

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned on the left side of the slide.

Lenguaje Python

Funciones y Módulos

Temario

- Repaso
- Definición de funciones
- Alcances
- Módulos y paquetes

Lenguaje Python

Repaso

Repaso: ¿Qué vimos hasta ahora?

- Conceptos Iniciales
 - Software Libre
 - Características básicas de Python
- Tipos de datos disponibles
 - Inmutables vs. mutables
 - Operaciones
- Sintaxis del lenguaje
 - Uso de variables
 - Estructuras de control



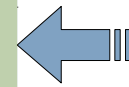
Lenguaje Python

Funciones en Python

Funciones en Python

Forma General:

```
def nombre_funcion(parametros):  
    sentencias  
    return <expresion>
```



El cuerpo de la función **debe** estar indentado!

Ejemplo:

```
def cuadrado(x):  
    return x ** 2
```

¿Cómo lo usamos?

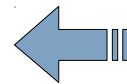
```
print (cuadrado(3))  
a=2+cuadrado(9)
```

Funciones en Python

Ejemplo sin sentencia return:

```
def imprimo_saludo(nombre):
```

```
    "Imprime un saludo a nombre"
```



Docstring

```
    print ("Hola" + nombre + "!!")
```

...

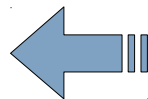
```
>>>imprimo_saludo("Vivi")
```

```
Hola Vivi!!
```

```
>>>print (imprimo_saludo("Vivi" ))
```

```
Hola Vivi!!
```

```
None
```



Las funciones SIEMPRE retornan un valor.... aún
valor "**None**"

Funciones en Python - Parámetros

```
def cambio(x,y):
```

```
    temp=x
```

```
    x=y
```

```
    y=temp
```

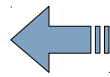
```
...
```

```
a=2
```

```
b=4
```

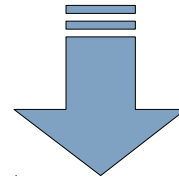
```
cambio(a,b)
```

```
print (a, b)
```

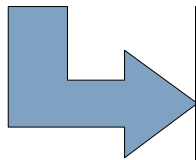


¿Cómo están pasadas?

Argumentos en Python
siempre son una copia de la
ref. al objeto pasado



Pensar: En los distintos tipos de
argumentos...



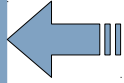
```
>>> a=2
>>> b=4
>>> cambio(a,b)
>>> print(a,b)
2 4
```

Ejemplo: strings, **números**, tuplas

Funciones en Python - Parámetros

```
def cambio(x):
```

```
    x[0]="cero"
```



En este ejemplo: Tipo
Lista!

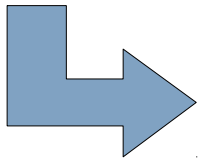
```
...
```

```
a=[1,2,3]
```

```
cambio(a)
```

```
¿print (a)?
```

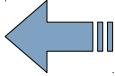
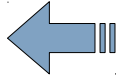
¿Qué pasó?

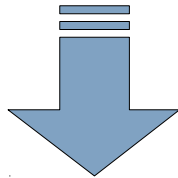


```
a=["cero",2,3]
```

Funciones en Python - Parámetros

```
def cambio(x):  
    y= x[:]  
    y[0]=0
```

 Si x es una lista
 y es una copia de x (“slicing”)



...

```
a=[1,2,3]  
cambio(a)  
print (a[0])
```

En este caso: el parámetro real NO se altera

También podría:

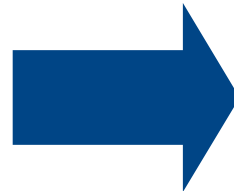
```
>>>cambio(a[:])
```

Mandamos una copia del parámetro real

Funciones en Python - Parámetros

```
def modifico1(x):  
    x=[1,2,3]  
    print(x)
```

```
a=["a", "b"]  
modifico1(a)  
print(a)
```



```
>>>  
[1, 2, 3]  
['a', 'b']  
>>> |
```

Resumiendo ...

```
def modifico2(x):  
    x[0]=[1,2,3]  
    print(x)
```

```
modifico2(a)  
print(a)
```



```
[[1, 2, 3], 'b']  
[[1, 2, 3], 'b']  
>>> |
```

¿Qué pasó?

