Algorithmic Approaches for Biological Data, Lecture #11

Katherine St. John

City University of New York
American Museum of Natural History

29 February 2016

Outline

- String Formatting

Outline

- String Formatting
- Dictionaries

Outline

- String Formatting
- Dictionaries
- Hashing

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries
- *Break*

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries
- *Break*
- More on Arrays:

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries
- *Break*
- More on Arrays:
  - Useful functions

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries
- *Break*
- More on Arrays:
  - Useful functions
  - Traversing Efficiently

Outline

- String Formatting
- Dictionaries
- Hashing
- Program Design:
  When to use what: lists, tuples, dictionaries
- *Break*
- More on Arrays:
  - Useful functions
  - Traversing Efficiently
  - Examples

String Formatting

- Python has a built-in string format function.

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic: *formatString*.format(*arguments*)

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic:
  formatString.format(*arguments*)
- Examples:
  `'{0} {1}'.format('one', 'two') #'one two'`

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic:
  formatString.format(*arguments*)
- Examples:

```
'{0} {1}'.format('one', 'two') #'one two'
'{1} {0}'.format('one', 'two') #'two one'
```

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic:
  formatString.format(*arguments*)
- Examples:

```
'{0} {1}'.format('one', 'two') #'one two'
'{1} {0}'.format('one', 'two') #'two one'
'{:>10}'.format('test') #'      test'
```

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic:
  formatString.format(*arguments*)
- Examples:

```
’{0} {1}’.format(’one’, ’two’) #’one two’
’{1} {0}’.format(’one’, ’two’) #’two one’
’{:>10}’.format(’test’) #’      test’
’{:,}’.format(1234567890) #’1,234,567,890’
```

String Formatting

- Python has a built-in string format function.
- Useful, but a bit cryptic:
  formatString.format(*arguments*)
- Examples:

```
’{0} {1}’.format(’one’, ’two’) #’one two’
’{1} {0}’.format(’one’, ’two’) #’two one’
’{:>10}’.format(’test’) #’      test’
’{:,}’.format(1234567890) #’1,234,567,890’
pts = 19.5
total = 22
’Correct answers:  {:.2%}’.format(pts/total)
    #’Correct answers:  88.64%’
```

Dictionaries

- Dictionaries are a collection data type that maps keys to values.

Dictionaries

- Dictionaries are a collection data type that maps keys to values.
- Canonical Example:

Dictionaries

- Dictionaries are a collection data type that maps keys to values.
- Canonical Example:

```
eng2sp = {}
eng2sp['one'] = 'uno'
eng2sp['two'] = 'dos'
eng2sp['three'] = 'tres'
```

Dictionaries

- Dictionaries are a collection data type that maps keys to values.
- Canonical Example:

```
eng2sp = {}
eng2sp['one'] = 'uno'
eng2sp['two'] = 'dos'
eng2sp['three'] = 'tres'
print eng2es
#{'three':'tres', 'two':'dos',
'one':'uno'}
```

Dictionaries

- Dictionaries are a collection data type that maps keys to values.
- Canonical Example:

```
eng2sp = {}
eng2sp['one'] = 'uno'
eng2sp['two'] = 'dos'
eng2sp['three'] = 'tres'
print eng2es
#{'three':'tres', 'two':'dos',
'one':'uno'}
print eng2es['two'] #'dos'
```

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.
  - `items()`: returns a list of the items (i.e. (key,value) pairs) in the dictionary.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.
  - `items()`: returns a list of the items (i.e. (key,value) pairs) in the dictionary.
  - `len()`: returns the number of items from the dictionary.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.
  - `items()`: returns a list of the items (i.e. (key,value) pairs) in the dictionary.
  - `len()`: returns the number of items from the dictionary.
  - `del`: delete items from the dictionary.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.
  - `items()`: returns a list of the items (i.e. (key,value) pairs) in the dictionary.
  - `len()`: returns the number of items from the dictionary.
  - `del`: delete items from the dictionary.
  - `get(key, altValue)`: returns altValue if key not found.

