



SIMA EXAMPLE P1

# Getting Started with SimaPy

Valid from Sima version 4.6

---





Sima Example

Getting Started with SimaPy

Date: October 2024

Valid from Sima version 4.6

**Prepared by: Digital Solutions at DNV**

E-mail support: [software.support@dnv.com](mailto:software.support@dnv.com)

E-mail sales: [digital@dnv.com](mailto:digital@dnv.com)

© DNV AS. All rights reserved.

This publication or parts thereof may not be reproduced or transmitted in any form or by any means, including copying or recording, without the prior written consent of DNV AS.

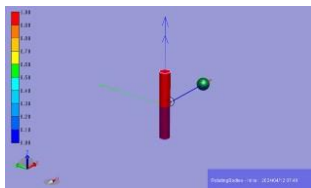
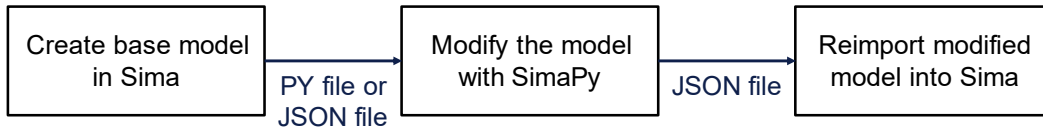


## Table of contents

1	INTRODUCTION.....	1
2	SETTING UP PYTHON DEVELOPMENT ENVIRONMENT .....	1
2.1	Installing Visual Studio Code (VS Code)	1
2.2	Installing Python 3.12.x	2
2.3	Installing Python extension for VS Code	3
2.4	Installing Jupyter Notebook Extension	3
2.5	Installing SimaPy Python Library	4
3	TESTING PYTHON DEVELOPMENT ENVIRONMENT .....	7
4	TESTING JUPYTER NOTEBOOK .....	9
5	CREATING SIMA MODEL WITH SIMAPY.....	12
5.1	Exporting an existing model as Python file in Sima	12
5.2	Checking the Contents of the Python File	14
5.3	Writing the Model as JSON File from SimaPy	17
5.4	Using Python Scripting to Modify the Model	18

# 1 INTRODUCTION

SimaPy is a Python library that can be used to interact with Sima. It can be used to create new Sima models, modify existing models, and run simulations without using the Sima GUI. SimaPy is mainly useful to automate the modelling of Sima analysis, which we will learn in this example.

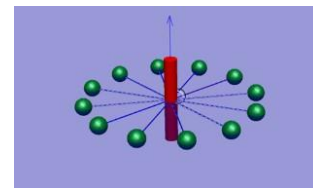


```

for i in np.arange(1,bodies_num):
    # Set the direction of the assembly
    current_dir = dir_start1 # degrees
    current_dir = np.deg2rad(current_dir)

    # Copy moving1 as base
    moving = moving1.copy()

    # Change some parameters of "moving"
    moving.name = f"MOVING{i+1}"
    moving.initialPosition.x+= np.cos(current_dir) * 20.0
    moving.initialPosition.y= np.sin(current_dir) * 20.0
  
```



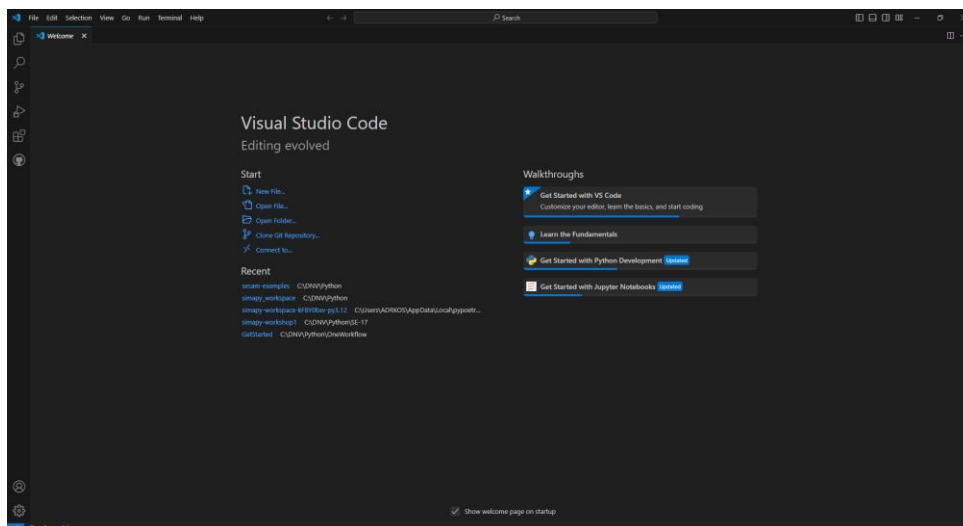
# 2 SETTING UP PYTHON DEVELOPMENT ENVIRONMENT

This chapter will walk you through setting up a development environment for SimaPy using Visual Studio Code (VS Code) and Python to kickstart your SimaPy experience. We will also set up Jupyter to use other SimaPy examples.

**Note:** Experienced Python users may skip the Python setup and go directly to Chapter 2.5, step 5.

## 2.1 Installing Visual Studio Code (VS Code)

Download and install VS Code from the [official website](#). VS Code is a free, open-source code editor that provides excellent support for various programming languages and extensions, making it a popular choice for developers.



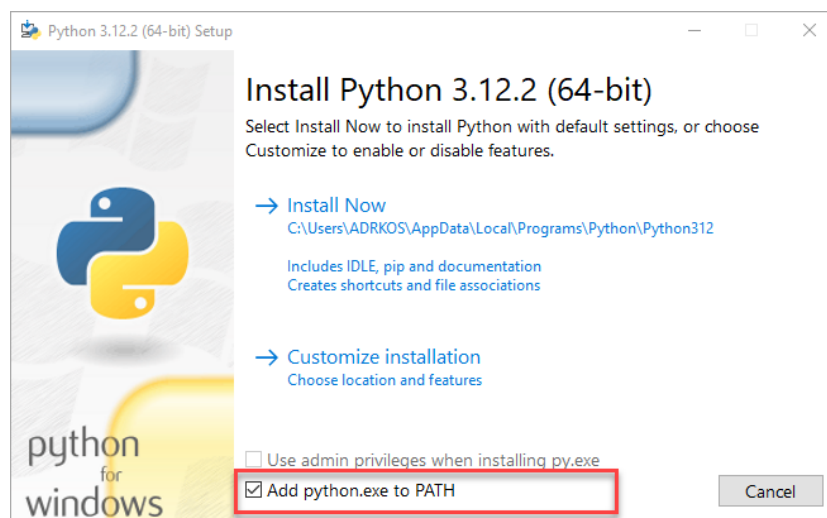
**Note:** We will use VS Code for this example, but you may use any other text editor or IDE you like. You may also change the colour theme to any of your likings.

## 2.2 Installing Python 3.12.x

To set up your SimaPy development environment, you will need to install Python 3.10.x or newer. If you already have a compatible version of Python installed, feel free to skip this step.

Follow these steps to get Python 3.12.x (the latest version at the time of writing this example) installed:

1. **Download Python 3.12.x:** Visit the [official Python website](#).
2. **Install Python 3.12.x:** Run the installer you downloaded in the previous step. One important note during the installation is to select the option **'Add python.exe to the PATH'** (as shown in the image below). This option is turned off by default, so make sure to activate it for a smoother development experience. Both 32-bit and 64-bit Python versions are supported.



3. **Verify the Default Python Version:** To ensure a smooth development experience, it's important to confirm the default Python version on your system, especially if you have multiple Python installations.

Follow these steps:

- Open a new terminal window: Windows start menu > type "cmd" > press Enter.
- In the terminal window, run the following command. This will display the installed version of Python.

```
python --version
```

The output will show the default Python version installed on your system. This information is useful for setting up your development environment and ensuring compatibility with SimaPy.

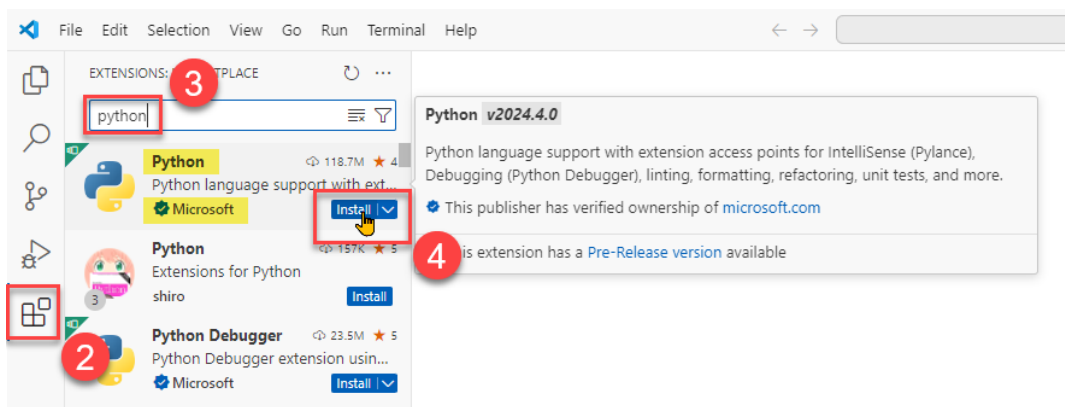


## 2.3 Installing Python extension for VS Code

The Python extension for VS Code offers rich support for the Python language. It comes with features such as IntelliSense (Pylance), linting, debugging, code navigation, code formatting, refactoring, variable explorer, test explorer, and more. It is a handy tool for Python developers.

