

Computational Aircraft Prototype Syntheses



Training Session 3

CAPS Analysis

ESP v1.18

Marshall Galbraith

galbramc@mit.edu

Massachusetts Institute of Technology

Bob Haines

haines@mit.edu

John F. Dannenhoffer, III

jfdannen@syr.edu

Syracuse University

- Python Basics
 - Lists, Tuples and Dictionaries
- Accessing/modifying analysis values
 - `set/getAnalysisVal`
- Analysis execution and outputs
 - `pre/postAnalysis`
 - `getAnalysisOutVal`
- DIRTY/CLEAN process
 - Tracking changes to inputs that impact outputs
- `capsGroup` attribute
 - Connecting geometry with analysis properties
- Suggested Exercises

- List: ordered changeable collection. Allows duplicates.
- Created with square brackets []

```
thislist = ["apple", "banana", "banana", "cherry"]
print(thislist) # Prints ["'apple', 'banana', 'banana', 'cherry']"
print(thislist[0]) # Prints 'apple'
print(thislist[-1]) # Prints 'cherry'

thislist[1] = "pear" # Change the banana to a pear
thislist.append(42) # Append 42 to the end of the list

for fruit in thislist: # Print each fruit (and 42) in the list via item
    print(fruit)

for i in range(len(thislist)): # Print each fruit (and 42) in the list via index
    print(thislist[i])
```

For more examples: www.w3schools.com/python/python_lists.asp

- Tuple: ordered unchangeable collection. Allows duplicates.
- Created with parenthesis ()

```
thistuple = ("apple", "banana", "banana", "cherry")
print(thistuple) # Prints "('apple', 'banana', 'banana', 'cherry')"
print(thistuple[0]) # Prints 'apple'
print(thistuple[-1]) # Prints 'cherry'

for fruit in thistuple: # Print each fruit in the tuple via item
    print(fruit)

for i in range(len(thistuple)): # Print each fruit in the tuple via index
    print(thistuple[i])

thistuple[1] = "pear" # Runtime error
```

For more examples: www.w3schools.com/python/python_tuples.asp

- Dictionary: unordered unchangeable indexed collection. No duplicates.
- Created with curly brackets { }
- key - value pairs separated by colon

```
thisdict = {
    "status" : "Don't panic",
    "Dolphin": "So long, and thanks for all the fish.",
    42       : "The answer"
    "Years of thought" : 7.5e6
}
print(thisdict["status"]) # Prints 'Don't panic'
print(thisdict[42]) # Prints 'The answer'

# Modify the answer
thisdict[42] = "The answer to the great question... Of life, the universe and everything..."

for key in thisdict: # Print each key in the dict
    print(key)

for key in thisdict: # Print each value in the dict
    print(thisdict[key])
```

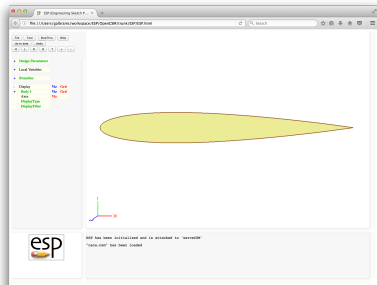
More examples: www.w3schools.com/python/python_dictionaries.asp

- Python Basics
 - Lists, Tuples and Dictionaries
- Accessing/modifying analysis values
 - set/getAnalysisVal
- Analysis execution and outputs
 - pre/postAnalysis
 - getAnalysisOutVal
- DIRTY/CLEAN process
 - Tracking changes to inputs that impact outputs
- capsGroup attribute
 - Connecting geometry with analysis properties
- Suggested Exercises

session03/naca.csm

```
# NACA design paramters
DESPMTR  thick  0.12  #frac of local chord
DESPMTR  camber 0.00  #frac of local chord

# Construct the airfoil
UDPRIM  naca  Thickness thick  Camber camber
ATTRIBUTE capsAIM $xfoilAIM;tsfoilAIM
```



- Analysis values of an AIM are set or accessed with set/getAnalysisVal
- Available AIM analysis values listed in documentation

session03/xfoil_1_AnalysisVal.py

```
# Load xfoil aim
print ("\n==> Loading xfoilAIM")
xfoil = myProblem.loadAIM(aim = "xfoilAIM",
                          analysisDir = "workDir_xfoil_1")

# Set Mach number
xfoil.setAnalysisVal("Mach", 0.5 )

# Print the modified mach number
mach = xfoil.getAnalysisVal("Mach")
print ("\n==> Modified Mach =", mach)
```

