


# Semana 2: Introduccion a Python

SISTEMAS EMBEBIDOS   
(EMBEDED SYSTEMS)

Grado Dual en Industria Digital

Campus Vitoria

Curso 2020-2021



**Deusto**

Facultad de Ingeniería  
Ingeniaritza Fakultatea

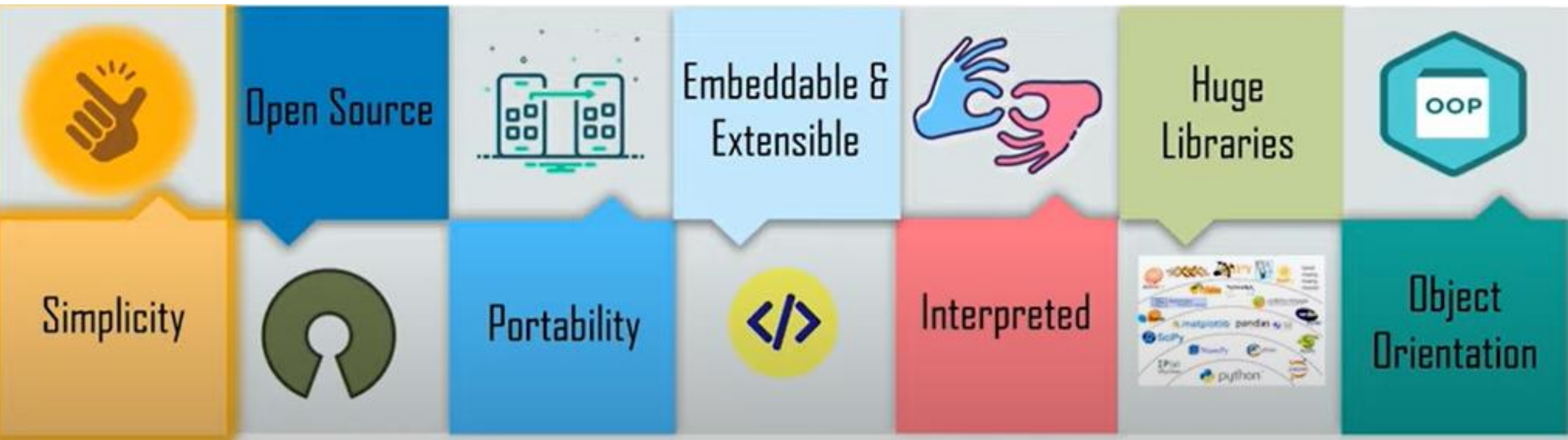
# Origen de Python

- Creado por Guido van Rossum a principios de los 90 con características de OO.
- El nombre es un tributo a los *MonthlyPyhton*
- 1997: Python 1.0
- 2010: Python 2.7
- 2015: Python 3.5
- 20xx: Python 3.8



- ☐ Permite tanto programación estructurada como orientada a objetos
- ☐ Tipos dinámicos: las variables no se declaran con tipo, pueden adquirir cualquier tipo-valor dinámicamente
- ☐ Gestión dinámica de memoria con recolector de basura
- ☐ Late binding (enlace tardío) en la resolución de nombres
- ☐ Muy legible y simple, poco verboso
- ☐ Los programas de otros desarrolladores se entienden fácilmente
- ☐ Muy extensible

# Por que es Python tan popular ??





# Oportunidades Laborales



# Holamundo.py

- ❑ El nivel de indentación/tabulación marca el control de flujo y los dos puntos ":" indican inicio de bloque:
- ❑ Los comentarios, con # o """ si es multilinea

```
print ("Hola mundo!")

for n in range(1,5):
    print ("Iteracion ",n)
    print ("Hola mundo")
    print ("Ahora estoy fuera del bucle")
# Esto es un comentario
""" Esto es un comentario
en varias líneas """
```

# Actividad

- En Python bajo Linux normalmente tenemos el intérprete en `/usr/bin` pudiendo existir varias versiones instaladas simultáneamente
- Examinar las versiones existentes en `/usr/bin` y ver cual está asociada al link simbólico Python
- Arrancar python, python2 y python3
- Salir con `exit()`

# Actividad

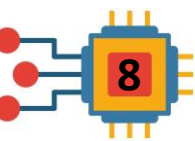
- En Python bajo Linux normalmente tenemos el intérprete en /usr/bin pudiendo existir varias versiones instaladas simultáneamente
- Examinar las versiones existentes en /usr/bin y ver cual está asociada al link simbólico Python

```
$ ls | grep "python"  
$ ls /usr/bin/python*
```