To install it in VS Code, follow these steps:

1. **Open VS Code.**
2. Click the **Extensions** icon (the four squares icon) in the Activity Bar.
3. In the search bar, **type “python”**.
4. Click the **Install** button next to the Python extension by Microsoft.
5. Once the extension is installed, **restart VS Code**.



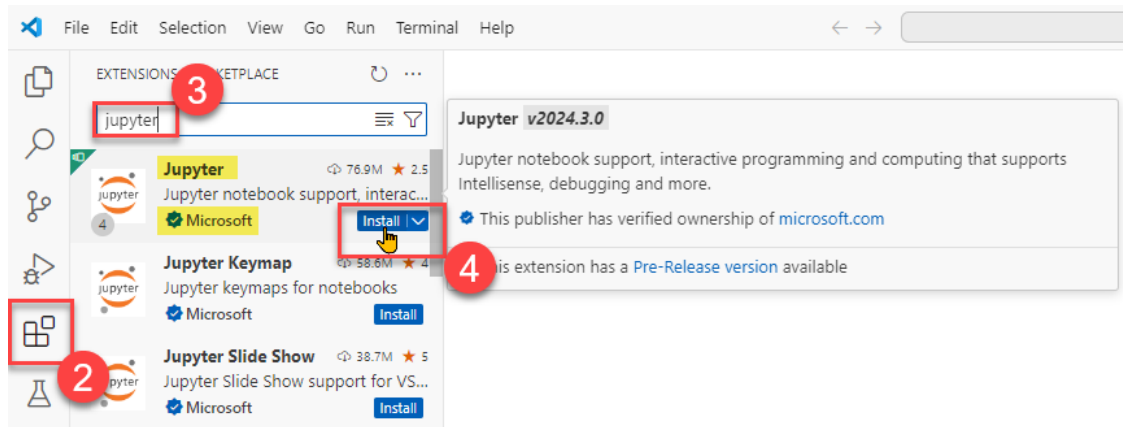
VS Code should automatically detect the Python language and provide syntax highlighting and other features.

## 2.4 Installing Jupyter Notebook Extension

The Jupyter Notebook extension for VS Code provides basic notebook support for language kernels that are supported in Jupyter Notebooks today and allows any Python environment to be used as a Jupyter kernel. Future examples of SimaPy will be delivered as Jupyter notebooks, so it is recommended to install it now.

To install the Jupyter Notebook extension in VS Code, follow these steps:

1. **Open VS Code.**
2. Click the **Extensions** icon (the four squares icon) in the Activity Bar.
3. In the search bar, **type “Jupyter”**.
4. Click the **Install** button next to the Jupyter extension by Microsoft.
5. Once the extension is installed, **restart VS Code**.

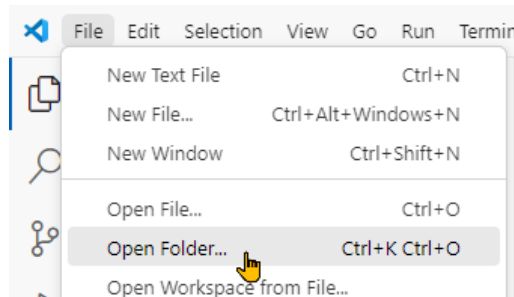


The Jupyter Notebook extension will enable you to create, edit, and run Jupyter notebooks directly in VS Code.

## 2.5 Installing SimaPy Python Library

Follow these steps to create your project folder and install the SimaPy Python library:

1. Open **VSCode**.
2. Create and open a project folder, for example, **C:\DNV\Python\simapy\_workspace** folder (File > Open Folder). You may need to create the folder manually in Windows explorer.

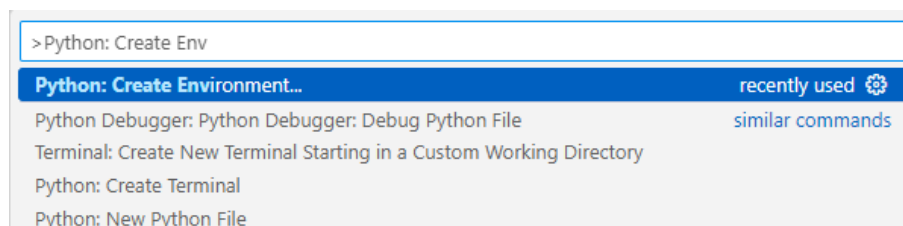


3. (optional) Activate a **virtual environment**.

**Note:** A virtual environment is a tool that helps to keep dependencies (libraries) and configurations of a Python project separate from other projects. For SimaPy, virtual environment is mainly useful when you want to use different Sima and SimaPy versions in a machine without conflicts. While you do not have to use virtual environment, it is a good practice to use it in your projects.

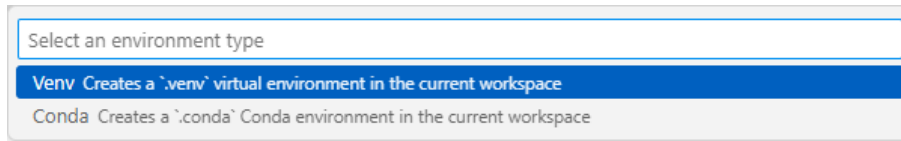
To set up a local virtual environment in VS Code, you can follow these steps:

- Open the Command Palette (**Ctrl+Shift+P**), search for the **Python: Create Environment** command, and select it.

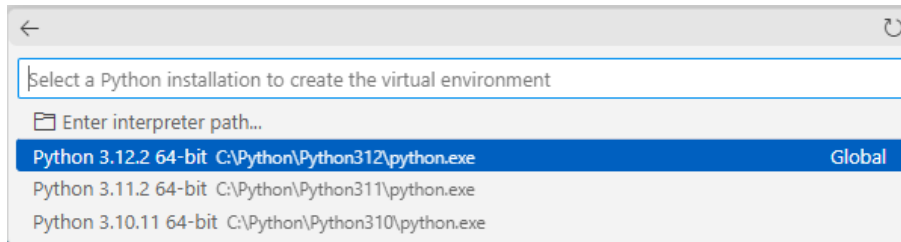




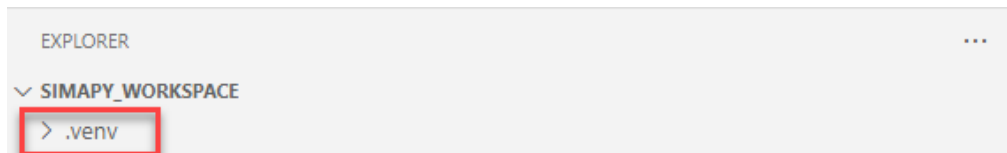
- Select **Venv**.



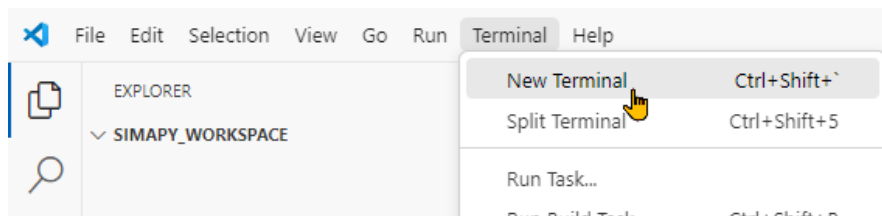
- Select **Python 3.12.x** you just installed.



