Grado en Ingeniería Informática

Administración de Sistemas I

ÍNDICE

- Python básico
  - Sintaxis.
  - Instrucciones básicas.
  - Tipo de datos.
  - Orientación a objetos

Historia

- Creado por Guido van Rossum a finales de los 80 en el Centro para las Matemáticas y la Informática.

  (http://www.python.org/~guido/).
- Python se le conoce además por el título BDFL "*Benevolent Dictator for Life*"

Hola Mundo

```
#!/usr/bin/env python
print "Hola Mundo" # "Hola Mundo"

print "hola", "mundo" # "hola mundo"

print "Hola" + "Mundo" # "HolaMundo"
```

Características

- Legible y elegante.
- Simple y poderoso.
  - Minimalista: todo aquello innecesario no hay que escribirlo (;, {, }, '\n')
  - Muy denso: pocas líneas de código para obtener resultados.
  - Soporta objetos y estructuras de datos de alto nivel: strings, listas, etc.
  - Múltiples niveles de organizar código: funciones, clases, módulos, y paquetes
    - Python standard library (http://www.python.org/doc/current/lib/lib.html) contiene un sinfín de clases de utilidad
  - Si hay áreas que son lentas se pueden reemplazar por plugins en C o C++, siguiendo la API para extender o empotrar Python en una aplicación

Características

- De scripting
  - No tienes que declarar constantes y variables antes de utilizarlas
  - No requiere paso de compilación/linkage
    - La primera vez que se ejecuta un script de Python se compila y genera bytecode que es luego interpretado
  - Alta velocidad de desarrollo y buen rendimiento
- Se puede utilizar en múltiples plataforma
- Open source
  - Razón por la cual la Python Library sigue creciendo
- De propósito general
  - Puedes hacer en Python todo lo que puedes hacer con C# o Java

Características

- Uso de tabulaciones.
  - Tabula una vez para indicar comienzo de bloque
  - Des-tabula para indicar el final del bloque

Código en C/Java	Código en Python

`if (x) {`
`  if (y) {`
`  f1();`
`  }`
`  f2();`
`}`
	`if x:`
	`  if y:`
	`  f1()`
	`  f2()`

Características

- Python vs. Perl vs. Java.
  - Perl está basado en awk, sed, and shell scripting y su misión es hacer las tareas de administradores de sistemas más sencillas
  - Python está basado e inspirando en OOP (Object-oriented programming)
  - Python es un lenguaje de scripting
  - Todo lo que puedes hacer con Java también lo puedes hacer con Python
    - Incluso puedes acceder a través de Python a las API de Java si usas Jython (http://www.jython.org) es la implementación de Python en Java, que permite acceder a todas las APIs de Java

Características

- Ventajas:
  - Generación de prototipos de sistema
  - Elaboración de aplicaciones cliente
  - Desarrollo web y de sistemas distribuidos
  - Desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente
- Inconvenientes:
  - Programación de bajo nivel (system-programming), como programación de drivers y kernels
    - Python es de demasiado alto nivel, no hay control directo sobre memoria y otras tareas de bajo nivel
  - Aplicaciones que requieren alta capacidad de computo (alternativa programación en C)

Instalación

- Descarga de la última versión (3.3.4 o 2.7.6).
  - http://www.python.org/download/
- Linux
  - En Debian Sarge: `apt-get install python3`
  - Para Fedora y Mandrake se pueden obtener los rpms de: http://www.python.org/2.4/rpms.html
    - `rpm -iv python2.4-2.4-1pydotorg.i386.rpm`
- Online
  - http://www.compileonline.com

¿Python 2.x o 3.x?