Funciones en Python - Parámetros

Resumiendo ...

Pensar:

```
x=[0,0,0]
def cambio(y):
    y=[1,2,3]
```

```
def cambio1(y):
    y[:]=[1,2,3]
```

¿Qué diferencia hay?

```
.....
cambio(x)

.....
cambio1(x)

.....
```

Funciones en Python - Parámetros

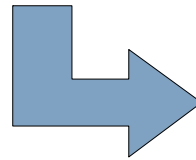
Pensar:

```
def cambio(y):  
    y[0]=100000
```

```
.....  
x=(0,0,0)  
cambio(x)  
.....
```

¿Y si
trabajamos
con una tupla?

¿Qué piensan que pasa
acá?



!!!ERROR!!!

Funciones en Python

¿Y si tenemos que retornar más de un valor?

```
1 def proceso_numeros(lista):
2     min=9999999
3     max=-9999999
4     sum=0
5     for x in lista:
6         if x>max:
7             max=x
8         if x<min:
9             min=x
10        sum+=x
11    return (min, max, 0 if len(lista)==0 else sum/len(lista))
12
13 lista=[11,2,3,4]
14 print(proceso_numeros(lista))
```

Suponemos que queremos definir una función que retorne el máx, el mín y el promedio de varios números...

Se retorna una **tupla** con los valores

Funciones en Python - Parámetros

Parámetros por defecto

```
def imprimo_saludo(nombre, texto="Hola "):  
    "Imprime un saludo a nombre"  
    print (texto + nombre + "!!")
```

```
>>>imprimo_saludo("Vivi")  
Hola Vivi!!
```

Argumentos por defecto
siempre al final

```
>>>imprimo_saludo("Vivi", "Chau " )  
Chau Vivi!!
```

Funciones en Python - Parámetros

Parámetros por defecto

```
def imprimo_saludo(nombre="Vivi", texto="Hola "):  
    "Imprime un saludo a nombre"  
    print (texto + nombre + "!!")
```

```
>>> imprimo_saludo(texto="Chau ")
```

Chau Vivi!!

Puedo invocarlos en otro orden pero
nombrando el parámetro

```
>>> imprimo_saludo(texto="Chau ", nombre="Clau")
```

Chau Clau!!

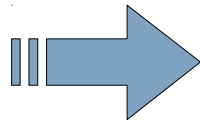
Funciones en Python - Parámetros

Parámetros por defecto

Importante: Los valores por defecto se evalúan **UNA única vez** en la **definición** de la función.

```
def f(a, L=[]):  
    L.append(a)  
    return L
```

```
print (f(1))  
print (f(2))  
print (f(3))
```



Imprime:
[1]
[1, 2]
[1, 2, 3]

Funciones en Python

Argumentos variables

Es posible definir funciones con un número variable de argumentos.

Ejemplo:

```
>>>
>>> def sumo_parametros(*param):
    '''Esta funcion recibe un numero variable de valores numericos
    y retorna la suma de todos ellos'''
    return sum(param)

>>> sumo_parametros(1,2,3,4,5,6,7,8,9)
45
>>> |
```

Representa a la lista
variable de argumentos
representada como una
tupla

El docstring ocupa
más de una línea

Funciones en Python

Variables Locales y Globales

```
x=12
a=13
def miFuncion(a):
    x=9
    a=10
```

- ➡ Variables locales enmascaran las globales
- ➡ Acceso a las globales mediante **global**

```
def miFuncion(a):
    global x
    x=9
    a=10
```

Aunque No es una buena práctica de programación, utilizar variables globales

Funciones en Python - Parámetros

Volviendo a analizar los parámetros por defecto

Dijimos que... Los valores por defecto se evalúan **UNA única vez** en la **definición** de la función.

