
python™
riding the snake

# Ficha Técnica


**Paradigma**: multiparadigma: orientado a objetos, imperativo, funcional
**Apareció en**: 1991
**Diseñado por**: Guido van Rossum
**Última versión**: 3.1 (27 de junio de 2009)
**Tipo de dato**: fuertemente tipado, dinámico
**Implementaciones**: CPython, Jython, IronPython, PyPy
**Dialectos**: Stackless Python, RPython
**Influido por**: ABC, Tcl, Perl, Modula-3, Smalltalk, ALGOL 68, C, Haskell, Icon, Lisp, Java
**Ha influido**: Ruby, Boo, Groovy, Cobra, D
**Sistema operativo**: Multiplataforma
**Licencia de software**: Python Software Foundation License
**Web**: http://www.python.org/

# ¿Por qué Python?


*Calidad de Software:*

El código de Python fué diseñado para ser legible, y por ende reusable y manipulable. Aunque nunca haya visto ni una sola línea de código de Python en su vida, es muy probable que pueda leer un código relativamente sencillo y poder descifrar qué hace. Tiene soporte para POO.

*Productividad en el desarrollo:*

Los programas en Python en general son cortos ya que requieren menos sentencias que los lenguajes más populares como C++ o Java. Los programas en Python corren inmediatamente, no necesitan ser compilados.

*Portabilidad del Programa:*

La mayoría de los programas pueden correr en varias plataformas sin necesidad de ser modificados. Para cualquier entorno se ofrencen un set de librerías standard que permiten la manipulación de interfaces gráficas, bases de datos, sistemas basados en web y más...

# ¿Por qué Python?


*Librerías:*

Python trae una gran colección de librerías con funcionalidad portable conocida como la *standard library.* Ésta librería soporta un set importante de tareas de programación a nivel de aplicación, desde marching de patrones de texto hasta scripting de redes. Además Python puede ser extensible en el sentido que pueden agregarse innumerables colecciones de librerías externas, entre ellas las NumPy y astroLib que veremos más adelante.

*Integración de Componentes:*

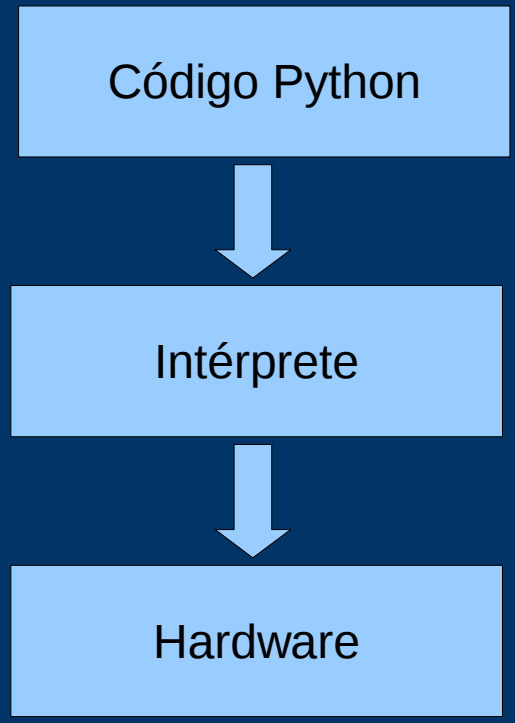
Python no es una herramienta “solitaria”, puede interactuar con librerías de C, C++.

*¡Divertido!:*

Debido a la facilidad del uso y a las herramientas provistas “out of the box”, pueden hacer que programar en Python sea divertido, ¡cualidad que impacta directamente en la productividad!.

# Ejecución de Programas

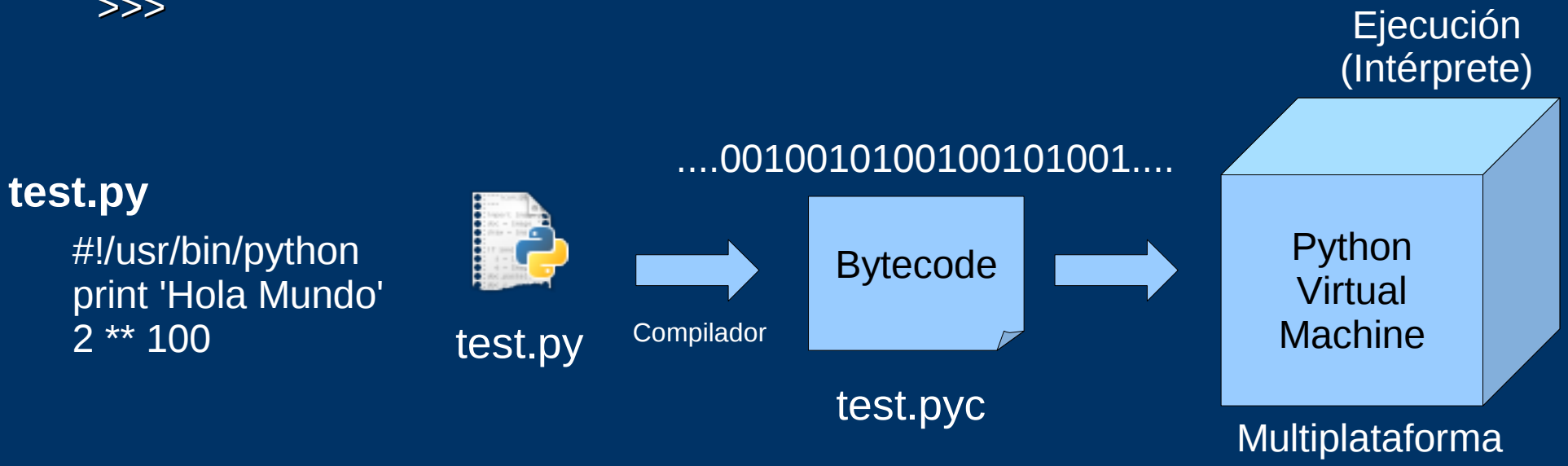
## Intérprete




```
[fernandoph@shenxiu ~]$ python
Python 2.5.2 (r252:60911, Jan  4 2009, 17:40:26)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# Ejecución de Programas