- Arrancar python, python2 y python3
- Salir con exit()

Para cambiar la versión de Python2 a Python 3:

<https://linuxconfig.org/how-to-change-from-default-to-alternative-python-version-on-debian-linux>





- Background
- Syntax
- Types / Operators / Control Flow
- Functions
- Classes
- Tools

# What is Python

- Multi-purpose (Web, GUI, Scripting, etc.)
- Object Oriented
- Interpreted
- Strongly typed and Dynamically typed
- Focus on readability and productivity

# Syntax

# Hello World

```
#!/usr/bin/env python  
print "Hello World!"
```

hello\_world.py

```
# En Python 3 son necesarios  
# los paréntesis  
print ("Hello World!")
```

# Indentation

- Most languages don't care about indentation
- Most humans do
- We tend to group similar things together



# Indentation

```
/* Bogus C code */  
if (foo)  
    if (bar)  
        baz(foo, bar);  
else  
    qux();
```

The else here actually belongs to the 2nd if statement



# Indentation

```
/* Bogus C code */  
if (foo) {  
    if (bar) {  
        baz(foo, bar);  
    }  
else {  
    qux();  
}}
```

The else here actually belongs to the 2nd if statement

# Indentation

```
/* Bogus C code */  
if (foo)  
if (bar)  
baz(foo, bar);  
else  
qux();
```

I knew a coder like this

# Indentation

```
/* Bogus C code */  
if (foo) {  
    if (bar) {  
        baz(foo, bar);  
    }  
    else {  
        qux();  
    }  
}
```

You should always be explicit

# Indentation

```
# Python code
if foo:
    if bar:
        baz(foo, bar)
    else:
        qux()
```

Python embraces indentation

# Comments

```
# A traditional one line comment
```

```
"""
```

```
Any string not assigned to a variable is  
considered a comment.  
This is an example of a multi-line comment.  
"""
```

```
"This is a single line comment"
```

# Types



# Variables

- No es necesario ni declarar las variables ni especificar el tipo de datos que contienen
  - `valor=1`
  - `valor="hola"`
- Asignación múltiple
  - `a=b=c=10`
- Python tiene 6 tipos estándar:
  - Numeros
  - Booleanos
  - String
  - Lista
  - Tupla
  - Diccionario

```
# Integers Numbers  
year = 2010  
year = int("2010")
```

```
# Floating Point Numbers  
pi = 3.14159265  
pi = float("3.14159265")
```

```
# Fixed Point Numbers  
from decimal import Decimal  
price = Decimal("0.02")
```

- Cuatro subtipos
  - Entero
    - 1234
  - Enteros largos (no en Python3)
    - 123123123L
  - Flotantes
    - 123.23e-4
  - Complejos
    - 34+5j

# Strings

```
# This is a string
name = "Nowell Strite (that\"s me)"

# This is also a string
home = 'Huntington, VT'

# This is a multi-line string
sites = '''You can find me online
on sites like GitHub and Twitter.'''

# This is also a multi-line string
bio = """If you don't find me online
you can find me outside."""
```

# Strings

Hello

0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> cadena="Esto es un texto"
>>> cadena[0:4]
'Esto'
>>> cadena[5:]
'es un texto'
>>> cadena[-1]
'o'
>>> cadena[0:12:2]
'Et su '
>>> cadena[:-5]
'Esto es un '
>>> cadena[-5:]
'texto'
```



## Strings

Name	Purpose
<code>len(s)</code>	Calculate the length of the string <code>s</code>
<code>+</code>	Add two strings together
<code>*</code>	Repeat a string
<code>s.find(x)</code>	Find the first position of <code>x</code> in the string <code>s</code>
<code>s.count(x)</code>	Count the number of times <code>x</code> is in the string <code>s</code>
<code>s.upper()</code> <code>s.lower()</code>	Return a new string that is all uppercase or lowercase
<code>s.replace(x, y)</code>	Return a new string that has replaced the substring <code>x</code> with the new substring <code>y</code>
<code>s.strip()</code>	Return a new string with whitespace stripped from the ends
<code>s.format()</code>	Format a string's contents



# Null is None