Entonces.... ¿qué imprime este código?

```
i = 5  
  
def f(arg=i):  
    print(arg)  
  
i = 6  
f()
```

Funciones en Python

Atributos de las funciones

funcion.__doc__: es una cadena que contiene un texto que aparece en la **primer línea** de la definición de la función (**Docstring**)

funcion.__name__: es una cadena con el nombre la función

funcion.__defaults__: es una tupla con los valores por defecto de los parámetros opcionales

Ejemplo:

```
def imprimo_saludo(nombre, texto="Hola "):  
    "Imprime un saludo a nombre"  
    print(texto + nombre + "!!")
```

```
print(imprimo_saludo.__doc__)  
print(imprimo_saludo.__defaults__)  
print(imprimo_saludo.__name__)
```

Funciones en Python

```
1 def proceso_numeros(lista):
2     def calculo_promedio(lista):
3         sum=0
4         for x in lista:
5             sum+=x
6         return (0 if len(lista)==0 else sum/len(lista))
7
8     min=9999999
9     max=-9999999
10
11     for x in lista:
12         if x>max:
13             max=x
14         if x<min:
15             min=x
16
17     return (min, max, calculo_promedio(lista))
18
19 lista=[1,2,3,4]
20 print(proceso_numeros(lista))
21
```

- Debe estar definida **antes** de usarse
- Por legibilidad, siempre al principio.
- Es **local** a proceso_numeros()

Funciones anidadas

Funciones en Python: Expresiones Lambda

¿Qué son las expresiones lambda?

- Funciones sencillas
- Anónimas

Forma general:

lambda parametros: expresion

↓
Es equivalente a

```
def nombre funcion(parametros):  
    expresion
```

No está
presente

Ejemplos:

```
def prod(a,b):  
    return a*b
```

⇒ lambda a,b: a*b

```
def prod1(a,b=1):  
    return a,b
```

⇒ lambda a,b=1: a*b

Funciones en Python: Expresiones Lambda

Un ejemplo

Usando expresiones
lambda

```
lista = [lambda x: x * 2, lambda x: x * 3]
```

```
param = 4
```

```
for accion in lista:
```

```
    print(accion(param))
```

¿Qué imprime?

Funciones en Python: Expresiones Lambda

Otro ejemplo

Usando expresiones
lambda

```
>>> def make_incrementor (n): return lambda x: x + n
```

```
>>>
```

```
>>> f = make_incrementor(2)
```

```
>>> g = make_incrementor(6)
```

```
>>>
```

```
>>> print (f(42), g(42))
```

```
44 48
```

```
>>> print (make_incrementor(22)(33))
```

```
55
```

Funciones en Python: Expresiones Lambda

Funciones map y filter

➤ map

```
>>> numeros = [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100]
```

```
>>> def doble(n):  
    return n*2;
```

```
list(map(doble, numeros))
```

```
[144, 202, 216, 216, 222, 88, 64, 238, 222, 228, 216, 200]
```

Equivalente a:

```
>>> numeros = [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100]
```

```
>>> list(map(lambda n: 2*n, numeros))
```

```
[144, 202, 216, 216, 222, 88, 64, 238, 222, 228, 216, 200]
```

Usando expresiones
lambda

IMPORTANTE: Diferencia
entre Python 2 y 3: map y
filter en Python 2 devuelven
listas pero en Python 3
devuelven un iterador

Funciones en Python: Expresiones Lambda

Funciones map y filter

+ filter

Usando expresiones
lambda

```
>>> numeros = [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100]
```

```
>>> def par(n):  
    return n%2==0;
```

```
list(filter(par, numeros))  
[72, 108, 108, 44, 32, 114, 108, 100]
```

IMPORTANTE: Diferencia entre Python 2 y 3: map y filter en Python 2 devuelven listas pero en Python 3 devuelven un iterador