```
[fernandoph@shenxiu ~]$ python
Python 2.5.2 (r252:60911, Jan  4 2009, 17:40:26)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hola Mundo'
Hola Mundo
>>> 2 ** 100
1267650600228229401496703205376L
>>>
```

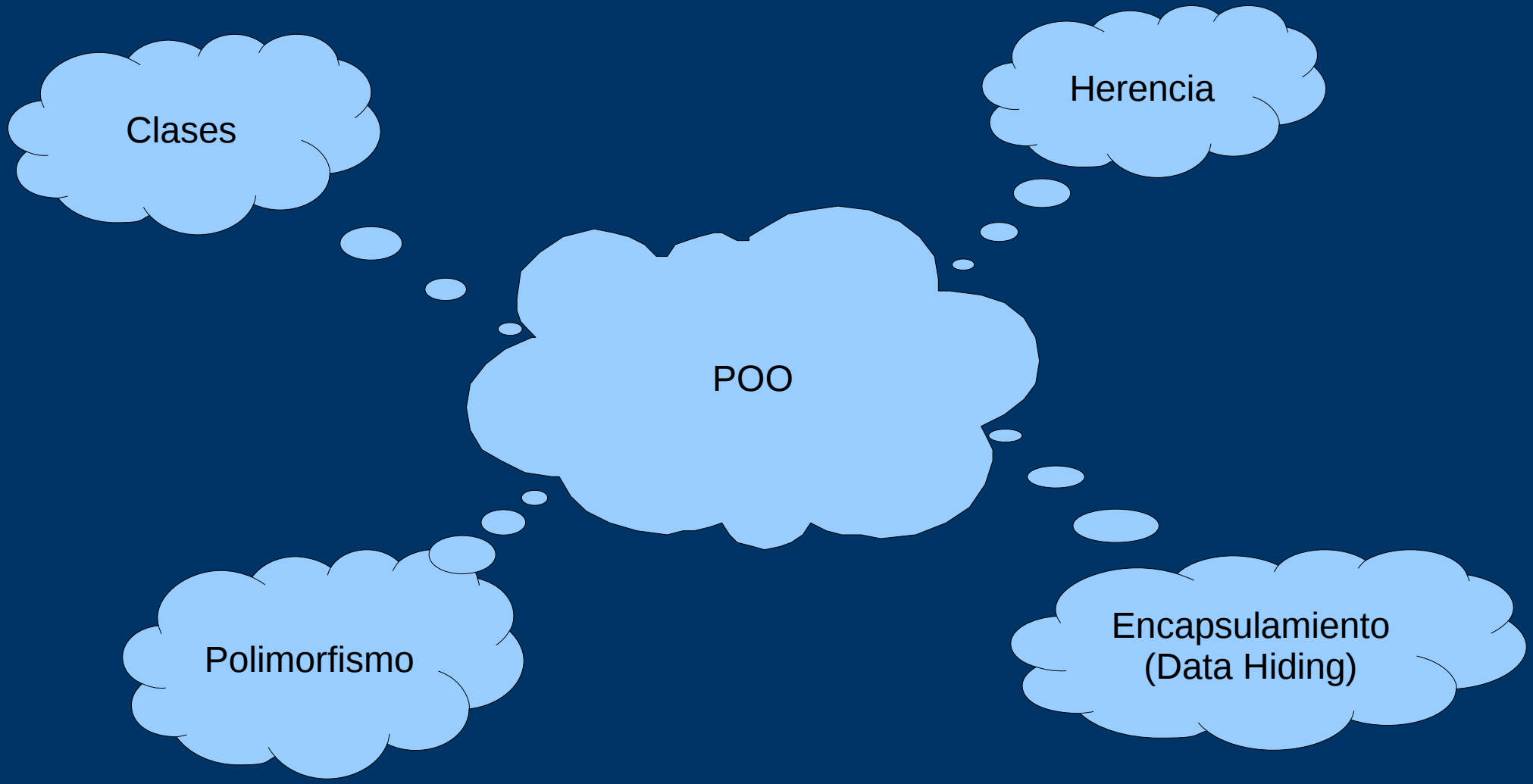



**¿Por qué compilar el código?**

Los programas compilados se ejecutan más rápido que los interpretados

# Programando en Python

## Programación Orientada a Objetos (POO)

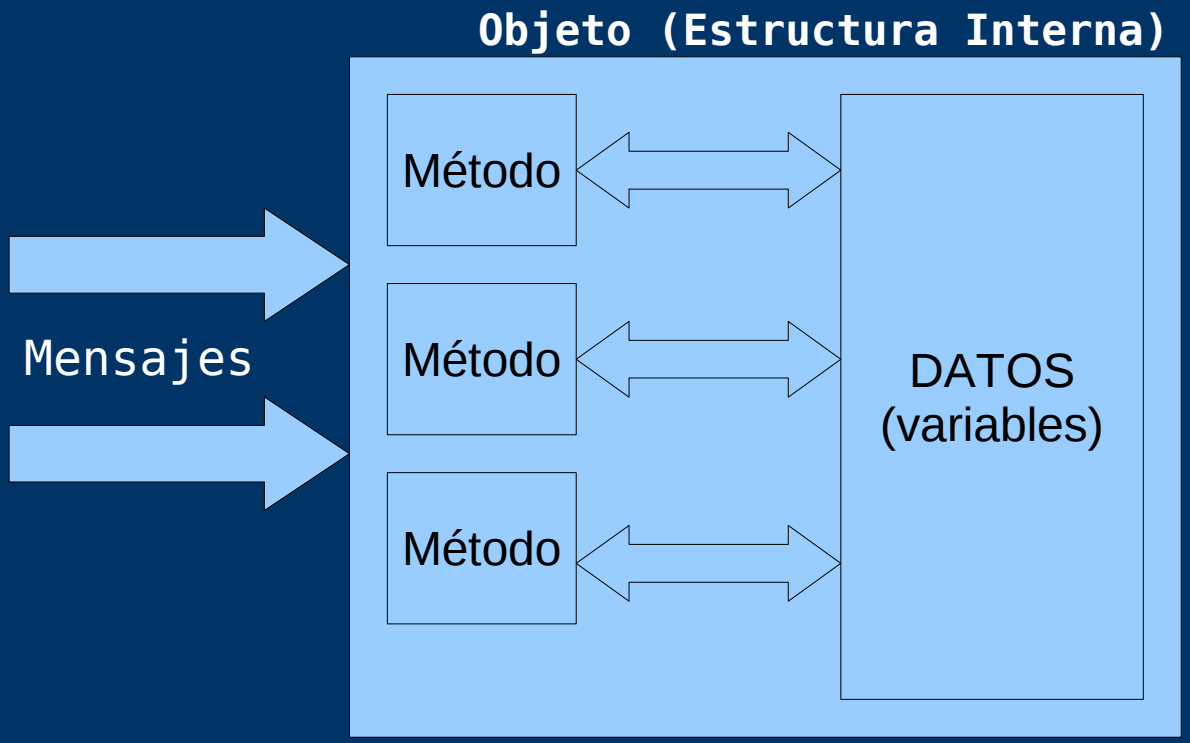


# Programando en Python

## Programación Orientada a Objetos (POO)

**Programa**: Colección de Objetos que colaboran enviándose mensajes

**Objeto**: Es una entidad computacional que se caracteriza por su capacidad de responder mensajes. Éstos mensajes definen su comportamiento. Además de su comportamiento, tienen una estructura interna que en general se materializa como variables, las cuales caracterizan al objeto. Éstas variables están encapsuladas (son accesibles **solo** por el objeto al cual pertenecen)




**TODO** está incluído en los métodos.

# Programando en Python

## Programación Orientada a Objetos (POO)

**Clase:** Es una descripción abstracta de las características que poseen los objetos que pertenecen a ésa clase. En particular en una clase describimos qué mensajes van a responder los objetos, como va a ser la estructura interna, y el código de los métodos para responder ésos mensajes

Un Objeto es un **INSTANCIA** de una clase