```
optional_data = None
```

- Python proporciona:
  - Listas: flexibles, heterogéneas. Se delimitan con []
    - [23, 56.10, “Hola”]
  - Diccionarios: correspondencias clave-valor. Se delimitan con {}
    - {“luz”:56, “temperatura”:23.5, “acceso”:“admin”}
  - Tuplas: series de datos inmutables. Se delimitan con ()
    - (“frio”, “templado”, “calor”)

```
# Lists can be heterogeneous
favorites = []

# Appending
favorites.append(42)

# Extending
favorites.extend(["Python", True])

# Equivalent to
favorites = [42, "Python", True]
```



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

```
len(numbers)
```

```
# 5
```

```
numbers[0]
```

```
# 1
```

```
numbers[0:2]
```

```
# [1, 2]
```

```
numbers[2:]
```

```
# [3, 4, 5]
```

- Borrado de un elemento por su subíndice
  - del numbers[1]
- Borrado de un elemento por su valor
  - palabras.remove(“hola”)
- Acceso al último y penúltimo elemento
  - print(numbers[-1])
  - print(numbers[-2])

Name	Purpose
<code>len(x)</code>	Calculate the length of the list <b>x</b>
<code>x.append(y)</code>	Add the object <b>y</b> to the end of the list <b>x</b>
<code>x.extend(y)</code>	Extend the list <b>x</b> with the contents of the list <b>y</b>
<code>x.insert(n, y)</code>	Insert object <b>y</b> into the list <b>x</b> before position <b>n</b>
<code>x.pop()</code> <code>x.pop(n)</code>	Remove and return the first object in the list Remove and return the object at position <b>n</b>
<code>x.count(y)</code>	Count the number of times object <b>y</b> is in the list
<code>x.sort()</code>	Sorts the list <b>x</b> in-place
<code>sorted(x)</code>	Returns sorted version of <b>x</b> (does not change <b>x</b> )
<code>x.reverse()</code>	Reverses the list <b>x</b> in-place



- Crear en el interprete una lista llamada valores mezclada de 2 enteros, 2 reales y 2 strings
- Mostrar por pantalla:
  - Elemento 1
  - Elementos del 0 al 3, incluidos
- Añadir un entero al final de la lista
- Modificar el elemento 1 asignando el valor “Casa”
- Volver a visualizar la lista
- Eliminar el elemento con subíndice 1 de la lista y volverla a visualizar

- Son una colección inmutable de elementos separados por comas y entre ()

```
tupla=("casa",24,True)
```

- Se acceden mediante posición

```
tupla [1]          # el valor 24
```

```
tupla[0:2]         # ("casa",24)
```

- Operaciones:

```
len(tupla)
```

```
# 3
```

```
"casa" in tupla
```

```
# True
```

```
tupla+tupla
```

```
# ('casa', 24, True, 'casa', 24, True)
```



## Dictionaries

```
person = {}

# Set by key / Get by key
person['name'] = 'Nowell Strite'

# Update
person.update({
    'favorites': [42, 'food'],
    'gender': 'male',
})

# Any immutable object can be a dictionary key
person[42] = 'favorite number'
person[(44.47, -73.21)] = 'coordinates'
```

# Dictionary Methods

```
person = {'name': 'Nowell', 'gender': 'Male'}
```

```
person['name']
```

```
person.get('name', 'Anonymous')
```

```
# 'Nowell Strite'
```

```
person.keys()
```

```
# ['name', 'gender']
```

```
person.values()
```

```
# ['Nowell', 'Male']
```

```
person.items()
```

```
# [('name', 'Nowell'), ('gender', 'Male')]
```

# Actividad

- Crear un carpeta `labpython` y dentro de ella crear con nano un fichero `diccionario1.py`
- Crea un diccionario llamado `persona` con las claves `nombre`, `apellido` y `edad`, y unos valores cualquiera
- Visualiza la edad accediendo a su clave
- Salir de nano y ejecutar con las siguientes alternativas:  
`python diccionario1.py`  
`python3 diccionario1.py`
- Añadir la siguiente línea al principio del fichero  
`#!/usr/bin/env python`
- Hacer el fichero ejecutable con `chmod a+x diccionario1.py` y ejecutar con  
`./diccionario1.py`
- Añade al diccionario tras visualizar la edad, una nueva clave-valor llamada `profesion` con el valor “ingeniero”
- Visualiza la persona

```
# This is a boolean  
is_python = True
```

```
# Everything in Python can be cast to boolean  
is_python = bool("any object")
```