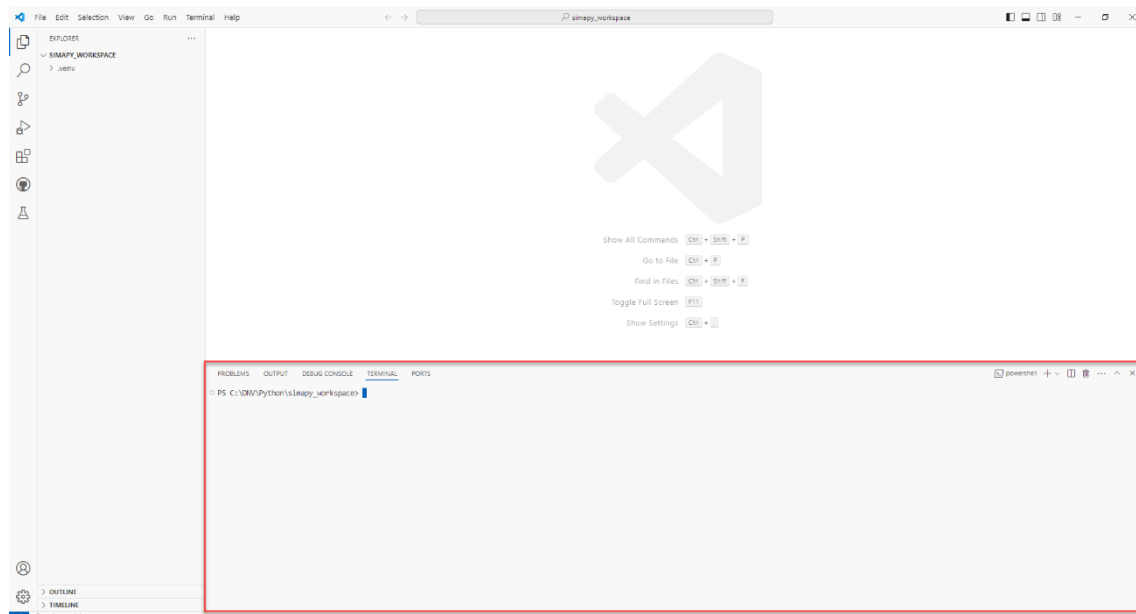
A `.venv` folder will be created in your project folder. This folder will contain and isolate the Python libraries for this project.



- 4. Click **Terminal > New Terminal** to create a new terminal.



A new terminal will be shown at the bottom of the VS Code screen.







(only if you are using the virtual environment) **Activate the virtual environment** by running the following command:

```
.\.venv\Scripts\Activate
```

You will see (.venv) at the start of the command line indicating the virtual environment is activated:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\DNV\Python\simapy_workspace> .\venv\Scripts\Activate
○ (.venv) PS C:\DNV\Python\simapy_workspace> █
```

5. Run the following terminal command to **install the latest SimaPy and its dependencies**:

```
pip install simapy
```

SimaPy library will be downloaded and installed:

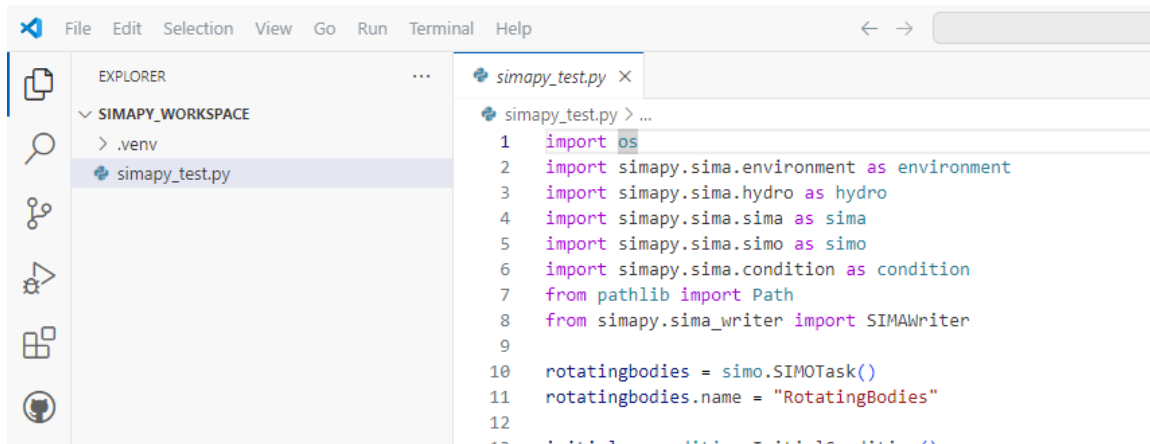
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] [ ] ... ^ x
● PS C:\DNV\Python\simapy_workspace> pip install simapy
Collecting simapy
  Downloading simapy-4.6.0-py3-none-any.whl.metadata (1.1 kB)
Collecting dmtpy==0.3.4 (from simapy)
  Downloading dmtpy-0.3.4-py3-none-any.whl.metadata (846 bytes)
Collecting numpy (from simapy)
  Using cached numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
Collecting h5py (from dmtpy==0.3.4->simapy)
  Downloading h5py-3.11.0-cp312-cp312-win_amd64.whl.metadata (2.5 kB)
Download simapy-4.6.0-py3-none-any.whl (2.0 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 2.8 MB/s eta 0:00:00
Download dmtpy-0.3.4-py3-none-any.whl (14 kB)
Using cached numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
Download h5py-3.11.0-cp312-cp312-win_amd64.whl (3.0 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 9.0 MB/s eta 0:00:00
Installing collected packages: numpy, h5py, dmtpy, simapy
Successfully installed dmtpy-0.3.4 h5py-3.11.0 numpy-1.26.4 simapy-4.6.0
○ PS C:\DNV\Python\simapy_workspace> █
```

### 3 TESTING PYTHON DEVELOPMENT ENVIRONMENT

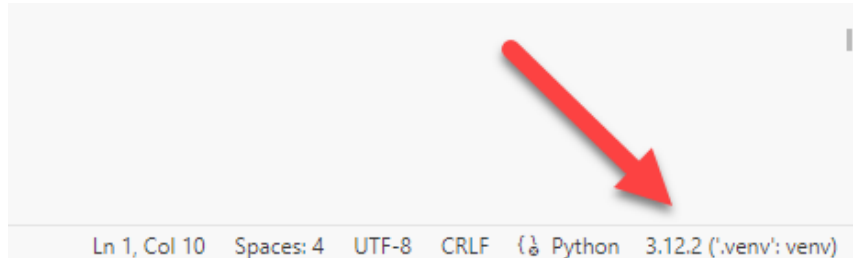
We will test if the Python development environment and SimaPy works correctly in this chapter.

To test if the installed Python development environment and the SimaPy library work correctly, follow these steps:

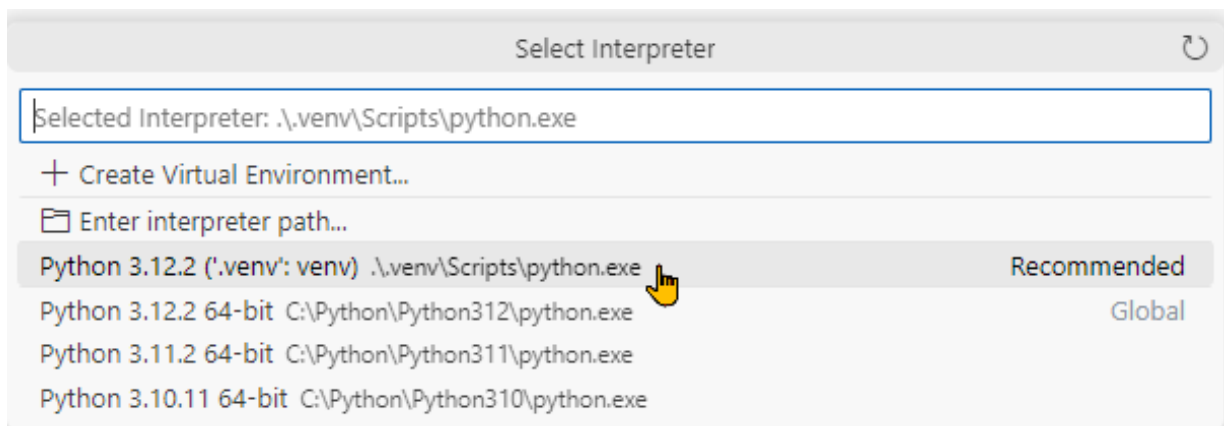
1. Copy **simapy\_test.py** file from the example input files to your project folder.
2. **Open the file** in VSCode. You will see a populated Python script. For now, ignore the contents.



