

Sistemas Web Desconectados

Defossé Nahuel, van Haaster Diego Marcos

30 de mayo de 2009

Índice general

| | |
|---|-----------|
| 1. Introducción | 1 |
| 1.1. Motivación | 1 |
| 1.2. Objetivos | 2 |
| 1.3. Alcance | 2 |
| 2. | 3 |
| 3. Tecnologías Web | 4 |
| 3.1. Recursos | 4 |
| 3.2. Lenguajes de programación para la web | 4 |
| 3.3. El lenguaje de programación Python | 4 |
| 3.3.1. Desarrollo del lenguaje | 6 |
| 3.3.2. Tipos de datos y funciones | 6 |
| 3.3.3. Espacio de nombres en Python | 7 |
| 3.3.4. Sobrecarga de operadores | 8 |
| 3.3.5. Funciones en Python | 8 |
| 3.3.6. Clases en Python | 8 |
| 3.3.7. Expresiones regulares en Python | 9 |
| 3.4. Frameworks web | 9 |
| 3.4.1. CGI | 9 |
| 3.4.2. WSGI | 10 |
| 3.5. Django | 11 |
| 4. Tecnologías Web del lado del Cliente | 15 |
| 4.1. Estructura de un navegador | 15 |
| 4.2. Evolución del lenguaje en el cliente | 15 |
| 4.3. Propuesta de extensión de Google Gears | 15 |
| 4.4. Propuesta de extensión y aprovechamiento de los avances en cliente | 15 |
| 4.5. Propuesta de framework MTV en el cliente | 15 |
| 5. Integración y metodología de desarrollo con framework web | 16 |
| 5.1. Mecanismos de extensión de Django | 16 |
| 5.1.1. Comandos Personalizados | 16 |
| 5.2. Integración | 17 |
| 5.3. Convivencia | 17 |
| 5.3.1. Modos de trabajo | 17 |

| | |
|--|-----------|
| 6. Lineas futuras | 18 |
| 6.1. Sitio de administración | 18 |
| 6.2. Historial de navegación | 18 |
| 6.3. Workers con soporte para Javascript 1.7 | 18 |
| A. Protopy | 19 |
| A.1. Modulos | 19 |
| A.1.1. builtin | 20 |
| A.1.2. sys | 20 |
| A.1.3. exception | 20 |
| A.1.4. event | 20 |
| A.1.5. timer | 20 |
| A.1.6. ajax | 20 |
| A.1.7. dom | 20 |
| A.2. sys | 20 |
| A.3. exception | 21 |
| A.4. event | 21 |
| A.5. timer | 22 |
| A.6. ajax | 22 |
| A.7. dom | 23 |
| A.8. builtin | 23 |
| A.9. Extendiendo Javascript | 27 |
| A.9.1. String | 27 |
| A.9.2. Number | 29 |
| A.9.3. Date | 29 |
| A.9.4. Element | 29 |
| A.9.5. Forms | 30 |
| A.9.6. Forms.Element | 30 |
| B. Doff | 32 |
| C. MIME | 33 |
| C.1. Introducción | 34 |
| D. Plataforma Mozilla | 36 |

Índice de figuras

| | |
|--|----|
| 3.1. Estructura básica de Django | 12 |
| 3.2. Esquema de flujo de una aplicación Django | 14 |

Índice de cuadros

Agradecimientos

A nuestros familiares

Resumen

Capítulo 1

Introducción

Yo sólo puedo mostrarte la
puerta. Tú eres quien debe
atravesarla.

Morfeo

1.1. Motivación

Hoy más que ayer, pero seguramente menos que mañana, Internet es “la red de redes”. El alto contenido de información Hoy en día Internet supone más que un medio para obtener información, su constante expansión a convertido a esta red en un terreno muy atractivo para la implementación de sistemas de información.

Las aplicaciones web son populares debido a lo práctico que resulta el navegador web como cliente de acceso a las mismas. También resulta fácil actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. En la actualidad, existe una gran oferta de frameworks web para facilitar el desarrollo de aplicaciones web. Una ventaja significativa de las aplicaciones web es que funcionan independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para los múltiples sistemas operativos, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Las aplicaciones web tienen ciertas limitaciones en las funcionalidades que ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio, como dibujar en la pantalla o arrastrar y soltar, que no están soportadas por las tecnologías web estándar. Los desarrolladores web, generalmente, utilizan lenguajes interpretados o script en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez. Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con tecnologías en el lado del servidor. Los sistemas operativos actuales de propósito general cuentan con un navegador web, con posibilidades de acceso a bases de datos y almacenamiento de código y recursos. La web, en el ámbito del software, es un medio singular por su ubicuidad y sus estándares abiertos. El conjunto de normas que rigen la forma en que se generan y transmiten los documentos a través de la web son regulados

por la W3C (Consorcio World Wide Web). La mayor parte de la web está soportada sobre sistemas operativos y software de servidor que se rigen bajo licencias OpenSource¹ (Apache, BIND, Linux, OpenBSD, FreeBSD). Los lenguajes con los que son desarrolladas las aplicaciones web son generalmente OpenSource, como e PHP, Python, Ruby, Perl y Java. Los frameworks web escritos sobre estos lenguajes utilizan alguna licencia OpenSource para su distribución; incluso frameworks basados en lenguajes propietarios son liberados bajo licencias OpenSource.

1.2. Objetivos

Podemos decir que las aplicaciones tradicionales, que no hacen uso de la web, son más robustas ya que no dependen de una conexión. Por lo tanto, sería deseable poder dotar a las aplicaciones web de la capacidad de trabajar cuando no cuentan con conexión. Si bien los elementos necesarios para llevar a cabo esta tarea están disponibles actualmente, no están contemplados en los diseños de los frameworks web. Es decir, cuando una determinada aplicación web debe ser transportada al cliente, es necesario escribir el código de soporte específico para esa aplicación. Un framework no constituye un producto per sé, sino una plataforma sobre la cual construir aplicaciones. Consideramos que sería beneficioso aportar una extensión a un framework web OpenSource que brinde facilidades para transportar las aplicaciones web, basadas en éste, al cliente de manera que la aplicación que haga uso de nuestra extensión pueda ser ejecutada a posteriori en el navegador en el cual ha sido descargada. El framework web será elegido tras un estudio de las características que consideramos más importantes para el desarrollo veloz, como la calidad del mapeador de objetos (entre las características más importantes de éste buscaremos eficiencia en las consultas a la base de datos, ejecución demorada para encadenamiento de consultas, implementación de herencia, baja carga de configuración), la simplicidad para enlazar url's a funciones controladoras, extensibilidad del sistema de escritura de plantillas. Buscaremos frameworks que permitan la ejecución transversal de cierto tipo de funciones, para ejecutar tareas como compresión de salida, sustitución de patrones de texto, caché, control de acceso, etc.

