

---

# **Learneth Documentation**

**Filip Mertens**

**Feb 02, 2022**



# CONTENTS

<b>1</b>	<b>About this plugin</b>	<b>1</b>
<b>2</b>	<b>The default tutorials</b>	<b>3</b>
<b>3</b>	<b>Creating tutorials</b>	<b>5</b>
3.1	Setting up your repository . . . . .	5
<b>4</b>	<b>Using the plugin</b>	<b>15</b>
4.1	Importing files . . . . .	15
4.2	Using tutorials . . . . .	16
<b>5</b>	<b>Requirements</b>	<b>23</b>
<b>6</b>	<b>Installation</b>	<b>25</b>
<b>7</b>	<b>Configuration</b>	<b>27</b>



## ABOUT THIS PLUGIN

This plugin is used in the [Remix IDE](#). Remix is a powerful, open source tool that helps you write Solidity contracts straight from the browser. Read more about [Remix here](#).

By default the plugin provides step by step tutorials on how to *use the features of the Remix IDE, code in the Solidity language* and many other topics related to it.

The plugin also allows the user to experiment with code. The plugin is able to evaluate the user's input and display answers and solutions to the assignments.

The plugin also allows anyone to **create new tutorials** to present to their audience.



## THE DEFAULT TUTORIALS

By default the plugin loads its tutorials from a [Remix workshops repository](#). It's a set created by the Remix team to get you started on a wide range of topics.





## CREATING TUTORIALS

To create a new set of tutorials you will need to set up a new repository on github. You and your audience will then be able to import those tutorials into the plugin. Follow these instructions to structure your repository correctly.

### 3.1 Setting up your repository

You can create your own workshops that can be imported in the plugin. When importing a github repo the plugin will look for a directory structure describing the tutorials.

Take a look at this example: [Remix workshops repository](#)

#### 3.1.1 Basic concepts

**Each tutorial is a group of ‘steps’.**

For example the tutorial called “Remix Basics” contains steps to teach you:

- The UI
- How to compile files
- How to deploy
- ...

A step is one task the user needs to do, or one thing they need to learn. For example how to deploy contracts. But it can be anything you like.

A step contains a file describing what the student needs to do or learn.

It can also contain several **code files**:

- **solidity, js or vyper files.** These will be loaded automatically when the step opens. Your step will describe the file or tell users what to do with it, for example, add something and compile.
- **answer files.** These are files containing the correct answer of the step.
- **test files.** Solidity Unit Testing files.

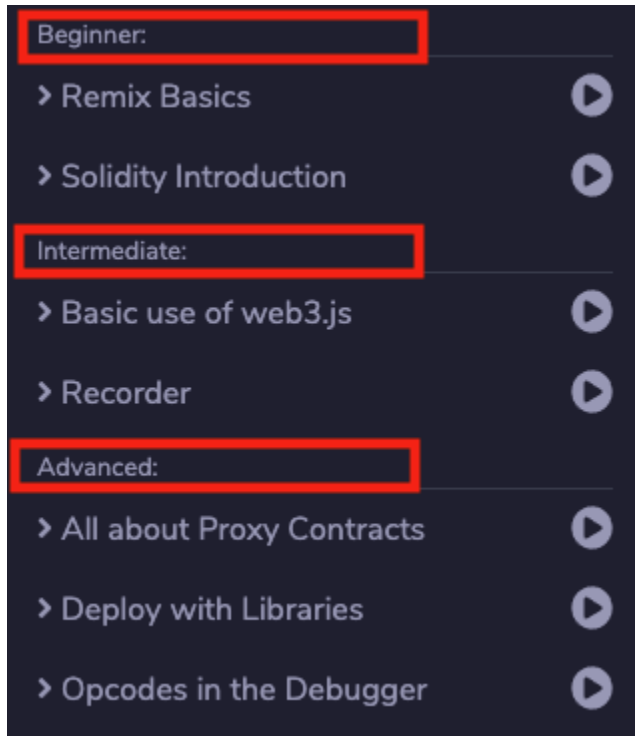
More on

**A list of tutorials.**

You can have more than one tutorial in your repo, it’s like a list of tutorials. And each tutorial has its own directory and its own configuration.