- Analysis values can be tuples/lists and toggled

session03/xfoil_1_AnalysisVal.py

```
# Print the default value of None
print("\n==> Default Alpha =", xfoil.getAnalysisVal("Alpha"))

# Set Alpha number
xfoil.setAnalysisVal("Alpha", 2.5 )
print("\n==> Modified Alpha =", xfoil.getAnalysisVal("Alpha"))

# Set list of Alpha
xfoil.setAnalysisVal("Alpha", [0.0, 3.0, 5.0, 7.0, 8.0] )
print("\n==> Modified Alpha =", xfoil.getAnalysisVal("Alpha"))

# Unset Alpha back to None
xfoil.setAnalysisVal("Alpha", None )
print("\n==> Unset Alpha =", xfoil.getAnalysisVal("Alpha"))
```

- Available analysis values in xfoil AIM documentation

- Python Basics
 - Lists, Tuples and Dictionaries
- Accessing/modifying analysis values
 - set/getAnalysisVal
- Analysis execution and outputs
 - pre/postAnalysis
 - getAnalysisOutVal
- DIRTY/CLEAN process
 - Tracking changes to inputs that impact outputs
- capsGroup attribute
 - Connecting geometry with analysis properties
- Suggested Exercises

- Load AIM and set analysis values

session03/xfoil_2_Analysis.py

```
print ("\n==> Loading xfoilAIM")
xfoil = myProblem.loadAIM(aim = "xfoilAIM",
                          analysisDir = "workDir_xfoil_2")

print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.setAnalysisVal("Mach", 0.5 )
xfoil.setAnalysisVal("Re", 1.0E6 )

# Set list of Alpha
xfoil.setAnalysisVal("Alpha", [0.0, 3.0, 5.0, 7.0, 8.0] )
```

- Run preAnalysis to generate xfoil input files

```
session03/workDir_xfoil_2/caps.xfoil
```

```
session03/workDir_xfoil_2/xfoilInput.txt
```

```
session03/xfoil_2_Analysis.py
```

```
print ("\n==> Loading xfoilAIM")  
xfoil = myProblem.loadAIM(aim = "xfoilAIM",  
                           analysisDir = "workDir_xfoil_2")
```

```
# Run AIM pre-analysis  
print ("\n==> Running preAnalysis")  
xfoil.preAnalysis()
```

- Execute xfoil in session03/workDir_xfoil_2

session03/xfoil_2_Analysis.py

```
##### Run xfoil #####  
print ("\n\n==> Running xFoil.....")  
  
currentDirectory = os.getcwd() # Get current working directory  
os.chdir(xfoil.analysisDir)    # Move into test directory  
  
# Run xfoil via system call  
os.system("xfoil < xfoilInput.txt > Info.out");  
  
os.chdir(currentDirectory)    # Move back to top directory  
#####
```

- CAPS currently does not execute analysis tools
- Driving program responsible for execution

- Run postAnalysis to indicate completion and parse output files

session03/xfoil_2_Analysis.py

```
# Run AIM post-analysis
print ("\n==> Running postAnalysis")
xfoil.postAnalysis()
```

- Get results from the analysis with getAnalysisOutVal

```
# Retrieve Alpha, Cl and Cd
print ("\n==> Retrieve analysis results")
Alpha = xfoil.getAnalysisOutVal("Alpha")
Cl     = xfoil.getAnalysisOutVal("CL")
Cd     = xfoil.getAnalysisOutVal("CD")

print()
print("--> Alpha =", Alpha)
print("--> Cl   =", Cl)
print("--> Cd   =", Cd)
```

- Helper function for running the analysis

session03/xfoil_3_Analysis.py

```
def run_xfoil(xfoil):
    # Run AIM pre-analysis
    print ("\n==> Running preAnalysis")
    xfoil.preAnalysis()

    ##### Run xfoil #####
    print ("\n\n==> Running xFoil.....")

    currentDirectory = os.getcwd() # Get current working directory
    os.chdir(xfoil.analysisDir)    # Move into test directory

    # Run xfoil via system call
    os.system("xfoil < xfoilInput.txt > Info.out");

    os.chdir(currentDirectory)    # Move back to top directory
    #####

    # Run AIM post-analysis
    print ("\n==> Running postAnalysis")
    xfoil.postAnalysis()
```

