

Programación Técnica y Científica

Grado en Ingeniería Informática

Tema 1

Introducción a la programación en

Python

Esquema

- Introducción a la PTC
- Cuestiones básicas de Python
- Funciones
- Tipos de datos básicos
- Estructuras de control
- Tipos de datos más complejos
- Ficheros

Programación técnica y científica

- La programación científica está relacionada con la **construcción de modelos matemáticos** y **técnicas de análisis cuantitativo** usando ordenadores para **analizar y resolver** problemas **científicos y de ingeniería**.
- Es habitual la aplicación de **simulaciones** por ordenador y otras formas de computación a partir de **análisis numéricos** e **informática teórica** para resolver problemas de distintas disciplinas científicas y técnicas.

Aplicaciones

- Simulaciones numéricas:
 - reconstrucción y comprensión de eventos (terremotos, tsunamis, etc),
 - Predicciones de tiempo o del comportamiento de partículas subatómicas.
- Ajustes de modelos y análisis de datos:
 - Ajustar modelos o resolver ecuaciones (exploración de pozos petrolíferos),
 - Teoría de grafos para modelar redes.
- Optimización computacional

Ámbitos I

- Ingeniería aeroespacial e ingeniería mecánica.
- Biología y medicina
- Química.
- Ingeniería civil
- Ingeniería informática, ingeniería eléctrica e ingeniería en telecomunicaciones
- Ingeniería industrial
- Ciencia de los materiales

Ámbitos II

- Ingeniería nuclear: modelado de explosiones nucleares, simulación del proceso de fusión.
- Ingeniería petrolífera: modelado de reservas petrolíferas, exploración de petróleo y gas
- Predicción meteorológica, investigación climática, Computación geofísica (datos sísmicos)
- Simulación de campos de batallas y juegos militares.
- Sistemas astrofísicos.

Cuestiones importantes

- Algunos aspectos son muy importantes, como la precisión:

¿Cual 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

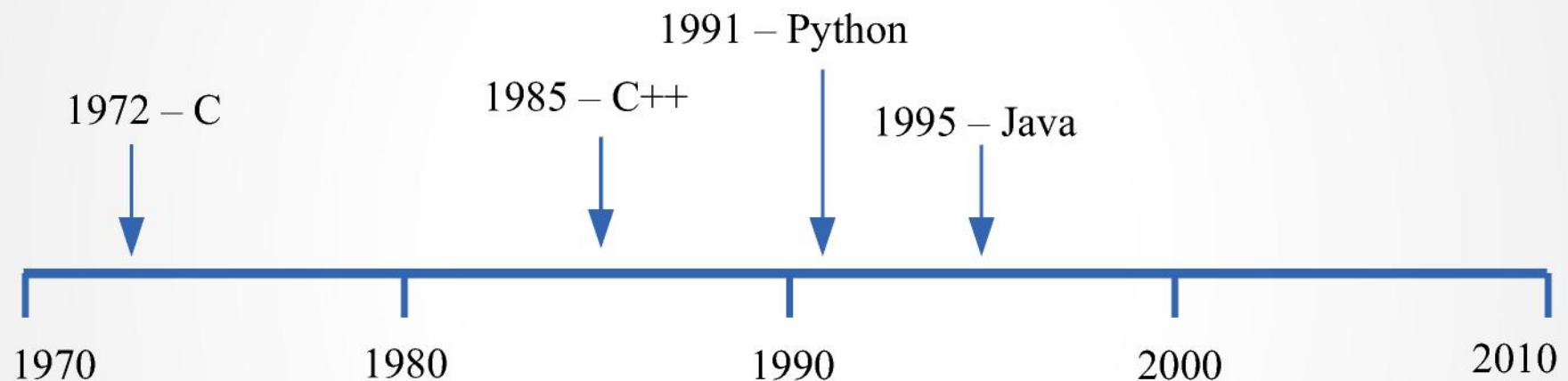
Métodos y algoritmos

- 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

Lenguajes de Programación Científica

- Python
- Matlab, Mathematica, SciLab, Octave
- R (computación estadística y gráfica)

Historia de los Lenguajes de Programación



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, ...

