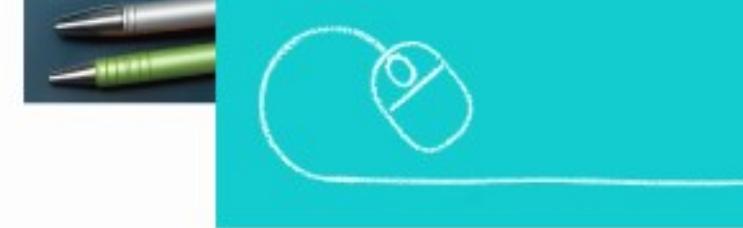


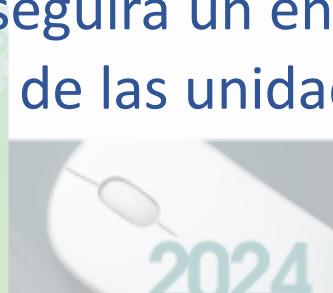
# Python para principiantes

Daniel Garrido  
[dgm@uma.es](mailto:dgm@uma.es)



# Objetivos del curso

- Este curso tiene como objetivo principal que el alumno adquiera los conocimientos necesarios para poder utilizar el lenguaje de programación Python.
- El curso comenzará desde los aspectos más básicos con 2 objetivos principales:
  - Que cualquier pueda después realizar sus propios programas en Python
  - Poder aplicar Python en diferentes campos de aplicación en códigos ya desarrollados.
- El curso seguirá un enfoque eminentemente práctico con ejercicios al final de cada una de las unidades que lo componen.



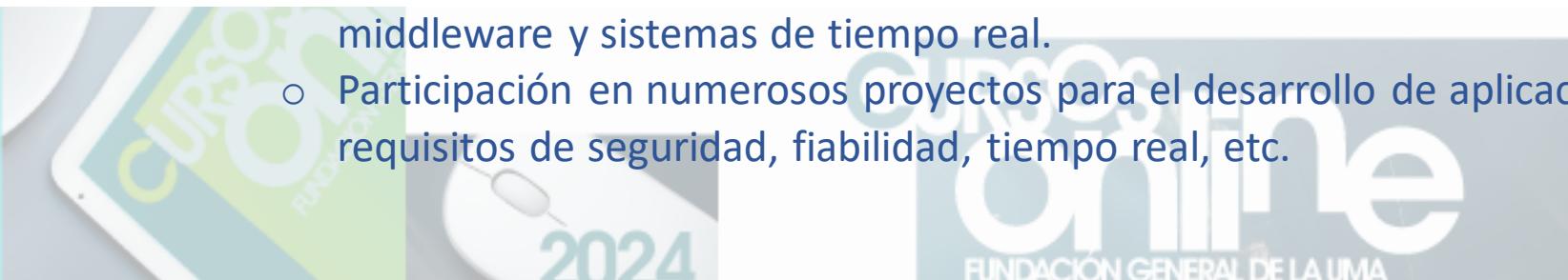
# Profesorado

**Daniel Garrido Márquez ([dgm@uma.es](mailto:dgm@uma.es))**

- Profesor Contratado Doctor
- Departamento de Lenguajes y Ciencias de la Computación
- ETSI Informática
- Área de Lenguajes y Sistemas Informáticos

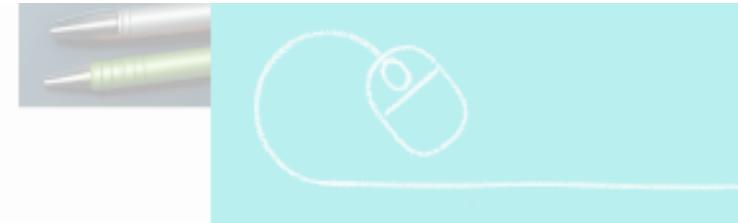
Algunas notas biográficas...

- Ingeniero en Informática y Doctor por la Universidad de Málaga.
- Docencia oficial: Elementos de Programación, Laboratorio de Programación, Programación Concurrente, ...
- Docencia no oficial: Instituto de Astrofísica de Canarias, Ayuntamiento de Málaga, Master Big Data, ...
- Labor investigadora: artículos en congresos y revistas relacionados con el campo de la simulación, middleware y sistemas de tiempo real.
- Participación en numerosos proyectos para el desarrollo de aplicaciones en sistemas críticos con requisitos de seguridad, fiabilidad, tiempo real, etc.



# Contenidos

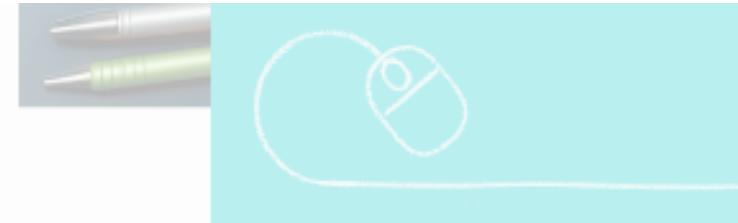
- Características básicas de Python
- Expresiones
- Condicionales y bucles
- Funciones
- Cadenas de caracteres
- Listas, diccionarios y tuplas
- Ficheros
- Complementario (no evaluable):
  - Objetos
  - Visualización de datos
  - Interfaces Gráficas de Usuario



Python para principiantes

# Dinámica del curso

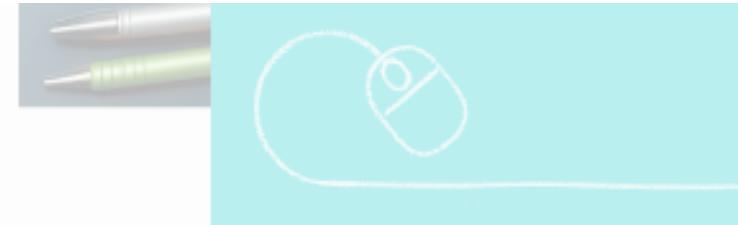
- Se dispondrá de material para las diversas partes del curso según los contenidos indicados anteriormente. Se propondrán pequeñas pruebas para que el alumno pueda ir comprobando sus progresos.
- Al final de cada parte se propondrá la realización de tareas para afianzar los conocimientos adquiridos y que serán obligatorias para superar la evaluación
- Se recomienda realizar en orden los contenidos del curso



Python para principiantes

# Evaluación final

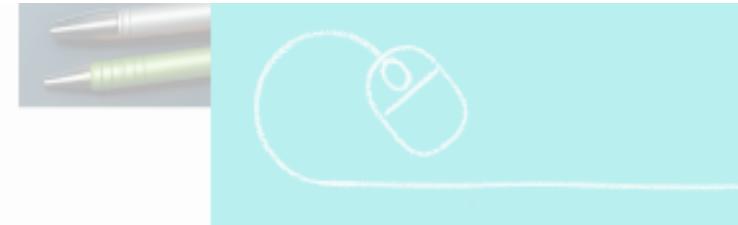
- La evaluación final del curso estará basada en la correcta realización de las diversas prácticas propuestas al final de las diferentes partes del contenido.



Python para principiantes

# Software

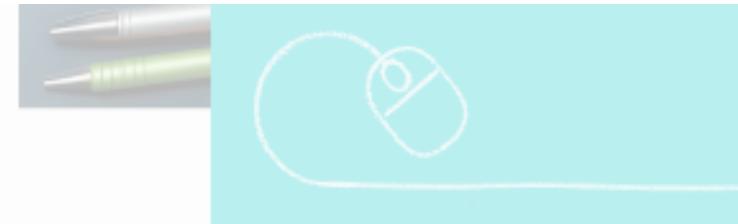
- Instalación de Python 3 y de algún editor de texto tales como Visual Studio Code, Atom, Notepad++.
- Presentación disponible en Campus Virtual con explicación detallada



Python para principiantes

# Bibliografía

- Python for EveryBody
  - <https://www.py4e.com/>
  - La mayoría de los contenidos del curso se basan en este libro. Incluye página web para poder hacer las tareas online
- [Python Crash Course / Matthes, Eric.](#)



Python para principiantes

# Instalando Python

## Unidad 0

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Configurando tu entorno Python

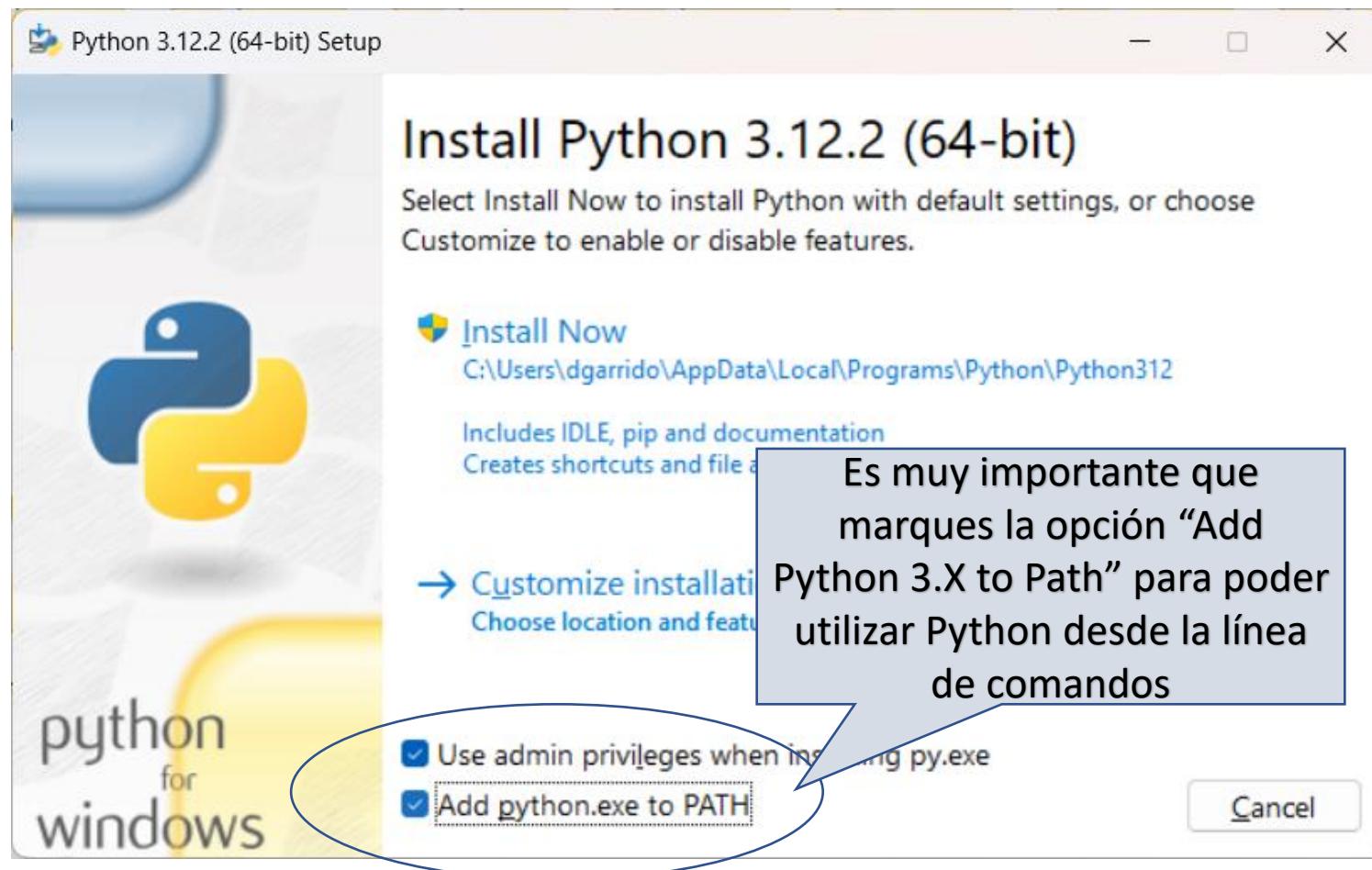
En esta presentación encontrarás indicaciones de cómo configurar tu entorno Python en Windows y Linux/Macintosh.

El objetivo fundamental del curso es que aprendas Python, así que nos vamos a centrar en el lenguaje. Puedes usar el editor o entorno de desarrollo que quieras, aunque bastará con un simple editor de texto y algunas nociones para el manejo de la línea de comandos.

# Instalando Python 3 en Windows

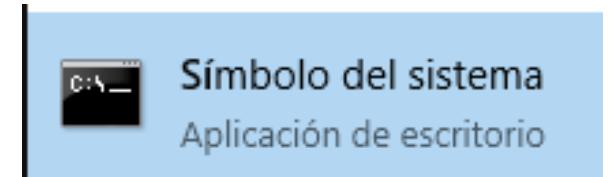
- Cualquier versión reciente de Python es aceptable para este curso. Si ya tienes Python 3.X instalado en tu ordenador puedes continuar sin tener que volver a instalarlo.
- Descarga e instala Python 3.x desde:
  - <http://www.python.org/download/>
- Asegúrate de marcar la opción "Add Python 3.X to PATH" para poder ejecutar Python desde la línea de comandos
- Descarga e instala el editor de texto que prefieras. Algunas sugerencias:
  - <https://code.visualstudio.com/>
  - <https://notepad-plus-plus.org/>
  - <http://atom.io>

# Instalando Python 3 en Windows



# Instalando Python 3 en Windows

- Si todo va bien, cuando abras el símbolo del sistema y escribas **python** deberías ver algo parecido:



A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe - python". The window shows the following text output:

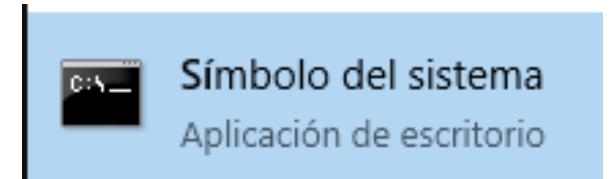
```
C:\Users\dgarrido>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> -
```

# Algunas indicaciones para Windows

- Notas sobre “Símbolo del sistema”

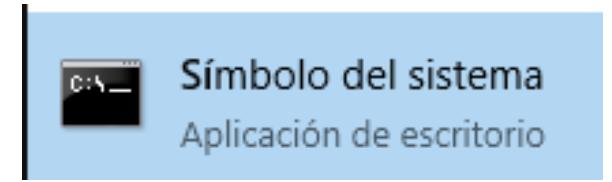
- Vamos a utilizar el símbolo del sistema para ejecutar nuestros programas Python en Windows
- Aunque no se requiere un conocimiento “experto”, sí que tienen que conocerse algunas nociones
- Cuando el símbolo del sistema arranca, te encuentras en tu carpeta de usuario, que varía según la versión de Windows. Por ejemplo:
  - Windows 10/11: C:\Users\alumno
- El indicador de la línea de comandos generalmente da alguna indicación de dónde te encuentras en la jerarquía de carpetas, pero puedes usar el comando **cd** para ver exactamente dónde te encuentras. Por ejemplo:

```
C:\Users\alumno> cd  
C:\Users\alumno
```

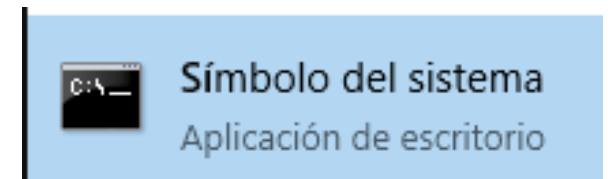


# Algunas indicaciones para Windows

- ¿Dónde ir?
- Generalmente, lo primero que querrás hacer es ir a la carpeta donde esté tu programa python. Por ejemplo, supongamos que está en el escritorio (Desktop), entonces debes escribir lo siguiente:
  - `cd Desktop` (suponiendo que estabas en la carpeta `C:\users\[tu usuario]`)
- El comando **dir** te mostrará los ficheros que haya en el directorio actual
- Y con el comando **cd ..** Puedes “subir” un nivel en la jerarquía de carpetas y volver a donde estuvieras.
- “Truco”: cuando en cualquier comando (p.ej. `cd`) estés escribiendo el nombre de una carpeta o fichero, puedes pulsar la Tecla TAB y Windows completará el nombre que estabas tratando de escribir



# Algunas indicaciones para Windows



- ¿Cómo ejecutar tu programa Python desde la línea de comandos?
- Escribe
  - `python firstprog.py` (donde `firstprog` es el nombre de tu programa Python)
- O simplemente escribe el nombre de tu programa `python`
  - `firstprog.py`
- Nota: si después vuelves a pulsar la tecla de flecha arriba, volverá a aparecer el ultimo comando introducido

# Configurando Python 3 en Linux/Macintosh

- Python 3 viene instalado en la mayoría de distribuciones principales de Linux, así como en Mac OS
- En caso de que no fuera así en tu distribución, puedes seguir las indicaciones de la misma o consultar en:
  - <https://www.python.org/downloads/>
- Descarga e instala el editor de texto que prefieras. Algunas sugerencias:
  - <https://code.visualstudio.com/>
  - <http://atom.io>

# Instalando Python 3 en Linux/Macintosh

- Para comprobar que tu instalación Python es correcta, abre una terminal y escribe **python3**. Deberías ver algo similar:
  - En Linux, la versión 2 de Python suele estar instalada, por lo que para ejecutar programas Python 3, debes escribir **python3**



A screenshot of a terminal window titled "alumno@debian: ~". The window has a standard window title bar with icons for close, minimize, and maximize. Below the title bar is a menu bar with options: Archivo, Editar, Ver, Buscar, Terminal, and Ayuda. The main area of the terminal shows the command "alumno@debian:~\$ python3" followed by the Python 3.5.3 startup message: "Python 3.5.3 (default, Sep 27 2018, 17:25:39) [GCC 6.3.0 20170516] on linux". It also includes the help message: "Type "help", "copyright", "credits" or "license" for more information." A cursor is visible at the bottom left of the terminal window.

# Algunas indicaciones para Linux/Mac

- Notas sobre la terminal



- Vamos a utilizar la terminal para ejecutar nuestros programas Python en Linux/Mac
- Aunque no se requiere un conocimiento “experto”, sí que tienen que conocerse algunas nociones
- Cuando la terminal arranca, te encuentras en tu directorio de usuario. Por ejemplo:
  - /home/alumno
- Para saber dónde te encuentras en la jerarquía de directorios, puedes usar el comando **pwd**. Por ejemplo:

```
alumno@ubuntu:~$ pwd  
/home/alumno
```

# Algunas indicaciones para Linux/Mac

- ¿Dónde ir?
- Generalmente, lo primero que querrás hacer es ir al directorio donde esté tu programa python. Por ejemplo, supongamos que está en el escritorio (Desktop), entonces debes escribir lo siguiente:
  - `cd Desktop` (suponiendo que estabas en `/home/[tu usuario]`)
- El comando `ls -l` te mostrará los ficheros que haya en el directorio actual
- Y con el comando `cd ..` Puedes “subir” un nivel en la jerarquía de directorios y volver a donde estuvieras.
- “Truco”: cuando en cualquier comando (p.ej. `cd`) estés escribiendo el nombre de un directorio o fichero, puedes pulsar la Tecla TAB y se completará el nombre que estabas tratando de escribir



# Algunas indicaciones para Linux/Mac



- ¿Cómo ejecutar tu programa Python desde la terminal?
- Escribe
  - `python3 firstprog.py` (donde `firstprog` es el nombre de tu programa Python)
- Nota: si después vuelves a pulsar la tecla de flecha arriba, volverá a aparecer el ultimo comando introducido

# Notas finales

- Si no puedes o no quieres instalar Python en tu ordenador, puedes probar otras alternativas como [replit](#), [Trinket](#), [PythonAnywhere](#), [Cloud9](#), o [CodeAnywhere](#).
- En la web de Python for EveryBody <https://www.py4e.com/> también podrás realizar online muchas de las tareas del curso.

# ¿Por qué programar?

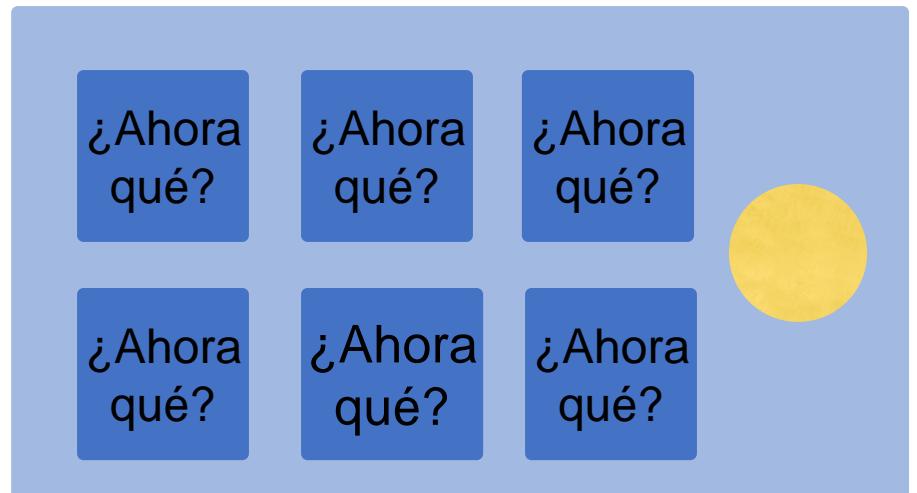
## Unidad 1

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Los ordenadores están para ayudar...

- Están construidos con un propósito: hacer las cosas por nosotros
- Pero es necesario hablar su propio lenguaje para describir qué queremos que hagan
- Los usuarios lo tienen fácil: alguien le puso las instrucciones al ordenador en forma de programas, y ahora solo hay que elegir cuáles usar



# Usuarios versus Programadores

- Los usuarios ven a los ordenadores como un conjunto de **herramientas**: procesador de textos, hoja de cálculo, mapa, lista de tareas, etc.
- Los programadores aprenden el “**comportamiento**” de los ordenadores y sus lenguajes
- Los programadores tienen herramientas que les permiten construir nuevas herramientas
- Los programadores a veces escriben herramientas para otros usuarios, y otras veces pequeñas “**utilidades**” para ellos mismos que les permiten **automatizar** tareas

# ¿Por qué ser un programador?

- Para realizar alguna tarea – somos el usuario y el programador
  - Limpiar los datos de la encuesta
- Para producir algo que otros usen – una tarea de programación
  - Solucionar un problema de rendimiento en el software Sakai
  - Añadir un libro de invitados a un sitio web



Desde el punto de vista del creador de software, **construimos** el software. Los usuarios finales son nuestros amos – a los que debemos agradar – y frecuentemente nos pagan cuando están contentos por nuestro trabajo. Nos enfrentamos a los **datos**, la **información** y las **redes** en lugar de los usuarios. El **hardware** y el **software** son nuestros amigos y aliados en esta búsqueda.

# ¿Qué es el Código? ¿El Software? ¿Un Programa?

- Una secuencia de instrucciones almacenadas
  - Es un producto de nuestra inteligencia en el ordenador
  - Se nos ocurre algo y lo codificamos, y se lo damos a alguien más para que **ahorre tiempo** cuando realice la misma tarea
- Una pieza de arte creativa – particularmente cuando hacemos que la experiencia del usuario sea buena

# Programas para Humanos...



<http://www.youtube.com/watch?v=vlzwuFkn88U>

# Programas para Humanos...

Mientras suena la música:

Mano izquierda fuera y arriba

Mano derecha fuera y arriba

Girar mano izquierda

Girar mano derecha

Mano izquierda en el hombro derecho

Mano derecha en el hombro izquierdo

Mano izquierda en la cabeza

Mano derecha en la cabeza

Mano izquierda en el cinturón

Mano derecha en el cinturón

Mano izquierda en la nalga izquierda

Mano derecha en la nalga derecha

Contoneo

Contoneo

Salta



<http://www.youtube.com/watch?v=vlzwuFkn88U>

# Programas para Python...

Averiguar la palabra  
que más aparece en un  
archivo

the clown ran after the car and the car ran into the tent and  
the tent fell down on the clown and the car



Imagen: [https://www.flickr.com/photos/allan\\_harris/4908070612/](https://www.flickr.com/photos/allan_harris/4908070612/) Attribution-NoDerivs 2.0 Generic (CC BY-ND 2.0)

```
name = input('Nombre fichero: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

```
python words.py
Nombre fichero: words.txt
to 16
```

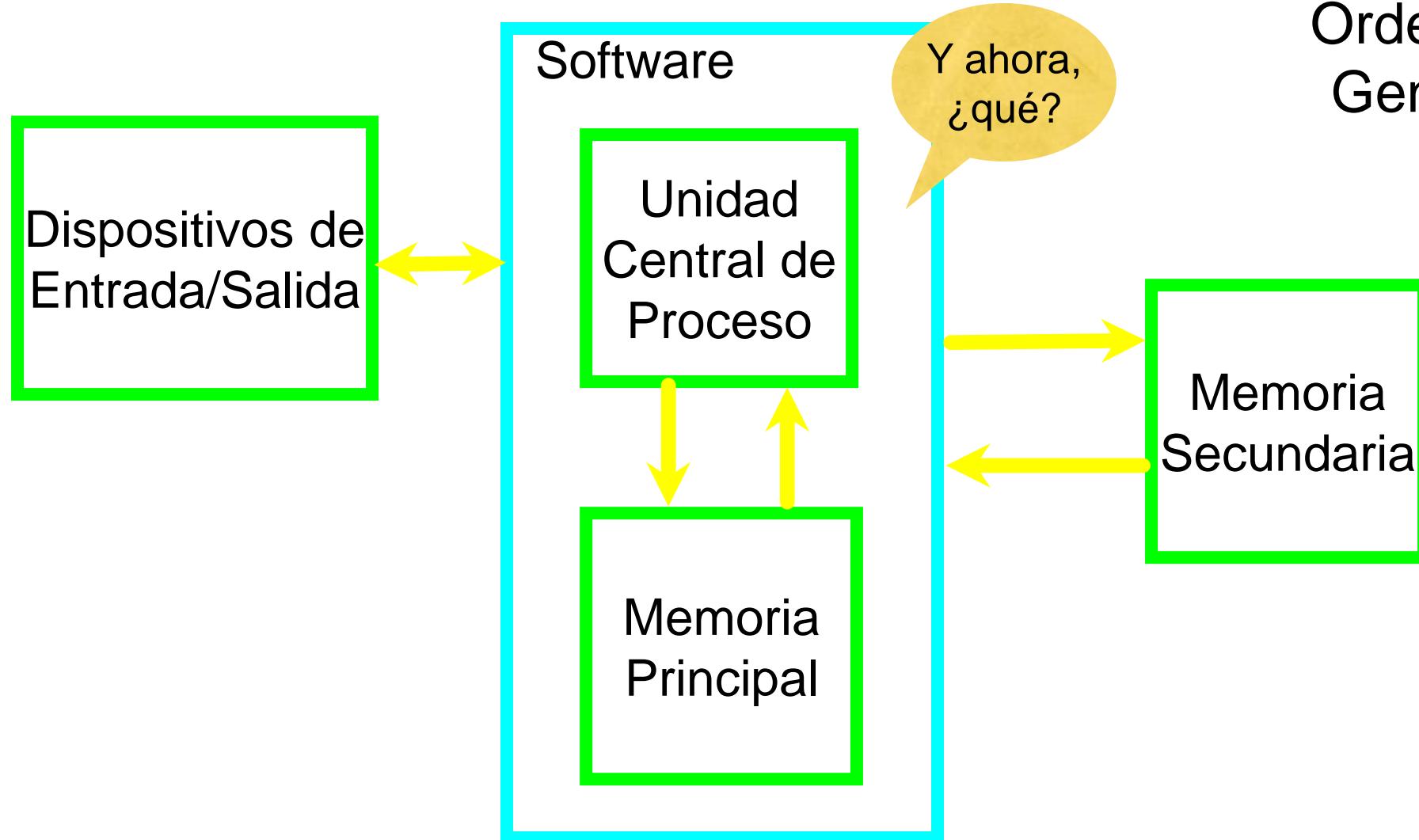
```
python words.py
Nombre fichero : clown.txt
the 7
```

# Arquitectura Hardware



<http://upload.wikimedia.org/wikipedia/commons/3/3d/RaspberryPi.jpg>

## Ordenador Genérico



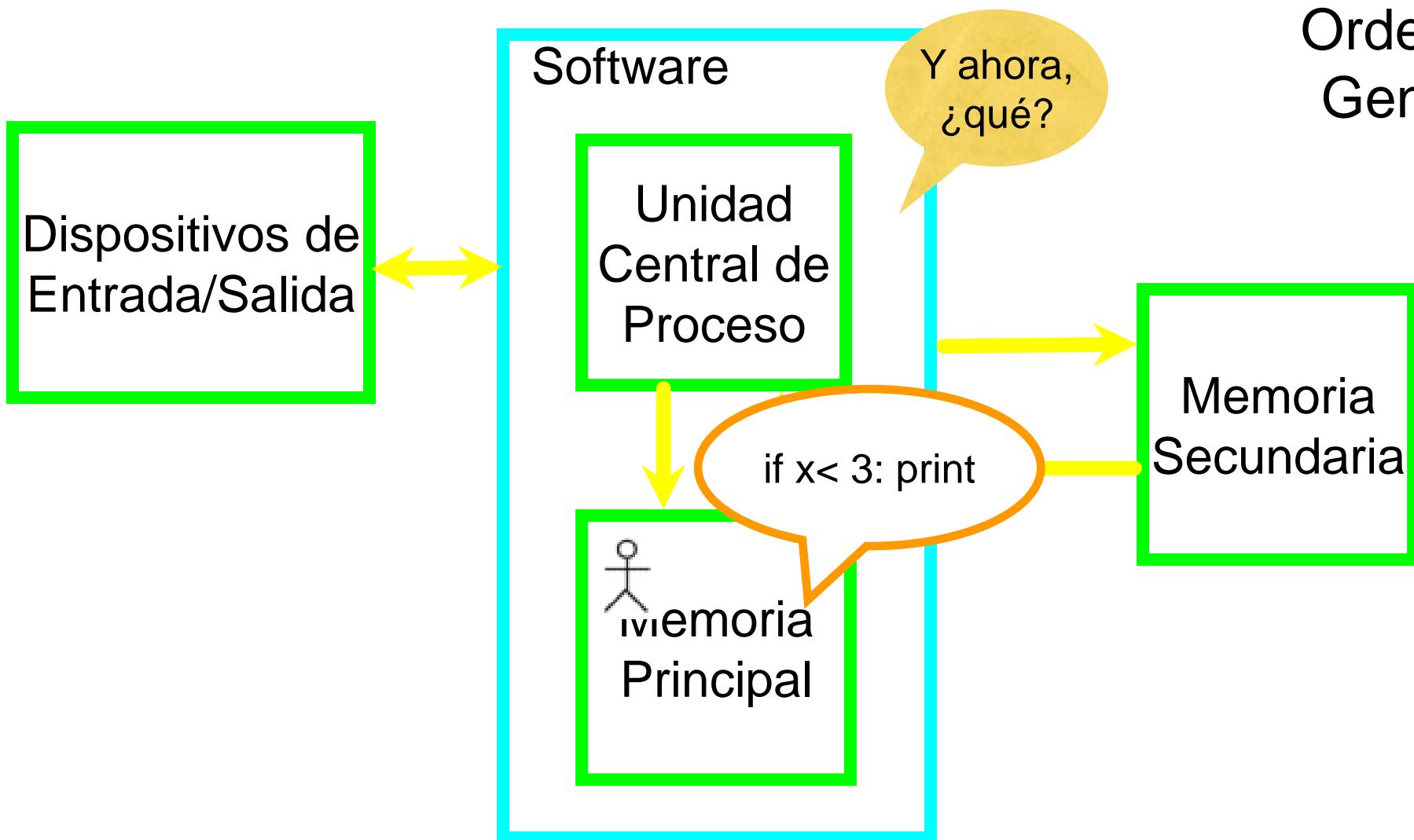
# Definiciones

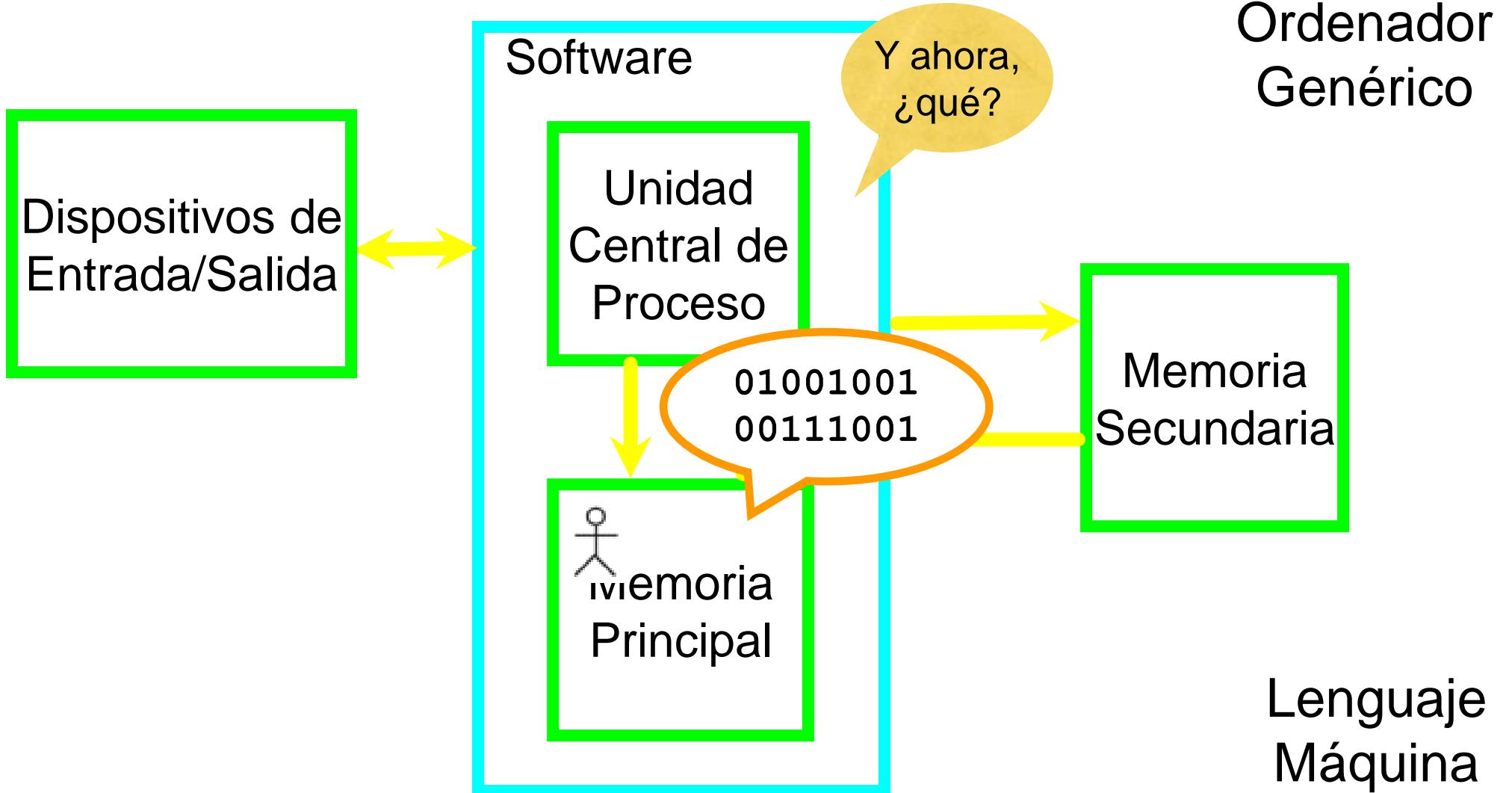


Y ahora,  
¿qué?

- **Unidad Central de Proceso (CPU):** Ejecuta el programa - La CPU siempre está esperando “qué hacer a continuación”. No exactamente un cerebro – muy torpe pero muy muy rápido
- **Dispositivos de Entrada:** Teclado, ratón, pantalla táctil
- **Dispositivos de Salida:** Pantalla, altavoces, impresora
- **Memoria Principal:** Almacenamiento temporal “pequeño” y rápido – se pierde tras un reinicio – memoria RAM
- **Memoria Secundaria:** Almacenamiento temporal más grande y lento – permanece hasta que se borre – disco duro / memoria USB

# Ordenador Genérico





# CPU quemada



<http://www.youtube.com/watch?v=y39D4529FM4>

# Disco Duro en acción



<http://www.youtube.com/watch?v=9eMWG3fwEU>

# Python como Lenguaje

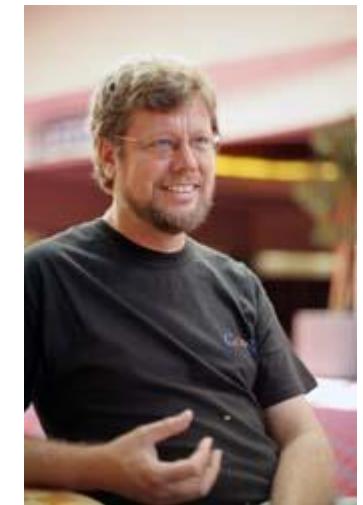
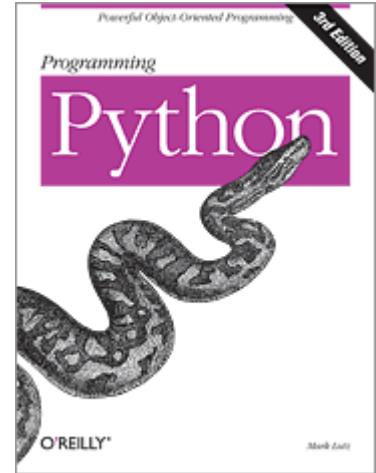
**Parsel** es la lengua de las serpientes y de aquellos que pueden hablar con ellas.

Alguien que puede hablar Parsel es conocido [en inglés] como un Parselmouth. Es una habilidad muy poco común, y es hereditaria. Prácticamente todos los hablantes de Parsel son descendientes de Salazar Slytherin.



<http://harrypotter.wikia.com/wiki/Parseltongue>

**Python** es el lenguaje del intérprete Python y de aquellos que pueden hablar con él. Alguien que puede hablar Python es conocido como un **Pythonista**. Es una habilidad muy poco común, y es hereditaria. Prácticamente todos los Pythonistas conocidos usan software desarrollado inicialmente por **Guido van Rossum**.



# Principiantes: Errores de Sintaxis

- Necesitamos aprender el lenguaje Python para poder **comunicar** nuestras instrucciones a Python. Al principio cometeremos muchos errores y hablaremos farfullando como niños pequeños.
- Cuando cometes un error, el ordenador no tiene piedad. El mensaje que se obtiene es “**syntax error**” (error de sintaxis) – él conoce el lenguaje y tú lo estás aprendiendo. Parece que Python es cruel y sin sentimientos..
- Sólo recuerda que eres inteligente y que **puedes aprender**. El ordenador es simple y muy muy rápido, pero no puede aprender. Así que es más fácil para ti aprender Python que para el ordenador aprender español...

# Hablando a Python

```
csev$ python3
```

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>
```

Y ahora,  
¿qué?



```
csev$ python3
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  5 2015, 21:12:44)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> x = 1
>>> print(x)
1
>>> x = x + 1
>>> print(x)
2
>>> exit()

Esta es una buena prueba para asegurarte de
que tienes Python correctamente instalado.
Con quit() también puedes finalizar la sesión
interactiva.
```

¿Qué le decimos?

