# Introducción a la biblioteca estándar de Python

Antonio Espín Herranz

# Contenidos

- Módulos
- Multi-Hilos
- Registros
- Trabajar con listas
- Aritmética en coma flotante.

# Módulos

- **os**: funciones para interactuar con el sistema operativo.
  - os.listdir, os.path.isfile
- **glob**: para buscar archivos mediante comodines.
  - glob.glob(‘*.py’)
- **sys**: argumentos de la línea de comandos.
  - sys.argv
- **re**: expresiones regulares
  - re.match, re.search

# Módulos

- **math**: librería matemática
  - math.sin, math.cos, math.pi, math.sqrt
- **urllib.request**: Acceso a internet
- **datetime**, date, time: (dentro del módulo datetime). Tiempos.
- **timeit**: para medir el rendimiento del código,
  - >>> from timeit import Timer
  - >>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
- Mas info: https://docs.python.org/es/3.8/tutorial/stdlib.html

# Hilos: thread

- Python también soporta la programación multihilos, donde podemos crear varias hebras de ejecución que pueden colaborar en la realización de una tarea.

- Características:
  - También llamados procesos ligeros.
  - Se ejecutan dentro del mismo proceso.
  - Una parte del código de mi programa se ejecuta por varios hilos.
  - No requieren cambios de contexto.
  - Comparten los recursos entre si.

# El GIL

- **GIL**: Global Interpreter Lock.
  - Permite la ejecución de hilos de forma que sólo un thread puede ejecutarse a la vez, independientemente del número de procesadores con el que cuente la máquina.
  - Cada cierto número de instrucciones de bytecode la máquina virtual para la ejecución del thread y elige otro de entre los que estaban esperando.
  - Por defecto el cambio de thread se realiza cada 10 instrucciones de bytecode,

# Conceptos

- ***Bloqueo mutuo (deadlock):***
  - *Es el* ***bloqueo irreversible de un conjunto de hilos o procesos****.*
    - Un bloqueo mutuo es lo que sucede cuando un programa se te queda "colgado" y te ves en la obligación de "matar un proceso" ya que el conjunto de hilos o procesos bloqueados, no tiene solución.

- ***Condición (o estado) de carrera:***
  - *Es aquel que se produce cuando* ***varios hilos o procesos intentan modificar de forma simultánea a un mismo recurso****.*
    - Si más de un hilo o recurso intenta modificar el estado o valor de un mismo recurso al mismo tiempo, los datos dejan de ser confiables y por consiguiente, es correcto afirmar que **los datos quedan corruptos**.

# Threads en Python

- El trabajo con threads se lleva a cabo en Python mediante el **módulo thread**.

- **El módulo threading** que se apoya en el primero para proporcionarnos una API de más **alto nivel**, más completa, y orientada a objetos.

# Threads en Python

- El **módulo threading** contiene una **clase Thread** que debemos **extender** para crear nuestros propios hilos de ejecución.

- El **método run** contendrá el código que queremos que ejecute el thread.

- El **constructor** de la clase tiene que llamar a **Thread.__init__(self)** para inicializar el objeto correctamente.

# Ejemplo

```
import threading
class MiThread(threading.Thread):
    def __init__(self, num):
            threading.Thread.__init__(self)
            self.num = num

    def run(self):
            print "Soy el hilo", self.num
```

- **El código principal**:
  - **Crea** varios hilos.
  - **Nunca** se **llama** directamente al método **run**.
  - Al **llamar a start** el hilo empieza a ejecutar el método run.

```
print "Soy el hilo principal"

for i in range(0, 10):
    t = MiThread(i)
    t.start()
    # Ojo, si ejecutamos join, hilo a hilo hace que los hilos se ejecuten uno detrás de otro.
    t.join()
    # Si no se lanza join, main termina pero los hilos continúan ejecutándose.
```

El método **join** se utiliza para que el hilo que ejecuta la llamada se bloquee hasta que finalice el **thread** sobre el que se llama

# Threads en Python

- Otra posibilidad (pero es mejor heredar de la clase Thread).

```
import threading
    def imprime(num):
        print “Soy el hilo”, num

print “Soy el hilo principal”
for i in range(0, 10):
    t = threading.Thread(target=imprime, args=(i, ))
    t.start()
```

# Mecanismos de Sincronización

- Necesitamos mecanismos de sincronización que nos garanticen el acceso exclusivo a una región crítica (la RC. se considera cualquier recurso a la que pueden acceder 2 o más hilos simultáneamente).

