

# Introducción a la programación para Ciencia de Datos: Python

Miguel García Silvente

## Esquema

- Cuestiones básicas de Python
- Funciones
- Tipos de datos básicos
- Estructuras de control
- Tipos de datos más complejos
- Ficheros

# Cuestiones importantes

- Algunos aspectos son muy importantes, como la precisión:

¿Cuál es el resultado?

$$0.3 - (0.1 + 0.1 + 0.1)$$

$$-5.551115123125783e-17$$

- Generación de números aleatorios
- Aproximación de funciones
- Enteros de gran tamaño

Prof.Miguel García Silvente

3

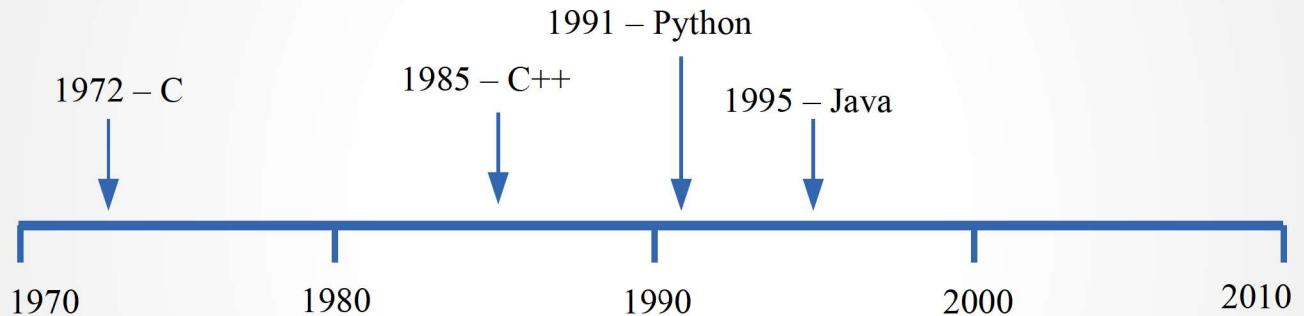
# Métodos y algoritmos

- Análisis y visualización de datos.
- Modelado y simulación.
- Análisis numérico (aproximación numérica).
- Series de Taylor.
- Cálculo de derivadas (diferenciación automática, diferencias finitas).
- Métodos de integración.
- Resolución de ecuaciones diferenciales.
- Métodos probabilísticos.
- etc

Prof.Miguel García Silvente

4

# Historia de los Lenguajes de Programación



Prof.Miguel García Silvente

5

## Lenguaje Script

- Python es un lenguaje de tipo script.
- Un script es un programa de alto nivel que suele ser corto.
- Otros lenguajes script: Unix shells, Tcl, Perl, Python, Ruby, Scheme, Rexx, JavaScript, VisualBasic, ...

Prof.Miguel García Silvente

6

# Ventajas de los lenguajes Script

- Interpretado, no es necesario compilar.
- Desarrollo del software más rápido.
- No hay que declarar las variables.
- Gran cantidad de bibliotecas y herramientas.

Prof.Miguel García Silvente

7

## Programas más cortos

- Supongamos que queremos leer los siguientes datos:

1.1 9 5.2

1.762543E-02

0 0.01 0.001

9 3 7

El código en python sería:

`F = open(filename, 'r')`

`n = F.read().split()`

Prof.Miguel García Silvente

8

# Algunas características de Python

- Multiplataforma.
- Manejo de matrices. **Numpy**
- Representación de datos. **Matplotlib**
- Funciones de cálculo científico. **Scipy**
- Cálculo simbólico. **Sympy**
- Evaluar expresiones: **eval**.