La World Wide Web, o *web*, durante los últimos años ha ganado terreno como plataforma para aplicaciones del variado tipo. Diversas tecnologías fueron formuladas para convertir el escenario inicial, donde la web se limitaba a ser una gran colección de documentos enlazados (*hipertexto*), para llegar a ser...

Vamos a realizar un breve análisis sobre las tecnologías que son utilizadas en la web.

Luego un análisis de las tecnologías del cliente, haciendo hincapié en ...

http://es.wikiquote.org/wiki/The_Matrix

1.3. Alcance

Aca ponemos hasta donde nos vamos a llegar.

Capítulo 2

Capítulo 3

Tecnologías Web

3.1. Recursos

Acá hablamos brevemente de Mozilla (cuentito de como evolucionó Javascript), Javascript y como se perfila como estándar de interconexión de sistemas (mashups), Python y CGI).—

3.2. Lenguajes de programación para la web

3.3. El lenguaje de programación Python

El lenguaje de programación Python fue creado por Guido van Rossum en el año 1991. Python es un lenguaje de programación multiparadigma. Es decir, permite al programador utilizar diferentes formas de resolución de problemas

- programación orientada a objetos
- programación estructurada
- programación funcional

Otros paradigmas (como programación lógica) están soportados mediante el uso de extensiones ¹².

Python usa tipo de dato dinámico y reference counting para el manejo de memoria. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Nuevos módulos se pueden escribir fácilmente en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. Aunque el diseño de Python es de alguna manera hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

¹PySWIP <http://code.google.com/p/pyswip/>

²python-logic

Filosofía del lenguaje

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" (ünpythonic.^{en} inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en *El Zen de Python*

1. Bello es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Ralo es mejor que denso.
7. La legibilidad cuenta.
8. Los casos especiales no son tan especiales como para quebrantar las reglas.
Aunque lo práctico gana a la pureza.
9. Los errores nunca deberían dejarse pasar silenciosamente.
A menos que hayan sido silenciados explícitamente.
10. Frente a la ambigüedad, rechaza la tentación de adivinar.
11. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
Aunque esa manera puede no ser obvia al principio a menos que usted sea Holandés
12. Ahora es mejor que nunca
Aunque nunca es a menudo mejor que ya
13. Si la implementación es difícil de explicar, es una mala idea.
14. Si la implementación es fácil de explicar, puede que sea una buena idea
15. Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Desde la versión 2.1.2, Python incluye estos puntos (en su versión original en inglés) como un huevo de pascua que se muestra al ejecutar

```
>>> import this
```

3.3.1. Desarrollo del lenguaje

3.3.2. Tipos de datos y funciones

Python cuenta con los siguientes tipos de datos incorporados: *int*, *str*, *bool*, *float*, *complex*. Estos tipos de datos son denominados **inmutables**, es decir, para que su valor cambie, la instancia anterior es destruida.

Python no cuenta con el tipo de dato arreglo, sin embargo cuenta con 3 sustitutos. El tipo de dato tupla (*tuple()*) es un tipo de dato inmutable, que agrupa de manera *ordenada* un conjunto de objetos del mismo o diferente tipo. El concepto de orden de los elementos es importante, ya que python tampoco provee la estructura *for* tradicional de lenguajes procedurales como C o Pascal. Una tupla puede ser considerada como un arreglo fijo de valores que pueden tener diferentes tipos. Las tuplas pueden ser accedidas secuencialmente (*iteradas*) en orden ascendente o descendente, se puede acceder al n-ésimo item, pero no puede ser modificadas se generará una excepción debido a que es un tipo de datos estático.

```

1 a = (1, 2, 3) # Una tupla de enteros
2 b = (1, 1.0, 'pasta') # Una tupla con un conjunto dispar
3                       # de elementos
4 c = ( (1, 2), (3, 4), (5, 6) ) # Una tupla que tiene
5                               # por elementos otras tuplas

```

Python soporta asignación a con la sintaxis de tupla, es decir, al colocar un conjunto de nombres separados por comas en el lado derecho de una asignación y un conjunto de valores o variables sobre la izquierda, la asignación es posicional.

Listing 3.1: Asignación en Python

```

1 a, b = 1, 2      # Es equivalente a (a, b) = (1, 2)
2 a, b = b, a      # Operación de intercambio sin auxiliar
3 c = (1, 2, 3, 'bacalao')
4 w, x, y, z = c   # Asignación posicional

```

Listing 3.2: Asignación en Python

```

1 function pepe() {}

```

Una lista es una colección ordenada de objetos del mismo o diferente tipo que pueden ser modificados. Básicamente la lista se comporta como una tupla mutable. `a = [1, 2, 3, 4]` `b[0] = 4`

Los *bloques* en Python son conjuntos de sentencias que se encuentran a bajo un cierto nivel de indentación. Un bloque comienza con dos puntos (:) y termina cuando se pierde el nivel de indentación que lo definía. Veremos ejemplos de bloques con las sentencias de control de flujo.

