PTC-Temariov2.pdf

CAZZ

Programacion Tecnica y Cientifica

4º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

PTC (Programación tecnica y cientifica)

A continuación se va a probar a realizar todos los ejemplos de codigo en python usados en el temario de PTC

Temario PTC:

Tema 1: Introducción a Python

Indice

La salida no es la deseada:

In [1]:
```
print(0.3 - (0.1 + 0.1 + 0.1))
```

```
-5.551115123125783e-17
```

In [11]:
```
print("División entera: {}\nDivision decimal: {}".format((2//3), (2/3)))
```

```
División entera: 0
Division decimal: 0.6666666666666666
```

Se puede reutilizar el resultado anterior con _

- El la primera linea añadir #!/usr/bin/env python3
- Dar permisos: chmod u+x nombre.py
- ./nombre.py

In [2]:
```
x = 5
print("idx: {}".format(id(x)))
y = x
print("idy: {}".format(id(y)))
x = 7
print("x: {}, y: {}".format(x, y))
print("idx: {}".format(id(x)))
print("idy: {}".format(id(y)))
```

```
idx: 94462386060736
idy: 94462386060736
x: 7, y: 5
idx: 94462386060800
idy: 94462386060736
```

In [3]:
```
def sumar(x, n):
    return x + n

def cociente_resto(a , b):
    return a/b, a%b

print(sumar(3,2))
print(cociente_resto(6, 4))
x, y = cociente_resto(6, 4)
```

```
5
(1.5, 2)
```

Documentacion:

In [28]:
```
def fact(n):
    """
    fact(n) asume que n es un entero positivo y devuelve 1
    """
    return 1

print(fact.__doc__)
```

```
    fact(n) asume que n es un entero positivo y devuelve 1
```

En vez de utilizar una variable temporal como intermediaria

In [90]:
```
def swap(a, b):
    return b, a
```

Alias:

In [95]:
```
f = swap
a = 10
b = 20
```

```
print(f)
print("Es la direccion de memoria en decimal:", id(f))

print("a: {}, b: {}".format(a, b))
a, b = f(a, b)
print("a: {}, b: {}".format(a, b))
```

```
<function swap at 0x7f0928613ca0>
Es la direccion de memoria en decimal: 139677308894368
a: 10, b: 20
a: 20, b: 10
```

In [99]:

```
def  func(a, b = 10):
    return a + b

print(func(6))

func(b = 12, a = 10)
print(func(b=10, a=12))
```

```
16
22
```

Funciones lambda

In [107…

```
suma = lambda x,y : x+y
print(suma(10,15))
f = lambda : 1
f()
```

```
25
```

Out[107… 1

Operaciones:

- Operaciones: + - / // * % divmod abs, int
- Operaciones bit a bit: | & ^ ~ << >> float bin
- Comparar nunca con 0 siempre con numero 1e-10, como en el ejemplo

In [116…

```
x = 5
print(x.bit_length())
print(bin(x))
print(hex(x))
print(oct(x))
print(abs(0.1 + 0.1 + 0.1 - 0.3) < 1e-10)
print((0.1 + 0.1 + 0.1 == 0.3))
print(0.1 + 0.1 + 0.1)
```

```
3
0b101
0x5
0o5
True
False
0.30000000000000004
```

In [121…

```
import math
```

```
from math import exp
from math import *
from math import exp as e
x = e(12)
print(x)
```

```
162754.79141900392
```

Complex:

In [124…

```
x = 3 + 2j
x.conjugate()
```

Out[124…

```
(3-2j)
```

Str:

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

In [132…

```
cad = "\\windows"
cad = r"\windows"

cad = "Hola mundo"
print(cad[0:3])
print(cad[0:-1])
print(cad[::-1])
```

```
Hol
Hola mund
odnum aloH
```

In [135…

```
x = "1234"
type(x)
```

Out[135…

```
str
```

In [139…

```
x = 34
print(type(x))

isinstance(x, int)
```

```
<class 'int'>
```

Out[139…

```
True
```

Entrada estandar:

In [141…

```
x = int(input("Numero datos: "))
```

```
Numero datos: 3
```

Salida estandar:

In [147…

```
x = 2
print("El cuadrado de", x, "es", x * x)
```

```
print("El cuadrado de {} es {}".format(x, x**2))
print("El cuadrado de %d es %d" %(x, x**2))

print("esto","es","un","test")
print("esto","es","un","test", sep="")
print("esto","es","un","test", sep="|", end="")

print("El cuadrado de {num1} es {num2}".format(num1=x, num2=x**2))
```

```
El cuadrado de 2 es 4
El cuadrado de 2 es 4
El cuadrado de 2 es 4
esto es un test
estoesuntest
esto|es|un|testEl cuadrado de 2 es 4
```

If else:

In [149…

```
x = 2
if x < 0:
    print("Negativo")
elif x == 0:
    print("Cero")
else:
    print("Positivo")
```

```
Positivo
```

Funcion lambda con if else:

In [151…

```
x = 100
resultado = (-1 if x < 0 else 1)
print(resultado)

fact = lambda n : 1 if n == 1 or n == 0 else n*fact(n-1)
fact(5)
```

```
1
```

Out[151… 120

Bucles for:

In [153…

```
for l in (12, 45, 23, 9, 12, 20):
    print(l)
```

```
12
45
23
9
12
20
```

In [156…

```
for i in range(20):
    print(i, end="")

print("")
```

```
for r in range(1, 5, 2):
    print(r, end="")
```

```
0123456789101112131415161718 19
13
```

- break -> rompe el bucle y no lo ejecuta mas
- continue -> se salta el codigo que queda en el bucle para pasar a la siguiente iteracion

In [159…

```
for x in (1,2,3,4,5,6):
    print(x)
else: print("Bucle terminado")

print("")

for x in (1,2,3,4,5,6):
    print(x)
    if x > 3: break
    else: print("Bucle llega hasta el final")
```

```
1
2
3
4
5
6
Bucle terminado

1
Bucle llega hasta el final
2
Bucle llega hasta el final
3
Bucle llega hasta el final
4
```

In [160…

```
for i in range(12):
    print(i)
    i += 2
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

Iterar con varias variables zip:

- El bucle termina cuando itera sobre la lista mas corta

In [161…

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

```
0 100
1 101
```

```
2 102
3 103
4 104
5 105
6 106
7 107
8 108
9 109
10 110
11 111
```

In [162…

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

```
0 0 23
1 1 45
2 2 16
```

Bucles while:

In [163…

```python
x = 1
n = 10
while x < n:
    x = x * 2
```

Tipos de datos mas complejos:

- Tupla: tuple ( )
- Lista: list [ ]
- Diccionario: dic
- Conjunto: set
- Array: array

Mutabilidad:

- Un objeto es inmutable si una vez creado no se puede modificar (No tiene funciones que permitan cambiar su contenido)
- Los objetos mutables poseen funciones para cambiar su contenido #### Tipos de datos secuencia:
- Tuplas:
  - Inmutable
  - Ordenada
  - Tipos compuestos
- Cadenas de caracteres
  - Inmutables
  - Similares a una tupla
- Listas
  - Mutable
  - Ordenada
  - range tambien es un tipo de lista

In [167…

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

```
li = ["abc", 34, 4.34, 23]
st = "Hello world"
st = 'Hello world'
st = """Esta es una cadena
con varias lineas comillas triples"""
```

In [168…
```
tu[1]
```

Out[168…
```
'abc'
```

In [169…
```
li[1]
```

Out[169…
```
34
```

In [170…
```
st[1]
```

Out[170…
```
's'
```

```
Se pueden usar indices negativos tambien
 h  o  l  a
 0  1  2  3
-4 -3 -2 -1
```

Tambien se puede trocear:

In [175…
```
tu = (23, 'abc', 4.56, (2,3), 'def')
print(tu[1:-1])
print(tu[1:4])
print(tu[:2])
print(tu[2:])
```

```
('abc', 4.56, (2, 3))
('abc', 4.56, (2, 3))
(23, 'abc')
(4.56, (2, 3), 'def')
```

Copia vs referencia:

In [186…
```
lista = ['hola', 1, 'adios']
print(lista[:])
referencia = lista
print(referencia)

referencia[0] = 2
print("\nLista: ", lista)
print("Referencia: ", referencia)

lista = ['hola', 1, 'adios']
copia = lista[:]

print()

print(lista)
print(copia)

copia[0] = 2
```

```
print("\nLista: ", lista)
print("Copia: ", referencia)
```

```
['hola', 1, 'adios']
['hola', 1, 'adios']

Lista:  [2, 1, 'adios']
Referencia:  [2, 1, 'adios']

['hola', 1, 'adios']
['hola', 1, 'adios']

Lista:  ['hola', 1, 'adios']
Copia:  [2, 1, 'adios']
```

Varoles nulos:

- Listas: t = []
- Cadenas de carateres = ""
- Tuplas = t ()

Operador "in":

In [189…
```
t = [1, 2, 3, 4, 5]
3 in t
4 not in t

a='abcde'
'cd' in a
```

Out[189… True

Operador "+":

In [194…
```
(1,2,3) + (4,5,6)
[1,2,3] + [4,5,6]
"Hello" + " " + "World"
```

Out[194… 'Hello World'

Operador "*":

In [195…
```
(1,2,3) * 3
[1,2,3] * 3
"Hello" * 3
```

Out[195… 'HelloHelloHello'

Las tuplas al ser inmutables son mas rapidas que las listas

In [197…
```
t = (1,2, [3,4])
t[2][1] = 5
t
```

Out[197… (1, 2, [3, 5])

Paso de parametros mutables:

In [201…
```
def f(v,i):
    v[i] += 1

aux = [1,2,3,4]
f(aux, 2)
aux
```

Out[201… [1, 2, 4, 4]

Operaciones sobre listas:

In [205…
```
li = [1,11,3,4,5]
li.append('a')
print(li)
li.insert(2, 'i')
li
```

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

Out[205… [1, 11, 'i', 3, 4, 5, 'a']

+, extend, append:

- (+) crea una lista nueva, con una nueva referencia a memoria
- extend opera directamente sobre la lista li
- append incluye un nuevo elemento al final de la lista

In [223…
```
print("ID antes de extend:", hex(id(li)))
li.extend([9,8,7])
print("ID despues de extend:", hex(id(li)))

li = li + [2]
print("ID despues de +:", hex(id(li)))
print("Se puede observar como crea una nueva referencia a memoria")

li = [2,3,4]
li.append([10,11,12])
print(li)
```

```
ID antes de extend: 0x7f09285e10c0
ID despues de extend: 0x7f09285e10c0
ID despues de +: 0x7f0928626440
Se puede observar como crea una nueva referencia a memoria
[2, 3, 4, [10, 11, 12]]
```

Algunos ejemplos, index, count, remove, reverse, sort:

- li.sort(key = funcion) Ordena la lista usando una comparacion definida por el usuario

In [232…
```
li = ['a','b','c','b']
print(li.index('b'))
print(li.count('b'))
```

```
li.remove('b')
print(li)

li = [5,2,6,8]
print(li)
li.reverse()
print(li)

li.sort()
print(li)

cads = ['hola','12','a','abc']
print(cads)
cads.sort(key = lambda x : len(x))
print(cads)
```

```
1
2
['a', 'c', 'b']
[5, 2, 6, 8]
[8, 6, 2, 5]
[2, 5, 6, 8]
['hola', '12', 'a', 'abc']
['a', '12', 'abc', 'hola']
```

Listas: insercion y borrado:

In [248…

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

print("Borrar elementos")
l[2:3] = []
print(l)
del l[-1]
print(l)

print("\nInsertar elementos")
l[2:2] = [3]
print(l)
l[:0] = [7]
print(l)
```

```
Borrar elementos
[1, 2, 4, 5]
[1, 2, 4]

Insertar elementos
[1, 2, 3, 4]
[7, 1, 2, 3, 4]
```

Detalles del tipo tupla:

- La coma es el operador de creacion de una tupla (no lo es el parentesis) 1, -> (1,)
- Python usa parentesis por claridad (1,) -> (1,)
- Es imprescindible el uso de la coma (1) -> 1
- Existen tuplas vacias: () , tuple()

Devolver valores en funciones:

- Se pueden devolver varios datos
- Si devuelvo varios valores con , (es una tupla)

- Para ello mejor poner() ó [] si quiero que devuelva una lista ##### Para devolver una lista no olvidar [ ]

In [254…

```
def f(x):
    return x, x*2, x**2

aux = f(2)
print(aux)

a,b,c = f(2)
print("a: {}, b: {}, c: {}".format(a,b,c))
```

```
(2, 4, 4)
a: 2, b: 4, c: 4
```

Conversion de lista a tupla y viceversa:

```
* li = list(tu)
* tu = tuple(li)
```

Uso de listas:

In [259…

```
L = [1,2,3,4,5]
suma = 0
for i in range(len(L)):
    suma += L[i]
suma

suma = 0
for i in L:
    suma += i
suma
```

Out[259… 15

In [260…

```
L = [1,2,3,4,5]
L_pares = []
for i in L:
    if i % 2 == 0:
        L_pares.append(i)
L_pares
```

Out[260… [2, 4]

Compresión de listas:

```
Python permite crear una lista con todos los valores en una
unica instrucción, es mas rapido que hacerlo valor a valor.
<lista> = [ <expr> for <var> in ...]

Ejemplo: gradosC = [convertir_F_C(i) for i in gradosF]
```

In [268…

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

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

```
[('a', 1), ('b', 2), ('c', 7)]
[('a', 3), ('b', 6), ('c', 21)]
[3, 6, 21]
```

In [277…

```
y = []
n = 5
h = 2
for i in range(n+1):
    y.append(i*h)

print(y)

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

```
[0, 2, 4, 6, 8, 10]
[0, 2, 4, 6, 8, 10]
```

In [280…

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

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0,
0, 0, 0], [0, 0, 0, 0, 0]]
```

Comparativa:

```
tabla2 = []
for C, F in zip(gradosC, gradosF):
    fil = [C, F]
    tabla2.append(fil)

* Alternativa:
 tabla2 = [[C,F] for C,F in zip(gradosC, gradosF)]
```

In [305…

```
nums = [1,2,3,4]
frutas = ["Manzanas", "Melocotones", "Peras", "Platanos"]
print([(i,f) for i,f in zip(nums, frutas)])
print("")
print([(i,f) for i in nums for f in frutas])

print("")
list1 = [3,4,5]
mult = [item * 3 for item in list1]
print(mult)

print("")
palabras = ["esto","es","una","lista","de","palabras"]
items = [ palabra[0] for palabra in palabras]
print(items)
```

```
[(1, 'Manzanas'), (2, 'Melocotones'), (3, 'Peras'), (4, 'Platanos')]

[(1, 'Manzanas'), (1, 'Melocotones'), (1, 'Peras'), (1, 'Platanos'), (2, 'Man
zanas'), (2, 'Melocotones'), (2, 'Peras'), (2, 'Platanos'), (3, 'Manzanas'),
(3, 'Melocotones'), (3, 'Peras'), (3, 'Platanos'), (4, 'Manzanas'), (4, 'Melo
cotones'), (4, 'Peras'), (4, 'Platanos')]
```

```
[9, 12, 15]

['e', 'e', 'u', 'l', 'd', 'p']
```

Operaciones adicionales:

```
* enumerate(L) devuelve un par (indice, elemen)
    for i, c in enumerate(L):
        L[i] = i + c
* sorted(L) ordena la lista
* reversed(L) inversa la lista
```

In [325…

```
L = [[0,0]] * 5
print(L)
L[2][0] = 7
print(L)
print("Esto ocurre porque es la misma referencia!\n")

L = ["abc"] * 3
print(L)
L = [L] * 3
print(L)
L[2][0] = 'cde'
print(L)
```

```
[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
[[7, 0], [7, 0], [7, 0], [7, 0], [7, 0]]
Esto ocurre porque es la misma referencia!

['abc', 'abc', 'abc']
[['abc', 'abc', 'abc'], ['abc', 'abc', 'abc'], ['abc', 'abc', 'abc']]
[['cde', 'abc', 'abc'], ['cde', 'abc', 'abc'], ['cde', 'abc', 'abc']]
```

Conversion de str a list:

In [328…

```
l = []
s = "Hola"
l += s
l
```

Out[328…

```
['H', 'o', 'l', 'a']
```

Resumen de Operaciones sobre listas:

1. a = [] crea una lista vacia
2. A = [1.3, 4, 'script.py'] asigna un valor
3. a.append(elem) añade un elemento al final
4. a + [1,3] concatena dos litas y da una nueva referencia
5. a[3] accede a un elemento
6. a[-1] accede al ultimo elemento
7. a[1:3] devuelve una sublista
8. del a[3] elimina un elemento
9. a.remove(4.4) elimina un elemento con ese valor
10. a.index('run.py') devuelve la posicion donde se encuentra dicho elemento (ValueError si no esta)
11. a.find('run.py') devuelve la posicion donde se encuentra un elemento (-1 si no esta)
12. a.count(v) numero de repeticiones de un elemento

13. len(a) numero de elementos
14. min(a) minimo elemento
15. max(a) maximo elemento
16. sum(a) suma todos los elementos
17. a.sort() ordena una lista
18. as = sorted(a) devuelve una version ordenada
19. b[3][0][2] indexado anidado
20. isinstance(a, list) devuelve True si es una lista
21. l1.copy() devuelve una copia de la lista

Ejemplo string + list

- readline() Lee una linea completa linea = sys.stdin.readline()
- readlines() Lee una serie de lineas, cada linea va en una posicion de la lista. texto = sys.stdin.readlines()
- split() divide una cadena y devuelve una lista de subcadenas palabras = cad.split() datos = cad.split("es")
- strip() elimina separadores de ambos extremos

In [337…

```
palabras = 'el gato sentado sobre el coche'.split()
print(palabras)
print(' el gato '.strip())
print(' el gato '.lstrip())
```

```
['el', 'gato', 'sentado', 'sobre', 'el', 'coche']
el gato
el gato
```

In [350…

```
primos = [2,3,5,7,11,13,17,19]
print([p for p in primos if p % 3 > 1])
print([[0,0,0] for x in range(2,5)])
print([93 % p for p in primos if 93 % p != 0])

matriz = [[1,2,3],[4,5,6],[7,8,9]]
print([[fil[i] for fil in matriz] for i in range(len(matriz))])

[[0 for i in range(5)] for j in range(6)]
```

```
[2, 5, 11, 17]
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
[1, 3, 2, 5, 2, 8, 17]
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

Out[350…

```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

Tipos de datos asociativos:

- Conjunto: set {} Coleccion ordenada de datos de distinto tipo, debe ser de tipo "hasheable" (se puede aplicar una funcion hash sobre el)
- Las listas no pueden pertenecer a un set, ya que: unhashable type: 'list'
- Diccionario: dic

In [356…

```
s1 = {1,3,5,4,3,2,1}
```

```
print(s1)

s2 = {1,'a'}
print(s2)

s1 = {(1,'a'),(2,'b')}
print(s1)

conj = {(3,5),(6,2),"Hola"}
print(conj)
```

```
{1, 2, 3, 4, 5}
{1, 'a'}
{(1, 'a'), (2, 'b')}
{(6, 2), 'Hola', (3, 5)}
```

Tipos de datos conjunto:

Existe set como version mutable y frozenset e inmutableSet como inmutables

In [363…

```
dias_semana = {'lun','mar','mie','jue','vie'}
print(dias_semana)

dias_semana = set(('lun','mar','mie','jue','vie'))
print(dias_semana)

dias_semana.add('sab')
print(dias_semana)

'mie' in dias_semana

dias_semana.remove('mie')
```

```
{'vie', 'mar', 'mie', 'lun', 'jue'}
{'vie', 'mar', 'mie', 'lun', 'jue'}
{'vie', 'mar', 'mie', 'lun', 'sab', 'jue'}
```

Operaciones sobre conjuntos

- add(x) añade un elemento
- remove(x) si no esta el elemento genera una excepcion KeyError:8
- discard(x) elimina elemento pero si no esta no genera ninguna excepcion
- pop() devueve y elimina un elemento aleatorio
- copy() copia todos los elementos
- clear() elimina todos los elementos
- Incluye:
  - union
  - intersection
  - difference
  - symmetric_difference

In [371…

```
s1 = {1,3,5,7,4,3,2,1}
s2 = {2,4,6,8,6,4,3,2,1}

union_s1_s2 = s1.union(s2)
print(union_s1_s2)
```

```
intersect_s1_s2 = s1.intersection(s2)
print(intersect_s1_s2)

diff = s1.difference(s2)
print(diff)

s_diff = s1.symmetric_difference(s2)
print(s_diff)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{1, 2, 3, 4}
{5, 7}
{5, 6, 7, 8}
```

Iterar sobre un conjunto:

In [376…

```
s1 = {1,3,5,7,4,3,2,1}
for letra in set("zbatgdab"):
    print(letra)

print("")

for i in s1:
    print(i)
```

```
z
b
g
t
a
d

1
2
3
4
5
7
```

Compresion de conjuntos:

De forma similar a las listas

In [398…

```
a = {x for x in 'abracadabra' if x not in 'abc'}
print(a)

vocalesPalabra = {x for x in 'esto es una prueba' if x in 'aeiou'}
print(vocalesPalabra)

L = [34,45,87,90,32,4]
mayores50 = {x for x in L if x > 50}
print(mayores50)
```

```
{'r', 'd'}
{'u', 'a', 'e', 'o'}
{90, 87}
```

Tipo de dato dic:

```
Tipo diccionario: guarda pares (clave, valor)
Poder acceder de forma eficiente a través de la clave
Las claves deben ser inmutables, los valores no tienen
restricciones
* Se pueden guardar claves y valores de distintos tipos
* Operaciones: buscar, borrar, modificar
* Conocidos como tablas hash o arrays asociativos
* Se usa {} separando clave y valor con : y cada par con ,
* Los diccionarios no tienen ningun orden preestablecido
```

