

# An Overview of Python Programming

## Lesson 3: Iteration and Lists

Nick Featherstone  
[feathern@colorado.edu](mailto:feathern@colorado.edu)

# Outline

- Lists
- Tuples & Dictionaries
- Loops

# Lists in Python

- Multiple values can be grouped together into lists
- Just use square brackets [ ]
- Values can have different datatypes

```
a = 1.0
b = [ 1, 2, a, 4]
print(b[0])
print(b[2])
print(b)
```

Indexed starting with '0'

```
print(b)
print (b[2] is a)
a = 2
print ( b[2] is a)
print ( b[2] , a)
```

memory management: values  
copied – once they have to be...

# Nested (Multi-dimensional) Lists

- We can have lists of lists:
- Indexing uses two square brackets

```
a = [ 1 , 2]  
b = [ 3 , 4]  
c = [ a , b , 5 ]
```

**Note:**

- `c[0]` and `c[1]` are 2-element lists
- `c[2]` is a scalar.

```
print( c[ 0 ] )      # a  
print( c[ 1 ] )      # b  
print( c[ 0 ][ 0 ] ) # a[0]  
print( c[ 0 ][ 1 ] ) # a[1]  
print( c[ 1 ][ 0 ] ) # b[0]  
print( c[ 1 ][ 1 ] ) # b[1]  
print( c[ 2 ] )      # 5
```

# Nested Lists: Memory

- Be careful!
- Python **does not** automatically **copy** lists...

```
a = [ 1 , 2 ]  
b = [ 3 , 4 ]  
c = [ a , b ]
```

```
print( a[ 0 ][ 0 ] , c[ 0 ][ 0 ] )  
c[ 0 ][ 0 ] = 4  
print( a[ 0 ][ 0 ] , c[ 0 ][ 0 ] )
```

# Cloning Lists

- If we want distinct copies, use the slice operator “ : ”

```
a = [ 1, 2]
b = a
b[0] = 5
print(a[0], b[0])
```

*Aliased!*

```
a = [ 1, 2]
b = a[:]
b[0] = 5
print( a[0], b[0] )
```

```
a = [ 1, 2]
b = [ 3, 4]
c = [ a[:], b[:] ]
```

```
print( a[ 0 ][ 0 ], c[ 0 ][ 0 ] )
c[ 0 ][ 0 ] = 4
print( a[ 0 ][ 0 ], c[ 0 ][ 0 ] )
```

# Sublists

- We can **copy** a portion of a list using the slice operator

```
a = [ 1 , 2 , 3 , 4 , 5 ]  
b = a[ 2 : 4 ]  
print( len( b ) )  
print( b[ 0 ] , b[ 1 ] )
```

*len function*

returns number of  
elements in a list

## Note:

- b is [ a[ 2 ] , a[ 3 ] ]
- b *is not* [ a[ 2 ] , a[ 3 ] , a[ 4 ] ]

# Lists and functions

- Avoid unwanted side-effects by passing list clones to a function

```
def mod0( a ):  
    a[ 0 ] = 2
```

Side Effect

```
b = [ 0 , 0 ]  
mod0( b )  
print( b ) # [ 0 , 2 ]
```

No Side Effect

```
b = [ 0 , 0 ]  
mod0( b[ : ] )  
print( b ) # [0, 0]
```



# append & del

- The **append method** grows a list
- The **del** statement deletes elements or sublists

```
a = [ ] # init empty list
a.append( 1 )
print( len( a ) , a )
a.append( 4 )
print( len( a ) , a )
a.append( 8 )
print( len( a ) , a )
```

```
a = [ 4 , 8 , 12 , 13 ]
print( a )
del a[ 0 : 2 ]
print( a )
del a[ 0 ]
print( a )
```

# List Initialization: Replication

- Occasionally useful to initialize a list with known value
- Use the \* operator to replicate values from an existing list or list expression

1-dimensional list

```
a = [ 1 , 2 ]  
b = 3 * a  
print( b )
```

b is [ 1, 2, 1, 2, 1, 2 ]

Nested list

```
a = [ 1 , 2 ]  
b = 3*[ a ]  
print( b )
```

b is [ [ 1, 2 ], [ 1, 2 ], [ 1, 2 ] ]

# List Initialization: Replication

- Extends naturally to higher dimensions

```
a = [ [ 1 , 2 ] , [ 3 , 4 ] ]  
b = 2 * a  
print( b )
```

b is [ [1, 2], [3, 4], [1, 2], [3, 4] ]

```
a = [ [ 1,2], [ 3, 4] ]  
b = 2*[a]  
print(b)
```

b is [ [ [1, 2], [3, 4] ], [ [1, 2], [3, 4] ] ]

