Algorithmic Approaches for Biological Data, Lecture #6

Katherine St. John

City University of New York
American Museum of Natural History

8 February 2016

Outline

Computing With Strings

- String variables

Outline

Computing With Strings

- String variables
- Simple string processing

Outline

Computing With Strings

- String variables
- Simple string processing
- Built-in string methods

Outline

Computing With Strings

- String variables
- Simple string processing
- Built-in string methods
- Break

Outline

Computing With Strings

- String variables
- Simple string processing
- Built-in string methods
- Break
- Strings as Lists, Lists as Strings

Outline

Computing With Strings

- String variables
- Simple string processing
- Built-in string methods
- Break
- Strings as Lists, Lists as Strings
- Using Files

String Variables

```
s = "I love Python!"
message = "Hello"
first = "Teddy"
last = "Roosevelt"
```

String Variables

```
s = "I love Python!"
message = "Hello"
first = "Teddy"
last = "Roosevelt"
```

- Concatenating ('adding') strings:
  `print "message"+"message"`

String Variables

```
s = "I love Python!"
message = "Hello"
first = "Teddy"
last = "Roosevelt"
```

- Concatenating ('adding') strings:
  ```
  print "message"+"message"
  print message+"message"
  ```

String Variables

```
s = "I love Python!"
message = "Hello"
first = "Teddy"
last = "Roosevelt"
```

- Concatenating ('adding') strings:
  ```
  print "message"+"message"
  print message+"message"
  print message+message
  ```

String Variables

```
s = "I love Python!"
message = "Hello"
first = "Teddy"
last = "Roosevelt"
```

- Concatenating ('adding') strings:

  ```
  print "message"+"message"
  print message+"message"
  print message+message
  ```

- Repetition operator:

  ```
  print 3*s
  ```

Accessing Strings

```
s = "I love Python!"
```

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!

- Can use whole string, or look at individual elements.

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!

- Can use whole string, or look at individual elements.
- `s[start:stop]` gives the substring that begins at `start` and goes up to but not including the stop.

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!

- Can use whole string, or look at individual elements.
- `s[start:stop]` gives the substring that begins at `start` and goes up to but not including the stop.
- Can also have a `step`: `s[start:stop:step]`.

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!

- Can use whole string, or look at individual elements.
- `s[start:stop]` gives the substring that begins at `start` and goes up to but not including the stop.
- Can also have a `step`: `s[start:stop:step]`.
- `s[:x]` is the substring of `s` starting at 0 and going up to, but not including the element with index `x`.

Accessing Strings

```
s = "I love Python!"
```

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Can access from end of list, by using negative indices.

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
`I`		`l`	`o`	`v`	`e`		`P`	`y`	`t`	`h`	`o`	`n`	`!`
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Can access from end of list, by using negative indices.
- Examples:

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Can access from end of list, by using negative indices.
- Examples:
  - `s[-2]` is `"n"`

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Can access from end of list, by using negative indices.
- Examples:
  - `s[-2]` is `"n"`
  - `s[0:-1]` is `"I love Python"`

Accessing Strings

```
s = "I love Python!"
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
I		l	o	v	e		P	y	t	h	o	n	!
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Can access from end of list, by using negative indices.
- Examples:
  - `s[-2]` is `"n"`
  - `s[0:-1]` is `"I love Python"`
  - `s[-3:]` is `"on!"`

Group Work