# Elementos de Python

- Vocabulario / Palabras - Variables y palabras reservadas  
(Unidad 2)
- Estructura de las sentencias – patrones de sintaxis válidos  
(Unidades 3-5)
- Estructura de programa – construyendo un programa para  
un propósito

```
name = input('Nombre fichero: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Una corta “historia”  
sobre cómo contar  
palabras con  
Python en un  
archivo

```
python words.py
Nombre fichero: words.txt
to 16
```

# Palabras Reservadas

No se pueden utilizar las palabras reservadas como nombres / identificadores

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

# Sentencias o Líneas

x = 2

x = x + 2

print(x)

Sentencia de Asignación  
Asignación con expresión  
Sentencia print

Variable

Operador

Constante

Función

# Programando Párrafos

# Scripts Python

- Interactuar con Python en el **intérprete** sirve para experimentos y programas de 3 o 4 líneas de largo
- La mayoría de los programas son mucho más largos, así que hay que **escribirlos en un archivo** y decirle a Python que ejecute las órdenes de ese archivo.
- En cierto sentido, estamos “dando a Python un guión” (**script**).
- Como convención, se suma “**.py**” como sufijo al final de estos archivos para indicar que contienen código Python.

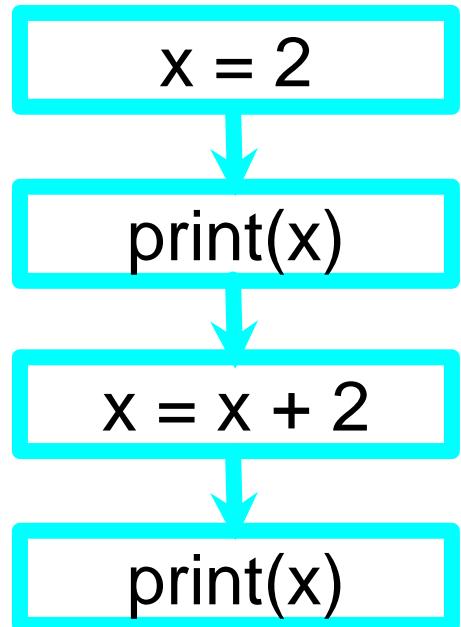
# Interactivo versus Script

- Interactivo
  - Se escribe línea a línea cada vez y Python responde
- Script
  - Se introduce una secuencia de sentencias (líneas) en un archivo usando un editor de texto y Python ejecuta las sentencias del archivo.

# Pasos de programa o Flujo de programa

- Como una receta o instrucciones de instalación, un programa es una secuencia de pasos para ser hechos en orden.
- Algunos pasos son condicionales – se pueden saltar.
- A veces un paso o grupo de ellos son repetidos.
- A veces almacenamos un conjunto de pasos para ser utilizados una y otra vez en diferentes partes de nuestro programa (Unidad 4).

# Pasos Secuenciales



Programa:

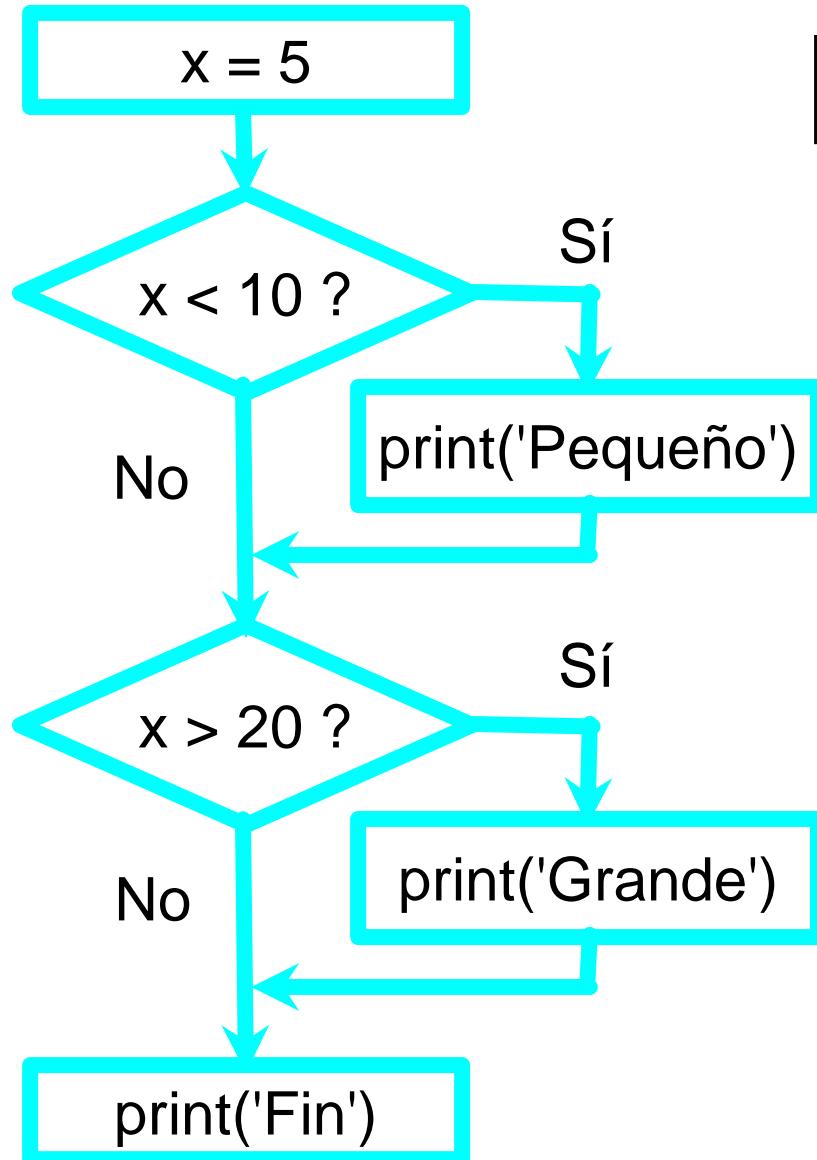
```
x = 2  
print (x)  
x = x + 2  
print (x)
```

Salida:

```
2  
4
```

Cuando un programa se está ejecutando, va de un paso al siguiente. Como programadores, establecemos “rutas” que el programa debe seguir.

# Pasos Condicionales



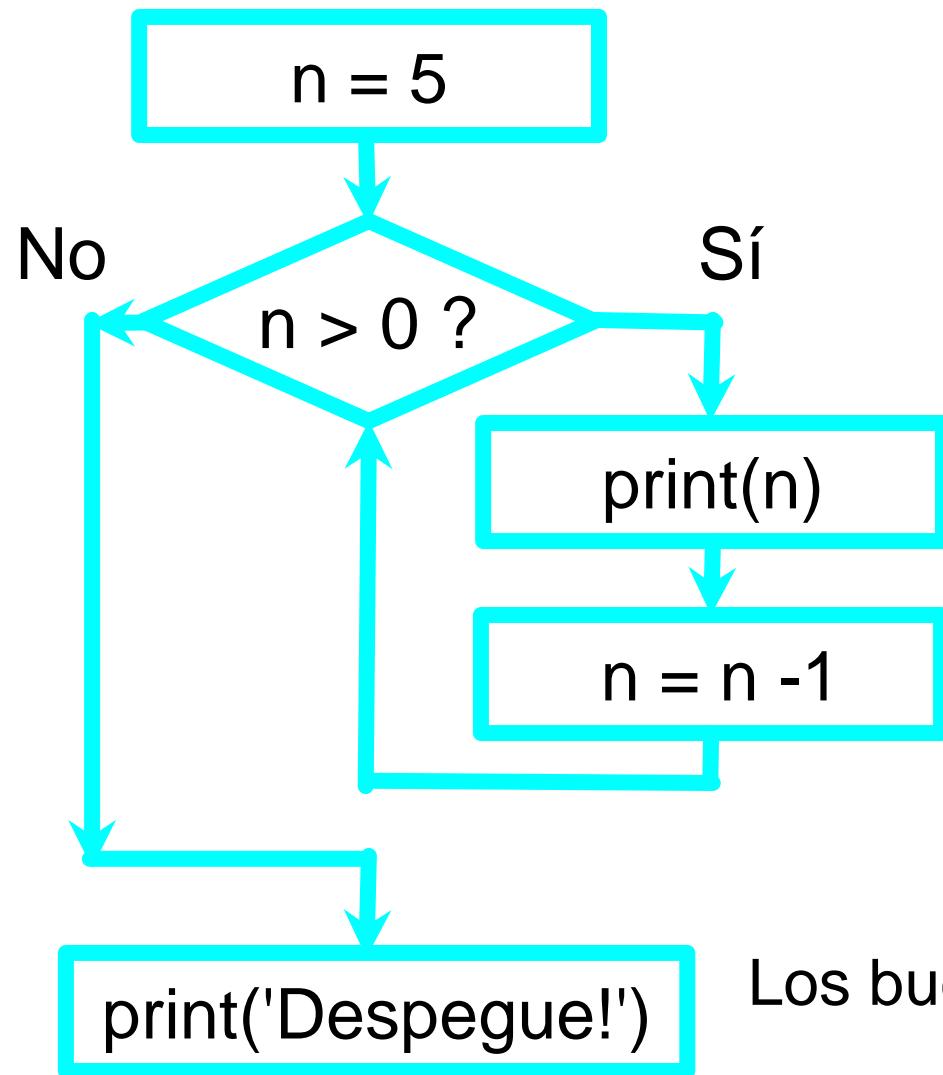
Programa:

```
x = 5
if x < 10:
    print('Pequeño')
if x > 20:
    print('Grande')
print('Fin')
```

Salida:

Pequeño  
Fin

# Pasos Repetidos



Programa:

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Despegue!')
```

Salida:

5	
4	
3	
2	
1	
Despegue!	

Los bucles (pasos repetidos) tienen variables de iteración que van cambiando a lo largo del bucle.

```
name = input('Nombre fichero: ')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Secuencial  
Repetido  
Condicional

Una “historia” corta en Python sobre cómo contar palabras en un archivo

```
name = input('Nombre fichero: ')
handle = open(name, 'r')

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1
```

```
bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Una instrucción utilizada para leer datos del usuario

Una sentencia sobre actualizar una de las muchas cuentas

Un párrafo sobre cómo encontrar el elemento más grande en una lista

# Resumen

Esta es una visión rápida del contenido del curso

Volveremos a ver estos conceptos a lo largo del curso

Se centra en una visión global

# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Variables, Expresiones y Sentencias

## Unidad 2

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Constantes

- Los **valores fijos** tales como **números, letras y cadenas de caracteres (strings)** son conocidos como “**constantes**” porque sus valores no cambian
- Las constantes numéricas se representan tal cual (como **números**)
- Las constantes de cadenas caracteres utilizan **comillas simples ('')** o **dobles ("")**

```
>>> print(123)  
123  
>>> print(98.6)  
98.6  
>>> print('Hello world')  
Hello world
```

# Palabras Reservadas

No se pueden utilizar las palabras reservadas como nombres de variables / identificadores

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

# Variables

- Una variable es un lugar con “**nombre**” en la memoria donde un programador puede **almacenar datos y recuperarlos más tarde** usando el nombre de la variable
- Los programadores eligen los nombres de las variables
- El contenido de una variable se puede cambiar con la **sentencia de asignación**

x = 12.2

y = 14

x 12.2

y 14

# Variables

- Una variable es un lugar con “**nombre**” en la memoria donde un programador puede **almacenar datos y recuperarlos más tarde** usando el nombre de la variable
- Los programadores eligen los nombres de las variables
- El contenido de una variable se puede cambiar con la **sentencia de asignación**

x = 12.2

y = 14

x = 100

x ~~12.2~~ 100

y 14

# Reglas para los Nombres de Variables en Python

- Deben comenzar con una letra o subrayado \_
- Solo pueden contener letras, números y subrayados
- Se **distinguen** mayúsculas de minúsculas

Bien: spam eggs spam23 \_speed

Mal: 23spam #sign var.12

Diferentes: spam Spam SPAM

# Mnemónicos para Nombres de Variables

- Dado que son los programadores quienes eligen los nombres, hay unas “**mejores prácticas**”
- Elegir un buen nombre es fundamental para que se entienda tu código
- Los nombres de las variables tienen que ayudar a recordar lo que queremos que almacenen (“mnemónico” = “ayuda a recordar”)

<http://en.wikipedia.org/wiki/Mnemonic>

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print(x1q3p9afd)
```

¿Qué hace este  
fragmento de  
código?

```
x1q3z9ocd = 35.0  
x1q3z9afd = 12.50  
x1q3p9afd = x1q3z9ocd * x1q3z9afd  
print(x1q3p9afd)
```

```
a = 35.0  
b = 12.50  
c = a * b  
print(c)
```

¿Qué hacen estos  
fragmentos de  
código?

```
x1q3z9ocd = 35.0                                a = 35.0
x1q3z9afd = 12.50                               b = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd             c = a * b
print(x1q3p9afd)                                print(c)
```

¿Qué hacen estos  
fragmentos de  
código?

```
horas = 35.0
precio = 12.50
total = horas * precio
print(total)
```

# Sentencias o Líneas

x = 2

x = x + 2

print(x)

Sentencia de Asignación  
Asignación con expresión  
Sentencia print

Variable

Operador

Constante

Función

# Sentencias de Asignación

- Asignamos un valor a una variable usando la sentencia de asignación (=)
- Una sentencia de asignación consiste en una **expresión** en la parte derecha y una **variable** en la parte izquierda para almacenar el resultado

$$x = 3.9 * x * (1 - x)$$

Una variable es una localización en memoria utilizada para almacenar un valor (0.6)



$$x = 3.9 * x * (1 - \frac{0.6}{x})$$

A diagram illustrating the evaluation of an expression. The expression  $x = 3.9 * x * (1 - \frac{0.6}{x})$  is shown. Orange arrows point from the numbers 0.6 and 3.9 to their respective positions in the expression. Another orange arrow points from the value 0.4 to the term  $\frac{0.6}{x}$ . The result of the evaluation, 0.936, is shown at the bottom.

La parte derecha es una expresión.  
Una vez que la expresión ha sido evaluada, el resultado es colocado en (asignado a) x.

El valor almacenado en una variable puede ser **actualizado** reemplazando el antiguo valor (0.6) con uno nuevo (0.936).



$$x = 3.9 * x * (1 - \frac{0.6}{x})$$

A diagram illustrating the evaluation of the expression  $3.9 * x * (1 - \frac{0.6}{x})$ . The expression is shown above. Orange arrows point from the number  $0.6$  to its position in the term  $\frac{0.6}{x}$ , and from the number  $3.9$  to its position in the term  $3.9 * x$ . Another orange arrow points from the intermediate value  $0.4$  (calculated as  $1 - \frac{0.6}{x}$ ) to the term  $(1 - \frac{0.6}{x})$ . A final orange arrow points from the value  $0.936$  to the final result of the expression.

La parte derecha es una expresión.  
Una vez que la expresión ha sido evaluada, el resultado es colocado en (asignado a) x.

# Expresiones...

# Expresiones Numéricas

- Debido a la falta de símbolos matemáticos en los teclados, se usan algunos símbolos específicos para expresar las operaciones matemáticas clásicas
- Por ejemplo:
  - El asterisco es la multiplicación
  - Las potencias (elevar un número a una potencia) utilizan un doble \*\*

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

# Expresiones Numéricas

```
>>> xx = 2  
>>> xx = xx + 2  
>>> print(xx)  
4  
>>> yy = 440 * 12  
>>> print(yy)  
5280  
>>> zz = yy / 1000  
>>> print(zz)  
5.28
```

```
>>> jj = 23  
>>> kk = jj % 5  
>>> print(kk)  
3  
>>> print(4 ** 3)  
64
```

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

# Orden de Evaluación

- Cuando usamos varios operadores juntos Python debe saber **qué hacer primero**
- Esto es lo que se llama “**precedencia** de operadores”
- ¿Qué operador “toma precedencia” sobre los otros?

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

# Reglas de Precedencia de Operadores

- De la precedencia más alta a la más baja:
    - Los paréntesis son siempre respetados
    - Exponenciación (elevar a una potencia)
    - Multiplicación, División y Resto
    - Suma y Resta
    - De izquierda a derecha
- Paréntesis
- Potencia
- Multiplicación
- Suma
- Izquierda a Derecha

```
>>> x = 1 + 2 ** 3 / 4 * 5
```

```
>>> print(x)
```

```
11.0
```

```
>>>
```

Paréntesis

Potencia

Multiplicación

Suma

Izquierda a

Derecha

```
1 + 2 ** 3 / 4 * 5
```

```
1 + 8 / 4 * 5
```

```
1 + 2 * 5
```

```
1 + 10
```

```
11
```



# Precedencia de Operadores

- Recordar las reglas anteriores
- Utilizar paréntesis **en caso de duda**
- Mantener las expresiones matemáticas lo suficientemente **simples** para que sean fáciles de entender
- “Romper” series largas de operaciones matemáticas para hacerlas más claras

Paréntesis

Potencia

Multiplicación

Suma

Izquierda a  
Derecha

# Tipos...

# ¿Qué significa el “Tipo”?

- En Python las variables y constantes tienen un “**tipo**”
- Python conoce la diferencia entre un número entero y una cadena de caracteres (string)
- Por ejemplo “+” significa “sumar” si algo es un número y “concatenar” si algo es una cadena de caracteres

```
>>> ddd = 1 + 4  
>>> print(ddd)  
5  
>>> eee = 'hello ' + 'there'  
>>> print(eee)  
hello there
```

concatenar = poner junto

# Cuestiones con Tipos

- Python conoce el “tipo” de todo
- Algunas operaciones están prohibidas
- No se puede “sumar 1” a un string
- Podemos preguntar a Python el tipo de algo usando la función **type()**

```
>>> eee = 'hello ' + 'there'  
>>> eee = eee + 1  
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>TypeError: Can't convert  
'int' object to str implicitly  
>>> type(eee)  
<class 'str'>  
>>> type('hello')  
<class 'str'>  
>>> type(1)  
<class 'int'>  
>>>
```

# Varios Tipos de Números

- Los números tienen dos tipos principales
  - Los número **enteros**:  
-14, -2, 0, 1, 100, 401233
  - Los números **flotantes** (con decimales): -2.5 , 0.0, 98.6, 14.0
- Hay otros tipos de números – variaciones de flotantes y enteros

```
>>> xx = 1
>>> type(xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class 'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>>
```

# Conversiones de Tipo

- Cuando se ponen enteros y flotantes en una expresión, el entero es **convertido** implícitamente a flotante
- Este comportamiento se puede controlar con las funciones **int()** y **float()**

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

# División Entera

- La división de enteros produce como resultado un número flotante

```
>>> print(10 / 2)  
5.0  
>>> print(9 / 2)  
4.5  
>>> print(99 / 100)  
0.99  
>>> print(10.0 / 2.0)  
5.0  
>>> print(99.0 / 100.0)  
0.99
```

**Esto era diferente en Python 2.x**

# Conversiones de Strings

- Se puede utilizar `int()` y `float()` para convertir entre cadenas de caracteres y enteros
- Se obtendrá un error si el string no tiene un número “bien escrito”

```
>>> sval = '123'  
>>> type(sval)  
<class 'str'>  
>>> print(sval + 1)  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
TypeError: Can't convert 'int' object  
to str implicitly  
>>> ival = int(sval)  
>>> type(ival)  
<class 'int'>  
>>> print(ival + 1)  
124  
>>> nsv = 'hello bob'  
>>> niv = int(nsv)  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
ValueError: invalid literal for int()  
with base 10: 'x'
```

# Entrada de Datos

- Podemos hacer que Python lea datos del usuario utilizando la función **input()**
- La función **input()** retorna una cadena de caracteres

```
nam = input('Who are you? ')
print('Welcome', nam)
```

Who are you? Chuck  
Welcome Chuck

# Convirtiendo la Entrada del Usuario

- Si queremos leer un número del usuario, debemos **convertir de string a número** utilizando la función correspondiente
- Después veremos que hacer con datos de entrada incorrectos



```
inp = input('Europe floor?')  
usf = int(inp) + 1  
print('US floor', usf)
```

Europe floor? 0  
US floor 1

# Comentarios en Python

- Todo lo que se escriba después de un `#` es **ignorado** por Python
  - Ojo, excepto si estamos dentro de una cadena de caracteres.
    - P.ej. “hola # aquí #”
- ¿Por qué comentar?
  - - Para **describir** lo que va a ocurrir en una secuencia de código
  - **Documentar** quién escribió el código u otra información auxiliar
  - **Desactivar** una línea de código – quizás temporalmente

```
# Obtener el nombre del archivo y abrirla
name = input('Nombre fichero:')
handle = open(name, 'r')

# Contar las apariciones de palabras
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

# Encontrar la palabra más común
bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# Todo hecho
print(bigword, bigcount)
```

# Resumen

- Tipos
- Palabras reservadas
- Variables (mnemónicos)
- Operadores
- Precedencia de operadores
- División entera
- Conversión entre tipos
- Entrada de usuario
- Comentarios (#)

# Ejercicio

Escribir un programa que pregunte al usuario un número de horas y el precio por hora para calcular el total a pagar.

Introduzca las horas: 35

Introduzca el precio/hora: 2.75

Total: 96.25

# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)



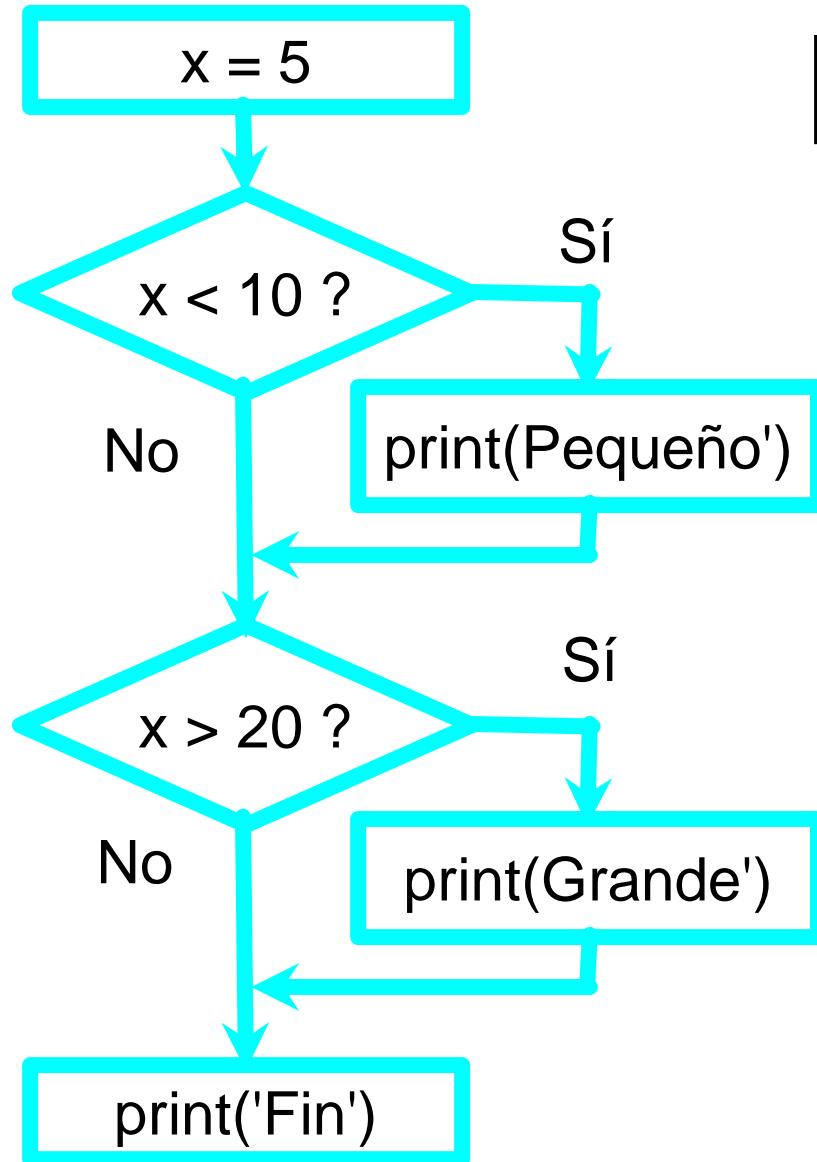
# Ejecución Condicional

## Unidad 3

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Pasos Condicionales



Programa:

```
x = 5
if x < 10:
    print('Pequeño')
if x > 20:
    print('Grande')
print('Fin')
```

Salida:

Pequeño  
Fin

# Operadores de Comparación

- Una expresión booleana hace una pregunta y produce un resultado **Sí** o **No** que puede usarse para controlar la ejecución del programa
- Las expresiones booleanas usan operadores de comparación que evalúan a **True** / **False** o Sí / No
- True y False son palabras reservadas de Python
- Los operadores de comparación miran el valor de las variables pero no las cambian

Python	Significado
<	Menor que
<=	Menor que o igual a
==	Igual a
>=	Mayor que o Igual a
>	Mayor que
!=	Distinto
Recuerda “=” se utiliza para la asignación.	

[http://es.wikipedia.org/wiki/George\\_Boole](http://es.wikipedia.org/wiki/George_Boole)

# Operadores de Comparación

```
x = 5
if x == 5 :
    print('Igual a 5')
if x > 4 :
    print('Mayor que 4')
if x >= 5 :
    print('Igual que o mayor a 5')
if x < 6 : print('Menor que 6')
if x <= 5 :
    print('Menor que o igual a 5')
if x != 6 :
    print('Distinto de 6')
```

Igual a 5

Mayor que 4

Igual que o mayor a 5

Menor que 6

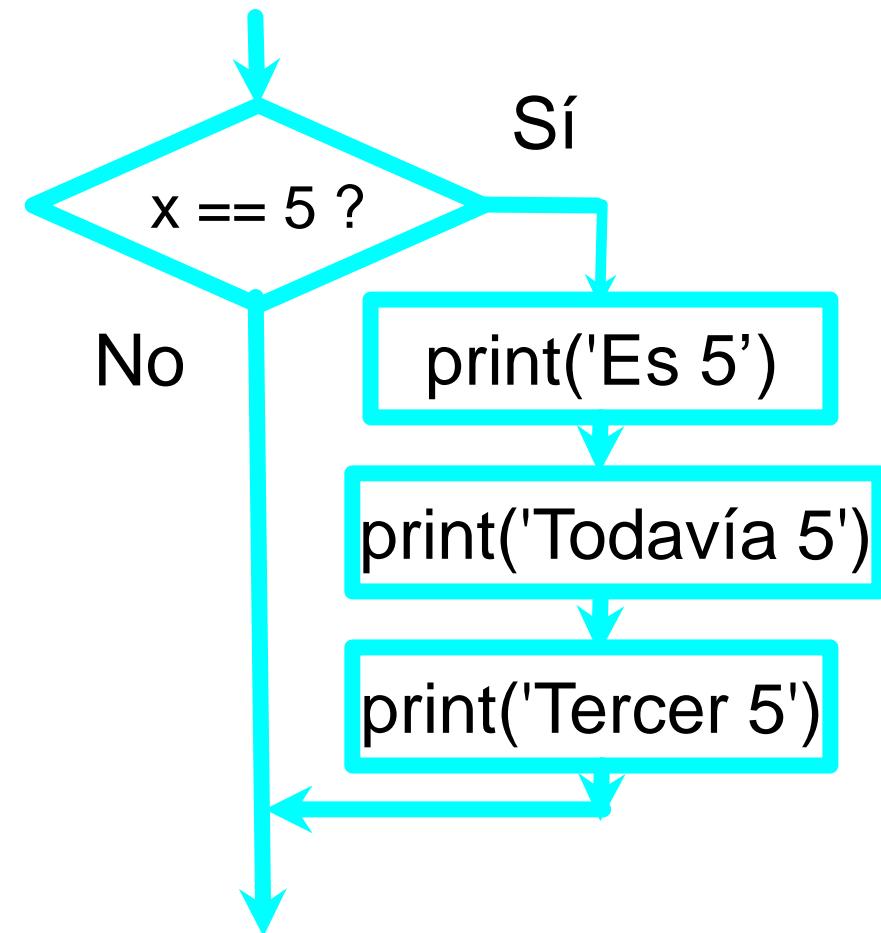
Menor que o igual a 5

Distinto de 6

# Decisiones Simples

```
x = 5  
print('Antes de 5')  
if x == 5 :  
    print('Es 5')  
    print('Todavía 5')  
    print('Tercer 5')  
print('Después de 5')  
print('Antes de 6')  
if x == 6 :  
    print('Es 6')  
    print('Todavía 6')  
    print('Tercer 6')  
print('Después de 6')
```

Antes de 5  
Es 5  
Todavía 5  
Tercer 5  
Después de 5  
Antes de 6  
Después de 6

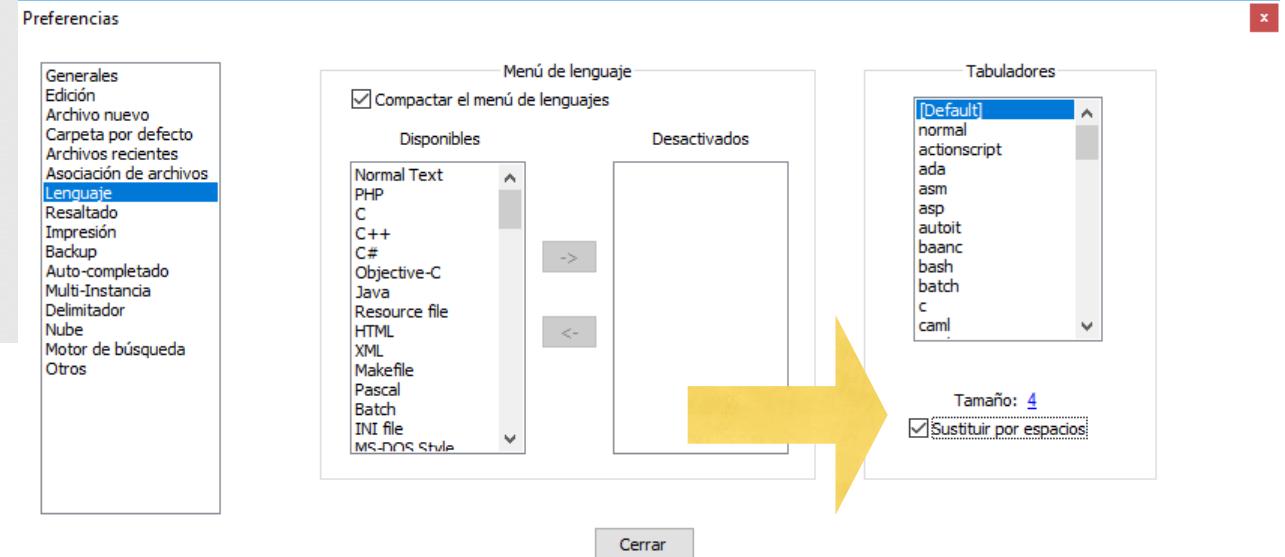
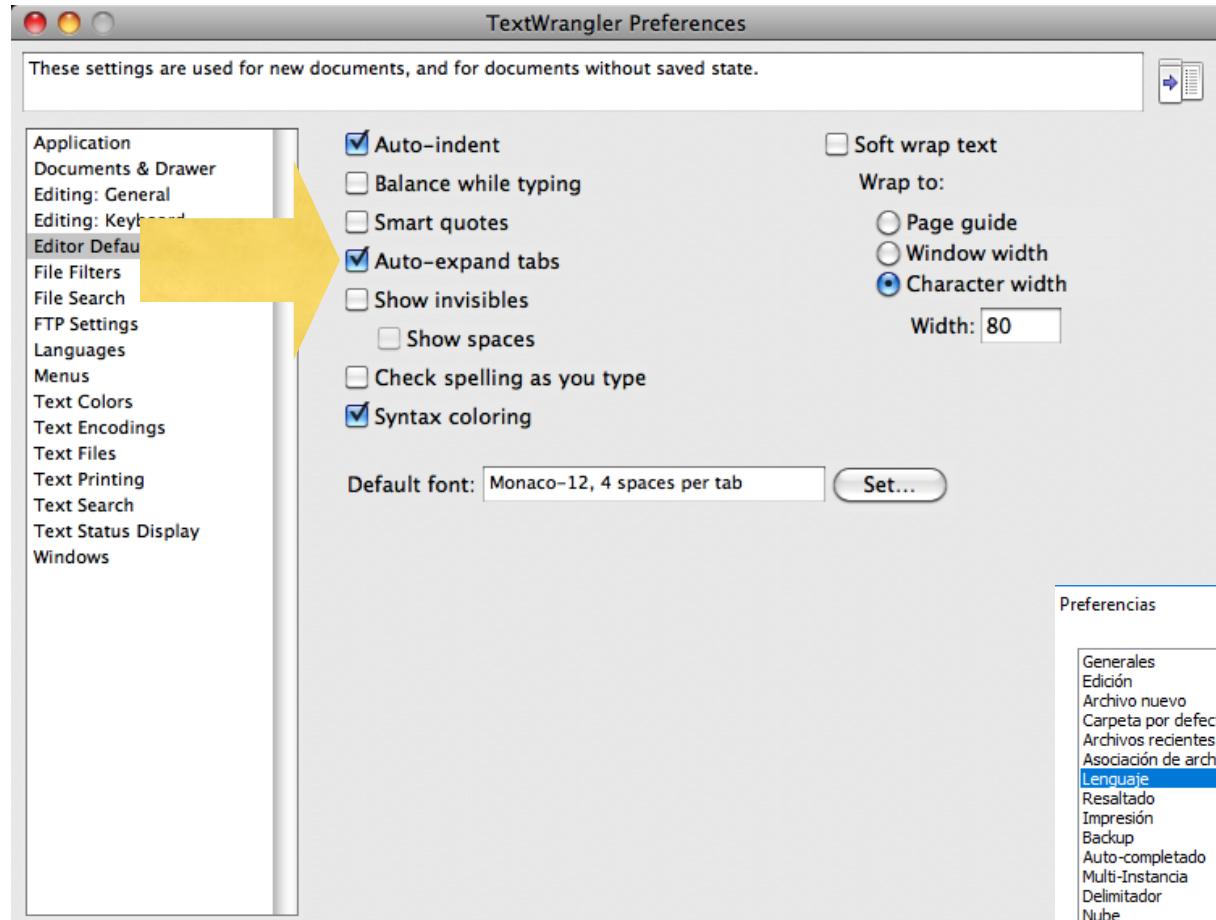


# Indentación

- La indentación (escribir más a la derecha o más a la izquierda que la línea anterior) se utiliza en Python para **agrupar** bloques de código
- Hay que **incrementar** la indentación después de una sentencia if o for (después de : )
- Hay que **mantener** la indentación para indicar el ámbito del bloque (si seguimos dentro del if/for)
- Hay que **reducir** la indentación al mismo nivel que la sentencia if o for para indicar la finalización del bloque
- Las líneas en blanco se ignoran – no afectan a la indentación
- Los comentarios se ignoran también con respecto a la indentación

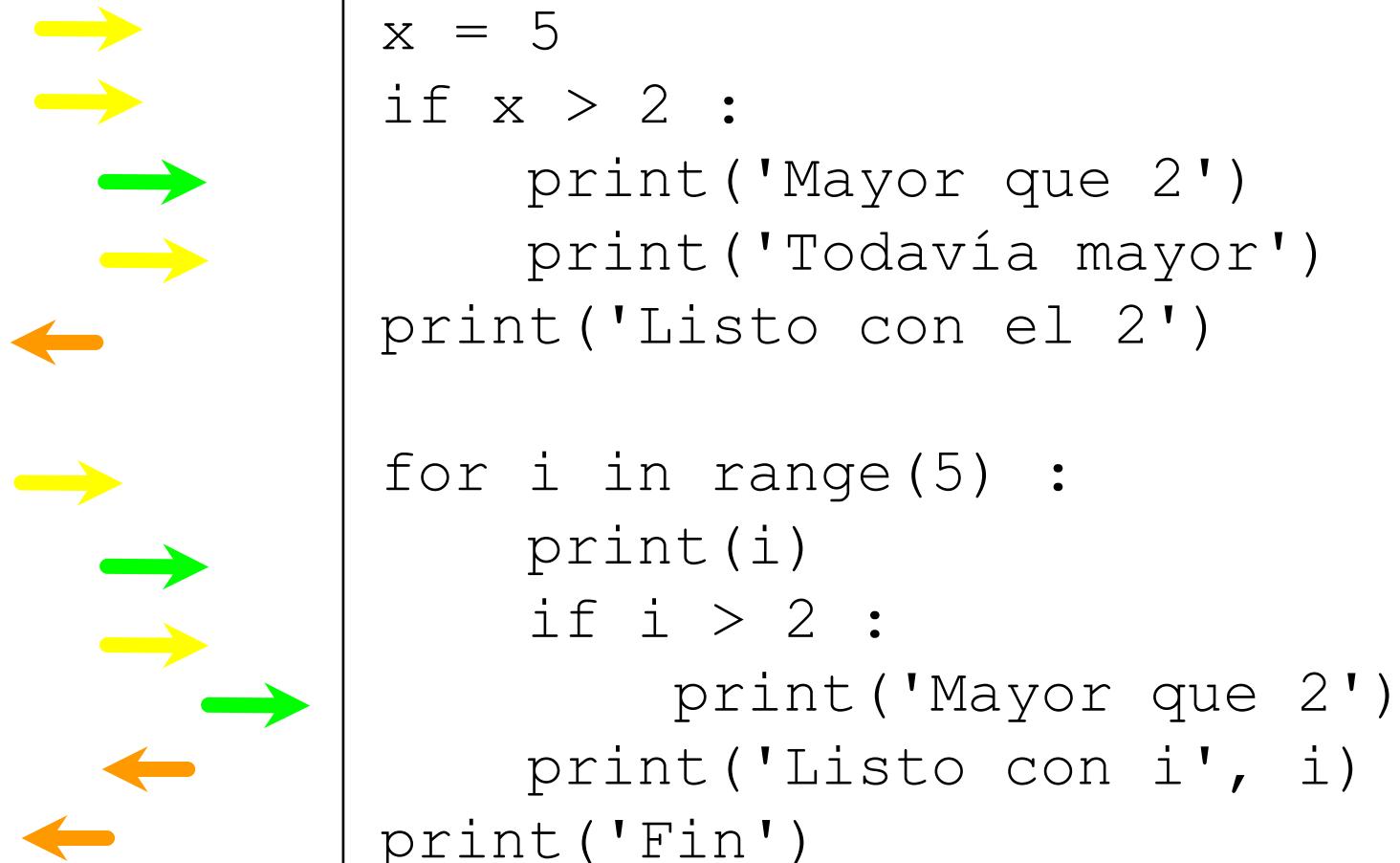
# Advertencia: ¡¡Desactivar la tabulación!!

- El editor Atom utiliza espacios automáticamente en archivos con extensión ".py" (bueno!)
- La mayoría de los editores pueden **convertir las tabulaciones en espacios** – **asegúrate** de habilitarlo
  - - NotePad++: Settings -> Preferences -> Language Menu/Tab Settings o bien Configuración -> Preferencias -> Lenguaje -> Tabuladores
  - - TextWrangler: TextWrangler -> Preferences -> Editor Defaults
- Python se preocupa **\*mucho\*** sobre la indentación de las líneas. Si mezclas tabulaciones y espacios, puedes obtener “**indentation errors**” incluso si todo parece correcto



Esto te ahorrará  
muchos problemas  
si usas Atom.

## incrementar / mantener después de if o for decrementar para indicar el fin de bloque



The diagram illustrates the flow of control in two snippets of Python code. Yellow arrows point from left to right, indicating the direction of execution. Green arrows point from right to left, indicating exits from loops or blocks. Orange arrows point from right to left, indicating exits from loops or blocks.

```
x = 5
if x > 2 :
    print('Mayor que 2')
    print('Todavía mayor')
print('Listo con el 2')

for i in range(5) :
    print(i)
    if i > 2 :
        print('Mayor que 2')
        print('Listo con i', i)
    print('Fin')
```

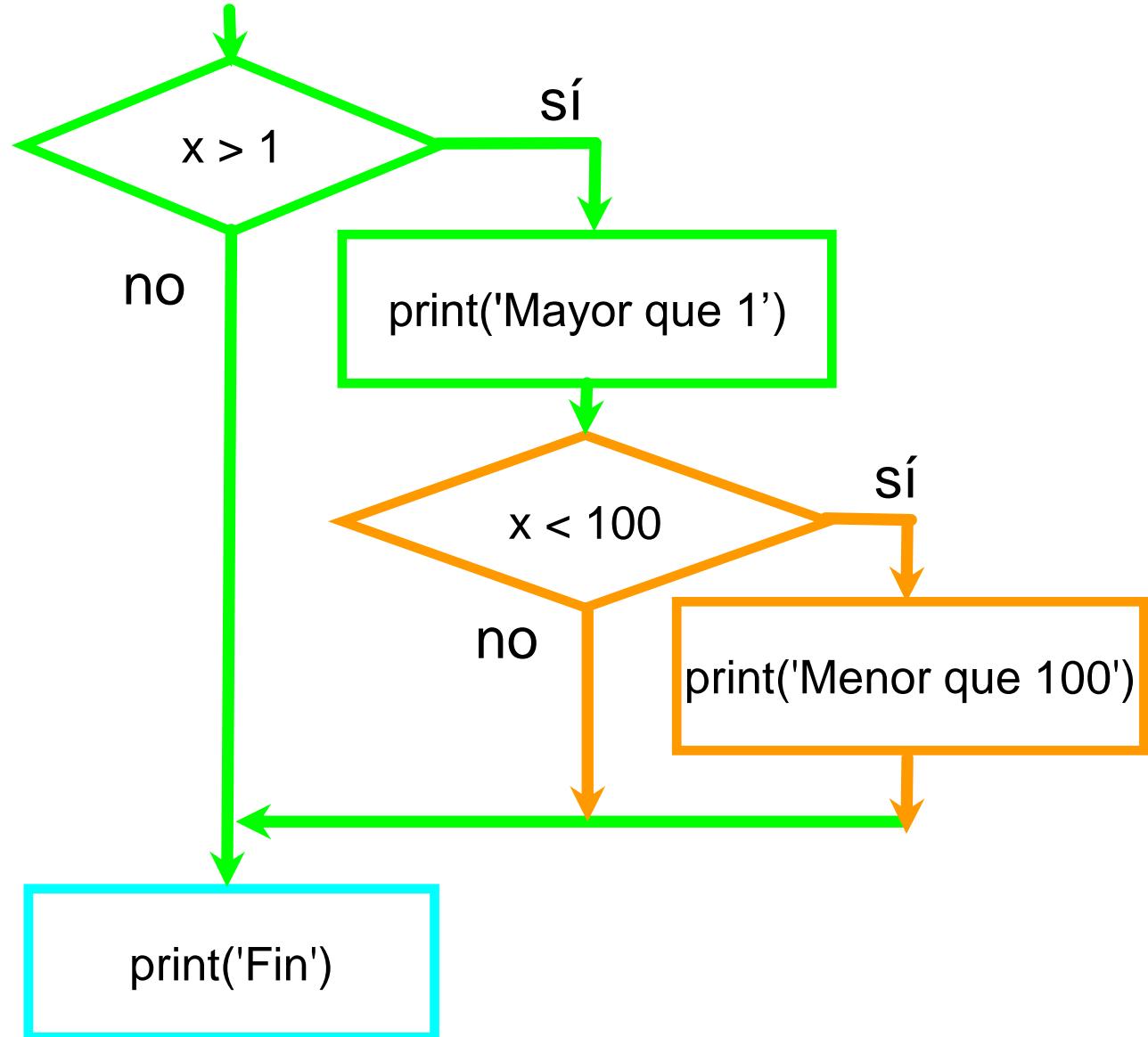
# Pensar sobre el inicio/fin de los bloques

```
x = 5
if x > 2 :
    print('Mayor que 2')
    print('Todavía mayor')
print('Listo con el 2')
```

```
for i in range(5) :
    print(i)
    if i > 2 :
        print('Mayor que 2')
    print('Listo con i', i)
print('Fin')
```

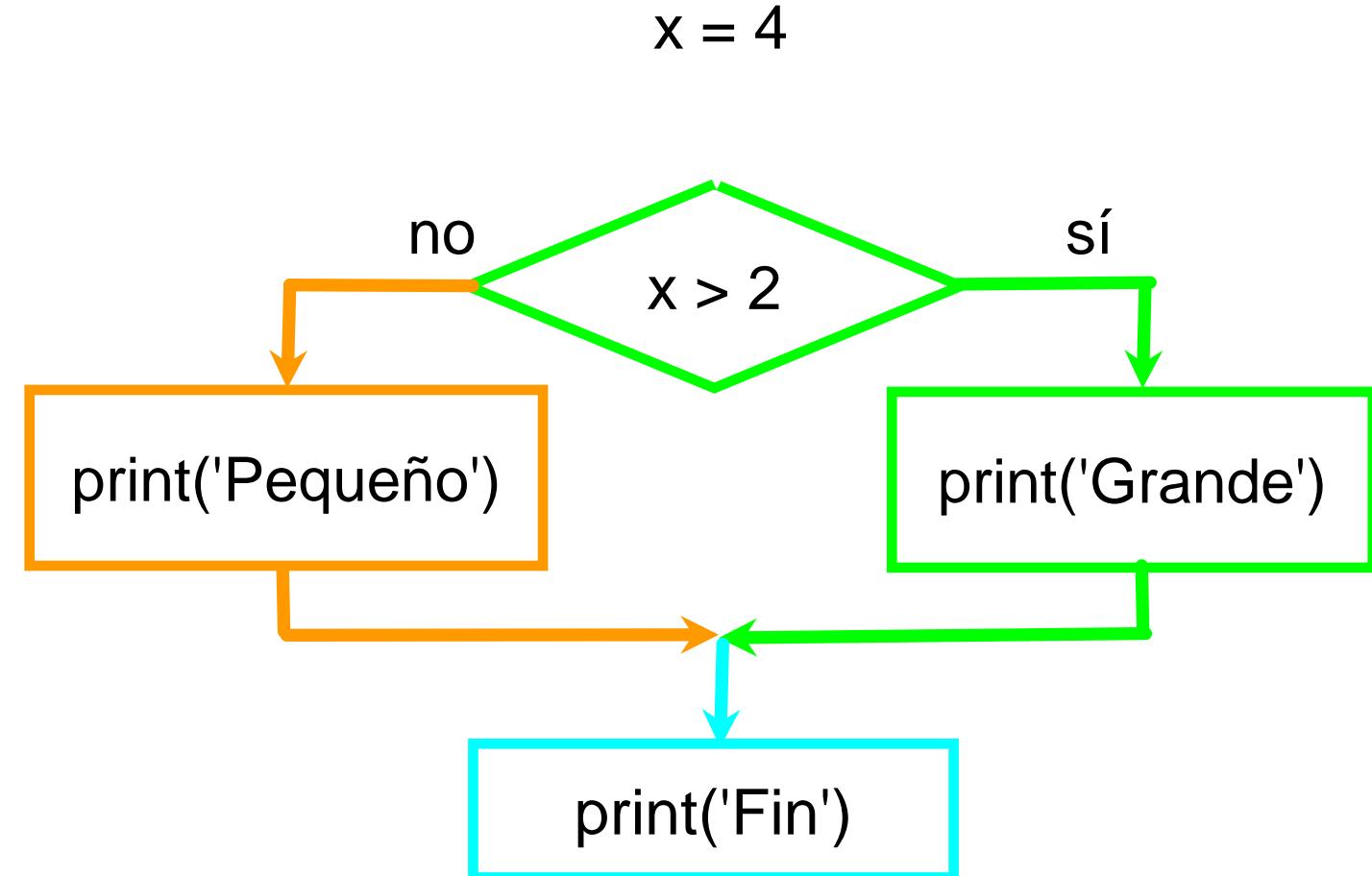
# Decisiones Anidadas