Equivalente a:

```
>>> numeros = [72, 101, 108, 108, 111, 44, 32, 119, 111, 114, 108, 100]  
>>> list(filter(lambda n: n%2==0, numeros))  
[72, 108, 108, 44, 32, 114, 108, 100]
```



Lenguaje Python

Módulos y Paquetes

Módulos

```
funciones.py x usoModulos.py x
1 def procesoNumeros(arreglo):
2     min=arreglo[0]
3     max=arreglo[0]
4
5
6
7
8
9
10
11
12

funciones.py x usoModulos.py x
1 import funciones
2
3 x=[1,1,1,1,1,1,2]
4
5 print procesoNumeros(x)
6
7
8
```

Sentencia **import**

Un **módulo** es un archivo (con extensión .py) que contiene sentencias y definiciones.

Módulos: sentencia import

Permite acceder a funciones y variables definidas en otro módulo.

Ejemplo:

#fib.py

```
def fibonacci(a, num1=0, num2=1)
    print (num1)
    while num2<a:
        print (num2)
        num2=num1+num2
        num1=num2-num1
```

>>> import fib ➡ Sólo se importa la función fibonacci.

>>> fib.fibonacci(12) Debo invocar la función para ejecutarla

Archivos .py vs. archivos .pyc

Módulos: Espacio de nombres

Relaciona nombres con objetos.

- ➔ Locales
- ➔ Globales
- ➔ `__builtins__`

Formas:

`import miModulo`

En este caso, todos los ítems definidos dentro de “miModulo” serán locales a “miModulo”. Para usarlo debo hacerlo **con notación puntual**

```
>>> import fib
>>> fib.fibonacci(12)
```

Sin la notación puntual (fib.) dará error.

Atributo `__name__`; es el nombre del módulo (archivo.py)

Se importan sólo una vez por sesión del intérprete

Uso de
`reload()`

Cursada 2014

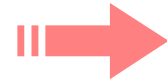
Módulos: Espacio de nombres

Otras Formas:

```
from miModulo import unaFuncion
```

Sólo se importa `unaFuncion` de `miModulo`. (no el nombre del módulo)

```
from miModulo import *
```



No recomendado!!

En este caso, todos los ítems formarán parte del namespace actual y podrían existir conflictos de nombres!!

```
>>> from fib import fibonacci
```

```
>>> fibonacci(12)
```

fibonacci pasó a formar parte del espacio de nombres local!!!

Módulos: search path

Veamos este ejemplo...

```
>>> import pp
```

El módulo pp no existe
(no lo encuentra)

```
Traceback (most recent call last):  
  File "<pyshell#0>", line 1, in <module>  
    import pp  
ImportError: No module named pp  
>>> |
```

Rutas de búsquedas predefinidas ([search path](#))

Directorio actual + otros directorios definidos en la variable de ambiente PYTHONPATH

```
>>> import sys  
>>> print sys.path  
['/home/clau', '/usr/lib/python2.5/idlelib', '/usr/lib/python2.5', '/usr/lib/python2.5/plat-lin  
ux2', '/usr/lib/python2.5/lib-tk', '/usr/lib/python2.5/lib-dynload', '/usr/local/lib/python2.  
5/site-packages', '/usr/lib/python2.5/site-packages', '/usr/lib/python2.5/site-packages/Numeri  
c', '/usr/lib/python2.5/site-packages/PIL', '/var/lib/python-support/python2.5', '/var/lib/pyt  
hon-support/python2.5/gtk-2.0']  
>>> |
```