Asignacion, acceso y actualizacion:

In [409…

```
datos = {'nombre':'Pepe', 'edad':40}
datos = dict(nombre='Pepe', edad=40)

print(datos)

print(datos['nombre'])
print(datos['edad'])

datos['nombre'] = "Francisco"
datos['dni'] = "77022.."

print(datos)
```

```
{'nombre': 'Pepe', 'edad': 40}
Pepe
40
{'nombre': 'Francisco', 'edad': 40, 'dni': '77022..'}
```

Asignacion directa:

In [419…

```
ciudades=['Madrid', 'Granada', 'Valencia']
temps_ciudad = [29,30,28]

temps = dict(zip(ciudades, temps_ciudad))
print(temps)
```

```
{'Madrid': 29, 'Granada': 30, 'Valencia': 28}
```

Borrar datos:

In [427…

```
temps = dict(zip(ciudades, temps_ciudad))
del temps['Valencia']
print(temps)

temps.pop('Madrid')
print(temps)

temps['Granada'] = None
print(temps)
print(temps)


temps.clear()
print(temps)
```

```
{'Madrid': 29, 'Granada': 30}
{'Granada': 30}
{'Granada': None}
```

```
{'Granada': None}
{}
```

Iterar y buscar:

In [441…

```
temps = dict(zip(ciudades, temps_ciudad))

for i in temps:
    print(i, temps[i])

print("")

for i,x in enumerate(temps):
    print(i, x, temps[x])


if 'Granada' in temps:
    print("\nTemperatura Granada: ", temps['Granada'])
else:
    print("\nNo hay temperaturas para Granada")
```

```
Madrid 29
Granada 30
Valencia 28

0 Madrid 29
1 Granada 30
2 Valencia 28

Temperatura Granada:  30
```

Obtener claves y valores:

In [454…

```
print(temps.keys())
print(temps.values())
print("")
print(sorted(temps.keys()))
print(sorted(temps.values()))


print("\nOrdenar los elementos: ")
for i in sorted(temps.keys()):
    temp = temps[i]
    print(i, temp)
```

```
dict_keys(['Madrid', 'Granada', 'Valencia'])
dict_values([29, 30, 28])

['Granada', 'Madrid', 'Valencia']
[28, 29, 30]

Ordenar los elementos:
Granada 30
Madrid 29
Valencia 28
```

Obtener claves y valores (II):

In [461…

```
for ciudad, temp in temps.items():
    print(ciudad,':',temp)
```

```
print("")

for ciudad, temp in sorted(temps.items()):
    print(ciudad,':',temp)
```

```
Madrid : 45
Granada : 30
Valencia : 28
Alamar : 34

Alamar : 34
Granada : 30
Madrid : 45
Valencia : 28
```

In [466…

```
estaciones = {'primavera':
                  {'marzo','abril','mayo'},
              'verano':
                  {'junio','julio','agosto'}
             }

for i,j in estaciones.items():
    print(i,':',j)

print("")

for i,j in estaciones.items():
    for k in j:
        print(k)
```

```
primavera : {'mayo', 'abril', 'marzo'}
verano : {'julio', 'agosto', 'junio'}

mayo
abril
marzo
julio
agosto
junio
```

Numero de parametros variable en una funcion:

Se indica con * delante del parametro

In [476…

```
def func(*datos):
    print("")
    for i in datos:
        print(i)
        if type(i) == list and len(i) > 1:
            for j in i:
                print(j)

func(5)
func(5,6,7)
func([1,'a','c'],['adios','h'])
```

```
5

5
6
```

```
7

[1, 'a', 'c']
1
a
c
['adios', 'h']
adios
h
```

Parametros con nombre en una funcion:

Se indica con ** delante del parametro

In [472…

```
def func(**datos):
    for i, j in datos.items():
        print(i,j)

func(pepe=1234,juan=343)
```

```
pepe 1234
juan 343
```

Tipo de dato array:

- Todos los datos del mismo tipo: modulo array
- Se crean indicado el tipo de dato #### array(tipo, datos)

codigo	tipo C	Tipo Python	Tamaño min bytes
'c'	char	character	1
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode char	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	long	2
'l'	signed long	int	4
'L'	unsigned long	long	4
'f'	float	float	4
'd'	double	float	8

Tipo de dato fichero:

- Un fichero es una secuencia bytes almacenada en memoria externa
- Python maneja archivos como secuencias de caracteres. Solo se leen y escriben cadenas
- Es necesario abrir fichero para usarlo y despues cerrarlo #### Operaciones sobre ficheros:
  - open(nombre, modo)
    - f = open("datos.txt")
  - f.close()
  - os.path.isfile(nombre) comprobar si existe un fichero
  - os.stat(nombre) comprueba permisos
  - f.read(num) lee un numero de bytes de un archivo, si no se indica se lee hasta el final
  - f.readline() lee una linea completa
  - f.readlines() Leer todo el archivo por lineas (en una lista)

- seek(num, donde) donde puede ser 0:inicio, 1:actual, 2:final (se posiciona sobre un byte concreto
- tell() conocer la posicion actual del fichero
- for linea in fich: (itera sobre las lineas del fichero (como una lista)
- with open("fichero.txt", "rb") as f: (como un bloque)

Ficheros de texto:

In [481…

```
f = open("datos.txt")
for line in f:
    print(line)
f.close()
```

```
hola

esto es una prueba

de python

encantado

no olvides suscribirte

fin
```

Ficheros binarios:

In [492…

```
with open("datos.txt", "rb") as f:
    byte = f.read(1)
    while byte:
        print(byte)
        byte = f.read(1)
```

```
b'h'
b'o'
b'l'
b'a'
b'\n'
b'e'
b's'
b't'
b'o'
b' '
b'e'
b's'
b' '
b'u'
b'n'
b'a'
b' '
b'p'
b'r'
b'u'
b'e'
b'b'
b'a'
b'\n'
b'd'
b'e'
b' '
b'p'
```

```
b'y'
b't'
b'h'
b'o'
b'n'
b'\n'
b'e'
b'n'
b'c'
b'a'
b'n'
b't'
b'a'
b'd'
b'o'
b'\n'
b'n'
b'o'
b' '
b'o'
b'l'
b'v'
b'i'
b'd'
b'e'
b's'
b' '
b's'
b'u'
b's'
b'c'
b'r'
b'i'
b'b'
b'i'
b'r'
b't'
b'e'
b'\n'
b'f'
b'i'
b'n'
b'\n'
```

Consultas:

* name devuelve el nombre del fichero
* closed devuelve si el fichero esta cerrado
* mode devuelve el modo de apertura

Leer datos y ponerlos en dos listas de numeros reales:

In [494…

```
x = []; y = []
lineas = open("datos1.txt")
for linea in lineas:
    xval, yval = linea.split()
    x.append(float(xval))
    y.append(float(yval))
lineas.close()

print(x)
print(y)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
```

rstrip() para eliminar salto de linea, espacios al principio y al final " banana " -> "banana"

In [498…

```
codigos = {}
f = open("codigos.txt", "r")

for line in f:
    [codigo, lugar] = line.rstrip().split(' ')
    codigos[codigo] = lugar
f.close()
print(codigos)
```

```
{'1234': 'mi_casa', '1235': 'mi_perro', '1236': 'mi_loco'}
```

Guardar datos:

* write(cadena) escribir cadenas, cada cadena se guarda a continuacion de la anterior(sin separadores)
    * f.write('Hola'); f.write('adios') -> Holaadios
* f.writelines(secuencia) varias lineas al mismo tiempo

In [2]:

```
with open("salida.txt", "w") as f:
    f.write("Esto")
    f.write(" es ")
    f.write("una prueba")

L = ['esto', ' es ', 'una prueba']
with open("salida2.txt", "w") as f:
    f.writelines(L)
```

Escribir datos formateados:

In [503…

```
with open("cupra.txt", "w") as f:
    for i in range(100, 200):
        f.write('{:3d} {:7d}'.format(i, i**2))
        f.write('\n')
```

Entrada y salida estandar:

Modulo sys

* import sys
* sys.stdin -> Entrada estandar, abierto para lectura
* sys.stdout -> Salida estandar, abierta para escritura
* sys.stderr -> Salida de error, abierta para escritura

VER EJEMPLO entrada_salida_estandar.py

In [507…

```
import sys
for line in sys.stdin:
    sys.stdout.write(line)
```

```
line = sys.stdin.readline()
while line:
    sys.stdout.write(line)
    line = sys.stdin.readline()
```

Informacion sobre el sistema

* sys.path devuelve una lista de strings con el path del sistema
* sys.argv Lista de argumentos del script
* sys.maxint el entero mas grande que procesa la CPU
* sys.platform devuelve una cadena que identifica la plataforma
* sys.version version de python

In [510…

```
import sys
print(sys.path)
print(sys.argv)
print(sys.platform)
print(sys.version)
```

```
['/home/cazz', '/home/cazz/Documents/anaconda3/lib/python38.zip', '/home/caz
z/Documents/anaconda3/lib/python3.8', '/home/cazz/Documents/anaconda3/lib/pyt
hon3.8/lib-dynload', '', '/home/cazz/.local/lib/python3.8/site-packages', '/h
ome/cazz/Documents/anaconda3/lib/python3.8/site-packages', '/home/cazz/Docume
nts/anaconda3/lib/python3.8/site-packages/IPython/extensions', '/home/cazz/.i
python']
['/home/cazz/Documents/anaconda3/lib/python3.8/site-packages/ipykernel_launch
er.py', '-f', '/home/cazz/.local/share/jupyter/runtime/kernel-04bf4360-fb41-4
0fc-ac58-7eefd279dc12.json']
linux
3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0]
```

Ejemplo ficheros csv

* Leer datos separados por comas

In [10]:

```
with open("datos.csv", "r") as f:
    datos = f.readline().split(',')

import csv
with open("datos.csv", "r") as f:
    data = list(tuple(rec) for rec in csv.reader(f, delimiter=","))
print(data)
```

```
[('hola', '1', 'que', 'tal'), ('esto', 'es', '4', 'prueba')]
```

Generar datos en la web

In [3]:

```
import urllib.request
f = urllib.request.urlopen('http://www.python.org/')
print(f.read(100).decode('utf-8'))
```

```
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]
-->
<!-
```

Operaciones:

- Leer datos de la web
- Leer pagina de la web
- Enviar datos en la web
- Autentificacion HTTP
- Modulo requests

In [13]:
```
import requests
page = requests.get('https://pbaquero.webs.ull.es/html/miprimerapagina.htm')
print(page)

page.content
```

```
<Response [200]>
```

Out[13]:
```
b'<html>\n\n<head>\n<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
\n<meta name="ProgId" content="FrontPage.Editor.Document">\n<meta http-equiv
="Content-Type" content="text/html; charset=windows-1252">\n<title>Una prueba
HTML</title>\n</head>\n\n<body>\n\n<h2>Mi primera p\xe1gina</h2>\n<h3> 
</h3>\n<h3>Hagamos un ejemplo muy b\xe1sico</h3>\n<div>\n   <p><i>\n  <b
>1. Qu\xe9 tal te parece si creamos un fichero y lo llamamos:</b></i></p>\n
<p align="center">\n  <b>prueba.html</b></div>\n<pre><b><i>2.<font face="Time
s New Roman" size="3">Ahora escribimos:</font></i></b></pre>\n<blockquote>\n
<blockquote>\n<pre>&lt;html&gt;\n&lt;head&gt;\n    &lt;title&gt;Esta es mi pr
imera pagina&lt;/title&gt;\n&lt;/head&gt;\n&lt;body&gt;\n    &lt;h1&gt;Esto e
s un encabezado&lt;/h1&gt;\n    &lt;p&gt;Y esto es un parrafo, donde podemos
escribir todo el rollo que se nos ocurra.\n&lt;/body&gt;\n&lt;/html&gt;</pre>
\n  </blockquote>\n</blockquote>\n<p>Si quieres ver como se saldr\xeda puedes
verlo <a href="ejemplo/ejemplo1.html">\naqu\xed</a>.</p>\n<p>Ah! No pongas ac
entos a las palabras, HTML no es muy listo y no los entiende \nsi no se los e
xplicas. Si tienes curiosidad coge el c\xf3digo de una p\xe1gina con \npalabr
as con acentos y mira como se hace. </p>\n<p> </p>\n<p><b><i>3. Ahora vo
y a explicar que en que consisten lo que se ha escrito. </i>\n</b></p>\n<p>a.
En HTML lo que queramos que se haga con un texto, una presentaci\xf3n o \ncua
lquier cosa debe estar encerrado entre dos etiquetas:</p>\n<blockquote>\n  <b
lockquote>\n<pre>&lt;etiqueta par\xe1metros&gt; lo que sea &lt;/etiqueta&gt;
</pre>\n  </blockquote>\n<p>No siempre es necesario cerrar las etiquetas. </p
>\n</blockquote>\n<p> </p>\n<p>b. Para indicar que se trata de un docume
nto HTML el c\xf3digo lo cerramos \nentre:</p>\n<blockquote>\n  <blockquote>
\n<pre>&lt;html&gt; </pre>\n<pre>... </pre>\n<pre>&lt;/html&gt;</pre>\n  </bl
ockquote>\n</blockquote>\n<p> </p>\n<p>c. La cabecera de un documento HT
ML se encierra entre:</p>\n<blockquote>\n  <blockquote>\n    <pre>&lt;head&g
t;\n  ...\n&lt;/head&gt;</pre>\n  </blockquote>\n<p>Lo que escribe dentro de
la cabecera no se ve dentro de la presentaci\xf3n y \ncontiene informaci\xf3n
sobre el documento como puede ser el t\xedtulo, o otras muchas \ncosas m\xe1
s. En nuestro ejemplo hemos puesto el t\xedtulo del documento,  que \nco
mo puedes comprobar no aparece dentro de la p\xe1gina sino como t\xedtulo de
la \nventana donde aparece la p\xe1gina.</p>\n  <blockquote>\n<p><font face
="Courier New" size="2">&lt;title&gt;Esta es mi primera pagina&lt;/title&gt;
</font></p>\n  </blockquote>\n</blockquote>\n<p>d. El cuerpo del documento se
encierra entre:</p>\n<blockquote>\n  <blockquote>\n<p><font face="Courier Ne
w" size="2">&lt;body&gt; </font></p>\n<p><font face="Courier New" size
="2">... </font></p>\n<p><font face="Courier New" size="2">&lt;/body&gt;</fon
t></p>\n  </blockquote>\n</blockquote>\n<p>Dentro del cuerpo he escrito:</p>
\n<blockquote>\n  <blockquote>\n<pre>&lt;h1&gt; ... &lt;/h1&gt;</pre>\n  </bl
ockquote>\n</blockquote>\n<p> <tt>\n&lt;h1&gt;</tt> (cabecera 1) aumenta el t
ama\xf1o del tipo de letra \npara que se resalte. </p>\n<p>Si queremos separa
r el texto anterior con un espacio vertical correspondiente \nescribimos &lt;
p&gt;, en este caso no hace falta cerrar la etiqueta:</p>\n<pre>&lt;P&gt;Esta
es la primera vez que hago una p\xe1gina y ya me considero un fiera. </pre>\n
<p>A continuaci\xf3n, se ver\xe1 como se da formato al texto, pero sin compli
carnos la \nvida!.</p>\n\n</body>\n\n</html>'
```

Guardar estructuras de datos:

- Abrimos el dichero y pasamos los datos a str, cerramos el fichero

- Lo mismo podemos hacer para leeer estructuras de datos
- Ejemplo:

In [9]:
```
lista = ['text1', 'text2']
a = [[1.3, lista], 'texto']
f = open('tmp.dat', 'w')

f.write(str(a))
f.close()
```

In [15]:
```
f = open('tmp.dat', 'r')
datos = eval(f.readline())
print(datos)
```

```
[[1.3, ['text1', 'text2']], 'texto']
```

Guardar datos en binario y recuperarlos

- Con ayuda del modulo struct
- ord() pasa a ordinal, chr() pasa a char

In [16]:
```
import struct
st = struct.pack(<formato>, <dato1>, <dato2>, ...)
(Usar unpack para desempaquetarlos)
```

Guardar datos completos: (Para guardar variables)

- Se debe usar el modulo pickle

In [24]:
```
import pickle
l = ['hola', '5']

pickle.dump(l, open('prueba.txt', 'wb'))
l2 = pickle.load(open('prueba.txt', 'rb'))
print(l2)
print(type(l2))
```

```
['hola', '5']
<class 'list'>
```

Trabajar con datos en el disco:

- Permite manejar datos sin cargarlos

In [27]:
```
import shelve
database = shelve.open('datos_shelve.bin')
database['a1'] = 1234
database['a2'] = 543
database['a3'] = 234
database['a123'] = (1234, 543, 234)

if 'a1' in database:
    a1 = database['a1']
    print(a1)
```

```
del database['a2']
database.close()
```

```
1234
```

Programación Funcional:

- Python usa dunciones de orden superior, cuyos parametros son, a su vez, funciones:
- map
- reduce
- filter #### Las funciones lambda se suelen utilizar como parametros

Map:

```
map(<funcion>, <objeto iterable>, ...)
La función se aplica a cada uno de los elementos del objeto
Hay que hacer un cambio de tipo a list() porque devuelve un iterador
```

In [29]:

```
nums = [0,4,7,2,1,0,9,3,5,6,8,0,3]
nums = list(map(lambda x: x % 5, nums))
print(nums)
```

```
[0, 4, 2, 2, 1, 0, 4, 3, 0, 1, 3, 0, 3]
```

In [36]:

```
def cuadrado(x):
    return x*x

print(list(map(cuadrado, range(10,20))))

lista = [(x*x) for x in range(10,20)]
print(lista)
```

```
[100, 121, 144, 169, 196, 225, 256, 289, 324, 361]
[100, 121, 144, 169, 196, 225, 256, 289, 324, 361]
```

Muchos programadores de python prefieren usar compresión de listas

In [39]:

```
print([2*x+1 for x in[10,20,30]])
print(list(map(lambda x: 2*x+1, [10,20,30])))
```

```
[21, 41, 61]
[21, 41, 61]
```

In [44]:

```
sec = range(8)
l = list(map(lambda x,y: (x,y), sec, list(map(lambda x: x*x, sec))))
print(l)
```

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49)]
```

In [48]:

```
a = [1,2,3,4]
b = [17,12,11,10]
c = [-1,-4,5,9]

print(list(map(lambda x,y: x+y, a,b)))
print(list(map(lambda x,y,z: x+y+z, a,b,c)))
```

```
[18, 14, 14, 14]
[17, 10, 19, 23]
```

Lista de puntos 3D, lista con distancias, usando programación funcional

In [50]:

```
from math import sqrt
puntos = [(2,1,3),(5,7,-3),(2,4,0),(9,6,8)]
def distancia(punto):
    x,y,z = punto
    return sqrt(x**2 + y**2 + z**2)

distancias = list(map(distancia, puntos))
print(distancias)
```

```
[3.7416573867739413, 9.1104335791443, 4.47213595499958, 13.45362404707371]
```

Filter:

```
filter(<funcion>, <objeto iterable>, ...)
La función se aplica a cada uno de los elementos del objeto
Hay que hacer un cambio de tipo a list() porque devuelve un iterador
```

In [53]:

```
nums = [0,4,7,2,1,0,9,3,5,6,8,0,3]
nums = list(filter(lambda x: x != 0, nums))
print(nums)
```

```
[4, 7, 2, 1, 9, 3, 5, 6, 8, 3]
```

In [54]:

```
def par(x):
    return x % 2 == 0

cuadrado = lambda x: x*x

list(map(cuadrado, filter(par, range(10,20))))
```

Out[54]:

```
[100, 144, 196, 256, 324]
```

In [73]:

```
from math import pi
from math import isnan
NaN = float("nan")
puntuaciones = [
                [NaN,12,.5,78,pi],
                [2,13,.5,.7,pi/2],
                [2,NaN,.5,78,pi],
                [2,14,.5,39,1,-pi]
               ]

def es_NaN(respuestas):
    for num in respuestas:
        if isnan(float(num)):
            return False
        return True

valid = list(filter(es_NaN, puntuaciones))
print(valid)
```

```
print("MEJOR OPCION:")
list(filter(lambda x: NaN not in x, puntuaciones))
```

```
[[2, 13, 0.5, 0.7, 1.5707963267948966], [2, nan, 0.5, 78, 3.141592653589793],
[2, 14, 0.5, 39, 1, -3.141592653589793]]
MEJOR OPCION:
```

Out[73]:
```
[[2, 13, 0.5, 0.7, 1.5707963267948966],
 [2, 14, 0.5, 39, 1, -3.141592653589793]]
```

Reduce:

reduce(<funcion>, <objeto iterable>,[inicializador])
Se aplica a cada elemento del objeto iterable junto con el resultado hasta ese momento, acumlando los resultados.
Funcion debe tener dos argumentos
Inicialiador si existe se usara como primer arg para aplicar la función
Importar el modulo from functools import reduce

In [4]:
```
from functools import reduce

def mas(x, y):
    return (x + y)

lista = [1,2,3,4,5,6]
print(reduce(mas, lista))

lista = ['h','e','l','l','o']
print(reduce(mas, lista))
```

```
21
hello
```

In [79]:
```
nums = [1,2,3,4,5,6,7,8]
nums = list(reduce(lambda x,y: (x,y), nums))
print(nums)
```