Ventajas de los lenguajes Script

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

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:

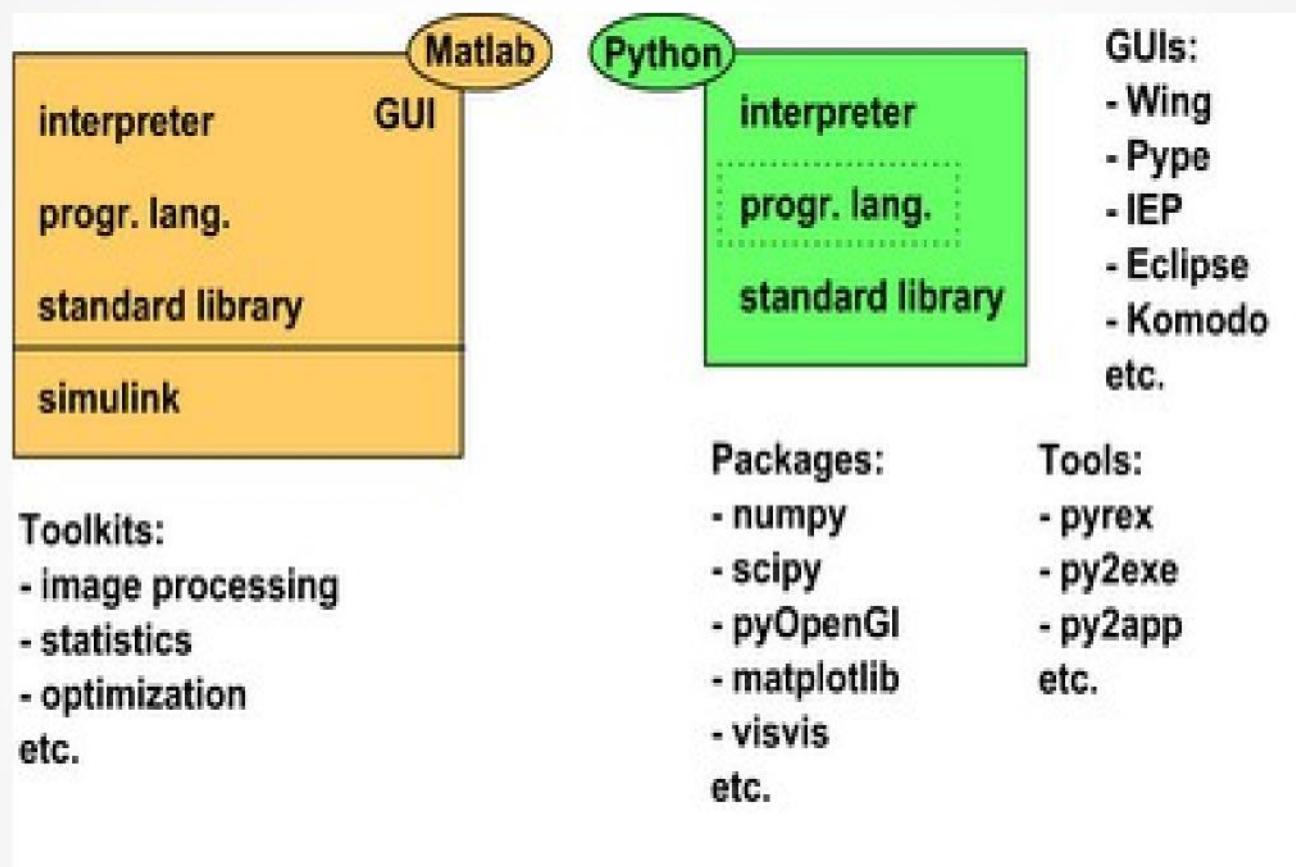
```
F = open(filename, 'r')
```

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

Algunas características de Python

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

Matlab vs Python I



<https://sites.google.com/site/pythonforscientists/python-vs-matlab>

Matlab vs Python II

Matlab:

- Código propietario.
- Caro.
- Portabilidad más difícil de realizar.
- Espacios de nombres (reciente).