- No hay muchas diferencias en sintaxis (print "Hola" a print("Hola")).
- Si existe mucha diferencia:
  - Internamente (rendimiento, manejo de memoria, etc...).
  - Compatibilidad de paquetes, módulos y frameworks bastante populares en Python (cStringIO, Django, etc...).
- Más información en:
  - https://wiki.python.org/moin/Python2orPython3
  - http://blog.mkaufmann.com.ar/posts/caracteristicas-de-python-27-33-que-no-conocia/

Características

- Para arrancar el intérprete (Python interactivo) ejecutar:

```
C:\>python
Python 2.7,6 (#60, Nov 30 2004, 11:49:19) [MSC v.1310
  32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for
  more information.
>>>
```

- Un comando simple:

```
>>> print "Hola Mundo"
Hola Mundo
>>>
```

- Para salir del intérprete Ctrl-D (en Linux) o Ctrl-Z (en Windows) o:

```
>>> import sys
>>> sys.exit()
$
```

Características

- Python desde script:
  - Guardar las siguientes sentencias en fichero: `holamundo.py`

```
#!/usr/bin/env python
print “Hola Mundo!"
```

- Ejecutar el script desde línea de comando:

```
$ python holamundo.py
Hola Mundo!
$
```


Características

- Sentencias y bloques.
- Las sentencias acaban en nueva línea, no en `;`
- Los bloques son indicados por *tabulación* que sigue a una sentencia acabada en '`:`'. E.j. (`bloque.py`):

```
# comentarios de línea se indican con carácter '#'
name = "Diego1" # asignación de valor a variable
if name == "Diego":
     print "Aupa Diego"
else:
     print "¿Quién eres?"
     print "¡No eres Diego!"

$ python bloque.py
¿Quién eres?
¡No eres Diego!
```

Características

- Los identificadores sirven para nombrar variables, funciones y módulos
  - Deben empezar con un carácter no numérico y contener letras, números y '_'
  - Python es *case sensitive* (sensible a la capitalización)
- Palabras reservadas:
  - `and elif global or assert else if pass break except import print class exec in raise continue finally is return def for lambda try del from not while`
- Variables y funciones delimitadas por ___ corresponden a símbolos implícitamente definidos:
  - `__name__` nombre de función
  - `__doc__` documentación sobre una función
  - `__init__()` constructor de una clase
  - `__dict__`, diccionario utilizado para guardar los atributos de un objeto

Características

- Tipos de datos.
- Numéricos (integer, long integer, floating-point, and complex)

```
>>> x = 4
>>> int (x)
4
>>> long(x)
4L
>>> float(x)
4.0
>>> complex (4, .2)
(4+0.2j)
```

Características

- Operadores lógicos y de Comparación
  - Los operadores básicos en Python son:
    - la *<<y lógica>> (and)*
    - *la <<o lógica>> (or)*
    - *el <<no lógico>> (not)*
  - Es posible combinar valores lógicos y operadores lógicos para formar *expresiones lógicas.*
    - >>> *True and False* /↱ False
    - >>> not *True* /↱False
    - >>> (*True and False) or True* /↱True
    - >>> *True and True or False* /↱True
    - >>> *False and True or True* /↱True
    - >>> *False and True or False* /↱False

Características

- Operadores lógicos y de Comparación

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

  - La comparación de desigualdad en el lenguaje de programación C se denota con != y en Pascal con <>. Python permite usar cualquiera de los dos símbolos

Características

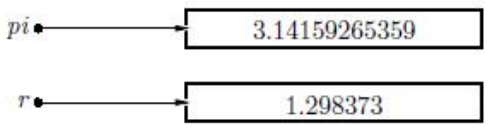
- Operadores lógicos y de Comparación

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4
Igual que	==	Binario	—	5
Distinto de	!=	Binario	—	5
Menor que	<	Binario	—	5
Menor o igual que	<=	Binario	—	5
Mayor que	>	Binario	—	5
Mayor o Igual que	>=	Binario	—	5
Negación	not	Unario	—	6
Conjunción	and	Binario	Por la izquierda	7
Disyunción	or	Binario	Por la izquierda	8

El nivel de precedencia 1 es el de mayor prioridad

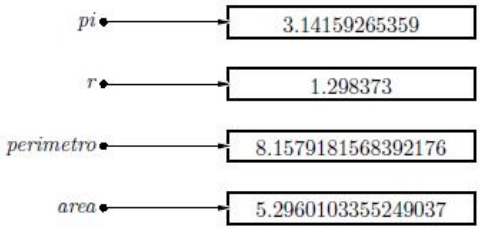
Características

- Variables y Asignaciones
  - *>>> pi = 3.14159265359 ↵*
  - >>> r = *1.298373 ↵*
  - >>> 2 * *pi * ↵*



Python reserva un espacio en la memoria, almacena el valor en él y crea una asociación entre el nombre de la variable y la dirección de memoria de dicho espacio.

