CSC108H Lecture 11

Dan Zingaro

October 3, 2012

Strings and Indices

- Since a string is a sequence, we can use Python index notation to extract its characters
- Assume `s` is a string
- Then, `s[i]` for `i` ≥ 0 extracts character `i` from the left
- Be careful: the first character in a string has index 0, not 1!
- We can also use a negative index `i` to extract a character beginning from the right
- e.g. If `s = "abcde"`, then
  - `s[0]` is `a`
  - `s[1]` is `b`
  - `s[-1]` is `e`
  - `s[-3]` is `c`

Python Slice Syntax

- Python slice syntax allows us to extract a segment of characters from a string
- `s[i:j]` extracts characters beginning at `s[i]` and ending at but not including `s[j]`
- If we leave out the first index, Python defaults to using index 0 to begin the slice
- Similarly, if we leave out the second index, Python defaults to using index `len(s)` to end the slice
- `s[:]` therefore gives us a copy of the entire string
- We can use negative indices in the slice syntax as well

ConcepTest

What is the output of the following code?

```
game = 'Lost Vikings'
print(game[5:-1])
```

- A. `kings`
- B. `king`
- C. `Viking`
- D. `Vikings`
- E. `ikings`

ConcepTest

What is the output of the following code?

```
game = 'Lost Vikings'
print(game[2:-6])
```

- A. `st V`
- B. `ost V`
- C. `iking`
- D. `st Vi`
- E. `Viking`

ConcepTest

What is the output of the following code?

```
game = 'Lost Vikings'
print(game[-6:11])
```

- A. `ost Vikings`
- B. `ost Viking`
- C. `ikings`
- D. `iking`
- E. `Vikings`

Methods

- **Method**: a function specific to one type of object
- Methods are called using `object.method` syntax

```
s = "Hi CSC108!"
s.lower()
lower(s) # wrong!
```

String Methods

- Use `dir(str)` to get a list of string methods
- Use `help(str.x)` for help on method name `x`

- `S.find(substring)`: return the index of the first occurrence of `substring` in `S` starting from the left, or -1 if `substring` is not found
- `S.replace(old, new)`: return `s` but with all occurrences of `old` replaced by `new`
- `S.count(substring)`: return the number of times `substring` occurs in `S`
- `S.startswith(substring)`: return `True` exactly when `S` begins with `substring`
- `S.endswith(substring)`: return `True` exactly when `S` ends with `substring`

ConcepTest

What is the output of this code?

```
s = 'Mississauga'
t = len(s.replace('ss', 'a'))
print(t)
```

- A. 11
- B. ss
- C. 10
- D. Miaaiaaauga'
- E. None of the above

ConcepTest

```
>>> help(str.center)
Help on method_descriptor:
center(...)
S.center(width[, fillchar]) -> str
Return S centered in a string of length width. Padding is
done using the specified fill character (default is a space)
```

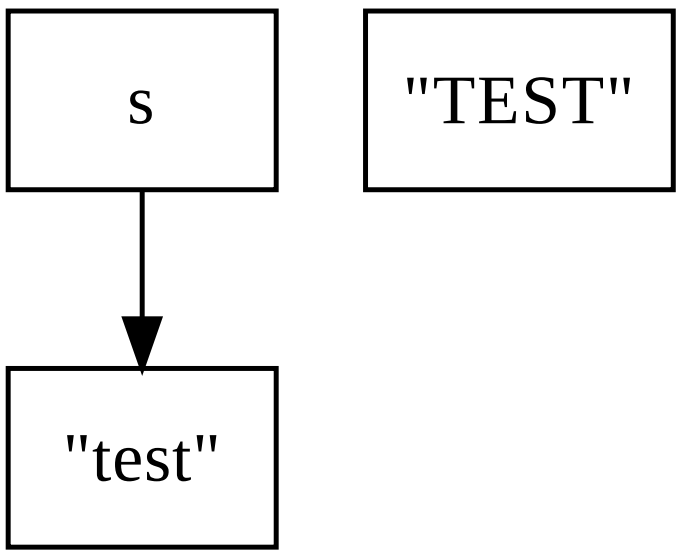
What is the string produced by the following:
`'cave'.center(8, 'x')`

- A. `'xxcavexx'`
- B. `'  cave  '`
- C. `'xxxxcavexxxx'`
- D. `'    cave    '`

Strings are Immutable (immutable_string.py)

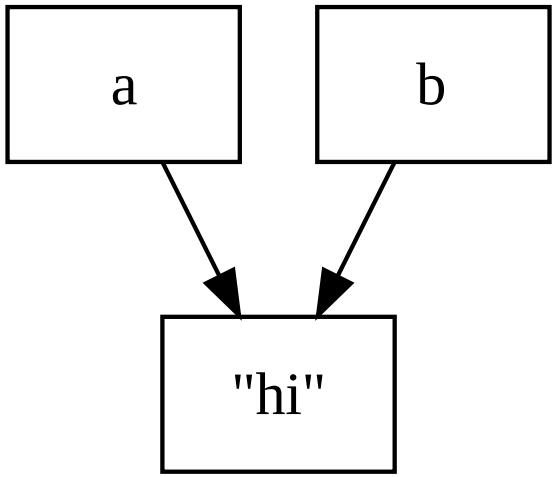
- String objects are **immutable**: means "cannot change"
- Methods that "look like" they are changing a string are actually creating a new string object
- Below, the string referenced by `s` is not modified; a new string is created, but immediately lost (since it is not referred to by a variable)

```
s = "test"
s.upper()
print(s)
```

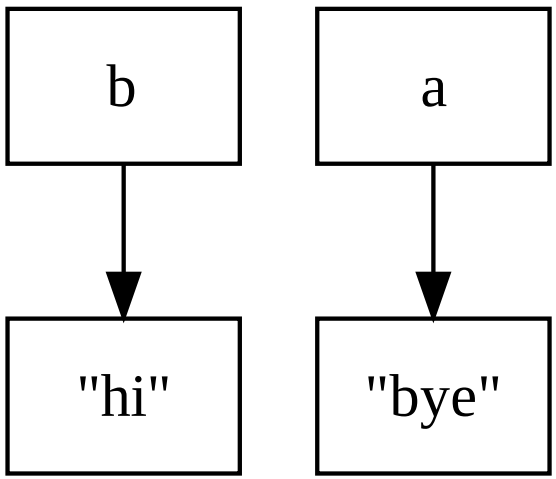


Shared References and Strings

```
a = 'hi'
b = a
```

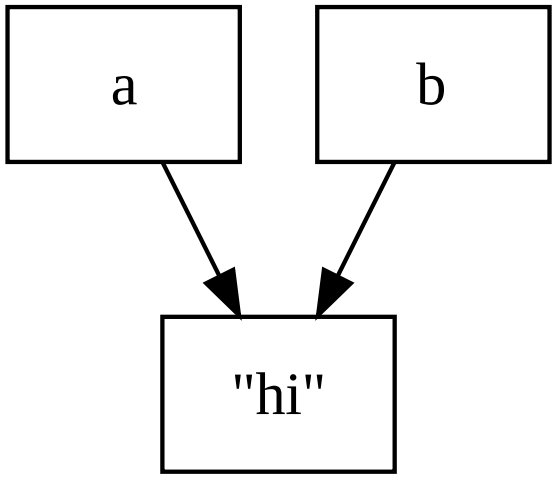



```
a = 'bye'
```



Shared References and Strings...

```
a = 'hi'
b = a
```




```
a = a.upper()
```