CSC108H Lecture 12

Dan Zingaro

October 5, 2012

String Comparisons

- We can use relational operators such as < and > on strings
- There is a well-defined ordering on strings: **a** comes before **b** which comes before **c** etc.
- There is also an ordering for the uppercase characters, and for digits
- Strings are compared from left to right using character codes
  - `ord` converts characters to codes
  - `chr` converts codes to characters
- If one string is a prefix of another, it is considered <

ConcepTest

```
ch = 'w'
val = ord('a') + 6 - (ord('z') - ord(ch) + 1)
mystery = chr(val)
```

What is the value of `mystery`?

- A. `'a'`
- B. `'b'`
- C. `'c'`
- D. `'d'`
- E. `'e'`

ConcepTest

```
ch = 'x'
val = ord('a') + 4 - (ord('z') - ord(ch) + 1)
mystery = chr(val)
```

What is the value of `mystery`?

- A. `'a'`
- B. `'b'`
- C. `'c'`
- D. `'d'`
- E. `'e'`

String Formatting

```
>>> 'I got {0}% on the test'.format(40 / 80 * 100)
'I got 50.0% on the test.'
>>> 'I got {m}% on the test'.format(m=40 / 80 * 100)
'I got 50.0% on the test'
```

- a **format string** contains fields like `{0}` or `{m}`
- Fields can be positional (numeric) or named
- Fields are replaced by the arguments to `format`

String Formatting...

- For argument `n`
  - Use `{n}` to insert the argument as-is
  - Use `{n:w}` for minimum width `w`
  - Use `{n:.p}` for precision `p`
  - Use `{n:w.p}` for minimum width `w` **and** precision `p`
- Precision is the number of digits to produce; accepted only for floats

ConcepTest

What is printed by this code?

```
s1 = '0'
s2 = 'a{' + s1 + '}b'
print(s2.format(4))
```

- A. `a{0}b`
- B. `ab`
- C. `a4b`
- D. The code does not run

Example 1

Complete the following function according to its docstring.

```
def rep_chars(s, num):
  '''(str, int) -> str
  Return a string consisting of each character of s
  repeated num times.

  >>> rep_chars('abc', 2)
  'aabbcc'
  '''
```

Example 2

Consider strings where each character is one of the following three characters:

- `g` for "good"
- `b` for "bad"
- `u` for "unusable"

The **goodness** of a string follows these two rules:

- The goodness of a string containing one or more **u**s is 0.
- Otherwise, the goodness of a string is equal to the number of **g**s in the string.

For example, the goodness of `"gbbgb"` is 2, and the goodness of `"gubgb"` is 0.

```
def goodness(s):
  '''(str) -> int

  Return goodness of s.
  '''
```