String: Text as a Value

- String are quoted characters
  - "’abc d’" (Python prefers)
  - "’abc d’" (most languages)

- How to write quotes in quotes?
  - Delineate with “other quote”
  - Example: " ’ "

- Solution: escape characters
  - Format: \ + letter

<table>
<thead>
<tr>
<th>Char</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>’</td>
<td>single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>

String are Indexed

- s = 'abc d'
- Access characters with []
  - s[0] is ‘a’
  - s[4] is ‘d’
  - s[5] causes an error
  - s[0:2] is ‘ab’ (excludes c)
  - s[2:] is ‘c d’

- Called “string slicing”

- s = 'Hello all'
- What is s[3:6]?

|s|0|1|2|3|4|5|6|7|
|---|---|---|---|---|---|---|---|
|H|e|l|l|o| |a|l|

Other Things We Can Do With Strings

- **Operation** in: s1 in s2
  - Tests if s1 “a part of” s2
  - Say s1 a substring of s2
  - Evaluates to a bool
- **Examples:**
  - s = 'abraodabra'
  - 'a' in s == True
  - 'abd' in s == True
  - 'too' in s == False

- **Function** len: len(s)
  - Value is # of chars in s
  - Evaluates to an int
- **Examples:**
  - s = 'abraodabra'
  - len(s) == 11
  - len(s[1:9]) == 4
  - s[1:len(s)-1] == 'bracadabr'

Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a function call
  - Will explain the meaning of this later
- Function expressions have the form `fun(x,y,…)`

Examples (math functions that work in Python):
- `round(2.54)`
- `max(a=0,4)`

Built-In Functions

- You have seen many functions already
  - Type casting functions: `int()`, `float()`, `bool()`
  - Dynamically type an expression: `type()`
  - Help function: `help()`
- Getting user input: `raw_input()`
- **print <string>** is not a function call
  - It is simply a statement (like assignment)
  - But it is in Python 3.x: `print(<string>)`
Method: A Special Type of Function

- Methods are unique (right now) to strings
- Like a function call with a “string in front”
  - Usage: `string method(x,y...)`
  - The string is an implicit argument
- Example: `upper()`
  - `s = 'Hello World'
  - `s.upper()` == 'HELLO WORLD'
  - `s[1:5].upper()` == 'ELLO'
  - `'abc'.upper()` == 'ABC'

Will see why we do it this way later in course

Examples of String Methods

- `s.index(s2)`
  - Position of the first instance of `s2` in `s`
- `s.count(s2)`
  - Number of times `s2` appears inside of `s`
- `s.strip()`
  - A copy of `s` with white-space removed at ends

Examples of String Methods

- `s1.index(s2)`
  - Position of the first instance of `s2` in `s1`
- `s1.count(s2)`
  - Number of times `s2` appears inside of `s1`
- `s.strip()`
  - A copy of `s` with white-space removed at ends

Built-in Functions vs Modules

- The number of built-in functions is small
  - [http://docs.python.org/2/library/functions.html](http://docs.python.org/2/library/functions.html)
- Missing a lot of functions you would expect
  - Example: `cos()`, `sqrt()`
- Module: file that contains Python code
  - A way for Python to provide optional functions
  - To access a module, the `import` command
  - Access the functions using module as a prefix

Example: Module math

- `import math`
- `math.cos(0)`
- `3.141592653589793`
- `math.cos(math.pi)`
- `-1.0`

Other Modules

- io
  - Read/write from files
- random
  - Generate random numbers
  - Can pick any distribution
- string
  - Useful string functions
- sys
  - Information about your OS

Example: Module math

- `import math`
- `math.pi`
- `3.141592653589793`
- `math.sin(math.pi)`
- `-1.0`

Using the from Keyword

- Be careful using from!
- Namespaces are safer
  - Modules might conflict (functions w/ same name)
  - What if import both?
- Example: Turtles
  - Use in Assignment 4
  - 2 modules: turtle, tkturtle
  - Both have func. Turtle()