Sufficient answers to EXUDYN questions

	EXUDYN General
1	-Exudyn is a C++ based Python library for efficient simulation of flexible multibody dynamics systems. -It is the follow up code of the previously developed multibody code HOTINT, which Johannes Gerstmayr started during his PhD-thesis. -Exudyn is particularly useful for setting up complex multibody models which may include rigid and flexible bodies, joints, loads, and other components. ### Application (there is a broad variation of answers to this question , so will depend on the model) -Due to its Python integration, Exudyn can be coupled with other environments and tools such as optimization frameworks, statistical tools, data analysis platforms, machine learning, and more. This makes it highly versatile for interdisciplinary research and applications. -Exudyn is adept at simulating complex systems that involve interactions between multiple bodies and forces, making it suitable for engineering, biomechanics, robotics, and vehicle dynamics, among other fields. -It serves as a powerful tool for educational purposes ….
2	## (The answer should at least name Dr. Johannes Gerstmayr) - Exudyn was developed primarily by Johannes Gerstmayr at the University of Innsbruck.
3	-Rigid and Flexible Bodies -Joints and Constraints -Loads and Forces Etc.
4	- It was developed using C++ - It utilizes Python for user interference
5	##prerequisits - It is recomended to use conda environment (better managment of dependencies) - Install Python on system! - (Optional but recommended) Create a new conda environment (!codesample !) ##installation - Install Required Libraries: numpy matplotlib scipy … - Pip install exudyn (Clone git-hub repo could also be an answer)
6	This is a bit of a trick question because there are no specific system requirements mentioned in the docs. But there are tested systems in the docs like e.g Mac OS 11.x ‘Big Sur’, Mac Mini (2021), Apple M1, 16GB Memory… Also good answer additionally would be: - Exudyn is compatible with Windows, Linux, and MacOS. - Exudyn supports Python versions from 3.8 to 3.12.

	EXUDYN General
7	-Nodes -Markers -Sensors -Loads -Objects
8	-Static analysis (settings accelerations and velocities to zero,Outputs may be displacements, rotations, stresses, joint or support forces, etc) -Dynamic analysis (initial conditions and using time integration) -Sensitivity analysis (Investigate the influence of small variations of the parameters (dimensions, material) … this are just some possible answers but sufficient (for me)

	EXUDYN General
9	To set up a multibody system simulation in Exudyn, including defining bodies, joints, and forces, follow these steps: 1. **Imports and initialize the system:** import exudyn as exu from exudyn.utilities import * import numpy as np - Create a `SystemContainer` and add a `MainSystem` to it. SC = exu.SystemContainer() mbs = SC.AddSystem() 2. **Define Parameters:** - Set the necessary parameters like mass, spring constant, damping coefficient, and external force. mass = 1.6 spring = 4000 damper = 8 force = 80 3. **Add Nodes:** - Add nodes to the system. Nodes provide the coordinates and degrees of freedom for the simulation. A ground node and a mass point node are typical examples. n1 = mbs.AddNode(Point(referenceCoordinates=[L, 0, 0], initialCoordinates=[u0, 0, 0], initialVelocities=[v0, 0, 0])) nGround = mbs.AddNode(NodePointGround(referenceCoordinates=[0, 0, 0]) 4. **Add Bodies:** - Define and add bodies to the system. Each body typically needs a mass and a node to which it is attached. massPoint = mbs.AddObject(MassPoint(physicsMass=mass, nodeNumber=n1)) 6. **Add Joints and Constraints:** - Define joints and constraints to connect different parts of the system. For instance, a spring-damper can be added between two markers. mbs.AddObject(CoordinateSpringDamper(markerNumbers=[groundMarker, nodeMarker], stiffness=spring, damping=damper)) 7. **Add Forces:** - Apply external forces to the system. Forces are typically applied to markers. mbs.AddLoad(LoadCoordinate(markerNumber=nodeMarker, load=force)) 8. **Assemble the System:** - Assemble the system to link all components and prepare it for simulation. mbs.Assemble() 9. **Simulation Settings:** - Set up the simulation parameters, such as the time span and step size. tEnd = 1 h = 0.001 simulationSettings = exu.SimulationSettings() simulationSettings.timeIntegration.numberOfSteps = int(tEnd / h) simulationSettings.timeIntegration.endTime = tEnd 10. **Run the Simulation:** - Execute the simulation with the defined settings exu.SolveDynamic(mbs, simulationSettings)

