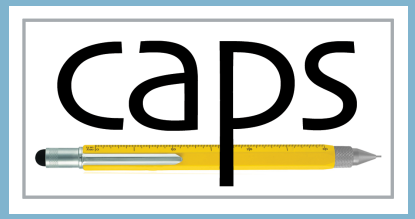
Computational Aircraft Prototype Syntheses



Training Session 3
CAPS Analysis
ESP v1.22

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
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

Python Basics: Lists

- 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`

Python Basics: Tuples

- 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(thislist[i])

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

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

Python Basics: Dictionaries

- Dictionary: unordered changeable 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`

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

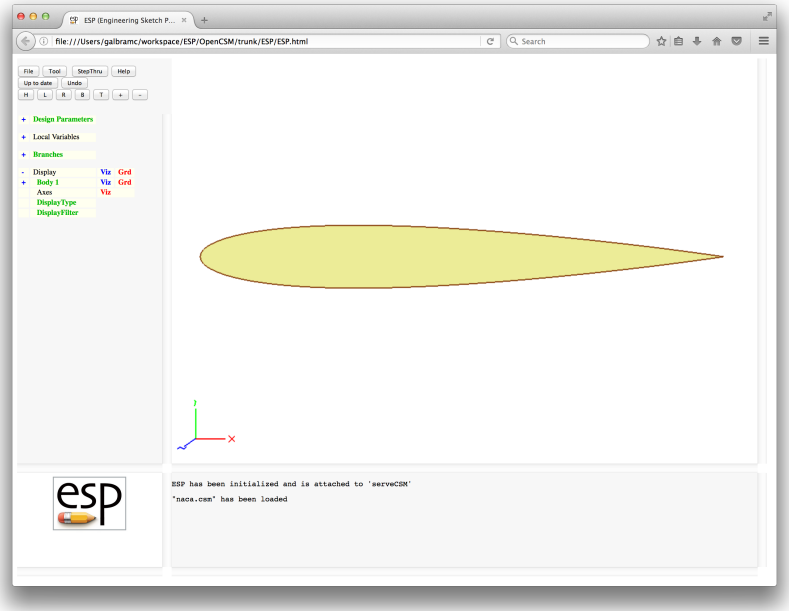
XFOIL Analysis and NACA Airfoil Geometry

XFOIL

- Airfoil performance analysis
- Integral Boundary Layer Theory + Potential Panel Method
  - Subsonic flow (Prandtl–Glauert)
- e^N Transition Model

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;msesAIM;tsfoilAIM
```



Creating analysis AIM with pyCAPS

- Analysis AIM created with Problem.analysis.create
  - Problem.analysis is a Sequence of Analysis Objects
  - create method adds Analysis instance to Problem.analysis Sequence
- "aim" is the string AIM type
- "name" is the unique key Problem.analysis Sequence

session03/01_xfoil_input.py

```
# Load geometry [.csm] file
filename = "naca.csm"
print ("\n==> Loading geometry from file \""+filename+"\"...")
capsProblem = pyCAPS.Problem(problemName = "design_naca",
                             capsFile = filename,
                             outLevel = 1)

# Create xfoil aim
print ("\n==> Creating xfoilAIM")
xfoil = capsProblem.analysis.create(aim = "xfoilAIM",
                                    name = "xfoil")

# Analysis added to capsProblem.analysis Sequence
assert(xfoil == capsProblem.analysis["xfoil"])
```

Setting/accessing analysis inputs with pyCAPS

- Analysis inputs are set/accessed with Analysis.input Sequence of Value Objects

session03/01_xfoil_input.py

```
# Set Mach number
xfoil.input.Mach = 0.5

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

Setting/accessing analysis inputs with pyCAPS

- Analysis values can be tuples/lists and toggled

session03/01_xfoil_input.py

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

# Set Alpha number
xfoil.input.Alpha = 2.5
print("\n==> Modified Alpha =", xfoil.input["Alpha"].value)

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

# Unset Alpha back to None
xfoil.input.Alpha = None
print("\n==> Unset Alpha =", xfoil.input["Alpha"].value)
print()
```

- Available analysis input values in xfoil AIM documentation

xfoil AIM Documentation

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

Running analysis with pyCAPS: Inputs

- Create AIM and set analysis values
  - autoExec used to toggle automatic execution when available

session03/02_xfoil_Analysis.py

