Servicios y Aplicaciones en Redes de Ordenadores (2014-15)

Grado en Ingeniería de Sistemas de Telecomunicación (URJC)

Jesús M. González Barahona, Gregorio Robles Martínez

http://cursosweb.github.io
GSyC, Universidad Rey Juan Carlos

21 de enero de 2015

Presentación de la asignatura

Datos, datos, datos

- Profesores:
  1. Jesús M. González Barahona (jgb @ gsyc.urjc.es)
  2. Gregorio Robles (grex @ gsyc.urjc.es)
- Grupo de Sistemas y Comunicaciones (GSyC)
- Despachos: 003 Biblioteca y 110 Departamental III
- Horario: M (9:00-11:00) y J (9:00-11:00)
- Tutoría: X de 16:00 a 19:00 (en los propios Laboratorios)
- Aula 314, Aulario III (esporádicamente)
- Laboratorio 209 Laboratorios III (habitualmente)

Campus virtual: `http://campusonline.urjc.es/`
GitHub: `http://cursosweb.github.io/`

¿De qué va todo esto?

Entendiendo cómo funciona la web

En concreto...

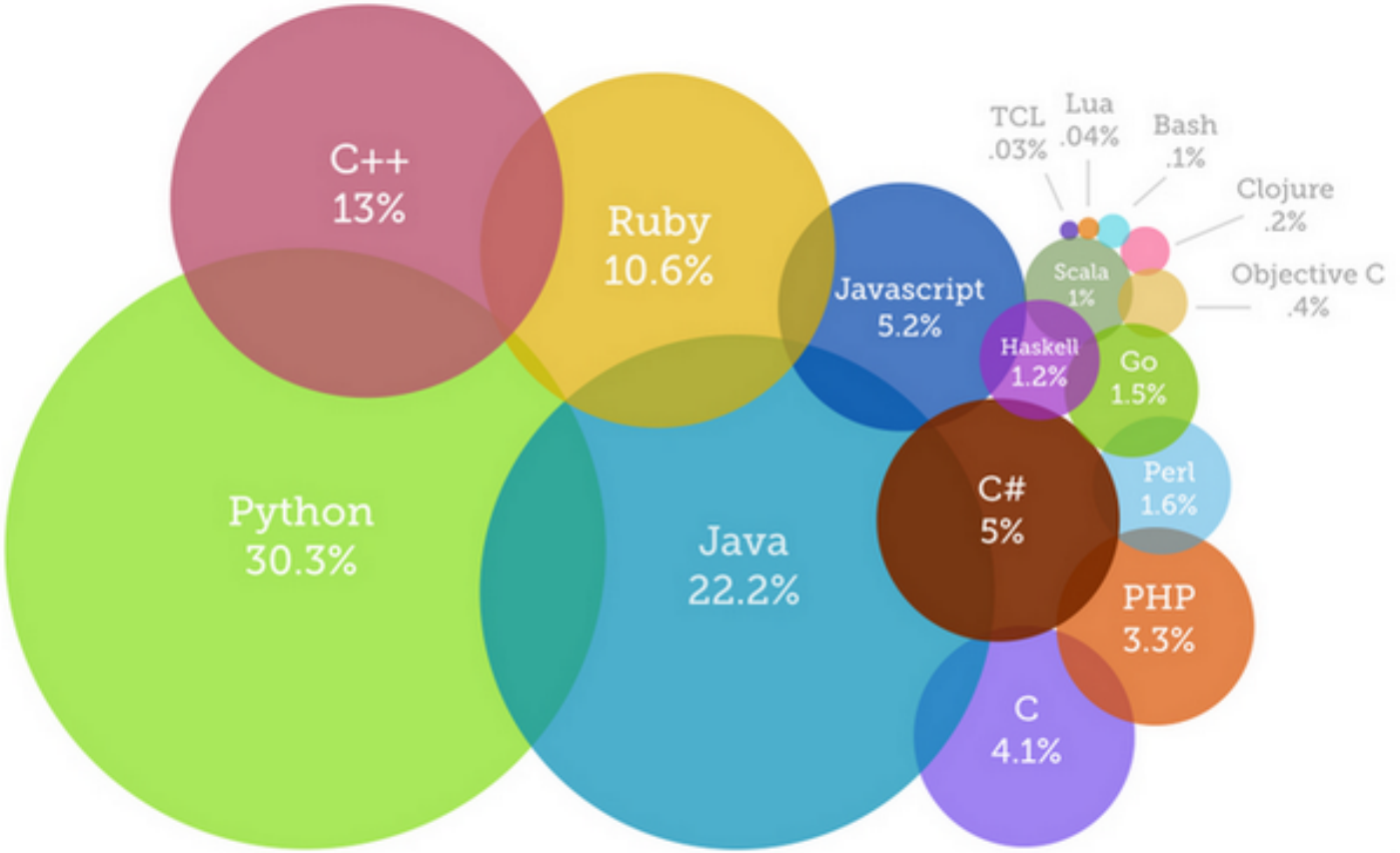
- Cómo se construyen los sistemas reales que se usan en Inet
- Qué tecnologías se están usando
- Qué esquemas de seguridad hay
- Cómo encajan las piezas
- En la medida de lo posible, “manos en la masa”