**Levels**

To make it easy to categorize your tutorials we have 3 levels, beginner, intermediate and advanced. You can set this in the config of your tutorial.



### 3.1.2 Root file structure

It is important you adhere to the directory structure for the system to work, loading any repo with sol files won't work. So for example

Basics	add id in config.yml	2 months ago
DeployWithLibraries	add id in config.yml	2 months ago
OpcodesInTheDebugger	add id in config.yml	2 months ago
ProxyContract	update proxy contract & recoder .yml files	21 days ago
Recorder	update proxy contract & recoder .yml files	21 days ago
Solidity-intro	add id in config.yml	2 months ago
Web3Client	update proxy contract & recoder .yml files	21 days ago
.gitignore	add gitignore	2 years ago
README.md	Update README.md	2 months ago

The readme.md in the root directory is not used by the plugin.

Each directory in the root is a **tutorial or workshop**.

For example *Basics* is a tutorial that contains steps teaching the basics of the Remix IDE.

### 3.1.3 Naming your tutorials

The name of the tutorial that is displayed in the LearnEth plugin will be either the

- name of this directory, for example 'Basics' OR
- a name provided in a yml file.

This config.yml file lives in each directory and is *required* by the system.

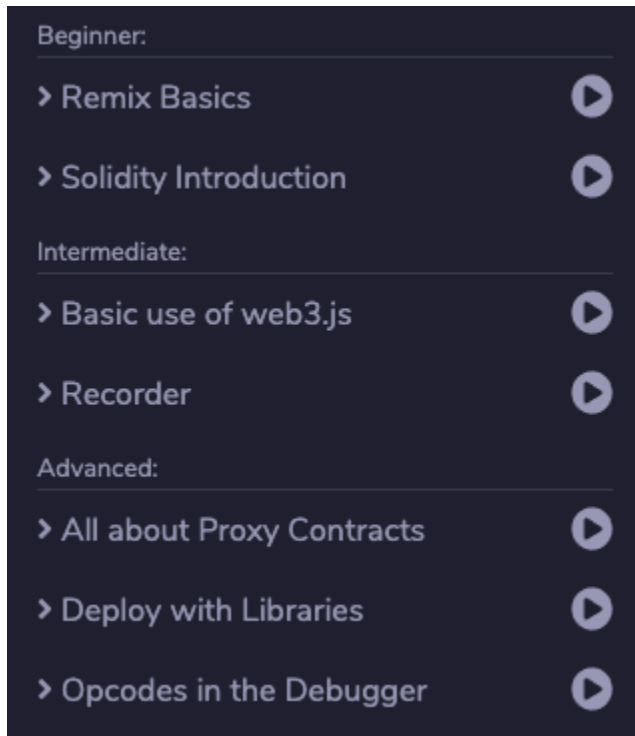
..	
1_Pragma	cleanup of folder names in solidity intro
2_variables	cleanup of folder names in solidity intro
3_Modify_Variables	Merge pull request <a href="#">#98</a> from ethereum/yann300-patch-6
4_Retrieve_Variables	update folder names
5_Constructor	update all but error handling
6_Address_and_Mapping	update to Solidity Intro
7_Events	update all but error handling
8_Error_Handeling	cleanup of folder names in solidity intro
README.md	update to Solidity Intro
config.yml	add id in config.yml

config.yml:

```
---
name: Remix Basics
```

This name will appear everywhere in the UI.

