

Algebra computacional con Sympy

Contenido

- Algebra computacional con **Sympy**
- Cálculo

Basado en [Taming math and physics using SymPy](#) y [Sympy : Symbolic Mathematics in Python](#)

Conceptos básicos de **Sympy**

```
# Importar librerías de cálculo matemático
from sympy import *
import numpy as np

# Importar librería para gráficos
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Forzar una expresión como simbólica
expr = S("1/7")
print ("Tipo expresión: ", expr, type(expr))

# Obtener su valor numérico
num = N("1/7")
print ("Tipo numérico: ", num, type(num))

# Operaciones para Racional y Flotante de sympy
print (expr+1)
print (num+1)
```

```
Tipo expresión: 1/7 <class 'sympy.core.numbers.Rational'>
Tipo numérico: 0.142857142857143 <class 'sympy.core.numbers.Float'>
8/7
1.14285714285714
```

```
# Cadena de texto sencilla
x='1/7'
print(type(x))

# Expresión simbólica de Sympy, usando comando S("expresion")
y=S('1/7')
print(type(y))
```

```
<class 'str'>
<class 'sympy.core.numbers.Rational'>
```

```
# Ejemplos de expresiones y operaciones con las mismas
## Como racional
print(S('1/7'))

## Como numérico
print(N('1/7'))

## Operación como racional
print(S('1/7')+1)

## Operación como numérico
print(N('1/7')+1)

## Tipos racional y flotante
print(type(S('1/7')),type(N('1/7')))
```

```
1/7
0.142857142857143
8/7
1.14285714285714
<class 'sympy.core.numbers.Rational'> <class 'sympy.core.numbers.Float'>
```

```
# pi es la expresión de Sympy, mientras que np.pi es la de Numpy

print(pi, "\t", type(pi))
print(pi + 1, "\t", type(pi+1))
print(N(pi+1), "\t", type(N(pi+1)))
print(np.pi, "\t", type(np.pi))
print(np.pi+1, "\t", type(np.pi+1))
```

```

pi      <class 'sympy.core.numbers.Pi'>
1 + pi  <class 'sympy.core.add.Add'>
4.14159265358979  <class 'sympy.core.numbers.Float'>
3.141592653589793 <class 'float'>
4.141592653589793 <class 'float'>

```

```

# Poniendo a Sympy a que imprima resultados bonitos
init_printing(use_latex=True)
pi, expr

```

$(\pi, 1/7)$

```

# Poniendo a Sympy a que imprima resultados NO bonitos
init_printing(use_latex=False)
pi, expr

```

$(\pi, 1/7)$

Expresiones

```

# Usando variables regulares de Python
t=2
y=3*t
y

```

6

```

# Tiene el tag: raises-exception, para que aunque tiene error continue compilando el resto del código
# Usando variables simbólicas sin Sympy

expr = 2*x + 3*x - sin(x) - 3*x + 42

```

```

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-47-519f78f4039f> in <module>
      3 # Usando variables simbólicas sin Sympy
      4
----> 5 expr = 2*x + 3*x - sin(x) - 3*x + 42

TypeError: unsupported operand type(s) for -: 'str' and 'sin'

```

Definir variables

```

# Definir variables simbólicas con Sympy

## Definir las variables
x,y,z = symbols("x y z")

## Ahora sí, escribir la expresión
expr = 2*x + 3*x - sin(x) - 3*x + 42
expr

```

$2 \cdot x - \sin(x) + 42$

```

nu, mu = symbols("n, m")
nu*2+mu

```

$m + 2 \cdot n$

```

g, a = symbols("\gamma \alpha")
g*2+a

```

$\alpha + 2 \cdot \gamma$

```

r, p, t = symbols("r, \phi, \theta")
2*r+3*p+5*t

```

$3 \cdot \phi + 5 \cdot \theta + 2 \cdot r$

```
## Revisión de tipos luego de Definir las variables x,y,z

w=3
print(type(w))
print(type(x))
```

```
<class 'int'>
<class 'sympy.core.symbol.Symbol'>
```

```
# Poniendo a Sympy a que imprima resultados bonitos, de esta celda en adelante

init_printing(use_latex=True)
```

Expandir y factorizar

```
# Expandir

expand( (x+3)**2 )
```

$$x^2 + 6x + 9$$

```
# Factorizar

factor( x**2 + 5*x + 6 )
```

$$(x + 2)(x + 3)$$

Sustituciones

```
# Definir la expresión

expr = (sin(x) + cos(y)) / 2
expr
```

$$\sin(x)/2 + \cos(y)/2$$

```
# Sustituir "x" por 1, e "y" por 2  
  
expr.subs({x: 1, y:2})
```

$$\cos(2)/2 + \sin(1)/2$$

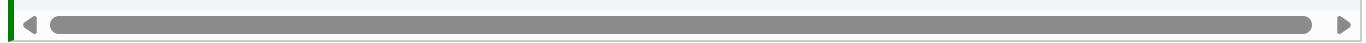
```
# Obtener el valor numérico flotante  
  
N(expr.subs({x: 1, y:2}))
```

$$0.212662074130377$$

```
# Sustituir "x" por otra expresión  
  
expr.subs({x: y**2+1})
```

$$\sin(y^2 + 1)/2 + \cos(y)/2$$

```
# Sustituir por otra expresión, y la única variable "y", sustituirla por el número 2  
  
expr.subs({x: y**2+1}).subs({y: 2})
```


$$\sin(5)/2 + \cos(2)/2$$

```
# Obtener el valor numérico flotante, dos opciones  
  
print(N(expr.subs({x: y**2+1}).subs({y: 2})))  
print(expr.subs({x: y**2+1}).subs({y: 2}).n())
```

$$\begin{aligned}-0.687535555605140\\-0.687535555605140\end{aligned}$$

Prueba de igualdad

```
# Definimos algunas expresiones
```

```
p = (x-5)*(x+5)
q = x**2 - 25
z = (x**2-9)/(x-3)
w = x+3
```

```
# No se puede comparar directamente usando el ==
```

```
print(p == q)
print(p - q == 0 )
```

False
False

```
print(z == w)
print(z - w == 0 )
```

False
False

```
simplify(z)
```

$x + 3$

```
# Ejemplo 1
```

```
print(simplify(p - q))
print(simplify(p - q) == 0)
```

0
True

```
# Ejemplo 2
```

```
print(simplify(z))
print(simplify(z)==w)
```

```
x + 3
True
```

```
# Ejemplo 3

expr1 = (x**2-9)/(x-3)
expr2 = (x+3)

expr1.equals(expr2)
```

```
True
```

```
# Verificación del ejemplo 3

simplify(expr1)
```

$x + 3$

Solucionadores

```
# Encontrar el valor de X
solve(5*x-10)
```

[2]

```
# Encontrar el valor de y
solve(5*x-x**2-y,y)
```

$[x(5 - x)]$

```
# Encontrar valores de x
s=solve(x**2 + 5*x +6)
s[0],s[1]
```

$(-3, - 2)$

```
# Tipo para lista y los elementos
type(s),type(s[0])
```

(list, sympy.core.numbers.Integer)

```
# Encontrar valores de x y mostrarlos
solve(x**2+5*x+6)
```

[-3, - 2]

```
# Encontrar valores de x y mostrarlos además del tipo de datos
sol = solve( x**2 + 2*x - 8, x)
sol[0], sol[1], type(sol[0]), type(sol[1])
```

(-4, 2, sympy.core.numbers.Integer, sympy.core.numbers.Integer)

```
# Generar una expresión usando un valor de la solución
sol[0]+x
```

$x - 4$

```
# Al agregarse un flotante a la suma se vuelve flotante el resultado
print(sol[0]+1.)
type(sol[0]+1.)
```

-3.000000000000000

sympy.core.numbers.Float

```
# Encontrando las famosas raíces de la ecuación cuadrática
a, b, c = symbols('a b c')
solve( a*x**2 + b*x + c, x)
```

$\left[\frac{(-b + \sqrt{-4ac + b^2})}{2a}, \frac{(-b - \sqrt{-4ac + b^2})}{2a} \right]$

```
# Encontrando sólo una de las raíces la ecuación cuadrática
sol = solve( a*x**2 + b*x + c, x)[0]
sol
```

$$(-b + \sqrt{-4ac + b^2})/2a$$

```
# Substituyendo por valores y obteniendo una raíz o número complejo
sol.subs({a:1, b:2, c:3})
```

$$-1 + \sqrt{2}i$$

```
# Substituyendo por valores y obteniendo un número real
sol.subs({a:1, b:5, c:6})
```

$$-2$$

Completar el cuadrado de modo que: $x^2 - 4x + 7 = (x - h)^2 + k$ para algunas constantes h y k .

Resolvemos $(x - h)^2 + k - (x^2 - 4x + 7) = 0$

```
# Resolver
h, k = symbols('h k')

# Encontrar valores de "h" y de "k"
solve( (x-h)**2 + k - (x**2-4*x+7), [h,k] )
```

$$[(2, 3)]$$

```
# Verificar que efectivamente h==2 y que k==3
((x-2)**2 + 3).expand()
```

$$x^2 - 4x + 7$$

Solucionar el sistema de ecuaciones:

$$x^2 + y = 0$$

$$3y - x = 0$$

```
# Se resuelve indicandole las dos ecuaciones
solve([x**2+y, 3*y-x])
```

```
[{x: -1/3, y: -1/9}, {x: 0, y: 0}]
```

```
# Otro ejemplo de un sistema de ecuaciones
print(solve([x+y-6, x-y-2]))
```

```
{x: 4, y: 2}
```

```
# Ejemplo de solución de una inecuación
print(solve(x-2>3))
```

```
(5 < x) & (x < oo)
```

```
# Explorar más a profundidad la documentación
help (solve)
```

Help on function solve in module sympy.solvers.solvers:

```
solve(f, *symbols, **flags)
    Algebraically solves equations and systems of equations.
```

Explanation

=====

Currently supported:

- polynomial
- transcendental
- piecewise combinations of the above
- systems of linear and polynomial equations
- systems containing relational expressions

Examples

=====

The output varies according to the input and can be seen by example:

```
>>> from sympy import solve, Poly, Eq, Function, exp
>>> from sympy.abc import x, y, z, a, b
>>> f = Function('f')
```

Boolean or univariate Relational:

```
>>> solve(x < 3)
(-oo < x) & (x < 3)
```

To always get a list of solution mappings, use flag dict=True:

```
>>> solve(x - 3, dict=True)
[{x: 3}]
>>> sol = solve([x - 3, y - 1], dict=True)
>>> sol
[{x: 3, y: 1}]
>>> sol[0][x]
3
>>> sol[0][y]
1
```

To get a list of *symbols* and set of solution(s) use flag set=True:

```
>>> solve([x**2 - 3, y - 1], set=True)
([x, y], {(-sqrt(3), 1), (sqrt(3), 1)})
```

Single expression and single symbol that is in the expression:

```
>>> solve(x - y, x)
[y]
>>> solve(x - 3, x)
[3]
```

```
>>> solve(Eq(x, 3), x)
[3]
>>> solve(Poly(x - 3), x)
[3]
>>> solve(x**2 - y**2, x, set=True)
([x], {(-y,), (y,)})
>>> solve(x**4 - 1, x, set=True)
([x], {(-1,), (1,), (-I,), (I,)})
```

Single expression with no symbol that is in the expression:

```
>>> solve(3, x)
[]
>>> solve(x - 3, y)
[]
```

Single expression with no symbol given. In this case, all free *symbols* will be selected as potential *symbols* to solve for. If the equation is univariate then a list of solutions is returned; otherwise - as is the case when *symbols* are given as an iterable of length greater than 1 - a list of mappings will be returned:

```
>>> solve(x - 3)
[3]
>>> solve(x**2 - y**2)
[{x: -y}, {x: y}]
>>> solve(z**2*x**2 - z**2*y**2)
[{x: -y}, {x: y}, {z: 0}]
>>> solve(z**2*x - z**2*y**2)
[{x: y**2}, {z: 0}]
```

When an object other than a Symbol is given as a symbol, it is isolated algebraically and an implicit solution may be obtained. This is mostly provided as a convenience to save you from replacing the object with a Symbol and solving for that Symbol. It will only work if the specified object can be replaced with a Symbol using the subs method:

```
>>> solve(f(x) - x, f(x))
[x]
>>> solve(f(x).diff(x) - f(x) - x, f(x).diff(x))
[x + f(x)]
>>> solve(f(x).diff(x) - f(x) - x, f(x))
[-x + Derivative(f(x), x)]
>>> solve(x + exp(x)**2, exp(x), set=True)
([exp(x)], {(-sqrt(-x),), (sqrt(-x),)})
```



```
>>> from sympy import Indexed, IndexedBase, Tuple, sqrt
>>> A = IndexedBase('A')
>>> eqs = Tuple(A[1] + A[2] - 3, A[1] - A[2] + 1)
>>> solve(eqs, eqs.atoms(Indexed))
{A[1]: 1, A[2]: 2}
```

* To solve for a symbol implicitly, use implicit=True:

```
>>> solve(x + exp(x), x)
```

```
[ -LambertW(1)]
>>> solve(x + exp(x), x, implicit=True)
[-exp(x)]
```

* It is possible to solve for anything that can be targeted with subs:

```
>>> solve(x + 2 + sqrt(3), x + 2)
[-sqrt(3)]
>>> solve((x + 2 + sqrt(3), x + 4 + y), y, x + 2)
{y: -2 + sqrt(3), x + 2: -sqrt(3)}
```

* Nothing heroic is done in this implicit solving so you may end up with a symbol still in the solution:

```
>>> eqs = (x*y + 3*y + sqrt(3), x + 4 + y)
>>> solve(eqs, y, x + 2)
{y: -sqrt(3)/(x + 3), x + 2: -2*x/(x + 3) - 6/(x + 3) + sqrt(3)/(x + 3)}
>>> solve(eqs, y*x, x)
{x: -y - 4, x*y: -3*y - sqrt(3)}
```

* If you attempt to solve for a number remember that the number you have obtained does not necessarily mean that the value is equivalent to the expression obtained:

```
>>> solve(sqrt(2) - 1, 1)
[sqrt(2)]
>>> solve(x - y + 1, 1) # !\ -1 is targeted, too
[x/(y - 1)]
>>> [_.subs(z, -1) for _ in solve((x - y + 1).subs(-1, z), 1)]
[-x + y]
```

* To solve for a function within a derivative, use ``dsolve``.

Single expression and more than one symbol:

* When there is a linear solution:

```
>>> solve(x - y**2, x, y)
[(y**2, y)]
>>> solve(x**2 - y, x, y)
[(x, x**2)]
>>> solve(x**2 - y, x, y, dict=True)
[{y: x**2}]
```

* When undetermined coefficients are identified:

* That are linear:

```
>>> solve((a + b)*x - b + 2, a, b)
{a: -2, b: 2}
```

* That are nonlinear:

```
>>> solve((a + b)*x - b**2 + 2, a, b, set=True)
([a, b], {(-sqrt(2), sqrt(2)), (sqrt(2), -sqrt(2))})
```

* If there is no linear solution, then the first successful attempt for a nonlinear solution will be returned:

```
>>> solve(x**2 - y**2, x, y, dict=True)
[ {x: -y}, {x: y} ]
>>> solve(x**2 - y**2/exp(x), x, y, dict=True)
[ {x: 2*LambertW(-y/2)}, {x: 2*LambertW(y/2)} ]
>>> solve(x**2 - y**2/exp(x), y, x)
[ (-x*sqrt(exp(x)), x), (x*sqrt(exp(x)), x) ]
```

Iterable of one or more of the above:

* Involving relational or bools:

```
>>> solve([x < 3, x - 2])
Eq(x, 2)
>>> solve([x > 3, x - 2])
False
```

* When the system is linear:

* With a solution:

```
>>> solve([x - 3], x)
{x: 3}
>>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y)
{x: -3, y: 1}
>>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y, z)
{x: -3, y: 1}
>>> solve((x + 5*y - 2, -3*x + 6*y - z), z, x, y)
{x: 2 - 5*y, z: 21*y - 6}
```

* Without a solution:

```
>>> solve([x + 3, x - 3])
[]
```

* When the system is not linear:

```
>>> solve([x**2 + y - 2, y**2 - 4], x, y, set=True)
([x, y], {(-2, -2), (0, 2), (2, -2)})
```

* If no *symbols* are given, all free *symbols* will be selected and a list of mappings returned:

```
>>> solve([x - 2, x**2 + y])
[ {x: 2, y: -4} ]
>>> solve([x - 2, x**2 + f(x)], {f(x), x})
[ {x: 2, f(x): -4} ]
```

* If any equation does not depend on the symbol(s) given, it will be eliminated from the equation set and an answer may be given implicitly in terms of variables that were not of interest:

```
>>> solve([x - y, y - 3], x)
```

```
{x: y}
```

Additional Examples

```solve()`` with check=True (default) will run through the symbol tags to eliminate unwanted solutions. If no assumptions are included, all possible solutions will be returned:`

```
>>> from sympy import Symbol, solve
>>> x = Symbol("x")
>>> solve(x**2 - 1)
[-1, 1]
```

By using the positive tag, only one solution will be returned:

```
>>> pos = Symbol("pos", positive=True)
>>> solve(pos**2 - 1)
[1]
```

Assumptions are not checked when ```solve()``` input involves relational or bools.

When the solutions are checked, those that make any denominator zero are automatically excluded. If you do not want to exclude such solutions, then use the `check=False` option:

```
>>> from sympy import sin, limit
>>> solve(sin(x)/x) # 0 is excluded
[pi]
```

If `check=False`, then a solution to the numerator being zero is found:  $x = 0$ . In this case, this is a spurious solution since  $\sin(x)/x$  has the well known limit (without discontinuity) of 1 at  $x = 0$ :

```
>>> solve(sin(x)/x, check=False)
[0, pi]
```

In the following case, however, the limit exists and is equal to the value of  $x = 0$  that is excluded when `check=True`:

```
>>> eq = x**2*(1/x - z**2/x)
>>> solve(eq, x)
[]
>>> solve(eq, x, check=False)
[0]
>>> limit(eq, x, 0, '-')
0
>>> limit(eq, x, 0, '+')
0
```

### \*\*Disabling High-Order Explicit Solutions\*\*

When solving polynomial expressions, you might not want explicit solutions (which can be quite long). If the expression is univariate, ```CRootOf``` instances will be returned instead:

```
>>> solve(x**3 - x + 1)
[-1/((-1/2 - sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)) - (-1/2 -
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3, -(-1/2 +
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3 - 1/((-1/2 +
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)), -(3*sqrt(69)/2 +
27/2)**(1/3)/3 - 1/(3*sqrt(69)/2 + 27/2)**(1/3)]
>>> solve(x**3 - x + 1, cubics=False)
[CRootOf(x**3 - x + 1, 0),
 CRootOf(x**3 - x + 1, 1),
 CRootOf(x**3 - x + 1, 2)]
```

If the expression is multivariate, no solution might be returned:

```
>>> solve(x**3 - x + a, x, cubics=False)
[]
```

Sometimes solutions will be obtained even when a flag is False because the expression could be factored. In the following example, the equation can be factored as the product of a linear and a quadratic factor so explicit solutions (which did not require solving a cubic expression) are obtained:

```
>>> eq = x**3 + 3*x**2 + x - 1
>>> solve(eq, cubics=False)
[-1, -1 + sqrt(2), -sqrt(2) - 1]
```

### \*\*Solving Equations Involving Radicals\*\*

Because of SymPy's use of the principle root, some solutions to radical equations will be missed unless check=False:

```
>>> from sympy import root
>>> eq = root(x**3 - 3*x**2, 3) + 1 - x
>>> solve(eq)
[]
>>> solve(eq, check=False)
[1/3]
```

In the above example, there is only a single solution to the equation. Other expressions will yield spurious roots which must be checked manually; roots which give a negative argument to odd-powered radicals will also need special checking:

```
>>> from sympy import real_root, S
>>> eq = root(x, 3) - root(x, 5) + S(1)/7
>>> solve(eq) # this gives 2 solutions but misses a 3rd
[CRootOf(7*x**5 - 7*x**3 + 1, 1)**15,
CRootOf(7*x**5 - 7*x**3 + 1, 2)**15]
>>> sol = solve(eq, check=False)
>>> [abs(eq.subs(x,i).n(2)) for i in sol]
[0.48, 0.e-110, 0.e-110, 0.052, 0.052]
```

The first solution is negative so ``real\_root`` must be used to see that it satisfies the expression:

```
>>> abs(real_root(eq.subs(x, sol[0])).n(2))
0.e-110
```

If the roots of the equation are not real then more care will be necessary to find the roots, especially for higher order equations. Consider the following expression:

```
>>> expr = root(x, 3) - root(x, 5)
```

We will construct a known value for this expression at  $x = 3$  by selecting the 1-th root for each radical:

```
>>> expr1 = root(x, 3, 1) - root(x, 5, 1)
>>> v = expr1.subs(x, -3)
```

The ``solve`` function is unable to find any exact roots to this equation:

```
>>> eq = Eq(expr, v); eq1 = Eq(expr1, v)
>>> solve(eq, check=False), solve(eq1, check=False)
([], [])
```

The function ``unrad``, however, can be used to get a form of the equation for which numerical roots can be found:

```
>>> from sympy.solvers.solvers import unrad
>>> from sympy import nroots
>>> e, (p, cov) = unrad(eq)
>>> pvals = nroots(e)
>>> inversion = solve(cov, x)[0]
>>> xvals = [inversion.subs(p, i) for i in pvals]
```

Although ``eq`` or ``eq1`` could have been used to find ``xvals``, the solution can only be verified with ``expr1``:

```
>>> z = expr - v
>>> [xi.n(chop=1e-9) for xi in xvals if abs(z.subs(x, xi).n()) < 1e-9]
[]
>>> z1 = expr1 - v
>>> [xi.n(chop=1e-9) for xi in xvals if abs(z1.subs(x, xi).n()) < 1e-9]
[-3.0]
```

## Parameters

=====

f :

- a single Expr or Poly that must be zero
- an Equality
- a Relational expression
- a Boolean
- iterable of one or more of the above

symbols : (object(s) to solve for) specified as

- none given (other non-numeric objects will be used)
- single symbol
- denested list of symbols  
(e.g., ``solve(f, x, y)``)
- ordered iterable of symbols  
(e.g., ``solve(f, [x, y])``)

```

flags :
 dict=True (default is False)
 Return list (perhaps empty) of solution mappings.
 set=True (default is False)
 Return list of symbols and set of tuple(s) of solution(s).
 exclude=[] (default)
 Do not try to solve for any of the free symbols in exclude;
 if expressions are given, the free symbols in them will
 be extracted automatically.
 check=True (default)
 If False, do not do any testing of solutions. This can be
 useful if you want to include solutions that make any
 denominator zero.
 numerical=True (default)
 Do a fast numerical check if *f* has only one symbol.
 minimal=True (default is False)
 A very fast, minimal testing.
 warn=True (default is False)
 Show a warning if ``checksol()`` could not conclude.
 simplify=True (default)
 Simplify all but polynomials of order 3 or greater before
 returning them and (if check is not False) use the
 general simplify function on the solutions and the
 expression obtained when they are substituted into the
 function which should be zero.
 force=True (default is False)
 Make positive all symbols without assumptions regarding sign.
 rational=True (default)
 Recast Floats as Rational; if this option is not used, the
 system containing Floats may fail to solve because of issues
 with polys. If rational=None, Floats will be recast as
 rationals but the answer will be recast as Floats. If the
 flag is False then nothing will be done to the Floats.
 manual=True (default is False)
 Do not use the polys/matrix method to solve a system of
 equations, solve them one at a time as you might "manually."
 implicit=True (default is False)
 Allows ``solve`` to return a solution for a pattern in terms of
 other functions that contain that pattern; this is only
 needed if the pattern is inside of some invertible function
 like cos, exp, ect.
 particular=True (default is False)
 Instructs ``solve`` to try to find a particular solution to a linear
 system with as many zeros as possible; this is very expensive.
 quick=True (default is False)
 When using particular=True, use a fast heuristic to find a
 solution with many zeros (instead of using the very slow method
 guaranteed to find the largest number of zeros possible).
 cubics=True (default)
 Return explicit solutions when cubic expressions are encountered.
 quartics=True (default)
 Return explicit solutions when quartic expressions are encountered.
 quintics=True (default)
 Return explicit solutions (if possible) when quintic expressions
 are encountered.

```

## See Also

=====

rsolve: For solving recurrence relationships

dsolve: For solving differential equations

**Sympy a Python y Numpy**Ver [Sympy Numeric Computation](#)

```
Definir las variables simbólicas
x,y,z = symbols("x y z")

Tomar la primera solución
sol = solve(a*x**2 + b*x + c, x)[0]
print("Solución 1 simbólica",sol)

Evaluar para un caso particular
s1 = N(sol.subs({a:1, b:5, c:-3}))
print("Solución 1 caso particular: ",s1)

Obtener la raíz cuadrada con sympy
ss1 = sqrt(s1)
print (ss1, type(ss1))

Obtener la raíz cuadrada con numpy
ns1 = np.sqrt(float(s1), dtype=complex) # Se le debe indicar para complejo
print (ns1, type(ns1))
```

```
Solución 1 simbólica (-b + sqrt(-4*a*c + b**2))/(2*a)
Solución 1 caso particular: 0.541381265149110
0.735786154496746 <class 'sympy.core.numbers.Float'>
(0.7357861544967463+0j) <class 'numpy.complex128'>
```

```
Uso de Lambdify
Este módulo proporciona funciones convenientes para transformar expresiones SymPy
en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamente

Ejemplo 1
exp1 = x**2+5*x+6
f1 = lambdify(x, exp1)
print(f1(-1))

Ejemplo 2
exp2 = (sin(x) + x**2)/2
f2 = lambdify(x, exp2)
print(f2(10))

Ejemplo 3
print(f1(np.array([1,2,3])))
print([f2(i) for i in [1,2,3]])
```

2  
49.72798944455531  
[12 20 30]  
[0.9207354924039483, 2.454648713412841, 4.570560004029933]

```
Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit f2(np.arange(100))
```

46.1 µs ± 1.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)  
8.58 µs ± 509 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

```
Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit [f2(i) for i in range(100)]
```

46.3 µs ± 503 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)  
321 µs ± 124 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

```
Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%%timeit f1(np.arange(100)) # No funciona pues no usa "Numpy"
%timeit f2(np.arange(100))
```

9.49 µs ± 1.35 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)

```
Uso de Lambdify para una FUNCIÓN DE DOS (2) ARGUMENTOS
Este módulo proporciona funciones convenientes para transformar expresiones SymPy
en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamente

Definir la expresión
expr = (sin(x) + cos(y))/2

Conversión la función con lambdify
f = lambdify([x, y], expr, "numpy")

Evaluar de a un valor
print(f(10,1))

Conjunto de valores
print(f([10,10],[1,1]))
```

-0.0018594025106150047  
[-0.0018594 -0.0018594]

## Sumatoria

Resolver la si siguiente sumatoria  $sum = \sum_{i=1}^n i$

```
Definimos los símbolos que vamos a usar
n, i = symbols('n i')

Definimos la sumatoria
S = summation(i, (i, 1, n))

Mostramos el resultado
print(factor(S))
```

```
n*(n + 1)/2
```

```
Evaluamos la sumatoria para n = 3
print(S.subs(n, 3))
```

6

## Derivar

```
diff(x**2)
```

$2x$

```
diff(x**2+x*y, x)
```

$2x + y$

```
diff(x**2+x*y, y)
```

$x$

```
diff(exp(x))
```

$e^x$

```
La diferenciación conoce algunas reglas (por ejemplo, la regla del producto)
diff(x**2*sin(x))
```

$x^2\cos(x) + 2x\sin(x)$

## Integral

```
Integral indefinida
integrate(x**2)
```

$x^3/3$

```
Ejemplo 1 Integral definida
#integrate(expr, (variable,desde,hasta))
integrate(x**2, (x,0,1))
```

$1/3$

```
Ejemplo 2 Integral definida
integrate(x**2, (x,-1,3))
```

$28/3$

## Recta tangente a una función

La recta tangente a la función  $f(x)$  en  $x = x_0$  es la linea que pasa a través del punto  $(x_0, f(x_0))$  y tiene la misma pendiente que la función en ese punto. La recta tangente a la función  $f(x)$  en el punto  $x = x_0$  está descrita por la ecuación \$

$$T_1(x) = f(x_0) + f'(x_0)(x - x_0)$$

¿Cuál es la ecuación de la recta tangente a  $f(x) = \frac{1}{2}x^2$  en  $x_0 = 1$ ?

```
Crear las variables
x, f = symbols("x f")
```

```
Crear la función
f = 1./2*x**2
print ('f(x)\t= ', f)
```

$$f(x) = 0.5*x^2$$

```
Encontrar la derivada
df = diff(f,x)
print ("f'(x)\t=", df)
```

```
f'(x) = 1.0*x
```

```
Encontrar la recta tangente a la función
x0 = 1
T_1 = f.subs({x:x0}) + df.subs({x:x0}) * (x - x0)
print ("T1(x)\t=", T_1)
```

```
T1(x) = 1.0*x - 0.5
```

```
Comprobar que el valor en el eje "y" de la función y la recta tangente son iguales
print(f.subs({x:1}))
T_1.subs({x:1}) - f.subs({x:1})
```

```
0.500000000000000
```

0

```
Comprobar que el valor de la pendiente de la función y la recta tangente son iguales
print(diff(f,x).subs({x:1}))
diff(T_1,x).subs({x:1}) - diff(f,x).subs({x:1})
```

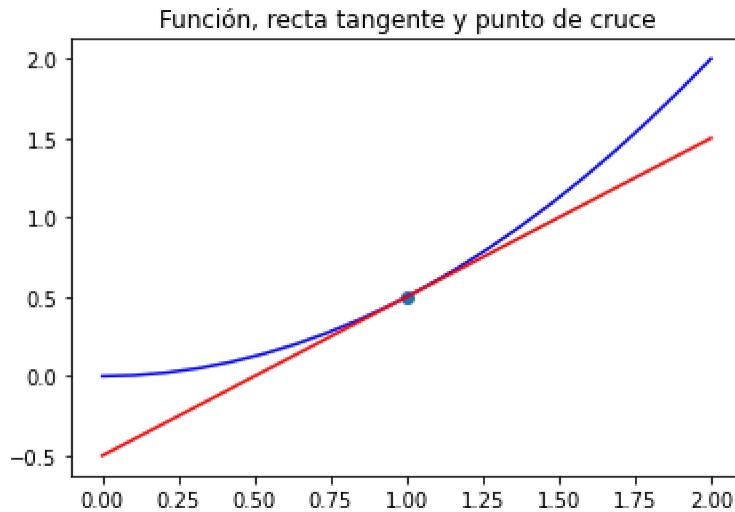
```
1.00000000000000
```

0

```
Obtener funciones de Python a partir de expresiones
fp = lambdify(x, f, "numpy")
tp = lambdify(x, T_1, "numpy")
```

```
Graficar
xr = np.linspace(0,2,20)
plt.plot(xr, fp(xr), color="blue") # función
plt.plot(xr, tp(xr), color="red") # recta tangente
plt.scatter(x0, fp(x0)) # punto de cruce, x0 == 1
plt.title("Función, recta tangente y punto de cruce")
```

Text(0.5, 1.0, 'Función, recta tangente y punto de cruce')



## Ecuaciones diferenciales

Resolver  $\frac{dy}{dt} = y(t) + t$

```
Crear variables
t, C1 = symbols("t C1")

Crear una función no definida con el parámetro cls=Function
y = symbols("y", cls=Function)

Crear una expresión con la función no definida y la variable
dydt = y(t)+t

Crear la ecuación a resolver
eq = dydt-diff(y(t),t)
eq
```

$$t + y(t) - \frac{d}{dt}y(t)$$

```
Resolver. Con dsolve se resuelven ecuaciones diferenciales
yt = dsolve(eq, y(t))
yt
```

$$y(t) = (C_1 + (-t - 1)e^{-t})e^t$$

```
Obtener el resultado
yt.rhs
```

$$(C_1 + (-t - 1)e^{-t})e^t$$

```
Verificar la solución, forma 1
simplify(dydt.subs({y(t): yt.rhs}) - diff(yt.rhs,t))
```

0

```
Verificar la solución, forma 2
expr1 = dydt.subs({y(t): yt.rhs})
expr2 = diff(yt.rhs,t)

expr1.equals(expr2)
```

True

```
Usando condición inicial y(0) = 2, obtener C1
eq1 = Eq(yt.rhs.subs({t:0}).evalf(), 2.) # Eq(f,g) -> f=g
sol = solve([eq1], [C1])
sol
```

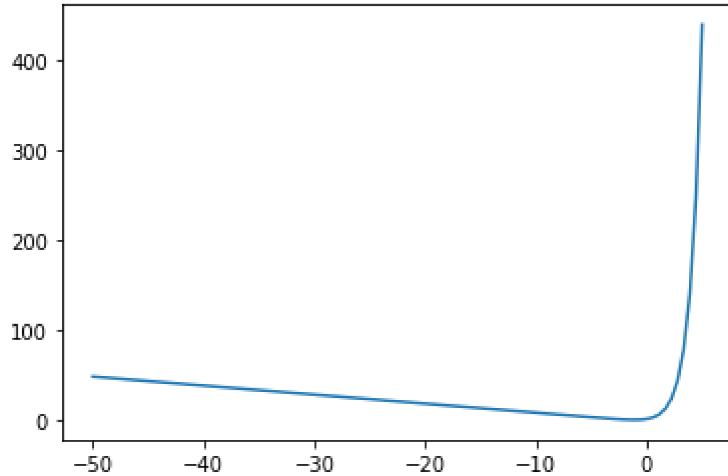
$$\{C_1 : 3.0\}$$

```
Definir función para C1=3, con la variable "t"
C1_val = 3
fY = lambdify(t, yt.rhs.subs({C1: C1_val}), "numpy") # C1_val == 3
print(yt.rhs.subs({C1: C1_val}))

Evaluar en un sólo valor de tiempo
print(fY(0))

Evaluar en varios valores de tiempo y graficar
t_vals = np.linspace(-50,5,100)
plt.figure()
plt.plot(t_vals, fY(t_vals))
plt.show()
```

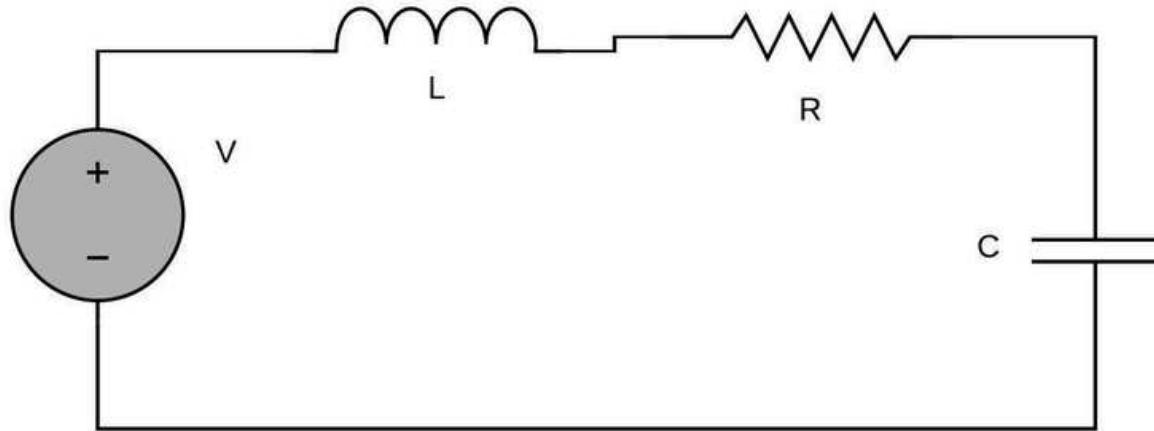
((-t - 1)\*exp(-t) + 3)\*exp(t)  
2.0



## Circuito RLC

Modelo:

$$\begin{aligned}\frac{di}{dt} &= -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi \\ \frac{dv}{dt} &= \frac{1}{C}i(t)\end{aligned}$$



```
Definir simbolos y funciones
t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)
didt=i.diff(t)
dvdt=v.diff(t)
```

```
Primera expresión
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -Ri(t)/L + vi/L - v(t)/L$$

```
Segunda expresión
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = i(t)/C$$

```
Sustitución, suponiendo valores conocidos
expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
Sustitución, suponiendo valores conocidos
expr2=expr2.subs(C,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
Resolver usando dsolve. Se encontrará i(t) y v(t)
s=dsolve([expr1,expr2])
s
```

$$i(t) = -(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2), \quad v(t) = C_1e^{-t/2}\cos(\sqrt{3}t/2) - C_2e^{-t/2}\sin(\sqrt{3}t/2) + \sin^2(\sqrt{3}t/2) + \cos^2(\sqrt{3}t/2)$$

```
Sustituir para un i(0)=0.1 y v(0)=0.1
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1) #i corriente
print(eq1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1) #v voltaje
print(eq2)
sol = solve([eq1,eq2], [C1,C2])
sol
```

$$\begin{aligned} & \text{Eq}(-0.5*C1 - 0.866025403784439*C2, 0.1) \\ & \text{Eq}(C1 + 1.0, 0.1) \end{aligned}$$

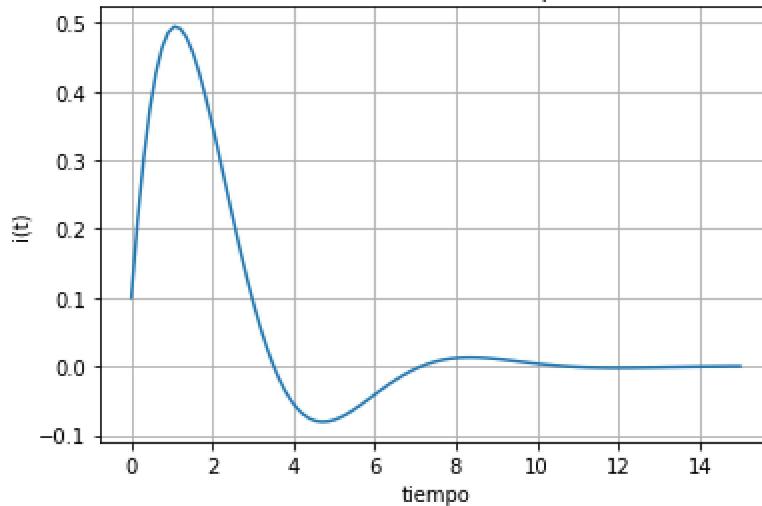
$$\{C_1 : -0.9, \quad C_2 : 0.404145188432738\}$$

```
Ecuación corriente
e1 = lambdify(t,s[0].rhs.subs(sol), 'numpy')
Ecuación voltaje
e2 = lambdify(t,s[1].rhs.subs(sol), 'numpy')
```

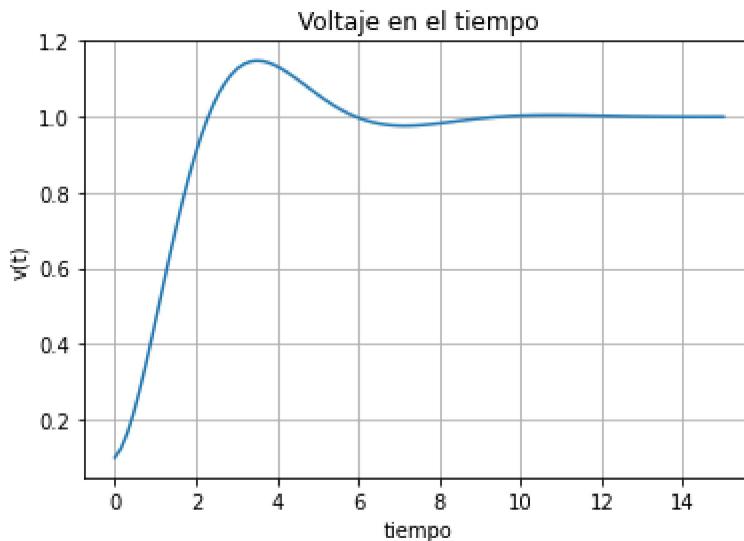
```
Vector de valores en el tiempo
t_vals = np.linspace(0,15,100)

Graficar
plt.figure()
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
plt.title("Corriente en el tiempo")
plt.show()
```

Corriente en el tiempo



```
Graficar
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
plt.title("Voltaje en el tiempo")
plt.show()
```



## Oscilador armónico amortiguado

```
Crear la ecuación del oscilador
t, w = symbols("t \omega_0", positive=True)
d = symbols("\xi", real=True)
x = Function("x", real=True)
eq = w**2*x(t) + 2*w*d*x(t).diff(t) + x(t).diff(t,t)
eq
```

$$\omega_0^2 x(t) + 2\omega_0 \xi \frac{dx}{dt} x(t) + \frac{d^2}{dt^2} x(t)$$

Evaluar y resolver para cuando  $\omega = 0$  y  $\xi = 1/10$

```
Evaluar para las condiciones iniciales
eq = eq.subs(w, 1).subs(d, Rational(1,10))

Resolver la ecuación diferencial
sol = dsolve(eq,x(t))
sol
```

$$x(t) = (C_1 \sin(3\sqrt{11}t/10) + C_2 \cos(3\sqrt{11}t/10)) e^{-t/10}$$

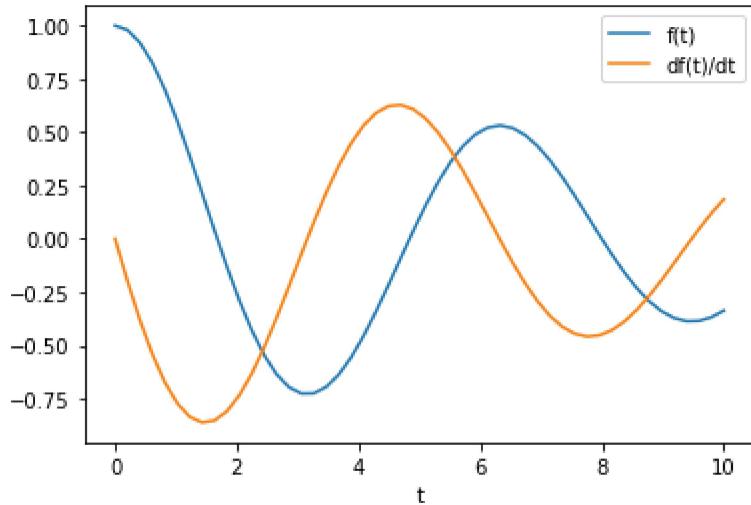
```
Usando estas condiciones iniciales x(0) = 1, y para dx(0)/dt = 0
C1, C2 = symbols("C1, C2")
const = solve([sol.rhs.subs(t,0) - 1, sol.rhs.diff(t).subs(t, 0)], [C1,C2])
print ("const =", const)

Reemplazar C1 y C2 en la solución
initvalue_solution = sol.rhs.subs(C1, const[C1]).subs(C2, const[C2])
print ("Aplicando las condiciones iniciales \nx(t) = ")
initvalue_solution
```

```
const = {C1: sqrt(11)/33, C2: 1}
Aplicando las condiciones iniciales
x(t) =
```

$$(\sqrt{11} \sin(3\sqrt{11}t/10)/33 + \cos(3\sqrt{11}t/10)) e^{-t/10}$$

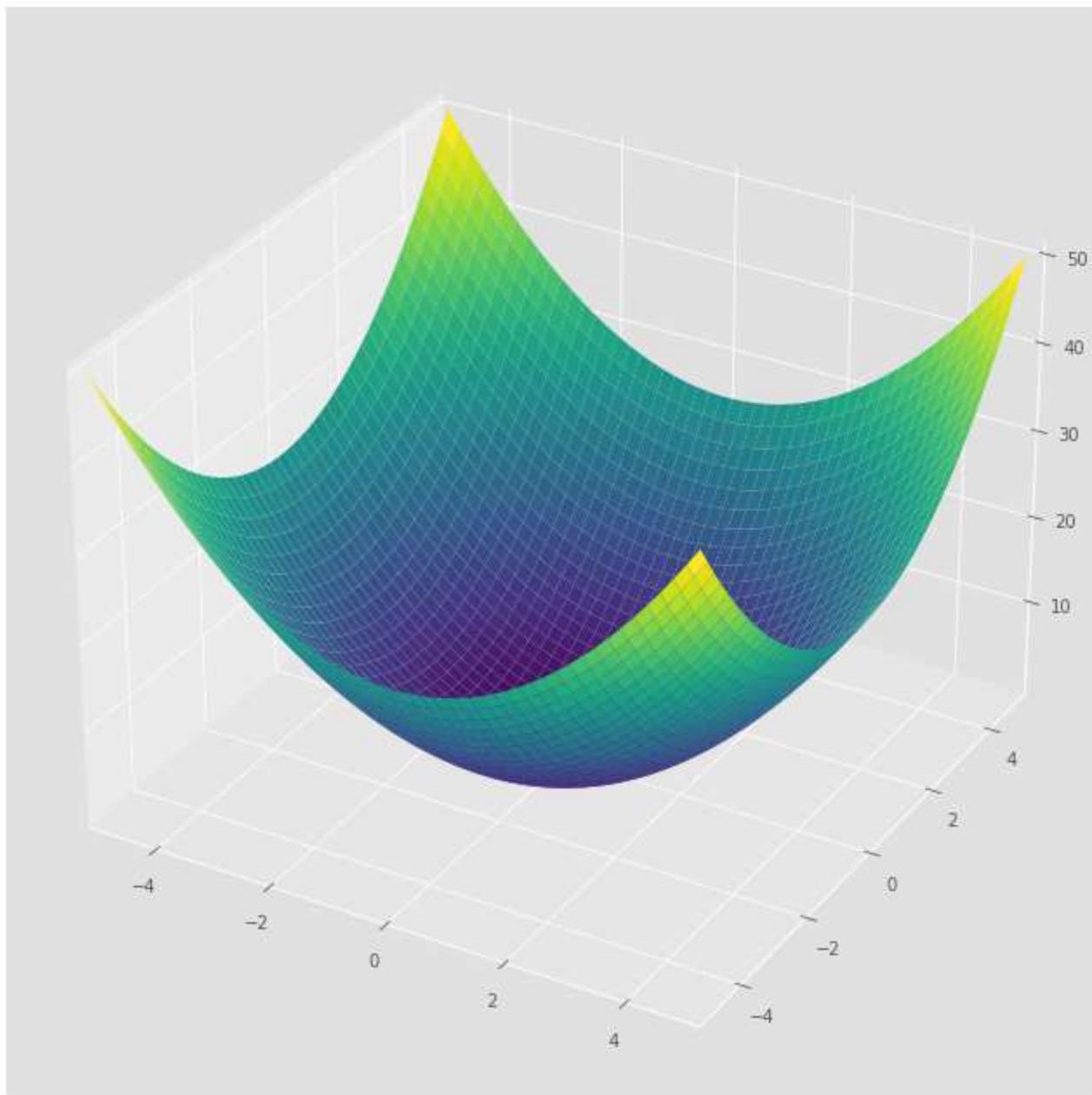
```
Graficar
T = np.linspace(0, 10)
F = lambdify(t, initvalue_solution, "numpy")
dFdT = lambdify(t, initvalue_solution.diff(), "numpy")
plt.plot(T, F(T))
plt.plot(T, dFdT(T))
plt.legend(("f(t)", "df(t)/dt"))
plt.xlabel("t")
plt.show()
```



## Gráfico en 3D

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
```

```
plot3d(x**2+y**2, (x, -5, 5), (y, -5, 5))
```



```
<sympy.plotting.plot.Plot at 0x7f0ced2dd760>
```

## Calculo del mínimo en una variable de forma analítica

```
x,y =symbols("x,y")
y=x**2+5*x+6
fp = lambdify(x, y, "numpy")
```

y

$$x^2 + 5x + 6$$

df=diff(y)

df

$$2x + 5$$

pc=solve(df)

N(pc[0])

$$-2.5$$

d2f=diff(df)

d2f

$$2$$

cy=y.subs({x:N(pc[0])})

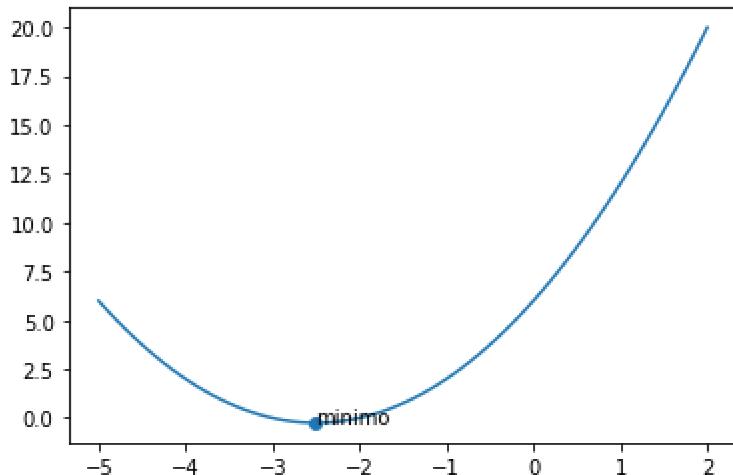
cy

$$-0.25$$

z=np.linspace(-5,2,100)

```
plt.plot(z,fp(z))
plt.scatter(pc[0],fp(pc[0]))
plt.text(pc[0],fp(pc[0]),'minimo')
```

```
Text(-5/2, -1/4, 'minimo')
```



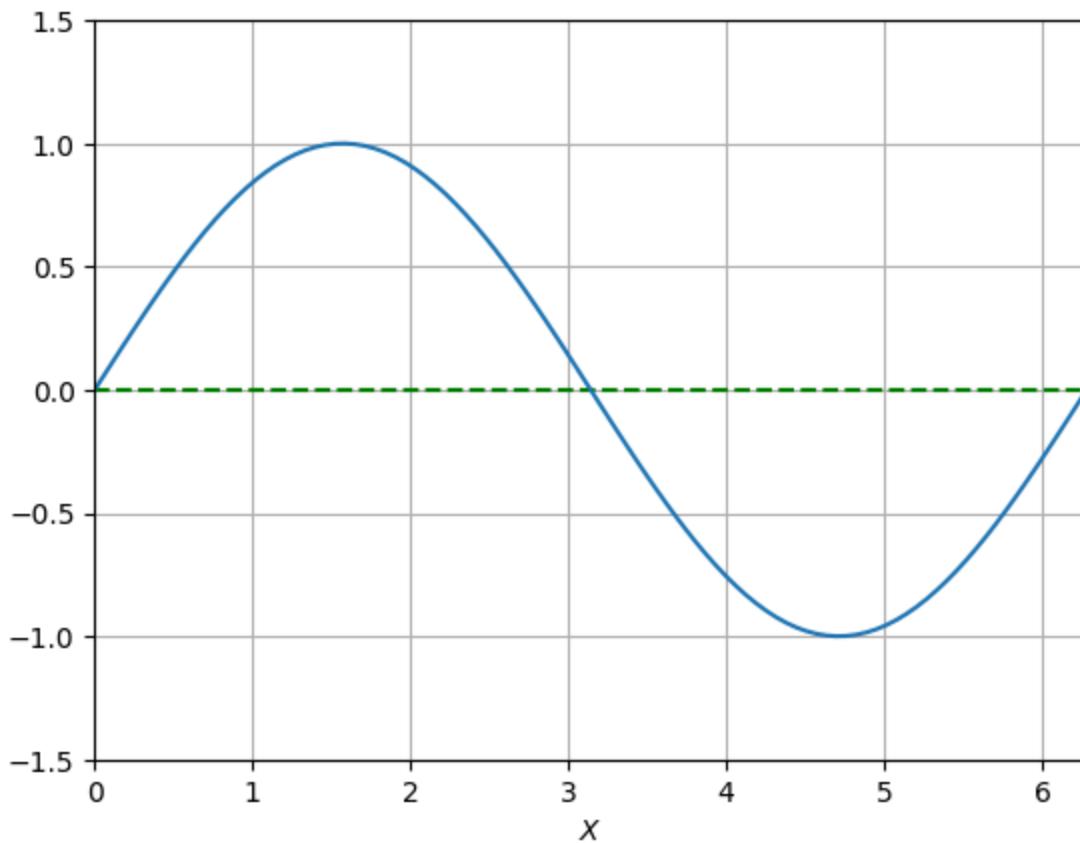
## Graficas dinámicas

```
Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML;
rc('animation', html='html5');
```

```
Plantilla sobre la cual se realiza el gráfico
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

fig, ax = plt.subplots()
ax.set_xlabel('X')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.5, 1.5)
ax.grid(True)
plt.plot([0, 2*np.pi], [0, 0], '--', color='green')
plt.plot(x,y)

Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea, = ax.plot([],[],'o',color = 'r', label = 'Curva sin(x)')
```



```
Realizamos una función que grafique el punto específico uno a uno

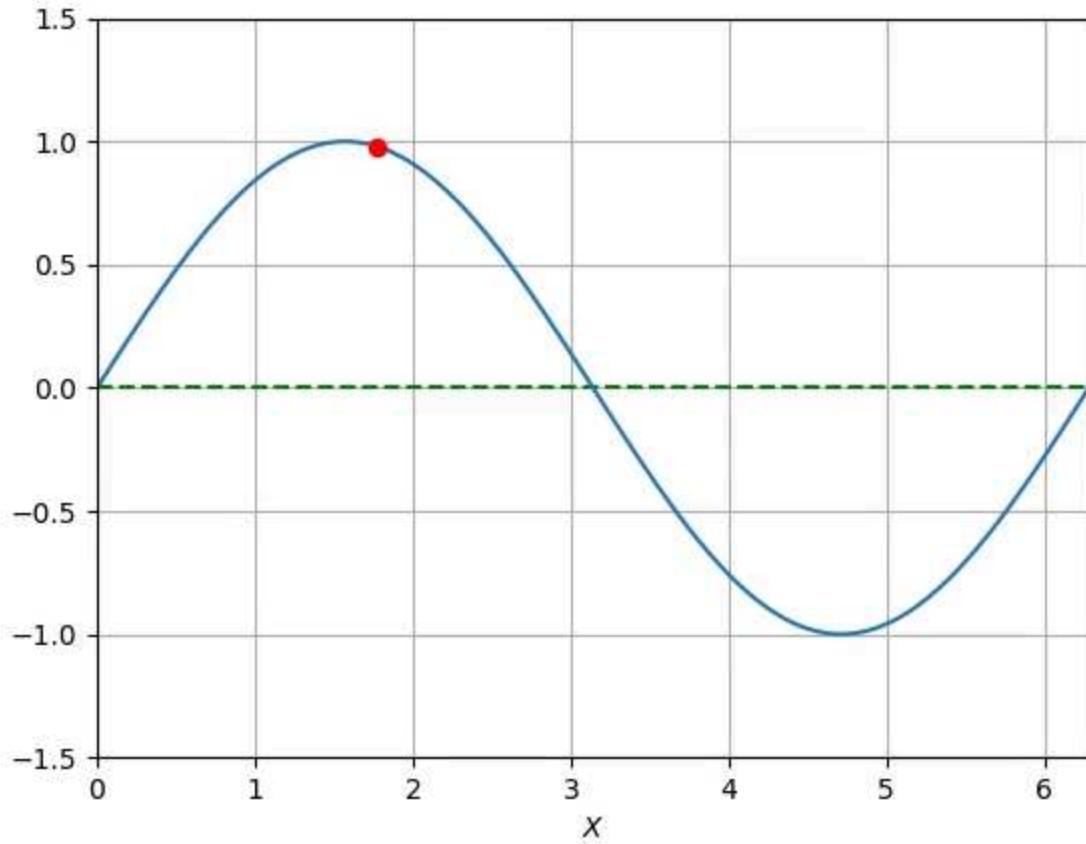
frames = 100;
def graficar(i):

 x = np.linspace(0, 2 * np.pi, 100)
 y = np.sin(x)

 linea.set_data(x[i],y[i])
 return (linea,)
```

```
Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=100)
anim
```

```
<ipython-input-56-d6bd03730612>:7: MatplotlibDeprecationWarning: Setting data with a n
 linea.set_data(x[i],y[i])
```



Enlace de interés para animación:

[Más sobre animaciones](#)

# Automatizaciones

## Contenido

- Automatizaciones
- Guardar mapas
- Envío de correos con python
- Dash

Con Python podemos llegar a realizar desde operaciones sencillas de datos hasta visualizaciones complejas o automatizaciones de procesos.

En Python hay muchas librerías y frameworks creados por la comunidad para resolver muchos procesos. Así que si tienes un proceso que quieras realizar, busca si ya existe una librería con la que puedas facilitarte la programación y en unas cuantas líneas de código.



```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```
!wget https://raw.githubusercontent.com/python-visualization/folium-example-data/main/
```

```
--2024-07-10 17:23:54-- https://raw.githubusercontent.com/python-visualization/folium
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 18
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 87688 (86K) [text/plain]
Saving to: 'us_states.json.1'
```

```
us_states.json.1 0%[] 0 ---KB/s
us_states.json.1 100%[=====] 85.63K ---KB/s in 0.02s
```

```
2024-07-10 17:23:54 (3.79 MB/s) - 'us_states.json.1' saved [87688/87688]
```

```
import pandas as pd

import folium
import json

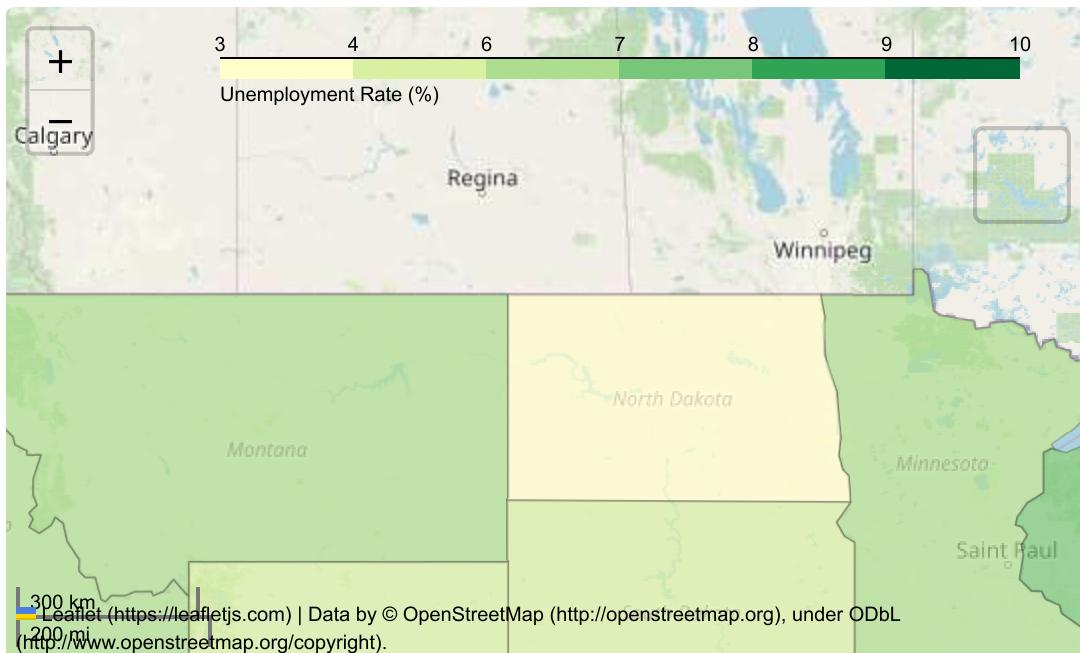
state_geo = json.load(open("us_states.json"))

state_data = pd.read_csv(
 "https://raw.githubusercontent.com/python-visualization/folium-example-data/main/u
)
state_data.head()
```

	State	Unemployment
0	AL	7.1
1	AK	6.8
2	AZ	8.1
3	AR	7.2
4	CA	10.1

```
m = folium.Map(location=[48, -102], zoom_start=5, width="80%", height="80%", control_scale=True)
Agreguemos la informacion del datafram en el mapa
folium.Choropleth(
 geo_data=state_geo, # data geográfica (JSON con los estados)
 name="choropleth", # nombre del mapa
 data=state_data, # datafram con los datos
 columns=["State", "Unemployment"], # columnas del datafram
 key_on="feature.id", # llave para unir los datos
 fill_color="YlGn", # colores de los estados
 fill_opacity=0.7, # opacidad del color
 line_opacity=0.2, # opacidad de la linea
 legend_name="Unemployment Rate (%)", # nombre de la leyenda
 highlight=True, # resaltar el estado al pasar el mouse por encima
 nan_fill_color='white',
 nan_fill_opacity=0.5,
).add_to(m) # Agregar este gráfico al mapa anterior

folium.LayerControl().add_to(m) # Agregar controles al mapa para moverlo
m
```



```
m.save("map.html")
```

Enviar correos electrónicos desde aplicaciones de Python es una tarea común y necesaria en muchos proyectos, desde notificaciones automáticas hasta boletines informativos. Python facilita esta tarea mediante la biblioteca estándar `smtplib`, que proporciona una interfaz simple y efectiva para enviar correos electrónicos utilizando el protocolo SMTP (Simple Mail Transfer Protocol).

```
Importemos las librerías
from email.message import EmailMessage
import smtplib

definamos la información que necesitamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
```

```
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "Correo de prueba de automatización"
email.set_content(mensaje)
```

Para poder autenticarte en el servidor de google necesitas la clave de aplicaciones para poder generar una clave, esta es diferente a la contraseña del correo

### [Crear Clave de aplicación](#)

```
smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacion
smtp.sendmail(remitente, destinatario, email.as_string()) # Enviamos el correo
smtp.quit() # Cerramos la conexión
```

```
(221,
b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b1094bsm1286049173.43 - gsmtp')
```

```
Este sería el código para enviar desde un correo de outlook.
smtp = smtplib.SMTP("smtp-mail.outlook.com", port=587)
smtp.starttls()
smtp.login(remitente, "clave_de_outlook_123") # Autenticación con la contraseña del correo
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

## Envío de adjuntos.