Ejemplos

- ¿Qué es una aplicación web?
- ¿Cómo construir un servicio REST?
- Acaba con la magia de los servicios web
- ¿Qué es el web 2.0?
- ¿Cómo se hace un mashup?
- ¿Cómo puedo interaccionar con los servicios más populares?
- ¿Qué es HTML5?

Fundamentos filosofales de la asignatura

Lenguaje de Programación: Python



Primer Mandamiento:
Amarás Python por encima de (casi) todo.

Plataforma: Django

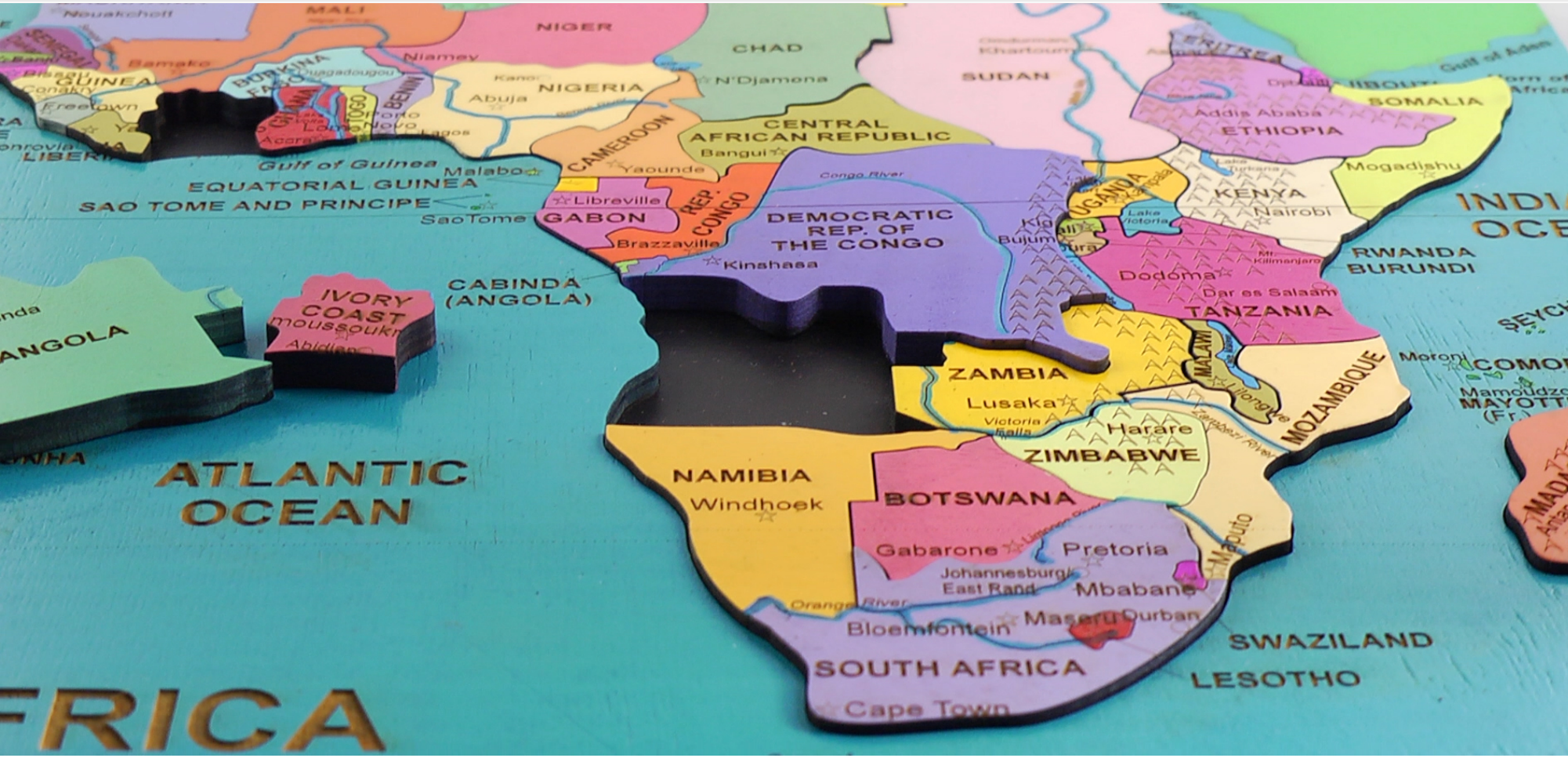
Primera Generación	Segunda Generación	Tercera Generación
HTML	PHP	RAILS
CGI	JSP	**DJANGO**
	PERL	SYMFONY

Segundo Mandamiento:
No tomarás el nombre de Django en vano.