So for example in the main list of tutorials:

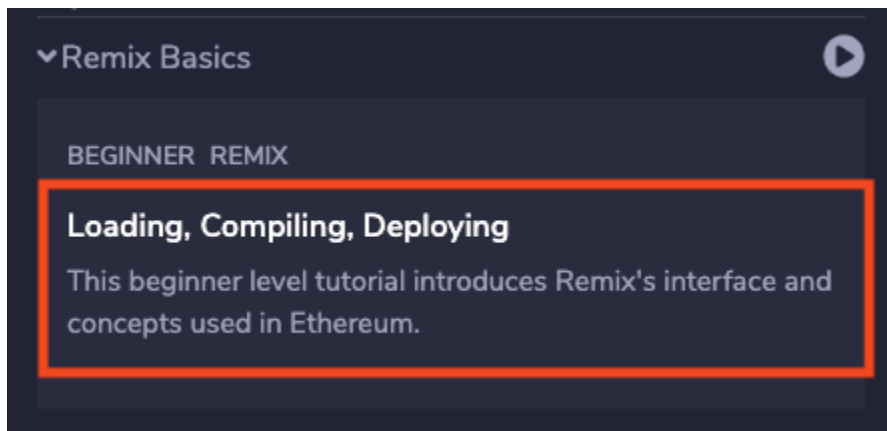


### 3.1.4 Sorting tutorials

The tutorials are grouped by level, ie Beginner, and then sorted alphabetically by name.

### 3.1.5 Description of your tutorial

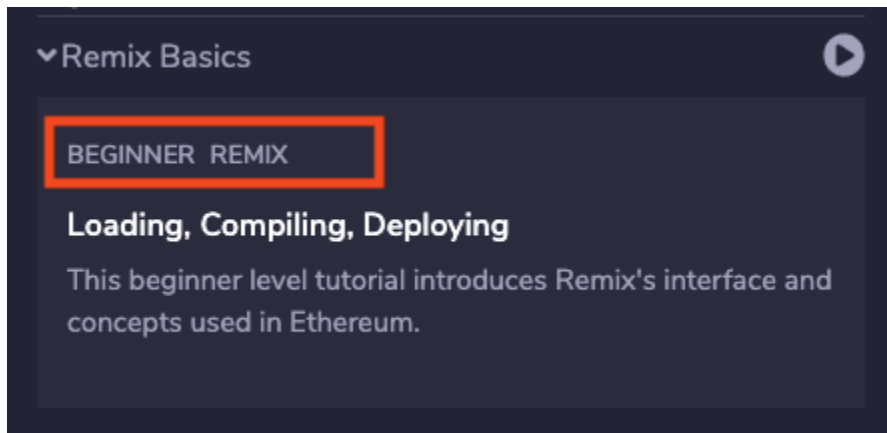
The description is shown in the overview in the list of tutorials:



This is the content of the readme.md in the tutorial directory. It's in markdown.

### 3.1.6 Tags & levels

As mentioned before a tutorial can have a level, but it doesn't have to have one. It can also have tags. In this example the tag is REMIX, the level is BEGINNER.



You can set the level and tags in the config.yml file

```
level: 1
tags:
  - Remix
  - tag2
  - tag3
```

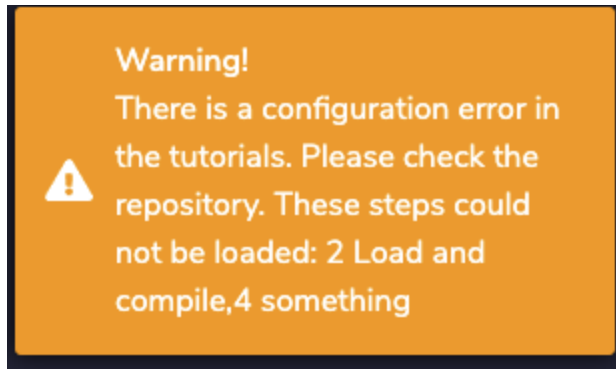
Level 1 is beginner. Level 2 is intermediate. Level 3 is Advanced.