## Introducción a Python

- Historia
- Instalación y ejecución
- Tipos de datos

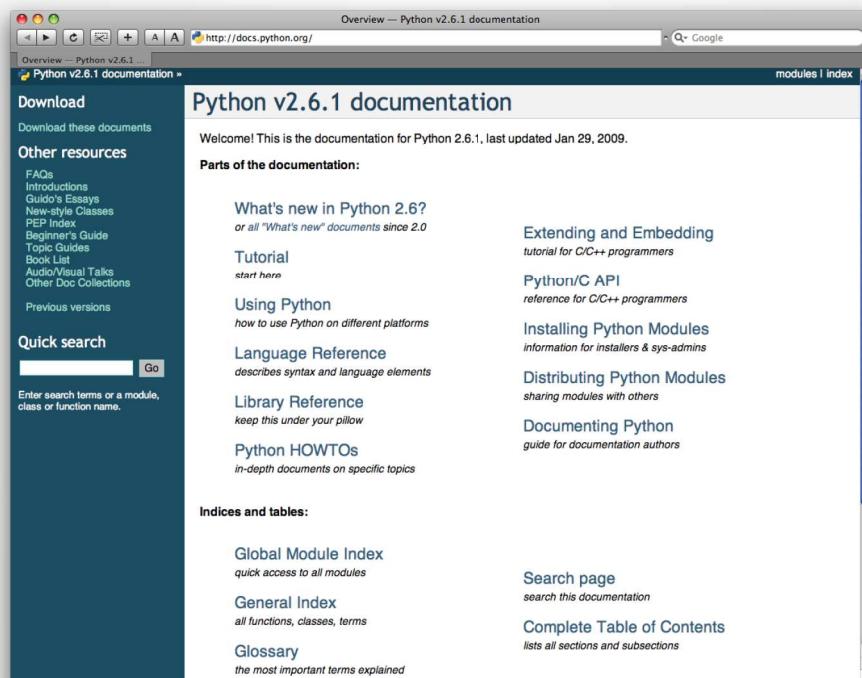
# Breve historia de python

- Fue creado en Holanda en el año 1991 por Guido van Rossum.
- Su nombre se debe a los Monty Python.
- Open source desde el inicio.
- Considerado un lenguaje script.
- Escalable, orientado a objetos y funcional desde su diseño inicial.
- Mayoritariamente usado por muchas empresas, por ejemplo: youtube, Google, NASA, etc.
- Implementado en C.

Prof.Miguel García Silvente

11

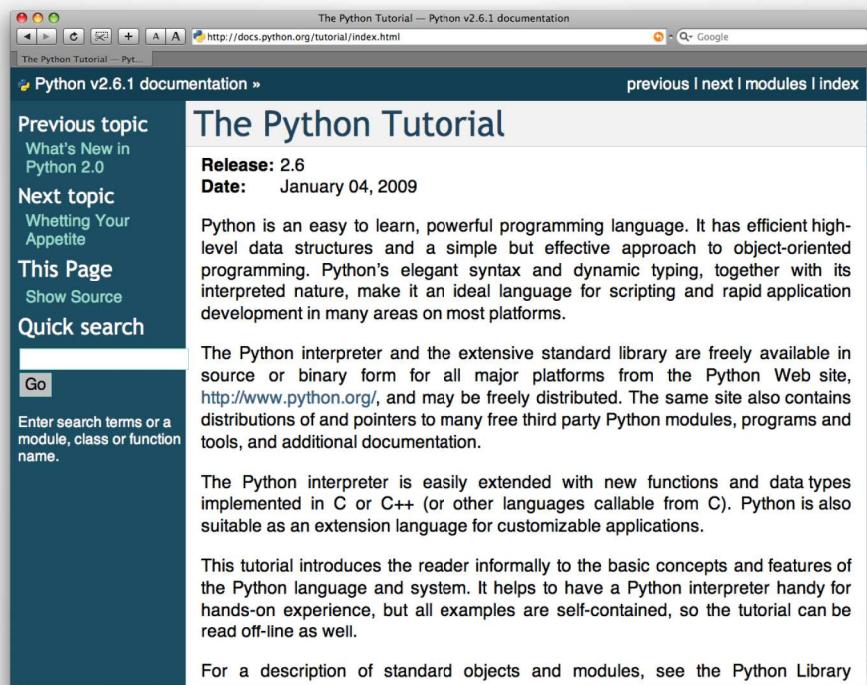
## http://docs.python.org/



Prof.Miguel García Silvente

12

# Tutorial de Python



Prof.Miguel García Silvente

13

## Incompatibilidad entre versiones

- Las versiones 2 y 3 son incompatibles en muchos aspectos.
- Por ejemplo:

- v3:

```
>>> 2/3
```

```
0.666666666666666
```

```
>>> print x # Error, hay que usar ()
```

- v2:

```
>>> 2/3
```

```
0
```

Prof.Miguel García Silvente

14

# Ejecución interactiva

>>> es el prompt de Python e indica que Python está preparado para que le demos una orden.

```
>>> "Hello, world"  
Hello, world  
>>> print("Hello, world")  
Hello, world  
>>> 2+3  
5  
>>> print(2+3)  
5  
>>> print("2+3=", 2+3)  
2+3= 5  
>>>
```