Bloques y estructuras de control

La sentencia *if* ejecuta el *if* condicion: `bloquesicondicionverdaderaelse : bloquesicondicionfalsa`

3.3.3. Espacio de nombres en Python

Cuando comenzamos a ejecutar el intérprete de Python de forma interactiva o cuando lo invocamos para ejecutar un script, están disponibles un conjunto de funciones, clases, clases de excepciones. El conjunto de estos nombres se conoce como ámbito de nombres `__builtin__` (*incorporado*) y en este encontramos: `int()`, `char()`, `str()`, `object`, `Exception`, `TypeError`, `ValueError`, `True`, `False`, `range`, `map`, `zip`, `locals`, `globals`, entre otras. Pueden verse en el interprete interactivo mediante

```
>>> dir(__builtins__)
```

Un *ámbito de nombres* en Python es un mapeo de nombres a objetos. La mayoría de los ámbitos de nombres están implementados mediante diccionarios, but that's normally not noticeable in any way (except for performance), y podría cambiar en el futuro. Otros ejemplos de ámbitos de nombre son los nombres globales en un módulo; los valores locales en la invocación de una función y en cierta forma, los atributos de un objeto también forman un ámbito de nombres. Es importante tener en cuenta que no existe ninguna relación entre nombres de diferentes ámbitos de nombres.

Un *módulo* es un archivo con definiciones y sentencias en Python. El conjunto de todas las sentencias que no están indentadas, forman el ámbito de nombres global del modulo, las funciones, variables y clases que están suelen decirse que estan a “nivel módulo”. Los elementos que pueden ser parte del ámbito de nombres del módulo son:

- una definición de una función
- una variable
- una expresión lambda
- el nombre de otro módulo importado

Un número, el llamado a función o el lanzado de una excepción no alteran el ámbito de nombres. Un módulo puede ser cargado desde el intérprete interactivo en el ámbito de nombres global o desde otro módulo (en su propio espacio de nombres) a través de la sentencia `import os`. En este caso, el intérprete buscará secuencialmente en el `PYTHONPATH` un módulo llamado **os**, en caso de encontrarlo, lo cargará y en el ámbito de nombres actual generará una referencia con el nombre “os”. A esta actividad se la llama importación y el interprete garantiza que un módulo se cargará de unicamente una sola vez, en otras palabras, existe una única instancia de un módulo. En este caso, si varios módulos importan a “os”, solo la primer ocurrencia realiza la importación efectiva. El resto de las sentencias `import` solamente generan referencias a la primera “instancia” de un módulo.

La sentencia `import` permite a su vez hacer una importación selectiva de símbolos de un módulo, por ejemplo: `from os import path`. En este caso, `import` crea la instancia del módulo en la máquina virtual, pero no expone todo el contenido al ámbito de nombres local, sino que únicamente genera una referencia al símbolo `path`.

El comodín `*` permite importar todos los símbolos definidos en un módulo al espacio de nombres local. `from os import *`

Esta técnica debe ser utilizada con cuidado, debido a que “contamina” el ámbito de nombres local. Para mitigar este problema, puede definirse una variable `__all__` al comienzo del archivo, con una tupla en la cual se definen los símbolos que se desean exportar. Suponiendo que el siguiente código se encuentra en el módulo `promedio`:

```
__all__=('calcular_promedio',)
def calcular_promedio(numeros): return suma(numeros)/len(numeros)
def suma(lista):
    return float(sum(lista))
```

Al realizar `from promedio import *`

El único símbolo que encontraremos en nuestro espacio de nombres será, `calcular_promedio`.

3.3.4. Sobrecarga de operadores

3.3.5. Funciones en Python

En python, una función se define de la siguiente manera: `def funcion(argumento1, argumento2):` `''' Doc de la funcion '''` `print argumento1, argumento2` Python soporta lo que se ha dado en llamar *empaquetado de argumentos* tanto en la definición como en la invocación de una función.

El empaquetado más simple es el del tipo lista, que consiste en la habilidad de una función de recibir argumentos variables en formato lista. Para usar esta característica es necesario utilizar una signatura especial (agregando un asterisco).

```
def promedio(*lista): ''' Calcula promedio ''' return sum(lista) / float(len(lista))
```

utilizando varios builtins

Ejemplo de utilización: `promedio(1, 2, 4, 5, 5)` 3.3999999999999999

Esta signatura de argumentos se puede combinar con argumentos posicionales. `def dividir_suma_por(numero, *lista): return sum(lista)/numero` La capacidad de recibir listas puede ser utilizada a la inversa. Es decir, conociendo que una función recibe una cantidad de argumentos, “acomodar” en una lista los argumentos que se desean utilizar. Utilizando la función `promedio` anterior

```
lista = [1, 2, 4, 6, 7]
promedio(*lista)
```

4.0

3.3.6. Clases en Python

En el lenguaje Python, todos los tipos de datos son objetos. No existe el concepto de tipo de dato primitivo (como en java). Las clases definidas por el usuario son a su vez objetos. A partir de la versión 2.2 del lenguaje, aparecen new-style-classes que permiten utilizar descriptores (base para implementación de propiedades) y metaclasses. Una metaclass es una clase que define la estructura de otra clase y es una forma dinámica de definir la estructura de una clase. Esta característica es considerada compleja, pero es utilizada en varios componentes de Django (como la generación de formularios a partir de la definición de un modelo o la generación de interfaces de CRUD en la administración integrada).

Instanciación de una clase

En Python no existe el concepto de constructor como en los lenguajes C++, Objective-C o Java. La instanciación de una clase tiene dos fases: una fase de

creación estructural y otra de inicialización. Estos métodos utilizan los nombres reservados `__new__` e `__init__`.

*necesitamos saber que son `*largs` y `**kwargs`*

3.3.7. Expresiones regulares en Python

Las expresiones reuglares proveen un medio conciso y flexible de identificar cadenas en un texto, como caracteres en particular, palabras o patrones de caracteres. Las expresiones regulares (a menudo abreviadas **regex**, o **regex****p**, o en plural **regexes**, **regexps**, o incluso **regexen**) son escritas en un lenguaje formal que puede ser interpretado por un procesador de expresiones regulares. Las expresiones regulares se utilizan en muchos editores de texto, utilitarios³ y lenguajes de programación.

POSIX Prel-like

El módulo *re* en python provee un motor de análisis de expresiones regulares. La sintaxis de definición de expresiones con captura nombrada de grupos.