- El módulo de threading proporciona los locks (o semáforos binarios) que permiten el acceso exclusivo a un recurso.

- Hay otros mecanismos como son: semáforos, eventos o colas sincronizadas (Queue)

# Locks

- Los **locks**, también llamados **mutex** (de mutual exclusion), cierres de exclusión mutua, cierres o candados, son objetos con **dos estados** posibles: **adquirido** o **libre**.

- Cuando un thread adquiere el candado, los demás threads que lleguen a ese punto posteriormente y pidan adquirirlo se bloquearán hasta que el thread que lo ha adquirido libere el candado, momento en el cuál podrá entrar otro thread.

# Locks

- El candado se representa mediante la **clase Lock**.
- Para adquirir el candado se utiliza el **método acquire** del objeto, al que se le puede pasar un booleano para indicar si queremos esperar a que se libere (True) o no (False).
- Si indicamos que no queremos esperar, el método devolverá True o False dependiendo de si se adquirió o no el candado, respectivamente.
- **Por defecto, si no se indica nada, el hilo se bloquea indefinidamente.**

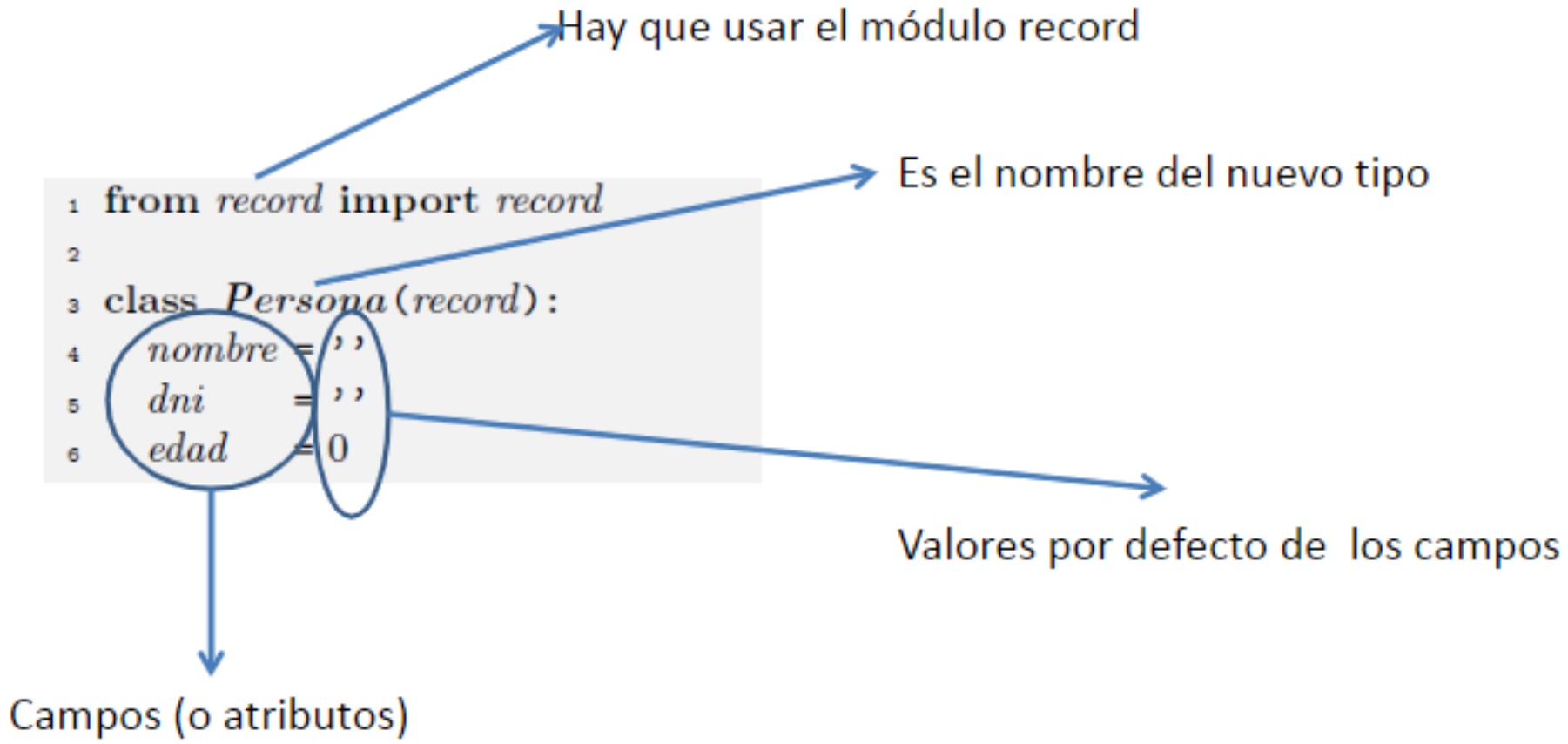
# Ejemplo

```
lista = []
lock = threading.Lock()

def anyadir(obj):
    lock.acquire()
    lista.append(obj)
    lock.release()

def obtener():
    lock.acquire()
    obj = lista.pop()
    lock.release()
    return obj
```

