From MATLAB to Python

Rob Campbell
Research Computing
Purdue University

What's the difference between MATLAB and Python?

$2,250

Differences

Trivial vs ***Significant***

Examples…

- Syntax & keywords
- Libraries (toolboxes)
- IDE (GUI, editor, debugger)
- Empty & Boolean values
- Naming conventions

Examples…

- Value vs. reference
- Collections (arrays, lists)
- Indexing and slicing
- Array expansion
- Behavior of objects

Python emphasizes readability and ease-of-use.

Syntax and Operators

```
% ex0: Syntax, MATLAB
% Ref: wikipedia.org/wiki/Euclidean_algorithm

foo = 252;
bar = 105;

while foo ~= bar

   if foo > bar
       foo = foo - bar;
   else
       bar = bar - foo;
   end
end

disp(foo);
```

```
# ex0: Syntax, Python
# Ref: wikipedia.org/wiki/Euclidean_algorithm

foo = 252
bar = 105

while foo != bar:

   if foo > bar:
       foo = foo - bar
   else:
       bar = bar - foo



print(foo)
```

Functions

```
% ex1: Functions, MATLAB

gcd = euclidgcd(252, 105);
disp(gcd);


function foo = euclidgcd(foo, bar)
   % EUCLIDGCD Return greatest common divisor

   while foo ~= bar

       if foo > bar
           foo = foo - bar;
       else
           bar = bar - foo;
       end
   end
end
```

```
# ex1: Functions, Python




def euclid_gcd(foo, bar):
   """Return greatest common divisor."""

   while foo != bar:

       if foo > bar:
           foo = foo - bar
       else:
           bar = bar - foo

   return foo


gcd = euclid_gcd(252, 105)
print(gcd)
```

Libraries, Built-ins, Toolboxes

```
% ex2: Libraries, MATLAB



foo = gcd(252, 105);
disp(foo);
```

```
# ex2: Libraries, Python

import math

foo = math.gcd(252, 105)
print(foo)
```

Arrays, Vectors, and Matrixes

```
% ex3: Arrays, MATLAB



foo = [252 253 254];
bar = [105 106 107];
baz = gcd(foo, bar);

disp(baz);
```

```
# ex3: Arrays, Python

import numpy as np

foo = np.array([252, 253, 254])
bar = np.array([105, 106, 107])
baz = np.gcd(foo, bar)

print(baz)
```

Indexing

```
% ex4: Indexing, MATLAB

foo = [252 253 254];
bar = [105 106 107];
baz = gcd(foo, bar);

if baz(1) == 21
   disp('Twenty-one!')
end
```

```
# ea4: Indexing, Python

foo = np.array([252, 253, 254])
bar = np.array([105, 106, 107])


if np.gcd(foo, bar)[0] == 21:
   print('Twenty-one!')
```

Loops

```
% ex5: Loops, MATLAB

foo = [252, 253, 254];
bar = [105, 106, 107];
baz = gcd(foo, bar);

for i=1:length(baz)  % 1:3 --> 1, 2, 3

   if baz(i) == 21
       disp('Twenty-one!')
   end
end

if any(baz==21):
      print('Twenty-one again!')
```

```
# ex5: Loops, Python

foo = np.array([252, 253, 254])
bar = np.array([105, 106, 107])
baz = np.gcd(foo, bar)

for i in range(len(baz)):  # range(3) --> 0, 1, 2

   if baz[i] == 21:
       print('Twenty-one!')



if 21 in baz:
     print('Twenty-one again!')
```