3.4. Frameworks web

Acá tenemos que justificar por que django

3.4.1. CGI

Interfaz de entrada común (en inglés Common Gateway Interface, abreviado CGI) es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME⁴. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGIs.

Las aplicaciones CGI fueron una de las primeras maneras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este programa puede estar hecho en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

CGI ha hecho posible la implementación de funciones nuevas y variadas en las páginas web, de tal manera que esta interfaz rápidamente se volvió un estándar, siendo implementada en todo tipo de servidores web.

Ejecución de CGI

A continuación se describe la forma de actuación de un CGI de forma esquemática:

1. En primera instancia, el servidor recibe una petición (el cliente ha activado un URL que contiene el CGI), y comprueba si se trata de una invocación de un CGI.

³ Como las utilidades de consola UNIX *sed*, *grep*, etc.

⁴Ver apéndice sobre MIMEC

2. Posteriormente, el servidor prepara el entorno para ejecutar la aplicación. Esta información procede mayoritariamente del cliente.
3. Seguidamente, el servidor ejecuta la aplicación, capturando su salida estándar.
4. A continuación, la aplicación realiza su función: como consecuencia de su actividad se va generando un objeto MIME que la aplicación escribe en su salida estándar.
5. Finalmente, cuando la aplicación finaliza, el servidor envía la información producida, junto con información propia, al cliente, que se encontraba en estado de espera. Es responsabilidad de la aplicación anunciar el tipo de objeto MIME que se genera (campo `CONTENT_TYPE`), pero el servidor calculará el tamaño del objeto producido.

Intercambio de información: Variable de Entorno

Variables de entorno que se intercambian de cliente a CGI:

QUERY_STRING Es la cadena de entrada del CGI cuando se utiliza el método GET sustituyendo algunos símbolos especiales por otros. Cada elemento se envía como una pareja `Variable=Valor`. Si se utiliza el método POST esta variable de entorno está vacía

CONTENT_TYPE Tipo MIME de los datos enviados al CGI mediante POST. Con GET está vacía. Un valor típico para esta variable es: `Application/X-www-form-urlencoded`

CONTENT_LENGTH Longitud en bytes de los datos enviados al CGI utilizando el método POST. Con GET está vacía

PATH_INFO Información adicional de la ruta (el "path") tal y como llega al servidor en el URL

REQUEST_METHOD Nombre del método (GET o POST) utilizado para invocar al CGI

SCRIPT_NAME Nombre del CGI invocado

SERVER_PORT Puerto por el que el servidor recibe la conexión

Variables de entorno que se intercambian de servidor a CGI:

SERVER_SOFTWARE Nombre y versión del software servidor de www

SERVER_NAME Nombre del servidor

GATEWAY_INTERFACE Nombre y versión de la interfaz de comunicación entre servidor y aplicaciones CGI/1.12

3.4.2. WSGI

The Web Server Gateway Interface defines a simple and universal interface between web servers and web applications or frameworks for the Python programming language. The latest version 3.0 of Python, released in December 2008, is already supported by `mod_wsgi` (*a module for the Apache Webserver*).

¿Por qué WSGI?

Historically Python web application frameworks have been a problem for new Python users because, generally speaking, the choice of web framework would limit the choice of usable web servers, and vice versa. Python applications were often designed for either CGI, FastCGI, *mod_python* or even custom API interfaces of specific web-servers.

WSGI⁵ (sometimes pronounced 'whiskey' or 'wiz-gee') was created as a low-level interface between web servers and web applications or frameworks to promote common ground for portable web application development. WSGI is based on the existing CGI standard.

Descripción general de la especificación

The WSGI has two sides: the "server" or "gateway" side, and the "application" or "framework" side. The server side invokes [clarification needed] a callable object (usually a function or a method) that is provided by the application side. Additionally WSGI provides middleware; WSGI middleware implements both sides of the API, so that it can be inserted "between" a WSGI server and a WSGI application – the middleware will act as an application from the server's point of view, and as a server from the application's point of view.

Un módulo middleware puede realizar operaciones como:

- Routing a request to different application objects based on the target URL, after changing the environment variables accordingly.
- Allowing multiple applications or frameworks to run side-by-side in the same process
- Load balancing and remote processing, by forwarding requests and responses over a network
- Perform content postprocessing, such as applying XSLT stylesheets

Aplicación de ejemplo

A WSGI compatible "Hello World" application in Python syntax: *continuar*

3.5. Django

Django es un framework web escrito en Python⁶ el cual sigue vagamente el concepto de Modelo Vista Controlador. Ideado inicialmente como un administrador de contenido para varios sitios de noticias, los desarrolladores encontraron que su CMS era lo suficientemente genérico como para cubrir un ámbito más amplio de aplicaciones. Fue liberado⁷ bajo la licencia BSD en Julio del 2005 como Django Web Framework en honor a Django Reinhart. En junio del 2008 fue anunciada la creación de la Django Software Foundation, la cual se hace cargo hasta la fecha del desarrollo y mantenimiento.

⁵PEP 333, Python Web Server Gateway Interface v1.0

⁶Ver apartado sobre el lenguaje Python 3.3

⁷En el ámbito del software libre, la liberación es la fecha en la cual se pone a disposición de la comunidad del software en cuestión

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos (**django.contrib.admin**), que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para syndicar contenido via RSS y/o Atom, "páginas planas" que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

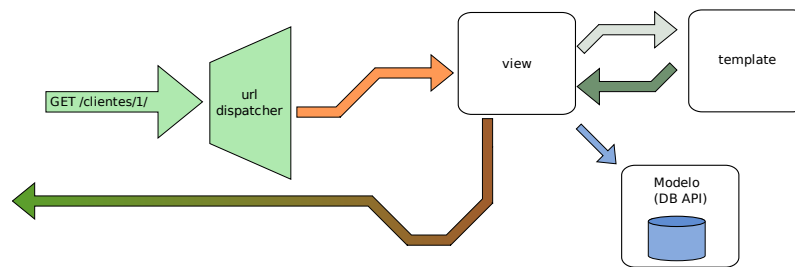


Figura 3.1: Estructura básica de Django

Django como framework de desarrollo consiste en un conjunto de utilidades de consola que permiten crear y manipular proyectos y aplicaciones.

