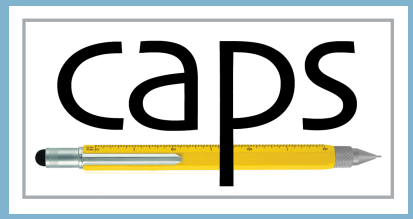
Computational Aircraft Prototype Syntheses



Training Session 2
CAPS Geometry

ESP v1.19

Marshall Galbraith galbramc@mit.edu
Bob Haimes haimes@mit.edu
Massachusetts Institute of Technology

John F. Dannenhoffer, III
jfdannen@syr.edu
Syracuse University

Overview

- Python Basics
  - Strings, Functions, and Classes
- Loading and viewing geometry via pyCAPS
  - pyCAPS.Problem
- Accessing/modifying DESPMTR
  - geometry.despmtr
  - geometry.save, .writeParameters, .readParameters
  - geometry.bodies
  - Value Object Sequence .value Shortcut
- Accessing SET and @values using OUTPMTR
  - geometry.outpmtr
- Directing bodies to AIMs
  - Attribute `capsAIM`
  - Attribute `capsIntent`
- Suggested Exercises

Python Basics: Strings

- String literals are defined with single or double quotes
- Alternating single/double quotes can be used to print quotes

```
print( "Hello World!" == 'Hello World!' ) #Prints: True
print( '"Hello World!"' ) #Prints: "Hello World!"
print( "'Hello World!'" ) #Prints: 'Hello World!'
```

- String concatenated with plus operator

```
hello = "Hello"
world = 'World'
print('"' + hello + ' ' + world + "!" + '"' ) #Prints: "Hello World!"
```

For more examples: `www.w3schools.com/python/python_strings.asp`

Python Basics: Functions

- Function defined with def keyword
- Indentation defines the body of the function
- Function arguments can have default values
- Function arguments can be specified as key value pairs

```
def my_function(thing1 = 1, thing2 = 2):
    print("thing1 =", thing1, "thing2 =", thing2)

my_function()                        # Prints: thing1 = 1 thing2 = 2
my_function(2,3)                     # Prints: thing1 = 2 thing2 = 3
my_function(thing1 = 4)              # Prints: thing1 = 4 thing2 = 2
my_function(thing2 = 6, thing1 = 5) # Prints: thing1 = 5 thing2 = 6

# Function with a return value
def my_sum(arg1, arg2):
    return arg1 + arg2

print( my_sum(1,2) ) # Prints: 3
print( my_sum("Hello ", "World!") ) # Prints: Hello World!
```

For more examples: `www.w3schools.com/python/python_functions.asp`

Python Basics: Classes

- Python is an object oriented language
  - Objects store data and methods to manipulate that data
- A class stores the blueprint to instantiate an object
- An object is instantiated by calling the class constructor
  - The optional __init__ method instantiates an object

```
class MyClass:
    def __init__(self, x = 5):
        self.x = x
    def printx(self):
        print(self.x) # Print the value of x to the terminal

p1 = MyClass()
p1.x = 10
p1.printx()  # Prints: 10

p2 = MyClass()
p2.printx()  # Prints: 5
p1.printx()  # Prints: 10
```