```
[((((((1, 2), 3), 4), 5), 6), 7), 8]
```

Calcular media de una lista de numeros usando programación funcional:

In [81]:
```
nums = [92,27,63,43,88,38]

media = reduce(lambda x,y: x+y, nums) / len(nums)
print(media)
```

```
58.5
```

In [84]:
```
f = lambda a,b: a if (a > b) else b
reduce(f, [47,11,42,102,13])
```

Out[84]:
```
102
```

Map reduce:

In [12]:
```
email = ['the', 'this', 'annoy', 'the', 'the']
def inEmail(x):
    if(x == 'the'):
        return 1
    else:
        return 0

print(list(map(inEmail, email)))
reduce((lambda x, xs: x+xs), map(inEmail, email))
```

```
[1, 0, 0, 1, 1]
```

Out[12]:
```
3
```

Manejo de errores:

- try/except
- try/finally -> se realiza ocurra o no la excepcion
- raise -> genera una excepción de de forma manual
- assert -> genera excepcion de forma concidicional

In [17]:
```
c = [1,2,3,"5"]

suma_de_c = 0
for numero in c:
    if isinstance(numero, int):
        suma_de_c += numero

print(suma_de_c)

suma_de_c = 0

for n in c:
    try:
        suma_de_c += numero
    except TypeError:
        pass

print(suma_de_c)
```

```
6
0
```

In [25]:
```
def division(x,y):
    try:
        if y == 0:
            raise TypeError
        return x/y
    except TypeError as error:
        print("No se puede dividir entre 0")


division(6,0)
```

```
No se puede dividir entre 0
```

Generadores:

In [32]:
```
lista_uno = [1,2,3,4,5,6]
def pares(lista):
    for i in lista:
        if i % 2 == 0:
            yield i

nums = pares(lista_uno)
print(nums)

for i in nums:
    print(i)
```

```
<generator object pares at 0x7f373d515cf0>
2
4
6
```

In [88]:
```
from time import gmtime, strftime
def generador():
    while True:
        yield strftime("%a, %d %b %Y %H:%M:%S + 0000", gmtime())

generadorInstancia = generador()
next(generadorInstancia)
```

Out[88]:
```
'Mon, 02 Nov 2020 14:31:28 + 0000'
```

Cálculo simbolico:

In [7]:
```
from sympy import *
x = symbols('x')
a = Integral(cos(x) * exp(x), x)
Eq(a, a.doit())
```

Out[7]:
$$\int e^{x} \cos\left(x\right) dx = \frac{e^{x} \sin\left(x\right)}{2} + \frac{e^{x} \cos\left(x\right)}{2}$$

In [8]:
```
diff(sin(x)*exp(x)*cos(x))
```

Out[8]:
$$-e^{x} \sin^{2}\left(x\right) + e^{x} \sin\left(x\right) \cos\left(x\right) + e^{x} \cos^{2}\left(x\right)$$

Tema 2: Computación numérica en Python y visualización de datos científicos

Indice:

* Matplotlib
* SciPy

In [9]:
```
import numpy as np
```

Tipo de dato ndarray

- Vector N-dimensional
- sin(x) equivalente a [sin(x[i]) for i in range(N)]
- x+y equivalente a [x[i] + y[i] for i in range(N)]

Tipos NumPy (dtype)

```
boolean 'b'
integer 'i'
unsigned integer 'u'
float 'f'.'d'
complex 'c'
strings 'S','a'
Object
Records
```

Arrays numpy

- Todas las filas la misma longitud
- Todas las entradas deben ser del mismo tipo
- Mucho mas eficiente que las listas

In [17]:
```
import numpy as np
a = np.array([[1,2,3],[4,5,6]], float)
print(a)

print("Dimensiones:", a.shape)
print("Numero de elementos:", a.itemsize)
print(a.dtype, a.dtype.type)
print(type(a[0,0]))
```

```
[[1. 2. 3.]
 [4. 5. 6.]]
Dimensiones: (2, 3)
Numero de elementos: 8
float64 <class 'numpy.float64'>
<class 'numpy.float64'>
```

In [33]:
```
a = np.array([0,1,2,3])
print("Tipo de array:", type(a))
print("Tipo de array dtype:", a.dtype)
print("Con esto sabemos que un elemento tiene 64 bits")
print("Bytes de un elemento:", a.itemsize)
print("Dimension del vector:", a.shape)
print("Numero de elementos:", a.size)
print("Numero de bytes en total:", a.nbytes)
print("Numero de dimensiones:", a.ndim)
b = a.copy()
```

```
Tipo de array: <class 'numpy.ndarray'>
```

```
Tipo de array dtype: int64
Con esto sabemos que un elemento tiene 64 bits
Bytes de un elemento: 8
Dimension del vector: (4,)
Numero de elementos: 4
Numero de bytes en total: 32
Numero de dimensiones: 1
```

Buscar informacion:

In [14]:
```
#np.lookfor('create array')
```

Creación de vectores nulos:

In [51]:
```
a = zeros(3)
print(a)

Z = zeros(3, 1)
print(Z)

a = ones(3)
print(a)
```

```
Matrix([[0, 0, 0], [0, 0, 0], [0, 0, 0]])
Matrix([[0], [0], [0]])
Matrix([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
```

Arrays con secuencias:

- linspace(a,b,n) genera n valores equispaciados empezando en a y terminando en b
- arange(a,b,x,tipo) genera un array con valores entre a y b (sin incluir b) de x en x valores
- reshape(n matrices, filas, columnas)
- shape(filas,columnas)

In [63]:
```
x = np.linspace(-5,5,11)
print(x)

a = np.r_[-5:5:11j]
print(a)
```

```
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
```

In [81]:
```
a = np.arange(-5,5,1,float)
print(a)

a = np.arange(24).reshape(2,3,4)
print(a)
```

```
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.]
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Eficiencia (timeit)

In [47]:
```
l = range(1000)
%timeit [i**2 for i in l]

a = np.arange(1000)
%timeit a**2
```

```
192 µs ± 6.03 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
968 ns ± 5.63 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Creacion a partir de una funcion:

In [120…
```
def mi_func(i,j):
    return (i+1)*(j+4-i)

#Un array 3x6 con a[i,j] = myfunc(i,j):
a = np.fromfunction(mi_func, (3,6))
print(a)
```

```
[[ 4.  5.  6.  7.  8.  9.]
 [ 6.  8. 10. 12. 14. 16.]
 [ 6.  9. 12. 15. 18. 21.]]
```

Creacion de arrays aleatorios:

- np.random.uniform(inicio,fin,elementos)
- reshape(z,y,x)

In [141…
```
a = np.random.uniform(0,1,6).reshape(1,3,2)
print(a)
print(a.shape)

a = np.random.uniform(0, 10, (3,3))
print("\n",a)
print(a.shape)
```

```
[[[0.1987719  0.44395362]
  [0.92476734 0.67143436]
  [0.0815231  0.57265925]]]
(1, 3, 2)

 [[1.20662198 9.99963724 9.97380347]
 [6.94712143 3.73879844 4.01510789]
 [9.25099413 2.44858842 6.6401268 ]]
(3, 3)
```

Conversión a lista:

- Dos formas la mas eficiente a.tolist()

In [145…
```
list(a)
```

Out[145…
```
[array([1.20662198, 9.99963724, 9.97380347]),
 array([6.94712143, 3.73879844, 4.01510789]),
 array([9.25099413, 2.44858842, 6.6401268 ])]
```

In [144…
```
a.tolist()
```

Out[144…
```
[[1.206621977560015, 9.999637237786395, 9.973803472503109],
 [6.947121427369083, 3.738798440981772, 4.0151078896665435],
 [9.250994130048472, 2.4485884177273487, 6.640126803938178]]
```

Conversión a numpy:

- Funcion asarray()

In [154…
```
#obj = np.asarray(obj)
#obj = np.asarray(obj, order='Fortran')

def f(sec):
    a = np.asarray(sec)

f([1,2,3])
f(zeros(10))
```

Asignación de elementos:

In [164…
```
a = zeros(4,1)
a.fill(-4.8)
print(a)
a[1] = 5.8
print(a)
```

```
Matrix([[-4.8], [-4.8], [-4.8], [-4.8]])
Matrix([[-4.8], [5.80000000000000], [-4.8], [-4.8]])
```

Eficiencia:

In [297…
```
import numpy as np
import time

def bucles_version(n):
    t1 = time.time()
    X = range(n)
    Y = range(n)
    Z = []
    for i, j in zip(X,Y):
        Z.append(i+j)
    return time.time()-t1

def compresion_version(n):
    t1 = time.time()
    X = range(n)
    Y = range(n)
    Z = [i + j for i,j in zip(X,Y)]
    return time.time()-t1

def numpy_version(n):
    t1 = time.time()
    X = np.arange(n)
    Y = np.arange(n)
    Z = X + Y
    return time.time()-t1

n = 10000000
```

```
print(bucles_version(n))
print(compresion_version(n))
print(numpy_version(n))
```

```
0.9794528484344482
0.7014822959899902
0.057921648025512695
```

Vectores multidimensionales:

In [175…
```
a = np.array([[0,1,2,3],
              [10,11,12,13]
             ])

a.shape
a.size
a.ndim
a[1]
```

Out[175… `array([10, 11, 12, 13])`

Cambiar dimensiones:

In [183…
```
a = np.array([0, 1.2, 4, -9.1, 5, 8])
print(a.size)
a.reshape(2,3)
```

```
6
```

Out[183…
```
array([[ 0. ,  1.2,  4. ],
       [-9.1,  5. ,  8. ]])
```

Trocear un vector:

In [232…
```
def mi_func(i,j):
    return (2*i*5)+j

#Un array 3x6 con a[i,j] = myfunc(i,j):
a = np.fromfunction(mi_func, (6,6))
b = a.astype(int)
print(b)

b[0,3:5]
b[4:,4:]
b[:,2]
b[2::2,::2]
```

```
[[ 0  1  2  3  4  5]
 [10 11 12 13 14 15]
 [20 21 22 23 24 25]
 [30 31 32 33 34 35]
 [40 41 42 43 44 45]
 [50 51 52 53 54 55]]
```

Out[232…
```
array([[20, 22, 24],
       [40, 42, 44]])
```

Asignacion de elementos:

- No funciona!

In [284…
```
a = zeros(40,1).reshape(5,8)
a
```

Out[284…
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In [295…
```
a[0:0] = 1
a
#a[:,::2]
```

Out[295…
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Trozos como referencia:

- Si se cambia un valor en un trozo, se hace sobre el original

Indexación indirecta

In [308…
```
a = np.arange(0,80,10)
print(a)

#indexación indirecta
y = a[[1,2,-3]]
print(y)

#take
y = np.take(a,[1,2,3])
print(y)
```

```
[ 0 10 20 30 40 50 60 70]
[10 20 50]
[10 20 30]
```

Seleccion de valores con datos logicos:

In [316…
```
mask = np.array([0,1,1,0,0,1,0,0], dtype=bool)
print(mask)

#seleccion de valores
y = a[mask]
print(y)

#con compress
```

```
y = np.compress(mask, a)
print(y)
```

```
[False  True  True False False  True False False]
[2. 3. 6.]
[2. 3. 6.]
```

Ejemplos de seleccion de valores:

In [335…

```
a = np.linspace(1,8,8)
b = a.astype(int)
print(b)

b[[1,6,7]] = 10
print(b)

b[range(2,8,3)] = -2
print(b)

print(b[b<0])

b[b<0] = b.max()
print(b)
```

```
[1 2 3 4 5 6 7 8]
[ 1 10  3  4  5  6 10 10]
[ 1 10 -2  4  5 -2 10 10]
[-2 -2]
[ 1 10 10  4  5 10 10 10]
```

In [340…

```
#b[b%2 == 1]

b = np.array([np.arange(5),np.arange(5),np.arange(6)])
b
```

Out[340…

```
array([array([0, 1, 2, 3, 4]), array([0, 1, 2, 3, 4]),
       array([0, 1, 2, 3, 4, 5])], dtype=object)
```

Indexación indirecta en 2D:

In [353…

```
def mi_func(i,j):
    return (2*i*5)+j

#Un array 3x6 con a[i,j] = myfunc(i,j):
a = np.fromfunction(mi_func, (6,6))
b = a.astype(int)
print(b)

b[(0,1,2,3,4),(1,2,3,4,5)]
b[3:,[0,2,5]]
mask = np.array([1,0,1,0,0,1], dtype=bool)
b[mask,2]
```

```
[[ 0  1  2  3  4  5]
 [10 11 12 13 14 15]
 [20 21 22 23 24 25]
 [30 31 32 33 34 35]
 [40 41 42 43 44 45]
 [50 51 52 53 54 55]]
```

Out[353…

```
array([ 2, 22, 52])
```

Bucles:

In [364…

```
a = np.array([[4,5,6,7,8,9],[6,8,10,12,14,16],[6,9,12,15,18,2]])
for i in a:
    print(i, ":", end="\n")
    for j in i:
        print(j)
```

```
[4 5 6 7 8 9] :
4
5
6
7
8
9
[ 6  8 10 12 14 16] :
6
8
10
12
14
16
[ 6  9 12 15 18  2] :
6
9
12
15
18
2
```

In [365…

```
for e in a.flat:
    print(e)
```

```
4
5
6
7
8
9
6
8
10
12
14
16
6
9
12
15
18
2
```

In [369…

```
for index, value in np.ndenumerate(a):
    print(index, value)
```

```
(0, 0) 4
(0, 1) 5
(0, 2) 6
(0, 3) 7
(0, 4) 8
(0, 5) 9
(1, 0) 6
(1, 1) 8
(1, 2) 10
```

```
(1, 3) 12
(1, 4) 14
(1, 5) 16
(2, 0) 6
(2, 1) 9
(2, 2) 12
(2, 3) 15
(2, 4) 18
(2, 5) 2
```

Modelo de memoria numpy:

- Se puede acceder a cada dimension del ndarray saltando un determinado numero de bytes
- Si la memoria es contigua se puede precalcular los saltos. Si se guardan con el orden:
- Fortran -> acceso + rapido a la primera dimensión
- C -> acceso + rapido a la ultima

In [374…

```
print(a.strides) #numero de bytes/dimension
print(a.flags.fortran, a.flags.contiguous)
```

```
(48, 8)
False True
```

Operaciones sobre vectores:

- Las operaciones se realizan elemento a elemento

In [392…

```
a = np.array([[1,2,3],[4,5,6]], int)
b = 3*a - 1

print(sum(a))

#Sumar a lo largo de un eje(o multiplicar con prod)
#sum(a, axis=0)
```

```
[5 7 9]
```

In [404…

```
a = np.array([1,2,3,4])
b = np.array([2,3,4,5])

a+b

x = np.arange(11.)
a = (2*pi)/10
x*a

#Se puede aplicar sobre una funcion
```

Out[404…

```
array([0, 0.2*pi, 0.4*pi, 0.6*pi, 0.8*pi, 1.0*pi, 1.2*pi, 1.4*pi, 1.6*pi,
       1.8*pi, 2.0*pi], dtype=object)
```

Calcular minimo y máximo:

- a.min(axis=0)
- a.argmin(axis=0)
- De forma equivalente con amax, argmax

In [20]:

```
import numpy as np
a = np.array([[2,3,0,1],[1,2,3,4]])
print(a.min(axis=0))
print(a.min(axis=1))

print(a.argmin(axis=0))
```

```
[1 2 0 1]
[0 1]
[1 1 0 0]
```

Normalizar un array:

In [430…

```
zmax, zmin = z.max(), z.min()
z = (z-zmin)/(zmax-zmin)
print("\n\n",z)
```

```
[[4.95731713 7.91141982 2.18833531 7.46195837 4.54111047]
 [4.08262327 9.22643486 1.50720644 1.30982851 3.81890077]
 [5.41986603 9.44370158 3.23258769 0.97441979 9.4738726 ]
 [8.57034822 0.26036864 1.17893682 8.29376381 1.38474291]
 [5.69471136 4.77775414 3.06686479 0.56607079 9.36835208]]


 [[0.5097896  0.83041709 0.20925445 0.78163419 0.46461605]
 [0.41485353 0.97314401 0.13532721 0.11390453 0.38623005]
 [0.55999296 0.99672535 0.32259378 0.0775005  1.        ]
 [0.90193477 0.         0.09969803 0.87191531 0.12203547]
 [0.58982367 0.49030049 0.30460682 0.03317979 0.98854719]]
```

Operaciones estadisticas:

- mean() -> media
- average -> promedio
- average, wieghts -> media ponderada
- std -> desviacion estandar
- var -> varianza

In [444…

```
a = np.array([[1,2,3],[4,5,6]],float)
print(a.mean())
print(a.mean(axis=0))

np.average(a, axis=0)
np.average(a, weights=[1,2], axis=0)
a.std(axis=0)
a.var(axis=0)
```

```
3.5
[2.5 3.5 4.5]
```

Out[444… `array([2.25, 2.25, 2.25])`

Buena normalizacion:

In [449…

```
z = np.random.uniform(0, 10, (5,5))
print(z)

media = z.mean()
desviacion = z.std()
```

```
z = (z - media)/desviacion

print("\n\n", z)
```

```
[[8.96485587 0.27887208 5.73301327 8.05643111 4.20512433]
 [5.25266837 2.86142593 6.58646233 0.90300409 8.22265309]
 [8.07414288 1.91616674 6.3830302  8.32464616 4.69827092]
 [5.80353033 1.0890278  4.6241949  8.86351833 4.64982775]
 [4.66215271 1.96004669 0.38728889 6.15861756 5.6463128 ]]


 [[ 1.49476269 -1.75708822  0.28482833  1.15466744 -0.28718138]
 [ 0.10499723 -0.79023399  0.60434183 -1.52342623  1.21689748]
 [ 1.16129836 -1.14411929  0.52818109  1.25508155 -0.1025576 ]
 [ 0.31122844 -1.45378284 -0.13029011  1.45682404 -0.12069371]
 [-0.1160795  -1.12769156 -1.71649923  0.44416569  0.25236948]]
```

Otros metodos:

- clip(x,y) -> valores por debajo de x se sustituyen por y
- round(decimal=x) -> redondea al decimal x
- ptp(axis=x) -> calcula distancia punto a punto por columnas

In [455…

```
a = np.array([[1,2,3],[4,5,6]], float)
print(a)

a = a.clip(3,5)
print("\n", a)
```

```
[[1. 2. 3.]
 [4. 5. 6.]]

 [[3. 3. 3.]
 [4. 5. 5.]]
```

In [458…

```
a = np.array([1.35, 2.5, 1.5])
a.round(decimals=1)
```

Out[458… array([1.4, 2.5, 1.5])

Distancia punto a punto por columnas:

In [460…

```
a = np.array([[1,2,3],[4,5,6]], float)
print(a.ptp(axis = 0))

#0 el maximo para todos los pares:
a.ptp(axis=None)
```

```
[3. 3. 3.]
```

Out[460… 5.0

Resumen:

```
Atributos basicos:
* a.dtype
* a.shape
* a.size
```

* a.itemsize
* a.nbytes
* a.ndim

Operaciones sobre la forma del ndarray
* a.flat
* a.flatten(): Devuelve una copia 1D del array multidimensional
* a.ravel(): Igual que flatten pero como una "vista"
* a.resize(nuevo tam)
* a.swapaxes(axis1,axis2) -> equivalente a traspuesta ->
a.transpose(*axes) -> a.T (abreviado)
* a.squeeze(): Elimina las dimensiones de longitud 1
* a.copy()
* a.fill(valor)

Conversion:
* a.tolist()
* a.tostring()
* a.astype(dtype)
* a.byteswap(False): Convierte el orden de byte(big <-> little
endian)

Numeros complejos:
* a.real
* a.imag
* a.conjugate()
* a.conj()

* a.dump(fichero): Guarda el array en un archivo se lee con load()
* a.dumps(): Devuelve el pickle binario del array como un string
* a.tofile(fild, sep="", formart="%s"): Salida ascii formateada a
fichero

* a.nonzero(): Devuelve indices de elementos que no son cero
* a.sort(axis=x)
* a.argsort(axis=x): Devuelve los indices para los elementos
ordenados
* a.searchsorted(b): Devuelve los indices donde se encontrarian los
elementos de b en a

* a.clip(low, high)
* a.round(decimals=x)
* a.cumsum(axis=None): Suma acumulada de un eje
* a.cumprod(axis=None): Producto acumulado a los largo de un eje

Metodos de reduccion: reducen el tamaño del array a una dimension,
aplica la operacion a lo largo del eje, si el eje es None se realiza
para todos los elementos.
* a.sum(axis=None)
* a.prod(axis=None)
* a.min(axis=None)
* a.max(axis=None)
* a.argmin(axis=None)
* a.argmax(axis=None)
* a.ptp(axis=None): Calcula a.max(axis) - a.min(axis)
* a.mean(axis=None)
* a.std(axis=None)

```
* a.var(axis=None)

* a.any(axis=None): True si algun valor es distinto de 0
* a.any(axis=None): True si todos los valores son distintos de 0
```

Funciones estadisticas:

In [21]:

```
import numpy as np

a = np.arange(0,100,2)
b = a + 20
correlacion = np.corrcoef(a,b)
print(correlacion)
```

```
[[1. 1.]
 [1. 1.]]
```

Operadores binarios matematicos:

```
a + b -> add(a,b)
a - b -> subtract(a,b)
a % b -> remainder(a,b)
a * b -> multiply(a,b)
a / b -> divide(a,b)
a ** b -> power(a,b)
```

Operadores lógicos:

```
equal ==
greater_equal >=
logical_and
logical_not
not_equal !=
less <
logical_or
greater >
less_equal <=
logical_xor
```

Operadores bit a bit:

```
bitwise_and &
bitwise_or |
invert ~
bitwise_xor
right_shift(a,shifts)
left_shift(a,shifts)
```

Otras funciones:

```
sin(x)
cos(x)
arcos(x)
sinh(x)
coshh(x)
arcosh(x)
arctan(x)
```

```
arcsin(x)
arctan2(x,y)
arctanh(x)
arcsinh(x)
------------------
exp(x)              log(x)
log10(x)            sqrt(x)
absolute(x)         conjugate(x)
negative(x)         ceil(x)
floor(x)            fabs(x)
hypot(x)            fmod(x,y)
maximun(x,y)        minimun(x,y)
```

Vector de registros:

- Un item puede incluir campos de diferentes tipos
- Un campo se describe con un objeto y un numero de bytes de desplazamiento. Así se pueden encadenar registros.
- El contructor interpreta los elementos de la tupla como campos.

In [26]:

```
dt = np.dtype("i4,f8,a5") #i entero, f flotante, a string
print(dt.fields)

a = np.array([(1,2.0,"Hello"),(2,3.0,"World")], dtype=dt)
print(a)

print(a['f2'])
```

```
{'f0': (dtype('int32'), 0), 'f1': (dtype('float64'), 4), 'f2': (dtype('S5'),
12)}
[(1, 2., b'Hello') (2, 3., b'World')]
[b'Hello' b'World']
```

Entrada/Salida:

- Para leer datos en formato texto
  - **datos = np.loadtxt('datos.txt')**
- Para guardar datos en formato texto
  - **np.savetext('datos2.txt', datos)**

Propoagación (Broadcasting):

- Con multiples entradas, son difundibles a la misma dimension
- Todos los arrays se modifican para tener mismas dimensiones
- Se expanden las dimensiones de longitud 1
- Permite hacer operaciones sobre elementos, filas o col de forma mas rápida

In [30]:

```
x = np.array([1,2,3,4])
y= np.array([[10],[20],[30]])
print(np.add(x,y)) # x es: (4,) y la funcion la ve como (1,4)
                   # y es: (3,1)
                   # Resultado (3,4)
```

```
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

Normas de propagacion:

- Los ejes de ambos arrays deben ser 1 o tener el mismo tamaño.
- En caso contrarip **ValueError**

In [25]:
```
import numpy as np

a = np.array((0,10,20,30))
b = np.array((0,1,2))
print("A: {}\nB: {}\n".format(a, b))


y = a[:, np.newaxis]
y = a[:, None] # Es lo mismo que la linea anterior
print(y)

z = a[:,None] + b
z
```

```
A: [ 0 10 20 30]
B: [0 1 2]

[[ 0]
 [10]
 [20]
 [30]]
```

Out[25]:
```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Funciones universales:

- Ufuncs son objetos que evaluan rapidamente una funcion elemento a elemento
- En arrays 1D es equivalente a compresión de listas pero mucho mas rapido

- A continuación ejemplos de ufun

In [49]:
```
import math
print(type(np.exp))

x = np.array([1,2,3,4,5])
print(np.exp(x))
%timeit np.exp(x)

print([math.exp(val) for val in x])
%timeit [math.exp(val) for val in x]
```

```
<class 'numpy.ufunc'>
[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
823 ns ± 24.6 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
[2.718281828459045, 7.38905609893065, 20.085536923187668, 54.598150033144236,
148.4131591025766]
1.45 µs ± 6.7 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [53]:
```
import timeit
timeit.timeit('import math; import numpy; x=numpy.array([1,2,3,4,5]); x = [ma
```

Out[53]: 2.7577400549998856

In [54]:
```
timeit.timeit('import numpy; x=numpy.array([1,2,3,4,5]); numpy.exp(x)', numbe
```

Out[54]: 1.8638780219989712

Todas las funciones amtematicas de comparacion, logicas y bit a bit que tienen dos operados se pueden usar

In [57]:
```
a = np.arange(8)
print(a)
print(np.add.reduce(a))
```

```
[0 1 2 3 4 5 6 7]
28
```

In [3]:
```
import numpy as np

a = np.arange(24).reshape(6,4)
print(a)

print("\n", np.add.reduce(a, axis=0))
print(np.add.reduce(a, axis=1))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]

 [60 66 72 78]
[ 6 22 38 54 70 86]
```

In [68]:
```
a = np.arange(8)
print(np.add.accumulate(a))
```

```
[ 0  1  3  6 10 15 21 28]
```

In [7]:
```
a = np.arange(24).reshape(6,4)
print(a,"\n\n")

print(np.add.accumulate(a, axis=0))
print("\n", np.add.accumulate(a, axis=1))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]


[[ 0  1  2  3]
 [ 4  6  8 10]
 [12 15 18 21]
 [24 28 32 36]
 [40 45 50 55]
 [60 66 72 78]]

 [[ 0  1  3  6]
```

```
 [ 4  9 15 22]
 [ 8 17 27 38]
 [12 25 39 54]
 [16 33 51 70]
 [20 41 63 86]]
```

In [77]:

```
a = np.arange(5)
b = np.arange(10,15)
print("A:",a,"\nB:", b)

np.add.outer(a,b) # Aplica ufunc a todos los pares(a,b)
```

```
A: [0 1 2 3 4]
B: [10 11 12 13 14]
```

Out[77]:

```
array([[10, 11, 12, 13, 14],
       [11, 12, 13, 14, 15],
       [12, 13, 14, 15, 16],
       [13, 14, 15, 16, 17],
       [14, 15, 16, 17, 18]])
```

In [83]:

```
a = np.arange(5)
print(a)
print(np.add.reduceat(a,[1,2])) #indices [axis, dtype, out]

print(np.add.reduceat(a,[1,3])) #indices [axis, dtype, out]

a = np.arange(24).reshape(6,4)
print("\n\n",a)
np.add.reduceat(a,[1,2])
```

```
[0 1 2 3 4]
[1 9]
[3 7]


 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

Out[83]:

```
array([[ 4,  5,  6,  7],
       [56, 60, 64, 68]])
```

Vectorización:

Esto provoca fallo:

Debido a que no se puede pasar un vector, hay que vectorizarlo para aplicarlo componente a componente

In [10]:

```
def sinc(x):
    if x == 0.0:
        return 1.0
    else:
        w = math.pi*x
        return math.sin(w)/w

sinc(np.array([1.3,1.5]))
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-10-a489e42136f8> in <module>
      6         return math.sin(w)/w
      7
----> 8 sinc(np.array([1.3,1.5]))

<ipython-input-10-a489e42136f8> in sinc(x)
      4     else:
      5         w = math.pi*x
----> 6         return math.sin(w)/w
      7
      8 sinc(np.array([1.3,1.5]))

TypeError: only size-1 arrays can be converted to Python scalars
```

Alternativa:

In [87]:
```
from numpy import vectorize
vsinc = vectorize(sinc)
vsinc(np.array([1.3,1.5]))
```

Out[87]:
```
array([-0.19809085, -0.21220659])
```

In [12]:
```
def f_sin(x):
    x1 = np.ones(x.size, float)
    w = math.pi * x
    x2 = math.sin(w)/w
    return where(x==0,x1,x2)
```

Numeros aleatorios:

- Generar numeros aleatorios escalares:

In [129…
```
import random
random.seed(2198)
print("Numero aleatorio uniforme(0,1):", random.random())
print("Numero aleatorio uniforme(-1,1):", random.uniform(-1,1))
print("Numero aleatorio normal(0,1):", random.gauss(0,1))
print("Numero aleatorio entero(1,10):", random.randint(1,10))
```

```
Numero aleatorio uniforme(0,1): 0.7230685205563604
Numero aleatorio uniforme(-1,1): -0.8741062425025439
Numero aleatorio normal(0,1): 1.4243342332687365
Numero aleatorio entero(1,10): 3
```

In [134…
```
from numpy import random
random.seed(12)
n = 10

u = random.random(n)
print(u)

u = random.uniform(-1,1,n)
print("\n", u)

m = 3
s = 2
u = random.normal(m,s,n)
print("\n", u)
```

```
[0.15416284 0.7400497  0.26331502 0.53373939 0.01457496 0.91874701
 0.90071485 0.03342143 0.95694934 0.13720932]

 [-0.43234329  0.21216637  0.88845027  0.70547108 -0.99548153  0.04245205
  0.10407527 -0.02924517  0.53626831 -0.67856649]

 [5.19191224 0.5696624  5.68471274 2.75570042 5.02503095 1.17226171
 0.94093959 5.4195929  4.00374461 3.27769235]
```

Tipo matriz:

- Se eliminara en un futuro

In [139…

```
x1 = np.array([1,2,3], float)
x2 = np.matrix(x1)
print(x2)

x3 = np.mat(x1).transpose()
print(x3)

print(type(x3))
```

```
[[1. 2. 3.]]
[[1.]
 [2.]
 [3.]]
<class 'numpy.matrix'>
```

In [146…

```
A = np.eye(3)
print(A)

A = np.mat(A)
print("\n", A)

y2 = x2*A
print("\n", y2)

y3 = A*x3
print("\n", y3)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

 [[1. 2. 3.]]
[[1.]
 [2.]
 [3.]]
```

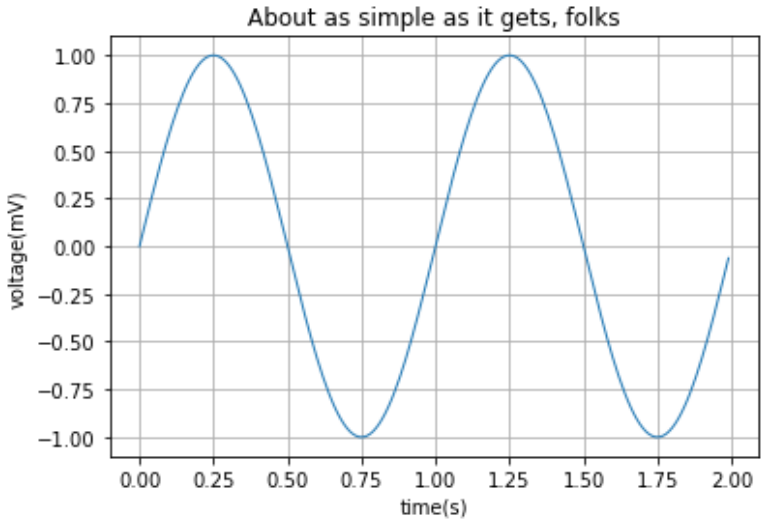
Representación gráfica en python:

- **Matplotlib**

In [11]:

```
import pylab as plt
import numpy as np
import math
```

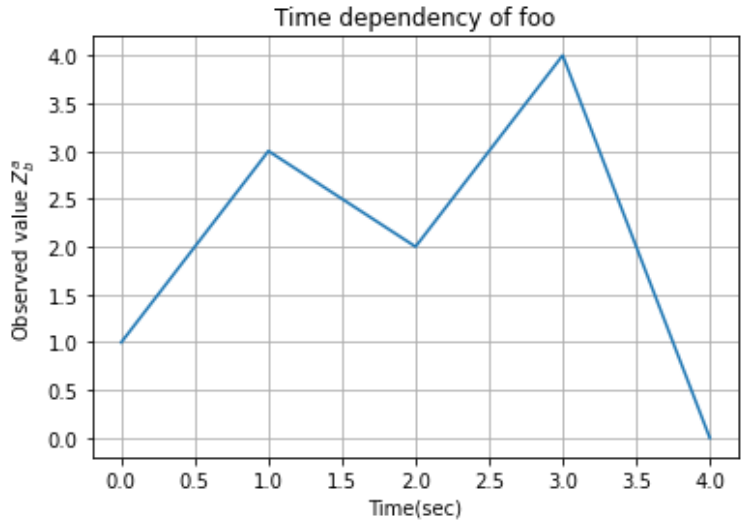
```
t = np.arange(0.0,2.0,0.01)
s = np.sin(2*math.pi*t)

plt.plot(t, s, linewidth=1.0)
plt.xlabel('time(s)')
plt.ylabel('voltage(mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)
plt.show()
```



Grafica simple con Latex:

In [17]:

```
plt.plot([1,3,2,4,0])
plt.title('Time dependency of foo')
plt.xlabel('Time(sec)')
plt.ylabel(r'Observed value $Z^{a}_{b}$')
plt.grid(True)
plt.show()
```



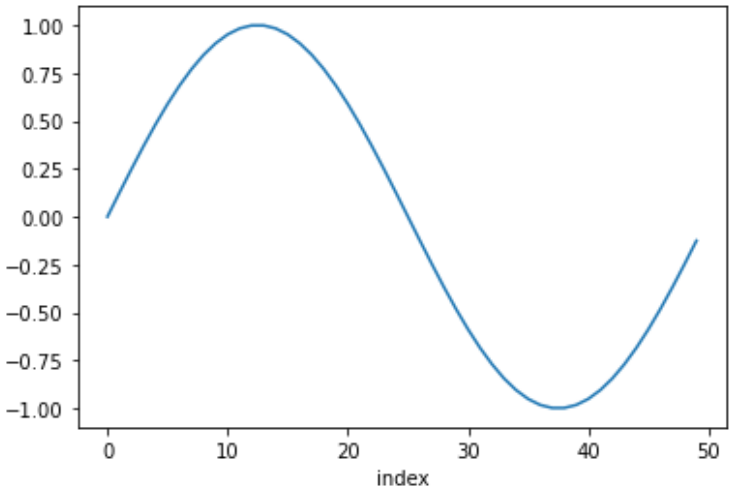
Gráficas de datos:

In [32]:

```
x = np.arange(50)*2*math.pi/50
y = np.sin(x)
```

```
plt.plot(y)
plt.xlabel("index")
```

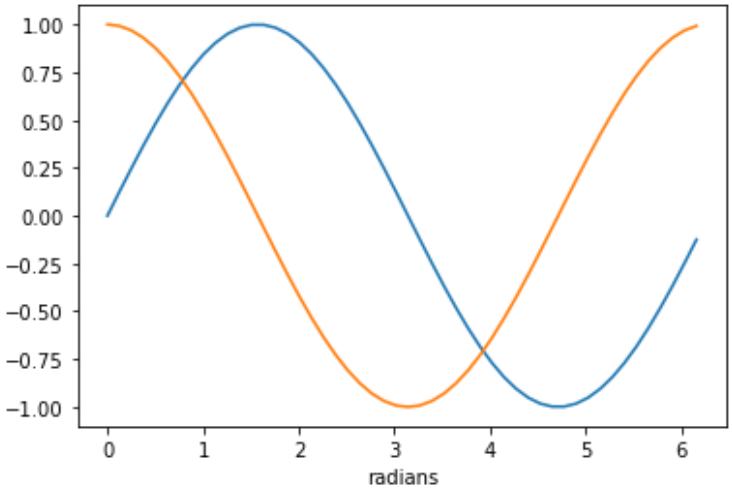
Out[32]: Text(0.5, 0, 'index')



In [24]:

```
x1 = np.arange(50)*2*math.pi/50
y1 = np.cos(x)
plt.plot(x,y,x1,y1)
plt.xlabel("radians")
```

Out[24]: Text(0.5, 0, 'radians')

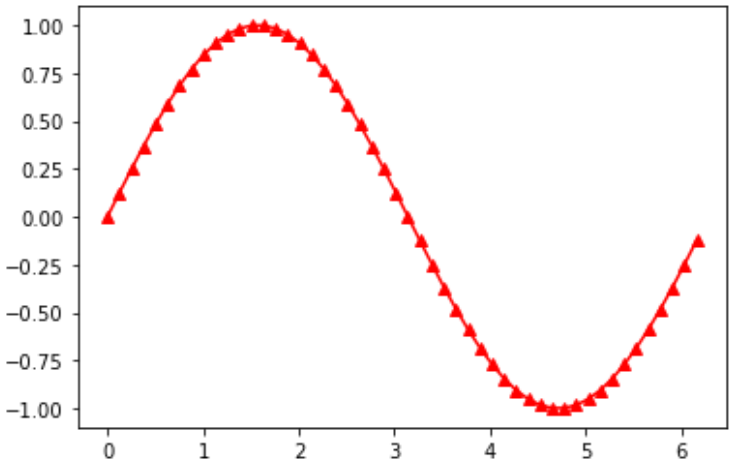


Formatos de linea:

In [33]:

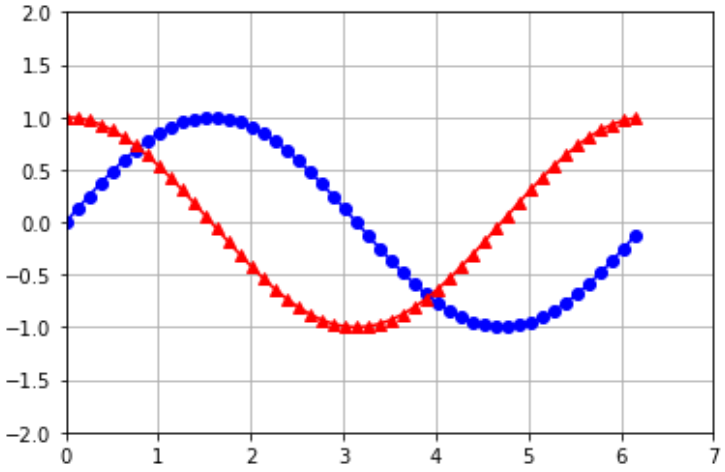
```
plt.plot(x,np.sin(x),'r-^')
```

Out[33]: [<matplotlib.lines.Line2D at 0x7f36d5f01040>]



In [38]:

```
plt.plot(x,y,'b-o',x1,y1,'r-^')
plt.axis([0,7,-2,2])
plt.grid(True)
```



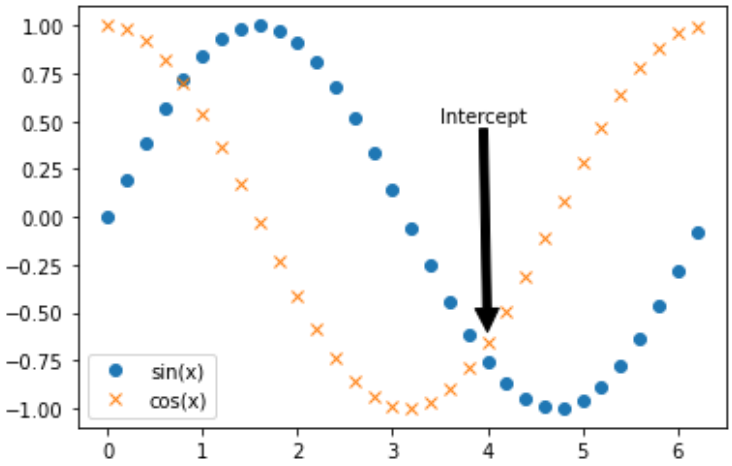
Gráfica dispersa con leyenda:

- Estilo de linea: ['-' | '--' | -. | ':' | 'steps' | ...]
- Marcadores: ['+' | ',' | '.' | '1' | '2' | '3' | '4']

In [51]:

```
x = np.arange(0.,2.*math.pi,0.2)
print("Shape de x", x.shape)

plt.plot(x, np.sin(x), 'o', label="sin(x)")
plt.plot(x, np.cos(x), 'x', label="cos(x)")

plt.legend()
# Anotación
plt.annotate('Intercept', xy=(4, -0.60), xytext=(3.5,0.5), arrowprops=dict(fa
plt.show()
plt.savefig("plot.svg") # Formatos pdf, png, ps, eps
plt.close()
```

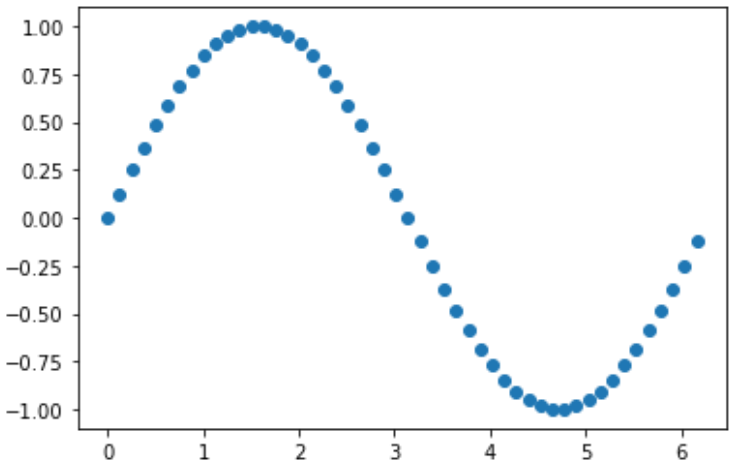
```
Shape de x (32,)
```



Gráficas con datos dispersos:

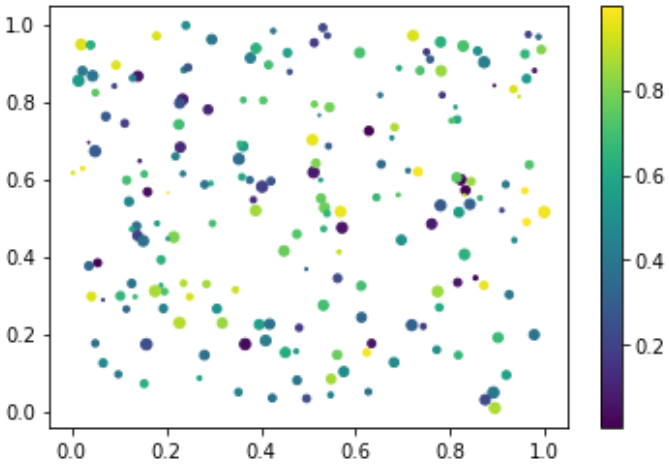
In [55]:
```
x = np.arange(50)*2*math.pi/50
y = np.sin(x)
plt.scatter(x,y)
```

Out[55]: <matplotlib.collections.PathCollection at 0x7f36d59d70d0>



In [68]:
```
from numpy import random as nprandom

x = nprandom.rand(200)
y = nprandom.rand(200)
size = nprandom.rand(200)*30
color = nprandom.rand(200)
plt.scatter(x,y,size,color)
plt.colorbar()
```

Out[68]: <matplotlib.colorbar.Colorbar at 0x7f36d558ad90>

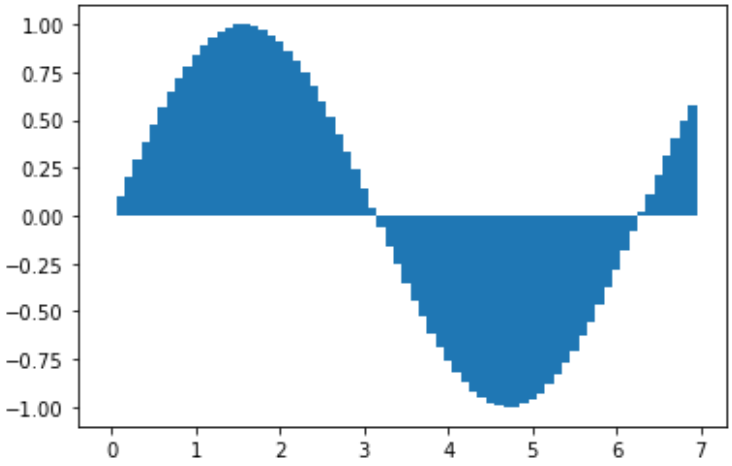


Gráficos de barras:

In [97]:
```
x = np.arange(0,7,0.1)
print(x[1], x[0])
plt.bar(x,np.sin(x),width=x[1]-x[0])
```

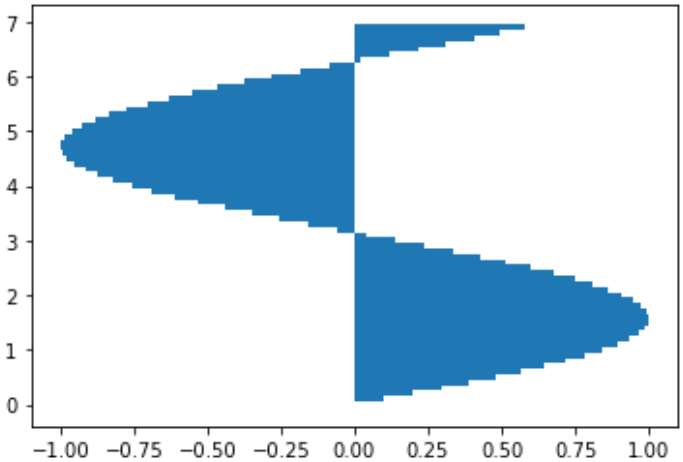
```
0.1 0.0
```

Out[97]:
```
<BarContainer object of 70 artists>
```



In [85]:
```
plt.barh(x,np.sin(x),height=x[1]-x[0])
```

Out[85]:
```
<BarContainer object of 70 artists>
```

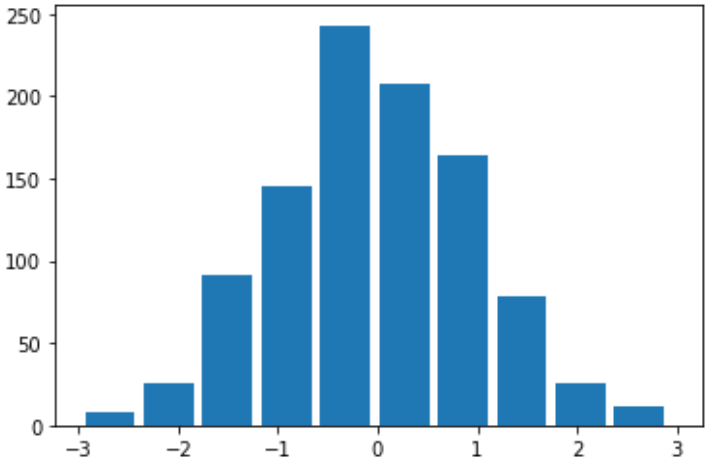


Gráficos de barras

Histogramas:

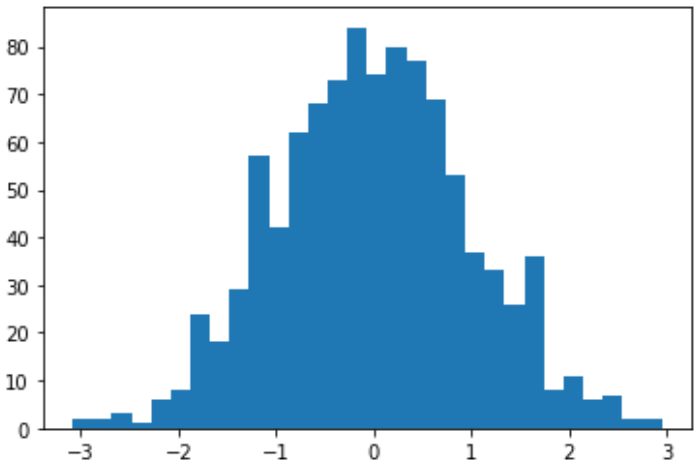
In [90]:
```
plt.hist(nprandom.randn(1000), width=0.5)
```

Out[90]:
```
(array([  8.,  26.,  91., 145., 243., 208., 164.,  78.,  26.,  11.]),
 array([-2.93530164, -2.34521034, -1.75511903, -1.16502772, -0.57493642,
         0.01515489,  0.6052462 ,  1.1953375 ,  1.78542881,  2.37552011,
         2.96561142]),
 <a list of 10 Patch objects>)
```



In [87]:
```
plt.hist(nprandom.randn(1000),30)
```

Out[87]:
```
(array([ 2.,  2.,  3.,  1.,  6.,  8., 24., 18., 29., 57., 42., 62., 68.,
        73., 84., 74., 80., 77., 69., 53., 37., 33., 26., 36.,  8., 11.,
         6.,  7.,  2.,  2.]),
 array([-3.08493388, -2.88375979, -2.6825857 , -2.48141161, -2.28023752,
        -2.07906343, -1.87788934, -1.67671525, -1.47554116, -1.27436707,
        -1.07319298, -0.87201889, -0.6708448 , -0.46967071, -0.26849662,
        -0.06732253,  0.13385156,  0.33502565,  0.53619974,  0.73737383,
         0.93854792,  1.13972201,  1.3408961 ,  1.54207019,  1.74324428,
         1.94441837,  2.14559246,  2.34676655,  2.54794064,  2.74911472,
         2.95028881]),
 <a list of 30 Patch objects>)
```



Tema 3: Programación interfaces de usuario Python

Existen varias:

- Tkinter (Tk)
- wxPython (wxWindows)
- PyQt (Qt)
- PyGtk (GTK)
- PySide (Qt)
- PythonWin (MFC)

In [16]:

```
import tkinter as tk

def main():
    root = tk.Tk()
    root.grid()
    etiqueta = tk.Label(root, text="Hola mundo")
    etiqueta.grid()
    root.mainloop()

main()
```

In [7]:

```
def botonPulsado():
    global raiz
    raiz.destroy()

def main():
    global raiz

    root = tk.Tk()
    root.geometry("200x100")
    etiqueta = tk.Label(root, text="Hola mundo")
    etiqueta.grid(row=0, column=0)
    boton = tk.Button(root, text="Salir", command=botonPulsado)
    boton.grid(row=0, column=1)

    raiz = root
    root.mainloop()

main()
```

Creando entrada de texto:

In [12]:

```
import tkinter as tk
entryBox = None

def botonPulsado():
    global entryBox
    txt = entryBox.get()
    print("El texto es:", txt)

def crearCajaTexto(parent):
    global entryBox
    entryBox = tk.Entry(parent)
    entryBox.grid()

def main():
```

```
    root = tk.Tk()
    miBoton = tk.Button(root, text="Mostrar texto", command=botonPulsado)
    miBoton.grid()

    crearCajaTexto(root)
    root.mainloop()

main()
```

```
El texto es: Hola
```

Mirar, carpeta de ejemplos Tkinter + calculadora.py

Tema 5 - Python Avanzado

Indice:

Clases

Una clase define un tipo de dato junto con sus operaciones y su estado

Elementos de una clase:

Un tipo de dato junto con sus operaciones y su estado. Algunas operaciones:

```
__init__ constructor
__del__ se llama justo antes de destruir el objeto
__repr__ la representación oficial en forma de string
__str__ la conversión a string
self para acceder al propio objeto
```

Ejemplo de clase:

In [3]:

```
class Yate:
    def __init__(self, inclinacion, velocidad):
        self.inclinacion = inclinacion
        self.velocidad = velocidad

    def __repr__(self):
        return "La inclinacion es: {}\nLa velocidad es: {}".format(self.incli


yate1 = Yate(2.5, 20)
print(yate1)
```

```
La inclinacion es: 2.5
La velocidad es: 20
```

La clase object

Todas las clases en Python heredan de la clase object

Crear objetos

- Se usa el nombre de la clase para crear un nuevo objeto
- El objeto se destruye cuando no queda ninguna referencia

Manipulación de objetos

In [17]:
```
class Punto:
    cont = 0 #Atributo global de la clase
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        Punto.cont += 1

    def posicion(self):
        return self.x, self.y

    def desplazar(self, dx, dy):
        self.x += dx
        self.y += dy

    def cuantos_puntos(self):
        return Punto.cont

    def __repr__(self):
        return "(x,y): ({},{})\nNumero de puntos creados: {}".format(self.x,

p1 = Punto(3,4)
print(str(p1))

p2 = Punto(14,21)
print(p2)

p2.posicion()
```

```
(x,y): (3,4)
Numero de puntos creados: 1
(x,y): (14,21)
Numero de puntos creados: 2
```

Out[17]:
```
(14, 21)
```

Los atributos de un objeto

Los atributos se pueden definir cuando sean necesarios Pueden aparecer solo en un objeto:

pass para definir clase vacia

In [9]:
```
class Box:
    pass

b1 = Box()
b2 = Box()

b1.alto = 100
b1.ancho = 50
b1.color = "red"

b2.alto = 100
b2.ancho = 50
```

```
print(b2.ancho)
```

```
50
```

Ejemplo de herencia:

In [19]:

```
class Punto3D(Punto):
    z = 0 # Atributo instancia de la clase

    def __init__(self, x, y ,z):
        Punto.__init__(self, x, y)
        self.z = z

    def desplazar(self, dx, dy, dz):
        Punto.desplazar(self, dx, dy)
        self.z += dz

    def __repr__(self):
        info = str(Punto.__repr__(self))
        info += "\nZ: {}".format(self.z)
        return info

    def __repr__(self):
        return "(x,y,z): ({},{},{})\nNumero de puntos creados: {}".format(sel

p3d = Punto3D(5,4,3)
print(p3d)
```

```
(x,y,z): (5,4,3)
Numero de puntos creados: 5
```

In [28]:

```
class Pila:
    def __init__(self):
        self.items = []

    def push(self, x):
        self.items.append(x)

    def pop(self):
        x = self.items[-1]
        del self.items[-1]
        return x

    def empty(self):
        return len(self.items) == 0

    def __str__(self):
        pila = ""
        for i in self.items[::-1]:
            pila += str(i) + ", "
        return pila

x = Pila()
print(x.empty())
x.push(1)
print(x.empty())
print(x)
x.push(3)
x.push(9)
```

```
x.push("hola")
print(x)
x.pop()
print(x)
```

```
True
False
1,
hola, 9, 3, 1,
9, 3, 1,
```

Ejemplo de clase derivada (I)

In [31]:

```
class PilaModificada(Pila):
    def seleccionar(self, n):
        size = len(self.items)
        assert 0 <= n < size
        return self.items[size-1-n]

y = PilaModificada()
y.push(1)
y.push("adios")
y.push("claro")
print(y)
print(y.seleccionar(1))
```

```
claro, adios, 1,
adios
```

Otro ejemplo de herencia:

In [32]:

```
class Animal:
    def __init__(self, name):
        self.name = name

class Gato(Animal):
    def hablar(self):
        return 'Miau!'

class Perro(Animal):
    def hablar(self):
        return 'Guau!'
```

Restricciones de acceso

Loos datos son publicos aunque se usa la convencion de usar __<dato> para indicar que es privado

In [19]:

```
class Ejemplo:
    cont = 0
    def __init__(self):
        self.pub = 0
        self._priv = 1
        self.__priv2 = 2

    def contador(self):
        Ejemplo.cont += 1
        return Ejemplo.cont
```

```
ej = Ejemplo()
print(ej._priv)
ej.__priv2
```

```
1
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-19-0cef36016d09> in <module>
     12 ej = Ejemplo()
     13 print(ej._priv)
---> 14 ej.__priv2

AttributeError: 'Ejemplo' object has no attribute '__priv2'
```

In [37]:
```
print(ej._Ejemplo__priv2) # Permite acceso
```

```
2
```

Limitaciones de copy

In [48]:
```
import copy
pila1 = Pila()
print(hex(id(pila1)))
pila2 = copy.copy(pila1)
print(hex(id(pila2)))
pila3 = copy.deepcopy(pila1)
print(hex(id(pila3)))
```

```
0x7f06a465be20
0x7f06a465b070
0x7f06a465bdc0
```

Métodos intrínseco (I)

dict Diccionario que permite conocer los atributos de la clase y sus valores, así como modificarlos **call** Permite usar un objeto como una funcion

In [49]:
```
for i in ej.__dict__.keys():
    print(i, ej.__dict__[i])
```

```
pub 0
_priv 1
_Ejemplo__priv2 2
```

In [52]:
```
class Factorial:
    def __init__(self):
        self.cache = {}
    def __call__(self, n):
        if n not in self.cache:
            if n == 0:
                self.cache[n] = 1
            else:
                self.cache[n] = n * self.__call__(n-1)
        return self.cache[n]

fact = Factorial()
for i in range(10):
    print("{}! = {}".format(i, fact(i)))
```

```
print(fact.cache)
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
{0: 1, 1: 1, 2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880}
```

Sobrecarga de operadores: datos numéricos

In [67]:

```
class Tiempo:
    def __init__(self, hrs, min):
        self.hrs = hrs
        self.min = min

    def __add__(self, t):
        hrs = self.hrs + t.hrs
        min = self.min + t.min
        if min >= 60:
            hrs = hrs + 1
            min = min - 60
        return Tiempo(hrs, min)

    def __iadd__(self, t):
        self.hrs = self.hrs + t.hrs
        self.min = self.min + t.min
        if self.min >= 60:
            self.hrs = self.hrs + 1
            self.min = self.min - 60
        return self

    def __repr__(self):
        return "Tiempo es " + str(self.hrs) + "h " + str(self.min) + "s"

t1 = Tiempo(4, 23)
print(t1)
t2 = Tiempo(4, 39)
print(t2)
t3 = t1 + t2
print(t3)

t3 += t1
print(t3)
```

```
Tiempo es 4h 23s
Tiempo es 4h 39s
Tiempo es 9h 2s
Tiempo es 13h 25s
```

Sobrecarga de operadores

Operador	Método de la clase

Operador	Método de la clase
-	**sub**(self, other) **rsub**(self, other) **isub**(self, other)
+	**add**(self, other) **radd**(self, other) **iadd**(self, other)
*	**mul**(self, other) **rmul**(self, other) **imul**(self, other)
/	**div**(self, other) **rdiv**(self, other) **idiv**(self, other)
%	**mod**(self, other) **rmod**(self, other) **imod**(self, other)

Operador Unario	Método de la clase
-	**neg**(self)
+	**pos**(self)

Operador	Método de la clase
==	**eq**(self, other)
!=	**ne**(self, other)
<	**lt**(self, other)
>	**gt**(self, other)
<=	**le**(self, other)
>=	**ge**(self, other)

Sobrecarga de operadores: otros tipos

Se puede emular el comportamiento de otros datos

- Secuencias
- Troceados
- Diccionarios

Metodos: por ejemplo para diccionarios:

```
__len__(self)
__getitem__(self, key)
__setitem__(self, key, value)
__delitem__(self, key)
```

Sobrecarga para el tipo list

In [73]:

```
class Mi_lista:
    def __init__(self, p1, p2):
        self.theList = []
        self.theList.append(p1)
        self.theList.append(p2)
```

```
    def __add__(self, other):
        result = []

        for item in self.theList:
            result.append(item)
        for item in other.theList:
            result.append(item)

        return result

    def __str__(self):
        return str(self.theList)

t1 = Mi_lista(1,2)
print(t1)
t2 = Mi_lista(3,4)
print(t2)
t3 = t1 + t2
print(t3)
```

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

Sobrecarga de [] usando getitem

In [76]:

```
class Mis_datos:
    def __init__(self, datos="ABDCDEFG"):
        self.data = datos

    def __getitem__(self, i):
        return self.data[i]

obj = Mis_datos()
print(obj[1])

for item in obj: # Se usa []
    print(item, end=" ")
```

```
B
A B D C D E F G
```

Atributos no definidos, procedimiento default

Llamar a un metodo no definido genera una excepcion

A menos que se defina __getattr__()

In [80]:

```
class Mi_clase:
    def __getattr__(self, metodo):
        print("Metodo no encontrado:", metodo)
        return self.myDefault

    def myDefault(self):
        print("default()")

    def f(self): print("f()")
    def g(self): print("g()")
    def h(self): print("h()")
```

```
a = Mi_clase()
a.f()
a.g()
a.h()
a.dummy()
```

```
f()
g()
h()
Metodo no encontrado: dummy
default()
```

Metodos estaticos:

Son comunes a todos los objetos de la clase se indican con @staticmethod

In [33]:

```
class A:
    def foo(self, x):
        print("Ejecutando foo(%s,%s)" %(self,x))

    @classmethod
    def class_foo(cls, x):
        print("Ejecutando class_foo(%s,%s)" %(cls,x))

    @staticmethod
    def static_foo(x):
        print("Ejecutando static_foo(%s)" %(x))

a = A()
a.foo(3)
A.class_foo(5)
A.static_foo(4)
```

```
Ejecutando foo(<__main__.A object at 0x7f7b7eadeb80>,3)
Ejecutando class_foo(<class '__main__.A'>,5)
Ejecutando static_foo(4)
```

Clases abstractas

No implementan algunos metodos y se definen como abstractos

In [85]:

```
from abc import ABCMeta, abstractmethod, abstractproperty
class Formas:
    __metaclass__ = ABCMeta

    @abstractmethod
    def mostrar(self): pass

    @abstractmethod
    def nombre(self): pass

# Se deben implementar todos los metodos abstractos

class Circulo(Formas):
    def __init__(self, nombre):
        self.nom = nombre

    def mostrar(self):
        print("Circulo", self.nom)
```

```
    def nombre(self):
        return (self.nom)

c = Circulo("Paco")
c.mostrar()
c.nombre()
```

```
Circulo Paco
```

Out[85]:
```
'Paco'
```

@property

Se puede indicar que un atributo se modifica y se consulta mediante dos metodos

- OJO SE PONE: _

In [88]:
```
class Celsius:
    def __init__(self, temperatura=0):
        self.temperatura = temperatura

    def to_fahrenheit(self):
        return (self.temperatura * 1.8) + 32

    def get_temperatura(self):
        print("Obteniendo valor")
        return self._temperatura

    def set_temperatura(self, value):
        if value < -273:
            raise ValueError("No es posible tener temperaturas por debajo de
        print("Asignando valor")
        self._temperatura = value

    temperatura = property(get_temperatura, set_temperatura)

c = Celsius()
c.temperatura = -30 #Se llama al setter
print(c.temperatura) # Se llama al getter
```

```
Asignando valor
Asignando valor
Obteniendo valor
-30
```

Clases de tipo excepcion

Se pueden definir nuevas excepciones

In [90]:
```
class MyError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)

try:
    raise MyError(2*2)
except MyError as e:
    print("Excepcion generada, valor:", e.value)
```

```
raise MyError('oh!')
```

```
Excepcion generada, valor: 4
---------------------------------------------------------------------------
MyError                                   Traceback (most recent call last)
<ipython-input-90-a5124c4711de> in <module>
     11     print("Excepcion generada, valor:", e.value)
     12
---> 13 raise MyError('oh!')