Prof.Miguel García Silvente

15

# Ejecución interactiva en UNIX

```
% python  
>>> 3+3  
6  
% python3
```

- Para salir:
  - En Unix, escribe CONTROL-D
  - En Windows, escribe CONTROL-Z + <Enter>
  - También con exit()

Prof.Miguel García Silvente

16

# Instalación

- Python está preinstalado en muchos sistemas unix, incluyendo Linux y MAC OS X
- Las versiones preinstaladas pueden no ser las más recientes (2.6 es de Nov 2008)
- Descargar de <http://python.org/download/>
- Distribuciones: Python(x,y), Winpython, Anaconda
- Python incluye una biblioteca con módulos estandar.
- IDEs:
  - Ninja IDE, Spyder
  - Eclipse with Pydev (<http://pydev.sourceforge.net/>)

Prof.Miguel García Silvente

17

# Cuestiones básicas I

- No existe un **main()** o función principal
- Se puede solicitar ayuda con **help(x)**
- Los bloques se inician con : y se determinan con la indentación.
- Es sensible a mayúsculas y minúsculas.
- Gestión de memoria: usa un recolector de basura (basado en referencias).

Prof.Miguel García Silvente

18

## Cuestiones básicas II

- Se ejecuta un archivo desde el inicio hasta el fin.
- No es necesario usar ; al final de cada sentencia.
- Las variables no se declaran. Se crean al asignarle un valor.
- El tipo de dato es dinámico y se puede conocer con `type()`

## Cuestiones básicas III

- La extensión para los archivos suele ser .py
- Las extensiones .pyc .pyo son versiones “compiladas”
- Los comentarios se indican con #
- Los módulos se incorporan con `import`
- Posee una gran cantidad de módulos.
- Existe una guía de estilo `PEP-8`

# Uso como calculadora

- La definición de una función es muy simple:

```
>>> 3 + 2
```

5

- Se puede reutilizar el último resultado:

```
>>> 3 + 2
```

5

```
>>> 7 * _
```

35

- `import math` para incorporar funciones matemáticas

# Ejecución de programas en UNIX

- Ejecutar el script con el intérprete

```
% python fact.py
```

- Convertir el fichero en ejecutable

- Añadiendo el path de python como la primera línea del archivo

```
#!/usr/bin/python3
```

- Darle permisos de ejecución

```
% chmod a+x fact.py
```

- Invocar el fichero desde la línea de órdenes

```
% ./fact.py
```

# Eficiencia

- Python es un lenguaje interpretado.
  - Más lento que un lenguaje compilado C/C++.
  - Más flexible en el diseño y la codificación.
- Las bibliotecas están escritas en C/C++.
- Tiene módulos para medir la eficiencia.
  - Profile.
  - Cprofile.
- Es posible mejorar la eficiencia, por ej. usando Cython.

Prof.Miguel García Silvente

23

## Asignación (I)

- Se asocia un nombre a una referencia a un objeto
  - *Las asignaciones crean referencias, no copias.*
- Una variable se crea la primera vez que aparece en la parte izquierda de una asignación:

```
>>> x = 5          >>> x = 7  
>>> id(x)        >>> y  
505910928         ????  
>>> y = x  
>>> id(y)  
505910928
```

Prof.Miguel García Silvente

24

# Asignación (II)

- Asignar valores a más de una variable:

```
>>> x, y = 2, 3  
>>> x  
2  
>>> y  
3
```

Facilita el intercambio de valores entre variables:

```
>>> x, y = y, x
```

- Las asignaciones devuelven valores

```
>>> a = b = x = 2
```

Prof.Miguel García Silvente

25

# Acceso a una variable inexistente

Se genera un error

```
>>> y
```

```
Traceback (most recent call last):  
  File "<pyshell#16>", line 1, in -toplevel-  
    y  
NameError: name 'y' is not defined  
>>> y = 3  
>>> y  
3
```

Prof.Miguel García Silvente

26

# Nombres de variables

- Son sensibles a mayúsculas y minúsculas y no pueden empezar por un número. Pueden contener letras, números y caracteres subrayado.

bob Bob \_bob \_2\_bob\_ bob\_2 BoB

- Palabras reservadas:

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

# Convenciones

La comunidad Python recomienda las siguientes:

- Para funciones, métodos y atributos usar letras minúsculas y usar el carácter subrayado \_
- Usar todo mayúsculas para las constantes.
- Primera letra en mayúscula para las clases
- Atributos: interface, \_internal, \_\_private

# Funciones (I)

- Se usa la palabra reservada **def**
- La definición de una función es muy simple:

```
def saludo(nombre):  
    print("Hola", nombre)  
    print("¿Cómo estás?")
```

- Se podría usar con datos distintos:

```
saludo("Pepe")  
saludo(3452)
```

# Funciones (II)

- Si no hay **return**, la función devuelve **None**
- Los parámetros se pasan como referencias por valor.
- No es necesario especificar tipos:

```
def sumar(x, n) :  
    return x+n
```

- Se pueden devolver varios valores al mismo tiempo:

```
def cociente_resto(a, b) :  
    return a/b, a%b
```

```
x, y = cociente_resto(12, 5)
```

# Funciones: documentación

Se documenta usando """

```
def fact(n) :  
    """fact(n) asume que n es un entero positivo y  
    devuelve el factorial de n.""""  
    assert(n>0)  
    ...
```

Se puede consultar la documentación con

```
fact.__doc__
```

# Funciones: swap

- Correcta

```
def swap(a, b) :  
    return b, a
```

- Incorrecta

```
def swap_incorrecto(a, b) :  
    temp = a  
    a = b  
    b = temp
```

# Funciones: alias y parámetros por defecto

- Se pueden usar alias

f = swap

b, a = f(a,b) :

- Parámetros por defecto

def func(a, b = 10) :

    return a + b

print (f(6))

- Uso de nombres de parámetros

func(b=12, a= 10)

func(10,12)

Prof.Miguel García Silvente

33

# Funciones lambda

- Una forma abreviada
- <función> = lambda <params> : <valores devueltos>

>>> suma = lambda x, y : x+y

>>> suma(10, 15)

>>> f = lambda : 1

Prof.Miguel García Silvente

34

# Tipos de datos básicos

- int/long
- bool
- float
- complex
- str

Prof.Miguel García Silvente

35

## Tipos de datos: int

- Precisión ilimitada en decimal, octal (0o) y hexadecimal (0x)
- Operaciones: = + - / // \* % \*\* divmod abs int
- Operaciones bit a bit: | & ^ ~ << >> bin  
bit\_length

```
>>> x = 34
```

```
>>> x
```

```
34
```

```
>>> x / 5
```

```
6.8
```

Prof.Miguel García Silvente

36

## Tipos de datos: bool

- Valores lógicos True False
- False se asocia al 0 y True a los demás valores
- Operadores: = and or not

```
>>> x = True
```

```
>>> x
```

```
True
```

```
>>> x = true
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'true' is not defined
```

## Tipos de datos: float

- Se representan en base 2. Se pierde precisión si no es exacta su representación en base 2, por ejemplo, 0.1

```
>>> 0.1 + 0.1 + 0.1 == 0.3
```

- Operaciones: = + - / // \* % \*\* divmod abs, int
- Operaciones bit a bit: | & ^ ~ << >> float bin  
x.bit\_length()

## Tipos de datos: complex

- La parte imaginaria se indica con j  
`>>> x = 3 + 2j`
- Operaciones: = + - / \* % \*\* abs imag real  
// (python2)

## Tipos de datos: str (I)

- Son tipos inmutables
- Se pueden usar ' o "
- Operadores [] +
- Se usa """ cuando la cadena ocupa varias líneas.

`x = """cadena  
multilínea"""`

## Tipos de datos: str (II)

- Funciones: len, lower, lstrip, replace, split, swapcase, upper, count, find, join, partition, str (o repr)
- La barra \ hay que “escaparla” con otra \  
`cad = "\windows"`
- Una forma de evitarlo es  
`cad = r"\windows"`

Prof.Miguel García Silvente

41

## Tipos de datos: str (III)

### Operador []

```
>>> cad = "Hola mundo"  
>>> cad [0:3]  
'Hol'  
>>> cad[3:-1]  
'a mund'  
>>> cad[-1:0:-1]  
'odnum alo'
```