### 3.1.7 Step configuration

You have the option of defining which steps you have in your tutorial. By default the directories are used to define the steps. But you can override this by adding the steps object to the config.yml file:









```
level: 1
tags:
  - Remix
  - tag2
  - tag3
steps:
  - name: 1 intro ui
    path: 1._Interface_introduction
  - name: 2 Load and compile
    path: 2_Load_and_compile // this directory does not exist
  - name: 3 Deploy
    path: 3._Deploy_to_the_JavascriptVM
  - name: 4 something
    path: 4._does_not_exist // this directory does not exist
```

When loading the repo these steps will be mapped to their corresponding directories. If you have an error there, for example when a directory does not exist the app will display a warning like this:



### 3.1.8 Name of a step

There are two ways of doing this. Default behavior: The name of a step is just the name of directory it resides in, but without any `_`. If you have steps defined in your `config.yml` those will be used and directories will be ignored.

-  1\_Pragma
-  2\_variables
-  3\_Modify\_Variables
-  4\_Retrieve\_Variables
-  5\_Constructor
-  6\_Address\_and\_Mapping
-  7\_Events
-  8\_Error\_Handeling

## Solidity Introduction

1 Pragma »

2 variables »

3 Modify Variables »

4 Retrieve Variables »

5 Constructor »

6 Address and Mapping »

7 Events »

8 Error Handeling »

### 3.1.9 Sorting steps

There are two ways of doing this. Default behavior: Steps are sorted alphabetically according to the name of the directory. So it's best to precede each step name by a number. If you have steps defined in your `config.yml` those will be used and directories will be ignored. The sorting is the order in which the steps are set in the `config.yml`, not alphabetically.

### 3.1.10 Step description

When you open a step, you see a text describing what to do or learn. This text is provided by the markdown file in the directory of each step.

In this example the file would be here: `Solidity-intro/1_Pragma/pragma.md`

## 1 Pragma

### Specify the compiler version with Pragma

The first line of a contract tells the compiler which version to use. This prevents you from adding functionality that a compiler does not support.

Here is a pragma that sets solidity to a specific version. `pragma solidity =0.5.2;`

The syntax starts with `pragma solidity`, and uses standard mathematical symbols to define the range of versions in minimum and maximum pairs.

#### Try it out!

Add a pragma statement above `contract SimpleStorage` that specifies a compiler version greater than or equal to 0.4.0, and less than 0.7.0.

If you get stuck, [read the pragma documentation](#).

When you're finished, compile the contract to see if the syntax is correct.

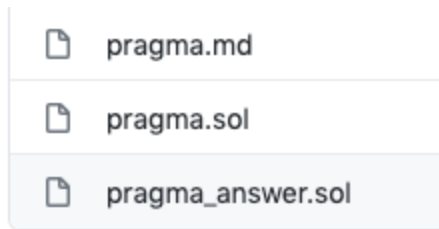
### 3.1.11 File Types & loading files

Each step can contain one file of each type. So this means:

- a markdown file describing the step
- a solidity file (.sol)
- a js file (.js)
- a vyper file (.vy)
- a test file The naming convention for these is name\_test.sol
- an answer file. The naming convention for these is name\_answer.sol

When a step loads the code files are loaded automatically into the Remix IDE. But not the answer file. The test file is only used by the Unit Testing system and not loaded in the IDE.





### 3.1.12 Answer files

Answer files are just files that display an answer. This can be anything.

If the filename contains `_answer` it will be shown in the interface and users can click on 'SHOW ANSWER'.

**Try it out!**

Add a `pragma` statement above `contract SimpleStorage` that specifies a compiler version greater than or equal to 0.4.0, and less than 0.7.0.

If you get stuck, [read the pragma documentation](#).

When you're finished, compile the contract to see if the syntax is correct.

Show answer

Next

### 3.1.13 Test files

These are Solidity Unit Testing files.

They are run through the Unit testing system when a user clicks on: 'CHECK ANSWER'.