Funciones predefinidas: dir()

dir(): retorna una lista ordenada de strings con los nombres definidos en el módulo.

```
>>> import pygame
>>> dir(pygame)
['ACTIVEEVENT', 'ANYFORMAT', 'ASYNCBLIT', 'AUDIO_S16', 'AUDIO_S16LSB', 'AUDIO_S16MSB', 'AUDIO_S16SYS', 'AUDIO_S8', 'AUDIO_U16', 'AUDIO_U16LSB', 'AUDIO_U16MSB', 'AUDIO_U16SYS', 'AUDIO_U8', 'Color', 'DOUBLEBUF', 'FULLSCREEN', 'GL_ACCUM_ALPHA_SIZE', 'GL_ACCUM_BLUE_SIZE', 'GL_ACCUM_GREEN_SIZE', 'GL_ACCUM_RED_SIZE', 'GL_ALPHA_SIZE', 'GL_BLUE_SIZE', 'GL_BUFFER_SIZE', 'GL_DEPTH_SIZE', 'GL_DOUBLEBUFFER', 'GL_GREEN_SIZE', 'GL_MULTISAMPLEBUFFERS', 'GL_MULTISAMPLES', 'GL_RED_SIZE', 'GL_STENCIL_SIZE', 'GL_STEREO', 'HAT_CENTERED', 'HAT_DOWN', 'HAT_LEFT', 'HAT_LEFTDOWN', 'HAT_LEFTUP', 'HAT_RIGHT', 'HAT_RIGHTDOWN', 'HAT_RIGHTUP', 'HAT_UP', 'HWACCEL', 'HWPALLETTE', 'HWSURFACE', 'IYUV_OVERLAY', 'JOYAXISMOTION', 'JOYBALLMOTION', 'JOYBUTTONDOWN', 'JOYBUTTONUP', 'JOYHATMOTION', 'KEYDOWN', 'KEYUP', 'KMOD_ALT', 'KMOD_CAPS', 'KMOD_CTRL', 'KMOD_LALT', 'KMOD_LCTRL', 'KMOD_LMETA', 'KMOD_LSHIFT', 'KMOD_META', 'KMOD_MODE', 'KMOD_NONE', 'KMOD_NUM', 'KMOD_RALT', 'KMOD_RCTRL', 'KMOD_RMETA', 'KMOD_RSHIFT', 'KMOD_SHIFT', 'K_0', 'K_1', 'K_2', 'K_3', 'K_4', 'K_5', 'K_6', 'K_7', 'K_8', 'K_9', 'K_AMPERSAND', 'K_asterisk', 'K_AT', 'K_BACKQUOTE', 'K_BACKSLASH', 'K_BACKSPACE', 'K_BREAK', 'K_CAPSLOCK', 'K_CARET', 'K_CLEAR', 'K_COLON', 'K_COMMA', 'K_DELETE', 'K_DOLLAR', 'K_DOWN', 'K_END', 'K_EQUALS', 'K_ESCAPE', 'K_EURO', 'K_EXCLAIM', 'K_F1', 'K_F10', 'K_F11', 'K_F12', 'K_F13', 'K_F14', 'K_F15', 'K_F2', 'K_F3', 'K_F4', 'K_F5', 'K_F6', 'K_F7', 'K_F8', 'K_F9', 'K_FIRST', 'K_GREATER', 'K_HASH', 'K_HELP', 'K_HOME', 'K_INSERT', 'K_KP0', 'K_KP1', 'K_KP2', 'K_KP3', 'K_KP4', 'K_KP5', 'K_KP6', 'K_KP7', 'K_KP8', 'K_KP9', 'K_KP_DIVIDE', 'K_KP_ENTER', 'K_KP_EQUALS', 'K_KP_MINUS', 'K_KP_MULTIPLY', 'K_KP_PERIOD', 'K_KP_PLUS', 'K_LALT', 'K_LAST', 'K_LCTRL', 'K_LEFT', 'K_LEFTBRACKET', 'K_LEFTPAREN', 'K_LESS', 'K_LMETA', 'K_LSHIFT', 'K_LSUPER', 'K_MENU', 'K_MINUS', 'K_MODE', 'K_NUMLOCK', 'K_PAGEDOWN', 'K_PAGEUP', 'K_PAUSE', 'K_PERIOD', 'K_PLUS', 'K_POWER', 'K_PRINT', 'K_QUESTION', 'K_QUOTE', 'K_QUOTEDBL', 'K_RALT', 'K_RCTRL', 'K_RETURN', 'K_RIGHT', 'K_RIGHTBRACKET', 'K_RIGHTPAREN', 'K_RMETA', 'K_RSHIFT', 'K_RSUPER', 'K_SCROLLLOCK', 'K_SEMICOLON', 'K_SLASH', 'K_SPACE', 'K_SYSREQ', 'K_TAB', 'K_UNDERSCORE', 'K_UNKNOWN', 'K_UP', 'K_a', 'K_b', 'K_c', 'K_d', 'K_e', 'K_f', 'K_g', 'K_h', 'K_i', 'K_j', 'K_k', 'K_l', 'K_m', 'K_n', 'K_o', 'K_p', 'K_q', 'K_r', 'K_s', 'K_t', 'K_u', 'K_v', 'K_w', 'K_x', 'K_y', 'K_z', 'MOUSEBUTTONDOWN', 'MOUSEBUTTONUP', 'MOUSEMOTION', 'NOEVENT', 'NOFRAME', 'NUMEVENTS', 'OPENGL', 'OPENGLBLIT', 'Overlay', 'PREALLOC', 'QUIT', 'RESIZABLE', 'RLEACCEL', 'RLEACCELOK', 'Rect', 'SRCALPHA', 'SRCOLORKEY', 'SWSURFACE', 'SYSWMEVENT', 'Surface', 'SurfaceType', 'TIMER_RESOLUTION', 'USEREVENT', 'UYVY_OVERLAY', 'VIDEOEXPOSE', 'VIDEORESIZE', 'YUY2_OVERLAY', 'YV12_OVERLAY', 'YVU_OVERLAY', '__builtins__', '__doc__', '__file__', '__name__', '__path__', '__rect_constructor__', '__rect_reduce__', '__version__', '__warningregistry__', '_check_darwin', 'base', 'cdrom', 'color', 'colordict', 'constants', 'cursors', 'display', 'draw', 'error', 'event', 'fastevent', 'font', 'get_error', 'get_sdl_version', 'image', 'init', 'joystick', 'key', 'mixer', 'mouse', 'movie', 'movieext', 'msg', 'overlay', 'packager_imports', 'quit', 'rect', 'register_quit', 'segfault', 'sndarray', 'sprite', 'string', 'surface', 'surfarray', 'sysfont', 'time', 'transform', 'ver', 'vernum', 'version']
>>> |
```