```
s = "I love Python!"
message = "Hello"
first = "Teddy"

last = "Roosevelt"
```

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

1.
   ```
   lll = s[2:6]
   print lll*3
   print lll+first*5
   ```

2.
   ```
   for c in message:
       print c
   ```

3.
   ```
   for i in range(5):
       print message[i]
   ```

4.
   ```
   for i in range(4,-1,-1):
       print message[i]
   ```

5.
   ```
   print message[-1::-1]
   ```

6.
   ```
   for c in message[-1::-1]:
       print c
   ```

7.
   ```
   for i in range(60):
           print "-",
       print
   ```

9.
   ```
   repeat = ""
   for i in range(20):
       repeat = repeat + "TA"
   ```

10.
    ```
    prefix = "outputRun"
    suffix = ".nex"
    fileNames = []
    for i in range(10):
        fileNames.append(prefix + str(i) + suffix)
    ```

11.
    ```
    prefix = "http://rest.ensemblgenomes.org/homology/id/"
    suffix = "?compara=pan_homology&content-type=application/json"
    genes = ["AT3G52260","AT3G52240","AT3G52610", "AT3G52150"]
    fileNames = []
    for g in genes:
        fileNames.append(prefix + g + suffix)
    ```

Useful String Methods

Python has built-in functions for strings:

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance.
    Also: `s.find(pattern, start, stop)`.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance.
    Also: `s.find(pattern, start, stop)`.
  - `s.count("AUG")` returns # of start codons.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
    - `s.upper()` returns an upper case version.
    - `s.find("AUG")` returns index of first occurrance.
      Also: `s.find(pattern, start, stop)`.
    - `s.count("AUG")` returns # of start codons.
      Also: `s.count(pattern, start, stop)`.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance. Also: `s.find(pattern, start, stop)`.
  - `s.count("AUG")` returns # of start codons. Also: `s.count(pattern, start, stop)`.
  - `s.isdigit()` returns true if all are digits, false otherwise.

Useful String Methods

Python has built-in functions for strings:

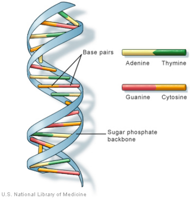
- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance.
    Also: `s.find(pattern, start, stop)`.
  - `s.count("AUG")` returns # of start codons.
    Also: `s.count(pattern, start, stop)`.
  - `s.isdigit()` returns true if all are digits, false otherwise.
  - `s.replace(old, new)` returns string with all occurrences of `old` with `new`.

Useful String Methods

Python has built-in functions for strings:

- `len(s)` returns the length of the string.
- Almost all others operate on the string:
  - `s.upper()` returns an upper case version.
  - `s.find("AUG")` returns index of first occurrance.
    Also: `s.find(pattern, start, stop)`.
  - `s.count("AUG")` returns # of start codons.
    Also: `s.count(pattern, start, stop)`.
  - `s.isdigit()` returns true if all are digits, false otherwise.
  - `s.replace(old, new)` returns string with all occurrences of `old` with `new`.
  - `s.strip()` returns string with leading and trailing whitespace removed.

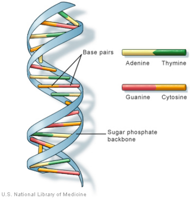
Group Work



In pairs or triples:

- Using the string methods, write Python code that counts the number of 'A','C','G', and 'T' in a string.

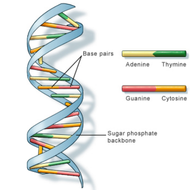
Group Work



In pairs or triples:

- Using the string methods, write Python code that counts the number of 'A','C','G', and 'T' in a string.
- Write pseucode (high-level description) of how you would make a list of all the sequence names in a FASTA formatted file.

Group Work



U.S. National Library of Medicine

In pairs or triples:

- Using the string methods, write Python code that counts the number of 'A','C','G', and 'T' in a string.
- Write pseucode (high-level description) of how you would make a list of all the sequence names in a FASTA formatted file.

Example:

```
>Rosalind_6404
CCTGCGGAAGATCGGCACTAGAATAGCCAGAACCGTTTCTCTGAGGCTTCCGGCCTTCCC
TCCCACTAATAATTCTGAGG
>Rosalind_5959
CCATCGGTAGCGCATCCTTAGTCCAATTAAGTCCCTATCCAGGCGCTCCGCCGAAGGTCT
ATATCCATTTGTCAGCAGACACGC
>Rosalind_0808
CCACCCTCGTGGTATGGCTAGGCATTCAGGAACCGGAGAACGCTTCAGACCAGCCCGGAC
TGGGAACCTGCGGGCAGTAGGTGGAAT
```

Break

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.
- For both, you can access individual elements (indexing) or substrings (slicing).

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.
- For both, you can access individual elements (indexing) or substrings (slicing).
- One major difference:
  - You can change elements of lists, but strings are immutable.

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.
- For both, you can access individual elements (indexing) or substrings (slicing).
- One major difference:
    - You can change elements of lists, but strings are immutable.
    - Can look at, but not change, individual elements in a string.

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.
- For both, you can access individual elements (indexing) or substrings (slicing).
- One major difference:
  - You can change elements of lists, but strings are immutable.
  - Can look at, but not change, individual elements in a string.
  - Can look at, and change, individual elements in a list.

Strings as Lists, Lists as Strings

In Python, lists and strings are similar

- Both are sequences of elements.
- For both, you can access individual elements (indexing) or substrings (slicing).
- One major difference:
  - You can change elements of lists, but strings are immutable.
  - Can look at, but not change, individual elements in a string.
  - Can look at, and change, individual elements in a list.
  - Demo at shell.

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

-
  ```
  s = "I love Python"
  words = s.split()
  ```

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

-
  ```
  s = "I love Python"
  words = s.split()
  ```
- Can also specify what the delimiter for the splitting should be:
  s.split('o')

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

-
  ```
  s = "I love Python"
  words = s.split()
  ```
- Can also specify what the delimiter for the splitting should be:
  s.split('o')

-
  ```
  specimen = "DOT 84 FLUID 11383, Ceyx lepidus collectoris, Solomon
  Islands, New Georgia Group, Vella Lavella Island, Oula River camp,
  , , 07 47 30 S, 156 37 30 E, Paul R. Sweet, 7-May-04,,PRS-2672,
  Tissue Fluid"

  fields = specimen.split(",")
  ```

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

-
  ```
  s = "I love Python"
  words = s.split()
  ```
- Can also specify what the delimiter for the splitting should be:
  s.split('o')

-
  ```
  specimen = "DOT 84 FLUID 11383, Ceyx lepidus collectoris, Solomon
  Islands, New Georgia Group, Vella Lavella Island, Oula River camp,
  , , 07 47 30 S, 156 37 30 E, Paul R. Sweet, 7-May-04,,PRS-2672,
  Tissue Fluid"

  fields = specimen.split(",")
  ```
- When using Excel files, can also export as tab-separated, and can use as the delimiters:

From Strings to Lists and Back

The `split()` method allows you to break up a string and store it in a list:

-
  ```
  s = "I love Python"
  words = s.split()
  ```
- Can also specify what the delimiter for the splitting should be:
  s.split('o')

-
  ```
  specimen = "DOT 84 FLUID 11383, Ceyx lepidus collectoris, Solomon
  Islands, New Georgia Group, Vella Lavella Island, Oula River camp,
  , , 07 47 30 S, 156 37 30 E, Paul R. Sweet, 7-May-04,,PRS-2672,
  Tissue Fluid"
  ```

  ```
  fields = specimen.split(",")
  ```
- When using Excel files, can also export as tab-separated, and can use as the delimiters:

  ```
  fields = specimen.split('\t')
  ```

From Strings to Lists and Back

The `join()` method takes a list and returns a string:

- Odd syntax:
  `delimiter.join(myList)`

From Strings to Lists and Back

The `join()` method takes a list and returns a string:

- Odd syntax:
  `delimiter.join(myList)`
- `state = "Mississippi"`

From Strings to Lists and Back

The `join()` method takes a list and returns a string:

- Odd syntax:
  `delimiter.join(myList)`

