1 Properties of Strings

sequences From the python documentation, we know that "Strings are sequences of Unicode code points." We'll explain what Unicode is in the another lecture, but the easiest way to think about strings is just as sequences of characters. Characters in Python are still strings—often strings of length one, but not always. Sequences in Python support several operations. Below, suppose that `s` is a sequence.

indexing `s[i]` returns the character at position i in the string

slicing Similar to ranges, `s[i:j:k]` returns the substring from position i to $j-1$ (inclusive) using step k. One can use empty i and j to grab prefixes or suffixes

length Use `len(s)` to find the length of a string

immutable Once a python string, always a python string. And always the *same* python string. You cannot change the contents of a python string—only generate new python strings based on old ones. For example:

```
>>> s = "brent drove 3.14 miles"
>>> s[0] = "t"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

2 String methods

Strings in Python have type `str`. You'll notice that

```
>>> type("a")
<class 'str'>
```

Classes define *types*. Classes also have methods, which are functions associated with objects of a given type. This means that any string object `s` has associated with it several functions that operate on the string. These functions are called using the *dot* notation: `s.bar(...)` means you have an object `s` and a method `bar` associated with it. Later in the course we'll talk about defining our own types by creating new classes with associated methods. Below, let `s` be a string object.

split This methods splits a string into constituent parts based on a separator string. A common usage is to split a string into its non-whitespace characters using the default `s.split()`. For example, `"brent drove 3.14 miles".split()` returns the list `['brent', 'drove', '3.14', 'miles']`.

join This method joins a list of strings using `s` as its separating character. For example, `"++".join(['brent', 'drove', '3.14', 'miles'])` returns `"brent++drove++3.14++miles"`.

upper Returns a new string with all the characters of `s` converted to uppercase

lower Returns a new string with all the characters of `s` converted to uppercase

find Given a search string `sub`, returns the lowest index in `s` where `sub` appears. If `sub` does not appear, this method returns -1.

The python API can be found here: `https://docs.python.org/3.4/library/stdtypes.html#textseq`

3 Formatting strings

```
>>> "{} drove {} miles".format("Brent", 3.14)
'Brent drove 3.14 miles'
>>> "{dave} drove {ten} miles".format(dave="Brent", ten=3.14)
'Brent drove 3.14 miles'
>>> "{1} drove {0} miles".format(3.14, "Brent")
'Brent drove 3.14 miles'
```

See `https://docs.python.org/3/library/string.html#formatstrings` for many more examples.

4 Examples

```
def palindrome(s):
    """Returns True if and only if s is a palindrome"""
    return (s == s[::-1])

def palindrome2(s):
    """Returns True if and only if s is a palindrome"""
    n = len(s)
    for i in range(n):
        if (s[i] != s[n-1-i]):
            return False
    return True

def palindrome3(s):
    """Returns True if and only if s is a palindrome"""
    n = len(s)
    for i in range(len(s)//2):
        if (s[i] != s[n-1-i]):
            return False
    return True


s = "a man a plan a canal panama"
s = "".join(s.split())
print(s)

print(palindrome(s))
print(palindrome2(s))
print(palindrome3(s))
```

```
$ python3 strings.py
amanaplanacanalpanama
True
True
True
```