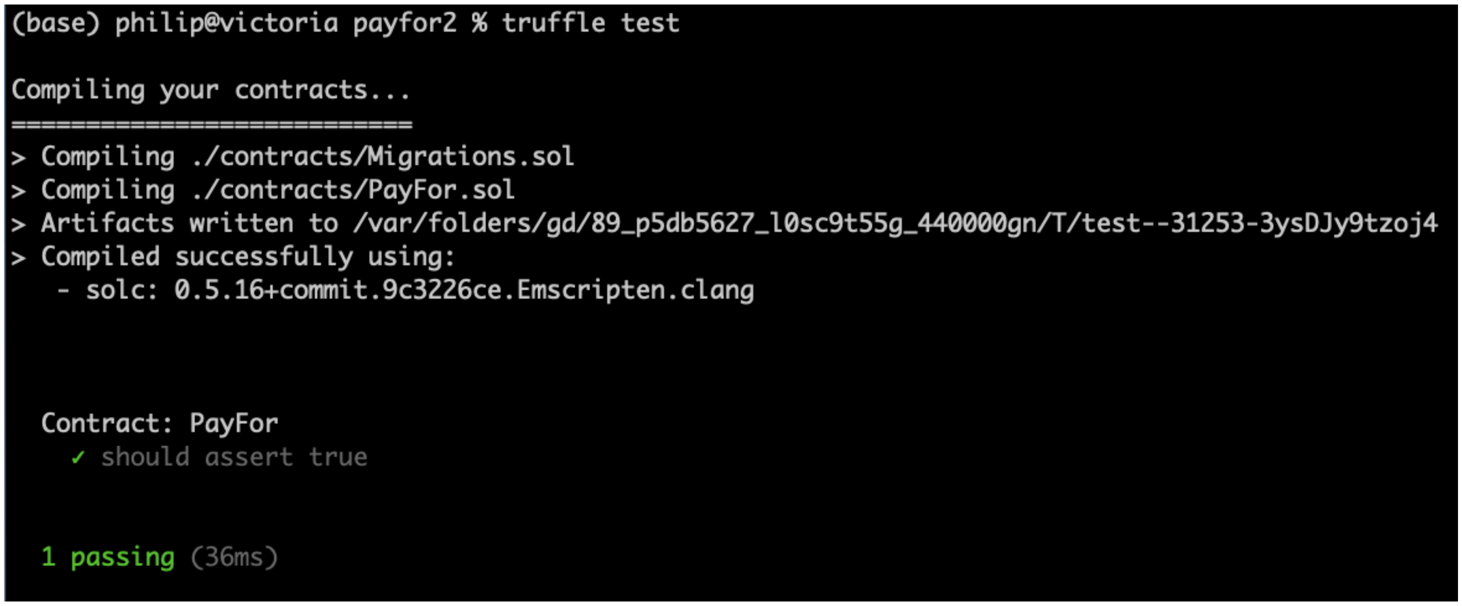
```
$ truffle comile
$ truffle migrate
```

Save the output from the migrate - it is important!

and... Run the empty test

```
$ truffle test
```

It procures a cute output:



```
(base) philip@victoria payfor2 % truffle test

Compiling your contracts...
===========================
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/PayFor.sol
> Artifacts written to /var/folders/gd/89_p5db5627_l0sc9t55g_440000gn/T/test--31253-3ysDJy9tzoj4
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang



  Contract: PayFor
    ✓ should assert true


  1 passing (36ms)
```