**Try it out!**

Create a variable that is:

- a 256 bit sized unsigned integer
- call it `storedData`
- set it equal to 3

If you get stuck, [read the integer variable documentation](#).

Check Answer Show answer

If the test fails the UI displays an error.

Check Answer Show answer

**Errors**

```
Error: Error from IDE : .learneth/Solidity
Introduction/2_variables/variables.sol:7:12:
DeclarationError: Undeclared identifier. return
storedData; ^-----^
```

If the test succeeds the user can proceed to the next step:

Check Answer Show answer

Next

Well done! No errors.

The user can always skip a step by picking a step in the steps overview or using the navigation.

## USING THE PLUGIN

Read about the plugin interface and how to use it here.

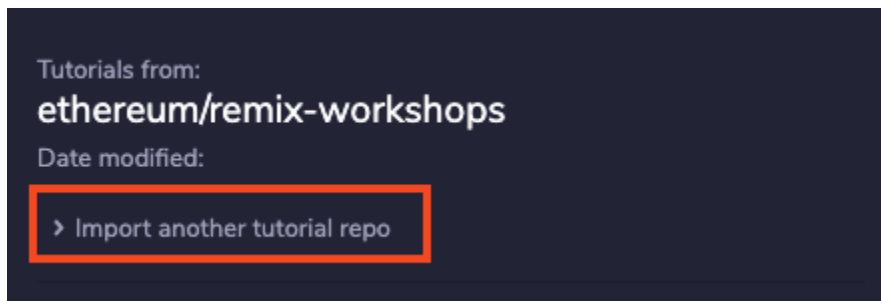
### 4.1 Importing files

#### 4.1.1 Loading the default tutorials

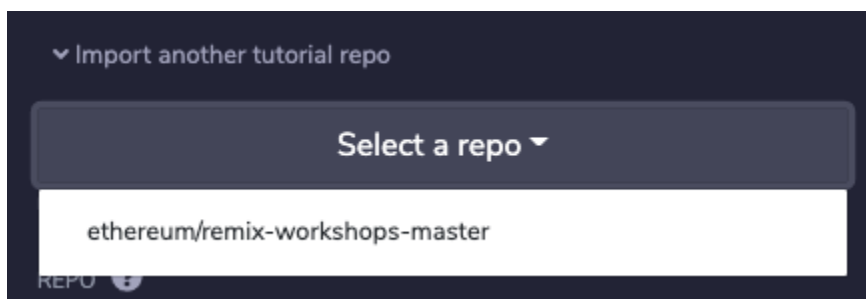
The default tutorials created by the Remix team are loaded by default on first use.

Once you loaded another repository that will be loaded on startup instead of the default ones.

You can get back the default tutorials by using the import section.

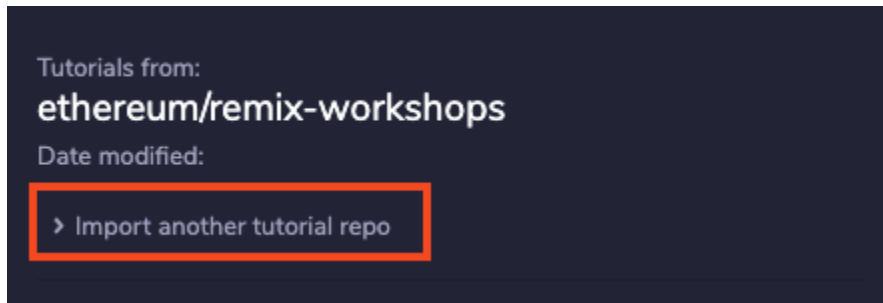


There will be a dropdown. Select "ethereum/remix-workshops-master".



## 4.1.2 Importing another repository

Open the import section.