Estructura de un proyecto

Un proyecto funciona como un contenedor de aplicaciones que se rigen bajo la misma base de datos, los mismos templates y las mismas clases de middleware.

Una aplicación es un paquete que contiene al menos los módulos **models.py** y **views.py**, y generalmente suele agregarse un módulo **urls.py**.

Un proyecto Django consiste en 3 módulos⁸ básicos:

Módulo **manage.py**

Esta es la interfase con el framework. Este módulo permite crear aplicaciones, testear que los modelos de una aplicación estén bien definidos (validación), iniciar el servidor de desarrollo, crear volcados de la base de datos y restaurarlos (restaurarlos (*fixtures*, utilizados en casos de pruebas y para precarga de datos conocidos).

⁸Un módulo en Python, es un archivo con extensión `.py`

Módulo `settings.py`

El módulo *settings* define la configuración transversal a las aplicaciones de usuario. En este módulo no se suelen definir más que constantes. Dentro de estas constantes encontramos la base de datos sobre la cual trabaja el ORM, el(los) directorio(s) de las plantillas, las clases middleware, ubicación de los medios estáticos⁹. En este módulo se definen la lista de aplicaciones instaladas.

Módulo `urls.py`

Este módulo define las asociaciones entre las URL y las funciones (vistas) que las atienden. Para generar código más modular, django permite delegar urls que cumplan con cierto patrón a un otro módulo. Típicamente este módulo se llama también `urls` y es parte de una aplicación (Ej: tratar todo lo que comience con `/clientes/` con el módulo `mi_proyecto.mi_aplicacion.urls`).

Una *expresión regular* es una forma de definir un patrón en una cadena. Mediante ésta técnica se realizan validaciones y búsquedas de elementos en cadenas. En python se define además una forma de otorgarle un alias a los elementos buscados (en contraposición a la forma tradicional que utiliza un índice numérico). Esta particularidad de las expresiones regulares ha sido explotada para la asociación de las URLs a las funciones que las atienden. Cuando el cliente realiza acceso a una URL dentro de un proyecto django, esta es chequeada contra cada patrón definido como url, en caso de éxito, se ejecuta la función asociada o vista. Si la expresión regular tiene definido grupos nombrados, cada subcadena pasa a ser argumento

pasan a ser argumentos de la vista, es decir, cada grupo nombrado, pasa a ser argumento de la función asociada.

Elementos de una aplicación Django

Una aplicación consiste en 2 módulos fundamentales.

Módulo `models.py`

En este módulo se definen los modelos.

Módulo `views.py`

En este módulo se definen las vistas. Una vista es una función que recibe como primer argumento un objeto `HttpRequest`¹⁰, el cual encapsula la información proveniente del request, como el método (GET, POST), los elementos de la query http.

⁹ Un medio estático es todo contenido que no se genera dinámicamente, como imágenes, librerías de javascript, contenido para embeber como archivos multimedia u elementos

¹⁰ [Documentación oficial sobre HttpRequest en `django`project.com](#)

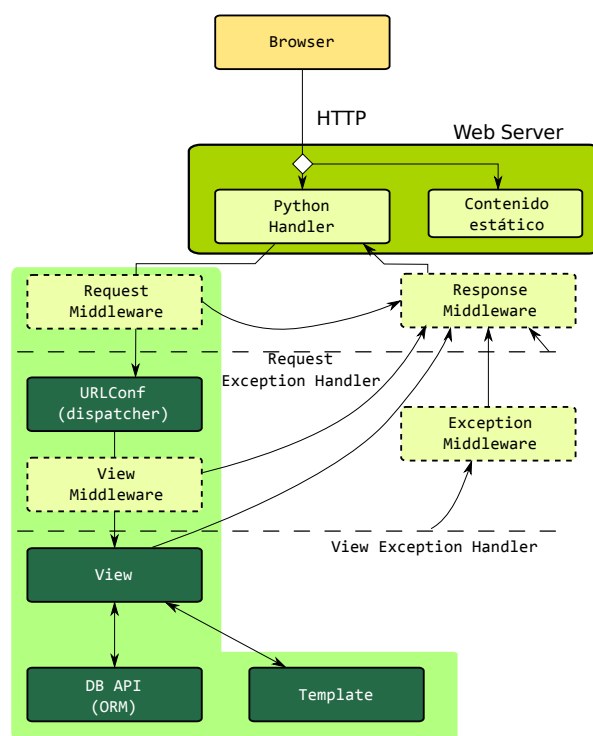


Figura 3.2: Esquema de flujo de una aplicación Django

Capítulo 4

Tecnologías Web del lado del Cliente

- 4.1. Estructura de un navegador
- 4.2. Evolución del lenguaje en el cliente
- 4.3. Propuesta de extensión de Google Gears
- 4.4. Propuesta de extensión y aprovechamiento de los avances en cliente
- 4.5. Propuesta de framework MTV en el cliente

Capítulo 5

Integración y metodología de desarrollo con framework web

5.1. Mecanismos de extension de Django

5.1.1. Comandos Personalizados

Para poder realizar extensiones sobre la funcionalidad de Django sus desarrolladores han añadido la posibilidad de crear comandos personalizados. Un comando personalizado se define en un módulo, dentro de una aplicación y es ejecutado de manera similar a cualquier otro comando provisto por el framework (como *syncdb*, *runserver*, *dumpdata*).

De igual manera que los comandos provistos por el framework, los comandos personalizados son ejecutados mediante el script de administración de proyecto *manage.py* o *django-admin.py*.

Un comando se define dentro de una aplicación. No existen comandos a nivel proyecto. Para que estos comandos sean detectados por el módulo de administración de proyectos deben respetar la siguiente estructura dentro del directorio de la aplicación

```
management/  
  __init__.py  
  commands/  
    __init__.py  
    mi_comando.py
```

Además la aplicación debe estar incluida en **INSTALLED_APPS** dentro de el módulo de configuración de proyecto **settings.py**.