For more examples: `www.w3schools.com/python/python_classes.asp`

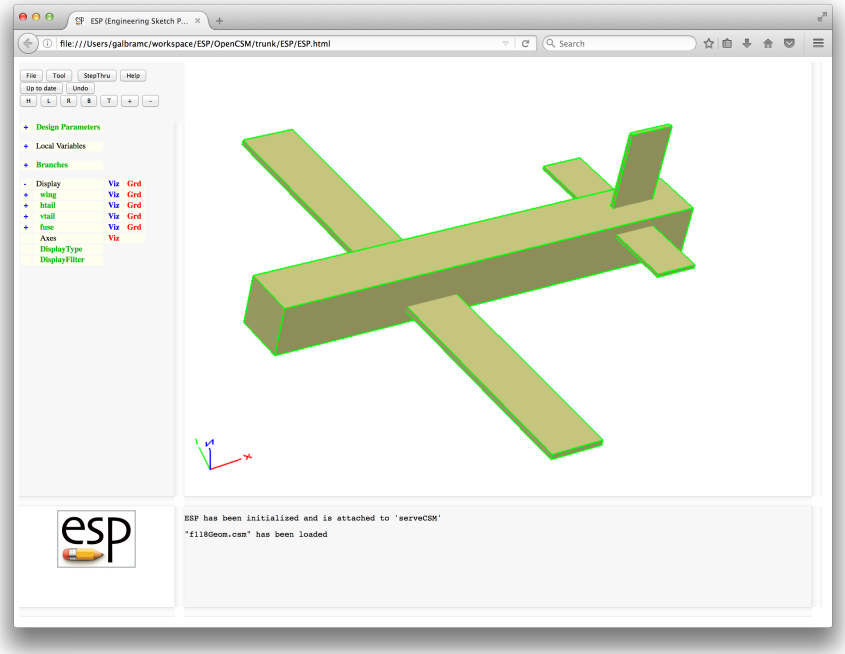
Overview

- Python Basics
  - Strings, Functions, and Classes
- Loading and viewing geometry via pyCAPS
  - pyCAPS.Problem
- Accessing/modifying DESPMTR
  - geometry.despmtr
  - geometry.save, .writeParameters, .readParameters
  - geometry.bodies
  - Value Object Sequence .value Shortcut
- Accessing SET and @values using OUTPMTR
  - geometry.outpmtr
- Directing bodies to AIMs
  - Attribute `capsAIM`
  - Attribute `capsIntent`
- Suggested Exercises

F-118 Boxter Geometry

session02/f118-A.csm

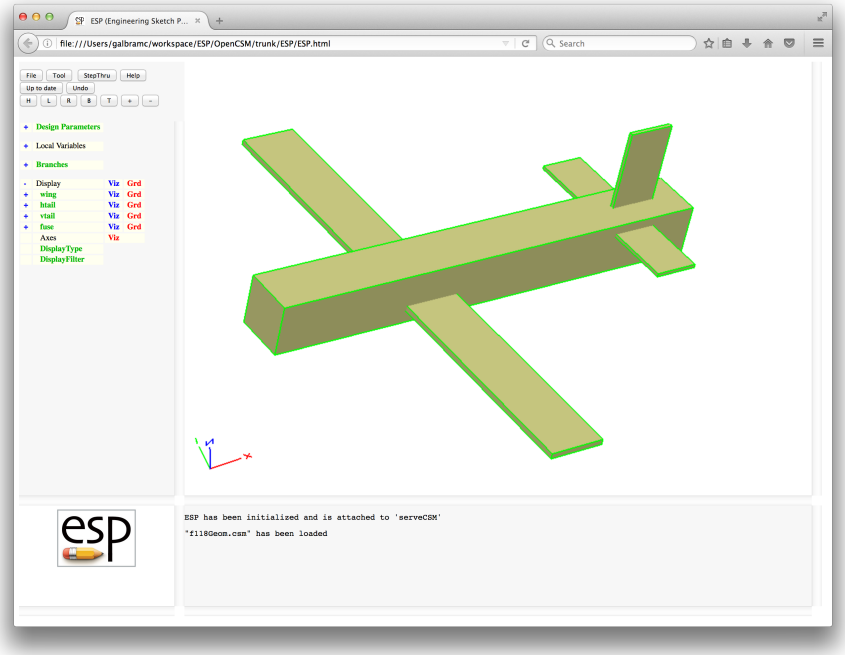
```
# F-118A Boxster

# wing design parameters
DESPMTR   wing:area         4240     # area
DESPMTR   wing:aspect       9.00     # aspect ratio
DESPMTR   wing:thick        0.10     # thickness ratio
DESPMTR   wing:xroot        54.0     # xloc at root LE
DESPMTR   wing:zroot        -5.0     # zloc at root LE

# horizontal tail design parameters
DESPMTR   htail:area        1210     # htail area
DESPMTR   htail:aspect      4.15     # htail aspect ratio
DESPMTR   htail:thick       0.08     # htail thickness
DESPMTR   htail:xroot        145     # xloc of root LE
DESPMTR   htail:zroot          5     # zloc of root LE

# vertical tail design parameters
DIMENSION vtail 5 1
DESPMTR   vtail             "610; \ # 1 vtail area
                            1.80; \ # 2 vtail aspect ratio
                            0.08; \ # 3 vtail thickness
                             150; \ # 4 xloc of root LE
                               9"   # 5 zloc of root LE

# fuselage design parameters
DESPMTR   fuse:length        180     # fuselage length
DESPMTR   fuse:width          20     # width  of fuselage
DESPMTR   fuse:height         20     # height of mid fuselage
```



