MI:Scripting Toolkit for Python 2.0

Introducing the new Streamlined API

ANSYS GRANTA

www.grantadesign.com

© Granta Design | CONFIDENTIAL

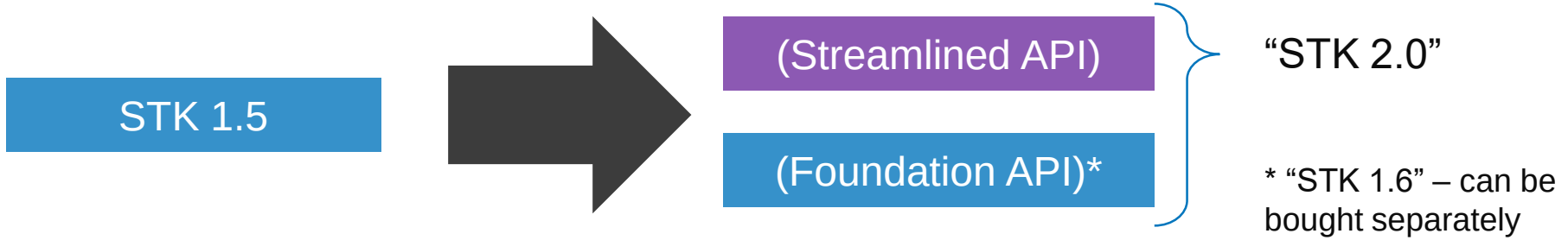
Agenda

- STK 2.0 - Introductory guide to the ‘Streamlined API’ (~ 25 mins)
  - API structure: general workflow
  - Basic Actions: Beginning a session, Accessing & creating records, fetching and updating attributes, bulk operations
  - Some examples of features: subsets, criteria searching, complex attribute types
- Streamlined API “cheat sheet” (~ 1 mins)
- 3 simple examples of applications (~ 10 mins)
  - Folder coloring tool
  - Subset assigning tool
  - Database summarising tool
- Questions (remainder)

Introduction to the two APIs of STK 2.0

- "Foundation" API (python 2.7 & 3.5+)
  - The API you are familiar with
  - Request & response-based actions, verbose
- "Streamlined" API (python 3.5+)
  - Same functionality as foundation API
  - Wholly new syntax
  - Method-based actions, concise
  - Primary focus for the future

AVAILABLE NOW!



© Granta Design | CONFIDENTIAL

Recap: Foundation API

```
AR = gdl.AttributeReference(DBK,PR,ID)
AV = gdl.AttributeValue(…)
SC = gdl.RecordSearchCriterion(SV,AR)
DT = gdl.XXXXDataType(…)
SV = gdl.XXXXSearchValue(…)
SR = gdl.SearchResult(…)
```

```
import GRANTA_MIScriptingToolkit as gdl
```

```
S = gdl.GRANTA_MISession(servicelayer, username, password, domain)
```

```
TR = gdl.TableReference(name,DBK)
PR = gdl.PartialTableReference(name)
TR = gdl.TreeRecord(…)
IR = gdl.ImportRecord(…)
RR = gdl.RecordReference(…)
```

S.BrowseService

GetDatabases()
- List all available DBs

Response returns…
- List of **MIDatabase** objs

GetTables(DBK)
- List all tables for a DB
- Access list of all attributes & table references

Response returns…
- List of **TableDetail** objs

GetChildNodes(RR)
- List all TreeRecords below current node
- Non-recursive
- Access record references

Response returns…
- List of **TRs**

S.SearchService

SimpleTextSearch(DBK,str)
- Search a table for a simple string

Response returns…
- List of **SRs**

CriteriaSearch(DBK,SC:list)
- Search over DB using SCs

Response returns…
- List of **SRs**

S.DataExportService

GetRecordAttributesByRef(AR:list, RR:list)
- Provide:
- RecordReference(s)
- AttributeReference(s)
- Get Data!

Response returns…
- List of **RecordData** objs

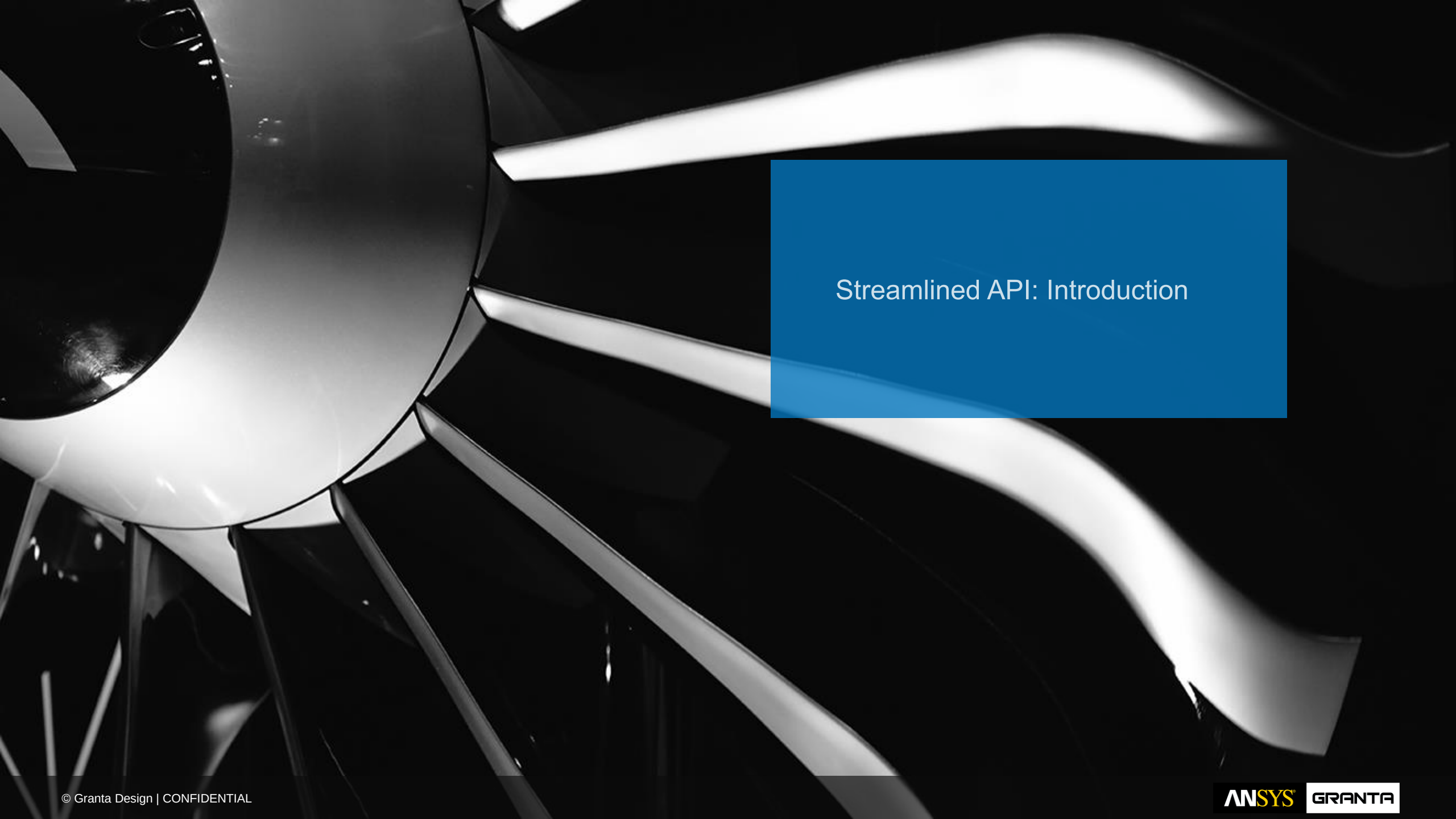
S.DataImportService

SetRecordAttributes(IR:list)
- Provide list of ImportRecord objects, which contain
- RecordReference(s) and AttributeReference(s)

Response returns…
- List of **NamedRecord** objs

```
Request = ServiceRequest(…) (e.g. gt = GetTables(…))
Response = ServiceResponse(Request) (resp = GetTablesResponse(gt))
```