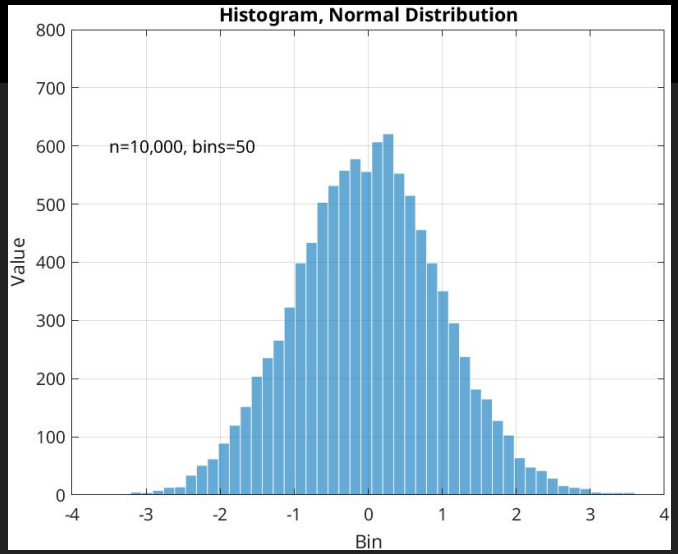
Plots

```
% ex7: Plots, MATLAB



foo = randn(10000, 1);

histogram(foo, 50, 'edgecolor', 'w');
xlabel('Bin');
ylabel('Value');
title('Histogram, Normal Distribution');
text(-3.5, 600, 'n=10,000, bins=50');
axis([-4, 4, 0, 800]);
grid('on');
shg();
```

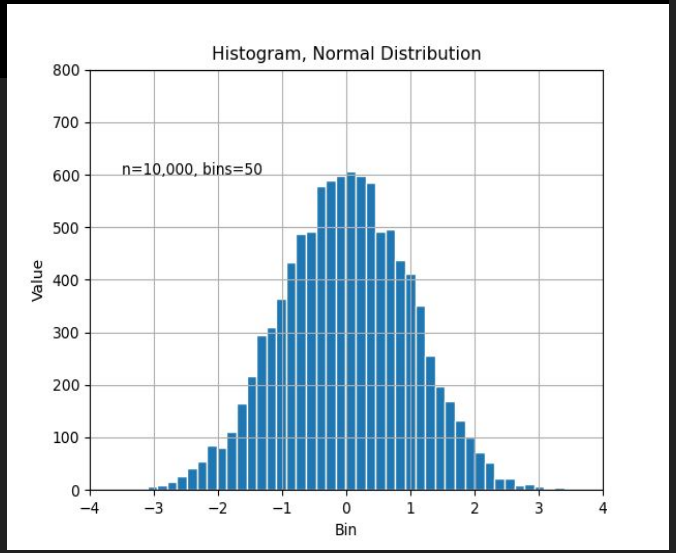



```
# ex7: Plots, Python

import matplotlib.pyplot as plt

foo = np.random.randn(10000, 1)

plt.hist(foo, 50, edgecolor='w')
plt.xlabel('Bin')
plt.ylabel('Value')
plt.title('Histogram, Normal Distribution')
plt.text(-3.5, 600, 'n=10,000, bins=50')
plt.axis([-4, 4, 0, 800])
plt.grid(True)
plt.show()
```



More on Syntax and Keywords

```
% ex8: Misc., MATLAB

foo = 1;      % Boolean value (alt: "true")
bar = [];     % Empty value
baz = 'abc';  % Character array (alt: "abc", string)

% Exception handling
try

   % Expressions
   if foo || (isempty(bar) && strcmp(baz, 'abc'))  %
       disp('Yep');
   else
       disp('Nope');
   end

   % Multiple return values
   [foo, bar] = meshgrid([1 2], [1 2 3]);
   foo = meshgrid([1 2], [1 2 3]);  % only 1st
   [~, foo] = meshgrid([1 2], [1 2 3]);  % only 2nd

catch ME

   % Formatted output
   fprintf('Error: %s\n', ME.identifier);
end
```

```
# ex8: Misc., Python

foo = True    # Boolean value
bar = None    # Empty value
baz = 'abc'   # String

# Exception handling
try:

   # Expressions
   if foo or (bar is None and baz == 'abc'):
       print('Yep')
   else:
       print('Nope')


   # Multiple return values ("unpacking")
   foo, bar = np.meshgrid([1, 2], [1, 2, 3])
   foo = np.meshgrid([1, 2], [1, 2, 3])  # only 1st
   _, foo = np.meshgrid([1, 2], [1, 2, 3])  # only 2nd

except Exception as e:

   # Formatted output
   print(f'Error: {e}')
```