F-118 Boxter Geometry

session02/f118-A.csm

```
#------------------------------------------------
# set available output parameters
OUTPMTR  wing:wet
OUTPMTR  wing:volume

DIMENSION htail:prop 2 1
OUTPMTR   htail:prop

DIMENSION vtail:prop 2 1
OUTPMTR   vtail:prop

OUTPMTR  fuse:wet
OUTPMTR  fuse:volume

#================================================
# Wing
SET       wing:span      sqrt(wing:aspect*wing:area)
SET       wing:chord     wing:area/wing:span

BOX  wing:xroot  -wing:span/2  wing:zroot   wing:chord  wing:span  wing:chord*wing:thick
SELECT body
    ATTRIBUTE _name   $Wing

SET  wing:wet    @area
SET  wing:volume @volume
```



pyCAPS Problem Object

- pyCAPS.Problem provides the context for a CAPS session
  - Multiple pyCAPS.Problems may be instantiated, but cannot interact
- `problemName` is a directory name for analysis/restart files

session02/f118_1_Geom.py

```
#-------------------------------------------------------------------------------

# Import pyCAPS module
import pyCAPS

#-------------------------------------------------------------------------------

# Load geometry [.csm] file
# This parses the csm file, but does not build geometry
filename = "f118-A.csm"
print ('\n==> Loading geometry from file "'+filename+'"...')
myProblem = pyCAPS.Problem(problemName = "workDir_1_Geom",
                           capsFile = filename,
                           outLevel = 1)
```

Importing Modules: `www.w3schools.com/python/python_modules.asp`

Viewing Geometry with pyCAPS

- Problem.geometry is a pyCAPS.ProblemGeometry class instance
- Visualize with ESP using myProblem.geometry.view
  - Geometry is built just-in-time

session02/f118_1_Geom.py

```
# myProblem.geometry is a class representing bodies on the stack
# Build and view the geometry with ESP
print ('\n==> Bulding and viewing geometry...')
myProblem.geometry.view()
```

Overview

- Python Basics
  - Strings, Functions, and Classes
- Loading and viewing geometry via pyCAPS
  - pyCAPS.Problem
- Accessing/modifying DESPMTR
  - geometry.despmtr
  - geometry.save, .writeParameters, .readParameters
  - geometry.bodies
  - Value Object Sequence .value Shortcut
- Accessing SET and @values using OUTPMTR
  - geometry.outpmtr
- Directing bodies to AIMs
  - Attribute `capsAIM`
  - Attribute `capsIntent`
- Suggested Exercises