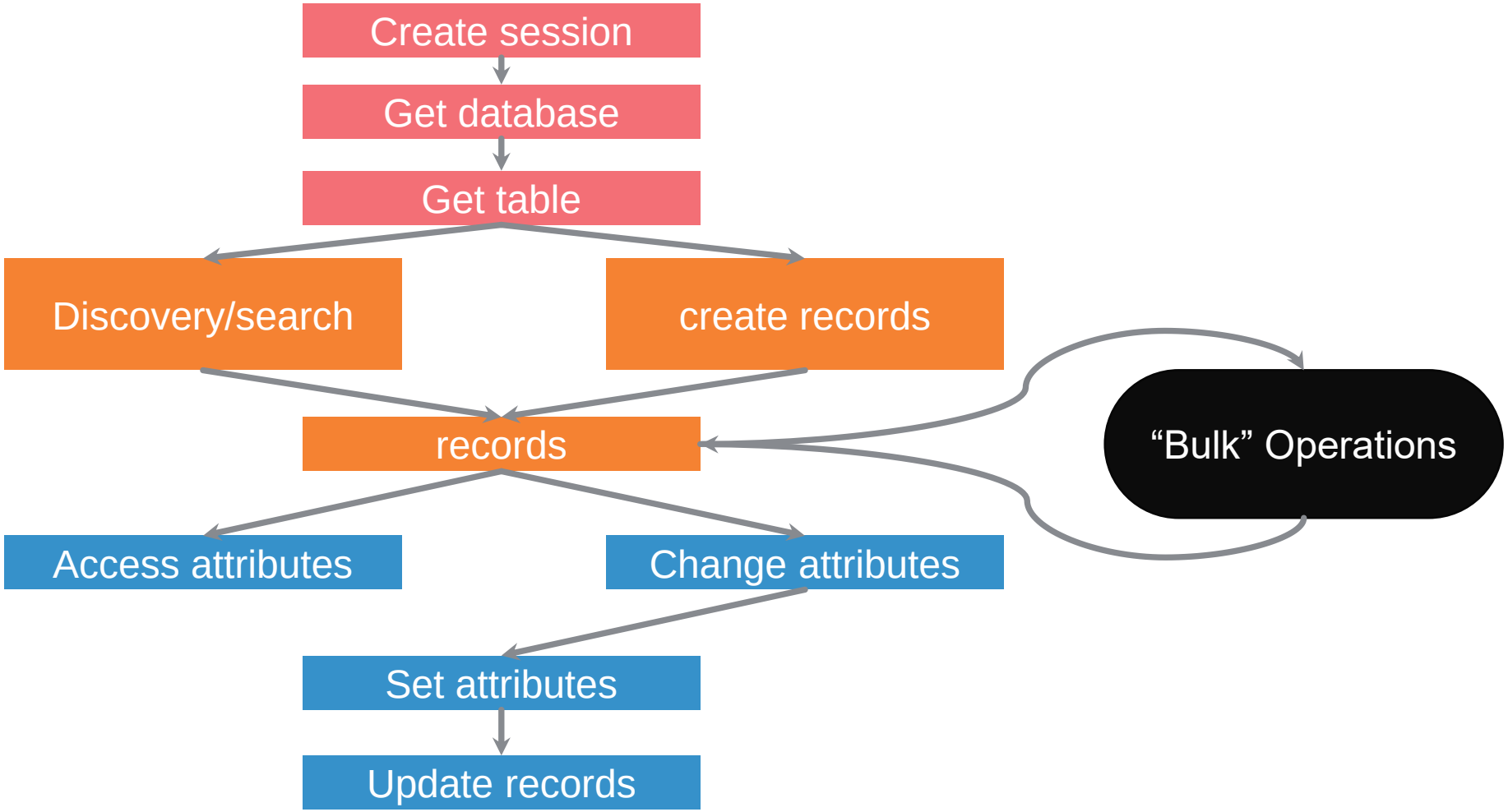
© Granta Design | CONFIDENTIAL



Streamlined API: Introduction

© Granta Design | CONFIDENTIAL

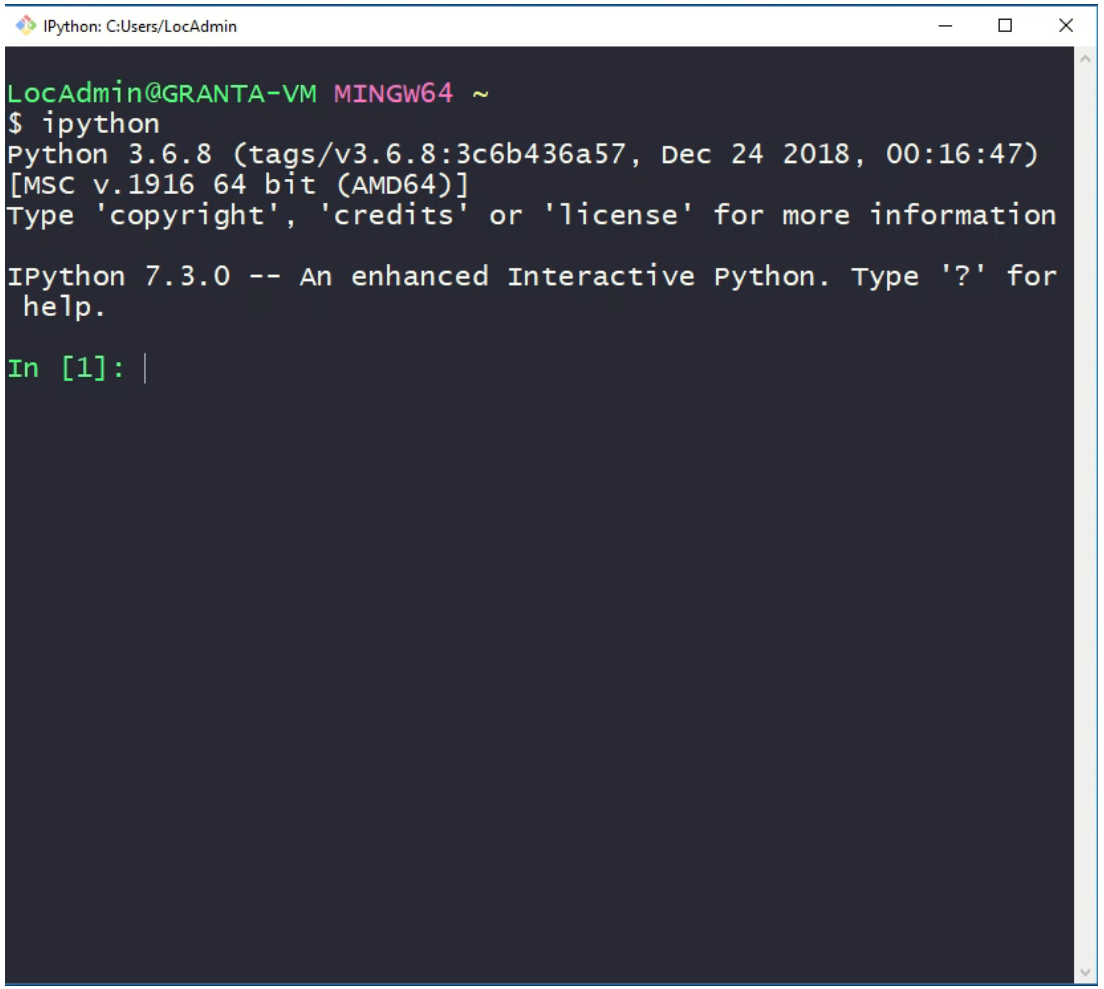
API Structure: an overview



© Granta Design | CONFIDENTIAL

Basic Actions: Beginning a session

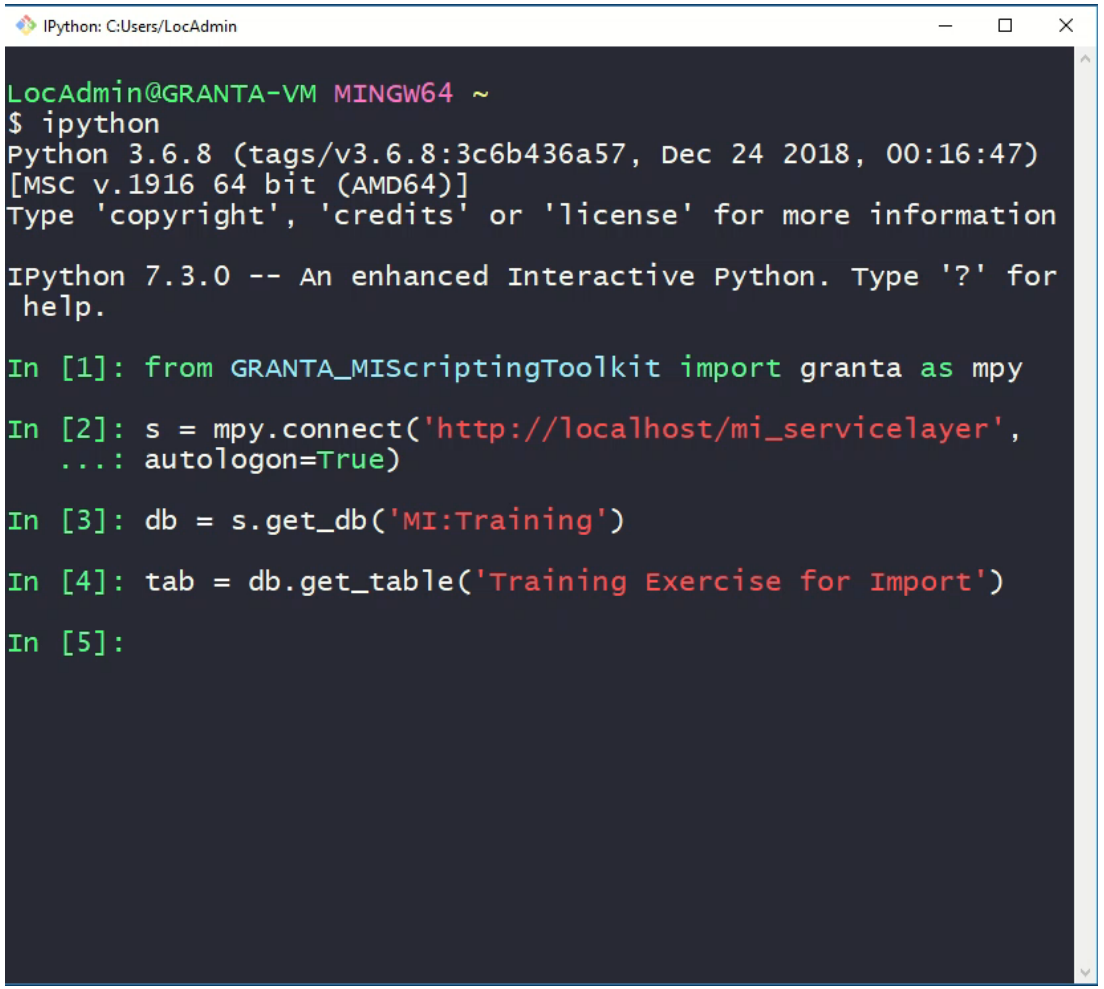
1. `from GRANTA_MIScriptingToolkit import granta as mpy`
   a) Import the new layer
2. `mi = mpy.connect('https://yourserver/mi_servicelayer', 'user', 'pwd', 'domain')`
   a) Begin a session
3. `db = mi.get_db('MI:Training')`
   a) Get a database object
   b) view available databases with 'mi.dbs'
4. `tab = db.get_table('MaterialUniverse')`
   a) Get a table object from the database object
   b) View available tables with 'db.tables'
5. `tab.attributes`
   a) View the attributes in the table as **AttributeDefinition**s



