

Computational Aircraft Prototype Syntheses



Training Session 2

CAPS Geometry

ESP v1.22

Marshall Galbraith

galbramc@mit.edu

Massachusetts Institute of Technology

Bob Haimes

haimes@mit.edu

John F. Dannenhoffer, III

jfdannen@syr.edu

Syracuse University

- 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`
- Phasing
- Suggested Exercises

- 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

- 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 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

- 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`
- Phasing
- Suggested Exercises

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

```

CFGPMTR nrow            1
DIMENSION vtail nrow 6
DESPMTR vtail           "610; \ # 1 vtail area
                        1.80; \ # 2 vtail aspect ratio
                        0.08; \ # 3 vtail thickness
                        150; \ # 4 xloc of root LE
                        0; \ # 5 yloc of root LE
                        9"  # 6 zloc of root LE

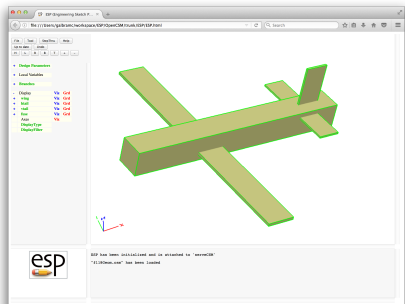
```

fuselage design parameters

```

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

```



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

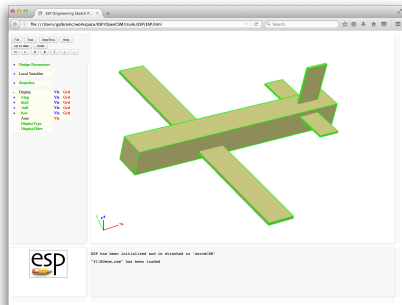
BOX wing:xroot -wing:span/2 wing:zroot wing:chord wing:span wing:chord*wing:thick

SELECT body

ATTRIBUTE _name \$Wing

SET wing:wet @area

SET wing:volume @volume



- `pyCAPS.Problem` 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
- `outLevel` sets verbosity (0-off, 1-default, 2-debug)

session02/1_f118_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+'...')  
capsProblem = pyCAPS.Problem(problemName = "design_f118",  
                             capsFile = filename,  
                             outLevel = 1)
```

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

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

session02/1_f118_Geom.py

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

- 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`
- Phasing
- Suggested Exercises

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

session02/2_f118_DESPMTR.py

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

- DESPMTR \leftrightarrow 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
```

session02/2_f118_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 DIMENSIONed DESPMTR
 - Modify copy of array values

session02/f118-A.csm

```
# vertical tail design parameters
CFGPMTR   nrow          1
DIMENSION vtail nrow 6
DESPMTR   vtail          "610; \ # 1 vtail area
                        1.80; \ # 2 vtail aspect ratio
                        0.08; \ # 3 vtail thickness
                        150; \ # 4 xloc of root LE
                        0; \ # 5 yloc of root LE
                        9"  # 6 zloc of root LE
```

session02/2_f118_DESPMTR.py

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

- Swapping parameter values

session02/2_f118_DESPMTR.py

```
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
```

- Create a 2nd vertical tail

session02/f118-A.csm

```
CFGPMTR      nrow          1
DIMENSION vtail nrow 6
DESPMTR      vtail
              "610; \ # 1 vtail area
              1.80; \ # 2 vtail aspect ratio
              0.08; \ # 3 vtail thickness
              150; \ # 4 xloc of root LE
              0; \ # 5 yloc of root LE
              9" # 6 zloc of root LE
```

```
PATBEG i vtail.nrow
SET      vtail:span      sqrt(vtail[i,1]*vtail[i,2])
SET      vtail:chord      vtail[i,1]/vtail:span
SET      vtail:thick      vtail:chord*vtail[i,3]

BOX      vtail[i,4]      vtail[i,5]-vtail:thick/2      vtail[i,6]      vtail:chord      vtail:thick      vtail:span
```

session02/2_f118_DESPMTR.py

```
# Make a 2nd vertical tail
vtail = [vtail[:,vtail[:,1]]
f118.cfgpmtr["nrow"].value = 2
vtail[0][4] = -f118.despmtr["fuse:width"].value/2*0.8
vtail[1][4] = f118.despmtr["fuse:width"].value/2*0.8
f118.despmtr["vtail"].value = vtail

# create a matrix
# change the CSM matrix size
# spanwise tail positions
```

