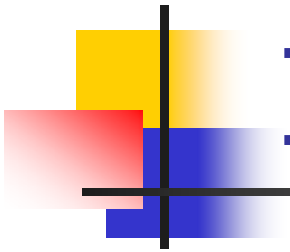
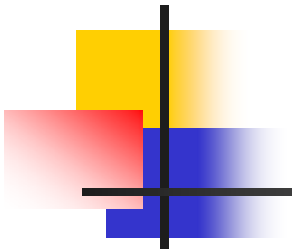


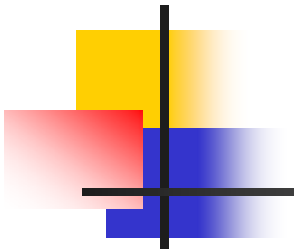
Introducción a Python

- Francisco Barranco Expósito
- Antonio Guerrero Galindo
- Manuel Entrena Casas
- Alvaro González Nonay



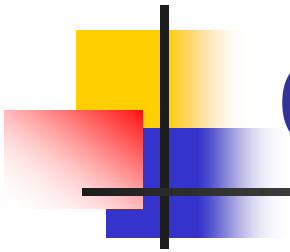
¿Qué es Python?

- Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipado dinámico, tipos de datos dinámicos de muy alto nivel, y clases.
- Python combina potencia con una sintaxis muy clara. Tiene interfaces a muchas llamadas al sistema y bibliotecas, así como también a varios sistemas de ventanas, y es extensible en C o C++.
- También es utilizable como un lenguaje de extensión para aplicaciones que necesiten interfaces programables. Finalmente, Python es portable, corre en muchas variantes de Unix, en la Mac, y en PCs bajo MS DOS, Windows, Windows NT, y OS/2.



Historia

- Fue creado a principios de los 90 por Guido Van Rossum en el Stichting Mathematisch Centrum (Instituto Nacional de Investigación de Matemáticas y Ciencias de la Computación en Holanda), como sucesor de un lenguaje llamado ABC.
- En 1995, Guido continuó su trabajo en la Corporation for National Research Initiatives (Corporación Nacional de Iniciativas de Investigación), en Virginia, EE.UU; donde lanzó varias versiones del software.
- En Mayo de 2000, Guido y el grupo de desarrolladores del nucleo de Python se trasladaron a los laboratorios de BeOpen.com.
- En Octubre del mismo año se mudaron a Digital Creations, actualmente Zope Corporation.
- En 2001 se fundó la Python Software Foundation (PSF), organización sin ánimo de lucro, para poseer la propiedad intelectual sobre Python.



Comparaciones: Java vs Python

Los programas Python generalmente son más lentos en ejecución que los Java, pero se desarrollan en mucho menos tiempo.

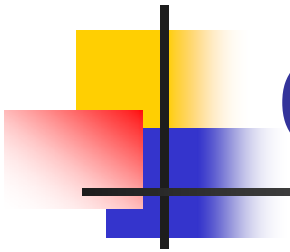
Se tarda del orden de 3 a 5 veces menos que un programa equivalente en Java.

Tal diferencia podría atribuirse a los tipos de datos de alto nivel integrados y al tipado dinámico de Python.

Por ejemplo, un programador de Python no emplea el tiempo en declarar los tipos de los argumentos o las variables, y la potencia polimórfica de las listas y diccionarios de Python, cuyo soporte sintáctico está dentro del lenguaje, encuentra un uso en casi todos los programas Python.

A causa del tipado en tiempo de ejecución, la ejecución de Python debe trabajar más:

Por ejemplo, cuando se evalua la expresión a+b, primero inspecciona los objetos a y b, para hallar su tipo, desconocido en tiempo de compilación. Luego invoca la operación de suma apropiada, que puede ser un método sobrecargado definido por el usuario.

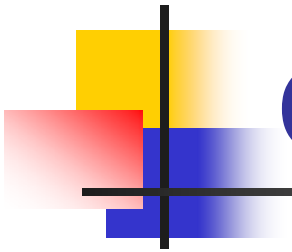


Comparaciones: Java vs Python

Por otra parte, Java, puede desarrollar una suma de enteros o reales muy eficiente, pero requiere la declaración de variables para a y b, y no permite la sobrecarga del operador suma en instancias definidas por el usuario.