Dictionaries

- Uses curly braces: { and } for list, but brackets for access (like lists).
- Stores keys and values in any order.
- Useful commands:
  - `keys()`: returns a list of the keys in the dictionary.
  - `values()`: returns a list of the values in the dictionary.
  - `items()`: returns a list of the items (i.e. (key,value) pairs) in the dictionary.
  - `len()`: returns the number of items from the dictionary.
  - `del`: delete items from the dictionary.
  - `get(key, altValue)`: returns altValue if key not found.
- *PythonTutor demo*

Group Work

In pairs/triples, work out (and then try at the shell or `pythonTutor`):

1.
```
sub1 = "python string!"
sub2 = "an arg"
a = "i am a {0}".format(sub1)
b = "with {kwarg}!".format(kwarg=sub2)
```

2.
```
’The r{0} in Sp{0} stays m{0}ly in the pl{0}s.’.format(’ain’)
```

3.
```
d = [i/3.0 for i in range(0,20,2)] for j in range(len(d)):
    print ’{:6.2f}.format(j)
```

4.
```
s = "mississippi"
counts = {}
for c in s:
    counts[c] = counts.get(c,0)
tot = sum(counts.values())
```

5.
```
for i in range(5):
    for j in range(5):
        print j,
    print
```

6.
```
data = "GATGGAACTTGACTACGTAAATT"
cod = {}
for i in range(0,len(data),3):
    counts[data[i:i+3]] = counts.get(data[i:i+3],0)
print cod.keys()
```

7. Write a program that takes a file and will print out the 10 words that occur most often.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.
  Example: lines in a file.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.
  Example: lines in a file.
- Tuples: information that's meaning comes from being grouped together.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.
  Example: lines in a file.
- Tuples: information that's meaning comes from being grouped together.
  Example: coordinates of point in 3D space.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.
  Example: lines in a file.
- Tuples: information that's meaning comes from being grouped together.
  Example: coordinates of point in 3D space.
- Dictionaries: storing data for some, but not all, possible values.

When to Use What: Lists, Tuples, Dictionaries

- Lists: sequence of usually similar information, each processes separately.
  Example: lines in a file.
- Tuples: information that's meaning comes from being grouped together.
  Example: coordinates of point in 3D space.
- Dictionaries: storing data for some, but not all, possible values.
  Example: counting words that occur in a file (versus keeping count for all possible words).

Break

AMNH Anthropology Collections

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:

  ```
  import numpy as np
  ```

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:

  ```
  import numpy as np
  ```
- Common commands:

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:

  `import numpy as np`
- Common commands:
  - `np.zeros(x)`: creates an array of x zeros.

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:
  `import numpy as np`
- Common commands:
  - np.zeros(x): creates an array of x zeros.
  - np.ones(x): creates an array of x ones.

Recall: Arrays in numpy

- The numpy module is part of scipy (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:
  `import numpy as np`
- Common commands:
  - `np.zeros(x)`: creates an array of `x` zeros.
  - `np.ones(x)`: creates an array of `x` ones.
  - `np.arrange(start, stop, step)`: like the `range()` function but returns an ndarray.

Recall: Arrays in `numpy`

- The `numpy` module is part of `scipy` (and part of anaconda).
- Focuses on linear algebra (manipulation of vectors and matrices).
- To use:

  `import numpy as np`
- Common commands:
  - `np.zeros(x)`: creates an array of `x` zeros.
  - `np.ones(x)`: creates an array of `x` ones.
  - `np.arrange(start, stop, step)`: like the `range()` function but returns an ndarray.
  - `np.linspace(start, stop, n)`: creates an array of n numbers evenly spaced between start and stop.

Recall: Multidimensional Arrays

- The `numpy` module supports multidimensional arrays.

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:
  `A = np.array([ [3.4, 8.7, 9.9], [1.1, -7.8, -0.7], [4.1, 12.3, 4.8]])`

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:

```
A = np.array([ [3.4, 8.7, 9.9], [1.1,
-7.8, -0.7], [4.1, 12.3, 4.8]])
print A
```

Recall: Multidimensional Arrays

- The `numpy` module supports multidimensional arrays.
- Example:

```
A = np.array([ [3.4, 8.7, 9.9], [1.1,
-7.8, -0.7], [4.1, 12.3, 4.8]])
print A
print A.ndim
```

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:

```
A = np.array([ [3.4, 8.7, 9.9], [1.1,
-7.8, -0.7], [4.1, 12.3, 4.8]])
print A
print A.ndim
print A.shape
```

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:
  ```
  A = np.array([ [3.4, 8.7, 9.9], [1.1,
  -7.8, -0.7], [4.1, 12.3, 4.8]])
  print A
  print A.ndim
  print A.shape
  ```
- ndim gives the number of dimensions of the array.