Listing 5.1: Comando personalizado básico en Django

```
1  
2 from django.core.management.base import NoArgsCommand  
3
```

```
4 class Command(NoArgsCommand):
5     help = '''Descripci n del comando '''
6     def handle_noargs(self, **options):
7         print "Soy un comando personalizado"
```

Desde el punto de vista del lenguaje, la creación de un comando es la creación de un módulo `management.commands.nombre_comando`, en el cual se extiende a alguna clase base `Command` (que se encuentran en `django.core.management.base`).

Las clases base disponibles son `NoArgsCommand`, para la implementación de comandos que no reciben argumentos; `AppLabelCommand` para la implementación de comandos que reciben como argumento el nombre de una aplicación instalada y

5.2. Integración

5.3. Convivencia

5.3.1. Modos de trabajo

Los distintos modos de trabajo de la aplicación son:

Estado online sin soporte offline

En este modo la aplicación trabaja directamente sobre el servidor de aplicación y depende completamente de una conexión. Al activar e implementar el soporte offline para nuestra aplicación, en el servidor; el cliente debe ser consciente de este cambio y pasar a soportar los modos.

Estado online con soporte offline

Estado offline con soporte offline

Capítulo 6

Lineas futuras

- 6.1. Sitio de administración
- 6.2. Historial de navegación
- 6.3. Workers con soporte para Javascript 1.7

Apéndice A

Protopy

El desarrollo de un framework en javascript que funcione del lado del cliente presupone gran cantidad de código viajando de un lado al otro de la conexión. Previendo básicamente este postulado desarrollamos una librería que brinde el soporte a los requerimientos más básicos. Esta librería constituye la base para posteriores construcciones más complejas en el cliente y aun herramientas que simplifican el desarrollo client-side.

A.1. Módulos

Uno de los principales inconvenientes a los que protopy da solución es a la inclusión dinámica de funcionalidad bajo demanda, esto es logrado mediante los módulos. Básicamente un módulo en un archivo con código javascript que reside en el servidor y es obtenido y ejecutado en el cliente.

Listing A.1: Estructura de un módulo

```
1 //Archivo: tests/module.js
2 require('event');
3
4 var h1 = $('titulo');
5
6 function set_texto(txt) {
7     h1.update(txt);
8 }
9
10 function get_texto() {
11     return h1.innerHTML;
12 }
13
14 event.connect($('titulo'), 'click', function(event) {
15     alert('El texto es: ' + event.target.innerHTML);
16 });
17
18 publish({
19     set_texto: set_texto,
20     get_texto: get_texto
```

21 });

Listing A.2: Test

```
>>> require('tests.module')
GET http://localhost:8080/tests/protopy/module.js
Object __file__=/tests/protopy/module.js
>>> module.get_texto()
"Test de modulo"
>>> module.set_texto('Un titulo')
>>> require('tests.module', 'get_texto')
get_texto()
>>> get_texto()
"Un titulo"
>>> require('tests.module', '*')
>>> set_texto('Hola luuu!!!')
>>> get_texto()
"Hola luuu!!!"
```

A.1.1. builtin

A.1.2. sys

A.1.3. exception

A.1.4. event

A.1.5. timer

A.1.6. ajax

A.1.7. dom

A.2. sys

version

version: 0.8,

browser

browser: IE: !!(!window.attachEvent navigator.userAgent.indexOf('Opera')
 === -1), Opera: navigator.userAgent.indexOf('Opera') <-1, WebKit: naviga-
 tor.userAgent.indexOf('AppleWebKit/') <-1, Gecko: navigator.userAgent.indexOf('Gecko')
 <-1 navigator.userAgent.indexOf('KHTML') === -1, MobileSafari: !!naviga-
 tor.userAgent.match(/Apple.*Mobile.*Safari/), features: XPath: !!document.evaluate,
 SelectorsAPI: !!document.querySelector, ElementExtensions: !!window.HTMLElement,
 SpecificElementExtensions: document.createElement('div')[_proto_]document.createElement('div')[_proto_]!!=document.crea

get_gears

get_gears: get_gears,

register_path

register_path: function(module, path)

module_url

module_url: function(name, postfix)

modules

modules: ModuleManager.modules,

paths

paths: ModuleManager.paths

A.3. exception

```
var exception = ModuleManager.create('exceptions', 'built-in', Exception:
Exception, AssertionError: type('AssertionError', Exception), AttributeError:
type('AttributeError', Exception), LoadError: type('LoadError', Exception), Key-
Error: type('KeyError', Exception), NotImplementedError: type('NotImplementedError',
Exception), TypeError: type('TypeError', Exception), ValueError: type('ValueError',
Exception), );
```

A.4. event**connect**

connect(object, event, context, method)

disconnect

disconnect(handle)

subscribe

subscribe(topic, context, method)

unsubscribe

unsubscribe(handle)

publish

publish(topic, args)

connectpublisher

connectpublisher(topic, obj, event)

fixevent

fixevent: function(),

stopevent

stopevent: function(),

keys

keys: BACKSPACE: 8, TAB: 9, CLEAR: 12, ENTER: 13, SHIFT: 16, CTRL: 17, ALT: 18, PAUSE: 19, CAPSLOCK : 20, ESCAPE : 27, SPACE : 32, PAGEUP : 33, PAGEDOWN : 34, END : 35, HOME : 36, LEFTARROW : 37, UPARROW : 38, RIGHTARROW : 39, DOWNARROW : 40, INSERT : 45, DELETE : 46, HELP : 47, LEFTWINDOW : 91, RIGHTWINDOW : 92, SELECT : 93, NUMPAD₀ : 96, NUMPAD₁ : 97, NUMPAD₂ : 98, NUMPAD₃ : 99, NUMPAD₄ : 100, NUMPAD₅ : 101, NUMPAD₆ : 102, NUMPAD₇ : 103, NUMPAD₈ : 104, NUMPAD₉ : 105, NUMPADMULTIPLY : 106, NUMPADPLUS : 107, NUMPADENTER : 108, NUMPADMINUS : 109, NUMPADPERIOD : 110, NUMPADDIVIDE : 111, F1 : 112, F2 : 113, F3 : 114, F4 : 115, F5 : 116, F6 : 117, F7 : 118, F8 : 119, F9 : 120, F10 : 121, F11 : 122, F12 : 123, F13 : 124, F14 : 125, F15 : 126, NUMLOCK : 144, SCROLLLOCK : 145);

A.5. timer**setTimeout**

setTimeout: window.setTimeout,

setInterval

setInterval: window.setInterval,

clearTimeout

clearTimeout: window.clearTimeout,

delay

delay: function(f)

defer

defer: function(f)

A.6. ajax**Request**

new Request: Request,

Response

new Response: Response,

toQueryParams

toQueryParams(string, separator) \rightarrow *object*

toQueryString

toQueryString(params) -¿string

A.7. dom**query**

query: query,

A.8. builtin**publish**

publish(object)

require

require(module[, object...]) -¿module — object

\$

\$(id) -¿HTMLElement \$(id ...) -¿[HTMLElement...]