- Compute polar for a range of angles of attack

session03/xfoil_3_Analysis.py

```
print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.setAnalysisVal("Mach", 0.5 )
xfoil.setAnalysisVal("Re", 1.0E6 )

# Set list of Alpha
xfoil.setAnalysisVal("Alpha", [0.0, 3.0, 5.0, 7.0, 8.0] )

# Run xfoil
run_xfoil(xfoil)

# Retrieve Alpha, Cl and Cd
print ("\n==> Retrieve analysis results")
Alpha = xfoil.getAnalysisOutVal("Alpha")
Cl     = xfoil.getAnalysisOutVal("CL")
Cd     = xfoil.getAnalysisOutVal("CD")

print()
print("--> Alpha =", Alpha)
print("--> Cl   =", Cl)
print("--> Cd   =", Cd)
```

- Switch to compute polar for a range of lift coefficients

session03/xfoil_3_Analysis.py

```
# Unset Alpha, otherwise it will be included in the next analysis
xfoil.setAnalysisVal("Alpha", None )

# Set specific Cl values instead
xfoil.setAnalysisVal("CL", [0.0, 0.1, 0.15, 0.3, 0.4] )

# Run xfoil
run_xfoil(xfoil)

# Retrieve Alpha, Cl and Cd
print ("\n==> Retrieve analysis results")
Alpha = xfoil.getAnalysisOutVal("Alpha")
Cl     = xfoil.getAnalysisOutVal("CL")
Cd     = xfoil.getAnalysisOutVal("CD")

print()
print("--> Alpha =", Alpha)
print("--> Cl   =", Cl)
print("--> Cd   =", Cd)
```

- Setup analysis values

session03/xfoil_4_Camber.py

```
# Load geometry [.csm] file
filename = "naca.csm"
print ("\n==> Loading geometry from file \""+filename+"\"")
naca = myProblem.loadCAPS(filename)

# Load xfoil aim
print ("\n==> Loading xfoilAIM")
xfoil = myProblem.loadAIM(aim = "xfoilAIM",
                          analysisDir = "workDir_xfoil_4")

print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.setAnalysisVal("Mach", 0.5 )
xfoil.setAnalysisVal("Re", 1.0E6 )

# Set list of Alpha
xfoil.setAnalysisVal("Alpha", [0.0, 1.0, 3.0] )
```

- Execute sequence of cambers

session03/xfoil_4_Camber.py

```
# List of cambers to analyze
Camber = [0.00, 0.01, 0.04, 0.07]

Alpha = []; Cl = []; Cd = []
for camber in Cambers:
    # Modify the camber
    naca.setGeometryVal("camber", camber)

    # Run xfoil
    run_xfoil(xfoil)

    # Append Alpha, Cl and Cd
    print ("\n==> Retrieve analysis results")
    Alpha.append(xfoil.getAnalysisOutVal("Alpha"))
    Cl.append(xfoil.getAnalysisOutVal("CL"))
    Cd.append(xfoil.getAnalysisOutVal("CD"))

print()
print("--> Cambers =", Cambers)
print("--> Alpha   =", Alpha)
print("--> Cl     =", Cl)
print("--> Cd     =", Cd)
```

- Python Basics
 - Lists, Tuples and Dictionaries
- Accessing/modifying analysis values
 - set/getAnalysisVal
- Analysis execution and outputs
 - pre/postAnalysis
 - getAnalysisOutVal
- **DIRTY/CLEAN** process
 - Tracking changes to inputs that impact outputs
- capsGroup attribute
 - Connecting geometry with analysis properties
- Suggested Exercises

- Calling `setGeometryVal` marks geometry as DIRTY
 - Geometry always built just-in-time
- Calling `setAnalysisVal` or `setGeometryVal` marks the AIM as DIRTY
- CAPS does not execute analysis, cannot execute just-in-time
- Errors reported accessing `getAnalysisOutVal` if AIM is DIRTY
- Errors reported running `preAnalysis` if AIM is CLEAN
 - Avoids inefficiencies with unnecessary calls to `preAnalysis`

- Execution without errors

session03/xfoil_5_CleanDirty.py

```
print("\n1. No Errors ", "-"*80)

# Set Mach and Reynolds number
xfoil.setAnalysisVal("Mach", 0.5 )
xfoil.setAnalysisVal("Re", 1.0E6 )

# Set list of Alpha
print("\n==> Setting alpha sequence")
xfoil.setAnalysisVal("Alpha", [0.0, 3.0, 5.0, 7.0, 8.0] )

# Run xfoil
run_xfoil(xfoil)

# Retrieve Cl
Cl = xfoil.getAnalysisOutVal("CL")
print("\n--> Cl      =", Cl)
```