```
x = 42
if x > 1 :
    print('Mayor que 1')
    if x < 100 :
        print('Menor que 100')
print('Fin')
```



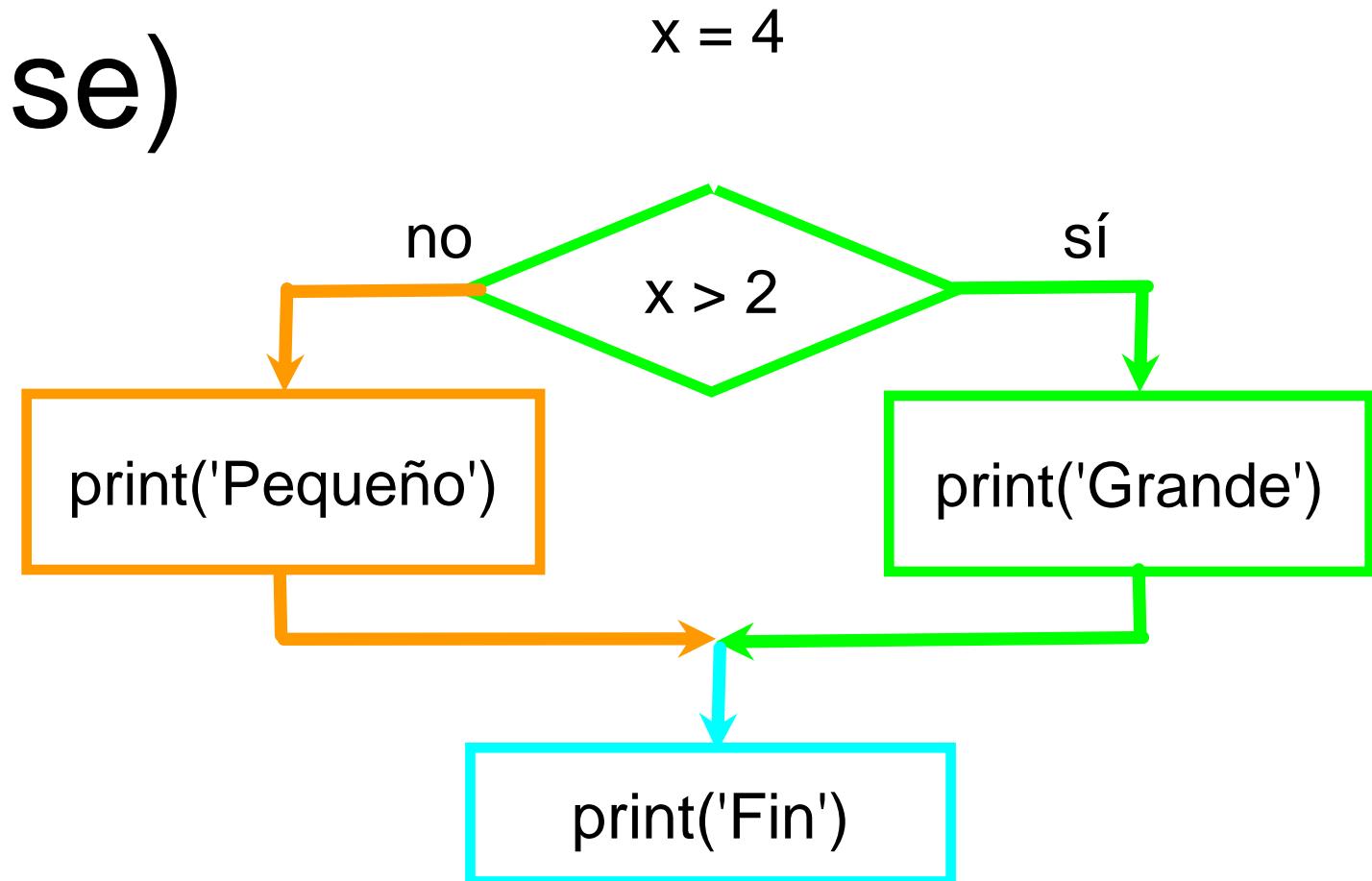
# Decisiones con alternativa (else)

- A veces querremos hacer algo si una expresión lógica es cierta y otra cosa si la expresión es falsa
- Es como una bifurcación en un camino – debemos **elegir** uno u otro pero no los dos



# Decisiones con alternativa (else)

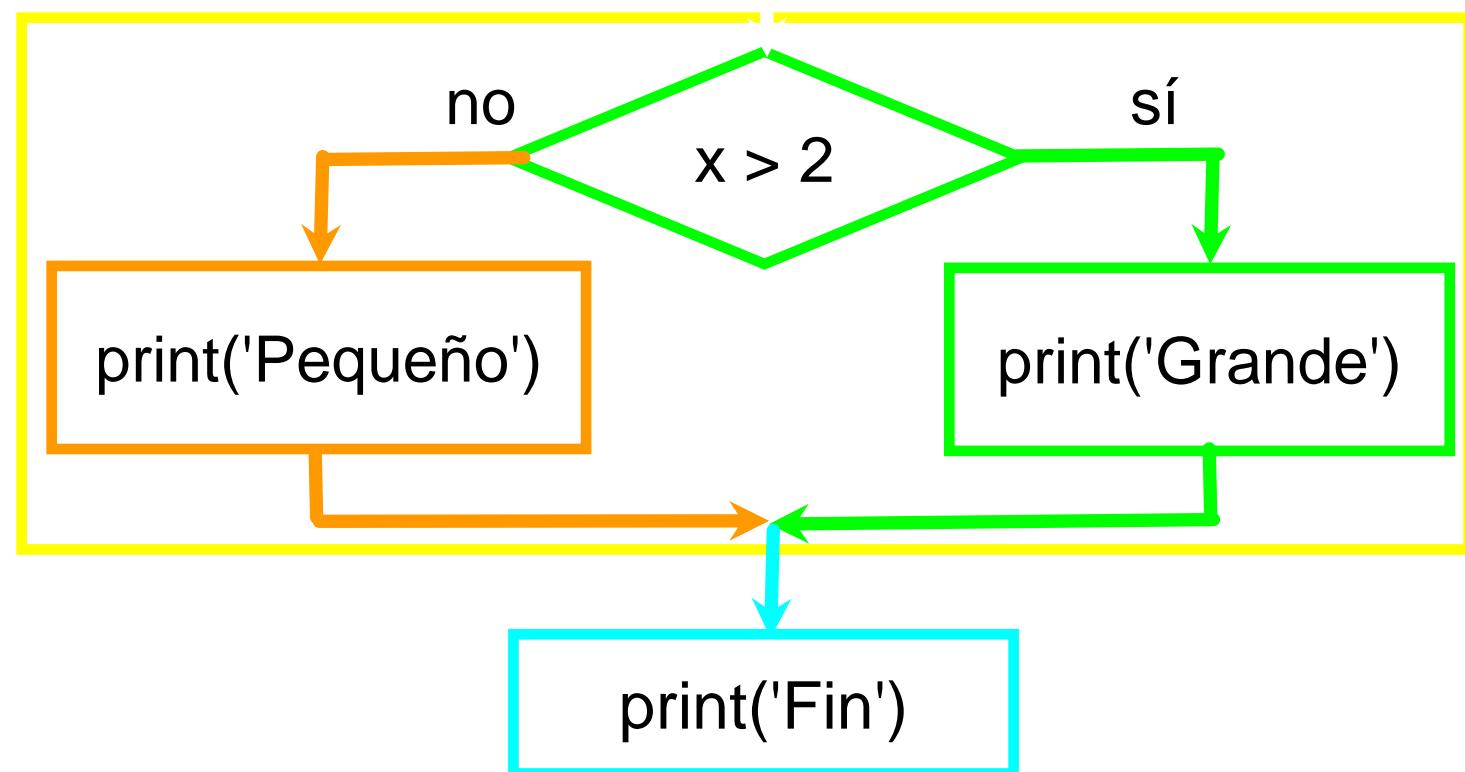
```
x = 4  
if x > 2 :  
    print('Grande')  
else :  
    print('Pequeño')  
print('Fin')
```



# Visualiza los bloques

```
x = 4  
if x > 2 :  
    print('Grande')  
else :  
    print('Pequeño')  
  
print('Fin')
```

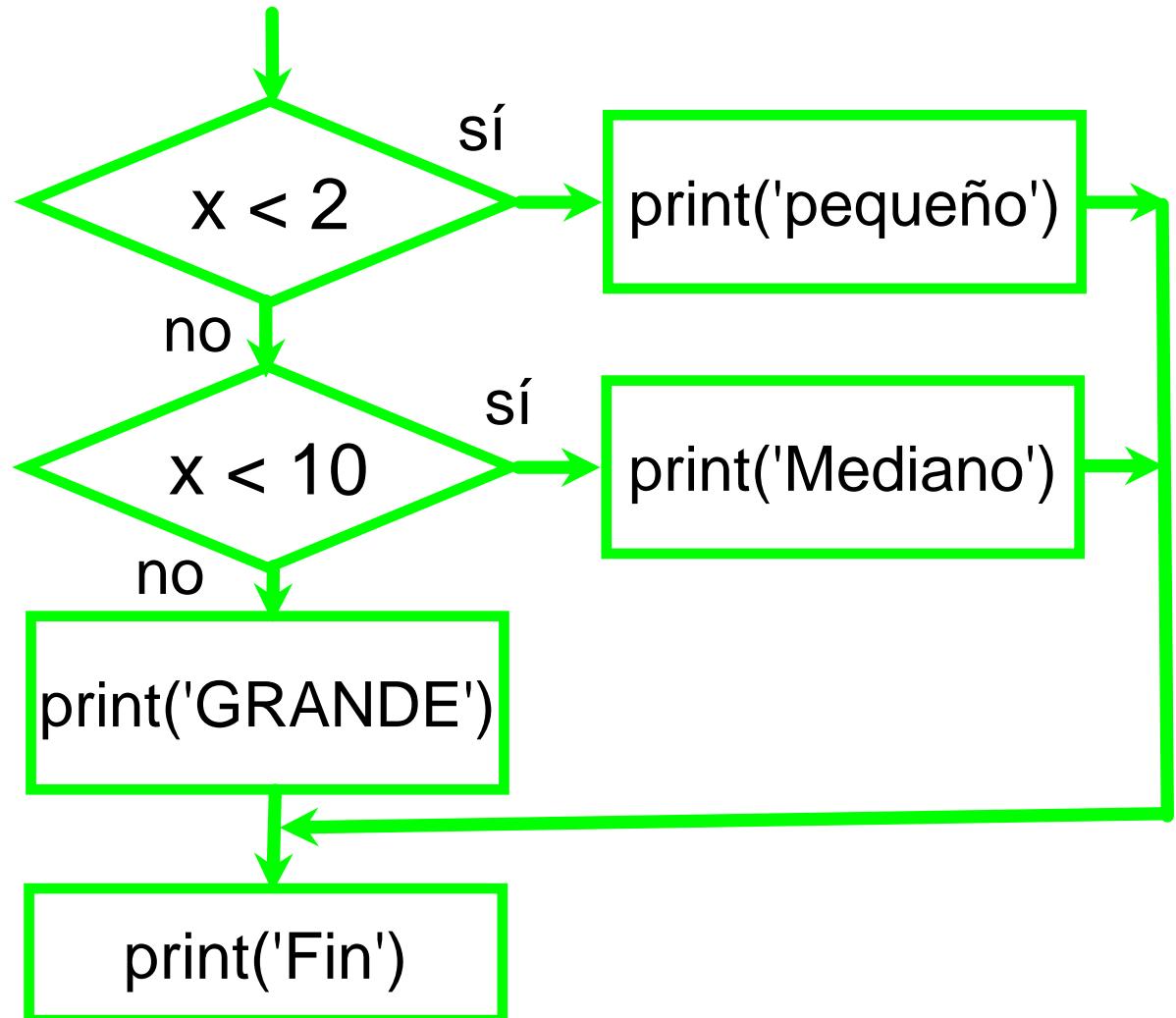
$x = 4$



# Más Estructuras Condicionales...

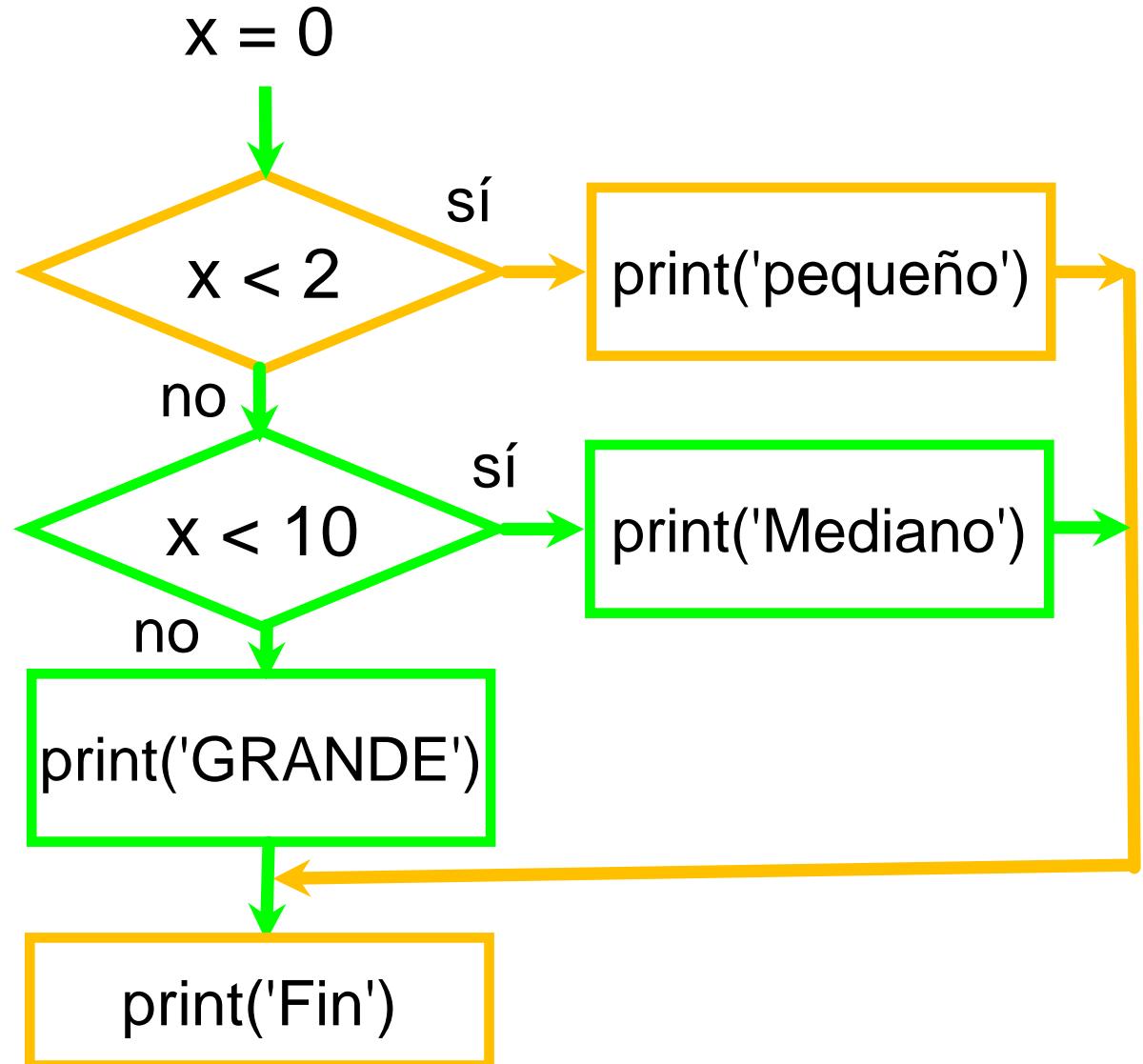
# Múltiples rutas

```
if x < 2 :  
    print('pequeño')  
elif x < 10 :  
    print('Mediano')  
else :  
    print('GRANDE')  
print('Fin')
```



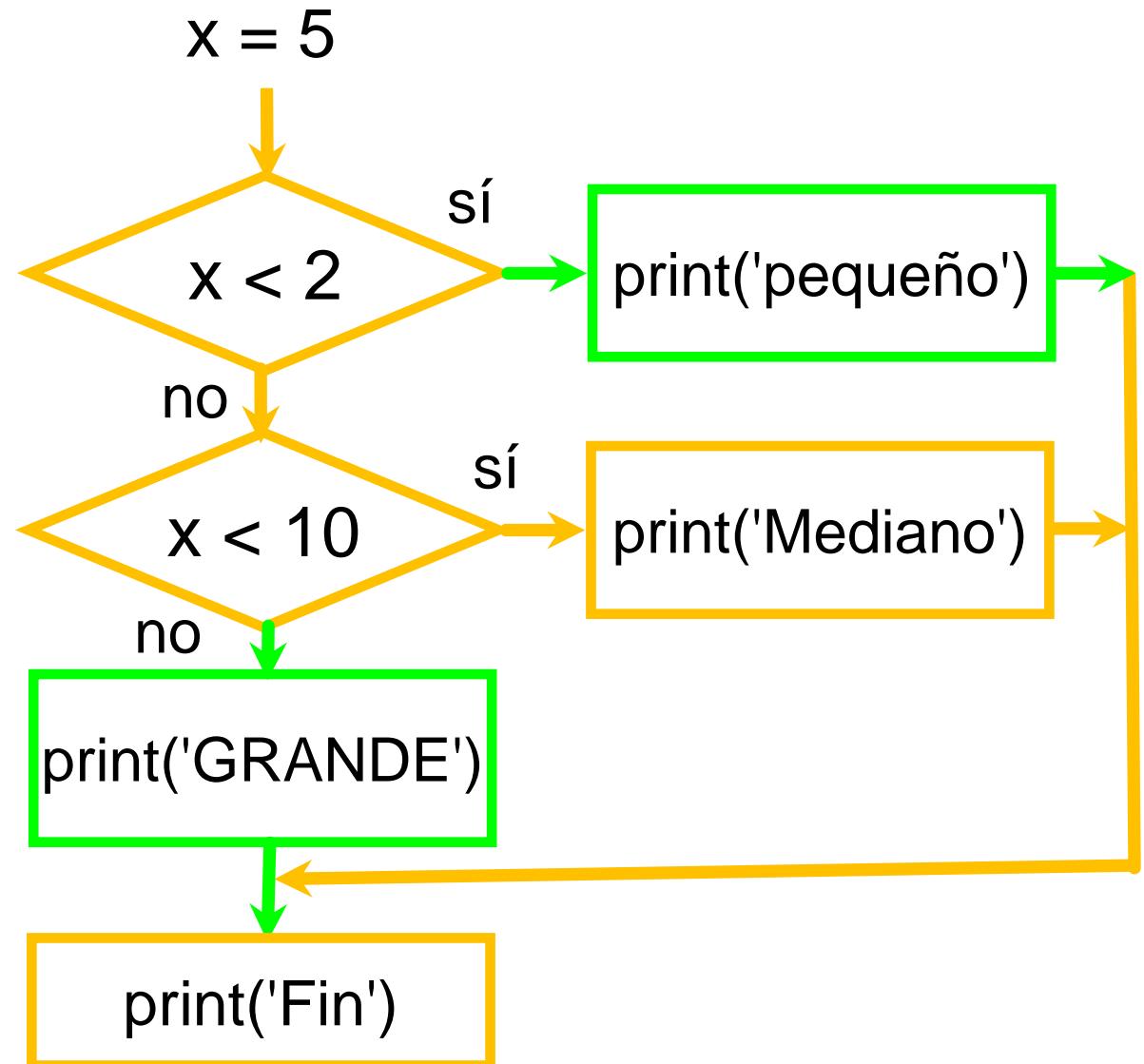
# Múltiples rutas

```
x = 0
if x < 2 :
    print('pequeño')
elif x < 10 :
    print('Mediano')
else :
    print('GRANDE')
print('Fin')
```



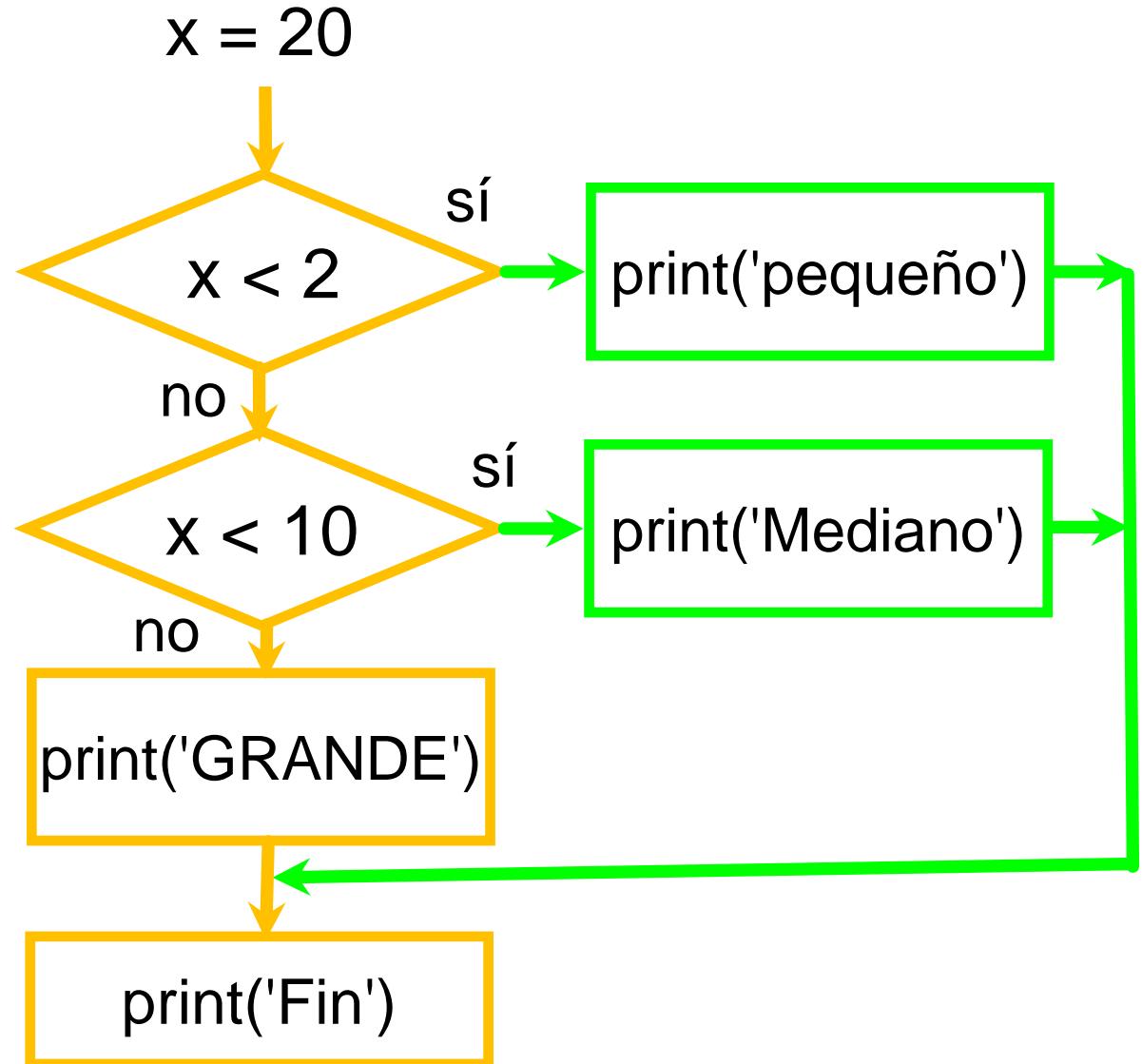
# Múltiples rutas

```
x = 5
if x < 2 :
    print('pequeño')
elif x < 10 :
    print('Mediano')
else :
    print('GRANDE')
print('Fin')
```



# Múltiples rutas

```
x = 20
if x < 2 :
    print('pequeño')
elif x < 10 :
    print('Mediano')
else :
    print('GRANDE')
print('Fin')
```



# Múltiples rutas

```
# Sin else
x = 5
if x < 2 :
    print('Pequeño')
elif x < 10 :
    print('Mediano')
print('Fin')
```

```
if x < 2 :
    print('Pequeño')
elif x < 10 :
    print('Mediano')
elif x < 20 :
    print('Grande')
elif x < 40 :
    print('Enorme')
elif x < 100:
    print('Prodigioso')
else :
    print('Gigantesco')
```

# Puzzles con múltiples rutas

¿Qué mensaje **nunca** se imprimirá independientemente del valor de x?

```
if x < 2 :  
    print('Menor que 2')  
elif x >= 2 :  
    print('2 o más')  
else :  
    print('Algo más')
```

```
if x < 2 :  
    print('Menor que 2')  
elif x < 20 :  
    print('Menor que 20')  
elif x < 10 :  
    print('Menor que 10')  
else :  
    print('Algo más')
```

# Operadores and y or

¿Y si queremos comprobar que se cumplen varias condiciones al mismo tiempo? Operador **and**

```
if x > 2 and x < 8:  
    print('Entre 2 y 8')
```

¿Y si queremos comprobar que se cumple alguna condición? Operador **or**

```
if x < 2 or x > 8:  
    print('Menor que 2 o mayor que 8')
```

# Operador not

Cuando buscamos que una condición no se cumpla

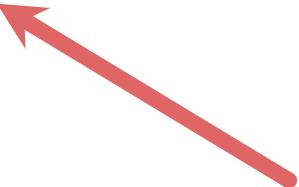
```
if not x > 2:  
    print('No mayor que 2')
```

# La estructura try / except

- Las secciones de código "**peligrosas**" se rodean con try y except
- Si el código dentro del try funciona – el except se **salta**
- Si el código dentro del try **falla** – se **ejecuta** el bloque del except

```
$ python3 notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
istr = int(astr)ValueError: invalid literal
for int() with base 10: 'Hola Bob'
```

```
$ cat notry.py
astr = 'Hola Bob'
istr = int(astr)
print('First', istr)
astr = '123'
istr = int(astr)
print('Second', istr)
```

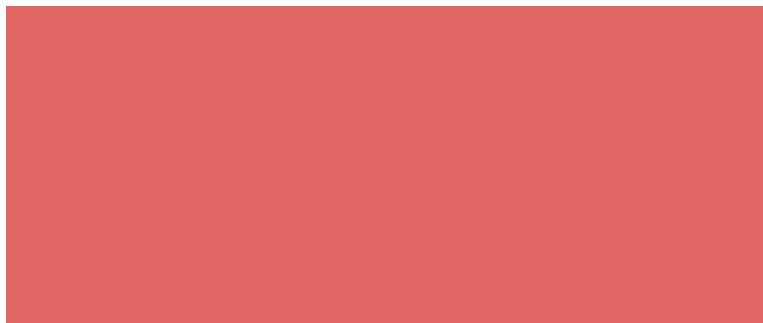


Fin

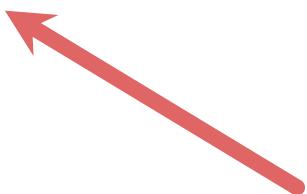
El  
programa  
para aquí



```
$ cat notry.py
astr = 'Hola Bob'
istr = int(astr)
```

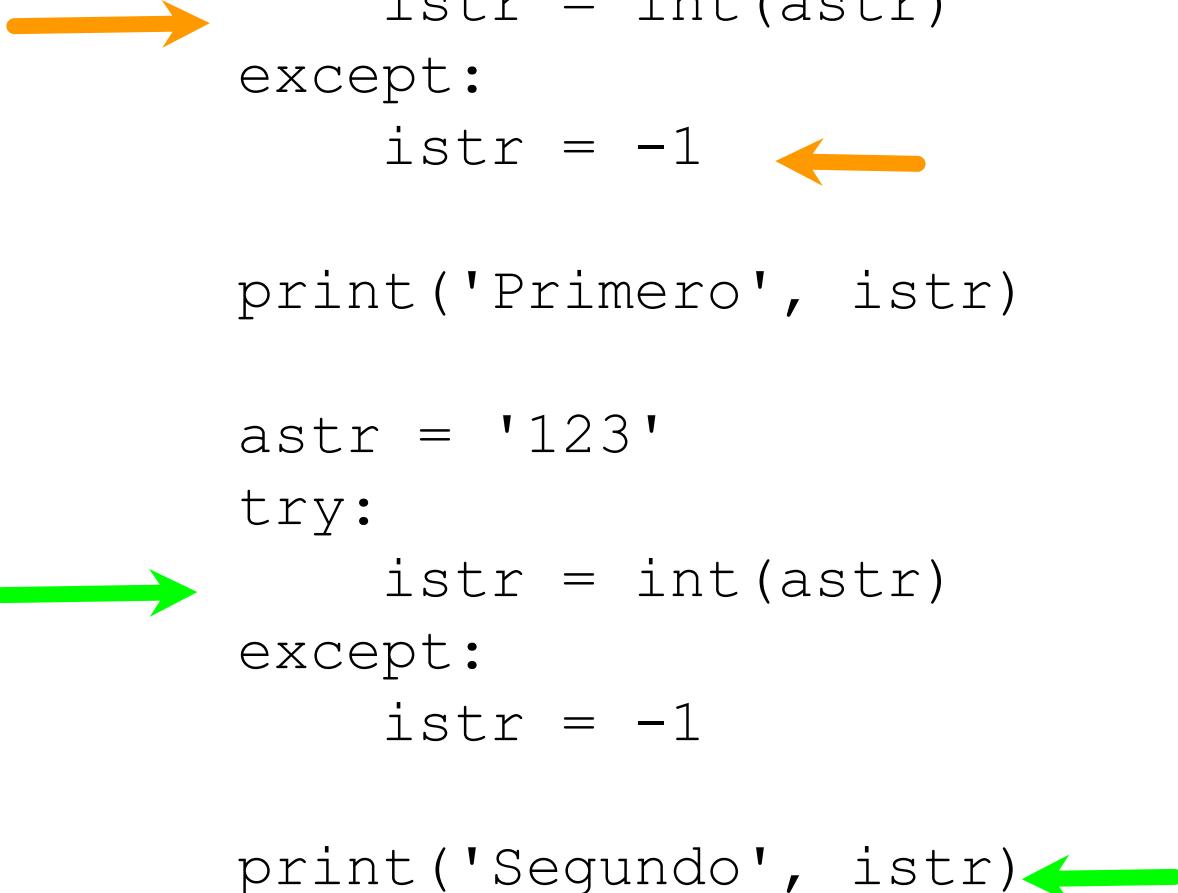


```
$ python3 notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
    istr = int(astr)
ValueError: invalid literal
for int() with base 10: 'Hola Bob'
```



Fin

```
astr = 'Hola Bob'  
try:  
    istr = int(astr)  
except:  
    istr = -1  
  
print('Primero', istr)  
  
astr = '123'  
try:  
    istr = int(astr)  
except:  
    istr = -1  
  
print('Segundo', istr)
```



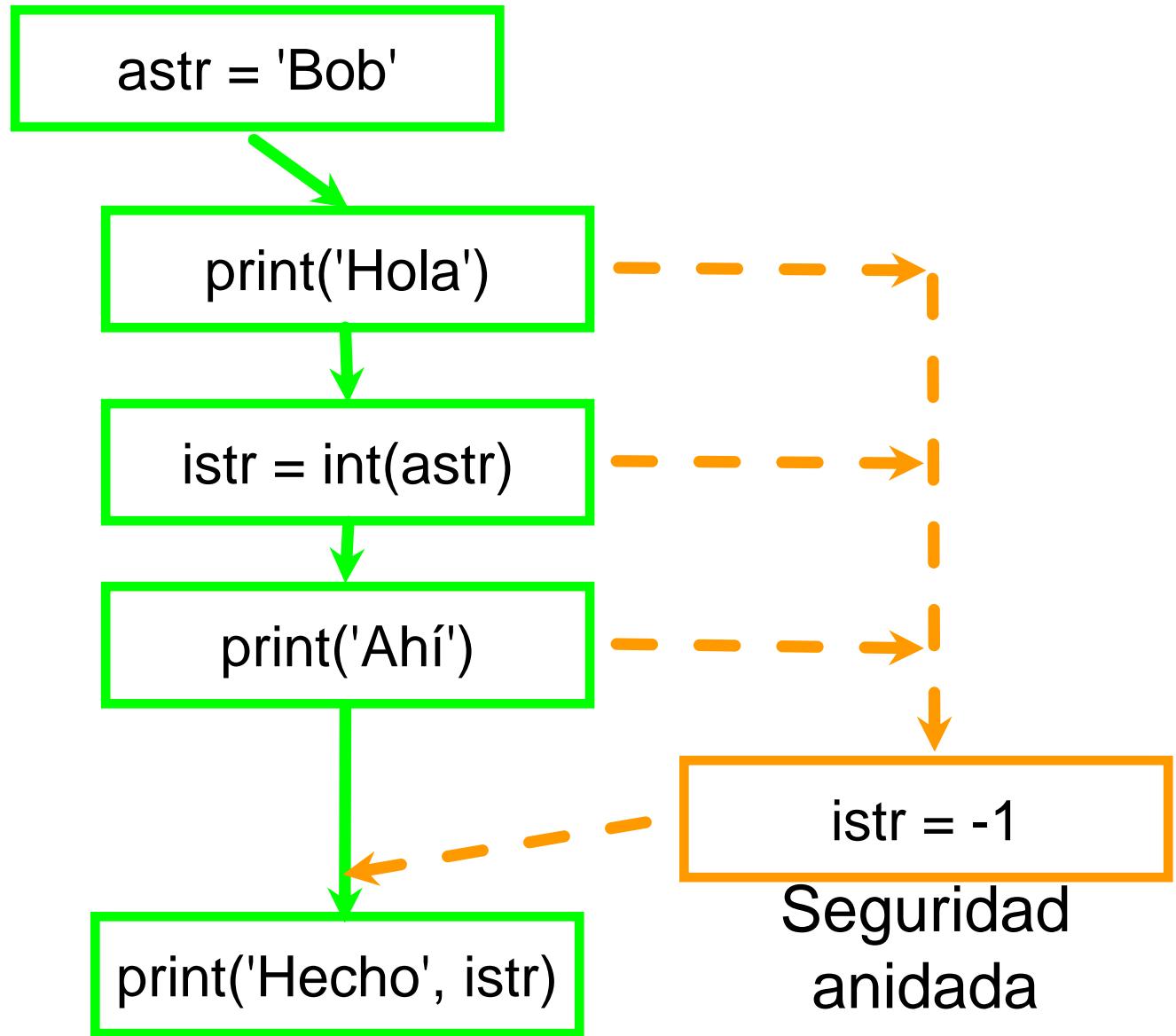
Cuando la primera conversion falla  
– salta al except y el programa sigue.

```
$ python tryexcept.py  
Primero -1  
Segundo 123
```

Cuando la segunda conversion funciona – salta el except y el programa sigue.

# try / except

```
astr = 'Bob'  
try:  
    print('Hola')  
    istr = int(astr)  
    print('Ahí')  
except:  
    istr = -1  
  
print('Hecho', istr)
```



# Ejemplo try / except

```
rawstr = input('Dime un número:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print('Bien hecho')
else:
    print('No es un número')
```

```
$ python3 trynum.py
Dime un número:42
Bien hecho
$ python3 trynum.py
Dime un número:forty-two
No es un número
$
```

# Resumen

- Operadores de comparación  
== <= >= > < !=
- Indentación
- Decisiones simples
- Decisiones con alternativa:  
if: and else:
- Decisiones anidadas
- Decisiones con múltiples ramas usando elif
- try / except para compensar errores

## Ejercicio 1

Reescribe el programa de pagos para incrementar en un factor de 1.5 las horas que se trabajen por encima de 40 horas (hasta 40 horas todas se cobran a precio normal)

Introduzca las horas: 45

Introduzca el precio/hora: 10

Total: 475.0

$$475 = 40 * 10 + 5 * 15$$

## Ejercicio 2

Reescribe el programa de pagos usando try y except para que se manejen adecuadamente entradas no numéricas (nota: no hace falta que se vuelvan a pedir los números si falla)

Introduzca las horas: 20

Introduzca el precio/hora: nueve

Error, por favor introduzca un número

Introduzca las horas: cuarenta

Error, por favor introduzca un número



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

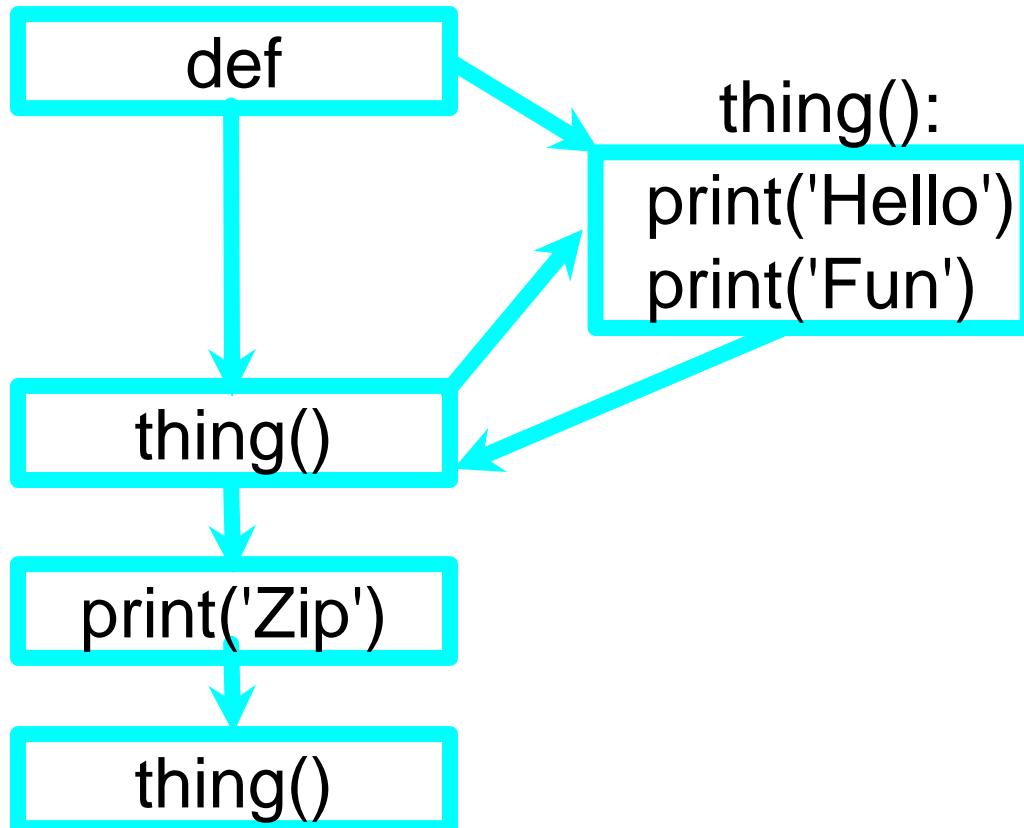
# Funciones

## Unidad 4

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Pasos almacenados (y reutilizados)



Programa:

```
def thing():
    print('Hello')
    print('Fun')
```

Salida:

```
thing()
print('Zip')
thing()
```

Hello  
Fun  
Zip  
Hello  
Fun

Llamaremos "**funciones**" a estas piezas reutilizables

# Funciones en Python

Hay dos tipos de funciones en Python.

- **Funciones internas** proporcionadas como parte de Python -  
print(), input(), type(), float(), int() ...
- Funciones que **definiremos nosotros mismos** y que  
podremos usar después

Trataremos a las funciones internas como "nuevas" palabras  
reservadas  
(es decir, evitaremos usarlas como nombres de variables)

# Definición de Funciones

- En Python una función es algún código reutilizable que toma **argumento(s)** como entradas, hace algún **procesamiento**, y **retorna** un resultado o resultados
- Las funciones se definen utilizando la palabra reservada **def**
- Podemos llamar/**invocar** a las funciones utilizando el **nombre** de la función, paréntesis y los argumentos en una expresión

**Asignación**      Argumento  
**big = max('Hello world')**

'w'      Resultado

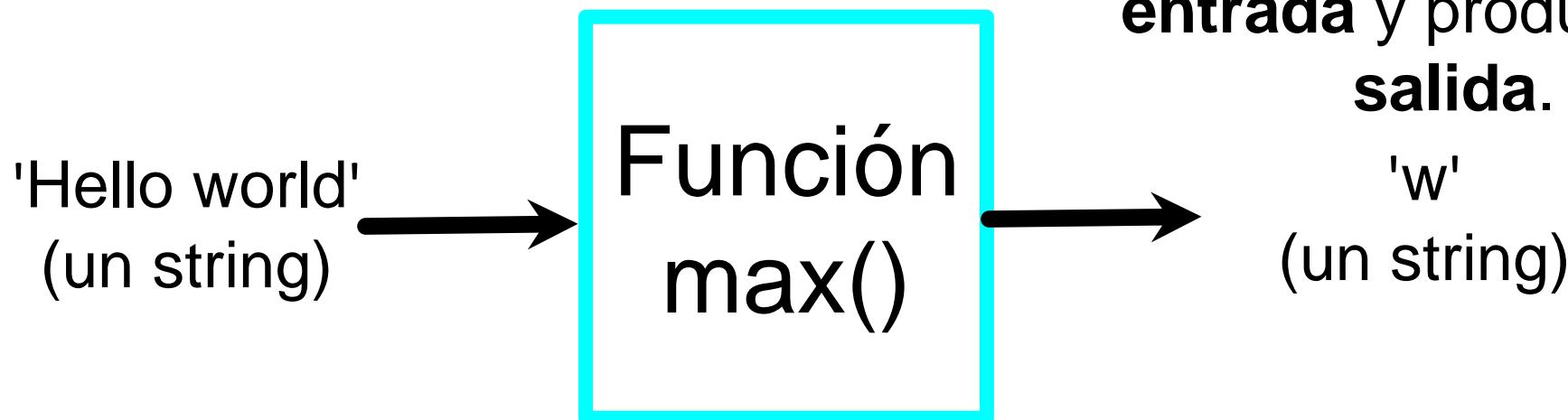
```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

>>>
```

# Función max

```
>>> big = max('Hello world')  
>>> print(big)  
w
```

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.



Guido escribió este código

# Función max

```
>>> big = max('Hello world')
>>> print(big)
w
```

'Hello world'  
(un string)

```
def max(inp):
    blah
    blah
    for x in inp:
        blah
        blah
```

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.

'w'  
(un string)

Guido escribió este código

# Conversiones de tipo

- Cuando pones un entero y un flotante en una expresión, el entero es convertido implícitamente a flotante
- Podemos controlar esto con las funciones internas **int()** y **float()**

```
>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>> print(1 + 2 * float(3) / 4 - 5)
-2.5
>>>
```

# Conversiones de cadenas

- También se pueden usar `int()` y `float()` para conversiones entre **strings y enteros**
- Obtendremos un error si el string no tiene un **número bien formado**