	EXUDYN General
10	-Newton’s basic principles -The Lagrange-d’Alembert principle -Generalized Principle of Virtual Work -Virtual displacements
11	Expecting a general answer that includes: - necessary imports - SC = exu.SystemContainer() - mbs = SC.AddSystem() - Some additional information about the Environmental settings. - Parameters for simulation - Parameters for visualization This question should more or less test the information span of the model and is very general, which makes it harder to answer
12	Exudyn:SimulationSettings:TimeIntegrationSettings General parameters used in time integration; specific parameters are provided in the according solver settings, e.g. for generalizedAlpha.
13	Description of parameter initialStepSize in the Exudyn structure TimeIntegrationSettings *** h_{init}: if automaticStepSize=True, initial step size; if initialStepSize==0, max. stepSize, which is (endTime-startTime)/numberOfSteps, is used as initial guess; a good choice of initialStepSize may help the solver to start up faster. initialStepSize has the Python type float and the default value "0". The parameter initialStepSize is located at: simulationSettings The parameter initialStepSize can be accessed directly using: simulationSettings.initialStepSize Description of parameter minimumStepSize in the Exudyn structure TimeIntegrationSettings *** h_{min}: if automaticStepSize=True or adaptiveStep=True: lower limit of time step size, before integrator stops with adaptiveStep; lower limit of automaticStepSize control (continues but raises warning). minimumStepSize has the Python type float and the default value "1e-8". The parameter minimumStepSize can be accessed directly using: simulationSettings.minimumStepSize

	EXUDYN General
14	simulationSettings.timeIntegration.newton (many options depending on the need) - Setting of tolerances (absolute, relative) - useModifiedNewton(bool) - maxModifiedIterations - numericalDifferentiation.addReferenceCoordinatesToEpsilon - numericalDifferentiation.minimumCoordinateSize - numericalDifferentiation.relativeEpsilon - modifiedNewtonContractivity
15	Description of parameter recordImagesInterval in the Exudyn structure SolutionSettings *** record frames (images) during solving: amount of time to wait until next image (frame) is recorded; set recordImages = -1. if no images shall be recorded; set, e.g., recordImages = 0.01 to record an image every 10 milliseconds (requires that the time steps / load steps are sufficiently small!); for file names, etc., see VisualizationSettings.exportImages. recordImagesInterval has the Python type float and the default value "-1.". The parameter recordImagesInterval is located at: simulationSettings The parameter recordImagesInterval can be accessed directly using: simulationSettings.recordImagesInterval
16	*** Description of parameter sensorsStoreAndWriteFiles in the Exudyn structure SolutionSettings flag (true/false); if false, no sensor files will be created and no sensor data will be stored; this may be advantageous for benchmarking as well as for special solvers which should not overwrite existing results (e.g. ComputeODE2Eigenvalues); settings this value to False may cause problems if sensors are required to perform operations which are needed e.g. in UserSensors as input of loads, etc. sensorsStoreAndWriteFiles has the Python type bool and the default value "true". The parameter sensorsStoreAndWriteFiles is located at: simulationSettings The parameter sensorsStoreAndWriteFiles can be accessed directly using: simulationSettings.sensorsStoreAndWriteFiles flag (true/false); if true, sensor output is appended to existing file (otherwise created) or in case of internal storage, it is appended to existing currently stored data; this allows storing sensor values over different simulations sensorsStoreAndWriteFiles [type = bool, default = True]: simulationSettings.solutionSettings.sensorsStoreAndWriteFiles

	EXUDYN General
17	Description of parameter sensorsAppendToFile in the Exudyn structure SolutionSettings *** flag (true/false); if true, sensor output is appended to existing file (otherwise created) or in case of internal storage, it is appended to existing currently stored data; this allows storing sensor values over different simulations. sensorsAppendToFile has the Python type bool and the default value "false". The parameter sensorsAppendToFile is located at: simulationSettings sensorsAppendToFile [type = bool, default = False]: simulationSettings.solutionSettings.sensorsAppendToFile
18	*** Description of parameter maxIterations in the Exudyn structure NewtonSettings *** maximum number of iterations (including modified + restart Newton iterations); after that total number of iterations, the static/dynamic solver refines the step size or stops with an error. maxIterations has the Python type int and the default value "25". The parameter maxIterations is located at: simulationSettings.timeIntegration or simulationSettings.staticSolver
19	*** Description of parameter adaptiveStep in the Exudyn structure TimeIntegrationSettings *** True: the step size may be reduced if step fails; no automatic stepsize control. adaptiveStep has the Python type bool and the default value "true". The parameter adaptiveStep is located at: simulationSettings The parameter adaptiveStep can be accessed directly using: simulationSettings.adaptiveStep