Por todo esto, Python es mucho mejor empleado como lenguaje de unión, mientras que Java se caracteriza más como un lenguaje de implementación de bajo nivel.

Los dos juntos hacen una excelente combinación: los componentes se desarrollan en Java y son combinados para formar aplicaciones con Python.



Comparaciones: Smalltalk vs Python

Quizás la mayor diferencia entre Python y Smalltalk es que Python tiene una sitanxis más de "flujo principal", lo que echa una mano al entrenamiento de programación.

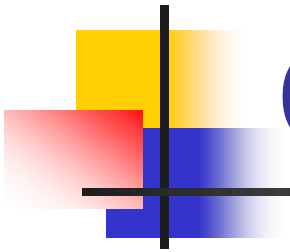
Como Smalltalk, Python tiene tipado y asignación dinámica, y todo es un objeto.

Sin embargo, Python distingue entre tipos de objetos integrados de clases definidas por el usuario y actualmente no permite herencia en tipos integrados.

La biblioteca estándar de tipos de datos está mas depurada, pero la de Python está más orientada a tratar con Internet y todo el mundo del WWW: Email, HTML, FTP, etc.

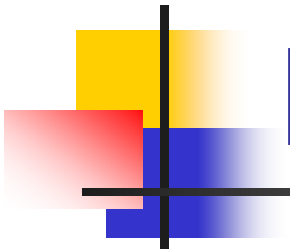
Python tiene una filosofía distinta, desde el punto de vista del entorno de desarrollo. Mientras que Smalltalk tiene un monolítico "sistema de imagen" que comprende tanto el entorno como el programa de usuario, Python almacena los módulos estándar y los modulos de usuario en archivos individuales que pueden ser fácilmente reubicados y distribuidos fuera del sistema.

Una consecuencia, entre otras, es que hay más de una opción a la hora de conectar una interfaz gráfica de usuario a un programa Python.



Características

- Posee una sintaxis sencilla: rápido aprendizaje.
- Tratamiento de excepciones con nombre.
- Extensible a otros sistemas software.
- Flexible en el tratamiento del lenguaje:un módulo que interactuará con un sistema externo puede ser probado con una "imitación" del sistema escrito en Python.
- Es un lenguaje dinámicamente interpretado.
- Es orientado a objetos:
  - Herencia múltiple.
  - Ligadura dinamica.
  - Polimorfismo.
  - Su núcleo es también orientado a objetos: jerarquía de clases.
- Portable: está implementado en C estándar usando E/S Posix.
- Gratuito, y de libre distribución.



Desventajas

Python no pretende se perfecto para todos los propósitos.

Podemos suponer que la mayoría de los programas escritos hoy en día podrían pasarse a Python, pero él sólo no sería suficiente para la cantidad de aplicaciones que están orientadas a componentes compilados.

Por ejemplo, no está indicado para las siguientes aplicaciones:

- Algoritmos de compresión de datos:

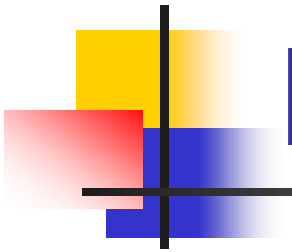
  Estos algoritmos traducen un flujo de datos a una forma más pequeña.

Esto implica examinar cada byte, y hacer un tratamiento estadístico. Para grandes cantidades de datos los compresores escritos en Python resultan demasiado lentos.

- Controladores de dispositivos:

Al igual que antes, una aplicación que realice millones de operaciones en punto flotante no uniformes será demasiado lenta en Python.

- Operaciones críticas de bases de datos.
- Operaciones altamente especializadas.