```
>>> sval = '123'  
>>> type(sval)  
<class 'str'>  
>>> print(sval + 1)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: cannot concatenate 'str'  
and 'int'  
>>> ival = int(sval)  
>>> type(ival)  
<class 'int'>  
>>> print(ival + 1)  
124  
>>> nsv = 'hello bob'  
>>> niv = int(nsv)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ValueError: invalid literal for int()
```

# Funciones escritas por nosotros...

# Construyendo nuestras propias funciones

- Creamos una nueva función utilizando la palabra clave **def** seguida por el **nombre** de la función y por los **parámetros** (opcionalmente) entre paréntesis
- **Indentamos** el código de la función
- Esto define la función pero **no ejecuta** el cuerpo de la función

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

```
x = 5  
print('Hello')
```

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all day.')  
  
print('Yo')  
x = x + 2  
print(x)
```

```
print("I'm a lumberjack, and I'm okay.")  
print('I sleep all night and I work all day.')
```

Hello  
Yo

7

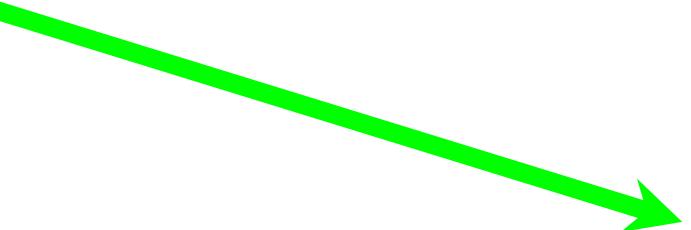
# Definiciones y Uso

- Una vez que hemos definido una función, podemos llamarla (o **invocarla**) tantas veces como queramos
- En esto consiste el patrón **almacenar y reutilizar**

```
x = 5
print('Hello')

def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

print('Yo')
print_lyrics()
x = x + 2
print(x)
```



Hello  
Yo  
I'm a lumberjack, and I'm okay.  
I sleep all night and I work all day.  
7

# Argumentos

- Un argumento es un **valor** que pasamos a la función como **entrada** cuando llamamos a la función
- Utilizamos argumentos para que la función pueda hacer **diferentes cosas** cuando la llamemos en **diferentes ocasiones**
- Ponemos los argumentos en paréntesis después del **nombre** de la función

```
big = max('Hello world')
```



Argumento

# Parámetros

- Un parámetro es una **variable** que usamos en la definición de una función. Es un **"manejador"** que permite al código de la función **acceder a los argumentos** utilizados en una invocación en particular de una función.

```
>>> def greet(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     else:  
...         print('Hello')  
...  
>>> greet('en')  
Hello  
>>> greet('es')  
Hola  
>>> greet('fr')  
Bonjour  
>>>
```

# Valores de Retorno

- Frecuentemente una función toma sus argumentos, hace algún cálculo, y **retorna un valor** para ser usado como resultado de la función en la expresión de llamada. La palabra clave **return** se utiliza para esto.

```
def greet():
    return "Hello"
print(greet(), "Glenn")
print(greet(), "Sally")
```

Hello Glenn  
Hello Sally

# Valores de Retorno

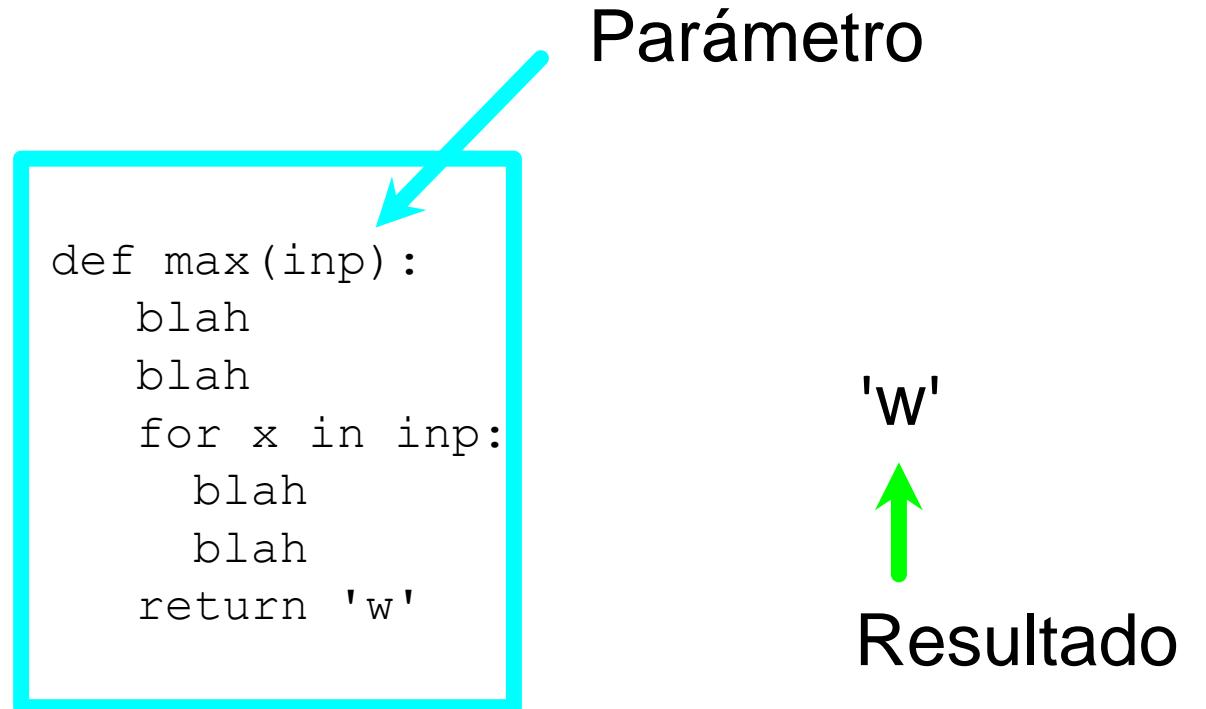
- Una función “fructífera” (o no vacía) es aquella que produce un resultado (o valor de retorno)
- La sentencia `return` finaliza la ejecución de la función y "envía de vuelta" el resultado de la función

```
>>> def greet(lang):  
...     if lang == 'es':  
...         return 'Hola'  
...     elif lang == 'fr':  
...         return 'Bonjour'  
...     else:  
...         return 'Hello'  
...  
>>> print(greet('en'), 'Glenn')  
Hello Glenn  
>>> print(greet('es'), 'Sally')  
Hola Sally  
>>> print(greet('fr'), 'Michael')  
Bonjour Michael  
>>>
```

# Argumentos, Parámetros y Resultados

```
>>> big = max('Hello world')
>>> print(big)
w
```

'Hello world'  
Argumento



# Múltiples Parámetros / Argumentos

- Podemos definir más de un parámetro en la definición de una función
- Simplemente añadimos más argumentos cuando llamemos a la función
- Los argumentos y parámetros se **"emparejan"** por el orden usado y deben coincidir en número

```
def addtwo(a, b):  
    added = a + b  
    return added
```

```
x = addtwo(3, 5)  
print(x)
```

8

# Funciones void (vacías)

- Cuando una función no retorna un valor, se dice que estamos llamando a una función vacía (`void`)
- Las funciones que retornan valores se denominan "no vacías"
- Esto no quiere decir que hayan "fallado", sino que no retornan nada, simplemente **ejecutan un conjunto de instrucciones**

# Función o No función...

- Organiza tu código en "**párrafos**" – captura una **idea completa** y "dale un nombre"
- No te repitas – haz que funcione una vez y entonces **reutiliza**
- Si algo se vuelve demasiado largo o complejo, **divídelo** en trozos lógicos y pon estos trozos en funciones
- Cuando tengas muchas funciones con código que utilizas una y otra vez, haz una **librería** – quizás compartir con otros...

# Resumen

- Funciones
- Funciones internas
- Conversiones de tipo (int, float)
- Conversiones de cadenas
- Parámetros
- Argumentos
- Resultados(funciones no vacías)
- Funciones vacías (void)
- ¿Por qué utilizar funciones?

## Ejercicio

Reescribe el cálculo del pago por horas de la unidad anterior y escribe una función llamada calculapago que toma dos parámetros (horas y precio) y retorna el total a pagar.

Introduzca las horas: 45

Introduzca el precio/hora: 10

Total: 475.0

$$475 = 40 * 10 + 5 * 15$$

## Ejercicio

Es decir, debes definir una función con la siguiente cabecera:

```
def calculapago(horas,precio_hora):
```

La función realizará el cálculo del total a pagar y acabará retornando el pago total con **return**

Desde fuera de la función, deberías poder hacer algo así como:

```
print(calculapago(horas,precio_hora))
```



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Funciones (lambda)

## Unidad 4

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Funciones lambda

- Las denominadas funciones “lambda” son una forma rápida y concisa de trabajar con funciones en Python
- Son funciones “anónimas” (no tienen nombre) con una sintaxis más restrictiva pero más concisa que las funciones normales

Primer ejemplo directamente en el intérprete:

```
>>> (lambda x: x + 1)(2)
```

Salida: 3

¿Qué hemos hecho aquí?

Hemos definido una función que toma como argumento “x” y devuelve “x+1”. A esta función la llamamos con el valor 2, devolviendo 2+1 (3)

# Funciones lambda

- Podemos dar nombre a las funciones lambda

```
>>> add_one = lambda x: x + 1
```

Y después usarlas como cualquier otra función

```
>>> add_one(2)
```

- Esto sería equivalente a haber definido la siguiente función:

```
def add_one(x):
```

```
    return x + 1
```

# Funciones lambda

- Podemos definir funciones lambda con más de un argumento

```
>>> full_name = lambda first, last: f'Full name: {first.title()} {last.title()}'  
>>> full_name('guido', 'van rossum')
```

'Full name: Guido Van Rossum'

Fijaos en que no es necesario poner ( ) para los argumentos first y last y que tampoco necesitamos return para indicar qué devuelve la función

# Características de las funciones lambda

- Solo pueden contener expresiones y no sentencias en su cuerpo
- Se escriben como una sola línea.
  - Aunque el código esté dividido en varias líneas usando paréntesis o string multilíneas, conceptualmente está en una sola línea
- Pueden ser llamadas inmediatamente
- No se pueden utilizar anotaciones de tipo
  - (esto es, no se pueden indicar los tipos de los argumentos de la función)

# Cuándo usar o no funciones lambda

- A veces se abusa de las funciones lambda cuando su uso dificulta la lectura
- En otras ocasiones su uso puede hacer más fácil o con un estilo más “funcional” el código desarrollado
  - Las funciones lambda se usan de forma intrínseca con funciones como map() y filter(). Prueba estos ejemplos:
    - `list(map(lambda x: x.upper(), ['cat', 'dog', 'cow']))`
    - `list(filter(lambda x: 'o' in x, ['cat', 'dog', 'cow']))`
  - Se usan también en algunas funciones que reciben funciones como argumentos:

```
ids = ['id1', 'id2', 'id30', 'id3', 'id22', 'id100']
>>> print(sorted(ids)) # Ordenación lexicográfica
['id1', 'id100', 'id2', 'id22', 'id3', 'id30']
>>> sorted_ids = sorted(ids, key=lambda x: int(x[2:])) # Ordenación por números
```

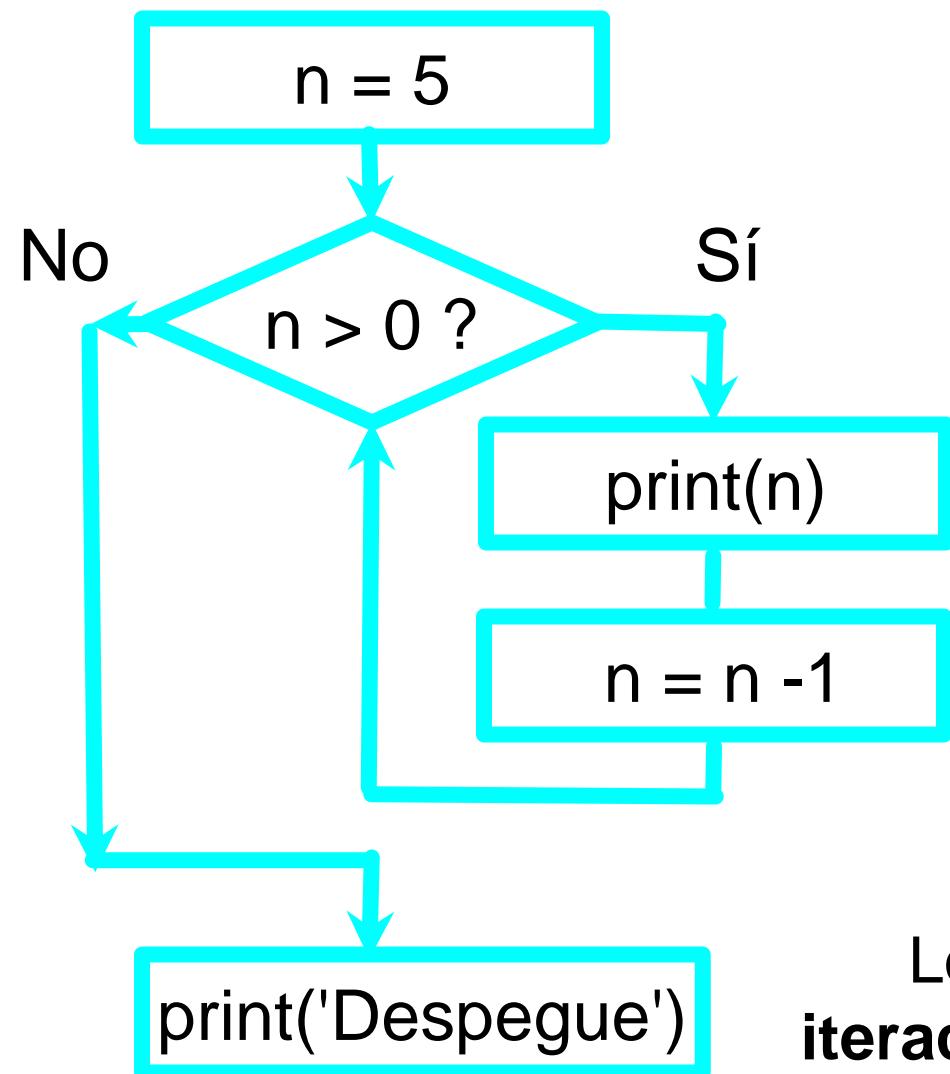
# Bucles

## Unidad 5

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Pasos Repetidos



Programa:

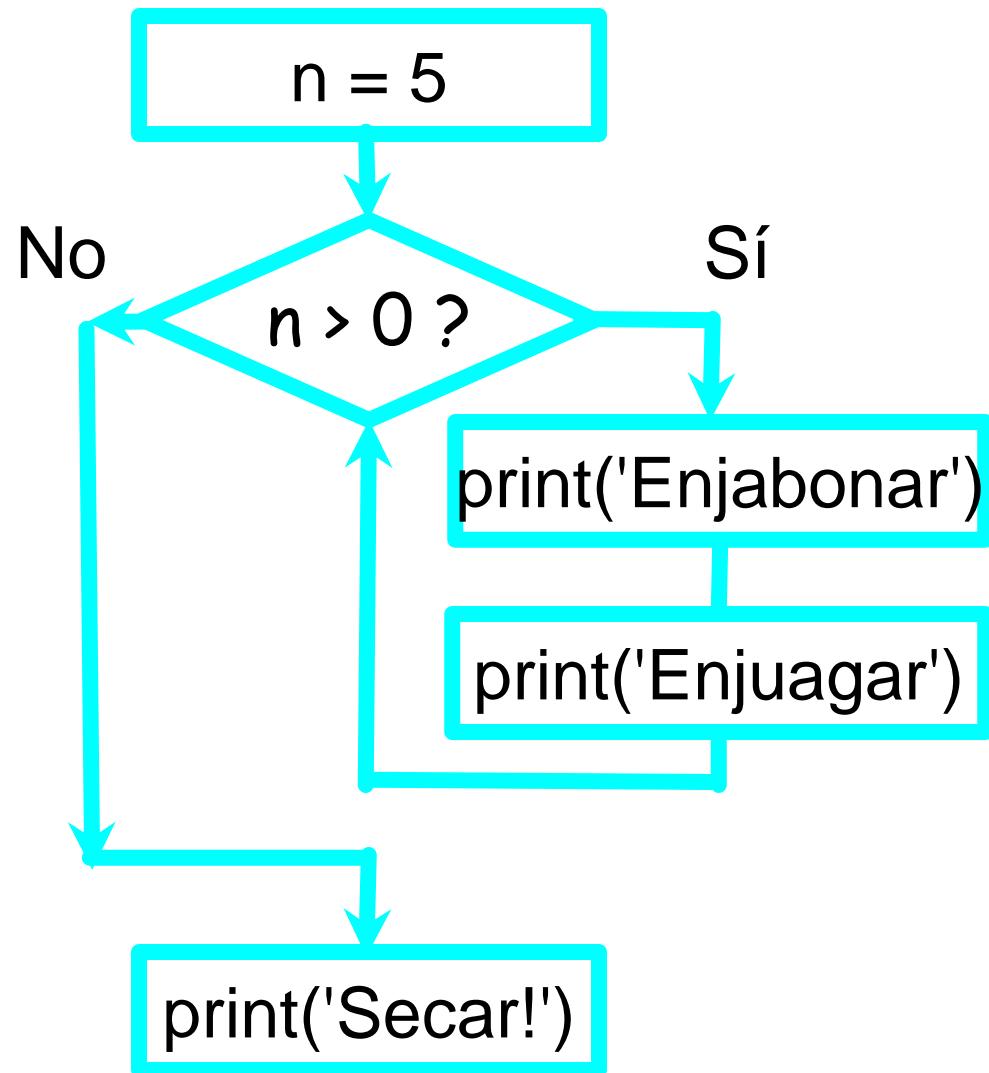
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print ('Despegue! ')
print(n)
```

Salida:

5  
4  
3  
2  
1  
Despegue!  
0

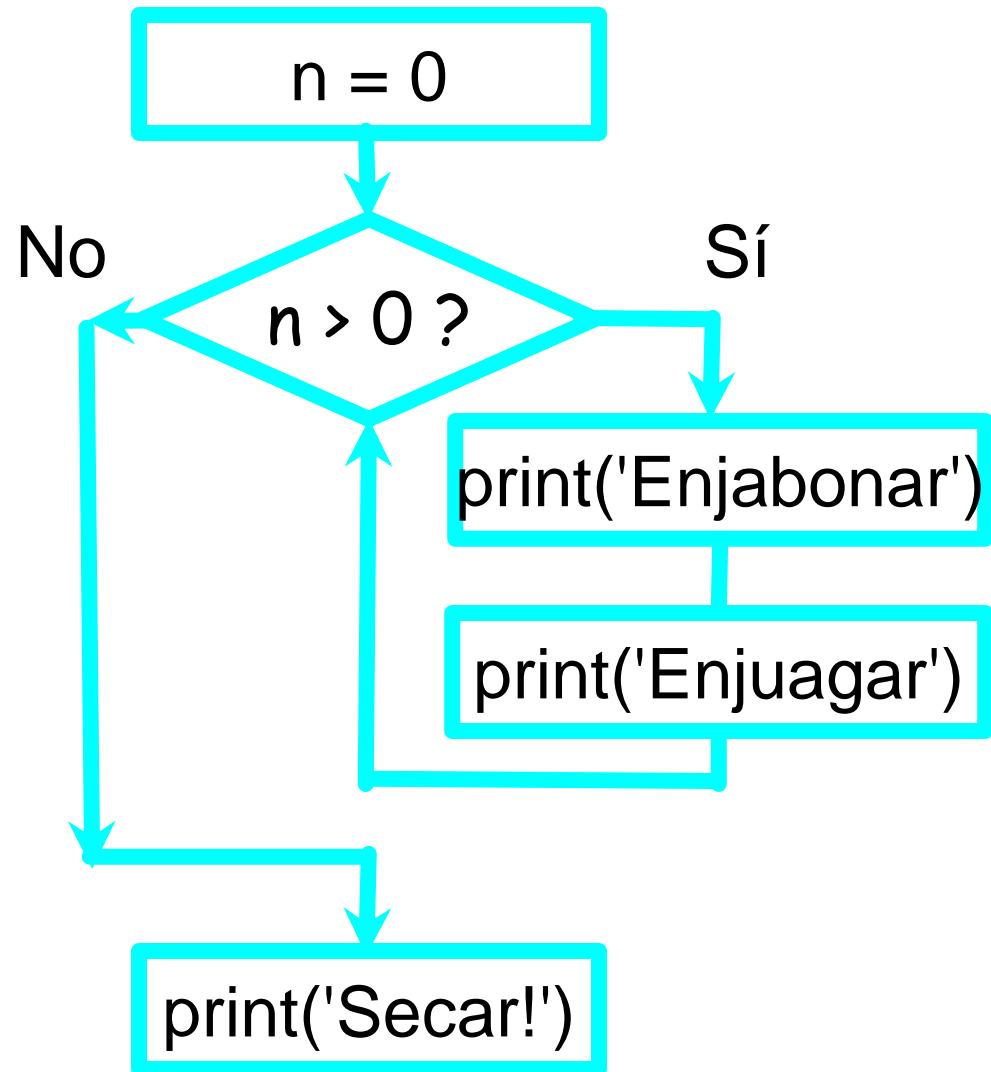
Los bucles (pasos repetidos) tienen **variables de iteración** que **cambian** cada vez que se ejecuta el bucle (iteración). Frecuentemente estas variables de iteración forman una **secuencia de números**.

# Un bucle infinito



```
n = 5
while n > 0 :
    print('Enjabonar')
    print('Enjuagar')
    print('Secar!')
```

¿Qué está mal en este bucle?



# Otro bucle

```
n = 0  
while n > 0 :  
    print('Enjabonar')  
    print('Enjuagar')  
    print('Secar')
```

¿Qué hace este bucle?

# Rompiendo un buce

La sentencia **break** finaliza el bucle actual y "salta" a la sentencia inmediatamente posterior al mismo

Es como comprobar la condición del bucle en cualquier parte del cuerpo del bucle

```
while True:  
    line = input('> ')  
    if line == 'hecho' :  
        break  
    print(line)  
print('Hecho!')  
  
> hola  
hola  
> fin  
fin  
> hecho  
Hecho!
```

# Rompiendo un buce

La sentencia **break** finaliza el bucle actual y "salta" a la sentencia inmediatamente posterior al mismo

Es como comprobar la condición del bucle en cualquier parte del cuerpo del bucle

```
while True:  
    line = input('> ')  
    if line == 'hecho' :  
        break  
    print(line)  
print('Hecho!')
```

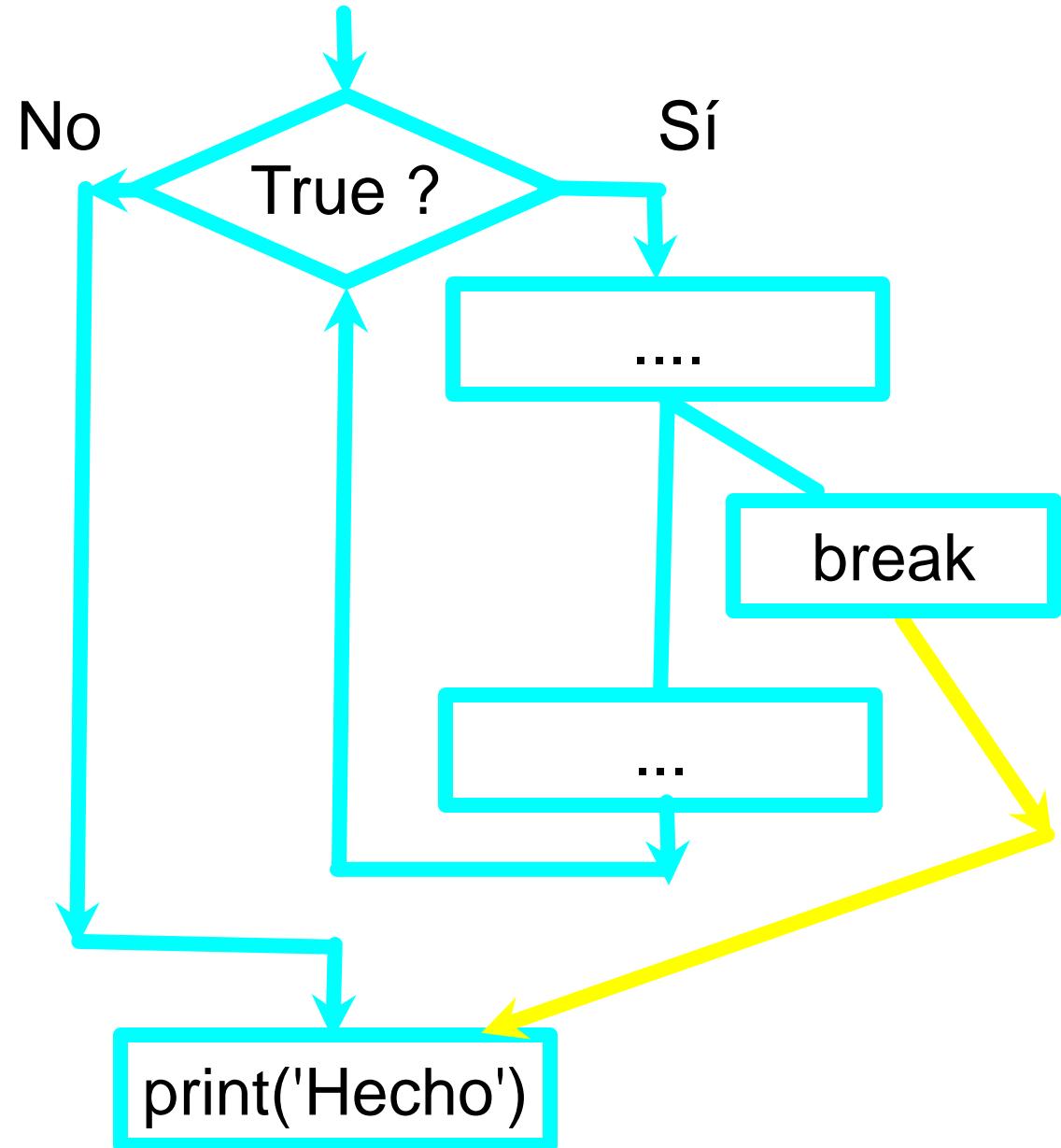
> hola  
hola  
> fin  
fin  
> hecho  
Hecho!



```
while True:  
    line = input('> ')  
    if line == 'hecho' :  
        break  
    print(line)  
print('Hecho!')
```



[http://en.wikipedia.org/wiki/Transporter\\_\(Star\\_Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



# Terminando una iteración con continue

La sentencia **continue** finaliza la iteración actual y salta al inicio del bucle para comenzar la próxima iteración

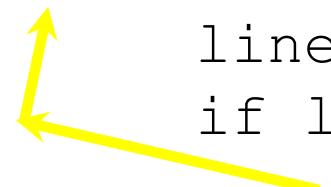
```
while True:  
    line = input('> ')  
    if line[0] == '#':  
        continue  
    if line == 'hecho':  
        break  
    print(line)  
print('Hecho!')
```

```
> hola  
hola  
> # ignora esto  
> imprime esto!  
imprime esto!  
> hecho  
Hecho!
```

# Terminando una iteración con continue

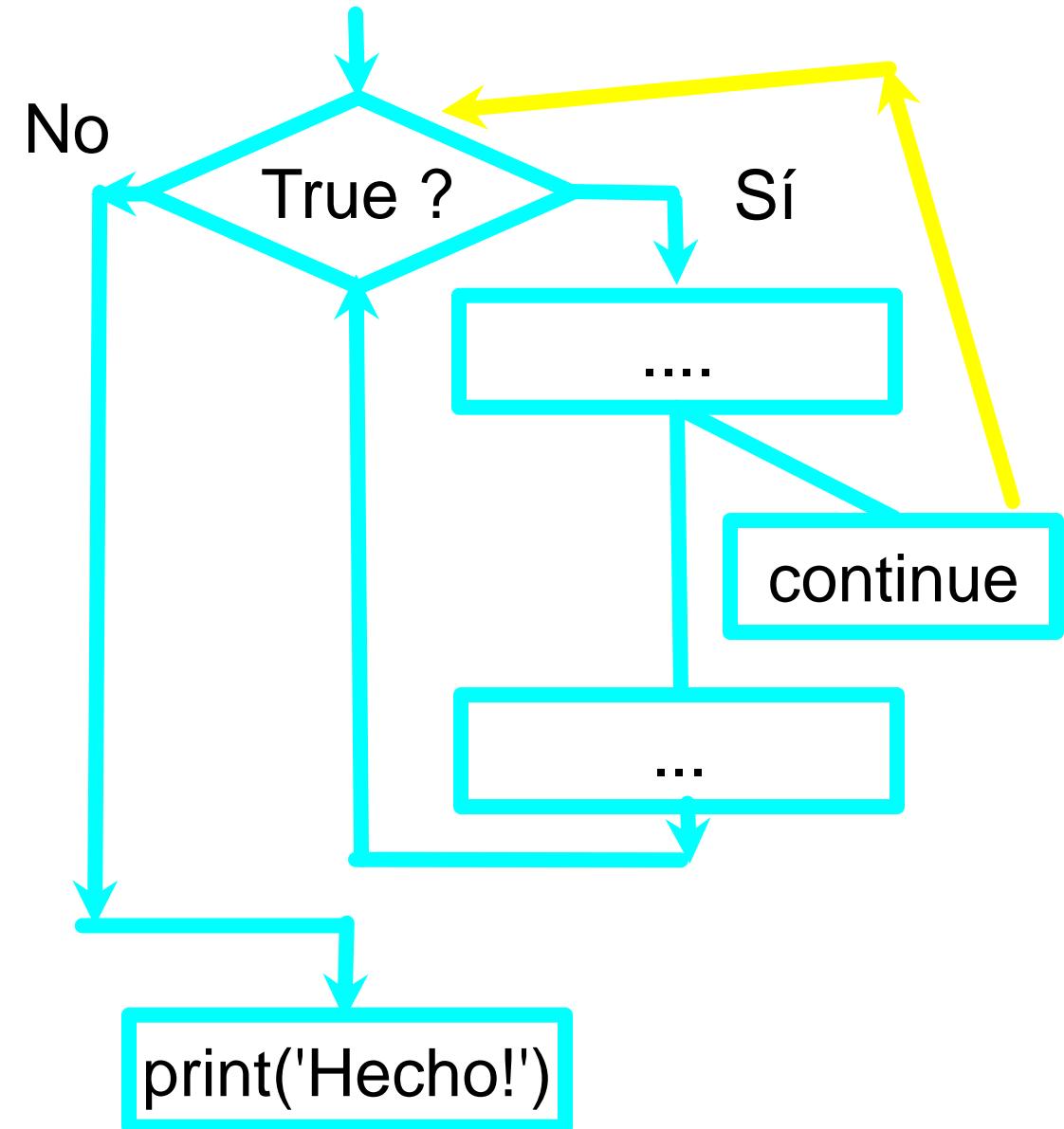
La sentencia **continue** finaliza la iteración actual y salta al inicio del bucle para comenzar la próxima iteración

```
while True:  
    line = input('> ')  
    if line[0] == '#':  
        continue  
    if line == 'hecho':  
        break  
    print(line)  
print('Hecho!')
```



```
> hola  
hola  
> # ignora esto  
> imprime esto!  
Imprime esto!  
> hecho  
Hecho!
```

```
while True:  
    line = input ('> ')  
    if line[0] == '#':  
        continue  
    if line == 'hecho':  
        break  
    print(line)  
print('Hecho!')
```



# Bucles Indefinidos

Los **bucles while** tienen un número de iteraciones indefinidas porque se ejecutan hasta que una condición lógica se vuelve falsa (**False**)

En los bucles anteriores era bastante fácil ver si terminarían o se convertirían en bucles "infinitos"

A veces es más difícil estar seguro de cuándo terminará un bucle

# Bucles Definidos

- Iterando sobre un conjunto de elementos...

# Bucles Definidos

- Frecuentemente tenemos una lista de elementos – por ejemplo, las líneas en un fichero – un **conjunto finito** de "cosas"
- En Python podemos escribir un bucle que itere sobre cada uno de estos elementos utilizando la construcción **for**
- Estos bucles se denominan “**bucles definidos**” porque ejecutan un número exacto de iteraciones
- Podemos decir que los “bucles definidos iteran sobre los miembros de un conjunto”

# Un bucle definido simple

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Despegue! ')
```

5  
4  
3  
2  
1  
Despegue!

# Un bucle definido con strings

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Feliz Año Nuevo:', friend)
print('Hecho!')
```

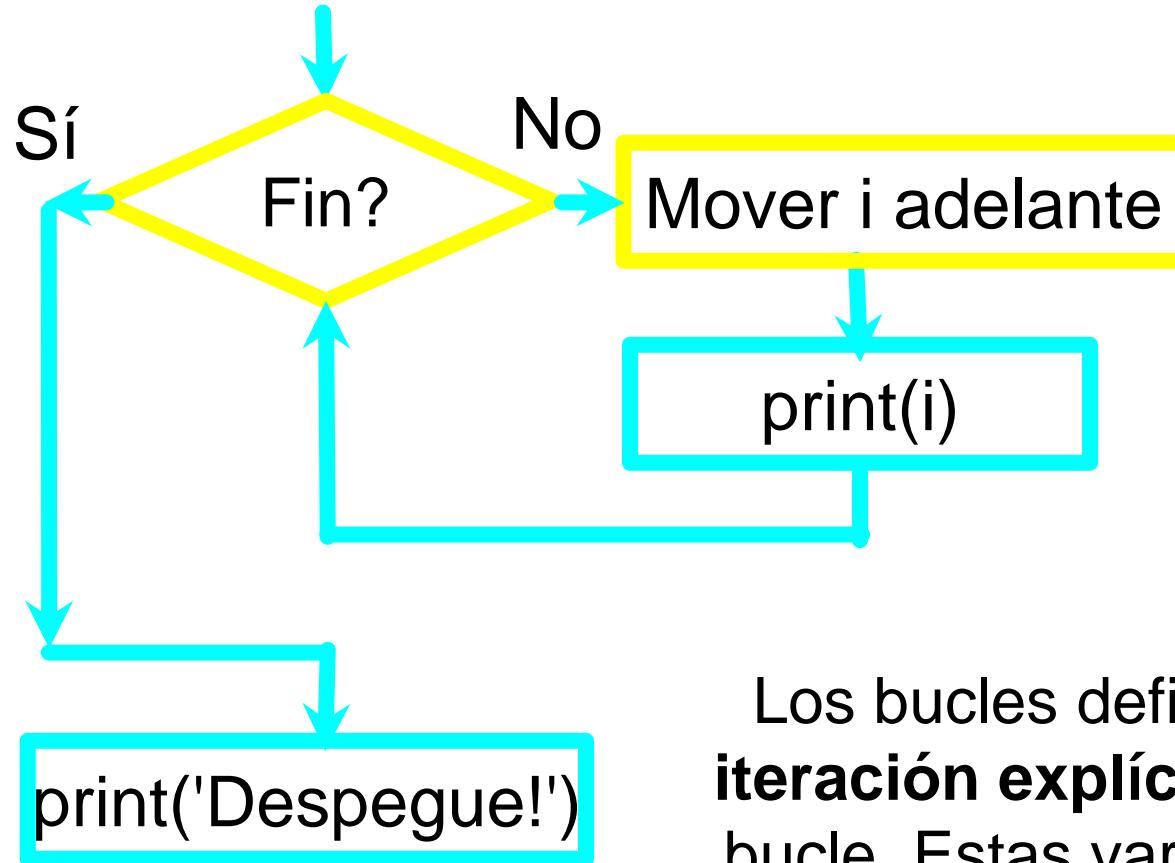
Feliz Año Nuevo: Joseph

Feliz Año Nuevo : Glenn

Feliz Año Nuevo : Sally

Hecho!

# Un bucle definido simple



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Despegue!')
```

5  
4  
3  
2  
1  
Despegue!

Los bucles definidos (**bucles for**) tienen **variables de iteración explícitas** que cambian en cada iteración del bucle. Estas variables de iteración se mueven a través de la **secuencia o conjunto**.

# Mirando in...

- La variable de iteración “**itera**” a través de la secuencia (conjunto ordenado)
- El bloque (cuerpo) de código es **ejecutado** una vez **para cada valor** en (in) la secuencia
- La variable de iteración se mueve a través de todos los valores en (in) la secuencia

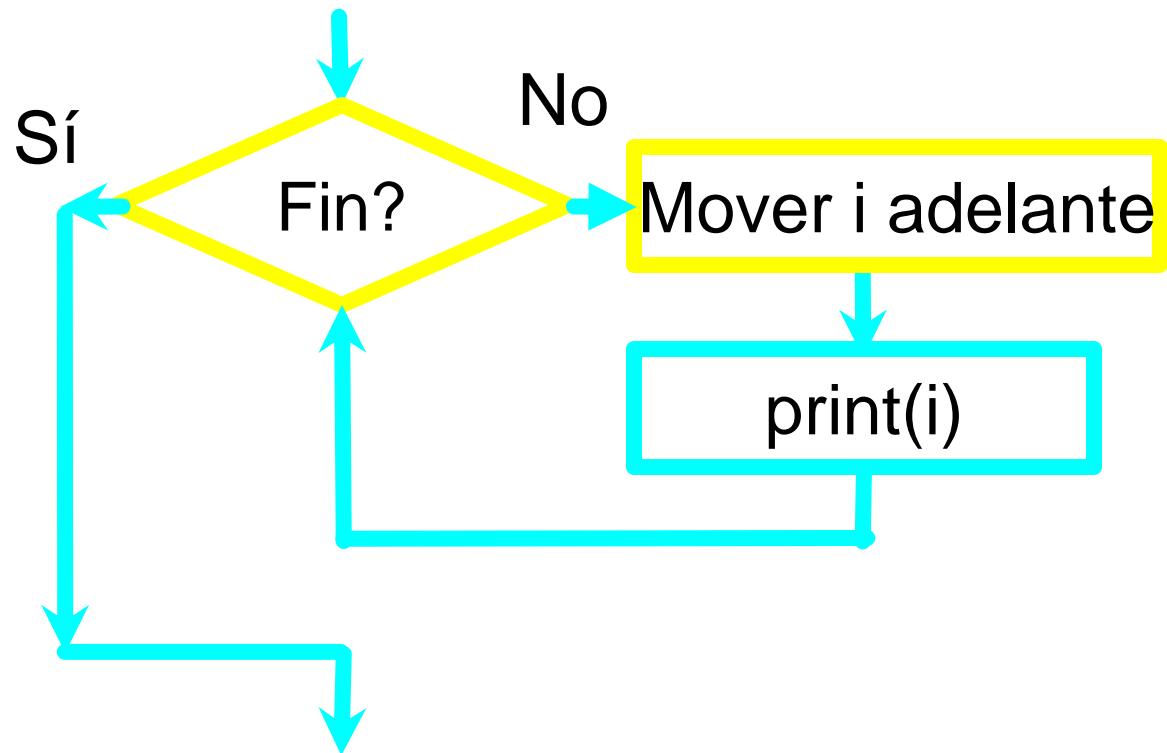
Variable de iteración



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

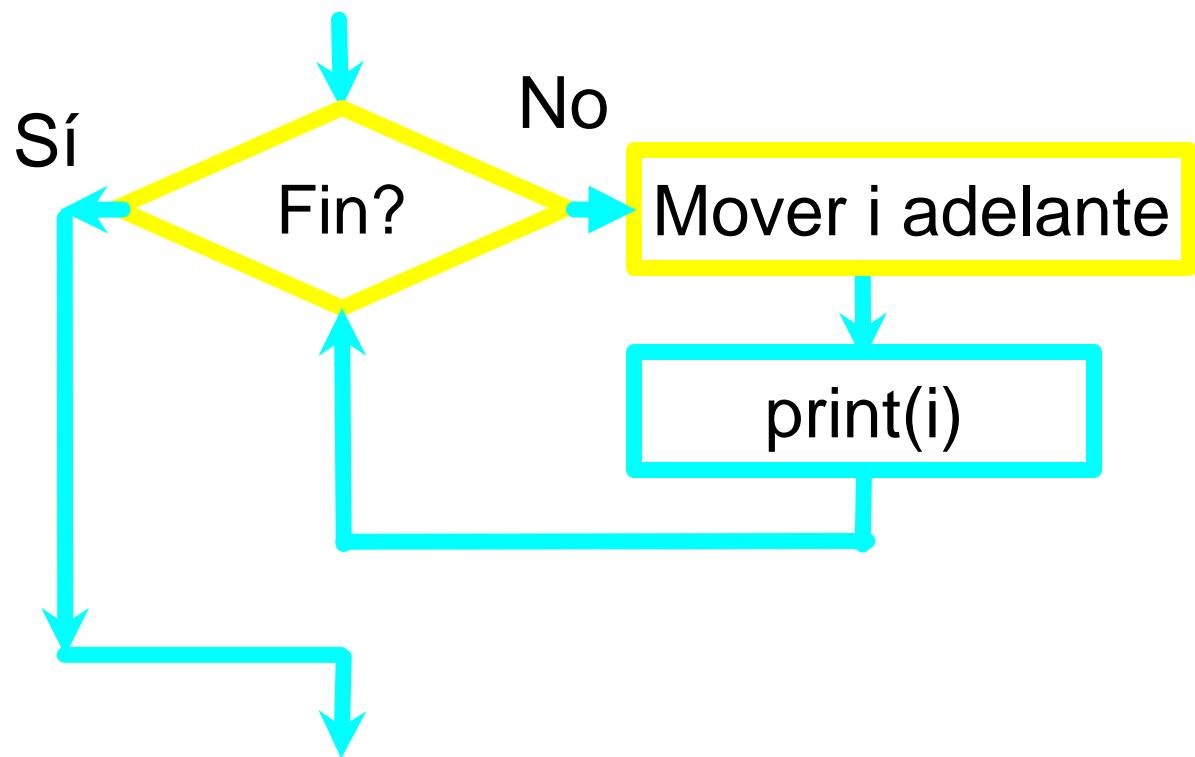
Secuencia de cinco elementos



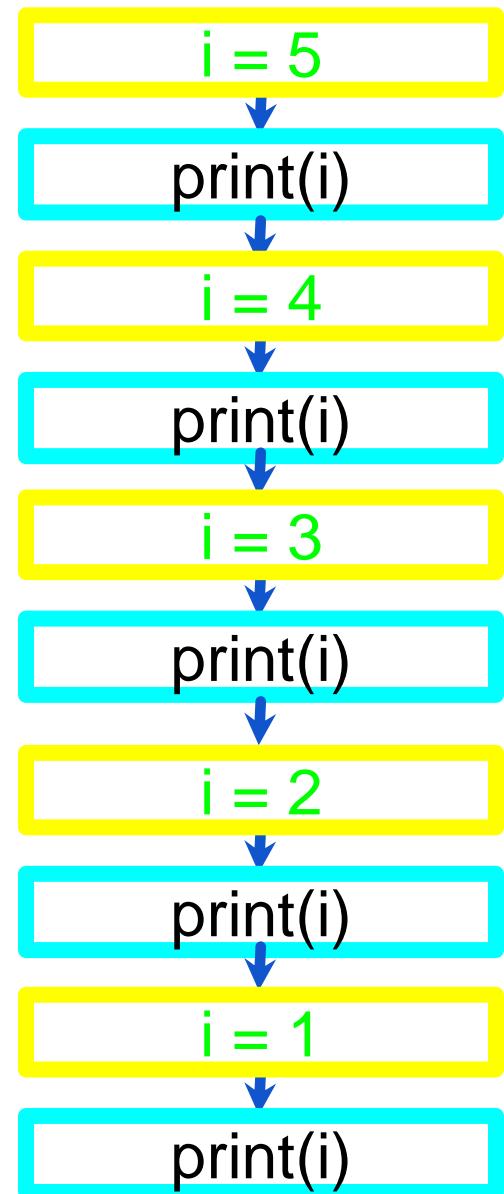


```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

La variable de iteración “**itera**” a través de la secuencia (conjunto ordenado)  
 El bloque (cuerpo) de código es **ejecutado** una vez **para cada valor** en (in) la secuencia  
 La variable de iteración se mueve a través de todos los valores en (in) la secuencia