Funciones predefinidas:locals() y globals()

globals() y **locals()**: Retornan los nombres en los espacios de nombres local y global.

```
>>> def miFuncion(un):
    x=12
    y=13
    print locals()

>>> miFuncion(1)
{'y': 13, 'x': 12, 'un': 1}
>>>
>>>
>>> print globals()
{'__builtins__': <module '__builtin__' (built-in)>, '__name__': '__main__', '__doc__': None, 'miFuncion': <function miFuncion at 0x828c684>}
>>>|
```

El tipo retornado es un diccionario. Por lo tanto:

locals().keys() ➡ ¿Qué retorna?

El módulo Collections

Define tipos de datos alternativos a los predefinidos dict, list, set, y tuple:

Counter: es una colección desordenada de pares claves-valor, donde las claves son los elementos de la colección y los valores son la cantidad de ocurrencias

Ejemplos:

```
from collections import Counter
```

```
cnt = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])  
print(Counter('abracadabra').most_common(1))
```

El módulo Collections

deque: permite implementar pilas y colas

```
from collections import deque
```

```
d = deque('abcd')
```

```
d.append("e")      → agrega al final
```

```
d.pop()           → eliminar último
```

```
d.popleft()        → elimina primero
```

Paquetes

- Permiten crear una estructura jerárquica de módulos.
- Se acceden por notación puntual.
- Es un directorio donde se ubican los archivos con los módulos.