```
Importemos las librerias
from email.message import EmailMessage
import smtplib

definamos la información que necesitamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "¡Enviado desde Python!"

email.set_content(mensaje)

smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacion
```

```
(235, b'2.7.0 Accepted')
```

```
Adjuntar un archivo .html que creamos del mapa

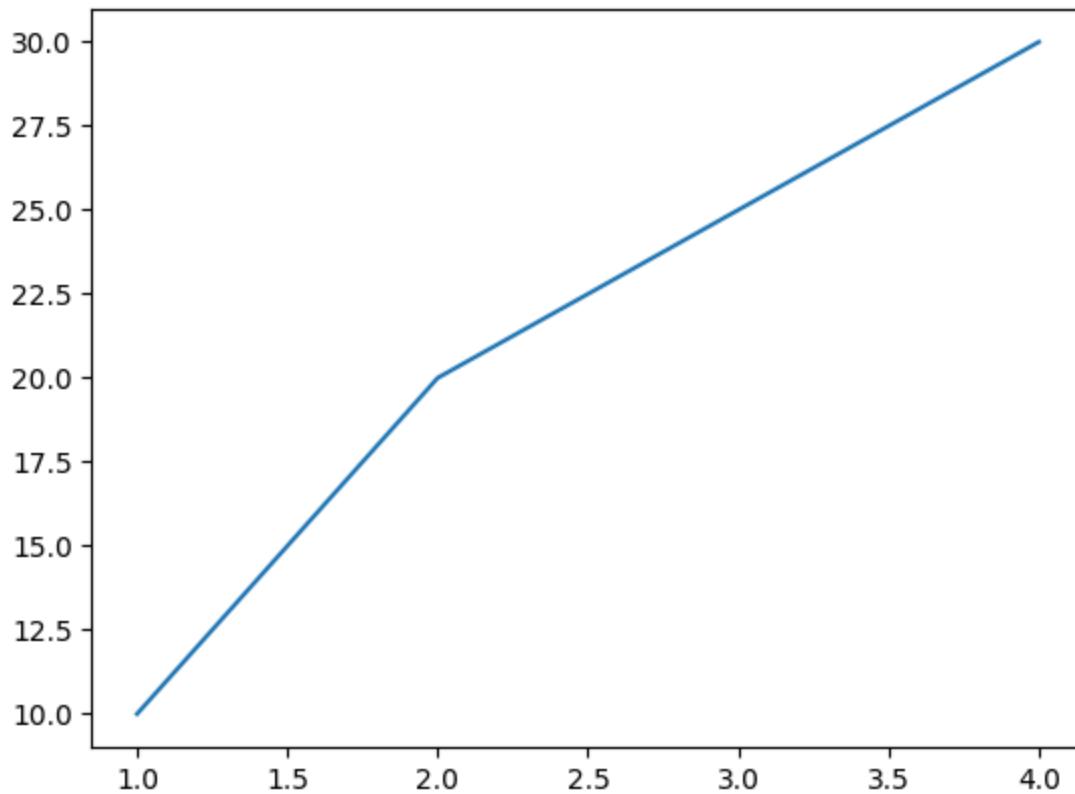
with open("map.html", "rb") as f:
 email.add_attachment(
 f.read(),
 filename="map.html",
 maintype="text",
 subtype="html"
)

import matplotlib.pyplot as plt
Crear y adjuntar una gráfica como archivo .png
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.title('Gráfica de ejemplo')

plt.savefig("grafica.png") # Tenemos que guardar el archivo

with open("grafica.png", "rb") as f:
 email.add_attachment(
 f.read(),
 filename="grafica.png",
 maintype="image",
 subtype="png"
)
```

### Gráfica de ejemplo



```
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

```
(221,
b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b0d427sm1299772173.64 - gsmtp')
```

## Envío masivo de correos.

```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024/main/emails.csv")
df
```

	Unnamed: 0	first_name	last_name			email	email_verified
0	aaron	Aaron	Davis		aaron6348@gmail.com		
1	acook	Anthony	Cook		cook@gmail.com		
2	adam.saunders	Adam	Saunders		adam@gmail.com		
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com			
4	adrian.blair	Adrian	Blair		adrian9335@gmail.com		
...	...	...	...	...	...	...	
470	wilson	Robert	Wilson		robert@yahoo.com		
471	wking	Wanda	King		wanda.king@holt.com		
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com			
473	young	Jessica	Young		jessica4028@yahoo.com		
474	zachary.neal	Zachary	Neal		zneal@gmail.com		

475 rows × 8 columns



Vamos a enviarle correos a nuestras cuentas verificadas y con un balance mayor a 95 USD

```
df_verified = df[(df["email_verified"] == True) & (df["balance"] > 95)]
df_verified
```

	Unnamed: 0	first_name	last_name		email	email_verified
68	christopher	Christopher	NaN	christopher@gmail.com		True
74	clayton6074	Robert	Clayton	robert@gmail.com		True
84	crane203	Valerie	Crane	valerie7051@hotmail.com		True
139	eugene4448	Eugene	NaN	eugene@gmail.com		True
208	jennifer.wong	Jennifer	Wong	jwong@yahoo.com		True
234	joseph2431	Joseph	Harris	joseph@gmail.com		True
245	karen.snow	Karen	Snow	ksnow@yahoo.com		True
259	king	Billy	King	billy.king@hotmail.com		True
260	king3246	Brittney	King	brittney@yahoo.com		True
296	margaret265	Margaret	NaN	margaret@gmail.com		True
357	paul6364	Paul	Hill	paul2980@hotmail.com		True
361	peter	Peter	Grimes	pgrimes@yahoo.com		True
398	sbennett	Steven	Bennett	sbennett@yahoo.com		True
453	twhite	Timothy	White	white5136@hotmail.com		True

# Veamos que le vamos a mandar a cada uno con la informacion del dataset

```
for index, row in df_verified.iterrows():
 print(f"Hello,{row['first_name']} this is your balance: {row['balance']}. This is
```

```
Hello,Christopher this is your balance: 96.96. This is just a test
Hello,Robert this is your balance: 97.05. This is just a test
Hello,Valerie this is your balance: 98.69. This is just a test
Hello,Eugene this is your balance: 95.38. This is just a test
Hello,Jennifer this is your balance: 95.1. This is just a test
Hello,Joseph this is your balance: 95.47. This is just a test
Hello,Karen this is your balance: 99.38. This is just a test
Hello,Billy this is your balance: 98.8. This is just a test
Hello,Brittney this is your balance: 98.79. This is just a test
Hello,Margaret this is your balance: 96.14. This is just a test
Hello,Paul this is your balance: 98.62. This is just a test
Hello,Peter this is your balance: 96.79. This is just a test
Hello,Steven this is your balance: 97.38. This is just a test
Hello,Timothy this is your balance: 99.9. This is just a test
```

```
Ahora creamos una función que envíe el correo y que como parametro reciba, mensaje)
Importemos las librerías
from email.message import EmailMessage
import smtplib

def send_mail(mensaje,destinatario):
 remitente = "remitente@gmail.com"

 email = EmailMessage()

 email["From"] = remitente
 email["To"] = destinatario
 email["Subject"] = "|Enviado desde Python test!"

 email.set_content(mensaje)

 smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otro
 smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicaci
 smtp.sendmail(remitente, destinatario, email.as_string())
 smtp.quit()
```

```
Ahora enviamos un correo por cada usuario del dataframe
for index, row in df_verified.iterrows():
 mensaje = f"Hello,{row['first_name']} this is your balance: {row['balance']}. This :"
 destino = row['email']
 send_mail(mensaje,destino)
```

Dash es un framework de Python diseñado para crear aplicaciones web interactivas de manera rápida y sencilla. Desarrollado por Plotly, Dash permite a los usuarios construir aplicaciones que integran visualizaciones de datos dinámicas y una interfaz de usuario intuitiva sin necesidad de conocimientos avanzados de desarrollo web.

# Estructura de una Aplicación Dash

Una aplicación típica de Dash consta de tres partes principales:

1. Layout: Define la estructura visual de la aplicación, especificando los componentes y su disposición en la página. El layout se describe utilizando estructuras de datos de Python.
2. Callbacks: Define las interacciones entre los componentes. Los callbacks son funciones que especifican cómo los componentes deben actualizarse en respuesta a eventos, como clics de botones o cambios en un formulario.
3. Servidor: Dash incluye un servidor web integrado basado en Flask, lo que permite ejecutar la aplicación localmente durante el desarrollo y luego desplegarla fácilmente en producción.

```
!pip install dash # Instalemos la libreria de Dash
```

```
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
```

```
Este pedazo de código es para poder correr Dash sobre los notebooks sin errores
Nativamente, dash no funciona en Colab entonces debemos colocar estas dos líneas
from dash import jupyter_dash
jupyter_dash.default_mode="external"
```

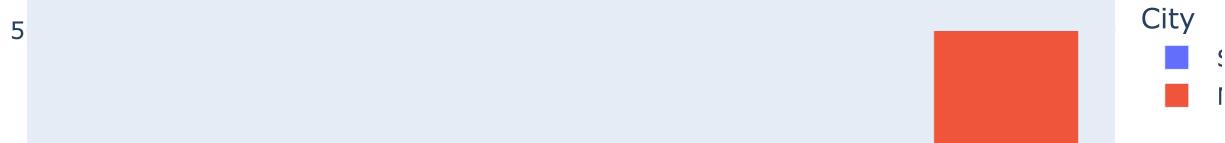
```
from dash import Dash, html, dcc # Estos son los componentes de Dash que vamos a utilizar
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos
import pandas as pd
```

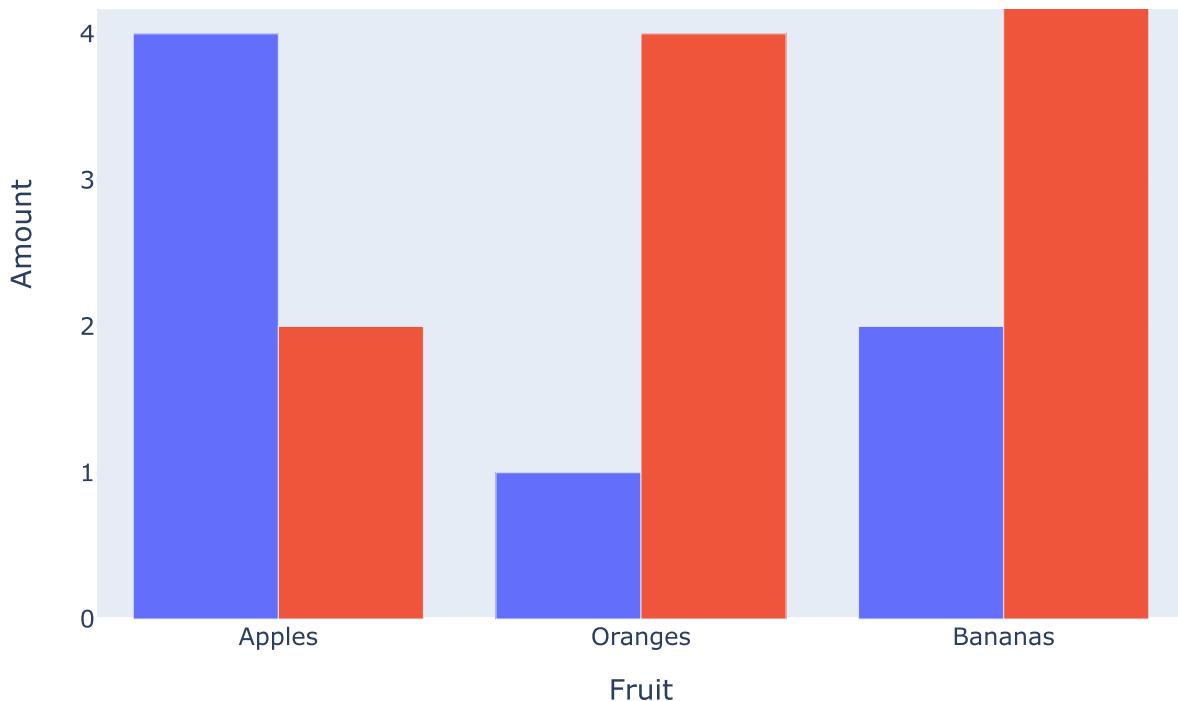
```
app = Dash(__name__) # Creamos un objeto Dash, esta será la aplicación.
```

```
df = pd.DataFrame({
 "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
 "Amount": [4, 1, 2, 2, 4, 5],
 "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})
df
```

	Fruit	Amount	City
0	Apples	4	SF
1	Oranges	1	SF
2	Bananas	2	SF
3	Apples	2	Montreal
4	Oranges	4	Montreal
5	Bananas	5	Montreal

```
Generamos un gráfico dinámico que luego vamos a poner en el dashboard.
fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")
fig
```





```
Definir el layout (distribución) de la aplicación
app.layout = html.Div(children=[
 html.H1(children='Hello Dash'),
 html.Div(children='''
 Dash: A web application framework for your data.
 '''),
 dcc.Graph(
 id='example-graph',
 figure=fig
)
])

Ejecutar la aplicación
if __name__ == '__main__':
 app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

Dash app running on:

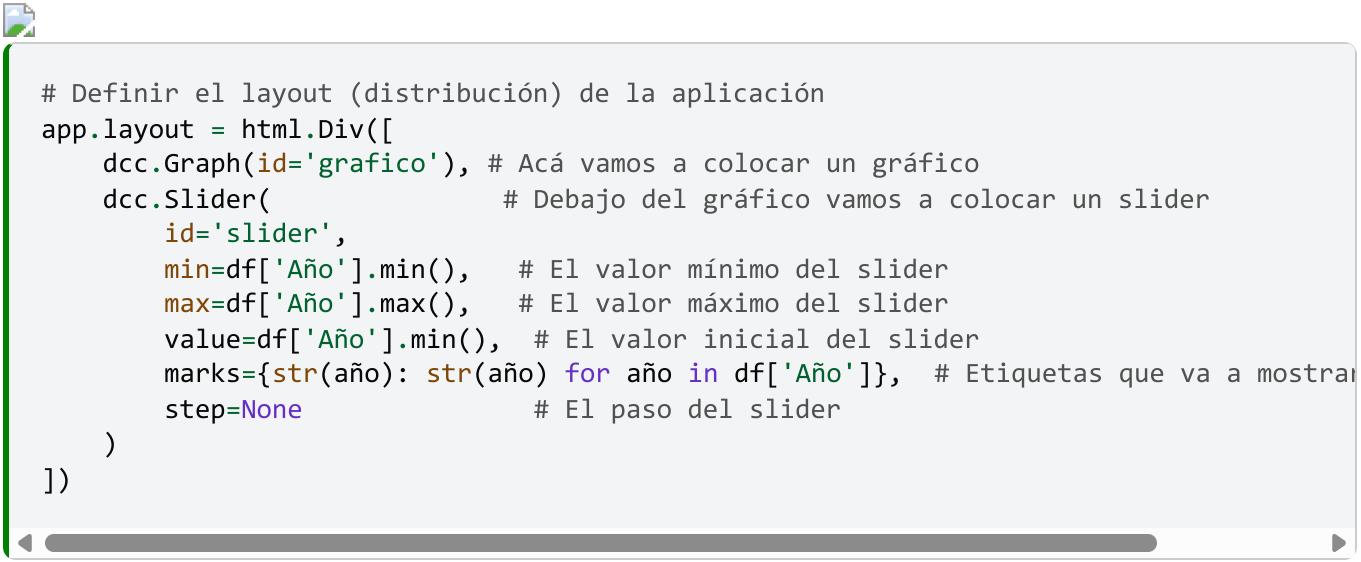
## Ejemplo básico 2

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos

Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
Datos de ejemplo
df = pd.DataFrame({
 "Año": [2010, 2011, 2012, 2013, 2014],
 "Valor": [10, 15, 13, 17, 20]
})

Inicializar la aplicación
app = Dash()
```



```
Definir el layout (distribución) de la aplicación
app.layout = html.Div([
 dcc.Graph(id='grafico'), # Acá vamos a colocar un gráfico
 dcc.Slider(# Debajo del gráfico vamos a colocar un slider
 id='slider',
 min=df['Año'].min(), # El valor mínimo del slider
 max=df['Año'].max(), # El valor máximo del slider
 value=df['Año'].min(), # El valor inicial del slider
 marks={str(año): str(año) for año in df['Año']}, # Etiquetas que va a mostrar el slider
 step=None # El paso del slider
)
])
```

```
Definir el callback para actualizar el gráfico
@app.callback(
 Output('grafico', 'figure'), # Definir en qué objeto del layout colocar la salida
 [Input('slider', 'value')]) # Definir en qué objeto del layout está el elemento de entrada
def actualizar_grafico(año_seleccionado):
 df_filtrado = df[df['Año'] == año_seleccionado]
 fig = px.bar(df_filtrado, x='Año', y='Valor')
 return fig
```

```
Ejecutar la aplicación
if __name__ == '__main__':
 app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

Dash app running on:

## Ejemplo intermedio

```
importamos pandas para leer el archivo csv
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder_u
```

	<b>country</b>	<b>continent</b>	<b>year</b>	<b>lifeExp</b>	<b>pop</b>	<b>gdpPercap</b>
<b>0</b>	Afghanistan	Asia	1952	28.801	8425333	779.445314
<b>1</b>	Afghanistan	Asia	1957	30.332	9240934	820.853030
<b>2</b>	Afghanistan	Asia	1962	31.997	10267083	853.100710
<b>3</b>	Afghanistan	Asia	1967	34.020	11537966	836.197138
<b>4</b>	Afghanistan	Asia	1972	36.088	13079460	739.981106
...	...	...	...	...	...	...
<b>3308</b>	Zimbabwe	Africa	1987	62.351	9216418	706.157306
<b>3309</b>	Zimbabwe	Africa	1992	60.377	10704340	693.420786
<b>3310</b>	Zimbabwe	Africa	1997	46.809	11404948	792.449960
<b>3311</b>	Zimbabwe	Africa	2002	39.989	11926563	672.038623
<b>3312</b>	Zimbabwe	Africa	2007	43.487	12311143	469.709298

3313 rows × 6 columns

```
df.keys()
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos

Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
Creamos un objeto Dash, esta será la aplicación.
app = Dash()
```

```
Vamos a definirle la organización del tablero que vamos a crear
app.layout = html.Div(
 [
 html.H1("Animated GDP and population over decades"),
 html.P("Select an animation:"),
dcc.RadioItems(
 id="selection",
 options=["GDP - Scatter", "Population - Bar"],
 value="GDP - Scatter",
,
 html.Iframe(# Agreguemos el mapa que hicimos al principio
 srcDoc=open("map.html", "r").read(),
 width="100%",
 height="600px",
,
 dcc.Loading(dcc.Graph(id="graph"), type="cube"),
]
)
```

```

El callback que va a actualizar la figura cuando se modifique el elemento de input.
@app.callback(
 Output("graph", "figure"), Input("selection", "value") # Definir los objetos de sa
)
def display_animated_graph(selection):
 animations = { # Definimos un diccionario que contiene para cada opción un tipo de
 "GDP - Scatter": px.scatter(# El primero es un scatter entonces le asignamos
 df, # Establecemos el nombre del dataframe
 x="gdpPercap", # Establecemos el eje x
 y="lifeExp", # Establecemos el eje y
 animation_frame="year", # Establecemos el eje de animación
 animation_group="country", # Establecemos el grupo de animación
 size="pop", # Establecemos el tamaño del punto proporciona
 color="continent", # Establecemos el color del punto proporcional
 hover_name="country", # Establecemos el nombre que va a mostrar cada
 log_x=True, # Establecemos el eje x en escala logarítmica
 size_max=55, # Establecemos el tamaño máximo del punto
 range_x=[100, 100000], # Establecemos el rango del eje x
 range_y=[25, 90], # Establecemos el rango del eje y
),
 "Population - Bar": px.bar(# El segundo es un bar entonces le asignamos a la
 df, # Establecemos el nombre del dataframe
 x="continent", # Establecemos el eje x
 y="pop", # Establecemos el eje y
 color="continent", # Establecemos el color del punto proporcional
 animation_frame="year", # Establecemos el eje de animación
 animation_group="country", # Establecemos el grupo de animación
 range_y=[0, 4000000000], # Establecemos el rango del eje y
),
 }
 return animations[selection] # Retornamos el gráfico seleccionado que es el gráfi

Run the app
if __name__ == '__main__':
 app.run(debug=True)

```

Dash app running on:

# «Data cleaning» y «Filtros»

## Contenido

- Tratamiento de valores Faltantes
- Ejercicio
- Operaciones con dataframes
- Retos

Para responder a preguntas a partir de los datos, pandas es una librería muy útil. Sin embargo, los datos no siempre vienen limpios y listos para ser analizados. En este notebook, veremos cómo limpiar datos y cómo filtrarlos para responder a preguntas específicas.

```
Vamos a cargar un dataset de ejemplo.

import pandas as pd

df = pd.read_csv('data/titanic.csv')
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A 211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticid
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0

891 rows × 12 columns

Revisa algunos de los métodos anteriores para revisar la data y explorar el tamaño de la información y las columnas.

# TU CÓDIGO AQUÍ

## Tratamiento de valores Faltantes

Un problema común en los datos es la presencia de valores faltantes. Estos valores pueden ser NaN, None, NaT, entre otros. Para tratar estos valores, podemos usar el método `fillna()` para reemplazar los valores faltantes por un valor específico. También podemos usar el método `dropna()` para eliminar las filas que contienen valores faltantes.

Con el método `isnull()`, podemos identificar los valores faltantes en un DataFrame. Este método devuelve un DataFrame booleano con True en las posiciones donde hay valores faltantes y False en las posiciones donde no hay valores faltantes.

Si usamos `fillna()` se reemplazarán los valores faltantes por el valor que le pasemos como argumento. Si usamos `dropna()` se eliminarán las filas que contienen valores faltantes.

```
df.fillna(' ')
df.head(10) # Note que al parecer el cambio no se ha realizado. ¿Por qué? ¿Qué podemos
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df_cleaned = df.fillna(' ')
df_cleaned.head(10) # Ahora sí se ha realizado el cambio.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	-	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	<b>PassengerId</b>	<b>Survived</b>	<b>Pclass</b>	<b>Name</b>	<b>Sex</b>	<b>Age</b>	<b>SibSp</b>	<b>Parch</b>	<b>Ticket</b>
<b>9</b>		10	1	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df.fillna(' ', inplace=True) # Esta es otra forma de tratar los valores nulos,
solo que en esta le especificamos que queremos que sea un espacio.
```

```
Otras formas de completar los valores nulos son:
- fillna(0) para completar con ceros.
- fillna('Desconocido') para completar con un string.
- fillna(df['columna'].mean()) para completar con el promedio de la columna.
- fillna(df['columna'].median()) para completar con la mediana de la columna.
- fillna(df['columna'].mode()[0]) para completar con la moda de la columna.
- fillna(method='ffill') para completar con el valor anterior.
- fillna(method='bfill') para completar con el valor siguiente.
```

## Ejercicio

Realiza la limpieza de solo una columna del dataframe y que los valores nulos se rellenen por el promedio de la columna.

```
TU CÓDIGO AQUÍ
```

## Operaciones con dataframes

Para realizar operaciones y filtrar información del dataframe o calcular nuevas columnas a partir de los datos ya existentes podemos utilizar funcionalidades especializadas de pandas. Vamos a revisar algunas cosas que podemos hacer con los dataframes.

## Operaciones de indexación booleana

```
Indexación booleana
La indexación booleana es una técnica que nos permite seleccionar un subconjunto de
que cumpla con cierta condición.

Por ejemplo, si queremos seleccionar los pasajeros que son mujeres, podemos hacer lo
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0		1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1		2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2		3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
1. revisemos los posibles valores que puede tomar la columna
df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
2. Como solo quiero seleccionar los que son mujeres hago lo siguiente:
result = df['Sex'] == 'female'

result # así, en result tenemos una serie de booleanos que nos indican si el pasajero
Pero nosotros queremos es tener los registros de las mujeres, para eso hacemos lo si
```

```
0 False
1 True
2 True
3 True
4 False
...
886 False
887 True
888 True
889 False
890 False
Name: Sex, Length: 891, dtype: bool
```

```
df[result]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17543
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O <sup>n</sup> 310128
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	11383
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347740
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	23773
...	...	...	...	...	...	...	...	...	...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	23042
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

PassengerId		Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

```
O lo que es lo mismo:
df[df['Sex'] == 'female']
Esta sintaxis se puede leer como: del dataframe df, vamos a seleccionar las filas de
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17543
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O <sup>n</sup> 310128
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	11383
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347740
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	23773
...	...	...	...	...	...	...	...	...	...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	23042
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

## Operaciones de indexación con operadores lógicos

```
Pensemos ahora que queremos saber cuales fueron los pasajeros de la clase mayor a 1
df[df['Pclass'] > 1]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2113128
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O <sup>n</sup> 310.101802
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734540802
5	6	0	3	Moran, Mr. James	male	NaN	0	0	3303550802
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	34934540802
...	...	...	...	...	...	...	...	...	...
884	885	0	3	Suthehall, Mr. Henry Jr	male	25.0	0	0	SOTON 3934540802
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3834540802
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2134540802
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 134540802
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3734540802

675 rows × 12 columns

```
Ahora extraigamos los pasajeros que son mujeres, sobrevivieron y mayores de 20 años.
condicion = (df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20) # Note que pa
Esta not
df[condicion]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	...
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monopeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

```
modo PRO
df[(df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...	...	...	...	...	...	...	...	...	...
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monopeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

## Operaciones entre columnas

Si quiero crear una columna nueva puedo simplemente decir `df['nueva_columna'] = df['columna1'] + df['columna2']` y se creará una nueva columna con la suma de los valores de las columnas 1 y 2.

Veamos como quedaría el código para realizar estas operaciones. en las que sumemos 10 a la columna Fare y luego ese resultado lo vamos a dividir entre la edad para cada uno de los pasajeros.

```
df['mi_nueva_columna'] = 10 + df['Fare'] # Podemos crear nuevas columnas a partir de :
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
df['mi_otra_columna'] = df['mi_nueva_columna'] / df['Age'] # Podemos crear nuevas columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

## Ordenar los valores de un resultado

Si queremos ordenar los valores de un resultado, podemos usar el método `sort_values()`.

```
Ordenemos los valores de una columna de forma ascendente, luego de sacar los registros
condicion = (df['Age'] > 20)
df[condicion].sort_values('Age', ascending=True)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
<b>227</b>	228	0	3	male	20.5	0	0	A/5 21173	7.2500
<b>37</b>	38	0	3	male	21.0	0	0	A./5. 2152	8.0500
<b>56</b>	57	1	2	female	21.0	0	0	C.A. 31026	10.5000
<b>436</b>	437	0	3	female	21.0	2	2	W./C. 6608	34.3750
<b>106</b>	107	1	3	female	21.0	0	0	343120	7.6500
...	...	...	...	...	...	...	...	...	...
<b>116</b>	117	0	3	male	70.5	0	0	370369	7.7500
<b>96</b>	97	0	1	male	71.0	0	0	PC 17754	34.6542
<b>493</b>	494	0	1	male	71.0	0	0	PC 17609	49.5042
<b>851</b>	852	0	3	male	74.0	0	0	347060	7.7750
<b>630</b>	631	1	1	male	80.0	0	0	27042	30.0000

535 rows × 13 columns

## Operaciones sobre columnas

Si quieres sacar medidas estadísticas de una columna puedes usar los métodos `mean()`, `median()`, `std()`, `min()`, `max()`, `count()`, `sum()`, entre otros.

Aquellas columnas que tienen valores NaN no se incluyen en el cálculo de estas medidas. Si quieres incluir los valores NaN en el cálculo de estas medidas, puedes usar el argumento `skipna=False`.

```
df['Age'].sum() # Podemos hacer operaciones sobre las columnas.
```

```
np.float64(21205.17)
```

```
df.Age.mean() # También podemos hacerlo de esta forma, usando el nombre de la columna
pero si el nombre de la columna tiene espacios o caracteres especiales
```

```
np.float64(29.69911764705882)
```

## Transformar textos de un dataframe

Ahora miremos otro ejemplo, que pasa si quiero separar de nombre el apellido y el nombre en dos columnas diferentes. Para esto podemos usar el método `str.split()` que nos permite separar un texto en base a un separador.

```
df['Apellido'] = df['Name'].str.split(',').str[0] # Podemos crear nuevas columnas a partir de una sola
df['Nombre'] = df['Name'].str.split(',').str[1] # Podemos crear nuevas columnas a partir de una sola
df.head(3)
```

	PassengerId	Survived	Pclass	Name		Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris		male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...		female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina		female	26.0	0	0	STON/O2. 3101282

## Eliminar columnas de un dataframe

```
Eliminar las columnas específicas de un DF
df.drop(columns=['mi_nueva_columna'], inplace=True) # Podemos eliminar columnas que no
df.drop(columns=['Name','mi_otra_columna'], inplace=True) # Podemos eliminar columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

# Cambiar el nombre de las columnas

Ahora vamos a cambiar los nombres de las columnas al español. Para esto podemos usar el método `rename()` que nos permite cambiar el nombre de las columnas de un dataframe.

```
df.rename(columns={'Apellido':'Apellido_Pasajero', 'Nombre':'Nombre_Pasajero'}, inplace=True)
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

## Ver solo algunas columnas

Veamos como se puede seleccionar solo algunas columnas de un dataframe.

```
pedazo = df[['Nombre_Pasajero', 'Apellido_Pasajero', 'Survived', 'Pclass']]
pedazo.head(3)
```

	Nombre_Pasajero	Apellido_Pasajero	Survived	Pclass
0	Mr. Owen Harris	Braund	0	3
1	Mrs. John Bradley (Florence Briggs Thayer)	Cumings	1	1
2	Miss. Laina	Heikkinen	1	3

# Funciones de agregación

Para realizar operaciones de agregación en un dataframe, podemos usar el método `agg()`. Este método nos permite aplicar una función de agregación a una o más columnas de un dataframe.

```
Funciones de agregación
Las funciones de agregación nos permiten realizar operaciones sobre un conjunto de columnas.

df['Age'].mean() # Por ejemplo, podemos calcular el promedio de la edad de los pasajeros.

df['Age'].max() # O la edad máxima.

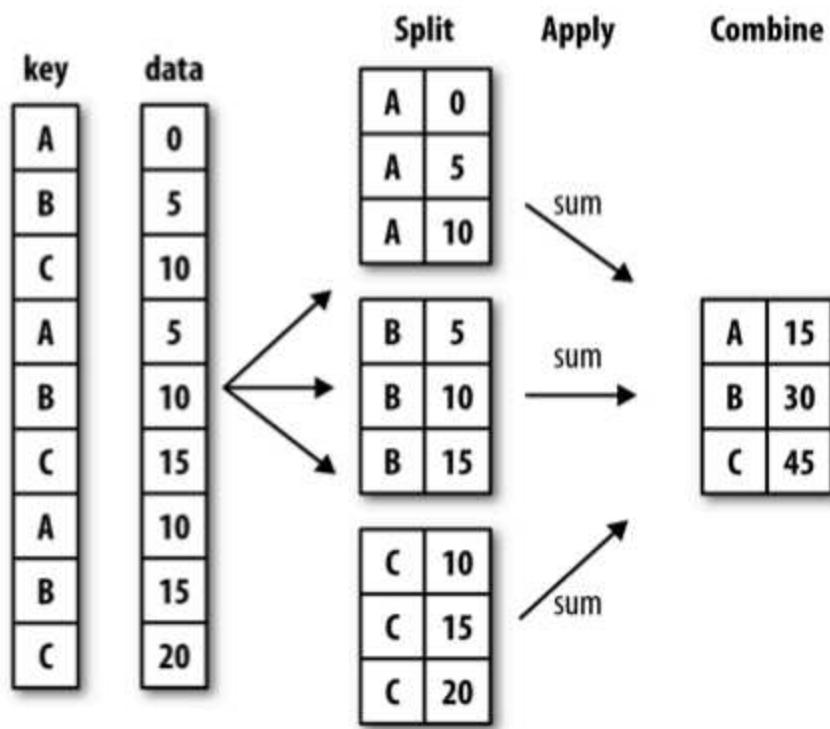
otra sintaxis es usando el método agg

df['Age'].agg('mean') # Por ejemplo, podemos calcular el promedio de la edad de los pasajeros.
```

```
np.float64(29.69911764705882)
```

# Funciones de agrupación

Para realizar operaciones de agrupación en un DataFrame, podemos usar el método `groupby()`. Este método divide el DataFrame en grupos según los valores de una o más columnas. Luego, podemos aplicar funciones de agregación a estos grupos.



```
Vamos a agrupar nuestros datos por sobrevivientes o no y calcular el promedio de la edad
Tradicionalmente lo haríamos así:

mean_1 = df[df['Survived']==1]['Age'].mean() # Promedio de la edad de los sobrevivientes
mean_0 = df[df['Survived']==0]['Age'].mean() # Promedio de la edad de los no sobrevivientes

print(mean_1)
print(mean_0)

Entonces tendríamos que conocer cada una de las etiquetas para hacer los filtros, p
```

28.343689655172415  
30.62617924528302

```
df.groupby('Survived')['Age'].sum() # Esto nos retorna una serie en la que saca el valor
Esto se puede leer como: Agrupe los registros del dataframe df por la columna Survived
```

Survived  
0 12985.50  
1 8219.67  
Name: Age, dtype: float64

```
df.groupby('Survived')['Age'].agg('sum') # Esto es igual a la anterior, solo que usamos agg()
df.groupby('Survived')['Age'].aggregate('sum') # Esto es igual a la anterior, solo que usamos aggregate()
```

```
Survived
0 12985.50
1 8219.67
Name: Age, dtype: float64
```

## Unir varios dataframes usando merge

Esta es una opción muy útil cuando tenemos varios dataframes y queremos unirlos en uno solo. Para esto podemos usar el método `merge()` que nos permite unir dos dataframes en base a una o más columnas en común.

```
import pandas as pd

transactions = pd.read_csv('data/transactions.csv', index_col=0)
requests = pd.read_csv('data/requests.csv', index_col=0)
```

```
Revisemos los dataframes obtenidos
transactions.head(2)
```

	<b>sender</b>	<b>receiver</b>	<b>amount</b>	<b>sent_date</b>
<b>0</b>	stein	smoyer	49.03	2018-01-24
<b>1</b>	holden4580	joshua.henry	34.64	2018-02-06

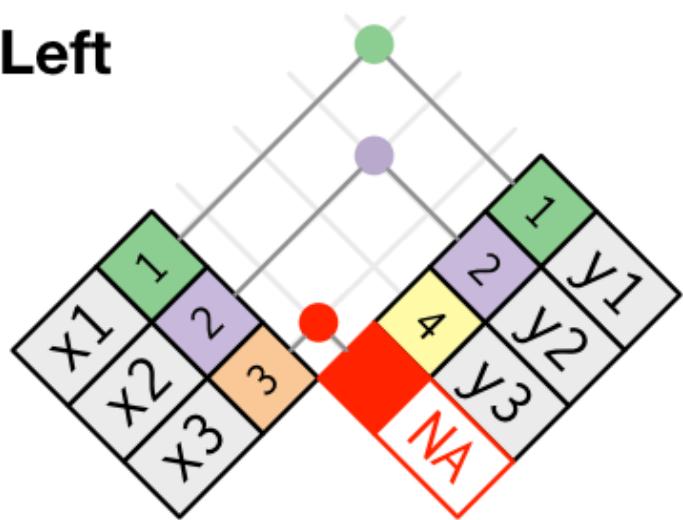
```
Revisemos los dataframes obtenidos
requests.head(2)
```

	<b>from_user</b>	<b>to_user</b>	<b>amount</b>	<b>request_date</b>
<b>0</b>	chad.chen	paula7980	78.61	2018-02-12
<b>1</b>	kallen	lmoore	1.94	2018-02-23

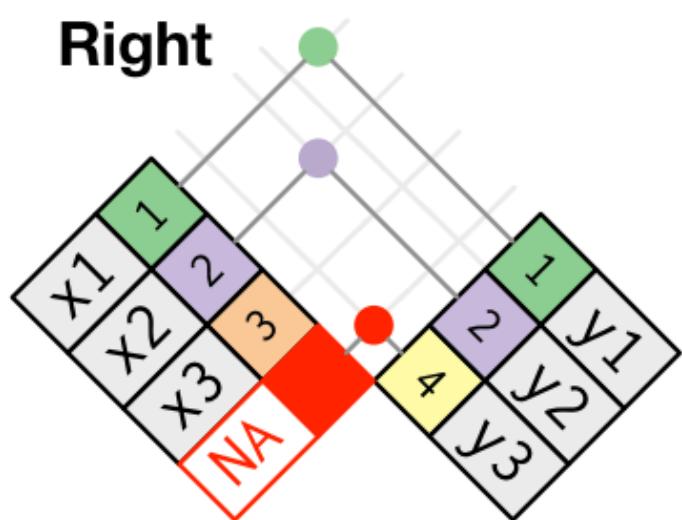
Me gustaría ver todas las solicitudes que tienen una transacción coincidente basada en los usuarios y el monto involucrado.

Para hacer esto, combinaremos ambos conjuntos de datos.

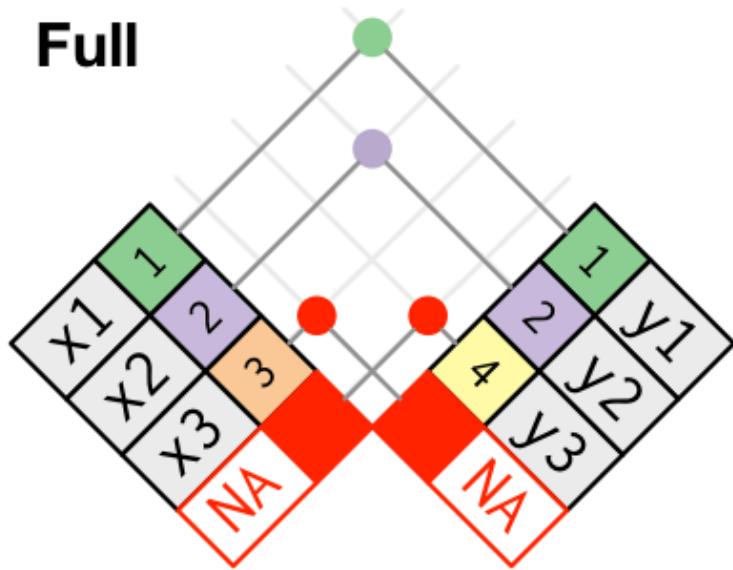
Crearemos un nuevo conjunto de datos utilizando el método DataFrame.merge.

**Left**

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

**Right**

key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

**Full**

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

```
Como estamos llamando a merge en el DataFrame `requests`, se considera el lado izquierdo
successful_requests = requests.merge(
 # Y `transactions` es el lado derecho
 transactions,
 # Así que ahora alineamos las columnas que harán que la unión tenga sentido. esto
 left_on=['from_user', 'to_user', 'amount'],
 right_on=['receiver', 'sender', 'amount']
)
Veamos cómo nos fue
successful_requests.head()
```

	from_user	to_user	amount	request_date	sender	receiver
0	chad.chen	paula7980	78.61	2018-02-12	paula7980	chad.chen
1	kallen	lmoore	1.94	2018-02-23	lmoore	kallen
2	gregory.blackwell	rodriguez5768	30.57	2018-03-04	rodriguez5768	gregory.blackwell
3	kristina.miller	john.hardy	77.05	2018-03-12	john.hardy	kristina.miller
4	lacey8987	mcguire	54.09	2018-03-13	mcguire	lacey8987

## Retos

### Reto 1

Vamos a usar la data de sobrevivientes del Titanic. Vamos responder a los siguientes enunciados:

1. ¿Cuántos pasajeros sobrevivieron y cuántos no sobrevivieron?
2. ¿Cuántos pasajeros de cada clase sobrevivieron?
3. ¿Cuántos hombres y mujeres sobrevivieron?
4. Usando groupby(), calcula la edad promedio para cada sexo.
5. Calcula tasa promedio de supervivencia para todos los pasajeros.

6. Calcula este ratio de supervivencia para todos los pasajeros menores de 25 años (recuerda: filtrado/indexación booleana)

```
Tu código aquí
```

# Ejecución de código en múltiples lenguajes de programación

## Contenido

- Ejecutar código en C
- Ejecutar código en C++
- Ejecutar código en R

## Ejecutar código en C

### El Hola Mundo

```
Versión de GCC
!gcc --version
```

```
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
%%writefile welcome.c
#include <stdio.h> /* incluye biblioteca donde se define E/S */
int main(){
 printf("Hola mundo en C");
 return 0;
}
```

Writing welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Hola mundo en C

## Condicionales en C

```
Uso de if, else, else if
%%writefile welcome.c

#include <stdio.h>
int main(){
 int myNum = 10; // Is this a positive or negative number?

 if (myNum > 0) {
 printf("The value is a positive number.");
 } else if (myNum < 0) {
 printf("The value is a negative number.");
 } else {
 printf("The value is 0.");
 }
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

The value is a positive number.

```
Uso de switch
%%writefile welcome.c
#include <stdio.h>
int main(){
 int day = 4;

 switch (day) {
 case 1:
 printf("Monday");
 break;
 case 2:
 printf("Tuesday");
 break;
 case 3:
 printf("Wednesday");
 break;
 case 4:
 printf("Thursday");
 break;
 case 5:
 printf("Friday");
 break;
 case 6:
 printf("Saturday");
 break;
 case 7:
 printf("Sunday");
 break;
 }
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Thursday

## Ciclos en C

```
Uso de While
%%writefile welcome.c
#include <stdio.h>
int main(){
 int i = 0;

 while (i < 5) {
 printf("%d\n", i);
 i++;
 }
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
./welcome
```

```
0
1
2
3
4
```

```
Uso de do while
%%writefile welcome.c
#include <stdio.h>
int main(){

 int i = 0;

 do {
 printf("%d\n", i);
 i++;
 }
 while (i < 5);

 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
4
```

```
Uso de for
%%writefile welcome.c
#include <stdio.h>
int main(){

 int i;

 for (i = 0; i < 5; i++) {
 printf("%d\n", i);
 }

 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
4
```

```
Uso de for anidado
%%writefile welcome.c
#include <stdio.h>
int main(){
 int i, j;

 // Outer loop
 for (i = 1; i <= 2; ++i) {
 printf("Outer: %d\n", i); // Executes 2 times

 // Inner loop
 for (j = 1; j <= 3; ++j) {
 printf(" Inner: %d\n", j); // Executes 6 times (2 * 3)
 }
 }

 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Outer: 1
 Inner: 1
 Inner: 2
 Inner: 3
Outer: 2
 Inner: 1
 Inner: 2
 Inner: 3
```

```
Uso de break en un while
%%writefile welcome.c
#include <stdio.h>
int main(){
 int i = 0;

 while (i < 10) {
 if (i == 4) {
 break;
 }
 printf("%d\n", i);
 i++;
 }

 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0
1
2
3
```

## Funciones en C

```
Crear una función y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction() {
 printf("I just got executed!\n");
}

int main(){
 myFunction(); // Llamar a la función
 myFunction(); // Llamar a la función por segunda vez
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

I just got executed!  
I just got executed!

```
Crear una función con entrada de Múltiples parámetros y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction(char name[], int age) {
 printf("Hello %s. You are %d years old.\n", name, age);
}

int main() {
 myFunction("Liam", 3);
 myFunction("Jenny", 14);
 myFunction("Anja", 30);
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Hello Liam. You are 3 years old.
Hello Jenny. You are 14 years old.
Hello Anja. You are 30 years old.
```

```
Crear una función que Retorna un resultado y utilizarla
%%writefile welcome.c
#include <stdio.h>

int myFunction(int x) {
 return 5 + x;
}

int main() {
 printf("Result is: %d", myFunction(3));
 return 0;
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Result is: 8
```

## Matrices en C

```
Crear un Array 1D (Vector) y recorrerlo usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
 int myNumbers[] = {25, 50, 75, 100};
 int i;

 for (i = 0; i < 4; i++) {
 printf("%d\n", myNumbers[i]);
 }
 return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
25
50
75
100
```

```
Crear un Array 2D (Matriz) y recorrerla usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
 int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

 int i, j;
 for (i = 0; i < 2; i++) {
 for (j = 0; j < 3; j++) {
 printf("%d\n", matrix[i][j]);
 }
 }
 return 0;
}
```

Overwriting welcome.c

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
1
4
2
3
6
8
```

## Ejecutar código en C++

```
%%writefile welcome.cpp
#include <iostream>
int main()
{
 std::cout << "Hola mundo en C++";
 return 0;
}
```

Writing welcome.cpp

```
%%bash
g++ welcome.cpp -o welcome
```

```
%%bash
./welcome
```

```
Hola mundo en C++
```

## Ejecutar código en R

```
Instalar versión compatible
!pip install rpy2==3.5.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu
Collecting rpy2==3.5.1
 Downloading rpy2-3.5.1.tar.gz (201 kB)
 201.7/201.7 KB 15.2 MB/s eta 0:00:00
?25h Preparing metadata (setup.py) ... ?25l?25hdone
Requirement already satisfied: cffi>=1.10.0 in /usr/local/lib/python3.8/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)
Requirement already satisfied: tzlocal in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)
Building wheels for collected packages: rpy2
 Building wheel for rpy2 (setup.py) ... ?25l?25hdone
 Created wheel for rpy2: filename=rpy2-3.5.1-cp38-cp38-linux_x86_64.whl size=318371 s
 Stored in directory: /root/.cache/pip/wheels/6b/40/7d/f63e87fd83e8b99ee837c8e3489081
Successfully built rpy2
Installing collected packages: rpy2
 Attempting uninstall: rpy2
 Found existing installation: rpy2 3.5.5
 Uninstalling rpy2-3.5.5:
 Successfully uninstalled rpy2-3.5.5
Successfully installed rpy2-3.5.1
```

```
activate R magic
%load_ext rpy2.ipython
```

```
!R --version
```

```
R version 4.2.2 Patched (2022-11-10 r83330) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
https://www.gnu.org/licenses/.
```

```
%%R
i <- 1
while (i < 6) {
 print(i)
 i <- i + 1
}
```

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

# Introducción

## Contenido

- **Introducción**
- Ciencia de datos
- Aprendizaje automático
- Más recursos
- Como saber la versión de Python
- Como saber los recursos de GPU
- Como saber la versión de Linux
- Tutorial Python
- Conexión con Google drive
- Asignación automática de tipos de variables
- Gestión de la memoria
- Casteo de variables
- Operaciones Aritméticas
- Funciones útiles
- Numeros Aleatorios
- Libreria Math
- Operaciones con cadena de texto
- Tuplas (Datos inmutables)
- Listas (Datos no inmutables)
- Diccionarios
- Entrada y salida de datos
- Operadores relacionales
- Operadores lógicos
- Estructuras condicionadas
- Estructuras repetitivas

- Funciones
- Paso por valor y referencia
- Manejo de excepciones y errores.

Si ya conoces Colab, mira este video para aprender sobre las tablas interactivas, la vista histórica de código ejecutado y la paleta de comandos.



## ¿Qué es Colab?

Colab, o «Colaboratory», te permite escribir y ejecutar código de Python en tu navegador, con

- Una configuración lista para que empieces a programar
- Acceso gratuito a GPU
- Facilidad para compartir

Seas **estudiante, científico de datos o investigador de IA**, Colab facilita tu trabajo. Mira [este video introductorio sobre Colab](#) para obtener más información, o bien comienza a usarlo más abajo.

El documento que estás leyendo no es una página web estática, sino un entorno interactivo denominado **notebook de Colab**, que permite escribir y ejecutar código.

Por ejemplo, esta es una **celda de código** con una secuencia de comandos Python corta que calcula un valor, lo almacena en una variable y devuelve el resultado:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

```
type(seconds_in_a_day)
```

```
int
```

A fin de ejecutar el código en la celda anterior, haz clic en él para seleccionarlo y luego presiona el botón de reproducción ubicado a la izquierda del código o usa la combinación de teclas «Command/Ctrl + Intro». Para editar el código, solo haz clic en la celda y comienza a editar.

Las variables que defines en una celda pueden usarse en otras:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Los notebooks de Colab te permiten combinar **código ejecutable** y **texto enriquecido** en un único documento, junto con **imágenes**, **HTML**, **LaTeX** y mucho más. Los notebooks que crees en Colab se almacenan en tu cuenta de Google Drive. Puedes compartir fácilmente los notebooks de Colab con amigos o compañeros de trabajo para que realicen comentarios o los editen. Si quieres obtener más información, consulta la [Descripción general de Colab](#). Para crear un nuevo notebook de Colab, ve al menú Archivo que aparece más arriba o usa este vínculo: [crear un nuevo notebook de Colab](#).

Los notebooks de Colab son notebooks de Jupyter que aloja Colab. Para obtener más información sobre el proyecto Jupyter, visita [jupyter.org](#).

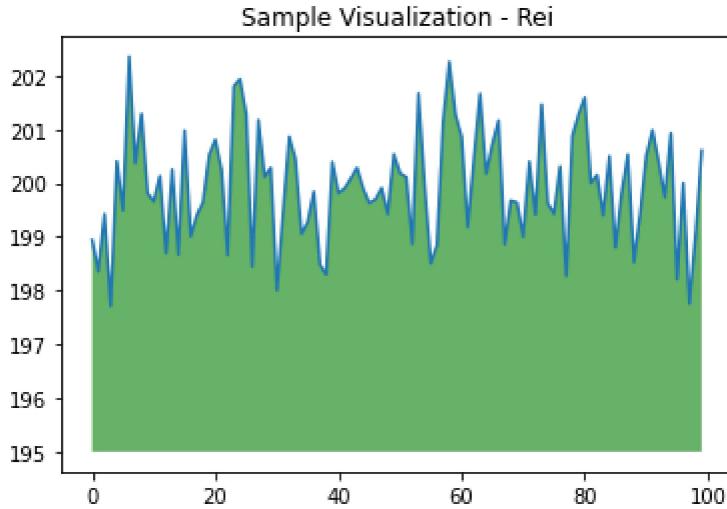
Con Colab, puedes aprovechar por completo las bibliotecas más populares de Python para analizar y visualizar datos. La celda de código que se incluye a continuación usa **NumPy** para generar algunos datos aleatorios y **matplotlib** para visualizarlos. Para editar el código, haz clic en la celda y comienza a editar.

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

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization - Rei")
plt.show()
```



Puedes importar datos propios a notebooks de Colab desde tu cuenta de Google Drive (incluso desde hojas de cálculos), GitHub y muchas otras fuentes. Para obtener más información acerca de la importación de datos y cómo puede usarse Colab para fines relacionados con la ciencia de datos, consulta los vínculos de [Cómo trabajar con datos](#).

Colab te permite importar un conjunto de datos de imágenes, entrenar un clasificador de imágenes en él y evaluar el modelo con solo [unas pocas líneas de código](#). Los notebooks de Colab ejecutan código en los servidores alojados en la nube de Google, lo que significa que puedes aprovechar al máximo el hardware de Google, incluidas las [GPU y TPU](#), independientemente de la potencia de tu máquina. Lo único que necesitas es un navegador.

Entre los usos que se la da a Colab en la comunidad de aprendizaje automático, se encuentran los siguientes:

- Introducción a TensorFlow
- Desarrollo y entrenamiento de redes neuronales
- Experimentación con TPU
- Diseminación de investigación de IA

- Creación de instructivos

Para ver notebooks de Colab de ejemplo que muestran los usos del aprendizaje automático, consulta los [ejemplos](#) que se incluyen a continuación.

## Cómo trabajar con notebooks en Colab

- [Descripción general de Colaboratory](#)
- [Guía para usar Markdown](#)
- [Cómo importar bibliotecas y luego instalar dependencias](#)
- [Cómo guardar y cargar notebooks en GitHub](#)
- [Formularios interactivos](#)
- [Widgets interactivos](#)

## Cómo trabajar con datos

- [Cómo cargar datos: Drive, Hojas de cálculo y Google Cloud Storage](#)
- [Gráficos: visualización de datos](#)
- [Cómo comenzar a usar BigQuery](#)

## Curso intensivo de aprendizaje automático

Estos son algunos de los notebooks del curso de aprendizaje automático en línea de Google.

Para obtener más información, consulta el [sitio web del curso completo](#).

- [Introducción a Pandas DataFrame](#)
- [Regresión lineal con tf.keras usando datos sintéticos](#)

## Uso de aceleración de hardware

- [TensorFlow con GPU](#)
- [TensorFlow con TPU](#)

# Ejemplos destacados

- [NeMo Voice Swap](#): Utiliza el kit de herramientas de NeMo para IA conversacional de Nvidia si quieres cambiar una voz en un fragmento de audio por otra generada por computadora.
- [Reentrenamiento de un clasificador de imágenes](#): compila un modelo de Keras sobre un clasificador de imágenes previamente entrenado para distinguir flores.
- [Clasificación de texto](#): clasifica opiniones sobre películas de IMDB como *positivas* o *negativas*.
- [Transferencia de estilos](#): usa el aprendizaje profundo para transferir el estilo de una imagen a otra.
- [Codificador universal de oraciones en varios idiomas para preguntas y respuestas](#): usa un modelo de aprendizaje automático para responder preguntas del conjunto de datos SQuAD.
- [Interpolación de videos](#): predice lo que sucedió en un video entre el primer y el último fotograma.

```
#Versión de python
!python --version
```

Python 3.8.10

```
Recursos de GPU
!nvidia-smi
!/usr/local/cuda/bin/nvcc --version
```

Tue Feb 21 19:33:36 2023

```
+-----+
| NVIDIA-SMI 510.47.03 Driver Version: 510.47.03 CUDA Version: 11.6 |
+-----+
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
+-----+
| 0 Tesla T4 Off | 00000000:00:04.0 Off | 0 |
| N/A 67C P0 28W / 70W | 0MiB / 15360MiB | 0% Default |
| | | N/A |
+-----+
+-----+
| Processes:
| GPU GI CI PID Type Process name GPU Memory
| ID ID
+-----+
| No running processes found
+-----+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_Mar_8_18:18:20_PST_2022
Cuda compilation tools, release 11.6, V11.6.124
Build cuda_11.6.r11.6/compiler.31057947_0
```

```
Versión de Ubuntu
!lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 20.04.5 LTS
Release: 20.04
Codename: focal
```

<https://www.w3schools.com/python/default.asp>

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

En Python no es necesario declarar explícitamente el tipo de las variables. El intérprete trata de inferir el tipo de las variables según se usan. Igualmente, las operaciones tienen semántica distinta para distintos tipos de datos. Fíjate en el ejemplo siguiente.

```
a = 10
b = 3
c = 3.
d = "holaworld"
e = True
f = 1e3
g = [1,2,3] # La lista es mutable
h = (1,2,3) # La tupla es inmutable
i = 5j
j = 1+2j

print ("a=",type(a))
print ("b=",type(b))
print ("c=",type(c))
print ("d=",type(d))
print ("e=",type(e))
print ("f=",type(f))
print ("g=",type(g))
print ("h=",type(h))
print ("i=",type(i))
print ("j=",type(j))
```

```
a= <class 'int'>
b= <class 'int'>
c= <class 'float'>
d= <class 'str'>
e= <class 'bool'>
f= <class 'float'>
g= <class 'list'>
h= <class 'tuple'>
i= <class 'complex'>
j= <class 'complex'>
```

## Tipo de datos BOOLEANO

El tipo de datos *boolean* (booleano) denota el rango de *valores lógicos* que consiste de dos elementos *true* y *false* y está definido de la siguiente manera:

$$\text{type boolean} = (\text{false}, \text{true})$$

Los siguientes *operadores* (operaciones) están definidos sobre argumentos de este tipo.

- AND: conjunción lógica,

$$\wedge : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- OR: disyunción lógica, inclusiva,

$$\vee : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- NOT: negación lógica,

$$\neg : \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

True and False , True or False, not True

(False, True, False)

## Operadores relacionales

Todos los operadores relacionales producen un resultado de tipo *Booleano*. Los seis operadores relacionales básicos son,

$$=, \neq, <, \leq, \geq, >$$

Por ejemplo, de acuerdo a los valores que sean asignados a las variables  $x$  y  $y$ , el valor de la siguiente expresión es *false* o *true* :

$$(x + y \geq 0 \wedge x * y < 0) \vee \neg(x * x \geq 20).$$

Los cuatro operadores relacionales  $<$ ,  $\leq$ ,  $\geq$ ,  $>$  solo pueden ser aplicados a tipos de datos ordenados (escalares).

`2>1, True<False`

(True, False)

## Tipo de datos INTEGER

EL tipo *integer* representa el conjunto de *números enteros*. Los siguientes operadores están definidos para este tipo de datos.