```
# All of these things are equivalent to False  
these_are_false = False or 0 or "" or {} or []  
or None
```

```
# Most everything else is equivalent to True  
these_are_true = True and 1 and "Text" and  
{ 'a': 'b' } and ['c', 'd']
```

# Operators

# Arithmetic

```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a + 1       # 11
c = a - 1       # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```





## String Manipulation

```
animals = "Cats " + "Dogs "  
animals += "Rabbits"  
# Cats Dogs Rabbits
```

```
fruit = ', '.join(['Apple', 'Banana', 'Orange'])  
# Apple, Banana, Orange
```

```
date = '%s %d %d' % ('Sept', 11, 2010)  
# Sept 11 2010
```

```
name = '%(first)s %(last)s' % {  
    'first': 'Nowell',  
    'last': 'Strite'}  
# Nowell Strite
```

- Probar en el intérprete

`"Hola"[0:2]`

`"Hola"*3`

`"o" in "Hola"`

`"Tu nombre es %s" % "John"`

`"Tu nombre es %s y tu apellido %s" % ("John", "Rambo")`

`"Tu nombre es %s y tu apellido %s, edad %d" % ("John", "Rambo", 32)`

`"La edad de %(nombre)s es %(edad)d" % {"edad":34, "nombre":"Mikel"}`

`"La temperatura es %f" % 23.4`

`"La temperatura es %.2f" % 23.4`

# Logical Comparison

```
# Logical And  
a and b
```

```
# Logical Or  
a or b
```

```
# Logical Negation  
not a
```

```
# Compound  
(a and not (b or c))
```

# Identity Comparison

```
# Identity
1 is 1 == True

# Non Identity
1 is not '1' == True

# Example
bool(1) == True
bool(True) == True

1 and True == True
1 is True == False
```

# Arithmetic Comparison

## # Ordering

a > b

a >= b

a < b

a <= b

## # Equality/Difference

a == b

a != b

## Salida a pantalla

- `print("Hola")`
- `print("Hola ", "Mundo")`
  - Diferente interpretación en Python2 que en Python3

## Lectura de teclado

- `valor=input("Teclea tu edad: ")`
  - Diferente en Python2 y Python3, ya que en Python 2 evalúa expresiones (por ejemplo, escribe 2+2)

# Control Flow



# Conditionals

```
grade = 82
if grade >= 90:
    if grade == 100:
        print 'A+'
    else:
        print "A"
elif grade >= 80:
    print "B"
elif grade >= 70:
    print "C"
else:
    print "F"

# B
```



**Deusto**

Facultad de Ingeniería  
Ingeniaritza Fakultatea

# For Loop

```
for x in range(10): #0-9  
    print x
```

```
fruits = ['Apple', 'Orange']
```

```
for fruit in fruits:  
    print fruit
```

# Expanded For Loop

```
states = {  
    'VT': 'Vermont',  
    'ME': 'Maine',  
}  
  
for key, value in states.items():  
    print '%s: %s' % (key, value)
```

# While Loop

```
x = 0  
while x < 100:  
    print x  
    x += 1
```

- Hacer un programa ejecutable en **Pyhton 3** denominado palabras.py
- Solicitar al usuario palabras en un bucle while y guardarlas en una lista hasta que escriba “fin”
- Posteriormente recorrer las palabras de la lista en un bucle for y por cada una mostrar por pantalla el número de letras que tiene
- La palabra de control “fin” no debe aparecer en la lista procesada

# List Comprehensions

- Useful for replacing simple for-loops.

```
odds = [ x for x in range(50) if x % 2 ]
```

```
odds = []  
for x in range(50):  
    if x % 2:  
        odds.append(x)
```

# Functions



**Deusto**

Facultad de Ingeniería  
Ingeniaritza Fakultatea

# Basic Function

```
def my_function():  
    """Function Documentation"""  
    print "Hello World"
```





## Function Arguments

```
# Positional
def add(x, y):
    return x + y

# Keyword
def shout(phrase='Yipee!'):
    print phrase

# Positional + Keyword
def echo(text, prefix=''):
    print '%s%s' % (prefix, text)
```

# Arbitrary Arguments

```
def some_method(*args, **kwargs):  
    for arg in args:  
        print arg  
  
    for key, value in kwargs.items():  
        print key  
  
some_method(1, 2, 3, name='Numbers')
```