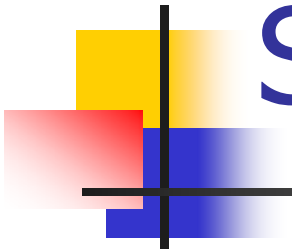
Python en la actualidad: usos

Algunos de los proyectos más importantes que utilizan Python son:

- AstraZeneca usa Python para la investigación cooperativa de medicamentos:

  AstraZeneca, una de compañías farmacéuticas más importantes del mundo, utiliza Python para reducir costes e incrementar la productividad en los procesos de identificación farmacológicos.
- En Philips, la Linea de Semiconductores es sobre Python: codifica la lógica que controla la producción de semiconductores.
- ForecastWatch.com usa Python para ayudar a los meteorólogos en sus previsiones.
- Control de tráfico aéreo: Python y Jython proporcionan las trazas de la interfaz y procesador usados en el control de tráfico de los aeropuertos.
- Industrial Light & Magic es sobre Python: La compañía de efectos especiales que creo StarWars, usa Python para unir los miles de computadores y componentes de hardware en su producción de gráficos.
- La compañía marítima Tribon Solutions usa Python para incrementar su eficiencia en el diseño y construcción de buques.
- Python colabora con la misión Espacial Shuttle: La United Space Alliance utiliza Python para enviar soluciones de ingeniería de calidad a un bajo coste.

Sintaxis Python



- Comentarios : detrás de #
- Asignación : se usa =
- Ej:

```
>>> a=3+2J
>>> # Esto es un comentario
>>>B=7 #asignación
```

Sintaxis Python

- Definición de funciones:
- >>>def nombrefunción(arg1,arg2..):

... instrucción1

... instrucción2

... .

... instrucciónN

...

- La indentación delimita qué instrucciones pertenecen a la función.
- A no ser que se use return, la función devuelve None.

Sintaxis Python

- If:
- >>>if condicion:

```
...             instrucciones
...     elif condicion:
...             instrucciones
...     else:
...             instrucciones
...
```

Sintaxis Python

- For:

>>>for variable in variable_lista:

... instrucciones

...

- Range: usado de la forma range(i) genera la lista [0,1,...,i].
- usado de la forma range(i,j) genera la lista [i,i+1,...,j-1]

Sintaxis Python

- filter(función, lista_arg) devuelve una lista con los elementos x de lista_arg que cumplen f(x) = true.

- >>>def f(x): return x%2 != 0 and x%3 != 0

...

- >>>filter(f, range(2,25))

[5,7,11,13,17,19,23]

Sintaxis Python

- map(función, lista_arg) devuelve la lista:

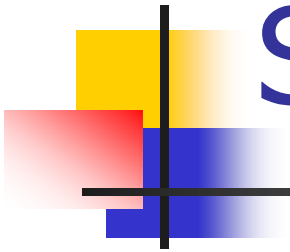
[f(lista_arg[0]),f(lista_arg[1]),...,f(lista_arg[n])]

- >>>def cubo(x): return x*x*x

...

- >>> map(cubo, range(1,5))

[1,8,27,64]



Sintaxis Python

- reduce(funcion, lista_arg) llama a funcion con los dos primeros elementos de lista_arg, luego la llama con el tercero y el resultado anterior, etc. Si sólo hay un elemento se devuelve este, y si no hay ninguno da error.

Sintaxis Python

- >>>def add(x,y): return x+y

...

- >>>reduce(add, range(1,11))

55

Sintaxis Python

- List Comprehensions: operaciones muy intuitivas con listas
- >>>vec = [2,4,6]
- >>>[3 * x for x in vec]

[6,12,18]

- >>>[3 * x for x in vec if x > 3]

[12,18]

- >>>[[x,x**2] for x in vec]

[[2,4],[4,16],[6,36]]

Sintaxis Python

- Definición y uso de Clases:
- >>>class NombreDeClase:

... sentencia 1

... .

... sentencia N

...

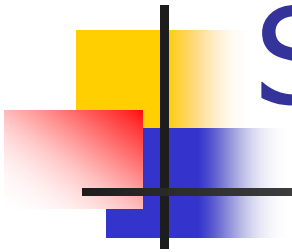
- >>>variable = NombreDeClase ()

Sintaxis Python

```
>>>class MiClase:
...                 "Una clase de ejemplo"
...                 i = 12435
...                 def f(self):
...                             return "hola mundo"
...
```

```
>>>x = MiClase()
>>>x.i
12435
>>>x.f()
"hola mundo"
```

Sintaxis Python



- Se puede incluir una función, llamada init, que se ejecute siempre que se cree una instancia de la clase.

- Def __init__(self):
  instrucciones

- init no puede hacer return.