Metodología

- Objetivo principal: conceptos básicos de construcción de sitios web modernos
- Clases de teoría y de prácticas, pero...
- Teoría en prácticas, prácticas en teoría
- Uso de resolución de problemas para aprender
- Fundamentalmente, entender lo fundamental

Fundamentos filosofales de la asignatura



Aprender no puede ser aburrido

Las Clases

- Empezamos en punto
- 10 minutos con un tema motivacional
  - Gadgets tecnológicos
  - Aplicaciones
  - Cuestiones interesantes
  - ...
- Generalmente, explicación de los conceptos más importantes y luego realización de ejercicios
- No hay descanso
- Ejercicios para hacer fuera de clase (y entregar)

Fundamentos filosofales de la asignatura

Evaluación

- Teoría (obligatorio): nota de 0 a 4.
- Práctica final (obligatorio): nota de 0 a 2.
- Opciones y mejoras práctica final: nota de 0 a 3
- Prácticas incrementales: 0 a 1
- Ejercicios en foro/GitHub: nota de 0 a 2
- Nota final: Suma de notas, moderada por la interpretación del profesor
- Mínimo para aprobar:
  - Aprobado en teoría (2) y práctica final (1), y
  - 5 puntos de nota final en total

Evaluación (2)

- Evaluación teoría: prueba escrita
- Evaluación prácticas incrementales (evaluación continua):
  - entre 0 y 1 (sobre todo las extensiones)
  - es muy recomendable hacerlas
- Evaluación práctica final
  - posibilidad de examen presencial para práctica final
  - ¡tiene que funcionar en el laboratorio!
  - enunciado mínimo obligatorio supone 1, se llega a 2 sólo con calidad y cuidado en los detalles
- Opciones y mejoras práctica final:
  - permiten subir la nota mucho
- Evaluación ejercicios (evaluación continua):
  - preguntas y ejercicios en foro/GitHub
- Evaluación extraordinaria:
  - prueba escrita (si no se aprobó la ordinaria)
  - nueva práctica final (si no se aprobó la ordinaria)

Prácticas finales

Ejemplos del pasado:

- Servicio de apoyo a la docencia
- Sitio de intercambio de fotos
- Aplicación web de autoevaluación docente
- Agregador de blogs (canales RSS)
- Agregador de microblogs (Identi.ca, Twitter)

Ejemplos de prácticas finales de otros años

- Fernando Yustas:
  `https://www.youtube.com/watch?v=TUUMVEaBzeg`
- Miguel Ariza: `https://www.youtube.com/watch?v=fVBx9cGPjWs`

(puedes buscar en YouTube por muchos más ejemplos)

¡Ánimo!

Aquí se enseñan cómo son las cosas
que se usan en el mundo real

Las buenas noticias son...
que no son tan difíciles

Cookies HTTP

Cookies

- Forma de mantener estado en un protocolo sin estado (HTTP)
- Forma de almacenar datos en el lado del cliente
- Dos versiones principales:
  - Versión 0 (Netscape)
  - Version 1 (RFC 2109 / RFC 2965)

Cabeceras HTTP para cookies (version 0)

- Set-Cookie: De servidor a navegador

```
Set-Cookie: statusmessages="deleted"; Path=/;
  Expires=Wed, 31-Dec-97 23:59:59 GMT
```

  - Nombre de la cookie: statusmessages
  - Valor: "deleted"
  - Path: / (todo el sitio del que se recibió)
  - Expira: 31-Dec-97 23:59:59 GMT

- Cookie: De navegador a servidor

```
Cookie: statusmessages="deleted"
```

  - Nombre de la cookie: statusmessages
  - Valor: "deleted"

RFC 2965 (version 1) tiene: "Cookie", "Cookie2" y "Set-Cookie2"

Estructura de Set-Cookie

- Nombre y valor (obligatorios)
  - Uso normal: "nombre=valor"
  - También valor nulo: "nombre="
- Fecha de expiración
  - "Expires=fecha"
  - Si no tiene, cookie no persistente (sólo en memoria)
- Path: camino para el que es válida
  - "Path=/camino"
  - Prefijo de caminos válidos
- Domain: dominio para el que es válida
  - El servidor ha de estar en el dominio indicado
  - Si no se indica, servidor que sirve la cookie
- Seguridad: se necesita SSL para enviar la cookie
  - Campo "Secure"
- Campos separados por ";"