© Granta Design | CONFIDENTIAL

Basic Actions: Accessing & creating **Records**

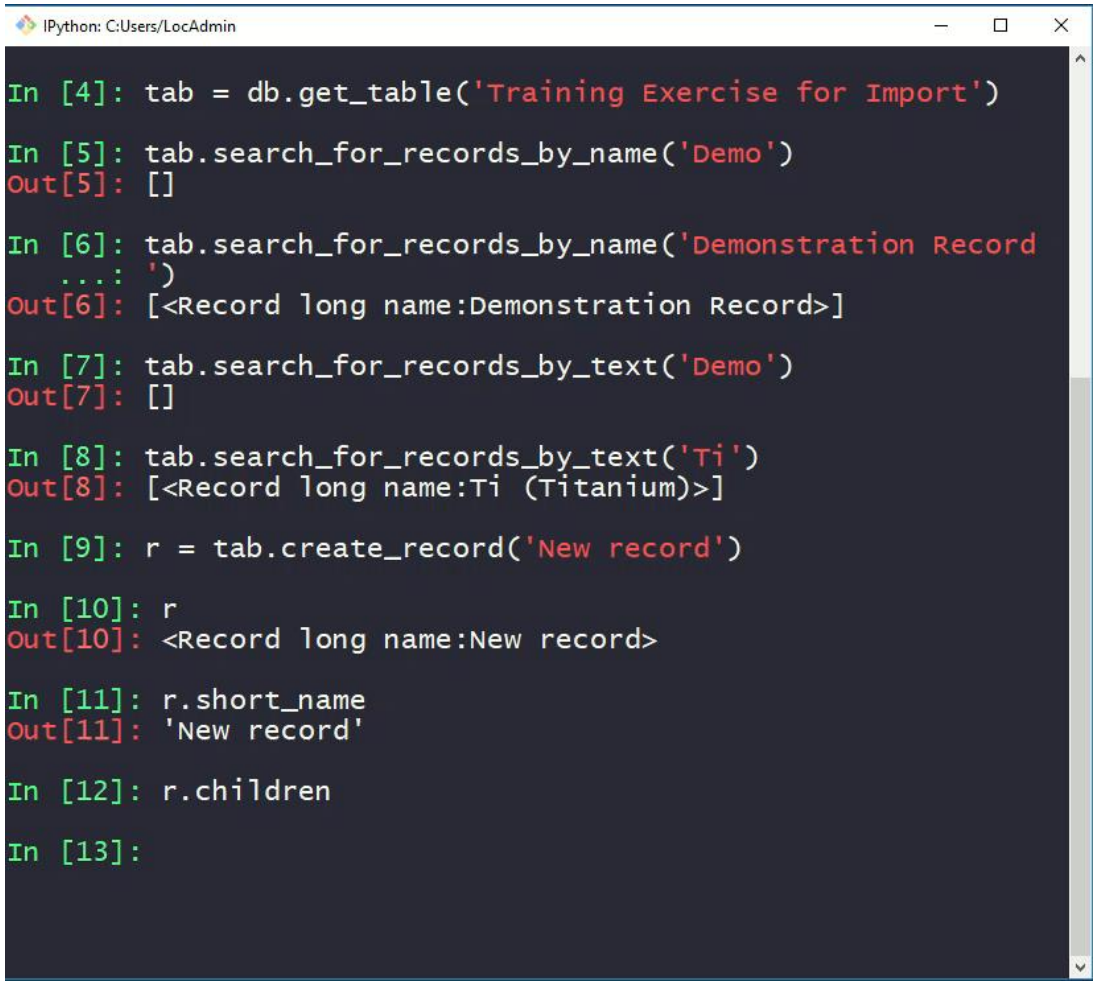
- Simple Search/Get
  - Session, Database and Table level
  - `mi/db/tab.search_for_records_by_text('text')`
  - `mi/db/tab.search_for_records_by_name('name')`
  - `mi/db/tab.get_record_by_id(hguid='…')`
- Create
  - Table level
  - `create_record(name)`
    - Creates local record object
    - Must be **`updated`** to take effect on server
  - `path_from(start, [path, of, folders])`
    - Creates all folders/record on server
    - **AND** returns last imported record



© Granta Design | CONFIDENTIAL

Basic Actions: Fetching & updating **Attributes**

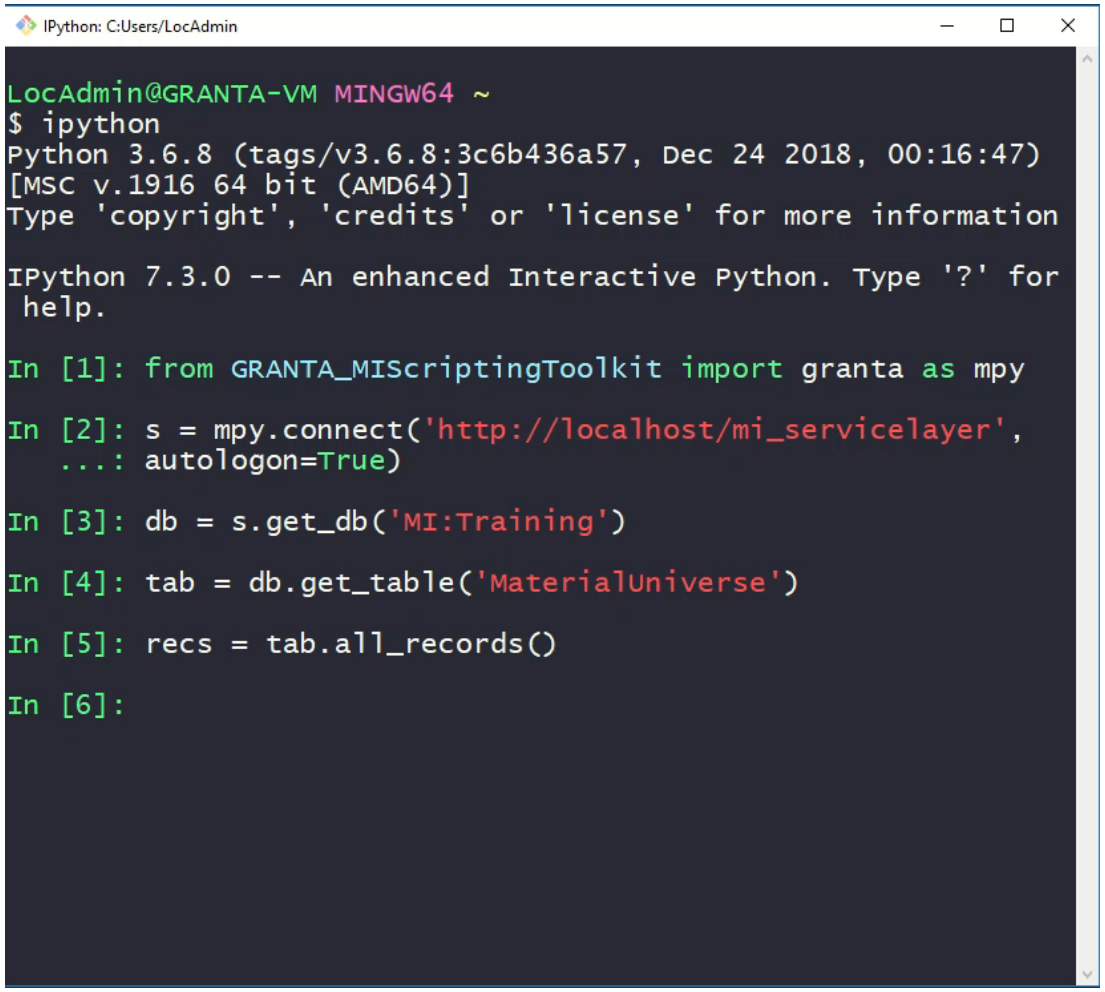
- **Record.**attributes
  - Dictionary of attributes, indexed by name
  - In a new record: all empty **Attribute** objects
  - Loaded automatically if dictionary not yet cached
  - Specific objects for each type, e.g.
    - **AttributePoint**
    - **AttributeTabular**
    - **AttributeRange**
  - To edit: change their “value” property in **most** cases
- Record.set_attributes(attributes)
  - set **Attribute** objects to be **update**d
- mi.**update**(records)
  - Push changes to Granta
  - Creates record if does not already exist



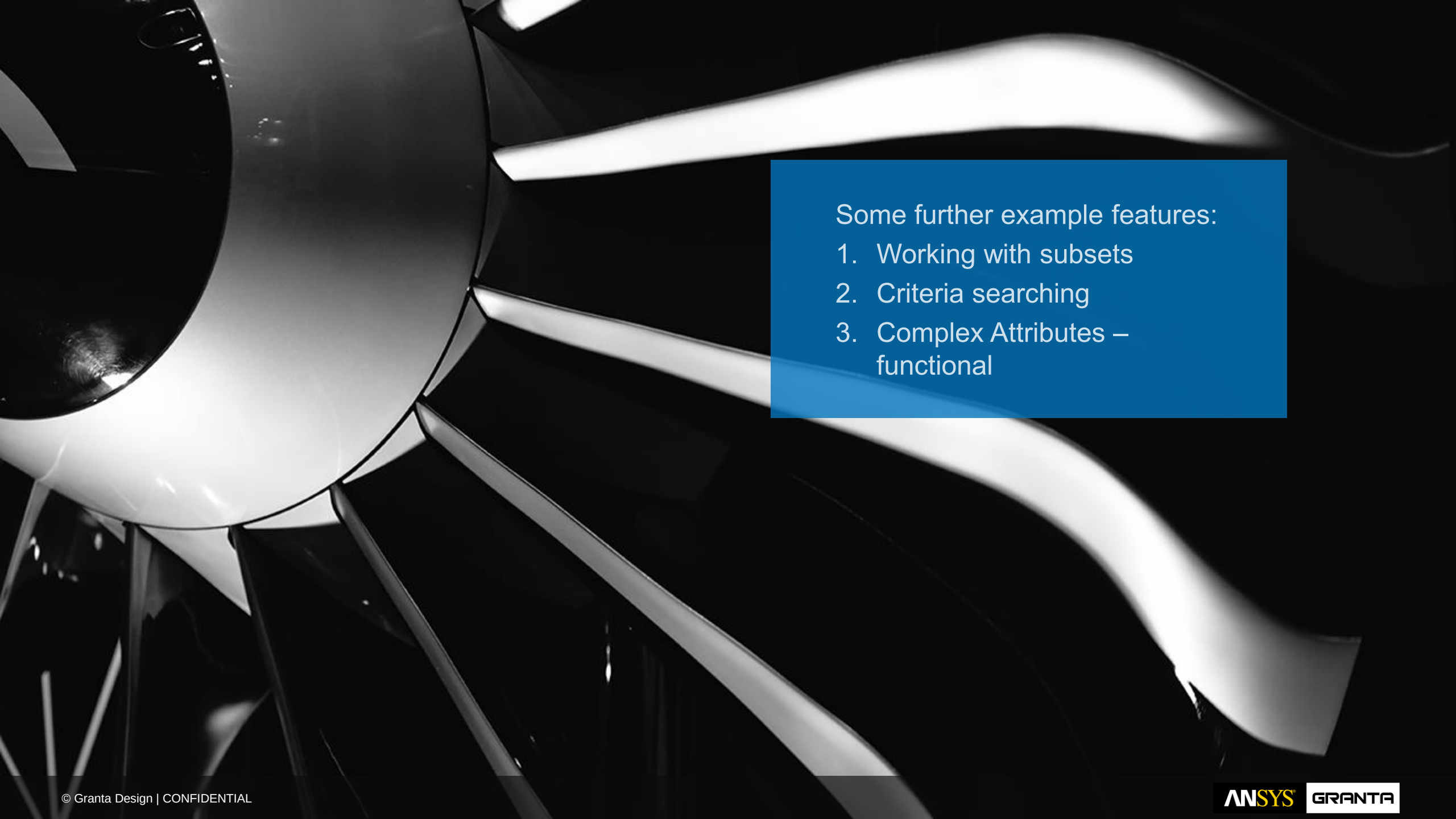
© Granta Design | CONFIDENTIAL

Basic Actions: “bulk” operations
“bulk” methods increase efficiency and transparency

- Most efficient ways of performing tasks
  - Requests are batched together
  - user controls size
- Operations do not return anything
  - Operate on the provided objects only
- **Table**.bulk_fetch(records, attributes)
  - Populate each **Record** with only the **Attribute**s in ‘attributes’
- **Table**.bulk_link_fetch(records, link_groups)
  - See cheat sheet for details on links
- **Session**.bulk_delete_or_withdraw_records(records)
- **Session**.bulk_fetch_release_states(records)