Una clase es una **FÁBRICA** de objetos

**Information Hiding**: NUNCA es importante cómo está construído un objeto, sólo importa QUÉ HACE.
Ésto permite modificaciones importantes en el objeto sin que los clientes se enteren (si no cambió la interfaz).

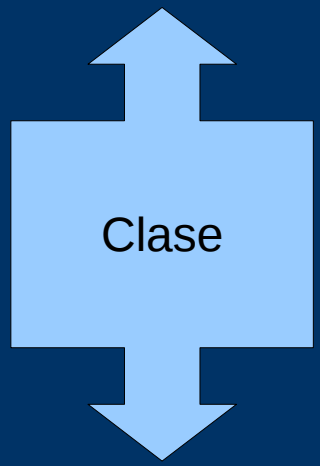
En el momento de desarrollo, es productivo pensar en los objetos primero como “qué hace” y luego “cómo lo hace”.

# Programando en Python

## Programación Orientada a Objetos (POO)

**Polimorfismo:** Se da cuando objetos de clases distintas pueden responder al mismo mensaje, aunque cada uno con su propio comportamiento.

**Concreta**: es posible crear objetos a partir de ésa clase

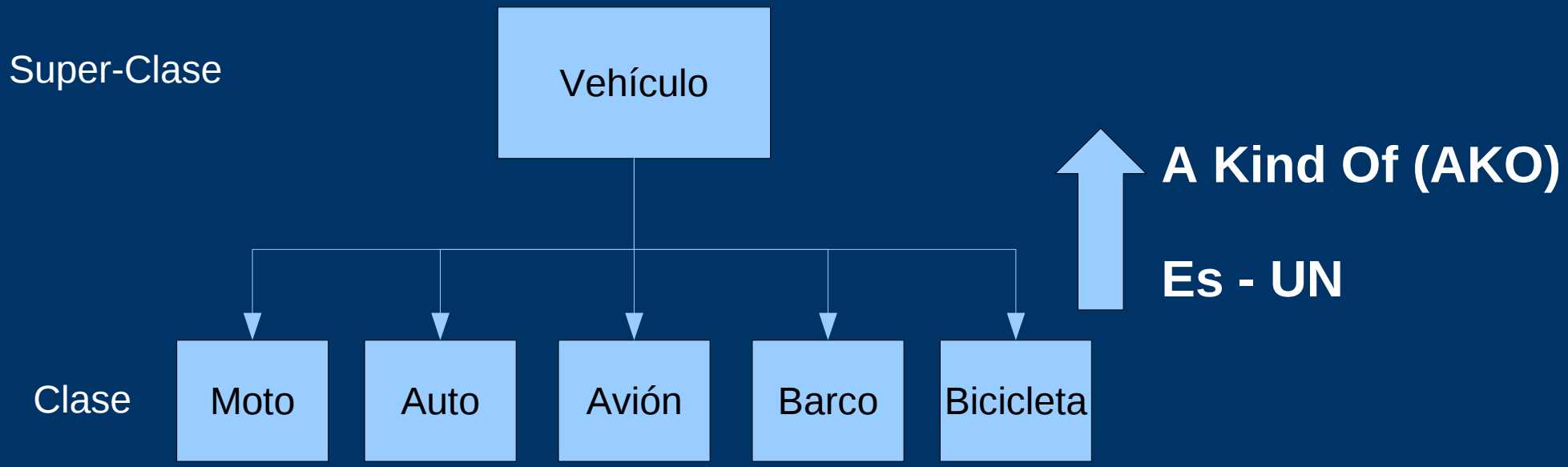



**Abstracta**: No se crean objetos a partir de ésta clase (concepto **AKO**). Generalización

# Programando en Python

## Programación Orientada a Objetos (POO)

**Herencia:** significa que se puede especificar que una clase sea una "especialización" o "descripción específica" de otra clase (llamada super-clase). **Concepto AKO ó ES-UN.**




**¡¡¡Reutilización de código!!!**

# Programando en Python

## Programación Orientada a Objetos (POO)

### ¿Qué hereda un objeto?

**Herencia de Comportamiento**
Si un objeto recibe un mensaje que no entiende, buscar'a el método para el mensaje en la super-clase que le corresponde.