```
# Create xfoil aim
print ("\n==> Creating xfoilAIM")
xfoil = capsProblem.analysis.create(aim = "xfoilAIM",
                                    name = "xfoil",
                                    autoExec = False)

print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.input.Mach = 0.5
xfoil.input.Re   = 1.0e6

# Set list of Alpha
xfoil.input.Alpha = [0.0, 3.0, 5.0, 7.0, 8.0]
```

Running analysis with pyCAPS: preAnalysis

- Run preAnalysis to generate xfoil input files

  session03/design_naca/Scratch/xfoil/caps.xfoil
  session03/design_naca/Scratch/xfoil/xfoilInput.txt
- "Scratch" is the default Phase
- "xfoil" in the path is the AIM name

session03/02_xfoil_Analysis.py

```
print ("\n==> Creating xfoilAIM")
xfoil = capsProblem.analysis.create(aim = "xfoilAIM",
                                    name = "xfoil",
                                    autoExec = False)
```

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

Running analysis with pyCAPS: Execution

- Execute xfoil in session03/design_naca/Scratch/xfoil
- Use Analysis.system method for system call!
  - Executes in xfoil.analysisDir
  - Required for "Journaling/Continuation"

session03/02_xfoil_Analysis.py

```
# Run xfoil via system call
print ("\n\n==> Running xFoil......")
xfoil.system("xfoil < xfoilInput.txt > Info.out");
```

Running analysis with pyCAPS: postAnalysis

- Run postAnalysis to indicate completion and parse output files

session03/02_xfoil_Analysis.py

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

- Get outputs with analysis.output Value Object Sequence

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

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

Running analysis with pyCAPS using autoExec

- Create AIM with autoExec = True (default)
- Explicitly execute with runAnalysis (optional)

session03/03_xfoil_AutoExec.py

```
# Create xfoil aim
print ("\n==> Creating xfoilAIM")
xfoil = capsProblem.analysis.create(aim = "xfoilAIM",
                                    name = "xfoil",
                                    autoExec = True)
```

```
# Run analysis (optional)
xfoil.runAnalysis()
```

- Auto execution available for many AIMs
- Not implemented for analysis with MPI
- Auto execution used throughout rest of the training

Multi analysis with the same AIM: Alpha

- Compute polar for a range of angles of attack

session03/04_xfoil_MultiAnalysis.py

```
print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.input.Mach = 0.5
xfoil.input.Re   = 1.0e6

# Set list of Alpha
xfoil.input.Alpha = [0.0, 3.0, 5.0, 7.0, 8.0]

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

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

Multi analysis with the same AIM: CL

- Switch to compute polar for a range of lift coefficients

session03/04_xfoil_MultiAnalysis.py

```
# Unset Alpha, otherwise it will be included in the next analysis
xfoil.input.Alpha = None

# Set specific Cl values instead
xfoil.input.CL = [0.0, 0.1, 0.15, 0.3, 0.4]

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

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

Analysis with geometric changes: Setup

- Setup analysis values

session03/05_xfoil_Camber.py

```
print ("\n==> Setting analysis values")
# Set Mach and Reynolds number
xfoil.input.Mach = 0.5
xfoil.input.Re   = 1.0e6