Variables, Values, and References

```
% ex9: Variables, MATLAB

% Single value
foo = 1;
bar = foo;  % Copy value
bar = 99;

disp([foo bar]);  % 1 99

% Values in container (array)
foo = [1, 2 ,3];
bar = foo;     % Make a COPY of entire container
bar(1) = 99;

disp([foo bar]);   % [1 2 3]  [99 2 3]
```

```
# ex9: Variables, Python

# Single value, "immutable"
foo = 1
bar = foo  # Copy value
bar = 99

print(foo, bar)  # 1 99

# Values in container (list), "mutable"
foo = [1, 2, 3]
bar = foo     # Point b at SAME container in memory!
bar[0] = 99

print(foo, bar)   # [99, 2, 3] [99, 2, 3] !!!
```

Passing to a function

```
% ex10: Passing, MATLAB




foo = [1 2 3];
munge(foo);
disp(foo);    % 1 2 3

function munge(bar)
   bar(1) = 99;
end
```

```
# ex10: Passing, Python

def munge(bar):
   bar[0] = 99


foo = [1, 2, 3]
munge(foo)
print(foo)    # 99, 2, 3  !!!
```

Collecations (Containers)

```m
% ex11: Collections, MATLAB

% Same type: array (vector, matrix)
foo = [1 2 3];
bar = [97.1 98.2 99.3];
baz = foo .* bar;  % Can do math, note .*

% Diff types: cell array
foo = {1 'two' 3.33};
bar = {'abc' 'def'};  % Can do same type
bar{1} = 'xyz';  % Note diff brackets

% Named fields: struct
baz = struct('shape', 'round', ...
             'color', 'red', ...
             'count', 42);
baz.age = 99;  % New field
% baz.4 = 'four'  % Restricted field names




disp([foo bar baz])
```

```py
# ex11: Collections. Python

# Same type: numpy's ndarray
foo = np.array([1, 2, 3])
bar = np.array([97.1, 98.2, 99.3])
baz = foo * bar  # Can do math, note *

# Diff types: list
foo = [1, 'two', 3.33]
bar = ['abc', 'def']  # Can do same type
bar[0] = 'xyz'  # Note same brackets

# Named fields: dictionary
baz = {'shape': 'round',
       'color': 'red',
       'count': 42}
baz['age'] = 99  # New field
baz[4] = 'four'  # Unrestricted field names

# Unchangeable: tuple
foo = (1, 'two', 3.33)
bar = foo[2]  # bar=3.33
# foo[2] = 4.44  # ERROR tuples can't change

# Unique: Set
foo = {1, 1, 2, 2, 3, 3}
bar = len(foo)  # bar=3

print(foo, bar, baz)
```

More on Arrays

```
% ex12: More Arrays, MATLAB



% Commas
foo = [1 2 3 4];
bar = [1, 2, 3, 4];  % Commas optional

% Autofill
foo = zeros(10);
bar = ones(20);
baz = 1:5;

% Multidimensional
foo = [1 2 3; 4 5 6];  % Note semicolon
bar = size(foo);
baz = ndims(foo);

% Concatenation
foo = [[1 2] [3 4]];
bar = [[1 2]; [3 4]];

disp(foo)
```

```
# ex12: More Arrays, MATLAB

import numpy as np

# Commas

foo = np.array([1, 2, 3, 4])  # Commas required

# Autofill
foo = np.zeros(10)
bar = np.ones(20)
baz = np.arange(1, 6)

# Multidimensional
foo = np.array([[1, 2, 3], [4, 5, 6]])  # list of lists
bar = foo.shape
baz = foo.ndim

# Concatenation
foo = np.hstack([[1, 2], [3, 4]])  # --> [1, 2, 3, 4]
bar = np.vstack([[1, 2], [3, 4]])  # 2x2

print(foo, bar, baz)
```