-
  ```
  state = "Mississippi"
  iDel = state.split("i")
  ```

From Strings to Lists and Back

The `join()` method takes a list and returns a string:

- Odd syntax:
  `delimiter.join(myList)`

-
  ```
  state = "Mississippi"
  iDel = state.split("i")
  newS = I".join(iDel)
  ```
- Converting comma-separated to tab-separated:

From Strings to Lists and Back

The `join()` method takes a list and returns a string:

- Odd syntax:

  ```
  delimiter.join(myList)
  ```

-
  ```
  state = "Mississippi"
  iDel = state.split("i")
  newS = I".join(iDel)
  ```

- Converting comma-separated to tab-separated:

  ```
  fields = specimen.split(",")
  specTab = ’\t’.join(fields)
  ```

Files

infile.txt
`Hello!`
`This is`
`a`
`test.`
`123`

Files

infile.txt
`Hello!`
`This is`
`a`
`test.`
`123`

```
"Hello!\nThis is \na \ntest.\n123"
```

Files

infile.txt
Hello!
This is
a
test.
123

```
"Hello!\nThis is \na \ntest.\n123"
```

- Text files are multi-lined strings.

Files

infile.txt
`Hello!`
`This is`
`a`
`test.`
`123`

```
"Hello!\nThis is \na \ntest.\n123"
```

- Text files are multi-lined strings.
- Lines are indicated by '\n' characters.

Files Commands

- Opening a file:

Files Commands

- Opening a file:

```
infile = open('data.txt', 'r')
```

Files Commands

- Opening a file:
  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```

Files Commands

- Opening a file:

```
infile = open('data.txt', 'r')
outfile = open('log.txt', 'w')
```

- Reading from a file:

Files Commands

- Opening a file:
  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```
- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.

Files Commands

- Opening a file:

  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```

- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.

Files Commands

- Opening a file:

  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```

- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.
  - `infile.readlines()`: read the file into a list of strings.

Files Commands

- Opening a file:

```
infile = open('data.txt', 'r')
outfile = open('log.txt', 'w')
```

- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.
  - `infile.readlines()`: read the file into a list of strings.
- Closing a file:

Files Commands

- Opening a file:

  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```

- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.
  - `infile.readlines()`: read the file into a list of strings.
- Closing a file:

  ```
  infile.close()
  ```

- Writing to a file:

Files Commands

- Opening a file:

  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```

- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.
  - `infile.readlines()`: read the file into a list of strings.
- Closing a file:

  ```
  infile.close()
  ```

- Writing to a file:

  ```
  outfile.write(s)
  ```

Files Commands

- Opening a file:
  ```
  infile = open('data.txt', 'r')
  outfile = open('log.txt', 'w')
  ```
- Reading from a file:
  - `infile.read()`: reads the entire file into a single string.
  - `infile.readline()`: read the next line of the file.
  - `infile.readlines()`: read the file into a list of strings.
- Closing a file:
  ```
  infile.close()
  ```
- Writing to a file:
  ```
  outfile.write(s)
  ```
- Demo at shell.

Group Work

In pairs/triples:

Group Work

In pairs/triples:

- Write a program will double space a file.

Group Work

In pairs/triples:

- Write a program will double space a file.
- Write a program that asks the user for a input and output file. Your program should copy the contents of the input file to the output and number the lines.

Group Work

In pairs/triples:

- Write a program will double space a file.
- Write a program that asks the user for a input and output file. Your program should copy the contents of the input file to the output and number the lines.
- Write a program that takes as input a FASTA file and prints out the number of sequences in the file.

Recap

(Image by Ron Dunn)

- Lecture Wednesday at 1pm.

Recap

(Image by Ron Dunn)

- Lecture Wednesday at 1pm.
- Email lab reports to `kstjohn@amnh.org`

Recap

(Image by Ron Dunn)

- Lecture Wednesday at 1pm.
- Email lab reports to `kstjohn@amnh.org`
- Challenges available at `rosalind.info`