Estructura de Cookie

- Lista de pares nombre - valor
- Cada par corresponde a un "Set-Cookie"
- Se envían las cookies válidas para el dominio y el path de la petición HTTP
- Si no se especificó dominio en "Set-Cookie", el del servidor
- Si no se especificó camino en "Set-Cookie", todo

```
Cookie: user=jgb; last=5; edited=False
```

Límites para las cookies

- Originalmente: 20 cookies del mismo dominio
- La mayoría de los navegadores: 30 o 50 cookies del mismo dominio
- Cada cookie: como mucho 4 Kbytes

Gestión de sesión en HTTP

- Mediante cookies: normalmente, identificador de sesión en la cookie

```
Set-Cookie: session=ab34cd-34fd3a-ef2365
```

- Reescritura de urls: se añade identificador a la url

```
http://sitio.com/path;session=ab34cd-34fd3a-ef2365
```

- Campos escondidos en formularios HTML

```
<form method="post" action="http://sitio.com/path">
  <input type="hidden" name="session" value="ab34cd-34fd3a
  ...
  <input type="submit">
</form>
```

Referencias

- Persistent Client State HTTP Cookies
  (especificación original de Netscape)
  `http://curl.haxx.se/rfc/cookie_spec.html`
- RFC 2109: HTTP State Management Mechanism
  `http://tools.ietf.org/html/rfc2109`
- RFC 2965: HTTP State Management Mechanism
  `http://tools.ietf.org/html/rfc2965`

Prácticas: Introducción a Python

Características

Python es un lenguaje:

- de alto nivel
- interpretado
- orientado a objetos (todo son objetos)
- dinámicamente tipado (frente a estáticamente tipado)
- fuertemente tipado (frente a débilmente tipado)
- sensible a mayúsculas/minúsculas

Un pequeño ejemplo

- Podemos usar Python como calculadora (ojo: $3/2 = 1$)
- Sensible a mayúsculas / minúsculas
- Comentarios: #

```
#!/usr/bin/python
print "hola mundo"    # esto es un comentario
euros=415
pesetas=euros*166.386
print str(euros) + " euros son "+ str(pesetas) + " pesetas"
```

Sangrado y separadores de sentencias

- ¡En Python NO hay llaves ni `begin-end` para encerrar bloques de código! Un mayor nivel de sangrado indica que comienza un bloque, y un menor nivel indica que termina un bloque.
- Ejemplo:

```
#!/usr/bin/python
def a_centigrado(x):
    """Convierte grados farenheit en grados centigrados"""
    return (x-32)*(5/9.0)


def a_farenheit(x):
    """Convierte grados centígrados en grados farenheit"""
    return (x*1.8)+32
```

Condicional

Sentencia `if`:

```
#!/usr/bin/python
x = 3
if x :
     print 'verdadero'
else:
     print 'falso'
```

Nótese como el caracter `:` introduce cada bloque de sentencias.

Cadenas

- No existe tipo `char`
- Comilla simple o doble
  `print "hola"` o `print 'hola'`
  `print 'me dijo "hola"'`
  más legible que `print 'me dijo \'hola\''`
- Puede haber caracteres especiales
  `print "hola\nque tal"`
- El operador + concatena cadenas, y el * las repite un número entero de veces
- Se puede acceder a los caracteres de cadenas mediante índices y rodajas como en las listas. Pero las cadenas son inmutables

Listas

- Tipo de datos predefinido en Python, va mucho más allá de los arrays
- Es un conjunto **indexado** de elementos, no necesariamente homogéneos
- Sintaxis:Identificador de lista, mas índice entre corchetes
- Cada elemento se separa del anterior por un caracter ,

```
a=['rojo','amarillo']
a.append('verde')
print a
print a[2]
print len(a)

b=['uno',2, 3.0]
```

- El primer elemento tiene índice 0.
- Un índice negativo accede a los elementos empezando por el final de la lista. El último elemento tiene índice -1.
- Pueden referirse **rodajas** (*slices*) de listas escribiendo dos índices entre el caracter :
- La rodaja va desde el **primero, incluido**, al **último, excluido**.
- Si no aparece el primero, se entiende que empieza en el primer elemento (0)
- Si no aparece el segundo, se entiende que termina en el último elemento (incluido).

```
#!/usr/bin/python
a=[0,1,2,3,4]
print a      # [0, 1, 2, 3, 4]
print a[1]   # 1
print a[0:2] # [0,1]
print a[3:]  # [3,4]
print a[-1]  # 4
print a[:-1] # [0, 1, 2, 3]
print a[:-2] # [0, 1, 2]
```

