Strings
Writing strings

• Strings can be written in single quotes, '...', or double quotes, "...">
  – These strings cannot contain an “actual” newline
  – Certain special characters can be written in these strings by escaping them with a backslash
    – \n=newline, \t=tab, \'=single quote, \"=double quote, \\= backslash, \uhhhh=unicode character hhhh
• Strings can be written in triple quotes, '''...''' or """..."""
  – Triply-quoted strings can contain unescaped newline and unescaped single and double quote marks
• A raw string is one in which all characters just mean themselves
  – To create a raw string, just prefix it with r or R
  – IDLE example: strings.py
• These rules are for strings that appear in code; input is always “raw”
Basic operations

• A string is a sequence of characters, and can be treated as such
  – Example: `s = "abc123"`
  – `s[2]` is 'c'

• Strings are **immutable** (cannot be changed)

• However, you can create a new string, and even assign it back to the original variable
  – Example: `s = "xyz"`
More operations

• Strings can be concatenated (joined) with +
  – Example: s = "apple"
  – s + 's' is 'apples'

• Strings can be “multiplied” with *
  – 'bum ' * 3 + 'BUM!' is 'bum bum bum BUM!'

• Both of the above points are also true for normal lists

• You can use the in and not in tests with strings
  – 'cap' in 'escape sequence' is True
Methods on strings

• These are methods, not functions, so use the method syntax, `string.method()`, not `function(string)`

• Some useful methods:
  – `s.isupper()` and `s.islower()` test whether all letters in `s` are uppercase or lowercase, respectively
  – `s.upper()` and `s.lower()` converts all letters in `s` to uppercase or lowercase, respectively
    • Because strings are immutable, these methods return a new string, i.e., `s` itself is not changed
  – `s.isalpha()` tests if all characters in `s` are alphabetic
  – `s.isdigit()` tests if all characters in `s` are digits
  – `s.isspace()` tests if all characters in `s` are whitespace
  – whitespace includes spaces, tabs, newlines, and a few other nonprinting characters
More methods

- `s.lstrip()`, `s.rstrip()`, `s.strip()` removes whitespace from the left end, the right end, or both ends of `s`  
  - These methods don’t change `s`, they return a new string
- `s.startswith(substring)`, `s.endswith(substring)` test whether `s` starts with, or ends with, substring
- `s.find(substring)` finds substring in `s` and returns its index, or -1 if not found
- `sep.join(sequence)` inserts the string `sep` between the elements of `sequence`, and returns a new string  
  - Example: `, .join(['one', 'two', 'three']) = 'one, two, three'"
Even more methods

- The methods `ljust`, `rjust`, and `center` will left-justify, right-justify, or center a string in a field of a given width

- Example: `s = 'abc'`
  - `s.ljust(6)` is `'abc '`
  - `s.rjust(6)` is `' abc`
  - `s.ljust(6, '*')` is `‘abc***’`

- As always, these methods return a **new string**

- These methods probably won’t affect functionality but programmers like things to be ordered
Many other methods exist

- Will explain as/if we need them
- One example is the **format** method
  - The format method looks for braces, `{}`, in a string, and substitutes values for those braces
  - Example: `
    one={}, half={}
  `.format(1, 1/2) is ‘one=1, half=0.5’
  - **format** has many options