	EXUDYN General
20	Description of parameter jacobianConnectorDerivative in the Exudyn structure NumericalDifferentiationSettings *** True: for analytic Jacobians of connectors, the Jacobian derivative is computed, causing additional CPU costs and not beeing available for all connectors or markers (thus switching to numerical differentiation); False: Jacobian derivative is neglected in analytic Jacobians (but included in numerical Jacobians), which often has only minor influence on convergence. jacobianConnectorDerivative has the Python type bool and the default value "true". The parameter jacobianConnectorDerivative is located at: simulationSettings.timeIntegration.newton or simulationSettings.staticSolver.newton The parameter jacobianConnectorDerivative can be accessed directly using: simulationSettings.timeIntegration.newton.jacobianConnectorDerivative or simulationSettings.staticSolver.newton.jacobianConnectorDerivative simulationSettings.timeIntegration.newton.numericalDifferentiation.jacobianConnectorDerivative = False
21	Description of parameter writeInitialValues in the Exudyn structure SolutionSettings *** flag (true/false); if true, initial values are exported for the start time; applies to coordinatesSolution and sensor files; this may not be wanted in the append file mode if the initial values are identical to the final values of a previous computation. writeInitialValues has the Python type bool and the default value "true". The parameter writeInitialValues is located at: simulationSettings The parameter writeInitialValues can be accessed directly using: simulationSettings.writeInitialValues
22	Description of parameter pauseAfterEachStep in the Exudyn structure SimulationSettings *** pause after every time step or static load step(user press SPACE). pauseAfterEachStep has the Python type bool and the default value „false". pauseAfterEachStep [type = bool, default = False]: .simulationSettings.pauseAfterEachStep pause after every time step or static load step(user press SPACE)

	EXUDYN General
23	*** Description of parameter taskSplitTasksPerThread in the Exudyn structure Parallel *** this is the number of subtasks that every thread receives; minimum is 1, the maximum should not be larger than 100; this factor is 1 as long as the taskSplitMinItems is not reached; flag is copied into MainSystem internal flag at InitializeSolverData(...). taskSplitTasksPerThread has the Python type int and the default value "16". The parameter taskSplitTasksPerThread is located at: simulationSettings The parameter taskSplitTasksPerThread can be accessed directly using: simulationSettings.taskSplitTasksPerThread taskSplitTasksPerThread [type = PInt, default = 16]: simulationSettings.parallel.taskSplitTasksPerThread this is the number of subtasks that every thread receives; minimum is 1, the maximum should not be larger than 100; this factor is 1 as long as the taskSplitMinItems is not reached; flag is copied into MainSystem internal flag at InitializeSolverData(...) There is no precise answer to what happens when taskSplit… exeeds 100. But based on the coontext I would expect performance issues.
24	*** Description of parameter showNumbers in the Exudyn structure VSettingsConnectors *** flag to decide, whether the connector(=object) number is shown. showNumbers has the Python type bool and the default value "false". The parameter showNumbers is located at: SC.visualizationSettings The parameter showNumbers can be accessed directly using: SC.visualizationSettings.showNumbers
25	*** Description of parameter showCurrent in the Exudyn structure VSettingsSensorTraces *** show current trace position (and especially vector quantity) related to current visualization state; this only works in solution viewer if sensor values are stored at time grid points of the solution file (up to a precision of 1e-10) and may therefore be temporarily unavailable. showCurrent has the Python type bool and the default value "true". The parameter showCurrent is located at: SC.visualizationSettings The parameter showCurrent can be accessed directly using: SC.visualizationSettings.showCurrent