  - *>>> pi = 3.14159265359 ↵*
  - *>>> = 1.298373 ↵*
  - *>>> perimetro = 2 * pi * ↵*
  - *>>> area = pi * **2 ↵*



Las asignaciones son mudas, no producen ninguna salida por consola

Nota: == no es = (comparar no es asignar)

Características

- Variables y Asignaciones
  - Un identificador no puede coincidir con una *palabra reservada o palabra clave.*
  - Lista con todas las palabras reservadas de Python: and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass,print, raise, return, try, while y yield.
  - Python distingue entre mayúsculas y minúsculas.
- Asignaciones
  - El incremento de *una variable se denota:*
  - >>> *variable+= 1 ↵* (No puede haber espacio alguno entre el + y el =.)
  - Ejemplos:
    - z += 2 z *= 2 z /= 2 z -= 2 z %= 2 z **= 2

Características

- Variables no inicializadas
  - En Python, la primera operación sobre una variable debe ser la asignación de un valor.
  - No se puede usar una variable a la que no se ha asignado previamente un valor:
    - Ejemplo

      >>> *a+ 2 ↵*

      Traceback (most recent call last):

      File "<stdin>", line 1, in ?

      NameError: name 'a' is not defined

Características

- Operadores a nivel de bit
  - El operador binario & calcula la operación *<<and>> bit a bit*
  - *El operador* binario | calcula la operación *<<or>> bit a bit*
  - *El operador binario ˆ calcula la <<or exclusiva>>* bit a bit (que devuelve cierto si y sólo si los dos operandos son distintos)
  - El operador unario ˜ invierte los bits de su operando
  - Los operadores binarios << y >> desplazan los bits a izquierda o derecha tantas posiciones como se indique.
  - Ejemplo:

En decimal		En binario	
Expresión	Resultado	Expresión	Resultado
5 & 12	4	00000101 & 00001100	00000100
5 | 12	13	00000101 | 00001100	00001101
5 ^ 12	9	00000101 ^ 00001100	00001001
~5	-6	~00000101	11111010
5 << 1	10	00000101 << 00000001	00001010
5 << 2	20	00000101 << 00000010	00010100
5 << 3	40	00000101 << 00000011	00101000
5 >> 1	2	00000101 >> 00000010	00000010

Características

- Funciones Predefinidas
  - Función abs(): cálculo del valor absoluto. Ejemplo abs(-3)
  - Función float(): conversión a flotante, acepta además argumentos de tipo cadena (la convierte a número flotante que representa).
    Ejemplos float(3)= 3.0 float('3.2e10')=32000000000.0
  - Función int: conversión a entero.
    Ejemplos int(-2.3)=-2 int('2')=2
  - Función str: conversión a cadena, se le asigna como argumento un número y devuelve una cadena.
    Ejemplos str(2.3)= '2.3' str(234E45)='2.34e+45'
  - Función round: redondea el número al flotante más próximo cuya parte decimal sea más próxima
    Ejemplos round(2.9)=3.0 round(2.1451,2)=2.15
  - Ejemplos *abs(-23) % int(7.3) =*2 *abs(round(-34.2765,1)) =*34.3

Características

- Funciones definidas en módulos
  - El módulo math

    from *math import sin* from *math import sin, cos*

    Si deseamos importar todas las funciones realizaremos **from *math import ****

  Funciones en el módulo math

  - *sin() Seno de , que debe estar expresado en radianes.*
  - *cos() Coseno de , que debe estar expresado en radianes.*
  - *tan() Tangente de , que debe estar expresado en radianes.*
  - *exp() El número elevado a .*
  - *ceil() Redondeo hacia arriba de*
  - *floor() Redondeo hacia abajo de*
  - *log() Logaritmo natural (en base) de*
  - *log10() Logaritmo decimal (en base 10) de*
  - *sqrt() Raíz cuadrada de*

