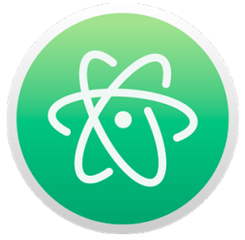
Lecture 7: Strings

Check-in and Reminders

- Reminder: **Homework 2 due now:** place in folders up front
- Lab 3 released on Friday
- Lab 2 you were given an algorithm and you had to implement it
- Lab 3 you have to come up the algorithm (to solve word puzzles!)
    - Advice: **sketch out your algorithm outline on paper first**
    - **Code later!** (Jumping to code often leads to errors)
- In terms of problem infrastructure:
    - Dealing with multiple python files for the same time
    - Topics to refresh: for loops, nested loops, file reading, strings

Do You Have Any Questions?



Review from Last Lecture

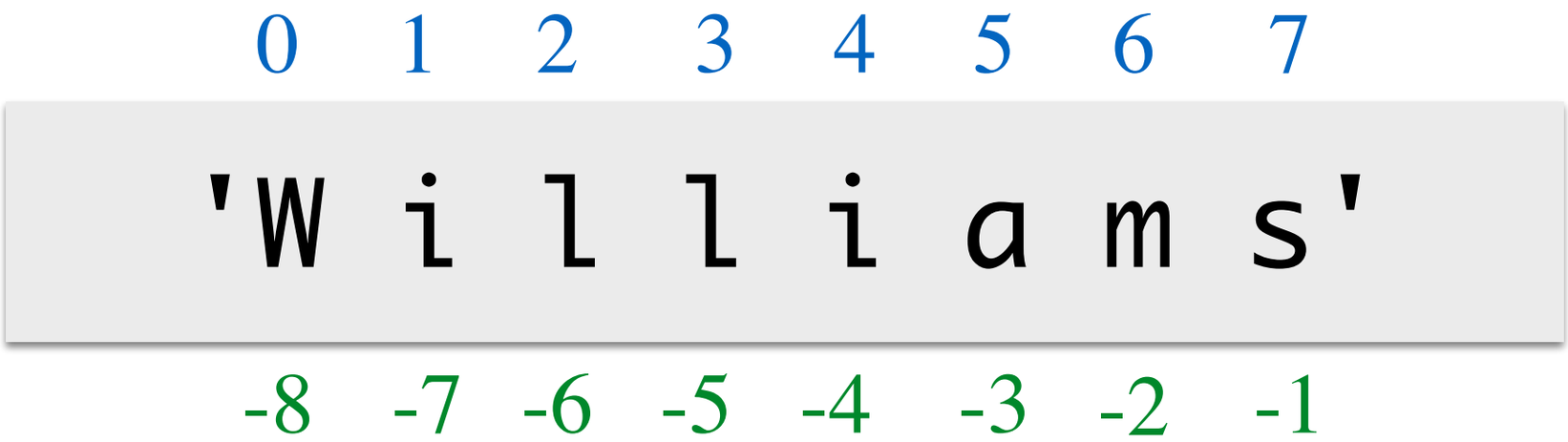
- What is the different purpose of modules and scripts?
- What is the purpose of special variable `__name__`?
- When do statements placed within the guarded `if __name__ == '__main__':` block get executed?
- When do statements placed within the guarded `if __name__ == '__main__':` block **do not get executed**?
- What sorts of things should we place within `if __name__ == '__main__'`?
- What is the purpose of the `__all__` special variable in a module?

Today's Class: Topics Outline

- How to slice and dice sequences to create new sequences
- Play with built-in string functions
    - strip, split, join
    - lower, upper
    - replace
- sorted sequences
    - sorted function
    - .sort on lists
- Format printing!

Review: How Do Indices Work?

- Indices in Python are both positive and negative.



word = 'Williams'

Review: upper(), lower()

- You can convert a string to uppercase or lowercase using Python's inbuilt `upper()` and `lower()` methods
- They return a **new string** with the corresponding case

```
In [1]: originalWord = 'Williams'

In [2]: newUpperWord = originalWord.upper()
        newLowerWord = originalWord.lower()

In [3]: newUpperWord
Out[3]: 'WILLIAMS'

In [4]: newLowerWord
Out[4]: 'williams'

In [5]: originalWord  # original word does not change
Out[5]: 'Williams'
```

Slicing Operation [:]

```
In [1]: word = 'Willaims'

In [2]: word[1:4]
Out[2]: 'ill'

In [3]: word[:3]
Out[3]: 'Wil'

In [4]: word[2:]
Out[4]: 'llaims'

In [5]: word[4:100] # notice no indexError
Out[5]: 'aims'
```

Slicing Operation with Optional Step

```
In [1]: word = 'Williams'

In [2]: word[:6:2]  # optional step argument
Out[2]: 'Wli'

In [3]: word[::2]
Out[3]: 'Wlim'

In [4]: word[::-1]  # reverse
Out[4]: 'smailliW'

In [5]: word[2::-2]
Out[5]: 'lW'
```

String to Lists

We can create a list from a string in several different ways.

