1 String slicing, ord and chr, and methods

A slice is a substring from the start index up to but not including the end index.

```python
c >>> s = 'computer science'
 >>> s[9:16] # negative slicing
 'science'
 >>> s[9:len(s)]
 'science'
 >>> s[9:] # default end
 'science'
 >>> s[:7] # default start
 'compute'
 >>> s[:] # makes a copy
 'computer science'
```

The `ord` and `chr` functions translate characters to numbers and back. The encoding is based on ASCII (http://en.wikipedia.org/wiki/ASCII), an encoding developed at the advent of computing. Example: `ord('A')` returns 65, which happens to be the ASCII encoding of ‘A’; `chr(65)` returns 'A'.

<table>
<thead>
<tr>
<th>String Method</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower</td>
<td>none</td>
<td>Returns a lowercase copy</td>
</tr>
<tr>
<td>upper</td>
<td>none</td>
<td>Returns an uppercase copy</td>
</tr>
<tr>
<td>capitalize</td>
<td>none</td>
<td>Returns a copy with first letter capitalized</td>
</tr>
<tr>
<td>isalpha</td>
<td>none</td>
<td>Returns True if all characters are alphabetic</td>
</tr>
<tr>
<td>isdigit</td>
<td>none</td>
<td>Returns True if all characters are digits</td>
</tr>
<tr>
<td>endswith</td>
<td>sub</td>
<td>Returns True if string ends with sub</td>
</tr>
<tr>
<td>startswith</td>
<td>sub</td>
<td>Returns True if string starts with sub</td>
</tr>
<tr>
<td>find</td>
<td>sub, start, end</td>
<td>Returns the lowest index where substring sub occurs or -1 if no occurrences. The start and end parameters are optional.</td>
</tr>
<tr>
<td>count</td>
<td>sub, start, end</td>
<td>Returns the number of non-overlapping occurrences of the substring sub in the string. The start and end parameters are optional.</td>
</tr>
<tr>
<td>replace</td>
<td>old, new</td>
<td>Returns a copy of string with every occurrence of substring old replaced with substring new.</td>
</tr>
<tr>
<td>strip</td>
<td>none</td>
<td>Returns a copy with leading and trailing whitespace removed</td>
</tr>
<tr>
<td>split</td>
<td>sep</td>
<td>Returns a list of “words” in string, splitting on whitespace. If optional separator sep is given, it splits on that instead.</td>
</tr>
<tr>
<td>join</td>
<td>list of strings</td>
<td>Returns a string that consists of the strings in list joined together with this string separating them.</td>
</tr>
</tbody>
</table>
A table of string methods followed by examples. For a complete list of methods, type `dir(str)` into the IDLE shell and then use `help` to get more information, as in `help(str.join)`. Remember: strings are immutable so none of these methods change the string: they return copies.

```python
>>> s = 'banana!'
>>> s.find('!')
6
>>> s.find('!', 3, 6)  # up to but not including end
-1
>>> 'banana!'.count('a')
3
>>> s.count('ana')  # non-overlapping
1
>>> s.replace('an', 'wah')
'bwahwaha!
>>> s = ' spacey banana '
>>> s.strip()
'spacey banana'
>>> s = 'hello jello fellow'
>>> s.split()
['hello', 'jello', 'fellow']
```

2 Exercises

Solutions are presented in class and also included in the moodle version of this handout.

1. Assume variable `s` is a string that refers to someone’s full name, as in `'Tom Brady'`. Write a single expression that produces the name with the first name abbreviated, as in `'T. Brady'`. Hint: Use the `find` and slicing.

   Solution:
   ```python
   s = 'Tom Brady'
   print s[0] + '. ' + s[s.find(' ')+1:]
   s = 'Peyton Manning'
   print s[0] + '. ' + s[s.find(' ')+1:]
   ```

2. Assume variables `s1` and `s2` refer to strings. Write a single expression that produces the index of the second occurrence of `s2` in `s1`. If `s2` does not occur twice in `s1`, the expression should
produce -1. For example, if s1 is "banana" and s2 is "ana", your expression should return 3. You cannot use indexing nor slicing; you can use find and arithmetic. Hint: call find twice.

Solution:

```python
s1 = 'banana'
s2 = 'ana'
print s1.find(s2, s1.find(s2) + 1)
s1 = 'apple'
s2 = 'p'
print s1.find(s2, s1.find(s2) + 1)
```

3. Write a function acronym that takes a string representing a phrase or name and returns the equivalent acronym, upper-cased. Example: if the input is 'computer science investigation', it should return 'C.S.I.'. Hint: use a for loop and the split and join and upper methods.

Solution:

```python
def acronym(s):
    '''(str) -> str
    Expects s to be a string of several words.
    Returns the corresponding acronym consisting of initial of each word in uppercase, separated by periods.
    >>> acronym('computer science investigation')
    'C.S.I.'
    >>> acronym('abbreviated coded rendition of name yielding meaning')
    'A.C.R.O.N.Y.M.'
    '''
    words = s.split()
    initials = []
    for word in words:
        initials += [word[0]]
    return '.'.join(initials).upper() + '.'
```

4. Write a function mycount that takes two strings s and sub and returns the number of nonoverlapping occurrences of sub in s. You cannot use the count method. Hint: use a while loop and the find method, making use of the optional start parameter of find.
Solution:

# version 1: uses a loop and a half

```python
def mycount(s, sub):
    '''(str, str) -> int
    Returns the number of non-overlapping occurrences of sub in s.'''
    count = 0
    idx = s.find(sub)  # half a loop
    while idx > -1:
        count += 1
        idx = s.find(sub, idx+len(sub))
    return count
```

# version 2: uses an infinite loop with break

```python
def mycount2(s, sub):
    '''(str, str) -> int
    Returns the number of non-overlapping occurrences of sub in s.'''
    count = 0
    idx = 0
    while True:
        count += 1
        idx = s.find(sub, idx+len(sub))
    return count
```
idx = s.find(sub, idx)
if idx == -1:
    break
count += 1
idx += len(sub)
return count

Some material adapted from Campbell and Gries.