Características

- Funciones definidas en módulos
  - El módulo sys

  Funciones que acceden al sistema operativo

  Funciones en el módulo sys:

  exit: anula la ejecución del intérprete

  maxint: número entero más grande con el que se puede trabajar.
  (Python 3 no pone límites, dependerá de la máquina)

  version: versión de Python

  Ejemplo

  from *sys import maxint, version*

  maxint

  2147483647

Características

- Métodos
  - Funciones especiales invocadas por ciertos tipos de datos (cadenas…).
  - Sintaxis: objeto.metodo ()

    Método lower

    cadena = 'Un ␣ EJEMPLO ␣ de ␣ Cadena' ↱

    cadena.lower() → 'un ejemplo de cadena'

    Método Upper

    'Otro ␣ ejemplo'.*upper()* → 'OTRO EJEMPLO'

    Método title

    'Juan ␣ CANO'.*title()* → 'Juan Cano'

    Método replace

    una_cadena = 'abc'.replace('b', '-') ↱ una_cadena → 'a-c'

Características

- Entrada/Salida
  - Lectura de datos

  raw_input: detiene la ejecución del programa y espera a que el usuario escriba un texto

  Ejemplo:

```
from math import pi
radio = float(raw_input('Introduce el radio:  '))
volumen = 4.0 / 3.0 * pi * radio ** 3
print volumen
```

Desde la línea de órdenes Python estará esperando una entrada desde teclado, recibirá una cadena y la función float la convertirá a flotante, para asignársela a la variable radio.

- Ejecución: python volumen_esfera.py ↱ Dame el radio: 3 → 113.097335529

Características

- La orden print
  - **print** (printredirect.py)
  - stdout en Python es sys.stdout, stdin es sys.stdin:

  Ejemplo

```
from math import pi
print 'Programa para el cálculo del volumen de una esfera.'
radio = float(raw_input('Dame el radio (en metros): '))
volumen = 4.0/3.0 * pi * radio ** 3
print 'Volumen de la esfera:',
print volumen, 'metros cúbicos'
```

Si print finaliza con una coma evitamos que imprima en una nueva línea, mostrándose un carácter en blanco.

Características

- Print: el operador %o

Ejemplo:

```
numero = int(raw_input('Dame␣un␣número:␣'))
print '%d elevado a %d es %4d' % (numero, 2, numero ** 2)
print '%d elevado a %d es %4d' % (numero, 3, numero ** 3)
print '%d elevado a %d es %4d' % (numero, 4, numero ** 4)
print '%d elevado a %d es %4d' % (numero, 5, numero ** 5)
```

Resultado

```
Dame un número: 3
3 elevado a 2 es␣␣␣␣9
3 elevado a 3 es␣␣␣27
3 elevado a 4 es␣␣␣81
3 elevado a 5 es␣␣243
```

Características

- Print
  - Las cadenas con formato son especialmente útiles para representar adecuadamente números flotantes.

  Marcas: para enteros y flotantes

  Ejemplo

  nombre = raw_input('Inserta el ␣nombre: ␣')

  print 'Hola, ␣%s.' % *nombre*

  Ejemplo

  'Programa para el cálculo del perímetro y el área de un rectángulo.'

  altura = float(raw_input('Dame a altura (en metros): '))

  anchura = float(raw_input('Dame la anchura (en metros): '))

  *area = altura * anchura*

  *perimetro = 2 * altura + 2 * anchura*

  print 'El ␣perímetro ␣es ␣de ␣%6.2f ␣metros.' % *perimetro*

  print 'El ␣´area ␣es ␣de ␣%6.2f ␣metros ␣cuadrados.' % *area*

Características

- Comentarios
- Se inician con el símbolo #*todo texto desde la almohadilla* hasta el final de la línea se considera comentario y, en consecuencia, es omitido por Python.