- adición  $+$ : *integer*  $\times$  *integer*  $\rightarrow$  *integer*
- sustracción  $-$  : *integer*  $\times$  *integer*  $\rightarrow$  *integer*
- multiplicación  $*$  : *integer*  $\times$  *integer*  $\rightarrow$  *integer*
- división con truncamiento **div**: *integer*  $\times$  (*integer*  $- \{0\})  $\rightarrow$  *integer*  
La operación *a div b* no está definida para  $b = 0$ .$
- residuo de la división entera **mod** : *integer*  $\times$  *integer*<sup>+</sup>  $\rightarrow$  *integer*  
La operación *a mod b* solo está definida para  $b$  un número entero positivo.

Profundizar en la representación de enteros en python:

- [https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To store integers%2C the computers,numbers to represent the integers.&text=By using 8 bits%2C you,8 – 1 %3D 255 integers.](https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To%20store,integers%2C%20the%20computers%2C%20numbers%20to%20represent%20the%20integers.&text=By%20using%208%20bits%2C,you,%208%20-%201%20=%20255%20integers.)

```
import numpy as np
z=np.uint8(255)
z
```

255

```
z,type(z),z nbytes
```

(255, numpy.uint8, 1)

```
x=np.int8(4)+np.int8(6)
y=np.uint8(100)+np.uint8(27)
```

```
x, y, type(x), type(y)
```

(10, 127, numpy.int8, numpy.uint8)

## Tipo de datos REAL

- Todo intervalo de números reales contiene infinitos números.
- Los números reales son densos.
- El tipo *real* no representa exactamente a los números reales.
- El tipo *real* es un conjunto finito de representantes de intervalos de números reales.
- El tipo *real* no es un tipo ordinal.
- Cursos de Sistemas Númericos, Análisis numérico.

Los siguientes operadores están definidos para este tipo de datos.

- adición  $+$ : *real*  $\times$  *real*  $\rightarrow$  *real*
  - sustracción  $-$  : *real*  $\times$  *real*  $\rightarrow$  *real*
  - multiplicación  $*$  : *real*  $\times$  *real*  $\rightarrow$  *real*
  - división  $/$  : *real*  $\times$  (*real*  $- \{0\}$ )  $\rightarrow$  *real*
- La operación  $a/b$  no está definida para  $b = 0$ .

## Char

### Tipo de datos CHAR

El tipo *char* denota un *conjunto de caracteres*, finito y ordenado.

- 26 letras
- 10 dígitos
- caracteres especiales, como el carácter para espacio y otros símbolos de puntuación.

## Funciones de transferencia y funciones predecesor y sucesor

Las funciones de transferencia del tipo *char* al tipo *integer* y viceversa, están definidas por:

- $ord(c)$  es el número entero, llamado ordinal, del carácter  $c$  en el conjunto de caracteres.
- $chr(i)$  es el carácter con número ordinal  $i$ .

Por tanto, se tienen las siguientes relaciones

$$chr(ord(c)) = c \quad \text{y} \quad ord(chr(i)) = i$$

y el ordenamiento del conjunto de caracteres está definido por:

$$c_1 < c_2 \text{ si y solamente si } ord(c_1) < ord(c_2)$$

Las funciones predecesor y sucesor están definidas respectivamente por:

$$pred(c) = chr(ord(c) - 1) \quad \text{y} \quad succ(c) = chr(ord(c) + 1)$$

```
ord("a"),ord("A"), ord("b"),ord("c"),ord("d")
```

```
(97, 65, 98, 99, 100)
```

```
'a'>'b'
```

```
False
```

```
'a'>'A'
```

```
True
```

```
chr(97),chr(98)
```

```
('a', 'b')
```

## Profundizar en la gestión de memoria en Python

- [customprogrammingsolutions/python-memory-usage](https://customprogrammingsolutions.com/python-memory-usage)
- [https://codeblessu.com/python/size-of-data-types.html#:~:text=Use sys.,to store it in memory](https://codeblessu.com/python/size-of-data-types.html#:~:text=Use%20sys.%20to%20store%20it%20in%20memory)
- <https://www.lesinskis.com/images/CPythonMemoryStructurePosterJanisLesinskisAlyshalannetta.pdf>

```
from sys import getsizeof
import sys as yi
```

```
help(getsizeof)
```

Help on built-in function getsizeof in module sys:

```
getsizeof(...)
 getsizeof(object [, default]) -> int

 Return the size of object in bytes.
```

```
a=1000000000000000000000000000000000000000000000000000000000000000
print ("a=",a,'el tamaño es=', getsizeof(a))
print ("b=", b,'el tamaño es=',getsizeof(b))
print ("c=",c,'el tamaño es=',getsizeof(c))
print ("d=",d,'el tamaño es=',getsizeof(d))
```

```
a= 1000000000000000000000000000000000000000000000000000000000000000 el tamaño es= 40
b= 3 el tamaño es= 28
c= 3.0 el tamaño es= 24
d= holaWorld el tamaño es= 58
```

```
a = 5
b = np.int8(5)
c = 'reine'
d = np.array([1,2,3,4]).astype('int8')

print(type(a), getsizeof(a))
print(b.nbytes, type(b))
print(type(c), getsizeof(c))
print(d.nbytes, type(d))
```

```
<class 'int'> 28
1 <class 'numpy.int8'>
<class 'str'> 54
4 <class 'numpy.ndarray'>
```

```
x=np.int8(128)
```

`x, x nbytes, type(x)`

(-128, 1, numpy.int8)

$$y=40$$

`getsizeof(y), type(y)`

(28, int)

`type(x), sizeof(x)`

(int, 96)

¿Por qué el resultado de las dos últimas divisiones no es el mismo?

```
a=10
b=3
c=3.
print ("a =",a,"; b =",b, " ; c =",c)
print ("a/b =", a/b)
print ("a/c =", a/c)
print ("a//b =", a//b)
print ("a//c =", a//c)
```

```
a = 10 ; b = 3 ; c = 3.0
a/b = 3.333333333333335
a/c = 3.333333333333335
a//b = 3
a//c = 3.0
```

```
Variable = """
Esta es una cadena
de varias líneas
"""
```

```
print(Variable)
```

```
Esta es una cadena
de varias líneas
```

```
type(Variable)
```

```
str
```

```
getsizeof(Variable)
```

```
110
```

```
x=np.array(['a','b','c'])
```

```
x.nbytes,type(x)
```

```
(12, numpy.ndarray)
```

```
x='2'
y='3'
z=x+y
print('z=',z)

x1=int(x)
y1=int(y)
z1=x1+y1
print('z1=',z1)
```

```
z= 23
z1= 5
```

```
print("Suma = ", 4+5)
print("Resta = ", 4-5)
print("Multiplicación = ", 4*5)
print("División = ", 4/5)
print("División entera = ", 10//3)
print("Residuo = ", 10%3)
print("Potencia = ", 2**3)
print("Raíz = ", 2**0.5)
```

```
Suma = 9
Resta = -1
Multiplicación = 20
División = 0.8
División entera = 3
Residuo = 1
Potencia = 8
Raíz = 1.4142135623730951
```

```
print("Redondear números= ",round(3.867483,3))
print("Valor Absoluto= ",abs(-4))
print("Convertir número a decimal= ",float('3.5'))
print("Convertir número a hexadecimal=", hex(5))
print("Convertir cadena o número decimal a entero=", int(6.9))
print("Convertir número a cadena de texto=", str(6)+str(6))
```

```
Redondear números= 3.867
Valor Absoluto= 4
Convertir número a decimal= 3.5
Convertir número a hexadecimal= 0x5
Convertir cadena o número decimal a entero= 6
Convertir número a cadena de texto= 66
```

```
import random
print('imprime un numero aleatorio entero=' ,random.randrange(3,10))
print('selecciona aleatoriamente=' ,random.choice(["uno", "dos", "tres"]))
print('imprime un numero aleatorio decimal entre 0 y 1=' ,random.random())
print('imprime una letra aleatoria=' ,random.choice("AEIOU"))
```

```
imprime un numero aleatorio entero= 9
selecciona aleatoriamente= dos
imprime un numero aleatorio decimal entre 0 y 1= 0.5008250519253324
imprime una letra aleatoria= U
```

```
print('imprime un numero aleatorio decimal entre 0 y 4=' ,int(4*random.random()))
```

```
imprime un numero aleatorio decimal entre 0 y 4= 3
```

```
import math
print("Factorial=",math.factorial(3))
print("PI=",math.pi)
print("E=",math.e)
print("seno=",math.sin(math.pi/6))
print("aseño=",math.degrees(math.asin(0.5)))
print("Logaritmo Natural=",math.log(math.exp(2)))
print("Logaritmo en base 10=",math.log10(100))
print("Logaritmo en base 2=",math.log2(8))
print("Potencia=",math.pow(3,2))
print("Raiz cuadrada=",math.sqrt(4))
```

```
Factorial= 6
PI= 3.141592653589793
E= 2.718281828459045
seno= 0.4999999999999994
aseño= 30.00000000000004
Logaritmo Natural= 2.0
Logaritmo en base 10= 2.0
Logaritmo en base 2= 3.0
Potencia= 9.0
Raiz cuadrada= 2.0
```

```
import numpy as np
np.float(5/0)
```

```
<ipython-input-83-136236f1537d>:2: DeprecationWarning: `np.float` is a deprecated alias
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rele
np.float(5/0)
```

```

ZeroDivisionError Traceback (most recent call last)
<ipython-input-83-136236f1537d> in <module>
 1 import numpy as np
----> 2 np.float(5/0)

ZeroDivisionError: division by zero
```

5/0

```

ZeroDivisionError Traceback (most recent call last)
<ipython-input-82-0106664d39e8> in <module>
----> 1 5/0

ZeroDivisionError: division by zero
```

```
#Suma o concatenación
x="Un divertido"+"programa"+ "de" + "radio"
print(x)
#Multiplicación de una cadena con un número
y=3*"programas"
print(y)
#Obtener longitud de una cadena
print('longitud de la cadena:',len(y))
```

Un divertidoprogamaderadio  
programasprogramasprogramas  
longitud de la cadena: 27

```
#Acceder a una posición de la cadena
cadena = "programA"
print('acceder a una posicion determinada:',cadena[0])
print('acceder a la última posición:',cadena[-1])

Comillas simples
cadena = 'Texto entre comillas simples'
print (cadena)
print (type(cadena))

Comillas dobles
cadena = "Texto entre comillas dobles"
print (cadena)
print (type(cadena))
```

```
acceder a una posicion determinada: p
acceder a la última posición: A
Texto entre comillas simples
<class 'str'>
Texto entre comillas dobles
<class 'str'>
```

cad='reinel'

cad[2:5]

'ine'

```
cadenaesc = 'Texto entre \n\n\n \t\t\tcomillas simples'
print (cadenaesc)
print (type(cadenaesc))
```

Texto entre

comillas simples  
<class 'str'>

Las tuplas son más rápidas que las listas. Si define un conjunto **constante** de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.

Una tupla es una secuencia de items ordenada e inmutable.

Los items de una tupla pueden ser objetos de cualquier tipo.

Para especificar una tupla, lo hacemos con los elementos separados por comas dentro de **paréntesis**.

Una tupla con únicamente dos elementos es denominada par.

Para crear una tupla con un único elemento (singleton), se añade una coma al final de la expresión.

Para definir una tupla vacía, se emplean unos paréntesis vacíos.

```
tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
print(tupla)
print(type(tupla))
print(tupla[1:4])
print('cuantas veces hay un elemento=',tupla.count(25))
print('dice en que posicion esta el elemento=',tupla.index(15))
#tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.
```

```
('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
<class 'tuple'>
(15, 2.8, 'otro dato')
cuantas veces hay un elemento= 2
dice en que posicion esta el elemento= 1
```

```

TypeError Traceback (most recent call last)
<ipython-input-6-1233f0186c5b> in <module>
 5 print('cuantas veces hay un elemento=',tupla.count(25))
 6 print('dice en que posicion esta el elemento=',tupla.index(15))
----> 7 tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.

TypeError: 'tuple' object does not support item assignment
```

```
tupla, type(tupla), tupla[1:-1],tupla.count(25),tupla.index(15)
```

```
(('cadena de texto', 15, 2.8, 'otro dato', 25, 25),
 tuple,
 (15, 2.8, 'otro dato', 25),
 2,
 1)
```

```
Usando el tag: raises-exception, se evita que se detenga en compilación
#tupla[0]=2
```

Una lista es una secuencia ordenada de elementos **mutable**.

Los items de una lista pueden ser objetos de distintos tipos.

Para especificar una lista se indican los elementos separados por comas en el interior de **CORCHETES**.

Para denotar una lista vacía se emplean dos corchetes vacíos.

```
a = [1,2,3, "hola", [10, "nunca", 90], -32]
print ("Toda la lista= ",a)
print(type(a))
print ("Longitud de la lista= ", len(a))
print("Acceder a un rango de posiciones=",a[0:4])
print("Acceder a una posición específica= ",a[4])
print("Acceder a la última posición= ",a[-1])
```

```
Toda la lista= [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
<class 'list'>
Longitud de la lista= 6
Acceder a un rango de posiciones= [1, 2, 3, 'hola']
Acceder a una posición específica= [10, 'nunca', 90]
Acceder a la última posición= -32
```

```
print("Lista= ",a)
a.append("Nuevo último")
print("Lista con nuevo último= ",a)
a[1]='nuevo1'
print("Lista con nuevo elemento en las posición 1= ",a)
a.insert(2,'nuevo2')
print("Lista con nuevo elemento en las posición 2= ",a)
```

```
Lista= [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
Lista con nuevo último= [1, 2, 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
Lista con nuevo elemento en las posición 1= [1, 'nuevo1', 3, 'hola', [10, 'nunca', 90]
Lista con nuevo elemento en las posición 2= [1, 'nuevo1', 'nuevo2', 3, 'hola', [10, '
```

a,type(a),len(a)

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último'],
list,
8)
```

```
for i in range(len(a)):
 print (i, "-->", a[i])
```

```
0 --> 1
1 --> nuevo1
2 --> nuevo2
3 --> 3
4 --> hola
5 --> [10, 'nunca', 90]
6 --> -32
7 --> Nuevo último
```

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

```
1
nuevo1
nuevo2
3
hola
[10, 'nunca', 90]
-32
Nuevo último
```

```
print(a)
print (a[2:])
print (a[-3:])
print (a[4])
```

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
['nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
[[10, 'nunca', 90], -32, 'Nuevo último']
hola
```

```
a = [1,2,3,"hola", [10, "nunca", 90], -32]
print (a[4])
print (a[4][0])
print (a[4][1])
print (a[4][1][2:])
```

```
[10, 'nunca', 90]
10
nunca
nca
```

Los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados. En Python, un diccionario es una colección **no-ordenada** de valores que son accedidos a través de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos. Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior. No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves. La información almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario. Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

```
Definir una variable diccionario
futbolistas = dict()

futbolistas = {
 1: "Casillas", 6: "Iniesta", 3: "Piqué",
 5: "Puyol",
 7: "Villa", 8: "Xavi Hernández",
 9: "Torres", 11: "Capdevila",
 14: "Xavi Alonso", 15: "Ramos",
 16: "Busquets"
}

Recorrer el diccionario, imprimiendo clave - valor
print ("Vemos que los elementos no van \"ordenados\":")
for k, v in futbolistas.items():
 print ("{} --> {}".format(k,v))
```

Vemos que los elementos no van "ordenados":

```
1 --> Casillas
6 --> Iniesta
3 --> Piqué
5 --> Puyol
7 --> Villa
8 --> Xavi Hernández
9 --> Torres
11 --> Capdevila
14 --> Xavi Alonso
15 --> Ramos
16 --> Busquets
```

```
futbolistas.items(),futbolistas.keys(),futbolistas.values()
```

```
(dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa')]),
 dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16]),
 dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres'])
```

```
Nº de elementos que tiene un diccionario
numElemen = len(futbolistas)
print ("\nEl número de futbolistas es de {}".format(numElemen))

Imprimir las claves que tiene un diccionario
keys = futbolistas.keys();
print ("\nLas claves de nuestro diccionario son : {}".format(keys))

Imprimir los valores que tiene un diccionario
values = futbolistas.values();
print ("\nLos valores de nuestro diccionario son : {}".format(values))
```

El número de futbolistas es de 11

Las claves de nuestro diccionario son : dict\_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16])

Los valores de nuestro diccionario son : dict\_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres', 'Busquets', 'Ramos', 'Capdevila', 'Alonso', 'Hernández', 'Torres'])

```
Obtener el valor de un elemento dada su clave
elem = futbolistas.get(6)
print ("\nEl futbolista con clave 6 es {}".format(elem))

Insertamos un elemento en el diccionario
si la clave ya existe, el elemento NO se inserta
futbolistas.setdefault(10, 'Cesc')
print ("\nInsertamos el elemento con clave 10 y valor Cesc")
print ("Ahora el diccionario queda : {}".format(futbolistas))

Añadimos, de otro modo, un elemento al diccionario
si la clave ya existe, el valor se cambia por este nuevo
futbolistas[22] = 'Navas'
print ("\nAñadimos un nuevo elemento, ahora el diccionario queda: ")
print (futbolistas)
```

El futbolista con clave 6 es Iniesta

Insertamos el elemento con clave 10 y valor Cesc

Ahora el diccionario queda : {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7:

Añadimos un nuevo elemento, ahora el diccionario queda:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
Eliminamos un elemento del diccionario dada su clave
futbolistas.pop(22)
print ("\nEliminamos el elemento con clave 22")
print ("Ahora el diccionario queda: {}".format(futbolistas))

Hacemos una copia del diccionario
futbolistas_copia = futbolistas.copy()
print ("\nLa copia del diccionario tiene los valores:")
print (futbolistas_copia)

Borramos los elementos de un diccionario
futbolistas_copia.clear()
print ("\nVaciamos el diccionario nuevo creado, ahora los valores en el: {}".format(fu
```

Eliminamos el elemento con clave 22

Ahora el diccionario queda: {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: '

La copia del diccionario tiene los valores:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

Vaciamos el diccionario nuevo creado, ahora los valores en el: {}

```
Comprobamos si existe o no una clave en un diccionario
if 2 in futbolistas:
 print ("\nEl futbolista con la clave 2 existe en el diccionario.")
else:
 print ("\nEl futbolista con la clave 2 NO existe en el diccionario.")

if 8 in futbolistas:
 print ("\nEl futbolista con la clave 8 existe en el diccionario.")
else:
 print ("\nEl futbolista con la clave 8 NO existe en el diccionario.")
```

El futbolista con la clave 2 NO existe en el diccionario.

El futbolista con la clave 8 existe en el diccionario.

```
Creamos un diccionario a partir de una lista de claves
keys = ['nombre', 'apellidos', 'edad']
datos_usuario = dict.fromkeys(keys, 'null')
print ("\nCreamos un diccionario a partir de la lista de claves:")
print (keys)
print ("y con valor 'null'")
print ("Así el diccionario queda: {}".format(datos_usuario))

Creación de un diccionario
diccionario=dict({1:'rei',2:'pepe',3:'javier'})
print(diccionario)
```

Creamos un diccionario a partir de la lista de claves:

```
['nombre', 'apellidos', 'edad']
y con valor 'null'
Así el diccionario queda: {'nombre': 'null', 'apellidos': 'null', 'edad': 'null'}
{1: 'rei', 2: 'pepe', 3: 'javier'}
```

```
Devolvemos los elementos del diccionario en una tupla
tupla = futbolistas.items()
print ("\nEl diccionario convertido en tupla queda así:")
print (tupla)

Unimos dos diccionarios existentes
suplentes = {
 4:'Marchena', 12:'Valdes', 13:'Mata',
 17:'Arbeloa', 19:'Llorente', 20:'Javi Martinez',
 21:'Silva', 23:'Reina'
}

print ("\nUnimos el diccionario: ")
print (futbolistas)
futbolistas.update(suplentes) # Aquí hacemos la unión de los diccionarios
print ("con el diccionario:")
print (suplentes)
print ("siendo el resultado:")
print (futbolistas)
```

El diccionario convertido en tupla queda así:  
dict\_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa'),  
  
Unimos el diccionario:  
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',  
con el diccionario:  
{4: 'Marchena', 12: 'Valdes', 13: 'Mata', 17: 'Arbeloa', 19: 'Llorente', 20: 'Javi Mar  
siendo el resultado:  
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
#Entrada de datos
x=input('ingrese el dato 1: ');
y=input('ingrese el dato 2: ');
print('tipo de datos',type(x),type(y))
print('suma de datos sin casteo',x+y)
print('suma de datos con casteo',float(x)+float(y))
```

```
ingrese el dato 1: 5
ingrese el dato 2: 6
tipo de datos <class 'str'> <class 'str'>
suma de datos sin casteo 56
suma de datos con casteo 11.0
```

Operador	Descripción	Ejemplo
<code>==</code>	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
<code>!=</code>	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<code>&lt;</code>	¿es a menor que b?	<code>r = 5 &lt; 3 # r es False</code>
<code>&gt;</code>	¿es a mayor que b?	<code>r = 5 &gt; 3 # r es True</code>
<code>&lt;=</code>	¿es a menor o igual que b?	<code>r = 5 &lt;= 5 # r es True</code>
<code>&gt;=</code>	¿es a mayor o igual que b?	<code>r = 5 &gt;= 3 # r es True</code>

```
a = 5
b = 5
a1 = 7
b1 = 3
c1 = 3

cadena1 = 'Hola'
cadena2 = 'Adios'

igual
c = a == b
print (c)

cadenas = cadena1 == cadena2
print (cadenas)

diferente
d = a1 != b
print (d)

cadena0 = cadena1 != cadena2
print (cadena0)

mayor que
e = a1 > b1
print (e)

menor que
f = b1 < a1
print (f)

mayor o igual que
g = b1 >= c1
print (g)

menor o igual que
h = b1 <= c1
print (h)
```

```
True
False
True
True
True
True
True
True
```

```
5==5
```

True

```
7!=9
```

True

```
cadena1 = 'Hola'
```

```
cadena1
```

'Hola'

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

```
x=3
y=7
if x==3 and y==7:
 print('si')
else:
 print('no')
```

si

```
if condicion1:
 instruccion1
 instruccion2
else:
 if condición2:
 instruccion3
 instruccion4
 else:
 instruccion5
 instruccion6
fin del if
instruccion7
instruccion8
```

```
if 5>2:
 print('si')
 print('si2')
elif 1>3:
 print('sino si')
else:
 print('no')
 print('PorFuera')
print('PorFuera2')
```

si  
si2  
PorFuera2

## Ejercicios

1. Realice un programa que pida la edad de una persona en años. Si la edad es mayor o igual a 18, el programa debe imprimir la cadena: 'ADULTO'. Si la edad es menor a 18 se debe imprimir: 'MENOR DE EDAD'.
2. Diseñe un algoritmo que determine si un número es o no es, par positivo.
3. Diseñe un algoritmo que lea un número de tres cifras y determine si es igual al revés del número.

# Escribe aquí tu código

```
#ciclo for
print("Ciclo for en un rango de datos: ")
for i in range(2,10):
 print(i)

print('Ciclo for en una colección de datos:')
x=[3,4,5,7,8,"Hola"]
for i in x:
 print("El elemento es= ",i)
for i in range(len(x)):
 print("El elemento de la posición",i,'es:',x[i])
```

Ciclo for en un rango de datos:

2  
3  
4  
5  
6  
7  
8  
9

Ciclo for en una colección de datos:

El elemento es= 3  
El elemento es= 4  
El elemento es= 5  
El elemento es= 7  
El elemento es= 8  
El elemento es= Hola  
El elemento de la posición 0 es: 3  
El elemento de la posición 1 es: 4  
El elemento de la posición 2 es: 5  
El elemento de la posición 3 es: 7  
El elemento de la posición 4 es: 8  
El elemento de la posición 5 es: Hola

```
print('Ciclo for en un diccionario:')
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
for nombre, color in frutas.items():
 print (nombre, "es de color", color)
for llave in frutas:
 print(llave, 'es de color', frutas[llave])
```

Ciclo for en un diccionario:

Fresa es de color roja  
Limon es de color verde  
Papaya es de color naranja  
Manzana es de color amarilla  
Guayaba es de color rosa  
Fresa es de color roja  
Limon es de color verde  
Papaya es de color naranja  
Manzana es de color amarilla  
Guayaba es de color rosa

frutas

```
{'Fresa': 'roja',
'Limon': 'verde',
'Papaya': 'naranja',
'Manzana': 'amarilla',
'Guayaba': 'rosa'}
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
frutas
```

```
{'Fresa': 'roja',
'Limon': 'verde',
'Papaya': 'naranja',
'Manzana': 'amarilla',
'Guayaba': 'rosa'}
```

len(frutas)

5

```
frutas.get('Fresa')
```

'roja'

```
for i in range(10):
 print(i)
```

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

```
v=['a' , 'b' , 'c' , 'd']

for i in range(len(v)):
 print(i,"-->",v[i])
```

```
0 --> a
1 --> b
2 --> c
3 --> d
```

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

```
a
b
c
d
```

```
#ciclo while
cont=0
while cont<10:
 print(cont)
 cont+=2
```

```
0
2
4
6
8
```

```
v=['a','b','c','d']
```

```
for i in range(len(v)):
 print(i,v[i])
```

```
0 a
1 b
2 c
3 d
```

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

```
a
b
c
d
```

## Ejercicio

- Realice un programa que lea las notas de un estudiante, despues devolver el promedio, la mejor y la peor nota.

```
Escribe aquí tu código
```

## Algoritmo Multiplicación 1

Multiplica los numeros naturales **x** y **w** **w**, sumando **x** veces el número número; el resultado lo asigna a la variable **z**

## Nociones fundamentales

Multiplicar dos números naturales  $x$  y  $y$ , y denotar su producto por  $z$

Paso 1:  $z := 0;$

$u := x; \rightsquigarrow x = 5$

Paso 2: **repetir las instrucciones**

$z := z + y; \rightsquigarrow y = 13$

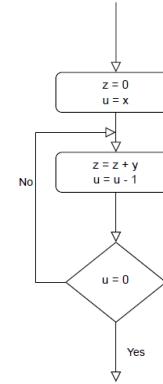
$u := u - 1;$

**hasta que  $u = 0$**

Valor de las variables

Paso	$z$	$u$
1	0	5
2	13	4
2	26	3
2	39	2
2	52	1
2	<b>65</b>	0

Multiplicación de dos números naturales  $x$  y  $y$



Pedimos introducir el primer número para multiplicar

```
x=int(input('Escribir un numero para multiplicar: '))
```

Escribir un numero para multiplicar: 6

Pedimos introducir el segundo número para multiplicar

```
w=int(input('Escribir otro numero para multiplicar: '))
```

Escribir otro numero para multiplicar: 5

Inicialización y asignación de variable

```
z=0
```

```
u=x
```

Realizamos la operación de multiplicación con sumas y estructuras repetitivas

```
while u>0:
 z=z+w
 u=u-1
```

Mostramos el resultado

```
print('El resultado de la multiplicacion es: ', z)
```

El resultado de la multiplicacion es: 30

Condensando todo el algoritmos en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir un numero para multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
paso1=1
z=0
u=x
paso2=2
print('Paso', ' ', 'Z', ' ', 'U')
print(paso1, ' ', z, ' ', u)
while u>0:
 z=z+w
 u=u-1
 print(paso2, ' ', z, ' ', u)
print('El resultado de la multiplicacion es: ', z)
```

Escribir un numero para multiplicar: 5  
 Escribir otro numero para multiplicar: 6  
 Paso Z U  
 1 0 5  
 2 6 4  
 2 12 3  
 2 18 2  
 2 24 1  
 2 30 0  
 El resultado de la multiplicacion es: 30

## Algoritmo División 1

Divide dos números naturales **x** y **w**, utilizando restas; el cociente es asignado a la variable **q** y el residuo a **r**.

Dividir el número natural  $x$  por el número natural  $y$ , y denotar por  $q$  el cociente y el residuo por  $r$ .

#### Teorema del cociente y del residuo:

Dado cualquier entero  $a$  y un entero positivo  $b$  existen dos enteros únicos  $q$  y  $r$  tales que:

$$a = q * b + r$$

donde  $0 \leq r \leq b$ .  $q$  se llama el cociente y  $r$  el residuo.

Notación:  $q = a \text{ div } b$  y  $r = a \text{ mod } b$

$$a = (a \text{ div } b) * b + (a \text{ mod } b)$$

**División entera de  $x$  entre  $y$ :** el cociente  $q$  es el número de veces que se puede restar (sustraer) el divisor  $y$  al dividendo  $x$  de tal forma que el resultado (cociente) sea no negativo.

Dividir el número natural  $x$  por el número natural  $y$ , y denotar por  $q$  el cociente y el residuo por  $r$ .

Paso 1:  $q := 0;$

$$r := x; \quad \rightsquigarrow x = 100$$

Paso 2: mientras que  $r \geq y$  repetir

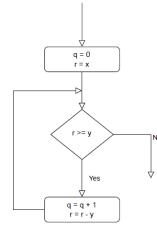
$$q := q + 1;$$

$$r := r - y; \quad \rightsquigarrow y = 15$$

Valor de las variables

Paso	$q$	$r$
1	0	100
2	1	85
2	2	70
2	3	55
2	4	40
2	5	25
2	6	10

División de dos números naturales  $x$  y  $y$



Pedimos introducir el dividendo de la división

```
x=int(input('Escribir el dividendo: '))
```

Escribir el dividendo: 9

Pedimos introducir el divisor de la división

```
w=int(input('Escribir el divisor: '))
```

Escribir el divisor: 2

Inicialización y asignación de variable

```
q=0
r=x
```

Realizamos la operación de división con restas y estructuras repetitivas

```
while r>=w:
 q=q+1
 r=r-w
```

Mostramos el cociente y el residuo

```
print('El cociente de la división es: ',q,'\\n El residuo de la división es: ',r)
```

```
El cociente de la división es: 4
El residuo de la división es: 1
```

Condensando todo el algoritmo en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir el dividendo: '))
w=int(input('Escribir el divisor: '))
paso1=1
q=0
r=x
paso2=2
print('Paso', ' ', 'q', ' ', 'r')
print(paso1, ' ', q, ' ', r)
while r>=w:
 q=q+1
 r=r-w
 print(paso2, ' ', q, ' ', r)
print('El cociente de la división es: ',q, '\n El residuo de la división es: ',r)
```

```
Escribir el dividendo: 9
Escribir el divisor: 2
Paso q r
1 0 9
2 1 7
2 2 5
2 3 3
2 4 1
El cociente de la división es: 4
El residuo de la división es: 1
```

Python es un lenguaje **indentado**, no usa corchetes para delimitar el alcance de las estructuras de programación sino que se fija en los **cambios de indentación**.

No se declara el tipo de los argumentos de las funciones. La semántica de la implementación ha de estar preparada para funcionar con los tipos de datos que quieras.

```
import numpy as np
def funcion_1(a,b):
 r = a**2
 return r+b

def greatest(a,b):
 if a>b:
 return a
 else:
 return b
```

```
funcion_1 (10.4,2)
```

```
110.16000000000001
```

```
funcion_1 (10.4, np.array([2,4]))
```

```
array([110.16, 112.16])
```

```
funcion_1(np.array([1,5]),np.array([3,2]))
```

```
array([4, 27])
```

```
m1 = np.array([[3,4],[1,1]])
m2 = np.array([[5,6],[0,0]])
```

```
funcion_1 (m1,m2)
```

```
array([[14, 22],
 [1, 1]])
```

```
greatest(10,2)
```

```
10
```

Podemos definir valores por defecto para los argumentos de las funciones y llamarlas usando explícitamente el nombre de los argumentos. Además, las funciones pueden devolver varios valores.

```
def f_power(x, p=2):
 return x**p
```

```
f_power(x=3)
```

9

```
f_power(p=4, x=3)
```

81

```
def f_power(x, p=2):
 return x**p, x*p
```

```
r = f_power(p=4, x=3)
print(r[0],r[1],"\n")
```

81 12

```
r1, r2 = f_power(p=4, x=3)
print(r1)
print(r2, "\n")
```

81

12

```
a,b = 10, np.array([10,4,-3])
print(a)
print(b)
```

10

[10 4 -3]

```
def f_power(x, p=2):
 return x**p

def f_powers(x, p1=2, p2=3):
 return x**p1, x**p2
```

```
f_power(4)
```

```
16
```

```
f_power(4,3)
```

```
64
```

```
f_powers(4, p1=3)
```

```
(64, 64)
```

```
xp1, xp2 = f_powers(4, p2=4, p1=3)
print("power1",xp1, "power2", xp2)
```

```
power1 64 power2 256
```

Dependiendo del tipo de dato que enviamos a la **función**, podemos diferenciar dos comportamientos:

**Paso por valor:** Se crea una copia local de la variable dentro de la función.

**Paso por referencia:** Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

**Los tipos simples se pasan por valor:** Enteros, flotantes, cadenas, lógicos...

**Los tipos compuestos se pasan por referencia:** Listas, diccionarios, conjuntos...

### Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(x):
 x = 2*x
 return x
```

```
x=10
doblar_valor(x)
print(x,doblar_valor(x))
```

10 20

Para modificar los tipos simples podemos devolverlos modificados y reasignarlos:

```
x = 10
x = doblar_valor(x)
print(x)
```

20

## Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(y):
 for i,n in enumerate(y):
 y[i] = 2*y[i]
```

```
y=[1,2,3]
```

```
doblar_valores(y)
```

```
print(y)
```

[2, 4, 6]

# Algoritmo multiplicación 2

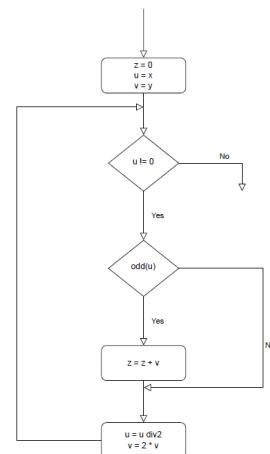
Otro algoritmo para la multiplicación de dos números naturales

Paso 1:  $z := 0; u := x; v := y$ ;  $\rightsquigarrow x = 9, y = 5$   
 Paso 2: mientras que  $u \neq 0$  repetir  
     si  $\text{odd}(u)$  entonces  $z := z + v$ ;  
      $u := u \text{ div } 2$ ;  
      $v := 2 * v$

Valor de las variables			
Paso	$z$	$u$	$v$
1	0	9	5
2	5	4	10
2	5	2	20
2	5	1	40
2	<b>45</b>	0	80

$$9 = 2 * 2 * 2 + 1 \quad 2(2(2(5))) + 5 = 45$$

Otro algoritmo para la multiplicación de dos números naturales



```
función para determinar el cociente
def funcionCociente(x,w):
```

```
 q=0
 r=x
 while r>=w:
 q=q+1
 r=r-w
 return q
```

```
función para determinar el residuo
def funcionResiduo(x,w):
```

```
 q=0
 r=x
 while r>=w:
 q=q+1
 r=r-w
 return r
```

```
función para determinar si un numero es par
def funcionPar(u):
```

```
 if funcionResiduo(u,2)==0:
 par=True
 else:
 par=False
 return par
```

```
def funcionMultiplicar(x,w):
 z=0
 u=x
 v=w
 while u!=0:
 if not funcionPar(u):
 z=z+v
 u=funcionCociente(u,2)
 v=2*v
 return z
```

```
x=int(input('Escribir un numero par multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
print('El resultado de la mutiplicación es: ', funcionMultiplicar(x,w))
```

Escribir un numero par multiplicar: 6  
 Escribir otro numero para multiplicar: 5  
 El resultado de la mutiplicación es: 30

## Catching

```
int(3.2)
```

3

```
int("hola")
```

```

ValueError Traceback (most recent call last)
<ipython-input-2-62025142aa33> in <cell line: 1>()
----> 1 int("hola")

ValueError: invalid literal for int() with base 10: 'hola'
```

Manejar cualquier tipo de error que pase en un bloque de código

```
def ejemplo_try_except(valor):
 try:
 resultado = 10 / valor
 print("El resultado es:", resultado)
 except Exception as e:
 print("Ha ocurrido un error:", e)

Ejemplos de llamadas a la función
ejemplo_try_except(2)
ejemplo_try_except(0)
ejemplo_try_except('cadena')
```

El resultado es: 5.0  
 Ha ocurrido un error: division by zero  
 Ha ocurrido un error: unsupported operand type(s) for /: 'int' and 'str'

### Manejando un tipo específico de error

```
def int_times_two(x):
 try:
 return(int(x)*2)
 except ValueError:
 return 0
```

int\_times\_two(2)

4

int\_times\_two(2.5)

4

int\_times\_two("hola")

0

### Manejo de la sentencia finally para que, pase o no pase un error, siempre ejecute lo que se coloque en la sentencia finally

```
def division_segura(dividendo, divisor):
 try:
 resultado = dividendo / divisor
 return resultado
 except ZeroDivisionError:
 print("¡Error! No puedes dividir entre cero.")
 return None
 finally:
 print("Operación finalizada.")
```

```
dividendo = 10
divisor = 2

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
 print("El resultado de la división es:", resultado_division)
```

Operación finalizada.  
El resultado de la división es: 5.0

```
dividendo = 10
divisor = 0

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
 print("El resultado de la división es:", resultado_division)
```

¡Error! No puedes dividir entre cero.  
Operación finalizada.

## Raising

Este tipo de sentencias se utilizan para generar errores personalizados dentro del código

```
def get_positive_integer_from_user():
 a = int(input("Enter a positive integer: "))
 if a <= 0:
 raise ValueError("That is not a positive number!")
 print ("thanks!")
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: 1
thanks!
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: -1
```

```

ValueError Traceback (most recent call last)
<ipython-input-13-ea5635cae962> in <cell line: 1>()
----> 1 get_positive_integer_from_user()

<ipython-input-11-a7d1640efb92> in get_positive_integer_from_user()
 2 a = int(input("Enter a positive integer: "))
 3 if a <= 0:
----> 4 raise ValueError("That is not a positive number!")
 5 print ("thanks!")

ValueError: That is not a positive number!
```

# Introducción a Pandas

## Contenido

- Series de pandas
- DataFrames de pandas
- Cargar Fuentes externas de datos

Pandas es una biblioteca de Python fundamental para la manipulación y el análisis de datos, ampliamente utilizada en el ámbito de la ciencia de datos y la ingeniería de datos. Proporciona estructuras de datos fáciles de usar y de alto rendimiento, como DataFrames y Series, que permiten realizar operaciones complejas de limpieza, transformación, y análisis de datos con una sintaxis intuitiva y eficiente.

```

¿Cómo importamos la librería?
import pandas as pd
```

## Series de pandas

Una pandas Series es una estructura de datos unidimensional en la biblioteca pandas de Python, similar a una columna en una tabla de datos o a un array en otros lenguajes de programación. Cada Series tiene un índice asociado, que son las etiquetas de las filas que permiten acceder a los elementos de la serie de manera rápida y eficiente. Los elementos de una Series pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

```
pd.Series([-2, -3, 0, 3, 2], dtype='int8')
```

↓

		Data
Index	0	-2
	1	-3
	2	0
	3	3
	4	2

dtype: int8

© w3resource.com

Lo que hace que las Series sean especialmente poderosas es su capacidad para manejar datos faltantes y realizar operaciones vectorizadas, lo que significa que puedes aplicar operaciones a todos los elementos de una Series de una sola vez, en lugar de tener que iterar sobre cada elemento individualmente. Además, las Series ofrecen una variedad de métodos y propiedades para realizar operaciones estadísticas, manipulación de datos, y transformaciones, lo que las convierte en una herramienta versátil y esencial para el análisis de datos en Python.

```
Vamos a crear una serie a partir de un diccionario

test_balance_data = {
 'pasan': 20.00,
 'treasure': 20.18,
 'ashley': 1.05,
 'craig': 42.42,
}
```

```
Para crear una serie se debe partir de un objeto de tipo diccionario
balances = pd.Series(test_balance_data)
balances
```

```

pasan 20.00
treasure 20.18
ashley 1.05
craig 42.42
dtype: float64
```

```
Acá tenemos una serie de pandas, que es una estructura de datos que se parece a un diccionario.
las etiquetas de las filas se obtienen
balances.keys()
tal cual como el un diccionario.
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
Alternativamente, se pueden obtener con el método index
balances.index
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
Si solo quieres los valores, puedes usar:
balances.values
```

```
array([20. , 20.18, 1.05, 42.42])
```

```
Ahora, creemos una serie a partir de una lista
```

```
unlabeled_balances = pd.Series([20.00, 20.18, 1.05, 42.42])
unlabeled_balances # Note que las etiquetas de las filas son números enteros que van del 0 al 3
```

```
0 20.00
1 20.18
2 1.05
3 42.42
dtype: float64
```

```
Si quieres personalizar esos index, puedes hacerlo de la siguiente manera
labeled_balances = pd.Series(
 [20.00, 20.18, 1.05, 42.42],
 index=['pasan', 'treasure', 'ashley', 'craig'])
labeled_balances
```

```
pasan 20.00
treasure 20.18
ashley 1.05
craig 42.42
dtype: float64
```

```
Si pasas un escalar, recuerda que es un valor único,
este se transmitirá a cada una de las claves especificadas en el argumento de palabras
pd.Series(20.0, index=["guil", "jay", "james", "ben", "nick"])
```

```
guil 20.0
jay 20.0
james 20.0
ben 20.0
nick 20.0
dtype: float64
```

## Acceso a los elementos de una Serie

```
import pandas as pd

Creamos una serie a partir de un diccionario que tiene nombres de personas como claves
test_balance_data = {
 'pasan': 20.00,
 'treasure': 20.18,
 'ashley': 1.05,
 'craig': 42.42,
}

balances = pd.Series(test_balance_data)
```

## Acceso por indexación como lista

```
Una Serie es ordenada e indexada (como las listas de Python)
print(balances[0])
print(type(balances[0]))
```

```
20.0
<class 'numpy.float64'>
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:2: FutureWarning: print(balances[0])
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:3: FutureWarning: print(type(balances[0]))
```

```
La forma correcta es:
balances.iloc[0]
```

```
np.float64(20.0)
```

```
Si quieres obtener el ultimo balance
balances[-1]
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/3307448855.py:2: FutureWarning: balances[-1]
```

```
np.float64(42.42)
```

## Acceso por la clave de índice

```
Acceder a un elemento por su clave(index según pandas)
balances['pasan']
```

```
np.float64(20.0)
```

```
Las series de pandas se comportan como diccionarios
for label,value in balances.items():
 print(label, value)
```

```
pasan 20.0
treasure 20.18
ashley 1.05
craig 42.42
```

```
Acceder a un elemento usando su clave como si fuera un atributo
balances.ashley
```

```
np.float64(1.05)
```

```
Acceder a los elementos explicitamente como lo menciona la documentación
print(balances.loc['ashley'])
print(balances.iloc[0])
```

```
1.05
20.0
```

```
Acceder a una porción de la serie
balances[['treasure':'craig']]
```

```
treasure 20.18
ashley 1.05
craig 42.42
dtype: float64
```

```
balances.iloc[0:3]
```

```
pasan 20.00
treasure 20.18
ashley 1.05
dtype: float64
```

## Vectorización y broadcasting

se trata de la capacidad de pandas para realizar operaciones element-wise (elemento a elemento) en estructuras de datos con diferentes formas, alineando automáticamente las

dimensiones según sea necesario.

```
import pandas as pd

test_balance_data = {
 'pasan': 20.00,
 'treasure': 20.18,
 'ashley': 1.05,
 'craig': 42.42,
}

test_deposit_data = {
 'pasan': 20,
 'treasure': 10,
 'ashley': 100,
 'craig': 55
}

balances = pd.Series(test_balance_data)
deposits = pd.Series(test_deposit_data)
```

## Vectorización

Aunque es posible recorrer cada elemento y aplicarlo a otro...

```
for label, value in deposits.items():
 print(f"Depositando {value} en la cuenta de {label}, que tiene {balances[label]}")
 balances[label] += value

balances
```

Depositando 20 en la cuenta de pasan, que tiene 20.0  
 Depositando 10 en la cuenta de treasure, que tiene 20.18  
 Depositando 100 en la cuenta de ashley, que tiene 1.05  
 Depositando 55 en la cuenta de craig, que tiene 42.42

pasan	40.00
treasure	30.18
ashley	101.05
craig	97.42
dtype: float64	

...es importante recordar apoyarse en la vectorización y omitir los bucles por completo. La vectorización es más rápida y, como puedes ver, más fácil de leer y escribir.

```
Se debe considerar que para esta operacion los indices deben coincidir, de lo contrario se genera un error
balances += deposits
balances
agrega 10 a la cuenta de 'james' en el diccionario de depositos para y ejecuta nueva operacion
```

```
pasan 60.00
treasure 40.18
ashley 201.05
craig 152.42
dtype: float64
```

```
balances -= deposits
balances
```

```
pasan 40.00
treasure 30.18
ashley 101.05
craig 97.42
dtype: float64
```

## Broadcasting

Esto lo que permite es sumar un escalar a cada uno de los elementos de la serie.

```
balances + 5
```

```
pasan 45.00
treasure 35.18
ashley 106.05
craig 102.42
dtype: float64
```

Hacer broadcast de una serie con otra serie es posible siempre y cuando ambas tengan el mismo índice. En caso de que no sea así, se devolverá NaN.

```
coupons = pd.Series(1, ['craig', 'ashley', 'james'])
coupons
```

```
craig 1
ashley 1
james 1
dtype: int64
```

```
Sumemos los balances con los cupones, el resultado es una nueva serie
balances + coupons
```

```
ashley 102.05
craig 98.42
james NaN
pasan NaN
treasure NaN
dtype: float64
```

```
balances
```

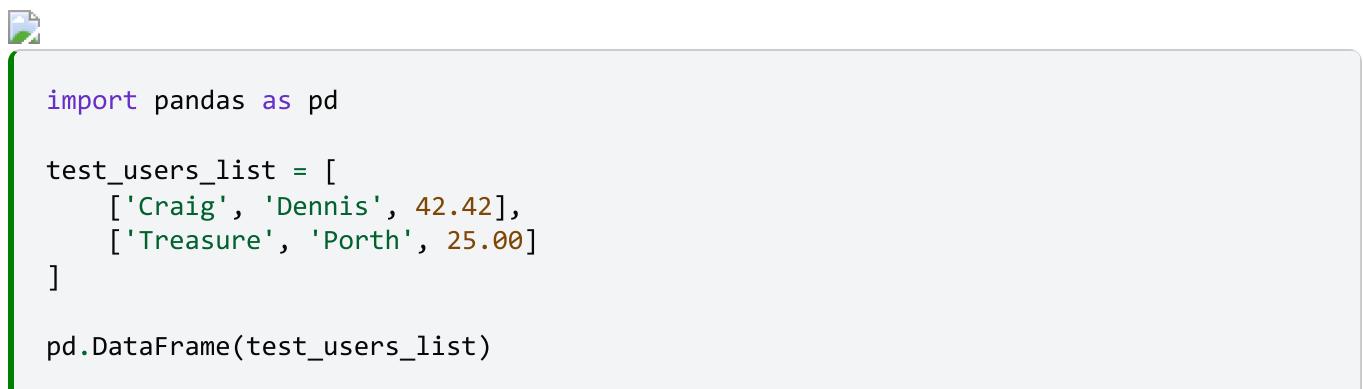
```
pasan 40.00
treasure 30.18
ashley 101.05
craig 97.42
dtype: float64
```

```
balances.add(coupons, fill_value=0) # Así se puede evitar el NaN y que no se pierda la
```

```
ashley 102.05
craig 98.42
james 1.00
pasan 40.00
treasure 30.18
dtype: float64
```

# DataFrames de pandas

Un DataFrame es una estructura de datos bidimensional en la biblioteca pandas de Python, similar a una tabla de datos o una hoja de cálculo en Excel. Cada DataFrame tiene un índice asociado, que son las etiquetas de las filas, y columnas, que son las etiquetas de las columnas. Los elementos de un DataFrame pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

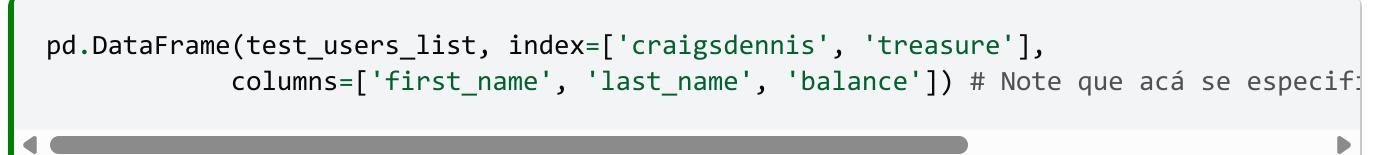


```
import pandas as pd

test_users_list = [
 ['Craig', 'Dennis', 42.42],
 ['Treasure', 'Porth', 25.00]
]

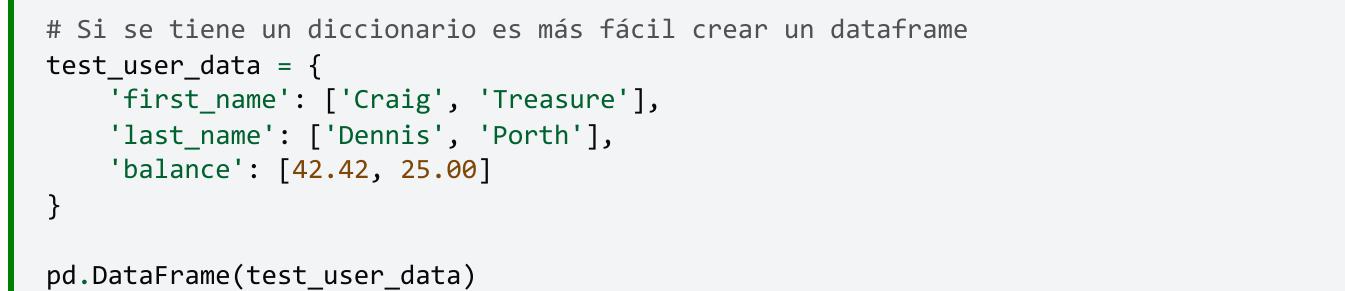
pd.DataFrame(test_users_list)
```

	0	1	2
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00



```
pd.DataFrame(test_users_list, index=['craigsdennis', 'treasure'],
 columns=['first_name', 'last_name', 'balance']) # Note que acá se especifica el índice y las columnas
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00



```
Si se tiene un diccionario es más fácil crear un dataframe
test_user_data = {
 'first_name': ['Craig', 'Treasure'],
 'last_name': ['Dennis', 'Porth'],
 'balance': [42.42, 25.00]
}

pd.DataFrame(test_user_data)
```

	<b>first_name</b>	<b>last_name</b>	<b>balance</b>
<b>0</b>	Craig	Dennis	42.42
<b>1</b>	Treasure	Porth	25.00

```
Se puede especificar los indices de las filas
pd.DataFrame(test_user_data, index=['craigslist', 'treasure'])
```

	<b>first_name</b>	<b>last_name</b>	<b>balance</b>
<b>craigslist</b>	Craig	Dennis	42.42
<b>treasure</b>	Treasure	Porth	25.00

## Acceso a los elementos de un DataFrame

```
Vamos a crear un dataframe con algunos datos de usuarios
import pandas as pd

test_user_data = { # Estos son los datos de los usuarios y el balance
 'first_name': ['Craig', 'Treasure', 'Ashley', 'Guil'],
 'last_name': ['Dennis', 'Porth', 'Boucher', 'Hernandez'],
 'balance': [42.42, 25.00, 2.02, 87.00]
}
test_user_names = ['craigslist', 'treasure', 'lindsay2000', 'guil'] # Supongamos que

users = pd.DataFrame(test_user_data, index=test_user_names)
users
```

	<b>first_name</b>	<b>last_name</b>	<b>balance</b>
<b>craigslist</b>	Craig	Dennis	42.42
<b>treasure</b>	Treasure	Porth	25.00
<b>lindsay2000</b>	Ashley	Boucher	2.02
<b>guil</b>	Guil	Hernandez	87.00

## Obtener una columna específica

Cada columna en un DataFrame es una Serie de pandas, por lo que puedes acceder a una columna específica utilizando la notación de corchetes y el nombre de la columna y el retorno es una serie.

```
balances = users['balance']
balances
```

```
craigsdennis 42.42
treasure 25.00
lindsay2000 2.02
guil 87.00
Name: balance, dtype: float64
```

```
La serie que retorna la celda anterior tiene una propiedad llamada name que retorna
balances.name
```

```
'balance'
```

Puedes obtener una fila de un Dataframe usando la propiedad .loc[] y pasando el índice de la fila, o el número.

```
users.loc['guil']
```

```
first_name Guil
last_name Hernandez
balance 87.0
Name: guil, dtype: object
```

```
users.iloc[3]
```

```
first_name Guil
last_name Hernandez
balance 87.0
Name: guil, dtype: object
```

## Recuperar un valor específico mediante encadenamiento

```
users['first_name']['craigdennis']
```

```
'Craig'
```

```
users.loc['craigdennis']['first_name']
```

```
'Craig'
```

```
users.loc['craigdennis', 'first_name']
```

```
'Craig'
```

```
users.at['craigdennis', 'first_name']
```

```
'Craig'
```

Usando las propiedades loc e iloc puedes dividir un DataFrame existente en uno nuevo.

En el ejemplo a continuación, usamos : en el eje de filas para seleccionar todas las filas, y especificamos qué columnas queremos recuperar usando una lista en el eje de columnas

```
Esto se lee como:
En el dataframe users, selecciona todas las filas, y dame solo las columnas 'balance'
users.loc[:, ['balance', 'last_name']]
```

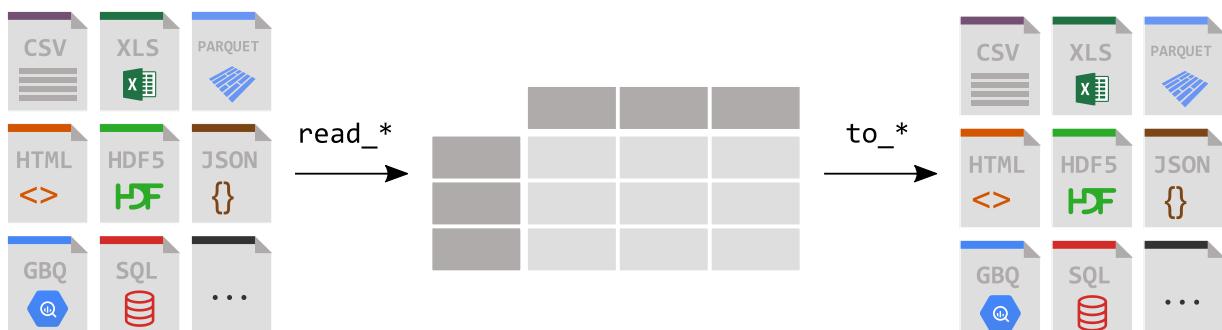
	<b>balance</b>	<b>last_name</b>
<b>craigslist</b>	42.42	Dennis
<b>treasure</b>	25.00	Porth
<b>lindsay2000</b>	2.02	Boucher
<b>guil</b>	87.00	Hernandez

```
O lo puedes indexar por posición como en las listas
users.iloc[1:3, 1:]
```

	<b>last_name</b>	<b>balance</b>
<b>treasure</b>	Porth	25.00
<b>lindsay2000</b>	Boucher	2.02

## Cargar Fuentes externas de datos

Naturalmente, cuando se trabaja con datos en la vida real, no siempre se tienen los datos en un DataFrame de pandas. A menudo, los datos se almacenan en archivos CSV, Excel, bases de datos SQL, o en la web. Afortunadamente, pandas proporciona una variedad de funciones para cargar datos desde diferentes fuentes y formatos, lo que facilita la importación y exportación de datos en Python.



La sintaxis general para cargar datos en un DataFrame

## de pandas es la siguiente:

```
import pandas as pd
pd.read_<formato>('ruta/al/archivo')
```

## La sintaxis general para guardar datos desde un DataFrame de pandas es la siguiente:

```
import pandas as pd
df.to_<formato>('ruta/al/archivo')
```

Los formatos más comunes para cargar y guardar datos en pandas son los siguientes:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

```
Vamos a importar los datos de un archivo CSV
1. Veamos lo que tiene el archivo users.csv
!head -n 5 data/users.csv # Este comando solo es una muestra del contenido del archivo
```

```
,first_name,last_name,email,email_verified,signup_date,referral_count,balance
aaron,Aaron,Davis,aaron6348@gmail.com,True,2018-08-31,6,18.14
acook,Anthony,Cook,cook@gmail.com,True,2018-05-12,2,55.45
adam.saunders,Adam,Saunders,adam@gmail.com,False,2018-05-29,3,72.12
adrian,Adrian,Fang,adrian.fang@teamtreehouse.com,True,2018-04-28,3,30.01
```

```
import pandas as pd

my_data = pd.read_csv('data/users.csv')
my_data # Note que hay una columna sin nombre, que es el índice y carga como Unnamed:
```

	Unnamed: 0	first_name	last_name	email	email_verified
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...	...	...	...	...	...
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

```
Usemos los parámetros para cargar el archivo CSV
my_data = pd.read_csv('data/users.csv', index_col=0)
Otros parametros que se pueden usar son:

sep: el separador de los datos. Algunos archivos CSV usan ; en lugar de ,
header: la fila que se usará como encabezado.
names: una lista con los nombres de las columnas.
skiprows: las filas que se deben omitir
na_values: los valores que se deben tratar como NaN
nrows: el número de filas que se deben leer
encoding: la codificación de caracteres que se debe usar

Existen otros parámetros que se pueden usar, pero estos son los más comunes.
Vamos a ver otros parámetros a lo largo de los ejemplos.

my_data
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>	
	<b>aaron</b>	Aaron	Davis	aaron6348@gmail.com	True
	<b>acook</b>	Anthony	Cook	cook@gmail.com	True
	<b>adam.saunders</b>	Adam	Saunders	adam@gmail.com	False
	<b>adrian</b>	Adrian	Fang	adrian.fang@teamtreehouse.com	True
	<b>adrian.blair</b>	Adrian	Blair	adrian9335@gmail.com	True
	...	...	...	...	.
	<b>wilson</b>	Robert	Wilson	robert@yahoo.com	False
	<b>wking</b>	Wanda	King	wanda.king@holt.com	True
	<b>wright3590</b>	Jacqueline	Wright	jacqueline.wright@gonzalez.com	True
	<b>young</b>	Jessica	Young	jessica4028@yahoo.com	True
	<b>zachary.neal</b>	Zachary	Neal	zneal@gmail.com	True

475 rows × 7 columns

## Exploraremos los datos que importamos

La forma más fácil si cargamos correctamente, podemos usar el método `.head()` para ver las primeras filas de un DataFrame, o `.tail()` para ver las últimas filas.

```
my_data.head() # Muestra las primeras 5 filas del dataframe
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>	
	<b>aaron</b>	Aaron	Davis	aaron6348@gmail.com	True
	<b>acook</b>	Anthony	Cook	cook@gmail.com	True
	<b>adam.saunders</b>	Adam	Saunders	adam@gmail.com	False
	<b>adrian</b>	Adrian	Fang	adrian.fang@teamtreehouse.com	True
	<b>adrian.blair</b>	Adrian	Blair	adrian9335@gmail.com	True

```
my_data.head(3)
```

	first_name	last_name	email	email_verified	signup
aaron	Aaron	Davis	aaron6348@gmail.com	True	2018-
acook	Anthony	Cook	cook@gmail.com	True	2018-
adam.saunders	Adam	Saunders	adam@gmail.com	False	2018-

```
Revisemos la cantidad de muestras que tiene el dataframe
len(my_data)
```

```
475
```

```
my_data.shape # Shape es un atributo que tiene todo DF (dataFrame) y retorna una tupla
```

```
(475, 7)
```

## Explorando la información

### Info

El método `.info()` proporciona una descripción concisa de un DataFrame, incluyendo el número de filas y columnas, el número de valores no nulos, el tipo de datos de cada columna, y la cantidad de memoria utilizada.