# Set list of Alpha
xfoil.input.Alpha = [0.0, 1.0, 3.0]
```

Analysis with geometric changes: Camber

- Execute sequence of cambers

session03/05_xfoil_Camber.py

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

Alpha = []; Cl = []; Cd = []
for camber in Cambers:
    # Modify the camber
    naca.despmtr.camber = camber

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

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

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

CAPS CLEAN/DIRTY Process

- Assigning geometry.despmtr marks geometry as DIRTY
  - Geometry always built just-in-time
- Assigning analysis.input or geometry.despmtr marks AIMs as DIRTY
- AIMs with auto execution and autoExec == True will run just-in-time
- Otherwise driver is responsible for executing DIRTY AIMs
- Errors reported accessing analysis.output if AIM is DIRTY
- Errors reported running preAnalysis if AIM is CLEAN
  - Avoids inefficiencies with unnecessary calls to preAnalysis

CAPS CLEAN/DIRTY Example

- Helper function for running the analysis

session03/06_xfoil_CleanDirty.py

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

    # Run xfoil via system call
    print ("\n\n==> Running xFoil......")
    xfoil.system("xfoil < xfoilInput.txt > Info.out");

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

CAPS Successful Execution

- Execution without errors

session03/06_xfoil_CleanDirty.py

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

# Set Mach and Reynolds number
xfoil.input.Mach = 0.5
xfoil.input.Re   = 1.0e6

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

# Run xfoil
run_xfoil(xfoil)

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

CAPS DIRTY Analysis

- Trying call analysis.output with DIRTY AIM due to analysis.input change

session03/06_xfoil_CleanDirty.py

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

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

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

CAPS DIRTY Geometry

- Trying get analysis.ouput with DIRTY AIM due to geometry.despmtr change

session03/06_xfoil_CleanDirty.py

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

# Modify a geometric parameter
print("\n==> Modifying camber")
capsProblem.geometry.despmtr.camber = 0.07

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

CAPS DIRTY postAnalysis

- Trying to call analysis.output without calling postAnalysis

session03/06_xfoil_CleanDirty.py

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

# Modify mach number
print("\n==> Modifying Mach")
xfoil.input.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.output.CL
    print("\n--> Cl    =", Cl)
except pyCAPS.CAPSError as e:
    print("\n==> CAPSError =", e)
```

CAPS CLEAN Error

- Calling preAnalysis with a CLEAN AIM

session03/06_xfoil_CleanDirty.py

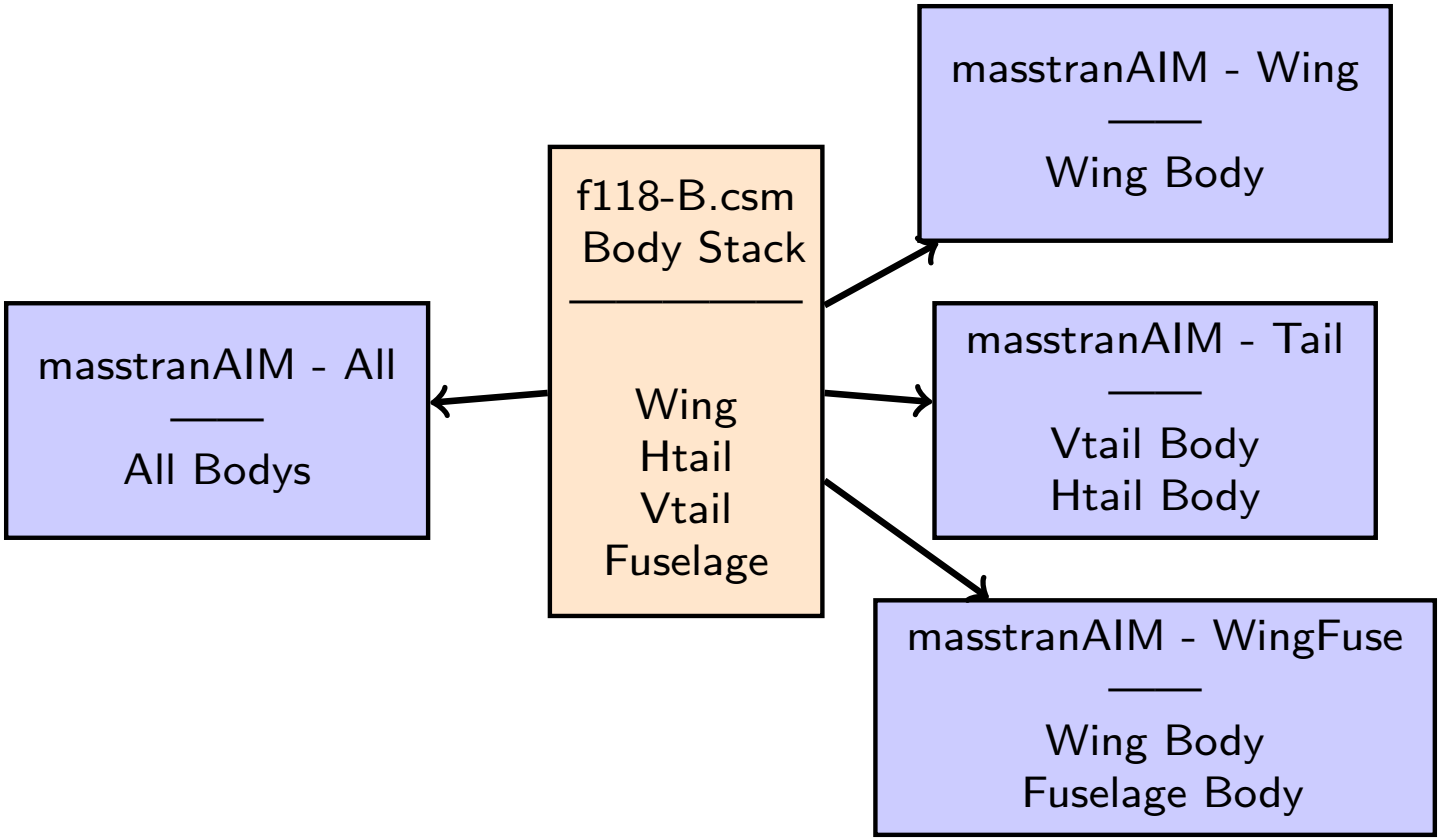
```
print("\n5. CLEAN Error ", "-"*80)

# Don't modify any analysis or geometry values

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

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

Directing Bodies to AIMs

- `masstranAIM`: Computes mass properties via a shell mesh
  - Compute mass properties for 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`

session03/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`

session03/07_masstran_AIM.py

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

# The AIM instance is also available in the Problem.analysis Sequence
assert(masstranAll == capsProblem.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

session03/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
```