Ejemplo

▼ Python
▼ Clases
▼ pruebas
▼ miPaquete

Puede estar vacío o puede utilizarse para hacer algunas inicializaciones generales del paquete.

__init__.py	12 bytes	script en Python
modulo1.py	142 bytes	script en Python
modulo2.py	142 bytes	script en Python
modulo3.py	142 bytes	script en Python

Paquetes

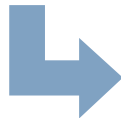
<http://docs.python.org/tutorial/modules.html>

Ejemplo más complejo

```
sound/
  __init__.py
  formats/
    __init__.py
    wavread.py
    wavwrite.py
    aiffread.py
    aiffwrite.py
    auread.py
    auwrite.py
    ...
  effects/
    __init__.py
    echo.py
    surround.py
    reverse.py
    ...
  filters/
    __init__.py
    equalizer.py
    vocoder.py
    karaoke.py
```

Top-level package
Initialize the sound package
Subpackage for file format conversions
Subpackage for sound effects
Subpackage for filters

```
import sound.effects.echo
```



```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

```
from sound.effects import echo
```



```
echo.echofilter(input, output, delay=0.7, atten=4)
```

Paquetes

`__all__`: es una variable que contiene una lista con los nombres de los módulos que deberían poder importarse cuando se encuentra la sentencia `from package import *`.

Ejemplo

The diagram illustrates a Python package structure for a 'sound' package. It is organized into three main subpackages: 'formats', 'effects', and 'filters'. Each subpackage contains an `__init__.py` file and several module files. The 'effects' subpackage is highlighted with a box around its `__init__.py` file, and an arrow points from this box to a separate box containing the `__all__` list. The `__all__` list is defined as `["echo", "surround", "reverse"]`, indicating that only these three modules are intended to be imported using `from sound.effects import *`.

```
sound/
  __init__.py
  formats/
    __init__.py
    wavread.py
    wavwrite.py
    aiffread.py
    aiffwrite.py
    auread.py
    auwrite.py
    ...
  effects/
    __init__.py
    echo.py
    surround.py
    reverse.py
    ...
  filters/
    __init__.py
    equalizer.py
    vocoder.py
    karaoke.py
    ...
```

Top-level package
Initialize the sound package
Subpackage for file format conversions

Subpackage for sound effects

Subpackage for filters

`__all__ = ["echo", "surround", "reverse"]`

Módulos estándares: sys

Existe una biblioteca de módulos estándar

Módulo sys

- **argv**: los argumentos en la línea de comandos
- **exit([arg])**: Sale del programa actual
- **modules**: es un diccionario con los nombres de los módulos
- **path**: las rutas donde buscar los módulos a cargar
- **platform**: contiene información sobre la plataforma
- **stdin**: el stream de entrada estándar
- **stdout**: el stream de salida estándar
- **stderr**: el stream del error estándar

Módulos estándares: os

Módulo os

- **environ**: Mapea las variables de ambiente
- **system(command)**: ejecuta un comando del sistema
- **sep**: separador dentro del path
- **pathsep**: separador para separar paths
- **linesep**: separador de línea ('\n', '\r', or '\r\n')
- **urandom(n)**: retorna n bytes de datos aleatorios

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned on the left side of the slide.

Lenguaje Python

Referencias

Referencias

- Diferencias entre Python 2 y 3

<http://www.diveintopython3.net/porting-code-to-python-3-with-2to3.html>

- Lo visto en clase...

- Sitio Oficial: <http://python.org/>
- Documentación en español:
<https://wiki.python.org/moin/SpanishLanguage>
- **De los libros en la Biblioteca:**
- Capítulos 4 y 5 Programming in Python 3
- Capítulos 6 y 10 Beginning Python
- Capítulos de Funciones (pág 34) y Módulos y Paquetes (pág 69) Python para todos

Lenguaje Python

Esta presentación se encuentra disponible bajo los términos la licencia
Creative Commons Atribución-Compartir Obras Derivadas Igual 2.5
Argentina License.