\$\$

\$\$ (cssRule) -¿[HTMLElement...]

type, El constructor de tipos

type(name, [base...] [, forType], forPrototype) -¿newType

Como ya se menciono anteriormente javascript es un lenguaje orientado a prototipos, para acercarnos un poco a la programacion de objetos, utilizamos una funcion constructora de tipos o clases a la que denominamos “type”.

Listing A.3: Definicion de tipos

```

1 var Animal = type('Animal', object, {
2   contador: 0,
3 }, {
4   __init__: function(especie) {
5     this.especie = especie;
6     this.orden = Animal.contador++;

```

```

7     }
8   });
9
10  var Terrestre = type('Terrestre', Animal, {
11    caminar: function() {
12      console.log(this.especie + ' caminando');
13    }
14  });
15
16  var Acuatico = type('Acuatico', Animal, {
17    nadar: function() {
18      console.log(this.especie + ' nadando');
19    }
20  });
21
22  var Anfibio = type('Anfibio', [Terrestre, Acuatico]);
23
24  var Piton = type('Piton', Terrestre, {
25    __init__: function(nombre) {
26      super(Terrestre, this).__init__(this.__name__);
27      this.nombre = nombre;
28    },
29    caminar: function() {
30      throw new Exception(this.especie + ' no camina');
31    },
32    reptar: function() {
33      console.log(this.nombre + ' la ' + this.especie.
34        toLowerCase() + ' esta reptando');
35    }
36  });
37
38  var doris = new Piton('Doris');
39  var ballena = new Acuatico('Ballena');
40  var rana = new Anfibio('Rana');

```

Listing A.4: Test

```

>>> doris
window.Piton especie=Piton orden=0 nombre=Doris __name__=Piton
>>> rana
window.Anfibio especie=Rana orden=2 __name__=Anfibio
>>> isinstance(rana, Terrestre)
true
>>> isinstance(doris, Animal)
true
>>> issubclass(Anfibio, Acuatico)
true
>>> issubclass(Piton, Animal)
true
>>> doris.caminar()
Exception: Piton no camina args=[1] message=Piton no camina
>>> doris.reptar()
Doris la piton esta reptando

```

extend

extend(destiny, source) -¿alteredDestiny

super

super(destiny, source) -¿alteredDestiny

isundefined

isundefined(destiny, source) -¿alteredDestiny

isinstance

isundefined(destiny, source) -¿alteredDestiny

issubclass

issubclass(destiny, source) -¿alteredDestiny

Arguments

new Arguments(destiny, source) -¿alteredDestiny

Template

new Template(destiny, source) -¿alteredDestiny

Dict

new Dict(destiny, source) -¿alteredDestiny

Set

new Set(destiny, source) -¿alteredDestiny

hash

hash(destiny, source) -¿alteredDestiny

id

id(destiny, source) -¿alteredDestiny

getattr

getattr(destiny, source) -¿alteredDestiny

setattr

setattr(destiny, source) -¿alteredDestiny

hasattr

hasattr(destiny, source) -¿alteredDestiny

assert

assert(destiny, source) -¿alteredDestiny

bool

bool(destiny, source) -¿alteredDestiny

callable

callable(destiny, source) -¿alteredDestiny

chr

chr(destiny, source) -¿alteredDestiny

ord

ord(destiny, source) -¿alteredDestiny

bisect

bisect(destiny, source) -¿alteredDestiny

equal

equal(destiny, source) -¿alteredDestiny

nequal

nequal(destiny, source) -¿alteredDestiny

number

number(destiny, source) -¿alteredDestiny

flatten

flatten(destiny, source) -¿alteredDestiny

include

include(destiny, source) -¿alteredDestiny

len

len(destiny, source) -¿alteredDestiny

array

array(iterable) -¿[element...]

print

print(destiny, source) -¿alteredDestiny

range

range(destiny, source) -¿alteredDestiny

string

string(destiny, source) -¿alteredDestiny

values

values(object) -¿[value...]

keys

keys(object) -¿[key...]

items

items(destiny, source) -¿alteredDestiny

inspect

inspect(destiny, source) -¿alteredDestiny

unique

unique(destiny, source) -¿alteredDestiny

xrange

xrange(destiny, source) -¿alteredDestiny

zip

zip(destiny, source) -¿alteredDestiny
santiago del estero norte 727, tiene un pino grande

A.9. Extendiendo Javascript

A.9.1. String

gsub

gsub(pattern, replacement) -¿string

sub

sub(pattern, replacement[, count = 1]) -¿string

subs

subs(pattern, replacement) -¿string

format

format(pattern, replacement) -¿string

inspect

inspect(*use_double_quotes*) -> *string*

truncate

truncate(length, truncation) -¿string

strip

strip() -¿string

striptags

striptags() -¿string

stripscripts

stripscripts() -¿string

extractscripts

extractscripts() -¿string

evalscripts

evalscripts() -¿string

escapeHTML

escapeHTML() -¿string

unescapeHTML

unescapeHTML() -¿string

succ

succ() -¿string

times

times(count[, separator = ”]) -¿string

camelize

camelize() -¿string

capitalize

capitalize() -¿string

underscore

underscore() -¿string

dasherize

dasherize() -¿string

startswith

startswith(pattern) -¿string

endswith

endswith(pattern) -¿string

blank

blank() -¿string

A.9.2. Number**format**

format(f, radix) -¿string

A.9.3. Date**toISOString**

toISOString() -¿string

A.9.4. Element**visible**

visible()

toggle

toggle()

hide`hide()`**show**`show()`**remove**`remove()`**update**`update(content)`**insert**`insert(insertions)`**select**`select(selector)`**A.9.5. Forms****disable**`disable()`**enable**`enable()`**serialize**`serialize()`**A.9.6. Forms.Element****serialize**`serialize()`**get_value**`get_value()`**set_value**`set_value(value)`**clear**`clear()`

present`present()`**activate**`activate()`**disable**`disable()`**enable**`enable()`