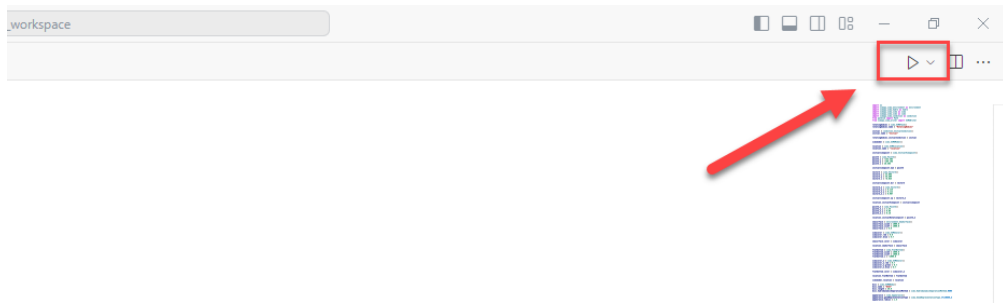
3. Verify if **Python 3.12.x** or **Python 3.12.x (.venv': venv)** (if you are using virtual environment) is selected at the bottom-right corner of the VSCode window.



If not, click it and select the **Python 3.12.x** interpreter path. Shown below is when using virtual environment.



4. Click the **Run Python File** icon at the top-right corner of the VSCode window to run the script.

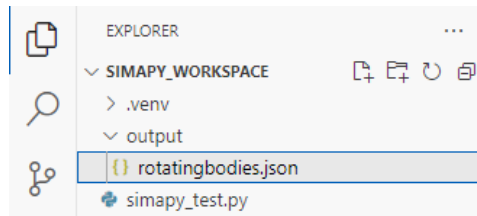


5. Once the process completes, “JSON file is written to output\rotatingbodies.json” will be printed.

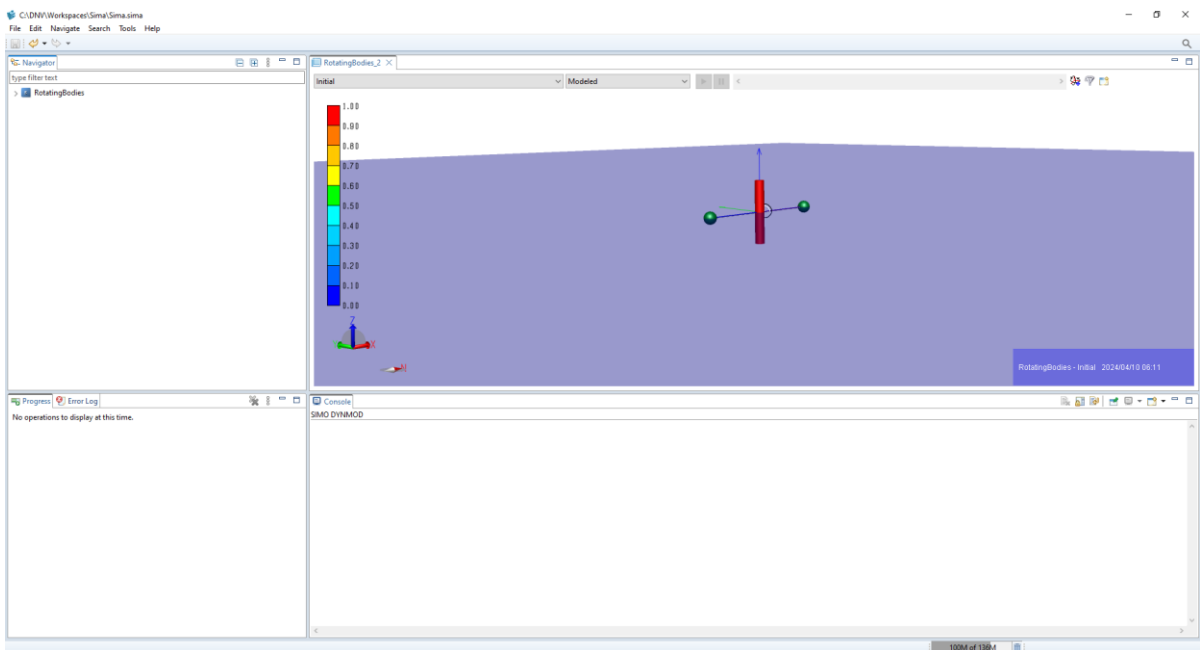
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS C:\DNV\Python\simapy_workspace> & c:/DNV/Python/simapy_workspace/.venv/Scripts/python.exe c:/DNV/Python/simapy_workspace/simapy_test.py
  JSON file outputted to output\rotatingbodies.json
○ PS C:\DNV\Python\simapy_workspace>
  
```

At the same time, **rotatingbodies.json** file is written to the output folder.



6. **Open Sima GUI**, import the **rotatingbodies.json** file. You will see a Simo task named *RotatingBodies* in the workspace.



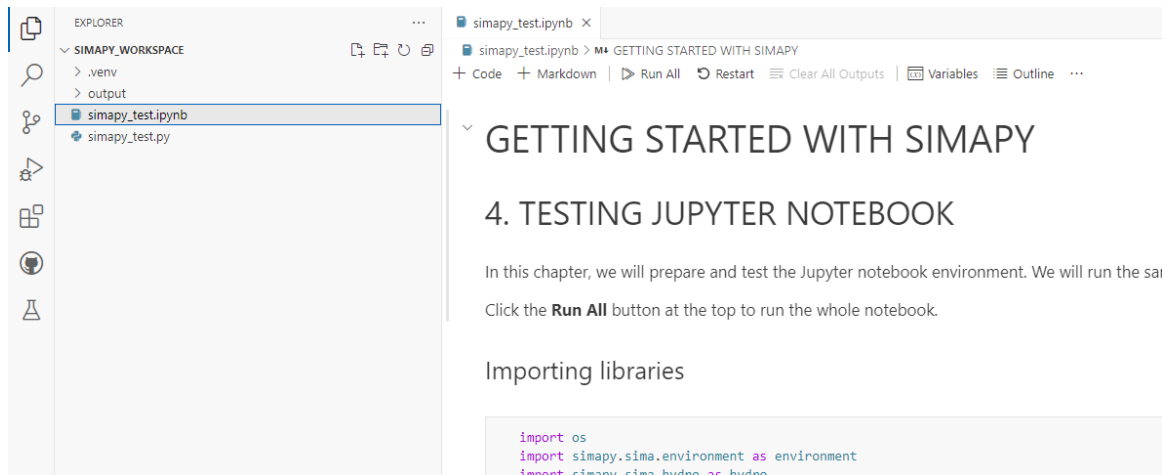
**Congratulations!** You have finished setting up the SimaPy development environment and ready to automate your Sima modelling using Python. To check if the Jupyter notebook also works correctly, go to the next chapter.

## 4 TESTING JUPYTER NOTEBOOK

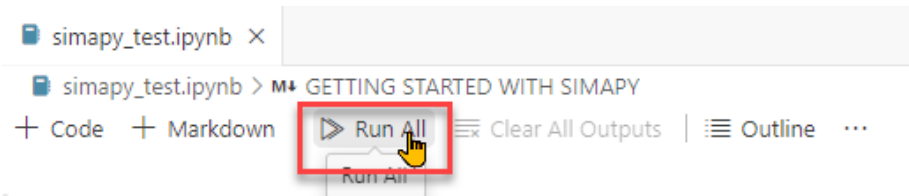
In this chapter, we will prepare and test the Jupyter notebook environment. We will run the same script we ran in the previous chapter with a Jupyter notebook and get another JSON file.

Follow these steps:

1. Copy **simapy\_test.ipynb** file from the example input files to your project folder.
2. **Open the file** in VSCode. You should see a populated Jupyter notebook. For now, ignore the contents.

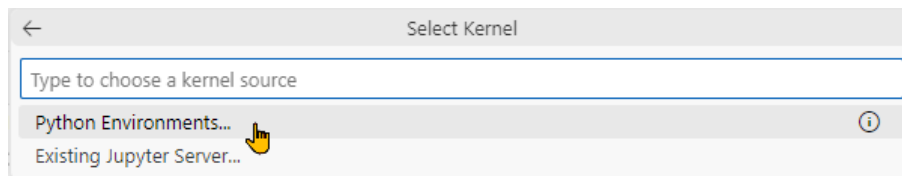


3. Click the **Run All** button at the top to run the whole notebook.

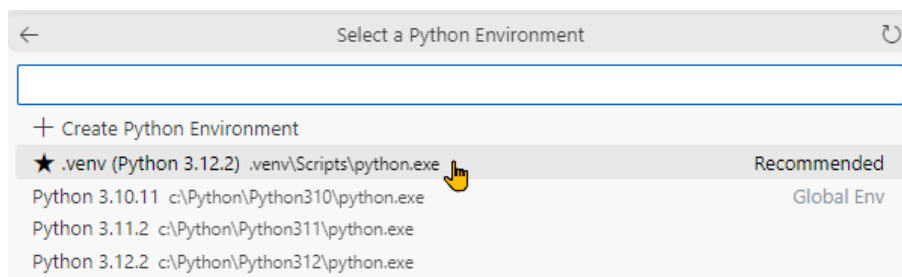


4. **If the Python environment is not yet selected**, you will be prompted to select one. Do the following:

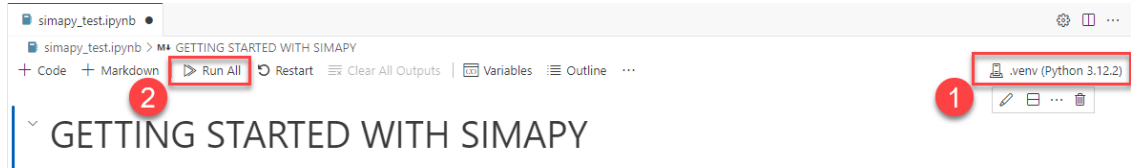
- **Select Python Environments.**



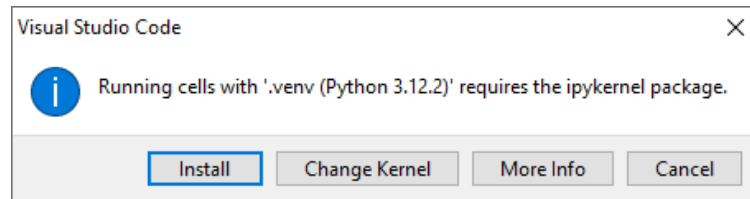