Prof.Miguel García Silvente

42

## Tipos de datos: str (IV)

### Operador []

```
>>> cad[:]  
'Hola mundo'  
>>> cad[::-1]  
'odnum aloH'
```

## Tipos de datos: str (V)

### Conversión de tipo

```
>>> cad = str(1/5)  
>>> cad = "El resultado de dividir " + str(x) + "  
entre " + str(y) + " es " + str(x/y)
```

# Tipos de datos dinámicos (duck typing)

- El tipo de cada variable se determina de forma dinámica
- El tipo de dato se consulta con type()

```
>>> x = "1234"
```

```
>>> type(x)
```

```
<type 'str'>
```

```
>>> x = 34
```

```
>>> type(x)
```

```
<type 'int'>
```

Prof.Miguel García Silvente

45

# Tipos de datos dinámicos (duck typing) II

- Asignación de tipo dinámico débil o fuerte

- Perl (débil):

```
$b = '1.2'
```

```
$c = 5*$b; # conversión implícita
```

```
# de tipo: '1.2' -> 1.2
```

- Python (fuerte):

```
b = '1.2'
```

```
c = 5*b
```

```
# es ilegal porque no hay conversión implícita
```

Prof.Miguel García Silvente

46

# E/S y estructuras de control

- Operaciones de E/S
- Sentencia condicional
- Bucles

## Entrada estándar

- Se utiliza input (en versión 2.x era raw\_input)  
`x = input("Num.datos: ")`
- x sería de tipo string, para que sea int:  
`x = int(input("Número:"))`

# Salida estándar (I)

- Se utiliza `print`
- Se puede usar de varias formas:

```
print ('el cuadrado de', x, 'es', x * x)
```

```
print('el cuadrado de {} es {}'.format(x, x**2))
```

```
print('el cuadrado de %d es %d' % (x, x**2))
```

# Salida estándar (II)

- Por defecto: se separan con espacios y cada `print` va a una línea.

```
>>> print('esto','es', 'un', 'test')
```

```
esto es un test
```

- Modificadores:

- para evitar que se separen por un espacio:

```
>>> print('esto','es', 'un', 'test', sep="")
```

- para evitar el final de línea:

```
estoesuntest
```

```
>>> print(item, " ", end=""); print ("2")
```

```
2 2
```

## Salida estándar (III)

- Uso de format

```
print('los datos son {} es {}'.format(x, x**2))
```

```
print('los datos son {1} es {0}'.format(x, x**2))
```

```
print('el cuadrado de {num1} es  
{num2}'.format(num1=x, num2=x**2))
```

## Sentencia condicional: if else

- La cláusula else es opcional:

```
if x< 0 :
```

```
    print("Negativo")
```

```
else :
```

```
    print("Positivo")
```

# Sentencia condicional: if elif

- Se pueden encadenar:

```
if x < 0 : print ("Negative")
```

```
elif x==0 :
```

```
    print("cero")
```

```
else :
```

```
    print ("Positivo")
```

## Función lambda con if else

```
>>> x = 100
```

```
>>> resultado = (-1 if x < 0 else 1)
```

```
>>> print (resultado)
```

```
1
```

```
>>> fact = lambda n : 1 if n==1 or n==0 else  
n*fact(n-1)
```

```
>>> fact(5)
```

```
120
```

# Bucles for (I)

- Se itera sobre una serie de elementos:

```
for l in (12, 45, 23, 9, 12, 20) :
```

```
    print (l)
```

- Se pueden generar con **range**:

```
for l in range(20) :
```

```
    print (l)
```

```
for r in range(1, 5, 2) :
```

```
    print (r)
```

Prof.Miguel García Silvente

55

# Bucles for (II)

- Se puede ejecutar código al final:

```
for x in (1,2,3,4,5,6) :
```

```
    print (x)
```

```
else:    print ("bucle terminado")
```

```
for x in (1,2,3,4,5,6):
```

```
    print (x)
```

```
    if x > 3 : break
```

```
else:    print (" bucle llega hasta el final")
```

Prof.Miguel García Silvente

56

## Bucles for (III)

- ¿Qué ocurre en el siguiente caso?