```
for i in [5, 4, 3, 2, 1] :
    print(i)
```



# El idioma de los bucles: qué hacemos en los bucles

Nota: Incluso aunque estos ejemplos son simples, los patrones aplican a toda clase de bucles

# Haciendo bucles “inteligentes”

El truco es **“saber algo”** en cada momento sobre el “proceso” completo

Cada iteración solo ve una entrada cada vez

Establecer algunas variables a **valores iniciales**

for thing in data:

Buscar algo o hacer algo **para cada entrada** de manera separada, actualizando una variable

Mirar las variables

# Iterando a través de un conjunto

```
print('Antes')
for thing in [9, 41, 12, 3, 74, 15] :
    print(thing)
print('Después')
```

```
$ python basicloop.py
Antes
9
41
12
3
74
15
Después
```

¿Cuál es el número más grande?

¿Cuál es el número más grande?

3

¿Cuál es el número más grande?

41

¿Cuál es el número más grande?

12

¿Cuál es el número más grande?

9

¿Cuál es el número más grande?

74

¿Cuál es el número más grande?

15

¿Cuál es el número más grande?

¿Cuál es el número más grande?

3

41

12

9

74

15

# ¿Cuál es el número más grande?

El más  
grande por  
ahora

largest\_so\_far

-1

# ¿Cuál es el número más grande?

3

largest\_so\_far

3

# ¿Cuál es el número más grande?

41

largest\_so\_far

41

# ¿Cuál es el número más grande?

12

largest\_so\_far

41

# ¿Cuál es el número más grande?

9

largest\_so\_far

41

¿Cuál es el número más grande?

74

largest\_so\_far

74

¿Cuál es el número más grande?

15

74

# ¿Cuál es el número más grande?

3

41

12

9

74

15

74

# Encontrando el valor mayor

```
largest_so_far = -1
print('Antes', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('Después', largest_so_far)
```

```
$ python largest.py
Antes -1
9 9
41 41
41 12
41 3
74 74
74 15
Después 74
```

Creamos una variable que contiene el valor más grande que hemos **visto por el momento**. Si el número actual que estamos mirando es más grande, entonces ese será el valor más grande que hemos visto por el momento.

# Más patrones de bucles...

# Contando en un bucle

```
count = 0
print('Antes', count)
for thing in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    print(count, thing)
print('Después', count)
```

```
$ python countloop.py
Antes 0
1 9
2 41
3 12
4 3
5 74
6 15
Después 6
```

Para **contar** cuántas veces ejecutamos un bucle, introducimos una **variable contador** que comienza en 0 y vamos sumando 1 en cada iteración del bucle.

# Sumando en un bucle

```
sum = 0
print('Antes', sum)
for thing in [9, 41, 12, 3, 74, 15] :
    sum = sum + thing
    print(sum, thing)
print('Después', sum)
```

```
$ python sumloop.py
Antes 0
9 9
50 41
62 12
65 3
139 74
154 15
Después 154
```

Para **totalizar** un valor en un bucle, introducimos una **variable de suma** que comienza en 0 y sumamos el valor actual a la suma en cada iteración del bucle.

# Calculando la media en un bucle

```
count = 0
sum = 0
print('Antes', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('Después', count, sum, sum / count)
```

```
$ python averageloop.py
Antes 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
Después 6 154 25.666
```

Para la media combinamos los patrones de cuenta y suma y dividimos cuando el bucle acaba.

# Filtrando en un bucle

```
print('Antes')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Número grande', value)
print('Después')
```

```
$ python search1.py
Antes
Número grande 41
Número grande 74
Después
```

Utilizamos una sentencia if en el bucle para capturar/filtrar los valores que estamos buscando.

# Búsqueda utilizando una variable booleana

```
found = False
print('Antes', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('Después', found)
```

```
$ python search2.py
Antes False
False 9
False 41
False 12
True 3
True 74
True 15
Después True
```

Si solo queremos saber si un valor fue encontrado, utilizamos una variable que inicializamos a False y ponemos a True en cuanto encontramos lo que estamos buscando.

# Cómo encontrar el valor más pequeño

```
largest_so_far = -1
print('Antes', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('Después', largest_so_far)
```

```
$ python largest.py
Antes -1
9 9
41 41
41 12
41 3
74 74
74 15
Después 74
```

¿Cómo podemos cambiarlo para encontrar el valor más pequeño en la lista?

# Encontrando el valor más pequeño

```
smallest_so_far = -1
print('Antes', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('Después', smallest_so_far)
```

Cambiamos el nombre de la variable a `smallest_so_far` e intercambiamos `>` por `<`

# Encontrando el valor más pequeño

```
smallest_so_far = -1
print('Antes', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('Después', smallest_so_far)
```

```
$ python smallbad.py
Antes -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
Después -1
```

Cambiamos el nombre de la variable a `smallest_so_far` e intercambiamos `>` por `<`

# Encontrando el valor más pequeño

```
smallest = None
print('Antes')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('Después', smallest)
```

```
$ python smallest.py
Antes
9 9
9 41
9 12
3 3
3 74
3 15
Después 3
```

Todavía tenemos una variable que es la más pequeña por el momento (`smallest`). En la primera iteración, `smallest` vale **None**, así que podemos hacer que `value` sea el más pequeño por el momento (`smallest`).

# Los operadores `is` , `is not`

```
smallest = None
print('Antes')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('Después', smallest)
```

- Python tiene el **operador `is`**, el cual puede ser usado en expresiones lógicas
- Significa “es lo mismo que”
- Similar a, pero más fuerte que `==`
- **`is not`** es también un operador lógico

# Resumen

- Bucles while (indefinidos)
- Bucles infinitos
- Usando break
- Usando continue
- Constantes None y variables
- Bucles for (definidos)
- Variables de iteración
- El idioma de los bucles
- Más grande o más pequeño

## Ejercicio

Escribe un programa que pida números al usuario hasta que escriba 'fin'. Una vez escrito 'fin', el programa debe mostrar el número más grande y el más pequeño. Si el usuario escribe algo diferente a un número, se debe capturar con un try/except e ignorar el valor introducido.

Probar con 7, 2, bob, 10, y 4

Nota: el ejercicio debe resolverse con los visto hasta el momento. No está permitido el uso de otros elementos como listas (que se verán posteriormente)



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)



# Strings

## Unidad 6

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# El tipo de datos string

- Un string es una **secuencia** de caracteres (**cadena de caracteres**)
- Los literales string utilizan **comillas** 'Hello' o "Hello"
- Para strings, **+** significa “**concatenar**”
- Cuando un string contiene números, es todavía un string
- Podemos convertir los números de un string en números utilizando **int()**

```
>>> str1 = "Hello"  
>>> str2 = 'there'  
>>> bob = str1 + str2  
>>> print(bob)  
Hellothere  
>>> str3 = '123'  
>>> str3 = str3 + 1  
Traceback (most recent call  
last): File "<stdin>", line 1,  
in <module>  
TypeError: cannot concatenate  
'str' and 'int' objects  
>>> x = int(str3) + 1  
>>> print(x)  
124  
>>>
```

# Leyendo y Convirtiendo

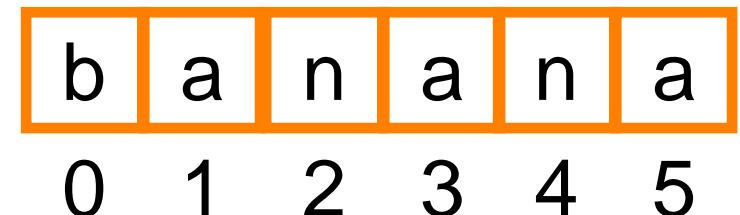
- Es mejor leer los datos usando strings y después procesarlos y **convertirlos** al tipo que necesitemos
- Esto nos da más control sobre situaciones de error y/o **entradas del usuario incorrectas**
- Los números de entrada deben convertirse desde strings

```
>>> name = input('Escribe: ')
Escribe:Chuck
>>> print(name)
Chuck
>>> apple = input('Escribe: ')
Escribe:100
>>> x = apple - 10
Traceback (most recent call
last): File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```



# Mirando dentro de los strings

- Podemos acceder a cualquier **carácter** simple de un string usando un índice especificado entre **corchetes [ ]**
- El índice debe ser un entero, y **comienzan en cero**
- El valor del índice puede ser una expresión calculada



```
>>> fruit = 'banana'  
>>> letter = fruit[1]  
>>> print(letter)  
a  
>>> x = 3  
>>> w = fruit[x - 1]  
>>> print(w)  
n
```

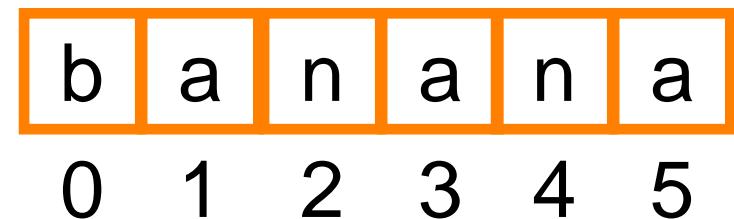
# Un carácter demasiado lejano

- Obtendremos un error python si intentamos acceder a un índice **más allá del final** del string
- Ten cuidado cuando construyas los índices y slices (ver más adelante)

```
>>> zot = 'abc'  
>>> print(zot[5])  
Traceback (most recent call  
last): File "<stdin>", line  
1, in <module>  
IndexError: string index out  
of range  
>>>
```

# Los strings tienen longitud

La función interna **len** proporciona la **longitud** de un string

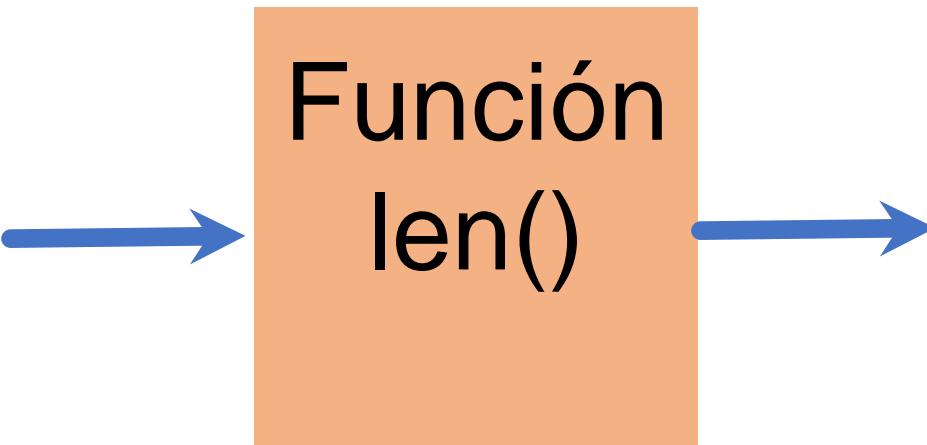


```
>>> fruit = 'banana'  
>>> print(len(fruit))  
6
```

# La función len

```
>>> fruit = 'banana'  
>>> x = len(fruit)  
>>> print(x)  
6
```

'banana'  
(un string)

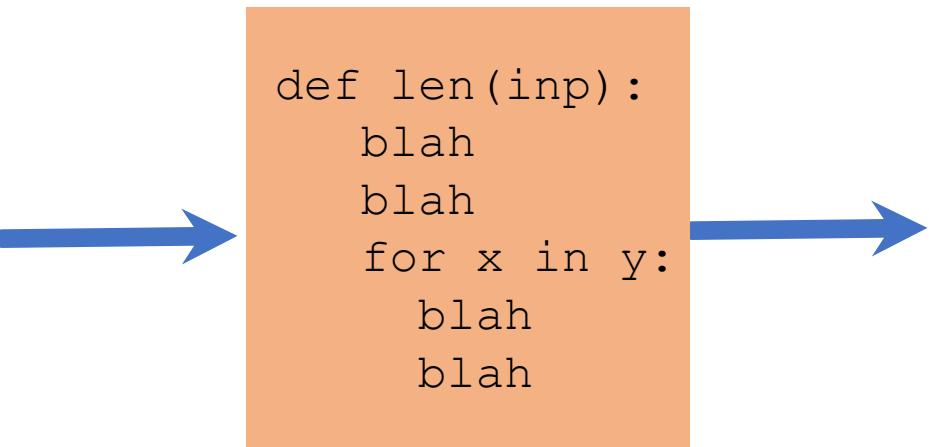


Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.

# La función len

```
>>> fruit = 'banana'  
>>> x = len(fruit)  
>>> print(x)  
6
```

'banana'  
(un string)



Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.

# Iterando sobre strings

- Usando un **bucle while**, una **variable de iteración**, y la función **len**, podemos mirar cada una de las letras de un string individualmente

```
fruit = 'banana'          0 b
index = 0                  1 a
while index < len(fruit): 2 n
    letter = fruit[index] 3 a
    print(index, letter)   4 n
    index = index + 1      5 a
```

# Iterando sobre strings

- Pero es mucho más elegante usar un **bucle for**
- La variable de iteración va tomando como valor **cada una de las letras** del string

```
fruit = 'banana'  
for letter in fruit:  
    print(letter)
```

b  
a  
n  
a  
n  
a

# Iterando sobre strings

- Pero es mucho más elegante usar un **bucle for**
- La variable de iteración va tomando como valor **cada una de las letras** del string

```
fruit = 'banana'  
for letter in fruit :  
    print(letter)
```

```
index = 0  
while index < len(fruit) :  
    letter = fruit[index]  
    print(letter)  
    index = index + 1
```

b  
a  
n  
a  
n  
a

# Iterando y contando

- Ejemplo de bucle simple que **itera** a través de las letras de un string y cuenta el número de **apariciones de la letra 'a'**

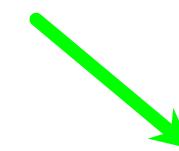
```
word = 'banana'  
count = 0  
for letter in word :  
    if letter == 'a' :  
        count = count + 1  
print(count)
```

# Mirando en detalle in

- La variable de iteración “**itera**” a través de la secuencia (conjunto ordenado)
- El bloque (cuerpo) de código es ejecutado una vez **para cada valor** en (in) la secuencia
- La variable de iteración **se mueve** a través de todos los valores en (in) la secuencia

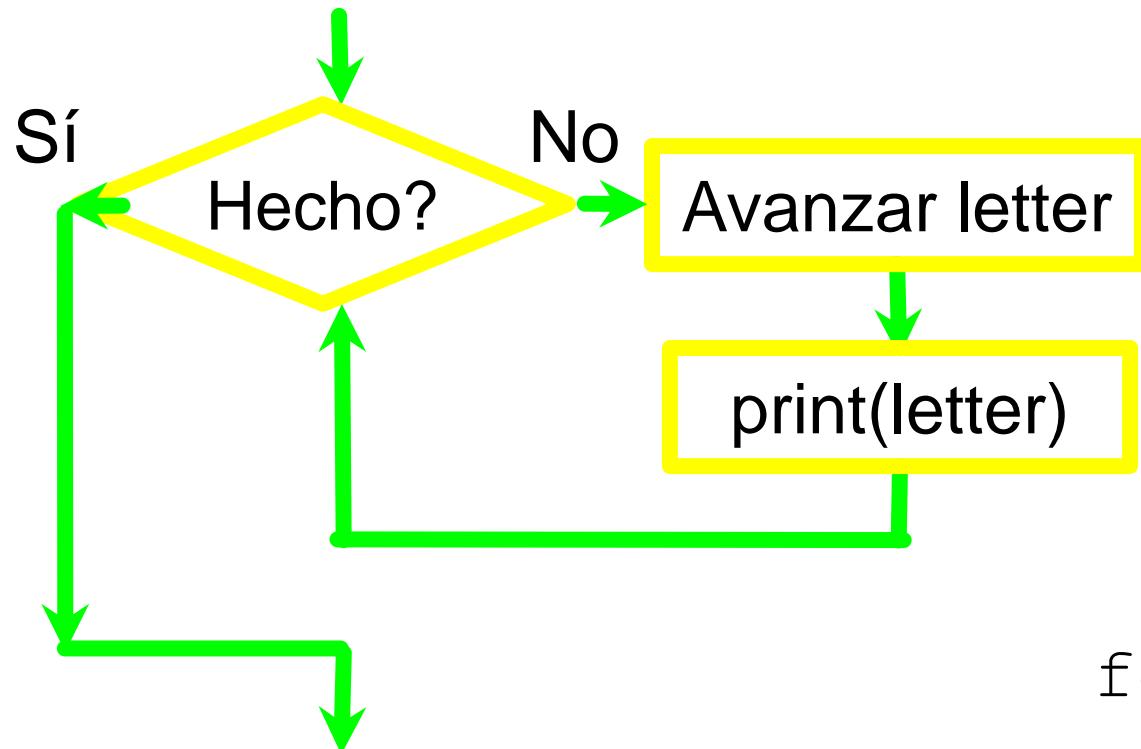
Variable de iteración

```
for letter in 'banana' :  
    print(letter)
```



String de 6 letras





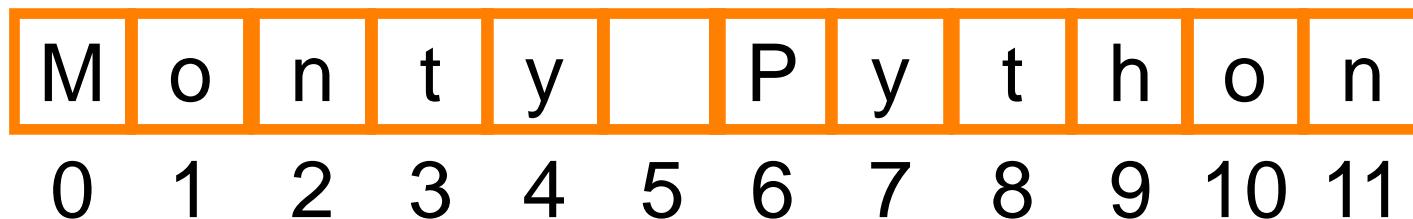
```
for letter in 'banana' :
    print(letter)
```

La variable de iteración “**itera**” a través del string y el bloque (cuerpo) de código es ejecutado una vez **para cada valor** en la secuencia

# Más operaciones de strings

# Cortando Strings (slices)

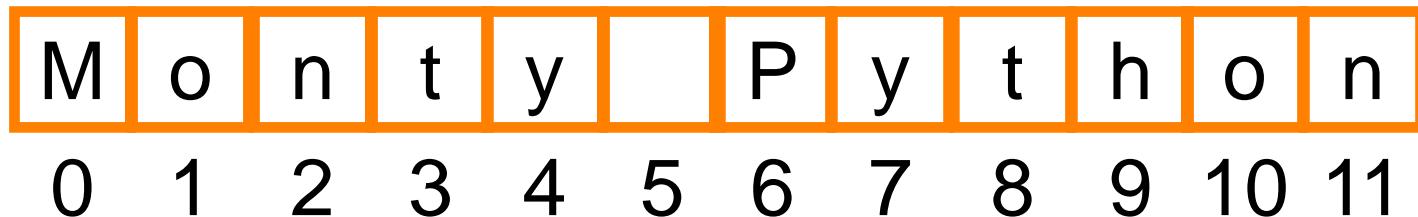
- Podemos mirar cualquier sección continua (rodaja) de un string utilizando el **operador dos puntos** (`:`)
- El primer número indica desde dónde **empezamos** a cortar
- El segundo número indica el **final** del slice (rodaja) – **pero sin incluirlo**
- Si el segundo número está más allá del final del string, se queda en el final
- Ten en cuenta que comenzamos en **0**



```
>>> s = 'Monty Python'  
>>> print(s[0:4])  
Mont  
>>> print(s[6:7])  
P  
>>> print(s[6:20])  
Python
```

# Cortando Strings (slices)

- Si no ponemos el primer o último número del slice, se asume que se está usando la primera o última letra del string respectivamente



```
>>> s = 'Monty Python'  
>>> print(s[:2])  
Mo  
>>> print(s[8:])  
thon  
>>> print(s[:])  
Monty Python
```

# Concatenación de strings

Cuando se aplica el  
**operador +** a strings,  
significa “**concatenación**”

```
>>> a = 'Hello'  
>>> b = a + 'There'  
>>> print(b)  
HelloThere  
>>> c = a + ' ' + 'There'  
>>> print(c)  
Hello There  
>>>
```

# Usando in como operador lógico

- La palabra clave `in` puede ser usada para comprobar si un **string** está "en" otro **string**
- El uso de `in` funciona como una expresión lógica que retorna **True o False** y puede ser usada en una sentencia `if`

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if 'a' in fruit :  
...     print('Encontrado!')  
  
...  
Encontrado!  
>>>
```

# Comparación de strings

```
if word == 'banana':  
    print('Todo bien, bananas.')  
  
if word < 'banana':  
    print('Tu palabra, ' + word + ', es menor que banana.')  
elif word > 'banana':  
    print('Tu palabra, ' + word + ', es mayor que banana.')  
else:  
    print('Todo bien, bananas.')
```

# La librería de strings

- Python tiene muchas funciones para el manejo de strings en la **librería de strings**
- Estas funciones son realmente internas en cada string – podemos invocarlas simplemente poniéndolas con el **operador punto (.)** a continuación del string ('string'.función())
- Estas funciones **no modifican el string**, en su lugar retornan un nuevo string resultante de aplicar la función

```
>>> greet = 'Hello Bob'  
>>> zap = greet.lower()  
>>> print(zap)  
hello bob  
>>> print(greet)  
Hello Bob  
>>> print('Hi There'.lower())  
hi there  
>>>
```

```
>>> stuff = 'Hello world'  
>>> type(stuff)  
<class 'str'>  
>>> dir(stuff)  
['capitalize', 'casefold', 'center', 'count', 'encode',  
'endswith', 'expandtabs', 'find', 'format', 'format_map',  
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',  
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',  
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',  
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',  
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',  
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',  
'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

`str.replace(old, new[, count])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.rfind(sub[, start[, end]])`

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.rindex(sub[, start[, end]])`

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.rjust(width[, fillchar])`

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.rpartition(sep)`

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

`str.rsplit(sep=None, maxsplit=-1)`

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

# Librería de strings

str.capitalize()

str.center(width[, fillchar])

str.endswith(suffix[, start[, end]] )

str.find(sub[, start[, end]] )

str.lstrip([chars])

str.replace(old, new[, count])

str.lower()

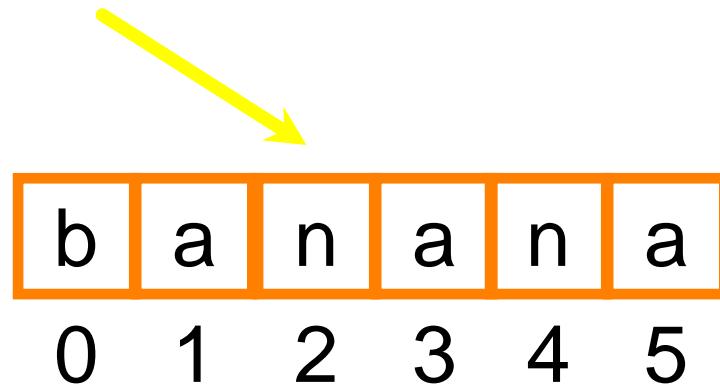
str.rstrip([chars])

str.strip([chars])

str.upper()

# Buscando en un String

- Usamos la función **find()** para buscar un **substring** dentro de otro string
- **find()** encuentra la primera aparición del substring
- Si el substring no se encuentra,, **find()** retorna -1
- Recuerda que las posiciones de los strings comienzan en cero



```
>>> fruit = 'banana'  
>>> pos = fruit.find('na')  
>>> print(pos)  
2  
>>> aa = fruit.find('z')  
>>> print(aa)  
-1
```

# Convirtiendo todo a MAYÚSCULAS

- Podemos conseguir copias de un string en minúsculas o mayúsculas
- Frecuentemente, cuando buscamos en un string con `find()`, primero debemos convertir el string a minúsculas para poder buscar independientemente de cómo haya escrito el usuario

```
>>> greet = 'Hello Bob'  
>>> nnn = greet.upper()  
>>> print(nnn)  
HELLO BOB  
>>> www = greet.lower()  
>>> print(www)  
hello bob  
>>>
```

# Buscar y Reemplazar

- La función **replace()** es como el "**buscar y reemplazar**" de un procesador de textos
- Reemplaza todas las ocurrencias del string buscado con el string de reemplazamiento y devuelve el nuevo string

```
>>> greet = 'Hello Bob'  
>>> nstr = greet.replace('Bob', 'Jane')  
>>> print(nstr)  
Hello Jane  
>>> nstr = greet.replace('o', 'X')  
>>> print(nstr)  
HellX BXb  
>>>
```

# Eliminando espacios

- Algunas veces nos interesará **quitar los espacios** al principio o al final de un string
- **lstrip()** o **rstrip()** borran los espacios en blanco a la izquierda o la derecha respectivamente
- **strip()** borra los espacios al principio y al final

```
>>> greet = 'Hello Bob '
>>> greet.lstrip()
'Hello Bob '
>>> greet.rstrip()
'Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

# Prefijos

```
>>> line = 'Please have a nice day'  
>>> line.startswith('Please')  
True  
>>> line.startswith('p')  
False
```

# Procesando (parsing) y extrayendo

21



31



From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'  
>>> atpos = data.find('@')  
>>> print(atpos)  
21  
>>> spos = data.find(' ',atpos)  
>>> print(spos)  
31  
>>> host = data[atpos+1 : spos]  
>>> print(host)  
uct.ac.za
```



# Dos clases de strings

Python 2.7.10

```
>>> x = '이광춘'  
>>> type(x)  
<type 'str'>  
>>> x = u'이광춘'  
>>> type(x)  
<type 'unicode'>  
>>>
```

Python 3.5.1

```
>>> x = '이광춘'  
>>> type(x)  
<class 'str'>  
>>> x = u'이광춘'  
>>> type(x)  
<class 'str'>  
>>>
```

En Python 3, todos los strings son Unicode

# Resumen

- El tipo String
- Leer/Convertir
- Indexando strings []
- Cortando strings [2:4]
- Iterando strings con for y while
- Concatenando strings con +
- Operaciones de strings
- La librería de strings
- Comparaciones de strings
- Buscando en strings
- Reemplazando texto
- Eliminando espacios

## Ejercicio 1

Utilizar `find()` y los slices (`:`) para extraer el número en un string como el que aparece a continuación. Convertir el número a flotante y mostrarlo.

¡Ojo! No asumir que el número es 0.8475, es solo un ejemplo. No sabemos qué número es, tenemos que extraerlo. Sí podemos suponer que el resto de la línea es igual. Esto es, comienza por "X-DSPAM-Confidence..."

"X-DSPAM-Confidence: 0.8475"

## Ejercicio 2

Pedir un nombre al Usuario de la forma:

NOMBRE APELLIDO1 APELLIDO2

Por ejemplo: José García Pérez

Dar la Vuelta al nombre y presentarlo en el formato:

APELLIDO1 APELLIDO2, NOMBRE

Por ejemplo: García Pérez, José

## Ejercicio 3

Pedir una contraseña al Usuario repetidas veces hasta que la contraseña sea considerada fuerte. Para ello, debe tener, como mínimo 8 caracteres y contener, al menos, tanto una mayúscula como un dígito

estaEslaCon8traseña sería válida

abcdefghijklmnpq no sería válida porque no cumple las condiciones

La solución debe incluir una función que reciba un string y retorne si es una contraseña fuerte o no

Pista: usar las funciones isdigit, isupper



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Strings (contenido adicional)

## Unidad 6

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Contenidos

- F-strings
- Triples comillas

# F-Strings en Python

- Los f-strings ofrecen una forma concisa y eficiente para trabajar con los strings
- Se añade un prefijo “f” o “F” a los strings
- Y se pueden añadir expresiones entre {} que son evaluadas en tiempo de ejecución

```
name = "Jane"  
age = 25  
  
print(f"Hello, {name}! You're {age} years old."  
  
Salida: Hello, Jane! You're 25 years old.
```

# Expresiones con f-strings

- Además de variables, se pueden incluir expresiones en los f-strings
- Las expresiones pueden incluir funciones y llamadas a métodos

```
print(f"2 * 21")  
Salida: 42
```

```
name = "Jane"  
age = 25
```

```
print(f"Hello, {name.upper()}! You're {age} years old.")
```

Salida: Hello, JANE! You're 25 years old.

```
print(f"[2**n for n in range(3, 9)]")  
Salida: [8, 16, 32, 64, 128, 256]
```

# Triples comillas

- Podemos definir strings usando triples comillas “”” para diferentes usos
  - Definir strings en múltiples líneas
  - Documentar nuestro código
- Para usar múltiples líneas:

```
texto = """Este es un string  
que ocupa varias líneas."""  
print(texto)
```

Salida:

Este es un string  
que ocupa varias líneas.

# Triples comillas

- Para documentar el código, se colocan justo debajo de la declaración de elementos como funciones o clases y no tienen efecto en la ejecución
- Luego podemos usar la herramienta **pydoc** para generar la documentación
  - En texto plano o HTML

```
python -m pydoc -w prueba
```

[index](#)  
**prueba** [d:\daniel\varios\prueba.py](#)

## Functions

### **suma(a, b)**

Esta función devuelve la suma de dos números.

```
def suma(a, b):
```

"""Esta función devuelve la suma de dos números."""

```
return a + b
```

Si la función suma está en el archivo prueba.py:

```
python -m pydoc prueba
```

Help on module prueba:

NAME

prueba

FUNCTIONS

suma(a, b)

Esta función devuelve la suma de dos números.

FILE

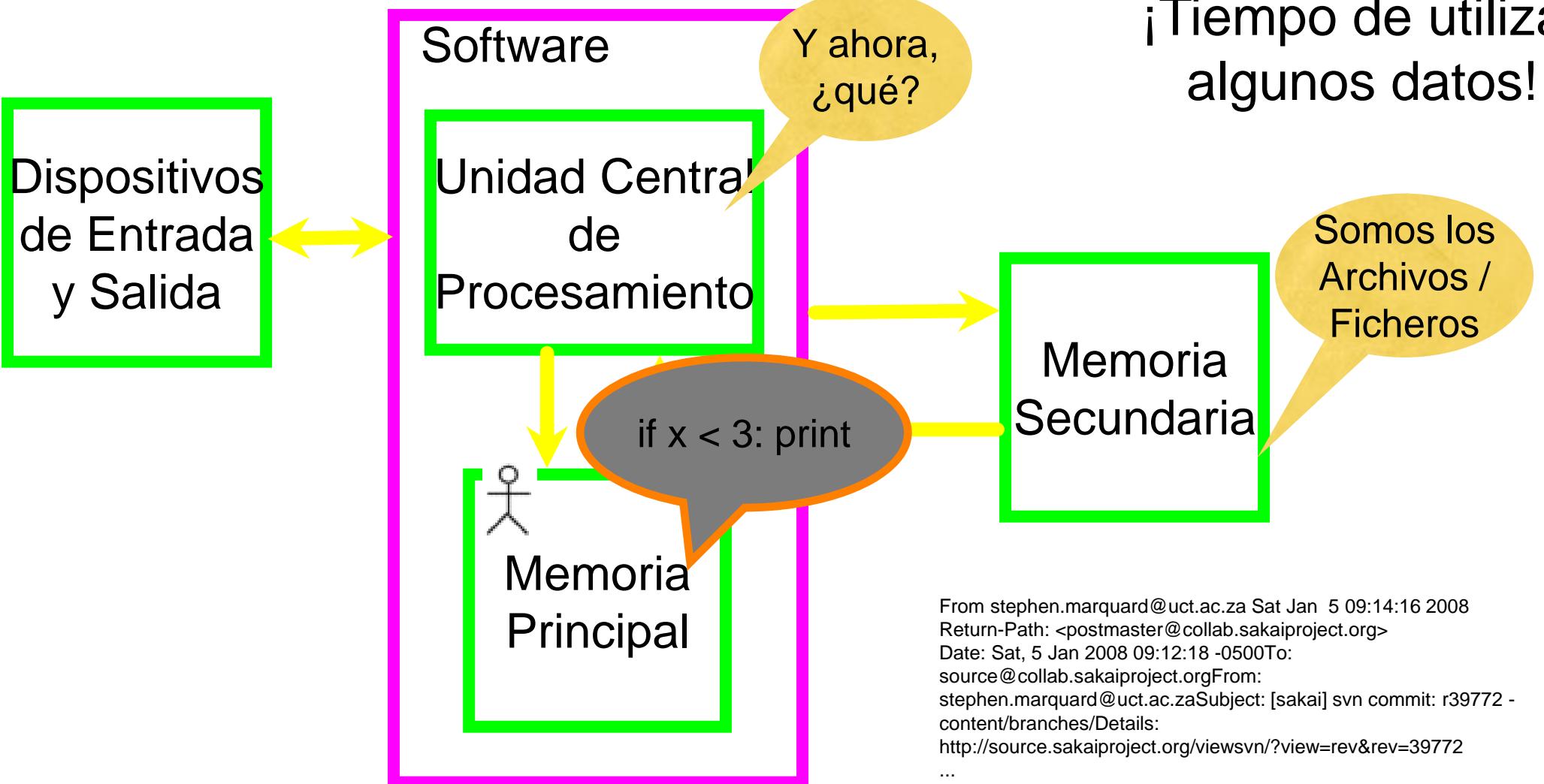
d:\daniel\varios\prueba.py

# Archivos

## Unidad 7

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)





From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008  
Return-Path: <postmaster@collab.sakaiproject.org>  
Date: Sat, 5 Jan 2008 09:12:18 -0500To:  
source@collab.sakaiproject.orgFrom:  
stephen.marquard@uct.ac.zaSubject: [sakai] svn commit: r39772 -  
content/branches/Details:  
<http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772>  
...

# Procesamiento de Archivos

Un archivo de texto puede ser visto como una **secuencia de líneas**

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Return-Path: <postmaster@collab.sakaiproject.org>

Date: Sat, 5 Jan 2008 09:12:18 -0500

To: source@collab.sakaiproject.org

From: stephen.marquard@uct.ac.za

Subject: [sakai] svn commit: r39772 - content/branches/

Details: <http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772>

Archivo mbox-short.txt en campus virtual

# Abriendo un Archivo

- Antes de leer los contenidos de un archivo, debemos decirle a Python **con qué archivo** vamos a trabajar y **qué vamos a hacer** con él
- Esto se hace con la función **open()**
- **open()** retorna un “**manejador de archivo**” – una variable que permite hacer operaciones sobre el archivo
- Similar a “Archivo -> Abrir” en un procesador de textos

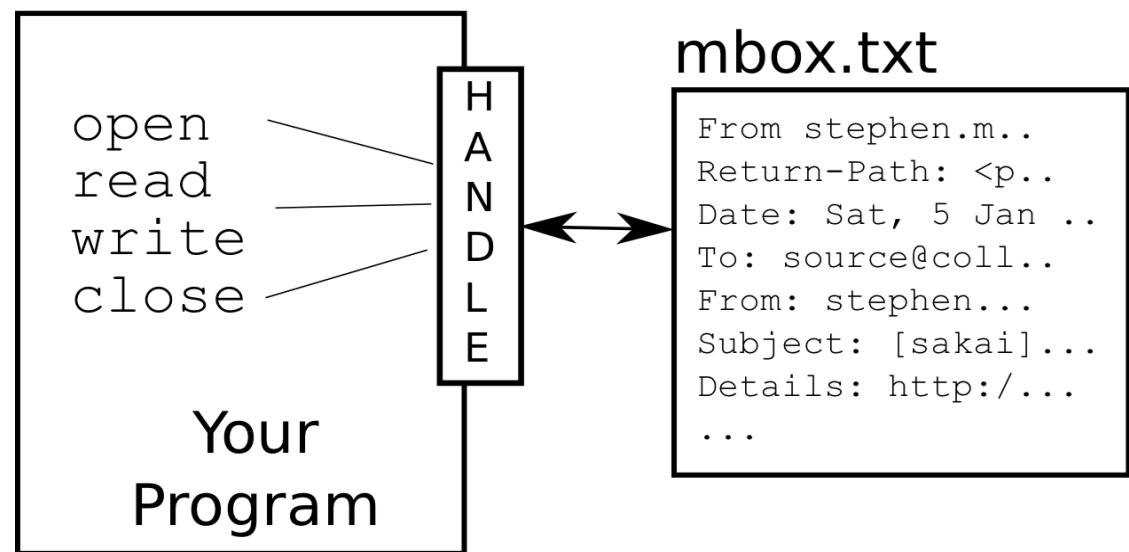
# Usando open()

```
fhand = open('mbox.txt', 'r')
```

- **handle = open(filename, mode)**
- Retorna un manejador para manipular el archivo
- filename es un string e indica el nombre del archivo
- mode es opcional y debería ser 'r' si vamos a leer del archivo y 'w' si vamos a escribir en él

# ¿Qué es un manejador?

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='cp1252'>
```



# Cuando no se encuentra un archivo

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundException: [Errno 2] No such file or
directory: 'stuff.txt'
```

# El carácter de nueva línea

- Se utiliza un carácter especial denominado de **nueva línea** (newline) para indicar cuándo acaba una línea y comienza otra
- Se representa en strings con `\n`
- Ojo! Sigue siendo un carácter – no dos

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

# Procesamiento de archivos

Un archivo de texto puede ser visto como una **secuencia de líneas**

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

# Procesamiento de archivos

Un archivo de texto tiene \n (nueva línea) **al final de cada línea**

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\n
To: source@collab.sakaiproject.org\n
From: stephen.marquard@uct.ac.za\n
Subject: [sakai] svn commit: r39772 - content/branches/\n
\n
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

# Leyendo archivos en Python

# El manejador de archivo como una secuencia

- Un manejador de archivo abierto para lectura puede ser tratado como una **secuencia de strings** donde cada línea en el archivo es un string en la secuencia
- Podemos usar un **bucle for** para iterar a través de la secuencia
- Recuerda - una secuencia es un conjunto ordenado (obtendremos el contenido del archivo en el mismo orden en el que está almacenado)

```
xfile = open('mbox.txt')  
for cheese in xfile:  
    print(cheese)
```

Una vez leído el archivo, tenemos que cerrarlo (`xfile.close()`) y volver a abrirlo si queremos volver a procesarlo. Esto es, no podemos "retroceder" en la lectura.

# Contando líneas en un archivo

- Abre un archivo
- Usa un bucle for para leer cada línea
- Cuenta las líneas e imprime su número

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print('Número de líneas:', count)
```

```
$ python open.py
Número de líneas: 132045
```

# Leyendo el archivo \*completo\*

- Podemos leer el archivo completo (\n incluidos) en un string simple

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

# Buscando en un archivo

- Podemos usar sentencias if en el bucle for para imprimir solamente las líneas que cumplan **cierto criterio**

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:'):
        print(line)
```

# OOPS!

¿Qué hacen estas líneas  
en blanco aquí?

From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

...

# OOPS!

¿Qué hacen estas líneas  
en blanco aquí)

- Cada línea de archivo tiene un carácter de nueva línea al final
- La sentencia print añade otro newline al final de cada línea

```
From: stephen.marquard@uct.ac.za\n\nFrom: louis@media.berkeley.edu\n\nFrom: zqian@umich.edu\n\nFrom: rjlowe@iupui.edu\n\n...
```

# Buscando en un archivo (arreglado)

- Podemos eliminar los espacios en blanco al final de un string usando la función **rstrip()**
- El newline es considerado un “espacio en blanco” y es eliminado

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)
```

From: stephen.marquard@uct.ac.za  
From: louis@media.berkeley.edu  
From: zqian@umich.edu  
From: rjlowe@iupui.edu  
....

# Saltando líneas con continue

- Si una línea no nos interesa, podemos saltarla con la sentencia continue

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:'):
        continue
    print(line)
```

# Usando `in` para seleccionar líneas

Podemos utilizar `in` en una línea como nuestro **criterio de selección**

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print(line)
```



```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

```
fname = input('Nombre de archivo: ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('Hubo', count, 'líneas con subject en', fname)
```

## Preguntar por el nombre de un archivo

Nombre de archivo: mbox.txt  
Hubo 1797 líneas con subject en mbox.txt

Nombre de archivo: mbox-short.txt  
Hubo 27 líneas con subject en mbox-short.txt

# Nombres de archivo incorrectos

```
fname = input('Nombre de archivo: ')
try:
    fhand = open(fname)
except:
    print('El archivo no pudo abrirse:', fname)
    quit()

count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('Hubo', count, 'líneas con subject en', fname)
```

Nombre de archivo: mbox.txt

Hubo 1797 líneas con subject en mbox.txt

Nombre de archivo: na na boo boo

El archivo no pudo abrirse: na na boo boo

# Escribiendo en archivos

- Para escribir en un archivo, tenemos que abrirlo con el modo "w" al hacer open
- ¡¡Cuidado, el contenido previo **se borrará!!!**
- También Podemos usar el modo "a" si lo que queremos es **añadir** datos al final del archivo
- El método **write** del manejador del archivo, permite escribir datos en él – retorna el número de caracteres escritos
- Al final, tenemos que cerrar el archivo con **close**, para asegurarnos de que los contenidos quedan escritos

```
fout = open('output.txt', 'w')  
  
>>> line1 = "Esto va a escribirse,\n"  
>>> fout.write(line1)  
22  
  
fout.close()
```

```
fout = open('output.txt', 'a')  
  
>>> line1 = "Y esto se añade.\n"  
>>> fout.write(line1)  
17  
  
fout.close()
```

# Resumen

- Almacenamiento secundario
- Abriendo un archivo –
  - manejador de archivo
- Estructura de archivo –
  - el carácter newline
- Leyendo un archivo línea a línea con un bucle for
- Buscando líneas
- Leyendo nombres de archivos
- Tratando con nombres equivocados
- Escribiendo en archivos

**Ejercicio** Escribir un programa que pregunte por un nombre de archivo, lo abra, y lo lea mostrando las líneas de la forma:  
X-DSPAM-Confidence: 0.8475

Contar las líneas y extraer los valores flotantes de cada una de ellas, calculando el promedio y mostrándolo al final.