La misma sintaxis se aplica a las cadenas

```
a="niño"
print a[-1]
```

Bucles

Sentencia `for`:

```
>>> a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
>>> a = ['had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 had
1 a
2 little
3 lamb
```

Diccionarios

- Es un conjunto **desordenado** de elementos
- Cada elemento del diccionario es un par clave-valor.
- Se pueden obtener valores a partir de la clave, pero no al revés.
- Longitud variable
- Elementos heterogéneos
- Hace las veces de los *registros* en otros lenguajes
- Atención: Se declaran con `{}`, se refieren con `[]`

- Asignar valor a una clave existente reemplaza el antiguo
- Una clave de tipo cadena es sensible a mayúsculas/minúsculas
- Pueden añadirse entradas nuevas al diccionario
- Los diccionarios se mantienen desordenados
- Los valores de un diccionario pueden ser de cualquier tipo
- Las claves pueden ser enteros, cadenas y algún otro tipo
- Pueden borrarse un elemento del diccionario con `del`
- Pueden borrarse todos los elementos del diccionario con `clear()`

Más sobre diccionarios

```
#!/usr/bin/python
pais={'de': 'Alemania', 'fr': 'Francia', 'es': 'España'}
print pais; print pais["fr"]

extension={}
extension['py']='python'
extension['txt']='texto plano'
extension['ps']='PostScript'

for x in pais.keys():
   print x, pais[x]

del pais['fr']
print len(pais)
print pais.has_key('es')
pais.clear()
```

Importar módulos

- `import nombre-módulo`
  permite acceder a los símbolos del módulo con la sintaxis `nombre-módulo.X`
- `from nombre-módulo import a, b, c`
  incorpora los símbolos a, b, c al espacio de nombres, siendo accesibles directamente (sin cualificarlos con el nombre del módulo)
- `from nombre-módulo import *`
  incorpora los símbolos del módulo al espacio de nombres, siendo accesibles directamente (sin cualificarlos con el nombre del módulo).

Ficheros

- `open(nombre_fichero,modo)` devuelve un objeto fichero
  modo:
  - `w`: Escritura. Destruye contenido anterior
  - `r`: Lectura. Modo por defecto
  - `r+`: Lectura y Escritura
  - `a`: Append
- `write(cadena)` escribe la cadena en el fichero
- `read()` devuelve el contenido del fichero
- `readlines()` devuelve una lista con cada línea del fichero
- `close()` cierra el fichero

```
#!/usr/bin/python
fich=open('/tmp/prueba','w')
fich.write("lunes\n")
fich.close()

fich=open('/tmp/prueba','a')
fich.write("martes\n")
fich.close()

fich=open('/etc/hosts','r')
maquinas=fich.read()
fich.close()

fich=open('/etc/hosts','r')
maquinas=fich.readlines()
fich.close()

for maquina in maquinas:
    print maquina,
```

Definición de variables

Python es

- fuertemente tipado (frente a débilmente tipado)
- sensible a mayúsculas/minúsculas

En Python la declaración de variables es implícita
(no hay declaración explícita)

- Las variables "nacen" cuando se les asigna un valor
- Las variables "desaparecen" cuando se sale de su ámbito

Sangrado y separadores de sentencias (II)

- Las sentencias se terminan al acabarse la línea (salvo casos especiales donde la sentencia queda "abierta": en mitad de expresiones entre paréntesis, corchetes o llaves).
- El caracter \ se utiliza para extender una sentencia más allá de una linea, en los casos en que no queda "abierta".
- El caracter : se utiliza como separador en sentencias compuestas. Ej.: para separar la definición de una función de su código.
- El caracter ; se utiliza como separador de sentencias escritas en la misma línea.

Tuplas

Tipo predefinido de Python para una lista inmutable.
Se define de la misma manera, pero con los elementos entre paréntesis.
Las tuplas no tienen métodos: no se pueden añadir elementos, ni cambiarlos, ni buscar con `index()`.
Sí puede comprobarse la existencia con el operador `in`.

```
>>> t = ("a", "b", "mpilgrim", "z", "example")
>>> t[0]
'a'
>>> 'a' in t
1
>>> t[0] =  "b"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

Utilidad de las tuplas:

- Son más rápidas que las listas
- Pueden ser una clave de un diccionario (no así las listas)
- Se usan en el formateo de cadenas

`tuple(li)` devuelve una tupla con los elementos de la lista `li`
`list(t)` devuelve una lista con los elementos de la tupla `t`

Funciones predefinidas

- `abs()` valor absoluto
- `float()` convierte a float
- `int()` convierte a int
- `str()` convierte a string
- `round()` redondea
- `raw_input()` acepta un valor desde teclado

Operadores

En orden de precedencia decreciente:

```
+x, -x, ~x    Unary operators
x ** y    Power
x * y, x / y, x % y    Multiplication, division, modulo
x + y, x - y    Addition, subtraction
x << y, x >> y    Bit shifting
x & y    Bitwise and
x | y    Bitwise or
x < y, x <= y, x > y, x >= y, x == y, x != y,
x <> y, x is y, x is not y, x in s, x not in s
                        Comparison, identity,
                        sequence membership tests
not x    Logical negation
x and y    Logical and
lambda args: expr            Anonymous function
```

- La declaración implícita de variables como en perl puede provocar resultados desastrosos

```
#!/usr/bin/perl
$sum_elementos= 3 + 4 + 17;
$media=suma_elementos / 3;    # deletreamos mal la variable
print $media;   # y provocamos resultado incorrecto
```

- Pero Python no permite referenciar variables a las que nunca se ha asignado un valor.

```
#!/usr/bin/python
sum_elementos= 3 + 4 + 17
media=suma_elementos / 3    # deletreamos mal la variable
print media;   # y el compilador nos avisa con un error
```

Operaciones sobre cadenas

`join()` devuelve una cadena que engloba a todos los elementos de la lista.
`split()` devuelve una lista dividiendo una cadena
`upper()` devuelve la cadena en mayúsculas
`lower()` devuelve la cadena en minúsculas
Estas funciones de biblioteca, como todas,
podemos encontrarlas en la *python library reference*
(disponible en el web en muchos formatos)

Operaciones sobre diccionarios

- `len(d)` devuelve el número de elementos de `d`
- `d.has_key(k)` devuelve 1 si existe la clave `k` en `d`, 0 en caso contrario
- `k in d` equivale a: `d.has_key(k)`
- `d.items()` devuelve la lista de elementos de `d`
- `d.keys()` devuelve la lista de claves de `d`

Más sobre cadenas

```
#!/usr/bin/python
import string
a="más vale pájaro en mano"
print string.split(a)
print string.upper(a)

b="que,cocodrilo,en,tobillo"
print string.split(b,',')

c=['rojo','amarillo','verde']
print string.join(c)
```

```
#!/usr/bin/python
x = int(raw_input("Please enter an integer: "))
if x < 0:
     x = 0
     print 'Negative changed to zero'
elif x == 0:
     print 'Zero'
elif x == 1:
     print 'Single'
else:
     print 'More'
```

No existe `switch/case`

Ambito de las variables

- Las variable declaradas fuera de una función son globales

```
#!/usr/bin/python
c=5
def f(x):
    return x+c

print f(3)    # 8
```

- Las variable declaradas dentro de una función son locales

```
#!/usr/bin/python
def f(x):
    c=5
    return x+c
print f(3)
print c    # ERROR: c es de ambito local
```

- Dentro de una función se puede ver una variable global pero no modificar

- A menos que se use la sentencia `global`

```
#!/usr/bin/python
c=5
def f(x):
    global c   #permite modificar una variable global
    c=c-1
    return x+c

print f(3)    #7
print c       #4
```

- Un poco más complicado:

```
#!/usr/bin/python
c=5
def f(x):
    c=4   #ahora c es variable local
    return x+c

print f(3)   # 7
print c      # 5
```

Más sobre listas

- `append()` añade un elemento al final de la lista
- `insert()` inserta un elemento en la posición indicada

```
>>> li
['a', 'b', 'mpilgrim', 'z', 'example']
>>> li.append("new")
>>> li
['a', 'b', 'mpilgrim', 'z', 'example', 'new']
>>> li.insert(2, "new")
>>> li
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new']
```

- `index()` busca en la lista un elemento y devuelve el índice de la primera aparición del elemento en la lista. Si no aparece se eleva una excepción.
- El operador `in` devuelve 1 si un elemento aparece en la lista, y 0 en caso contrario.