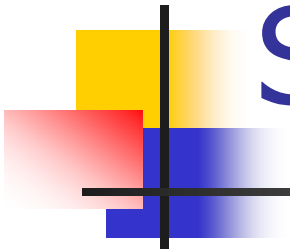
Sintaxis Python

- Las variables de instancia no necesitan declararse.

```
>>>x=MiClase()
>>>x.numero=7
```

- x.f() equivale a MiClase.f(x)
- >>>MiClase.f(x)

"hola mundo"



Sintaxis Python

- Para definir una subclase se escribe:
- class NombreDeClase(NombreSubclase)

- Se heredan funciones y variables de clase.

Sintaxis Python

```
>>>class MiSubClase(MiClase):
...                                     j=5
...                                     def g(self):
...                                             return 5
...
>>>x=MiSubClase()
>>>x.j
5
>>>x.i
12435
>>>x.f()
“hola mundo”
```

Sintaxis Python

- Podemos redefinir una función en la subclase:
- >>> class MiSubClase(MiClase):

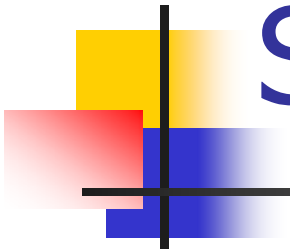
... def f(self):

... return “HOLA MUNDO”

...

- >>> x = MiSubClase()
- >>> x.f()

“HOLA MUNDO”



Sintaxis Python

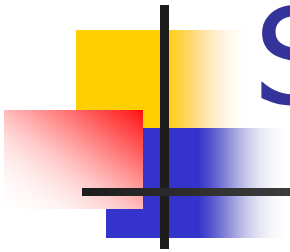
- Y para llamar al método de la superclase :

- >>>x.f()

"hola mundo"

- >>>MiClase.f(x)

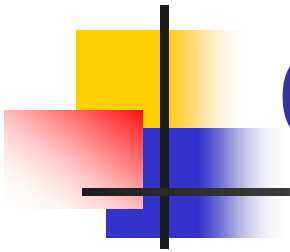
"hola mundo"



Sintaxis Python

- Es muy fácil dotar de herencia múltiple, basta con incluir más clases como argumentos en la definición:
- >>>class MiSubClase(Superclase1, Superclase2,...)

- A la hora de resolver un mensaje a un objeto de MiSubClase se buscará el método en la propia clase, si no se encuentra se busca en Superclase1 (y en sus superclases), luego en Superclase2, etc.



Clases básicas de Python

Numeros :

-Enteros

-Flotantes

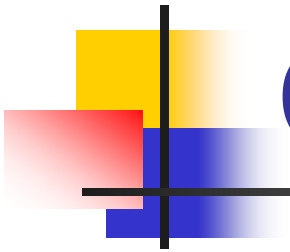
-Complejos

Cadenas

Tuplas

Listas

Diccionarios

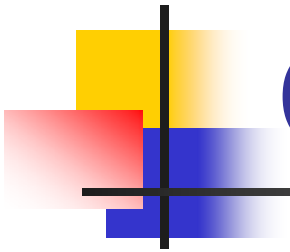


Clases básicas de Python

- Python como calculadora: Enteros

```
>>> 2+2
4
>>> 5*6
30
>>> 7/2       # La división entera redondea hacia abajo:
 2
```

Los operadores +, -, * y / funcionan como en otros lenguajes (p. ej. Pascal o C).



Clases básicas de Python

- Flotantes

```
>>> 3.0  #Esto es un flotante
```

Los operadores con tipos mixtos convierten el operando entero a coma flotante:

```
>>> 4 * 2.5 / 3.3
3.0303030303
>>> 7.0 / 2
3.5
```

Clases básicas de Python

- Números Complejos:

Los números imaginarios se escriben con el sufijo "`j`" o "`J`". Los números complejos con una parte real distinta de cero se escriben "(*real*+*imag*`j`)", y se pueden crear con la función "`complex(`*real*, *imag*`)`".