**for i in range(12) :**

**print(i)**

**i += 2**

## Iterar con varias variables: zip

- Sobre pares:

**for x,y in zip(range(12), range(100, 120)) :**

**print (x, y)**

- O tripletes:

**r = zip(range(20), range(12), (23, 45, 16))**

**for x, y, z in r :**

**print(x,y,z)**

# Bucles while

- Se termina cuando se da una condición:

`x = 1`

`while x < n :`

`x = x * 2`

# Tipos de datos más complejos

- tuple
- list
- dict
- set
- array

# Secuencias: Tuplas, Listas y Strings

Prof.Miguel García Silvente

61

## Tipos de secuencias

### 1. Tupla (*tuple*)

- Una secuencia *immutable* y ordenada de elementos
- Los elementos pueden ser de distinto tipo, incluyendo tipos compuestos

### 2. Cadena de caracteres (*str*)

- *Immutables*
- Similares a una tupla

### 3. Lista (*list*)

*Secuencia ordenada y mutable de elementos de distintos tipos.*

*(en python 3.x existe también el tipo range)*

Prof.Miguel García Silvente

62

# Tuplas, listas,strings I

- Las tuplas se definen usando comas (y paréntesis)  
`>>> tu = (23, 'abc', 4.56, (2,3), 'def')`
- Las listas usando corchetes y separando con comas  
`>>> li = ["abc", 34, 4.34, 23]`
- Las cadenas usando comillas (" , ' or """).

```
>>> st = "Hello World"  
>>> st = 'Hello World'  
>>> st = """Esta es una cadena con  
varias líneas que usacommillas triples."""
```

# Tuplas, listas,strings II

- Acceso a un elemento mediante corchetes
- *El primer elemento es el 0*

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')  
>>> tu[1]      # segundo item de la tupla.  
'abc'  
  
>>> li = ["abc", 34, 4.34, 23]  
>>> li[1]      # Segundo item de la lista.  
34  
  
>>> st = "Hello World"  
>>> st[1]      # Segundo carácter del string.  
'e'
```

# Índices positivos y negativos

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Índice **positivo**: desde la izquierda, empezando en 0

```
>>> t[1]
```

```
'abc'
```

Índice **negativo**: empezando por la derecha,  
empezando con -1

```
>>> t[-3]
```

```
4.56
```

## Trocear (I)

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

- Devuelve una copia de un contenedor compuesto por un subconjunto de los miembros. Empieza en el primer índice y termina antes del segundo:

```
>>> t[1:4]
```

```
('abc', 4.56, (2,3))
```

- Se pueden usar índices negativos:

```
>>> t[1:-1]
```

```
('abc', 4.56, (2,3))
```

## Trocear (II)

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

- Si se omite el primer índice se empieza en el primer elemento:

```
>>> t[:2]
```

```
(23, 'abc')
```

- Si se omite el segundo se llega hasta el último elemento:

```
>>> t[2:]
```

```
(4.56, (2,3), 'def')
```

## Copiar la secuencia completa

- [ :] hace una copia de la secuencia completa

```
>>> t[:]
```

```
(23, 'abc', 4.56, (2,3), 'def')
```

- Dos casos para objetos mutables:

```
>>> l2 = l1 # l2 es una referencia a l1,  
          # si se cambia uno, afecta al otro  
>>> l2 = l1[:] # copias independientes, 2  
referencias
```

# Valores nulos

- Listas:

```
>>> t = []
```

- Strings:

```
>>> cad = ""
```

- Tuplas:

```
>>> t = ()
```

# El operador ‘in’

- Comprueba si un valor aparece en un contenedor:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- En las cadenas, comprueba subcadenas

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

# El operador +

- El operador + produce una nueva tupla, lista o cadena concatenando los argumentos

```
>>> (1, 2, 3) + (4, 5, 6)  
(1, 2, 3, 4, 5, 6)
```

```
>>> [1, 2, 3] + [4, 5, 6]  
[1, 2, 3, 4, 5, 6]
```

```
>>> "Hello" + " " + "World"  
'Hello World'
```

Prof.Miguel García Silvente

71

# El operador \*

- Produce una nueva tupla, lista o cadena repitiendo el contenedor original.

```
>>> (1, 2, 3) * 3  
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

```
>>> [1, 2, 3] * 3  
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> "Hello" * 3  
'HelloHelloHello'
```

Prof.Miguel García Silvente

72

# Mutabilidad: Listas vs tuplas

Prof.Miguel García Silvente

73

## Las listas son mutables

```
>>> li = ['abc', 23, 4.34, 23]  
>>> li[1] = 45  
>>> li  
['abc', 45, 4.34, 23]
```

- Se *pueden cambiar directamente*.
- *li* apunta a la misma referencia de memoria cuando terminamos.