```
my_data.info() # Info es un método que retorna información sobre el dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 475 entries, aaron to zachary.neal
Data columns (total 7 columns):
 # Column Non-Null Count Dtype

 0 first_name 475 non-null object
 1 last_name 430 non-null object
 2 email 475 non-null object
 3 email_verified 475 non-null bool
 4 signup_date 475 non-null object
 5 referral_count 475 non-null int64
 6 balance 475 non-null float64
dtypes: bool(1), float64(1), int64(1), object(4)
memory usage: 26.4+ KB
```

## Counts

El método `.count()` cuenta, por cada columna, cuantos valores no nulos hay.

```
my_data.count()
```

```
first_name 475
last_name 430
email 475
email_verified 475
signup_date 475
referral_count 475
balance 475
dtype: int64
```

La mayoría de nuestras columnas incluyen valores para cada fila, pero parece que `last_name` tiene algunos valores faltantes. Los datos faltantes aparecerán como `np.nan` – no es un número de NumPy – en esos registros.

## Dtype

```
Revisemos los tipos de datos de las columnas
my_data.dtypes
```

```
first_name object
last_name object
email object
email_verified bool
signup_date object
referral_count int64
balance float64
dtype: object
```

a mayoría de los tipos de datos de estas columnas fueron inferidos o asumidos correctamente. Observa cómo email\_verified es automáticamente bool, referral\_count es un entero y balance un flotante. Esto es lo que hace pandas cuando usamos pd.read\_csv.

Sin embargo, una cosa a tener en cuenta es que el campo signup\_date es un objeto y no un datetime. Puedes convertir estos datos durante o después de la importación si lo necesitas, y haremos algunas de esas conversiones más adelante en este curso.

## Describe

El método DataFrame.describe es una excelente manera de obtener una idea general de todos los datos numéricos en tu DataFrame. Notarás que solo se devuelven las columnas que tienen datos numéricos, mientras que las que tienen valores booleanos o texto, como email\_verified y first\_name, se omiten.

Verás muchas agregaciones diferentes.

```
my_data.describe()
```

	<b>referral_count</b>	<b>balance</b>
<b>count</b>	475.000000	475.000000
<b>mean</b>	3.429474	49.933263
<b>std</b>	2.281085	28.280448
<b>min</b>	0.000000	0.050000
<b>25%</b>	2.000000	25.305000
<b>50%</b>	3.000000	51.570000
<b>75%</b>	5.000000	74.480000
<b>max</b>	7.000000	99.900000

## Otros métodos

```
El promedio
my_data.balance.mean()
```

```
np.float64(49.933263157894736)
```

```
La desviación estándar
my_data.balance.std()
```

```
np.float64(28.28044849675191)
```

```
La suma de todo el balance
my_data.balance.sum()
```

```
np.float64(23718.3)
```

```
El valor mínimo del balance
my_data.balance.min()
```

```
np.float64(0.05)
```

```
El valor máximo del balance
my_data.balance.max()
```

```
np.float64(99.9)
```

```
Obtener, de una columna en específico, el conteo de los valores únicos
my_data.email_verified.value_counts()
```

```
email_verified
True 389
False 86
Name: count, dtype: int64
```

## Reordenar columnas

Con pandas puedes ordenar rápidamente las columnas de un DF.

Ordenemos el DataFrame para que el usuario con el saldo más alto esté en la parte superior. Por defecto, se asume el orden ascendente, pero puedes cambiar eso configurando el argumento de palabra clave ascending en False.

```
my_data.sort_values(by='balance', ascending=False).head()
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>	<b>signup_</b>
<b>twhite</b>	Timothy	White	white5136@hotmail.com	True	2018-C
<b>karen.snow</b>	Karen	Snow	ksnow@yahoo.com	True	2018-C
<b>king</b>	Billy	King	billy.king@hotmail.com	True	2018-C
<b>king3246</b>	Brittney	King	brittney@yahoo.com	True	2018-C
<b>crane203</b>	Valerie	Crane	valerie7051@hotmail.com	True	2018-C

```
my_data.head()
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>
<b>aaron</b>	Aaron	Davis	aaron6348@gmail.com	True
<b>acook</b>	Anthony	Cook	cook@gmail.com	True
<b>adam.saunders</b>	Adam	Saunders	adam@gmail.com	False
<b>adrian</b>	Adrian	Fang	adrian.fang@teamtreehouse.com	True
<b>adrian.blair</b>	Adrian	Blair	adrian9335@gmail.com	True

Notarás que la llamada a `sort_values` en realidad creó un nuevo DataFrame. Si deseas cambiar permanentemente el orden predeterminado (ordenado por índice), puedes pasar `True` como argumento al parámetro de palabra clave `inplace`.

```
Ordenar primero por last_name y luego por first_name. Por defecto, np.nan aparece al principio
my_data.sort_values(by=['last_name', 'first_name'], inplace=True)
El orden ahora ha cambiado
my_data.head()
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>	<b>size</b>
<b>darlene.adams</b>	Darlene	Adams	adams@hotmail.com	True	1
<b>lauren</b>	Lauren	Aguilar	lauren.aguilar@summers.com	False	2
<b>daniel</b>	Daniel	Allen	allen@hotmail.com	False	3
<b>kallen</b>	Kathy	Allen	kathy@hotmail.com	False	4
<b>alvarado</b>	Denise	Alvarado	alvarado@hotmail.com	True	5

```
Si quieres ordenar por el índice, puedes usar sort_index

my_data.sort_index(inplace=True)
my_data.head()
```

	<b>first_name</b>	<b>last_name</b>	<b>email</b>	<b>email_verified</b>
aaron	Aaron	Davis	aaron6348@gmail.com	True
acook	Anthony	Cook	cook@gmail.com	True
adam.saunders	Adam	Saunders	adam@gmail.com	False
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	True
adrian.blair	Adrian	Blair	adrian9335@gmail.com	True

## Más fuentes de información con pandas

Exploraremos la forma en la que se cargan diferentes fuentes de datos con pandas, una vez cargados en un dataframe, se pueden aplicar los métodos y propiedades que hemos visto anteriormente.

## Cargar JSON

```
filepath = 'data/anscombe.json'

Leer el archivo JSON
data = pd.read_json(filepath)

Verificar los datos
data.head()
```

	<b>Series</b>	<b>X</b>	<b>Y</b>
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33

```
data.to_json('data/mi_archivojson.json')
```

## Cargar Excel

Carguemos el archivo data/datos\_rrss.xlsx en un DataFrame de pandas.

Primero revisemos el contenido del archivo Excel.

```
Para cargar los datos de un archivo Excel, se usa la función read_excel
Debemos instalar la librería openpyxl

%pip install openpyxl
```

```
Collecting openpyxl
 Downloading openpyxl-3.1.4-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et-xmlfile (from openpyxl)
 Using cached et_xmlfile-1.1.0-py3-none-any.whl.metadata (1.8 kB)
 Downloading openpyxl-3.1.4-py2.py3-none-any.whl (251 kB)
 251.4/251.4 kB 1.4 MB/s eta 0:00:00a 0:00:
?25hUsing cached et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.4
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

my_excel = pd.read_excel('data/datos_rrss.xlsx')
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Facebook	2449	True	2006
1	Twitter	339	False	2006
2	Instagram	1000	True	2010
3	YouTube	2000	False	2005
4	LinkedIn	663	False	2003

Esto por defecto carga la primera hoja del archivo Excel. Si deseas cargar una hoja específica, puedes hacerlo pasando el nombre de la hoja al argumento sheet\_name.

```
my_excel = pd.read_excel('data/datos_rrss.xlsx', sheet_name='Hoja3')
my_excel.head() # Note que este no tiene encabezado de la información
```

	<b>Facebook</b>	<b>2449</b>	<b>True</b>	<b>2006</b>
<b>0</b>	Twitter	339	False	2006
<b>1</b>	Instagram	1000	True	2010
<b>2</b>	YouTube	2000	False	2005
<b>3</b>	LinkedIn	663	False	2003
<b>4</b>	WhatsApp	1600	True	2009

```
Ajustar el encabezado
my_excel = pd.read_excel('data/datos_rrss.xlsx', sheet_name='Hoja3')
my_excel.columns = ['Nombre', 'Cantidad', 'ES_FBK', 'Año'] # De esta forma asignamos los nombres
my_excel.head()
```

	<b>Nombre</b>	<b>Cantidad</b>	<b>ES_FBK</b>	<b>Año</b>
<b>0</b>	Twitter	339	False	2006
<b>1</b>	Instagram	1000	True	2010
<b>2</b>	YouTube	2000	False	2005
<b>3</b>	LinkedIn	663	False	2003
<b>4</b>	WhatsApp	1600	True	2009

```
my_excel.to_excel('data/mi_archivo_excel.xlsx', index=False) # Guardar el archivo sin indice
```

## Cargar Directamente desde la web

Pandas también puede cargar datos directamente desde una URL. Esto es útil si los datos que deseas cargar están disponibles en la web y no necesitas descargarlos localmente.

### Leer CSV desde una URL

```
Ejemplo para cargar datos desde la web
import pandas as pd

url = 'https://raw.githubusercontent.com/datasets/investor-flow-of-funds-us/master/datasets/investor-flow-of-funds-us/master/data.csv'
data = pd.read_csv(url)
data.head()
```

	Date	Total Equity	Domestic Equity	World Equity	Hybrid	Total Bond	Taxable Bond	Municipal Bond	Total
0	2012-12-05	-7426	-6060	-1367	-74	5317	4210	1107	-2183
1	2012-12-12	-8783	-7520	-1263	123	1818	1598	219	-6842
2	2012-12-19	-5496	-5470	-26	-73	103	3472	-3369	-5466
3	2012-12-26	-4451	-4076	-375	550	2610	3333	-722	-1291
4	2013-01-02	-11156	-9622	-1533	-158	2383	2103	280	-8931

### Leer HTML desde una URL

Para este necesitamos instalar la librería lxml, que es una librería de Python que permite trabajar con archivos XML y HTML.

```
%pip install lxml
```

```
Requirement already satisfied: lxml in ./venv/lib/python3.11/site-packages (5.2.2)
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

URL de la página web que contiene la tabla
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_inflation_rate'

Leer todas las tablas en la página web
tables = pd.read_html(url)
tables
```

```
[0 1
0 NaN This article needs to be updated. Please help ..., \
 Country/Territory/Region/Group \
 Country/Territory/Region/Group
0 Aruba
1 Afghanistan
2 Angola
3 Albania
4 Andorra
...
245 ...
246 European Union
246 G20 (Group of Twenty)
247 G-77 (Group of 77)
248 OECD (Organisation for Economic Cooperation an...
249 Notes: WB: Consumer price index reflects chang...
```

```
World Bank consumer price indices (in %) \
0 2010
0 2.08
1 2.18
2 14.47
3 3.63
4 NaN
...
245 ...
245 1.53
246 NaN
247 NaN
248 1.81
249 Notes: WB: Consumer price index reflects chang...
```

```
\ \
0 2011
0 4.32
1 11.80
2 13.48
3 3.43
4 NaN
...
245 ...
245 3.29
246 NaN
247 NaN
248 3.37
249 Notes: WB: Consumer price index reflects chang...
```

```
\ \
0 2012
0 0.63
1 6.44
2 10.28
3 2.03
4 NaN
...
245 ...
245 2.66
246 NaN
247 NaN
```

```
248 2.53
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2013
```

```
1 -2.37
```

```
2 7.39
```

```
3 8.78
```

```
4 1.94
```

```
.. NaN
```

```
245 ...
```

```
246 1.22
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2014
```

```
1 0.42
```

```
2 4.67
```

```
3 7.28
```

```
4 1.63
```

```
.. NaN
```

```
245 ...
```

```
246 0.20
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2015
```

```
1 0.47
```

```
2 -0.66
```

```
3 9.35
```

```
4 1.90
```

```
.. NaN
```

```
245 ...
```

```
246 -0.06
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2016
```

```
1 -0.93
```

```
2 4.38
```

```
3 30.70
```

```
4 1.28
```

```
.. NaN
```

```
245 ...
```

```
246 0.18
```

```
247 NaN
```

```
248 0.44
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2017
```

```
1 -1.03
```

```
2 4.98
```

```
3 29.84
```

```
4 1.99
```

```
.. NaN
```

```
245 ...
```

```
246 1.43
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2018
```

```
1 3.63
```

```
2 0.63
```

```
3 19.63
```

```
4 2.03
```

```
.. NaN
```

```
245 ...
```

```
246 1.74
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2019
```

```
1 4.26
```

```
2 2.30
```

```
3 17.08
```

```
4 1.41
```

```
.. NaN
```

```
245 ...
```

```
246 1.63
```

```
247 NaN
```

```
248 NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0 2020
```

```
1 NaN
```

```
2 NaN
```

```
3 22.27
```

```
4 1.62
```

```
.. NaN
```

```
245 ...
```

```
246 0.48
```

```
247 NaN
```



248

NaN

249 Notes: WB: Consumer price index reflects chang...

[250 rows x 16 columns],

	Country/Territory/Region/Group	\
	Country/Territory/Region/Group	
	Country/Territory/Region/Group	

0

Aruba

1

Afghanistan

2

Angola

3

Albania

4

Andorra

..

...

245

European Union

246

G20 (Group of Twenty)

247

G-77 (Group of 77)

248 OECD (Organisation for Economic Cooperation an...

249 Notes: WB: Consumer price index reflects chang...

	United Nations consumer price indices	\
	2010	

Unnamed: 1\_level\_2

0

2.08

1

NaN

2

2.15

3

3.61

4

14.48

..

...

245

1.69

246

2.54

247

5.15

248

1.74

249 Notes: WB: Consumer price index reflects chang...

\

2011

Unnamed: 2\_level\_2

0

4.38

1

12.17

2

4.70

3

3.44

4

13.48

..

...

245

2.74

246

3.75

247

6.87

248

2.79

249 Notes: WB: Consumer price index reflects chang...

\

2012

Unnamed: 3\_level\_2

0

0.57

1

-28.64

2

1.43

3

2.04

```

4 10.28
..
245 ...
246 2.53
246 2.77
247 5.45
248 2.13
249 Notes: WB: Consumer price index reflects chang...

```

\

	2013
0	Unnamed: 4_level_2 -2.37
1	-9.90
2	0.19
3	1.93
4	8.78
..	...
245	1.37
246	2.40
247	5.15
248	1.53

249 Notes: WB: Consumer price index reflects chang...

\

	2014
0	Unnamed: 5_level_2 0.42
1	4.67
2	-0.28
3	1.61
4	7.30
..	...
245	0.44
246	2.38
247	4.56
248	1.59

249 Notes: WB: Consumer price index reflects chang...

\

	2015
0	Unnamed: 6_level_2 0.47
1	-1.55
2	-0.94
3	1.87
4	9.16
..	...
245	0.17
246	1.72
247	4.08
248	0.53

249 Notes: WB: Consumer price index reflects chang...

\

	2016
0	Unnamed: 7_level_2

```

0 -0.93
1 2.12
2 -0.57
3 1.29
4 30.69
..
245 ...
246 0.22
247 2.08
247 5.03
248 1.00
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2017
Unnamed: 8_level_2
0 -1.03
1 -10.57
2 1.34
3 1.98
4 29.84
..
245 ...
246 1.56
246 2.34
247 3.98
248 2.07
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2018
Unnamed: 9_level_2
0 3.63
1 0.63
2 0.38
3 2.03
4 19.63
..
245 ...
246 1.81
246 2.72
247 4.59
248 2.37
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2019
Unnamed: 10_level_2
0 3.94
1 2.30
2 0.75
3 1.41
4 17.08
..
245 ...
246 1.36
246 2.68
247 5.23
248 1.80
249 Notes: WB: Consumer price index reflects chang...

```

```
_2020
 Unnamed: 11_level_2
0 -1.31
1 5.60
2 -0.47
3 1.62
4 22.28
..
245 ...
246 0.53
247 2.06
247 5.63
248 1.10
```

249 Notes: WB: Consumer price index reflects chang...

```
_2021
 Unnamed: 12_level_2
0 0.73
1 5.13
2 1.78
3 2.04
4 25.77
..
245 ...
246 2.77
246 3.67
247 6.28
248 3.65
```

249 Notes: WB: Consumer price index reflects chang...

```
_2022
 Unnamed: 13_level_2
0 5.59
1 13.71
2 3.04
3 6.73
4 21.36
..
245 ...
246 8.89
246 7.97
247 7.72
248 8.94
```

249 Notes: WB: Consumer price index reflects chang...

```
[250 rows x 14 columns],
 vteLists of countries by financial rankings _
0 Trade
1 Investment
2 Funds
3 Budget and debt
4 Income and taxes
5 Bank rates
6 Currency
7 Other
```

## 8 Lists of countries by GDP rankings List of int...

```
vteLists of countries by financial rankings.1
0 Account balance % of GDP Exports by product me...
1 FDI received past FDI abroad GFI
2 Forex reserves Gold reserves Sovereign wealth ...
3 Government budget PPP % of GDP per capita Cred...
4 Tax rates Inheritance tax Tax revenue Wage ave...
5 Central bank interest rate Commercial bank pri...
6 Exchange rates to US$ Inflation rate
7 Financial Development Index Average annual lab...
8 Lists of countries by GDP rankings List of int...]
```

```
Imprimir el número total de tablas que se traen de la página web
print(f'Número total de tablas: {len(tables)}')
```

Número total de tablas: 4

```
Revisemos las tablas que tenemos
tables[0].head()
```

	0	1
0	NaN	This article needs to be updated. Please help ...

```
tables[1].head()
```

	<b>Country/Territory/Region/Group</b>	<b>World Bank consumer price indices (in %)</b>							
	<b>Country/Territory/Region/Group</b>	<b>2010</b>	<b>2011</b>	<b>2012</b>	<b>2013</b>	<b>2014</b>	<b>2015</b>	<b>2016</b>	
<b>0</b>	Aruba	2.08	4.32	0.63	-2.37	0.42	0.47	-0.93	
<b>1</b>	Afghanistan	2.18	11.80	6.44	7.39	4.67	-0.66	4.38	
<b>2</b>	Angola	14.47	13.48	10.28	8.78	7.28	9.35	30.70	
<b>3</b>	Albania	3.63	3.43	2.03	1.94	1.63	1.90	1.28	
<b>4</b>	Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
tables[2].head()
```

	<b>Country/Territory/Region/Group</b>	<b>United Nations consumer price indices</b>			
	<b>Country/Territory/Region/Group</b>	<b>2010</b>	<b>2011</b>	<b>2012</b>	<b>2013</b>
	<b>Country/Territory/Region/Group</b>	<b>Unnamed: 1_level_2</b>	<b>Unnamed: 2_level_2</b>	<b>Unnamed: 3_level_2</b>	<b>Unnamed: 4_level_2</b>
<b>0</b>	Aruba	2.08	4.38	0.57	-2.37
<b>1</b>	Afghanistan	NaN	12.17	-28.64	-9.90
<b>2</b>	Angola	2.15	4.70	1.43	0.19
<b>3</b>	Albania	3.61	3.44	2.04	1.93
<b>4</b>	Andorra	14.48	13.48	10.28	8.78

```
tables[3].head()
```

	vteLists of countries by financial rankings	vteLists of countries by financial rankings.1
0	Trade	Account balance % of GDP Exports by product me...
1	Investment	FDI received past FDI abroad GFI
2	Funds	Forex reserves Gold reserves Sovereign wealth ...
3	Budget and debt	Government budget PPP % of GDP per capita Cred...
4	Income and taxes	Tax rates Inheritance tax Tax revenue Wage ave...

Vamos a leer otras tablas a partir de HTML

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota')
print(f'Total de tablas obtenidas: {len(table_MN)})
```

Total de tablas obtenidas: 28

Con 38 tablas, puede ser un desafío encontrar la que necesitas. Para facilitar la selección de la tabla, utiliza el parámetro match para seleccionar un subconjunto de tablas. Podemos usar el título "Resultados de las elecciones en carreras estatales" para seleccionar la tabla:

```
%pip install html5lib
```

```
Requirement already satisfied: html5lib in ./venv/lib/python3.11/site-packages (1.1)
Requirement already satisfied: six>=1.9 in ./venv/lib/python3.11/site-packages (from
Requirement already satisfied: webencodings in ./venv/lib/python3.11/site-packages (f
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota', match='presidential')
len(table_MN)
```

1

```
df = table_MN[0]
df.head()
```

	<b>Year</b>	<b>Republican</b>		<b>Democratic</b>		<b>Third party</b>	
	<b>Year</b>	<b>No.</b>	<b>%</b>	<b>No.</b>	<b>%</b>	<b>No.</b>	<b>%</b>
<b>0</b>	2020	1484065	45.28%	1717077	52.40%	76029	2.32%
<b>1</b>	2016	1323232	44.93%	1367825	46.44%	254176	8.63%
<b>2</b>	2012	1320225	44.96%	1546167	52.65%	70169	2.39%
<b>3</b>	2008	1275409	43.82%	1573354	54.06%	61606	2.12%
<b>4</b>	2004	1346695	47.61%	1445014	51.09%	36678	1.30%

```
df.to_html('data/mi_tabla_descargada.html')
df.to_json('data/mi_tabla_descargada.json')
df.to_csv('data/mi_tabla_descargada.csv')
df.to_excel('data/mi_tabla_descargada.xlsx')
```

## Cargar datos de una base de datos SQL (Mid-level)

Pandas permite cargar datos directamente desde una base de datos SQL o NoSQL, como SQLite, MySQL, PostgreSQL, MongoDB, entre otros. Para ello, necesitas instalar los controladores de la base de datos correspondiente, como sqlalchemy, psycopg2, pymysql, pymongo, entre otros.

## SQL

```
%pip install sqlalchemy
```

```
Collecting sqlalchemy
```

```
 Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.6 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in ./venv/lib/python3.11/site
 Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl (2.1 MB)

```

```
 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 5.4 MB/s eta 0:00:00a 0:00:010m
```

```
?25hInstalling collected packages: sqlalchemy
```

```
Successfully installed sqlalchemy-2.0.31
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

from sqlalchemy import create_engine # Este es el método para crear la conexión con la base de datos

Crear la conexión con la base de datos
engine = create_engine('sqlite:///data/mibase.db') # Para otros motores debes cambiar la string de conexión de la base de datos

Leer los datos de una tabla específica. En este caso, la tabla rock_songs que es una tabla de la base de datos
data = pd.read_sql('SELECT * FROM rock_songs', engine)
data.head()
```

	<b>Song</b>	<b>Artist</b>	<b>Release_Year</b>	<b>PlayCount</b>
0	Caught Up in You	.38 Special	1982.0	82
1	Hold On Loosely	.38 Special	1981.0	85
2	Rockin' Into the Night	.38 Special	1980.0	18
3	Art For Arts Sake	10cc	1975.0	1
4	Kryptonite	3 Doors Down	2000.0	13

```
data = pd.read_sql_query("SELECT * FROM rock_songs WHERE Artist='AC/DC' ", engine)
data.head()
```

	<b>Song</b>	<b>Artist</b>	<b>Release_Year</b>	<b>PlayCount</b>
<b>0</b>	Back In Black	AC/DC	1980.0	97
<b>1</b>	Big Gun	AC/DC	1993.0	6
<b>2</b>	Dirty Deeds Done Dirt Cheap	AC/DC	1976.0	85
<b>3</b>	For Those About To Rock	AC/DC	1981.0	46
<b>4</b>	Hard As A Rock	AC/DC	1995.0	1

Estas son formas de traer datos desde una base de datos, depende del tamaño de los datos, o del problema, si es conveniente hacer la query y traer el resultado o traer toda la tabla y trabajar con ella.

```
Guardar el DataFrame en una tabla de la base de datos
data.to_sql('rock_song_filtered', con=engine, if_exists='replace', index=False)
```

20

[Imprimir en PDF](#)

# Manejo de ficheros

## Contenido

- Manejo de ficheros
- Ejercicio 1: Registro y Análisis de Asistencia
- Reto 2: Gestión de Inventario de Equipos
- Reto 1: Sistema de Gestión de Evaluaciones de Desempeño
- Reto 2: Sistema de Gestión de Proyectos de Infraestructura

El manejo de ficheros es una habilidad esencial en Python, especialmente cuando se trabaja con datos almacenados en archivos de texto, CSV, JSON y otros formatos. En esta clase, aprenderemos a leer y escribir archivos, así como a procesar y analizar los datos contenidos en ellos.

En diferentes aplicaciones es fundamental realizar manejo de archivos, tanto para el almacenamiento de información como para la carga de información de la aplicación general. Python ofrece un conjunto de funciones que hacen parte de la librería estándar y permiten la manipulación de archivos de texto plano. A continuación veremos el uso de las principales funciones.

### **Para abrir un archivo:**

```
filepath = 'ruta_del_archivo'
file = open('filepath/filename.txt', 'modo')
```

Como se puede observar un archivo puede abrirse en diferentes modos:



Para usar esta librería, en primer lugar crearemos un archivo desde cero, esto se puede hacer con los modos (x) o (w). **Verifique cual es la diferencia entre ambos**



```
Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 20:52:10-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPyth
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 41 [text/plain]
Saving to: 'archivo.txt'

archivo.txt 0%[] 0 --.-KB/s
archivo.txt 100%[=====] 41 --.-KB/s in 0s

2024-06-12 20:52:10 (525 KB/s) - 'archivo.txt' saved [41/41]
```

```
Abrir y leer un archivo de texto
with open("archivo.txt", "r") as archivo:
 contenido = archivo.read()
 print(contenido)
```

Hola, este es el contenido de un archivo.

```
Crear un archivo
folder='sample_data'
fileName = 'miArchivo'
fileExtension= 'txt'
filePath = folder + '/' + fileName + '.' + fileExtension
open(filePath, 'x')
```

```
<_io.TextIOWrapper name='sample_data/miArchivo.txt' mode='x' encoding='UTF-8'>
```

Podrá verificar que después de ejecutar las líneas anteriores se crea un archivo en la carpeta sample\_data. A continuación abriremos el archivo en modo (w) para escribir algo en él.

```
file = open(filePath, 'w')
```

Ahora con el archivo abierto, podemos escribir la información que se contenga en otra estructura, por ejemplo una lista.

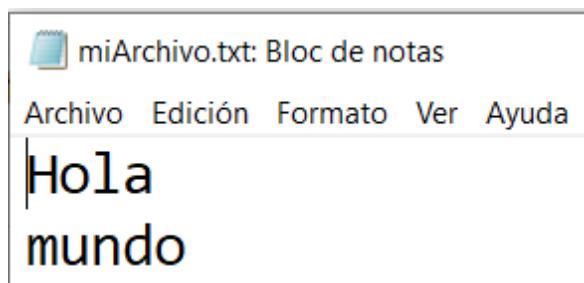
```
miLista = ["Hola","mundo"]
```

```
miLista = ['Hola','mundo']
for i in range(10):
 for palabra in miLista:
 file.write(palabra+'\t')
 file.write('\n')
```

Luego de procesar un archivo es necesario cerrarlo para que esté disponible para otras aplicaciones.

```
file.close()
```

En este momento haga el ejercicio de descargar el archivo y abrirlo desde su computador con un bloc de notas. Podrá ver algo como lo siguiente:



A partir de este archivo se pueden agregar nuevas líneas, para ello utilizaremos el modo (a), supongamos la siguiente tupla:

```
miTupla = ("Loren","ipsum","dolor","at","met")
```

```
miTupla = ('Loren','ipsum','dolor','at','met')
file = open(filePath,'a')
for palabra in miTupla:
 file.write(palabra+' ')
file.close()
```

```
with open(filePath, "w") as archivo:
 archivo.write("Hola, este es un archivo de texto.\n")
 archivo.write("Esta es una segunda línea.")
```

## Ejemplo 1:

Escribe un programa que cree un archivo de texto, escriba varias líneas, y luego lo lea e imprima su contenido.

```
with open("mi_archivo.txt", "w") as archivo:
 archivo.write("Primera línea.\n")
 archivo.write("Segunda línea.\n")
 archivo.write("Tercera línea.\n")
```

```
Leer el archivo de texto
with open("mi_archivo.txt", "r") as archivo:
 contenido = archivo.read()
 print(contenido)
```

```
Primera línea.
Segunda línea.
Tercera línea.
```

## Lectura de archivos

Ahora que ya tenemos un archivo creado y con contenido podemos abrirlo para leerlo.

```
file = open(filePath, 'r')
lista = list()
for linea in file:
 #print(type(linea))
 lista.append(linea)
 print(linea)

file.close()
print(lista)
```

## Lecto - escritura de archivos

Existen modos de lectoescritura como w+, r+ y a+

```
Lecto - escritura de archivos
f=open(filePath,"r+")
x=f.readlines()
print(x)
print(type(x))
f.write("\n Esto es otra línea")
f.close()
```

```
['Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur
<class 'list'>
```

## Leyendo archivos JSON

Estos archivos se utilizan ampliamente en APIs de sitios web. En apariencia son similares a los diccionarios de Python, pues almacenan los datos a manera de clave y valor. Al igual que en el caso de CSV la mejor alternativa para leer estos archivos es mediante Pandas. En este caso utilizaremos el archivo **anscombe.json**

```
import json

with open('sample_data/anscombe.json') as json_file:
 data = json.load(json_file)
 print(data)
```

```
[{'Series': 'I', 'X': 10.0, 'Y': 8.04}, {'Series': 'I', 'X': 8.0, 'Y': 6.95}, {'Series': 'I', 'X': 13.0, 'Y': 7.58}, {'Series': 'I', 'X': 9.0, 'Y': 8.81}, {'Series': 'I', 'X': 11.0, 'Y': 8.33}, {'Series': 'I', 'X': 14.0, 'Y': 9.96}, {'Series': 'I', 'X': 6.0, 'Y': 7.04}, {'Series': 'I', 'X': 12.0, 'Y': 9.14}, {'Series': 'I', 'X': 7.0, 'Y': 8.04}, {'Series': 'I', 'X': 10.0, 'Y': 9.14}, {'Series': 'II', 'X': 8.08, 'Y': 9.14}, {'Series': 'II', 'X': 9.14, 'Y': 8.04}, {'Series': 'II', 'X': 7.82, 'Y': 6.95}, {"Series": "III", "X": 7.91, "Y": 7.58}, {"Series": "III", "X": 8.77, "Y": 8.81}, {"Series": "III", "X": 8.33, "Y": 8.33}, {"Series": "III", "X": 9.23, "Y": 9.96}, {"Series": "III", "X": 7.5, "Y": 7.04}, {"Series": "III", "X": 10.8, "Y": 9.14}, {"Series": "III", "X": 8.81, "Y": 8.04}, {"Series": "III", "X": 9.96, "Y": 9.14}, {"Series": "III", "X": 8.33, "Y": 7.58}, {"Series": "IV", "X": 8.81, "Y": 8.04}, {"Series": "IV", "X": 8.04, "Y": 8.81}, {"Series": "IV", "X": 8.33, "Y": 9.96}, {"Series": "IV", "X": 8.77, "Y": 7.58}, {"Series": "IV", "X": 9.23, "Y": 8.33}, {"Series": "IV", "X": 7.5, "Y": 9.14}, {"Series": "IV", "X": 10.8, "Y": 8.81}, {"Series": "IV", "X": 8.33, "Y": 7.04}, {"Series": "IV", "X": 9.96, "Y": 8.33}, {"Series": "IV", "X": 8.81, "Y": 9.96}], [{"x": 10.0, "y": 8.04}, {"x": 8.0, "y": 6.95}, {"x": 13.0, "y": 7.58}, {"x": 9.0, "y": 8.81}, {"x": 11.0, "y": 8.33}, {"x": 14.0, "y": 9.96}, {"x": 6.0, "y": 7.04}, {"x": 12.0, "y": 9.14}, {"x": 7.0, "y": 8.04}, {"x": 10.0, "y": 9.14}, {"x": 8.08, "y": 9.14}, {"x": 9.14, "y": 8.04}, {"x": 7.82, "y": 6.95}, {"x": 7.91, "y": 7.58}, {"x": 8.77, "y": 8.81}, {"x": 8.33, "y": 8.33}, {"x": 9.23, "y": 9.96}, {"x": 7.5, "y": 7.04}, {"x": 10.8, "y": 9.14}, {"x": 8.81, "y": 8.04}, {"x": 9.96, "y": 9.14}, {"x": 8.33, "y": 7.58}, {"x": 8.81, "y": 8.81}, {"x": 8.04, "y": 9.96}, {"x": 8.33, "y": 7.04}, {"x": 9.23, "y": 8.33}, {"x": 7.5, "y": 9.14}, {"x": 10.8, "y": 8.81}, {"x": 8.33, "y": 7.58}, {"x": 9.96, "y": 8.33}, {"x": 8.81, "y": 9.96}]]
```

## Escribiendo archivos JSON

```
import json

Escribir en un archivo JSON
datos = {
 "nombre": "Ana",
 "edad": 30,
 "ciudad": "Madrid"
}

with open("archivo.json", "w") as archivo:
 json.dump(datos, archivo, indent=4)
```

## Escribiendo archivos csv

```
import csv

Escribir en un archivo CSV
with open("archivo.csv", "w", newline='') as archivo:
 escritor_csv = csv.writer(archivo)
 escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
 escritor_csv.writerow(["Ana", 30, "Madrid"])
 escritor_csv.writerow(["Carlos", 25, "Barcelona"])
```

## Leyendo archivos csv

```
import csv

Leer un archivo CSV
with open("archivo.csv", "r") as archivo:
 lector_csv = csv.reader(archivo)
 for fila in lector_csv:
 print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
```

## Ejemplo:

Escribe un programa que cree un archivo CSV, escriba varias filas de datos, y luego lo lea e imprima su contenido.

```

import csv

Escribir en un archivo CSV
with open("datos.csv", "w", newline='') as archivo:
 escritor_csv = csv.writer(archivo)
 escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
 escritor_csv.writerow(["Ana", 30, "Madrid"])
 escritor_csv.writerow(["Carlos", 25, "Barcelona"])
 escritor_csv.writerow(["Beatriz", 28, "Valencia"])

Leer el archivo CSV
with open("datos.csv", "r") as archivo:
 lector_csv = csv.reader(archivo)
 for fila in lector_csv:
 print(fila)

```

```

['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
['Beatriz', '28', 'Valencia']

```

## Ejemplo de análisis de datos desde un archivo CSV

Supongamos que tenemos un archivo CSV con datos de empleados y queremos calcular la edad promedio.

```

Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main

```

```
--2024-06-12 21:41:37-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 557 [text/plain]
```

```

Saving to: 'empleados.csv'

empleados.csv 0%[] 0 --.-KB/s
empleados.csv 100%[=====>] 557 --.-KB/s in 0s

2024-06-12 21:41:37 (20.5 MB/s) - 'empleados.csv' saved [557/557]
```

```

import csv

Leer datos de empleados y calcular la edad promedio
with open("empleados.csv", "r") as archivo:
 lector_csv = csv.DictReader(archivo)
 total_edad = 0
 contador = 0
 for fila in lector_csv:
 total_edad += int(fila["Edad"])
 contador += 1

 edad_promedio = total_edad / contador
 print(f"Edad promedio: {edad_promedio}")

```

Edad promedio: 30.133333333333333

## Ejemplo de análisis de datos desde un archivo JSON

Supongamos que tenemos un archivo JSON con datos de varios productos y queremos calcular el precio total de todos los productos.

```

Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main

```

```

--2024-06-12 21:42:00-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 1369 (1.3K) [text/plain]
Saving to: 'productos.json'

```

```

productos.json 0%[] 0 --.-KB/s
productos.json 100%[=====] 1.34K --.-KB/s in 0s

2024-06-12 21:42:00 (64.5 MB/s) - 'productos.json' saved [1369/1369]

```

```
import json

Leer datos de productos y calcular el precio total
with open("productos.json", "r") as archivo:
 productos = json.load(archivo)
 total_precio = sum(producto["precio"] for producto in productos)
 print(f"Precio total: {total_precio}")
```

Precio total: 370.0

Crea un programa para gestionar el registro de asistencia de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de asistencia en un archivo CSV.
2. Leer y mostrar los datos de asistencia desde el archivo.
3. Calcular y mostrar el porcentaje de asistencia de cada empleado.

```
Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:08-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main/asistencia.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.108.134
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 704 [text/plain]
Saving to: ‘asistencia.csv’
```

```
asistencia.csv 0%[] 0 --.-KB/s
asistencia.csv 100%[=====] 704 --.-KB/s in 0s

2024-06-12 21:47:08 (44.0 MB/s) - ‘asistencia.csv’ saved [704/704]
```

```
import csv

def agregar_asistencia(archivo, nombre, fecha, presente):
 with open(archivo, "a", newline='') as archivo_csv:
 escritor_csv = csv.writer(archivo_csv)
 escritor_csv.writerow([nombre, fecha, presente])
```

```
def leer_asistencia(archivo):
 with open(archivo, "r") as archivo_csv:
 lector_csv = csv.reader(archivo_csv)
 for fila in lector_csv:
 print(fila)
```

```
def calcular_asistencia(archivo):
 with open(archivo, "r") as archivo_csv:
 lector_csv = csv.reader(archivo_csv)
 empleados = {}
 for fila in lector_csv:
 nombre, _, presente = fila
 if nombre not in empleados:
 empleados[nombre] = {"asistencias": 0, "total": 0}
 empleados[nombre]["total"] += 1
 if presente == "True":
 empleados[nombre]["asistencias"] += 1

 for nombre, datos in empleados.items():
 porcentaje_asistencia = (datos["asistencias"] / datos["total"]) * 100
 print(f"{nombre}: {porcentaje_asistencia:.2f}% de asistencia")
```

```
Agregar registros de asistencia
agregar_asistencia("asistencia.csv", "Juan", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Carlos", "2023-06-01", False)
agregar_asistencia("asistencia.csv", "Juan", "2023-06-02", False)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-02", True)
```

```
Leer y mostrar los registros de asistencia
leer_asistencia("asistencia.csv")
```

```
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
```

```
Calcular y mostrar el porcentaje de asistencia de cada empleado
calcular_asistencia("asistencia.csv")
```

```
Juan: 50.00% de asistencia
Ana: 100.00% de asistencia
Carlos: 0.00% de asistencia
```

Crea un programa para gestionar el inventario de equipos en una institución pública. El programa debe permitir:

1. Guardar los datos del inventario en un archivo JSON.
2. Leer y mostrar los datos del inventario desde el archivo.
3. Calcular y mostrar el valor total del inventario.

```
Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:42-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main/inventario.json
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.108.134
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2982 (2.9K) [text/plain]
Saving to: ‘inventario.json’

inventario.json 0%[=====] 0 --.-KB/s
inventario.json 100%[=====] 2.91K --.-KB/s in 0s

2024-06-12 21:47:42 (16.1 MB/s) - ‘inventario.json’ saved [2982/2982]
```

```
import json

def agregar_equipo(archivo, equipo):
 try:
 with open(archivo, "r") as archivo_json:
 inventario = json.load(archivo_json)
 except FileNotFoundError:
 inventario = []

 inventario.append(equipo)

 with open(archivo, "w") as archivo_json:
 json.dump(inventario, archivo_json, indent=4)
```

```
def leer_inventario(archivo):
 with open(archivo, "r") as archivo_json:
 inventario = json.load(archivo_json)
 for equipo in inventario:
 print(equipo)
```

```
def calcular_valor_total(archivo):
 with open(archivo, "r") as archivo_json:
 inventario = json.load(archivo_json)
 valor_total = sum(equipo["valor"] for equipo in inventario)
 print(f"Valor total del inventario: {valor_total}")
```

```
Agregar equipos al inventario
agregar_equipo("inventario.json", {"nombre": "Computadora", "marca": "Dell", "modelo": "Inspiron", "año": 2021, "valor": 200}
agregar_equipo("inventario.json", {"nombre": "Impresora", "marca": "HP", "modelo": "LaserJet", "año": 2019, "valor": 200}
agregar_equipo("inventario.json", {"nombre": "Escáner", "marca": "Epson", "modelo": "Perfection", "año": 2020, "valor": 150}
```

```
Leer y mostrar los datos del inventario
leer_inventario("inventario.json")
```

```
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor': 200}
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 150}
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor': 200}
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 150}
```

```
calcular_valor_total("inventario.json")
```

Valor total del inventario: 2300

Crea un programa que gestione las evaluaciones de desempeño de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de las evaluaciones en un archivo JSON.
2. Leer y mostrar los datos de las evaluaciones desde el archivo.
3. Calcular y mostrar el puntaje promedio de desempeño por departamento.

### Requisitos:

- Cada evaluación debe contener el nombre del empleado, departamento, fecha de evaluación y puntaje.
- El programa debe ser capaz de agregar nuevas evaluaciones, actualizar puntajes de evaluaciones existentes y eliminar evaluaciones.
- Debe calcular el puntaje promedio de desempeño por departamento y mostrar un resumen de todos los empleados y sus evaluaciones.

```
Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
Tu código va acá
```

Crea un programa que gestione los proyectos de infraestructura en una institución pública. El programa debe permitir:

1. Guardar los datos de los proyectos en un archivo CSV.
2. Leer y mostrar los datos de los proyectos desde el archivo.
3. Calcular y mostrar el presupuesto total de los proyectos por estado (planificado, en curso, completado).

### Requisitos:

- Cada proyecto debe contener el nombre del proyecto, descripción, fecha de inicio, fecha de fin, estado y presupuesto.
- El programa debe ser capaz de agregar nuevos proyectos, actualizar el estado y presupuesto de proyectos existentes y eliminar proyectos.
- Debe calcular el presupuesto total por estado y mostrar un resumen de todos los proyectos.

```
Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
Tu código va acá
```

# Matrices y vectores

## Contenido

- Matrices y vectores
- Numpy
- Generación de matrices y vectores
- Operaciones vectorizadas
- Más operaciones vectorizadas
- Expresiones compactas / Comprehensions
- Matplotlib
- Imágenes

### Buscar un elemento en un arreglo

Dado un arreglo con componentes  $A[1], \dots, A[n]$  y un valor  $x$ , el siguiente algoritmo retorna en la variable  $q$  el valor *false* si no existe  $k$  tal que  $A[k] = x$ ; en caso contrario, retorna en la variable  $q$  el valor *true* y en la variable  $i$  el valor  $k$ .

$\{n > 0\}$   
 $i : 0..n; q : boolean;$   
 $A : array [1..n] of T;$   
 $\{\text{asignarle valores a las componentes de } A\}$

Paso 1:  $i := 0;$   
Paso 2: **repetir**  
           $i := i + 1;$   
           $q := A[i] = x$   
          **hasta que**  $q \vee (i = n)$

```
import numpy as np
```

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
q=True
n=len(A)
while q and (i!=n) :
 q=not(A[i]==x)
 i=i+1
if i==n and q==True:
 print('No se encontró el elemento')
else:
 print('la posición es:',i-1,'El valor es:',A[i-1])
```

la posición es: 4 El valor es: e

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
n=len(A)
while i!=n:
 if A[i]==x:
 break
 i=i+1
if i==n:
 print('No se encontró el elemento')
else:
 print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 4 El valor es: e

## Otra forma de buscar un elemento en un arreglo: (centinela)

Permite simplificar la condición de terminación del anterior algoritmo.

- Se le adiciona una componente al arreglo  $A$ .
- A la nueva componente se le asigna el valor  $x$ , que actua como **centinela** para la finalización de la búsqueda.

$$\{c = n + 1, n > 0\}$$

$$i : 0..c;$$

$$A : \text{array}[1..c] \text{ of } T;$$

Paso 1:  $i := 0;$   
 $A[c] := x;$

Paso 2: **repetir**  
 $i := i + 1;$   
**hasta que**  $A[i] = x;$

```
A=np.array(['a','b','c','d','e'])
x='c'
n=len(A)
A=np.append(A,x)

i=0
while not(A[i]==x):
 i=i+1
if i==n:
 print('No se encontró el elemento')
else:
 print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 2 El valor es: c

## Buscar un elemento en un arreglo ordenado

En este caso las componentes del arreglo están ordenadas, por ejemplo en orden creciente, es decir  $A[i] < A[j]$  para todo  $i < j$ .

El siguiente algoritmo tiene en cuenta que el arreglo está ordenado.

```

i, j, k : integer; q : boolean;
{n > 0}
A : array [1..n] of T;
Paso 1: i := 1; j := n; q := false
Paso 2: repetir
 k := (i + j) div 2;
 si A[k] = x entonces q := true sino
 si A[k] < x entonces i = k + 1 sino j := k - 1;
hasta que q $\vee (i > j)$
```

## Busqueda binaria

```

import numpy as np
A=np.array([2,3,5,6,8,10,11])
x=11
i=0
j=len(A)
q=True
while q and (i<=j):
 k=(i+j)//2
 if A[k]==x:
 q=False
 else:
 if A[k]<x:
 i=k+1
 else:
 j=k-1
if q==True:
 print('No se encontró el elemento')
else:
 print('la posición es:',k,'El valor es:',A[k])
```

la posición es: 6 El valor es: 11

## Producto escalar o producto interno

Dados las sucesiones de números  $(x_1, \dots, x_n)$  y  $(y_1, \dots, y_n)$  calcular el producto escalar

$$s = x_1 * y_1 + x_2 * y_2 + \dots + x_n * y_n = \sum_{i=1}^n x_i * y_i$$

Teniendo en cuenta la definición de esta suma en términos de la relación de recurrencia:

$$s_i = s_{i-1} + x_i * y_i, \quad s_0 = 0$$

se tiene el siguiente programa,

```
s : real; i : integer;
x, y : array [1..n] of real;
Paso 1: s := 0; i := 0;
Paso 2: repetir
 i := i + 1;
 s := s + x[i] * y[i];
 hasta que i = n;
```



```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
while i!=n:
 s=s+v1[i]*v2[i]
 i+=1
print('el producto escalar es: ',s)
```

el producto escalar es: 60

## Sentencia for

Otro algoritmo para hallar el producto escalar de dos sucesiones de números  $(x_1, \dots, x_n)$  y  $(y_1, \dots, y_n)$  es:

```
s : real; i : 1..n;
x, y : array [1..n] of real;
s := 0;
para i := 1 hasta n hacer
 s := s + x[i] * y[i];
```

```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
for i in range(n):
 s=s+v1[i]*v2[i]
print('el producto escalar es: ',s)
```

```
el producto escalar es: 60
```

## Encontrar el elemento máximo

Encontrar en un arreglo  $A$  el índice  $j$  tal que  $x_j = \max(x_m, \dots, x_n)$ .

$j, k : m..n;$   
 $x : \text{array } [m..n] \text{ of } T;$   
Paso 1:  $j := m;$   
Paso 2: **para**  $k := m + 1$  **hasta**  $n$  **hacer**  
          **si**  $x[k] > x[j]$  **entonces**  $j := k$

```
A=[1,2,4,5,3,6,7]
j=0
k=0
for k in range(1,len(A)):
 if A[k]>A[j]:
 j=k
print('el elemento maximo es:',A[j], 'en la posición: ',j)
```

```
el elemento maximo es: 7 en la posición: 6
```

## Sentencias for anidadas

### Ejemplo:

Dado un entero positivo  $n$ , calcular la suma  $1^1 + 2^2 + \dots + n^n$ .

```

x, i : 1..n;
s, potencia : integer;
s := 0;
para x := 1 hasta n hacer
 potencia := 1;
 para i := 1 hasta x hacer
 potencia = potencia * x;
 s := s + potencia;

```

```

n=3
s=0
for x in range(1,n+1):
 potencia=1
 for i in range(1,x+1):
 potencia=potencia*x
 s=s+potencia
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

```

n=3
s=0
for x in range(1,n+1):
 s=s+x**x
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

## Algoritmo para el ordenamiento de un arreglo

```

 $h, j, k : 1..n;$
 $x : \text{array } [1..n] \text{ of } T;$
 $u : T;$
para $h := 1$ hasta $n - 1$ hacer
 $j := h;$
 para $k := h + 1$ hasta n hacer
 si $x[k] > x[j]$ entonces $j := k$
 $u := x[h];$
 $x[h] := x[j]$
 $x[j] := u$

```

### Complejidad del algoritmo:

$$(n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n}{2}(n - 1).$$

```

x=np.random.randint(200, size=100)
h=0
for h in range(len(x)):
 j=h
 for k in range(h+1,len(x)):
 if x[k]>x[j]:
 j=k
 u=x[h]
 x[h]=x[j]
 x[j]=u
print(x)

```

198	194	192	187	181	180	179	179	178	177	175	175	172	172	171	168	166	164	163
161	161	160	158	156	154	152	147	146	146	146	146	142	140	135	135	130	128	126
124	120	119	118	118	116	112	110	107	102	101	101	100	100	95	89	88	86	
84	83	82	82	82	81	81	80	77	76	73	73	72	71	69	68	67	66	
61	61	57	55	49	45	42	41	41	40	39	36	35	35	29	23	22	21	
18	17	15	13	12	12	10	8	6	0									

Con la librería `numpy` se trabaja con matrices de forma natural. Fíjate cómo se declaran y cómo descubrimos sus dimensiones.

[https://s3.amazonaws.com/assets.datacamp.com/blog\\_assets/Numpy\\_Python\\_Cheat\\_Sheet.pdf](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)

```
import numpy as np

a = np.array([[1,2,3,4,5],
 [5,4,3,2,1],
 [9,8,7,6,5],
 [7,6,5,6,7],
 [2,2,2,3,3],
 [4,3,4,3,4],
 [5,1,1,4,1]]).astype('int32')

print(a, type(a), a nbytes, "\n")
print("a shape", a.shape, "\n")
print("a rows", a.shape[0], "\n")
print("a cols", a.shape[1])
```

```
[[1 2 3 4 5]
 [5 4 3 2 1]
 [9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]
 [4 3 4 3 4]
 [5 1 1 4 1]] <class 'numpy.ndarray'> 140

a shape (7, 5)

a rows 7

a cols 5
```

```
a[0,0]=100
a
```

```
array([[100, 2, 3, 4, 5],
 [5, 4, 3, 2, 1],
 [9, 8, 7, 6, 5],
 [7, 6, 5, 6, 7],
 [2, 2, 2, 3, 3],
 [4, 3, 4, 3, 4],
 [5, 1, 1, 4, 1]], dtype=int32)
```

```
a nbytes
```

140

```
v = np.array([2,3,4,5,6,7,3,12])
print("v shape", v.shape)
print("v elems", v.shape[0])
print(v, type(v))
```

```
v shape (8,)
v elems 8
[2 3 4 5 6 7 3 12] <class 'numpy.ndarray'>
```

```
v[3:-1]
```

```
array([5, 6, 7, 3])
```

```
np.min(v), np.max(v)
```

```
(2, 12)
```

Con la notación de índices accedemos a columnas o filas enteras, rangos de columnas o filas, elementos individuales o porciones de una matriz o un vector.

```
print("una fila ", a[2])
print("una fila ", a[2,:])
print("una columna ", a[:,2])
print("un elemento ", a[2,2])
print("varias filas \n", a[2:5])
print("varias columnas\n", a[:,1:3])
print("una porcion \n", a[2:5,1:3])
```

```

una fila [9 8 7 6 5]
una fila [9 8 7 6 5]
una columna [3 3 7 5 2 4 1]
un elemento 7
varias filas
[[9 8 7 6 5]
[7 6 5 6 7]
[2 2 2 3 3]]
varias columnas
[[2 3]
[4 3]
[8 7]
[6 5]
[2 2]
[3 4]
[1 1]]
una porcion
[[8 7]
[6 5]
[2 2]]

```

Muchas funciones de la librería `numpy` operan sobre una matriz completa, o de forma separada por columnas o filas según el valor del argumento `axis`.

```

print(a)
print("suma total", np.sum(a))
print("suma eje 0", np.sum(a, axis=0))
print("suma eje 1", np.sum(a, axis=1))
print("promedio total", np.mean(a))
print("promedio eje 0", np.mean(a, axis=0))
print("promedio eje 1", np.mean(a, axis=1))

```

```

[[100 2 3 4 5]
 [5 4 3 2 1]
 [9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]
 [4 3 4 3 4]
 [5 1 1 4 1]]
suma total 237
suma eje 0 [132 26 25 28 26]
suma eje 1 [114 15 35 31 12 18 12]
promedio total 6.771428571428571
promedio eje 0 [18.85714286 3.71428571 3.57142857 4. 3.71428571]
promedio eje 1 [22.8 3. 7. 6.2 2.4 3.6 2.4]

```

Las matrices en Python pueden tener un número arbitrario de dimensiones y podemos acceder a submatrices en la dirección o dimensión que queramos.

```
z = np.random.randint(-20,20, size=(5,5))
print(z)
```

```
[[-19 4 -13 -19 13]
 [19 6 -9 -9 -16]
 [-11 12 -6 -20 16]
 [-4 -1 -4 13 0]
 [-19 13 -1 9 8]]
```

```
np.sum(z, axis=1)
```

```
array([-34, -9, -9, 4, 10])
```

```
#m = np.random.randint(10, size=(3,3,3))
print("Matrix 3D completa\n", m)
print("-----\n", m[0,:,:])
print("-----\n", m[1,:,:])
print("-----\n", m[:,0,:])
print("-----\n", m[:,0,1])
print("-----\n", np.mean(m, axis=1))
```

```
Matrix 3D completa
```

```
[[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```
[[8 6 9]
 [0 3 0]
 [0 9 2]]]
```

```

```

```
[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```

```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```

```

```
[[7 0 8]
 [4 7 9]
 [8 6 9]]]
```

```

```

```
[0 7 6]
```

```

```

```
[[4.66666667 1.33333333 7.
 [2. 4.66666667 8.
 [2.66666667 6. 3.66666667]]]
```

`m.shape`

(3, 3, 3)

`m`

```
print("matrix identidad\n", np.eye(3))
print("vector de ceros", np.zeros(4))
print("matriz de ceros\n", np.zeros((3,2)))
print("matriz de unos\n", np.ones((2,3)))
print("vector rango", np.arange(10))
print("vector rango", np.arange(5,10))
print("vector espacio lineal", np.linspace(2,12,11))
print("matriz aleatoria según distribución uniforme [0,1]\n", np.random.random(size=(3
print("vector aleatorio de enteros entre 0 y 5", np.random.randint(5, size=10))
```

```

matrix identidad
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
vector de ceros [0. 0. 0. 0.]
matriz de ceros
[[0. 0.]
 [0. 0.]
 [0. 0.]]
matriz de unos
[[1. 1. 1.]
 [1. 1. 1.]]
vector rango [0 1 2 3 4 5 6 7 8 9]
vector rango [5 6 7 8 9]
vector espacio lineal [2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.]
matriz aleatoria según distribución uniforme [0,1]
[[0.57996022 0.65361661 0.99420518 0.12111156 0.26731741]
 [0.206147 0.25575192 0.66588831 0.55022348 0.76820253]
 [0.51384056 0.63172996 0.50836505 0.41256526 0.66835632]]
vector aleatorio de enteros entre 0 y 5 [0 3 3 0 3 3 2 1 2 1]

```

```

import numpy as np
v = np.array([10,12,13,15,20])
print(v)
print(v+1)
print(v*2)

a = np.random.randint(100, size=5)
print(a)

print(v.dot(a))

```

```

[10 12 13 15 20]
[11 13 14 16 21]
[20 24 26 30 40]
[60 83 36 31 82]
4169

```

```

v1=np.array([2,-1,1])
v2=np.array([-3,1,1])
print(v1.dot(v2))
print(v1*v2)
print(np.cross(v2,v1))
print(np.cross(v1,v2).dot(v1))
print(np.cross(v1,v2).dot(v2))

```

```
-6
[-6 -1 1]
[2 5 1]
0
0
```

```
m1=np.array([[1,2],[3,4]])
```

```
m2=np.array([[4,8],[9,1]])
```

```
m1*m2
```

```
array([[4, 16],
 [27, 4]])
```

```
m1.dot(m2)
```

```
array([[22, 10],
 [48, 28]])
```

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[6,5,4],[3,2,1]])
c = np.array([[1,2],[4,5]])
print(a)
print("--")
print(a.T)
print("--")
print(b)
print("--")
print(a.T.dot(a))
print("--")
print(a*b)
print("--")
x=np.array([[2,3],[4,5]])
z=np.array([[5,6,7],[2,3,1]])
print(x.dot(z))
```

```
[[1 2 3]
 [4 5 6]]
 --
 [[1 4]
 [2 5]
 [3 6]]
 --
 [[6 5 4]
 [3 2 1]]
 --
 [[17 22 27]
 [22 29 36]
 [27 36 45]]
 --
 [[6 10 12]
 [12 10 6]]
 --
 [[16 21 17]
 [30 39 33]]
```

Las operaciones vectorizadas también funcionan con expresiones *booleanas*. Fíjate cómo se indexa un vector con una expresión booleana para seleccionar un conjunto de elementos.