## Fibonacci

```
def fib(n):  
    """Return Fibonacci up to n."""  
    results = []  
    a, b = 0, 1  
    while a < n:  
        results.append(a)  
        a, b = b, a + b  
    return a
```

# Fibonacci Generator

```
def fib():  
    """Yield Fibonacci."""  
    a, b = 0, 1  
    while True:  
        yield a  
        a, b = b, a + b
```

```
for v in fib():  
    print (v)  
# Visualiza los números
```


# Actividad

- Diseñar en `calculadora_f.py` las funciones:
  - `sumar(op1,op2)`
  - `restar(op1,op2)`
  - multiplicar, dividir, elevar, ...
- En la operación `elevar(base,exponente)` hacer que el parámetro `exponente` sea opcional, siendo el valor por defecto 2 en caso de no incluirlo.
- Invocar las diferentes funciones con o son parámetros opcionales, mostrando el resultado por pantalla para validar el funcionamiento.
- EXTRA: Añadir al final del fichero un programa que muestre un menú con las operaciones (y una opción de salir), el usuario seleccione la deseada, luego introduzca los operandos, visualice el resultado y vuelva a mostrar el menú hasta que se seleccione la opción de salir.

# Classes

# Declaración de clase y atributos

```
class User():  
    nombre = "Nadie"  
    tecnico = False  
  
juan=User()  
pepe=User()  
print(pepe.nombre)  
juan.nombre = "Juan"  
pepe.nombre = "Pepe"  
print(pepe.nombre + " y " + juan.nombre)  
if not pepe.tecnico:  
    print(pepe.nombre + " no es tecnico")
```



# Constructor y métodos

```
class Persona():  
    def __init__(self, n="Anonimo", e=0):  
        self.nombre = n  
        self.edad = e  
  
    def envejecer(self, anyos):  
        self.edad += anyos
```

```
juan=Persona()  
pepe=Persona("Pepe", 34)  
pepe.envejecer(3)  
print("Juan tiene " + str(juan.edad))  
print("Pepe tiene " + str(pepe.edad))
```



## Actividad guiada

- En esta actividad llevaremos a cabo una serie de pasos viendo los errores que se producen si no lo hacemos bien.
- Crear un fichero ejecutable `persona.py`
- Crear una clase `Persona` con atributos `nombre` y `edad`.
- Definir un constructor `__init__` que reciba parámetros opcionales para dichos atributos, con valores por defecto de “Anonimo” y 0 respectivamente
- Definir una función `supernombre` que devuelve (return) el nombre con el prefijo “super”
  - Hacer este paso con `supernombre()` sin parámetros
- Crear una persona en una variable `p` con unos datos y mostrar por pantalla el resultado de hacer `supernombre()` sobre ella.
- Corregir la función `supernombre(self)` y probar de nuevo

## Actividad guiada

- Ahora mostrar por pantalla directamente la persona, por ejemplo `print (p)`
- Redefinir la función `__str__(self)` en `Persona` para que devuelva un texto como “Pepe tiene 20 anyos”
- Probar de nuevo `print (p)`

- Se indica con el nombre de la clase base de la cual se hereda entre paréntesis en la declaración de clase:

```
class SuperUser(User):  
    ...
```

- Se puede invocar el constructor de la clase base, desde la clase derivada (con diferentes sintaxis):
  - Python2 y Python3:  
    User.\_\_init\_\_(self, "John")
  - Solo Python3:  
    super().\_\_init\_\_("John")

## Actividad guiada

- Extender el mismo fichero persona.py anterior, creando tras Persona una clase Estudiante que herede de ella con un atributo universidad
- Crear un constructor para Estudiante que:
  - reciba los 3 atributos (nombre, edad y universidad, sin valores por defecto)
  - invoque al constructor de la clase base
  - establezca su atributo universidad con el valor recibido como parámetro
- Crear un estudiante llamado Mikel con valores cualesquiera y mostrar sus datos por pantalla.
- Hacer `print(mikel)`
- Añadir la función `__str__(self)` a Estudiante devolviendo tanto su nombre, edad como la universidad
- Hacer de nuevo `print(mikel)`

# Actividad guiada

```
class Persona():
    def __init__(self, nombre="Anonimo", edad=0):
        self.nombre = nombre
        self.edad = edad

    def envejecer(self, anyos):
        self.edad += anyos

    def supernombre(self):
        return "super" + self.nombre

    def __str__(self):
        return "%s tiene %d anyos" % (self.nombre, self.edad)

class Estudiante(Persona):
    def __init__(self, n, e, u):
        super().__init__(n,e)
        self.universidad = u
```

# Python's Way

- No interfaces
- Private attributes & methods can be *simulated* by starting (but not ending) with double underscores:

`__value`

- Special class methods start and end with double underscores.