- Display all design parameters

session02/2_f118_DESPMTR.py

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

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

session02/2_f118_DESPMTR.py

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

session02/3_f118_Save.py

```
# Build and save geometry
print ('\n==> Bulding and saving Wide Body geometry...')
f118.save("3_f118_Save_Wide.egads")
```

- View geometry with:
 - serveCSM 3_f118_Save_Wide.egads
 - serveCSM 3_f118_Save_Canard.egads

- 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
CFGPMTR      nrow      1
DIMENSION    vtail nrow 6
DESPMTR      vtail
              "610; \ # 1 vtail area
              1.80; \ # 2 vtail aspect ratio
              0.08; \ # 3 vtail thickness
              150; \ # 4 xloc of root LE
              0; \ # 5 yloc of root LE
              9"  # 6 zloc of root LE
```

session02/2_f118_DESPMTR.py

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

session02/4_f118_Shortcut.py

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

- 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
```

session02/4_f118_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)
```

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

session02/5_f118_writeParameters.py

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

# 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 + '"...')
capsProblem2 = pyCAPS.Problem(problemName = "design_f118",
                              capsFile = filename,
                              phaseName = "4_writeParameters_2",
                              outLevel = 0)

# Read back in the parameters
print ('\n==> Reading parameters from "' + paramFile + '"...')
capsProblem2.geometry.readParameters(paramFile)
```

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

session02/6_f118_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])
```

- 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`
- Phasing
- Suggested Exercises

- OUTPMTR accessed with geometry.outpmtr Sequence of ValueOut 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
```

session02/7_f118_OUTPMTR.py

```
# Create an alias to geometry  
f118 = capsProblem.geometry  
  
# Build and print OUTPMTRs  
print ("--> wing:wet      =", f118.outpmtr["wing:wet"].value )  
print ("--> wing:volume    =", f118.outpmtr.wing.volume )
```


- 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
```

session02/7_f118_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] )
```

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

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

session02/7_f118_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 non-OUTPMTR raises KeyError/AttributeError

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)
```

session02/7_f118_OUTPMTR.py

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

- 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`
- Phasing
- Suggested Exercises

- Phase stores the entire state of a Problem object
 - Should represent a “decision”
- Created by specifying phaseName
- Finalized with closePhase

session02/8_f118_Phasing.py

```
filename = "f118-A.csm"
print ('\n==> Loading geometry from file "' + filename + '"...')
capsProblem = pyCAPS.Problem(problemName = "design_f118",
                             capsFile = filename,
                             phaseName = "WideBody",
                             outLevel = 0)
```

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

```
# Closing the phase marks it as complete and static
capsProblem.closePhase()
```

- New Phase can branch off of existing Phase with phaseStart
- “Canard” phase inherits state from “WideBody” phase
- New phase may use updated csm file

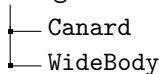
session02/8_f118_Phasing.py

```
# Open a new phase
capsProblem = pyCAPS.Problem(problemName = "design_f118",
                             phaseStart = "WideBody",
                             phaseName = "Canard",
                             outLevel = 0)

print ('\n==> Bulding and viewing geometry from WideBody phase...')
f118.view()
```

- Problem Directory contains phases

design_f118



- phaseUtil used to manipulate phases
- Do not manipulate phases manually!

```
Usage: phaseUtil prPath                -- list Phases
      phaseUtil prPath -l phName        -- show lock owner
      phaseUtil prPath -r phName        -- remove lock
      phaseUtil prPath -c phName newPath -- copy Phase
      phaseUtil prPath -p phName newName -- copy Phase in Problem
      phaseUtil prPath -d phName        -- delete Phase
      phaseUtil prPath -f newPath       -- full copy of Problem
```

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 session02/7_f118_OUTPMTR.py

Custom f118-A.csm

- Customize the f118-A.csm with geometry.despmtr
 - Start from a copy of session02/2_f118_DESPMTR.py

Custom phase

- Create a 3rd phase starting from “Canard” phase
 - Start from a different file than 8_f118_Phasing.py
- Create your own (optionally share it galbramc@mit.edu)