Prof.Miguel García Silvente

74

# Las tuplas son inmutables

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')  
>>> t[2] = 3.14
```

```
Traceback (most recent call last):  
  File "<pyshell#75>", line 1, in -toplevel-  
    tu[2] = 3.14  
TypeError: object doesn't support item assignment
```

- No es posible cambiar una tupla.
- *La inmutabilidad permiten que sean más rápidas que las listas.*

## Inmutabilidad vs mutabilidad

¿Qué ocurre en este caso?

```
>>> t = (1, 2, [3, 4])  
>>> t[2][1] = 5  
>>> t  
(1, 2, [3, 5])
```

# Paso de parámetros mutables

¿Qué ocurre en este caso?

```
def f(v, i) :  
    v[i] += 1
```

```
>>> aux = [1,2,3,4]  
>>> f(aux, 2)  
>>> aux  
[1, 2, 4, 4]
```

Prof.Miguel García Silvente

77

# Operaciones sobre listas

```
>>> li = [1, 11, 3, 4, 5]
```

```
>>> li.append('a')  
>>> li  
[1, 11, 3, 4, 5, 'a']
```

```
>>> li.insert(2, 'i') # en la posición 2 habrá una i  
>>> li  
[1, 11, 'i', 3, 4, 5, 'a']
```

Prof.Miguel García Silvente

78

# +, Extend, append

- `+` crea una lista nueva, con una nueva referencia a memoria
- `extend` opera directamente sobre la lista `li`.

```
>>> li.extend([9, 8, 7])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7]
```

- *Confusiones potenciales:*

- `extend` añade los elementos de la lista.
- `append` incluye un único elemento.

```
>>> li.append([10, 11, 12])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7, [10, 11, 12]]
```

Prof.Miguel García Silvente

79

## Operaciones sobre listas (I)

- Algunos ejemplos: `index`, `count`, `remove`, `reverse`, `sort`

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b') # índice de la 1a aparición
1
>>> li.count('b') # número de veces que aparece
2
>>> li.remove('b') # elimina la 1a aparición
>>> li
['a', 'c', 'b']
```

Prof.Miguel García Silvente

80

# Operaciones sobre listas (II)

```
>>> li = [5, 2, 6, 8]
```

```
>>> li.reverse() # invierte la lista  
>>> li  
[8, 6, 2, 5]
```

```
>>> li.sort()      # ordena la lista  
>>> li  
[2, 5, 6, 8]
```

```
>>> li.sort(key = <función>)  
# ordena la lista usando una comparación  
# definida por el usuario
```

```
cads = ['hola','12','a','abc']  
print (cads)  
cads.sort(key = lambda x: len(x))  
print (cads)
```

Prof.Miguel García Silvente

81

## Listas: inserción y borrado

```
>>> l = [1, 2, 3, 4, 5]
```

# Borrar elementos

```
>>> l[2:3] = []
```

# Insertar

```
>>> l[2:2] = [3]
```

```
>>> l[:0] = [7]
```

```
>>> del l[-1]
```

Prof.Miguel García Silvente

82

# Detalles del tipo tupla

- La coma es el operador de creación de una tupla (no es el paréntesis)

```
>>> 1,  
(1,)
```

- Python usa paréntesis por claridad

```
>>> (1,)  
(1,)
```

- No se debe olvidar la coma:

```
>>> (1)  
1
```

- Existen las tuplas vacías

```
>>> ()  
<empty tuple>  
>>> tuple()  
()
```

Prof.Miguel García Silvente

83

# Devolver valores en funciones (I)

- Se pueden devolver varios datos

```
def f(x) :  
    return x, x*x, x**x # o return (x,x*x,x**x)  
aux = f(7) # aux es de tipo tuple
```

```
a,b,c = f(5)  
a, b = f(12) # Error
```

# Devolver valores en funciones (II)

- También se puede hacer usando una lista:

```
def f(x) :  
    return [x, x*2, x**2]  
aux = f(7) # aux es de tipo list
```

```
a,b,c = f(5)  
a, b = f(12) # Error
```

## Resumen: Tuplas vs. Listas

- Las listas son más lentas pero más potentes:
  - las listas se pueden modificar,
  - tienen multitud de operaciones y métodos,
  - las tuplas son inmutables y tienen menos características.
- Se puede convertir de un tipo al otro con las funciones `list()` y `tuple()`:

```
li = list(tu)  
tu = tuple(li)
```