  Ejemplo

```
# Programa: rectangulo.py
# Propósito: Calcula el perímetro y el  área de un rectángulo a partir de su altura y anchura.
# Autor: Sergio A.
# Fecha: 14/02/2013
# Petición de los datos (en metros)
altura = float(raw_input('Dame ␣la ␣altura ␣(en ␣metros): ␣'))
anchura = float(raw_input('Dame ␣la ␣anchura ␣(en ␣metros): ␣'))
 # Cálculo del área y del perímetro
area = altura * anchura
perimetro = 2 * altura + 2 * anchura
# Impresión de resultados por pantalla
print 'El perímetro es de %6.2f metros' % perímetro # solo dos decimales.
print 'El area es de %6.2f metros cuadrados' % área
```

Características

- Strings, delimitados por un par de (', " ,""")
  - Dos string juntos sin delimitador se unen
    ```
    >>> print "Hi" "there"
    Hithere
    ```
  - Los códigos de escape se expresan a través de '\':
    ```
    >>>print '\n'
    ```
  - Raw strings
    ```
    >>> print r'\n\\' # no se 'escapa' \n
    ```
  - Lo mismo ' que ", p.e. "\\[foo\\]" r'\[foo\]'
  - Algunos de los métodos que se pueden aplicar a un string son:
    ```
    >>> len('La vida es mucho mejor con Python.')
    >>> 34
    >>> 'La vida es mucho mejor con Python.'.upper()
    'LA VIDA ES MUCHO MEJOR CON PYTHON'
    >>> "La vida es mucho mejor con Python".find("Python")
    27
    >>> "La vida es mucho mejor con Python".find('Perl')
    -1
    >>> 'La vida es mucho mejor con Python'.replace('Python', 'Jython')
    'La vida es mucho mejor con Jython'
    ```


Características

- El módulo `string` de la Python library define métodos para manipulación de strings:

  ```
  >>> import string
  >>> s1 = 'La vida es mejor con Python'
  >>> string.find(s1, 'Python')
  21
  ```

- '%' es el operador de formateo de cadenas:

  ```
  >>> provincia = ‘Teruel'
  >>> "La capital de la provincia de %s es %s"  % (provincia, “Teruel")
  'La capital de la provincia de Teruel es Teruel'
  ```

  - Los caracteres de formateo son los mismos que en C, p.e. d, f, x

Características

- Si deseamos escribir caracteres con acentos es necesario introducir la siguiente línea al comienzo de un programa Python:
  - `# -*- coding: iso-8859-1 -*-`
- Los strings en formato unicode se declaran precediendo el string de una 'u':
  - `print u'¿Qué tal estás?'`
- Listas []. Indexadas por un entero comienzan en 0:

  >>> meses = ["Enero", "Febrero"]
  >>> print meses[0] -→Enero
  >>> meses.append("Marzo")
  >>> print meses → ['Enero', 'Febrero', 'Marzo']

Características

- Listas []
  - Dos puntos (`:`) permite trabajar con una porción de la lista, el elemento indicado por el segundo parámetro no se incluye:

    >>> print meses[1:2]
    ['Febrero']
  - Más (+) es el operador de concatenación:

    >>> print meses+meses
    ['Enero', 'Febrero', 'Marzo', 'Enero', 'Febrero', 'Marzo']
  - Las listas pueden contener cualquier tipo de objetos Python:

```
>>> meses.append (meses)
>>> print meses
['Enero', 'Febrero', 'Marzo', ['Enero', 'Febrero', 'Marzo' ]]
>>> meses.append(1)
['Enero', 'Febrero', 'Marzo', ['Enero', 'Febrero', 'Marzo' ],1]
```

Características

- Listas []
  - Para añadir un elemento a una lista:

```
>>> items = [4, 6]
>>> items.insert(0, -1)
>>> items
[-1, 4, 6]
```

  - Para usar una lista como una pila, se pueden usar append y pop:

```
>>> items.append(555)
>>> items [-1, 4, 6, 555]
>>> items.pop()
555
>>> items [-1, 4, 6]
```


Características

- Tuplas ()
  - Semejante a las listas, pero no se pueden modificar

```
D:\>python
Python 2.7.6 (#60, Nov 30 2004, 11:49:19) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> mitupla = ('a', 1, "hola")
>>> mitupla[2]
'hola'
>>> dir(mitupla)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
    '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__',
    '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
    '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
    '__rmul__', '__setattr__', '__str__']
```