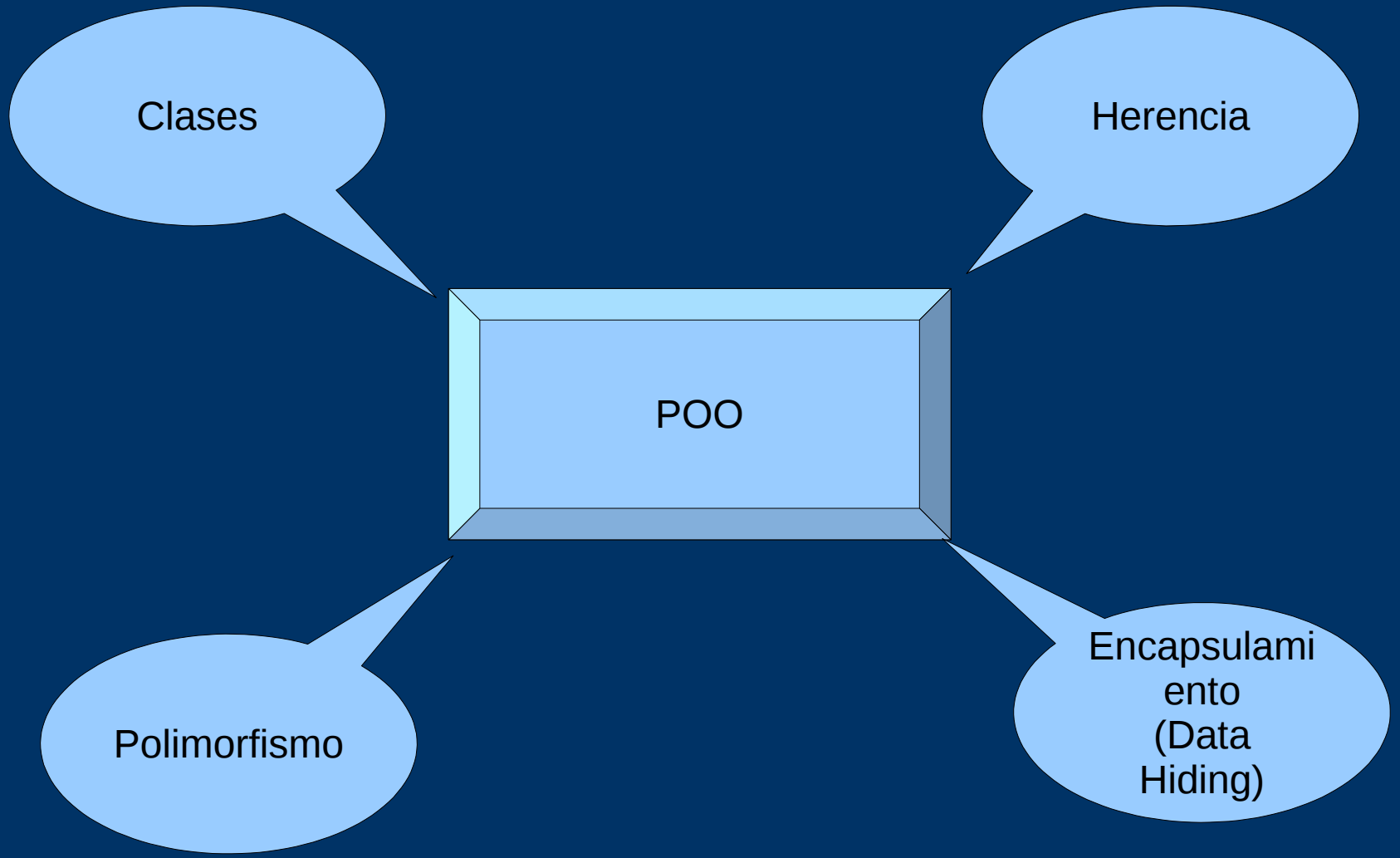
Se identifican métodos similares para definir clases abstractas (mensaje común a todos los objetos que pertenecen a la super-clase).


**Herencia de Estructura**
“La forma de los objetos”. En el momento que “creo” el objeto (instancio), los datos que tendrá ése objeto serán los datos del objeto más los datos de la super-clase.

**Binding Dinámico:** Decimos que hay binding es dinámico cuando para cada ejecución de una expresión, el binding puede cambiar. Ésto ocurre por ejemplo cuando una variable por ejemplo x representa en un momento del programa a un auto pero en otro momento representa a una bicicleta.

# Programando en Python

## Programación Orientada a Objetos (POO)



# Programamando en Python


(Finalmente)

## Lo más aburrido: Tipos de dato standard

**Números**: 1234, 3.1415, 999L, 3+4j, Decimal

**Strings**: “spam”, “guido's”

**Listas**: [1, [2, 'three'], 4]

**Diccionarios**: {'food': 'spam', 'taste':'yum'}

**Tuplas**: (1, 'spam', 4, 'U')

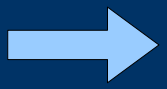
**Archivos**: myfile = open('eggs', 'r')

**Otros tipos**: Conjuntos, tipos, Ninguno, booleanos

# Programando en Python

## Tipos de dato Standard: Números