# List-like Objects: Tuples

- Similar to Lists, but immutable (can't change values)
- Use ( ) instead of [ ] during creation (**only**)

```
a = ( 1 , 2 )  
print( a[ 0 ] )  
a[ 0 ] = 2 # not allowed
```

```
a = ( 1 , 2 )  
b = ( 3 , 4 )  
c = [ a , b ]  
c[ 0 ] = 1      # OK – modifying list c  
c[ 1 ][ 0 ] = 2 # not OK – modifying tuple element
```

*Can have lists of tuples*

# Tuple Assignment

- Useful Python feature
- tuple *expression* 1 = tuple OR tuple expression 2
- values on right mapped 1-to-1 to values on left

```
( a, b, c ) = (1, 2, 3)
```

Create a,b,c and assign them values

```
tmp = a  
a = b  
b = tmp
```



```
( a , b ) = ( b , a )
```

Swap values

# List-like Objects: Dictionaries

- Key-value pairs
- Key (i.e., the index) must be immutable
- Initialize with { } ( not [ ] or ( ) )

```
var = { }  
var['Apple'] = 43  
var[8] = [ 'Orange', 2, 14.0]
```

```
print (var[ 'Apple' ]) 43
```

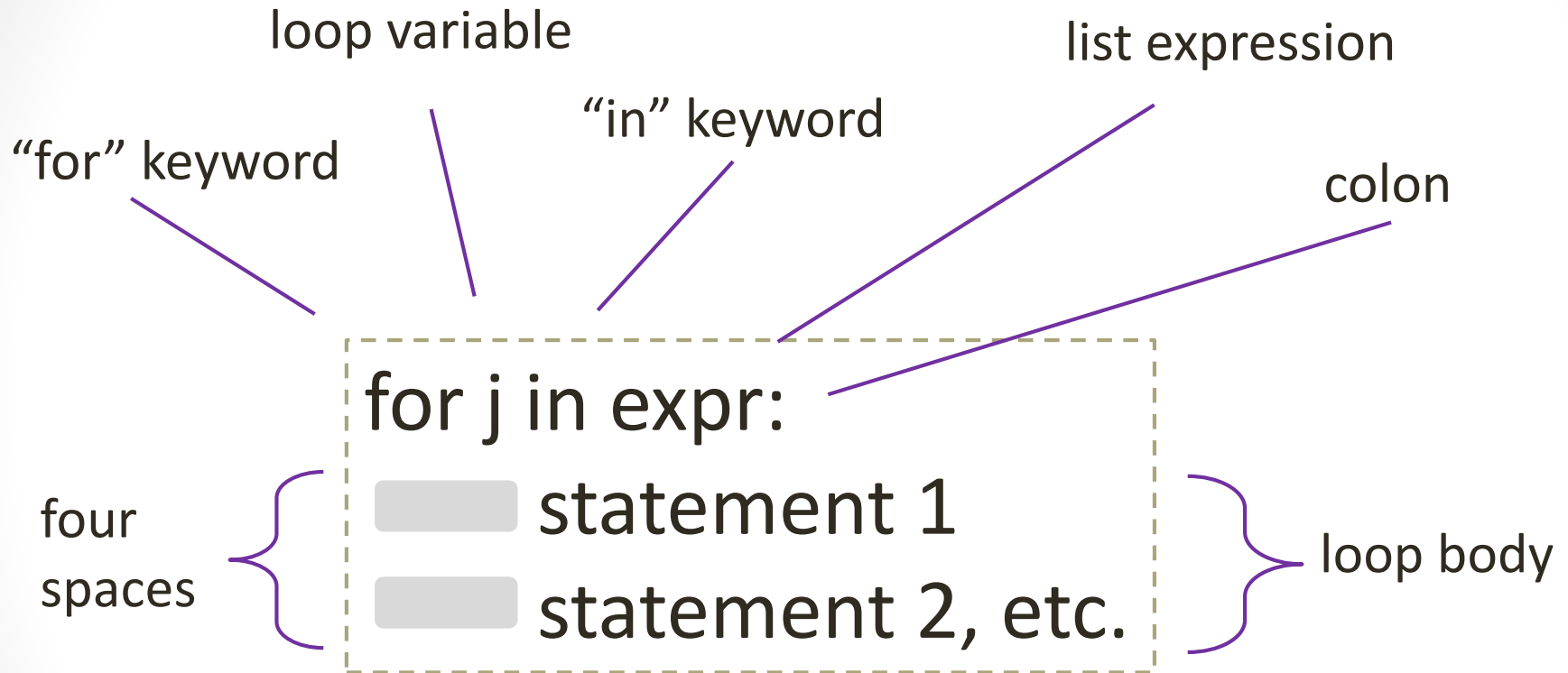
```
print( var[ 8 ] ) [ 'Orange', 2, 14.0]
```

```
print( var[ 8 ][ 2 ] ) 14.0
```