	EXUDYN General
26	*** Description of parameter exportAlgebraicCoordinates in the Exudyn structure SolutionSettings *** add algebraicCoordinates (=Lagrange multipliers) to solution file (coordinatesSolutionFile). exportAlgebraicCoordinates has the Python type bool and the default value "true". The parameter exportAlgebraicCoordinates is located at: simulationSettings The parameter exportAlgebraicCoordinates can be accessed directly using: simulationSettings.exportAlgebraicCoordinates
27	More than one possibilitiy: mbs.AddObject({'objectType': 'MassPoint', 'physicsMass': 2.5, 'nodeNumber': n3}) #, 'graphicsData': [graphics1]}) ## create mass point massPoint = mbs.CreateMassPoint(referencePosition=[L,0,0], initialDisplacement=[u0,0,0], initialVelocity=[v0,0,0], physicsMass=mass)
28	#add an load with user function: def UFforce(mbs, t, loadVector): #define time-dependent function: return [10+5*np.sin(t*10*2*pi),0,0] mbs.CreateForce(bodyNumber=b0, localPosition=[-0.5,0,0], loadVector=[10,0,0], loadVectorUserFunction=UFforce,) #load is 10N in x-direction
29	mbs = SC.AddSystem() #add a new system to work with

	EXUDYN General

30	#create spring damper between bodies (using local position) or between nodes
	#spring-damper may not have size 0; spring reference length is computed from reference configuration
	oSD = mbs.CreateSpringDamper(bodyOrNodeList=[oGround, m2],
	localPosition0=[6,0,0],
	localPosition1=[0,0,0],
	stiffness=1e3, damping=1e1,
	drawSize=0.2)
	
	#alternatively, we can use a CartesianSpringDamper; has spring and damper coefficients as list of x/y/z components
	#it has no reference length and acts on the coordinates of both objects:
	oCSD = mbs.CreateCartesianSpringDamper(bodyOrNodeList=[oGround, m2],
	localPosition0=[7,2,0],
	localPosition1=[0,0,0],
	stiffness=[20,0,1e4], #stiffness in x/y/z direction
	damping=[0.1,0,10],
	drawSize=0.2)
	
	#also a possibility but needs a lot more context for quality integration within system..
	(example of SpringDampers and CoordinateConstraints)
	mbs.AddObject(SpringDamper(markerNumbers = [mN0, mN1], stiffness = 1000, referenceLength=L))
	mbs.AddLoad(Force(markerNumber=mN1, loadVector=[10,0,0]))

31	oRB = mbs.CreateRigidBody(inertia=inertia,
	referencePosition=p0,
	referenceRotationMatrix=Alist[i],
	gravity=g,
	graphicsDataList=[graphicsBody])
	
	b0 = mbs.CreateRigidBody(inertia = inertiaCube,
	referencePosition = [0.5,0,0], #reference position x/y/z of COM
	referenceRotationMatrix=RotationVector2RotationMatrix([0,0,pi*0.5]),
	initialAngularVelocity=[2,0,0],
	initialVelocity=[0,4,0],
	gravity = [0,-9.81,0],
	graphicsDataList = [graphicsCube])

	EXUDYN General
32	spherical joint between two bodies; definition of joint position in global coordinates (alternatively in body0 local coordinates) for reference configuration of bodies; all markers are automatically computed Example: mbs.CreateSphericalJoint(bodyNumbers=[oGround, b0], position=[5.5,0,0], useGlobalFrame=True, jointRadius=0.06) #alternatively, using markers and objects: # mRigid11 = mbs.AddMarker(MarkerBodyRigid(bodyNumber = oRB1, localPosition = [0.5*L1,0,0])) #connection to piston # mPiston = mbs.AddMarker(MarkerBodyRigid(bodyNumber = oPiston1, localPosition = [0,0,0])) # mbs.AddObject(SphericalJoint(markerNumbers=[mRigid11,mPiston], # constrainedAxes=[1,1,0], # visualization=VObjectJointSpherical(jointRadius=1.5*a)))

	EXUDYN General

33	import exudyn as exu
	from exudyn.utilities import *
	
	SC = exu.SystemContainer()
	mbs = SC.AddSystem()
	
	#++++++++++++++++++++++++++++++++++
	
	oGround = mbs.CreateGround() #[0,0,0]
	
	oMass = mbs.CreateMassPoint(name='Mass',
	referencePosition=[1,0,0],
	physicsMass=4,
	gravity=[0,-9.81,0],
	drawSize=0.2,
	color=color4blue)
	
	mbs.CreateDistanceConstraint(bodyOrNodeList=[oGround, oMass],
	distance=1, #'none' will compute distance automatically
	drawSize=0.06)
	
	mbs.Assemble()
	
	tEnd = 5
	stepSize = 0.02 #smaller gives more accurate results
	
	# add simulation settings
	simulationSettings = exu.SimulationSettings()
	simulationSettings.timeIntegration.verboseMode = 1
	simulationSettings.timeIntegration.endTime = tEnd
	simulationSettings.timeIntegration.numberOfSteps = int(tEnd/stepSize)
	