Características

- Diccionarios {}
  - Arrays asociativos o mapas, indexados por una clave, la cual puede ser cualquier objeto Python, aunque normalmente es una tupla:

```
>>> mydict = {"altura" : "media", "habilidad" : "intermedia", "salario" :1000 }
>>> print mydict
{altura': 'media', 'habilidad': 'intermedia', 'salario': 1000}
>>> print mydict["habilidad"]
intermedia
```

  - Verificación de una clave en un diccionario usando has_key:

```
if mydict.has_key('altura'):
    print 'Nodo encontrado'
```

  - Lo mismo se podría hacer:

```
if 'altura' in mydict:
    print 'Nodo encontrado'
```


Características

- Control de Flujo: Condicionales

```
if condición:
    acciones
elif
    acciones
else
    acciones
```

  - *Ejemplo. Resolución ecuación de primer grado*

```
a= float(raw_input('Valor de a: '))
b= float(raw_input('Valor de b: '))
if a!= 0:
    x= -b/a
    print 'Solución: ',x
if a== 0:
    if b!= 0:
        print 'La ecuación no tiene solución.'
if b== 0:
        print 'La ecuación tiene infinitas soluciones.'
```

Características

- Control de Flujo: Condicionales
  - E.j. (condicional.py)

```
q = 4
h = 5
if q < h :
  print "primer test pasado"
elif q == 4:
  print “q tiene valor 4”
else:
  print "segundo test pasado"
>>> python condicional.py
primer test pasado
```

  - Operadores booleanos: "or," "and," "not"
  - Operadores relacionales: ==, >, <, !=

Características

- Control de Flujo: Bucles
  - for se utiliza para iterar sobre los miembros de una secuencia
    - Se puede usar sobre cualquier tipo de datos que sea una secuencia (lista, tupla, diccionario)
  - Ej. bucle.py

```
for x in range(1,5):
  print x
$ python bucle.py
1 2 3 4
```

  - La función range crea una secuencia descrita por ([start,] end [,step]), donde los campos start y step son opcionales. Start es 0 y step es 1 por defecto.

Características

- Control de Flujo: Bucles
  - while es otra sentencia de repetición. Ejecuta un bloque de código hasta que una condición es falsa.
  - break nos sirve para salir de un bucle
  - Por ejemplo:

```
reply = 'repite'
while reply == 'repite':
  print 'Hola'
  reply = raw_input('Introduce "repite" para hacerlo de nuevo: ')
```

```
Hola
Introduce "repite" para hacerlo de nuevo: repite
Hola
Introduce "repite" para hacerlo de nuevo: adiós
```


Características

- Funciones
  - Una función se declara usando la palabra clave def

```
# funcionsimple.py
def myfunc(a,b):
  sum = a + b
  return sum
print myfunc (5,6)
$ python funcionsimple.py →11
```

  - A una función se le pueden asignar parámetros por defecto:

```
# funcionvaloresdefecto.py
def myfunc(a=4,b=6):
  sum = a + b
  return sum
print myfunc()
print myfunc(b=8) # a es 4, sobreescribir b a 8
$ python funcion.py → 10  12
```

Características

- Funciones
  - Listas de argumentos y argumentos basados en palabras clave:

```
# funcionargumentosvariablesyconclave.py
def testArgLists_1(*args, **kwargs):
  print 'args:', args
  print 'kwargs:', kwargs
testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')
Mostrará: args: ('aaa', 'bbb') kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
def testArgLists_2(arg0, *args, **kwargs):
  print 'arg0: "%s"' % arg0
  print 'args:', args
  print 'kwargs:', kwargs
print '=' * 40
testArgLists_2('un primer argumento', 'aaa', 'bbb', arg1='ccc', arg2='ddd')
```

Mostrará arg0: "un primer argumento“ args: ('aaa', 'bbb') kwargs: {'arg1': 'ccc', 'arg2‘:'ddd'}

Características

- Ficheros.
  - Lectura de ficheros (leerfichero.py)

```
fh = open("holamundo.py") # open crea un objeto de tipo fichero
for line in fh.readlines() : # lee todas las líneas en un fichero
  print line,
fh.close()
$ python leerfichero.py
#!/usr/bin/python
print "Hola mundo“
```