Recall: Multidimensional Arrays

- The numpy module supports multidimensional arrays.
- Example:

```
A = np.array([ [3.4, 8.7, 9.9], [1.1,
-7.8, -0.7], [4.1, 12.3, 4.8]])
print A
print A.ndim
print A.shape
```

- ndim gives the number of dimensions of the array.
- shape gives the size of each dimension.

More on Arrays: Useful Functions

- The `numpy` module's array:

More on Arrays: Useful Functions

- The numpy module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

More on Arrays: Useful Functions

- The numpy module's array:
  ```
  import numpy
  x = np.array([[1,2,3],[4,5,6]]
  ```
- Adding matrices:

More on Arrays: Useful Functions

- The `numpy` module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

More on Arrays: Useful Functions

- The `numpy` module's array:

  ```
  import numpy
  x = np.array([[1,2,3],[4,5,6]]
  ```

- Adding matrices:

  ```
  print x+x
  ```

- Multiplying matrices: not * but dot()

More on Arrays: Useful Functions

- The `numpy` module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

- Multiplying matrices: not * but dot()

```
print x.dot(x)
```

More on Arrays: Useful Functions

- The `numpy` module's array:

  ```
  import numpy
  x = np.array([[1,2,3],[4,5,6]]
  ```

- Adding matrices:

  ```
  print x+x
  ```

- Multiplying matrices: not * but dot()

  ```
  print x.dot(x)
  ```

- Transposes: interchange element (i,j) with (j,i):

More on Arrays: Useful Functions

- The `numpy` module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

- Multiplying matrices: not * but dot()

```
print x.dot(x)
```

- Transposes: interchange element (i,j) with (j,i):

```
x.shape #(2,3)
```

More on Arrays: Useful Functions

- The numpy module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

- Multiplying matrices: not * but dot()

```
print x.dot(x)
```

- Transposes: interchange element (i,j) with (j,i):

```
x.shape #(2,3)
y = x.T #Transpose x
```

More on Arrays: Useful Functions

- The numpy module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

- Multiplying matrices: not * but dot()

```
print x.dot(x)
```

- Transposes: interchange element (i,j) with (j,i):

```
x.shape #(2,3)
y = x.T #Transpose x
y.shape #(3,2)
```

More on Arrays: Useful Functions

- The `numpy` module's array:

```
import numpy
x = np.array([[1,2,3],[4,5,6]]
```

- Adding matrices:

```
print x+x
```

- Multiplying matrices: not * but dot()

```
print x.dot(x)
```

- Transposes: interchange element (i,j) with (j,i):

```
x.shape #(2,3)
y = x.T #Transpose x
y.shape #(3,2)
```

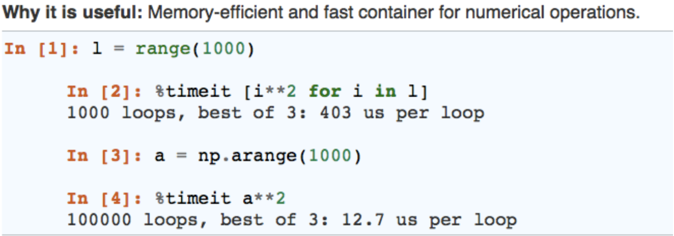
- Also can compute inverses, eigenvalues, eigenvectors,...

More on Arrays: Operations

- Use container operations when possible:

More on Arrays: Operations

- Use container operations when possible:



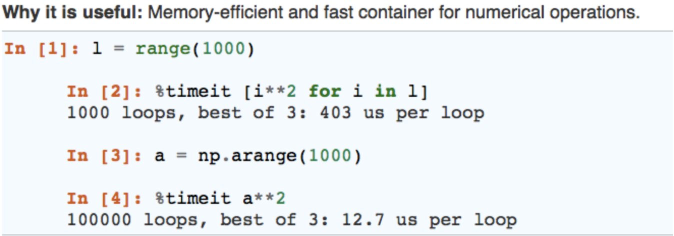
Why it is useful: Memory-efficient and fast container for numerical operations.

```
In [1]: l = range(1000)

      In [2]: %timeit [i**2 for i in l]
      1000 loops, best of 3: 403 us per loop

      In [3]: a = np.arange(1000)

      In [4]: %timeit a**2
      100000 loops, best of 3: 12.7 us per loop
```

scipy documentation

More on Arrays: Operations

- Use container operations when possible:



Why it is useful: Memory-efficient and fast container for numerical operations.

