- Threaded and non-threaded script code
- Callback functions
  - Dynamics callback functions
  - Joint callback functions
  - Contact callback function
  - Vision callback functions
  - Trigger callback functions
  - User config callback functions
- Script execution order
- Buffers
- Lua vs Python scripts
- Lua crash course
- Plugins
- CoppeliaSim's library
- Accessing scene objects programmatically
- Explicit and non-explicit calls
- CoppeliaSim API framework
  - Regular API reference
  - Regular API constants
  - Properties
    - Properties reference
- Simulation
  - Simulation dialog
- Tutorials
  - BubbleRob tutorial
  - Building a clean model tutorial
  - Line following BubbleRob tutorial
  - Inverse kinematics tutorial
  - External controller tutorial
  - Plugin tutorial
  - Robot language interpreter integration tutorial
- ROS tutorials
  - ROS tutorial
  - ROS 2 tutorial
- Compiling CoppeliaSim

Callback functions

Callbacks or callback functions are script functions that are triggered by a specific event. They represent the only entrance into script code. They can be categorized into user callbacks, and system callbacks:

- user callback functions are directly triggered by user code, or a plugin. This happens for instance when the user calls a script function in a different script via sim.callScriptFunction, or when the user clicks a custom UI button, which triggers a custom event. User callback functions can usually run in any type of script.
- system callback functions on the other hand are triggered by CoppeliaSim when a certain event happens, for instance when a script is initialized, or before switching to another scene. System callback functions have fixed function names, such as *sysCall_init* or *sysCall_beforeInstanceSwitch*. Not all system callback functions are supported by all types of scripts, e.g. some can only run in an add-on (e.g. *sysCall_addOnScriptSuspend*).

While most system callback functions operate in a relatively straight-forward manner and are easy to understand (see further down for a complete list of system callback functions), some require a little bit more explanations:

- Dynamics callback functions are called by the physics engine before and after each dynamics simulation step
- The contact callback function is called by the physics engine for each contact pair
- Joint callback functions are called for custom joint control
- Vision callback functions are called whenever an new image is acquires or applied
- Trigger callback functions are called when a certain trigger event is fired
- User config callback functions are called when the user double-clicks a user parameter icon

Following is a list of all supported system callback functions. Most support returning {cmd='cleanup'} or {cmd='restart'} to end or restart the script. Refer to the comments in the code below for details about individual functions:

Python | Lua

```python
#python

def sysCall_init():
    # Do some initialization here.
    pass

def sysCall_thread():
    # Entry point for threaded scripts.
    pass

def sysCall_cleanup():
    # Do some clean-up here.
    pass

def sysCall_nonSimulation():
    # Is executed when simulation is not running.
    # Not for simulation scripts or the main script.
    pass

def sysCall_beforeSimulation():
    # Simulation is about to start.
    # Not for simulation scripts or the main script.
    pass

def sysCall_afterSimulation():
    # Simulation has just ended.
    # Not for simulation scripts or the main script.
    pass

def sysCall_actuation():
    # Put some actuation code here.
    pass

def sysCall_sensing():
    # Put some sensing code here.
    pass

def sysCall_suspend():
    # Simulation is about to be suspended.
    pass

def sysCall_suspended():
    # Simulation is suspended.
    pass

def sysCall_resume():
    # Simulation is about to resume.
    pass

def sysCall_realTimeIdle():
    # Is executed when idle in real-time mode
    pass

def sysCall_beforeInstanceSwitch():
    # About to switch to another scene.
    # Not for simulation scripts or the main script.
    pass

def sysCall_afterInstanceSwitch():
    # Switched to another scene.
    # Not for simulation scripts or the main script.
    pass

def sysCall_beforeSave(inData):
    # About to save the scene, or about to set an undo/redo point
    print(str(inData['regularSave']), inData['file'])

def sysCall_afterSave(inData):
    # After the scene was saved, or after an undo/redo point was set
    print(str(inData['regularSave']), inData['file'])

def sysCall_afterLoad():
    # After the scene was loaded
    pass

def sysCall_selChange(inData):
    # After selection was changed
    for handle in inData['sel']:
        print(f"Object with handle {handle} is selected")

def sysCall_beforeCopy(inData):
    # Before one or several objects will be copied. Can be reentrant
    for handle in inData['objects']:
        print(f"Object with handle {handle} will be copied")

def sysCall_afterCopy(inData):
    # After one or several objects were copied. Can be reentrant
    for handle in inData['objects']:
        print(f"Object with handle {handle} was copied")
```

```python
def sysCall_afterCreate(inData):
    # After one or several objects were created. Can be reentrant
    for handle in inData['objects']:
        print(f"Object with handle {handle} was created")

def sysCall_beforeDelete(inData):
    # Before one or several objects will be deleted. Can be reentrant
    for handle in inData['objects']:
        print(f"Object with handle {handle} will be deleted")
    # inData['allObjects'] indicates if all objects in the scene will be deleted

def sysCall_afterDelete(inData):
    # After one or several objects were deleted. Can be reentrant
    for handle in inData['objects']:
        print(f"Object with handle {handle} was deleted")
    # inData['allObjects'] indicates if all objects in the scene were deleted

def sysCall_beforeMainScript():
    # Can be used to step a simulation in a custom manner.
    outData = {"doNotRunMainScript": False}  # when true, then the main script won't be executed
    return outData

def sysCall_addOnScriptSuspend():
    # Add-on execution is about to be suspended.
    # Only for add-ons.
    pass

def sysCall_addOnScriptResume():
    # Add-on execution is about to be resumed.
    # Only for add-ons.
    pass

def sysCall_dyn(inData):
    # See the dynamics callback function section for details
    return {...}

def sysCall_joint(inData):
    # See the joint callback function section for details
    return {...}

def sysCall_contact(inData):
    # See the contact callback function section for details
    return {...}

def sysCall_vision(inData):
    # See the vision callback function section for details
    # Only for customization scripts and simulation scripts.
    return {...}

def sysCall_trigger(inData):
    # See the trigger callback function section for details
    # Only for customization scripts and simulation scripts.
    return {...}

def sysCall_userConfig():
    # See the user config callback function section for details.
    # Only for customization scripts.
    pass

def sysCall_moduleEntry(inData):
    # Called when a user module menu entry is selected, e.g.
    print(f"Following module menu entry was selected: {inData['handle']}")
    # See sim.moduleEntry for details.

def sysCall_msg(msg, origin):
    # Synchronously called when sim.broadcastMsg is called, or when sysCall_init, sysCall_cleanup,
    # sysCall_addOnScriptSuspend, sysCall_addOnScriptResume or sysCall_userConfig is called. Can be reentrant
    pass

def sysCall_data(inData):
    # Asynchronously called when custom data properties attached to this script, or this script's parent are modified
    # Only for simulation- and customization scripts
    pass

def sysCall_event(cborBuffer):
    # Asynchronously called when an internal variable or state changes. Can be reentrant
    # Mainly used to synchronize an external GUI/viewer/renderer with CoppeliaSim.
    # See also sim.getGenesisEvents and sim.setEventFilters
    pass
```