MyError: 'oh!'
```

Iteradores:

```
* Crea una clase con los metodos __iter__() y __next__()
* Se llama a __iter__() al principio, normalmente devuelve self
* next() en cada iteracion
```

In [92]:

```
class Fib:
    def __init__(self, max=100):
        self.max = max

    def __iter__(self):
        self.a = 0
        self.b = 1
        return self

    def __next__(self):
        fib = self.a
        if fib > self.max:
            raise StopIteration
        self.a, self.b = self.b, self.a + self.b
        return fib

for f in Fib():
    print(f)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

Iteradores con generadores

In [93]:

```
class Fibonacci:
    def __init__(self):
        self.x, self.y = 0, 1

    def __iter__(self):
        return self

    def __str__(self):
```

```
        return str(self.x)

    def miGenerador(self):
        while self.x < 10000:
            yield self.x
            self.x, self.y = self.y, self.x + self.y
        return

f = Fibonacci()
g = f.miGenerador()
for i in g:
    print(f)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

Pandas

Biblioteca open soruce que permite: Trabajar con ficheros csv, excel, json, bases de datos sql

- Los deposita en datos (DataFrames),
- Reliza distintas tecnicas de analisis de datos

```
import pandas as pd
dataframe = pd.read_excel('fichero.xlsx', 'sheet_name')
```

- Añade estructuras de datos y herramientas para trabajar con datos en forma de tabla (similar a Series y Data Frames en R)
- Proporciona herramientas para manipulación de datos:
  - Cambiar dimensiones
  - Mezclar
  - Ordenar
  - Trocear
  - Agregar
  - etc
- Permite manejar datos desaparecidos

DataFrame

- Representa una tabla, una estructura similar a una hoja de cálculo
- Contiene una colección ordenada de columnas que pueden tener distintos valores (Numerico, string, logico, etc)
- Tiene indices tanto por filas como por columnas

Creación

In [19]:
```
import pandas as pd

data = {'region' : ['Andalucia', 'Andalucia', 'Madrid', 'Madrid', 'Galicia'],
        'año' : [2014, 2015, 2014, 2015, 2015],
        'poblacion' : [1.5, 1.7, 3.6, 4.2, 2.9]
       }

frame = pd.DataFrame(data)
frame
```

Out[19]:

	region	año	poblacion
0	Andalucia	2014	1.5
1	Andalucia	2015	1.7
2	Madrid	2014	3.6
3	Madrid	2015	4.2
4	Galicia	2015	2.9

Ordenar columnas

In [2]:
```
print(pd.DataFrame(data, columns=['año', 'region', 'poblacion']))
```

```
    año     region  poblacion
0  2014  Andalucia        1.5
1  2015  Andalucia        1.7
2  2014     Madrid        3.6
3  2015     Madrid        4.2
4  2015    Galicia        2.9
```

Añadir columnas

In [54]:
```
frame2 = pd.DataFrame(data,
                      columns=['año', 'region', 'poblacion', 'deuda'],
                      index=['uno', 'dos', 'tres' ,'cuatro', 'cinco'])

frame2
```

Out[54]:

	año	region	poblacion	deuda
uno	2014	Andalucia	1.5	NaN
dos	2015	Andalucia	1.7	NaN
tres	2014	Madrid	3.6	NaN
cuatro	2015	Madrid	4.2	NaN

	año	region	poblacion	deuda
cinco	2015	Galicia	2.9	NaN

Imprimir columnas

In [41]:
```
print(frame2.columns)
```

```
Index(['año', 'region', 'poblacion', 'deuda'], dtype='object')
```

Imprimir una columna usando . y []

In [43]:
```
print(frame2.año)
print(frame2["region"])
```

```
uno       2014
dos       2015
tres      2014
cuatro    2015
cinco     2015
Name: año, dtype: int64
uno       Andalucia
dos       Andalucia
tres         Madrid
cuatro       Madrid
cinco       Galicia
Name: region, dtype: object
```

Modificación de columnas

In [44]:
```
frame2['deuda'] = 16.5
frame2
```

Out[44]:

	año	region	poblacion	deuda
uno	2014	Andalucia	1.5	16.5
dos	2015	Andalucia	1.7	16.5
tres	2014	Madrid	3.6	16.5
cuatro	2015	Madrid	4.2	16.5
cinco	2015	Galicia	2.9	16.5

Modificación de columnas con arange

In [45]:
```
import numpy as np
frame2['deuda'] = np.arange(5.)
frame2
```

Out[45]:

	año	region	poblacion	deuda
uno	2014	Andalucia	1.5	0.0
dos	2015	Andalucia	1.7	1.0

	año	region	poblacion	deuda
tres	2014	Madrid	3.6	2.0
cuatro	2015	Madrid	4.2	3.0
cinco	2015	Galicia	2.9	4.0

Asignar una lista/serie

In [46]:
```
val = pd.Series([-1.2, -1.5, -1.7],
                index=['dos','cuatro','cinco']
               )
frame2['deuda'] = val
frame2
```

Out[46]:

	año	region	poblacion	deuda
uno	2014	Andalucia	1.5	NaN
dos	2015	Andalucia	1.7	-1.2
tres	2014	Madrid	3.6	NaN
cuatro	2015	Madrid	4.2	-1.5
cinco	2015	Galicia	2.9	-1.7

Añadir una columna con []

In [47]:
```
# Si no existe se añade
frame2['En costa'] = (frame2.region == 'Andalucia')
frame2
```

Out[47]:

	año	region	poblacion	deuda	En costa
uno	2014	Andalucia	1.5	NaN	True
dos	2015	Andalucia	1.7	-1.2	True
tres	2014	Madrid	3.6	NaN	False
cuatro	2015	Madrid	4.2	-1.5	False
cinco	2015	Galicia	2.9	-1.7	False

Borrar columnas

In [48]:
```
del frame2['En costa']
print(frame2.columns)
frame2
```

```
Index(['año', 'region', 'poblacion', 'deuda'], dtype='object')
```

Out[48]:

	año	region	poblacion	deuda
uno	2014	Andalucia	1.5	NaN
dos	2015	Andalucia	1.7	-1.2

	año	region	poblacion	deuda
tres	2014	Madrid	3.6	NaN
cuatro	2015	Madrid	4.2	-1.5
cinco	2015	Galicia	2.9	-1.7

Usando diccionarios de diccionarios

In [27]:
```
poblacion = {'Andalucia' : {2014:1.5, 2015:1.7},
             'Madrid' : {2014:3.6, 2015:4.2},
             'Galicia' : {2015:2.9}}

frame3 = pd.DataFrame(poblacion)
frame3
```

Out[27]:

	Andalucia	Madrid	Galicia
2014	1.5	3.6	NaN
2015	1.7	4.2	2.9

Traspuesta

In [28]:
```
frame3.T
```

Out[28]:

	2014	2015
Andalucia	1.5	1.7
Madrid	3.6	4.2
Galicia	NaN	2.9

Indices explicitos

In [29]:
```
print(pd.DataFrame(frame3,
                   index=[2014, 2015, 2016]))
```

```
      Andalucia  Madrid  Galicia
2014        1.5     3.6      NaN
2015        1.7     4.2      2.9
2016        NaN     NaN      NaN
```

Uso de indices

* Se puede usar un indice para mejorar las busquedas

In [55]:
```
print(frame2, "\n")
frame2 = frame2.set_index(['region'])
print(frame2.loc["Andalucia"])
```

```
         año     region  poblacion deuda
```

```
uno     2014  Andalucia        1.5   NaN
dos     2015  Andalucia        1.7   NaN
tres    2014     Madrid        3.6   NaN
cuatro  2015     Madrid        4.2   NaN
cinco   2015    Galicia        2.9   NaN

            año  poblacion deuda
region
Andalucia  2014        1.5   NaN
Andalucia  2015        1.7   NaN
```

Nombres para filas y columnas

In [50]:

```
frame3.index.name = 'año'
frame3.columns.name = 'region'
frame3
```

Out[50]:

region	Andalucia	Madrid	Galicia
año			
2014	1.5	3.6	NaN
2015	1.7	4.2	2.9

Valores

In [51]:

```
print(frame3.values)
print(type(frame3.values))
```

```
[[1.5 3.6 nan]
 [1.7 4.2 2.9]]
<class 'numpy.ndarray'>
```

Ejemplo con ficheros xls

- Se necesita los paquetes xlrd xlwd

```
import pandas as pd
poblacion = pd.read_escel('fichero.xls', 'Nombre')

print(poblacion["municipio"])
print(poblacion[poblacion["municipio"] == "Vera"])
print(poblacion[poblacion["municipio"] == "Vera"]["edad0015"].sum())
#Da resultado de la suma

poblacion = poblacion.set_index(['municipio'])
aux = poblacion.loc["Vera"]
print(aux.describe())
```

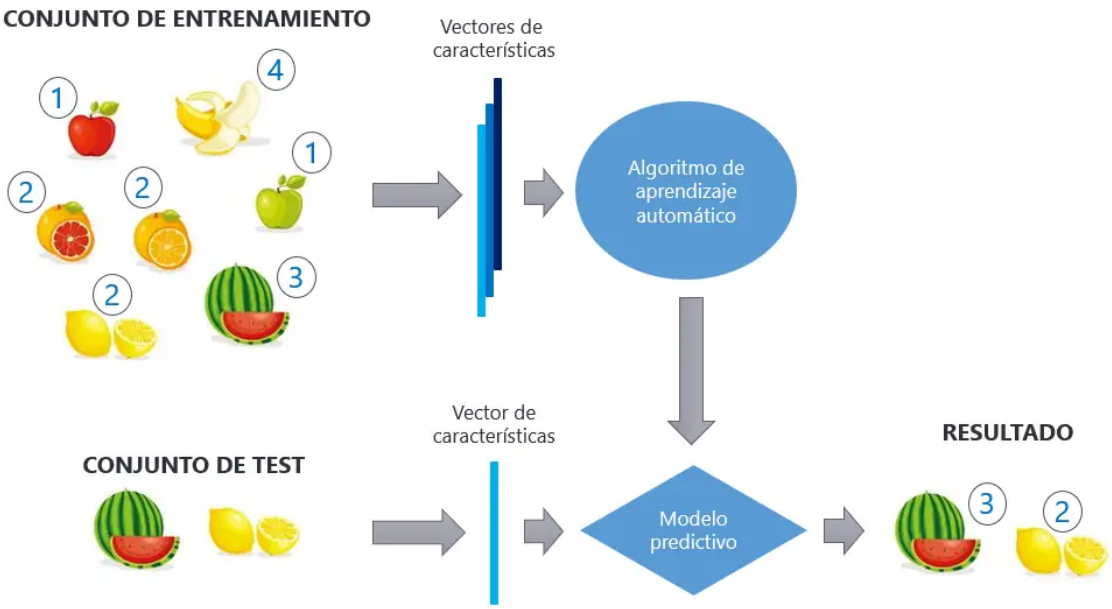
Scikit-learn

Aprendizaje automático y minería de datos

- Scikit-learn
- Orange

- Pandas
- MLpy
- MDP
- PyBrain

Ejemplo de clasificacion:



Ejemplo Scikit-learn

- Python Machine learning
- Se usa import sklearn
- Aprendizaje supervisado, ejemplos etiquetados
  - Clasificacion (binaria o multiclase)
  - Regresión
- Aprendizaje no supervisado
  - Clustering

Conjunto de datos de ejemplo

- Dentro del paquete sklearn

```
from sklearn import datasets
  iris = datasets.load_iris()
  digits = datasets.load_digits()
```

- Un dataset internamente es un diccionario que tiene datos y metadatos
- Datasets externos (otros links)

In [12]:

```
from sklearn import datasets
iris = datasets.load_iris() # Dataset iris
digits = datasets.load_digits() # Dataset numeros, predecir
# Imprimir los datos
print(iris.data)
print(digits) # Imprimo toda la informacion
```

```
[[5.1 3.5 1.4 0.2]
```

```
[4.9 3.  1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5.  3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  3.4 1.5 0.2]
[4.4 2.9 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[4.8 3.4 1.6 0.2]
[4.8 3.  1.4 0.1]
[4.3 3.  1.1 0.1]
[5.8 4.  1.2 0.2]
[5.7 4.4 1.5 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.5 1.4 0.3]
[5.7 3.8 1.7 0.3]
[5.1 3.8 1.5 0.3]
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
```

```
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
```

```
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]]), 'target': array([0, 1, 2, ...,
8, 9, 8]), 'frame': None, 'feature_names': ['pixel_0_0', 'pixel_0_1', 'pixel_
0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'pixel_0_7', 'pixel
_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3', 'pixel_1_4', 'pixel_1_5', 'pixe
l_1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_2_1', 'pixel_2_2', 'pixel_2_3', 'pix
el_2_4', 'pixel_2_5', 'pixel_2_6', 'pixel_2_7', 'pixel_3_0', 'pixel_3_1', 'pi
xel_3_2', 'pixel_3_3', 'pixel_3_4', 'pixel_3_5', 'pixel_3_6', 'pixel_3_7', 'p
ixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pixel_4_3', 'pixel_4_4', 'pixel_4_5',
'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pixel_5_1', 'pixel_5_2', 'pixel_5_3',
'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'pixel_5_7', 'pixel_6_0', 'pixel_6_1',
'pixel_6_2', 'pixel_6_3', 'pixel_6_4', 'pixel_6_5', 'pixel_6_6', 'pixel_6_7',
'pixel_7_0', 'pixel_7_1', 'pixel_7_2', 'pixel_7_3', 'pixel_7_4', 'pixel_7_5',
'pixel_7_6', 'pixel_7_7'], 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8,
9]), 'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
```

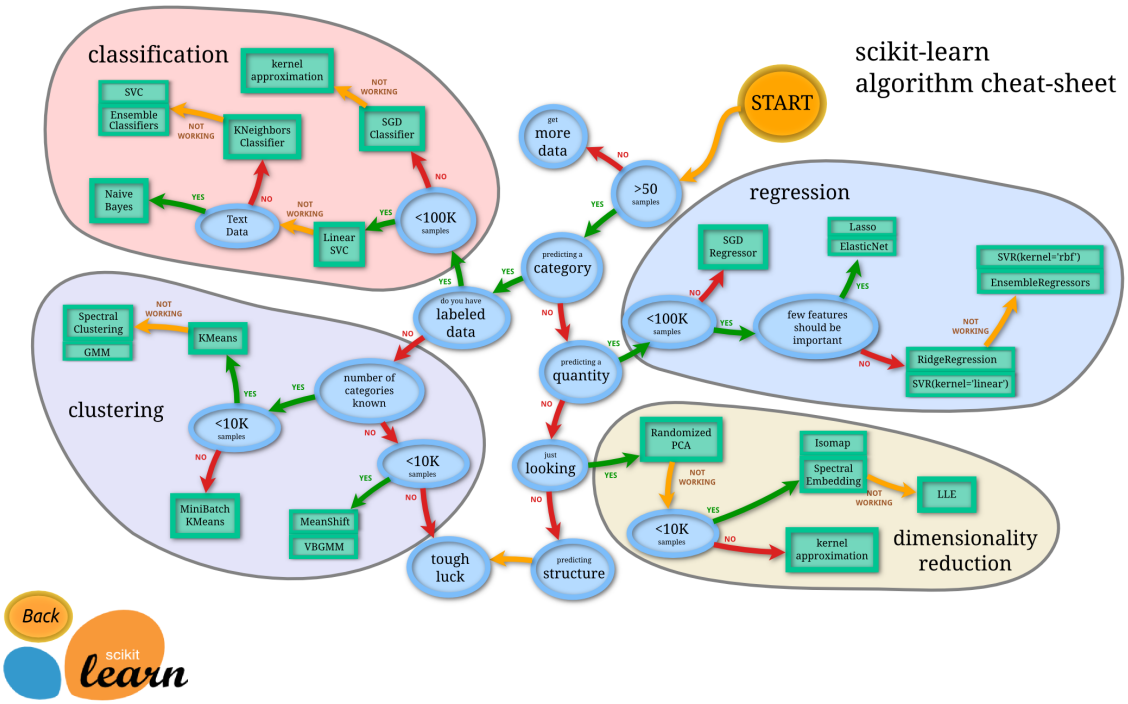
```
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]), 'DESCR': ".. _digits_datase
t:\n\nOptical recognition of handwritten digits dataset\n--------------------
-----------------------------\n\n**Data Set Characteristics:**\n\n    :Numbe
r of Instances: 5620\n    :Number of Attributes: 64\n    :Attribute Informati
on: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribute V
alues: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date:
July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digit
s datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Ha
ndwritten+Digits\n\nThe data set contains images of hand-written digits: 10 c
lasses where\neach class refers to a digit.\n\nPreprocessing programs made av
ailable by NIST were used to extract\nnormalized bitmaps of handwritten digit
s from a preprinted form. From a\ntotal of 43 people, 30 contributed to the t
raining set and different 13\nto the test set. 32x32 bitmaps are divided into
nonoverlapping blocks of\n4x4 and the number of on pixels are counted in each
block. This generates\nan input matrix of 8x8 where each element is an intege
r in the range\n0..16. This reduces dimensionality and gives invariance to sm
all\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garri
s, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A.
Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTI
R 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Co
mbining Multiple Classifiers and Their\n    Applications to Handwritten Digit
Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and En
gineering, Bogazici University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading
Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan and Xi Y
ao and A. Kai Qin.\n    Linear dimensionalityreduction using relevance weight
ed LDA. School of\n    Electrical and Electronic Engineering Nanyang Technolo
gical University.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal
Margin Classification\n    Algorithm. NIPS. 2000."}
```

In [16]:

```
# Data del dataset
# Un array de n_samples x n_features (muestras, caracteristicas)
print(digits.data)

# Target del dataset, predicción, del 0 al 9 los digitos
print(digits.target)

# Imagenes de los numeros, representan numeros
print(digits.images[0])
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[0 1 2 ... 8 9 8]
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

Algoritmos de scikit-lean



Aprendizaje y predicción

* Un estimador para clasificación es un objeto con los metodos:
    - fit(X, y)
    - predict(T)
* Por ejemplo para SVM
    - sklearn.svm.SVC
    - Ejemplo:

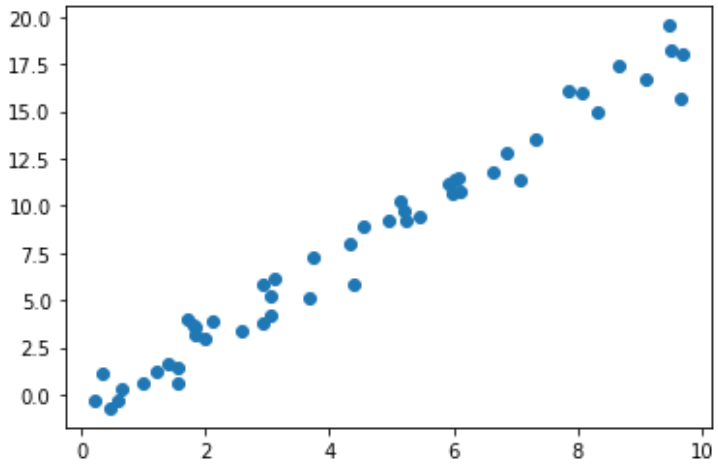
```
from sklearn import svm
clf = svm.SVC(gamma=0.001, C=100.)
```

Ejemplo de aprendizaje (I) [Regresión lineal]

In [27]:

```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)

plt.scatter(x, y)
```

Out[27]: <matplotlib.collections.PathCollection at 0x7fbc9c1a0e20>



Ejemplo de aprendizaje (II) [Regresión lineal]

In [39]:
```
from sklearn.linear_model import LinearRegression # Modelo lineal

model = LinearRegression(fit_intercept=True)
print(x.shape)
X = x[:, np.newaxis] # Los datos los pasa a una sola columna
print(X.shape)

model.fit(X, y)
print(model.coef_) # Coeficientes estimados para el problema de regresión lir

model.intercept_  # Término independiente en el modelo lineal. Establecer en
```

```
(50,)
(50, 1)
[1.9776566]
```

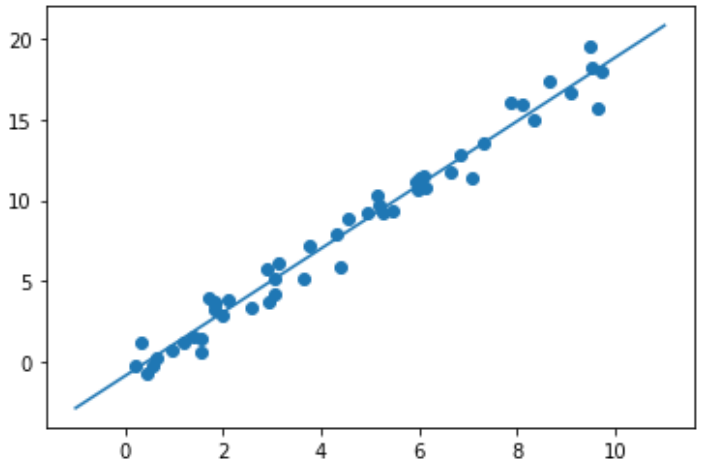
Out[39]:
```
-0.9033107255311164
```

Ejemplo de aprendizaje (III) [Regresión lineal]

In [47]:
```
# Predicción de datos
xfit = np.linspace(-1, 11) # Devuelve números espaciados uniformemente durant

print(xfit.shape)
Xfit = xfit[:, np.newaxis] # Los datos los pasa a una sola columna
print(Xfit.shape)

yfit = model.predict(Xfit)

plt.scatter(x,y)
plt.plot(xfit, yfit)
```

```
(50,)
(50, 1)
```