session03/07_masstran_AIM.py

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

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

Directing Bodies to AIMs with Attribution

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

session03/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[i,4]  vtail[i,5]-vtail:thick/2  vtail[i,6]   vtail:chord  vtail:thick  vtail:span
    SELECT body
        ATTRIBUTE capsAIM    $masstranAIM;astrosAIM
        ATTRIBUTE capsIntent $vtail;tail
        ATTRIBUTE _name      $Vtail+i
```

session03/07_masstran_AIM.py

```
# capsAIM == $masstranAIM and capsIntent == $tail
masstranTail = capsProblem.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)

session03/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
```

session03/07_masstran_AIM.py

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

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

The capsGroup attribute – Masstran Wing

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

session03/08_masstran_f118_Wing.py

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

# Create a masstran aim with the wing
masstran = capsProblem.analysis.create(aim = "masstranAIM",
                                       name = "masstran",
                                       capsIntent="wing")
```

The capsGroup attribute – Masstran Wing

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

session03/f118-C.csm

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

session03/08_masstran_f118_Wing.py

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

# Set the material
masstran.input.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.input.Property = {"wing:faces": shell}
```

The capsGroup attribute – Masstran F-118C

- capsGroups on all FACEs

session03/f118-C.csm

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

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

```
    SELECT face
        ATTRIBUTE capsGroup  $vtail:faces
    SELECT body
```

```
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
SELECT body
```

The capsGroup attribute – Masstran F-118C

- Properties assigned to capsGroups

session03/09_masstran_f118.py

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

# Set the materials
masstran.input.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.input.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}
```

Overview

- Python Basics
  - Lists, Tuples and Dictionaries
- AIM creation and accessing/modifying analysis values
  - analysis.input
- Analysis execution and outputs
  - pre/postAnalysis
  - analysis.output
- DIRTY/CLEAN process
  - Tracking changes to inputs that impact outputs
- Directing bodies to AIMs
  - Attributes `capsAIM` and `capsIntent`
- `capsGroup` attribute
  - Connecting geometry with analysis properties
- Journaling and Continuation
- Suggested Exercises

Journaling and Continuation

- CAPS records all calls to API functions in a "journal"
- Journal replayed when "re-running" script: Journal Continuation
  - Default enabled with phaseName
  - Default disabled without phaseName
- Script cannot be modified before Continuation

session03/10_masstran_f118_journal.py

Suggested Exercises

Thickness

- Plot airfoil polars for a range of airfoil thicknesses
  - Start from a copy of session03/05_xfoil_Camber_Plot.py

Custom masstran analysis

- Load wing, htail and fuselage into a masstranAIM
  - Start from a copy of session03/07_masstran_f118_AIM.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 session03/09_masstran_f118.py, create a new shell and/or material for the newly created `capsGroup`(s)

Suggested Exercises Cont.

F-118C CG Location

- Using session03/09_masstran_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`)