© Granta Design | CONFIDENTIAL

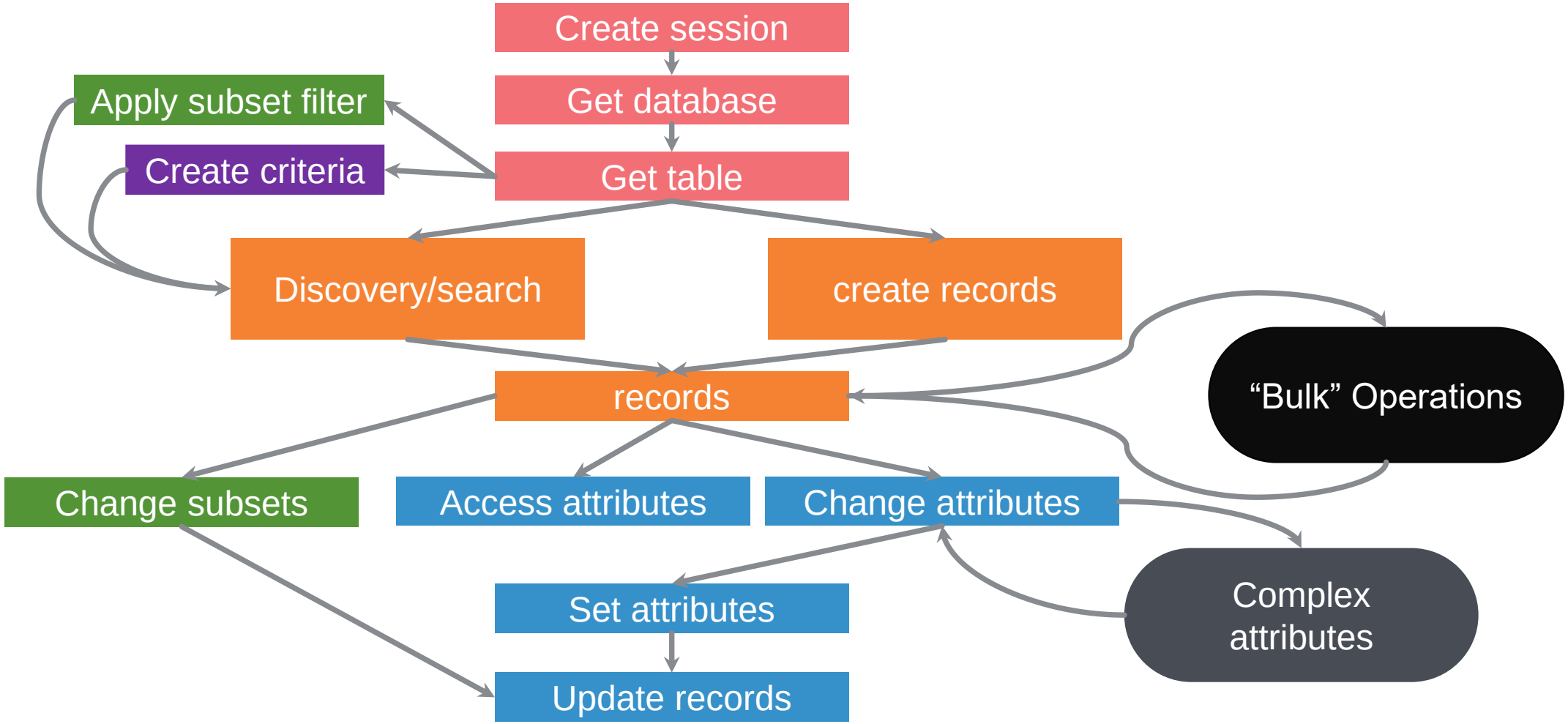


Some further example features:

1. Working with subsets
2. Criteria searching
3. Complex Attributes – functional

© Granta Design | CONFIDENTIAL

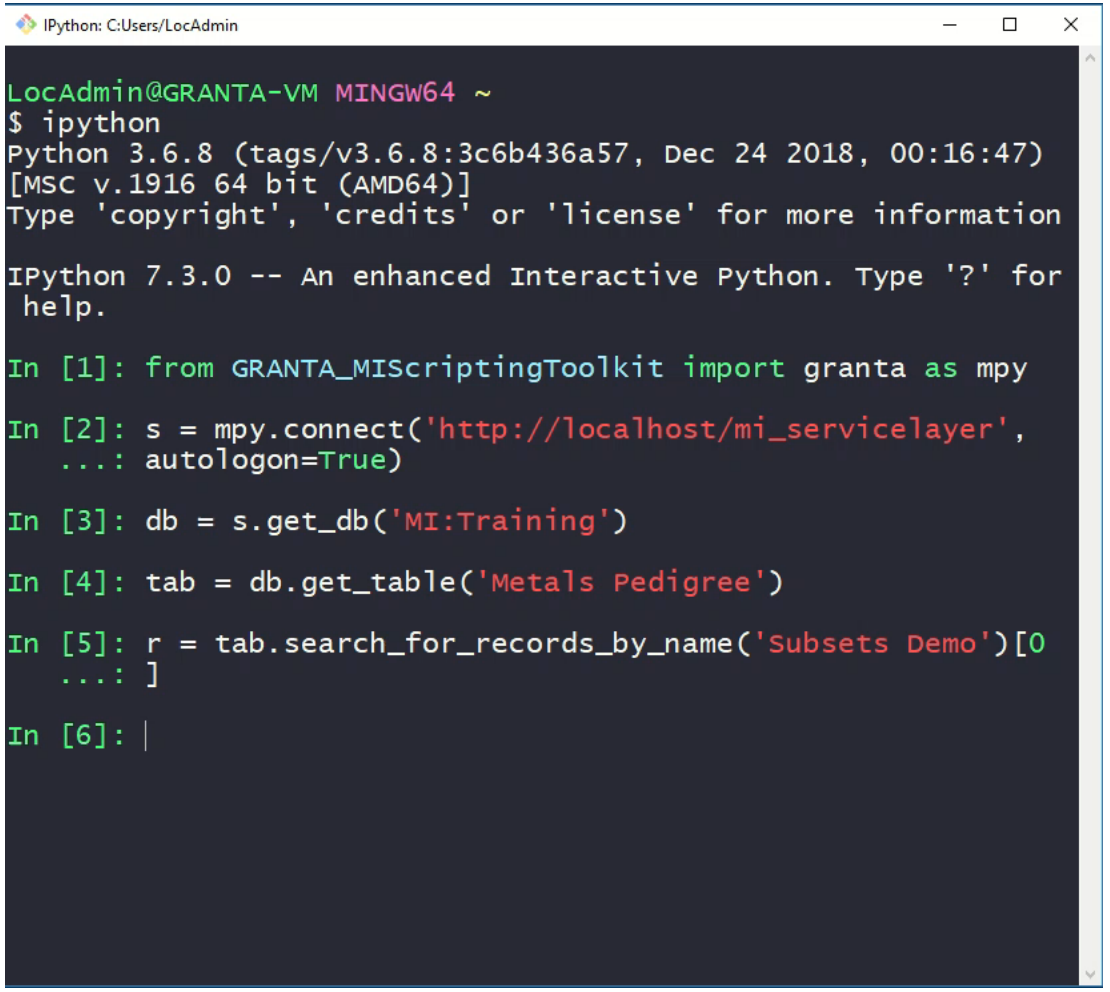
API Structure: example feature location in workflow



© Granta Design | CONFIDENTIAL

Example features: Working with subsets

- The **Table.**subsets property is a python set of strings
- This shows the subsets selected *for* the table
- These act as filters on searches
- The **Record.**subsets property is also a set of strings
- It shows the subsets the **Record** is *in*
- …and the subsets it **will** be in
- (NOT same as **Table**.subsets)



© Granta Design | CONFIDENTIAL

Example features: Criteria Searching
`mi/db/tab.search_for_records_where([crit1, crit2, …])`

- `ad = table.attributes['attribute']`
- `crit = ad.search_criterion(less_than,` - float
  `greater_than,` - float (+ less_than for "between")
  `contains,` - string (text & disc) or bool (logi)
  `contains_any,` - list of strings (disc only)
  `contains_all,` - list of strings (disc only)
  `does_not_contain,` - string (disc only)
  `exists,` - True or False (all types)
  `equal_to,` - integer (INPT only)
  `between_dates,` - tuple('YYYY-MM-DD', 'YYYY-MM-DD')
  `in_column)` - tabular column to search in

- One option per criterion, EXCEPT for
  - `in_column` (combine with any other option to search in that tabular column),
  - `less_than/greater_than` (combine these to get "between").
- No limit to number of criteria in a search_for_records_where() search