Se pueden descargar datos de muestra desde el archivo mbox-short.txt en Campus Virtual. El valor promedio resultante debería ser:  
0.7507185185185187



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Listas en Python

## Unidad 8

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Programación

- **Algoritmos**
  - Un conjunto de reglas o pasos para resolver un problema
- +- **Estructura de Datos**
  - Una forma particular de organizar datos en un ordenador

<https://es.wikipedia.org/wiki/Algoritmo>

[https://es.wikipedia.org/wiki/Estructura\\_de\\_datos](https://es.wikipedia.org/wiki/Estructura_de_datos)

# ¿Qué no es una "Colección?

La mayor parte de nuestras variables tienen un único valor en ellas – cuando ponemos un nuevo valor en la variable, el antiguo valor es sobreescrito

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

# Una lista es una clase de colección



- Una colección nos permite poner **muchos valores** en una sola “variable”
- Usar colecciones es interesante, porque podemos tener muchos valores en un solo "paquete".

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]
```

```
carryon = [ 'socks', 'shirt', 'perfume' ]
```

# Constantes de Listas

- Las constantes de Listas se indican **entre corchetes [ ]** y los elementos de la lista se separan por comas
- Un elemento de una lista puede ser cualquier objeto Python – incluso otra lista
- Una lista puede estar vacía

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

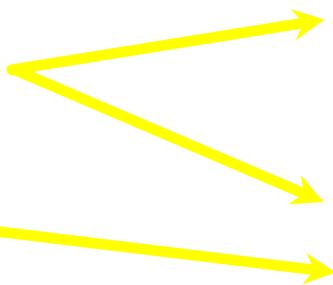
# ¡Ya usábamos listas!

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Despegue! ')
```

5  
4  
3  
2  
1  
Despegue!

# Listas y Bucles for – Los mejores amigos

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Feliz Año Nuevo:', friend)
print('Hecho!')
```



Feliz Año Nuevo: Joseph  
Feliz Año Nuevo : Glenn  
Feliz Año Nuevo : Sally  
Hecho!



# Mirando dentro de las listas

Como en los strings, podemos acceder a cualquier elemento de una lista usando su **índice** expresado con corchetes



```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]  
>>> print(friends[1])  
Glenn  
>>>
```

# Las listas son mutables

- Los strings son “inmutables” – no podemos cambiar los contenidos de un string – debemos crear un nuevo string para hacer cualquier cambio
- Las listas son “mutables” – **podemos cambiar un elemento de una lista usando su índice**

```
>>> fruit = 'Banana'  
>>> fruit[0] = 'b'  
Traceback  
TypeError: 'str' object does not  
support item assignment  
>>> x = fruit.lower()  
>>> print(x)  
banana  
>>> lotto = [2, 14, 26, 41, 63]  
>>> print(lotto)  
[2, 14, 26, 41, 63]  
>>> lotto[2] = 28  
>>> print(lotto)  
[2, 14, 28, 41, 63]
```

# ¿Cuántos elementos tiene una lista?

- La función **len()** toma una lista como parámetro, y retorna el **número de elementos** en la lista
- En realidad, len() nos dice el número de elementos de cualquier conjunto o secuencia (como los strings...)

```
>>> greet = 'Hello Bob'  
>>> print(len(greet))  
9  
  
>>> x = [ 1, 2, 'joe', 99]  
>>> print(len(x))  
4  
>>>
```

# Usando la función range

- La función **range** retorna una **lista de números** que varían desde 0 a uno menos de lo indicado en **range**
- Usando **list** y **range** podemos construir una lista de números
- Podemos construir un bucle usando **for** y **range**

```
>>> print(list(range(4)))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(list(range(len(friends))))
[0, 1, 2]
>>>
```

# Una historia de 2 bucles...

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
    print('Feliz Año Nuevo:', friend)

for i in range(len(friends)) :
    friend = friends[i]
    print('Feliz Año Nuevo:', friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(list(range(len(friends))))
[0, 1, 2]
>>>
```

**Feliz Año Nuevo: Joseph**  
**Feliz Año Nuevo: Glenn**  
**Feliz Año Nuevo: Sally**

# Concatenando Listas con +

Podemos crear una  
nueva lista uniendo dos  
existentes

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

# Las listas pueden ser troceadas usando :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Recuerda: Como en los strings, el segundo número indica el final,  
**pero sin incluirlo**

# Métodos de Listas

```
>>> x = list()
>>> type(x)
<class 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

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

# Construyendo una Lista

- Podemos crear una lista vacía e ir sumando elementos con el método **append**
- La lista permanece en orden y los nuevos elementos son añadidos **al final de la lista**

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

# ¿Está algo en una lista?

- Python proporciona dos operadores para comprobar si un elemento está en una lista (**in** y **not in**)
- Estos operadores lógicos retornan True o False
- No modifican la lista

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

# Ordenando Listas

- Una lista puede contener muchos elementos y los mantiene en el orden en el que se han añadido hasta que hagamos algo que cambie este orden
- Una lista puede ser **ordenada (sort)** (es decir, cambiamos el orden)

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]  
>>> friends.sort()  
>>> print(friends)  
['Glenn', 'Joseph', 'Sally']  
>>> print(friends[1])  
Joseph  
>>>
```

# Borrando elementos de una lista

- Si sabemos la posición a borrar, podemos usar **pop**
- Si no necesitamos el elemento, podemos utilizar el operador **del**
- E incluso podemos indicar qué queremos borrar si no sabemos la posición usando **remove**

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print(t)
['a', 'c']
>>> print(x)
b
```

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print(t)
['a', 'c']
```

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print(t)
['a', 'c']
```

# Funciones internas y Listas

- Hay varias funciones en Python que toman listas como parámetros
- ¿Recuerdas los bucles que construimos? ¡Así es mucho más simple!

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

```
total = 0
count = 0
while True :
    inp = input('Escribe un número: ')
    if inp == 'fin' : break
    value = float(inp)
    total = total + value
    count = count + 1
```

```
average = total / count
print('Media:', average)
```

Escribe un número: 3  
Escribe un número : 9  
Escribe un número : 5  
Escribe un número : fin  
**Media: 5.66666666667**

```
numlist = list()
while True :
    inp = input('Escribe un número: ')
    if inp == 'fin' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Media:', average)
```

# Mejores amigos: Strings y Listas

```
>>> abc = 'With three words'  
>>> stuff = abc.split()  
>>> print(stuff)  
['With', 'three', 'words']  
>>> print(len(stuff))  
3  
>>> print(stuff[0])  
With
```

```
>>> print(stuff)  
['With', 'three', 'words']  
>>> for w in stuff :  
...     print(w)  
...  
With  
Three  
Words  
>>>
```

**Split** divide un string en partes y produce una **lista de strings**. Después, podemos acceder a una palabra en particular o iterar a través de todas las palabras.

```
>>> line = 'A lot of spaces'  
>>> etc = line.split()  
>>> print(etc)  
['A', 'lot', 'of', 'spaces']  
>>>  
>>> line = 'first;second;third'  
>>> thing = line.split()  
>>> print(thing)  
['first;second;third']  
>>> print(len(thing))  
1  
>>> thing = line.split(';')  
>>> print(thing)  
['first', 'second', 'third']  
>>> print(len(thing))  
3  
>>>
```

Cuando no indicas un separador, se utilizan los espacios como separador, y múltiples espacios se tratan como uno solo

Se puede indicar **qué carácter** se quiere utilizar como **separador/delimitador** para usar en **split**

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

Sat  
Fri  
Fri  
Fri  
...

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

# El patrón doble split

A veces usamos split, y después volvemos a aplicar split sobre uno de los “trozos”

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
words = line.split()
email = words[1]
print(email[1])
```

# El patrón doble split

From **stephen.marquard@uct.ac.za** Sat Jan 5 09:14:16 2008

```
words = line.split()  
email = words[1]                                stephen.marquard@uct.ac.za  
print(email[1])
```

# El patrón doble split

From **stephen.marquard@uct.ac.za** Sat Jan 5 09:14:16 2008

```
words = line.split()  
email = words[1]                      stephen.marquard@uct.ac.za  
pieces = email.split('@')                ['stephen.marquard', 'uct.ac.za']  
print(pieces[1])
```

# Resumen

- Concepto de colección
- Listas y bucles for
- Indexando y buscando
- Mutabilidad de las listas
- Eliminando elementos
- Funciones: len, min, max, sum
- Troceando listas
- Métodos de listas: append, remove
- Ordenando listas
- Dividiendo strings en palabras
- Usando split para interpretar strings

## Ejercicio 1

Abre el archivo romeo.txt y léelo línea a línea. Para cada línea, divide la línea en palabras usando el método split(). El programa debería construir una lista de palabras. Para cada palabra de cada línea, comprobar si ya estaba en la lista, y si no, añádela. Cuando el programa termine, ordenar la lista y mostrar las palabras resultantes en orden alfabético.

El archivo romeo.txt puede descargarse desde el Campus Virtual

**Ejercicio 2** Abre el archivo mbox-short.txt y léelo línea a línea. Cuando encuentres una línea que comience con 'From ' como la siguiente:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16  
2008

trocea la línea usando split() e imprime la segunda palabra de la línea (es decir, la dirección de la persona que envió el mensaje). Al final, indica cuántas personas hubo (no es necesario tener en cuenta repeticiones).

Nota: asegúrate de no incluir las líneas que comiencen con 'From:'.

El archivo mbox-short.txt puede descargarse desde el Campus Virtual



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Diccionarios en Python

## Unidad 9

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# ¿Qué es una colección?



- Usar colecciones es interesante, porque podemos tener muchos valores en un solo "paquete".
- Hacemos esto teniendo más de un lugar “en” la variable
- Python proporciona formas de acceder a los diferentes lugares en la variable

# ¿Qué no es una "Colección?

La mayor parte de nuestras variables tienen un único valor en ellas – cuando ponemos un nuevo valor en la variable, el antiguo valor es sobreescrito

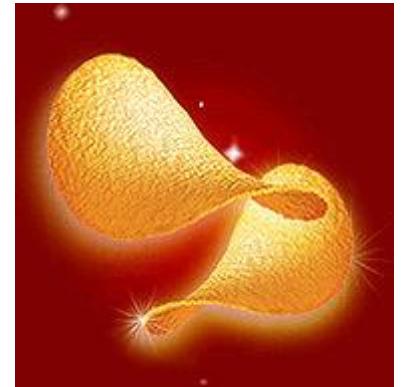
```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```



# Una historia de dos colecciones...

- **Lista**

- Una colección lineal de valores que permanecen en orden



- **Diccionario**

- Una “bolsa” de valores, cada uno con su propia etiqueta



# Diccionarios



calculadora

pañuelos

perfume

dinero

caramelo

[http://en.wikipedia.org/wiki/Associative\\_array](http://en.wikipedia.org/wiki/Associative_array)

# Diccionarios

- Los diccionarios son las colecciones de datos **más poderosas** de Python
- Los diccionarios nos permiten hacer **operaciones rápidas** similares a las de bases de datos en Python
- Los diccionarios tienen nombres diferentes en otros lenguajes
  - Arrays asociativos - Perl / PHP
  - Properties o Map o HashMap - Java
  - Propiedad Bag - C# / .NET



# Diccionarios

- Las listas permiten acceder a sus elementos basándose en la posición en la lista
- Los **diccionarios** son como bolsas – no hay orden
- Así que accedemos a lo que ponemos en el diccionario a través de una “etiqueta de búsqueda”

```
>>> bolso = dict()
>>> bolso['dinero'] = 12
>>> bolso['caramelos'] = 3
>>> bolso['pañuelos'] = 75
>>> print(bolso)
{'dinero': 12, 'pañuelos': 75, 'caramelos': 3}
>>> print(bolso['caramelos'])
3
>>> bolso['caramelos'] = bolso['caramelos'] + 2
>>> print(bolso)
{'dinero': 12, 'pañuelos': 75, 'caramelos': 5}
```

# Comparando listas y diccionarios

Los diccionarios son como listas excepto que usan **claves** (keys) en lugar de números para acceder a valores

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(183)  
>>> print(lst)  
[21, 183]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 183]
```

```
>>> ddd = dict()  
>>> ddd['age'] = 21  
>>> ddd['course'] = 182  
>>> print(ddd)  
{'course': 182, 'age': 21}  
>>> ddd['age'] = 23  
>>> print(ddd)  
{'course': 182, 'age': 23}
```

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(183)  
>>> print(lst)  
[21, 183]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 183]
```

Lista		
Clave	Valor	
[0]	21	lst
[1]	183	

```
>>> ddd = dict()  
>>> ddd['age'] = 21  
>>> ddd['course'] = 182  
>>> print(ddd)  
{'course': 182, 'age': 21}  
>>> ddd['age'] = 23  
>>> print(ddd)  
{'course': 182, 'age': 23}
```

Diccionario		
Clave	Valor	
['course']	182	ddd
['age']	21	

# Literales Diccionarios (Constantes)

- Los literales de diccionarios utilizan **llaves { }** y tienen una lista de pares clave: valor
- Se puede hacer un **diccionario vacío** simplemente con { }

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print(ooo)
{ }
>>>
```

¿El nombre más común?

# ¿El nombre más común?

marquard

zhen

csev

zhen

cwen

marquard

zhen

csev

cwen

zhen

csev

marquard

zhen

# ¿El nombre más común?

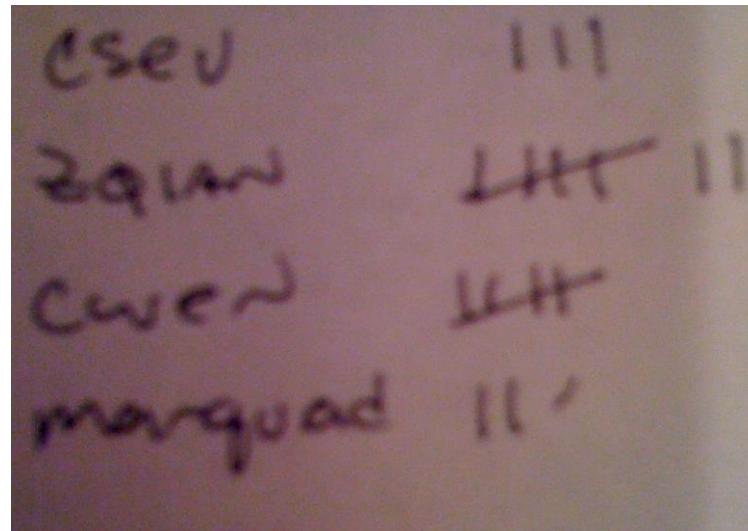
marquard

zhen

csev

zhen

cwen



csev

cwen

zhen

csev

marquard

zhen

# Muchos contadores con un diccionario

Un uso común de los diccionarios es **contar** cuántas veces “vemos” algo

```
>>> ccc = dict()  
>>> ccc['csev'] = 1  
>>> ccc['cwen'] = 1  
>>> print(ccc)  
{'csev': 1, 'cwen': 1}  
>>> ccc['cwen'] = ccc['cwen'] + 1  
>>> print(ccc)  
{'csev': 1, 'cwen': 2}
```

Clave	Valor
csev	111
zqian	LHHT 11
Cwen	LHHT
marqued	11'

# Errores con diccionarios

Es un **error** hacer referencia a una clave **que no está** en el diccionario

Podemos usar el operador **in** para ver si una clave está en el diccionario

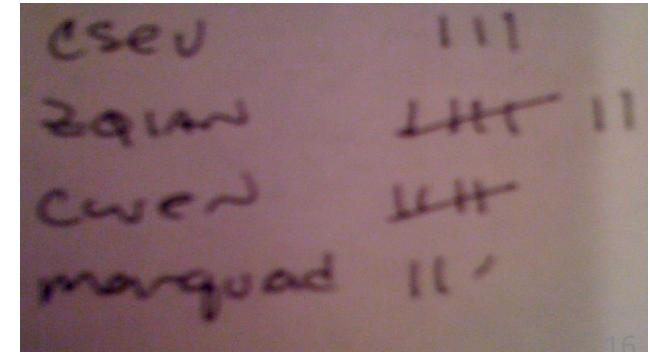
```
>>> ccc = dict()  
>>> print(ccc['csev'])  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
KeyError: 'csev'  
>>> 'csev' in ccc  
False
```

# Cuando vemos un nuevo nombre

Cuando encontramos un nuevo nombre, tenemos que sumar una nueva entrada al diccionario, y si es la segunda o posteriores veces que vemos el nombre, simplemente sumanos uno a la cuenta en el diccionario con ese nombre

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}



# El método get para diccionarios

El patrón para comprobar si un valor **está actualmente** en un diccionario y asumir un **valor por defecto** si no lo está, es tan común, que el método **get()** lo hace por nosotros

```
if name in counts:  
    x = counts[name]  
else :  
    x = 0  
  
x = counts.get(name, 0)
```

Valor por defecto si la clave no existe (sin error)

```
{'csev': 2, 'zqian': 1, 'cwen': 2}
```

# Cuenta simplificada con get()

Podemos usar `get()` y proporcionar un valor por defecto de 0 cuando la clave no está todavía en el diccionario – y entonces sumar solo 1

```
counts = dict()  
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']  
for name in names :  
    counts[name] = counts.get(name, 0) + 1  
print(counts)
```

Valor por  
defecto



{'csev': 2, 'zqian': 1, 'cwen': 2}

# Cuenta simplificada con get()

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```



<https://www.youtube.com/watch?v=2AoxCkySv34>

# Contando palabras en texto

Writing programs (or programming) is a very creative and rewarding activity. You can write programs for many reasons ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem. This book assumes that everyone needs to know how to program and that once you know how to program, you will figure out what you want to do with your newfound skills.

We are surrounded in our daily lives with computers ranging from laptops to cell phones. We can think of these computers as our “personal assistants” who can take care of many things on our behalf. The hardware in our current-day computers is essentially built to continuously ask us the question, “What would you like me to do next?”

Our computers are fast and have vast amounts of memory and could be very helpful to us if we only knew the language to speak to explain to the computer what we would like it to do next. If we knew this language we could tell the computer to do tasks on our behalf that were repetitive. Interestingly, the kinds of things computers can do best are often the kinds of things that we humans find boring and mind-numbing.

# Patrón de cuenta

```
counts = dict()
print('Escribe una línea:')
line = input('')

words = line.split()

print('Palabras:', words)

print('Contando....')
for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Cuentas', counts)
```

El patrón general para contar las palabras en una línea de texto es dividir la línea en palabras (**split**), y entonces iterar a través de las palabras usando un **diccionario** para llevar la cuenta de cada palabra de manera independiente.

```
python wordcount.py
```

Escribe una linea:

the clown ran after the car and the car ran into the tent  
and the tent fell down on the clown and the car

Palabras: ['the', 'clown', 'ran', 'after', 'the', 'car',  
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',  
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',  
'and', 'the', 'car']

Contando...

Cuentas {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1,  
'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7,  
'tent': 2}



<http://www.flickr.com/photos/71502646@N00/2526007974/>

```
counts = dict()
line = input('Escribe una línea:')
words = line.split()

print('Palabras:', words)
print('Contando...')

for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Cuentas', counts)
```



python wordcount.py

Escribe una línea:

the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car

Palabras: ['the', 'clown', 'ran', 'after', 'the', 'car', 'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and', 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the', 'car']

Counting...

Cuentas {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}

# Bucles for y diccionarios

Aunque los diccionarios no están almacenados en orden, podemos escribir un **bucle for** que itera por todas las entradas del diccionario – realmente **las claves**, y entonces mirar los **valores**

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print(key, counts[key])
...
jan 100
chuck 1
fred 42
>>>
```

# Recuperando listas de claves y valores

Se puede recuperar una lista de claves, valores o ítems (clave+valor) de un diccionario

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

¿Qué es una “tupla”? - Próximamente...

# Extra: ¡Dos variables de iteración!

- Podemos iterar a través de las **parejas clave-valor** de un diccionario usando **dos variables de iteración**

- En cada iteración, la primera variable es la clave y la segunda, el valor correspondiente a esa clave

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

jan 100  
chuck 1  
fred 42

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
name = input('Nombre de fichero: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

```
python words.py
Nombre de fichero: words.txt
to 16
```

```
python words.py
Nombre de fichero: clown.txt
the 7
```

Usando dos bucles anidados

# Resumen

- ¿Qué es una colección?
- Listas versus diccionarios
- Constantes de diccionarios
- La palabra más común
- Usando el método get()
- Búsqueda y falta de orden
- Escribiendo bucles con diccionarios
- Antícpo: tuplas
- Ordenando diccionarios

## Ejercicio

Escribe un programa que lea el archivo mbox-short.txt e indique quién ha enviado el mayor número de e-mails junto con el número de e-mails. El programa busca líneas con 'From ' y toma la segunda palabra de estas líneas como la persona que envió el e-mail.

Debes crear un diccionario Python que mapea el nombre del emisor con el número de veces que aparece en el archivo. Después de generar el diccionario, el programa iterá a través del diccionario para ver quién ha sido el emisor con más mensajes.



# Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Tuplas

## Unidad 10

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Las tuplas son como las listas

Las tuplas son otra clase de **secuencia** que se parece mucho a las listas – tienen elementos que pueden ser **indexados desde 0**

```
>>> x = ('Glenn', 'Sally', 'Joseph')      >>> for iter in y:  
>>> print(x[2])                      ...     print(iter)  
Joseph                           ...  
>>> y = ( 1, 9, 2 )                   1  
>>> print(y)                         9  
(1, 9, 2)                          2  
>>> print(max(y))                  >>>  
9
```

# Pero... las tuplas son “inmutables”

Al contrario que una lista, una vez crees una tupla, **no puedes alterar su contenido** – como un string

```
>>> x = [9, 8, 7]          >>> y = 'ABC'           >>> z = (5, 4, 3)
>>> x[2] = 6              >>> y[2] = 'D'            >>> z[2] = 0
>>> print(x)              Traceback: 'str'      Traceback: 'tuple'
>>>[9, 8, 6]                object does        object does
>>>                         not support item   not support item
>>>                         Assignment       Assignment
>>>
```

# Cosas que no se pueden hacer con tuplas

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
>>>
```

# Una historia de dos secuencias

```
>>> l = list()  
>>> dir(l)  
['append', 'count', 'extend', 'index', 'insert', 'pop',  
'remove', 'reverse', 'sort']  
  
>>> t = tuple()  
>>> dir(t)  
['count', 'index']
```

# Las tuplas son más eficientes

- Dado que Python sabe que las tuplas no tienen que modificarse, **son más simples y más eficientes** en términos de uso de memoria y rendimiento en comparación con las listas
- Así que en nuestros programas cuando hagamos "variables temporales" usaremos mejor tuplas en vez de listas

# Tuplas y Asignaciones

- También podemos poner una tupla en la parte izquierda de una sentencia de asignación
- Podemos incluso omitir los paréntesis

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```

# Tuplas y Diccionarios

El método **ítems()**  
de los diccionarios  
retorna una lista de  
tuplas (clave, valor)

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

# Las tuplas son comparables

Los operadores de comparación funcionan con tuplas y otras secuencias. Si el primer elemento es igual, Python continúa con el próximo hasta que algún elemento es diferente.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

# Ordenando listas de tuplas

Podemos tomar ventaja de la posibilidad de ordenar una lista de tuplas para obtener una versión ordenada de un diccionario

Primero ordenamos el diccionario por la clave usando el método **items()** y la función **sorted()**

```
>>> d = {'a':10, 'b':1, 'c':22}  
>>> d.items()  
dict_items([('a', 10), ('c', 22), ('b', 1)])  
>>> sorted(d.items())  
[('a', 10), ('b', 1), ('c', 22)]
```

# Usando sorted()

Podemos hacer esto incluso más directamente si usamos la función **sorted**, que toma una secuencia como parámetro y devuelve una secuencia ordenada

```
>>> d = { 'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[ ('a', 10), ('b', 1), ('c', 22) ]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

# Ordenar por valores

- Si pudiéramos construir una lista de tuplas de la forma **(valor, clave)**, podríamos ordenar por el valor
- Podemos hacer esto con un **for** que cree una lista de tuplas

```
>>> c = { 'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...     tmp.append( (v, k) )
...
>>> print(tmp)
[ (10, 'a'), (22, 'c'), (1, 'b') ]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[ (22, 'c'), (10, 'a'), (1, 'b') ]
```

## Las 10 palabras más comunes

```
fhand = open('romeo.txt')
counts = {}
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = []
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)

for val, key in lst[:10] :
    print(key, val)
```

# Una versión más corta

```
>>> c = { 'a':10, 'b':1, 'c':22}

>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )

[(1, 'b'), (10, 'a'), (22, 'c')]
```

Las **listas por comprensión** crean listas dinámicas. En este caso, hacemos una lista de tuplas invertidas y las ordenamos

<http://wiki.python.org/moin/HowTo/Sorting>

# Resumen

- Sintaxis de las tuplas
- Inmutabilidad
- Comparaciones
- Ordenación
- Tuplas en sentencias de asignación
- Ordenador de diccionarios por clave o valor

# Ejercicio

Escribe un programa que lea el archivo mbox-short.txt y muestre la distribución por horas del día de los mensajes.

Se puede extraer la hora de las líneas que comiencen por 'From ', extrayendo la hora completa y dividiendo una segunda vez con los ':' como delimitador.

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16  
2008

Una vez hayas obtenido el total de mensajes en cada hora, muestra los totales ordenados por hora. Deberías obtener el resultado que se muestra a la derecha

04 3  
06 1  
07 1  
09 2  
10 3  
11 6  
14 1  
15 2  
16 4  
17 2  
18 1  
19 1



## Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Objetos Python

## Unidad 11

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Introducción

- La Programación Orientada a Objetos es la evolución de la Programación Estructurada
- Trata de modelar los programas como si fueran “Objetos” del mundo real
  - Atributos de los objetos
  - Acciones que pueden realizar
- No tendrás la visión global hasta que lo utilices en el **contexto de un problema real complejo**

La Orientación a Objetos está en  
todas partes (y ya la habías estado  
usando)

## 5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

### 5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.append(x)`

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.extend(L)`

Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

`list.insert(i, x)`

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.remove(x)`

Remove the first item from the list whose value is `x`. It is an error if there is no such item.

`list.pop([i])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

## 12.6. `sqlite3` — DB-API 2.0 interface for SQLite databases

[Source code: Lib/sqlite3/](#)

---

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The `sqlite3` module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by [PEP 249](#).

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
            (date text, trans text, symbol text, qty real, price real)''')
```

# Comencemos con los programas



```
inp = input('Planta Europa?')  
usf = int(inp) + 1  
print('Planta USA', usf)
```

Planta Europa? 0  
Planta EEUU 1



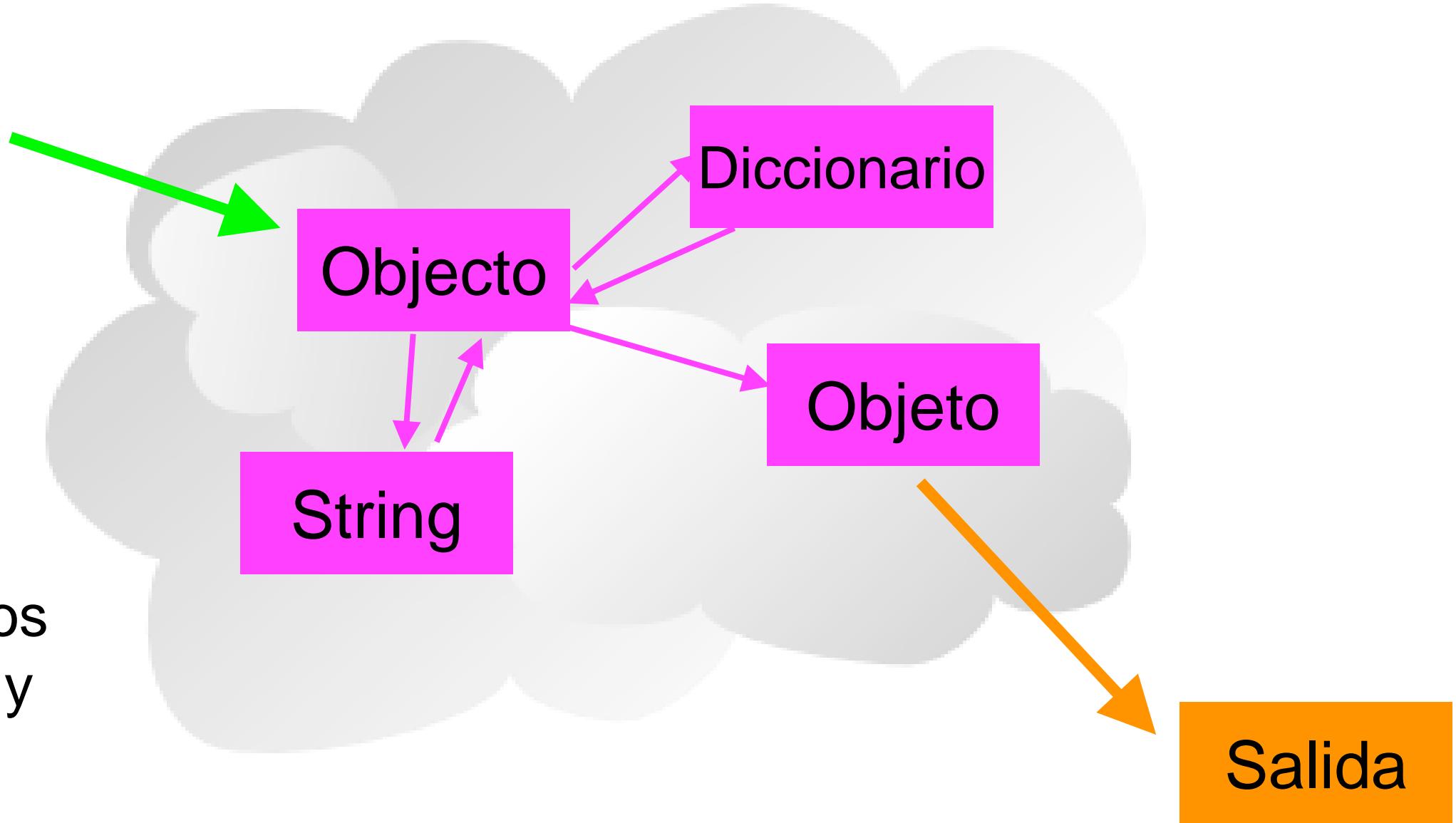
# Orientación a Objetos

- Un programa está hecho de muchos objetos **cooperando**
- En lugar de ser el "programa completo" – **cada objeto** es una **pequeña "isla"** dentro del programa trabajando cooperativamente con otros objetos
- Un programa está hecho de uno o más objetos trabajando juntos – los objetos hacen uso de las **capacidades de los demás**

# Objeto

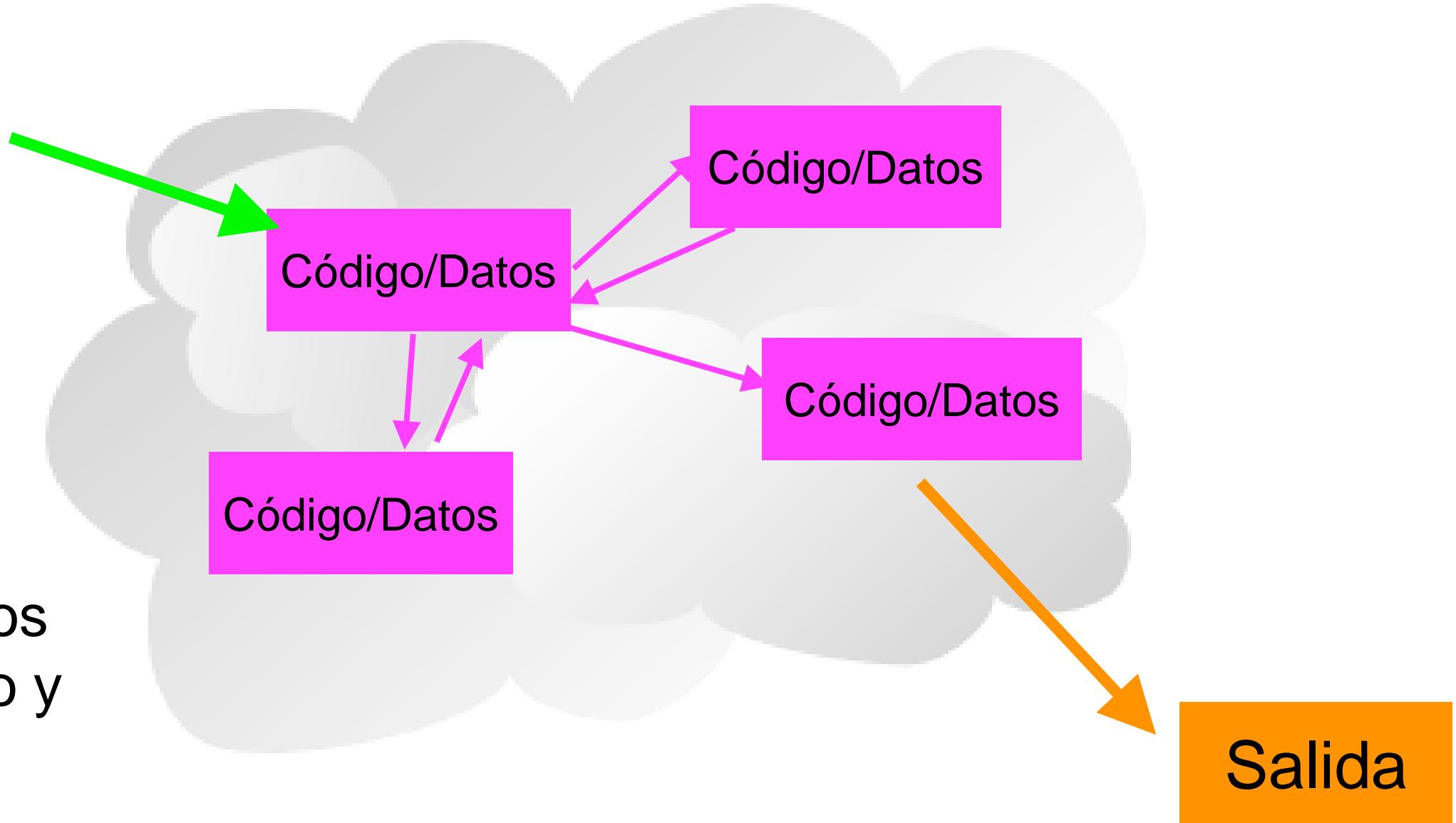
- Un Objeto incluye **Código y Datos**
- Un aspecto clave de la orientación a objetos es dividir un problema en partes más pequeñas manejables (**divide y vencerás**)
- Los objetos tienen límites/fronteras que nos permiten ignorar los detalles innecesarios
- Hemos estado usando objetos todo el rato: objetos string, objetos integer, objetos diccionarios, objetos listas...

**Entrada**



Los objetos  
se crean y  
usan

**Entrada**



Los objetos  
son código y  
datos

Entrada

Los objetos  
ocultan  
detalles –  
nos permiten  
ignorar el  
resto del  
programa

Código/Datos

Código/Datos

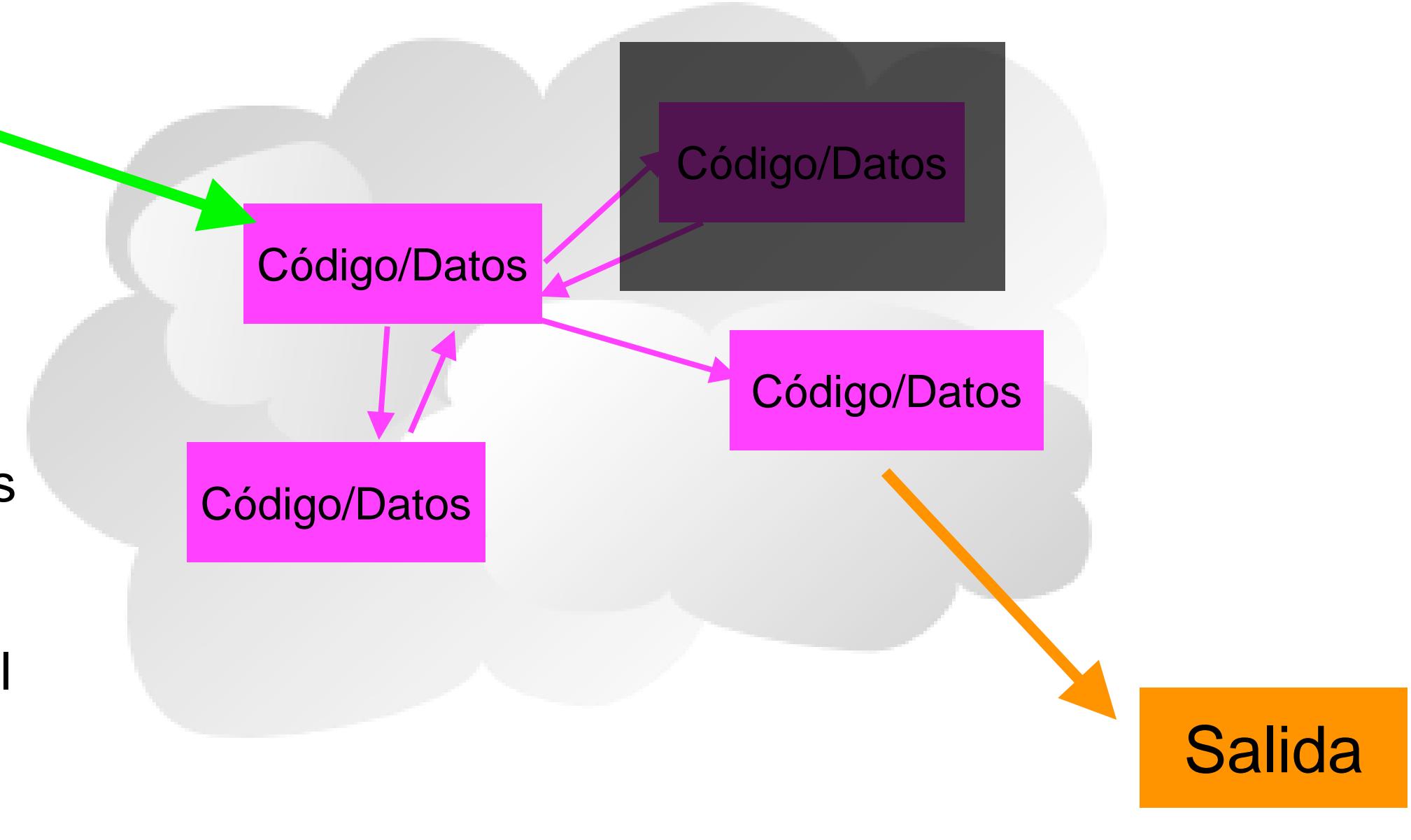
Código/Datos

Código/Datos

Salida

**Entrada**

Los objetos  
ocultan  
detalles –  
permiten al  
resto del  
programa  
ignorarlos



# Definiciones



- **Clase** – una plantilla
- **Método o Mensaje** – Una capacidad definida de una clase
- **Campo o atributo** – Dato en una clase
- **Objeto o Instancia** – Una instancia particular de una clase

# Terminología: Clase



Define las características **abstractas** de una cosa (**objeto**), incluyendo sus **características (sus atributos, campos o propiedades)** y su **comportamiento (las cosas que puede hacer, o métodos u operaciones)**. Se podría decir que una clase es una plantilla o plano que describe la naturaleza de algo. Por ejemplo, la clase Perro podría consistir de rasgos compartidos por todos los perros, tales como la raza y el color (características), y la habilidad para ladrar y sentarse (comportamientos).

# Terminología: Instancia



Podemos tener instancias u objetos de una clase. Las instancias son los objetos reales creados durante la ejecución del programa.

En "jerga" de programador, el objeto Lassie **es una instancia** de la clase Perro. El conjunto de valores de los atributos de un objeto en particular, forman su **estado**. El objeto consiste en el estado y el comportamiento definido en la clase del objeto.

Se usan los términos Objecto e Instancia como sinónimos

# Terminología: Método



Una habilidad de un objeto. En lenguaje, los métodos son "**verbos**". Lassie, siendo un Perro, tiene la habilidad para ladrar. Así que ladrar() es uno de los métodos de Lassie. Podría tener más métodos también, por ejemplo sentar() o comer() o pasear(). Dentro del programa, normalmente usar un método **solo afecta a un objeto en particular**; todos los Perros saben ladrar, pero solo necesitamos que lo haga uno en particular

Método y Mensaje son usados como sinónimos.

# Algunos objetos Python

```
>>> x = 'abc'  
>>> type(x)  
<class 'str'>  
>>> type(2.5)  
<class 'float'>  
>>> type(2)  
<class 'int'>  
>>> y = list()  
>>> type(y)  
<class 'list'>  
>>> z = dict()  
>>> type(z)  
<class 'dict'>
```

```
>>> dir(x)  
[ ... 'capitalize', 'casefold', 'center', 'count',  
'encode', 'endswith', 'expandtabs', 'find',  
'format', ... 'lower', 'lstrip', 'maketrans',  
'partition', 'replace', 'rfind', 'rindex', 'rjust',  
'rpartition', 'rsplit', 'rstrip', 'split',  
'splitlines', 'startswith', 'strip', 'swapcase',  
'title', 'translate', 'upper', 'zfill']  
>>> dir(y)  
[... 'append', 'clear', 'copy', 'count', 'extend',  
'index', 'insert', 'pop', 'remove', 'reverse',  
'sort']  
>>> dir(z)  
[..., 'clear', 'copy', 'fromkeys', 'get', 'items',  
'keys', 'pop', 'popitem', 'setdefault', 'update',  
'values']
```

# Una Clase de ejemplo



**class** es una palabra reservada

Cada objeto Fiestero tiene una parte de código

Decirle al objeto que "ejecute" el código `fiesta()` dentro de él

```
class Fiestero:  
    x = 0  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos",self.x)  
  
obj = Fiestero()  
  
obj.fiesta()  
obj.fiesta()  
obj.fiesta()
```

Esta es la plantilla para hacer objetos Fiestero

Cada objeto Fiestero tiene una parte de datos

Construye un objeto Fiestero y almacénalo en una variable

`Fiestero.fiesta(obj)`

```
class Fiestero:
```

```
    x = 0
```

```
        def fiesta(self) :  
            self.x = self.x + 1  
            print("Tan lejos",self.x)
```

```
obj = Fiestero()
```

```
obj.fiesta()
```

```
obj.fiesta()
```

```
obj.fiesta()
```

```
$ python party1.py
```

```
class Fiestero:
```

```
    x = 0
```

```
        def fiesta(self) :  
            self.x = self.x + 1  
            print("Tan lejos",self.x)
```

```
obj = Fiestero()
```

```
obj.fiesta()
```

```
obj.fiesta()
```

```
obj.fiesta()
```

```
$ python party1.py
```

obj



```
class Fiestero:  
    x = 0  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos",self.x)  
  
obj = Fiestero()  
  
obj.fiesta()  
obj.fiesta()  
obj.fiesta()
```

```
$ python party1.py  
Tan lejos 1
```

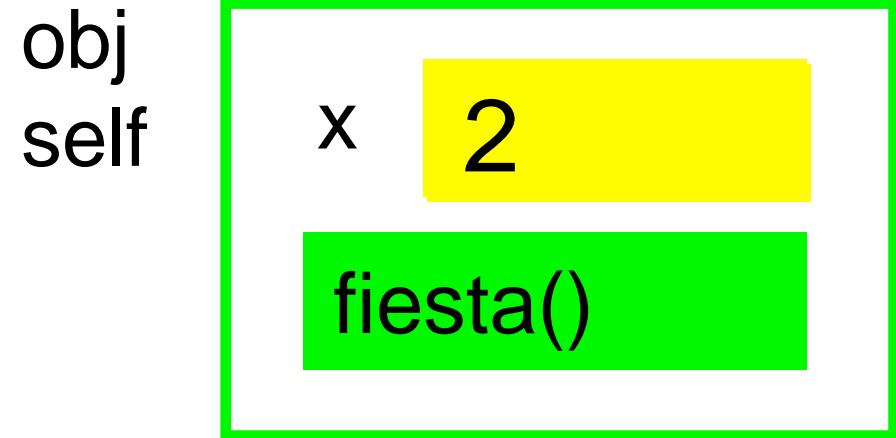
obj  
self



Fiestero.fiesta(obj)

```
class Fiestero:  
    x = 0  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos",self.x)  
  
obj = Fiestero()  
  
obj.fiesta()  
obj.fiesta()  
obj.fiesta()
```

```
$ python party1.py  
Tan lejos 1  
Tan lejos 2
```



Fiestero.fiesta(obj)

```
class Fiestero:  
    x = 0  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos",self.x)  
  
obj = Fiestero()  
  
obj.fiesta()  
obj.fiesta()  
obj.fiesta()
```

```
$ python party1.py  
Tan lejos 1  
Tan lejos 2  
Tan lejos 3
```

obj  
self



Fiestero.fiesta(obj)

# Jugando con dir() y type()

# Una forma sencilla de encontrar capacidades

- El comando **dir()** muestra capacidades
- Ignora las que empiezan con subrayados "\_" - son usadas por Python
- El resto son operaciones reales que el objeto puede realizar
- Es como **type()** – nos dice \*algo\* sobre una variable

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(y)
['__add__', '__class__',
 '__contains__', '__delattr__',
 '__delitem__', '__delslice__',
 '__doc__', ... '__setitem__',
 '__setslice__', '__str__',
 'append', 'clear', 'copy',
 'count', 'extend', 'index',
 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>>
```

```
class Fiestero:  
    x = 0  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos", self.x)  
  
obj = Fiestero()  
  
print("Type", type(obj))  
print("Dir ", dir(obj))
```

Podemos usar `dir()` para encontrar las “capacidades” de nuestra nueva clase.