- If you are using a virtual environment, select **venv**. Otherwise, select any Python 3.12.x.



- Verify if the **venv** or **Python 3.12.x** is shown at the right side of the notebook, click **Run All** again.



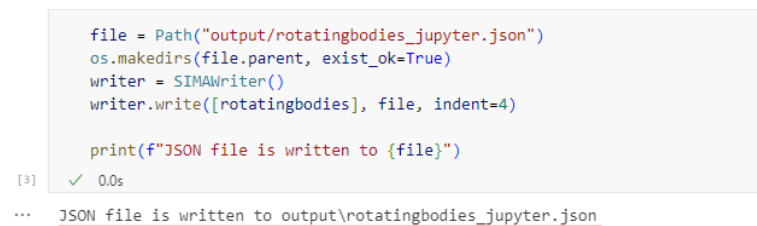
5. If **Jupyter kernel is not installed**, you will be prompted to install one. Click **Install**.



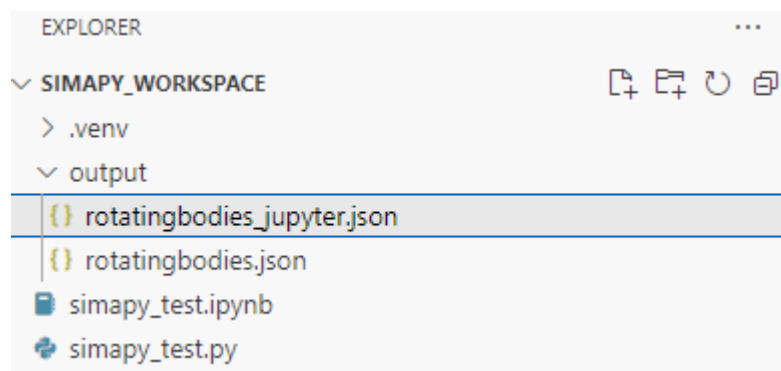
Wait for the Jupyter kernel to be installed and the notebook to be run.

6. **Scroll down to the end** of the notebook and see if the message “*JSON file is written to output\rotatingbodies\_jupyter.json*” should be printed.

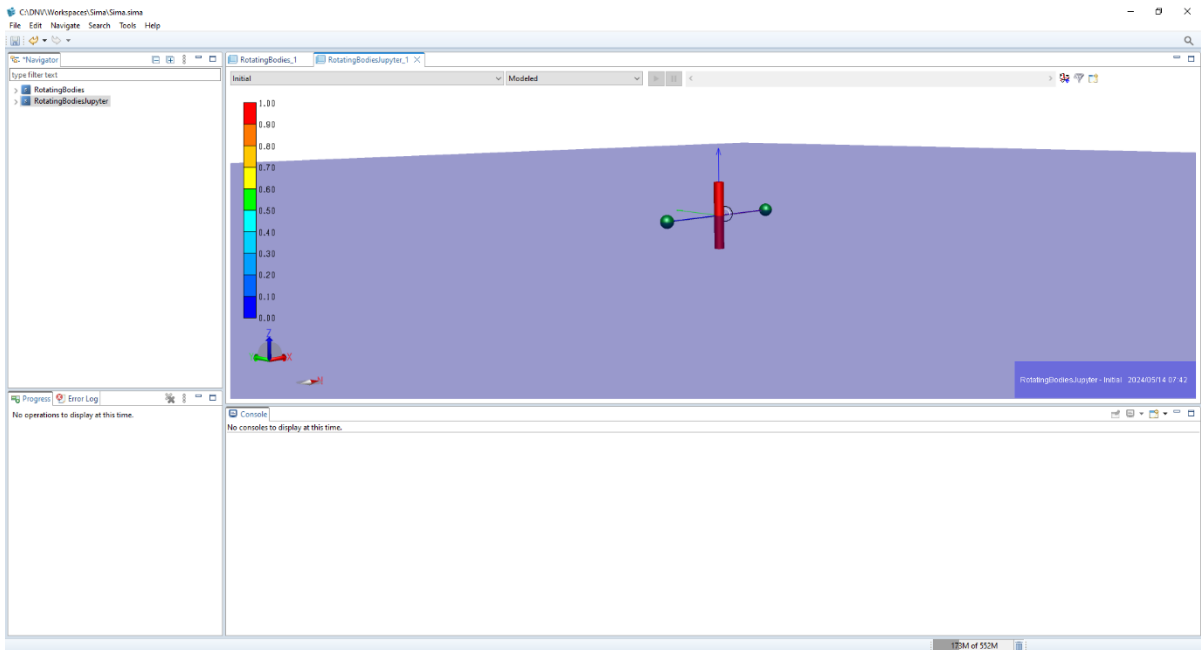
### Writing JSON file



Then, a **rotatingbodies\_jupyter.json** file is also written to the output folder.



7. **Open Sima GUI**, import the **rotatingbodies\_jupyter.json** file. You will see the Simo task *RotatingBodiesJupyter* in the workspace.



**Congratulations!** You have finished setting up the Jupyter notebook environment and now is ready to explore future SimaPy examples.

## 5 CREATING SIMA MODEL WITH SIMAPY

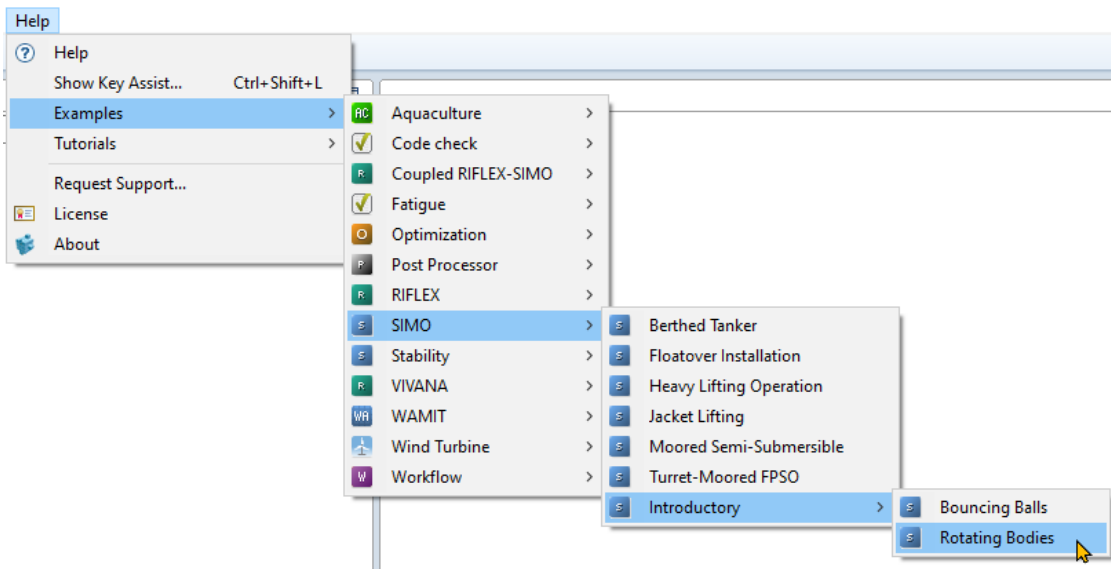
In this chapter, we will learn how to generate SimaPy codes with Sima and use it as a template to create a Sima model using SimaPy. At the end, we will import the created model into Sima.

### 5.1 Exporting an existing model as Python file in Sima

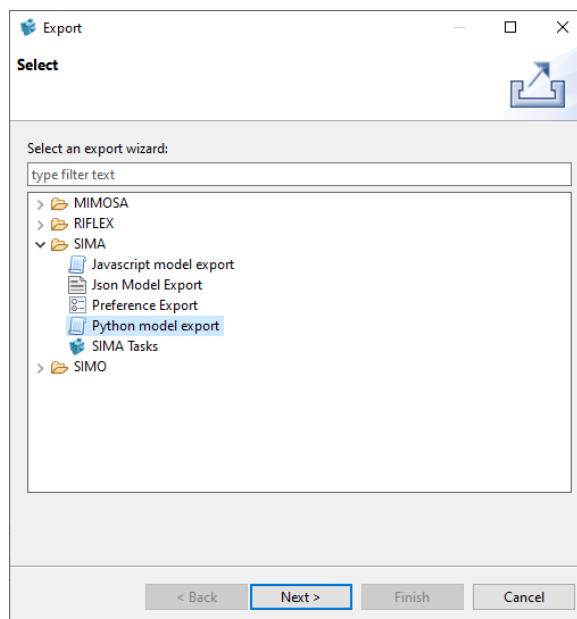
Sima can generate a SimaPy Python file for a model. This is mainly useful to learn and understand SimaPy.

Follow these steps to export a model as Python file:

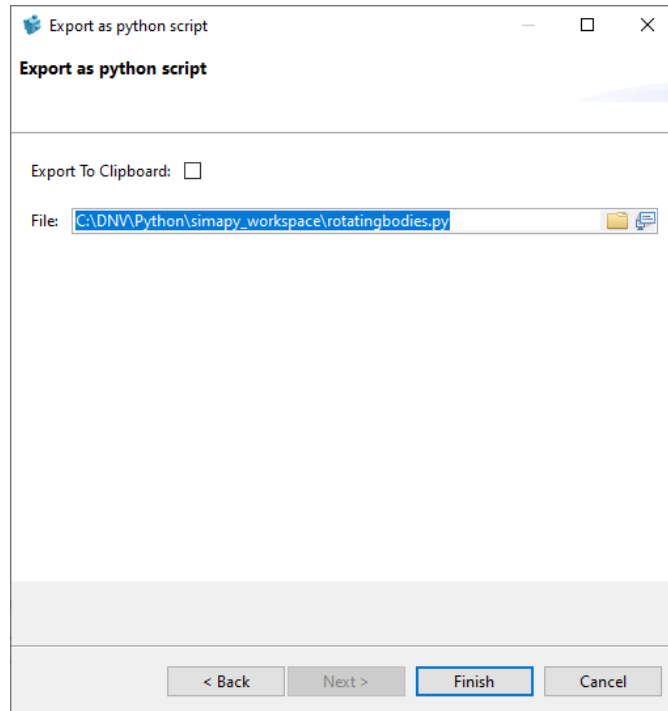
1. Open a new Sima workspace (the Sima GUI).
2. Create or import any Sima model. As an example, load *Rotating Bodies* examples by clicking **Help > Examples > SIMO > Introductory > Rotating Bodies**.



3. In the *Navigator*, right-click the **RotatingBodies** task and click **Export**. Then, select **SIMA > Python model export**.



- Click **Next** and specify the file's save location. Save it as **rotatingbodies.py** in your SimaPy project folder.



- Click **Finish** to save the Python file.
- In VSCode, open the **rotatingbodies.py** file and verify that SimaPy scripts has been generated. Note that sometimes the order of the lines might be different.