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
```

Para extraer una de las partes de un número complejo *z*, usa *z*`.real` y *z*`.imag`.

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Clases básicas Python

- Además de los números, Python también sabe manipular cadenas, que se pueden

```
>>> 'fiambre huevos‘
'fiambre huevos‘
>>> 'L\'Hospitalet'
"L'Hospitalet“
>>> "L'Hospitalet“
"L'Hospitalet"

>>> hola = "Esto es un texto bastante largo \
        con saltos de linea.”
>>> print hola
Esto es un texto bastante largo
       con saltos de linea.
```

Clases básicas de Python

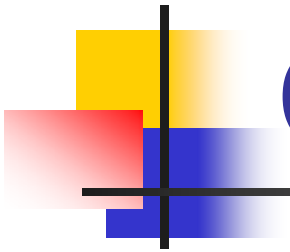
Se pueden concatenar cadenas con el operador + y repetir con el operador *

```
>>> palabra = 'Ayuda' + 'Z‘
>>> palabra
'AyudaZ‘
>>> '<' + palabra*5 + '>‘
'<AyudaZAyudaZAyudaZAyudaZAyudaZ>'
```

Se puede indexar una cadena. Como en C, el primer carácter de una cadena tiene el indice 0.

```
>>> palabra[4]
'a‘
 >>> palabra[0:2]                    #Notacion corte!!! Para subcadenas
'Ay‘
>>> palabra[2:4]
'ud'
```

Una vez creada una cadena no se puede modificar!!!

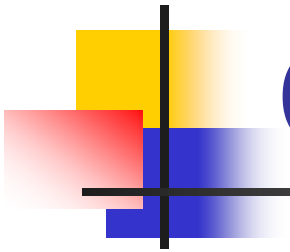


Clases básicas de Python

- Tuplas:

Una tupla consta de cierta cantidad de valores separada por comas, por ejemplo:

```
>>> t = 12345, 54321, '¡hola!‘
>>> t[0]
12345
>>> t
(12345, 54321, '¡hola!')
>>> # Se pueden anidar tuplas:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, '¡hola!'), (1, 2, 3, 4, 5))
```



Clases básicas de Python

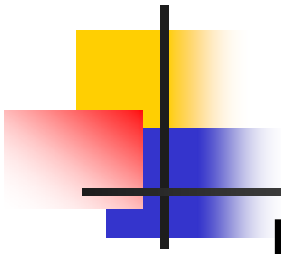
Para sacar los valores de una tupla se puede hacer uno a uno o mediante el desempaquetado de secuencias:

```
>>> x, y, z = t
```

Esto requiere que el numero de variables sea igual al numero de elementos de la tupla.

Las tuplas, como las cadenas, son inmutables, aunque se puede simular el mismo efecto mediante corte y concatenacion.

Otra opción es que la tupla contenga elementos mutables(como listas)



Clases básicas de Python

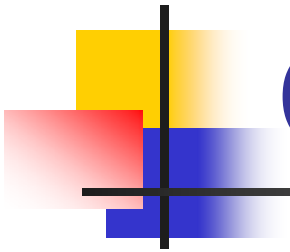
Listas:

De los tipos de datos secuenciales que hemos visto el mas versátil es el de lista.

```
>>> a = ['fiambre', 'huevos', 100, 1234]
>>> a
['fiambre', 'huevos', 100, 1234]
```

Las listas también se pueden cortar, concatenar, etc.:

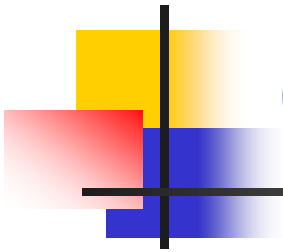
```
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['huevos', 100]
>>> a[:2] + ['bacon', 2*2]
['fiambre', 'huevos', 'bacon', 4]
```



Clases básicas de Python

es posible cambiar los elementos de una lista:

```
>>> a
['fiambre', 'huevos', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['fiambre', 'huevos', 123, 1234]
>>> a[0:2] = [1, 12]                    # Reemplazar elementos
>>> a
[1, 12, 123, 1234]
>>> a[0:2] = []                 # Quitar elementos
>>> a
[123, 1234]
>>> a[1:1] = ['puaj', 'xyzzy']          # Insertar cosas
>>> a
[123, 'puaj', 'xyzzy', 1234]
```



Clases básicas de Python

Como usar una lista como pila :

Para apilar un elemento, usa `append( )`. Para recuperar el elemento superior de la pila, usa `pop( )` sin un índice explícito. Por ejemplo:

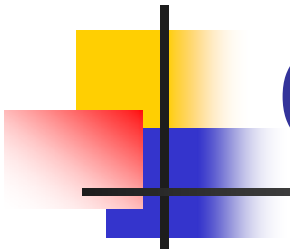
```
>>> pila = [3, 4, 5]
>>> pila.append(6)
>>> pila.append(7)
>>> pila
[3, 4, 5, 6, 7]
>>> pila.pop()
7
>>> pila
[3, 4, 5, 6]
>>> pila.pop()
6
>>> pila
[3, 4, 5]
```

Clases básicas de Python

Como usar una lista como cola:

El metodo usado para insertar elementos es el mismo append(), pero ahora para sacarlos usamos pop() con indice 0, que saca elementos de la lista por el principio :