Out[47]:
```
[<matplotlib.lines.Line2D at 0x7fbc9bd03070>]
```



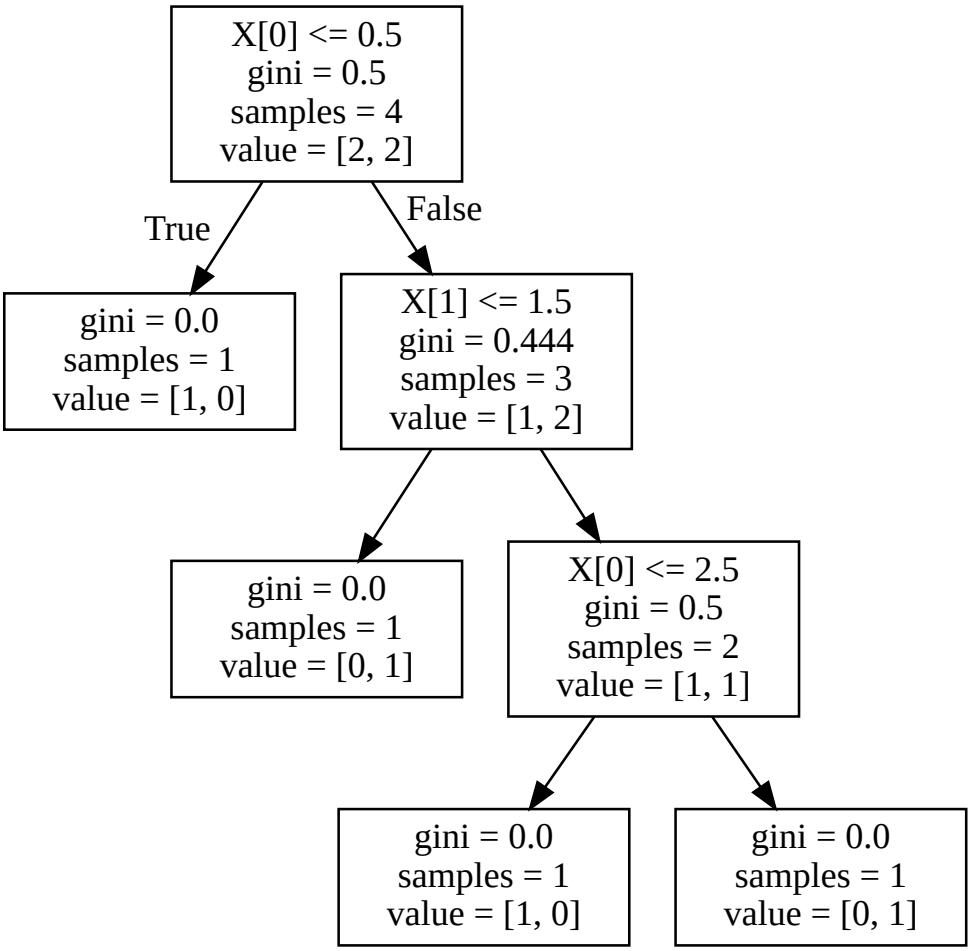
Ejemplo de aprendizaje (IV) [Arboles de decisión]

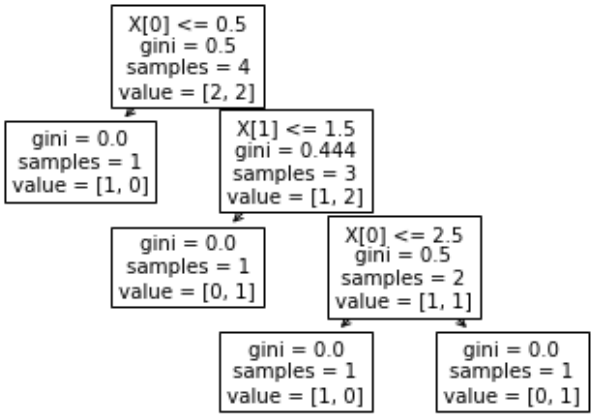
In [68]:

```
from sklearn import tree

X = [[0,0], [1,1], [2,2], [3,3]]
Y = [0, 1, 0, 1]

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)

tree.plot_tree(clf); # Ploteando el arbol

print(clf.predict([[2.,2.]]))
print(clf.predict_proba([[2.,2.]]))

import graphviz

dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph
```

```
[0]
[[1. 0.]]
```

Out[68]:





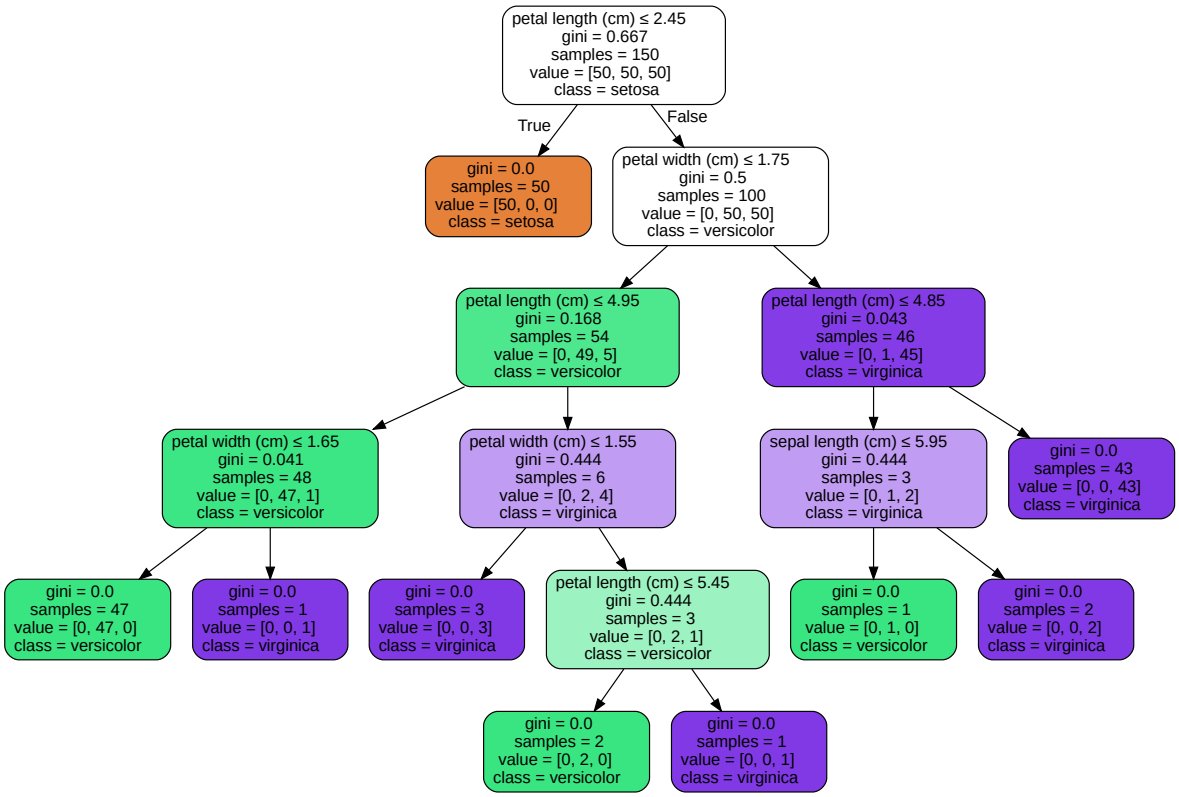
Ejemplo de aprendizaje (V) [Arboles de decision]

In [77]:

```
from sklearn.datasets import load_iris
from sklearn import tree

iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)

import graphviz

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
```

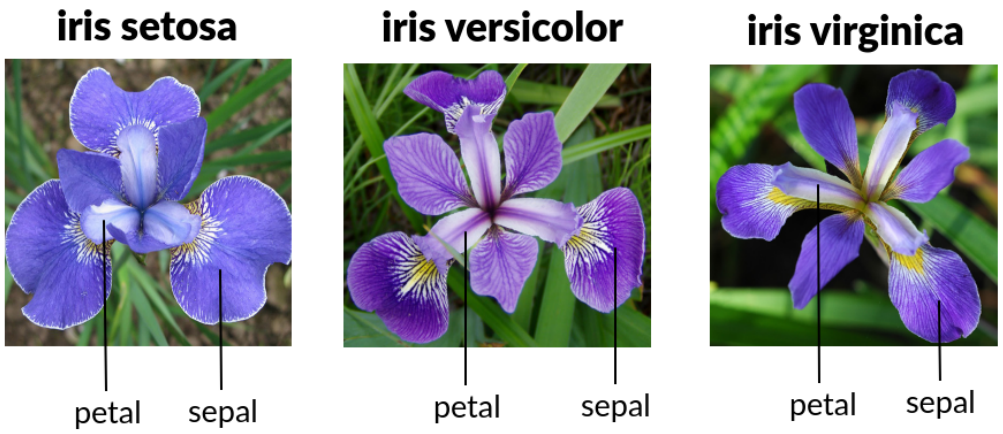
```
graph = graphviz.Source(dot_data)
graph.render("iris")
graph
```

Out[77]:



Ejemplo: Iris Dataset

Claisifica los tipos de flores usando los siguientes rasgos:
* Longitud del sépalo
* Ancho del sépalo
* Longitud del petalo
* Ancho del petalo



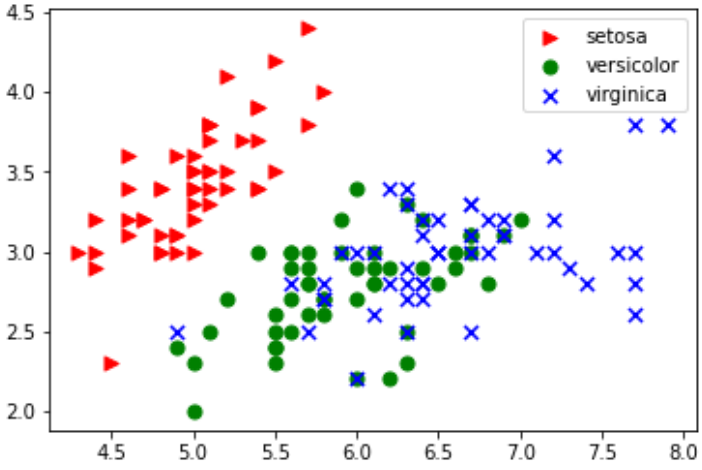
Visualización de datos:

In [92]:

```
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
import numpy as np
```

```
data = load_iris()
features = data['data'] #datos
feature_names = data['feature_names'] #Nombre de las caracteristicas
target = data['target'] # Objetivos, 0,1,2 [setosa, versicolor, virginica]

print(feature_names)
print(data.target)
print(data.target_names)

cad = []
cad2 = []

for t, marker, c in zip(range(3), ">ox", "rgb"): #> o x, #r g b
    aux = plt.scatter(features[target == t, 0], #x
                      features[target == t, 1], #y
                      marker = marker,
                      c=c,
                      s=50) #SIZE
    cad.append(aux)
    cad2.append(data.target_names[t])
plt.legend(cad, cad2, ncol=1, loc='upper right')
plt.show()
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```



Un ejemplo simple: Regresion

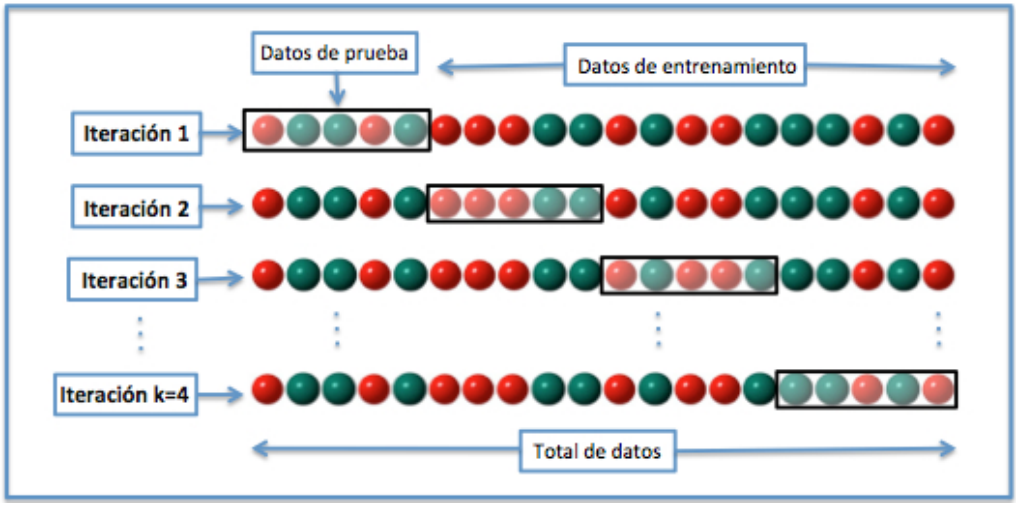
In [111…

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression # Modelo lineal

iris = load_iris()
X = iris.data[:, :2] # Dos de los rasgos
y = iris.target

# Normalizar datos
X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)

# Division en entrenamiento y test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rar

# Etiquetado binario
is_versicolor_tr = y_train == 0
binary_target_tr = np.zeros(len(X_train))
binary_target_tr[is_versicolor_tr] = 1
print(binary_target_tr) # Pasa las etiquetas a binario, boleanos

is_versicolor_tst = y_test == 0
binary_target_tst = np.zeros(len(X_test))
binary_target_tst[is_versicolor_tst] = 1

# Ajuste
model = LogisticRegression()
model_binary = LogisticRegression()
model_binary.fit(X_train, binary_target_tr)
model.fit(X_train, y_train)

# Exactitud del ajuste
tr_accuracy = np.mean(model.predict(X_train) == y_train)
tst_accuracy = np.mean(model.predict(X_test) == y_test)
#---------------------------------------------------------
tr_accuracyb = np.mean(model_binary.predict(X_train) == binary_target_tr)
tst_accuracyb = np.mean(model_binary.predict(X_test) == binary_target_tst)

print("Accuracy train: ", tr_accuracy)
print("Accuracy test: ", tst_accuracy)
print(model.predict(X[-1].reshape(1,-1)), y[1]) # Para predecir un dato que s
print(np.sum(y_train), np.sum(y_test))

print("\nBinary(ETIQUETADO BINARIO, MUCHO MEJOR)\nAccuracy train: ", tr_accur
print("Accuracy test: ", tst_accuracyb)
print(model_binary.predict(X[-1].reshape(1,-1)), y[1]) # Para predecir un dat
print(np.sum(y_train), np.sum(y_test))
```

```
[0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 1. 0. 0.]
Accuracy train:  0.79
Accuracy test:  0.82
[1] 0
103 47

Binary(ETIQUETADO BINARIO, MUCHO MEJOR)
Accuracy train:  0.99
Accuracy test:  1.0
[0.] 0
103 47
```

Validación cruzada (I)

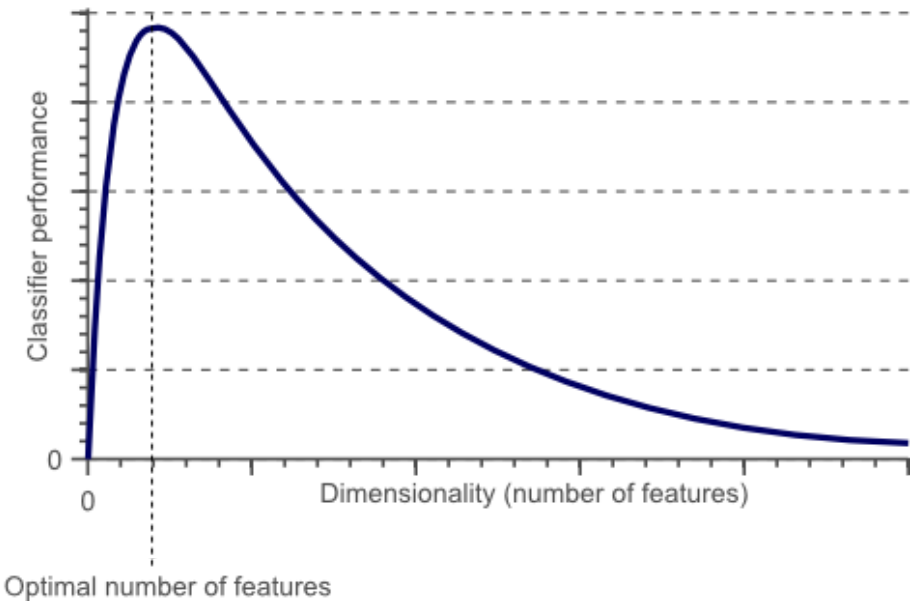


Validicación cruzada (II)

```
* La exactitud no debe depender de la suerte
    kf = KFold(n=len(binary_target), n_folds=5, shuffle=True)
* En kf tendríamos las n_fols divisiones
    for tr, tst in kf:
        tr_features = features[tr, :]
        tr_target = binary_target[tr]
        tst_features = features[tst, :]
        tst_target = binary_target[tst]
* Nunca se deben usar datos de entrenamiento para test
```

La maldición de la dimensionalidad

- Si el numero de variables no es suficiente no es posible conseguir una calidad deseable
- Si el numero es demasiado alto puede ocurrir lo mismo



Número de rasgos

- Dependera del problema y del tipo de clasificador
- Si el metodo tiende a sobreajustar se deben usar relativamente pocos rasgos. Ej: redes neuronales, KNN, arboles de decision

- Por el contrario, si el metodo generaliza bien, se deben usar muchos rasgos. El: naive Bayes, clasificadores lineales.
- ¿Cómo seleccionar el numero optimo? https://es.wikipedia.org/wiki/Selecci%C3%B3n_de_variable

Reducción de dimensionalidad

- Se usa para eliminar caracteristicar, y quedarse con las mejores
- Selección de rasgos
- Extraccion de rasgos:
  - PCA (Analisis de componentes principales)
  - Kernel PCA
  - Kernel PCA basada en grafos
  - LDA (Analisis discriminante lineal)
  - GDA (Analisis discriminante generalizado)

PCA (I)

- Se puede definir un número de componentes principales con las que quedarnos

```
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components,
          svd_solver='randomized', whiten=True)
pca = pca.fit(X_train)
```

- Se aplica a los datos antes de usar el clasificador

```
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

PCA (II)

1. Escoger el modelo

```
from sklearn.decomposition import PCA
```

2. Instanciar el modelo con hiperparametros

```
model_pca = PCA(n_components=2)
```

3. Ajustar los datos, y no se incluye

```
model_pca.fit(X_iris)
```

4. Transforma los datos a 2D

```
X_2D = model_pca.transform(X_iris)
```

PCA (III)

In [156…

```
import seaborn as sns
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]

sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-156-cb7ba88f8d25> in <module>
      3 iris['PCA2'] = X_2D[:, 1]
```

```
      4
----> 5 sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False)

~/Documents/anaconda3/lib/python3.8/site-packages/seaborn/regression.py in lm
plot(x, y, data, hue, col, row, palette, col_wrap, height, aspect, markers, s
harex, sharey, hue_order, col_order, row_order, legend, legend_out, x_estimat
or, x_bins, x_ci, scatter, fit_reg, ci, n_boot, units, seed, order, logistic,
lowess, robust, logx, x_partial, y_partial, truncate, x_jitter, y_jitter, sca
tter_kws, line_kws, size)
    576     need_cols = [x, y, hue, col, row, units, x_partial, y_partial]
    577     cols = np.unique([a for a in need_cols if a is not None]).tolist(
)
--> 578     data = data[cols]
    579
    580     # Initialize the grid

TypeError: unhashable type: 'list'
```

SVM

Support Vector Machine, esta considerado como el clasificador estandar

- Para clasificar con SVM:

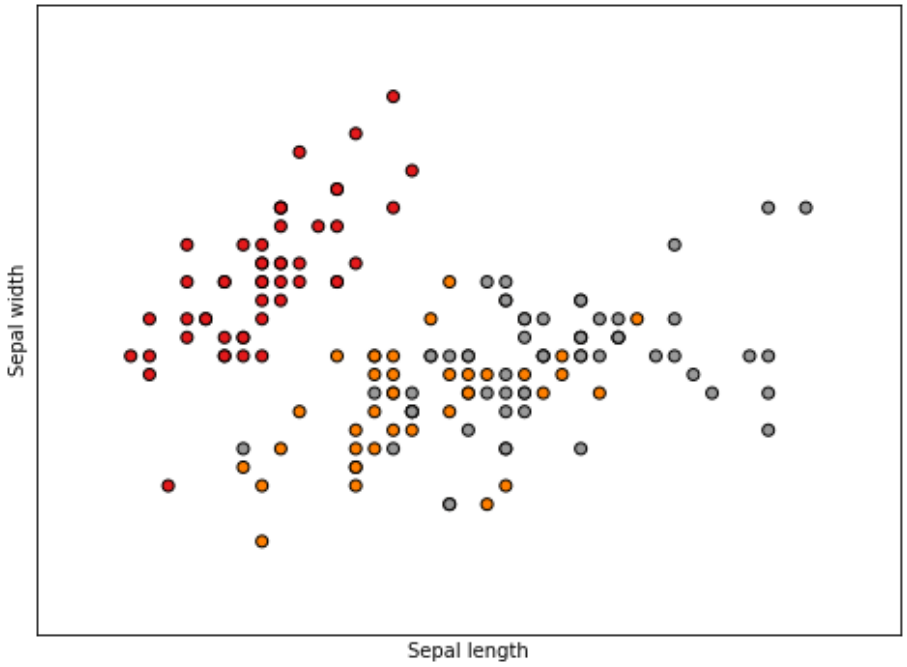
```
from sklearn.svm import SVC
...
model = SVC()
model.fit(tr_features, tr_target)
```

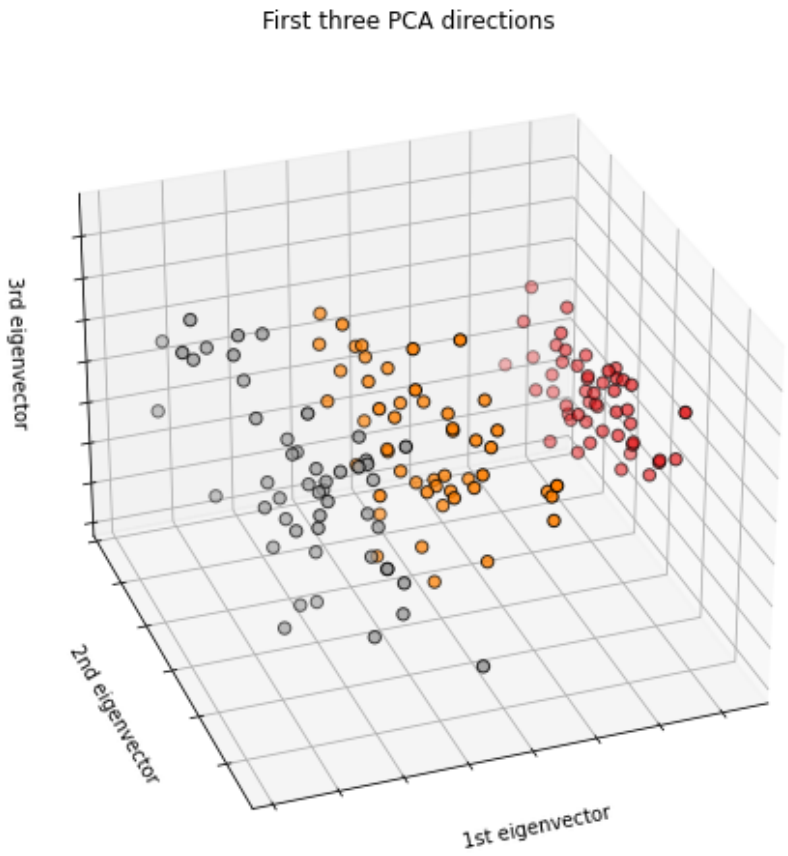
- Tiene varios parametros (¿Qué valores usamos?):
  - **C**: parametro de penalizacion para puntos "no separables".
  - **Kernel**: rbf, poly, linear
  - **Degree**: para el grado del kernel poly
  - **Gamma**: para el kernel rbf, (es la inversa del tamaño del radio del kernel)
  - ...

In [163…