# Loops in Python

- Three commonly used loop constructs:
  - for
  - while
  - enumerate

# For Loop Syntax



For each element in expr:

- Assign its value to j
- Execute statements in loop body



# For Loop Examples

- Try these:

```
a = [1,2,3]
for j in a:
    print(j)
```

```
a = (1,2,3)
for j in a:
    print(j)
```

```
a = [ [1,2] , [3,4] ]
for j in a:
    print(j)
```

```
a = ['Peter', 'Paul', 'Mary']
for j in range(3):
    print(j)
    print( a[j] )
```

*range* function

- range(n)
  - Integer sequence 0 through n-1
- range(m,n)
  - Integer sequence m through n-1

# Nesting Loops

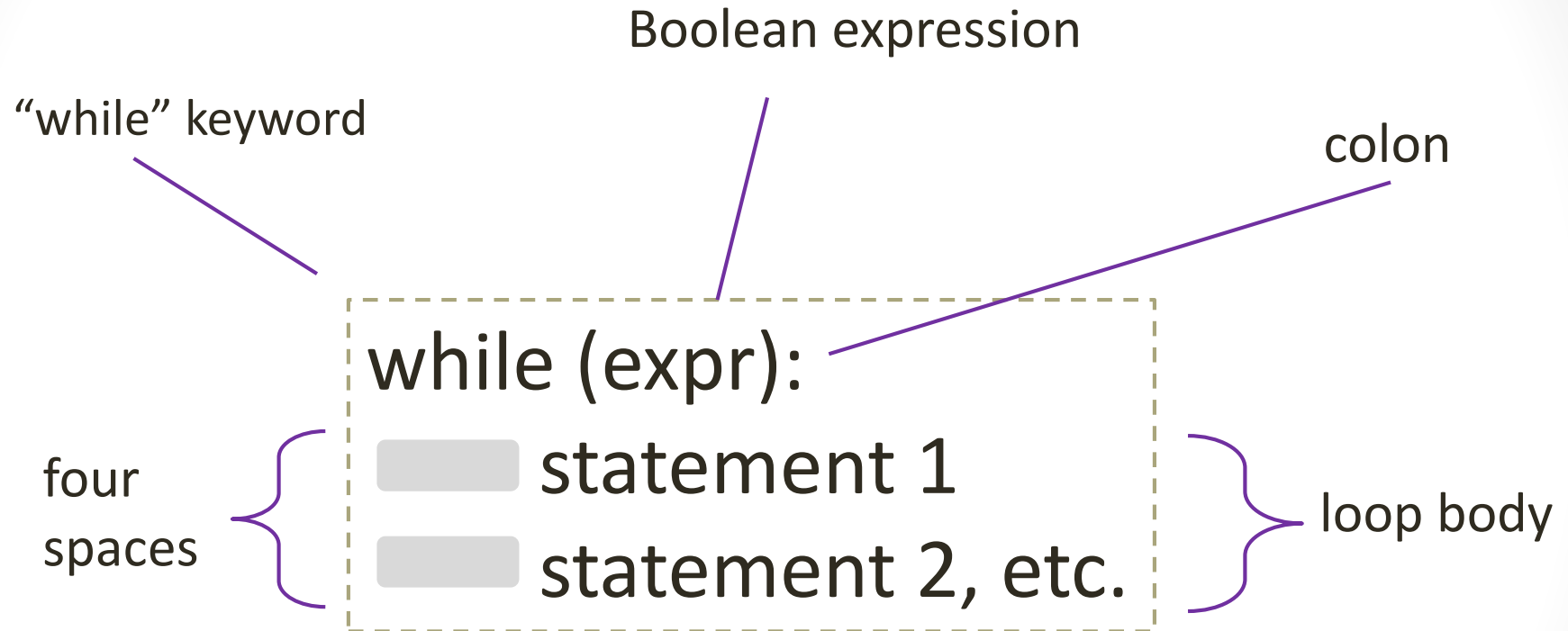
- Just like in any other language...
- Try this

```
a = [ [ 1 , 2 , 0 ] , [ 3 , 4 , 7 ] ]  
alen = len(a)  
for j in range(alen):  
    jlen = range(a[ j ])  
    for k in range(jlen):  
        print( j , k , ' : ' , a[ j ][ k ] )
```

# Exercise 1: For Loops

- Write a function that:
  - Accepts a single parameter, assumed to be a list of numbers
  - Returns the sum of those numbers
- Be sure to use a for loop
- Hint: don't forget about the `len` function

# While Loop Syntax



As long as `expr` is True:

- Execute statements in loop body

# While Loop Examples

- Try these:

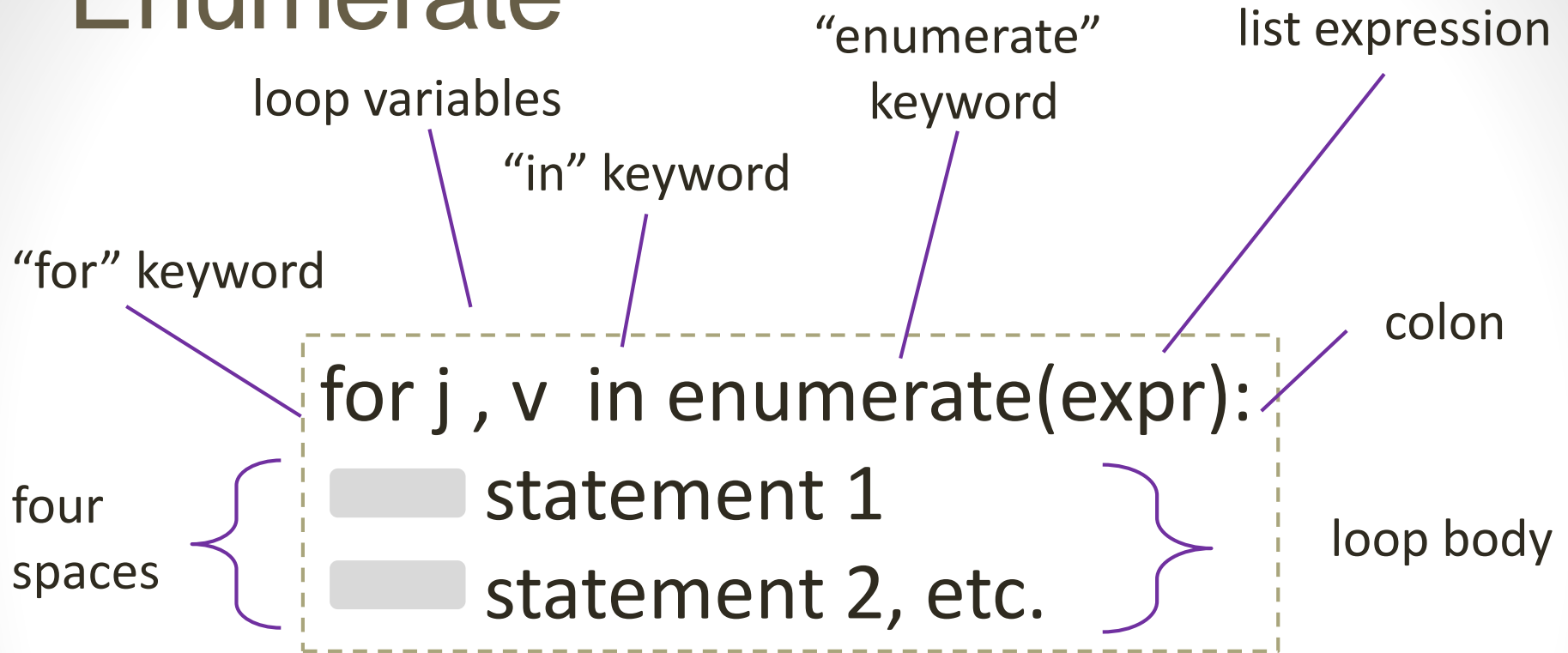
```
a = [ 1 , 2 , 3 ]  
j = 0  
n = len( a )  
while( j < n ):  
    print( j )  
    j += 1
```

```
a = [ [ 1 , 2 ] , [ 3 , 4 ] ]  
n = len( a )  
j = 0  
while ( j < n ):  
    print( a[ j ] )  
    for b in a[ j ]:  
        print( b )  
    j += 1
```

# Exercise 2: While Loops

- Write a function that:
  - Accepts two list parameters *a* and *b*
  - Returns the sum of  $a[j] * b[j]$  for all elements *j* in *a* and *b*.
  - Replaces  $b[j]$  with  $b[j] * a[j]$  as a *side effect*
- Be sure to use a while loop

# Enumerate



For each element in `expr`:

- Assign its value to `v`
- Assign a value of 0 through `len(expr)-1` to `j`
- Execute statements in loop body

# Enumeration Example

- Try this:

```
a = [ 'John' , 45.0 , 85 ]  
for j, v in enumerate(a):  
    print( j, v )
```



# Exercise 3: Enumerate

- Write a function that:
  - Accepts a single parameter, assumed to contain a list of string values
  - Returns a list of string values with their element index appended.
  - For example:
    - Input = [ 'Hello' , 'There']
    - Return value = ['Hello 0', 'There 1']
- Be sure to use enumerate