```
>>> cola = [“Pepe", “Fran", “Manolo"]
>>> cola.append(“Jose")          # llega Jose
>>> cola.append(“Carlos")         # llega Carlos
>>> cola.pop(0)
‘Pepe‘
>>> cola.pop(0)
‘Fran‘
>>> cola
['Manolo', ‘Jose', ‘Carlos']
```



Clases básicas de Python

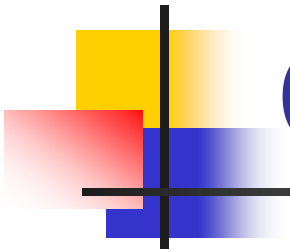
Diccionario:

A diferencia de las secuencias, que se indexan mediante un rango de números, los diccionarios se indexan mediante *claves*, que pueden ser de cualquier tipo inmutable.

Las operaciones principales sobre un diccionario son la de almacenar una valor con una clave dada y la de extraer el valor partiendo de la clave . También se puede eliminar una pareja *clave*: *valor* con `del`.

El metodo keys() devuelve todas las claves usadas en el diccionario.

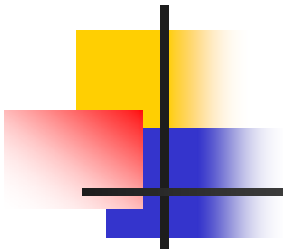
Un diccionario se crea si se coloca entre llaves parejas *clave: valor* separadas por comas. Si no hay ninguna pareja se crea el diccionario vacio.



Clases básicas de Python

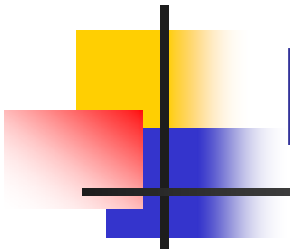
He aquí un pequeño ejemplo que utiliza un diccionario:

```
>>> tel = {‘primera': 4098, ‘segunda': 4139}
>>> tel[‘otra'] = 4127
>>> tel
{'segunda': 4139, ‘otra': 4127, ‘primera': 4098}
>>> tel[‘primera']
4098
>>> del tel[‘segunda']
>>> tel[‘tercera'] = 4127
>>> tel
{‘otra': 4127, ‘tercera': 4127, ‘primera': 4098}
>>> tel.keys()
[‘otra', ‘tercera', ‘primera']
```



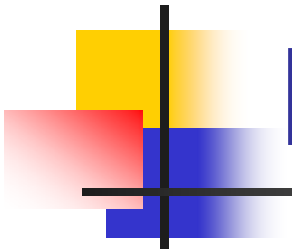
Ejemplo Correo

```
import smtplib
 import sys,os

def enviar_mensajes_a_lista_de_correo(mail_enviador,asunto,servidorsmtp,puertosmtp):

fichero_con_lista_direcciones=raw_input('Introducir path completo del fichero de
      direcciones: ')
fichero_con_cuerpo_mensaje=raw_input('Introducir path completo del fichero con el
      cuerpo del mensaje: ')
fichero_con_mensajes_enviados=raw_input('Introducir path completo para el fichero de
      mensajes enviados: ')
fld=open(fichero_con_lista_direcciones,'r')
fme=open(fichero_con_mensajes_enviados,'w')
```



Ejemplo Correo

```
for i in fld.readlines():
try:
mensaje = "From: "+mail_enviador+"\nSubject: "+asunto+"\nTo: "+i+"\n"
     fcm=open(fichero_con_cuerpo_mensaje,'r')
print '\nMensaje para: '+i
 for ii in fcm.readlines():
mensaje=mensaje+ii
fcm.close()
print 'Abierto fichero del cuerpo del mensaje...\n'
server = smtplib.SMTP(servidorsmtp,int(puertosmtp))
server.sendmail(mail_enviador, i, mensaje)
 fme.write('Mensaje enviado a:'+i)
server.quit()
```



Ejemplo Correo