Código Matlab vs Python I

Python:

- No hay que usar **end**
- No se usa () para las matrices, sino []
- Código un 10-20% más corto
- Los índices empiezan en 0
- Hay menos restricciones, por ej., se puede hacer:

```
>>> [1,2,4,7,11,14,16,17][n]
```

Código Matlab vs Python II

Python:

- Incluye diccionarios.
- POO más clara que en C++ y que en Matlab.
- Se puede combinar “main” con otras funciones en un mismo fichero.
- Organización del código
- Más variedad para programar GUIs

http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html

<http://www.stat.washington.edu/~hoytak/blog/whypython.html>

Código Matlab vs Python III

Python usa “propagación automática”

Dados A 20x1x15, B 20x12x1, C 1x12x15:

- En matlab

`D= bsxfun(@times, bsxfun(@minus,B,C), A);`

- En Python/NumPy

`D= A*(B-C)`

http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html
<http://www.stat.washington.edu/~hoytak/blog/whypython.html>

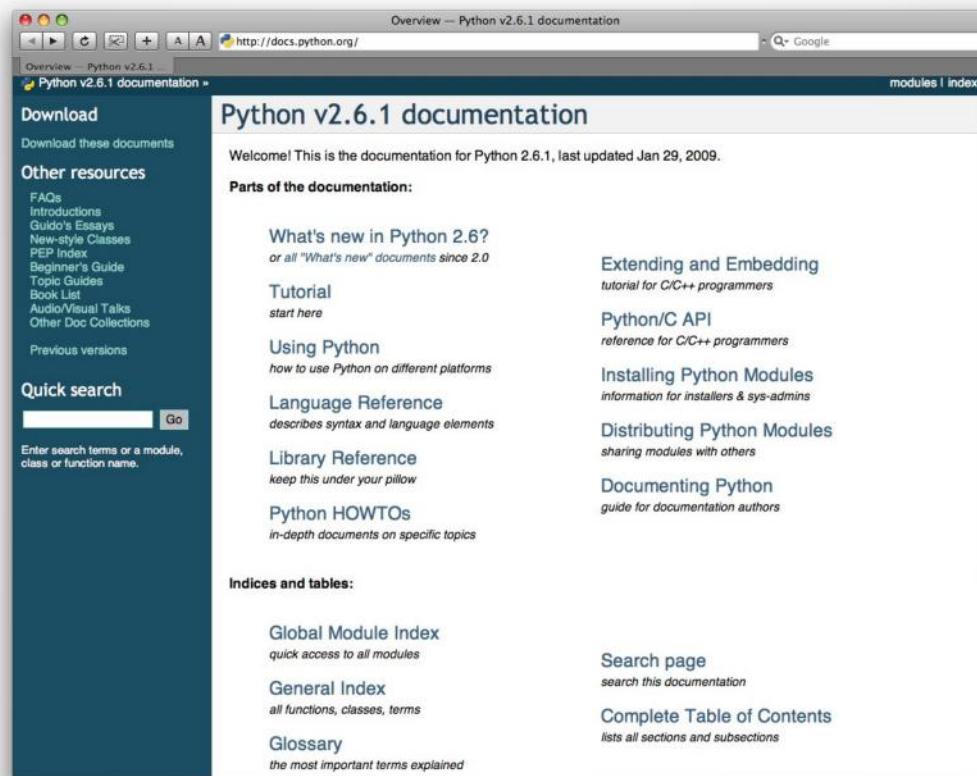
Introducción a Python

- Historia
- Instalación y ejecución
- Tipos de datos

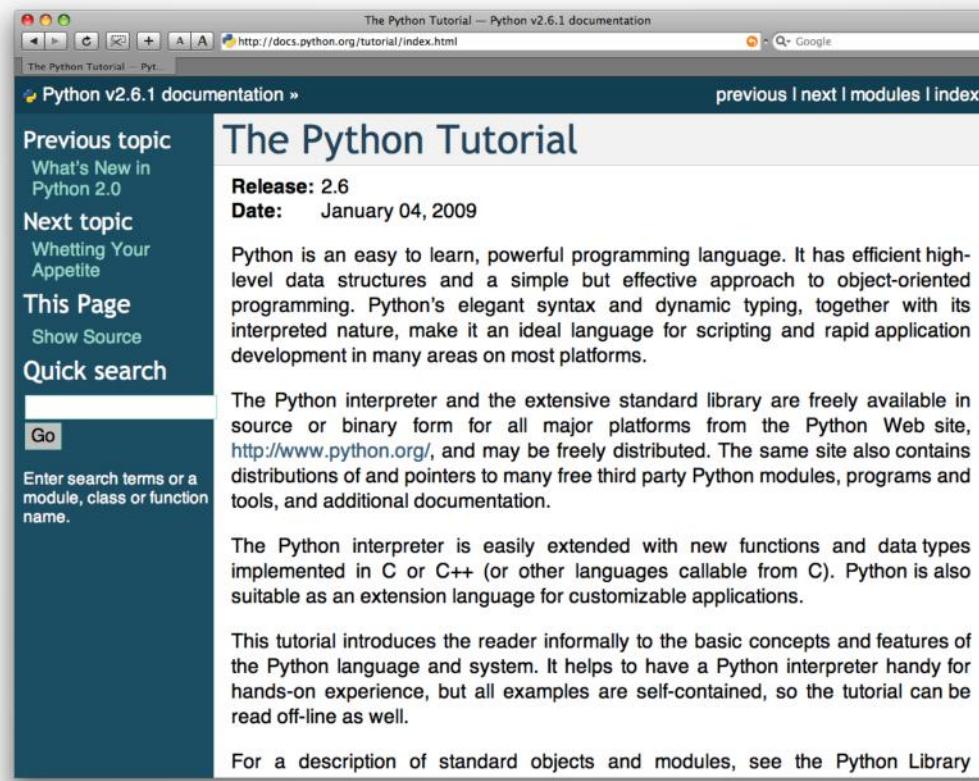
Breve historia de python

- 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

http://docs.python.org/



Tutorial de Python



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
```

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  
>>>
```

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()

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/>)

Cuestiones básicas I

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

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
```

- Previamente hay que usar `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

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 >>> id(x) 505910928 >>> y = x >>> id(y) 505910928	>>> x = 7 >>> y ????
--	--------------------------------

Asignación II

- Varias asignaciones al mismo tiempo:

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

```
>>> x
```

```
2
```

```
>>> y
```

```
3
```

Facilita el intercambio de variables

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