```
>>> 123+222
345
>>> 1.5*4
6.0
>>> 2**100
1267650600228229401496703205376L
>>>  7.6/3.4
2.2352941176470589
>>> round(5.6)
6.0
>>> round(5.5)
6.0
>>> round(5.4)
5.0
>>> (6 + 3j)
(6+3j)
>>> (5 + 4j) + (1 - 5j)
(6-1j)
>>>
>>>  import math
>>> math.pi
3.1415926535897931
```



Notar la L al final de la cadena numérica: Python automáticamente convierte cualquier entero a un entero “grande” cuando se necesita precisión extra.

```
>>> import random
>>> random.random()
0.9144142472244063 4
>>> random.choice([1,2,3,4])
3
```

# Programando en Python

## Tipos de dato Standard: Strings

Los Strings son un tipo especial de “secuencia”, las secuencias en Python son colecciones ordenadas de otros objetos. Las secuencias mantienen el órden izquierda-a-derecha entre los ítems que contiene. Estrictamente hablando, los strings son secuencias de strings de UN caracter.

```
>>> cadena = "una cadena relativamente larga"
>>> cadena
'una cadena relativamente larga'
>>> print cadena
una cadena relativamente larga
>>> len(cadena)
30
>>> cadena.islower()
True
>>> cadena.isupper()
False
>>> cadena.upper()
'UNA CADENA RELATIVAMENTE LARGA'
>>> cadena.isupper()
False
>>> cadena[0]
'u'
>>> >>> cadena[-4]
'a'
```

**Inmutabilidad**

```
>>> cadena[10:18]
' relativ'
>>> cadena[15:]
'tivamente larga'
>>> cadena + ' que ahora es mas larga'
'una cadena relativamente larga que ahora es mas larga'
>>> cadena * 10
'una cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente largauna cadena relativamente larga'
```

Notar que el signo mas (+) representa diferentes operaciones de acuerdo a los objetos con los que se opera, si los operadores son números entonces los sumará; si los operadores son cadenas entonces los concatenará. Un claro ejemplo de **Polimorfismo**

# Programando en Python

## Tipos de dato Standard: Strings

### Métodos específicos del tipo

Algunas de las operaciones de strings vistas anteriormente son en realidad operaciones de cualquier tipo que pertenezca a la super-clase secuencia, como por ejemplo las listas y las tuplas, los strings tienen operaciones propias (métodos), veamos algunos:

```
>>> cadena.find('tiva')
15
>>> cadena[15]
't'
>>> cadena.replace('larga', 'corta')
'una cadena relativamente corta'
>>> cadena.split(' ')
['una', 'cadena', 'relativamente', 'larga']
>>> dir(cadena)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__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']
>>> help(cadena.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
(END)
```

Listado de operaciones “online”

Ayuda “online”

# Programando en Python

## Tipos de dato Standard: Listas

```
>>> lista = [4502, "IAR", 3.1415, [1, "fer", "python"], 'i']
```

**Polimorfismo**

```
>>> len(lista)
5
>>> lista[3]
[1, 'fer', 'python']
>>> len(lista[3][2])
6
```

**Métodos específicos del tipo**

```
>>> lista.append('unlp')
>>> lista
[4502, 'IAR', 3.1415000000000002, [1, 'fer', 'python'],
'i', 'unlp']
>>> lista.pop(2)
3.1415000000000002
>>> lista
[4502, 'IAR', [1, 'fer', 'python'], 'i', 'unlp']
>>> lista.sort()
>>> lista
[4502, [1, 'fer', 'python'], 'IAR', 'i', 'unlp']
>>> lista.reverse()
>>> lista
['unlp', 'i', 'IAR', [1, 'fer', 'python'], 4502]
```

```
>>> matriz=[]   <-- Lista Vacía
>>> matriz.append([])
>>> matriz
[[]]
>>> matriz=[[1, 2, 3],
...              [4, 5, 6],
...              [7, 8, 9]]
>>> matriz
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matriz[0]
[1, 2, 3]
>>> matriz[0][2]
3
```

Éste tipo de matrices funcionan perfectamente para operaciones en pequeñas escalas. Para realizar tareas numéricas de mayor embergadura existe la librería NumPy.

# Programamando en Python

**Resumiendo:**
El objeto lista es una colección ordenada de otros objetos arbitrarios que puede crecer indefinidamente (si nos alcanza la RAM).

Las listas **PUEDEN** modificarse, contrariamente a lo que pasaba con los strings.

**Un ejemplo de Binding dinámico con Listas**

```
>>> lista = [4502, "IAR", 3.1415, [1, "fer", "python"], 'i']
```

```
>>> for item in lista:
...     print item
...
4502
IAR
3.1415
[1, 'fer', 'python']
i
```

el identificador de objeto 'item' hará **binding dinámico** con cada uno de los items de la lista de acuerdo como avanza el índice, representando a un objeto distinto cada vez.

Más sobre estructuras de control, en unos minutos...

# Programando en Python

## Tipos de dato Standard: Diccionarios

Los diccionarios en Python son sub-clases de la clase **Mapeos**.

La clase **Mapeos** no es ordinal, en cambio su mecánica es asociar palabras clave a valores. Los diccionarios son la única clase standard en Python cuya super-clase es un Mapeo. Pueden modificarse en tiempo de ejecución, permitiendo aumentar o disminuir su tamaño de acuerdo a la demanda del usuario, como las listas.