```
except:
print 'Error enviando el fichero a: '+ i
 fme.write('Error enviado el fichero a: '+i)
fld.close()
fme.close()
```

Ejemplo Correo

```
if __name__ == '__main__' :
 print '''Uso:
Este programa envia un mensaje a una lista de direcciones de e-mail.
El fichero de direcciones debe de tener una direccion de e-mail por linea.
El fichero con el cuerpo del mensaje debe de estar en formato texto.
El puerto de SMTP estandard es el 25.
'''
enviador_mensaje=raw_input('Introducir e-mail del enviador: ')
asunto_mensaje=raw_input('Introducir asunto del mensaje: ')
servidorsmtp_mensaje=raw_input('Introducir servidor de SMTP: ')
puertosmtp_mensaje=raw_input('Introducir puerto del SMTP : ')
enviar_mensajes_a_lista_de_correo(enviador_mensaje,asunto_mensaje,servid
    orsmtp_mensaje,puertosmtp_mensaje)
sys.exit()
```

ALGUNAS DIRECTIVAS

En Python existen una serie de directivas distribuidas en módulos (decisión de diseño) como son:

1. type: Devuelve el tipo de dato de cualquier objeto.

```
>>type (1)                    >>type([‘a’, ‘b’])
<type ‘int’>                  <type ‘list’>
```

2. str: Transforma un dato en una cadena:

```
>>str(1)                     >>str([‘a’,’b’])
’1’                          “[ ‘a’, ‘b’]”
```

3. dir: Devuelve una lista de los atributos y métodos de cualquier objeto: módulos, funciones, cadenas, listas ...

```
>>dir ([‘a’,’b’])
>>['index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

LÓGICA EN PYTHON

- En Python, and y or realizan las operaciones de lógica booleana como cabe esperar, pero no devuelven valores booleanos; devuelven uno de los valores reales que están comparando.

```
>>> 'a' and 'b'
'b'
>>> '' and 'b'
''
>>> 'a' and 'b' and 'c'
'c'
```

```
>>> 'a' or 'b'
'a'
>>> '' or 'b'
'b'
>>> '' or [] or {}
{}
```

La expresión bool ? a : b, que se evalúa como a si bool es verdadero, y b en caso contrario, se puede realizar en Python.

```
>>> a = ""
>>> b = "second"
>>> (1 and [a] or [b])[0]
```

FUNCIONES LAMBDA

- Tomadas de Lisp, definen funciones mínimas, de una línea, sobre la marcha que pueden usarse donde usemos una función.
- >>> **def f**(x):
  ... **return** x*2
  ...

  >>> g = **lambda** x: x*2 (1)
  >>> g(3)

Las funciones lambda, no pueden contener más de una expresión, ni tampoco órdenes, no se debe exprimir mucho la funcionalidad de este mecanismo, si se necesita algo más complejo, se deben definir funciones normales.

Las funciones lambda son una cuestión de estilo. Siempre se podrá definir una función normal, separada y utilizarla en su lugar.

CADENAS DE DOCUMENTACIÓN

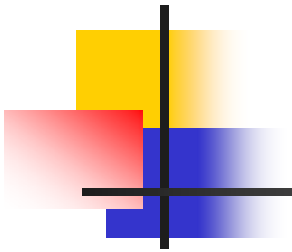
La primera línea debe ser un resumen corto y conciso de lo que debe hacer el método. No se hace constar el nombre y tipo del objeto, pues éstos están disponibles mediante otros modos. Esta línea debe empezar por mayúscula y terminar en punto.

```
>>> def mifuncion():
    """No hace nada, es un ejemplo de comentario

    COMENTARIO
    """
```

Si hay más líneas, la segunda debe ir en blanco, separando visualmente el resumen del resto de la descripción. Las siguientes deben ser párrafos que describan las convenciones de llamada de los objetos, sus efectos secundarios, etc.