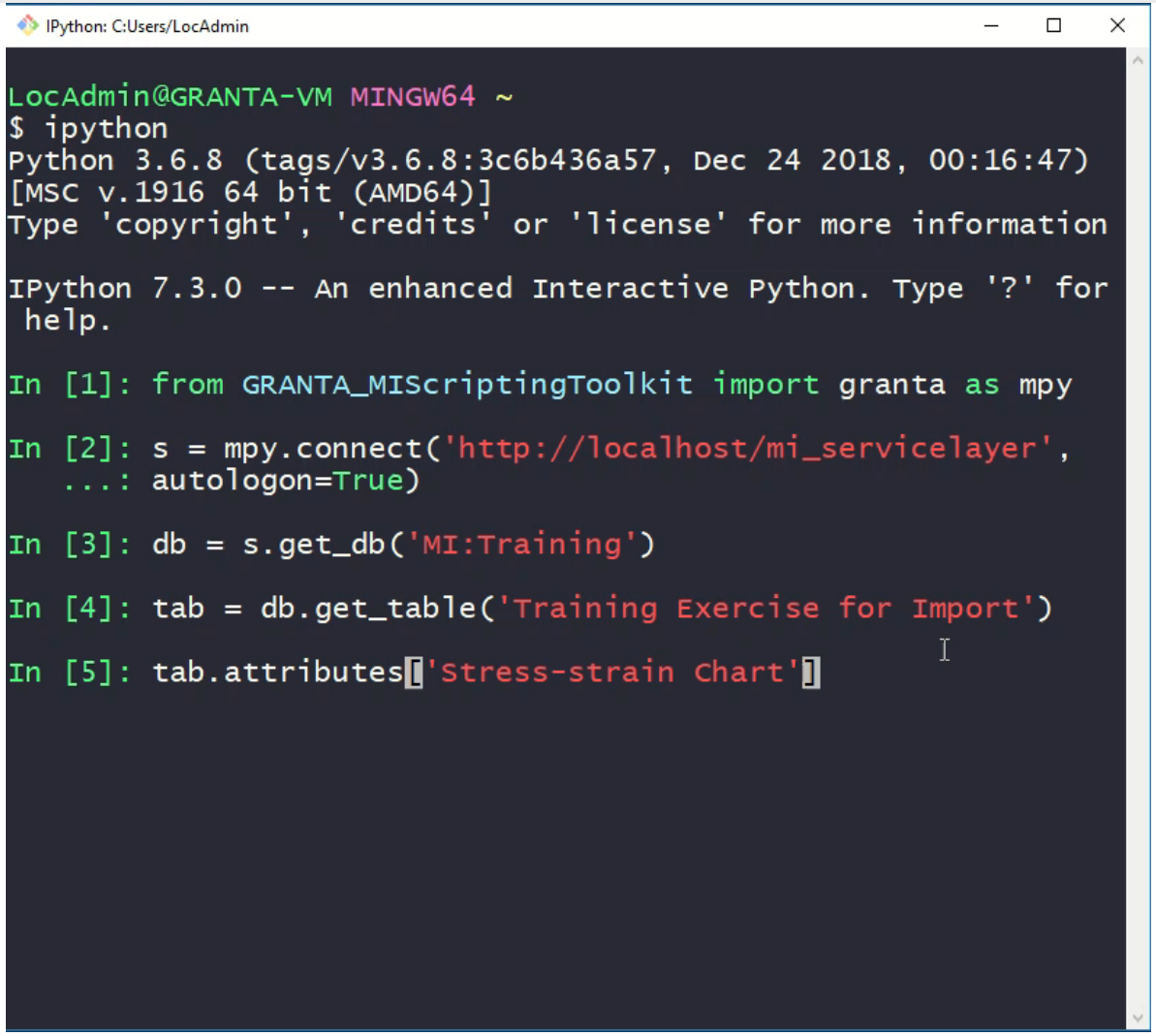
© Granta Design | CONFIDENTIAL

Example features: complex Attribute types

- Majority of **Attribute** objects changed by editing ‘value’ property
- **Except for:** Tabular, Functional, Point, File, and Picture
- Consider Functional: `fd = r.attributes['functional attribute']`
- Two functional attributes: **FunctionalSeries, FunctionalGrid**
  - ‘Functionally’ identical
  - Different import methods & some different class methods
- Functional data stored as list of lists, inc. column headers (`fd.data`)
- Data is **assumed to be ordered**
  - Series are found automatically on import
- Access parameters & change parameter units through `fd.parameters`
- Add data through `fd.add_point/add_range`
  - `Arg is a dict of entries: {'y': 0., 'Temperature': 3., 'Mass': 2.}`

© Granta Design | CONFIDENTIAL

Criteria searching for functional data & editing it



© Granta Design | CONFIDENTIAL

Streamlined API: cheat sheet (page 1)

```
from GRANTA_MIScriptingToolkit import granta as mpy
```

Session

Basics

- `s = mpy.connect(servicelayer, user, pwd, domain, autologon=False, timeout=300000)`
- `s.dbs` – view all available databases
- `s.get_db(name, db_key=None)` – get a database

Unique record acquisition method

- `s.records_from_string(text)` – get records from an MI12 clipboard-format string

Data updating

- `recs = s.update([rec1, rec2, …])`
- `recs = s.update_links([rec1, rec2, …])`

Data fetching

- `s.bulk_delete_or_withdraw_records(records)`
- `s.bulk_fetch_release_states(records)`

AttributeDefinition

Basics

- `ad = tab.attributes['attribute']`
- `ad.type`
- `ad.unit`
- `ad.parameters` (only on multi-point types)
- `ad.meta_attributes` – dict of metas

Database

Basics

- `db = s.get_db('MI:Training')`
- `db.tables` – view all available tables
- `db.get_table(name)` – get a table

Useful properties & methods

- `db.unit_system` – see the current unit system
- `db.unit_systems_available`
- `db.set_unit_system('Metric', absolute_temperatures=False)`
- `db.record_link_groups` – details of all the RLGs in the db

Searching (Session/Database/Table)

- `search_for_records_by_text('text')`
- `search_for_records_by_name('name')`
- `search_for_records_where([crit1, crit2, …])`
- `get_record_by_id(db_key, hguid=None, vguid=None, identity=None)`

SearchCriterion

- `crit = ad.search_criterion(less_than, greater_than, contains, contains_any, contains_all, does_not_contain, exists, equal_to, between_dates, in_column)`

Table

Basics

- `tab = db.get_table('MaterialUniverse')`
- `tab.attributes` – all attribute definitions

Useful properties & methods

- `tab.subsets` – current subset filter
- `tab.subsets_available`
- `tab.all_records(include_folders=False, include_generics=True)`

Record creation

- `rec = tab.create_record(name, parent=None, short_name=None, attributes=None, folder=False, subsets=set(), release_record=False, color=None)`
- `rec = tab.path_from(starting_node, tree_path, end_node=None, color=None)`
- `db.record_link_groups` – details of all the RLGs in the db

Data Fetching

- `tab.bulk_fetch(records, attributes, batch_size=100, parallelise=False)`
- `tab.bulk_link_fetch(records, link_groups, batch_size=100, parallelise=False)`

© Granta Design | CONFIDENTIAL

Streamlined API: cheat sheet (page 2)

Record

r = tab.**create_record**(name)

Basic Properties

- r.name – long name
- r.short_name – short name
- r.color – record color
- r.type (= 'Folder' or 'Record')
- r.path – path through tree
- r.parent – parent record/table
- r.children – immediate children
- r.history_guid – history GUID
- r.release_state – e.g. 'Released'

Advanced Properties

- r.subsets – set of subsets record is in (& will be in when updated)
- r.pseudo_attributes – **PseudoAttribute**s dict
- r.attributes – dict of **Attribute**s
- r.links – dict of sets of **Record**s
- r.flag_for_release – bool, release on push

Record Methods

- r.**set_attributes**(attrs) – prime attributes
- r.**set_links**(link_group, {Record, …}) – prime new links for import on push
- r.**delete_or_withdraw_record_on_server**()
- r.**clear_attributes**(attrs) – prime attributes for deletion on push

AttributeValue (PseudoAttributeValue - *)

a = r.attributes['attribute']

Basics (common to all)

- a.name* – name of attribute
- a.**is_empty**()* – (bool) True if empty
- a.type* – 4-char type, e.g. 'POIN'
- a.value* – "value" of the attribute
- a.meta_attributes – dict of **Attribute**s
- a.unit – unit symbol

AttributeInteger

- a.value – integer value

AttributeLogical

- a.value – Boolean value (True/False)

AttributeText, -Hyperlink, -Date

- a.value – string value

AttributeDiscrete

- a.value – list of strings
- a.order – value order in viewer (read only)

AttributeUnsupported

- a.value – None

AttributeRange

- a.value – {'low': 0., 'high': 1.}

AttributePicture, AttributeFile

- a.value – mpy.File or mpy.Picture type
- a.load(path, file_name) – load file
- a.save(path, file_name) – save file
- a.description – (**AttributeFile** only)

AttributeFunctional (FunctionalSeries & FunctionalGrid)

- a.data – data as list of lists
- a.add_point(entry) – "entry" is a dict of each point: {'y': 0., 'Tension': 1.}
- a.add_range(entry) – as before, but 'ymin' & 'ymax' must be supplied instead of 'y'.
- a.unit – y-axis unit
- a.parameters – parameters (change x-units here)

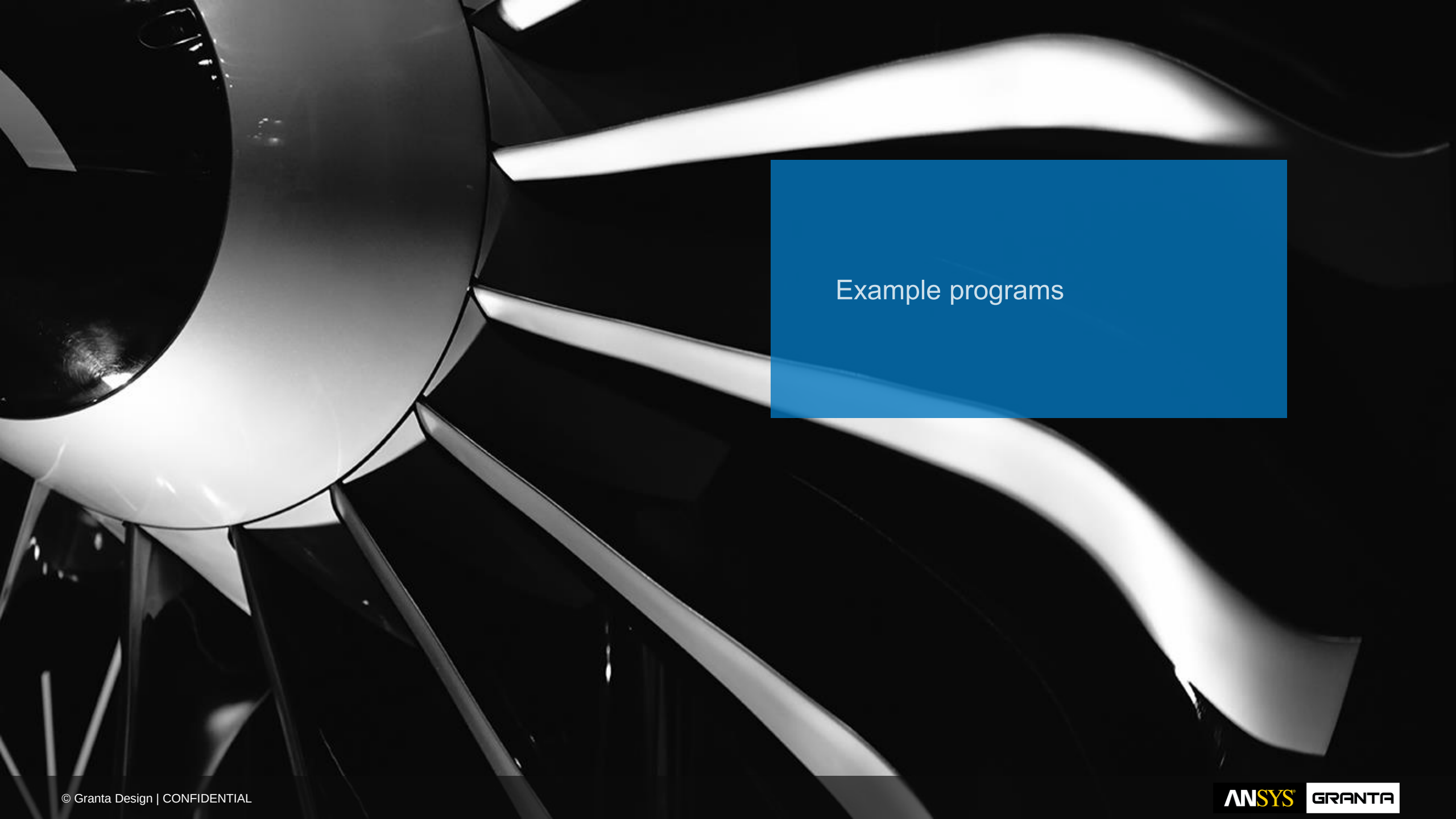
AttributeTabular

- a.**load**() – lazily loaded unless called
- a.data – raw data as list of lists
- a['col name', row_index] – access cell
- a.**add_row**(linking_value) – new row
- a.**delete_row**(row_index) – del row
- a.units – TabularUnits object
- a.columns – column names as list of strings