Slicing Arrays

```
% ex13: Slicing, MATLAB



foo = [11 22 33 44];

% Slicing makes copies
bar = foo(2:3);  % bar=[22 33]
bar(1) = 0;  % foo unchanged

baz = foo(2:end);  % Slice to end
baz = foo(end);  % Last value: "end means last one"
baz = foo(end-1); % Backwards indexing

disp([foo bar baz]);

foo = [1 2 3; 4 5 6];

% Colon alone, "all"
bar = foo(:, 2);  % 2 5
bar = foo(1, :);  % 1 2 3
bar = foo(:);  % 1x6: 1 4 2 5 3 6  (corrected, was "1 2 3 4 5 6")

% Partial indexes
baz = foo(1);  % 1
baz = foo(2);  % 4
baz = foo(3);  % 2  (corrected, was "5")


disp(foo);
```

```
# ex13: Slicing, MATLAB

import numpy as np

foo = np.array([11, 22, 33, 44])

# Slicing creates "view" into orig
bar = foo[1:2]  # bar=[22, 33]
bar[0] = 0  # foo=[0, 22. 33, 44], bar=[0, 33, 44]  !!!

baz = foo[1:]  # Slice to end
baz = foo[-1]  # Last value: "negative means wrap around"
baz = foo[-2]  # Backwards indexing

print(foo, bar, baz)

foo = np.array([[1, 2, 3], [4, 5, 6]])

# Colon alone, "all"
bar = foo[:, 1]  # [2, 5]
bar = foo[0, :]  # [1, 2, 3]
bar = foo[:]  # 2x3: [[1, 2, 3], [4, 5, 6]]  !!!

# Partial indexes
baz = foo[0]  # [1, 2, 3] !!!  Python: "[0,1]" same as [0][1]
baz = foo[1]  # [4, 5, 6]
baz = foo[2]  # ERROR! Index out of bounds!

print(foo, bar, baz)
```

Expanding Arrays

```
% ex14: Expanding Arrays, MATLAB



foo = [1, 2, 3];

% Append
foo(end+1) = 4;

% Auto expansion
foo(12) = 99;  % 1 2 3 4 0 0 0 0 0 0 0 99




disp(foo);
```

```
# ex14: Expanding Arrays, Python

import numpy as np

foo = np.array([1, 2, 3])

# Append
foo = np.append(foo, 4)

# Auto expansion not allowed
# foo[11] = 99;  # ERROR index out of bounds!

# Manual expansion
foo = np.hstack([foo, np.zeros(8)])
foo[11] = 99  # [1, 2, 3, 4, 0, 0, 0, 0, 0, 0, 0, 99]

print(foo)
```

Classes and Objects

```
classdef ex15moreoop < ex6oop  % Inheritance
   properties (Access = protected)
       baz  % Create protected attribute
   end
   methods
       function obj = ex15moreoop(foo, bar)
           obj = obj@ex6oop(foo, bar);  % Call super constructor
           obj.describe();
       end

       function baz = process(obj)
           baz = process@ex6oop(obj);  % Call super method
           obj.baz = baz;
       end
   end
   methods (Access = protected)
       function describe(obj)  % Protected method
           fprintf('foo: %d, bar: %d\n', length(obj.foo), ...
                                         length (obj.bar))
       end
   end
   methods (Static)
       function example = test()  % Static method
           example = ex15moreoop([252 253], [105 106]);
           gcd = example.process();
           example.check(gcd);
       end
   end
end
```

```
class Example15(Example6):  # Inheritance
   def __init__(self, foo, bar):
       super().__init__(foo, bar)  # Call super constructor
       self._describe()
       self._baz = None  # Protected attribute

   def process(self):
       self._baz = super().process()  # Call super method
       return self._baz

   def _describe(self):  # "Protected" method
       print(f'foo: {len(self.foo)}, bar: {len(self.bar)}')

   @staticmethod  # Static method
   def test():
       example = Example15([252, 253], [105, 106])
       gcd = example.process()
       example.check(gcd)
       return example

class Another():  # Mult classes in same file
   def __init__(self):
       print('Hi')


if __name__ == "__main__":  # Add'l code in same file
   example = Example15.test()
   example.qux = 3.14  # Create a new attribute
   print(example.__dict__)
   Another()
```

Not covered…

- Unpacking
- List comprehension ("one liner code") (dictionaries, etc.)
- Code/module/library search paths
- IDE (editor-debugger, "MATLAB GUI")
- Command line
- Naming conventions
- Building GUI applications
- Compiled code
- Packaging
- Literate computing (notebooks) (for research software engineering)
- Machine learning
- Multiprocessing (parallel computing)

…but most differences are trivial!

Links…

Python Built-in Functions	docs.python.org/3/library/functions
MATLAB Built-in Functions	www.mathworks.com/help/matlab/referencelist.html?type=function&category=index&s_tid=CRUX_lftnav_function_index
NumPy for MATLAB users	numpy.org/doc/stable/user/numpy-for-matlab-users
NumPy: the absolute basics for beginners	numpy.org/devdocs/user/absolute_beginners
Repo	github.com/rcpurdue/mat2py