- Using the `list` function on a string returns a list of all its characters
- `.split()` function on a string creates a list of words (which were separated by spaces in the string)

```
In [1]: word = 'Williams'

In [2]: list(word)
Out[2]: ['W', 'i', 'l', 'l', 'i', 'a', 'm', 's']


In [3]: phrase = "New England's weather is unpredictable."
        phrase.split()
Out[3]: ['New', "England's", 'weather', 'is', 'unpredictable.']
```

Lists of Strings to Strings

- If you have a list of strings, you can "join" them together in a string using Python's **`join`** method
- Join is a string method so it operates on a string, e.g. a string containing a space `' '` or a string containing `','`
- It returns a new string, e.g.

```
In [7]: ' '.join(['Birds', 'of', 'a', 'feather'])

Out[7]: 'Birds of a feather'

In [8]: ','.join(['Birds', 'of', 'a', 'feather'])

Out[8]: 'Birds,of,a,feather'
```

Mutability

Strings are Immutable

- Once you create a string, it cannot be changed!
- All functions that we have seen on strings return a new string and do not modify the original string

Lists are mutable

- Lists are mutable sequences
- As we saw, you can append to a list
- You can modify a list in many other ways: we will see this in the next lecture

Summary: Sequences Operations

Operation	Result
`x in seq`	True if an item of seq is equal to x
`x not in seq`	False if an item of seq is equal to x
`seq1 + seq2`	The concatenation of seq1 and seq2*
`seq*n, n*seq`	n copies of seq concatenated
`seq[i]`	i'th item of seq, where origin is 0
`seq[i:j]`	slice of seq from i to j
`seq[i:j:k]`	slice of seq from i to j with step k
`len(seq)`	length of seq
`min(seq)`	smallest item of seq
`max(seq)`	largest item of seq

* Concatenation is not supported on range objects

Image Source: (http://cs111.wellesley.edu/spring19)

Summary: String Methods

	Returned value
`word = 'Williams College'`	
`word.split()`	`['Williams','College']`
`word.upper()`	`'WILLIAMS COLLEGE'`
`word.lower()`	`'williams college'`
`word.replace('iams', 'eslley')`	`'Willeslley College'`
`word.replace('tent', 'eselley')`	`'Williams College'`
`newWord = '   Spacey College   '`	
`newWord.strip()`	`'Spacey College'`
`myList = ['Williams', 'College']`	
`' '.join(myList)`	`'Williams College'`

Remember. none of these operations change/affect the original string, they all return a new string

Lots More String Functions

- `word.find(s)`
    - Return the first (or last) position of string s in word. Returns -1 if not found.
- `s.isspace()`
  (or `islower, isupper, isalpha, isdigit, isalnum`).
    - Returns `True` if `s` is not empty and `s` is composed of white space (or lowercase, uppercase, or alphabetic letters, or digits, or either letters or digits).
- `word.count(s)`
    - Returns the number of (non-overlapping) occurences of `s` in word
- Many more: see `pydoc3 str`

\

Sorted Function

- The built-in function `sorted` which takes a sequence as input, creates and returns a **new list** where items of are ordered in **ascending** order.

```
In [1]: numbers = [35, -2, 17, -9, 0, 12, 19]
        sorted(numbers)
```

```
Out[1]: [-9, -2, 0, 12, 17, 19, 35]
```

- Notice that the original list is **unchanged**

```
In [2]: numbers
```

```
Out[2]: [35, -2, 17, -9, 0, 12, 19]
```

Sorted Function on Strings

- Strings can be sorted the same way: the ordering used for the sorting is dictated by the **ASCII values of the characters.**

```
In [3]: phrase = 'Red Code 1'
        sorted(phrase)
```

```
Out[3]: [' ', ' ', '1', 'C', 'R', 'd', 'd', 'e', 'e', 'o']
```

- Notice that spaces and special characters are first, following by numbers, followed by capital letters, and finally lower case
- You can check the ASCII value of any character using the **`ord`** function

```
In [4]: ord(' ')
```

```
Out[4]: 32
```

← ASCII value of space

Why Sort Strings?

- Gives us a canonical form, useful to find other strings made up of the same characters!
- Remember that when comparing strings, we should always make sure they are in the same case (which is why we use `.lower()` often)
- **Motivating example. Anagrams.**
  - Finding anagrams of a given word among a list of words
  - What do anagrams have in common?

Dormitory = Dirty room
School master = The classroom
Listen = Silent
Funeral = Real fun

Format Printing in Python

- A quick way to build strings with particular form is to use the `.format` function on them

Syntax: `myString.format(*args)`

`*args` means it takes zero or more arguments

- For every pair of braces (`{}`), format consumes one argument.
- Argument is converted to a string (with `str`) and concatenated with the remaining parts of the format string
- Especially useful in printing: called **format printing**

```
In [8]: "Hello, you {} world{}".format("silly",'!')  # creates a new string
Out[8]: 'Hello, you silly world!'

In [9]: print("Hello, {}.".format("you silly world!"))
        Hello, you silly world!.
```

Resume Exercise: bookStats

- Last lecture we were reading in the book Pride and Prejudice
- We converted it to a list of words using
    - strip function and split function
    - list accumulation
- We also built some functions in our module sequenceTools

- Lets apply some of these to figure out interesting things about pride and prejudice
- We can throw in some string functions we learnt today as well!
    - For example, how many palindromes are in the book?
    - What would happen if we took out conjunctions from the book?

Acknowledgments

These slides have been adapted from:

- http://cs111.wellesley.edu/spring19 and
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/