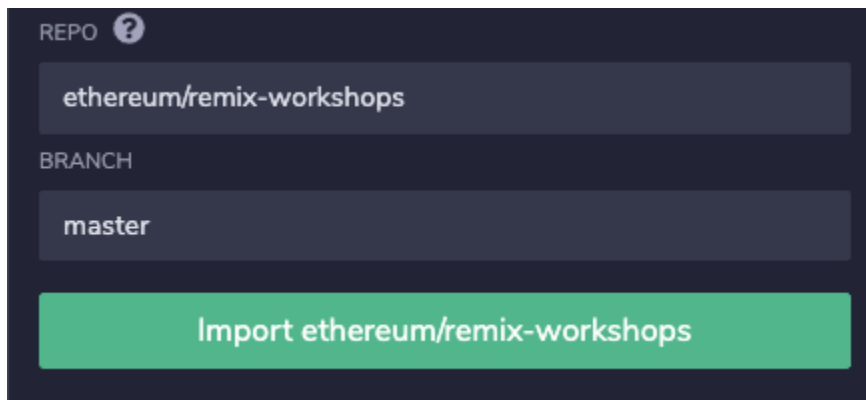
Fill in the details of your github repo and click import.

The repo name is basically the url of the repository on github but without github.com.

So this:



Becomes:



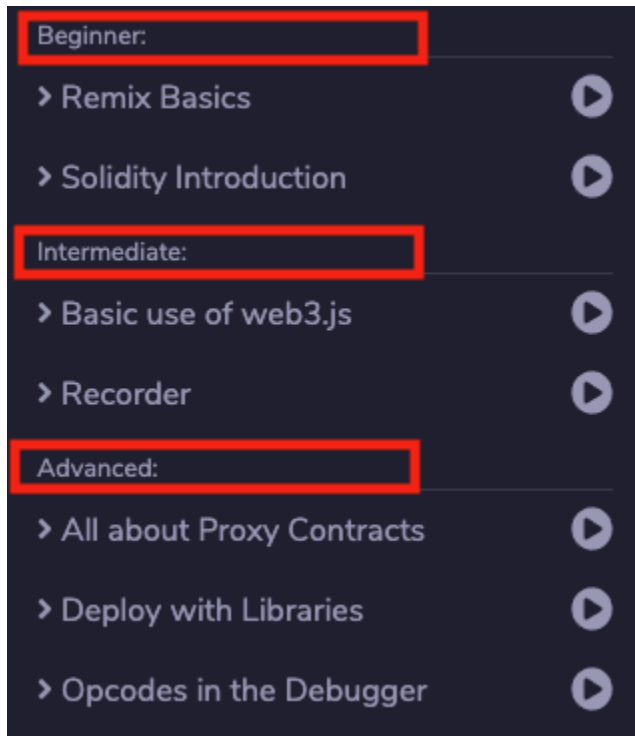
If you set up the repo correctly according to the steps outlined in Setting up your repository you should be fine.