Modifying Geometry DESPMTR with pyCAPS

- DESPMTR ↔ geometry.despmtr Sequence
  - Sequence of ValueIn Objects
  - Keys correspond to DESPMTR names in csm file
  - Iterate despmtr.keys(), despmtr.values(), despmtr.items()
- CFGPMTR ↔ geometry.cfgpmtr Sequence
- CONPMTR ↔ geometry.conpmtr Sequence

session02/f118-A.csm

```
# fuselage design parameters
DESPMTR   fuse:length       180    # fuselage length
DESPMTR   fuse:width         20    # width  of fuselage
DESPMTR   fuse:height        20    # height of mid fuselage
```

session02/f118_2_DESPMTR.py

```
# Create an alias to geometry
f118 = myProblem.geometry

# geometry.despmtr is a Sequence of ValueIn Objects of all DESPMTR defined in the csm file
# Set wide fuselage "fuse:width"
f118.despmtr["fuse:width"].value = 60
```

Modifying Geometry DESPMTR with pyCAPS

- DESPMTR ↔ geometry.despmtr Sequence

session02/f118-A.csm

```
# horizontal tail design parameters
DESPMTR   htail:area       1210    # htail area
DESPMTR   htail:aspect     4.15    # htail aspect ratio
DESPMTR   htail:thick      0.08    # htail thickness
DESPMTR   htail:xroot       145    # xloc of root LE
DESPMTR   htail:zroot         5    # zloc of root LE
```

session02/f118_2_DESPMTR.py

```
# Double the htail:area
htail_area = f118.despmtr["htail:area"].value
f118.despmtr["htail:area"].value = htail_area*2

print ("--> old htail:area = ", htail_area)
print ("--> new htail:area = ", f118.despmtr["htail:area"].value)
```

Modifying Geometry DESPMTR with pyCAPS

- Modifying DIMENSIONed DESPMTR

session02/f118-A.csm

```
# vertical tail design parameters
DIMENSION vtail 5 1
DESPMTR   vtail             "610; \ # 1 vtail area
                            1.80; \ # 2 vtail aspect ratio
                            0.08; \ # 3 vtail thickness
                             150; \ # 4 xloc of root LE
                               9"   # 5 zloc of root LE
```

session02/f118_2_DESPMTR.py

```
# Double the vtail area in an array
vtail = f118.despmtr["vtail"].value
vtail[0] *= 2
f118.despmtr["vtail"].value = vtail

# Build and view the geometry with ESP
print ('\n==> Bulding and viewing geometry...')
f118.view()
```

Modifying Geometry DESPMTR with pyCAPS

- Swapping parameter values

session02/f118_2_DESPMTR.py

```
# Build the Canard variant

# Reset the fuselage
f118.despmtr["fuse:width"].value = 20

htail_area = f118.despmtr["htail:area"].value
wing_area  = f118.despmtr["wing:area"].value

# Swap "wing:area" and "htail:area"
f118.despmtr["htail:area"].value = wing_area
f118.despmtr["wing:area"].value  = htail_area/2

# Rebuild and view geometry
print ('\n==> Bulding and viewing geometry...')
f118.view()

# Display all design parameters
print ('\n==> All design parameters')
for key in f118.despmtr.keys():
    print ('--> {:<12}'.format(key), '=', f118.despmtr[key].value)
```

Saving Geometry with pyCAPS

- Modified geometry is saved with geometry.save
  - Available extensions: .egads .stp .step .igs .iges .brep

session02/f118_2_DESPMTR.py

```
# Build and view the geometry with ESP
print ('\n==> Bulding and viewing geometry...')
f118.view()
```

session02/f118_3_Save.py

```
# Build and save geometry
print ('\n==> Bulding and saving geometry...')
f118.save("f118_3_Save_Wide.egads")
```

- View geometry with:
  - serveCSM f118_3_Save_Wide.egads
  - serveCSM f118_3_Save_Canard.egads