- Las asignaciones devuelven valores

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

Acceso a una variable inexistente

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
```

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)

Funciones lambda

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

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

```
>>> suma(10, 15)
```

```
>>> f = lambda : 1
```

Tipos de datos básicos

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

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
```

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
```

Prof.Miguel García Silvente

48

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 bit_length

Tipos de datos: complex

- La parte imaginaria se indica con j

```
>>> x = 3 + 2j
```

- Operaciones: = + - / // * % ** abs imag real

Tipos de datos: str

- Se pueden usar ' o “”
- Operadores [] +
- Se usa “”” cuando la cadena ocupa varias líneas.

x = “””cadena
multilínea”””

- Funciones: len, lower, lstrip, replace, split, swapcase, upper, count, find, join, partition, str (o repr)

Tipos de datos: str II

- Son tipos inmutables
- La barra \ hay que “escaparla” con otra \
`cad = “\\windows”`
- Una forma de evitarlo es
`cad = r”\\windows”`

Tipos de datos: str III

- Operador []

```
>>> cad = "Hola mundo"
```

```
>>> cad [0:3]
```

```
'Hol'
```

```
>>> cad[3:-1]
```

```
'a mund'
```

```
>>> cad[-1:0:-1]
```

```
'odnum alo'
```

Tipos de datos: str IV

- Operador []

```
>>> cad[:]
```

```
'Hola mundo'
```

```
>>> cad[::-1]
```

```
'odnum aloH'
```

Tipos de datos: str √

- 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'>
```

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`

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`
- `x = int(input("Número:"))`

Salida estándar

- 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

- Modificadores

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

esto es un test

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

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

- La cláusula else es opcional:

```
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

- 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)
```

Bucles for (II)

- Se itera sobre una serie de elementos:

```
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

Iterar con varias variables: zip

- Se itera sobre una serie de elementos:

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

```
r = zip(range(20), range(12), (23, 45, 16))  
for x, y, z in r :  
    print(x,y,z)
```

Bucles while

- Se itera sobre una serie de elementos:

x = 1

while x < n :

 x = x * 2

Tipos de datos más complejos

- tuple
- list
- dict
- set
- array

Secuencias: Tuplas y Listas

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)

Tuplas, listas y cadenas

- Las tuplas se definen usando paréntesis y comas

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Las listas usando corchetes y 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

- El acceso a un elemento usando 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 III

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

- Tuplas:

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

- Strings:

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

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'
```

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'
```

Mutabilidad: Listas vs tuplas

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.

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 permite 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]
```

Operaciones sobre listas

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

```
>>> li.append('a')      # se usa un método
```

```
>>> li
```

```
[1, 11, 3, 4, 5, 'a']
```

```
>>> li.insert(2, 'i')
```

```
>>> li
```

```
[1, 11, 'i', 3, 4, 5, 'a']
```

+, extend, append

- `+` crea una lista nueva que tiene 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]]
```

Operaciones sobre listas

- Algunos de ellos: 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']
```

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)
```

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]
```

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

```
>>> ()  
()  
>>> tuple()  
()
```

Devolver valores en funciones I

- Se pueden devolver varios datos

```
def f(x) :
```

```
    return x, x*x, x**2 # o return (x,x*x,x**2)
```

```
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 flexibles.
 - 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)
```

Uso de listas I

Usando bucles:

L = [1, 2, 3, 4, 5]

suma = 0

for i in range(len(L)) :

 suma += L[i]

Uso de listas II

Usando la filosofía de python

```
L = [1, 2, 3, 4, 5]
```

```
suma = 0
```

```
for i in L :
```

```
    suma += i
```

Uso de listas III

Usando bucles:

```
L = [1, 2, 3, 4, 5]
```

```
L_pares = []
```

```
for i in range(len(L)) :
```

```
    if L[i] % 2 == 0 :
```

```
        L_pares.append(L[i])
```

Uso de listas IV

Usando la filosofía de python:

```
L_pares = []
for i in L :
    if i % 2 == 0 :
        L_pares.append(i)
```

Comprensión de listas I

Python permite asignar a una lista todos los valores en una única instrucción.

Es más rápido que hacerlo valor a valor

```
<lista> = [ <expr> for <var> in ... ]
```

```
li = [('a', 1), ('b', 2), ('c', 7)]
```

```
[ n * 3 for (x, n) in li ]
```

Comprensión de listas II

- Usando bucles:

y = []

for i in range(n+1) :

 y.append(a+i*h)

- Usando comprensión de listas:

v = [a+i*h for i in range(n+1)]

- Otro ejemplo:

gradosC = [convertirF_C(i) for i in gradosF]

Comprensión de listas III

- Listas más complejas

```
L = [[0 for col in range(5)] for fil in range(6)]
```

```
tabla2 = []
```

```
for C, F in zip(gradosC, gradosF) :
```

```
    fil = [C, F]
```

```
    table2.append(fil)
```

- Alternativa

```
table2 = [[C, F] for C, F in zip(gradosC, gradosF)]
```

Comprensión de listas IV

```
nums = [1,2,3,4]
```

```
frutas = ["Manzanas", "Melocotones", "Peras",  
"Plátanos"]
```

```
print ([ (i,f) for i in nums for f in frutas])
```

```
list1 = [3,4,5]
```

```
mult = [item*3 for item in list1]
```

```
palabras = ["esto","es","una","lista","de","palabras"]
```

```
items = [ palabra[0] for palabra in palabras ]
```

Operaciones adicionales

- `enumerate(L)` devuelve un par (indice,elem)
`for i, c in enumerate(L) :`
$$L[i] = i + c$$
- `sorted(L)` devuelve una lista ordenada
- `reversed(L)` devuelve una lista con los elementos en orden inverso