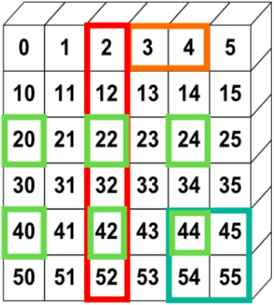
```
In [1]: l = range(1000)

      In [2]: %timeit [i**2 for i in l]
      1000 loops, best of 3: 403 us per loop

      In [3]: a = np.arange(1000)

      In [4]: %timeit a**2
      100000 loops, best of 3: 12.7 us per loop
```

scipy documentation

- These include most mathematical functions. To use them, remember to use the `np` prefix.
  Example: `np.log(x), np.sqrt(x), x**2, ...`

More on Arrays: Traversing Efficiently

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



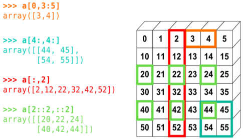
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.

More on Arrays: Traversing Efficiently

- When possible, use slices and indices to create a view of the array.



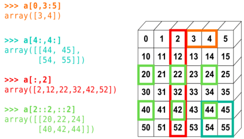
```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

More on Arrays: Traversing Efficiently

- When possible, use slices and indices to create a view of the array.
- Example:

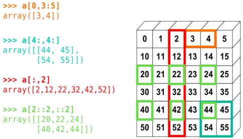


```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

More on Arrays: Traversing Efficiently



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

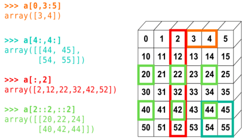
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.
- Example:

```
a = np.arange(10)
    #[0,1,2,3,4,5,6,7,8,9]
```

More on Arrays: Traversing Efficiently



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

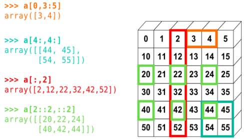
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.
- Example:

```
a = np.arange(10)
    #[0,1,2,3,4,5,6,7,8,9]
b = a[::2]
    # [0,2,4,6,8]
```

More on Arrays: Traversing Efficiently



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

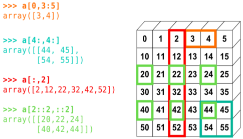
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.
- Example:

  ```
  a = np.arange(10)
      #[0,1,2,3,4,5,6,7,8,9]
  b = a[::2]
      # [0,2,4,6,8]
  b[0] = 12
      # Changes in both arrays
  ```

More on Arrays: Traversing Efficiently



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

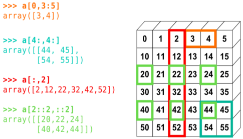
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.
- Example:

```
a = np.arange(10)
    #[0,1,2,3,4,5,6,7,8,9]
b = a[::2]
    # [0,2,4,6,8]
b[0] = 12
    # Changes in both arrays
b = a[::2].copy()
    # force a copy
```

More on Arrays: Traversing Efficiently



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

scipy documentation

- When possible, use slices and indices to create a view of the array.
- Example:

```
a = np.arange(10)
    #[0,1,2,3,4,5,6,7,8,9]
b = a[::2]
    # [0,2,4,6,8]
b[0] = 12
    # Changes in both arrays
b = a[::2].copy()
    # force a copy
a[a % 3 == 0] = -1
    # Boolean mask:
    #[-1,1,2,-1,4,5,-1,7,8,-1]
```

In Pairs

In pairs:

Assume: `import numpy`.

1. What does the following print:

```
a = np.arange(10)
a[::2] += 5
print a
```

2. What do the following print:

```
b = np.arange(12).reshape(3, 4)
print b print b*2
```

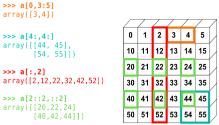
3. Write code that produces the array:

0	1	2	3	4	5
6	7	8	9	10	11

4. Write code that produces the array:

0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0

Recap

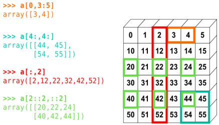


```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

- Using `matplotlib` & `numpy` in lab today.

Recap

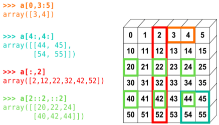


```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

- Using `matplotlib` & `numpy` in lab today.
- Email lab reports to `kstjohn@amnh.org`

Recap



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

- Using `matplotlib & numpy` in lab today.
- Email lab reports to `kstjohn@amnh.org`
- Challenges available at `rosalind.info`