Computer Science 1 — CSci 1100
Lecture 4 — Python Strings

Reading

This material is drawn from Chapters 3 of *Practical Programming* and Chapter 8 of *Think Python*. We aren't ready for the latter, but we will look at an example of something a bit more interesting than the introductory examples of *Practical Programming*. The documentation strings are covered in Chapter 4 of *Practical Programming*.

More Than Just Numbers

- Computers started with numbers, and still at their core they work with numbers — binary numbers at that.
- But, now we work with text.
- Sometimes the main data that your program is working with is text. Think about a mad lib program:
  - It reads text (nouns, adjectives, etc.) from the user
  - It fills the sentences with the (hopefully funny) user input
- Similarly, the main thing Facebook stores and shows is text: information about you, your status updates, comments on friends' updates, etc.
- Sometimes the function of your program is to compute numbers, but the program needs to present these numbers with text that explains what they mean.
  - Instead of outputting:

    ```
    2
    95
    ```

  - a much clearer presentation is:

    ```
    You got 2 out of 3 total questions correctly.
    Your grade is 95.
    ```

- All of these require a representation, a representation called *strings*.
  - This representation is another *abstraction*: it is implemented by the Python interpreter in terms of numbers, but we ignore this detail and think more about what we can do with strings.

We've Already Started With Strings

- A string is a sequence of 0 or more characters delimited by single quotes or double quotes.

```
'Rensselaer'
"Albany, NY"
'4 8 15 16 23 42'
''
```

- We have already used strings in our `print` statements

```
>>> print "Hello, world!"
```

- Strings form a third type, in addition to integers and floats.

Writing strings out: use of quotations and print

- Strings may be assigned to variables:

```
>>> s = 'Hello'
```

- You can use both quotes together. Try the following:

```
>>> s = "'He did what?'"
>>> s
>>> print s

>>> s = '"She told us that programming was easy."'
>>> s
>>> print s
```

- You cannot mix quotations though.

```
>>> 'Hello "
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
EOL while scanning string literal: <string>, line 1, pos 8
```

- What if you wanted to write: schrodinger's cat? We can mix different quotations:

```
>>> "schrodinger's cat"
```

Writing strings out: multiple lines

- Suppose the string spans multiple lines.
- We can use three quotes to start and end a multi-line string.

```
>>> s = '''this is a
... multi line
... string
... '''
```

- Anything goes in between the quotes.
- Note: you can also use triple double quotations:

```
>>> s = """this is a multi line string"""
```

Exercise: single, double, triple quote

Ok, which of the following are valid strings?

```
>>> "Monorail cat"s pose"
```

```
>>> "Ceiling cat's paw"
```

```
>>> '''Rebecca Black's
... "Friday"
... '''
```

```
>>> """Rebecca Black's "Friday" """
```

```
>>> 'Have you seen the "Incredibly Photogenic Guy"'s picture?'
```

```
>>> "Have you seen the 'Incredibly Photogenic Guy''s picture?"
```

Writing strings out: escaping characters

- Let us look closer at the multi-line strings.

  ```
  >>> s = '''this is a
  ... multi line
  ... string
  ... '''
  >>> s
  >>> 'this is a \nmulti line\nstring\n'
  ```

- `\n` is a special symbol called **newline** that indicates to Python the end of a line in string.
- The backslash `\` is an escape character. It allows you to change how you treat the next character:
  - `\'`: do not treat quote as special
  - `\n`: treat the letter n as special
  - `\t`: treat t as special, in this case it will be a tab character for spacing
- What if you wanted to print the following string?

  `Python treats \n as the new line character`
- So, you can put a quote in a string in one of two ways:

  ```
  >>> "schrodinger's cat"
  >>> 'schrodinger\'s cat'
  ```

String Operations: Concatenation

- Often, we need to combine multiple strings into a single one: when we combine a noun input for mad-lib with the rest of the sentence.
- Combining strings is called **concatenation**, it is accomplished using the '+' operator

```
>>> s = 'Hello'
>>> s2 = s + 'World!'
```

- Careful here! Spaces are not automatically created.

```
>>> s2
>>> 'HelloWorld!'
```

- When concantenating strings (not the variables they are associated with), the '+' operator is not even necessary.

```
>>> s2 = 'Hello ' 'World'
```

- Note that the '+' operator has different meanings: it adds numbers, but concatenates strings. This is called **overloading**.

String Operations: Replication

- Strings may be replicated:

```
>>>  s3 = 'abc' * 4
```

  results in `s3` being associated with

```
>>>  'abcabcabcabc'
```

- What happens when the multiplier is a 0 or a negative number? Try it!