AttributePoint

a.points – list of floats
a.parameters = list of dicts of form [{'param': val}] (one/point & same order)

© Granta Design | CONFIDENTIAL



Example programs

© Granta Design | CONFIDENTIAL

3 example scripts using STK 2.0

- Simple, short programs
  - No more than 100 lines each
- Solve real use cases suggested by customers
- Examples of mundane tasks that can now be automated
- Will be available after webinar

© Granta Design | CONFIDENTIAL

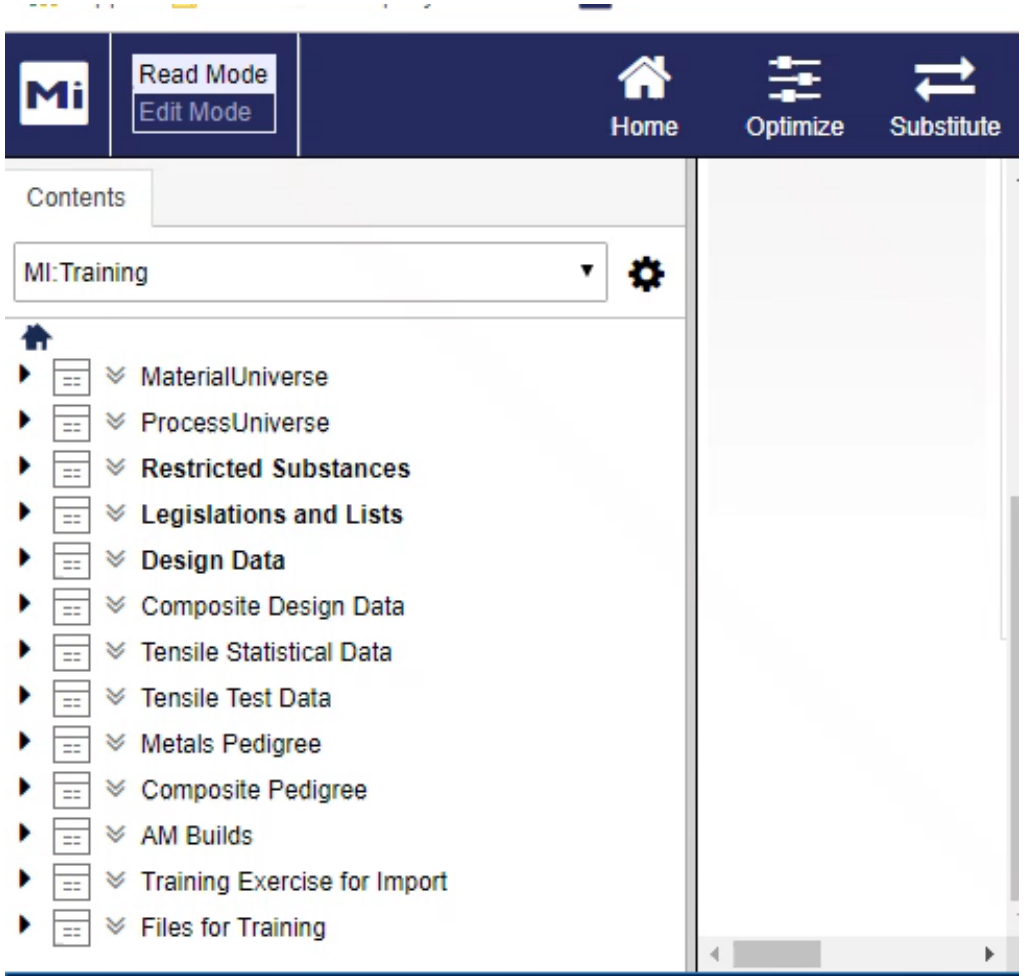
Folder Colorer

- Problem:
  - Need simple visual indicator of table structure
  - Which folders have records, which don't
- Solution:
  - Color! Color records & folders depending on their population
- Requirements:
  - Color folders in a table based on number of children
  - Black – many
  - Red – one child
  - White – empty

© Granta Design | CONFIDENTIAL

Folder Colorer: the script

```
from GRANTA_MIScriptingToolkit import granta as mpy

DB = 'MI:Training'
TAB = 'Training Exercise for Import'
full_color = 'Black'
one_rec_color = 'Red'
empty_color = 'White'

s = mpy.connect('localhost/mi_servicelayer', autologon=True)
tab = s.get_db(DB).get_table(TAB)

recs = tab.all_records(include_folders=True,
include_generics=True)
changed_recs = []

for r in recs:
    if r.type == 'Folder':
        if r.children:
            if len(r.children) == 1:
                r.color = one_rec_color
            else:
                r.color = full_color
        else:
            r.color = empty_color
        changed_recs.append(r)

s.update(changed_recs)
```



© Granta Design | CONFIDENTIAL

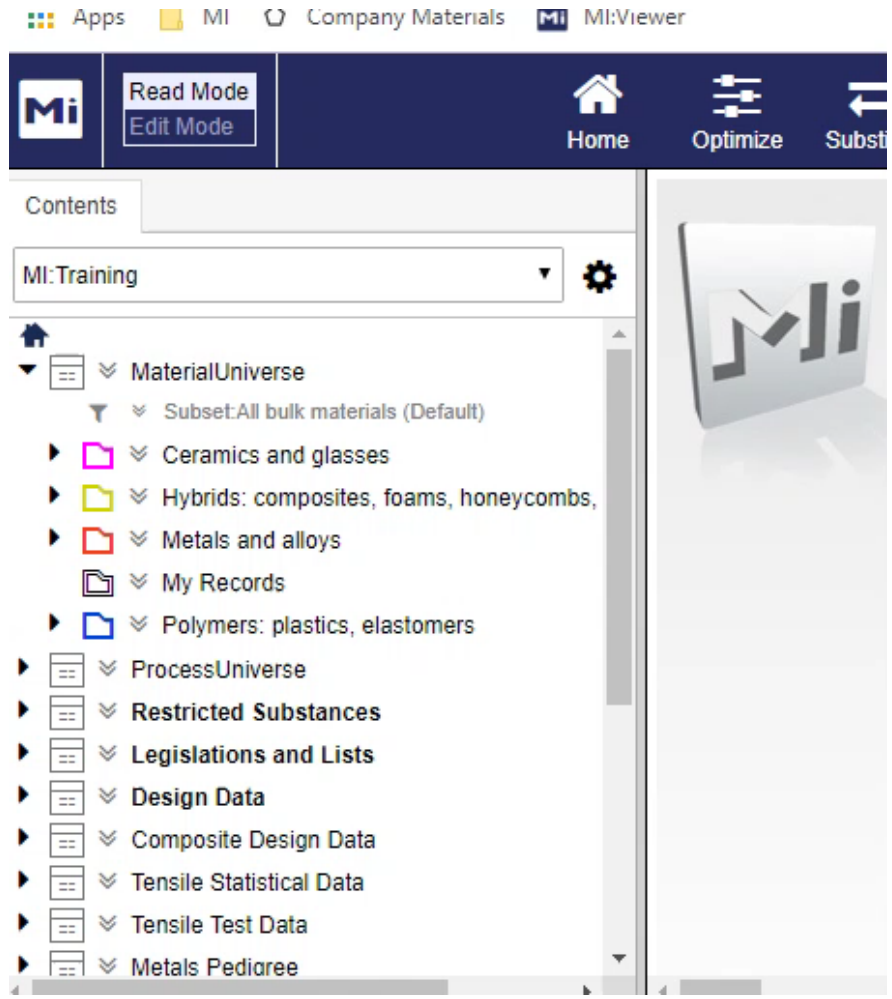
Subset Assigner

- Problem:
  - New subset(s) added to table
  - How to populate them?
  - Subsets relate to an attribute; assign based on value?
- Solution:
  - Periodically-run script that moves records between subsets
- Requirements:
  - Automatically add/remove records from subset(s) based on attribute value

© Granta Design | CONFIDENTIAL

Subset Assigner: the script

```
from GRANTA_MIScriptingToolkit import granta as mpy
attribute = 'Base'
base_to_subset = {'Oxide': 'Oxides', 'Fe (Iron)': 'Iron-
based'}
s = mpy.connect('localhost/mi_servicelayer', autologon=True)
tab = s.get_db('MI:Training').get_table('MaterialUniverse')
# generics included and folders excluded by default
recs = tab.all_records()
tab.bulk_fetch(recs, attributes=[attribute])
changed_recs = []
for r in recs:
    base = r.attributes[attribute].value[0]
    if base is not None and base in base_to_subset:
        r.subsets.add(base_to_subset[base])
        changed_recs.append(r)
# push changes!
s.update(changed_recs)
# In the script below, we do the reverse and de-populate the
# two subsets we just filled.
recs = tab.all_records(include_folders=True)
changed_recs = []
for r in recs:
    if {'Oxides', 'Iron-based'} & r.subsets:
        r.subsets.discard('Oxides')
        r.subsets.discard('Iron-based')
        changed_recs.append(r)
s.update(changed_recs)
```



© Granta Design | CONFIDENTIAL

Database Summarizer

- Problem:
  - I have a large dataset and I need a high-level idea of what's in it
  - What's populated and what's not?
- Solution:
  - Use STK to create summaries of all tables in a DB
- Requirements:
  - Database statistics presented in easily accessible manner, e.g. word doc
  - Automated report creation

© Granta Design | CONFIDENTIAL

Database Summarizer: the script

- Script iterates through all tables
  - `process_table` function on next slide
- Populates a dictionary with results
- This dictionary is saved to a json
  - Using the package ‘json’
- An additional script uses the python package: ‘python-docx’ to create word report

```
s = mpy.connect('localhost/mi_servicelayer',
autologon=True)

db = s.get_db('MI:Training')
tabs = db.tables
results = {}

for tab_ in tabs:
    print('processing table: {}'.format(tab_.name))
    results[tab_.name] = process_table(tab_)


with open('{}_results.json'.format(db.db_key), 'w') as
f:
    json.dump(results, f)
```