```
a = np.array([1,8,4,10,-4,5])
print("posiciones en a >4:", a>4)
print("elementos de a >4:",a[a>4])
```

```
posiciones en a >4: [False True False True False True]
elementos de a >4: [8 10 5]
```

```
a = np.random.randint(100, size=(6,10))
a
```

```
array([[14, 19, 5, 87, 99, 92, 31, 63, 18, 44],
 [81, 26, 65, 86, 2, 24, 99, 79, 71, 37],
 [42, 87, 8, 92, 63, 29, 31, 85, 75, 93],
 [59, 58, 97, 10, 12, 99, 86, 99, 28, 56],
 [76, 76, 75, 56, 49, 33, 51, 58, 41, 4],
 [35, 55, 41, 7, 42, 85, 53, 14, 22, 51]])
```

```
np.mean(a, axis=1)
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
%timeit np.mean(a, axis=1)
```

```
10.8 µs ± 2.4 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
43.7 µs ± 925 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Pregúntate siempre si puedes vectorizar algo y hazlo. Por ejemplo, Para dos matrices (**grandes**).

```
a = np.random.randint(100, size=(1000,100))
b = np.random.randint(200, size=(1000,100))
```

```
El número de elementos que son iguales
np.mean(a==b)
```

```
0.00512
```

```
El promedio de los elementos de a que son mayores a su correspondiente posición en b
np.mean(a[a>b])
```

```
66.36604323536555
```

```
El promedio de los elementos de b que son mayores a su correspondiente posición en a
np.mean(b[b>a])
```

```
122.37378653758725
```

```
Con matrices más pequeñas
a = np.random.randint(100, size=(10))
b = np.random.randint(200, size=(10))
print (a)
print (b)
```

```
[73 19 71 47 95 38 48 59 74 61]
[68 94 69 142 167 40 190 70 33 95]
```

```
el elemento en b correspondiente a la posición del elemento más grande en a
b[np.argmax(a)]
```

167

## Broadcasting

Normalmente, Numpy necesita que las dimensiones de las matrices coincidan cuando se hacen operaciones con ellas

```
a = np.random.randint(100, size=(3,5))
b = np.random.randint(10, size=(3,4))
print (a)
print (b)
a + b
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86 0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```

ValueError Traceback (most recent call last)
<ipython-input-14-860cc8167430> in <cell line: 5>()
 3 print (a)
 4 print (b)
----> 5 a + b

```

`ValueError: operands could not be broadcast together with shapes (3,5) (3,4)`

Pero numoy intentan expandir las operaciones si alguna de las dimensiones coincide

a

```
array([[83, 58, 25, 73, 33],
 [91, 73, 70, 60, 18],
 [20, 86, 0, 23, 62]])
```

a\*10

```
array([[830, 580, 250, 730, 330],
 [910, 730, 700, 600, 180],
 [200, 860, 0, 230, 620]])
```

# Veamos como funciona el reshape en la siguiente operación.  
`a + b[:,1].reshape(-1,1)`

```
array([[85, 60, 27, 75, 35],
 [97, 79, 76, 66, 24],
 [29, 95, 9, 32, 71]])
```

`b[:,1].reshape(-1,1)`

```
array([[2],
 [6],
 [9]])
```

`b[:,1]`

```
array([2, 6, 9])
```

```
a + b[:,1]
```

```

ValueError Traceback (most recent call last)
<ipython-input-20-d148e74cc6fe> in <cell line: 1>()
----> 1 a + b[:,1]
```

**ValueError:** operands could not be broadcast together with shapes (3,5) (3,)

```
Observa la forma de las filas
a + b.flatten()[:a.shape[1]]
```

```
array([[90, 60, 29, 73, 36],
 [98, 75, 74, 60, 21],
 [27, 88, 4, 23, 65]])
```

```
print (a)
print (b)
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86 0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```
b.flatten()
```

```
array([7, 2, 4, 0, 3, 6, 7, 1, 7, 9, 5, 2])
```

```
b.flatten()[:a.shape[1]]
```

```
array([7, 2, 4, 0, 3])
```

Fíjate cómo las siguientes expresiones son equivalentes:

```
a=15
if a > 10:
 s = "mayor que 10"
else:
 s = "menor que 10"

print(s)
```

mayor que 10

```
a = 15
s = "mayor que 10" if a > 10 else "menor que 10"
print(s)
```

mayor que 10

```
l=[]
for i in range(5):
 l.append(i)
print(l)
```

[0, 1, 2, 3, 4]

```
l=[i for i in range(5)]
print(l)
```

[0, 1, 2, 3, 4]

```
a = [10, -4, 20, 5]

#o = ["10A", "-4B", "20A", "5A"]

o = []
for i in a:
 if i<0:
 o.append(str(i)+"B")
 else:
 o.append(str(i)+"A")

print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
a = [10, -4, 20, 5]
def convert(x):
 return str(x)+"B" if x<0 else str(x)+"A"

o = [convert(i) for i in a]
print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
r = []
for i in range(10):
 r.append("el numero "+str(i))
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
r = ["el numero "+str(i) for i in range(10)]
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla', 'Guayaba':'rosa'}
for nombre, color in frutas.items():
 print (nombre, "es de color", color)

print()
r = [nombre+" es de color "+color for nombre, color in frutas.items()]
r
```

```
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
```

```
['Fresa es de color roja',
 'Limon es de color verde',
 'Papaya es de color naranja',
 'Manzana es de color amarilla',
 'Guayaba es de color rosa']
```

## Ejercicio

Construir una matriz aleatoria de cuatro letras (A,T,C,G) de tamaño 100x1000 luego cambiar las letras por las siguientes codificaciones:

Codificación 1: -2,-1,1,2

Codificación 2: 0,1,2,3

Codificación 3: 0001,0010,0100,1000

```
m=np.array([['A', 'T'], ['C', 'G']])
```

m

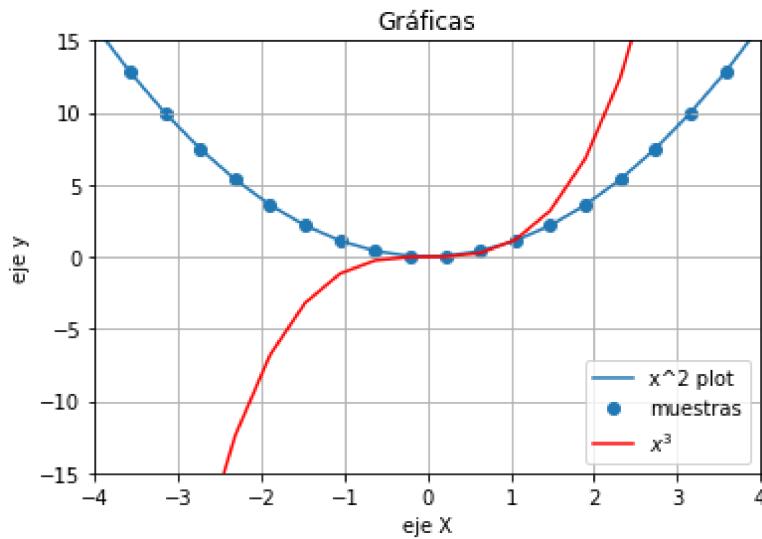
```
array([['A', 'T'],
 ['C', 'G']], dtype='<U1')
```

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

x = np.linspace(-4,4,20)# El vector de tabulación
y=x**2
z=x**3

plt.plot(x, y, label="x^2 plot")
plt.scatter(x, y, label="muestras")
plt.plot(x, z, label="x^3", color="red")
plt.xlim([-4,4])
plt.ylim([-15, 15])
plt.legend()
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Gráficas')
```

Text(0.5, 1.0, 'Gráficas')

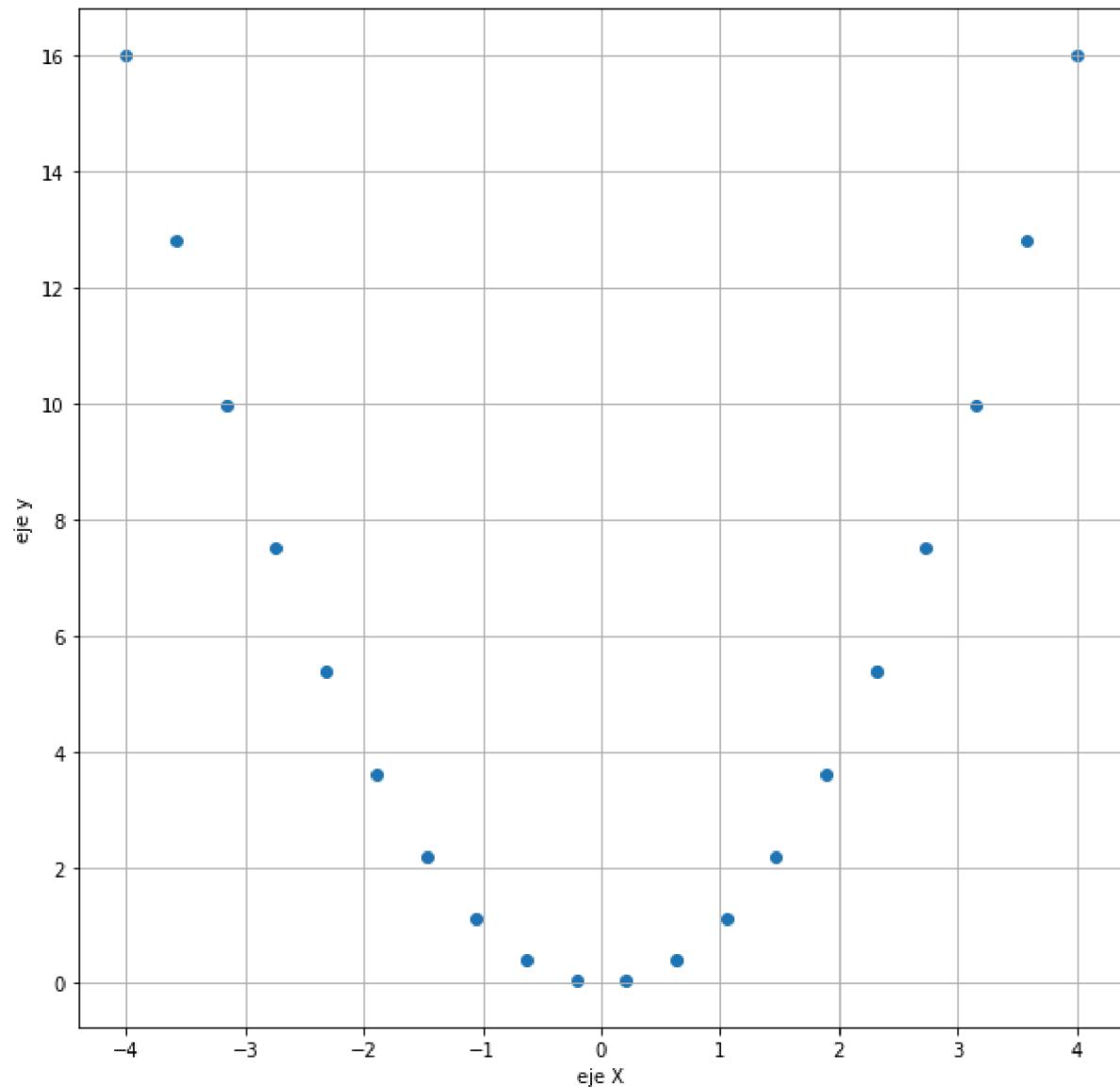


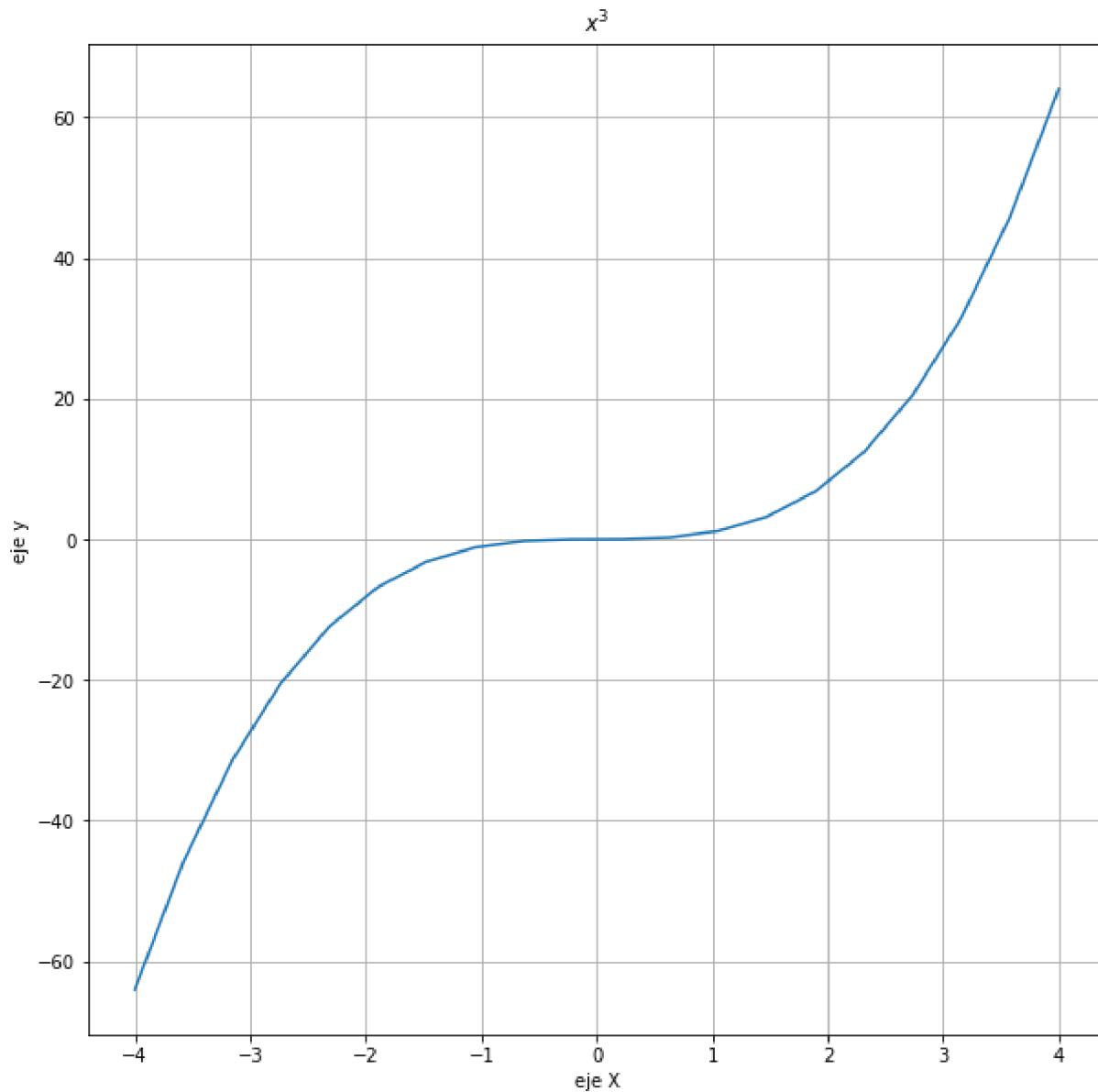
```
x = np.linspace(-4,4,20)
y=x**2

plt.figure(figsize=(10,10))
plt.scatter(x, y)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Parabola')

plt.figure(figsize=(10,10))
plt.plot(x, x**3)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('x^3')
```

Text(0.5, 1.0, '\$x^3\$')

**Parabola**

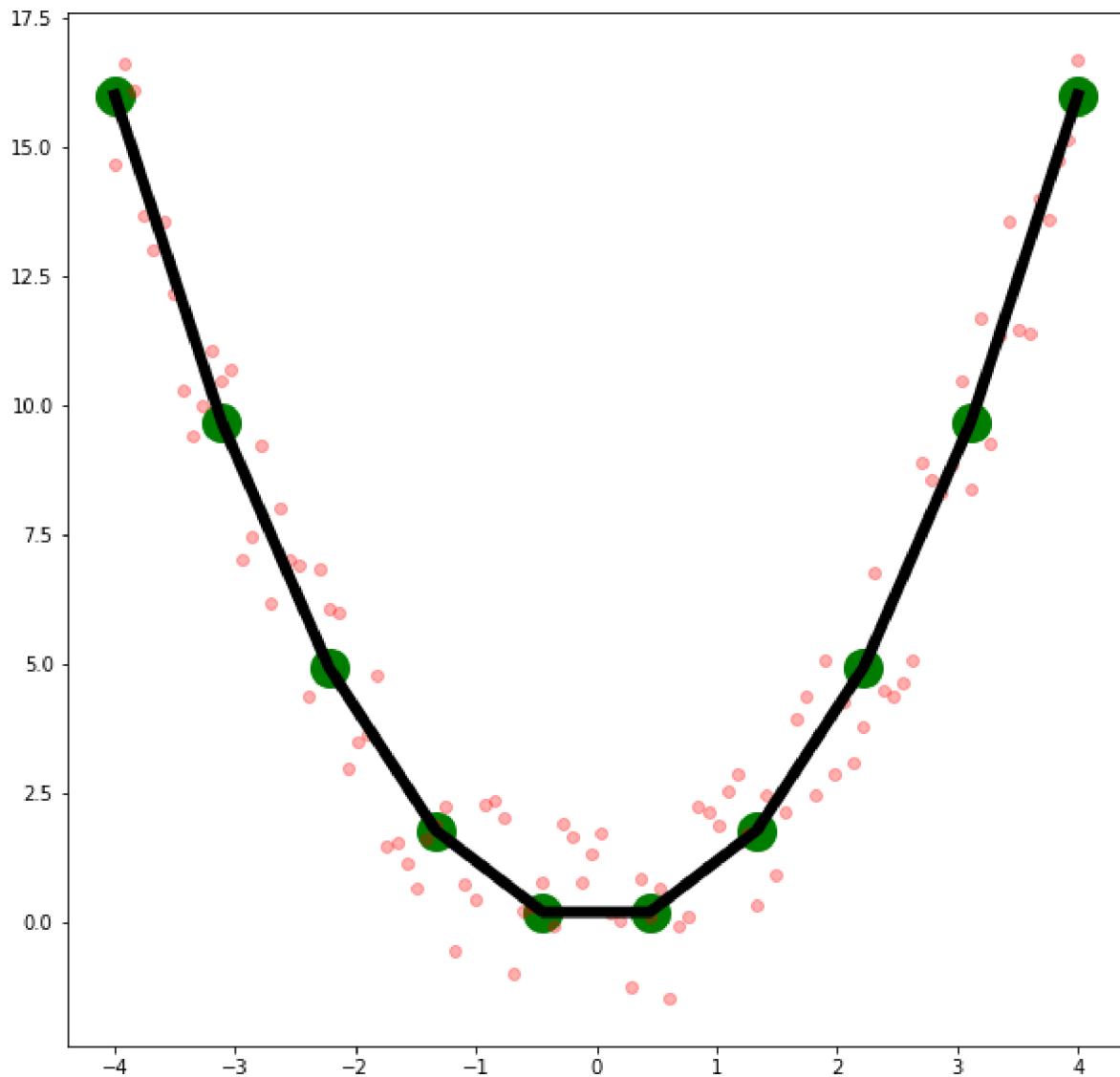


```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.figure(figsize=(10,10))
x = np.linspace(-4,4,10)

plt.plot(x, x**2, color="black", linewidth=6)
plt.scatter(x, x**2, c="green", s=400)

x_r = np.linspace(-4,4,100)
x_ruido = x_r**2 + (np.random.random(x_r.shape)-0.5)*4
plt.scatter(x_r,x_ruido, c="red", alpha=0.3)
```

```
<matplotlib.collections.PathCollection at 0x7ff4845f44f0>
```



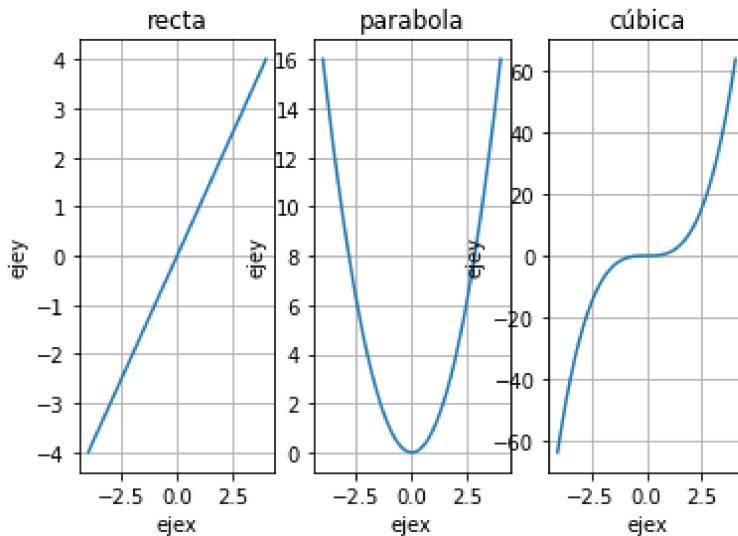
```

x=np.linspace(-4,4,100)
y=x
z=x**2
w=x**3
#plt.figure()
plt.subplot(1,3,1)
plt.plot(x,y)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('recta')
plt.grid()

#plt.figure()
plt.subplot(1,3,2)
plt.plot(x,z)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('parabola')
plt.grid()

#plt.figure()
plt.subplot(1,3,3)
plt.plot(x,w)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('cúbica')
plt.grid()

```



```

from google.colab import drive
drive.mount('/content/drive')

```

Mounted at /content/drive

```
#Rei
#path_data = "/content/drive/MyDrive/Machine Learning 2023-1 UManizales Maestría/Clase
#Arteaga
path_data = "/content/drive/MyDrive/Clases 2023-01/Programación Concurrente y Distribu

path_mesa = path_data+"/Mesa.jpg"
path_chestxray = path_data+"/ChestXRay.jpg"
```

```
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

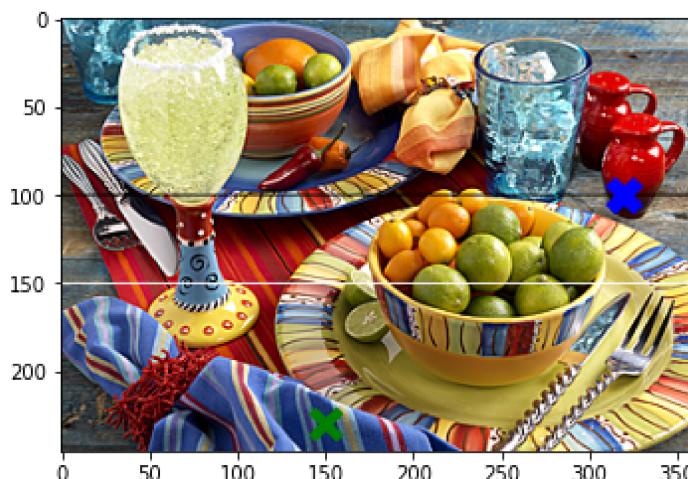
#Arteaga
img = io.imread(path_mesa)
img2 = io.imread(path_chestxray)

print("dimensiones", img.shape, "max", np.max(img), "min", np.min(img), "tipo", type(im

plt.scatter(320,100, marker="x", s=200, linewidth=7, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print("pixel at blue marker ", img[100,320,:])
print("pixel at green marker", img[230,150,:])
img[100,:,:]=0
img[150,:,:]=255
#plt.grid() # remove gridlines
plt.imshow(img)
```

```
dimensiones (246, 360, 3) max 255 min 0 tipo <class 'numpy.ndarray'>
pixel at blue marker [75 0 5]
pixel at green marker [24 48 112]
```

<matplotlib.image.AxesImage at 0x7ff45c6ccf10>



```
array([[[90, 108, 132],
 [224, 254, 254],
 [64, 110, 143],
 ...,
 [173, 188, 191],
 [144, 162, 164],
 [172, 189, 196]],

 [[34, 61, 80],
 [212, 223, 241],
 [156, 201, 230],
 ...,
 [173, 192, 198],
 [173, 193, 200],
 [158, 187, 191]],

 [[27, 57, 83],
 [134, 141, 157],
 [172, 231, 247],
 ...,
 [195, 205, 207],
 [188, 193, 197],
 [175, 186, 190]],

 ...,

 [[44, 52, 55],
 [36, 45, 52],
 [38, 43, 46],
 ...,
 [119, 113, 97],
 [120, 122, 109],
 [122, 125, 130]],

 [[57, 47, 38],
 [54, 46, 43],
 [55, 52, 47],
 ...,
 [109, 99, 87],
 [113, 110, 91],
 [105, 108, 101]],

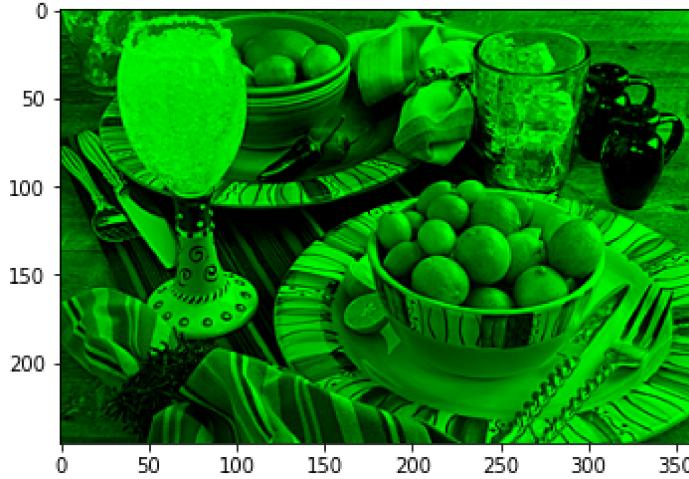
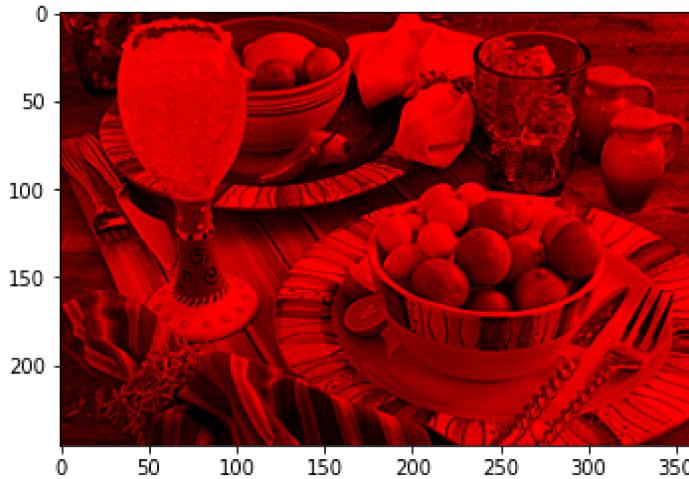
 [[58, 56, 57],
 [69, 70, 65],
 [61, 66, 69],
 ...,
 [139, 133, 119],
 [134, 119, 98],
 [132, 123, 108]]], dtype=uint8)
```

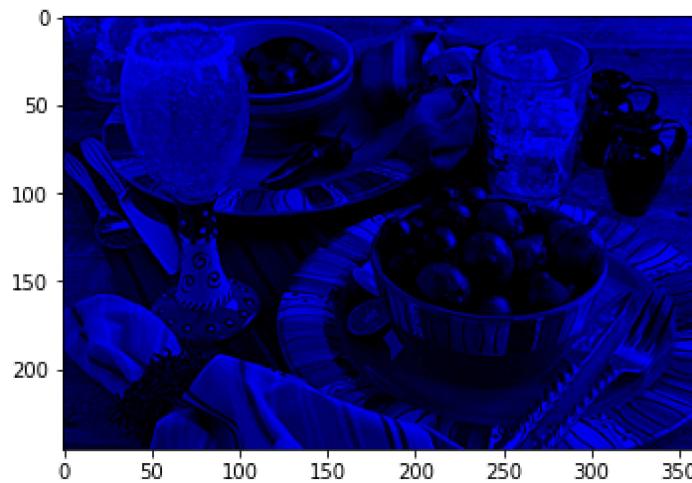
```
img = io.imread(path_mesa)
img[:, :, 1]=0
img[:, :, 2]=0
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 2]=0
plt.figure()
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 1]=0
plt.figure()
plt.imshow(img)
```

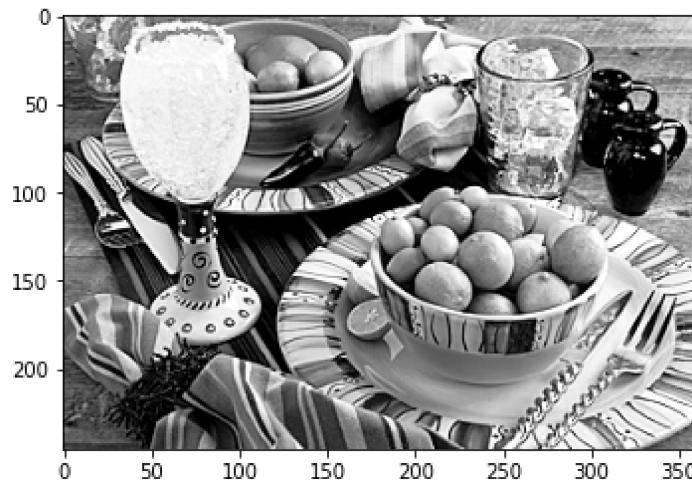
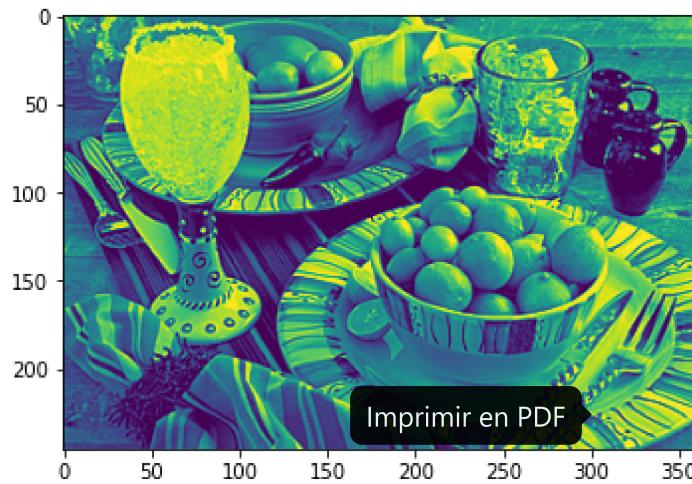
<matplotlib.image.AxesImage at 0x7ff45c4e0310>





```
img = io.imread(path_mesa)
plt.imshow(img[:, :, 1])
plt.figure()
plt.imshow(img[:, :, 1], cmap = plt.cm.Greys_r)
```

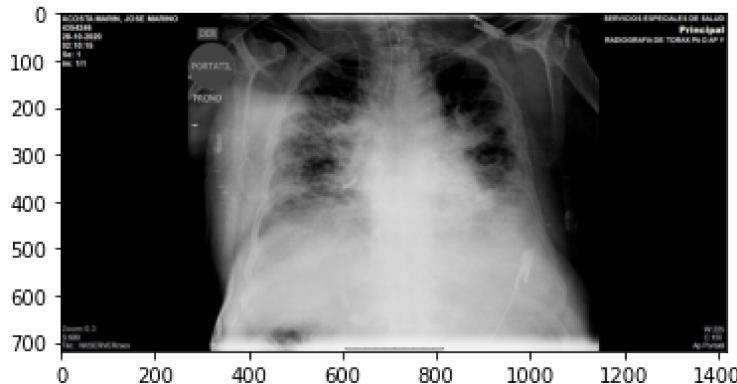
<matplotlib.image.AxesImage at 0x7ff45c3c1700>



```
Otra imagen
print(img2.shape)
plt.imshow(img2)
```

(718, 1418, 3)

<matplotlib.image.AxesImage at 0x7ff45c3cd940>

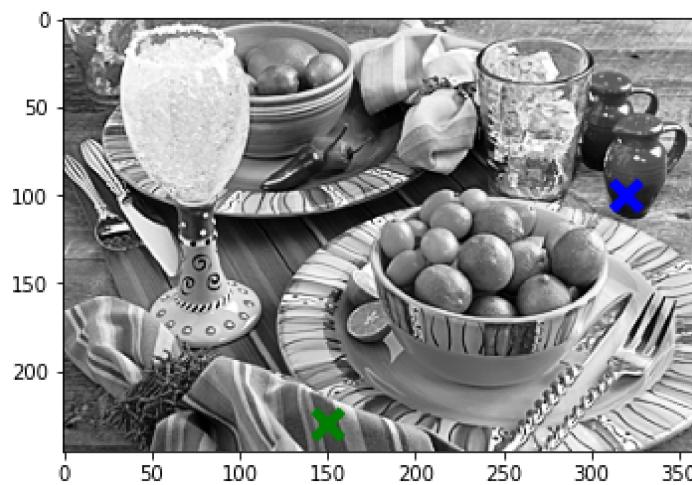


La manipulación de imágenes se *reduce* a realizar cálculos sobre los valores de luminosidad de cada pixel. Por ejemplo, obtenemos una versión en escala de grises promediando los valores RGB de cada pixel. Esto se hace de manera natural con la función `np.mean` y el argumento `axis` adecuado.

```
gimg = np.mean(img, axis=2)
print("dimensiones", gimg.shape, "max", np.max(gimg), "min", np.min(gimg))
plt.scatter(320,100, marker="x", s=200, linewidth=5, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print(img[100,320,:])
print("pixel at blue marker ", gimg[100,320])
print("pixel at green marker", gimg[230,150])
plt.grid() # remove gridlines
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

dimensiones (246, 360) max 255.0 min 0.0  
[75 0 5]  
pixel at blue marker 26.666666666666668  
pixel at green marker 61.333333333333336

<matplotlib.image.AxesImage at 0x7ff45c182640>



## Ejercicio

También podemos acceder a porciones (**parches**) de la imagen usando la notación natural de matrices de Python. Si además, reducimos la luminosidad de cada pixel a la mitad lo que hacemos es oscurecer la imagen.

1. Cargar nuevamente la imagen de la mesa
2. Visualizarla
3. Obtener una nueva imagen promediando los 3 canales RGB
4. Visualizarla
5. Extraer el jarrón de la posición 50:100 y 300:350 (ver imagen anterior)
6. Visualizarla
7. Dividir todos los pixeles del parche extraído (item 5) entre 2
8. Compara los parches (visualiza usando los parámetros: `cmap=>gray<`, `vmin=0`, `vmax=255`), el original y el de división de cada pixel entre 2 ¿Qué observa?

```
Escribe tu código aquí
gimg = np.mean(img, axis=2)
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

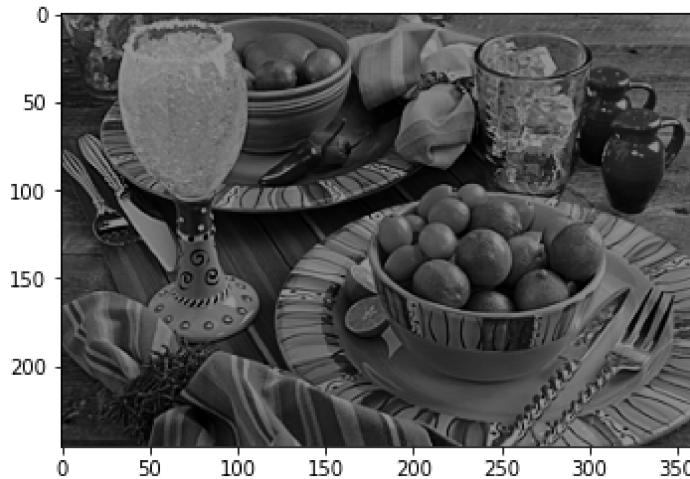
```
<matplotlib.image.AxesImage at 0x7ff45c160760>
```



```
gimg2=gimg/2
```

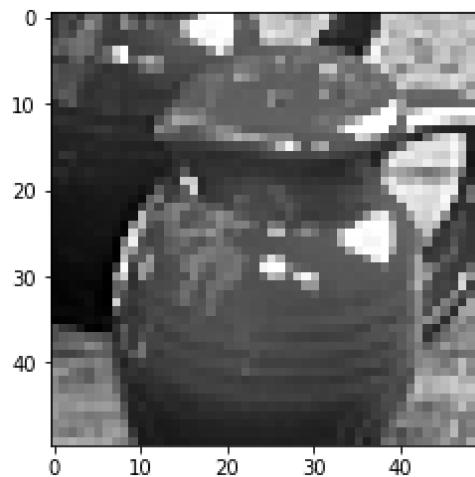
```
plt.imshow(gimg2, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff45c0a1b80>
```



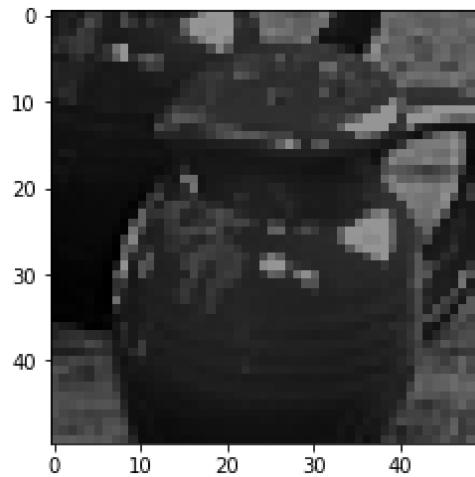
```
jarron=gimg[50:100 , 300:350]
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ecbee80>
```



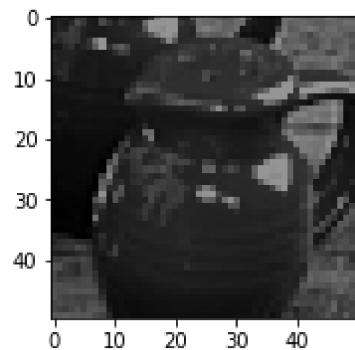
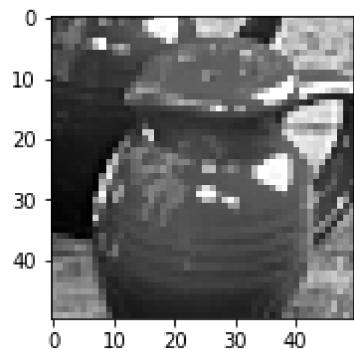
```
jarronOscurito=jarron/2
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ec2b160>
```



```
plt.subplot(1,2,1)
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
plt.subplot(1,2,2)
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44e835fa0>
```



# Método de euler para ecuaciones diferenciales.

## Contenido

- Método de euler para ecuaciones diferenciales.
- Actividad:

El método de Euler es un procedimiento numérico simple para resolver ecuaciones diferenciales ordinarias (EDOs) con un valor inicial dado. Es un método iterativo que comienza en un punto conocido y avanza paso a paso hasta llegar al punto deseado, utilizando la pendiente de la solución (es decir, la derivada) para avanzar en cada paso.

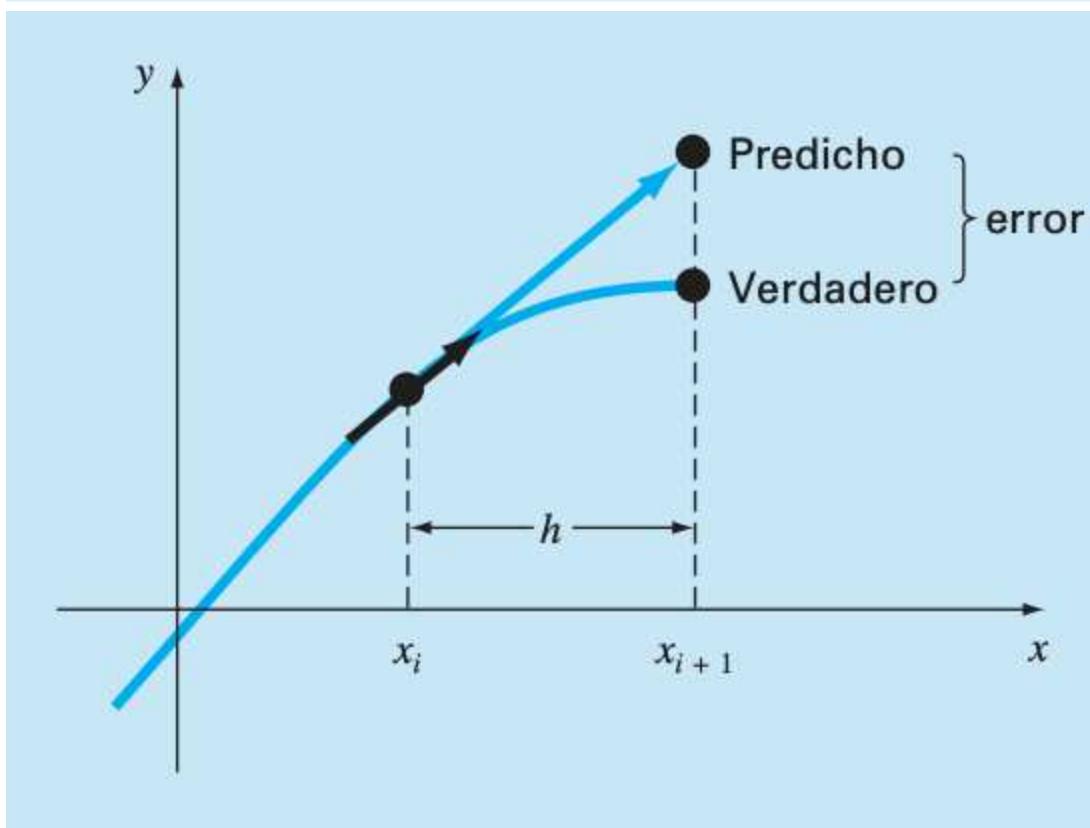
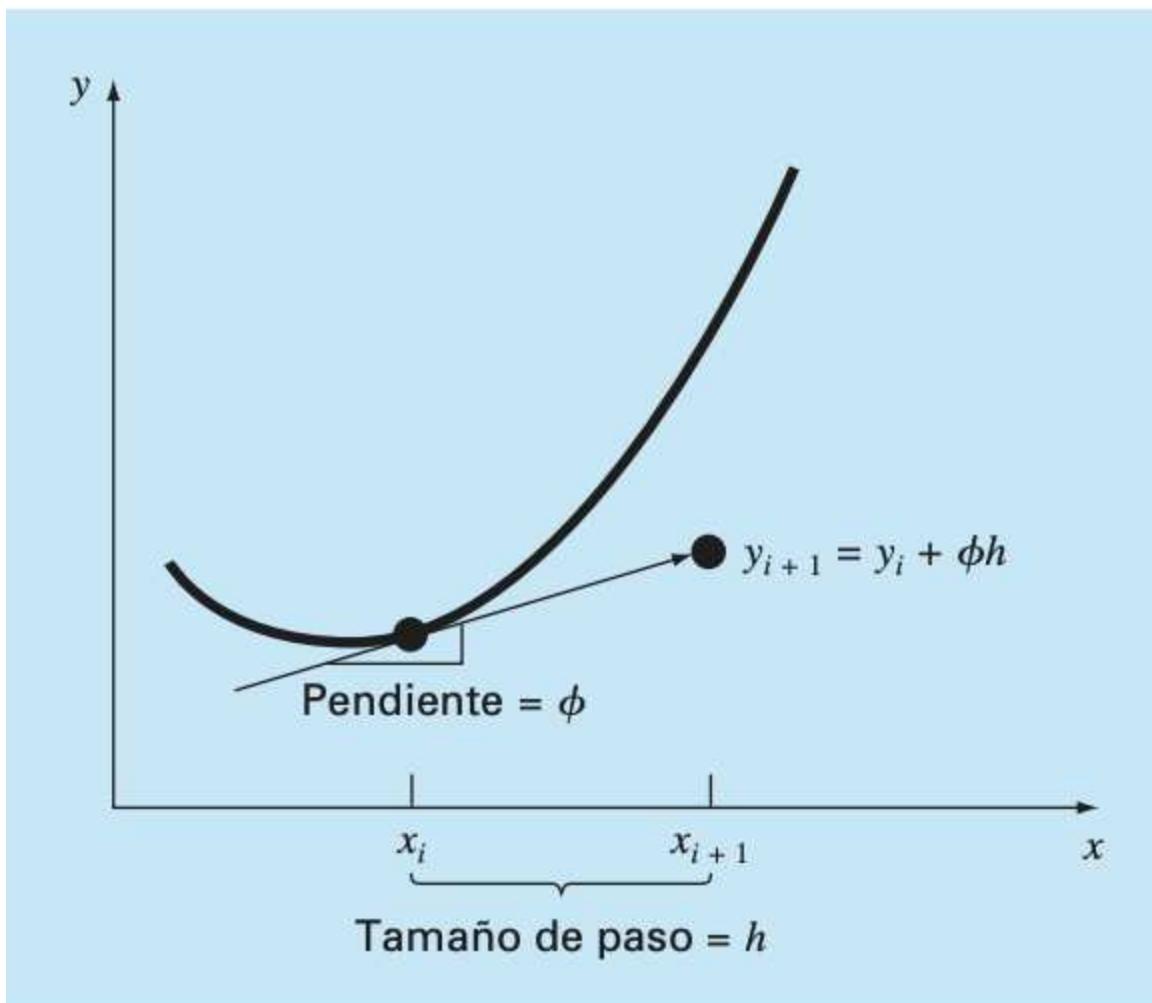
En este capítulo se enfoca la solución de ecuaciones diferenciales ordinarias de la forma:

$$\frac{dy}{dx} = f(x, y)$$

en términos matemáticos calculabamos el siguiente de la forma:

$$y_{i+1} = y_i + mh$$

Donde  $m$  es la pendiente estimada que se usa para extrapolar desde un valor anterior  $y_i$  a un nuevo valor  $y_{i+1}$  en una distancia  $h$ . Este valor se usa paso a paso para calcular un valor posterior y trazar la trayectoria de la solución. Normalmente, la pendiente representa la derivada de la función en el punto  $y_i$



# Algoritmo de Euler para la integración

1. Hacer  $y_{i+1} = y_i + h f'(x_i, y_i)$

2. Hacer  $x_i = x_{i+1}$

## Ejercicio:

Hallar la integral geométrica de la función  $y' = 2t$  con una condición inicial  $y(0)=0$  y un tiempo de integración de 0 hasta 0.6 y un paso de integración de 0.1

```
Definimos la función
import sympy as sp
import numpy as np
import pandas as pd

t = sp.symbols('t')
dy = 2*t

y0 = 0 # Condición inicial
ti = 0 # Tiempo inicial
h = 0.1 # paso de integración
tf = 0.6 # Tiempo final

Definimos los vectores donde almacenaremos el resultado.
x = np.arange(ti, tf + h, h)
y = np.empty_like(x)
y[0] = y0
print(x)
print(y)
```

```
[0. 0.1 0.2 0.3 0.4 0.5 0.6]
[0.00000000e+000 1.68821824e+195 2.51313104e+180 4.44032494e+252
 5.40026746e-310 0.00000000e+000 0.00000000e+000]
```

```
columnas = ['it','t','yi','yi+1']
tabla = pd.DataFrame(columns=columnas)
for i in range(0,len(y)-1):
 y[i+1] = y[i] + h * dy.subs({t:x[i]})
 nueva_fila = pd.DataFrame(data={'it':[i+1], 't':[round(x[i],2)], 'yi':[round(y[i],2)]})
 tabla = pd.concat([tabla,nueva_fila], ignore_index=True)

tabla.head()
```

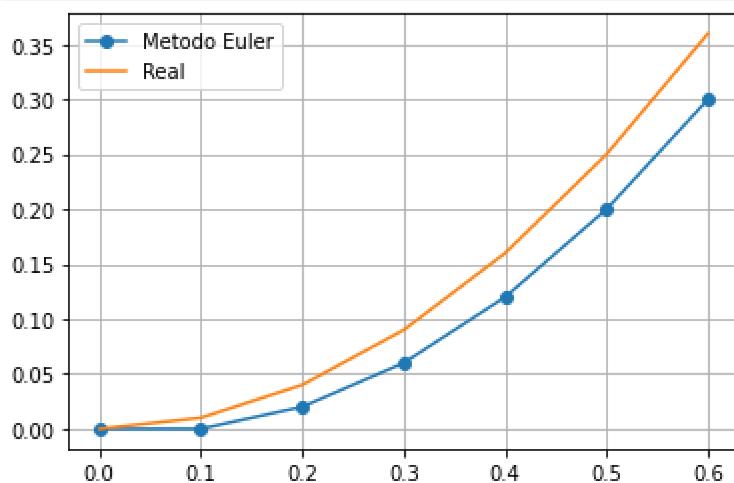
```
C:\Users\ricar\AppData\Local\Temp\ipykernel_9580\3804453467.py:6: FutureWarning: The b
 tabla = pd.concat([tabla,nueva_fila], ignore_index=True)
```

	it	t	yi	yi+1
0	1	0.0	0.00	0.00
1	2	0.1	0.00	0.02
2	3	0.2	0.02	0.06
3	4	0.3	0.06	0.12
4	5	0.4	0.12	0.20

```
import matplotlib.pyplot as plt
plt.figure()
fig, ax = plt.subplots()
ax.grid()
ax.plot(x,y, marker='o', label ="Metodo Euler")
ax.plot(x, x**2, label="Real")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1f5bb1e7730>
```

```
<Figure size 432x288 with 0 Axes>
```

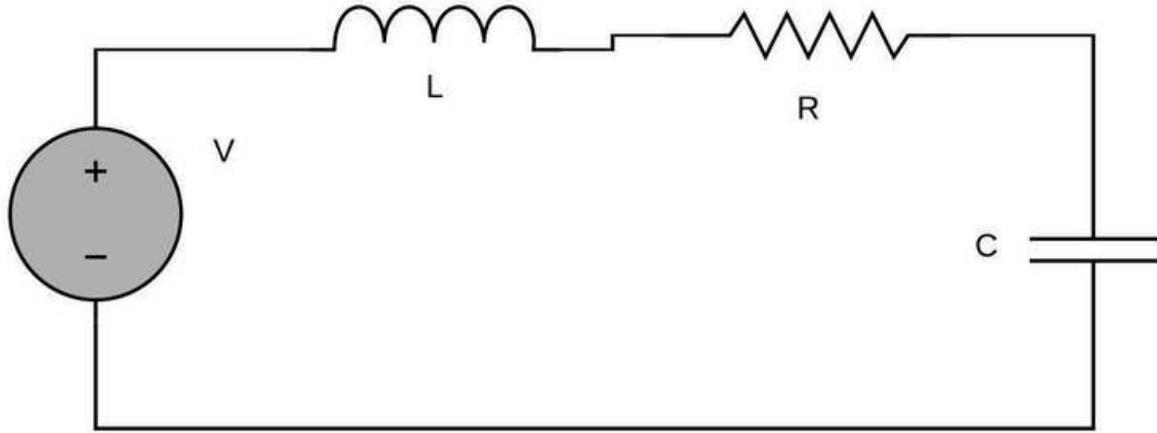


## Circuito RLC: Análisis analítico.

Vamos a deducir el modelo del siguiente circuito:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$

$$\frac{dv}{dt} = \frac{1}{C}i(t)$$



Definamos las ecuaciones del sistema en [sympy](#)

```
from sympy import *

t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)

didt=i.diff(t)
dvdt=v.diff(t)
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -\frac{Ri(t)}{L} + \frac{vi}{L} - \frac{v(t)}{L}$$

```
Asumimos valores de R,L y C = 1

expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = \frac{i(t)}{C}$$

```
Asumimos valores de R,L y C = 1

expr2=expr2.subs(L,1).subs(R,1).subs(C,1).subs(vi,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
init_printing(use_latex=True)
s=dsolve([expr1,expr2])
s
```

$$[i(t) = -(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2), v(t) = C_1e^{-t/2}\cos(\sqrt{3}t/2) - C_2e^{-t/2}\sin(\sqrt{3}t/2) + \sin^2(\sqrt{3}t/2) + \cos^2(\sqrt{3}t/2)]$$

```
Vamos a hallar los valores de las ctes C1 y C2
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1)
sol = solve([eq1,eq2], [C1,C2])
print(sol)
```

$$\{C1: -0.900000000000000, C2: 0.404145188432738\}$$

```
eq1
```

$$-0.5C_1 - 0.866025403784439C_2 = 0.1$$

```
s[0].rhs
```

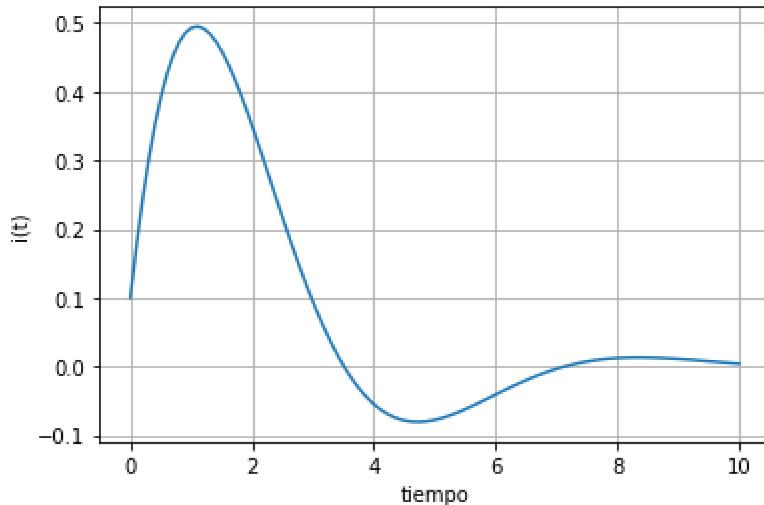
$$-(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2)$$

```
Generemos dos funciones que permitan reemplazar los valores de t dados para graficar

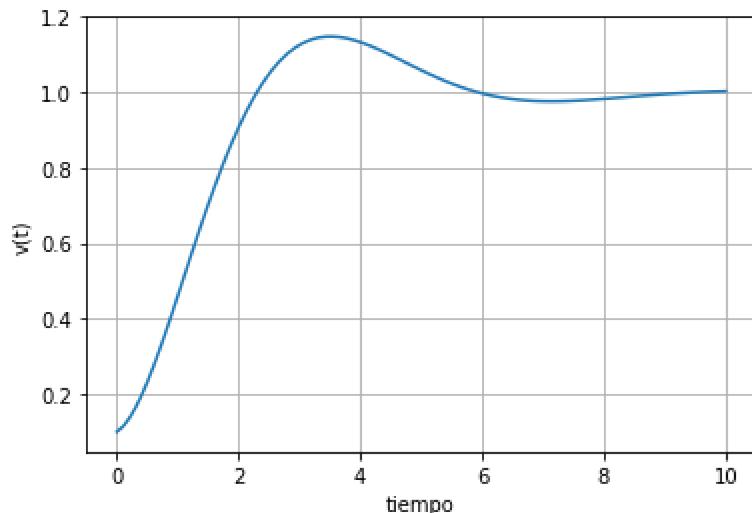
e1 = lambdify(t,s[0].rhs.subs(sol), 'numpy')
e2 = lambdify(t,s[1].rhs.subs(sol), 'numpy')
```

```
t_vals = np.linspace(0,10,100)
```

```
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
```



```
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
```



las gráficas para la corriente  $i(t)$  y el voltaje  $v(t)$  en función del tiempo para el circuito RLC analizado. Como puedes ver, ambas funciones muestran un comportamiento oscilatorio amortiguado, que es típico en este tipo de circuitos.

```

def euler_method_system(funcs, y0, t_range, dt):
 """
 Método de Euler para un sistema de ecuaciones diferenciales.

