python

Lecture III
Syntax

- Statements
- Output
- Variables
- Conditions
- Loops
- List Comprehension
- Function Calls
- Modules

Statements

- Statements normally end at the end of lines. No semicolons are needed.
- If a line ends before all open brackets are closed, the next line is considered to be part of the same statement:

  -
    ```
    x = [1, 2, 3,
         4, 5, 6]
    ```
- A statement can also be broken into several lines using a backslash at the end of the line:

  -
    ```
    X = 5 + \
        6
    ```

Output

- Python 2.x uses the `print` keyword for output. By default, `print` sends the output to the system's standard output, but this can be redirected to files.
- Any object can be printed. The `print` statement will always try to convert the object to be printed into some string representation.
- Printing `unicode` strings may raise errors in systems or IDEs that do not normally support unicode.

Variables

- Everything in Python is an object, and all objects that can be accessed are accessed through variables.
- Even built-in values and functions, such as `True`, `False`, `len`, `range`, etc. are variables and can be assigned to:
  -
    ```
    >>> print True
    True
    >>> True = 5
    >>> print True
    5
    ```

Variables

- Variables are created automatically when assigned to, but can be deleted manually using the `del` operator:

  -
    ```
    >>> x = 5
    >>> print x
    5
    >>> del x
    >>> print x
    ERROR
    ```

Variables

- Variables behave like references or pointers. Assignment is simply changing a pointer. It is NOT a copy:

  -
    ```
    x = [1, 2, 3]
    y = x
    y.append(8)
    ```
    y $\longrightarrow$ `[1, 2, 3, 8]`   x $\longrightarrow$ `[1, 2, 3, 8]`

Variable Naming

- Variables start with a letter or underscore, optionally followed by any number of letters, digits or underscores.
- Variables names are case sensitive: `var`, `Var`, `vAr`, and `VAR` are 4 different variables.

Variable Naming

- Variables starting with an underscore are considered "private" and are harder (but not impossible!) to access from outside their scope.
- Those starting with two underscores are "name mangled". They appear to the outside word under compiler-generated names.
- Those starting *and ending* with two underscores are "reserved". They have special meaning to Python and should be used only as prescribed.

Conditions

- The only simple conditional statement in Python is `if`. It is used as follows:
  -
```
if condition1:
        actions1
elif condition2:
        actions2
elif condition3:
        actions3
else:
        actions4
```

Conditions

- The *condition* part can be any expression – it will be automatically treated as a Boolean as per the rules discussed previously.
- The *actions* part is a group of statements to execute if the condition to which they belong is True. If no statements are to be executed in that block, use the `pass` (no-op) statement.
- The statements in each *actions* group should all be indented on the same level, and at least one more space than the `if` statement.

Loops

- Python has 2 looping statements: `while` and `for`.
- The `while` statement is used to repeat a statement of group of statements as long as some condition is true.
- The `for` statement is used to iterate over a an "iterable" object. Sequences like lists are the simplest and most common iterable objects. It is similar to "for each" in other languages.
- Like the `if` statement, both `while` and `for` identify their contents using indentation.

Loops

- Breaking out of a loop (either type) can be done using the `break` statement, while skipping the rest of the current iteration can be done using `continue`.
- Unlike most other languages, both looping statements in Python take an optional `else` clause. This is executed once when the loop is exited (when the condition is false in `while` loops or when the iterable is exhausted in `for` loops).

List Comprehension

- There's a shortcut for looping over a list to produce a new list. The syntax is:
  - `result = [`*`expression`* **`for`** *`variable`* **`in`** *`list`* **`if`** *`condition`*`]`
  - *`list`* is the list whose elements we are processing.
  - *`variable`* is the name referring to the current element.
  - *`condition`* is a filtering condition expression.
  - *`expression`* is an expression, usually using *`variable`*.

python

List Comprehension

- Examples:
  - `x = [1, 2, -3, 8, 9]`
  - `y = [i*i for i in x]` y $\longrightarrow$ `[1, 4, 9, 64, 81]`
  - `z = [num for num in x if num >= 0]` z $\longrightarrow$ `[1, 2, 8, 9]`
  - `u = [5 for number in x]` u $\longrightarrow$ `[5, 5, 5, 5, 5]`
  - `v = [x + [i] for i in x]`

    v $\longrightarrow$ `[[1, 2, -3, 8, 9, 1], [1, 2, -3, 8, 9, 2],`
    `[1, 2, -3, 8, 9, -3], [1, 2, -3, 8, 9, 8],`
    `[1, 2, -3, 8, 9, 9]]`

Function Calls

- Functions are called in Python in a way similar to most other languages, using the function name followed by a list of zero or more arguments between brackets.
- Functions can also be called on objects by prefixing the call with the name of a variable pointing to the object.
- Parameters can be positional or "keyword". Keyword arguments can come in any order and specify parameter name/value pairs.

Function Call Examples

- Ordinary calls:
  - `do_my_bidding()`
  - `abs(x)`
  - `say('hello')`
  - `max(4, 3, 5, 2)`
- Calls on objects:
  - `'hello'.upper()`
  - `'hello'.strip().upper().replace('ll', 'l')`
  - `[1, 2, 3].append(5)`
  - `'this is a word'.replace('word', 'sentence')`

Function Call Examples

- Calls with keywords arguments:
  - `refresh(completely=True)`
  - `[3, 1, 2].sort(reverse=True)`
  - `[3, 1, 2].sort(key=my_key)`
  - `dict(hello='goodbye', this='that')`
  - `f(1, 2, 3, other=8)`
  - `f(1, 2, other=8, 3)` ← ERROR
- Note: keyword arguments can't come before positional arguments.

Modules

- Modules are Python code libraries.
- Every Python script file can be used as a module, and the standard distribution provides a large library of such scripts.
- Modules can be imported into a Python script in several ways, all using the `import` statement.
- Once imported, a module is an object.
- Module documentation is usually included with the module. Use the `help()` function to access it!

Modules

- Modules in Python can be arranged into a tree structure using "packages".
- A package is simply a folder containing Python modules of other packages, in addition to a special file called `__init__.py`.
- The `__init__.py` file is often left empty. However, it can contain code to be executed when the package is first imported.

Modules

- Importing examples:
  - `import os`
  - `import xml.parsers.expat`
    - Imports the whole module and creates a variable referring to it.
  - `from sys import *`
    - Imports everything defined in the module and creates variables referring to the objects.
  - `from time import sleep, clock`
    - Imports specific objects from the module.

Modules

- Usage examples:
  - import os

```
os.mkdir('C:/example')
print os.name

if os.path.exists('C:/Program Files'):
    print 'Found the program files folder!'
```

  - from os import *

```
mkdir('C:/example')
print name

if path.exists('C:/Program Files'):
    print 'Found the program files folder!'
```

Modules

- Usage examples:

  -
    ```
    import time

      t = time.time()
      time.sleep(2.5)
      t2 = t - time.time()

      print 'Milliseconds passed: ', t2
    ```

  -
    ```
    from sys import version, platform

      print 'Your Python version is:', version
      print 'Your platform is:', platform
    ```