© Granta Design | CONFIDENTIAL

Database Summarizer: main function

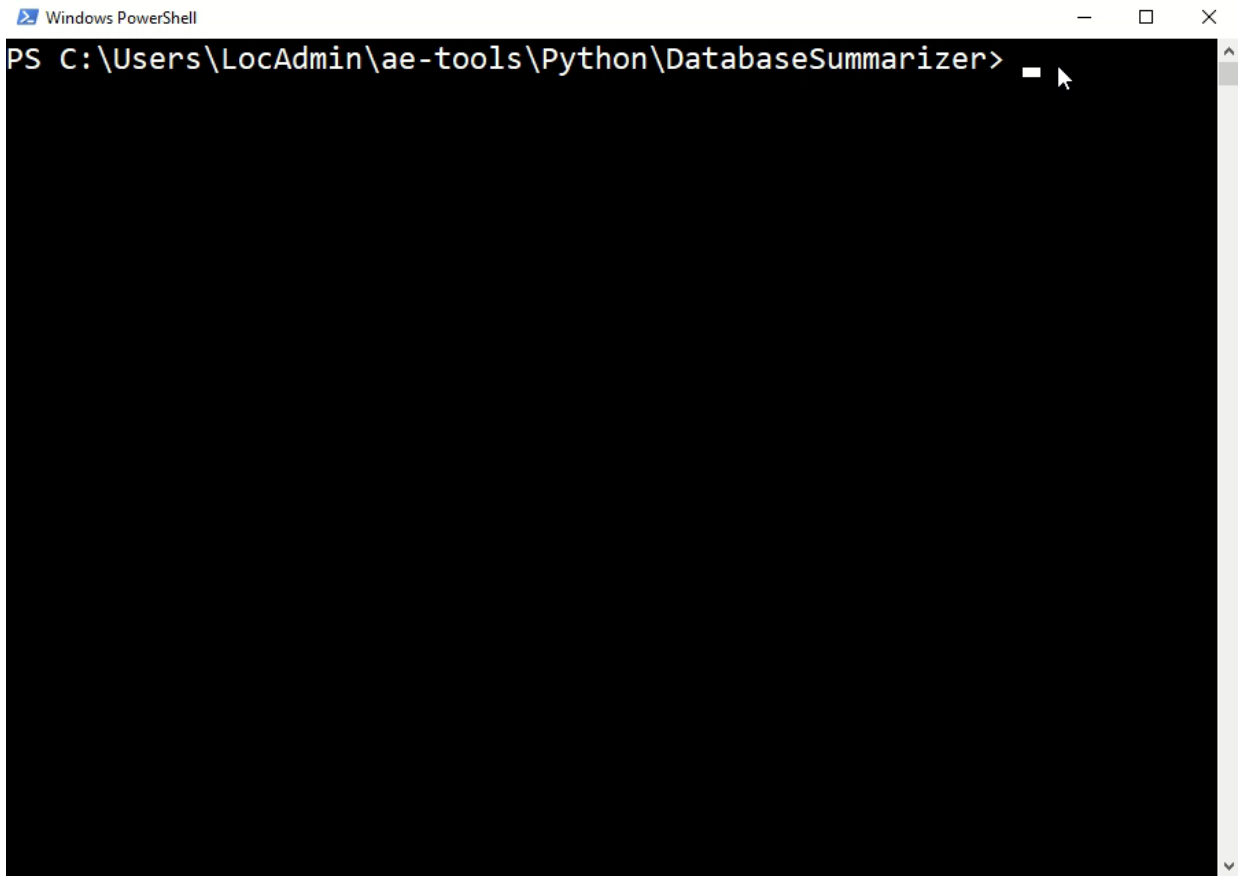
```python
def process_table(tab: mpy.Table):
    db = tab.mi.get_db(db_key=tab.db_key)
    attribute_pops = {a: 0 for a in tab.attributes}
    links_pops = {rlg.name: 0 for rlg in db.record_link_groups}
    all_recs = tab.all_records(include_folders=False, include_generics=True)
    num_recs = len(all_recs)
    tab.bulk_fetch(all_recs)
    tab.bulk_link_fetch(all_recs)
    partial_recs = 0
    full_recs = 0
    empty_recs = 0
    for r in all_recs:
        full = True
        partial = False
        full, partial = calc_population_of_record(attribute_pops, full, partial, r)
        if full:
            full_recs += 1
            partial = False
        if partial:
            partial_recs += 1
        if not partial and not full:
            empty_recs += 1
        for rlg, lv in r.links.items():
            links_pops[rlg] += len(lv)
    return {'recs': num_recs, 'full recs': full_recs, 'partial recs': partial_recs,
            'empty recs': empty_recs, 'attribute pops': attribute_pops, 'link pops': links_pops}
```

```python
def calc_population_of_record(attribute_pops,
full, partial, r):
    for a, v in r.attributes.items():
        try:
            empty = v.is_empty()
            insufficient_privileges = False
        except OSError as error:
            insufficient_privileges = True
        if not insufficient_privileges:
            if not empty:
                attribute_pops[a] += 1
                partial = True
            else:
                full = False
    return full, partial
```

© Granta Design | CONFIDENTIAL

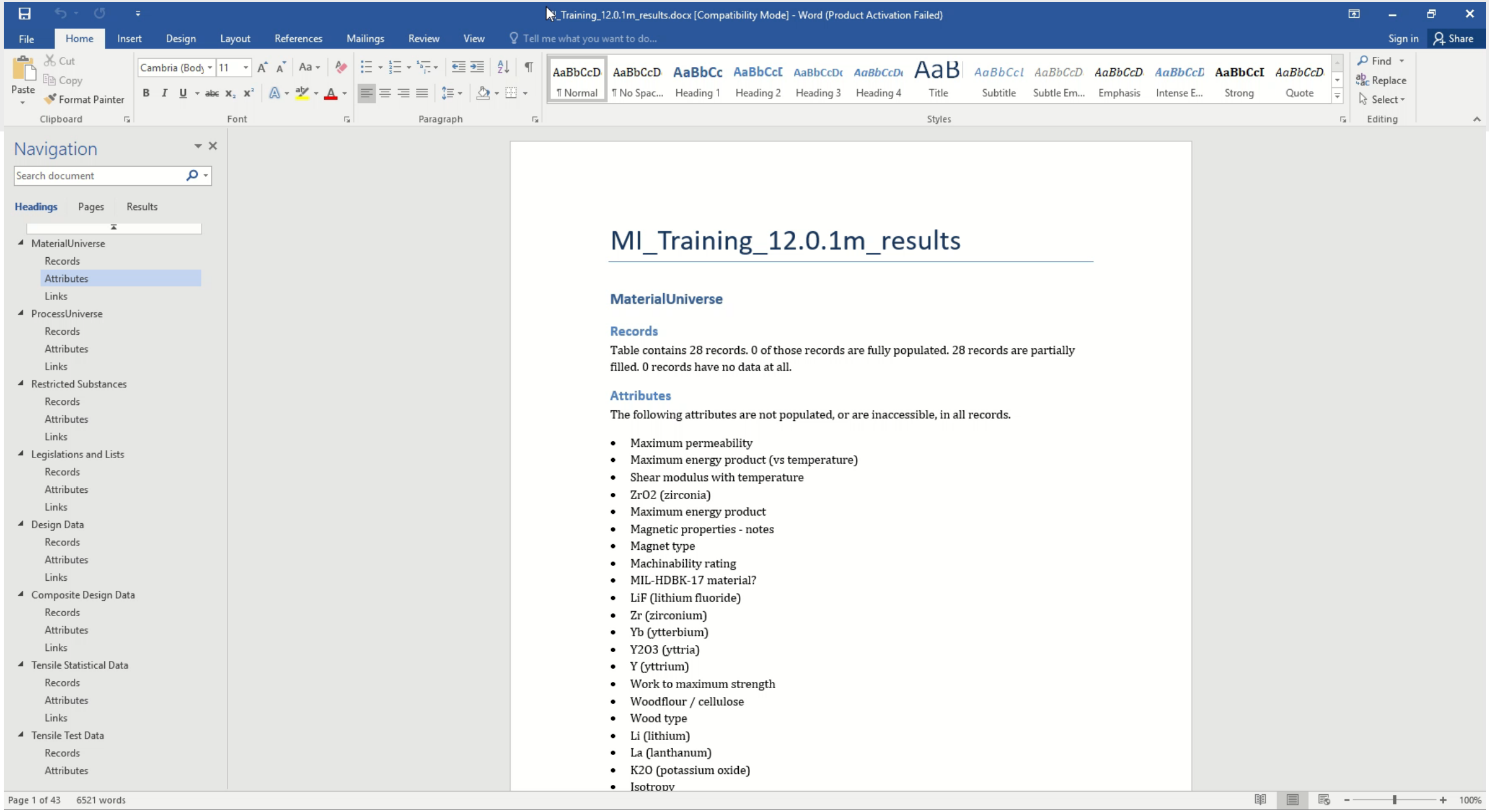
Json output; once complete, needs to be processed


```
"Training Exercise (completed)": {
     "recs": 2,
     "full recs": 0,
     "partial recs": 2,
     "empty recs": 0,
     "attribute pops": {
          "Description": 2,
          "Density": 2,
          "Composition": 2,
          "Material": 0,
     },
     "link pops": {
          "Tensile test data": 0,
          "Material batch": 0,
          "Tensile tests from this build": 0,
          "MaterialUniverse": 0,
          "Smart Link to MaterialUniverse": 0
     }
}
```