```

rotatingbodies.py x
rotatingbodies.py > ...
1 import simapy.sima.sima as sima
2 import simapy.sima.hydro as hydro
3 import simapy.sima.simo as simo
4 import simapy.sima.environment as environment
5 import simapy.sima.condition as condition
6 rotatingbodies = simo.SIMOTask()
7 rotatingbodies.name = "RotatingBodies"
8
9 initial = condition.InitialCondition()
10 initial.name = "Initial"
11
12 rotatingbodies.initialCondition = initial
13
14 simomodel = simo.SIMOModel()
15
16 location = simo.SIMOLocation()
17 location.name = "location"
18
19 initialviewpoint = sima.InitialViewpoint()
20
21 point3 = sima.Point3()
22 point3.x = -155.404
23 point3.y = -169.198
24 point3.z = 28.565
25
26 initialviewpoint.eye = point3
  
```

Next, we will look at the content of the Python file.



## 5.2 Checking the Contents of the Python File

We will use the `rotatingbodies.py` generated in the previous sub-chapter as our template to create another Sima model. However, before that, we will check the contents of the file to understand it better.

We will check each part of the Python file and compare it to the model in Sima GUI:

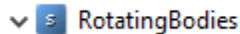
1. Importing SimaPy packages.

```
1 import simapy.sima.sima as sima
2 import simapy.sima.hydro as hydro
3 import simapy.sima.simo as simo
4 import simapy.sima.environment as environment
5 import simapy.sima.condition as condition
```

2. Defining an empty Sima Task (can be Simo, Reflex, etc.).

```
6 rotatingbodies = simo.SIMOTask()
7 rotatingbodies.name = "RotatingBodies"
```

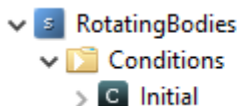
At this stage, the equivalent Sima model contains an empty *Task* named "RotatingBodies":



3. Defining an *InitialCondition* and appending it to the *Task*.

```
9 initial = condition.InitialCondition()
10 initial.name = "Initial"
11
12 rotatingbodies.initialCondition = initial
```

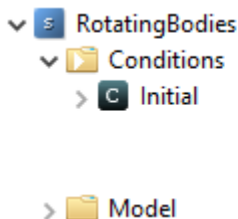
Now, we have an *Initial condition* in our Sima model:



4. Defining the *model folder* as a *Simo model folder*.

```
14 simomodel = simo.SIMOModel()
```

The *Model* folder is now present but not yet appended to the *Task*, it will be appended to the *Task* after all parameters have been defined.



We will set the *Task* aside and focus on the *Model* folder.

5. Defining an object (*Location*) and append it to the *Model*.

The sub-objects of the *Location* will be defined one by one:

```

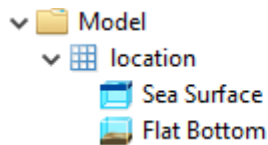
16 location = simo.SIMOLocation()
17 location.name = "location"
18
19 initialviewpoint = sima.InitialViewpoint()
20
21 point3 = sima.Point3()
22 point3.x = -155.404
23 point3.y = -169.198
24 point3.z = 28.565
...
42 location.initialViewpoint = initialviewpoint

```

Some other sub-objects such as *seaSurface* and *flatBottom* will be defined after this line. At the end, they will be appended to the *Model* folder:

```
78 simomodel.location = location
```

Now we have this equivalent Sima Model folder:



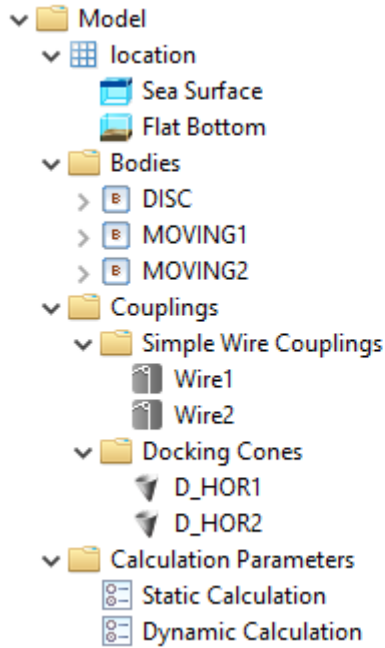
6. This will be repeated for all objects inside the *Model* folder.

Specifically:

- Line 80-166: *Simo body* named *DISC*, where it also includes:
  - Line 98-117: *Winch* mechanism named *WINCH\_1*.
  - Line 119-138: *Winch* mechanism named *WINCH\_2*.
- Line 168-222: *Simo body* named *MOVING1*.
- Line 224-278: *Simo body* named *MOVING2*.
- Line 280-288: *Simple wire coupling* named *Wire1*.
- Line 290-298: *Simple wire coupling* named *Wire2*.
- Line 300-354: *Docking cone* named *D\_HOR1*.
- Line 356-410: *Docking cone* named *D\_HOR2*.
- Line 412-418: *Simo static calculation parameters*.
- Line 420-429: *Simo dynamic calculation parameters*.



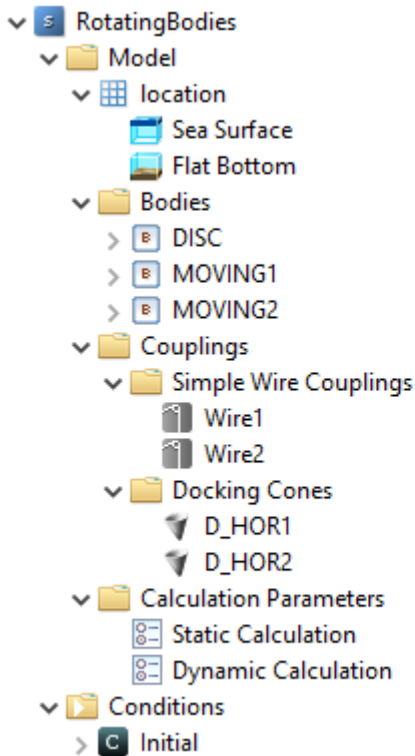
With this, our Sima Model folder has been populated as below:



7. In the last line, the *Model* folder is assigned to the *Task* defined in Step 2.