```
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
```

```
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```



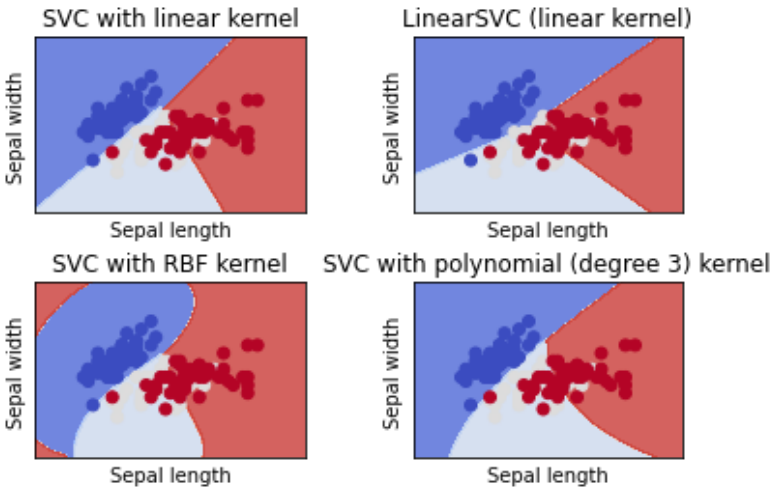


In [164…

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
y = iris.target

h = .02  # step size in the mesh

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
lin_svc = svm.LinearSVC(C=C).fit(X, y)

# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# title for the plots
titles = ['SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel']
```

```
for i, clf in enumerate((svc, lin_svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```

```
/home/cazz/Documents/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.
py:976: ConvergenceWarning: Liblinear failed to converge, increase the number
of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```



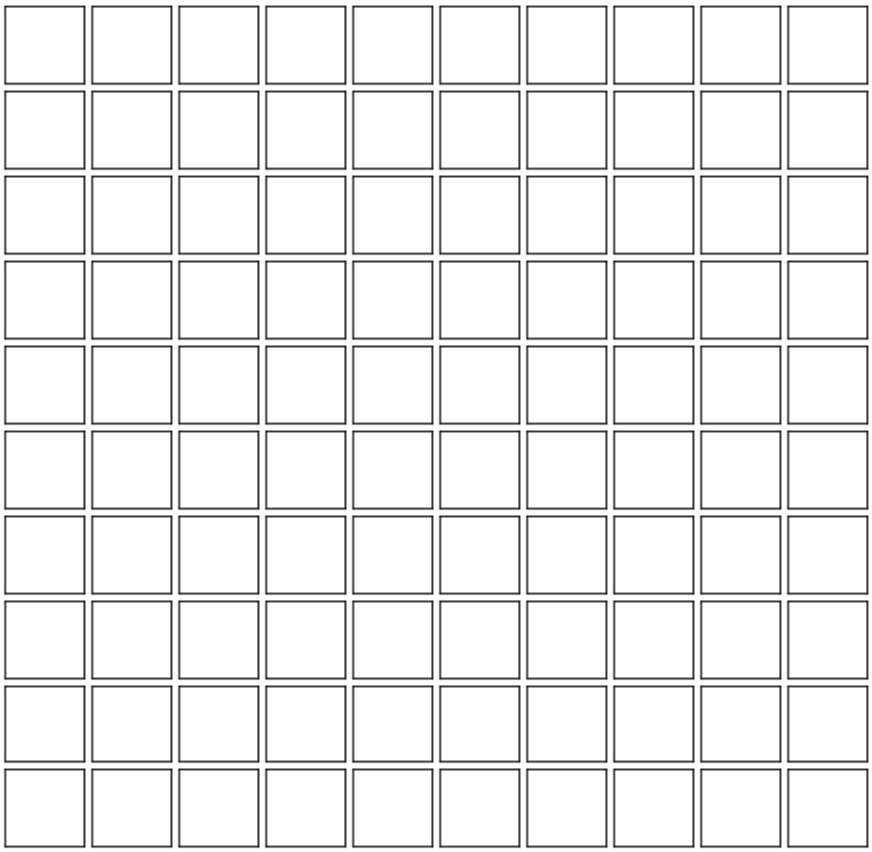
Ejemplo: reconocer digitos

In [181…

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

digits = load_digits()
print(digits.images.shape)
# print(digits.images)

fig, axes = plt.subplots(10, 10, figsize=(8,8),
                         subplot_kw={'xticks': [], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1)) # crea un p

for i, aux in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
```

```
    ax.text(0.05, 0.05, str(digits.target[i]), transform=ax.transAxes, color=
    ax.set_aspect("auto")
```

```
(1797, 8, 8)
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-181-67a51b6cad61> in <module>
     11
     12 for i, aux in enumerate(axes.flat):
---> 13     ax.imshow(digits.images[i], cmap='binary', interpolation='neares
t')
     14     ax.text(0.05, 0.05, str(digits.target[i]), transform=ax.transAxes
, color='green')
     15     ax.set_aspect("auto")

~/.local/lib/python3.8/site-packages/matplotlib/__init__.py in inner(ax, dat
a, *args, **kwargs)
   1445     def inner(ax, *args, data=None, **kwargs):
   1446         if data is None:
-> 1447             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1448
   1449         bound = new_sig.bind(ax, *args, **kwargs)

~/.local/lib/python3.8/site-packages/matplotlib/axes/_axes.py in imshow(self,
X, cmap, norm, aspect, interpolation, alpha, vmin, vmax, origin, extent, filt
ernorm, filterrad, resample, url, **kwargs)
   5516         if aspect is None:
   5517             aspect = rcParams['image.aspect']
-> 5518         self.set_aspect(aspect)
   5519         im = mimage.AxesImage(self, cmap, norm, interpolation, origi
n, extent,
   5520                               filternorm=filternorm, filterrad=filter
rad,

~/.local/lib/python3.8/site-packages/mpl_toolkits/mplot3d/axes3d.py in set_as
pect(self, aspect, adjustable, anchor, share)
    321         """
    322         if aspect != 'auto':
--> 323             raise NotImplementedError(
    324                 "Axes3D currently only supports the aspect argument "
    325                 f"'auto'. You passed in {aspect!r}."

NotImplementedError: Axes3D currently only supports the aspect argument 'aut
o'. You passed in 'equal'.
```



In [195…

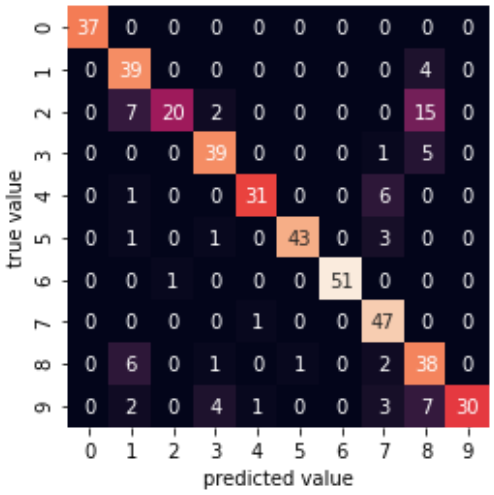
```
digits = load_digits()
X = digits.data
y = digits.target

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)

from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Out[195… 0.8333333333333334

In [200…

```
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(ytest, y_model)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value')
```

Out[200… Text(91.68, 0.5, 'true value')



Ajuste de los datos:

In [203…

```
clf.fit(digits.data[:-1], digits.target[:-1])
```

Out[203… SVC(kernel='poly')

In [218…

```
# Ejemplo claisificar digitos
print(digits.data[0].reshape(-1,1))
clf.predict(digits.data[0].reshape(1,-1))
clf.predict(digits.data[-1:])
```

```
[[ 0.]
 [ 0.]
 [ 5.]
 [13.]
 [ 9.]
 [ 1.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [13.]
 [15.]
 [10.]
 [15.]
 [ 5.]
 [ 0.]
 [ 0.]
 [ 3.]
 [15.]
 [ 2.]
 [ 0.]
 [11.]
 [ 8.]
 [ 0.]
 [ 0.]
 [ 4.]
 [12.]
 [ 0.]
 [ 0.]
 [ 8.]
 [ 8.]
 [ 0.]
 [ 0.]
 [ 5.]
 [ 8.]
```

```
[ 0.]
[ 0.]
[ 9.]
[ 8.]
[ 0.]
[ 0.]
[ 4.]
[11.]
[ 0.]
[ 1.]
[12.]
[ 7.]
[ 0.]
[ 0.]
[ 2.]
[14.]
[ 5.]
[10.]
[12.]
[ 0.]
[ 0.]
[ 0.]
[ 0.]
[ 6.]
[13.]
[10.]
[ 0.]
[ 0.]
[ 0.]]
```

Out[218… `array([8])`

Parametros del modelo (I)

- En el ejemplo, gamma aparece con un valor fijo.
- Los algoritmos tienen parametros que se pueden estimar:
  - Grid search
  - Random search
- Ejemplo:

```
param_grid = {'C': [1e3, 5e3, 1e4, 1e5], 'gamma':[0.0001, 0.0005,
0.0001, 0.0005, 0.01, 0.1]}
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
param_grid)
clf. clf.fit(X_train, y_train)
print("Best estimator found by grid search:")
print(clf.best_estimator_)
```

Parametros del modelo (II)

In [224…

```
import numpy as np
from scipy.stats import uniform as sp_rand
from sklearn import datasets
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV

dataset = datasets.load_diabetes()
param_grid = {'alpa': sp_rand()} # Una distribucion uniforme para muestrear a
model = Ridge() # Crea y ajusta usando regresión ridge regression model, con
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
rsearch.fit(dataset.data, dataset.target)
print(rsearch)
# resultados de la busqueda aleatoria del parametro
```

```
print(rsearch.best_socre)
print(rsearch.best_estimator_alpha)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-224-8530ae60747c> in <module>
      9 model = Ridge() # Crea y ajusta usando regresión ridge regression mod
el, con alpha
     10 rsearch = RandomizedSearchCV(estimator=model, param_distributions=par
am_grid, n_iter=100)
---> 11 rsearch.fit(dataset.data, dataset.target)
     12 print(rsearch)
     13 # resultados de la busqueda aleatoria del parametro

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py
in inner_f(*args, **kwargs)
     71                           FutureWarning)
     72         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)
})
---> 73         return f(**kwargs)
     74     return inner_f
     75

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_se
arch.py in fit(self, X, y, groups, **fit_params)
    734                 return results
    735
--> 736             self._run_search(evaluate_candidates)
    737
    738         # For multi-metric evaluation, store the best_index_, best_pa
rams_ and

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_se
arch.py in _run_search(self, evaluate_candidates)
   1527     def _run_search(self, evaluate_candidates):
   1528         """Search n_iter candidates from param_distributions"""
-> 1529         evaluate_candidates(ParameterSampler(
   1530             self.param_distributions, self.n_iter,
   1531             random_state=self.random_state))

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_se
arch.py in evaluate_candidates(candidate_params)
    706                                 n_splits, n_candidates, n_candidates *
 n_splits))
    707
--> 708                 out = parallel(delayed(_fit_and_score)(clone(base_est
imator),
    709                                                        X, y,
    710                                                        train=train, t
est=test,

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in __cal
l__(self, iterable)
   1027             # remaining jobs.
   1028             self._iterating = False
-> 1029             if self.dispatch_one_batch(iterator):
   1030                 self._iterating = self._original_iterator is not None
   1031

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in dispa
tch_one_batch(self, iterator)
    845                 return False
    846             else:
--> 847                 self._dispatch(tasks)
    848                 return True
    849

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in _disp
```

```
atch(self, batch)
    763         with self._lock:
    764             job_idx = len(self._jobs)
--> 765             job = self._backend.apply_async(batch, callback=cb)
    766             # A job can complete so quickly than its callback is
    767             # called before we get here, causing self._jobs to

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.p
y in apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/_parallel_backends.p
y in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in __cal
l__(self)
    250         # change the default number of processes to -1
    251         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 252             return [func(*args, **kwargs)
    253                     for func, args, kwargs in self.items]
    254

~/Documents/anaconda3/lib/python3.8/site-packages/joblib/parallel.py in <list
comp>(.0)
    250         # change the default number of processes to -1
    251         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 252             return [func(*args, **kwargs)
    253                     for func, args, kwargs in self.items]
    254

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_va
lidation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbose,
 parameters, fit_params, return_train_score, return_parameters, return_n_test
_samples, return_times, return_estimator, error_score)
    518             cloned_parameters[k] = clone(v, safe=False)
    519
--> 520         estimator = estimator.set_params(**cloned_parameters)
    521
    522     start_time = time.time()

~/Documents/anaconda3/lib/python3.8/site-packages/sklearn/base.py in set_para
ms(self, **params)
    247             key, delim, sub_key = key.partition('__')
    248             if key not in valid_params:
--> 249                 raise ValueError('Invalid parameter %s for estimator
 %s. '
    250                                  'Check the list of available paramet
ers '
    251                                  'with `estimator.get_params().keys()
`.' %

ValueError: Invalid parameter alpa for estimator Ridge(). Check the list of a
vailable parameters with `estimator.get_params().keys()`.
```

Guardar y recuperar un ajuste

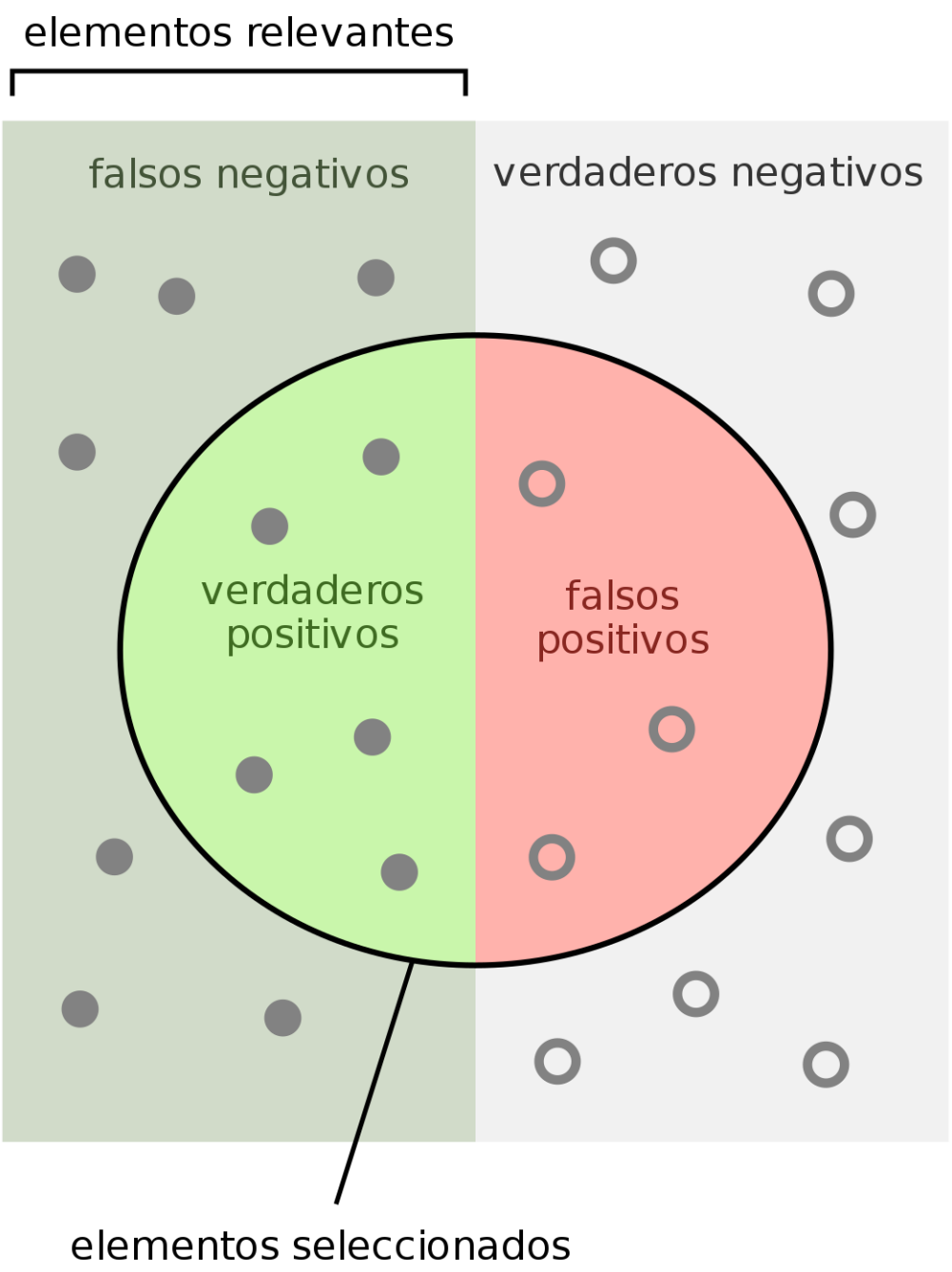
- Modulo pickle

```
import pickle
s = pickle.dumps(clf)
clf2 = pickle.loads(s)
```

- Es mas eficiente usar **joblib** (con grandes cantidades de datos)

```
from sklearn.externals import joblib
joblib.dump(clf, 'filename.pkl')
clf = joblib.load('filename.pkl')
```

Calidad de la clasificación



¿Cuántos objetos relevantes se seleccionaron?
i.e. Cuantas personas enfermas son identificadas como

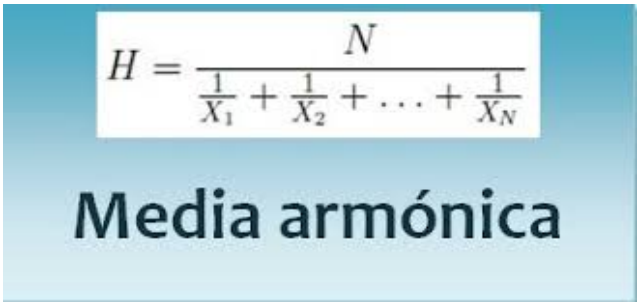
¿Cuántos elementos negativos se identifican como negativos?
i.e. Cuantas personas sanas son identificadas como no enfermas.

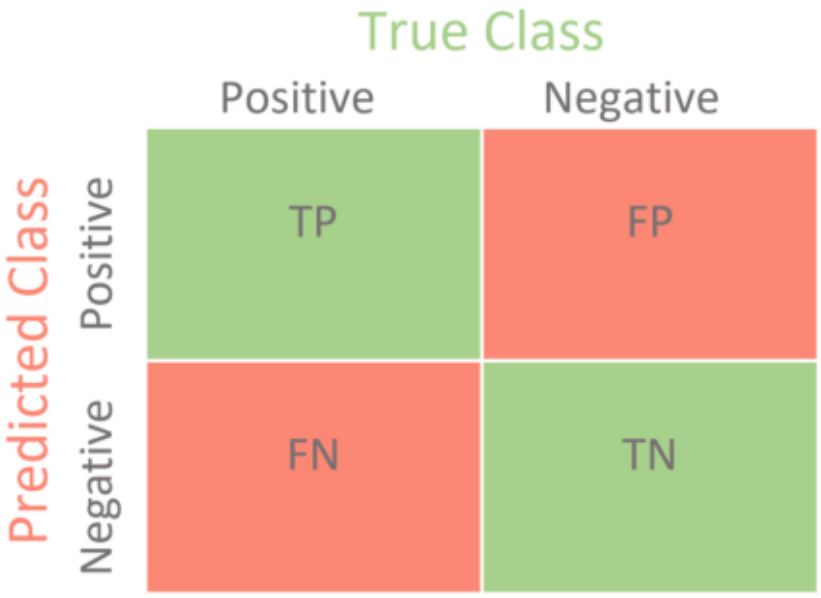
tales:

Sensibilidad=  | Especificidad = 

Medias (I)

- **F1 score**: Es la media armónica entre precisión y sensibilidad (recall)





$$F_1\text{–score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

Página para mirar medidas https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

In [230…

```python
import numpy as np
from sklearn.metrics import accuracy_score

y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2 ,3]

print(accuracy_score(y_true, y_pred))
print(accuracy_score(y_true, y_pred, normalize=False))

from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()
X, y = iris.data, iris.target

clf = svm.SVC(probability=True, random_state=0)
cross_val_score(clf, X, y, scoring='accuracy')
# Support es el numero de ocurrencias de cada clase
```

```
0.5
2
```

Out[230… `array([0.96666667, 0.96666667, 0.96666667, 0.93333333, 1.        ])`