To reload a contract you need a “truffle migrate --reset”.

```
(base) philip@victoria payfor % truffle migrate --reset

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



Starting migrations...
======================
> Network name:    'development'
> Network id:      1645743739818
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0xc2215471f6e946eb1990156ccd5367b81f540f5d4b775fdf7aa2fda6d3161330
   > Blocks: 0            Seconds: 0
   > contract address:    0x3b6bdDFC0A92E3BDa4dd4941D5319b40227cdE21
   > block number:        31
   > block timestamp:     1645795766
   > account:             0x5C046b3B982a2073584613213e40C22d6A876300
   > balance:             99.89637302
   > gas used:            191943 (0x2edc7)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00383886 ETH


   > Saving migration to chain.
   > Saving artifacts
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00383886 ETH


2_initial_migrations.js
=======================

   Replacing 'PayFor'
   ------------------
   > transaction hash:    0x768be0029d0a020c861bb22ffe7dd0efab40eb1f3a4a96f8d4191f70638c3f59
   > Blocks: 0            Seconds: 0
   > contract address:    0x921511c8972EA1e3c4A9Acf117714917eB3f6a15
   > block number:        33
   > block timestamp:     1645795766
   > account:             0x5C046b3B982a2073584613213e40C22d6A876300
   > balance:             99.8828177
   > gas used:            635428 (0x9b224)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01270856 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01270856 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.01654742 ETH
```

Take a look at the contract code.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract PayFor {

    struct productPriceStruct {
        uint256 price;
        bool isValue;
    }

    address payable owner_address;
    event ReceivedFunds(address sender, uint256 value, uint256 application, uint256 loc);
    event Withdrawn(address to, uint256 amount);
    event SetProductPrice ( uint256 product, uint256 minPrice );
    event LogDepositReceived(address sender);

    uint256 internal nPayments;
    uint256 internal paymentID;

    address[] private listOfPayedBy;
    uint256[] private listOfPayments;
    uint256[] private payFor;

    mapping (uint256 => productPriceStruct) internal productMinPrice;

    mapping (address => uint256) internal totalByAccount;

    constructor() public {
        owner_address = msg.sender;
        nPayments = 0;
    }

    /**
     * @return the address of the owner.
     */
    function owner() public view returns (address) {
        return owner_address;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == owner_address;
    }

    /**
     * @dev set the minimum price for a product.  Emit SetProductPrice when a price is set.
     */
    function setProductPrice(uint256 productNumber, uint256 minPrice) public onlyOwner {
        productMinPrice[productNumber] = productPriceStruct ( minPrice, true );

        emit SetProductPrice ( productNumber, minPrice );
    }
    /**
     * @return true for funds received.  Emit a ReceivedFunds event.
     */
    function receiveFunds(uint256 forProduct) public payable returns(bool) {
        // Check that product is valid
        require(productMinPrice[forProduct].isValue, 'Invalid product');
        // Validate that the sender has payed for the prouct.
        require(msg.value > productMinPrice[forProduct].price, 'Insufficient funds for product');

        uint256 pos;
        uint256 tot;
        nPayments++;
        pos = listOfPayments.length;
        listOfPayedBy.push(msg.sender);
        listOfPayments.push(msg.value);
        payFor.push(forProduct);
        tot = totalByAccount[msg.sender];
        totalByAccount[msg.sender] = tot + msg.value;
        emit ReceivedFunds(msg.sender, msg.value, forProduct, pos);
        return true;
    }

    /**
     * @return the total that has been payed by an account.
     */
    function getNPayments(address lookUp) public onlyOwner returns(uint256) {
        return ( totalByAccount[lookUp] );
    }

    /**
     * @return the number of paymetns.
     */
    function getNPayments() public onlyOwner payable returns(uint256) {
        return ( nPayments );
    }

    /**
     * @return the address that payeed with the payment amount and what was payed for.
     */
    function getPaymentInfo(uint256 n) public onlyOwner payable returns(address, uint256, uint256) {
        return ( listOfPayedBy[n], listOfPayments[n], payFor[n] );
    }

    /**
     * @dev widthdraw funds form the contract.
     */
    function withdraw( uint256 amount ) public onlyOwner returns(bool) {
        require(address(this).balance >= amount, "Insufficient Balance for withdrawl");
        address(owner_address).transfer(amount);
        emit Withdrawn(owner_address, amount);
        return true;
    }

    /**
     * @return the amount of funds that can be withdrawn.
     */
    function getBalanceContract() public view onlyOwner returns(uint256){
        return address(this).balance;
    }





    /**
     * @return Catch and save funds for abstrc transfer.
     */
    function() external payable {
        require(msg.data.length == 0);
        emit LogDepositReceived(msg.sender);
    }

}
```