 :param funcs: Lista de funciones que representan el sistema de ecuaciones diferenciales.
 :param y0: Condiciones iniciales para cada variable en el sistema.
 :param t_range: Rango de tiempo como una tupla (inicio, fin).
 :param dt: Paso de tiempo.
 :return: Tupla de listas (tiempos, valores de cada variable).
 """
 t_values = np.arange(t_range[0], t_range[1], dt)
 y_values = [np.array(y0)]

 for t in t_values[:-1]:
 y_current = y_values[-1]
 y_next = y_current + dt * np.array([f(*y_current, t) for f in funcs])
 y_values.append(y_next)

 return t_values, np.array(y_values).T

Funciones para el sistema de ecuaciones diferenciales
def func_i(i, v, t):
 return 1 - i - v

def func_v(i, v, t):
 return i

Paso de tiempo para el Método de Euler
dt = 0.1

Condiciones iniciales y rango de tiempo
initial_i = 0.1
initial_v = 0.1
time_range = (0, 10)

Condiciones iniciales para i y v
initial_conditions = [initial_i, initial_v]

Solución aproximada para i(t) y v(t) usando el Método de Euler para sistemas
t_values_euler, [i_values_euler_system, v_values_euler_system] = euler_method_system(
 [func_i, func_v], initial_conditions, time_range, dt
)

Graficando soluciones analíticas y aproximadas
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(t_vals, e1(t_vals), label='Analítico i(t)')
plt.plot(t_values_euler, i_values_euler_system, label='Aproximado i(t)', linestyle='--')
plt.xlabel('Tiempo')
plt.ylabel('Corriente (i(t))')
plt.title('Corriente en función del tiempo')
plt.grid(True)
plt.legend()

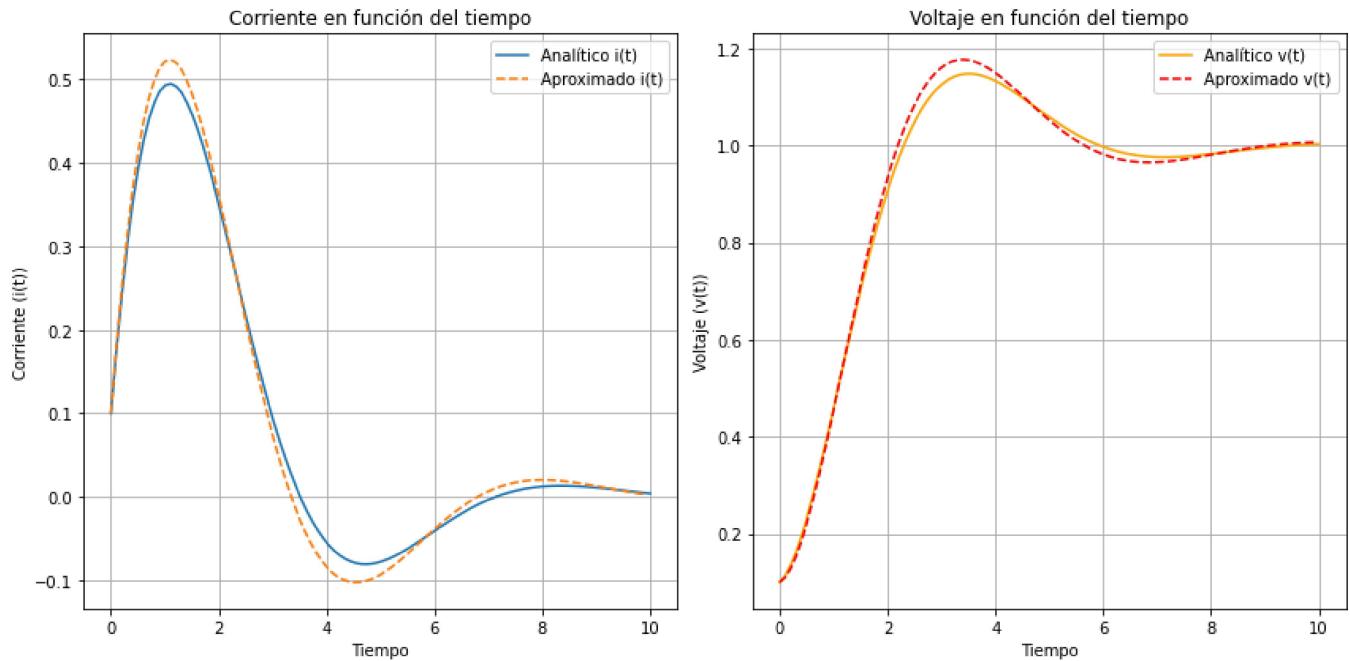
```

```

plt.subplot(1, 2, 1)
plt.plot(t_vals, e2(t_vals), label='Analítico v(t)', color='orange')
plt.plot(t_values_euler, v_values_euler_system, label='Aproximado v(t)', linestyle='--')
plt.xlabel('Tiempo')
plt.ylabel('Voltaje (v(t))')
plt.title('Voltaje en función del tiempo')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

```



Ajustar el código para que, dado un sistema de ecuaciones diferenciales, pueda calcular la solución del mismo dadas unas condiciones iniciales y un paso con el método de euler.

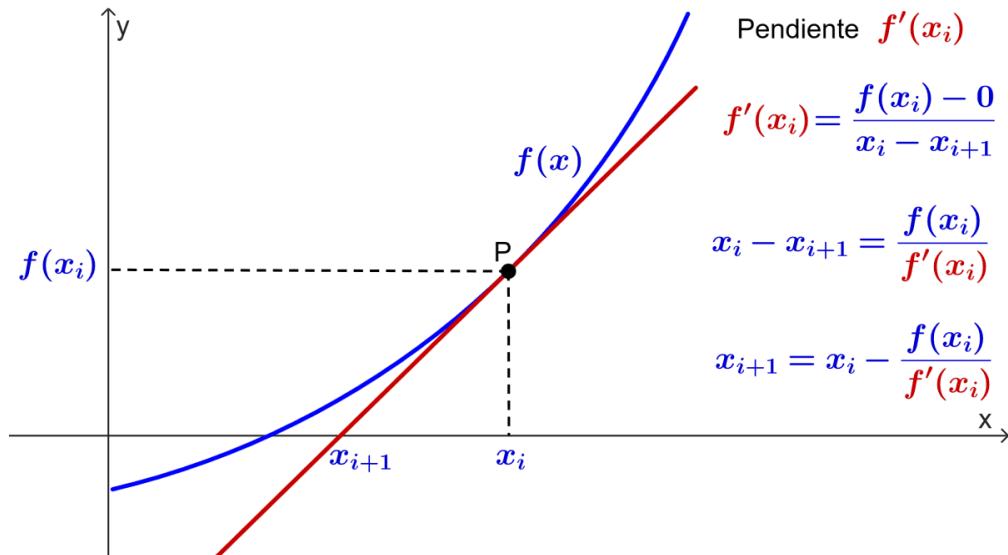
# Método de newton-raphson

## Contenido

- Algoritmo de Newton Raphson
- Veamos un ejemplo práctico
- Ejercicio:

Es el método más usado y trabaja como referencia la pendiente ( $m$ ) de la recta tangente en una raíz  $x_i$  incial para hallar el  $x_{i+1}$ .

Este método tiende a acercarse muy rápido a la raíz. Pero también puede tender a la divergencia rápidamente.



$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

$$f'(x_i)[x_i - x_{i+1}] = f(x_i)$$

$$f'(x_i)(x_i) - f'(x_i)(x_{i+1}) = f(x_i)$$

$$f'(x_i)(x_{i+1}) = f'(x_i)(x_i) - f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

# Algoritmo de Newton Raphson

1. Se calcula el siguiente  $x_{i+1}$  con la formula.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

2. Se repite el paso 1 hasta convergencia.

## Error relativo

$$e_r = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right|$$

## Tolerancia

Precisión que se requiere en el método!!

## Veamos un ejemplo práctico

Hallar la raíz de  $f(x) = e^{-x} - x$  con el método de Newton Raphson y una condición inicial  $x_i = 0$  y  $tol = 1\%$

## Solución

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

```
Tomamos el intervalo inicial y grafiquemos la función dentro de ese valor

x = sp.symbols('x')

y = sp.E**(-x)-x
print("La función es: ",y)

xi = 0

print(f"El punto inicial es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi : f({xi})={fxi}")
```

La función es:  $-x + \exp(-x)$   
 El punto inicial es: 0  
 la función evaluada en xi :  $f(0)=1$

```
@title Gráfica incial de la función y x_i dado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

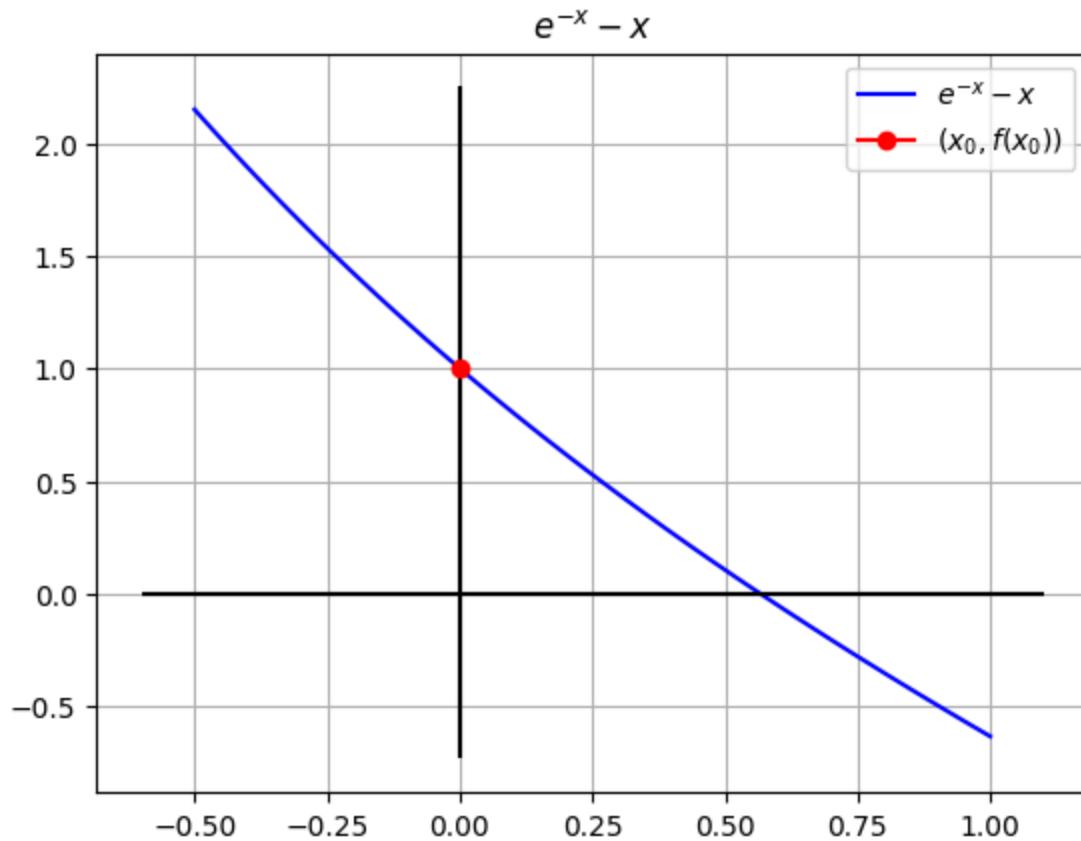
Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



Formula para el cálculo del  $x_{i+1}$  en el ejercicio dado

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{e^{-x} - x}{-e^{-x} - 1}$$

```
Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

La derivada de la función  $f(x)=-x + \exp(-x)$  ->  $f'(x)= -1 - \exp(-x)$   
 La derivada evaluada en xi  $f'(xi)= -2$

```
Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

La raíz supuesta está en x=0.5000

```
@title Gráfica función con el x_i y x_{i+1} calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

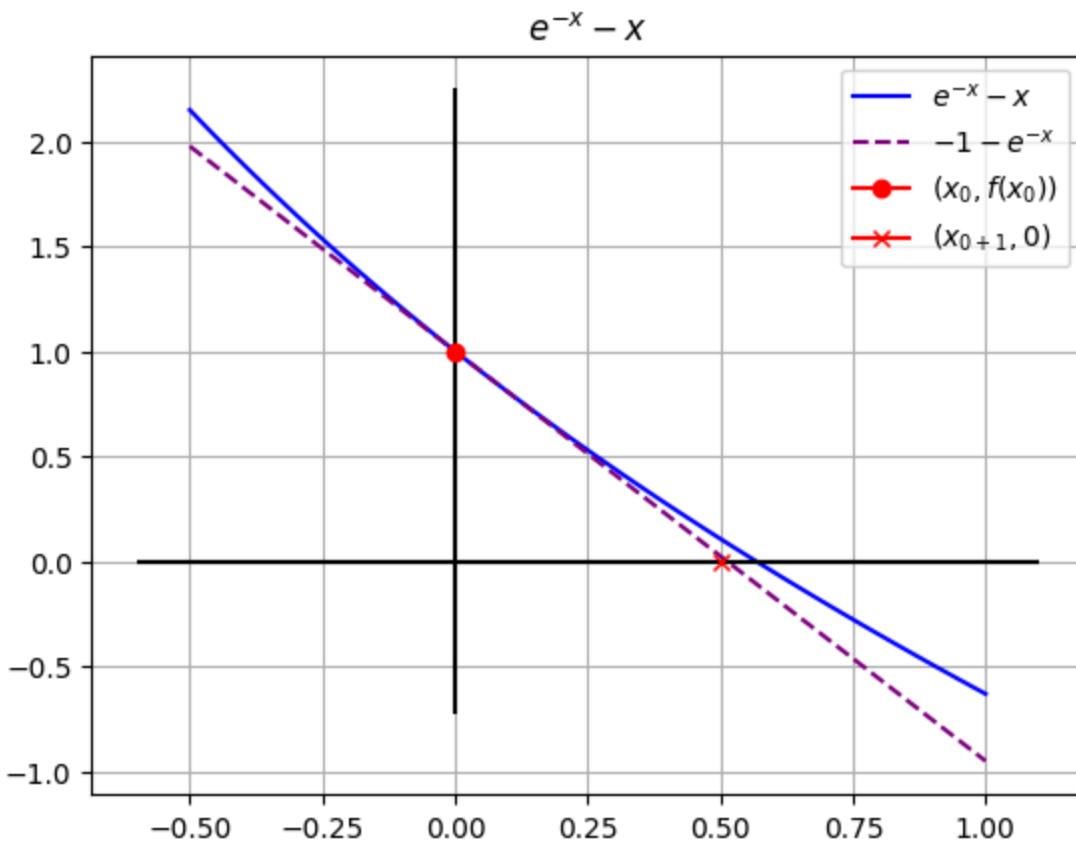
Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{0+1},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

El error del cálculo es: 100.0%

```
Generemos la tabla de iteraciones
@title Tabla de iteraciones
import pandas as pd

columnas = ['xi','Xi+1','f(xi)','f'(Xi)', 'er(%)']

primer_iter = {'xi':[xi], 'Xi+1':[xs], 'f(xi)':[fxi], "f'(Xi)": [dfxi], 'er(%)':[round(er,1)]}

tabla = pd.DataFrame(data=primer_iter, columns=columnas)
tabla.head(1)
```

	<b>xi</b>	<b>Xi+1</b>	<b>f(xi)</b>	<b>f'(Xi)</b>	<b>er(%)</b>
<b>0</b>	0	0.5000	1	-2	100.0

## Sigamos iterando...

```
Nuestro x_i+1 ahora se convierte en nuestro x_i
xi = xs
```

```
print(f"El punto inicial ahora es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi ahora es: f({xi})={fxi}")
```

El punto inicial ahora es: 0.5000  
la función evaluada en xi ahora es: f(0.5000)=0.1065

```
@title Gráfica incial de la función y x_i para la iteración 2
import numpy as np
plt.figure()

r = np.linspace(-0.5, 1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

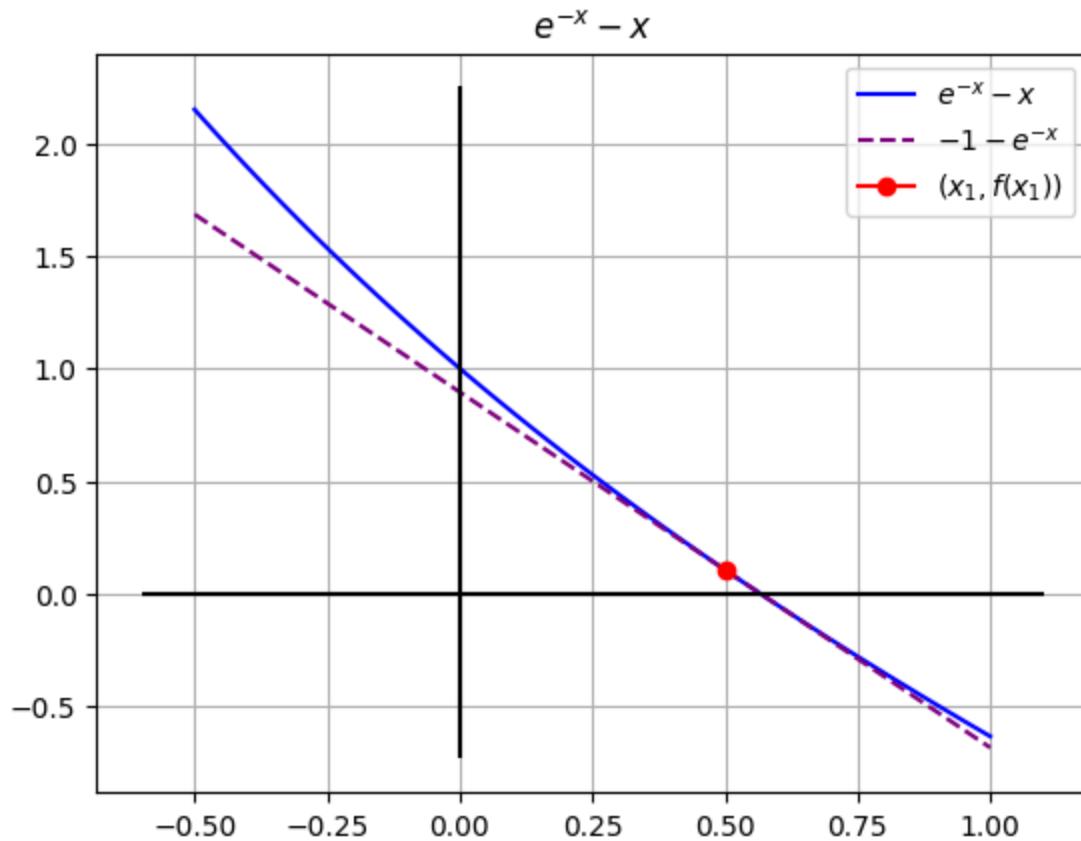
h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

Punto inicial
ax.plot([xi],[fxi], color='red', marker='o', label='${x_1,f(x_1)}$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

La derivada de la función  $f(x)=-x + \exp(-x)$  ->  $f'(x)= -1 - \exp(-x)$   
 La derivada evaluada en xi  $f'(xi)= -1.605$

```
Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

La raíz supuesta está en  $x=0.5664$

```
@title Gráfica función con el x_i y x_{i+1} calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

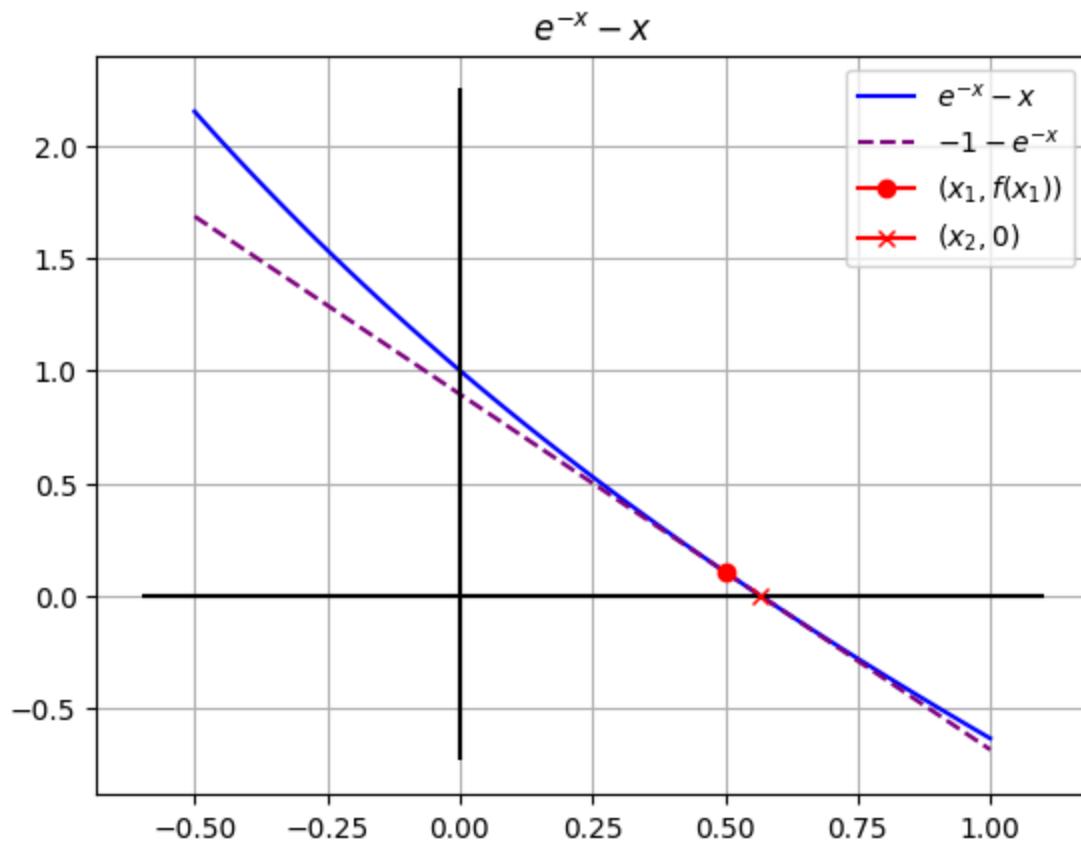
Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_2,0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

El error del cálculo es: 11.7%

```
@title Actualizamos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
tabla = tabla.append(nueva_fila, ignore_index=True)
tabla.head()
```

```
<ipython-input-141-30e46d477870>:3: FutureWarning: The frame.append method is deprecated
tabla = tabla.append(nueva_fila, ignore_index=True)
```

	$x_i$	$X_{i+1}$	$f(x_i)$	$f'(X_i)$	$er(\%)$
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72

## Ahora veamos el algoritmo completo

```

@title iteremos hasta que el error sea de 1%
from IPython.display import HTML, display, clear_output
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import sympy as sp
import pandas as pd

x = sp.symbols('x')

y = sp.E**(-x)-x
xi = 0

print("La función es: ",y)

columnas = ['xi','Xi+1','f(xi)','f'(Xi)','er(%)']
tabla = pd.DataFrame(columns=columnas)

tol = 1

er = tol+1
it = 1

ims = []

while er > tol:
 fxi = round(y.subs({x: xi}), 4)

 #####
 plt.figure()
 fig, ax = plt.subplots()

 r = np.linspace(-0.5,1, 100)
 fx = [y.subs({x:x_i}) for x_i in r]

 ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

 ## Plano cartesiano (Ejes)
 ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
 ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

 ## Punto inicial

 ax.plot([xi],[fxi], color='red', marker='o', label=f'$(x_{it}), f(x_{it})) = ({xi},{fxi})$')

 ## Calculemos la derivada
 dy = y.diff()
 ## Evaluemos la derivada en x_i
 dfxi = round(dy.subs({x:xi}),4)

 ## Calculemos el x_i+1 (siguiente)
 xs = round(xi - ((fxi)/(dfxi)),4)
 ax.plot([xs],[0], color='red', marker='x', label=f'$(x_{it+1},0) = ({xs},0)$')

 ims.append(ax)

 tol = 10**-it
 er = abs(fxi - xs)
 it += 1
 xi = xs

```

```
Pintar la recta tangente
h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()

Calculo del error
er = np.abs((xs-xi)/(xs)) * 100

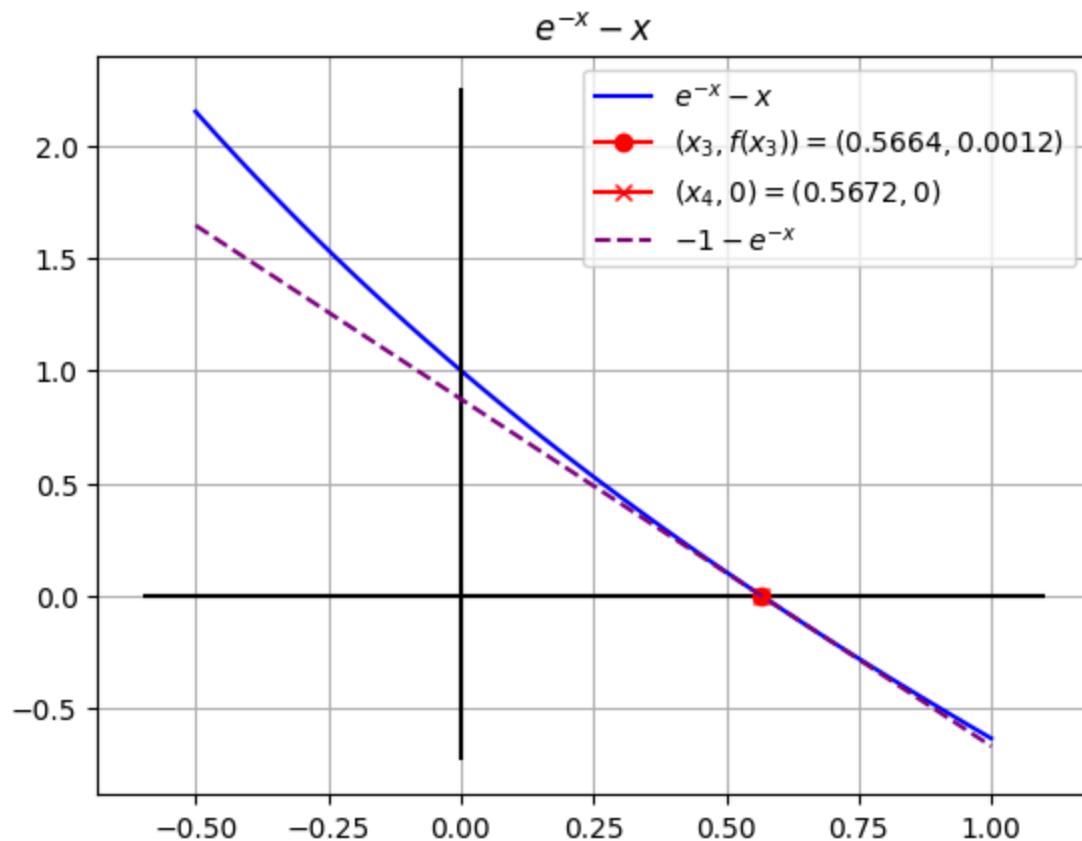
Actualicemos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
nueva_fila = pd.DataFrame([nueva_fila])
tabla = pd.concat([tabla, nueva_fila], ignore_index=True)

display(HTML(tabla.head(it).to_html()))

#Actualizamos la iteración y el valor de x actual (xi)
it += 1
xi = xs

print("")
input()
clear_output(wait=True)
```

<Figure size 640x480 with 0 Axes>



	$x_i$	$X_{i+1}$	$f(x_i)$	$f'(X_i)$	$er(\%)$
<b>0</b>	0	0.5000	1	-2	100.0
<b>1</b>	0.5000	0.5664	0.1065	-1.605	11.72
<b>2</b>	0.5664	0.5672	0.0012	-1.568	0.1412

```

@title Grafico dinámico (movie del método)
Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML;
import sympy as sp
rc('animation', html='html5');

x = sp.symbols('x')
y = sp.E**(-x)-x
dy = y.diff()
r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')
ax.set_title("$e^{-x} - x$")
ax.grid()

Se definen los atributos que debe tener la linea o en este caso el punto que se va a dibujar
linea1, = ax.plot([],[],'o',color = 'r', label = '')
linea2, = ax.plot([],[],'x',color = 'r', label = '')
linea3, = ax.plot([],[],'--',color = 'purple', label = '')

frames = len(tabla)

def graficar(i):

 xg = tabla['xi'].to_list()
 yg = tabla['f(xi)'].to_list()

 xs = tabla['Xi+1'].to_list()
 ys = tabla["f'(Xi)"].to_list()

 linea1.set_data(xg[i],yg[i])
 linea1.set_label(f"${x_{i}}, f(x_{i})= ({xg[i]},{yg[i]})$')

 linea1.set_data(xs[i],0)
 linea2.set_label(f"${x_{i}}, 0) = ({xs[i]},0)$')

 ## Pintar la recta tangente
 h=0.1
 dfx = (y.subs({x:xg[i]+h})-y.subs({x:xg[i]}))/h # derivative
 tan = y.subs({x:xg[i]})+dfx*(r-xg[i]) # tangent
 linea3.set_data(r,tan)
 linea3.set_label(f"${sp.latex(dy)}$")

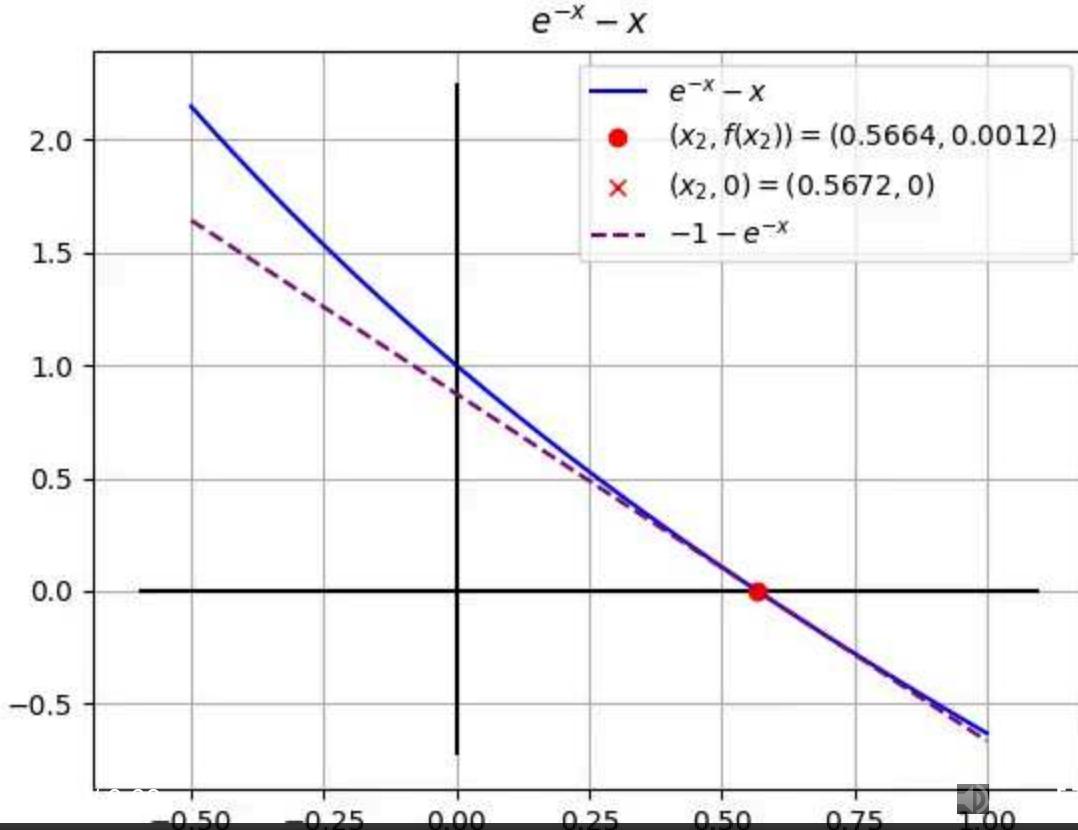
 ax.legend()
 return (linea1,linea2,linea3,)


```

```
plt.close()
```

```
Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=1000, repeat=False)
anim
```

```
<ipython-input-150-276efacf074c>:40: MatplotlibDeprecationWarning: Setting data with a
 linea1.set_data(xg[i],yg[i])
<ipython-input-150-276efacf074c>:44: MatplotlibDeprecationWarning: Setting data with a
 linea1.set_data(xs[i],0)
```



## Ejercicio:

Calcular la raíz de  $f(x) = \sin(x) - x$  en el punto  $x_i = 0.75$  con una  $tol <= 1\%$

Implementar una función que me permita dinámicamente modificar la función (pedirla por consola), la tolerancia y el punto inicial, además de la gráfica y que retorne la raíz, el error, la tabla de iteraciones y la gráfica dinámica de como itera el método.

TIP: Indagar sobre los formularios con entradas en COLAB.

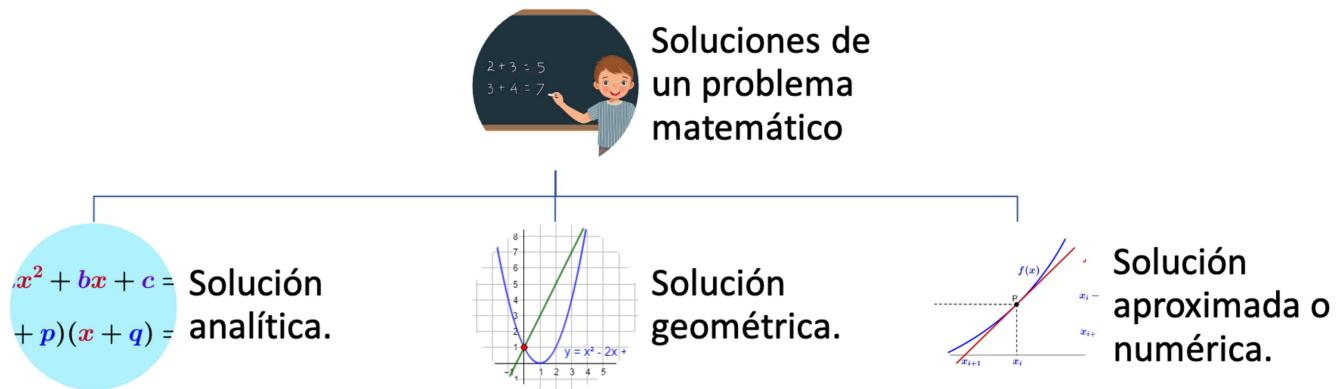
NOTA: Modificar el código del método para que se pase por consola la ecuación y se realice el método con esta entrada, mostrando la ecuación en el título y en los legend de forma dinámica.

# Métodos numéricos - Cálculo de raíces

## Contenido

- Solución de un problema matemático.
- Raíces de funciones

## Solución de un problema matemático.



## Raíces de funciones

Una raíz se define como una función igualada a cero o el intercepto de una curva con el eje X

$$f(x) = 0$$



### 💡 Encontremos las raíces de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

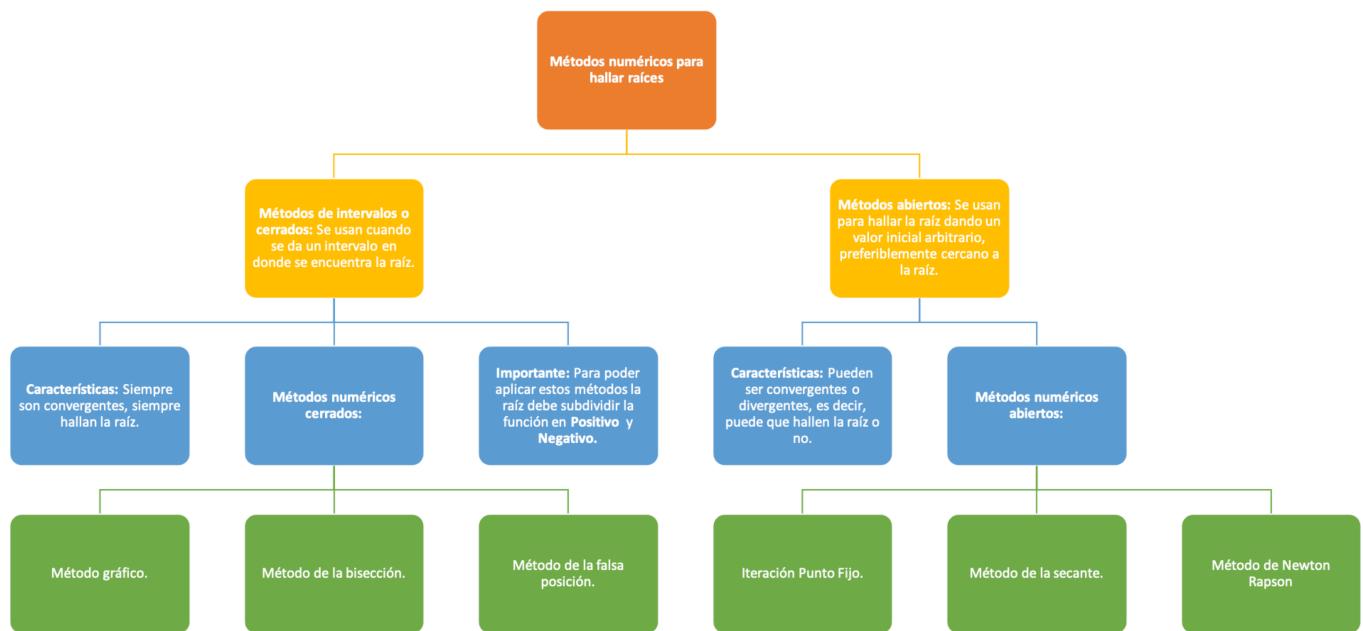
$$f(x) = x^2 + 5x + 6$$

### 💡 Encontremos las raíces de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

$$f(x) = x^2 + 2x + 1$$

## Métodos numéricos para hallar raíces



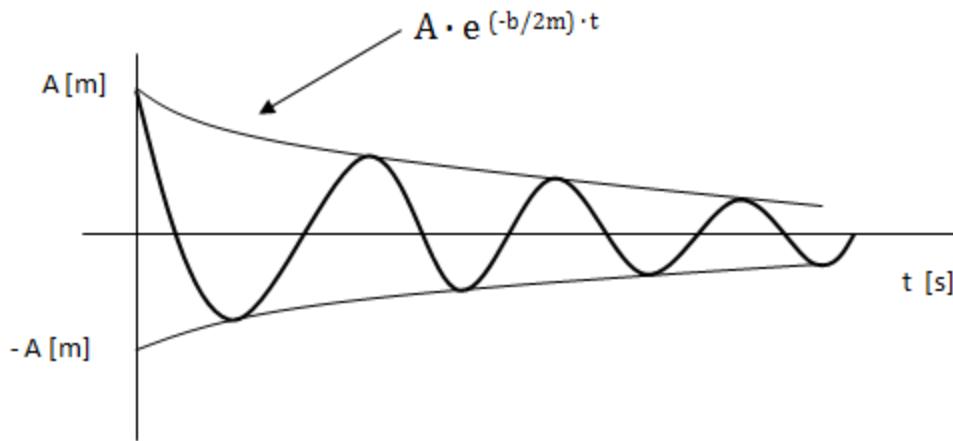
## Método gráfico:

El método gráfico sirve como método inicial para conocer dónde están las raíces de forma aproximada.

### Nota

Hallar una raíz geométricamente consiste en hallar el punto de cruce con el eje X

Encontremos las raíces de la siguiente función...



Este ejemplo tiene 8 raíces de multiplicidad 1

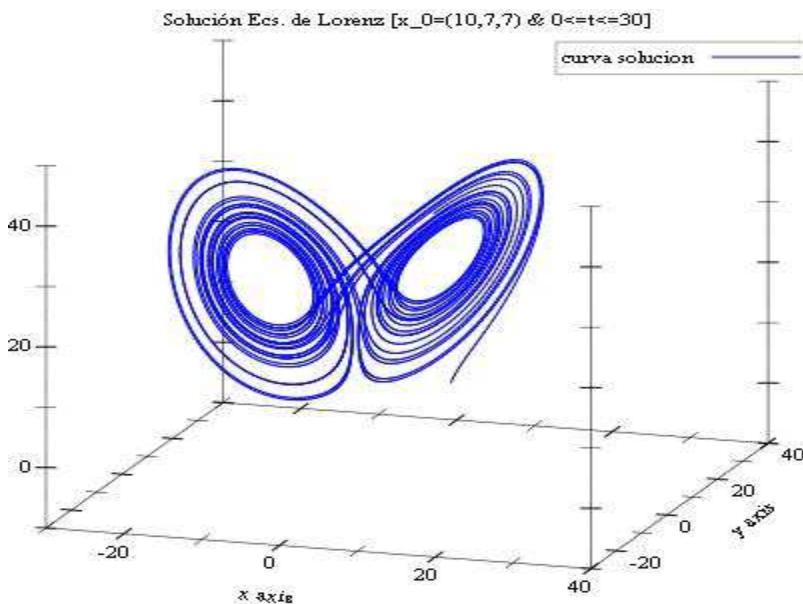
# Métodos Numéricos: Ecuaciones Diferenciales Ordinarias

## Contenido

- Métodos para solución de ecuaciones diferenciales.
- Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

## Métodos para solución de ecuaciones diferenciales.

Las ecuaciones diferenciales son una parte fundamental de las matemáticas aplicadas y la ingeniería, proporcionando una poderosa herramienta para modelar y entender fenómenos que varían con el tiempo o el espacio. Desde el movimiento de los planetas hasta el flujo de corriente en circuitos eléctricos, las ecuaciones diferenciales juegan un papel crucial en describir cómo cambian las cosas en el mundo real.



Sin embargo, la mayoría de las ecuaciones diferenciales no pueden resolverse con métodos analíticos, especialmente cuando se trata de modelos complejos o sistemas no lineales. Aquí es donde los métodos numéricos entran en juego. Estos métodos proporcionan aproximaciones prácticas para resolver ecuaciones diferenciales, permitiendo a los científicos e ingenieros obtener soluciones útiles donde las técnicas analíticas fallan o son demasiado complicadas para aplicarse.

---

## Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

- 1. Ingeniería y Física:** En el diseño de sistemas mecánicos, electrónicos y de control, las ecuaciones diferenciales modelan el comportamiento de componentes y sistemas. Por ejemplo, en la ingeniería eléctrica, los circuitos RLC, que combinan resistencias, inductancias y capacitancias, se analizan mediante ecuaciones diferenciales.
- 2. Meteorología y Modelado Climático:** Los modelos climáticos y meteorológicos utilizan ecuaciones diferenciales para predecir el clima y entender los patrones climáticos. Estos modelos son intrincadamente complejos y requieren el uso de métodos numéricos para su resolución.
- 3. Biología y Medicina:** En la farmacocinética, las ecuaciones diferenciales describen cómo los medicamentos se distribuyen y metabolizan en el cuerpo. También son fundamentales en la modelización de la dinámica de las poblaciones y la propagación de enfermedades.
- 4. Economía y Finanzas:** En la modelización financiera, las ecuaciones diferenciales se utilizan para entender la dinámica de los mercados y para la valoración de opciones y otros instrumentos financieros.
- 5. Ciencias de la Computación y la Inteligencia Artificial:** Los algoritmos de aprendizaje automático y redes neuronales a menudo implican resolver ecuaciones diferenciales para optimizar y entrenar modelos.

## Taller 1 - Programación Concurrente y Distribuida

**Nombre:** \_\_\_\_\_

**Código de estudiante:** \_\_\_\_\_

### Cuestionario Teórico

1. ¿Qué significa el seno de  $30^\circ$ ?
2. ¿A qué corresponde HPC en el contexto del curso?
3. ¿Es verdadero o falso que HPC es cualquier técnica computacional que soluciona un problema grande de forma más rápida que usando posiblemente sistemas simples?
4. ¿Cuáles de los siguientes conceptos se encuentran en HPC: Procesadores de alto rendimiento, Computación individualizada, Computación Paralela, Computación Distribuida, Computación en serie, Computación Grid, Procesadores de baja eficiencia?
5. ¿Cuál de las siguientes empresas no ofrece servicios de Cloud (servicios de infraestructura en la nube): Microsoft Azure, Amazon Web Services, Adobe Creative Cloud, Google Cloud, IBM Cloud, Oracle Cloud?
6. Ordene los siguientes componentes según su capacidad de procesamiento, de menor a mayor: CPU, GPU, HIVE, TPU.
7. **Evalúe como verdadero o falso:** La computación paralela consiste en varios sistemas acoplados por un secuenciador de trabajo sobre problemas relacionados.
8. **Evalúe como verdadero o falso:** La computación distribuida consiste en sistemas simples con varios procesadores trabajando sobre el mismo problema.
9. **Evalúe como verdadero o falso:** La computación Grid consiste en varios sistemas acoplados por software y redes para trabajar en conjunto en problemas simples o en problemas relacionados.
10. ¿Cómo se puede crear una función en Sympy que calcule la derivada de una función simbólica?
11. ¿Cómo se puede graficar una matriz NumPy utilizando Matplotlib?

12. Dado un arreglo NumPy de dos dimensiones, ¿cómo se calcula la suma de las filas y columnas eficientemente?
13. ¿Cómo se puede integrar una función simbólica con SymPy?
14. ¿Cómo se puede encontrar el número de elementos distintos en un arreglo NumPy?
15. Cuando cargas un archivo .csv tabular con datos estructurados, ¿cómo se carga usando Pandas?
16. Mencione las librerías utilizadas para graficar en Python.
17. ¿Qué estructura de datos nativa de Python es equivalente a los JSON?
18. Con 5 bits, ¿cuál es el número más grande que se puede representar con signo y sin signo?
19. ¿Cuál es la principal diferencia entre un entero de 32 bits y un entero de 64 bits?

### Ejercicios de Programación

20. Escribe un programa en Python que solicite al usuario una colección de números enteros. Primero, debe pedir el tamaño de la colección y luego solicitar los valores. El programa debe mostrar la colección ordenada de mayor a menor. Si el usuario introduce un número negativo, ese número no se debe incluir en el resultado final. (Ejemplo: Si la colección es [4, -1, 5, 3, 8], debe mostrar [8, 5, 4, 3] ).

21. Realice una función en Python que permita al usuario elegir cuál de las siguientes funciones visualizar:

- A)  $y = -\cos(x) + 2$
- B)  $y = x^2 - 1$
- C)  $y = -x^2 + 3x + 6$
- D)  $z = x^2 - y^2$

El programa debe solicitar un límite de graficado personalizado (rango de valores de x). Cada gráfica debe incluir los puntos críticos de la función y generar una animación que muestre cómo la gráfica se forma progresivamente. El usuario debe poder elegir entre visualizar una función o todas al mismo tiempo.

22. Usando los datos de sobrevivientes del Titanic disponibles en el siguiente [datasets/titanic.csv at master · datasciencedojo/datasets · GitHub](#), responda a los siguientes enunciados y genere gráficas para visualizar los resultados:
- ¿Cuántos pasajeros sobrevivieron y cuántos no?
  - ¿Cuántos pasajeros de cada clase sobrevivieron?
  - ¿Cuántos hombres y mujeres sobrevivieron?
  - Usando  
groupby(), calcule la edad promedio para cada sexo.
  - Calcule la tasa promedio de supervivencia para todos los pasajeros.
  - Calcule este ratio de supervivencia para todos los pasajeros menores de 25 años.

23. Considere un circuito RLC serie sin fuente externa, con

$R=4\Omega$ ,  $L=1 \text{ H}$ , y  $C=0.25 \text{ F}$ . La corriente

$$L \frac{d^2i}{dt^2} + R \frac{di}{dt} + \frac{1}{C} i = 0$$

Con condiciones iniciales:

- $i(0) = 1 \text{ A}$
- $\frac{di}{dt}(0) = 0 \text{ A/s}$

- a) Encuentre la solución analítica de la ecuación diferencial.
- b) Aproxime la solución usando el método de Euler para  $t \in [0,10]$  segundos con un paso  $h=0.01$ .
- c) Grafique ambas soluciones (analítica y aproximada) en una misma gráfica para comparar su precisión.

24. Desarrolla un programa en Python para gestionar el inventario de un supermercado usando exclusivamente la librería pandas. El sistema debe permitir agregar productos, vender productos y manejar un presupuesto inicial a través de un menú interactivo por consola. El inventario (nombre, cantidad, valor unitario) debe ser un DataFrame. Al



agregar, si el producto existe, aumenta la cantidad; si no, añade una nueva fila. Al vender, debe verificar el stock, descontar la cantidad y sumar el ingreso al presupuesto. El usuario debe poder consultar el inventario y el presupuesto en cualquier momento.

25. Desarrolla un programa en Python que encuentre el valor de  $x$  que minimiza una función continua en un intervalo. El usuario debe ingresar la función como cadena, el intervalo de búsqueda  $[a, b]$  y una tolerancia de error. Implemente el método de bisección, Newton-Raphson o gradiente descendente. El algoritmo debe detenerse cuando la derivada (o el cambio en  $x$ ) sea menor que la tolerancia o se alcance un máximo de iteraciones. Finalmente, muestre el valor de  $x$  que minimiza la función, el valor mínimo y el número de iteraciones realizadas.

# Unidad 1: Clusters en computación paralela/distribuida

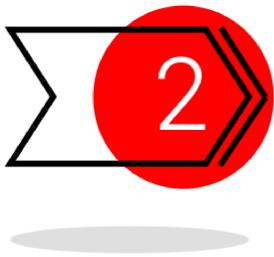
## Contenido

- Contenido de la unidad
- Introducción al HPC
- ¿Dónde corre lo que se va a programar en el curso?
- CPU & GPU & TPU
- CPU & GPU
- Actualidad del software para Big Data
- Aplicaciones del HPC

## Contenido de la unidad



Introducción a  
entornos  
HPC/HTC



Colas y  
calendarizadores



Computación  
Cloud



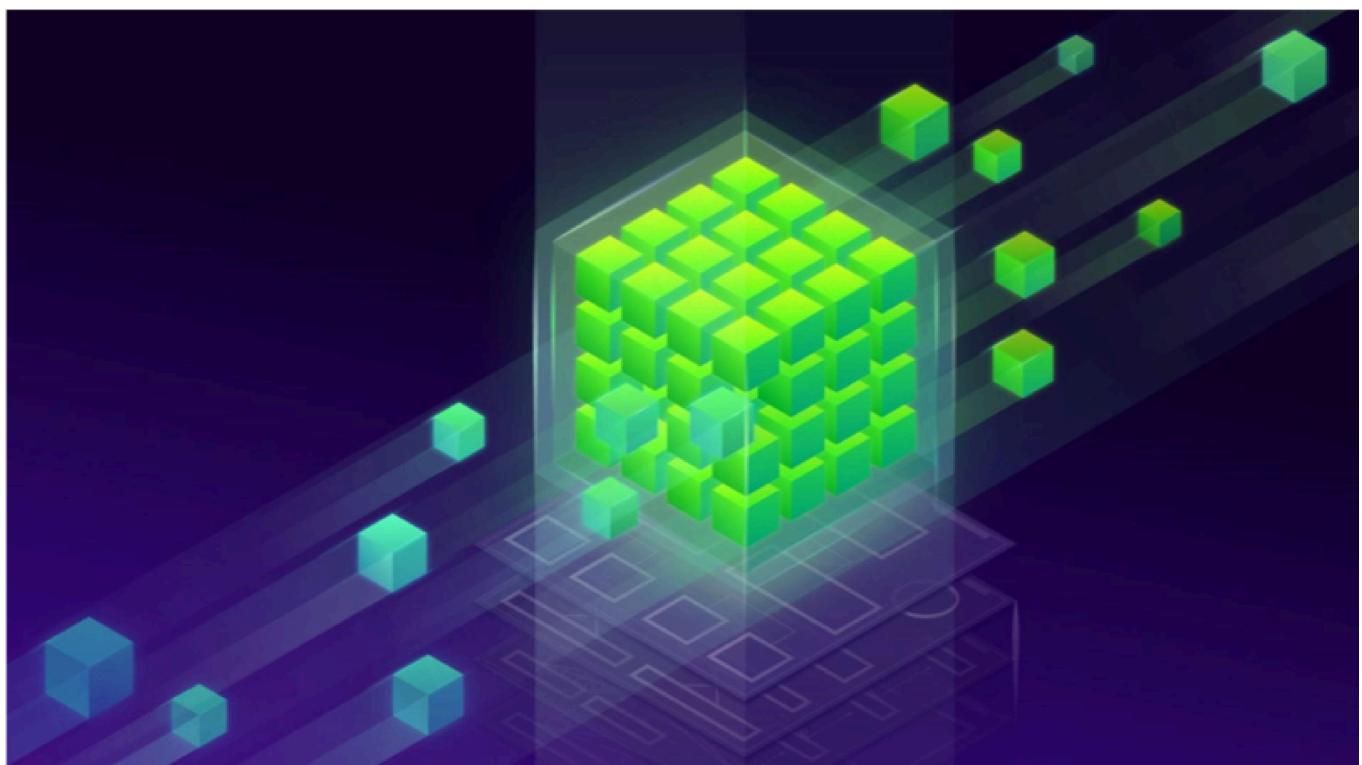
Repaso de  
Python y C para  
data science

# Introducción al HPC

**HPC:** Cualquier técnica computacional que soluciona un problema grande de forma más rápida que usando posiblemente sistemas simples.

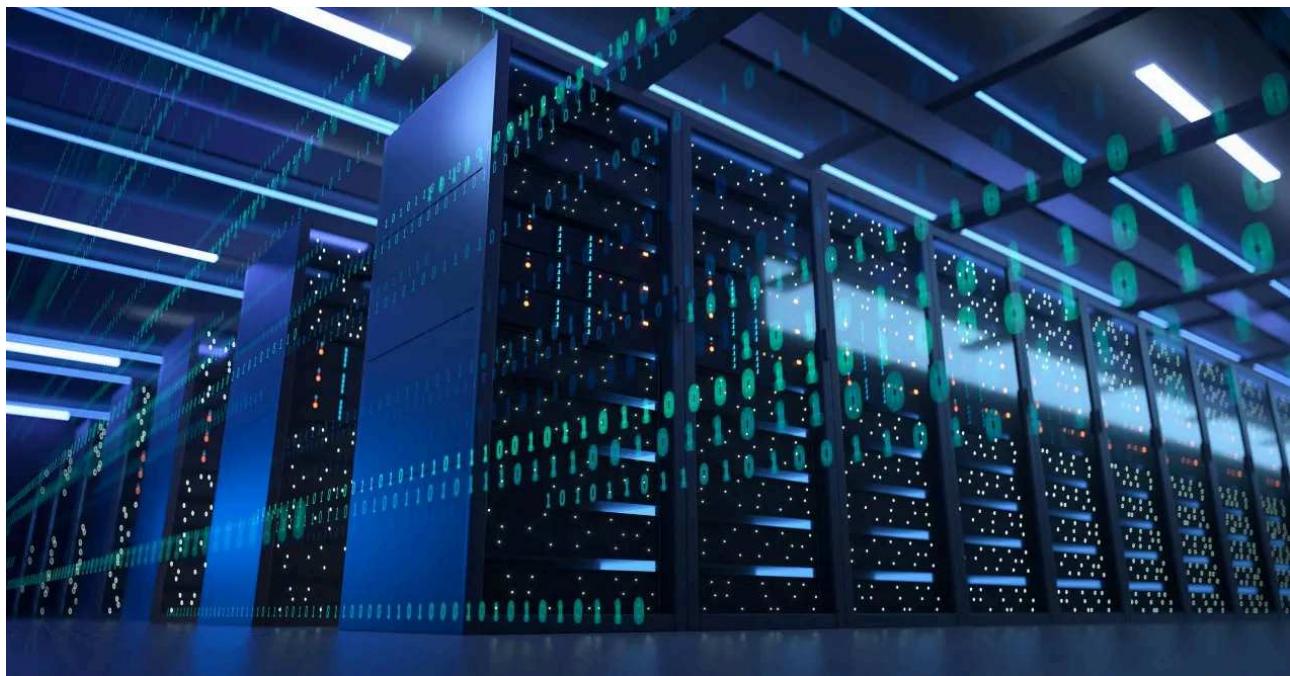
HPC ha tenido gran impacto sobre todas las áreas de ciencias computacionales e ingeniería en la academia, gobierno e industria.

Muchos problemas han sido solucionados con técnicas de **HPC** que eran imposibles de solucionar con estaciones de trabajo individuales o computadores personales.



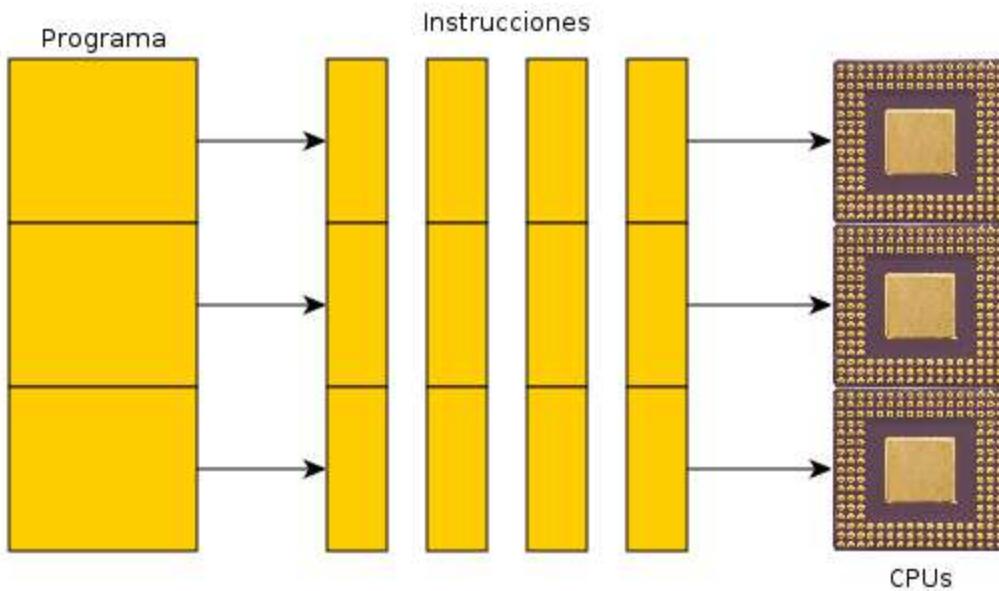
## Procesadores de alto rendimiento





## Computación paralela

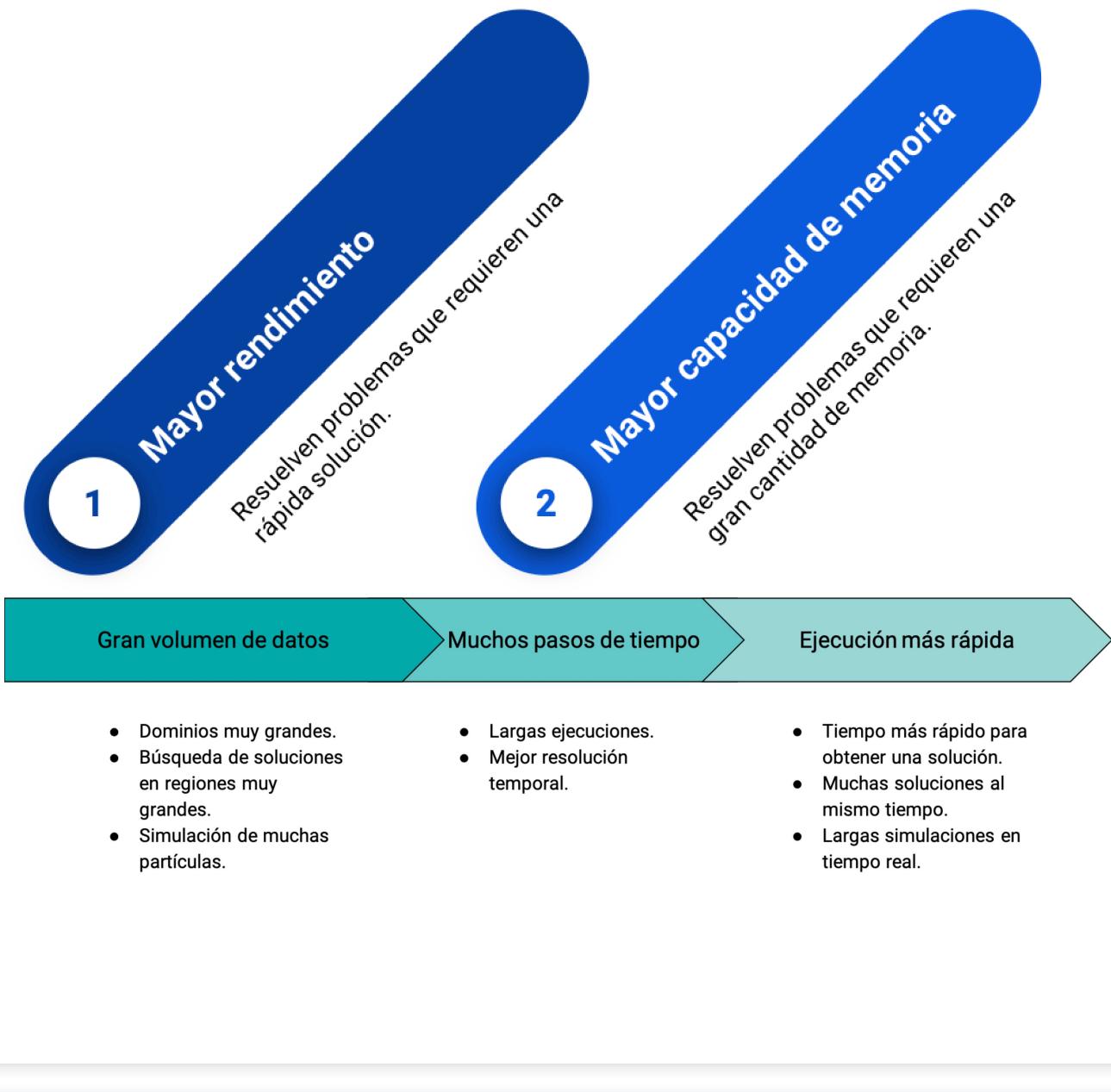
Sistemas simples con varios procesadores trabajando sobre el mismo problema



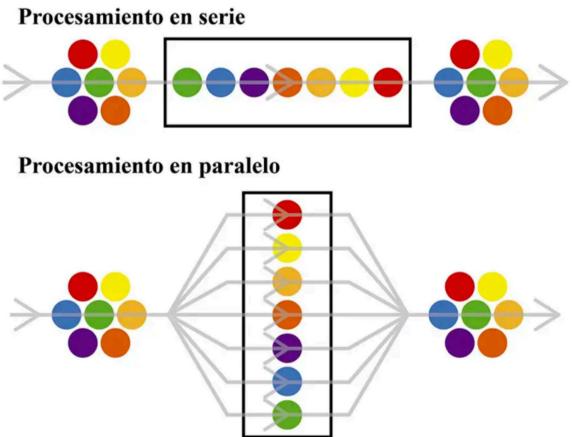
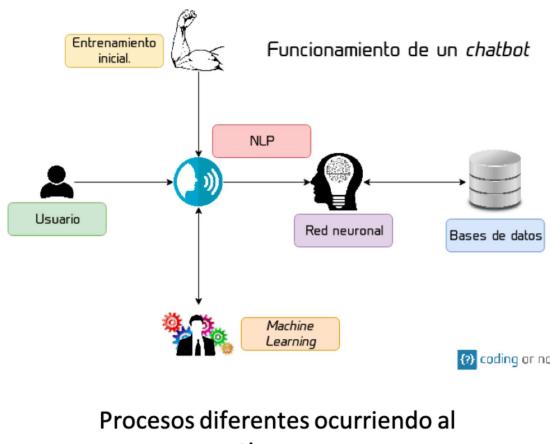
## Computación paralela Vs Computador paralelo

- Computación Paralela: el uso de múltiples computadores o procesadores trabajando en conjunto sobre una tarea común
- Computador Paralelo: un computador que contiene múltiples procesadores:

- Cada procesador trabaja sobre una sección del problema
- Los procesadores permiten intercambio de información con otros procesadores



## Computación/programación Concurrente



Procesos diferentes ocurriendo al tiempo

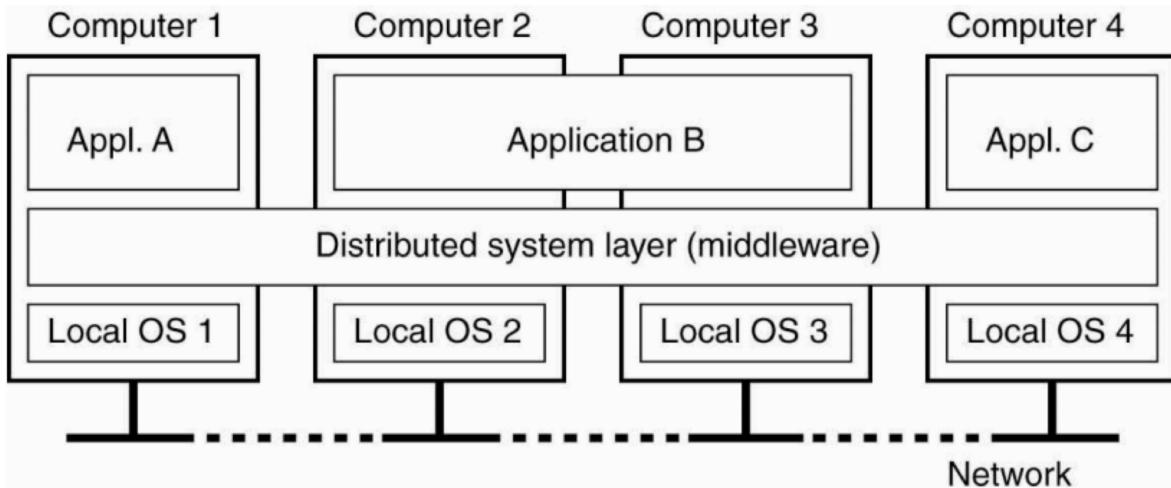
La programación concurrente se enfoca en manejar múltiples tareas al mismo tiempo (ya sea simultáneamente o dando la ilusión de simultaneidad), la programación paralela se enfoca en la división y ejecución simultánea de tareas para resolver un problema más grande en menos tiempo.