```
$ python party2.py  
Type <class '__main__.Fiestero'>  
Dir  ['__class__', ... 'fiesta', 'x']
```

# Intenta dir() con un string

```
>>> x = 'Hello there'  
>>> dir(x)  
['__add__', '__class__', '__contains__', '__delattr__',  
'__doc__', '__eq__', '__ge__', '__getattribute__',  
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',  
'__hash__', '__init__', '__le__', '__len__', '__lt__',  
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',  
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',  
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',  
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',  
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',  
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',  
'splitlines', 'startswith', 'strip', 'swapcase', 'title',  
'translate', 'upper', 'zfill']
```

# Ciclo de vida de los objetos

# Ciclo de vida de los objetos

- Los objetos son **creados, usados y descartados**
- Tenemos bloques especiales de código (métodos) que son llamados
  - Cuando se crea el objeto (**constructor**)
  - Cuando se destruye el objeto (**destructor**)
- Los constructores se usan mucho
- Los destructores rara vez se usan (en Python)

# Constructor

El propósito principal del constructor es **inicializar** algunas variables de instancia (los atributos) para que tengan valores apropiados cuando se cree el objeto

[https://es.wikipedia.org/wiki/Constructor\\_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Constructor_(inform%C3%A1tica))

```
class Fiestero:  
    x = 0  
  
    def __init__(self):  
        print('Me construyen')  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print("Tan lejos",self.x)  
  
    def __del__(self):  
        print('Me destruyen', self.x)  
  
obj = Fiestero()  
obj.fiesta()  
obj.fiesta()  
  
obj=42  
print('obj contiene',obj)
```

```
$ python party3.py  
Me construyen  
Tan lejos 1  
Tan lejos 2  
Me destruyen 2  
obj contiene 42
```

El constructor y el destructor son opcionales. El constructor es usado para inicializar variables. El destructor es rara vez usado

# Muchas instancias

- Podemos crear muchos objetos – la clase es la plantilla para el objeto
- Podemos almacenar cada objeto en su propia variable
- Podemos tener **muchas instancias de la misma clase**
- Cada instancia tiene **su propia copia** de las variables de instancia (atributos)

```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
s = Fiestero("Sally")  
j = Fiestero("Jim")  
  
s.fiesta()  
j.fiesta()  
s.fiesta()
```

Los constructores pueden tener **parámetros adicionales**. Esto sirve para inicializar los atributos de cada instancia en particular

party4.py

```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
s = Fiestero("Sally")  
j = Fiestero("Jim")  
  
s.fiesta()  
j.fiesta()  
s.fiesta()
```

```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
s = Fiestero("Sally")  
j = Fiestero("Jim")  
  
s.fiesta()  
j.fiesta()  
s.fiesta()
```

x: 0

nombre:

```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
s = Fiestero("Sally")  
j = Fiestero("Jim")  
  
s.fiesta()  
j.fiesta()  
s.fiesta()
```

Tenemos dos  
instancias  
independientes

x: 0

nombre: Sally

x: 0

nombre: Jim

```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
s = Fiestero("Sally")  
j = Fiestero("Jim")  
  
s.fiesta()  
j.fiesta()  
s.fiesta()
```

Sally construido  
Jim construido  
Sally contador 1  
Jim contador 1  
Sally contador 2

# Herencia

<http://www.ibiblio.org/g2swap/byt eofpython/read/inheritance.html>

# Herencia

- Cuando hacemos una nueva clase podemos reutilizar una clase existente y **heredar todas las capacidades (y datos) que tenga**, sumando entonces algo más en nuestra nueva clase
- Otra forma de **reutilización**
- Escribir una vez – reutilizar muchas veces
- La nueva **clase (hija)** tiene todas las capacidades de la vieja **clase (padre)** – y algunas más

# Terminología: Herencia



Las '**subclases**' son versiones especializadas de una clase, que hereda atributos y comportamiento de la clase padre, y puede introducir las suyas propias.

```

class Fiestero:
    x = 0
    nombre = ""

    def __init__(self,z):
        self.nombre=z
        print(self.nombre,"construido")

    def fiesta():
        self.x = self.x + 1
        print(self.nombre,"contador",self.x)

class Jugador(Fiestero):
    puntos = 0
    def jugada():
        self.puntos = self.puntos + 1
        self.fiesta()

        print(self.nombre,"puntos",self.puntos)

```

```

s = Fiestero("Sally")
s.fiesta()

j = Jugador("Jim")
j.fiesta()
j.jugada()

```

Jugador es una clase que **"extiende"** a Fiestero. Tiene todas las capacidades de Fiestero y más aún.

```

class Fiestero:
    x = 0
    nombre = ""

    def __init__(self,z):
        self.nombre=z
        print(self.nombre,"construido")

    def fiesta():
        self.x = self.x + 1
        print(self.nombre,"contador",self.x)

class Jugador(Fiestero):
    puntos = 0

    def jugada(self):
        self.puntos = self.puntos + 1
        self.fiesta()

        print(self.nombre,"puntos",self.puntos)

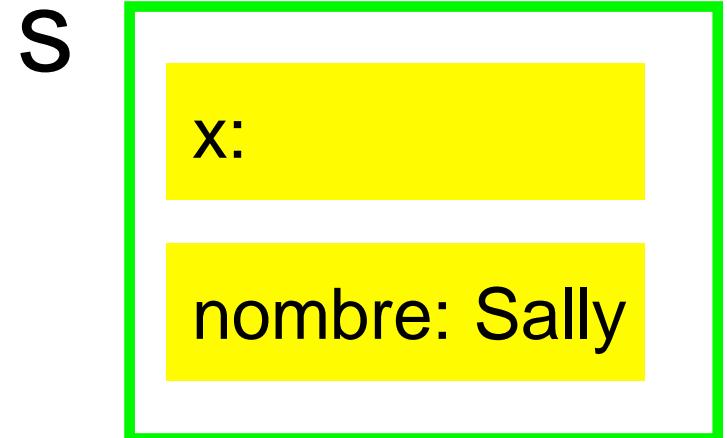
```

```

s = Fiestero("Sally")
s.fiesta()

j = Jugador("Jim")
j.fiesta()
j.jugada()

```



```
class Fiestero:  
    x = 0  
    nombre = ""  
  
    def __init__(self,z):  
        self.nombre=z  
        print(self.nombre,"construido")  
  
    def fiesta(self) :  
        self.x = self.x + 1  
        print(self.nombre,"contador",self.x)  
  
class Jugador(Fiestero):  
    puntos = 0  
    def jugada(self):  
        self.puntos = self.puntos + 1  
        self.fiesta()  
  
        print(self.nombre,"puntos",self.puntos)
```

```
s = Fiestero("Sally")  
s.fiesta()  
  
j = Jugador("Jim")  
j.fiesta()  
j.jugada()
```

j

x:

nombre: Jim

puntos:

# Definiciones

- **Clase** – una plantilla
- **Atributo** – Una variable dentro de una clase
- **Método** – Una función dentro de una clase
- **Objeto** – Una instancia particular de una clase
- **Constructor** – Código que se ejecuta cuando un objeto es creado
- **Herencia** – La habilidad para extender una clase y crear una nueva clase



# Resumen

- La **Programación Orientada a Objetos** es una forma muy estructurada de **reutilizar** código
- Podemos agrupar datos y funcionalidad juntos para crear muchas instancias independientes de una clase



## Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License.

Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

Spanish Version: Daniel Garrido (dgm@uma.es)

# Interfaces Gráficas de Usuario

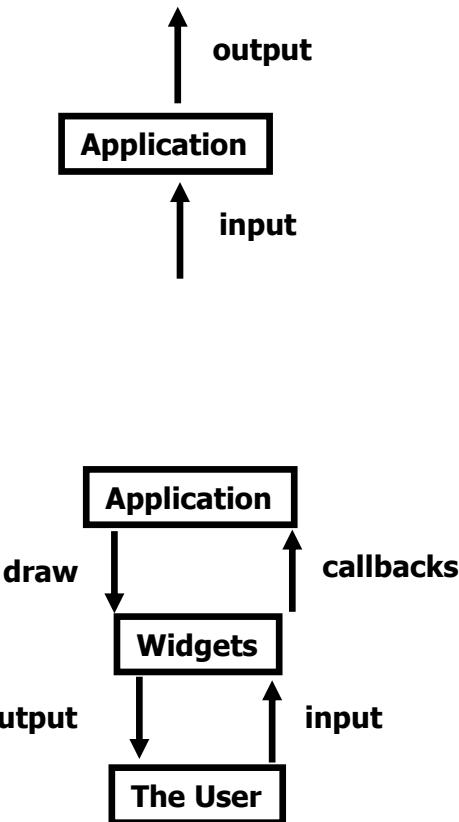
## Unidad 13

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Conceptos Generales

- Una Interfaz Gráfica de Usuario (GUI) permite interactuar con una aplicación de manera "amigable"
- Programación convencional:
  - La secuencia de operaciones es determinada por el programa
  - Lo que quieres que ocurra, ocurre cuando tú quieres
- Programación dirigida por eventos:
  - La secuencia de operaciones es determinada por la interacción del usuario con la interfaz de la aplicación
  - Cualquier cosa que ocurra, ocurre en cualquier momento



# La "experiencia de usuario"

- Los usuarios aprenden "por intentos"
  - Rara vez leen los manuales
  - Pensar cuidadosamente cuál tendría que ser el comportamiento por defecto de cualquier función de nuestra aplicación
- Deberíamos ayudar al usuario...
  - Usando teclas y opciones familiares (p.ej. Ctrl + C para copiar)
  - Incluyendo ayuda y tutoriales

# Diseñando para los usuarios

- En cada momento, pensar: ¿es obvio lo que hace?
- Hacer todas las pantallas tan simples como sea posible
- Desconectar funcionalidades hasta que se necesiten. Por ejemplo:

```
model_menu.entryconfig("Run model", state="disabled")
```

# Hasta que el usuario elija los ficheros, entonces:

```
model_menu.entryconfig("Run model", state="normal")
```

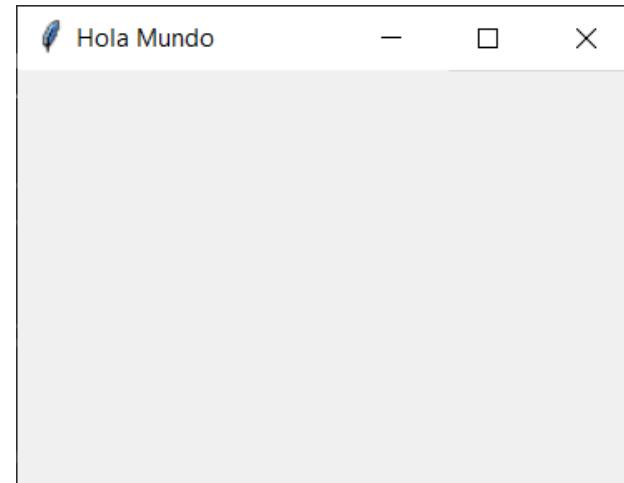
- Ocultar la funcionalidad compleja y las opciones para cambiar el comportamiento por defecto en los menús de "Opciones"

# Tkinter

- Tkinter (“**ToolKit interface**”) es una librería Python que permite desarrollar GUIs
  - <https://docs.python.org/3/library/tk.html>
- Es una capa sobre Tcl/Tk para Python y está incluida por defecto en Python 3 (no es la única opción)

```
import tkinter as tk

ventana1=tk.Tk()
ventana1.title("Hola Mundo")
ventana1.mainloop()
```



# Orientación a Objetos y GUIs

- Podemos aplicar Orientación a Objetos para hacer nuestro código más reutilizable

```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Hola Mundo")
        self.ventana1.mainloop()

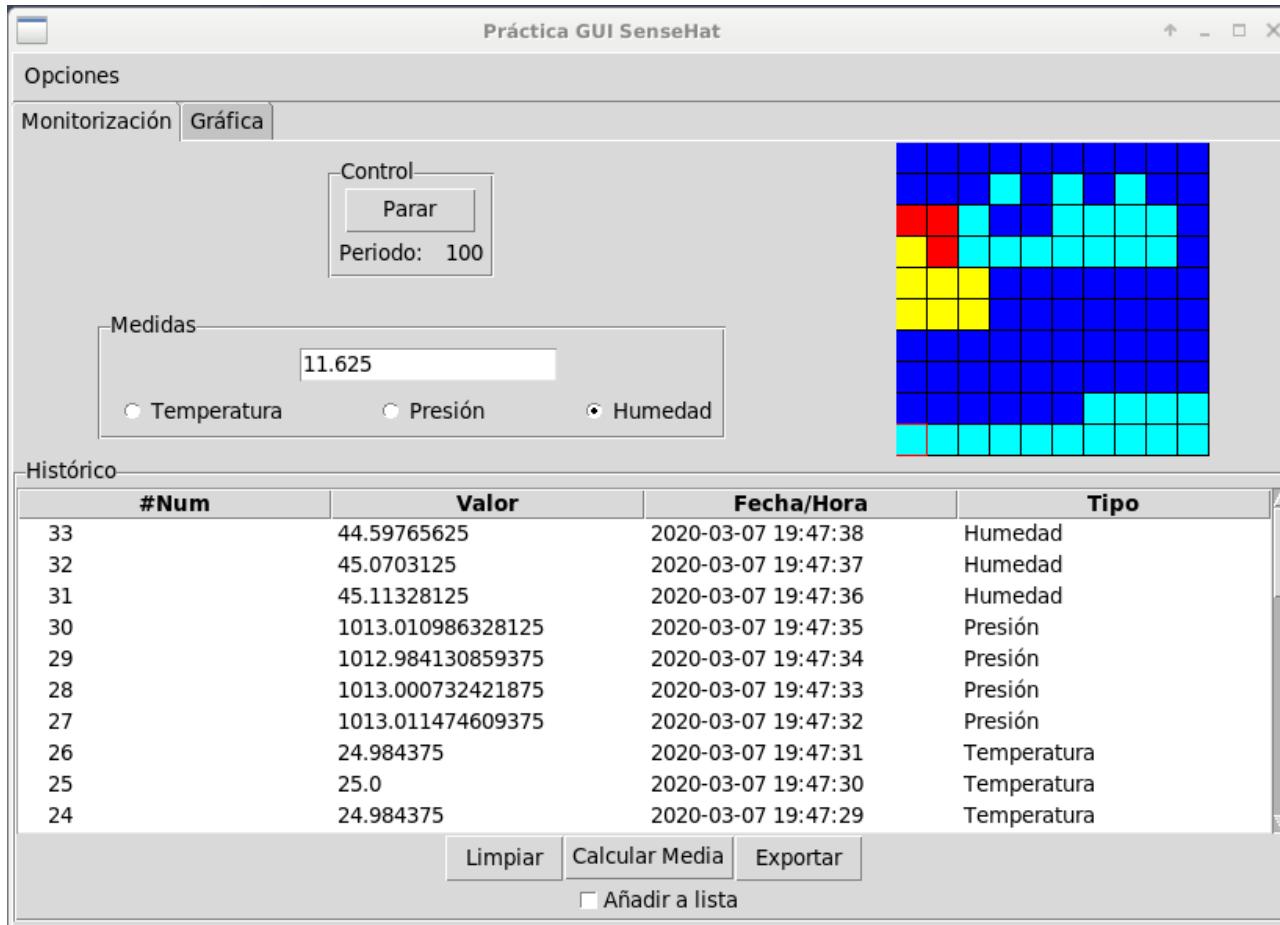
aplicacion1=Aplicacion()
```

# Repositorio de Ejemplos

- Todos los ejemplos (y más) de la presentación pueden encontrarse en el repositorio <https://github.com/dgarridouma/ejemplos-tkinter>

# Widgets

- Los widgets son elementos que se pueden añadir a nuestra Ventana y con los que interactuará el usuario. Ejemplos:
  - Buttons, Checkbuttons, Radiobuttons, Menus, Entry, Labels, Frames, Scale, Scrollbar, Canvas...



# Widgets

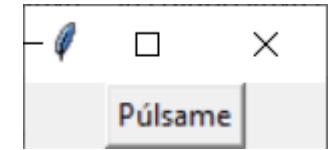
- Ejemplo: Añadir una Etiqueta (Label)
  - Crear la etiqueta
    - Pasar la Ventana donde se colocará y el texto
  - Definir la posición de la etiqueta ( `self.label.pack()` )
    - Indica que se coloque en la Ventana y se muestre
  - Comenzar el bucle de eventos ( `root.mainloop()` )



```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana=tk.Tk()
        self.label=tk.Label(self.ventana, text="Hola Mundo")
        self.label.pack()
        self.ventana.mainloop()

aplicacion1=Aplicacion()
```



# GUIs Manejo de Eventos

- Button:

- Para hacer que el usuario pueda interactuar con un botón, tenemos que usar **command** indicando qué método se ejecutará cuando se haga click sobre el botón

```
import tkinter as tk
from tkinter.messagebox import showinfo

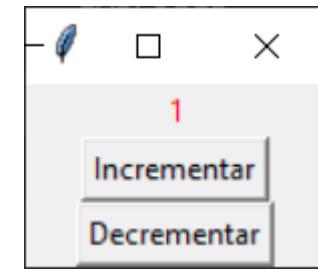
class Aplicacion:
    def __init__(self):
        self.ventana=tk.Tk()
        self.boton=tk.Button(self.ventana, text="Púlsame",
                             command=self.saludar)
        self.boton.pack()
        self.ventana.mainloop()

    def saludar(self):
        showinfo(message="Me has pulsado")

aplicacion1=Aplicacion()
```

# Ejercicio

- Implementar una interfaz de usuario como la que se muestra, de manera que cada vez que se pulse uno de los botones, el valor aumentará o disminuirá en una unidad
- Ayudas:
  - Utilizar un atributo "valor" (self.valor) en la clase, que contenga el valor actual mostrado (inicializado a 1)
  - Cuando se pulse un botón, podemos cambiar el texto de la etiqueta haciendo:
    - `self.label.config(text=self.valor)`
  - Podemos hacer que la etiqueta salga en rojo utilizando:
    - `self.label1.configure(foreground="red")`
  - Bonus: hacer que la etiqueta salga en azul si el valor es  $\geq 0$ , rojo en caso contrario



# Layouts

- Tkinter proporciona 3 formas diferentes de colocar los widgets dentro de su contenedor
  - No pueden mezclarse en el mismo contenedor
- Pack
  - Se indican posiciones relativas. Posiblemente el más fácil
- Grid
  - Se indica fila o columna. El tamaño se determina automáticamente
- Place
  - Permite indicar posiciones y medidas exactas en términos absolutos o relativos a otra ventana

# Pack layout

El método `pack()` especifica la posición de un widget dentro de su contenedor



Opción	Descripción
side	LEFT, RIGHT, TOP, BOTTOM,
fill	'both', 'x', 'y', o 'none'
expand	True o False

```
from tkinter import Tk, Label, PhotoImage, BOTTOM, LEFT, RIGHT, RIDGE
root = Tk()

text = Label(root, font=('Helvetica', 16, 'bold italic'), foreground='white', background='black', pady=10, text='Universidad de Málaga')
text.pack(side=BOTTOM)

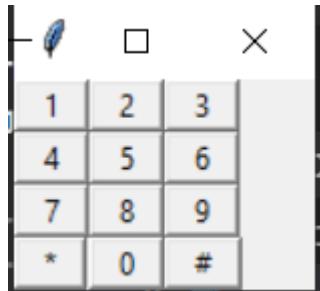
uma1 = PhotoImage(file='img/uma1.png')
umaLabel = Label(root, borderwidth=3, relief=RIDGE, image=uma1)
umaLabel.pack(side=LEFT)

uma2 = PhotoImage(file='img/uma2.png')
uma2Label = Label(root, image=uma2)
uma2Label.pack(side=RIGHT)

root.mainloop()
```

# Grid layout

El método `grid()` permite colocar widgets en una cuadrícula



## Opciones

`column`

`columnspan`

`row`

`rowspan`

```
from tkinter import Tk, Label, RAISED  
  
root = Tk()  
labels = [[ '1', '2', '3' ],  
          [ '4', '5', '6' ],  
          [ '7', '8', '9' ],  
          [ '*', '0', '#' ]]  
  
for r in range(4):  
    for c in range(3):  
        # crear label para fila r y columna c  
        label = Label(root,  
                      relief=RAISED,  
                      padx=10,  
                      text=labels[r][c])  
        # colocar label en fila r y columna c  
        label.grid(row=r, column=c)  
  
root.mainloop()
```

# Place layout

El método `place()` permite colocar widgets en posiciones específicas y con el tamaño indicado



```
import tkinter as tk
import random

root = tk.Tk()
root.geometry("170x200+30+30")

temas = ['Python', 'POO', 'Git', 'GUI', 'BBDD']
labels = range(5)
for i in range(5):
    ct = [random.randrange(256) for x in range(3)]
    ct_hex = "%02x%02x%02x" % tuple(ct)
    bg_colour = '#' + ''.join(ct_hex)
    l = tk.Label(root,
                 text=temas[i],
                 bg=bg_colour)
    l.place(x = 20, y = 30 + i*30, width=120,
            height=25)

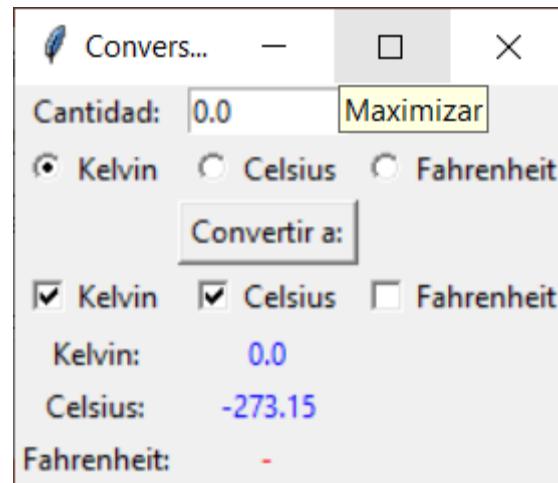
root.mainloop()
```

# Más Widgets...

- Vamos a ver más ejemplos de widgets del repositorio  
<https://github.com/dgarridouma/ejemplos-tkinter>
- En concreto vamos a ver ejemplos de los siguientes widgets:
  - Entry
  - Radiobutton
  - Checkbutton
  - Listbox y scroll

# Ejercicio

- Conversor de temperaturas: implementar un conversor de temperaturas con una interfaz similar a la siguiente



- Bonus: Añadir un control de lista a la interfaz que almacene todas las conversiones realizadas

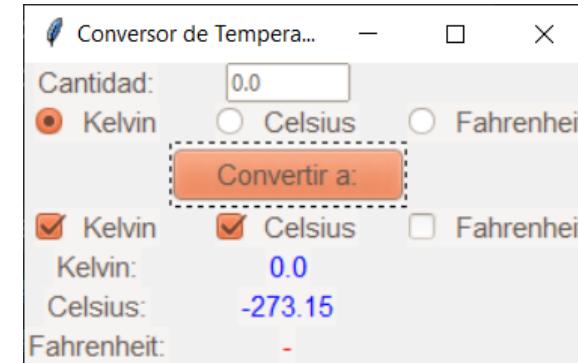
# ttk

- El modulo **ttk** proporciona acceso a "temas" en los widget
- La idea básica es separar el comportamiento de la apariencia (y un aspecto más "nativo")
- Para comenzar a usar Ttk
  - **from tkinter import ttk**
- Automáticamente varios widtget de Tk son reemplazados por las nuevas versiones
- Ttk viene con 6 nuevos widgets no incluidos en Tk
  - Combobox, Notebook, Progressbar, Separator, Sizegrip y Treeview

# Aplicando temas

- Vamos a cambiar el tema a nuestro conversor de temperaturas
  - <https://ttkthemes.readthedocs.io/en/latest/themes.html>
- Instalamos el paquete ttkthemes
  - pip3 install ttkthemes
- Importamos los paquetes necesarios
  - import tkinter as tk
  - import tkinter.ttk as ttk
  - from ttkthemes import ThemedStyle
- Aplicamos el tema a nuestra Ventana

```
self.ventana1=tk.Tk()
style = ThemedStyle(self.ventana1)
style.set_theme("radiance")
```
- Puede que tengamos que hacer algunos ajustes a nuestro código

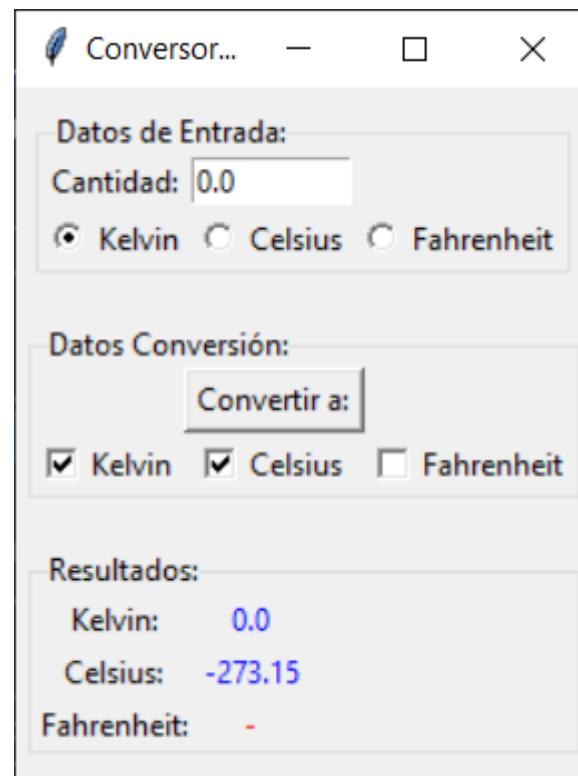


# Widgets Ttk

- Vamos a ver más ejemplos de widgets, incluyendo widgets Ttk y otros elementos como Menús: <https://github.com/dgarridouma/ejemplos-tkinter>
- En concreto vamos a ver ejemplos de los siguientes widgets:
  - Combobox
  - Menus
  - Notebook
  - Frame
  - Labelframe

# Ejercicio

- Vamos a mejorar la apariencia de nuestro conversor de unidades añadiendo widget LabelFrames. El resultado debe ser similar al siguiente:
  - Pista: en Resultados usar sticky="WE" en el LabelFrame

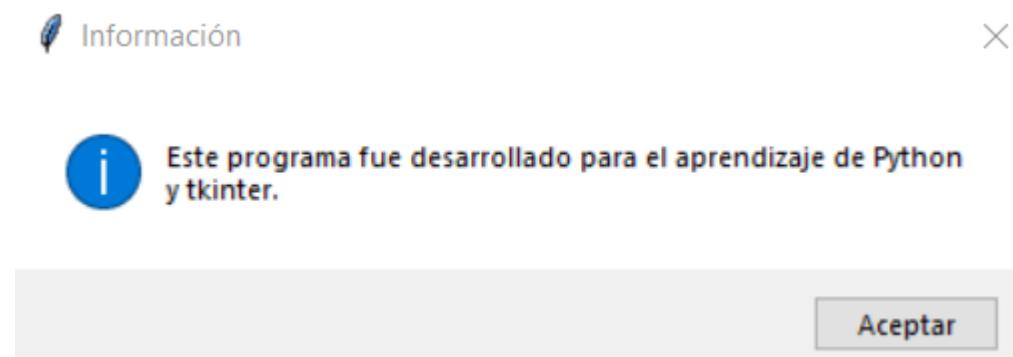


# Diálogos

- Los cuadros de información son un mecanismo muy útil para mostrar información al usuario
  - Hay de varios tipos (información, advertencia y error, sí/no)
- Ejemplo:

```
from tkinter import messagebox as mb
```

```
mb.showinfo("Información", "Este programa fue desarrollado para el  
aprendizaje de Python y tkinter.")
```



# Diálogos

- Los **diálogos** se usan para mostrar ventanas adicionales
  - Pueden ser usados para muy diversas tareas: recuperar información del usuario, modificar opciones, ...
- Se suelen desarrollar en una clase aparte
- Hay que dar la orden de mostrarlas y habitualmente tienen un comportamiento "modal". Hasta que no se cierran no se puede volver a la ventana inicial
- Una vez cerrada, podemos recoger el resultado

# Diálogos

- Definición

```
class DialogoTamano:

    def __init__(self, ventanaprincipal):
        self.dialogo=tk.Toplevel(ventanaprincipal)
        # Widgets del diálogo aquí
        self.dialogo.protocol("WM_DELETE_WINDOW", self.confirmar)
        self.dialogo.resizable(0,0)
        self.dialogo.grab_set()

    def mostrar(self):
        self.dialogo.wait_window()
        return (self.dato1.get(), self.dato2.get())

    def confirmar(self):
        self.dialogo.destroy()
```

# Diálogos

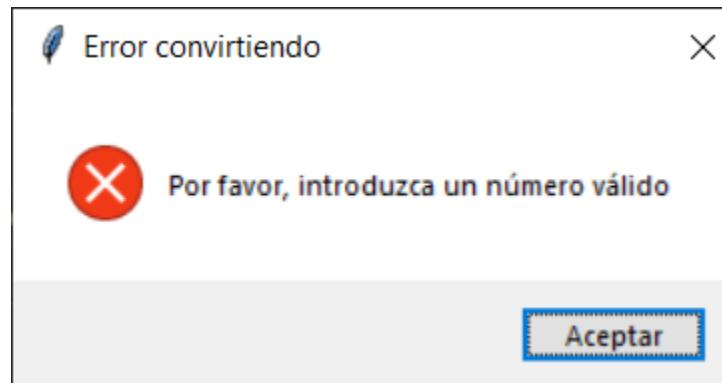
- Creación y recuperación de resultados

```
def configurar(self):  
    dialogo1 = DialogoTamano(self.ventana1)  
    s=dialogo1.mostrar()  
    self.ventana1.geometry(s[0]+"x"+s[1])
```

- Ejemplos completos en
  - <https://github.com/dgarridouma/ejemplos-tkinter>

# Ejercicio

- Vamos a comprobar que el número introducido en nuestro conversor es válido, y si no lo es, mostraremos un mensaje de error usando el método messagebox.showerror



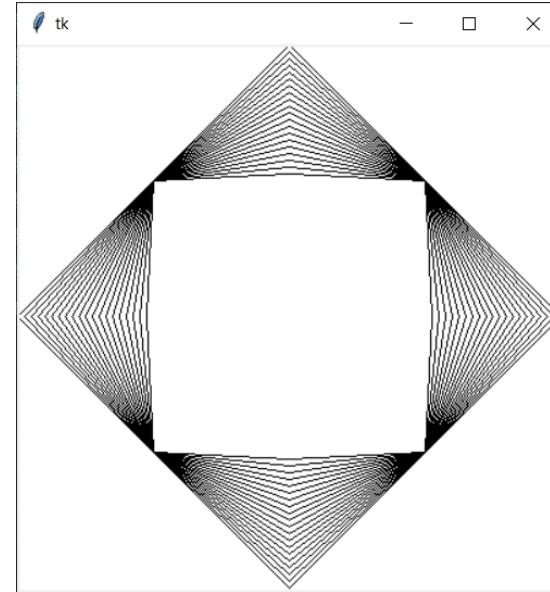
- Bonus: añadir un menu con una opción que muestre el típico diálogo "Acerca de"

# Y aún más widgets!

- <https://github.com/dgarridouma/ejemplos-tkinter>
- En concreto vamos a ver ejemplos de los siguientes widgets:
  - Spinbox
  - Progressbar
  - Scrolledtext
  - Treeview

# Canvas

- El widget Canvas permite mostrar elementos gráficos como líneas, rectángulos o círculos, así como controlar clicks del ratón o pulsaciones de teclas
- Una muestra...

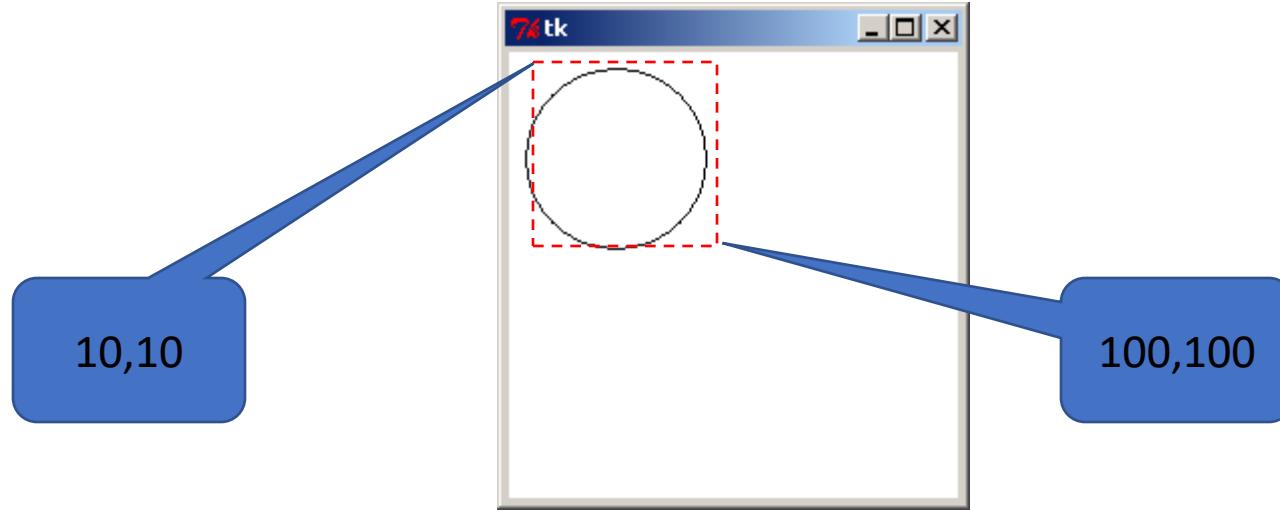


```
    self.canvas1=tk.Canvas(self.ventana1, width=400, height=400,
background="white")
    self.canvas1.pack()

    i=0
    while i<100:
        i+=5
        self.canvas1.create_line([(100,100),(200,100-i),
(300,100),(300+i,200),(300,300),(200,300+i),(100,300),
(100-i,200),(100,100)], smooth=0)
```

# Importante

## Saber cómo funcionan las coordenadas



```
from tkinter import *
tk = Tk()
c = Canvas(tk, width=800, height=600, bg="white")
c.create_oval(10,10,100,100)
c.pack()
tk.mainloop()
```

# Más ejemplos con Canvas

- <https://github.com/dgarridouma/ejemplos-tkinter>
- Entre otras cosas veremos:
  - Crear diversas figuras
  - Controlar la posición del ratón y sus pulsaciones
  - Controlar la pulsación de teclas
  - Asignar identificadores a las formas creadas
  - Mover y borrar las formas
  - Mostrar imágenes en el canvas

# Integración con matplotlib

- Las figuras de matplotlib pueden ser integradas en tkinter
  - [https://matplotlib.org/3.1.0/gallery/user\\_interfaces/embedding\\_in\\_tk\\_sgskip.html](https://matplotlib.org/3.1.0/gallery/user_interfaces/embedding_in_tk_sgskip.html)

```
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

class Aplicacion:
    def __init__(self):
        self.ventana=tk.Tk()
        self.fig = Figure()
        self.ax = self.fig.add_subplot(111)
        self.ax.set_xlabel("X axis")
        self.ax.set_ylabel("Y axis")
        self.ax.cla() # clear axis
        self.ax.grid() # configura grid
        self.ax.set_xlim(-10, 10)
        self.ax.set_ylim(0, 100)
```

# Integración con matplotlib

- Continuación...