```
>>> li
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new']
>>> li.index("example")
5
>>> li.index("new")
2
>>> li.index("c")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
ValueError: list.index(x): x not in list
>>> "c" in li
0
```

- `remove()` elimina la primera aparición de un elemento en la lista. Si no aparece, eleva una excepción.
- `pop()` devuelve el último elemento de la lista, y lo elimina. (Pila)
- `pop(0)` devuelve el primer elemento de la lista, y lo elimina. (Cola)

```
 >>> li
['a', 'b', 'new', 'mpilgrim', 'z', 'new', 'two', 'elements']
>>> li.remove("new")
>>> li
['a', 'b', 'mpilgrim', 'z', 'new', 'two', 'elements']
>>> li.remove("c")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
ValueError: list.remove(x): x not in list
>>> li.pop()
'elements'
>>> li
['a', 'b', 'mpilgrim', 'z', 'new', 'two']
```

- El operador + concatena dos listas, devolviendo una nueva lista
- El operador * concatena repetitivamente una lista a sí misma

```
>>> li = ['a', 'b', 'mpilgrim']
>>> li = li + ['example', 'new']
>>> li
['a', 'b', 'mpilgrim', 'example', 'new']
>>> li += ['two']
>>> li
['a', 'b', 'mpilgrim', 'example', 'new', 'two']
>>> li = [1, 2] * 3
>>> li
[1, 2, 1, 2, 1, 2]
```

- `sort()` ordena una lista. Puede recibir opcionalmente un argumento especificando una función de comparación, lo que enlentece notable su funcionamiento
- `reverse()` invierte las posiciones de los elementos en una lista.

Ninguno de estos métodos devuelve nada, simplemente alteran la lista sobre la que se aplican.

```
>>> li = ['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new', 'two', 'eleme
>>> li.sort()
>>> li
['a', 'b', 'elements', 'example', 'mpilgrim', 'new', 'new', 'two', 'z']
>>> li.reverse()
>>> li
['z', 'two', 'new', 'new', 'mpilgrim', 'example', 'elements', 'b', 'a']
```

Un programa en Python

```
def buildConnectionString(params):
    """Build a connection string from a dictionary of parameters.

    Returns string."""
    return ";".join(["%s=%s" % (k, v) for k, v in params.items()])

if __name__ == "__main__":
    myParams = {"server":"mpilgrim", \
                "database":"master", \
                "uid":"sa", \
                "pwd":"secret" \
                }
    print buildConnectionString(myParams)
```

Ejecución:

```
xterm$ python odbchelper.py
server=mpilgrim;uid=sa;database=master;pwd=secret
```

El atributo __name__ de un módulo

Los módulos son objetos, con ciertos atributos predefinidos.
El atributo `__name__`:

- si el módulo es importado (con `import`), contiene el nombre del fichero, sin trayecto ni extensión
- si el módulo es un programa que se ejecuta sólo, contiene el valor `__main__`

Puede escribirse ejecución condicionada a cómo se use el módulo:

```
if __name__ == "__main__":
    ...
```

Asignaciones múltiples y rangos

- Pueden hacerse también tuplas de variables:

```
>>> v = ('a', 'b', 'e')
>>> (x, y, z) = v
>>> x
'a'
```

- La función `range()` permite generar listas al vuelo:

```
>>> range(7)
[0, 1, 2, 3, 4, 5, 6]
>>> (MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
... FRIDAY, SATURDAY, SUNDAY) = range(7)
>>> MONDAY
0
>>> SUNDAY
6
```

Mapeo de listas

- Se puede mapear una lista en otra, aplicando una función a cada elemento de la lista:

```
>>> li = [1, 9, 8, 4]
>>> [elem*2 for elem in li]
[2, 18, 16, 8]
>>> li
[1, 9, 8, 4]
>>> li = [elem*2 for elem in li]
>>> li
[2, 18, 16, 8]
```

Filtrado de listas

- Sintaxis:

```
[expresión-mapeo for elemento in lista-orig if condición-filtrado]
```

- Ejemplos:

```
>>> li = ["a", "mpilgrim", "foo", "b", "c", "b", "d", "d"]
>>> [elem for elem in li if len(elem) > 1]        1
['mpilgrim', 'foo']
```

Control de flujo

Sentencia `while`:

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print b,
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Nótese el efecto de un caracter `,` al final de un `print`
Nótese otro modelo de asignación múltiple

- break sale de un bucle:

```
#!/usr/bin/python
a=10
while a > 0:
    print a,
    a=a-1
```

equivale a

```
#!/usr/bin/python
a=10
while 1:
    print a,
    if a==1:
        break
    a=a-1
```

Los nulos también tienen nombre

Sentencia nula: `pass`
Valor nulo: `None`

Uso de bibliotecas

- Llamada al shell

```
#!/usr/bin/python
import os
os.system('ls -l')
```

- Argumentos de linea de comandos

```
#!/usr/bin/python
import sys
print sys.argv[1:]
```

Las funciones de biblioteca podemos encontrarlas en la *python library reference* (disponible en el web en muchos formatos)

Excepciones

- Un programa sintácticamente correcto puede dar errores de ejecución

```
#!/usr/bin/python
while 1:
    x=int(raw_input("Introduce un nº"))
    print x
```

- Definimos una acción para determinada excepción