# Registros


Hay que usar el módulo record
Es el nombre del nuevo tipo
1 from record import record
2
3 class Persona(record):
4 nombre = ''
5 dni = ''
6 edad = 0
Valores por defecto de los campos
Campos (o atributos)

# Registros: crear nuevas variables

- Al igual que los objetos …
  - La creación de una variable del nuevo tipo se realiza así:

```
8 juan = Persona(nombre='Juan Paz', dni='12345678Z', edad=19)
9 ana  = Persona(nombre='Ana Mir',  dni='23456789Z', edad=18)
```

  - Esta operación se llama instanciación o construcción.
  - Las variables juan y ana son instancias o registros del tipo Persona

# Registros

- Acceso a los campos con el punto.
- La asignación con el punto: objeto.campo = valor.
- Los registros se pueden anidar:

```
from record import record

class Fecha(record):
    dia  = 1
    mes  = 1
    anyo = 1
```

```
from record import record
from fecha import fecha

class Persona(record):
    nombre = ''
    apellido = ''
    fecha_nacimiento = None
```

```
ana = Persona(nombre='Ana', \
              apellido='Paz', \
              fecha_nacimiento=Fecha(dia=31, mes=12, anyo=1990))
```

```
print ana.fecha_nacimiento.dia
```

# Trabajar con listas

- A partir del tipo list: **dir(list)**
- ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

# Aritmética en coma flotante

- Tener en cuenta la representación en coma flotante
- Por ejemplo: 1/3 siempre almacena internamente una aproximación, puede hacer redondeos a la hora de imprimir. O los podemos realizar nosotros con la función round indicando el número de decimales.

- Prueba en el interprete: .1 + .1 + .1
- Luego: round(.1+.1+.1, 10)

- En la mayoría de las máquinas hoy en día, los float se aproximan usando una fracción binaria con el numerador usando los primeros 53 bits con el bit más significativos y el denominador como una potencia de dos.
  - En el caso de 1/10 se puede almacenar como: `3602879701896397 / 2 ** 55`

- Enlace:
  - https://docs.python.org/es/3/tutorial/floatingpoint.html