Printing strings

- When you print a string:
  - Python automatically puts a newline after it.
  - It strips the outer layer of quotes.
  - It replaces the escaped characters with their values: new lines, tabs.
- You can print multiple things by listing them. A single space in inserted between different items.

```
>>> a = 1
>>> b = 2
>>> print a,b
```

- If you end a print statement with a comma, it assumes your list is continuing. Try the following script:

```
a = 1
b = 2
print a,
print b
```

Formatted printing

- You can also create a string where some pieces are filled in from arguments supplied at the end.

```
>>> a = 4
>>> print "String %s has %d characters" %('nice',a)
```

- Formatting commands:
  - `%s` for strings
  - `%d` for integers
  - `%f` for floats
- Let us provide a better format for the output of the function `area_and_volume` from Lecture 3.

```
def area_and_volume(radius, height):
    print "Given a cylinder"
    print "\twith radius\t%.2f" %radius
    print "\tand height\t%.2f\n" %height
    print "The surface area is\t%.2f" \
      %area_cylinder(radius,height)
    print "The volume is\t\t%.2f" \
      %volume_cylinder(radius,height)

>>> area_and_volume(4, 10)
Given a cylinder
        with radius     4.00
        and height      10.00

The surface area is     351.86
The volume is           502.65
```

Exercise

What is the result of the following operations? Try this without typing it in Python first.

```
>>> line = '''egg\tspam\nspam\tegg'''
>>> line
```

```
>>> print line
```

```
>>> line = 'nobody expect the %s inquisition.' %spanish
>>> print line
```

```
>>> print 'egg \\\n spam'
```

```
>>> print 'memes\' evolve' 'faster', 'than Internet''s' + ' growth'
```

Comments

- Comments are explanatory statements you put in programs.
- They should describe the meaning of your code, not its syntax.
- Inline comments appear after a line of code, preceeded with a `#` sign. It is recommended that you capitalize the first letter.

  ```
  area = length * width  # Compute the area of rectangle
  ```

- Use inline comments rarely, only if it is necessary to explain what that statement is doing.
- Always write comments that explain what a program or function does. Place a command at the beginning of a program and at the beginning of each function.
- Use strings with double quotations (even if they fit in a single line) for commenting the first line programs, functions and modules. These are called **Documentation Strings**.

  ```
  def ellipse_area(a=0.0, b=0.0):
      """Return the area of an ellipse with the given radius

      a is the half width of the ellipse
      b is the half height of the ellipse

      """

      return 3.14 * a * b
  ```

- You can actually print the documentation strings of functions in Python:

  ```
  >>> ellipse_area.__doc__
  ```

Reading strings

- Interactive programs require the user to provide input.
- You type your query into Google to search something. Your query is input to the Google search algorithm.
- In Python, you can read user input using the built-in `raw_input()` function. Reading stops when newline is encountered.

```
print "Enter a number: "
num = raw_input()
```

- The input is always read as string, even if it is a number.
- How can you make the prompt appear in the same line as the user input?

Type Conversions

- Even if a string contains a number, it is not treated as a number unless you explicitly convert it. What is the result of this?

```
>>> '123' + 4
```

- Any integer or float can be converted to a string, using the built in `str` function.

```
>>> int('123')
>>> str(5.0000)
```

- A string that has the form of an integer may be converted to an integer using the `int` function, and a string that has the form of a float may be converted to a float using the `float` function.
- When string does not have the proper format, an execution error occurs.

String Length

- The function `len` returns the number of characters in a string. For example,

```
>>> len('George')
6
```

- Try these, however:

```
>>> len('John Doe')
```

and

```
>>> len('Hello, World!')
```

and

```
>>> len('Hello, World!  ')
```

- Remember, the number of `len` gives the number of **characters** in a Python string, not just the number of letters.
- What is the result of this?

```
>>> len('Hello\n')
```

A word on characters

- When we are programming, we often use characters from the English Alphabet (lower and upper case), numbers, punctuation and some special control characters like space and tab. This set of basic characters is called ASCII (American Standard Code for Information Interchange).
- There are 128 ASCII characters.
- Strings may contains letters from other alphabets though. Unicode character set is much larger than ASCII and contains almost all letters from different languages and mathematical symbols, etc.
- Unicode characters cannot be represented as simple strings. You need to use an encoding to be able to store them as special strings. The most common encoding is UTF-8, especially for the Web.
- In Python, unicode strings are preceded by the letter u. When you store them in a file, you need to define an encoding.

```
>>> s = u'Sibel Adalı'
>>> s
>>> print s
```

Example: mad libs

- Let us write a simple mad libs program which will take in as input some words and output a sentence constructed from the user input.
- You need to prompt the user first about what type of input she should give.

```
"""mad_lib

   Reads input from user, and prints a mad_lib for python.
"""

print "enter a noun representing a feature:",
noun = raw_input()
print "enter a verb with -ing:",
verb_with_ing = raw_input()

print   "Due to the %s of the Python language,\n" \
        " the CS department decided to teach in Python\n" \
        " instead of %s with C++." %(noun, verb_with_ing)
```

Summary

- Strings represent character sequences — our third Python type
- String operations include addition (concatenate) and replication
- We can concatenate by '+' or by using formatted strings:

```
>>> 'a' + 'b'
>>> '%d eggs and %s spam' %(2,'no')
```

- Functions on strings may be used to determine length and to convert back and forth to integers and floats.
- Escape sequences change the meaning of special Python characters or make certain characters have special meaning.
- Some special characters of note: `\n` for new line, `\t` for tab. They all precede with `\`
- We can read input using `raw_input()`