## 4.2 Using tutorials

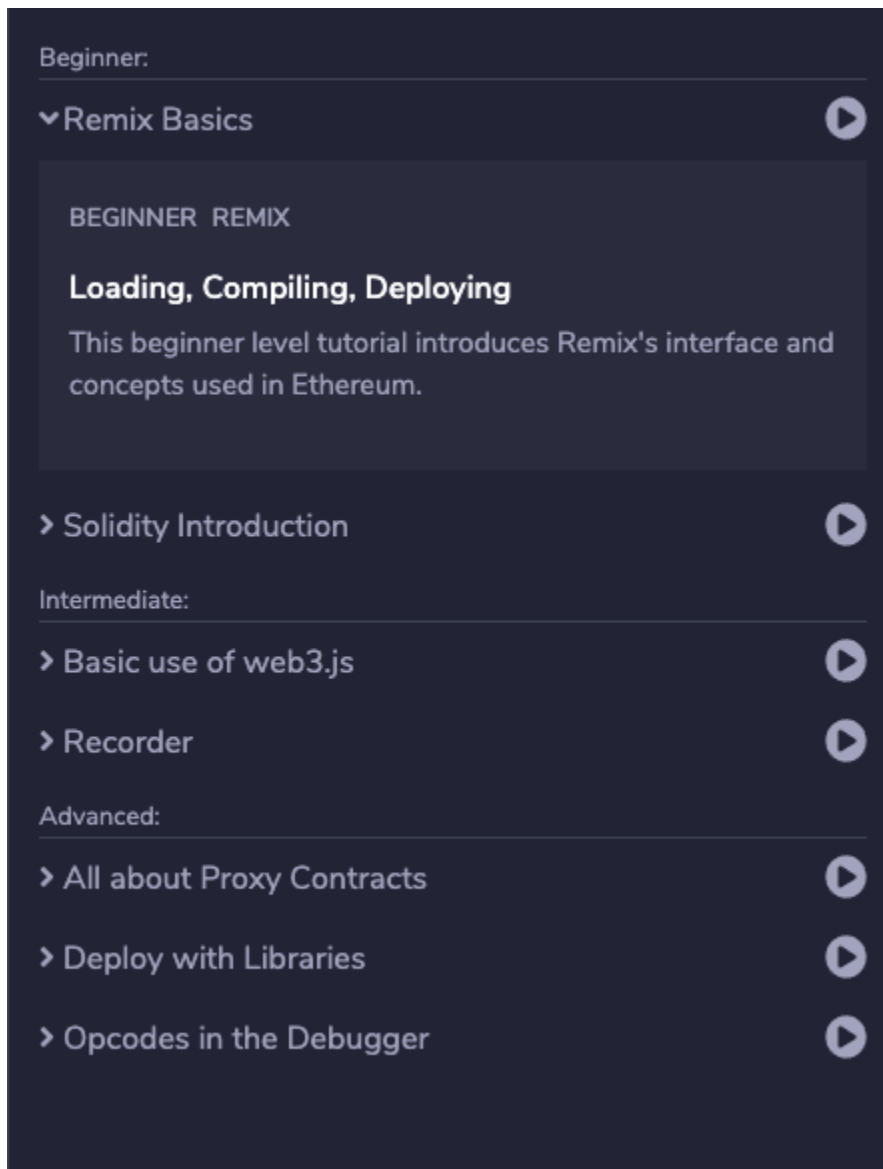
### 4.2.1 The tutorial list

The plugin opens with a list of tutorials you can use.

They are grouped by the level of difficulty.



Each tutorial provides a short description of what it's for.



Click on the **play button** to start.

## 4.2.2 Steps

Each tutorial consists of several steps you can go through.

## Solidity Introduction

1 Pragma »

2 variables »

