PyMAPDL cheat sheet

Version: v0.70.dev0

Connect to MAPDL remote session

Launch and connect to MAPDL. You can specify the version you want to use with `version` argument.

```
from ansys.mapdl.core import launch_mapdl
mapdl = launch_mapdl(version=242)
```

Exit a session

```
mapdl.exit()
```

Connect to an existing instance at IP address `192.168.1.30` and port `50001`:

```
mapdl = launch_mapdl(start_instance=False,
    ip='192.168.1.30', port=50001)
```

Create and exit a pool of instances:

```
# Create a pool of 10 instances
from ansys.mapdl.core import pool
mapdl_pool = pool.MapdlPool(10)
# Exit the pool
mapdl_pool.exit()
```

PyMAPDL CLI

Access MAPDL instances through CLI

```
# Start an MAPDL instance at port ``50051``
pymapdl start --port 50051
# List the current MAPDL instances and processes
pymapdl list
# Stop all the MAPDL instances
pymapdl stop --all
```

PyMAPDL commands

PyMAPDL commands are Python statements that act as a wrapper for APDL commands. For example: `ESEL,s,type,1` is translated as

```
mapdl.esel('s', 'type', vmin=1)
```

Help for the MAPDL class functions is accessible with:

```
help(mapdl.esel)
```

Most of the time, commands that start with * or / have these characters removed:

```
mapdl.prep7() # /PREP7
mapdl.get() # *GET
```

Load arrays from Python to MAPDL:

```
import numpy as np
np_array = np.array([[1,2,3], [4,5,6]])
mapdl.load_array("array_name", np_array)
```

Write parameters and access from or to the MAPDL database:

```
# Create a parameter from a NumPy array
mapdl.parameters['my_np_param'] = np_array
# Save a parameter to a NumPy array
saved_np_array = mapdl.parameters['my_np_param']
```

Access to specific model entity values with `get_array` and `get_value`.

```
# List the current selected node numbers
mapdl.get_array('NODE', item1='NLIST')
# Get the number of selected nodes
total_node = mapdl.get_value(entity='node',
    item1='count')
```

Convert APDL script to Python files

An existing APDL script can be converted to PyMAPDL format with the following commands:

```
import ansys.mapdl.core as pymapdl
pymapdl.convert_script(
  "mapdl_script.dat", "pymapdl_script.py"
)
```

Mesh and geometry of a model

Store the finite element mesh as a VTK UnstructuredGrid data object:

```
grid = mapdl.mesh.grid
```

Save element and node numbers to Python arrays with the `mapdl.mesh` and the `mapdl.geometry` classes.

```
# Get an array of the nodal coordinates
nodes = mapdl.mesh.nodes
# Save node numbers of selected nodes to an array
node_num = mapdl.mesh.nnum
# Save volume numbers of selected nodes to an array
volum_numbers = mapdl.geometry.vnum
# Save keypoint numbers of selected nodes to an
    array
keypoints = mapdl.geometry.keypoints
```

Solve an analysis

```
mapdl.solution()
mapdl.solve()
mapdl.finish()
```

Post-process results

The PostProcessing class is used for plotting and saving results to NumPy arrays.

```
mapdl.post1()
mapdl.set(1, 1)
mapdl.allsel()

# Plot nodal temperatures
mapdl.post_processing.plot_nodal_temperature()
# Save nodal temperatures to a Python array
nodal_temp=
    mapdl.post_processing.nodal_temperature()
```

You can store the command output following the Postprocessing object methods:

```
cmd = mapdl.prnsol("TEMP")
cmd.to_list()
```

Create nice plots

Use PyVista to interpolate data, save results and store them in the underlying UnstructuredGrid object:

```
from pyvista import Plotter
pl = Plotter()
pm = mapdl.post_processing.plot_element_stress(
  "X", return_plotter=True
)
pl.add_mesh(pm.meshes[0])
pl.show()
```

```
# Plot selected elements
mapdl.eplot()
# Plot selected volumes
mapdl.vplot()
# Plot selected areas
mapdl.aplot()
# Plot selected lines
mapdl.lplot()
```

```
# Testing one of the above plotting
mapdl.eplot()
```