Strings in Python

Early Character Encodings

- The idea of using codes to represent letters dates from before the time of Herman Hollerith, as described in Chapter 6.
- Most of you are familiar with the work of Samuel F. B. Morse, inventor of the telegraph, who devised a code consisting of dots and dashes. This scheme made it easier to transmit messages and paved the way for later developments in communication.

Strings as an Abstract Idea

- Characters are most often used in programming when they are combined to form collections of consecutive characters called strings.
- Strings are stored internally as a sequence of characters in sequential memory addresses.
- The internal representation is really just an implementation detail. For most applications, it is best to think of a string as an abstract conceptual unit rather than as the characters it contains.
- Python emphasizes the abstract view by defining a built-in string class that defines high-level operations on string values.

Selecting Characters from a String

- Conceptually, a string is an ordered collection of characters. In Python, the character positions in a string are identified by an index that begins at 0.
- For example, if `s` is initialized as `s = "hello, world"` the characters in `s` are arranged like this:

```
 0 1 2 3 4 5 6 7 8 9
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[0]` returns the one-character string "$h$" that appears at index 0.

Negative Indexing

- As a convenience, 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:

```
-0 -1 -2 -3 -4 -5 -6 -7 -8 -9
h e l l o . w o r l d
```

- You can select the "$d$" at the end of this string using the expression `s[-1]` which is shorthand for the positive indexing expression `s[len(s) - 1]`
Concatenation

- One of the most useful operations available for strings is concatenation, which consists of combining two strings end to end with no intervening characters.
- As you know from earlier in the semester, concatenation is built into Python in the form of the + operator.
- It is also important to recall that Python interprets the * operator as concatenation only if its operands are strings. If its operands are numbers, the * operator signifies addition.

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 a * n indicates a copies of the string a 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 \rightarrow "la" + "la" + "la"
```

Exercise: String Repetition

- Use string repetition to encode the following songs in as short a Python program as possible:

```
Let it be
I can see your halo
I can feel your halo
Whisper words of wisdom


Let it be
I can see your halo
I can feel your halo
I can feel your halo
I like it

—Beyoncé, “Halo,” 2008
```

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 end of the string.
- A slice specification may also contain a third component, which is called the stride. This component indicates how many positions are skipped between each selected character.
- The stride component can be negative, in which case the selection occurs backwards from the end of the string.

Exercise: Slicing

- Suppose that you have initialized ALPHABET as

```
ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

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

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

• What are the values of the following slice expressions?

- (a) ALPHABET[7:9]
- (b) ALPHABET[3:-1]
- (c) ALPHABET[14:12]
- (d) ALPHABET[:1]
- (e) ALPHABET[1::] 
- (f) ALPHABET[1:1::]
- (g) ALPHABET[0:5:2]
- (h) ALPHABET[:::1]
- (i) ALPHABET[5::2]
- (j) ALPHABET[14:10::2]

Comparing Strings

- Python allows you to call the standard relational operators to compare the values of two strings in a natural way. For example, if s1 and s2 are strings, the expression

```
s1 == s2
```

is True if the strings s1 and s2 contain the same characters.

- String comparisons involving the operators <, <=, >, and >= are implemented in a fashion similar to traditional alphabetic ordering: if the first characters match, the comparison operator checks the second characters, and so on.

- Characters are compared numerically using their Unicode values. For example, “cat” > “CAT” because the character code for “c” (99) is greater than the code for “C” (67). This style of comparison is called lexicographic ordering.
Common String Idioms

• When you work with strings, there are two idiomatic patterns that are particularly important:
  1. Iterating through the characters in a string.
     ```python
     for ch in s:
       ... body of loop that uses the character ch ...
     ```
  2. Growing a new string character by character.
     ```python
     result = ''
     for whatever loop header line fits the application :
       result += the next piece of the result
     ```

Reversing a String

```python
reverseString('stressed')
```

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 bokeper and committee would become comite. Write a function

```
removeDoubledLetters(s)
```

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

Using Methods in the String Class

• All other built-in operations on strings are defined as methods in Python’s string class. Before trying to use those methods individually, it is important to understand how those methods work at a more general level.
• Because strings are objects, Python uses the receiver syntax to call string methods. Thus, if s is a string, you would apply the method name using s.name(arguments).
• None of the methods in Python’s string class change the value of the string used as the receiver. What happens instead is that these methods return a new string on which the desired changes have been performed.
• Classes that prohibit clients from changing an object’s state are said to be immutable. Immutable classes have many advantages and play an important role in programming.

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.

Examples of the find Method

• The method find takes a string and returns the index within the receiver at which the first instance of that string begins. If the string is not found, find returns -1. For example, if s contains the string “hello, world”: 

```
s.find("h") → 0
s.find("o") → 4
s.find("all") → 1
s.find("s") → -1
```

• The find method takes an optional second argument that indicates the starting position for the search. Thus:

```
s.find("o", 5) → 8
```
Methods for Transforming Strings

- `str.lower()`
  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`.

Methods for Classifying Characters

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

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

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

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

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

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

- `str.isidentifier()`
  Returns True if this string is a legal Python identifier.

Exercise: Implement `upper`

• Suppose that Python’s string class had not included the `upper` method. How could you implement it as a function?