```
#!/usr/bin/python
while 1:
    try:
        x=int(raw_input("Introduce un nº:"))
        print x
    except ValueError:
        print ("Número incorrecto")
```

- Se puede indicar una acción para cualquier excepción pero es *muy* desaconsejable (enmascara otros errores)
- El programador puede levantar excepciones

```
#!/usr/bin/python
try:
    x=int(raw_input("Introduce un n°:"))
    print x
except :     # para cualquier excepción
    print ("Número incorrecto")

raise SystemExit
print "nunca se ejecuta"
```

Objetos en Python

Todo son objetos, en sentido amplio:

- Cualquier objeto puede ser asignado a una variable o pasado como parámetro a una función
- Algunos objetos pueden no tener ni atributos ni métodos
- Algunos objetos pueden no permitir que se herede de ellos

Ejemplos de objetos Python: Strings, listas, funciones, módulos...

Todos los objetos tienen:

- **Identidad**:
  - Nunca cambia.
  - El operador `is` compara la identidad de dos objetos.
  - La función `id()` devuelve una representación de la identidad (actualmente, su dirección de memoria).
- **Tipo**:
  - Nunca cambia.
  - La función `type()` devuelve el tipo de un objeto (que es otro objeto)
- **Valor**:
  - Objetos inmutables: su valor no puede cambiar
  - Objetos mutables: su valor puede cambiar

Contenedores: objetos que contienen referencias a otros objetos (ej.: tuplas, listas, diccionarios).

Cadenas de documentación

- No son obligatorias pero sí muy recomendables (varias herramientas hacen uso de ellas).
- La cadena de documentación de un objeto es su atributo `__doc__`
- En una sola línea para objetos sencillos, en varias para el resto de los casos.
- Entre triples comillas-dobles (incluso si ocupan una línea).
- Si hay varias líneas:
  - La primera línea debe ser una resumen breve del propósito del objeto. Debe empezar con mayúscula y acabar con un punto
  - Una línea en blanco debe separar la primera línea del resto
  - Las siguientes líneas deberían empezar justo debajo de la primera comilla doble de la primera línea

De una sola línea:

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    ...
```

De varias:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)

    """
    if imag == 0.0 and real == 0.0: return complex_zero
```

Documentando el código (tipo Javadoc)

- Permite documentar el código -generalmente las funciones- dentro del propio código
- Genera la documentación del código en formatos legibles y navegables (HTML, PDF...)
- Se basa en un lenguaje de marcado simple
- PERO... hay que mantener la documentación al día cuando se cambia el código

Ejemplo

```
def interseccion(m, b):
  """
  Devuelve la interseccion de la curva M{y=m*x+b} con el eje X.
  Se trata del punto en el que la curva cruza el eje X (M{y=0}).

  @type  m: numero
  @param m: La pendiente de la curva
  @type  b: numero
  @param b: La intersección con el eje Y

  @rtype:   numero
  @return:  la interseccion con el eje X de la curva M{y=m*x+b}.
  """
  return -b/m
```

Orientación a objetos

- Modelo de clases clásico
- En cada módulo (fichero) una o varias clases
- Todo en Python es un objeto (incluso una función)
- Los métodos de una clase se declaran como funciones
- Método __init__: inicialización en el momento de la instanciación
- Argumento "self" pasado a cada método: referencia al propio objeto
- Todos los métodos incluyen un argumento "self"

Orientación a objetos (ejemplo)

```
class aClass (parentClass)
   """Description of the class"""

   def method1 (self)
      """Description of method1"""
      codeForMethod1

   def __init__ (self, arg)
      """Description for initialization"""
      codeForInit
```

Orientación a objetos (ejemplo 2)

```
# Instantiation
anObject = aClass (arg)
# Calling method1
anObject.method1 ()
# Calling a parent method
anObject.methodParent ()
```

Referencias

- `http://www.python.org/doc`
  Documentación en línea de Python (incluyendo un Tutorial, los manuales de referencia, HOWTOS, etc. Usa la versión para Python 2.x
- `http://www.diveintopython.org/`
  "Dive into Python", por Mark Pilgrim. Libro para aprender Python, orientado a quien ya sabe programa con lenguajes orientados a objetos.
- `http://wiki.python.org/moin/BeginnersGuide/Programmers`
  Otros textos sobre Python, de interés especialmente para quien ya sabe programar en otros lenguajes.
- `http://python.net/~goodger/projects/pycon/2007/idiomatic/`
  "Code Like a Pythonista: Idiomatic Python", por David Goodger

Referencias (2)

- `http://en.wikibooks.org/wiki/Python_Programming`
  "Python Programming", Wikibook sobre programación en Python.
- `http://en.wikipedia.org/wiki/Python_(programming_language)`
  Python en la Wikipedia