```
>>> D = {'cantidad' : 5, 'vehiculo' : 'auto', 'opciones' : [1, 2, 3]}
>>> D
{'opciones': [1, 2, 3], 'vehiculo': 'auto', 'cantidad': 5}
>>> D['opciones']
[1, 2, 3]
>>> D['opciones'][2]
3
>>> D.items()
[('opciones', [1, 2, 3]), ('vehiculo', 'auto'), ('cantidad', 5)]
>>> D['peras']='2kg'
>>> D
{'peras': '2kg', 'opciones': [1, 2, 3], 'vehiculo': 'auto', 'cantidad': 5}
>>> 'peras' in D
True
```

# Programando en Python

## Tipos de dato Standard: Tuplas

Una tupla es una lista que no puede ser cambiada. Es una secuencia (como las listas) pero son inmutables (como los strings). Entonces, ¿para qué queremos un tipo de dato que es parecida a una lista pero que no podemos modificar?. Justamente la inmutabilidad de las tuplas hacen a su practicidad, las tuplas mantienen su integridad a lo largo de un programa, nadie puede modificarlas.

```
>>>  t = (0, 'Banana', 10, 20, 123)
>>> t
(0, 'Banana', 10, 20, 123)
>>> len(t)
5
>>> t[1]
'Banana'
>>> t[1] = 'Peras'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

# Programando en Python

## Tipos de dato Standard: Conjuntos

Los objetos “conjunto” son un modelado de los conjuntos que todos conocemos en matemática. En Python los conjuntos se crean llamando a la función set.

```
>>> frutas = set(['bananas', 'manzanas', 'frutillas', 'tomates'])
>>> frutas
set(['tomates', 'frutillas', 'bananas', 'manzanas'])
>>> verduras = set(['zapallos', 'cebollas', 'papas', 'tomates'])
>>> verduras
set(['tomates', 'cebollas', 'zapallos', 'papas'])
>>> frutas & verduras
set(['tomates'])
>>> frutas | verduras
set(['bananas', 'frutillas', 'zapallos', 'papas', 'tomates', 'cebollas',
'manzanas'])
>>> frutas - verduras
set(['frutillas', 'bananas', 'manzanas'])
>>> frutas.symmetric_difference(verduras)
set(['bananas', 'frutillas', 'zapallos', 'papas', 'cebollas', 'manzanas'])
```

# Programando en Python

## Tipos de dato Standard: Archivos

Los Archivos son un tipo especial de objeto, no hay un “constructor” de los objetos archivo. Para crear un objeto de la clase archivo se llama a la función “open” pasándole como parámetro el nombre del archivo y el modo.

```
>>>  archivo = open('archivo.txt', 'w')
>>> dir(archivo)
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__getattribute__',
'__hash__', '__init__', '__iter__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__str__', 'close', 'closed', 'encoding', 'fileno', 'flush',
'isatty', 'mode', 'name', 'newlines', 'next', 'read', 'readinto', 'readline',
'readlines', 'seek', 'softspace', 'tell', 'truncate', 'write', 'writelines',
'xreadlines']
>>> archivo.write('Hola Mundo\n')
>>> archivo.close()
>>> otro_archivo = open('archivo.txt', 'r')
>>> otro_archivo.read()
'Hola Mundo\n'
>>> otro_archivo.seek(0)
>>> bytes = otro_archivo.read()
>>> bytes
'Hola Mundo\n'
>>> bytes.split()
['Hola', 'Mundo']
```

# Programando en Python

## Estructuras de Control de Flujo de programa (seguimos aburridos)

| Sentencia | Rol | Ejemplo |
|---|---|---|
| Asignación | Crea referencias | a, b, c = 'puedo', 'asignar', 'varios' |
| Llamadas | Corre Funciones | string.find('buscame esto') |
| Imprimir | Imprime Objetos | print ('Que aburrido') |
| if/elif/else | Acciones de selección | If “guarda” in texto:<br>  print texto |
| for/else | Iteración de secuencias | for elemento in lista:<br>  print elemento |
| while/else | Iteraciones generales | while x < y<br>  print 'x es menor a y' |
| pass | Marcador de posición vacío | while True:<br>  pass |
| break,continue | Saltos en iteraciones | while True:<br>  If not line: break |
| try/except/finally | Capturadores de Excepciones | try:<br>  accion()<br>except:<br>  Print 'Error en accion' |
| raise | Disparadores de Excepciones | raise, endSearch, location |
| Import,from | Acceso a Módulos | import sys<br>from sys import stdin |
| def,return,yield | Constructores de funciones | def f (a, b, c = 1, *d):<br>  return a+b+c+d[0]<br>def gen(n):<br>  for i in n, yield i*2 |

# Programamando en Python

Qué sale y que entra en el mundo Python respecto de otros lenguajes.
Una pequeñisima introducción a lo que vamos a ver en unos segundos.

## ¿Qué Entra?

- El carácter “:” (dos puntos)

## ¿Qué Sale?

- Los paréntesis son opcionales.
- El fin de la línea es el fin de la sentencia (no hay carácter ';' como en C).
- El fin de la indentación es el fin de un bloque de sentencias.

# Programamando en Python

## Estructuras de Control de Flujo de programa (seguimos aburridos)

**- if / elif / else:**

```
if a > b:
      print “el mayor es %d\n” % a
elif a < b:
      print “el mayor es %d\n” % b
else:
      print “Son iguales”
```

**- while:**

```
while condición:
      sentenciasA
else:
      sentenciasB
```

Se ejecutan las sentenciasA **mientras** la condición sea verdadera, las sentenciasB se ejecutarán sólo si la condición se hace falsa y no se dió un caso de break dentro de sentenciasA.

