Algorithmic Approaches for Biological Data, Lecture #10

Katherine St. John

City University of New York
American Museum of Natural History

24 February 2016

Outline

- List & Strings
- Lists, Tuples, & Arrays
- `numpy` and arrays

Outline

- List & Strings
- Lists, Tuples, & Arrays
- `numpy` and arrays

Lists & Strings

- Lists & strings are both container data types.

Lists & Strings

- Lists & strings are both container data types.
- Built-in functions to go from one to another:

Lists & Strings

- Lists & strings are both container data types.
- Built-in functions to go from one to another:
  - `split()`: splits up a string into a list (of strings)

Lists & Strings

- Lists & strings are both container data types.
- Built-in functions to go from one to another:
  - `split()`: splits up a string into a list (of strings)
  - `join()`: joins up a list into string

Lists & Strings

- Lists & strings are both container data types.
- Built-in functions to go from one to another:
  - `split()`: splits up a string into a list (of strings)
  - `join()`: joins up a list into string
- Example:

```
s = "I+love+Python"
words = s.split("+")
newS = " ".join(words)
```

In Pairs

In pairs:

Assume that `data = f.read()`

- Replace all tabs in `data` with 4 spaces (many ways to do this).
- Count the number of lines in `data`.
- Replace all the newlines with a space.
- Count the number of words in data.
- Capitalize the first letter of every word in data.

Lists, Tuples, & Arrays

- In addition to lists, there are two other natural ways to represent sequences of information:

Lists, Tuples, & Arrays

- In addition to lists, there are two other natural ways to represent sequences of information:
  - Tuples: fixed length sequences

Lists, Tuples, & Arrays

- In addition to lists, there are two other natural ways to represent sequences of information:
  - Tuples: fixed length sequences
    Example: `coordinates = (5,4,3)`
  - Arrays: fixed length sequences for mathematical manipulation

Lists, Tuples, & Arrays

- In addition to lists, there are two other natural ways to represent sequences of information:
  - Tuples: fixed length sequences
    Example: `coordinates = (5,4,3)`
  - Arrays: fixed length sequences for mathematical manipulation
    Example:
    `vector = np.array([3.4,5.6,9.8])`

Lists, Tuples, & Arrays

- In addition to lists, there are two other natural ways to represent sequences of information:
  - Tuples: fixed length sequences
    Example: `coordinates = (5,4,3)`
  - Arrays: fixed length sequences for mathematical manipulation
    Example:
    `vector = np.array([3.4,5.6,9.8])`
    (Part of the `numpy` module)

Arrays in `numpy`

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

Arrays in `numpy`

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

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:

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`

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:

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.

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.

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.

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.

Multidimensional Arrays

- The `numpy` module supports multidimensional arrays.

Multidimensional Arrays

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

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]])`

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
```

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
```

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
```

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.

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.

Indexing

- Works similarly to lists and strings.

Indexing

- Works similarly to lists and strings.
- Example:

Indexing

- Works similarly to lists and strings.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0]
```

Indexing

- Works similarly to lists and strings.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0]
print F[-1]
```

Indexing

- Works similarly to lists and strings.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0]
print F[-1]
B = np.array([ [ [111, 112], [121, 122] ],[
[211, 212], [221, 222] ],
```

Indexing

- Works similarly to lists and strings.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0]
print F[-1]
B = np.array([ [ [111, 112], [121, 122] ],[
[211, 212], [221, 222] ],
print B[0][1][0]
```

Indexing

- Works similarly to lists and strings.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0]
print F[-1]
B = np.array([ [ [111, 112], [121, 122] ],[
[211, 212], [221, 222] ],
print B[0][1][0]
```

- Can also use a single set of brackets: `B[0,1,0]`

Slicing

- The general syntax for a one-dimensional array A:

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0:3]
```

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0:3]
print F[-1::-1]
```

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0:3]
print F[-1::-1]
B = np.array([ [ [111, 112], [121, 122]
],[ [211, 212], [221, 222] ],
```

Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0:3]
print F[-1::-1]
B = np.array([ [ [111, 112], [121, 122]
],[ [211, 212], [221, 222] ],
print B[:1][1:][:]
```

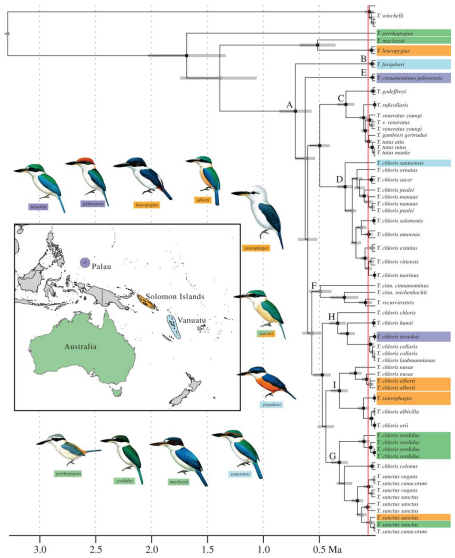
Slicing

- The general syntax for a one-dimensional array A: `A[start:stop:step]`
- Multidimensional is similar, but separate each coordinate by commas.
- Example:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print F[0:3]
print F[-1::-1]
B = np.array([ [ [111, 112], [121, 122]
],[ [211, 212], [221, 222] ],
print B[:1][1:][:]
```

- Can also use a single set of brackets: `B[:1,1:,:]`

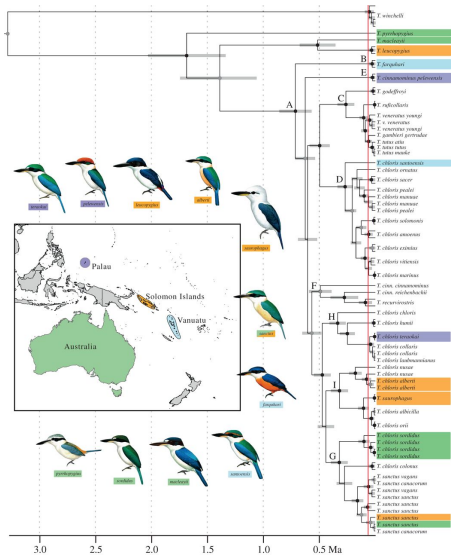
Recap



Anderson *et al.* 2014

- Using `matplotlib & numpy` in lab today.

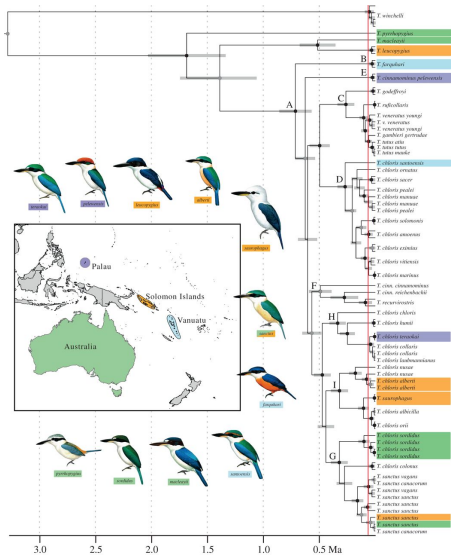
Recap



Anderson *et al.* 2014

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

Recap



Anderson *et al.* 2014

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