Apéndice B

Doff

Apéndice C

MIME

MIME (*Multipurpose Internet Mail Extensions*), (Extensiones de Correo de Internet Multipropósito), son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos. En sentido general las extensiones de MIME van encaminadas a soportar:

- texto en conjuntos de caracteres distintos de US-ASCII
- adjuntos que no son de tipo texto
- cuerpos de mensajes con múltiples partes (multi-part)
- información de encabezados con conjuntos de caracteres distintos de ASCII.

Prácticamente todos los mensajes de correo electrónico escritos por personas en Internet y una proporción considerable de estos mensajes generados automáticamente son transmitidos en formato MIME a través de SMTP. Los mensajes de correo electrónico en Internet están tan cercanamente asociados con el SMTP y MIME que usualmente se les llama mensaje SMTP/MIME.[1]

En 1991 la IETF (Internet Engineering Task Force) comenzó a desarrollar esta norma y desde 1994 todas las extensiones MIME están especificadas de forma detallada en diversos documentos oficiales disponibles en Internet.

MIME está especificado en seis RFCs (acrónimo inglés de Request For Comments) : RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 y RFC 2077.

Los tipos de contenido definidos por el estándar MIME tienen gran importancia también fuera del contexto de los mensajes electrónicos. Ejemplo de esto son algunos protocolos de red tales como HTTP de la Web. HTTP requiere que los datos sean transmitidos en un contexto de mensajes tipo e-mail aunque los datos pueden no ser un e-mail propiamente dicho.

En la actualidad ningún programa de correo electrónico o navegador de Internet puede considerarse completo si no acepta MIME en sus diferentes facetas (texto y formatos de archivo).

C.1. Introducción

El protocolo básico de transmisión de mensajes electrónicos de Internet soporta solo caracteres ASCII de 7 bit (véase también 8BITMIME). Esto limita los mensajes de correo electrónico, ya que incluyen solo caracteres suficientes para escribir en un número reducido de lenguajes, principalmente Inglés. Otros lenguajes basados en el Alfabeto latino es adicionalmente un componente fundamental en protocolos de comunicación como HTTP, el que requiere que los datos sean transmitidos como un e-mail aunque los datos pueden no ser un e-mail propiamente dicho. Los clientes de correo y los servidores de correo convierten automáticamente desde y a formato MIME cuando envían o reciben (SMTP/MIME) e-mails.

Encabezados MIME

MIME-Version

La presencia de este encabezado indica que el mensaje utiliza el formato MIME. Su valor es típicamente igual a "1.0" por lo que este encabezado aparece como:

MIME-Version: 1.0

Debe señalarse que los implementadores han intentado cambiar el número de versión en el pasado y el cambio ha tenido resultados imprevistos. En una reunión de IETF realizada en Julio 2007 se decidió mantener el número de versión en "1.0" aunque se han realizado muchas actualizaciones a la versión de MIME.

Content-Type

Este encabezado indica el tipo de medio que representa el contenido del mensaje, consiste en un tipo: type y un subtipo: subtype, por ejemplo:

Content-Type: text/plain

A través del uso del tipo multiparte (multipart), MIME da la posibilidad de crear mensajes que tengan partes y subpartes organizadas en una estructura arbórea en la que los nodos hoja pueden ser cualquier tipo de contenido no multiparte y los nodos que no son hojas pueden ser de cualquiera de las variedades de tipos multiparte. Este mecanismo soporta:

- mensajes de texto plano usando text/plain (este es el valor implícito para el encabezado Content-type:)
-
- texto más archivos adjuntos (multipart/mixed con una parte text/plain y otras partes que no son de texto, por ejemplo: application/pdf para documentos pdf, application/vnd.oasis.opendocument.text para OpenDocument text). Un mensaje MIME que incluye un archivo adjunto generalmente indica el nombre original del archivo con un encabezado Content-Disposition: por un atributo name de Content-Type, por lo que el tipo o

formato del archivo se indica usando tanto el encabezado MIME content-type y la extensión del archivo (usualmente dependiente del SO).

```
Content-Type: application/vnd.oasis.opendocument.text;  
             name="Carta.odt"  
Content-Disposition: inline;  
             filename="Carta.odt"
```

- reenviar con el mensaje original adjunto (multipart/mixed con una parte text/plain y el mensaje original como una parte message/rfc822)
- contenido alternativo, un mensaje que contiene el texto tanto en texto plano como en otro formato, usualmente HTML (multipart/alternative con el mismo contenido en forma de text/plain y text/html)
- muchas otras construcciones de mensaje

Apéndice D

Plataforma Mozilla

- Porque desarrollaron e implementaron Javascript 1.7
- Porque javascript 1.7 tomo semántica (y sintaxis???) de Python
- Porque es código abierto
- Porque es extensible mediante plugins
 - Tiene firebug
 - Gears y firebug = muy compardor para el desarrollador.
-

Protopy persigue acercar la semántica del Javascript 1.7 a la del lenguaje Python. Las funcionalidades principales son las siguientes:

- Ámbito de nombres
- Semántica de objetos, dentro de la cual se hace una adaptación de
 - Iteradores
 - Generadores
- Iteradores
- Generadores
- Tipos básicos de la librería estándar de python, entre los que se encuentran:
bool,

type Type sirve para defnir clases.

Bibliografía

- [1] [Versiones de Javascript, Jhon Reisig](#)