Saving Parameters with pyCAPS

- Geometry parameters are saved with geometry.writeParameters and geometry.readParameters

session02/f118_4_writeParameters.py

```
# Double the vtail area in an array
vtail = f118.despmtr["vtail"].value
vtail[0] *= 2
f118.despmtr["vtail"].value = vtail

# Write geometry parameters
paramFile = "f118_4.params"
print ('\n==> Saving parameters to "'+paramFile+'"...')
f118.writeParameters(paramFile)

print ('\n==> Loading 2nd geometry from file "'+filename+'"...')
myProblem2 = pyCAPS.Problem(problemName = "workDir_4_writeParameters_2",
                            capsFile = filename,
                            outLevel = 0)

# Read back in the parmeters
print ('\n==> Reading parameters from "'+paramFile+'"...')
myProblem2.geometry.readParameters(paramFile)

# View the geomtry
print ('\n==> Building and viewing the geometry...')
myProblem2.geometry.view()
```

Accessing pyEGADS bodies with pyCAPS

- pyEGADS bodies and length units are accessed with geometry.bodies (advanced usage)

session02/f118_5_bodies.py

```
# Get a dictionary of pyEGADS bodies and length units
# Keys use the body "_name" attribute or "Body_#".
bodys, units = f118.bodies()

# See $ESP_ROOT/doc/EGADS/egads.pdf for pyEGADS documentation
# See $ESP_ROOT/pyESP/testEGADS/test_egads.py for example uses
print ('\n==> pyEGADS egos...')
for key in bodys.keys():
    print(key, bodys[key])
```

The Value Object Sequence .value shortcut

- Value Objects have several methods
- 90% of usage is modifying .value property
- Sequence of Value Objects provides shortcut to .value property

session02/f118-A.csm

```
# vertical tail design parameters
DIMENSION vtail 5 1
DESPMTR   vtail            "610; \ # 1 vtail area
                           1.80; \ # 2 vtail aspect ratio
                           0.08; \ # 3 vtail thickness
                            150; \ # 4 xloc of root LE
                              9"   # 5 zloc of root LE
```

session02/f118_2_DESPMTR.py

```
# Double the vtail area in an array
vtail = f118.despmtr["vtail"].value
vtail[0] *= 2
f118.despmtr["vtail"].value = vtail
```

session02/f118_5_Shortcut.py

```
# Double the "vtail" area in an array
vtail = f118.despmtr.vtail
vtail[0] *= 2
f118.despmtr.vtail = vtail
```

The Value Object Sequence .value shortcut

- Value Objects have several methods
- 90% of usage is modifying .value property
- Colons in OpenCSM names create "groups" in ESP UI
- pyCAPS Value Object Sequence is "grouped"

session02/f118-A.csm

```
# fuselage design parameters
DESPMTR   fuse:length       180    # fuselage length
DESPMTR   fuse:width         20    # width  of fuselage
DESPMTR   fuse:height        20    # height of mid fuselage
```

session02/f118_6_Shortcut.py

```
# Set wide fuselage "fuse:width" via shortcuts
f118.despmtr["fuse:width"].value = 60
f118.despmtr.fuse.width = 60
f118.despmtr["fuse"].width = 60
f118.despmtr["fuse"]["width"].value = 60

# Double the "htail:area"
htail_area = f118.despmtr["htail"].area
f118.despmtr["htail"].area = htail_area*2
```

```
# Display grouped design parameters
print ('\n==> Grouped parameters')
for group in ["fuse", "wing", "htail"]:
    print ('-->', group)
    for key in f118.despmtr[group].keys():
        print ('    {:<12}'.format(group+":"+key),
               '=', f118.despmtr[group][key].value)
```

Overview

- Python Basics
  - Strings, Functions, and Classes
- Loading and viewing geometry via pyCAPS
  - pyCAPS.Problem
- Accessing/modifying DESPMTR
  - geometry.despmtr
  - geometry.save, .writeParameters, .readParameters
  - geometry.bodies
  - Value Object Sequence .value Shortcut
- Accessing SET and @values using OUTPMTR
  - geometry.outpmtr
- Directing bodies to AIMs
  - Attribute `capsAIM`
  - Attribute `capsIntent`
- Suggested Exercises