# Programando en Python

## Estructuras de Control de Flujo de programa

**- break**

"Sale" del loop más cercano "se salta" la fase de chequeo de condición.

**- continue**

"Salta" directamente a la fase de chequeo del bucle.

**- pass**

No hace nada, es una sentencia vacía.

**- for / else**

Es un iterador de secuencias, puede moverse a través de todos los ítems de cualquier objeto secuenciable ordenado, funciona con strings, listas, tuplas y todos los objetos secuenciables.

```
for <elemento> in <objeto_secuenciable>
    <sentenciasA>
else:
    <sentenciasB>
```

El bloque "else" funciona de igual manera que en el loop while: se ejecutarán las sentenciasB **siempre que** el loop for haya terminado satisfactoriamente, ésto es: que el objeto sencuenciable sea completamente iterado y no se haya dado una condición de "break"

# Programando en Python

## Estructuras de Control de Flujo de programa

### Funciones

**def** es código ejecutable. Las funciones en Python se escriben con ésta sentencia. A diferencia de otros lenguajes compilados, una función es código ejecutable en Python, ésto quiere decir que la función no existe hasta que se alcanza el punto donde se define la función, ésto permite por ejemplo anidar funciones dentro de bloques if-else, loops. El uso común de def es crear un módulo donde se definen funciones y luego importarlo.

**def** crea un objeto de clase función y se lo asigna al nombre de la función. Como en todas las asignaciones el nombre de la función se convierte en una referencia al objeto función instanciado.

**return** envia un objeto “de respuesta” a quien haya llamado a la función. Cuando se llama a una función, el módulo que invoca detiene su trabajo hasta que la función haya terminado de realizar las tareas especificadas, luego devuelve el control al modulo que llamó. El valor de retorno se convierte en el resultado del llamado a la función.

**Los Argumentos** se pasan por asignación (referencia a objeto), ésto quiere decir que los argumentos de función hacen referencia a cierto objeto para ser pasados como argumento.

**Las variables son locales** a la función y existen sólo cuando la función está corriendo.

# Programando en Python

## Estructuras de Control de Flujo de programa

## Módulos (import – from – reload)

Los módulos usualmente corresponden a archivos en Python que sirven al propósito de reutilización de código y datos. Los módulos pueden ser también “extensiones” que son módulos codificados en otros lenguajes (C, C++, C#, Java, etc). Cada archivo es un módulo y los módulos importan otros módulos para usar los nombres que éstos definen, los módulos se invocan con dos sentencias:

**import**
hace que un cliente (importador) obtenga todo un módulo

**from**
permite a los clientes obtener nombres particulares de un módulo (funciones, objetos, etc).

Existe también la función reload que no veremos hoy.

## Arquitectura de los programas en Python

Generalmente un programa Python consiste en múltiples archivos que contienen sentencias Python. El programa se estructura como un solo main (archivo) que tiene ninguno, uno o más archivos suplementarios llamados módulos.

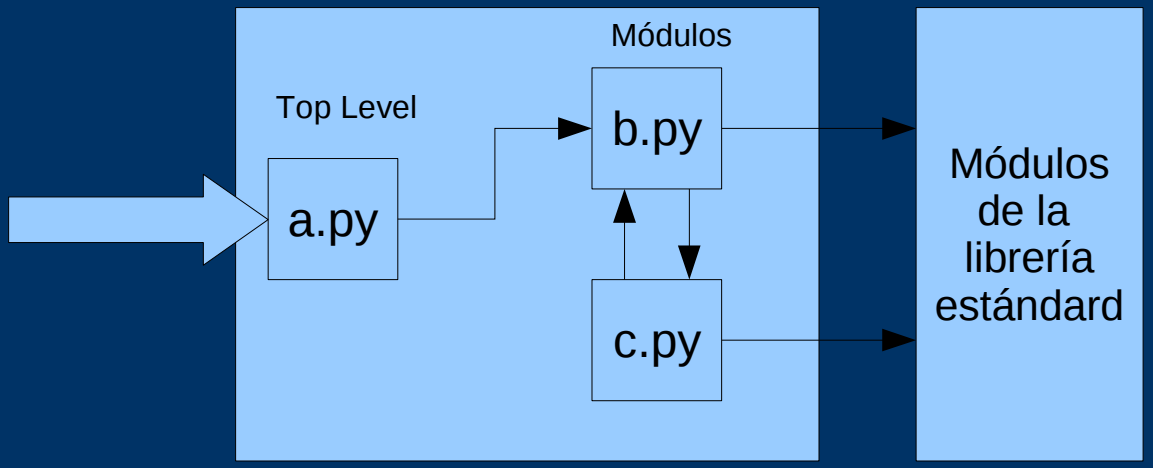
Los módulos generalmente no “hacen” nada cuando corren, en cambio definen las herramientas que van a usar los programas que los invocan. En última instancia, lo que hacemos es importar módulos y acceder a sus atributos para usar sus herramientas.

# Programando en Python

## Estructuras de Control de Flujo de programa

### Módulos (import – from – reload) continuación

Para aclarar un poco lo que estamos diciendo, vamos a suponer la siguiente estructura de un programa Python con múltiples archivos fuente.




