Strings in Python

Jerry Cain
CS 106AX
October 23, 2019
slides leveraged from those constructed by Eric Roberts

Selecting Characters from a String

• A string is (still) an ordered collection of characters. The character positions in a Python string are, as in most computer languages, identified by an index beginning at 0.
• For example, if `s` is initialized as `s = "hello, world"
the characters in `s` are arranged like this:

```
  h e l l o , w o r l d
```
• You can select an individual character using the syntax `s[k]`, where `k` is the index of the desired character. The expression `s[7]` returns the one-character string "w" that appears at index 7.

Negative Indexing

• Unlike JavaScript, Python allows you to specify a character position in a string by using negative index numbers, which count backwards from the end of the string. The characters in the "hello, world" string on the previous slide can therefore be numbered using the following indices:

```
   -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
```
• You can select the "w" toward the end of this string using the expression `s[-5]`, which is shorthand for the positive indexing expression `s[len(s) - 5]`

Concatenation

• One of the more familiar operations available to Python strings is concatenation, which consists of combining two strings end to end with no intervening characters.
• Concatenation is built into Python in the form of the + operator. This is consistent with how JavaScript (and Java, and C++) supports concatenation.
• Noteworthy difference between Python and JavaScript: Python interprets the + operator as concatenation only if both operands are strings. If one of the operands is something other than a string, then string concatenation isn’t applied.
• Restated, Python doesn’t automatically convert numbers to strings as JavaScript does.

Repetition

• In much the same way that Python redefines the * operator to indicate string concatenation, it also redefines the * operator for strings to indicate repetition, so that the expression `s * n` indicates n copies of the string `s` concatenated together.
• The expression "la" * 3 therefore returns "lalala", which is three copies of the string "la" concatenated together.
• Note that this interpretation is consistent with the idea that multiplication is repeated addition:

```
  "la" * 3 → "la" + "la" + "la"
```
• You can use this feature, for example, to print a line of 72 hyphens like this:

```
print("-") * 72
```

Exercise: String Repetition

• Use string repetition to encode the following songs in as short a Python program as possible:
  Let it be
  I can feel your halo

—Paul McCartney, "Let It Be", 1970

—Beyoncé, "Halo", 2008

• In 1984, computer scientist Don Knuth published a wonderful paper on "The Complexity of Songs" in which he concludes that Casey and the Sunshine Band hold the all-time record for

```
u = "uh huh" * 2
s = ("That's the way" + u + "I like it" + u + "(uh)" + "") * ∞
```
  I can see your halo
  I can see your halo
  I can see your halo

I can feel your halo
I can feel your halo
I can see your halo
Exercise: removeDoubledLetters

• In the early part of the 20th century, there was considerable interest in both England and the United States in simplifying the rules used for spelling English words, which has always been a difficult proposition. One suggestion advanced as part of this movement was to eliminate all doubled letters, so that bookkeeper would be written as bokpepe and committee would become comite. Write a function

```python
removeDoubledLetters(s)
```

that returns a new string in which any duplicated characters in a have been replaced by a single copy.

Exercise: Slicing

• Suppose that you have initialized ALPHABET as

```python
ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

so that the index numbers (in both directions) run like this:

```
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

• What are the values of the following slice expressions?

(a) ALPHABET[1::2]  
(b) ALPHABET[3:10]  
(c) ALPHABET[::3]  
(d) ALPHABET[::]  
(e) ALPHABET[14::12]  
(f) ALPHABET[1::1]

Methods for Finding Patterns

* `str.find(pattern)`
  Returns the first index of `pattern` in `str`, or -1 if it does not appear.

* `str.find(pattern, k)`
  Same as the one-argument version but starts searching from index `k`.

* `str.rfind(pattern)`
  Returns the last index of `pattern` in `str`, or -1 if it does not appear.

* `str.rfind(pattern, k)`
  Same as the one-argument version but searches backward from index `k`.

* `str.startswith(prefix)`
  Returns True if this string starts with `prefix`.

* `str.endswith(suffix)`
  Returns True if this string ends with `suffix`.

Slicing

• Python allows you to extract a substring by specifying a range of index positions inside the square brackets. This operation is known as slicing.

• The simplest specification of a slice is `[start:limit]`, where `start` is the index position at which the slice begins and `limit` is the index position before which the slice ends.

• The `start` and `limit` components of a slice are optional, but the colon must be present. If `start` is missing, it defaults to 0; if `limit` is missing, it defaults to the length of the string.

• A slice specification may also contain a third component called a stride, as with `[start:limit:stride]`. Strides indicate how many positions are omitted between selected characters.

• The stride component can be negative, in which case the selection occurs backwards from the end of the string.

Methods for Classifying Characters

* `str.islower()`
  Returns a copy of `str` with all letters converted to lowercase.

* `str.upper()`
  Returns a copy of `str` with all letters converted to uppercase.

* `str.capitalize()`
  Capitalizes the first character in `str` and converts the rest to lowercase.

* `str.strip()`
  Removes whitespace characters from both ends of `str`.

* `str.replace(old, new)`
  Returns a copy of `str` with all instances of `old` replaced by `new`.

* `str.isalpha()`
  Returns True if `ch` is a letter.

* `str.isdigit()`
  Returns True if `ch` is a digit.

* `str.isalnum()`
  Returns True if `ch` is a letter or a digit.

* `str.islower()`
  Returns True if `ch` is a lowercase letter.

* `str.isupper()`
  Returns True if `ch` is an uppercase letter.

* `str.isspace()`
  Returns True if `ch` is a whitespace character (space, tab, or newline).

* `str.isidentifier()`
  Returns True if this string is a legal Python identifier.
Revisiting Pig Latin in Python

```python
def toPigLatin(word):
    if word[0].lower() in 'aeiou':
        return word + 'way'
    else:
        firstVowel = findFirstVowel(word)
        return word[firstVowel:] + word[:firstVowel] + 'ay'

toPigLatin('this is pig latin')
```

Exercise: Implement `translate`

Pig Latin is just one example of a language game designed to render spoken words to be incomprehensible to the untrained ear. B-Language is another such language game—known primarily in Germany—where words are transformed such that every vowel cluster is reduplicated with a leading “b”.

Implement of `translate`, which accepts a lowercase (English) word and returns its B-Language translation.

Examples:
- `translate('quick')` ➞ "quibuick"
- `translate('spaghetti')` ➞ "spabaghebettibi"
- `translate('adieu')` ➞ "abadieubieu"
- `translate('audiophile')` ➞ "aubaudiobiophibilebe"
- `translate('queueing')` ➞ "queueibueueing"

Exercise: Implement `extract`

Uniform Resource Locators, or URLs, are string expressions used to identify where documents can be found on the Internet. Many URLs include query strings, which are substrings appearing after a “?” that further refine what portion of a document should be retrieved. For example:

```
https://www.google.edu/search?q=hello&tbm=vid
```

lists public `vid`eos associated with “hello”, which unsurprisingly includes a whole lot of Adele.

Exercise: Implement `extract`

Query strings are really just string serializations of maps in the sense you learned about JavaScript aggregates, where keys are separated from values by equal signs, and key-value pairs are separated from one another by ampersands.

Implement `extract`, which lists all key value pairs, one per line, embedded within the query string of a URL. For example:

```
```

would print the following in the Terminal:

```
Key: "view", Value: "catalog"
Key: "academicYear", Value: ""
Key: "page", Value: "0"
Key: "q", Value: "CS"
Key: "filter-departmentcode-CS", Value: "on"
Key: "filter-coursestatus-Active", Value: "on"
Key: "filter-term-Winter", Value: "on"
```

The End