- Trying call `getAnalysisOutVal` with DIRTY AIM due to `setAnalysisVal`

session03/xfoil_5_CleanDirty.py

```
print("\n2. DIRTY AnalysisVal Error ", "-"*80)

# Set a new alphas
print("\n==> Setting new alpha sequence")
xfoil.setAnalysisVal("Alpha", [1.0, 2.0] )

# Try to retrieve Cl without executing pre/postAnalysis
print("\n==> Attempting to get Cl")
try:
    Cl = xfoil.getAnalysisOutVal("CL")
    print("\n--> Cl   =", Cl)
except CAPSError as e:
    print("\n==> CAPSError =", e)
```

- Trying call `getAnalysisOutVal` with DIRTY AIM due to `setGeometryVal`

session03/xfoil_5_CleanDirty.py

```
print("\n3. DIRTY GeometryVal Error ", "-"*80)

# Modify a geometric parameter
print("\n==> Modifying camber")
myProblem.geometry.setGeometryVal("camber", 0.07)

# Try to retrieve Cl without executing pre/postAnalysis
print("\n==> Attempting to get Cl")
try:
    Cl = xfoil.getAnalysisOutVal("CL")
    print("\n--> Cl   =", Cl)
except CAPSError as e:
    print("\n==> CAPSError =", e)
```

- Trying to call `getAnalysisOutVal` without calling `postAnalysis`

session03/xfoil_5_CleanDirty.py

```
print("\n4. DIRTY pre- but no postAnalysis Error ", "-"*80)

# Modify mach number
print("\n==> Modifying Mach")
xfoil.setAnalysisVal("Mach", 0.3 )

# Run AIM pre-analysis
print ("\n==> Running preAnalysis but not running postAnalysis")
xfoil.preAnalysis()

# Retrieve Cl
print("\n==> Attempting to get Cl")
try:
    Cl = xfoil.getAnalysisOutVal("CL")
    print("\n--> Cl   =", Cl)
except CAPSError as e:
    print("\n==> CAPSError =", e)
```

- Calling preAnalysis with a CLEAN AIM

session03/xfoil_5_CleanDirty.py

```
print("\n5. CLEAN Error ", "-"*80)

# Don't modify any analysis or geometry values

try:
    # Run xfoil
    run_xfoil(xfoil)
except CAPSError as e:
    print("\n==> CAPSError =", e)
```

- Python Basics
 - Lists, Tuples and Dictionaries
- Accessing/modifying analysis values
 - set/getAnalysisVal
- Analysis execution and outputs
 - pre/postAnalysis
 - getAnalysisOutVal
- DIRTY/CLEAN process
 - Tracking changes to inputs that impact outputs
- **capsGroup** attribute
 - Connecting geometry with analysis properties
- Suggested Exercises

capsGroup attribute

- Tags groups of BODY/FACE/EDGE/NODE
 - Entities with same capsGroup value are in the group
- Specific use of capsGroup is AIM dependent

session03/masstran_6_f118_Wing.py

```
filename = "f118-C.csm"  
print ("\n=> Loading geometry from file \""+filename+"\n...")  
myProblem.loadCAPS(filename, verbosity=0)  
  
# Load the masstran aim with the wing  
masstran = myProblem.loadAIM(aim = "masstranAIM",  
                             analysisDir = "workDir_masstranWing",  
                             capsIntent="wing")
```

- Wing FACES tagged with \$wing:faces
- masstranAIM material and properties for "wing:faces"

session03/f118-C.csm

```
# 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
ATTRIBUTE capsGroup   $wing:faces
```

session03/masstran_6_f118_Wing.py

```
# Define material properties
unobtainium = {"density" : 7850}

# Set the material
masstran.setAnalysisVal("Material", ("Unobtainium", unobtainium))

# Define shell property
shell = {"propertyType"      : "Shell",
         "material"          : "Unobtainium",
         "membraneThickness" : 0.2}

# Associate the shell property with capsGroups defined on the geometry
masstran.setAnalysisVal("Property", ("wing:faces", shell))
```