- Processing script converts json to word

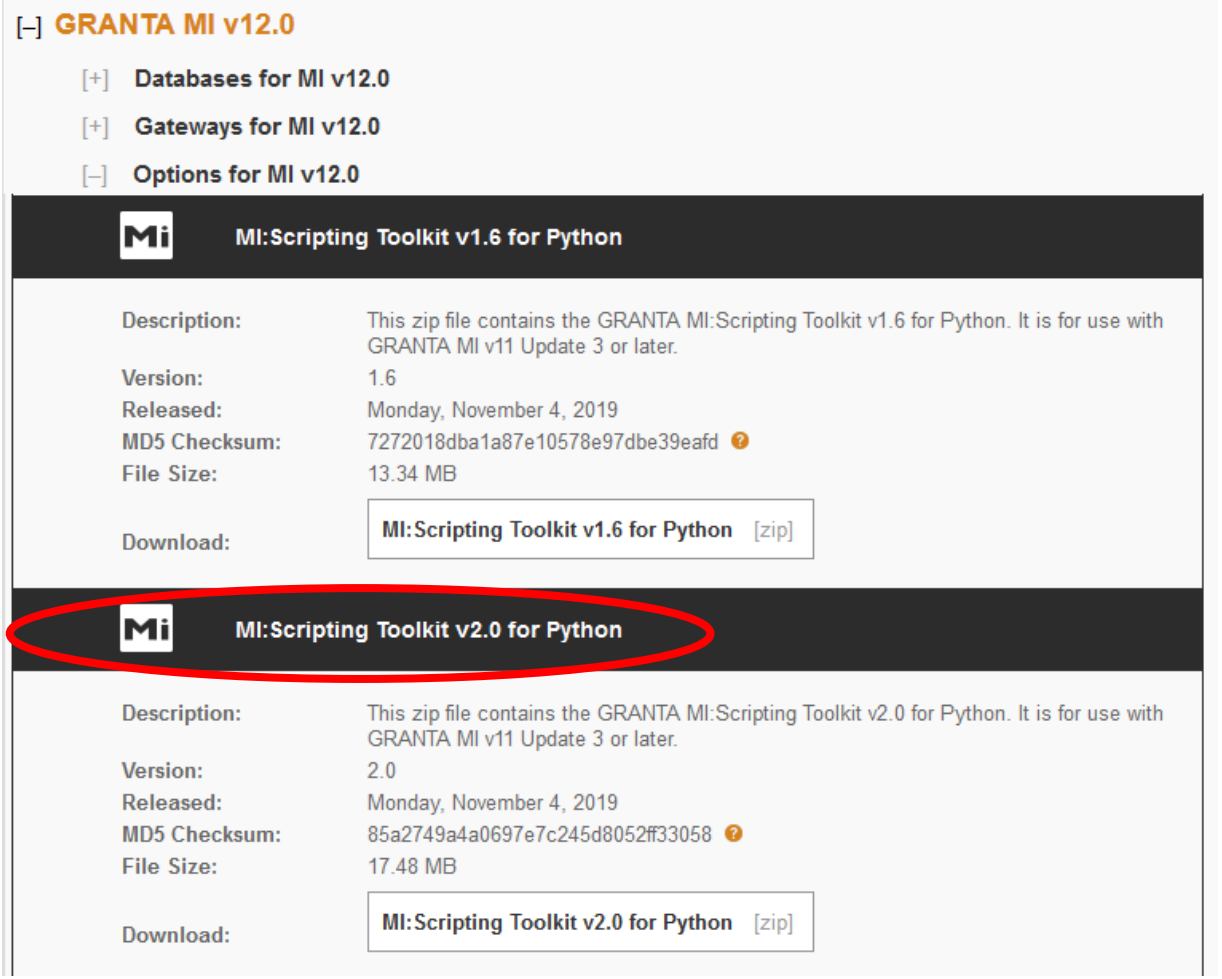
© Granta Design | CONFIDENTIAL



© Granta Design | CONFIDENTIAL

Summary

- Streamlined API
  - Complete overhaul of the STK API
  - Pythonic
  - Concise
  - Fast to write
  - Easy to read
  - Simple to maintain
  - Designed to increase productivity
- You can get it now!
  - Available for all existing STK owners
  - Go to your ‘My Downloads’
  - And…



© Granta Design | CONFIDENTIAL

Questions?

ANSYS GRANTA

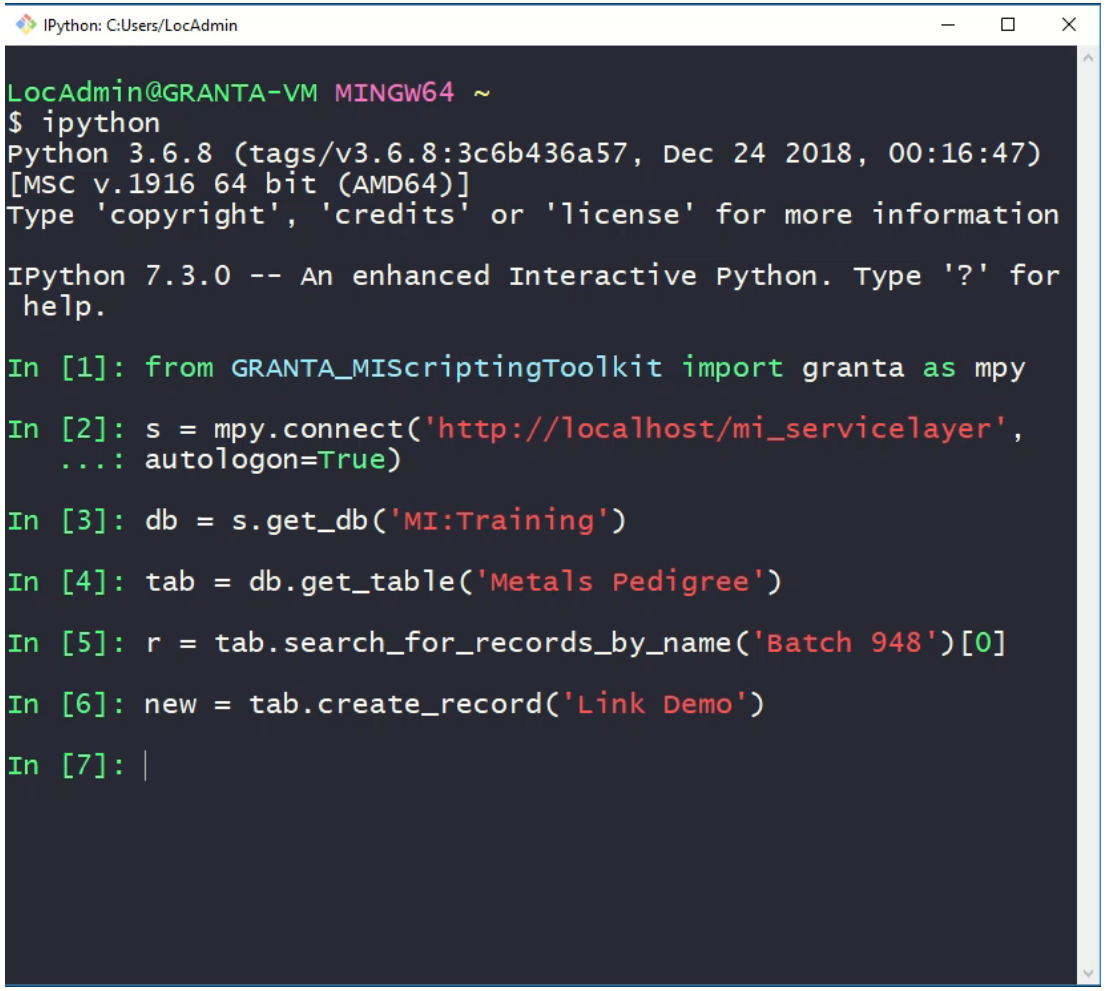
www.grantadesign.com

© Granta Design | CONFIDENTIAL

© Granta Design | CONFIDENTIAL

Advanced Techniques: Working with links

- links property is a dict of sets, indexed by record link group name
  - E.g. {'Forward': {rec1, rec2, …}}
- links "set" in same way as attributes
  - `r.set_links('Forward', {rec3, rec4})`
- Objects in each set actual **Record** objects
- Link group info available in **Database**.record_link_groups
  - List of **LinkGroupDetail** objects
  - Contain details of link group name, table from, table to, etc.
- Static, dynamic and cross-database links supported



© Granta Design | CONFIDENTIAL

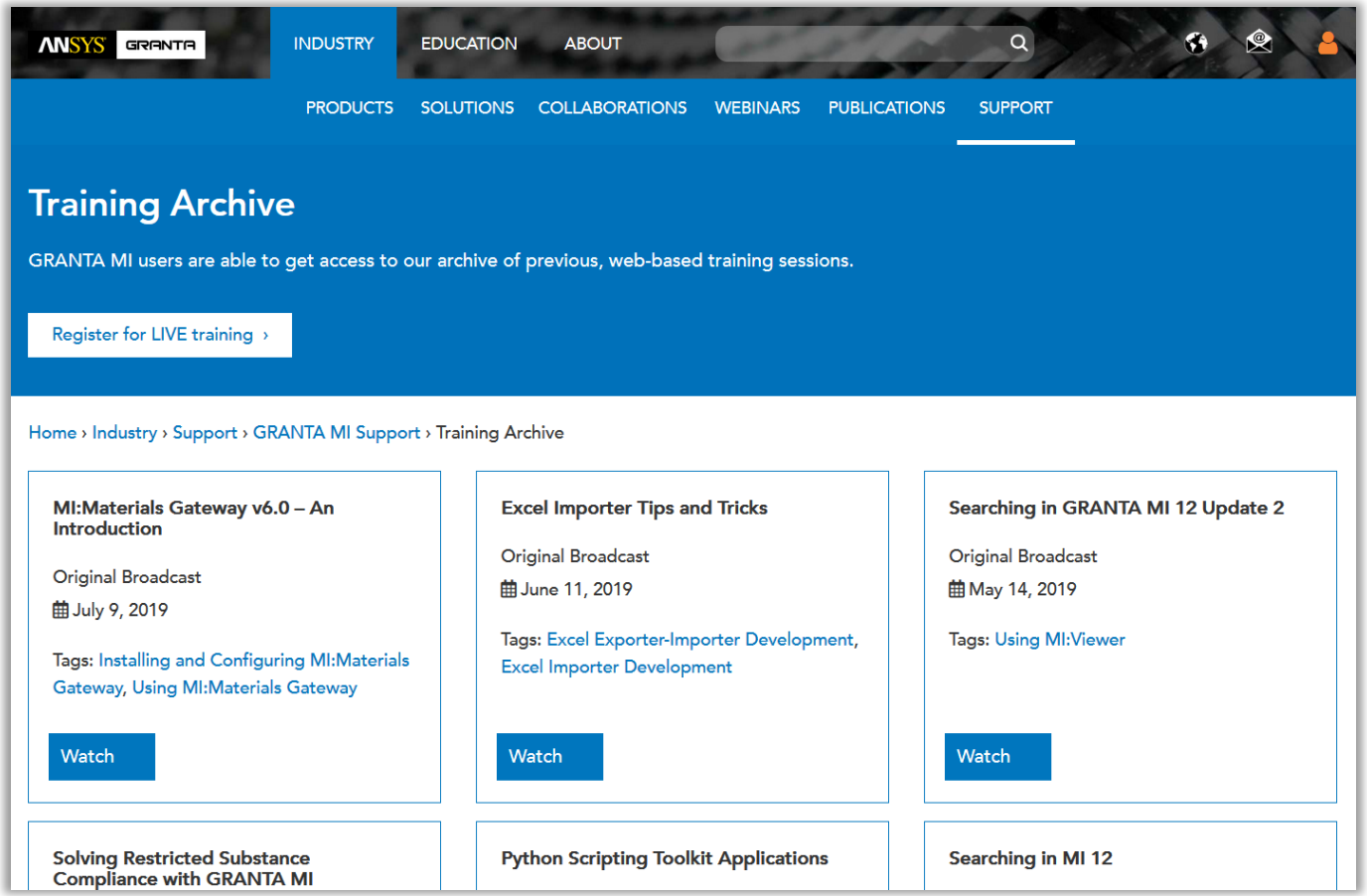
Advanced Techniques: complex AttributeValue types

- Majority of **Attribute** objects changed by editing 'value' property
- **Except for:** Tabular, Functional, Point, File, and Picture
- File & Picture are easy - `fp = r.attributes['binary attribute']`
  - `fp.value` – `mpy.File` or `mpy.Picture` object!
  - Load in a file using `fp.load(path, filename)`
  - & save locally with `fp.save(path, filename)`
  - Change description `fp.description = 'blah'` (`mpy.File` only)
- The others are more complicated
  - But, `object.value` always returns the object for these!
  - Case-study: Functional

© Granta Design | CONFIDENTIAL

GRANTA MI Monthly Online Training Archive

Find previous sessions by logging in: https://grantadesign.com/industry/support/granta-mi/training-archive/



Training schedule

Next training is December 10 – What's New in GRANTA MI 2020 R1

https://grantadesign.com/industry/support/granta-mi/training/