```
431 rotatingbodies.model = simomodel
```

Finally, our Sima model is complete:



However, as you may already noticed, we only have created the model in SimaPy without outputting anything. In the next sub-chapter, we will see how to write the model to a JSON file.

### 5.3 Writing the Model as JSON File from SimaPy

We will add a few lines to write the SimaPy model as a JSON file to be imported to Sima GUI using one of many features of SimaPy.

At the end of the Python file, add the following lines:

```
# Import additional packages
import os
from pathlib import Path
from simapy.sima_writer import SIMAWriter

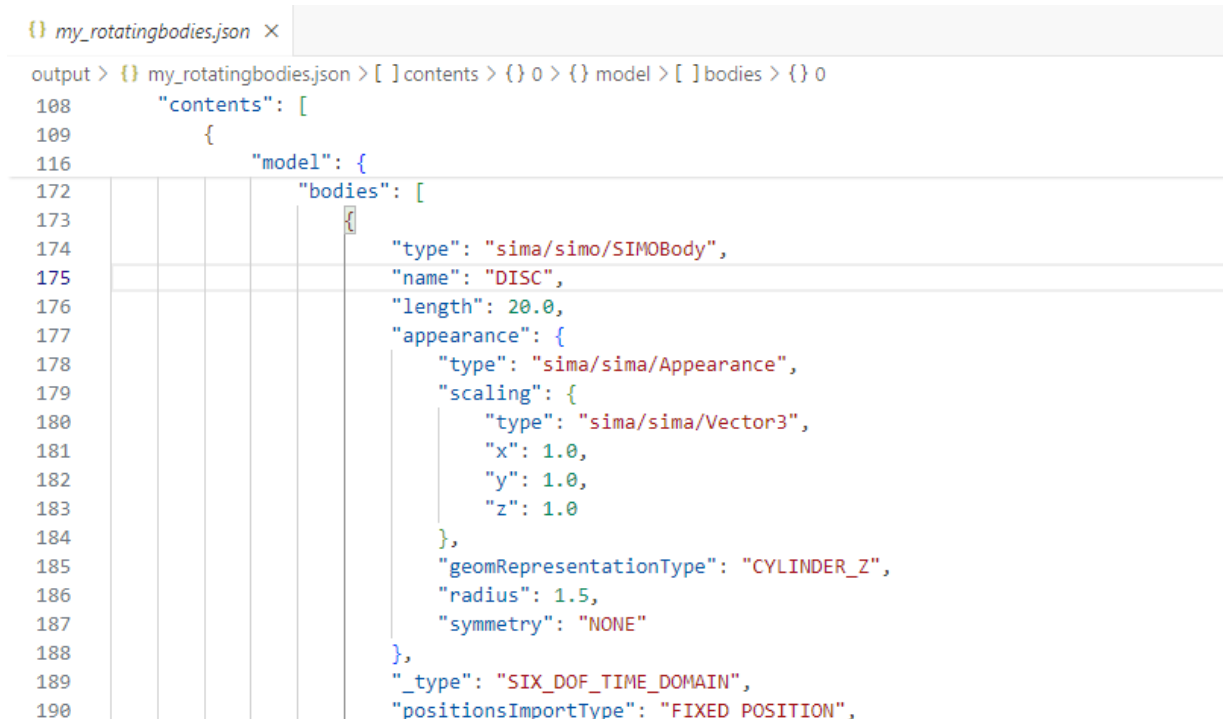
# Define output file path and file name
file = Path("output/my_rotatingbodies.json")

# Check if the folder is present, if not create the folder
os.makedirs(file.parent, exist_ok=True)

# Create a SIMAWriter instance and write the Task to the path above
writer = SIMAWriter()
writer.write([rotatingbodies], file, indent=4)

# Output to terminal indicating the location of the file
print(f"JSON file is written to {file}")
```

Run the Python script and a JSON file *my\_rotatingbodies.json* will be written into the *output* folder:



```
{ } my_rotatingbodies.json ×
output > { } my_rotatingbodies.json > [ ] contents > { } 0 > { } model > [ ] bodies > { } 0
108     "contents": [
109         {
116             "model": {
172                 "bodies": [
173                     {
174                         "type": "sima/simo/SIMOBody",
175                         "name": "DISC",
176                         "length": 20.0,
177                         "appearance": {
178                             "type": "sima/sima/Appearance",
179                             "scaling": {
180                                 "type": "sima/sima/Vector3",
181                                 "x": 1.0,
182                                 "y": 1.0,
183                                 "z": 1.0
184                             },
185                             "geomRepresentationType": "CYLINDER_Z",
186                             "radius": 1.5,
187                             "symmetry": "NONE"
188                         },
189                         "_type": "SIX_DOF_TIME_DOMAIN",
190                         "positionsImportType": "FIXED_POSITION",
```

This JSON file contains the same model as the original *Rotating Bodies* example. You can try to import it into Sima GUI and see it by yourself. Now you understand the content of the *simapy\_test.py* file we used in the previous chapter.

**Note:** Just remember this logic when working with SimaPy:

1. Create an object.
2. Define the parameters of the object.
3. Append it to a higher-level object.

In the next sub-chapter, we will use the power of Python scripting to modify the Sima model iteratively.

## 5.4 Using Python Scripting to Modify the Model

We will now use some Python scripting like for-loop or string manipulation to automate the modification of our model.

Firstly, copy the **rotatingbodies\_mod.py** file from the workshop input folder to your SimaPy project folder and open it. This file is a slightly modified version of the *rotatingbodies.py* from the previous sub-chapter.

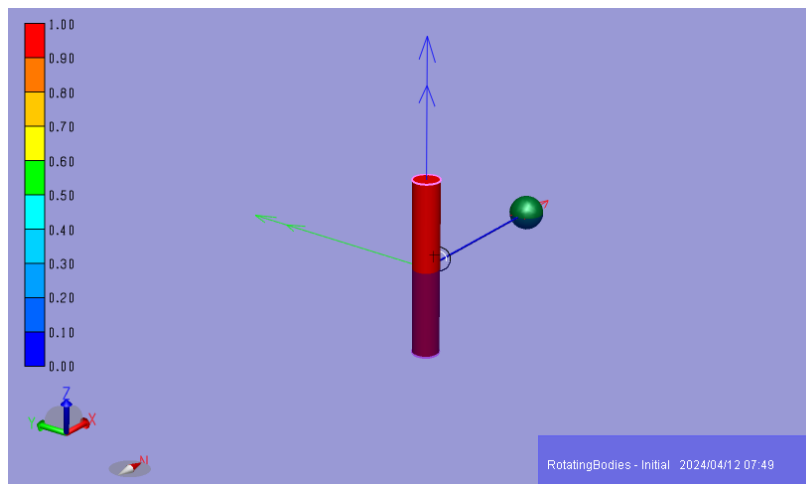
```

rotatingbodies_mod.py X
rotatingbodies_mod.py > ...
1  import numpy as np
2  import os
3  import simapy.sima.sima as sima
4  import simapy.sima.hydro as hydro
5  import simapy.sima.simo as simo
6  import simapy.sima.environment as environment
7  import simapy.sima.condition as condition
8  from pathlib import Path
9  from simapy.sima_writer import SIMAWriter
10
11 rotatingbodies = simo.SIMOTask()
12 rotatingbodies.name = f"RotatingBodies" # This will be changed
13
14 initial = condition.InitialCondition()
15 initial.name = "Initial"
16

```

There are mainly two changes:

- The second rotating body (*MOVING2*) and its associated objects were removed as shown below:



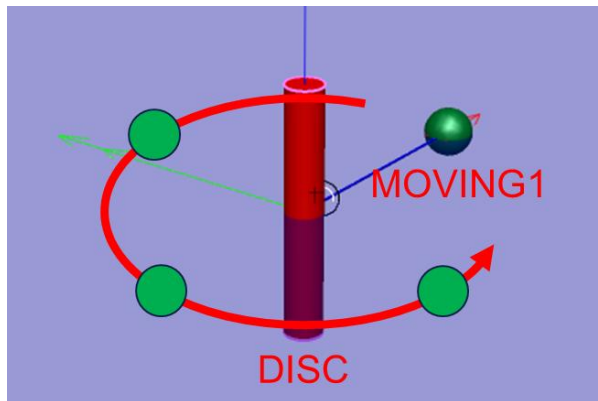
- A line to *import numpy* was added and other *import* lines were moved to the start of the file. It is considered a good practice to import the packages at the beginning.