BASES DE DATOS Y PYTHON

- Lo que es JDBC en Java es DB API en Python
- Para conectarnos a una base de datos usamos el método connect que devuelve un objeto de tipo conection
- El objeto connection tiene el método cursor() que sirve para recuperar un cursor de la BD. También: close(), commit(), rollback(), cursor() ...
- El objeto cursor define entre otros los siguientes métodos:
- → execute(), fetchone(), fetchmany([size]), fetchall()
- Hay varios módulos que implementan el estándar DB API:
- → DCOracle (http://www.zope.org/Products/DCOracle/) creado por Zope
- → MySQLdb (http://sourceforge.net/projects/mysql-python)

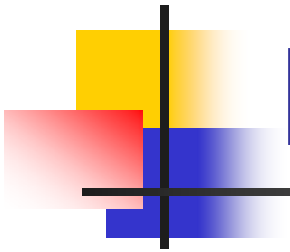


ERRORES Y EXCEPCIONES I

- Hay (al menos) dos tipos de errores diferenciables: los *errores de sintaxis* y las *excepciones*.
- Los errores de sintaxis son los más comunes en el intérprete:

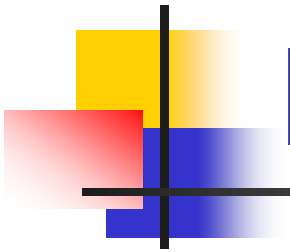
```
>>> while 1 print 'Hola mundo'
File "<stdin>", line 1
while 1 print 'Hola mundo'
^
SyntaxError: invalid syntax
```

- Se muestran el nombre del fichero y el número de línea para saber dónde buscar, si la entrada venía de un fichero.
- Los errores que se detectan en la ejecución se llaman *excepciones.*



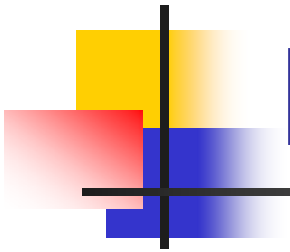
ERRORES Y EXCEPCIONES II

```
>>> 10 * (1/0)
Traceback (innermost last):
File "<stdin>", line 1
ZeroDivisionError: integer division or modulo
>>> 4 + variable*3
Traceback (innermost last):
File "<stdin>", line 1
NameError: variable
>>> '2' + 2
Traceback (innermost last):
File "<stdin>", line 1
TypeError: illegal argument type for built-in operation
```



ERRORES Y EXCEPCIONES III

- La última línea del mensaje de error indica qué ha pasado. Las excepciones pueden ser de diversos tipos, que se presentan como parte del mensaje.
- Es posible escribir programas que gestionen excepciones:
- >>> while 1:
- ... try:
- ... x = int(raw_input("Introduce un número: "))
- ... break
- ... except ValueError:
- ... print "No es un número. Pruebe de nuevo..."
- ...



ERRORES Y EXCEPCIONES IV

Una sentencia try puede contener más de una cláusula except, para capturar diferentes excepciones (no se ejecuta más de un gestor para una excepción) y puede capturar más de una excepción, nombrándolas dentro de una lista:

- ... except (RuntimeError, TypeError, NameError):

- La última cláusula except puede no nombrar ninguna excepción, en cuyo caso hace de comodín y captura cualquier excepción. Se debe utilizar esto con mucha precaución, pues es muy fácil enmascarar un error de programación real de este modo.
- Cuando salta una excepción, puede tener un valor asociado, también conocido como el/los *argumento/s* de la excepción.

ERRORES Y EXCEPCIONES V

```
>>> try:
... funcion()
... except NameError, x:
... print 'nombre', x, 'sin definir'
...
nombre funcion sin definir
```

- La sentencia raise (hacer saltar, levantar) permite que el programador fuerce la aparición de una excepción. Por ejemplo:

```
>>> raise NameError, 'MuyBuenas'
Traceback (innermost last):
File "<stdin>", line 1
NameError: MuyBuenas
```

ERRORES Y EXCEPCIONES VI

- El usuario puede definir sus propias excepciones, tal y como aparece en la transparencia siguiente:

```
  class E(RuntimeError):
def __init__(self, msg):
  self.msg = msg
def getMsg(self):
  return self.msg
  try:
  raise E('mi mensaje de error')
  except E, obj:
     print 'Msg:', obj.getMsg()
```

Visualizaría:

```
  >>>Msg: mi mensaje de error
```