```
x = range(-10,11)
y = [v*v for v in x]
self.line, = self.ax.plot(x, y, marker='o', color='orange')

self.graph = FigureCanvasTkAgg(self.fig,
                               master=self.ventana)
# Agg: Anti-Grain geometry rendering engine
self.graph.get_tk_widget().pack(side="top",fill='both',
                                expand=True)

self.ventana.mainloop()

Aplicacion()
```

- <https://github.com/dgarridomarquez/ejemplos-tkinter>



## Acknowledgements / Contributions

These slides are Copyright 2020- Daniel Garrido of the University of Málaga and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Daniel Garrido, University of Málaga

... Insert new Contributors and Translators here

# Visualización de Datos

## Unidad 12

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# Visualización de Datos

- Una de las características por la que Python se ha hecho tan popular en ámbitos muy diversos es por su facilidad para **recopilar, analizar y representar datos**
- En esta presentación veremos un par de ejemplos sobre cómo **recuperar información de Internet**, procesarla y crear gráficas que aporten **algo más de valor** sobre los datos originales
- También veremos cómo podemos instalar librerías externas usando el comando **pip** y utilizaremos las librerías **requests** y **matplotlib**
  - La librería **requests** permite acceder a recursos de Internet
  - La librería **matplotlib** se utiliza para visualización de datos

# Requisitos Previos

- En primer lugar, vamos a comprobar si tenemos las librerías que necesitamos instaladas. Para ello, abre la consola Python y escribe lo siguiente:

```
>>> import requests
```

Y a continuación

```
>>> import pylab
```

- Si no están instaladas (y en una instalación limpia de Python no lo están), verás errores como los siguientes:

```
>>> import requests
```

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ModuleNotFoundError: No module named  
'requests'

```
>>> import pylab
```

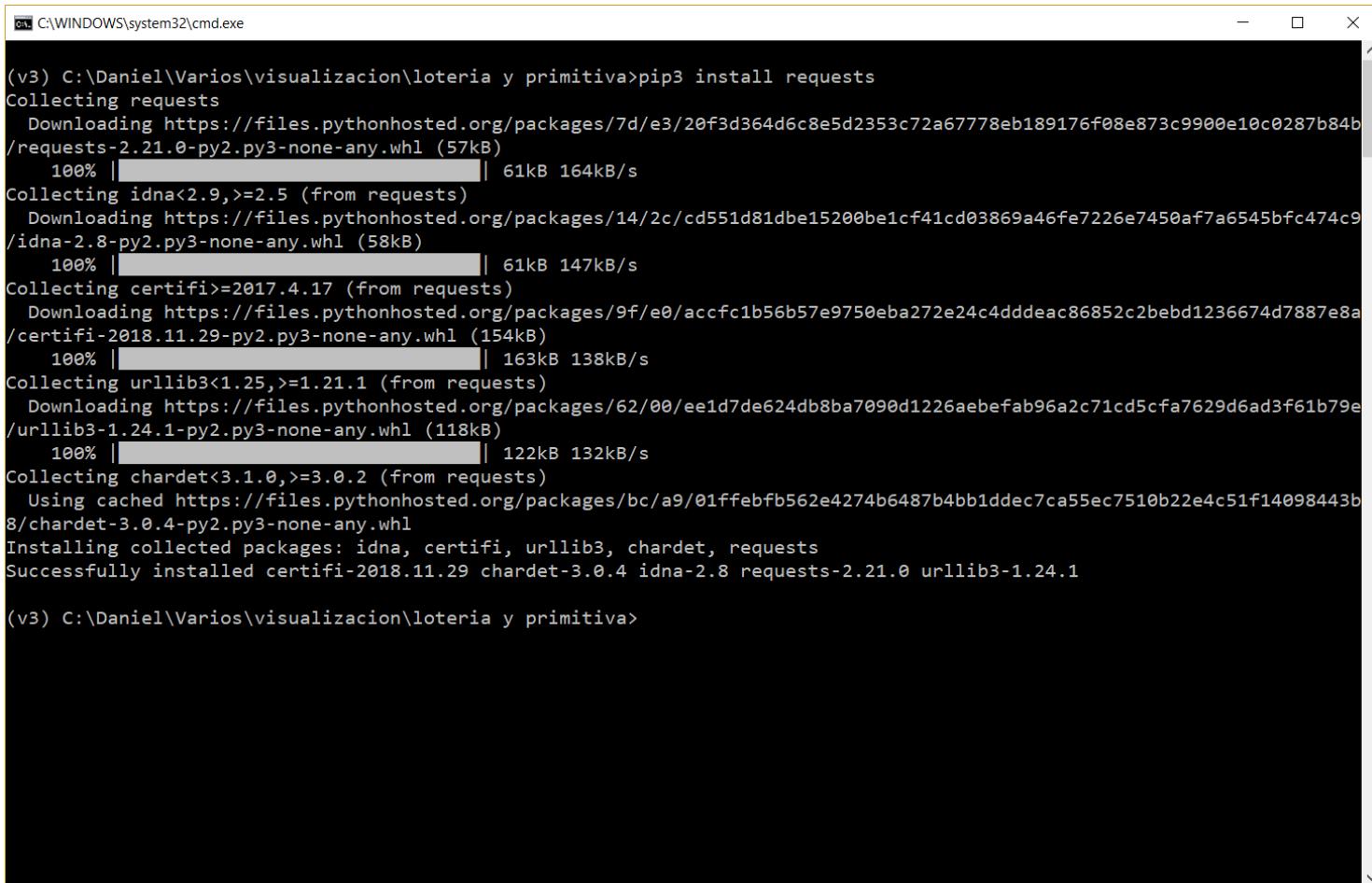
Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ModuleNotFoundError: No module named  
'pylab'

# Utilización de pip

- El comando pip permite **instalar librerías** de terceros en tu instalación python.
- Para instalar las librerías que necesitamos, ejecuta sobre tu terminal (**del Sistema Operativo**, no de Python!) lo siguiente:
  - **pip3 install requests**
  - **pip3 install matplotlib**
- Si todo va bien, deberías ver algo similar a la figura de la derecha
- pip3 es para Python 3. También deberías saber que hay otras alternativas más limpias (**virtualenv**), aunque no vamos a usarlas en estos ejemplos



```
C:\WINDOWS\system32\cmd.exe
(v3) C:\Daniel\Varios\visualizacion\loteria y primitiva>pip3 install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/7d/e3/20f3d364d6c8e5d2353c72a67778eb189176f08e873c9900e10c0287b84b
  /requests-2.21.0-py2.py3-none-any.whl (57kB)
    100% |██████████| 61kB 164kB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9
  /idna-2.8-py2.py3-none-any.whl (58kB)
    100% |██████████| 61kB 147kB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading https://files.pythonhosted.org/packages/9f/e0/accfc1b56b57e9750eba272e24c4dddeac86852c2bebd1236674d7887e8a
  /certifi-2018.11.29-py2.py3-none-any.whl (154kB)
    100% |██████████| 163kB 138kB/s
Collecting urllib3<1.25,>=1.21.1 (from requests)
  Downloading https://files.pythonhosted.org/packages/62/00/ee1d7de624db8ba7090d1226aebe fab96a2c71cd5cfa7629d6ad3f61b79e
  /urllib3-1.24.1-py2.py3-none-any.whl (118kB)
    100% |██████████| 122kB 132kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b
  /chardet-3.0.4-py2.py3-none-any.whl
Installing collected packages: idna, certifi, urllib3, chardet, requests
Successfully installed certifi-2018.11.29 chardet-3.0.4 idna-2.8 requests-2.21.0 urllib3-1.24.1

(v3) C:\Daniel\Varios\visualizacion\loteria y primitiva>
```

# Caso de uso 1: Evolución Euribor

- En este primer ejemplo, vamos a ver cómo podemos utilizar Python para estudiar la evolución del Euribor a lo largo del tiempo. Para ello vamos a utilizar datos originalmente disponibles de la siguiente web:
  - <https://www.emmi-benchmarks.eu/euribor-org/euribor-rates.html>
  - Tendrás el código definitivo al final

The screenshot shows the homepage of the EMMI website. At the top, there is a navigation bar with the EMMI logo and links for EURIBOR, Eonia, and Step. Below the logo, there is a search bar and a 'what's new' section. The main content area features a large image of a hand holding a globe with euro symbols on it. The page title is 'Euribor® Rates'. A message at the top states: 'In order to be compliant with new web security standards EMMI and STEP websites are now in HTTPS protocol. Should you have any issue accessing the sites or the data please do not hesitate to contact us'. Below this, another message says: 'The EMMI website only displays rates on a 24-hour delayed basis.' A detailed explanation of Euribor follows, mentioning its calculation methodology and the Euribor Code of Conduct. On the left side, there is a sidebar with links: Back To EMMI, Euribor®, History, Euribor Reform, Rates (which is highlighted in orange), Re-Fixing, Panel Banks, and Steering Committee.

# Caso de uso 1: Evolución Euribor

- Actualización 2021: Tras los últimos cambios de la web anterior. Los datos, no pueden descargarse directamente en formato CSV, por lo que usaremos un repositorio externo (la URL se indicará a continuación)
- En nuestro caso **descargaremos los datos directamente** desde nuestro programa Python
  - Esto tiene la ventaja de que si hubiera cambios (en algún otro ejemplo), no tenemos que volver a descargar los archivos

# Caso de uso 1: Evolución Euribor

- En la dirección [https://github.com/dgarridouma/python-course/blob/main/EURIBOR\\_2017.csv](https://github.com/dgarridouma/python-course/blob/main/EURIBOR_2017.csv) podéis encontrar los datos relativos a 2017:

,Jan17,Feb17,Mar17,Apr17,May17,Jun17,Jul17,Aug17,Sep17,Oct17,Nov17,Dec17,2017  
1w,-0.378,-0.379,-0.379,-0.379,-0.379,-0.379,-0.379,-0.379,-0.38,-0.379,-0.377,-0.379  
2w,-0.373,-0.372,-0.372,-0.372,-0.373,-0.373,-0.376,-0.376,-0.377,-0.377,-0.376,-0.372,-0.374  
1m,-0.371,-0.372,-0.372,-0.372,-0.373,-0.373,-0.373,-0.372,-0.372,-0.372,-0.372,-0.369,-0.372  
2m,-0.339,-0.341,-0.34,-0.34,-0.341,-0.342,-0.341,-0.34,-0.34,-0.34,-0.341,-0.339,-0.34  
3m,-0.326,-0.329,-0.329,-0.33,-0.329,-0.33,-0.33,-0.329,-0.329,-0.33,-0.329,-0.328,-0.329  
6m,-0.236,-0.241,-0.241,-0.246,-0.251,-0.267,-0.273,-0.272,-0.273,-0.274,-0.274,-0.271,-0.26  
9m,-0.152,-0.165,-0.171,-0.179,-0.179,-0.195,-0.206,-0.211,-0.218,-0.221,-0.219,-0.22,-0.195  
12m,-0.095,-0.106,-0.11,-0.119,-0.127,-0.149,-0.154,-0.156,-0.168,-0.18,-0.189,-0.19,-0.145

# Caso de uso 1: Evolución Euribor

- Los datos que podemos encontrarnos en Internet pueden ser **de muy diferentes tipos**, y en cada uno de ellos tendremos que **afrontarlo de una manera diferente**.
- En muchas ocasiones los datos se encuentran **incompletos, con errores o no se ajustan a nuestras necesidades**, por lo que tendremos que adaptarnos a lo que encontramos.
- En el caso que nos ocupa, encontramos en cada línea (si ignoramos la primera), la evolución del índice Euribor para 1 semana, 2 semanas, 1 mes, 2 meses, 3 meses, 6 meses, 9 meses y un año para cada uno de los meses del año
- Si nos damos cuenta, un archivo csv no es más que un archivo de texto donde la información se encuentra separada (en este caso) por comas ','
  - Por ejemplo, la línea:  
1w,-0.378,-0.379,-0.379,-0.379,-0.379,-0.379,-0.379,-0.379,-0.38,-0.379,-0.377,-0.379  
Nos indica que en enero de 2017 el valor del índice para 1 semana fue -0.378, para febrero -0.379 y así sucesivamente
- Con lo que hemos visto en unidades anteriores, seríamos capaces de procesar esta información para representar en una gráfica de líneas cómo ha ido evolucionando el índice

# Recuperación de la información - Euribor

- Para recuperar la información en nuestro programa vamos a usar la **librería requests**  
import requests
- En la variable url indicamos la dirección donde están nuestros datos. Para recuperarlos, tenemos que hacer una **petición get (protocolo http)**.

```
url = 'https://raw.githubusercontent.com/dgarridouma/python-course/main/EURIBOR_2017.csv'
```

```
response=requests.get(url)
```

- La respuesta queda almacenada en la variable response, y si queremos tener nuestro texto dividido en líneas, podemos hacer lo siguiente para tener una lista de strings!

```
texto=response.text.splitlines()
```

# Procesamiento de la información - Euribor

- Una vez tenemos el texto dividido en líneas, solo tenemos que ir recuperando los valores. Usaremos para ello un **bucle for**, el **método split** y **compresión de listas**!:

```
for linea in texto[1:]: # descartamos la primera línea de cabecera
    palabras_linea=linea.split(',')
    # creamos una lista de flotantes con los valores de la linea
    # descartamos el primer y ultimo valor de la lista por contener
    # el nombre del indicador y la media del año respectivamente
    valores=[float(x) for x in palabras_linea[1:len(palabras_linea)-1]]
```

Si quisiéramos verlos por pantalla, no tendríamos más que hacer:

```
print(valores)
```

Pero hemos dicho que queríamos verlos en una gráfica!

# Visualización de la información - Euribor

- La librería matplotlib, sobre la que puedes encontrar abundante información en <https://matplotlib.org/index.html> es ampliamente utilizada en Python para representar datos
- Es una librería muy extensa. En este caso de uso nos vamos a centrar en lo que nos interesa. No obstante, un primer ejemplo muy básico sería el siguiente:

```
import pylab as pl
```

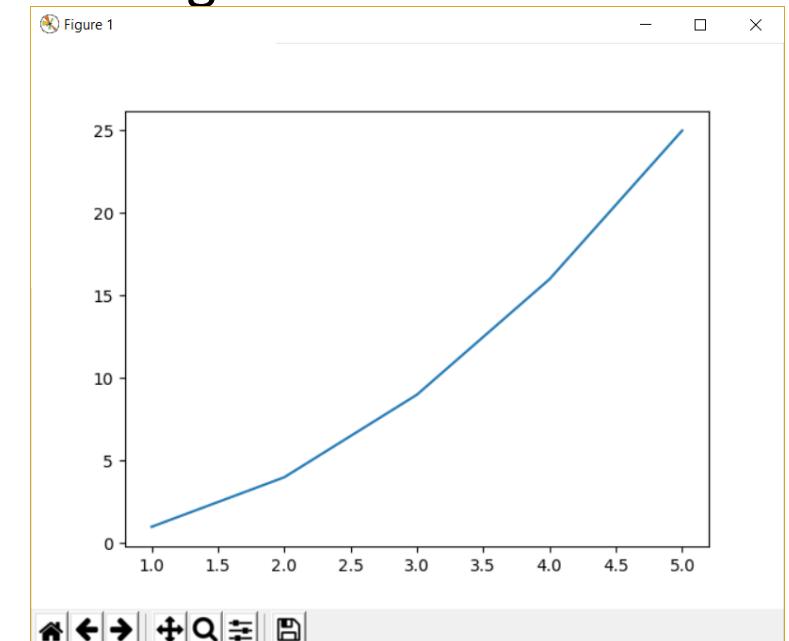
```
x = [1, 2, 3, 4, 5]
```

```
y = [1, 4, 9, 16, 25]
```

```
pl.plot(x, y)
```

```
pl.show()
```

Pruébalo antes en otro programa, y deberías ver algo así:



# Visualización de la información - Euribor

- En el caso que nos ocupa, queremos tener una serie de datos por cada indicador (semanal, mensual, etc.). Para ello bastará con hacer sucesivos "plot" en el bucle for anterior, que quedaría así:

```
for linea in texto[1:] :  
    palabras_linea=linea.split(',')  
    valores=[float(x) for x in  
palabras_linea[1:len(palabras_linea)-1]]  
    pl.plot(range(1,13),valores,  
marker='o',label= palabras_linea[0] )
```

Añadir al comienzo  
del programa:  
import pylab as pl

Hemos indicado también que queremos utilizar un marcador ('o') y que queremos poner un nombre a nuestra serie de datos (label)

# Visualización de la información - Euribor

- Nos faltan aún algunos detalles más:

```
pl.title("Evolucion EURIBOR 2017")
```

```
pl.xlabel("Mes")
```

```
pl.ylabel("Valor indicador")
```

```
pl.xticks(range(13),[x[:3] for x in list(calendar.month_name)])
```

```
pl.grid(True)
```

```
pl.legend()
```

Y finalmente:

```
pl.show()
```

Añadir al comienzo  
del programa:  
import calendar



Estamos usando compresión de listas y la  
**librería calendar** para recuperar los  
nombres del mes

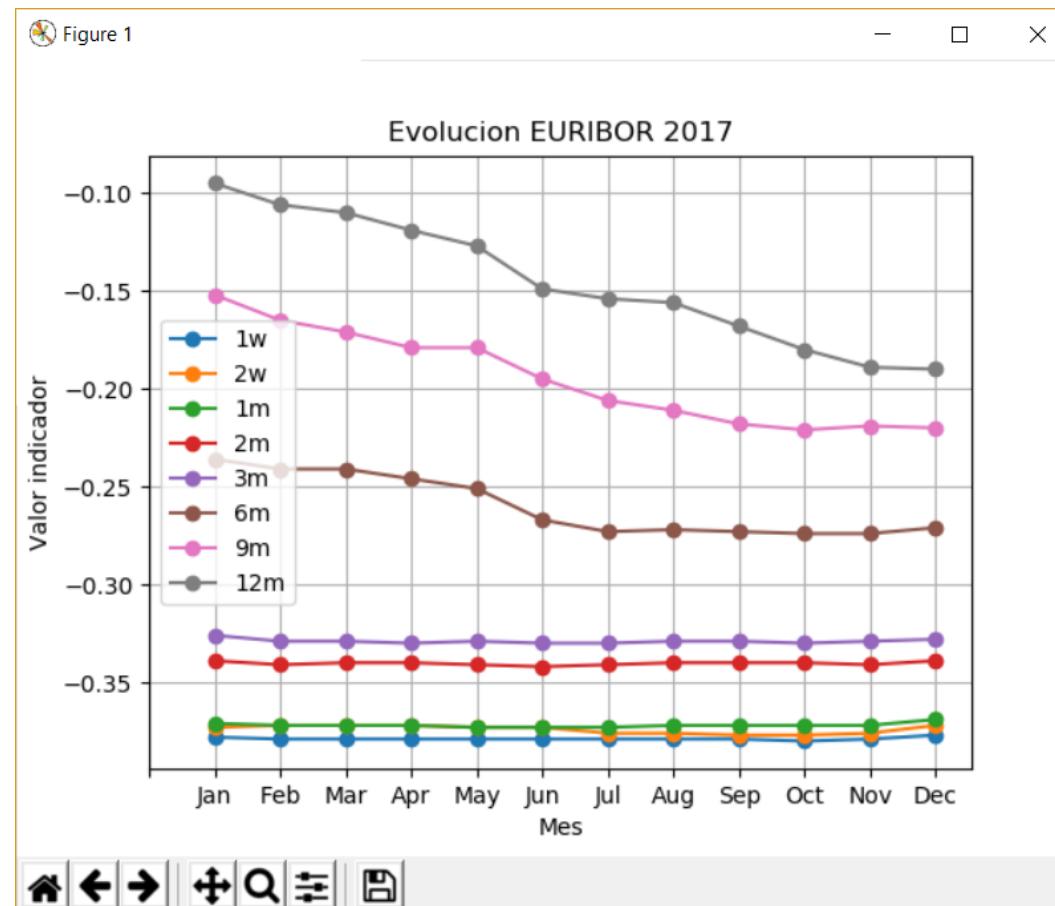
# Visualización de la información - Euribor

- Un último detalle. Si habéis probado el programa anterior, los nombres del mes (las 3 primeras letras) ¡salen en inglés!
- Para solucionarlo podemos usar la librería locale y decirle que use los nombres en español:

```
import locale  
...  
locale.setlocale(locale.LC_ALL, 'es_ES')
```

# Visualización de la información - Euribor

- Si todo ha ido bien, al ejecutar el programa deberías obtener lo siguiente:



# Código definitivo - Euribor

```
import pylab as pl
import requests
import calendar
import locale
locale.setlocale(locale.LC_ALL,'es_ES') # para obtener nombres meses
url = 'https://raw.githubusercontent.com/dgarridouma/python-course/main/EURIBOR_2017.csv'

response=requests.get(url)
texto=response.text.splitlines()
for linea in texto[1:]:
    palabras_linea=linea.split(',')

    # creamos una lista de flotantes con los valores de la linea
    # descartamos el primer y ultimo valor de la lista por contener
    # el nombre del indicador y la media del año respectivamente
    valores=[float(x) for x in palabras_linea[1:len(palabras_linea)-1]]
    pl.plot(range(1,13),valores, marker='o',label=palabras_linea[0])
```

# Código definitivo - Euribor

... (viene de la diapositiva anterior)

```
pl.title("Evolucion EURIBOR 2017")
pl.xlabel("Mes")
pl.ylabel("Valor indicador")
pl.xticks(range(13),[x[:3] for x in list(calendar.month_name)])
pl.grid(True)
pl.legend()
pl.show()
```

# Caso de uso 2: Lotería primitiva

- En este segundo ejemplo, vamos a ver algo más lúdico (lotería primitiva ;-)), analizando qué números salen con más frecuencia en los sorteos
- ¡Posiblemente no nos vamos a hacer ricos!, pero al menos veremos cómo hacer **gráficas de barras**
- En esta ocasión, hemos obtenido los datos de la siguiente dirección:

<http://www.lotoideas.com/primitiva-resultados-historicos-de-todos-los-sorteos/>



Histórico de los resultados de La Primitiva  
Actualizado frecuentemente

Resultados históricos de todos los sorteos de La Primitiva desde su inicio.  
Totalmente gratis y listos para su descarga en varios formatos.

Lotoideas.com - Histórico de Resultados - Primitiva

FECHA	COMBINACIÓN GANADORA	COMP.	R.	JOKER
22/12/2018	08 11 32 41 44 45	38	3	1092798
20/12/2018	08 24 25 31 43 46	12	8	7952896
15/12/2018	06 09 10 24 26 32	25	5	3459017
13/12/2018	02 10 41 42 44 45	33	3	3661223
8/12/2018	08 09 12 19 24 41	39	4	9759406
6/12/2018	19 20 32 37 42 49	46	0	6931344
1/12/2018	02 12 19 39 44 45	22	9	1687972
29/11/2018	08 18 26 40 44 49	04	7	3594429
24/11/2018	02 10 14 19 24 35	32	9	7216216
22/11/2018	09 22 39 43 45 47	04	4	0336382
17/11/2018	02 14 23 29 30 39	37	2	5045646
15/11/2018	06 26 36 39 40 45	12	8	5892581

# Formato de los datos - Primitiva

- En esta ocasión, los datos son suministrados nuevamente como archivos csv pero en dos diferentes, con datos desde 1985 a 2012 y desde 2013 al 2018 (o posteriores según cuándo hagas el curso)
- A continuación, veamos un fragmento de estos archivos:

FECHA,COMBINACIÓN GANADORA,,,,,,COMP.,R.,JOKER

8/12/2018,08,09,12,19,24,41,39,4,9759406

6/12/2018,19,20,32,37,42,49,46,0,6931344

1/12/2018,02,12,19,39,44,45,22,9,1687972

Los datos vienen ordenados por fecha, a continuación la combinación ganadora (6 números) y luego el complementario, reintegro y joker. Nos vamos a centrar únicamente en los **6 números de la combinación ganadora**

# Imports y obtención de los datos - Primitiva

- Nuevamente usaremos requests y pylab:

```
import requests  
import pylab as pl
```

- Como hemos visto, los datos están en 2 archivos diferentes, así que tendremos que hacer 2 peticiones get y unir el resultando:

```
url =  
'https://docs.google.com/spreadsheet/pub?key=0AhqMeY8ZOrNKdFJnYkMzX3JzN2lkRl  
ZZTXV2aldqS1E&single=true&gid=1&output=csv'  
  
response=requests.get(url)  
lineas=response.text.splitlines()  
  
url='https://docs.google.com/spreadsheet/pub?key=0AhqMeY8ZOrNKdFJnYkMzX3JzN2  
lkRlZZTXV2aldqS1E&single=true&gid=0&output=csv'  
  
response=requests.get(url)  
lineas+=response.text.splitlines()
```

# Procesamiento de los datos - Primitiva

- El procesamiento de los datos es similar a ejemplos realizados en unidades anteriores. Vamos a tener un **diccionario** donde iremos contabilizando **cuántas veces aparece cada número** en la combinación ganadora

```
for l in lineas[1:]:
    lsorteo=l.split(',')
    for num in lsorteo[1:6]:
        sdict[num]=sdict.get(num, 0)+1
```

sdict contiene el número de repeticiones para cada número

# Procesamiento de los datos - Primitiva

- Llegados a este punto, ya hemos construido el diccionario, pero en realidad, queremos tener "algo" ordenado por el número de apariciones de cada número. Vamos a utilizar **compresión de listas** y **sorted** para hacer esto.

```
tmp = list ()  
for k, v in sdic.items () :  
    tmp.append( (v, k) )
```

Creamos una lista auxiliar con **tuplas** con el valor (el número de repeticiones) como primer componente

```
top6=sorted(tmp,reverse=True) [:6]  
print (top6)
```

top6 almacena los 6 números que han aparecido más veces en los sorteos

# Procesamiento de los datos - Primitiva

- A continuación, preparamos los datos para la gráfica: etiquetas de las barras y valores de las repeticiones usando las tuplas anteriores:

```
top6_nums=[x[1] for x in top6]
```

```
print(top6_nums)
```

```
top6_repeticiones=[int(x[0]) for x in top6]
```

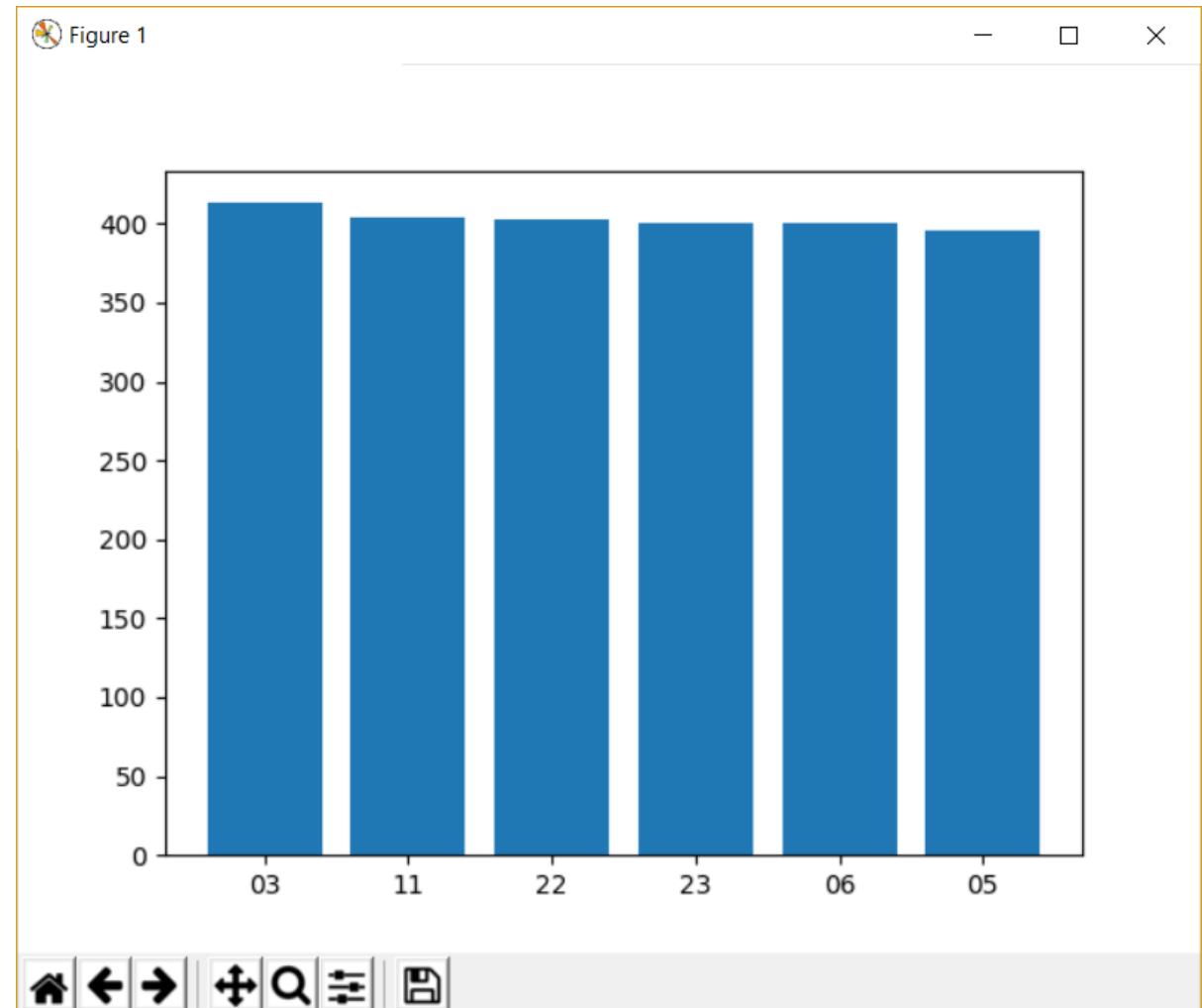
```
print(top6_repeticiones)
```

# Procesamiento de los datos - Primitiva

- Finalmente, creamos la gráfica y la mostramos:

```
pl.bar(range(6), top6_repeticiones,  
       align='center')  
  
puntos=range(6)  
  
pl.xticks(puntos, top6_nums)  
  
pl.show()
```

A la derecha podemos ver los 6 números que más veces aparecen, así como el número de repeticiones



# Código definitivo – Lotería Primitiva

```
import requests
import pylab as pl

url =
'https://docs.google.com/spreadsheet/pub?key=0AhqMeY8ZOrNKdFJnYkMzX3JzN21kRlZZTXV2aldqS1E&
single=true&gid=1&output=csv'
response=requests.get(url)
lineas=response.text.splitlines()

url='https://docs.google.com/spreadsheet/pub?key=0AhqMeY8ZOrNKdFJnYkMzX3JzN21kRlZZTXV2aldq
S1E&single=true&gid=0&output=csv'
response=requests.get(url)
lineas+=response.text.splitlines()

sdict=dict()
for l in lineas[1:]:
    lsorteo=l.split(',')
    for num in lsorteo[1:6]:
        sdict[num]=sdict.get(num,0)+1
```

# Código definitivo – Lotería Primitiva

```
... (viene de la diapositiva anterior)

tmp = list()
for k, v in sdict.items() :
    tmp.append( (v, k) )

top6=sorted(tmp,reverse=True) [:6]
print(top6)

top6_nums=[x[1] for x in top6]
print(top6_nums)

top6_repeticiones=[int(x[0]) for x in top6]
print(top6_repeticiones)

pl.bar(range(6),top6_repeticiones,align='center')
puntos=range(6)
pl.xticks(puntos,top6_nums)
pl.show()
```

# Resumen

- Instalación de librerías (pip)
- Tratamiento de datos de Internet
- Librería requests
- Peticiones get
- Archivos csv
- Matplotlib
- Gráficas de líneas y barras



## Acknowledgements / Contributions

These slides are Copyright 2018- Daniel Garrido of the University of Málaga and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Daniel Garrido, University of Málaga

... Insert new Contributors and Translators here

# Ejercicios Adicionales Tema 1

## El adivino

---

*¿Por qué pagar a un adivino cuando puedes programar tu propia fortuna?*

- Almacena la siguiente información en variables: número de hijos, nombre de tu pareja, ciudad, trabajo deseado.
- Mostrar por la consola algo así como: "Serás X en Y, y estarás casado con Z con quien tendrás N hijos."

## Calculador de Edad

---

*¿Quieres saber cuántos años tendrás? ¡Calculémoslo!*

- Almacena tu año de nacimiento en una variable.
- Almacena un año futuro en otra variable.
- Calcula las 2 posibles edades que tendrás ese año.  
Por ejemplo, si naciste en 1988, entonces en 2026 tendrás 37 o 38, dependiendo del mes que sea en 2026.
- Mostrar por la consola algo así como: "Tendré NN o NN en YYYY", sustituyendo los valores.

## La calculadora de suministros

---

*¿Nunca te has preguntado cuántos suministros de tu comida o bebida favorita necesitarás? ¡No te lo preguntes más!*

- Almacena tu edad en una variable.
- Almacena una edad máxima en otra variable.
- Indica una cantidad estimada por día.
- Calcula cuántas beberás o comerás hasta llegar a la edad máxima (ignora años bisiestos)
- Mostrar por la consola algo así como: "Necesitarás NN para llegar hasta la edad de X".

## Calculador de Áreas

---

Calcula las propiedades de un círculo, usando las definiciones de [aquí](#).

- Almacena un radio en una variable.
- Calcula la circunferencia basándose en el radio y muestra un mensaje similar "La circunferencia es NN".
- Calcula el área basándose en el radio, y muestra "El área es NN".

## Conversor de Temperatura

---

¡Hace frío! Vamos a hacer un conversor basado en [estos pasos](#).

- Almacena una temperatura en celsius en una variable.
- Convierte a fahrenheit y muestra "NN°C es NN°F".
- Ahora almacena una temperatura en fahrenheit.
- Convierte a celsius y muestra "NN°F es NN°C."

## ¿Cuánto tardas?

---

Escribe un programa que lea desde teclado un número de minutos e indica a cuántos días, horas y minutos corresponden. Por ejemplo, 4565 minutos corresponden a 3 días, 4 horas y 5 minutos.

## ¿Está muy lejos?

---

Escribe un programa que lea desde teclado las coordenadas de dos puntos (x,y) en el plano 2D y calcula la distancia entre esos dos puntos. Por ejemplo, los puntos (3, 5) y (6, 1) están a una distancia de 5.

Si tenemos 2 puntos A(x1,y1) y B(x2,y2), la fórmula de la distancia es:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

# Ejercicios Adicionales Tema 2

---

## ¿Qué número es mayor?

---

- Escribe una función llamada `greaterNum` que:
  - toma 2 argumentos, ambos números.
  - retorna el número que sea mayor.
- Llamar la función varias veces para asegurarse de que funciona y mostrar en el terminal el resultado (p.ej. "El número mayor entre 5 y 10 es 10.").

## El Traductor Mundial

---

- Escribe una función llamada `helloWorld` que:
  - toma 1 argumento, un código de idioma (p.ej. "es", "de", "en")
  - retorna "Hola, Mundo" para el idioma dado (al menos para 3). El valor por defecto debería ser en inglés.
- Llamar a la función con varios idiomas para ver que funciona.

## ¿Cuántos impares hay?

---

- Escribe un programa que lea de teclado 4 números e informe si, al menos, dos de ellos son impares

## El Calificador

---

- Escribe una función llamada `assignGrade` que:
  - toma 1 argumento, una puntuación de 0 a 100.
  - retorna un grado según la puntuación, "A", "B", "C", "D", o "F".
- Llamar a la función con diferentes puntuaciones para comprobar que funciona.

## El Calificador 2

---

- Comprueba los resultados de la función `assignGrade` del ejercicio anterior mostrando todos los valores desde 60 a 100. Por ejemplo "Para 88, tienes una B. Para 89, tienes una B. Para 90, tienes una A. Para 91, tienes una A.", etc.

## El Pluralizador (inglés)

---

- Escribe una función llamada `pluralize` que:
  - toma 2 argumentos, un nombre y un número.
  - retorna el número y la forma en plural (si aplica), como "5 cats" o "1 dog".
- Llamar a la función con diferentes cantidades para ver que funciona
- Bonus: Manejar algunos nombres colectivos como "sheep" o "geese".

## Par/Impar

---

- Escribe un bucle que itere de 0 a 20. Para cada iteración, se comprobará si el número es par o impar y se mostrará un mensaje por pantalla (p.ej. "2 es par").

## Tablas de Multiplicar

---

- Escribe un bucle que itere de 0 a 10. Para cada iteración del bucle, se multiplicará el número por 9 y se mostrará el resultado (p.ej. "2 \* 9 = 18").
- **Bonus:** Usar un bucle anidado para mostrar las tablas de 1 a 10 (100 resultados en total).

## ¿Este año es bisiesto?

---

- Escribe un programa que lea desde teclado un año e indique si el año es bisiesto o no. Los años bisiestos son aquellos que son múltiplos de 4 pero no de 100, con la excepción de los múltiplos de 400 que sí son años bisiestos.

Los siguientes ejercicios están basados en los ejercicios adicionales del tema 1, por lo que se pueden tomar sus soluciones como puntos de partida.

## El Adivino

---

*¿Por qué pagar a un adivino cuando puedes programar tu propia fortuna?*

- Escribe una función llamada `tellFortune` que:
  - toma 4 argumentos: número de hijos, nombre de la pareja, ciudad y trabajo deseado.
  - muestra tu futuro a la consola con mensajes como: "Serás X en Y, y estarás casado con Z con quien tendrás N hijos."

Mejor aún, que la función retorne el mensaje.

- Llama a la función varias veces con diferentes argumentos.

## El Calculador de Edad de Mascotas

Sabes la edad de tu perro en años humanos, pero ¿y en años perrunos? ¡Hagamos el cálculo!

- Escribe una función llamada `calculateDogAge` que:
  - toma 1 argumento: la edad del perro.
  - calcula la edad del perro basada en la conversión de 1 año humano equivale a 7 perrunos.
  - muestra el resultado por consola (o retórnalo) así: "Tu perro tiene NN años perrunos"
- Llama a la función varias veces con diferentes argumentos.
- **Bonus:** Suma un argumento adicional a la función con el ratio de conversión de humano a perro.

## La calculadora de suministros

---

*¿Nunca te has preguntado cuántos suministros de tu comida o bebida favorita necesitarás? ¡No te lo preguntes más!*

- Escribe una función llamada `calculateSupply` que:
  - toma 2 argumentos: edad, cantidad por día.
  - calcula la cantidad consumida para el resto de tu vida (basada en una constante de edad máxima).
  - muestra (o retorna) el resultado a la consola: "Necesitarás NN para llegar hasta la edad de X"
- Llama a la función varias veces con diferentes argumentos.
- **Bonus:** Acepta números flotantes para la cantidad por día, y redondea el resultado.

# Calculador de Áreas

---

Calcula las propiedades de un círculo, usando las definiciones de [aquí](#).

Crea una función llamada `calcCircumference`:

- Pasa el radio a la función.
- Calcula la circunferencia basada en el radio y muestra "La circunferencia es NN".

Crea una función llamada `calcArea`:

- Pasa el radio a la función.
- Calcula el área basada en el radio, y muestra "El área es NN".

# Conversor de Temperatura

---

¡Hace frío! Vamos a hacer un convesor basado en [estos pasos](#).

Crea una función llamada `celsiusToFahrenheit`:

- Recibe una temperatura celsius.
- La convierte a fahrenheit. Debes mostrar "NN°C es NN°F".

Crea una función llamada `fahrenheitToCelsius`:

- Recibe una temperatura fahrenheit.
- La convierte a celsius. Debes mostrar "NN°F es NN°C."

# Ejercicios Adicionales Tema 3

## MixUp

Crea una función llamada `mixUp`. Toma dos strings y retorna la concatenación de los dos (separados por un espacio) e intercambiando los dos primeros caracteres de cada. Se puede asumir que los strings tienen al menos 2 caracteres de largo. Por ejemplo:

```
mixUp('mix', 'pod'): 'pox mid'  
mixUp('dog', 'dinner'): 'dig donner'
```

## FixStart

Crea una función llamada `fixStart`. Toma un argumento, un string, y retorna una versión donde todas las ocurrencias del primer carácter han sido reemplazadas con '\*', excepto en la primera posición. Se puede asumir que el string tiene al menos un carácter. Por ejemplo:

```
fixStart('babble'): 'ba**le'
```

## Verbing

Crear una función llamada `verbing`. Toma un argumento, un string. Si la longitud es al menos 3, debería sumar 'ing' al final, a menos que termine en 'ing', en cuyo caso suma 'ly' en su lugar. Si la longitud del string es menor que 3, permanece sin cambios. Por ejemplo:

```
verbing('swim'): 'swimming'  
verbing('swimming'): 'swimmingly'  
verbing('go'): 'go'
```

## Not Bad

---

- Crear una función llamada `notBad` que toma un argumento, un string.
- Encuentra la primera ocurrencia de los substrings 'not' y 'bad'.
- Si 'bad' sigue a 'not', entonces debería sustituir el substring 'not'...'bad' completo con 'good' y retornar el resultado.
- En caso contrario, retorna el string original.

Por ejemplo:

```
notBad('This dinner is not that bad!'): 'This dinner is good!'
notBad('This movie is not so bad!'): 'This movie is good!'
notBad('This dinner is bad!'): 'This dinner is bad!'
```

## osrevnl - Inverso

---

Escribe una función que recibe una cadena de caracteres y devuelve como resultado todas las palabras de la cadena pero escritas al revés. Por ejemplo: "hola mundo" devolvería "aloh odnum".

## Palíndromo

---

Escribe una función que determine si una cadena de caracteres es un palíndromo o no. Un palíndromo es una cadena de caracteres que puede leerse igual de izquierda a derecha que de derecha a izquierda. Por ejemplo: "Dábale arroz a la zorra el abad" (ignorar acentos y mayúsculas).

## Tu información

---

Escribe un programa que pida al usuario su nombre, edad y dirección de e-mail y la escriba en un archivo con el siguiente formato:

Nombre: Gustavo.  
Edad: 37 años.  
Email: [gustavo37@gmail.com](mailto:gustavo37@gmail.com)

## Solo una línea

---

Escribe un programa que lea un archivo y transforme todos los caracteres de fin de línea '\n' en espacios. El resultado puede mostrarse por la consola o escribirse en otro archivo.

## Uniendo archivos

Escribe un programa que pida dos nombres de archivos y los una en un único fichero cuyo nombre también se pedirá al usuario.

## La palabra más larga

Escribir un programa que pida el nombre de un archivo, lo abra e indique cuál es la palabra más larga del mismo y su longitud. En caso de no existir el archivo, debe mostrar un mensaje por consola informando de ello.

## Tabla de Multiplicar en Archivos

---

Escribir una función que pida un número entero entre 1 y 10 y guarde en un archivo con el nombre tabla-n.txt la tabla de multiplicar de ese número, donde n es el número introducido.

Después, hacer otra función que pida un número entero entre 1 y 10 y lea el archivo con el nombre tabla-n.txt, donde n es el número introducido y lo muestre por la consola.

Escribir una tercera función que pida dos números n y m entre 1 y 10, lea el archivo tabla-n.txt con la tabla de multiplicar de ese número, y muestre por consola la línea m del archivo. Si el fichero no existe debe mostrar un mensaje por consola informando de ello.

## Frankenstein

---

Descargar del siguiente enlace <https://www.gutenberg.org/files/84/84-0.txt> el libro de Frankenstein y escribir un programa Python que indique cuántas veces aparece la palabra "monster".

## Un elefante se balanceaba...

---

Para la famosa nana:

"Un elefante se balanceaba sobre la tela de una araña, como veía que no se caía, fue y llamó a otro elefante.

Dos elefantes se balanceaban sobre la tela de una araña, como veían que no se caían, fueron a llamar a otro elefante.

..."

Escribir un programa Python que pida el número de elefantes y escriba la canción en el fichero elefantes.txt. Ojo, para el primer elefante la letra es distinta.

Bonus: No quedarse dormido al realizarlo.

# Ejercicios Adicionales Tema 4

## Tus mejores elecciones

---

- Crea una lista para almacenar tus preferencias (colores, presidentes, lo que sea)
- Para cada elección, muestra en la pantalla un mensaje como: "My #1 choice is blue."
- **Bonus:** Cambiar el mensaje para que muestre "My 1st choice, "My 2nd choice", "My 3rd choice", eligiendo el sufijo adecuado (en inglés) según el número.

## Rellenar lista

---

Escribe una función que cree una lista con un valor dado y el número de elementos a crear (repetidos).

Ejemplo de uso:

```
valueToFill = 'a'  
lista = fill(valueToFill, 3) # ['a', 'a', 'a']
```

## Invertir lista

---

Escribe una función que invierta los elementos de una lista. El propósito del ejercicio es simplemente practicar el uso de listas. No usar la función reverse.

Ejemplo de uso:

```
mireverse([1,2,3]) # [3, 2, 1]
```

## Compactar lista

---

Escribe una función que “compacte” los elementos de una lista eliminando todos los elementos “innecesarios” (que evalúan a falso):

Ejemplo de uso:

```
compact([1,3,False,4,5,0]) // [1, 3, 4, 5]
```

## Listas sin duplicados

---

Escribir una función que devuelva una lista sin elementos duplicados.

Ejemplo de uso:

```
unique([1,2,1,2,2,2,2,3,4,5,3,3,5]); # [1, 2, 3, 4, 5]
```

---

## La Ficha para Recetas

---

*¡Nunca olvides otra receta!*

- Crea un diccionario para almacenar información de tu receta favorita. Debería tener propiedades para **título**, **raciones** e **ingredientes** (una lista de strings).
- Mostrar en líneas separadas, la información de la receta. Por ejemplo:
  - Mole
  - Raciones: 2
  - Ingredientes:
    - canela
    - comino
    - cacao

---

## La Lista de Lectura

---

*¡Una lista de qué libros has leído y cuales te gustaría leer!*

- Crea una lista de diccionarios, donde cada diccionario describe un libro y tiene propiedades para el **título**, **autor** y **leído** (indicando si ya se ha leído).
- Itera a través de la lista de libros. Para cada libro, mostrar el título y autor así: "El Hobbit por J.R.R. Tolkien".
- Ahora usa un if/else para indicar si el libro se ha leído o no. Si se ha leído, mostrar algo así como: '[Leído] El Hobbit por J.R.R. Tolkien', y si no, '[Pendiente de leer] El Señor de los Anillos por J.R.R. Tolkien.'

---

## La Base de Datos de Películas

*¡Como IMDB, pero mucho más pequeña!*

- Crea un diccionario para almacenar información sobre tu película favorita: **título**, **duración** y **actores** (una lista de strings).
- Crea una función para mostrar la información de la película recibiendo un diccionario de tipo película. Algo así como: "Puff the Magic Dragon dura 30 minutos. Actores: Puff, Jackie, Living Sneezes."

## Contando repeticiones

---

- Escribir una función que a partir de una lista devuelva un diccionario que incluya cuántas veces aparece cada elemento en una lista.
- Por ejemplo:

```
sample_list = [11, 45, 8, 11, 23, 45, 23, 45, 89]
Printing count of each item {11: 2, 45: 3, 8: 1, 23: 2, 89: 1}
```

- Posteriormente, crea una tupla a partir de la lista sin duplicados.

## Guardando notas

---

- Escribir un programa en Python que permita guardar las notas obtenidas por los estudiantes. Para un mismo estudiante se pueden tener varias notas.
- En un diccionario, las claves serán los nombres de los estudiantes y los valores serán listas con las notas de cada estudiante. El programa pedirá por teclado repetidamente nombres de estudiantes y notas hasta que escriba 'fin'. Tras esto, se mostrará un listado con las notas de cada estudiante.

## Los meses del año

---

Escribir un programa en Python que almacene en una tupla los nombres de los meses del año y en otra tupla cuántos días tiene cada mes. Después, se pedirá al usuario que introduzca un número de mes y el programa responderá con un mensaje del tipo "Enero tiene 31 días". Si el mes es incorrecto, se mostrará un mensaje de error. El programa debe terminar cuando se introduzca el valor 0.

Bonus: escribir una función que recibe el número de mes del año y retorna una tupla con el nombre del mes y el número de días de ese mes.

## Sumando tuplas y diccionarios

---

- Escribe un programa que calcule la suma de los elementos de una tupla suponiendo que todos ellos son números válidos.
- Realizar lo mismo para los valores contenidos en un diccionario.

## Calculando precios

---

Escribe un programa que guarde en un diccionario los precios de ciertos productos mediante referencias. Después el programa pregunta al usuario una referencia y el número de unidades y si el producto existe se muestra el precio total. Si el producto no existe se muestra un mensaje indicativo.

## La Caja Registradora

---

Escribe una función llamada `cashRegister` que toma un diccionario lista de la compra. El diccionario contiene nombres de productos y precios (`itemName: itemPrice`). La función debería devolver el precio total de la lista de la compra.

Ejemplo

```
# Entrada
cartForParty = {
    banana: 1.25,
    handkerchief: 0.99,
    Tshirt: 25.01,
    apple: 0.60,
    nalgene: 10.34,
    proteinShake: 22.36
}

# Salida
print(cashRegister(cartForParty)) # 60.55
```