```

1 import os
2 from pathlib import Path
3 import numpy as np
4 import simapy.sima.condition as condition
5 import simapy.sima.environment as environment
6 import simapy.sima.hydro as hydro
7 import simapy.sima.sima as sima
8 import simapy.sima.simo as simo
9 from simapy.sima_writer import SIMAWriter

```

We will copy the body *MOVING1* and copy it iteratively around the *DISC* body **N** times:



To do so, we will add some lines to Python script. You can add them into the following part of the Python script:

```

295 # Additions start here
296
297 # Additions end here

```

Add the following:

1. Define the final number of bodies, change the name of the Task, and pre-calculate the direction change's step size. For now, let's try number of bodies equals 4.

```

bodies_num = 4
dir_step = 360.0/bodies_num
rotatingbodies.name = f"RotatingBodies_{bodies_num}"

```

2. Start a for-loop.

```
for i in np.arange(1,bodies_num):
```

3. Calculate the current direction of the body number *i*. Note there is an indentation.

```

    # Set the direction of the assembly
    current_dir = dir_step*i # degrees
    current_dir = np.deg2rad(current_dir)

```

4. Copy the body *MOVING1* to a Python variable, *moving*.

```

    # Copy moving1 as base
    moving = moving1.copy()

```

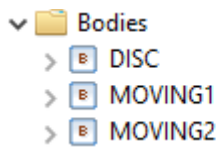
**Note: It is important to pass a copy of the object using `copy()` method** because SimaPy only pass the reference to the object if it is not used.

A consequence of passing just the reference to an object is that the original object will be modified when you modify the copied object. See the following two code snippets:

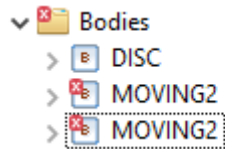
```
# Case A
moving = moving1.copy()
moving.name = "MOVING2"

# Case B
moving = moving1
moving.name = "MOVING2"
```

Case A will result in the following Sima model:



Meanwhile, case B will result in the following Sima model:



In case B, the name of the *MOVING1* (original object) was changed to *MOVING2* too, which is not our intention.

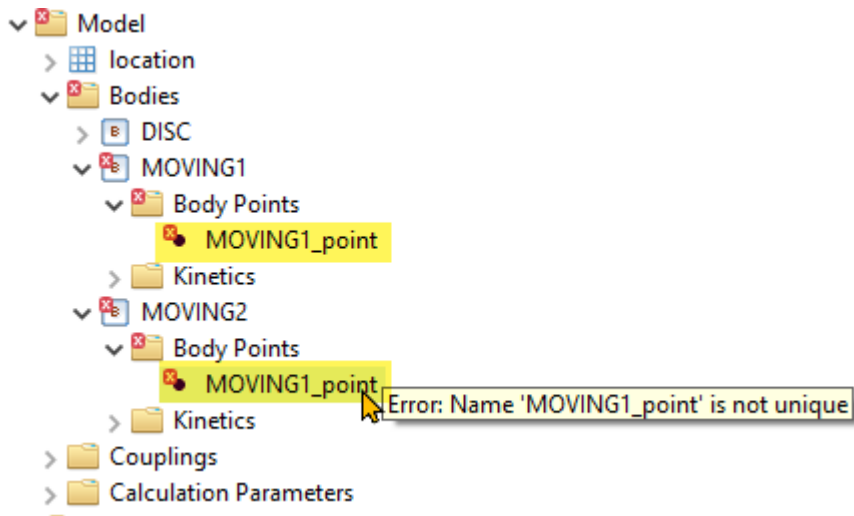
5. Change some parameters of the copied body, *moving*.

```
# Change some parameters of "moving"
moving.name = f"MOVING{i+1}"
moving.initialPosition.x = np.cos(current_dir) * 20.0
moving.initialPosition.y = np.sin(current_dir) * 20.0

moving_point = moving.bodyPoints[0]
moving_point.name = f"MOVING{i+1}_point"
```

**Note: The name of the attached body point needs to be changed.** This is because in Sima, these body points must have unique names as shown below:

# DNV



- Append the copied body to the model folder.

```
# Append it to the model folder
simomodel.bodies.append(moving)
```

Otherwise, our newly created body is not attached to the model folder.

- Copy the body point and the attached winch of the DISC body because we need to attach our wire later to it. Verify if it is appended to the DISC body.

```
# Copy the body point and the attached winch of the DISC body,
# change the names, and append it to DISC body.
discpoint = discpoint_1.copy()
discpoint.name = f"DISCpoint_{i+1}"
discpoint.winch.name = f"WINCH_{i+1}"
disc.bodyPoints.append(discpoint)
```

- Copy Wire1, change the name and end points (attach it to the newly created body MOVING2 and DISC), and append it to the model folder.

```
# Copy Wire1, change the name and endPoints,
# and finally append it to the model folder
wire = wire1.copy()
wire.name = f"Wire{i+1}"
wire.endPoint1 = discpoint
wire.endPoint2 = moving_point
simomodel.simpleWireCouplings.append(wire)
```

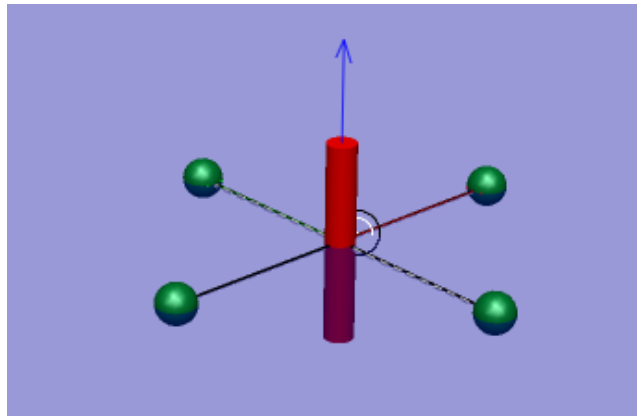
- Lastly, copy the docking cone D\_HOR1.



```
# Copy docking cone, change the name, direction, and associated bodies,  
# then append it to the model folder  
d_hor = d_hor1.copy()  
d_hor.name = f"D_HOR{i+1}"  
d_hor.dockingConeDirectionVector.x = np.cos(current_dir)  
d_hor.dockingConeDirectionVector.y = np.sin(current_dir)  
d_hor.dockingPinBody = moving  
d_hor.dockingConeBody = disc  
simomodel.dockingCones.append(d_hor)
```

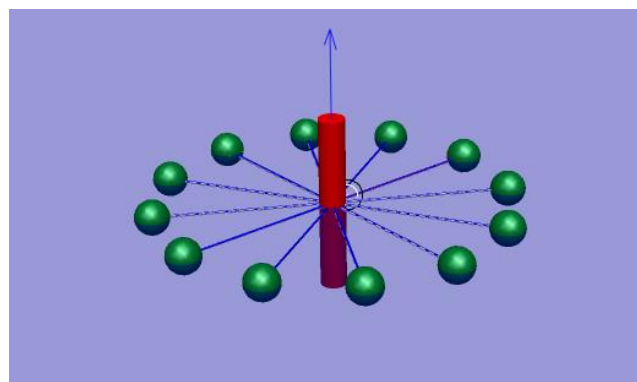
You can find the finished script in the file *rotatingbodies\_mod\_done.py*. Please compare your file with it.

10. Run the Python code to output the JSON file. Then, try to import it into Sima GUI.



Now we have 4 bodies in our model.

11. Change the value of `bodies_num` variable to a larger number and see what happens.



**Note:** Notice how we do not need to re-attach the model folder to the Task. This is because once we attach an object, no reattachment is needed.

**Congratulations!** Now you can leverage Python scripting to modify your SimaPy model.



## **About DNV**

DNV is an independent assurance and risk management provider, operating in more than 100 countries. Through its broad experience and deep expertise DNV advances safety and sustainable performance, sets industry standards, and inspires and invents solutions.

## **Digital Solutions**

DNV is a world-leading provider of digital solutions and software applications with focus on the energy, maritime and healthcare markets. Our solutions are used worldwide to manage risk and performance for wind turbines, electric grids, pipelines, processing plants, offshore structures, ships, and more. Supported by our domain knowledge and Veracity assurance platform, we enable companies to digitize and manage business critical activities in a sustainable, cost-efficient, safe and secure way.