## Computación Distribuida

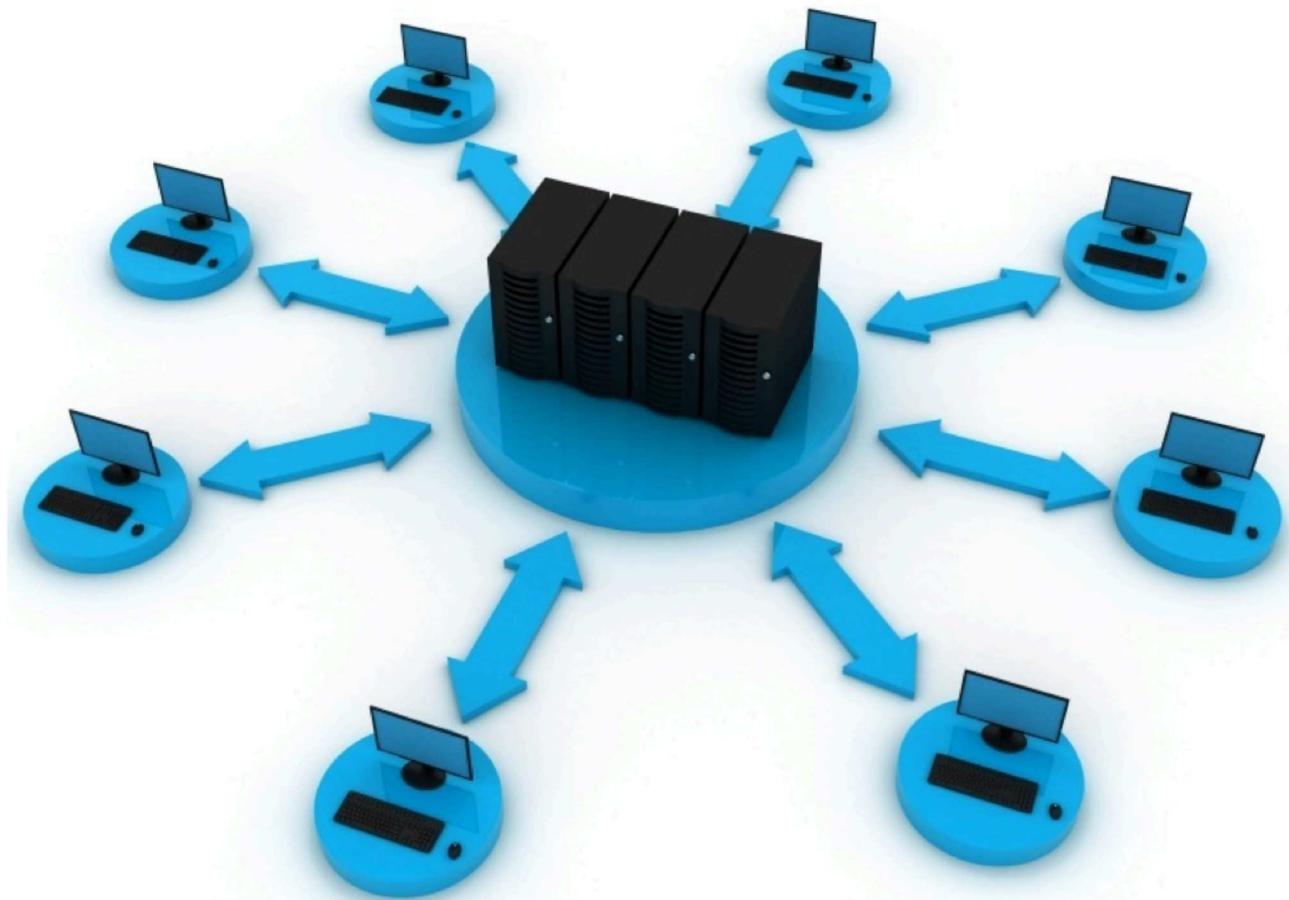
Varios sistemas acoplados por un secuenciador de trabajo sobre problemas relacionados



# Sistemas distribuidos



## Computación grid



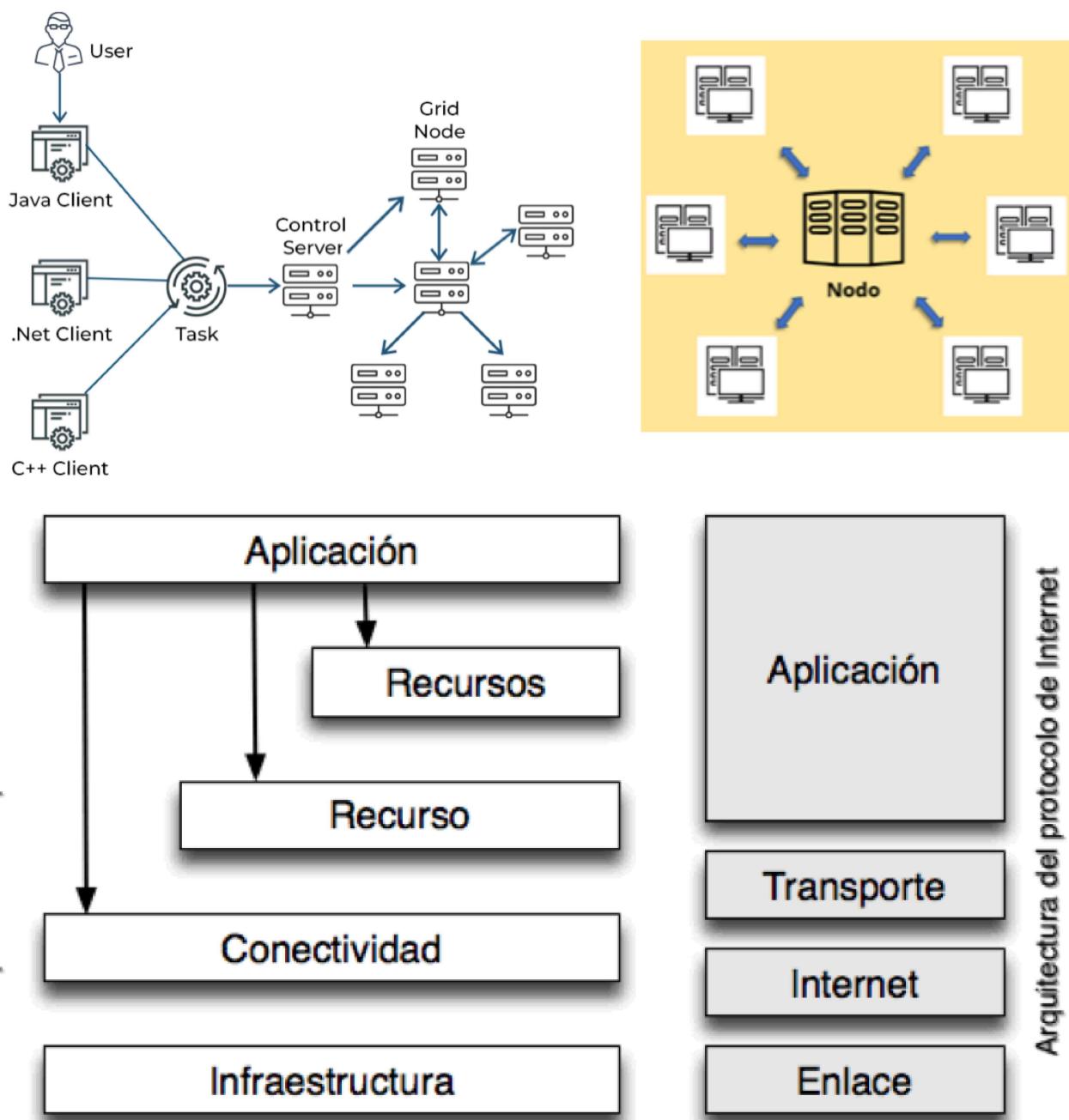
Varios sistemas acoplados por software y redes para trabajar en conjunto en problemas simples o en problemas relacionados.

Unión de Clusters de computadores, recursos computacionales, datos, grupos de investigación, científicos, etc., distribuidos geográficamente y conectados mediante redes WAN

## Funcionamiento de la computación grid

Middleware para comunicación transparente y explotación de recursos.

- El objetivo final es usar recursos remotos
- Se requieren conexiones de redes rápidas
- El grid busca el uso eficiente de los recursos
- Es esencial la seguridad centrada en: políticas de acceso, autenticación y autorización.
- Es importante estandarizar las aplicaciones grid.



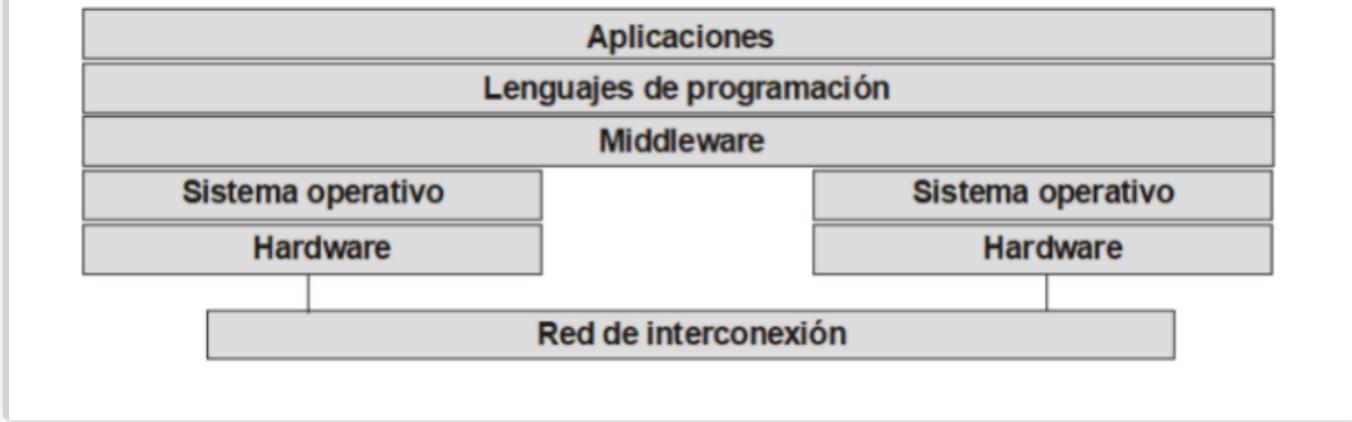
## Arquitectura Grid

- Capa de aplicación
- Capa de middleware
- Capa de recursos
- Capa de red

## Middleware

El auténtico cerebro del grid, se encarga de las siguientes funciones:

- Encontrar lugar conveniente para ejecutar la tarea solicitada por el usuario.
- Optimizar el uso de recursos que se encuentren dispersos.
- Organizar el acceso eficiente a los datos.
- Autenticar los diferentes elementos.
- Ejecutar tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar automáticamente la recuperación frente a fallos.
- Notificar el fallo, culminación de la ejecución de una tarea y sus resultados.

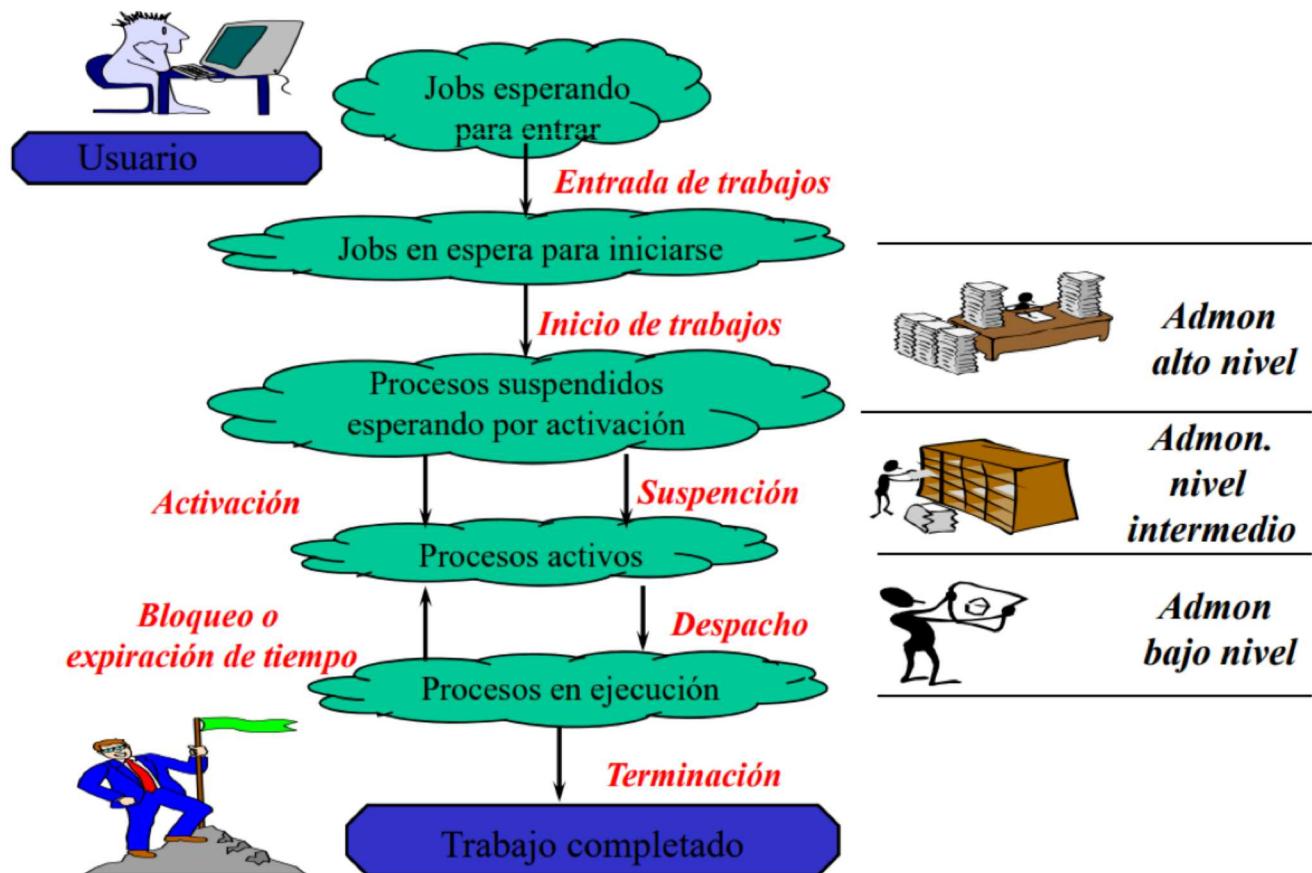


¿Dónde corre lo que se va a programar en el curso?

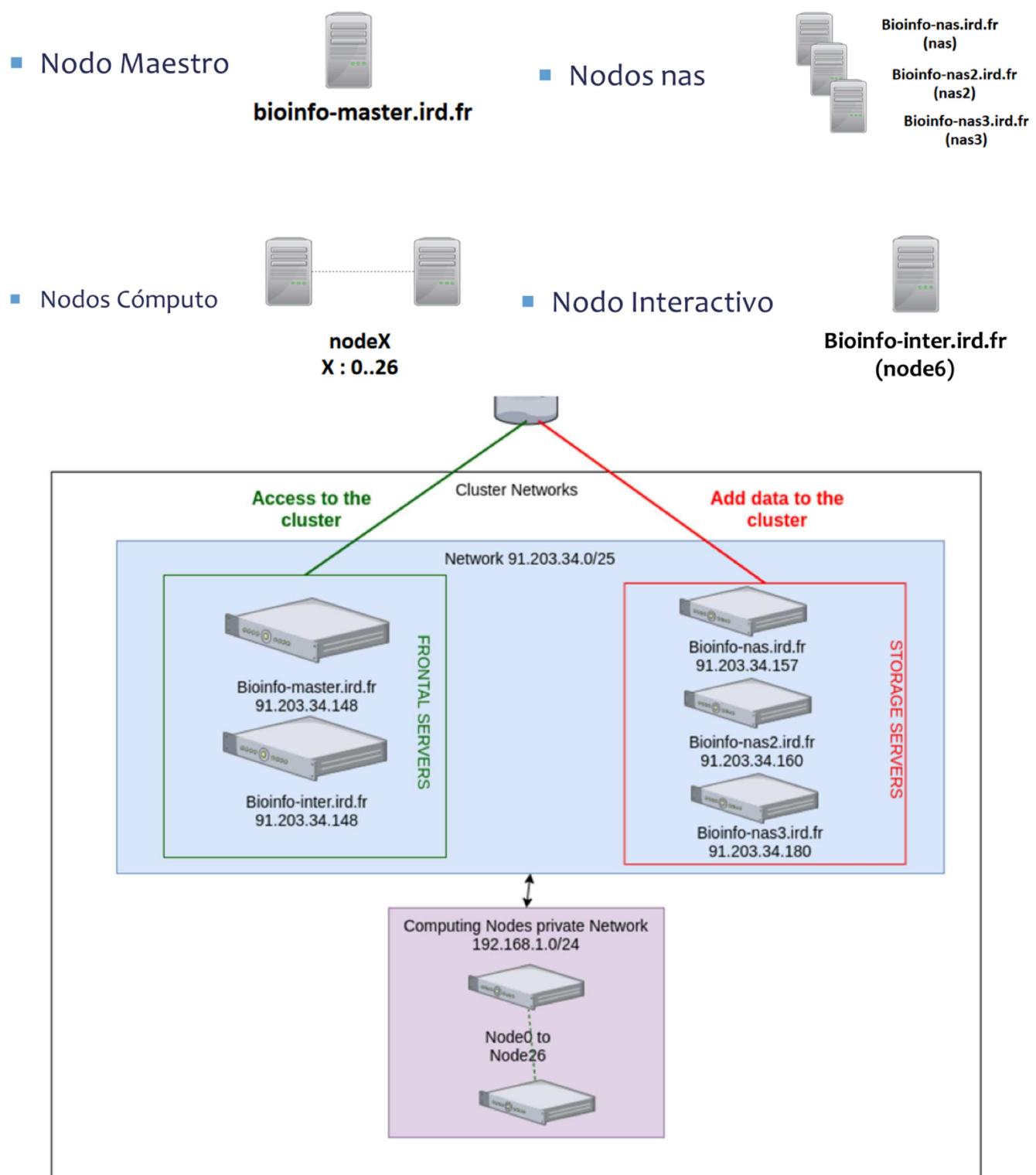
## Infraestructura: Cluster



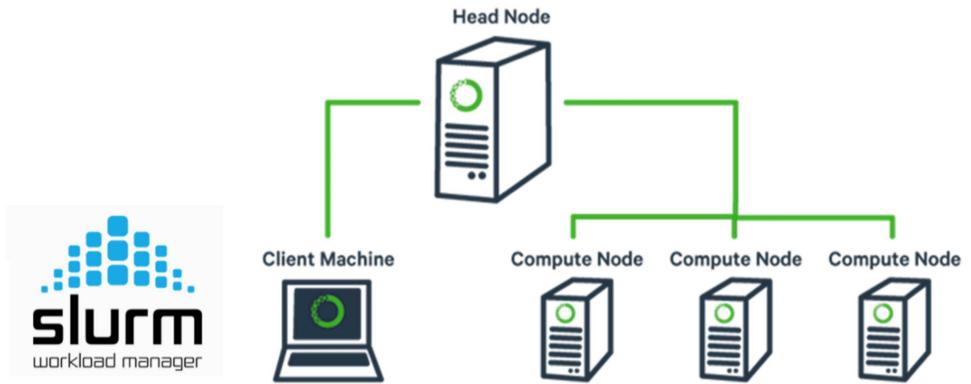
# Colas y calendarizadores



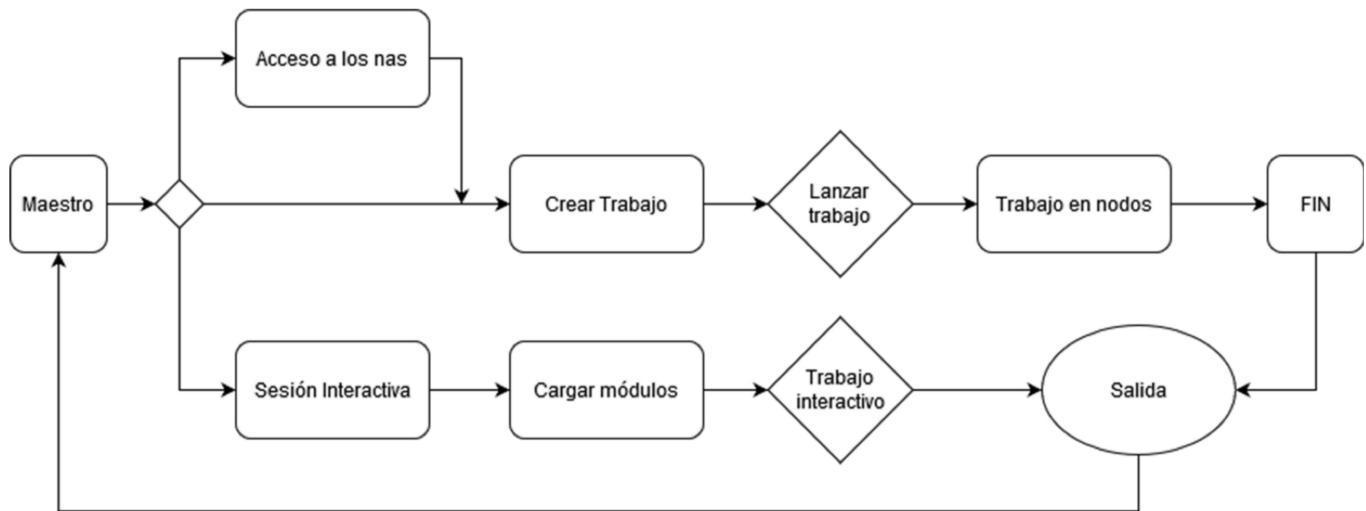
- Administrar recursos del Clúster
- Priorizar trabajos
- Limitar capacidad de cómputo
- Administrar usuarios



## Algunos calendarizadores

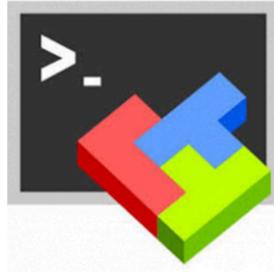


## Flujo de trabajo con calendarizadores

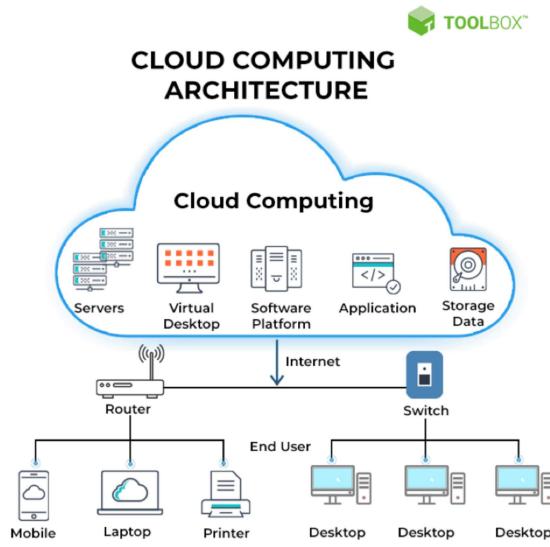


## Así se ve un job para calendarizador

```
#!/bin/bash
Define the job name
#SBATCH --job-name=test
Define the output file
#SBATCH --output=res.txt
Define the number of tasks
#SBATCH --ntasks=1
```



## Infraestructura: Cloud

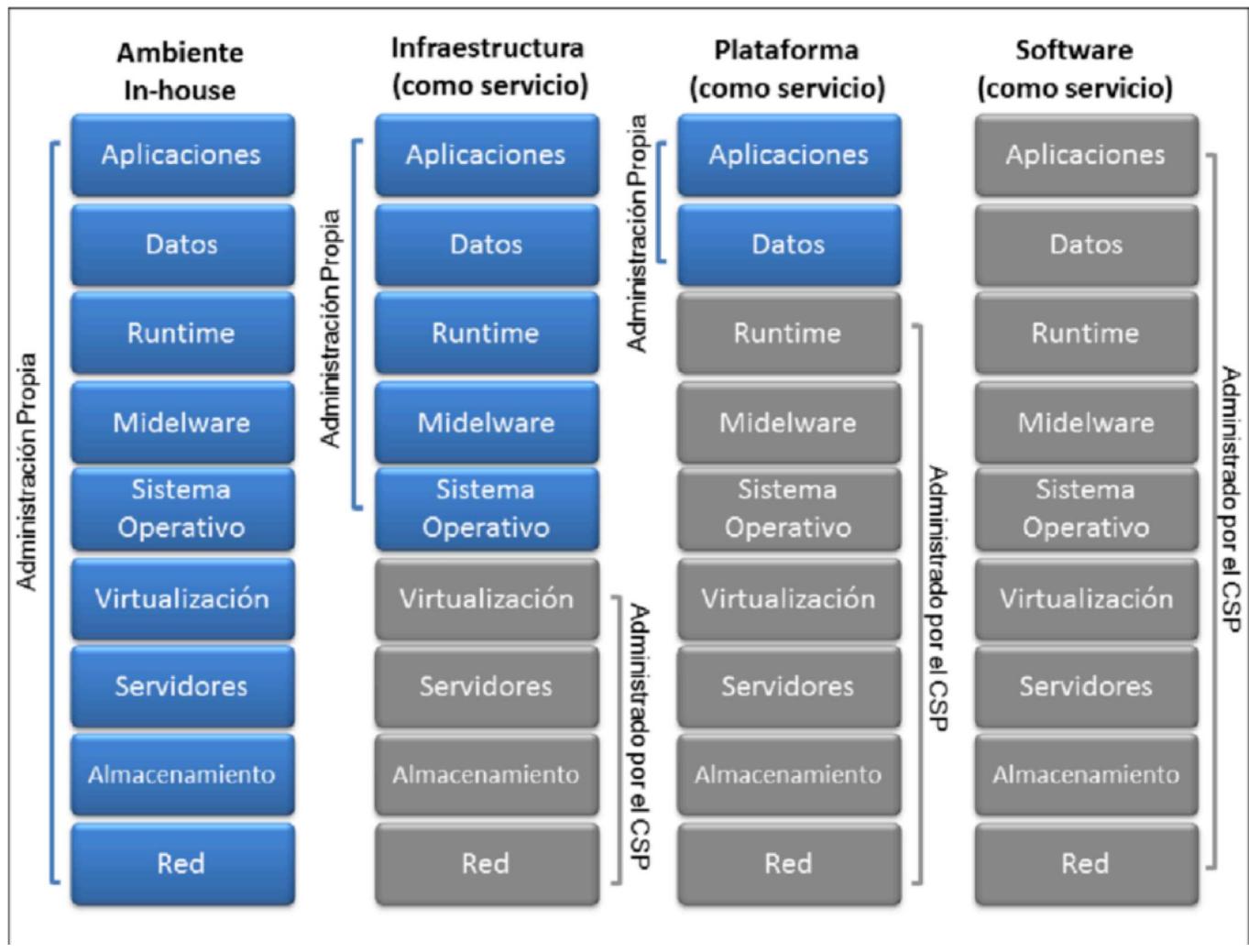


- Externalizar TI a proveedores de servicio
- Presentar transparencia al usuario final
- Asignación dinámica de recursos bajo demanda (Sin compartir) – Pago asociado
- Virtualización

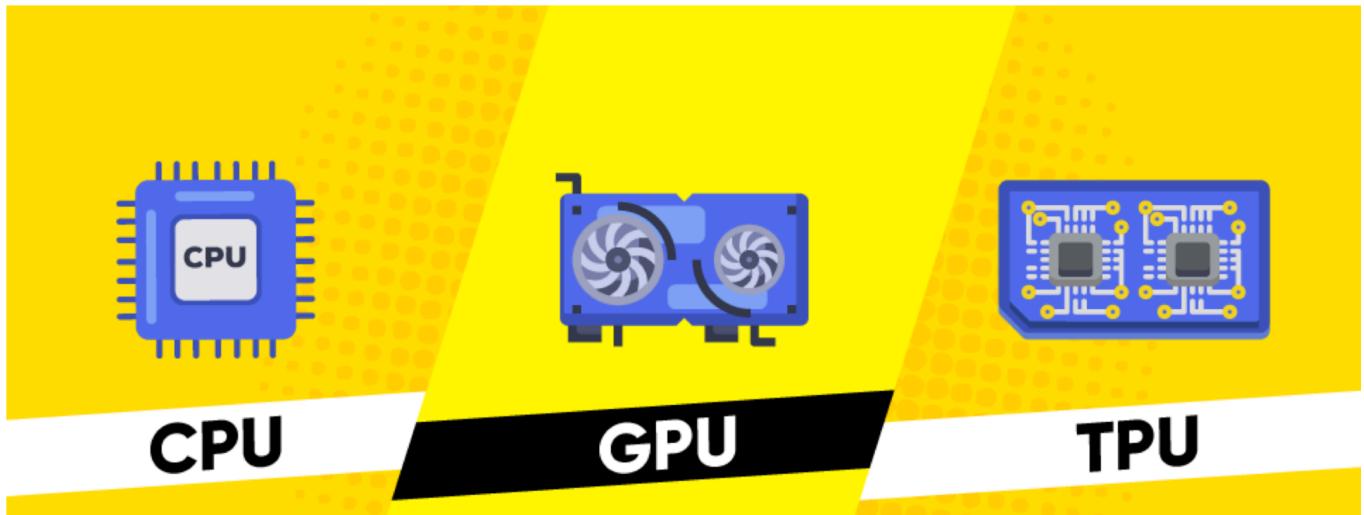
# Computación cloud

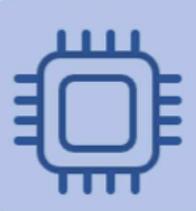


## Infraestructura tradicional Vs Cloud



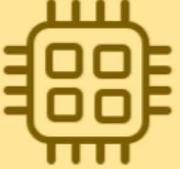
# CPU & GPU & TPU





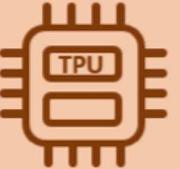
### CPU

- Small models
- Small datasets
- Useful for design space exploration



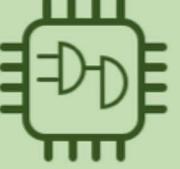
### GPU

- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL



### TPU

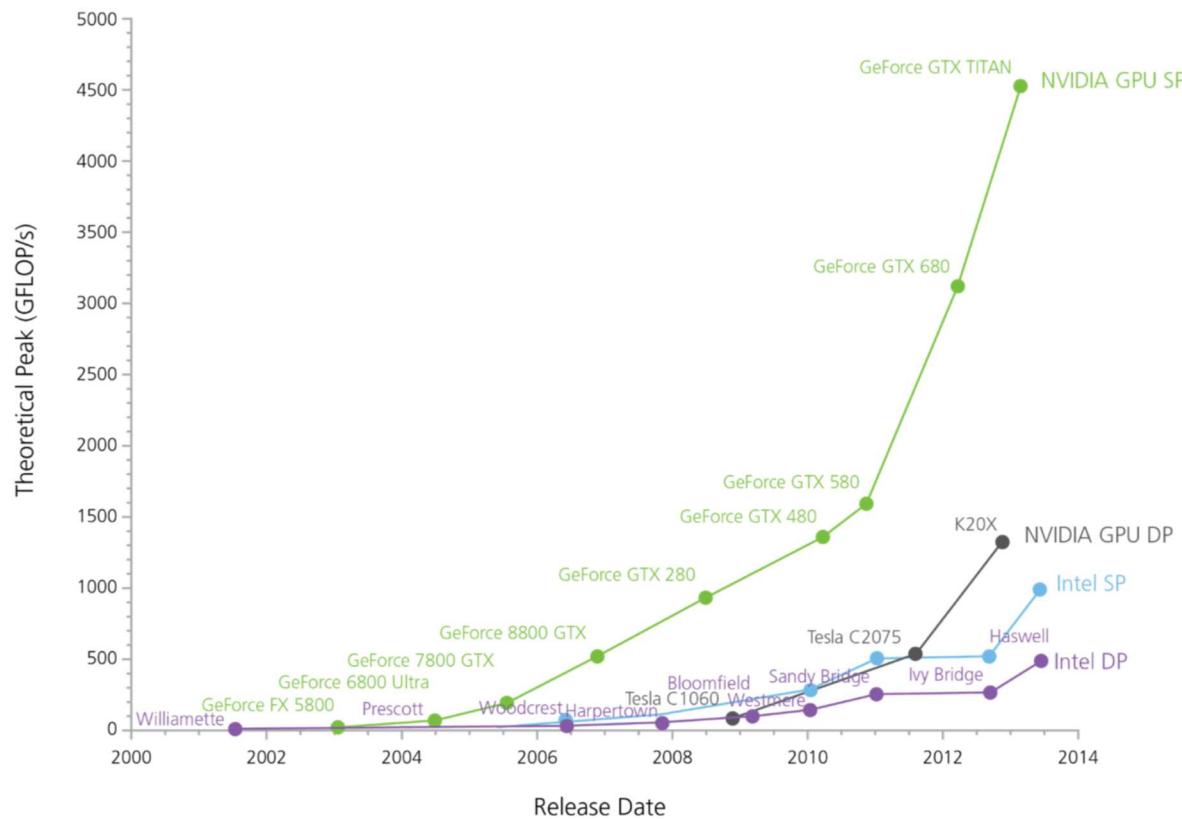
- Matrix computations
- Dense vector processing
- No custom TensorFlow operations



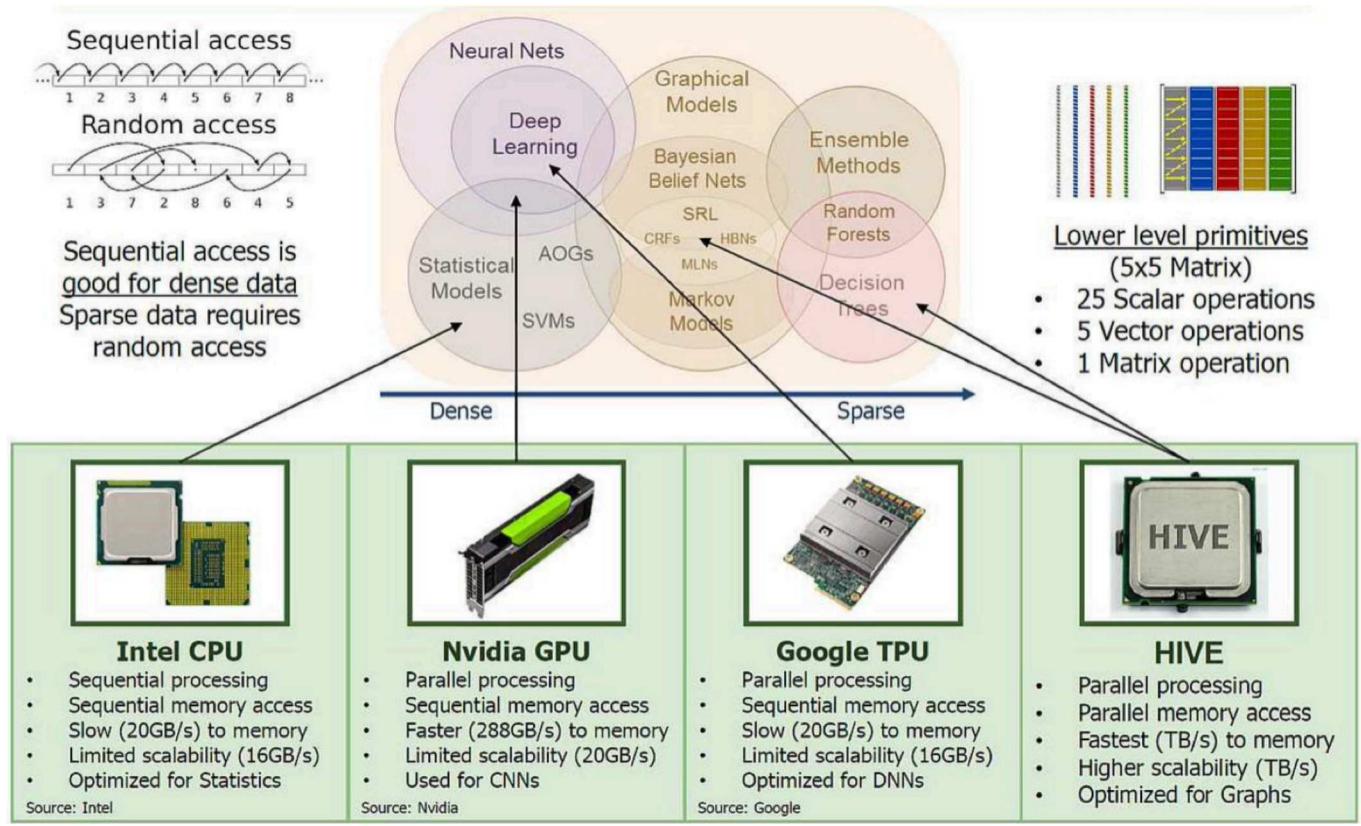
### FPGA

- Large datasets, models
- Compute intensive applications
- High performance, high perf./cost ratio

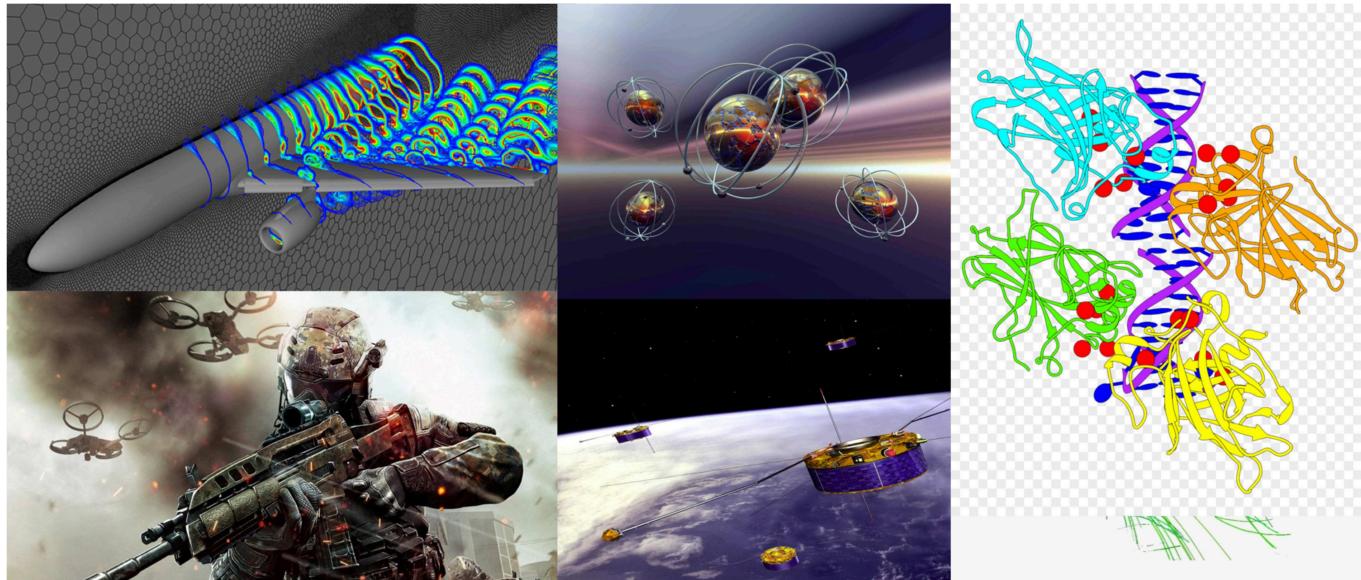
# CPU & GPU



# Actualidad del software para Big Data



## Aplicaciones del HPC



# Algebra computacional con Sympy

## Contenido

- Algebra computacional con **Sympy**
- Cálculo

Basado en [Taming math and physics using SymPy](#) y [Sympy : Symbolic Mathematics in Python](#)

## Conceptos básicos de **Sympy**

```
Importar librerías de cálculo matemático
from sympy import *
import numpy as np

Importar librería para gráficos
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Forzar una expresión como simbólica
expr = S("1/7")
print ("Tipo expresión: ", expr, type(expr))

Obtener su valor numérico
num = N("1/7")
print ("Tipo numérico: ", num, type(num))

Operaciones para Racional y Flotante de sympy
print (expr+1)
print (num+1)
```

```
Tipo expresión: 1/7 <class 'sympy.core.numbers.Rational'>
Tipo numérico: 0.142857142857143 <class 'sympy.core.numbers.Float'>
8/7
1.14285714285714
```

```
Cadena de texto sencilla
x='1/7'
print(type(x))

Expresión simbólica de Sympy, usando comando S("expresion")
y=S('1/7')
print(type(y))
```

```
<class 'str'>
<class 'sympy.core.numbers.Rational'>
```

```
Ejemplos de expresiones y operaciones con las mismas
Como racional
print(S('1/7'))

Como numérico
print(N('1/7'))

Operación como racional
print(S('1/7')+1)

Operación como numérico
print(N('1/7')+1)

Tipos racional y flotante
print(type(S('1/7')),type(N('1/7')))
```

```
1/7
0.142857142857143
8/7
1.14285714285714
<class 'sympy.core.numbers.Rational'> <class 'sympy.core.numbers.Float'>
```

```
pi es la expresión de Sympy, mientras que np.pi es la de Numpy

print(pi, "\t", type(pi))
print(pi + 1, "\t", type(pi+1))
print(N(pi+1), "\t", type(N(pi+1)))
print(np.pi, "\t", type(np.pi))
print(np.pi+1, "\t", type(np.pi+1))
```

```

pi <class 'sympy.core.numbers.Pi'>
1 + pi <class 'sympy.core.add.Add'>
4.14159265358979 <class 'sympy.core.numbers.Float'>
3.141592653589793 <class 'float'>
4.141592653589793 <class 'float'>

```

```

Poniendo a Sympy a que imprima resultados bonitos
init_printing(use_latex=True)
pi, expr

```

$(\pi, 1/7)$

```

Poniendo a Sympy a que imprima resultados NO bonitos
init_printing(use_latex=False)
pi, expr

```

$(\pi, 1/7)$

## Expresiones

```

Usando variables regulares de Python
t=2
y=3*t
y

```

6

```

Tiene el tag: raises-exception, para que aunque tiene error continue compilando el resto del código
Usando variables simbólicas sin Sympy

expr = 2*x + 3*x - sin(x) - 3*x + 42

```

```

TypeError Traceback (most recent call last)
<ipython-input-47-519f78f4039f> in <module>
 3 # Usando variables simbólicas sin Sympy
 4
----> 5 expr = 2*x + 3*x - sin(x) - 3*x + 42

TypeError: unsupported operand type(s) for -: 'str' and 'sin'

```

## Definir variables

```

Definir variables simbólicas con Sympy

Definir las variables
x,y,z = symbols("x y z")

Ahora sí, escribir la expresión
expr = 2*x + 3*x - sin(x) - 3*x + 42
expr

```

$2 \cdot x - \sin(x) + 42$

```

nu, mu = symbols("n, m")
nu*2+mu

```

$m + 2 \cdot n$

```

g, a = symbols("\gamma \alpha")
g*2+a

```

$\alpha + 2 \cdot \gamma$

```

r, p, t = symbols("r, \phi, \theta")
2*r+3*p+5*t

```

$3 \cdot \phi + 5 \cdot \theta + 2 \cdot r$

```
Revisión de tipos luego de Definir las variables x,y,z

w=3
print(type(w))
print(type(x))
```

```
<class 'int'>
<class 'sympy.core.symbol.Symbol'>
```

```
Poniendo a Sympy a que imprima resultados bonitos, de esta celda en adelante

init_printing(use_latex=True)
```

## Expandir y factorizar

```
Expandir

expand((x+3)**2)
```

$$x^2 + 6x + 9$$

```
Factorizar

factor(x**2 + 5*x + 6)
```

$$(x + 2)(x + 3)$$

## Sustituciones

```
Definir la expresión

expr = (sin(x) + cos(y)) / 2
expr
```

$$\sin(x)/2 + \cos(y)/2$$

```
Sustituir "x" por 1, e "y" por 2

expr.subs({x: 1, y:2})
```

$$\cos(2)/2 + \sin(1)/2$$

```
Obtener el valor numérico flotante

N(expr.subs({x: 1, y:2}))
```

$$0.212662074130377$$

```
Sustituir "x" por otra expresión

expr.subs({x: y**2+1})
```

$$\sin(y^2 + 1)/2 + \cos(y)/2$$

```
Sustituir por otra expresión, y la única variable "y", sustituirla por el número 2

expr.subs({x: y**2+1}).subs({y: 2})
```

$$\sin(5)/2 + \cos(2)/2$$

```
Obtener el valor numérico flotante, dos opciones

print(N(expr.subs({x: y**2+1}).subs({y: 2})))
print(expr.subs({x: y**2+1}).subs({y: 2}).n())
```

$$\begin{aligned} -0.687535555605140 \\ -0.687535555605140 \end{aligned}$$

## Prueba de igualdad

```
Definimos algunas expresiones
```

```
p = (x-5)*(x+5)
q = x**2 - 25
z = (x**2-9)/(x-3)
w = x+3
```

```
No se puede comparar directamente usando el ==
```

```
print(p == q)
print(p - q == 0)
```

False  
False

```
print(z == w)
print(z - w == 0)
```

False  
False

```
simplify(z)
```

$x + 3$

```
Ejemplo 1
```

```
print(simplify(p - q))
print(simplify(p - q) == 0)
```

0  
True

```
Ejemplo 2
```

```
print(simplify(z))
print(simplify(z)==w)
```

```
x + 3
True
```

```
Ejemplo 3

expr1 = (x**2-9)/(x-3)
expr2 = (x+3)

expr1.equals(expr2)
```

```
True
```

```
Verificación del ejemplo 3

simplify(expr1)
```

$x + 3$

## Solucionadores

```
Encontrar el valor de X
solve(5*x-10)
```

[2]

```
Encontrar el valor de y
solve(5*x-x**2-y,y)
```

$[x(5 - x)]$

```
Encontrar valores de x
s=solve(x**2 + 5*x +6)
s[0],s[1]
```

$(-3, - 2)$

```
Tipo para lista y los elementos
type(s),type(s[0])
```

(list, sympy.core.numbers.Integer)

```
Encontrar valores de x y mostrarlos
solve(x**2+5*x+6)
```

[-3, - 2]

```
Encontrar valores de x y mostrarlos además del tipo de datos
sol = solve(x**2 + 2*x - 8, x)
sol[0], sol[1], type(sol[0]), type(sol[1])
```

(-4, 2, sympy.core.numbers.Integer, sympy.core.numbers.Integer)

```
Generar una expresión usando un valor de la solución
sol[0]+x
```

$x - 4$

```
Al agregarse un flotante a la suma se vuelve flotante el resultado
print(sol[0]+1.)
type(sol[0]+1.)
```

-3.000000000000000

sympy.core.numbers.Float

```
Encontrando las famosas raíces de la ecuación cuadrática
a, b, c = symbols('a b c')
solve(a*x**2 + b*x + c, x)
```

$\left[ \frac{(-b + \sqrt{-4ac + b^2})}{2a}, \frac{(-b - \sqrt{-4ac + b^2})}{2a} \right]$

```
Encontrando sólo una de las raíces la ecuación cuadrática
sol = solve(a*x**2 + b*x + c, x)[0]
sol
```

$$(-b + \sqrt{-4ac + b^2})/2a$$

```
Substituyendo por valores y obteniendo una raíz o número complejo
sol.subs({a:1, b:2, c:3})
```

$$-1 + \sqrt{2}i$$

```
Substituyendo por valores y obteniendo un número real
sol.subs({a:1, b:5, c:6})
```

$$-2$$

Completar el cuadrado de modo que:  $x^2 - 4x + 7 = (x - h)^2 + k$  para algunas constantes  $h$  y  $k$ .

Resolvemos  $(x - h)^2 + k - (x^2 - 4x + 7) = 0$

```
Resolver
h, k = symbols('h k')

Encontrar valores de "h" y de "k"
solve((x-h)**2 + k - (x**2-4*x+7), [h,k])
```

$$[(2, 3)]$$

```
Verificar que efectivamente h==2 y que k==3
((x-2)**2 + 3).expand()
```

$$x^2 - 4x + 7$$

Solucionar el sistema de ecuaciones:

$$x^2 + y = 0$$

$$3y - x = 0$$

```
Se resuelve indicandole las dos ecuaciones
solve([x**2+y, 3*y-x])
```

```
[{x: -1/3, y: -1/9}, {x: 0, y: 0}]
```

```
Otro ejemplo de un sistema de ecuaciones
print(solve([x+y-6, x-y-2]))
```

```
{x: 4, y: 2}
```

```
Ejemplo de solución de una inecuación
print(solve(x-2>3))
```

```
(5 < x) & (x < oo)
```

```
Explorar más a profundidad la documentación
help (solve)
```

Help on function solve in module sympy.solvers.solvers:

```
solve(f, *symbols, **flags)
 Algebraically solves equations and systems of equations.
```

#### Explanation

=====

Currently supported:

- polynomial
- transcendental
- piecewise combinations of the above
- systems of linear and polynomial equations
- systems containing relational expressions

#### Examples

=====

The output varies according to the input and can be seen by example:

```
>>> from sympy import solve, Poly, Eq, Function, exp
>>> from sympy.abc import x, y, z, a, b
>>> f = Function('f')
```

Boolean or univariate Relational:

```
>>> solve(x < 3)
(-oo < x) & (x < 3)
```

To always get a list of solution mappings, use flag dict=True:

```
>>> solve(x - 3, dict=True)
[{x: 3}]
>>> sol = solve([x - 3, y - 1], dict=True)
>>> sol
[{x: 3, y: 1}]
>>> sol[0][x]
3
>>> sol[0][y]
1
```

To get a list of \*symbols\* and set of solution(s) use flag set=True:

```
>>> solve([x**2 - 3, y - 1], set=True)
([x, y], {(-sqrt(3), 1), (sqrt(3), 1)})
```

Single expression and single symbol that is in the expression:

```
>>> solve(x - y, x)
[y]
>>> solve(x - 3, x)
[3]
```

```
>>> solve(Eq(x, 3), x)
[3]
>>> solve(Poly(x - 3), x)
[3]
>>> solve(x**2 - y**2, x, set=True)
([x], {(-y,), (y,)})
>>> solve(x**4 - 1, x, set=True)
([x], {(-1,), (1,), (-I,), (I,)})
```

Single expression with no symbol that is in the expression:

```
>>> solve(3, x)
[]
>>> solve(x - 3, y)
[]
```

Single expression with no symbol given. In this case, all free \*symbols\* will be selected as potential \*symbols\* to solve for. If the equation is univariate then a list of solutions is returned; otherwise - as is the case when \*symbols\* are given as an iterable of length greater than 1 - a list of mappings will be returned:

```
>>> solve(x - 3)
[3]
>>> solve(x**2 - y**2)
[{x: -y}, {x: y}]
>>> solve(z**2*x**2 - z**2*y**2)
[{x: -y}, {x: y}, {z: 0}]
>>> solve(z**2*x - z**2*y**2)
[{x: y**2}, {z: 0}]
```

When an object other than a Symbol is given as a symbol, it is isolated algebraically and an implicit solution may be obtained. This is mostly provided as a convenience to save you from replacing the object with a Symbol and solving for that Symbol. It will only work if the specified object can be replaced with a Symbol using the subs method:

```
>>> solve(f(x) - x, f(x))
[x]
>>> solve(f(x).diff(x) - f(x) - x, f(x).diff(x))
[x + f(x)]
>>> solve(f(x).diff(x) - f(x) - x, f(x))
[-x + Derivative(f(x), x)]
>>> solve(x + exp(x)**2, exp(x), set=True)
([exp(x)], {(-sqrt(-x),), (sqrt(-x),)})
```

```
>>> from sympy import Indexed, IndexedBase, Tuple, sqrt
>>> A = IndexedBase('A')
>>> eqs = Tuple(A[1] + A[2] - 3, A[1] - A[2] + 1)
>>> solve(eqs, eqs.atoms(Indexed))
{A[1]: 1, A[2]: 2}
```

\* To solve for a symbol implicitly, use implicit=True:

```
>>> solve(x + exp(x), x)
```

```
[-LambertW(1)]
>>> solve(x + exp(x), x, implicit=True)
[-exp(x)]
```

\* It is possible to solve for anything that can be targeted with subs:

```
>>> solve(x + 2 + sqrt(3), x + 2)
[-sqrt(3)]
>>> solve((x + 2 + sqrt(3), x + 4 + y), y, x + 2)
{y: -2 + sqrt(3), x + 2: -sqrt(3)}
```

\* Nothing heroic is done in this implicit solving so you may end up with a symbol still in the solution:

```
>>> eqs = (x*y + 3*y + sqrt(3), x + 4 + y)
>>> solve(eqs, y, x + 2)
{y: -sqrt(3)/(x + 3), x + 2: -2*x/(x + 3) - 6/(x + 3) + sqrt(3)/(x + 3)}
>>> solve(eqs, y*x, x)
{x: -y - 4, x*y: -3*y - sqrt(3)}
```

\* If you attempt to solve for a number remember that the number you have obtained does not necessarily mean that the value is equivalent to the expression obtained:

```
>>> solve(sqrt(2) - 1, 1)
[sqrt(2)]
>>> solve(x - y + 1, 1) # !\ -1 is targeted, too
[x/(y - 1)]
>>> [_.subs(z, -1) for _ in solve((x - y + 1).subs(-1, z), 1)]
[-x + y]
```

\* To solve for a function within a derivative, use ``dsolve``.