  - Escritura de ficheros(escribirfichero.py)

```
fh = open("out.txt", "w")
fh.write ("estamos escribiendo ...\n")
fh.close()
$ python escribirfichero.py
$ cat out.txt
 estamos escribiendo ...
```

Características

- La orden print
  - **print** (printredirect.py)
  - stdout en Python es sys.stdout, stdin es sys.stdin:

```
import sys
class PrintRedirect:
    def __init__(self, filename):
        self.filename = filename
    def write(self, msg):
        f = file(self.filename, 'a')
        f.write(msg)
        f.close()
sys.stdout = PrintRedirect('tmp.log')
print 'Log message #1'
print 'Log message #2'
print 'Log message #3'
```


Características

- Variables Globales
  - Usar identificador global para referirse a variable global:

```
# variableglobal.py
NAME = "Manzana"
def show_global():
  name = NAME
  print '(show_global) nombre: %s' % name
def set_global():
  global NAME
  NAME = 'Naranja'
  name = NAME
  print '(set_global) nombre: %s' % name
show_global()
set_global()
show_global()
```

  - Resultado: (show_global) nombre: Manzana (set_global) nombre: Naranja
    (show_global) nombre: Naranja

Características

- Expresiones Regulares
- A través del módulo re, Python permite el uso de expresiones regulares similares a como se hace en Perl (una razón más para moverse de Perl a Python)

```
# regex/procesaUrlConRe.py
import re, urllib, sys
if len(sys.argv) <= 4:
    print "Usage: procesaUrl <url-a-procesar> <palabra-a-reemplazar> <nueva-palabra>
      <fichero-html-a-crear>"
    sys.exit(0)
print sys.argv[1]
s = (urllib.urlopen(sys.argv[1])).read() # lee el contenido de una url
# reemplaza todas las ocurrencias de "Artaza" por "artaza"
t = re.sub(sys.argv[2], sys.argv[3], s)
backupFile = open(sys.argv[4], "w")
backupFile.write(t)
backupFile.close()
print 'Fichero ' + sys.argv[4] + ' escrito con contenido de url: ' + sys.argv[1] + ' al
    reemplazar palabra ' + sys.argv[2] + ' con palabra ' + sys.argv[3]
```


Características

- Expresiones Regulares (Ejemplo)

```
# conseguir el titulo del documento HTML
tmatch = re.search(r'<title>(.*?)</title>', s, re.IGNORECASE)
if tmatch:
    title = tmatch.group(1)
    print 'Titulo de pagina ' + sys.argv[1] + ' es: ' + title

# extraer lista de enlaces url:
pat = re.compile(r'(http://[\w-]*[.\w-]+)')
addrs = re.findall(pat, s)

print 'La lista de enlaces encontrados en esta pagina es: '
for enlace in addrs:
    print enlace
```

Características

- Excepciones.
- Cada vez que un error ocurre se lanza una excepción, visualizándose un extracto de la pila del sistema. E.j. excepcion.py:

  ```
  #!/usr/bin/python
  print a
  $ python exception.py
  Traceback (innermost last): File "exception.py", line 2, in ? print a NameError: a
  ```

- Para capturar la excepción se usa except:

  ```
  try:
    fh=open("new.txt", "r")
  except IOError, e:
    print e
  $ python excepcion.py
  [Errno 2] No such file or directory: 'new.txt‘
  ```

- Personalización de excepciones: uso del comando raise:

  ```
  raise MyException
  raise SystemExitModules
  ```


Características

- Programación de Sistemas.
  - Python permite la programación de sistema tanto accediendo a la API de Windows (http://www.python.org/windows/index.html) como a las llamadas al sistema de UNIX (módulo os)
  - El módulo os nos da acceso a:
    - El entorno del proceso: getcwd(), getgid(), getpid()
    - Creación de ficheros y descriptores: close(), dup(), dup2(), fstat(), open(), pipe(), stat(), socket()
    - Gestión de procesos: execle(), execv(), kill(), fork(), system()
    - Gestión de memoria mmap()
  - El módulo threading permite la creación de threads en Python