**b.py:**

```
def sum (a, b):
        return a + b
```

**c.py:**

```
def prod (a, b):
        return a * b
```

**a.py:**

```
import b
import c

a, b = 2, 3

print b.sum(a, b)
print c.prod(a,b)
```

**a.py:**

```
from b import *
from c import *

a, b = 2, 3

print sum(a, b)
print prod(a,b)
```

# Programando en Python

## Algunos ejemplos de estructuras de control

**for / else:**

```
>>> utiles = ['lapiz', 'lapicera', 'goma', 'sacapuntas']
>>> for util in utiles:
...     print 'comprar ' + util
...
comprar lapiz
comprar lapicera
comprar goma
comprar sacapuntas
>>> for i in range (0, 5):
...     print i**2
...
0
1
4
9
16
>>>
```

**while / else:**

```
>>> lista = []
>>> n = 0
>>> while len(lista) <> 10:
...     lista.append(n)
...     n = n + 1
...
>>> lista
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> n
10
```

# Programando en Python

## Algunos ejemplos de estructuras de control

**Funciones:**

```
>>> def alaquinta(n):
...     return n ** 5
...
>>> alaquinta
<function alaquinta at 0xb7a7ed14>
>>> alaquinta(8)
32768
>>> f = alaquinta
>>> f(8)
32768
>>> f
<function alaquinta at 0xb7a7ed14>
>>>
```

**Funciones:**

```
>>> def factorial(x):
...     if x == 0:
...             return 1
...     else:
...             return x * factorial (x - 1)
...
>>> factorial(10)
3628800
```

# Referencias y Links

Libro **“Learning Python”**
By Mark Lutz
Editorial O'Reilly
ISBN-10: 0-596-51398-4
ISBN-14: 978-0-596-51398-6

http://www.iar.unlp.edu.ar/~fernandoph/pub/OReilly.Learning.Python.3rd.Edition.Oct.2007.pdf (SHHHHHH)

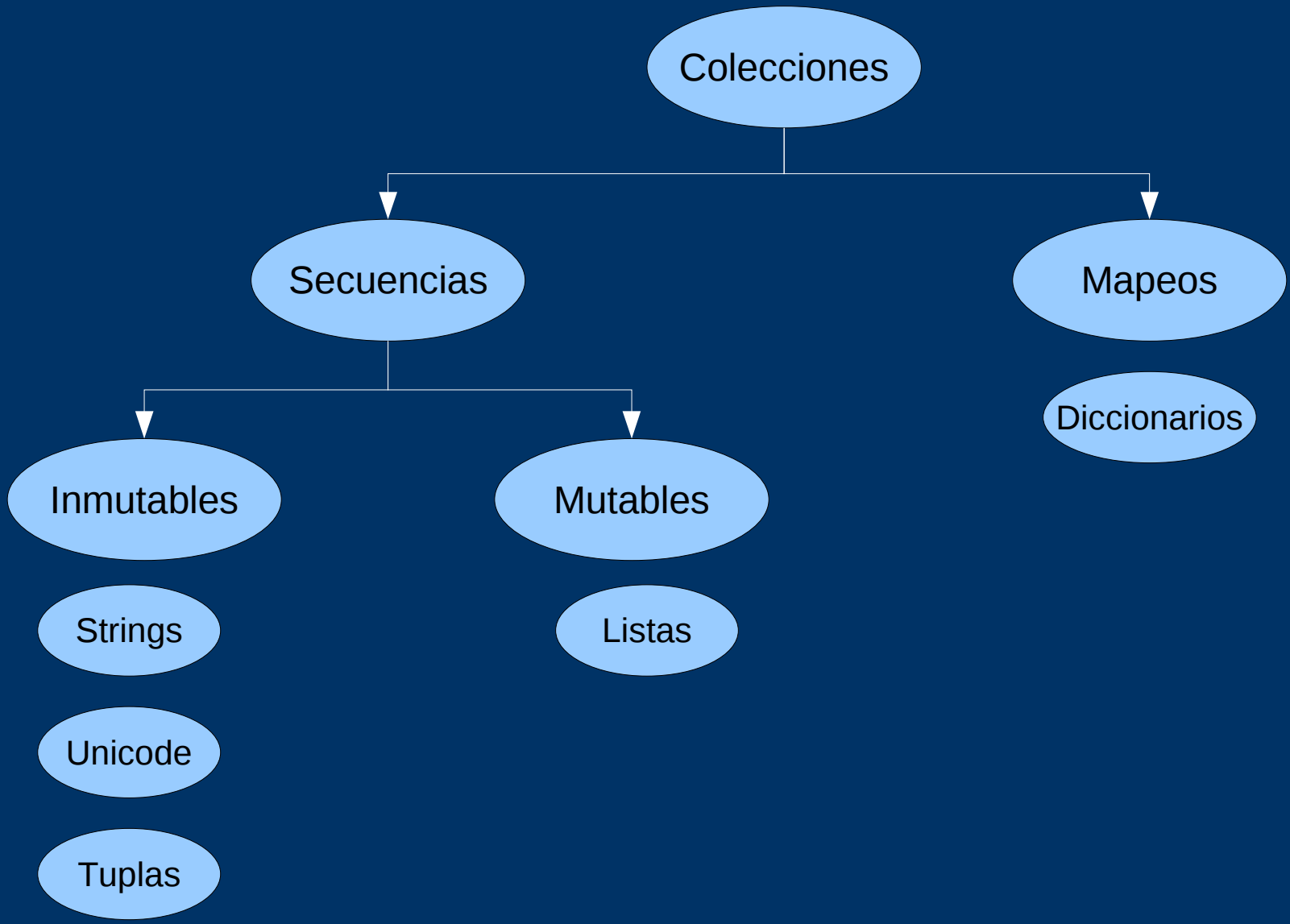
**Tutorial de Python de Guido van Rossum traducido al español por la comunidad PyAr**
http://trac.usla.org.ar/proyectos/python-tutorial

**Python con baterías extra:**
http://www.iar.unlp.edu.ar/~fernandoph/pub/Python+PySerial+GTK_WIN32/

**NumPy**
http://numpy.scipy.org/

**Astrolib**
http://www.scipy.org/AstroLib


# Dudas, comentarios, sugerencias


Colecciones
Secuencias
Mapeos
Diccionarios
Inmutables
Mutables
Strings
Listas
Unicode
Tuplas