3 Modify Variables »

4 Retrieve Variables »

5 Constructor »

6 Address and Mapping »

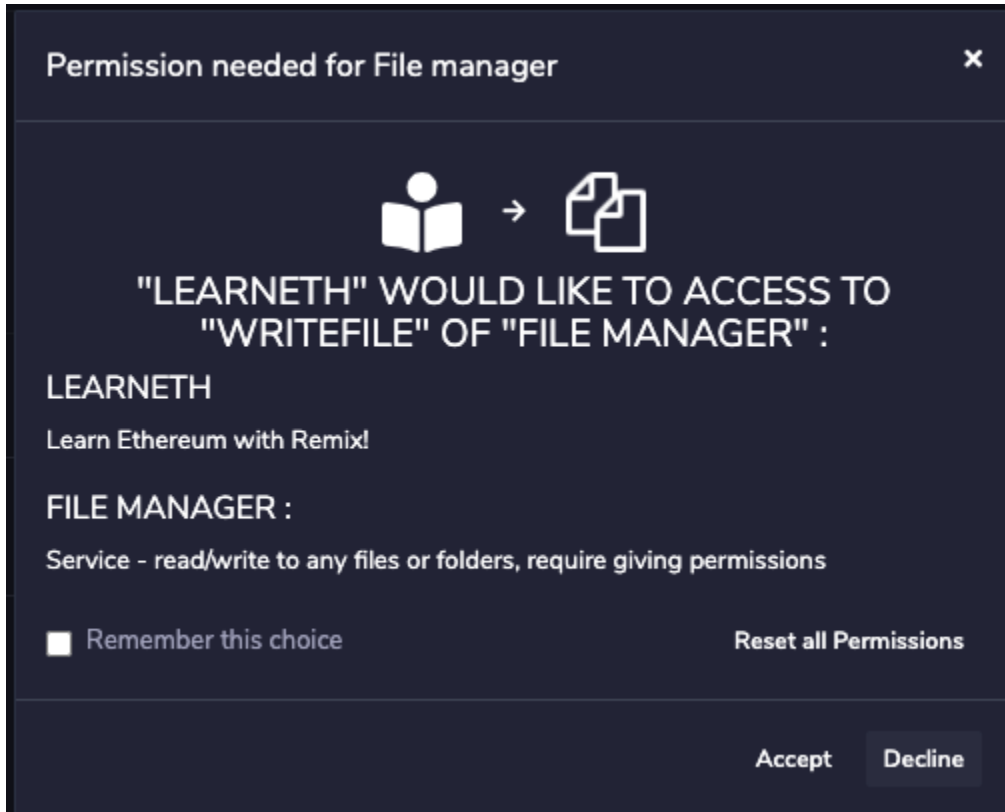
7 Events »

8 Error Handeling »

### 4.2.3 Loading files

Sometimes a step will load a file in the IDE.

It is best to check 'Remember this choice' so you only see this window once.



#### 4.2.4 Answers and checking answers

Sometimes a step contains a solution to the problem. It will be displayed on the right in the Remix IDE.

**Try it out!**

Add a pragma statement above `contract SimpleStorage` that specifies a compiler version greater than or equal to 0.4.0, and less than 0.7.0.

If you get stuck, [read the pragma documentation](#).

When you're finished, compile the contract to see if the syntax is correct.

Show answer

Next

Sometimes a step contains a test. This will test the code you've written and display feedback.



### Try it out!

Create a variable that is:

- a 256 bit sized unsigned integer
- call it `storedData`
- set it equal to 3

If you get stuck, [read the integer variable documentation](#).

Check Answer

Show answer

You have an error in your code:

Check Answer

Show answer

#### Errors

```
Error: Error from IDE : .learneth/Solidity
Introduction/2_variables/variables.sol:7:12:
DeclarationError: Undeclared identifier. return
storedData; ^-----^
```

You have succeeded in the test:

Check Answer

Show answer

Next

Well done! No errors.

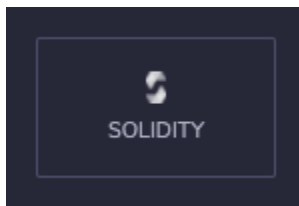
### 4.2.5 Navigation

- The home icon takes you back to the start
- The hamburger takes you to the list of steps in this tutorial
- Use the arrows to navigate to between steps
- Use the 'next' buttons to navigate to the next step



## REQUIREMENTS

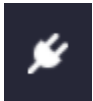
Some tutorials need a set of Solidity plugins to be loaded into the IDE, otherwise testing your code is not possible. On the home screen of the Remix IDE you will find a button to activate that set:



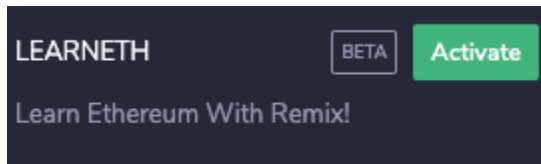


## INSTALLATION

Go to the plugin manager using the button in the sidebar



Find the LearnEth plugin the list and click 'activate'





## CONFIGURATION

That's it, no configuration is needed. On the sidebar you will find the LearnEth icon. Click that to start the plugin.