Accessing Geometry OUTPMTR with pyCAPS

- OUTPMTR accessed with geometry.outpmtr Sequence of Value Objects

session02/f118-A.csm

```
#-------------------------------------------------
# set available output parameters
OUTPMTR  wing:wet
OUTPMTR  wing:volume

```

```
BOX  wing:xroot  -wing:span/2  wing:zroot   wing:chord  wing:span  wing:chord*wing:thick
SELECT body
    ATTRIBUTE _name   $Wing

SET  wing:wet    @area
SET  wing:volume @volume
```

session02/f118_7_OUTPMTR.py

```
# Create an alias to geometry
f118 = myProblem.geometry

# Build and print OUTPMTRs
print ("--> wing:wet      =", f118.outpmtr["wing:wet"].value )
print ("--> wing:volume   =", f118.outpmtr.wing.volume )
```

Accessing Geometry OUTPMTR with pyCAPS

- OUTPMTR can be DIMENSIONed array/matrix

session02/f118-A.csm

```
DIMENSION htail:prop 2 1
OUTPMTR   htail:prop
```

```
BOX  htail:xroot  -htail:span/2  htail:zroot   htail:chord  htail:span  htail:chord*htail:thick
SELECT body
    ATTRIBUTE _name   $Htail

SET  htail:prop[1] @area
SET  htail:prop[2] @volume
```

session02/f118_7_OUTPMTR.py

```
# Accessing DIMENSIONed OUTPMTR
htail_prop = f118.outpmtr["htail"].prop
print ("--> htail:prop[0] =", htail_prop[0] )
print ("--> htail:prop[1] =", htail_prop[1] )
```

Accessing Geometry OUTPMTR with pyCAPS

- None returned for any OUTPMTR not SET

session02/f118-A.csm

```
DIMENSION vtail:prop 2 1
OUTPMTR   vtail:prop

OUTPMTR  fuse:wet
OUTPMTR  fuse:volume
```

```
SET  vtail:prop[1] @area
# vtail:prop[2] is not set
```

```
# fuse:wet and fuse:volume not set
```

session02/f118_7_OUTPMTR.py

```
# Accessing OUTPMTR that has not been set
print ("--> vtail:prop    =", f118.outpmtr["vtail:prop"].value )
print ()
print ("--> fuse:wet      =", f118.outpmtr["fuse:wet"].value )
print ("--> fuse:volume   =", f118.outpmtr.fuse.volume )
```

Accessing Geometry OUTPMTR with pyCAPS

- Accessing non-OUTPMTR gives CAPS_NOTFOUND error

session02/f118-A.csm

```
#-----------------------------------------------
# set available output parameters
OUTPMTR  wing:wet
OUTPMTR  wing:volume

DIMENSION htail:prop 2 1
OUTPMTR   htail:prop

DIMENSION vtail:prop 2 1
OUTPMTR   vtail:prop

OUTPMTR  fuse:wet
OUTPMTR  fuse:volume

#===============================================
# Wing
SET       wing:span      sqrt(wing:aspect*wing:area)
```