	# start solver
	mbs.SolveDynamic(simulationSettings=simulationSettings)
	
	mbs.SolutionViewer()

	EXUDYN General

34	import exudyn as exu
	from exudyn.utilities import *
	
	SC = exu.SystemContainer()
	mbs = SC.AddSystem()
	
	#++++++++++++++++++++++++++++++++
	
	oGround = mbs.CreateGround() #[0,0,0]
	
	oMass = mbs.CreateMassPoint(name='HeavyMass',
	referencePosition=[0,0,0],
	physicsMass=4,
	drawSize=0.2,
	color=color4red)
	
	oSD = mbs.CreateSpringDamper(bodyList=[oGround, oMass],
	stiffness=2500,
	damping=25,
	drawSize=0.1)
	
	#alternatively, we can use a CartesianSpringDamper; has spring and damper coefficients as list of x/y/z components
	#it has no reference length and acts on the coordinates of both objects:
	oCSD = mbs.CreateCartesianSpringDamper(bodyOrNodeList=[oGround, oMass],
	localPosition0=[0,0,0],
	localPosition1=[0,0,0],
	stiffness=[2500,0,0], #stiffness in x/y/z direction
	damping=[25,0,10],
	drawSize=0.1)
	
	#add load via marker:
	bodyMarker = mbs.AddMarker(MarkerBodyPosition(bodyNumber=oMass))
	mbs.AddLoad(LoadForceVector(markerNumber = bodyMarker, loadVector = [50,0,0]))
	
	mbs.Assemble()
	
	SC.visualizationSettings.nodes.drawNodesAsPoint = False
	SC.visualizationSettings.general.drawWorldBasis = True
	
	tEnd = 10
	stepSize = 0.02 #smaller gives more accurate results
	
	# add simulation settings for basicTutorial2024
	simulationSettings = exu.SimulationSettings()
	simulationSettings.timeIntegration.verboseMode = 1
	simulationSettings.timeIntegration.endTime = tEnd
	simulationSettings.timeIntegration.numberOfSteps = int(tEnd/stepSize)
	
	# start solver for basicTutorial2024
	mbs.SolveDynamic(simulationSettings=simulationSettings)
	
	mbs.SolutionViewer()

	EXUDYN General
35	import numpy as np matrix = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(matrix) zero_matrix = np.zeros((4, 4)) print(zero_matrix) one_matrix = np.ones((4, 4)) print(one_matrix) identity_matrix = np.eye(4) print(identity_matrix) random_matrix = np.random.rand(4, 4) print(random_matrix)
36	import numpy as np array_1d = np.array([1, 2, 3, 4, 5]) mean_1d = np.mean(array_1d) print("Mean of 1D array:", mean_1d) array_2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) mean_2d = np.mean(array_2d) print("Mean of 2D array:", mean_2d)

	EXUDYN General

37	import matplotlib.pyplot as plt
	import numpy as np
	
	# Generate data
	x = np.linspace(0, 10, 100) # 100 points from 0 to 10
	y = np.sin(x) # sine of x
	
	# Create the line plot
	plt.plot(x, y, label='Sine wave')
	
	# Customize the plot
	plt.xlabel('x values') # Label for the x-axis
	plt.ylabel('y values') # Label for the y-axis
	plt.title('Simple Line Plot') # Title of the plot
	plt.legend() # Add a legend
	plt.grid(True) # Add a grid
	
	# Show the plot
	plt.show()

38	import numpy as np
	
	# Generate a single random number between 0 and 1
	random_number = np.random.rand()
	print("Single random number:", random_number)
	
	# Generate an array of random numbers between 0 and 1
	random_array = np.random.rand(5)
	print("Array of random numbers:", random_array)
	
	# Generate a 2D array of random numbers between 0 and 1
	random_matrix = np.random.rand(3, 4)
	print("2D array of random numbers:\n

39	pip install scipy pip install scipy==1.9.1 # Replace 1.9.1 with your desired version

40	pip install pandas
	import pandas as pd
	
	# Reading a CSV file from a local path
	df = pd.read_csv('path/to/your/file.csv')
	print(df)