Características

- Ejemplo threads

```
#!/usr/bin/env python
import threading # threading/ejemplothreading.py
import urllib
class FetchUrlThread(threading.Thread):
  def __init__(self, url, filename):
    threading.Thread.__init__(self)
    self.url = url
    self.filename = filename
  def run(self):
    print self.getName(), "Fetching ", self.url
    f = open(self.getName()+self.filename, "w")
    content = urllib.urlopen(self.url).read()
    f.write(content)
    f.close()
    print self.getName(), "Saved in ", (self.getName()+self.filename)
urls = [ ('http://www.python.org', 'index.html'),
      ('http://www.google.es', 'index.html') ]
# Recuperar el contenido de las urls en diferentes threads
for url, file in urls:
  t = FetchUrlThread(url, file)
  t.start()
```


JYTHON

- Funcionalidades
  - Jython (http://www.jython.org) es una implementación open source en Java de Python que se integra de manera transparente con la plataforma Java.
  - Jython complementa a Java y es especialmente indicada para las siguientes tareas:
    - Empotrar scripts en aplicaciones Java, de modo que los usuarios finales puedan escribir scripts que añadan funcionalidad a la aplicación.
    - Experimentación interactiva por medio del intérprete interactivo suministrado por Jython que permite interactuar con los paquetes Java o aplicaciones en ejecución.
    - Desarrollo rápido de aplicaciones. Los programas en Python son típicamente entre 2 y 10 veces más cortos que los programas Java. Esto se traduce en una mayor productividad en la programación. La integración transparente de Jython y Java permite combinar estos dos lenguajes en productos.

Jython

- Documentación
  - Download Jython de:
    - http://www.jython.org/download.html
  - Para instalar simplemente ejecutar: `java jython-21`
    - Usa Lift-Off Java-Installer: http://liftoff.sourceforge.net/
  - Algunos ejemplos de Jython en:
    - http://www.jython.org/applets/index.html
  - Documentación básica sobre Jython disponible en:
    - http://www.jython.org/docs/usejava.html

IronPython

- Funcionalidades
  - En Marzo del 2004 en la conferencia PyCon fue presentado IronPython (http://ironpython.com/)
    - Una implementación Python orientada a las plataformas .NET y Mono.
  - Sus características principales son:
    - Tan rápida como la versión estándar de Python
    - Integrada de modo transparente con la Common Language Runtime de .NET. Desde IronPython podemos usar las librerías de clases .NET y extender otras clases .NET
    - Dinámico, soporta el modo de ejecución interactivo como Python
    - Opcionalmente estático, se pueden compilar ficheros Python para producir ejecutables (.exes) que pueden ejecutarse directamente o incluso dlls.
    - Soporta código gestionado (managed code)
    - No finalizado, todavía en versión alfa no puede ser utilizada para producción de software.
  - IronPython parece una contribución prometedora que nos permitirá programar aplicaciones .NET desde la cómoda y sencilla sintaxis de Python.

Uso de Python

- BitTorrent (http://bitconjurer.org/BitTorrent/), sistema P2P que ofrece mayor rendimiento que eMule
- PyGlobus, permite la programación de Grid Computing (http://www-itg.lbl.gov/gtg/projects/pyGlobus/)
- ZOPE (www.zope.org) es un servidor de aplicaciones para construir y gestionar contenido, intranets, portales, y aplicaciones propietarias
- Industrial Light & Magic usa Python en el proceso de producción de gráficos por ordenador
- Google usa Python internamente, lo mismo que Yahoo para su sitio para grupos
- Red Hat Linux utiliza Python para la instalación, configuración, y gestión de paquetes.
- Más historias de éxito de Python en: http://pbf.strakt.com/success

Bibliografía

- Python for Unix and Linux System Administration Jason Brittain and Ian F. Darwin O'REILLY ISBN: 978-0-596-51582-9
- PYTHON PARA ADMINISTRACION DE SISTEMAS UNIX Y LINUX. NOAH GIFT, JEREMY M. JONES, ANAYA MULTIMEDIA, 2009 ISBN 9788441525405
- http://www.ceibal.edu.uy/contenidos/areas_conocimiento/aportes/python_para_todos.pdf