`__init__, __doc__, __cmp__, __str__`



# Publico / privado

```
class Ejemplo:
    def __init__(self, valor):
        self.valor = valor
        self.__valorprivado = valor

    def getValorPrivado(self):
        return self.__valorprivado

    def __operacionPrivada(self, op1, op2):
        self.valor = op1 * op2

    def operacionPublica(self, op1, op2):
        self.__operacionPrivada(op1, op2)

p = Ejemplo(12)
print(p.valor) # 12
print(p.getValorPrivado()) # 12
p.operacionPublica(10, 2)
print(p.valor) # 20
print(p.getValorPrivado()) # 12
```

# Actividad Sensores

- **Atención: en los siguientes laboratorios usaremos las clases que diseñamos ahora por lo que es necesario conservarlas.**
- Crear un fichero sensor.py
- Definir una clase Medida con atributos “valor” y “unidad” que se reciben como parámetros en el constructor.
- Definir una clase Sensor con un atributo llamado “tipo” que se recibe como parámetro en el constructor y con otro atributo medidas que se inicializa como una lista vacía.
- Definir para cada una de las clases la función `__str__(self)` que permite invocar `str(...)` sobre una instancia de la clase, retornando una representación en string de la misma. En el caso del Sensor devolverá el tipo y el array de medidas.
- Hacer un programa que cree un sensor, le añada 3 medidas y al hacer `print(str(sensor))` o `print(sensor)` muestre, por ejemplo:

```
Sensor de temperatura  
30 grados  
29 grados  
27 grados
```



## Module imports

- Allows code isolation and re-use
- Adds references to variables/classes/functions/etc. into current namespace

# Imports

```
# Imports the datetime module into the
# current namespace
import datetime
datetime.date.today()
datetime.timedelta(days=1)

# Imports datetime and adds date and
# timedelta into the current namespace
from datetime import date, timedelta
date.today()
timedelta(days=1)
```

## More Imports

```
# Renaming imports
from datetime import date
from my_module import date as my_date

# This is usually considered a big No-No
from datetime import *
```

Listado de librerías estándar en Python 3:

<https://docs.python.org/3/library/>

# Módulos y ejecución principal

```
# fichero mod1.py
def func():
    print("func() en mod1.py")

print("ejecucion pricipal en mod1.py")

if __name__ == "__main__":
    print("mod1.py ejecutado directamente por el intérprete")
else:
    print("mod1.py se ha importado por otro modulo")

# fichero mod2.py
import mod1

print("Invocacion directa a mod1 desde mod2")
mod1.func()

if __name__ == "__main__":
    print("mod1.py ejecutado directamente por el intérprete")
else:
    print("mod1.py se ha importado por otro modulo")
```



## Error Handling

```
import datetime
import random

day = random.choice(['Eleventh', 11])
try:
    date = 'September ' + day
except TypeError:
    date = datetime.date(2010, 9, day)
else:
    date += ' 2010'
finally:
    print date
```

- Strings que deben aparecer como **primer contenido** de un módulo, clase o función.

```
def foo():  
    """  
    Python supports documentation for all modules,  
    classes, functions, methods.  
    """  
    pass  
  
# Access docstring in the shell  
help(foo)  
  
# Programatically access the docstring  
foo.__doc__
```



**Deusto**

Facultad de Ingeniería  
Ingeniaritza Fakultatea

# Package Management

```
easy_install pip
```

```
pip install django
```

```
pip install git+git://github.com/  
django/django.git#egg=django
```

# Control de tiempo

- Modulo time

```
import time
```

```
while True:  
    print ("Sigo aqui")  
    time.sleep (5.5)
```



- Modulo datetime
- **Tipos:** date, time, datetime, tzinfo, timezone, timedelta
- **Ejemplos:**

```
d = date.today()
print(d)
d = date (2016,11,8)
print("El año es " + str(d.year))
print(datetime.now())
print(datetime.utcnow())
dt=datetime.now()
print ("%02d:%02d:%02d" % (dt.hour, dt.minute, dt.second))
```