session02/f118_7_OUTPMTR.py

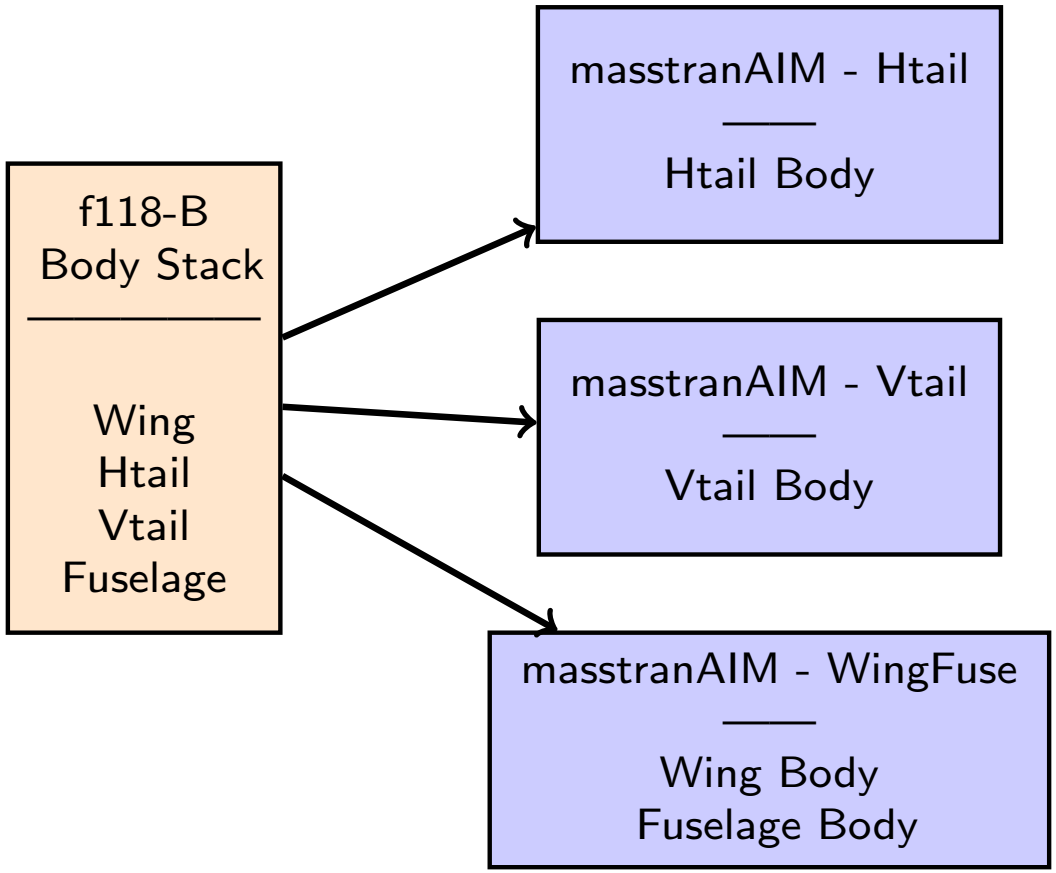
```
# Attempt to get a SET value not defined as OUTPMTR (causes a KeyError)
print ("--> wing:span     =", f118.outpmtr.wing.span )
```

Overview

- Python Basics
  - Strings, Functions, and Classes
- Loading and viewing geometry via pyCAPS
  - pyCAPS.Problem
- Accessing/modifying DESPMTR
  - geometry.despmtr
  - geometry.save, .writeParameters, .readParameters
  - geometry.bodies
  - Value Object Sequence .value Shortcut
- Accessing SET and @values using OUTPMTR
  - geometry.outpmtr
- Directing bodies to AIMs
  - Attribute `capsAIM`
  - Attribute `capsIntent`
- Suggested Exercises

Directing Bodies to AIMs

- `masstranAIM`: Computes mass properties via a shell mesh
  - Compute mass properties for different collections of bodies?
- Single Stack with all Bodies
- Direct bodies in Stack to different AIM instances



Directing Bodies to AIMs with Attribution

- `capsAIM` `BODY` attribute
  - String semicolon separated AIM names
  - AIMs suitable to use the body
- `capsIntent` `BODY` attribute
  - Optional string used to direct bodies to AIM instance
  - String semicolon separated names
  - Multiple bodies may have the same `capsIntent`

session02/f118-B.csm

```
SET       htail:span      sqrt(htail:aspect*htail:area)
SET       htail:chord     htail:area/htail:span

BOX  htail:xroot  -htail:span/2  htail:zroot   htail:chord  htail:span  htail:chord*htail:thick
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $htail;tail
    ATTRIBUTE _name      $Htail
```