- capsGroups on all FACES

session03/f118-C.csm

```
BOX wing:xroot -wing:span/2 wing:zroot wing:chord wing:span wing:chord*wing:thick
ATTRIBUTE capsGroup $wing:faces
```

```
BOX htail:xroot -htail:span/2 htail:zroot htail:chord htail:span htail:chord*htail:thick
ATTRIBUTE capsGroup $htail:faces
```

```
BOX vtail:xroot 0 vtail:zroot vtail:chord vtail:chord*vtail:thick vtail:span
ATTRIBUTE capsGroup $vtail:faces
```

```
BOX 0 -fuse:width/2 -fuse:height/2 fuse:length fuse:width fuse:height
```

```
SELECT face 1
```

```
ATTRIBUTE capsGroup $fuse:nose
```

```
SELECT face 2
```

```
ATTRIBUTE capsGroup $fuse:tail
```

```
SELECT face 3
```

```
ATTRIBUTE capsGroup $fuse:side
```

```
SELECT face 4
```

```
ATTRIBUTE capsGroup $fuse:side
```

```
SELECT face 5
```

```
ATTRIBUTE capsGroup $fuse:side
```

```
SELECT face 6
```

```
ATTRIBUTE capsGroup $fuse:side
```

- Properties assigned to capsGroups

session03/masstran_7_f118.py

```
# Define material properties
unobtainium = {"density" : 7850}
madeupium   = {"density" : 6890}

# Set the materials
masstran.setAnalysisVal("Material", [{"Unobtainium", unobtainium},
                                      {"Madeupium", madeupium}])

# Define shell properties
shell_1 = {"propertyType" : "Shell",
           "material"      : "unobtainium",
           "membraneThickness" : 0.2}

shell_2 = {"propertyType" : "Shell",
           "material"      : "madeupium",
           "membraneThickness" : 0.3}

# Associate the shell property with capsGroups defined on the geometry
masstran.setAnalysisVal("Property", [{"wing:faces" , shell_1}, {"htail:faces", shell_1},
                                      {"fuse:nose"  , shell_1}, {"fuse:tail"  , shell_1},
                                      {"vtail:faces", shell_2}, {"fuse:side"  , shell_2}])
```

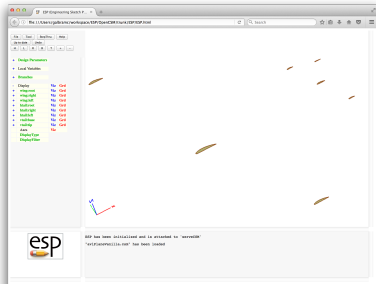
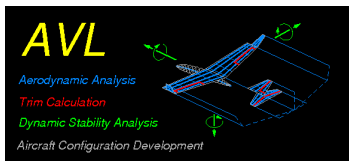
- AVL - Vortex Lattice Method: Geometry defined by airfoils
- capsGroup groups airfoils into surfaces

session03/avlPlaneVanilla.csm

```
UDPRIM      naca Thickness wing:thick Camber wing:camber
SCALE      wing:croot
ROTATEX    90      0      0
TRANSLATE  wing:xroot 0      wing:zroot
ATTRIBUTE  capsGroup      $Wing
```

```
UDPRIM      naca Thickness wing:thick Camber wing:camber
SCALE      htail:croot
ROTATEX    90      0      0
TRANSLATE  htail:xroot 0      htail:zroot
ATTRIBUTE  capsGroup $Htail
```

```
UDPRIM      naca Thickness vtail:thick
SCALE      vtail:croot
TRANSLATE  vtail:xroot 0      vtail:zroot
ATTRIBUTE  capsGroup $Vtail
```



- VLM meshing parameters defined via capsGroups

session03/avl_8_PlaneVanilla.py

```
# Load avl aim
print ("\n=> Loading avlAIM")
avl = myProblem.loadAIM(aim = "avlAIM",
                       analysisDir = "workDir_avl")

print ("\n=> Setting analysis values")
avl.setAnalysisVal("Alpha", 1.0)

# Set meshing parameters for each surface
wing = {"numChord"      : 4,
        "numSpanTotal" : 24}

htail = {"numChord"      : 4,
         "numSpanTotal" : 16}

vtail = {"numChord"      : 4,
         "numSpanTotal" : 10}

# Associate the surface parameters with capsGroups defined on the geometry
avl.setAnalysisVal("AVL_Surface", [{"Wing" , wing },
                                   ("Htail", htail),
                                   ("Vtail", vtail)])
```

Thickness

- Plot airfoil polars for a range of airfoil thicknesses
 - Start from a copy of `session03/xfoil_4_Camber_Plot.py`

New Shells and Material

- Change `capsGroup` value for the top and bottom faces of the fuselage for F-118C (either the same or two different values)
- Starting with `session03/masstran_7_f118.py`, create a new shell and/or material for the newly created `capsGroup(s)`

F-118C CG Location

- Using `session03/masstran_7_f118.py`, create an array of F-118C CG x-locations by modifying the `wing:xroot` location
- Create your own (optionally share it galbramc@mit.edu)