# CS151 Fall 2012 Lecture 12

Stephanie R Taylor

October 1, 2012

## 1 Administrative Topics

- Return Quiz 3
- Sorry, but Proj 3 grades won't be out until tomorrow.
- Stephanie's office hours today will be either the usual (1:30-3:30) or split (12:30-2 and 4-5).

## 2 Strings

### 2.1 Creating String Literals

We have seen how to create string literals, by placing some text between pairs of single quotes or pairs of double quotes, e.g. "hi" and 'ho'. Once a string has been created in Python, Python "forgets" which quotes were used to delimit it. In other words, the quotes are just used to create it and don't determine how the string is stored. For example, if we type

```
myString = "hi"
```

at the Python interpreter prompt (i.e. we use double quotes), then type

```
myString
```

it will print out `'hi'` (single quotes).

However, if we actually call `print`, it won't print the quotes at all, e.g.

```
print myString
```

results in `hi`.

### 2.1.1 Escape Characters

Strings can have special characters like tabs and carriage returns encoded with *escape sequences*. Escape sequences begin with a backslash. Table 1 shows a few that are useful when reading text that has been read in from a file.

Table 1: Selected Escape Sequences

| Escape Sequence | Meaning |
|---|---|
| \' | single quote |
| \" | double quote |
| \t | tab |
| \n | newline (return in Unix) |
| \r | carriage return |

An escape sequence is considered as a single character – e.g. `len('h\th')` is 3.

Carriage returns, newlines, tabs and spaces are all considered *whitespace*.

## 2.2 String Operations

We have seen how to concatenate two strings, e.g. "hi" +"ho" is "hiho".

There are several other operators that apply to strings shown in Table 2.

Here are some examples illustrating the operations (with the command typed at the Python prompt and the result that is displayed):

Table 2: String Operations

| Operation | Result |
|---|---|
| `x in s` | `True` if an item of `s` is equal to `x`, else `False` |
| `x not in s` | `False` if an item of `s` is equal to `x`, else `True` |
| `s + t` | the concatenation of `s` and `t` |
| `s * n`, `n * s` | `n` shallow copies of `s` concatenated |
| `s[i]` | `i`th item of `s`, origin 0 |
| `s[i:j]` | slice of `s` from `i` to `j` (but not including the `j`th item) |
| `s[i:j:k]` | slice of `s` from `i` to `j` with step `k` (but not including the `j`th item) |
| `len(s)` | length of `s` |

- `"man" in 'human'` ⇒ `True`
- `"woman"not in 'human'` ⇒ `True`
- `"hi" = "ho"` ⇒ `'hiho'`
- `"ho " * 3` ⇒ `'ho ho ho '`
- `'blast'[0]` ⇒ `'b'`
- `'blast'[1:5]` ⇒ `'last'`
- `'blast'[0:5:2]` ⇒ `'bat'`
- `len('blast')` ⇒ `5`
- `min('blast')` ⇒ `'a'`
- `max('blast')` ⇒ `'t'`

And all of these examples produce identical results if they are applied to a variable containing a string, e.g. after the line

```
myString = 'blast'
```

we reproduce the same results:

- `myString[0]` ⇒ `'b'`
- `myString[1:5]` ⇒ `'last'`
- `myString[0:5:2]` ⇒ `'bat'`
- `len(myString)` ⇒ `5`

This is true because the operations are performed on the *value* itself – symbols just let us get to the values.

## 2.3 String Methods

Guess what? Strings are objects, so they have methods.[1].

One of the most useful string methods is `find`, which searches for the first occurrence of one string within another and returns its location. Its syntax, along with that of several other string methods is in Table 3.

Here are several examples using the string methods:

- `"ho ho ho".upper()` ⇒ `"HO HO HO"`
- `"ho ho ho".split()` ⇒ `['ho', 'ho', 'ho']`
- `"ho,ho,ho".split(',')` ⇒ `['ho', 'ho', 'ho']`
- `"ho,ho,ho".split(',',1)` ⇒ `['ho', 'ho,ho']`

For a complete list of string methods, see

`http://docs.python.org/library/stdtypes.html#string-methods`

[1]In fact, all data types in Python are actually objects. Yes, even numeric types are objects, but there are no commonly used methods on numeric types. For the purposes of CS151, there is no need to think of numbers as objects, so we ignore that fact. New in Python 2.6 is a method called `hex` which returns the hexadecimal representation of a float

Table 3: Selected String Methods

| Operation | Result |
|---|---|
| `s.upper()` | returns a copy of `s` in upper case |
| `s.lower()` | returns a copy of `s` in lower case |
| `s.title()` | returns a copy of `s` in title case (the first letter of each "word" is upper case) |
| `s.strip()` | Return a copy of the string with the leading and trailing whitespace characters removed |
| `s.strip(chars)` | Return a copy of the string with the leading and trailing characters removed where the characters are specified by the string `chars`* |
| `s.split()` | Returns a list of words in the string, using whitespace as the delimiter string: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace |
| `s.split(sep)` | Return a list of the words in the string, using `sep` as the delimiter string |
| `s.split(sep, maxsplit)` | Return a list of the words in the string, using `sep` as the delimiter string with at most `maxsplit`s done |
| `s.find(findstr)` | If `findstr` is in `s`, then returns the index of the first character of `findstr` within `str`, else returns -1 |
| `s.find(findstr,start)` | The same as above, but doesn't begin lookinh in `s` until the `start`'th character |
| `s.find(findstr,start,end)` | The same as above, but doesn't look past the `end`'th character |

### 2.4 Strings Methods Return New Strings

Notice that all of these methods return a new string with the result. None of them are changing the original string. This is because string are immutable in Python. Once a string has been made, it will always be the same. But that doesn't mean we can't just reuse a variable, e.g.

```
str = 'defenestration of prague'
str = str.title()
```

Now `str` contains the string `'Defenestration Of Prague'`.

## 3 Converting strings to other types

In certain situations, such as when we are using command line arguments, we have strings that represent numbers. To use the numeric value of the string, we need to convert it. We saw this in Project 3.

- Converting to an integer value. Use the function `int()` For example, if you have a variable called `istr` and its contains a string representing an integer value (e.g. "3"), then you would call `int(istr)`.
- Converting to a floating point value. Using the function `float()`. For example, if you have a variable called `fstr` and its contains a string representing an integer value (e.g. "3.1"), then you would call `ifloat(fstr)`.
- Converting to a boolean value. There is no equivalent function. If you have a variable called `bstr` and it contains a string representing a boolean value (e.g. "True") and you want to put its boolean value in a variable called `truthiness`, The best thing to do is this:

  ```
  truthiness = bstr == "True"
  ```

  and if you want to be case insensitive, then you could use this code:

  ```
  truthiness = bstr.title() == "True"
  ```

# 4 Quiz Yourself

1. How would you use the slice operation to retrieve the substring “can” from the string “catatonic” (recall that the square brackets are used to slice strings)?

2. How would you separate the string ”2.3e-9” into two strings – one containing the mantissa and the other containing the exponent?

3. Familiarize yourself with the String Methods at http://docs.python.org/library/stdtypes.html How would you remove the trailing spaces from a string?

4. How would you check to see if all characters in a string are numbers?

5. How would you count the number of times “ACCT” appears in “ACCTTGGCACCT”?