Directing Bodies to AIMs with Attribution

- Create AIM with analysis.create
  - aim: The type of the AIM to create
  - capsIntent: List of strings to match with `capsIntent` attr
  - name: A unique name (key in analysis Sequence)
- Empty `capsIntent` loads all bodies with matching `capsAIM`

session02/f118_8_AIM.py

```
# capsAIM == $masstranAIM
masstranAll = myProblem.analysis.create(aim = "masstranAIM", capsIntent="",
                                        name = "All")

# The AIM instance is also available in the Problem.analysis Sequence
assert(masstranAll == myProblem.analysis["All"])

# Show the geometry used by the AIM
print("==> Geometry used by masstranAll instance with no capsIntent")
masstranAll.geometry.view()
```

Directing Bodies to AIMs with Attribution

- Creating masstranAIM with bodies
  `capsAIM` == $masstranAIM and `capsIntent` == $wing

session02/f118-B.csm

```
SET       wing:span      sqrt(wing:aspect*wing:area)
SET       wing:chord     wing:area/wing:span

BOX  wing:xroot  -wing:span/2  wing:zroot   wing:chord  wing:span  wing:chord*wing:thick
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $wing
    ATTRIBUTE _name      $Wing
```

session02/f118_8_AIM.py

```
# capsAIM == $masstranAIM and capsIntent == $wing
myProblem.analysis.create(aim = "masstranAIM", capsIntent="wing",
                          name = "Wing")

# Show the geometry used by the AIM
print("==> Geometry used by Wing instance with capsIntent='wing'")
myProblem.analysis["Wing"].geometry.view()
```

Directing Bodies to AIMs with Attribution

- Creating masstranAIM with bodies
  `capsAIM` == $masstranAIM and `capsIntent` == $tail

session02/f118-B.csm

```
BOX  htail:xroot  -htail:span/2  htail:zroot   htail:chord  htail:span  htail:chord*htail:thick
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $htail;tail
    ATTRIBUTE _name      $Htail
```

```
BOX  vtail[4]  0  vtail[5]   vtail:chord  vtail:chord*vtail[3]  vtail:span
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $vtail;tail
    ATTRIBUTE _name      $Vtail
```

session02/f118_8_AIM.py

```
# capsAIM == $masstranAIM and capsIntent == $tail
masstranTail = myProblem.analysis.create(aim = "masstranAIM", capsIntent="tail",
                                         name = "Tail")

# Show the geometry used by the AIM
print("==> Geometry used by Tail instance with capsIntent='tail'")
masstranTail.geometry.view()
#-----------------------------------------------------------------------------#
```

Directing Bodies to AIMs with Attribution

- Creating masstranAIM with bodies
  `capsAIM` == $masstranAIM and
  (`capsIntent` == $wing or `capsIntent` == $fuse)

session02/f118-B.csm

```
BOX  wing:xroot  -wing:span/2  wing:zroot   wing:chord  wing:span  wing:chord*wing:thick
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $wing
    ATTRIBUTE _name      $Wing
```

```
BOX  0  -fuse:width/2  -fuse:height/2   fuse:length  fuse:width  fuse:height
SELECT body
    ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
    ATTRIBUTE capsIntent $fuse
    ATTRIBUTE _name      $Fuselage
```

session02/f118_8_AIM.py

```
# capsAIM == $masstranAIM and (capsIntent == $wing or capsIntent == $fuse)
myProblem.analysis.create(aim = "masstranAIM", capsIntent=["wing","fuse"],
                          name = "WingFuse")
```

Suggested Exercises

Fix f118-A.csm

- `SET` fuse:wet and fuse:volume in session02/f118-A.csm
- Add wing:span as OUTPMTR in session02/f118-A.csm
- Rerun session02/f118_7_OUTPMTR.py

Custom f118-A.csm

- Customize the f118-A.csm with setGeometryVal
  - Start from a copy of session02/f118_2_DESPMTR.py

Custom masstran analysis

- Load wing, htail and fuselage into a masstranAIM
  - Start from a copy of session02/f118_8_AIM.py

- Create your own (optionally share it `galbramc@mit.edu`)