Single expression and more than one symbol:

\* When there is a linear solution:

```
>>> solve(x - y**2, x, y)
[(y**2, y)]
>>> solve(x**2 - y, x, y)
[(x, x**2)]
>>> solve(x**2 - y, x, y, dict=True)
[{y: x**2}]
```

\* When undetermined coefficients are identified:

\* That are linear:

```
>>> solve((a + b)*x - b + 2, a, b)
{a: -2, b: 2}
```

\* That are nonlinear:

```
>>> solve((a + b)*x - b**2 + 2, a, b, set=True)
([a, b], {(-sqrt(2), sqrt(2)), (sqrt(2), -sqrt(2))})
```

\* If there is no linear solution, then the first successful attempt for a nonlinear solution will be returned:

```
>>> solve(x**2 - y**2, x, y, dict=True)
[{x: -y}, {x: y}]
>>> solve(x**2 - y**2/exp(x), x, y, dict=True)
[{x: 2*LambertW(-y/2)}, {x: 2*LambertW(y/2)}]
>>> solve(x**2 - y**2/exp(x), y, x)
[(-x*sqrt(exp(x)), x), (x*sqrt(exp(x)), x)]
```

Iterable of one or more of the above:

\* Involving relational or bools:

```
>>> solve([x < 3, x - 2])
Eq(x, 2)
>>> solve([x > 3, x - 2])
False
```

\* When the system is linear:

\* With a solution:

```
>>> solve([x - 3], x)
{x: 3}
>>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y)
{x: -3, y: 1}
>>> solve((x + 5*y - 2, -3*x + 6*y - 15), x, y, z)
{x: -3, y: 1}
>>> solve((x + 5*y - 2, -3*x + 6*y - z), z, x, y)
{x: 2 - 5*y, z: 21*y - 6}
```

\* Without a solution:

```
>>> solve([x + 3, x - 3])
[]
```

\* When the system is not linear:

```
>>> solve([x**2 + y - 2, y**2 - 4], x, y, set=True)
([x, y], {(-2, -2), (0, 2), (2, -2)})
```

\* If no \*symbols\* are given, all free \*symbols\* will be selected and a list of mappings returned:

```
>>> solve([x - 2, x**2 + y])
[{x: 2, y: -4}]
>>> solve([x - 2, x**2 + f(x)], {f(x), x})
[{x: 2, f(x): -4}]
```

\* If any equation does not depend on the symbol(s) given, it will be eliminated from the equation set and an answer may be given implicitly in terms of variables that were not of interest:

```
>>> solve([x - y, y - 3], x)
```

```
{x: y}
```

### \*\*Additional Examples\*\*

```solve()`` with check=True (default) will run through the symbol tags to eliminate unwanted solutions. If no assumptions are included, all possible solutions will be returned:`

```
>>> from sympy import Symbol, solve
>>> x = Symbol("x")
>>> solve(x**2 - 1)
[-1, 1]
```

By using the positive tag, only one solution will be returned:

```
>>> pos = Symbol("pos", positive=True)
>>> solve(pos**2 - 1)
[1]
```

Assumptions are not checked when ```solve()``` input involves relational or bools.

When the solutions are checked, those that make any denominator zero are automatically excluded. If you do not want to exclude such solutions, then use the `check=False` option:

```
>>> from sympy import sin, limit
>>> solve(sin(x)/x) # 0 is excluded
[pi]
```

If `check=False`, then a solution to the numerator being zero is found: $x = 0$. In this case, this is a spurious solution since $\sin(x)/x$ has the well known limit (without discontinuity) of 1 at $x = 0$:

```
>>> solve(sin(x)/x, check=False)
[0, pi]
```

In the following case, however, the limit exists and is equal to the value of $x = 0$ that is excluded when `check=True`:

```
>>> eq = x**2*(1/x - z**2/x)
>>> solve(eq, x)
[]
>>> solve(eq, x, check=False)
[0]
>>> limit(eq, x, 0, '-')
0
>>> limit(eq, x, 0, '+')
0
```

Disabling High-Order Explicit Solutions

When solving polynomial expressions, you might not want explicit solutions (which can be quite long). If the expression is univariate, ```CRootOf``` instances will be returned instead:

```
>>> solve(x**3 - x + 1)
[-1/((-1/2 - sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)) - (-1/2 -
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3, -(-1/2 +
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)/3 - 1/((-1/2 +
sqrt(3)*I/2)*(3*sqrt(69)/2 + 27/2)**(1/3)), -(3*sqrt(69)/2 +
27/2)**(1/3)/3 - 1/(3*sqrt(69)/2 + 27/2)**(1/3)]
>>> solve(x**3 - x + 1, cubics=False)
[CRootOf(x**3 - x + 1, 0),
 CRootOf(x**3 - x + 1, 1),
 CRootOf(x**3 - x + 1, 2)]
```

If the expression is multivariate, no solution might be returned:

```
>>> solve(x**3 - x + a, x, cubics=False)
[]
```

Sometimes solutions will be obtained even when a flag is False because the expression could be factored. In the following example, the equation can be factored as the product of a linear and a quadratic factor so explicit solutions (which did not require solving a cubic expression) are obtained:

```
>>> eq = x**3 + 3*x**2 + x - 1
>>> solve(eq, cubics=False)
[-1, -1 + sqrt(2), -sqrt(2) - 1]
```

Solving Equations Involving Radicals

Because of SymPy's use of the principle root, some solutions to radical equations will be missed unless check=False:

```
>>> from sympy import root
>>> eq = root(x**3 - 3*x**2, 3) + 1 - x
>>> solve(eq)
[]
>>> solve(eq, check=False)
[1/3]
```

In the above example, there is only a single solution to the equation. Other expressions will yield spurious roots which must be checked manually; roots which give a negative argument to odd-powered radicals will also need special checking:

```
>>> from sympy import real_root, S
>>> eq = root(x, 3) - root(x, 5) + S(1)/7
>>> solve(eq) # this gives 2 solutions but misses a 3rd
[CRootOf(7*x**5 - 7*x**3 + 1, 1)**15,
CRootOf(7*x**5 - 7*x**3 + 1, 2)**15]
>>> sol = solve(eq, check=False)
>>> [abs(eq.subs(x,i).n(2)) for i in sol]
[0.48, 0.e-110, 0.e-110, 0.052, 0.052]
```

The first solution is negative so ``real_root`` must be used to see that it satisfies the expression:

```
>>> abs(real_root(eq.subs(x, sol[0])).n(2))
0.e-110
```

If the roots of the equation are not real then more care will be necessary to find the roots, especially for higher order equations. Consider the following expression:

```
>>> expr = root(x, 3) - root(x, 5)
```

We will construct a known value for this expression at $x = 3$ by selecting the 1-th root for each radical:

```
>>> expr1 = root(x, 3, 1) - root(x, 5, 1)
>>> v = expr1.subs(x, -3)
```

The ``solve`` function is unable to find any exact roots to this equation:

```
>>> eq = Eq(expr, v); eq1 = Eq(expr1, v)
>>> solve(eq, check=False), solve(eq1, check=False)
([], [])
```

The function ``unrad``, however, can be used to get a form of the equation for which numerical roots can be found:

```
>>> from sympy.solvers.solvers import unrad
>>> from sympy import nroots
>>> e, (p, cov) = unrad(eq)
>>> pvals = nroots(e)
>>> inversion = solve(cov, x)[0]
>>> xvals = [inversion.subs(p, i) for i in pvals]
```

Although ``eq`` or ``eq1`` could have been used to find ``xvals``, the solution can only be verified with ``expr1``:

```
>>> z = expr - v
>>> [xi.n(chop=1e-9) for xi in xvals if abs(z.subs(x, xi).n()) < 1e-9]
[]
>>> z1 = expr1 - v
>>> [xi.n(chop=1e-9) for xi in xvals if abs(z1.subs(x, xi).n()) < 1e-9]
[-3.0]
```

Parameters

=====

f :

- a single Expr or Poly that must be zero
- an Equality
- a Relational expression
- a Boolean
- iterable of one or more of the above

symbols : (object(s) to solve for) specified as

- none given (other non-numeric objects will be used)
- single symbol
- denested list of symbols
(e.g., ``solve(f, x, y)``)
- ordered iterable of symbols
(e.g., ``solve(f, [x, y])``)

```

flags :
    dict=True (default is False)
        Return list (perhaps empty) of solution mappings.
    set=True (default is False)
        Return list of symbols and set of tuple(s) of solution(s).
    exclude=[] (default)
        Do not try to solve for any of the free symbols in exclude;
        if expressions are given, the free symbols in them will
        be extracted automatically.
    check=True (default)
        If False, do not do any testing of solutions. This can be
        useful if you want to include solutions that make any
        denominator zero.
    numerical=True (default)
        Do a fast numerical check if *f* has only one symbol.
    minimal=True (default is False)
        A very fast, minimal testing.
    warn=True (default is False)
        Show a warning if ``checksol()`` could not conclude.
    simplify=True (default)
        Simplify all but polynomials of order 3 or greater before
        returning them and (if check is not False) use the
        general simplify function on the solutions and the
        expression obtained when they are substituted into the
        function which should be zero.
    force=True (default is False)
        Make positive all symbols without assumptions regarding sign.
    rational=True (default)
        Recast Floats as Rational; if this option is not used, the
        system containing Floats may fail to solve because of issues
        with polys. If rational=None, Floats will be recast as
        rationals but the answer will be recast as Floats. If the
        flag is False then nothing will be done to the Floats.
    manual=True (default is False)
        Do not use the polys/matrix method to solve a system of
        equations, solve them one at a time as you might "manually."
    implicit=True (default is False)
        Allows ``solve`` to return a solution for a pattern in terms of
        other functions that contain that pattern; this is only
        needed if the pattern is inside of some invertible function
        like cos, exp, ect.
    particular=True (default is False)
        Instructs ``solve`` to try to find a particular solution to a linear
        system with as many zeros as possible; this is very expensive.
    quick=True (default is False)
        When using particular=True, use a fast heuristic to find a
        solution with many zeros (instead of using the very slow method
        guaranteed to find the largest number of zeros possible).
    cubics=True (default)
        Return explicit solutions when cubic expressions are encountered.
    quartics=True (default)
        Return explicit solutions when quartic expressions are encountered.
    quintics=True (default)
        Return explicit solutions (if possible) when quintic expressions
        are encountered.

```

See Also

=====

rsolve: For solving recurrence relationships

dsolve: For solving differential equations

Sympy a Python y NumpyVer [Sympy Numeric Computation](#)

```
# Definir las variables simbólicas
x,y,z = symbols("x y z")

# Tomar la primera solución
sol = solve( a*x**2 + b*x + c, x)[0]
print("Solución 1 simbólica",sol)

# Evaluar para un caso particular
s1 = N(sol.subs({a:1, b:5, c:-3}))
print("Solución 1 caso particular: ",s1)

# Obtener la raíz cuadrada con sympy
ss1 = sqrt(s1)
print (ss1, type(ss1))

# Obtener la raíz cuadrada con numpy
ns1 = np.sqrt(float(s1), dtype=complex) # Se le debe indicar para complejo
print (ns1, type(ns1))
```

```
Solución 1 simbólica (-b + sqrt(-4*a*c + b**2))/(2*a)
Solución 1 caso particular:  0.541381265149110
0.735786154496746 <class 'sympy.core.numbers.Float'>
(0.7357861544967463+0j) <class 'numpy.complex128'>
```

```
# Uso de Lambdify
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamente

# Ejemplo 1
exp1 = x**2+5*x+6
f1 = lambdify(x, exp1)
print(f1(-1))

# Ejemplo 2
exp2 = (sin(x) + x**2)/2
f2 = lambdify(x, exp2)
print(f2(10))

# Ejemplo 3
print(f1(np.array([1,2,3])))
print([f2(i) for i in [1,2,3]])
```

2
49.72798944455531
[12 20 30]
[0.9207354924039483, 2.454648713412841, 4.570560004029933]

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit f2(np.arange(100))
```

46.1 µs ± 1.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
8.58 µs ± 509 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%timeit [f1(i) for i in range(100)]
%timeit [f2(i) for i in range(100)]
```

46.3 µs ± 503 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
321 µs ± 124 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

```
# Comparar tiempos
exp2 = (sin(x) + x**2)/2
f1 = lambdify(x, exp2, "math")
f2 = lambdify(x, exp2, "numpy")

%%timeit f1(np.arange(100)) # No funciona pues no usa "Numpy"
%timeit f2(np.arange(100))
```

9.49 µs ± 1.35 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)

```
# Uso de Lambdify para una FUNCIÓN DE DOS (2) ARGUMENTOS
# Este módulo proporciona funciones convenientes para transformar expresiones SymPy
# en funciones lambda que se pueden usar para calcular valores numéricos muy rápidamente

# Definir la expresión
expr = (sin(x) + cos(y))/2

# Conversión la función con lambdify
f = lambdify([x, y], expr, "numpy")

# Evaluar de a un valor
print(f(10,1))

# Conjunto de valores
print(f([10,10],[1,1]))
```

-0.0018594025106150047
[-0.0018594 -0.0018594]

Sumatoria

Resolver la si siguiente sumatoria $sum = \sum_{i=1}^n i$

```
# Definimos los símbolos que vamos a usar
n, i = symbols('n i')

# Definimos la sumatoria
S = summation(i, (i, 1, n))

# Mostramos el resultado
print(factor(S))
```

```
n*(n + 1)/2
```

```
# Evaluamos la sumatoria para n = 3
print(S.subs(n, 3))
```

6

Derivar

```
diff(x**2)
```

$2x$

```
diff(x**2+x*y, x)
```

$2x + y$

```
diff(x**2+x*y, y)
```

x

```
diff(exp(x))
```

e^x

```
# La diferenciación conoce algunas reglas (por ejemplo, la regla del producto)
diff(x**2*sin(x))
```

$x^2\cos(x) + 2x\sin(x)$

Integral

```
# Integral indefinida
integrate(x**2)
```

$x^3/3$

```
# Ejemplo 1 Integral definida
#integrate(expr, (variable,desde,hasta))
integrate(x**2, (x,0,1))
```

$1/3$

```
# Ejemplo 2 Integral definida
integrate(x**2, (x,-1,3))
```

$28/3$

Recta tangente a una función

La recta tangente a la función $f(x)$ en $x = x_0$ es la linea que pasa a través del punto $(x_0, f(x_0))$ y tiene la misma pendiente que la función en ese punto. La recta tangente a la función $f(x)$ en el punto $x = x_0$ está descrita por la ecuación \$

$$T_1(x) = f(x_0) + f'(x_0)(x - x_0)$$$

¿Cuál es la ecuación de la recta tangente a $f(x) = \frac{1}{2}x^2$ en $x_0 = 1$?

```
# Crear las variables
x, f = symbols("x f")
```

```
# Crear la función
f = 1./2*x**2
print ('f(x)\t=', f)
```

$f(x) = 0.5*x^2$

```
# Encontrar la derivada
df = diff(f,x)
print ("f'(x)\t=", df)
```

$f'(x) = 1.0*x$

```
# Encontrar la recta tangente a la función
x0 = 1
T_1 = f.subs({x:x0}) + df.subs({x:x0}) * (x - x0)
print ("T1(x)\t=", T_1)
```

$T1(x) = 1.0*x - 0.5$

```
# Comprobar que el valor en el eje "y" de la función y la recta tangente son iguales
print(f.subs({x:1}))
T_1.subs({x:1}) - f.subs({x:1})
```

0.500000000000000

0

```
# Comprobar que el valor de la pendiente de la función y la recta tangente son iguales
print(diff(f,x).subs({x:1}))
diff(T_1,x).subs({x:1}) - diff(f,x).subs({x:1})
```

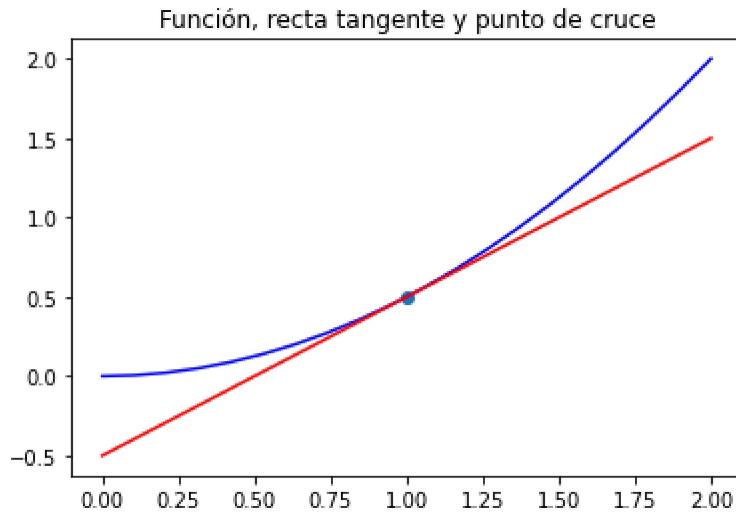
1.00000000000000

0

```
# Obtener funciones de Python a partir de expresiones
fp = lambdify(x, f, "numpy")
tp = lambdify(x, T_1, "numpy")
```

```
# Graficar
xr = np.linspace(0,2,20)
plt.plot(xr, fp(xr), color="blue") # función
plt.plot(xr, tp(xr), color="red") # recta tangente
plt.scatter(x0, fp(x0)) # punto de cruce, x0 == 1
plt.title("Función, recta tangente y punto de cruce")
```

Text(0.5, 1.0, 'Función, recta tangente y punto de cruce')



Ecuaciones diferenciales

Resolver $\frac{dy}{dt} = y(t) + t$

```
# Crear variables
t, C1 = symbols("t C1")

# Crear una función no definida con el parámetro cls=Function
y = symbols("y", cls=Function)

# Crear una expresión con la función no definida y la variable
dydt = y(t)+t

# Crear la ecuación a resolver
eq = dydt-diff(y(t),t)
eq
```

$$t + y(t) - \frac{d}{dt}y(t)$$

```
# Resolver. Con dsolve se resuelven ecuaciones diferenciales
yt = dsolve(eq, y(t))
yt
```

$$y(t) = (C_1 + (-t - 1)e^{-t})e^t$$

```
# Obtener el resultado
yt.rhs
```

$$(C_1 + (-t - 1)e^{-t})e^t$$

```
# Verificar la solución, forma 1
simplify(dydt.subs({y(t): yt.rhs}) - diff(yt.rhs,t))
```

0

```
# Verificar la solución, forma 2
expr1 = dydt.subs({y(t): yt.rhs})
expr2 = diff(yt.rhs,t)

expr1.equals(expr2)
```

True

```
# Usando condición inicial y(0) = 2, obtener C1
eq1 = Eq(yt.rhs.subs({t:0}).evalf(), 2.) # Eq(f,g) -> f=g
sol = solve([eq1], [C1])
sol
```

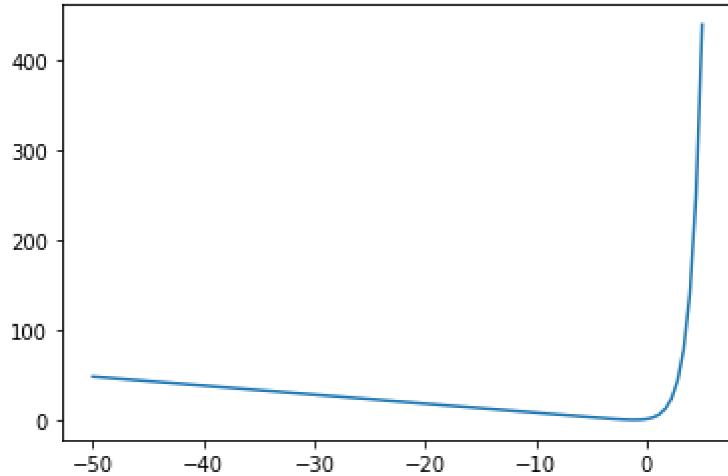
$$\{C_1 : 3.0\}$$

```
# Definir función para C1=3, con la variable "t"
C1_val = 3
fY = lambdify(t, yt.rhs.subs({C1: C1_val}), "numpy") # C1_val == 3
print(yt.rhs.subs({C1: C1_val}))

# Evaluar en un sólo valor de tiempo
print(fY(0))

# Evaluar en varios valores de tiempo y graficar
t_vals = np.linspace(-50,5,100)
plt.figure()
plt.plot(t_vals, fY(t_vals))
plt.show()
```

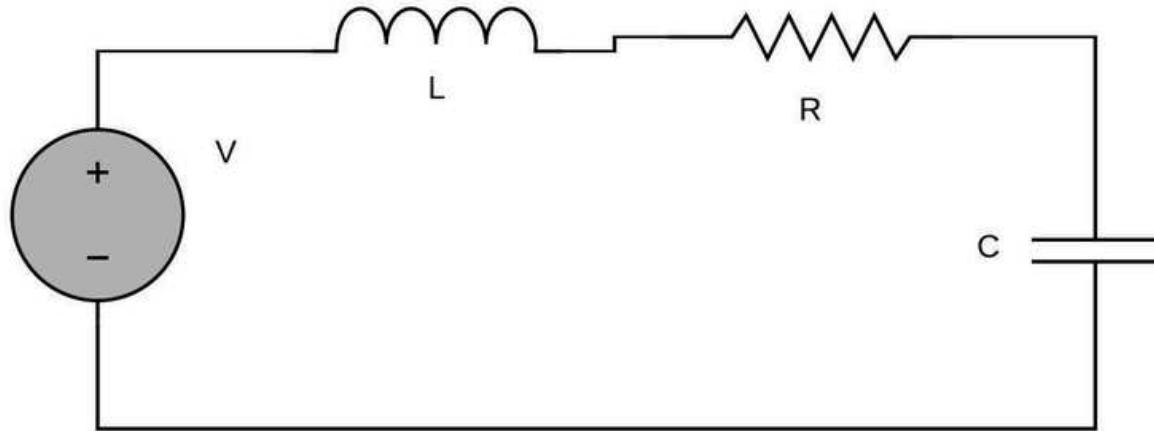
((-t - 1)*exp(-t) + 3)*exp(t)
2.0



Circuito RLC

Modelo:

$$\begin{aligned}\frac{di}{dt} &= -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi \\ \frac{dv}{dt} &= \frac{1}{C}i(t)\end{aligned}$$



```
# Definir simbolos y funciones
t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)
didt=i.diff(t)
dvdt=v.diff(t)
```

```
# Primera expresión
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -Ri(t)/L + vi/L - v(t)/L$$

```
# Segunda expresión
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = i(t)/C$$

```
# Sustitución, suponiendo valores conocidos
expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
# Sustitución, suponiendo valores conocidos
expr2=expr2.subs(C,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
# Resolver usando dsolve. Se encontrará i(t) y v(t)
s=dsolve([expr1,expr2])
s
```

$$i(t) = -(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2), \quad v(t) = C_1e^{-t/2}\cos(\sqrt{3}t/2) - C_2e^{-t/2}\sin(\sqrt{3}t/2) + \sin^2(\sqrt{3}t/2) + \cos^2(\sqrt{3}t/2)$$

```
# Sustituir para un i(0)=0.1 y v(0)=0.1
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1) #i corriente
print(eq1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1) #v voltaje
print(eq2)
sol = solve([eq1,eq2], [C1,C2])
sol
```

$$\begin{aligned} & \text{Eq}(-0.5*C1 - 0.866025403784439*C2, 0.1) \\ & \text{Eq}(C1 + 1.0, 0.1) \end{aligned}$$

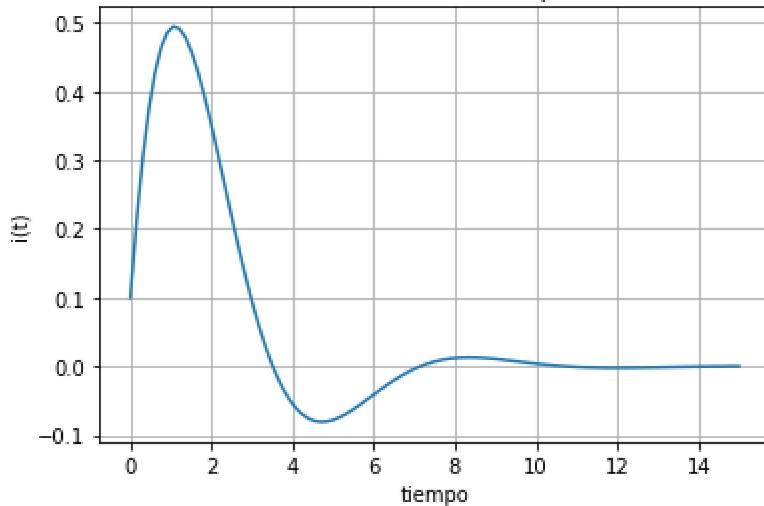
$$\{C_1 : -0.9, \quad C_2 : 0.404145188432738\}$$

```
# Ecuación corriente
e1 = lambdify(t,s[0].rhs.subs(sol), 'numpy')
# Ecuación voltaje
e2 = lambdify(t,s[1].rhs.subs(sol), 'numpy')
```

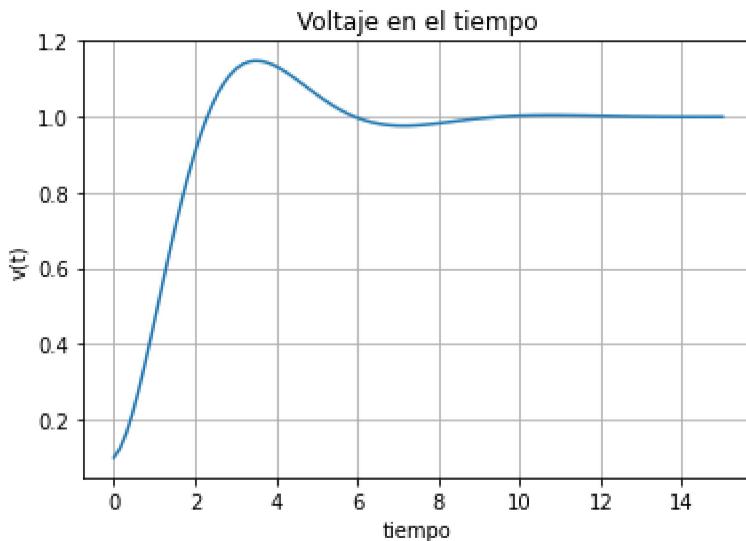
```
# Vector de valores en el tiempo
t_vals = np.linspace(0,15,100)

# Graficar
plt.figure()
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
plt.title("Corriente en el tiempo")
plt.show()
```

Corriente en el tiempo



```
# Graficar
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
plt.title("Voltaje en el tiempo")
plt.show()
```



Oscilador armónico amortiguado

```
# Crear la ecuación del oscilador
t, w = symbols("t \omega_0", positive=True)
d = symbols("\xi", real=True)
x = Function("x", real=True)
eq = w**2*x(t) + 2*w*d*x(t).diff(t) + x(t).diff(t,t)
eq
```

$$\omega_0^2 x(t) + 2\omega_0 \xi \frac{dx}{dt} x(t) + \frac{d^2}{dt^2} x(t)$$

Evaluar y resolver para cuando $\omega = 0$ y $\xi = 1/10$

```
# Evaluar para las condiciones iniciales
eq = eq.subs(w, 1).subs(d, Rational(1,10))

# Resolver la ecuación diferencial
sol = dsolve(eq,x(t))
sol
```

$$x(t) = (C_1 \sin(3\sqrt{11}t/10) + C_2 \cos(3\sqrt{11}t/10)) e^{-t/10}$$

```
# Usando estas condiciones iniciales x(0) = 1, y para dx(0)/dt = 0
C1, C2 = symbols("C1, C2")
const = solve([sol.rhs.subs(t,0) - 1, sol.rhs.diff(t).subs(t, 0)], [C1,C2])
print ("const =", const)

# Reemplazar C1 y C2 en la solución
initvalue_solution = sol.rhs.subs(C1, const[C1]).subs(C2, const[C2])
print ("Aplicando las condiciones iniciales \nx(t) = ")
initvalue_solution
```

```
const = {C1: sqrt(11)/33, C2: 1}
Aplicando las condiciones iniciales
x(t) =
```

$$(\sqrt{11} \sin(3\sqrt{11}t/10)/33 + \cos(3\sqrt{11}t/10)) e^{-t/10}$$

```
# Graficar
T = np.linspace(0, 10)
F = lambdify(t, initvalue_solution, "numpy")
dFdT = lambdify(t, initvalue_solution.diff(), "numpy")
plt.plot(T, F(T))
plt.plot(T, dFdT(T))
plt.legend(("f(t)", "df(t)/dt"))
plt.xlabel("t")
plt.show()
```

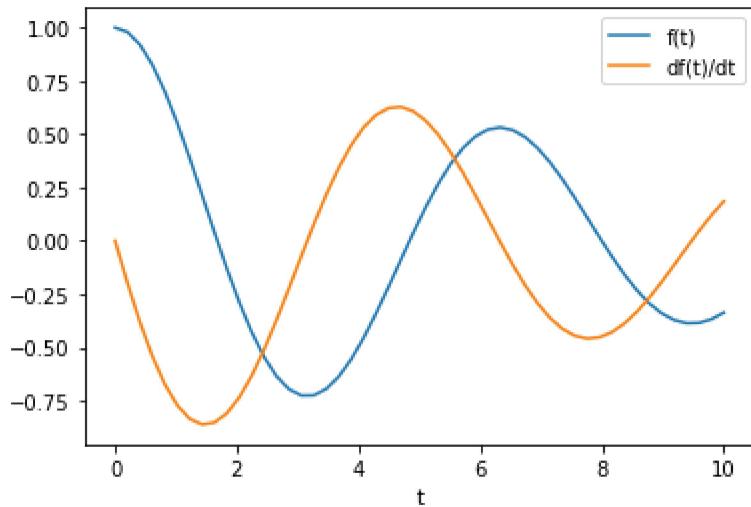
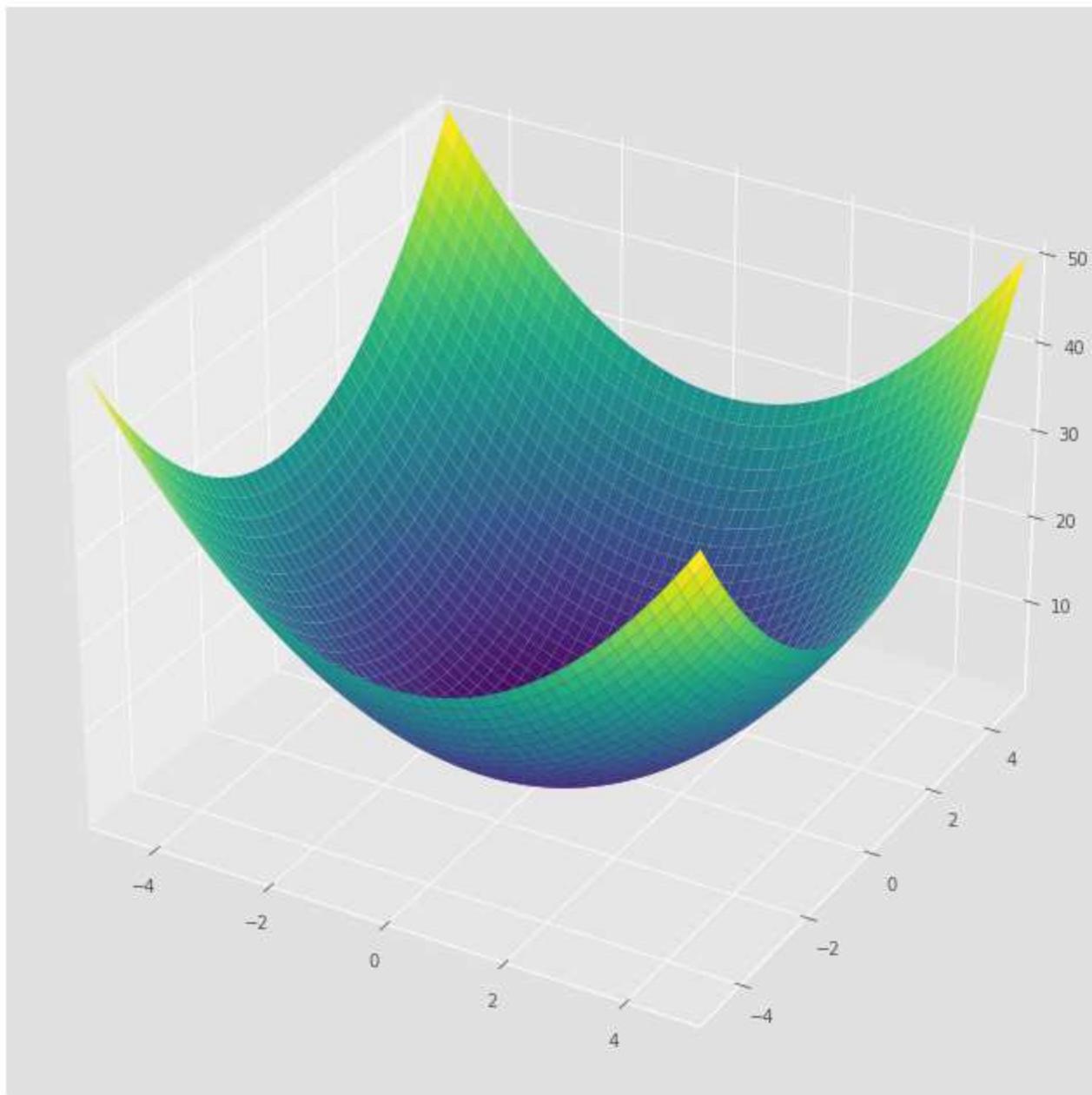


Gráfico en 3D

```
from sympy import symbols
from sympy.plotting import plot3d
x, y = symbols('x y')
```

```
plot3d(x**2+y**2, (x, -5, 5), (y, -5, 5))
```



```
<sympy.plotting.plot.Plot at 0x7f0ced2dd760>
```

Calculo del mínimo en una variable de forma analítica

```
x,y =symbols("x,y")
y=x**2+5*x+6
fp = lambdify(x, y, "numpy")
```

y

$$x^2 + 5x + 6$$

df=diff(y)

df

$$2x + 5$$

pc=solve(df)

N(pc[0])

$$-2.5$$

d2f=diff(df)

d2f

$$2$$

cy=y.subs({x:N(pc[0])})

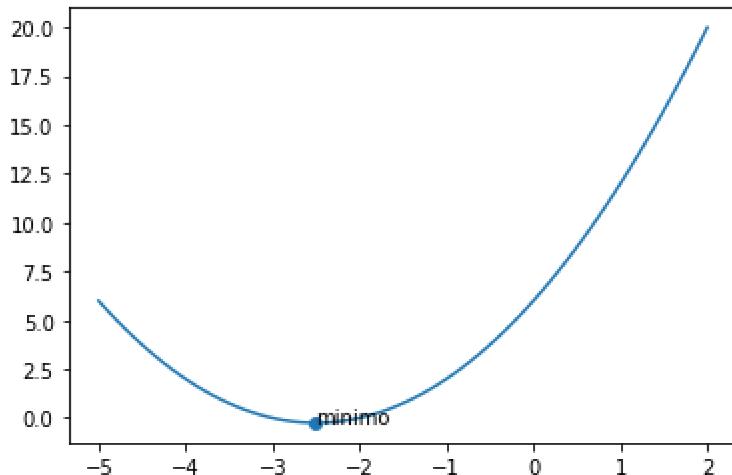
cy

$$-0.25$$

z=np.linspace(-5,2,100)

```
plt.plot(z,fp(z))
plt.scatter(pc[0],fp(pc[0]))
plt.text(pc[0],fp(pc[0]),'minimo')
```

```
Text(-5/2, -1/4, 'minimo')
```



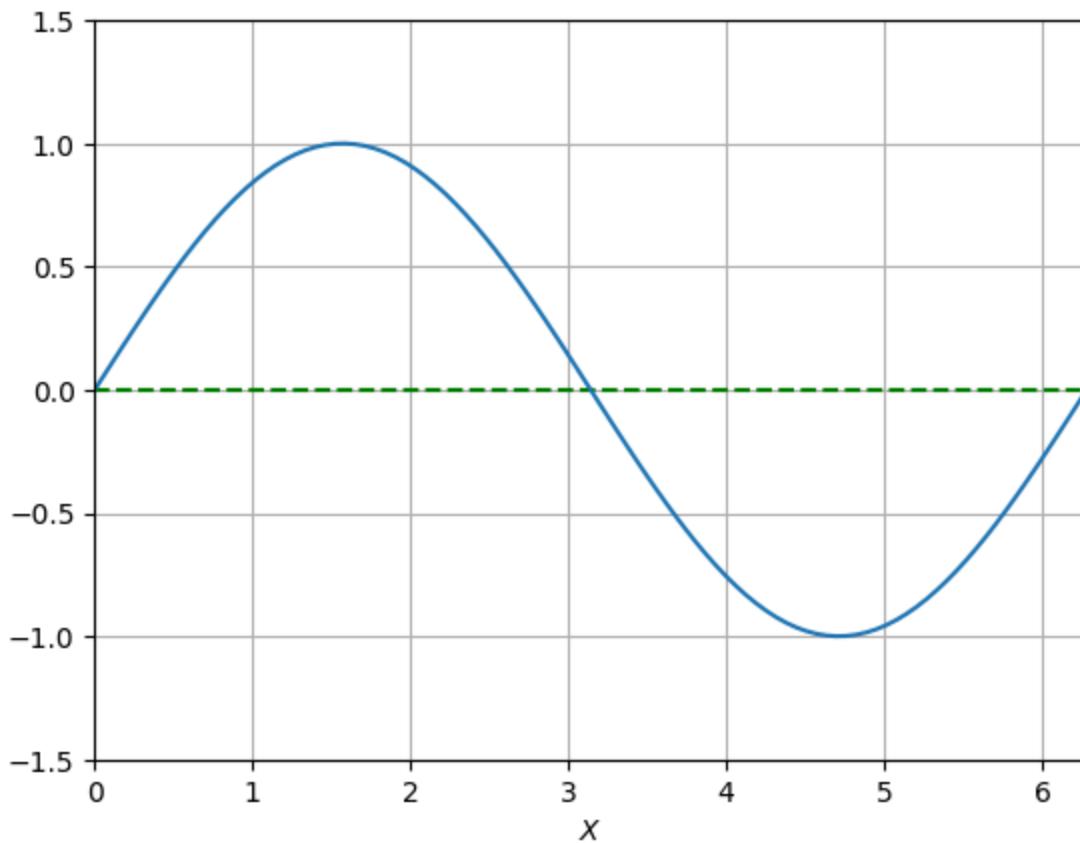
Graficas dinámicas

```
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML;
rc('animation', html='html5');
```

```
# Plantilla sobre la cual se realiza el gráfico
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

fig, ax = plt.subplots()
ax.set_xlabel('$X$')
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.5, 1.5)
ax.grid(True)
plt.plot([0, 2*np.pi], [0, 0], '--', color='green')
plt.plot(x,y)

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a
linea, = ax.plot([],[],'o',color = 'r', label = 'Curva sin(x)')
```



```
# Realizamos una función que grafique el punto específico uno a uno

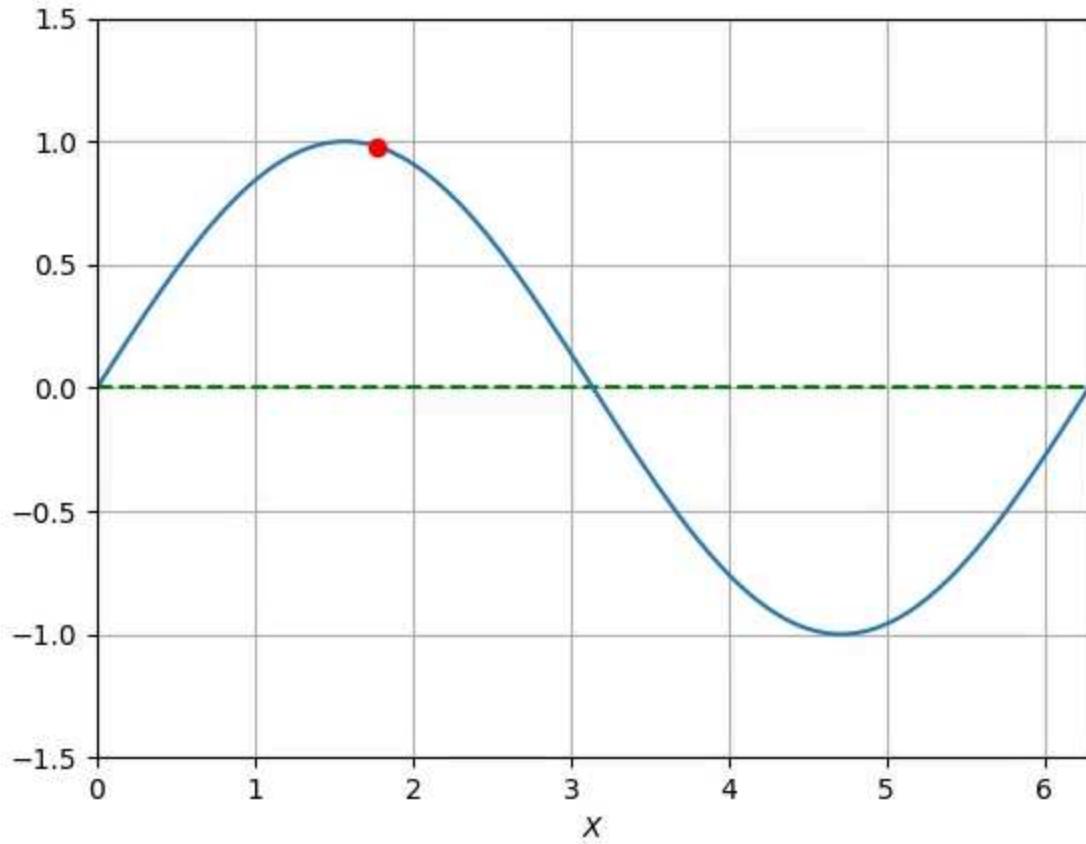
frames = 100;
def graficar(i):

    x = np.linspace(0, 2 * np.pi, 100)
    y = np.sin(x)

    linea.set_data(x[i],y[i])
    return (linea,)
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=100)
anim
```

```
<ipython-input-56-d6bd03730612>:7: MatplotlibDeprecationWarning: Setting data with a n
    linea.set_data(x[i],y[i])
```



Enlace de interés para animación:

[Más sobre animaciones](#)

Automatizaciones

Contenido

- Automatizaciones
- Guardar mapas
- Envío de correos con python
- Dash

Con Python podemos llegar a realizar desde operaciones sencillas de datos hasta visualizaciones complejas o automatizaciones de procesos.

En Python hay muchas librerías y frameworks creados por la comunidad para resolver muchos procesos. Así que si tienes un proceso que quieras realizar, busca si ya existe una librería con la que puedas facilitarte la programación y en unas cuantas líneas de código.



```
Requirement already satisfied: folium in /usr/local/lib/python3.10/dist-packages (0.14
Requirement already satisfied: branca>=0.6.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
```

```
!wget https://raw.githubusercontent.com/python-visualization/folium-example-data/main/
```

```
--2024-07-10 17:23:54-- https://raw.githubusercontent.com/python-visualization/folium
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 18
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:4
HTTP request sent, awaiting response... 200 OK
Length: 87688 (86K) [text/plain]
Saving to: ‘us_states.json.1’
```

```
us_states.json.1      0%[                               ]      0  ---KB/s
us_states.json.1    100%[=====] 85.63K  ---KB/s    in 0.02s
```

```
2024-07-10 17:23:54 (3.79 MB/s) - ‘us_states.json.1’ saved [87688/87688]
```

```
import pandas as pd

import folium
import json

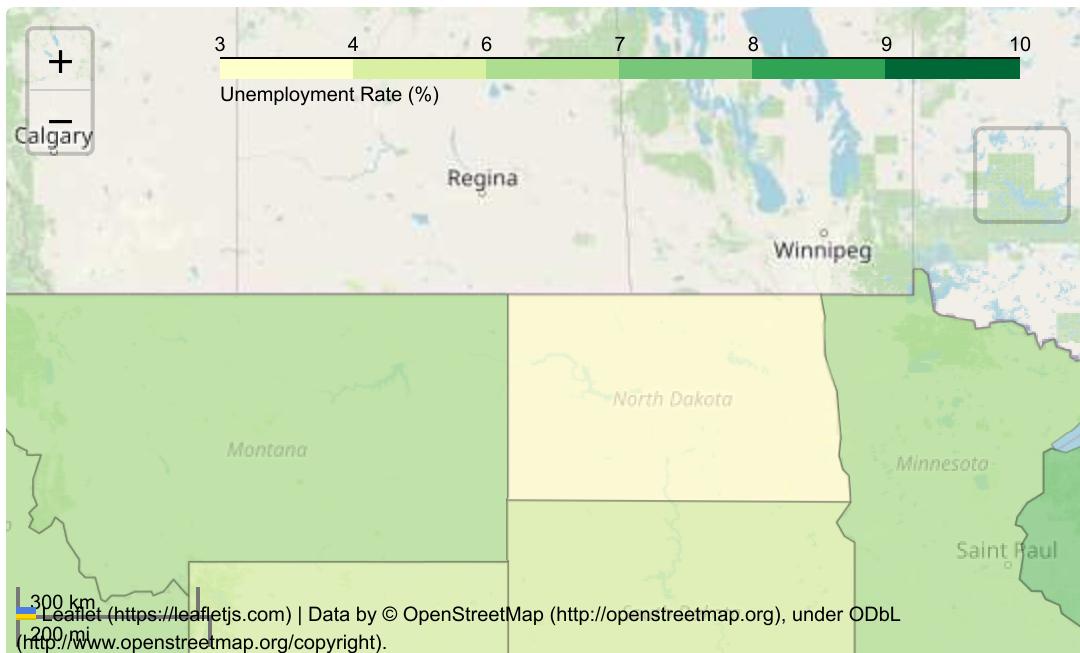
state_geo = json.load(open("us_states.json"))

state_data = pd.read_csv(
    "https://raw.githubusercontent.com/python-visualization/folium-example-data/main/u
)
state_data.head()
```

	State	Unemployment
0	AL	7.1
1	AK	6.8
2	AZ	8.1
3	AR	7.2
4	CA	10.1

```
m = folium.Map(location=[48, -102], zoom_start=5, width="80%", height="80%", control_scale=True)
# Agreguemos la informacion del datafram en el mapa
folium.Choropleth(
    geo_data=state_geo,           # data geográfica (JSON con los estados)
    name="choropleth",           # nombre del mapa
    data=state_data,             # datafram con los datos
    columns=["State", "Unemployment"], # columnas del datafram
    key_on="feature.id",         # llave para unir los datos
    fill_color="YlGn",            # colores de los estados
    fill_opacity=0.7,             # opacidad del color
    line_opacity=0.2,             # opacidad de la linea
    legend_name="Unemployment Rate (%)", # nombre de la leyenda
    highlight=True,               # resaltar el estado al pasar el mouse por encima
    nan_fill_color='white',
    nan_fill_opacity=0.5,
).add_to(m)                      # Agregar este gráfico al mapa anterior

folium.LayerControl().add_to(m) # Agregar controles al mapa para moverlo
m
```



```
m.save("map.html")
```

Enviar correos electrónicos desde aplicaciones de Python es una tarea común y necesaria en muchos proyectos, desde notificaciones automáticas hasta boletines informativos. Python facilita esta tarea mediante la biblioteca estándar `smtplib`, que proporciona una interfaz simple y efectiva para enviar correos electrónicos utilizando el protocolo SMTP (Simple Mail Transfer Protocol).

```
# Importemos las librerías
from email.message import EmailMessage
import smtplib

# definamos la información que necesitamos
remitente = "dirección@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
```

```
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "Correo de prueba de automatización"
email.set_content(mensaje)
```

Para poder autenticarte en el servidor de google necesitas la clave de aplicaciones para poder generar una clave, esta es diferente a la contraseña del correo

[Crear Clave de aplicación](#)

```
smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacion
smtp.sendmail(remitente, destinatario, email.as_string()) # Enviamos el correo
smtp.quit() # Cerramos la conexión
```

```
(221,
b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b1094bsm1286049173.43 - gsmtp')
```

```
# Este sería el código para enviar desde un correo de outlook.
smtp = smtplib.SMTP("smtp-mail.outlook.com", port=587)
smtp.starttls()
smtp.login(remitente, "clave_de_outlook_123") # Autenticación con la contraseña del correo
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

Envío de adjuntos.

```
# Importemos las librerias
from email.message import EmailMessage
import smtplib

# definamos la información que necesitamos
remitente = "direccion@gmail.com"
destinatario = "destinatario@ejemplo.com"

mensaje = "¡Hola, mundo!"
email = EmailMessage()

email["From"] = remitente
email["To"] = destinatario
email["Subject"] = "¡Enviado desde Python!"

email.set_content(mensaje)

smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otros
smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicacion
```

```
(235, b'2.7.0 Accepted')
```

```
# Adjuntar un archivo .html que creamos del mapa

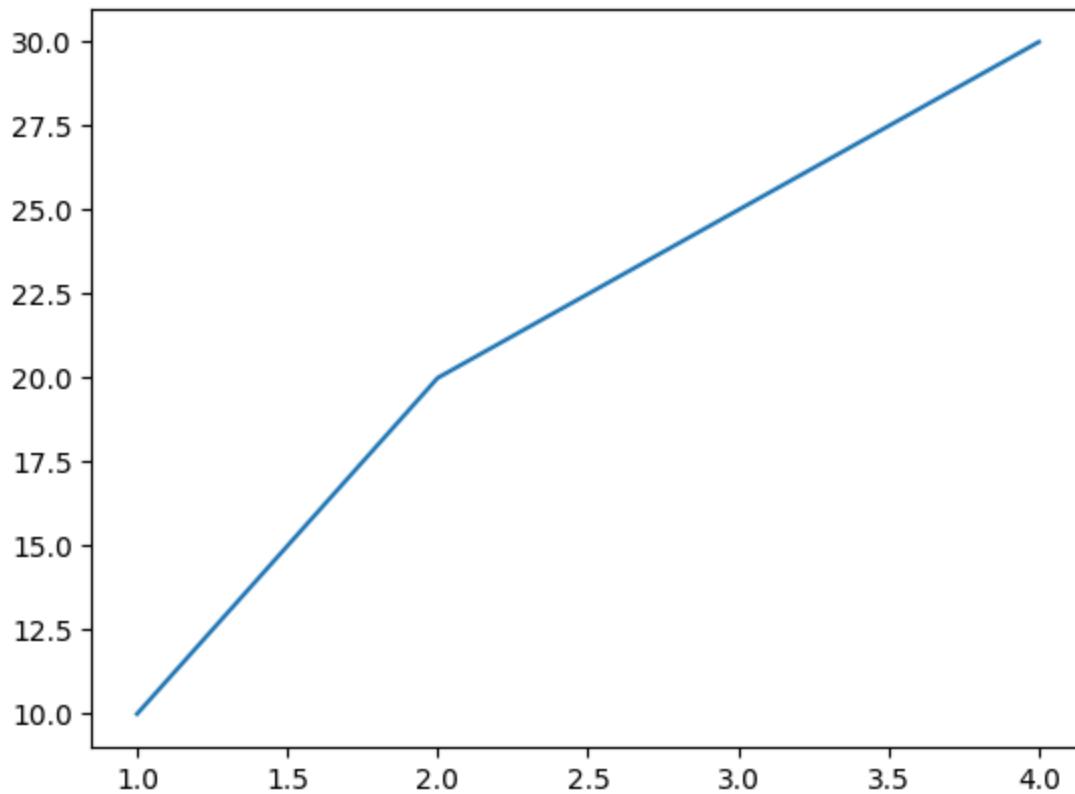
with open("map.html", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="map.html",
        maintype="text",
        subtype="html"
    )

import matplotlib.pyplot as plt
# Crear y adjuntar una gráfica como archivo .png
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.title('Gráfica de ejemplo')

plt.savefig("grafica.png") # Tenemos que guardar el archivo

with open("grafica.png", "rb") as f:
    email.add_attachment(
        f.read(),
        filename="grafica.png",
        maintype="image",
        subtype="png"
    )
```

Gráfica de ejemplo



```
smtp.sendmail(remitente, destinatario, email.as_string())
smtp.quit()
```

```
(221,
b'2.0.0 closing connection 8926c6da1cb9f-4c0b1b0d427sm1299772173.64 - gsmtp')
```

Envío masivo de correos.

```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024/main/emails.csv")
df
```

	Unnamed: 0	first_name	last_name			email	email_verified
0	aaron	Aaron	Davis		aaron6348@gmail.com		
1	acook	Anthony	Cook		cook@gmail.com		
2	adam.saunders	Adam	Saunders		adam@gmail.com		
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com			
4	adrian.blair	Adrian	Blair		adrian9335@gmail.com		
...	
470	wilson	Robert	Wilson		robert@yahoo.com		
471	wking	Wanda	King		wanda.king@holt.com		
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com			
473	young	Jessica	Young		jessica4028@yahoo.com		
474	zachary.neal	Zachary	Neal		zneal@gmail.com		

475 rows × 8 columns



Vamos a enviarle correos a nuestras cuentas verificadas y con un balance mayor a 95 USD

```
df_verified = df[(df["email_verified"] == True) & (df["balance"] > 95)]
df_verified
```

	Unnamed: 0	first_name	last_name		email	email_verified
68	christopher	Christopher	NaN	christopher@gmail.com		True
74	clayton6074	Robert	Clayton	robert@gmail.com		True
84	crane203	Valerie	Crane	valerie7051@hotmail.com		True
139	eugene4448	Eugene	NaN	eugene@gmail.com		True
208	jennifer.wong	Jennifer	Wong	jwong@yahoo.com		True
234	joseph2431	Joseph	Harris	joseph@gmail.com		True
245	karen.snow	Karen	Snow	ksnow@yahoo.com		True
259	king	Billy	King	billy.king@hotmail.com		True
260	king3246	Brittney	King	brittney@yahoo.com		True
296	margaret265	Margaret	NaN	margaret@gmail.com		True
357	paul6364	Paul	Hill	paul2980@hotmail.com		True
361	peter	Peter	Grimes	pgrimes@yahoo.com		True
398	sbennett	Steven	Bennett	sbennett@yahoo.com		True
453	twhite	Timothy	White	white5136@hotmail.com		True

Veamos que le vamos a mandar a cada uno con la informacion del dataset

```
for index, row in df_verified.iterrows():
    print(f"Hello,{row['first_name']} this is your balance: {row['balance']}. This is
```

```
Hello,Christopher this is your balance: 96.96. This is just a test
Hello,Robert this is your balance: 97.05. This is just a test
Hello,Valerie this is your balance: 98.69. This is just a test
Hello,Eugene this is your balance: 95.38. This is just a test
Hello,Jennifer this is your balance: 95.1. This is just a test
Hello,Joseph this is your balance: 95.47. This is just a test
Hello,Karen this is your balance: 99.38. This is just a test
Hello,Billy this is your balance: 98.8. This is just a test
Hello,Brittney this is your balance: 98.79. This is just a test
Hello,Margaret this is your balance: 96.14. This is just a test
Hello,Paul this is your balance: 98.62. This is just a test
Hello,Peter this is your balance: 96.79. This is just a test
Hello,Steven this is your balance: 97.38. This is just a test
Hello,Timothy this is your balance: 99.9. This is just a test
```

```
# Ahora creamos una función que envíe el correo y que como parametro reciba, mensaje )
# Importemos las librerías
from email.message import EmailMessage
import smtplib

def send_mail(mensaje,destinatario):
    remitente = "remitente@gmail.com"

    email = EmailMessage()

    email["From"] = remitente
    email["To"] = destinatario
    email["Subject"] = "|Enviado desde Python test!"

    email.set_content(mensaje)

    smtp = smtplib.SMTP_SSL("smtp.gmail.com") # Iniciamos la conexión a gmail, para otro
    smtp.login(remitente, "clave_de_gmail_123") # Autenticación con la clave de aplicaci
    smtp.sendmail(remitente, destinatario, email.as_string())
    smtp.quit()
```

```
# Ahora enviamos un correo por cada usuario del dataframe
for index, row in df_verified.iterrows():
    mensaje = f"Hello,{row['first_name']} this is your balance: {row['balance']}. This :"
    destino = row['email']
    send_mail(mensaje,destino)
```

Dash es un framework de Python diseñado para crear aplicaciones web interactivas de manera rápida y sencilla. Desarrollado por Plotly, Dash permite a los usuarios construir aplicaciones que integran visualizaciones de datos dinámicas y una interfaz de usuario intuitiva sin necesidad de conocimientos avanzados de desarrollo web.

Estructura de una Aplicación Dash

Una aplicación típica de Dash consta de tres partes principales:

1. Layout: Define la estructura visual de la aplicación, especificando los componentes y su disposición en la página. El layout se describe utilizando estructuras de datos de Python.
2. Callbacks: Define las interacciones entre los componentes. Los callbacks son funciones que especifican cómo los componentes deben actualizarse en respuesta a eventos, como clics de botones o cambios en un formulario.
3. Servidor: Dash incluye un servidor web integrado basado en Flask, lo que permite ejecutar la aplicación localmente durante el desarrollo y luego desplegarla fácilmente en producción.

```
!pip install dash # Instalemos la libreria de Dash
```

```
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (
```

```
# Este pedazo de código es para poder correr Dash sobre los notebooks sin errores
# Nativamente, dash no funciona en Colab entonces debemos colocar estas dos líneas
from dash import jupyter_dash
jupyter_dash.default_mode="external"
```

```
from dash import Dash, html, dcc # Estos son los componentes de Dash que vamos a utilizar
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos
import pandas as pd
```

```
app = Dash(__name__) # Creamos un objeto Dash, esta será la aplicación.
```

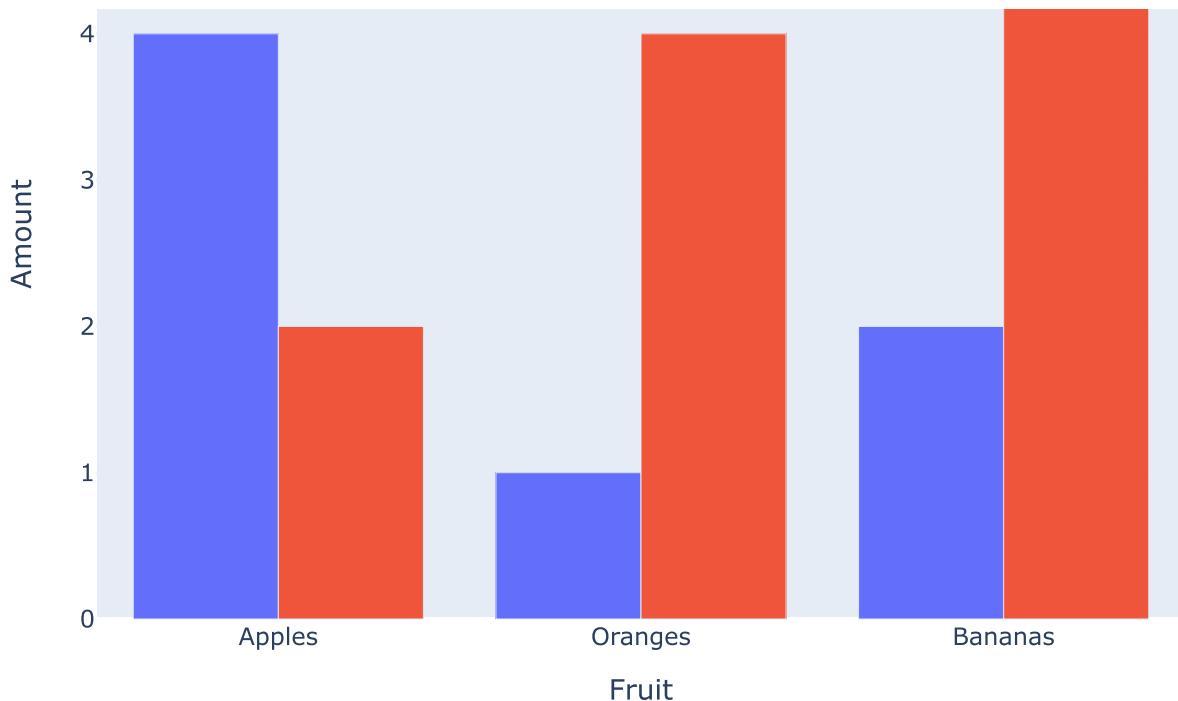
```
df = pd.DataFrame({
    "Fruit": ["Apples", "Oranges", "Bananas", "Apples", "Oranges", "Bananas"],
    "Amount": [4, 1, 2, 2, 4, 5],
    "City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})
df
```

	Fruit	Amount	City
0	Apples	4	SF
1	Oranges	1	SF
2	Bananas	2	SF
3	Apples	2	Montreal
4	Oranges	4	Montreal
5	Bananas	5	Montreal

```
# Generamos un gráfico dinámico que luego vamos a poner en el dashboard.
fig = px.bar(df, x="Fruit", y="Amount", color="City", barmode="group")
fig
```

5





```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),
    html.Div(children='''
        Dash: A web application framework for your data.
    '''),
    dcc.Graph(
        id='example-graph',
        figure=fig
    )
])

# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

Dash app running on:

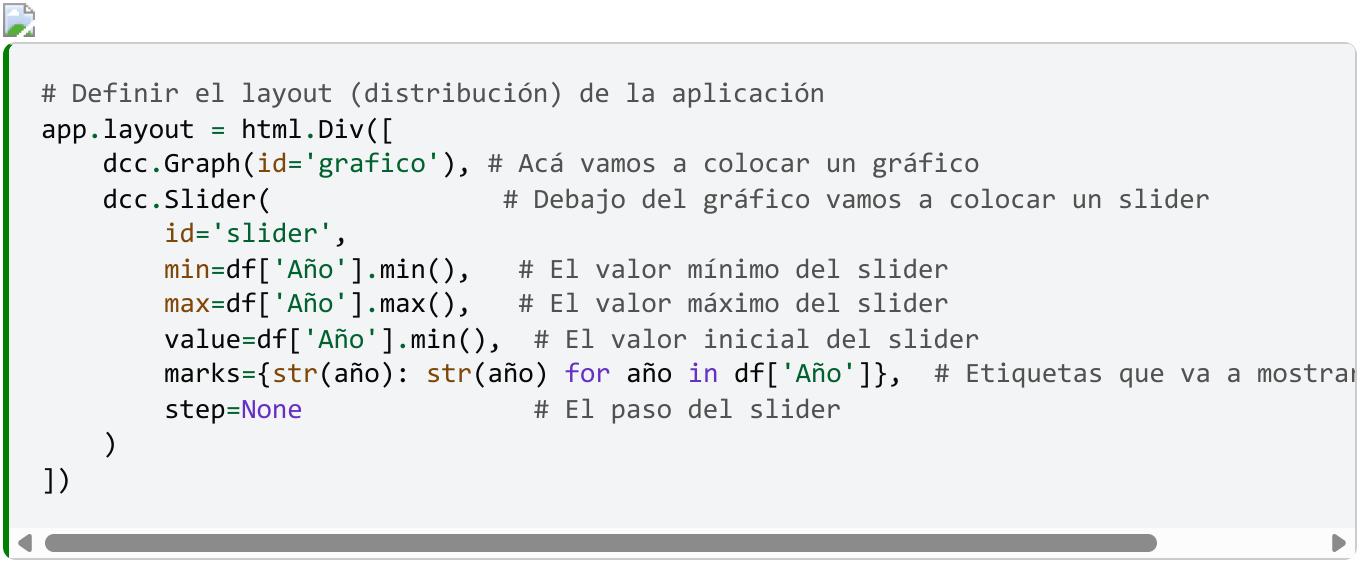
Ejemplo básico 2

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
# Datos de ejemplo
df = pd.DataFrame({
    "Año": [2010, 2011, 2012, 2013, 2014],
    "Valor": [10, 15, 13, 17, 20]
})

# Inicializar la aplicación
app = Dash()
```



```
# Definir el layout (distribución) de la aplicación
app.layout = html.Div([
    dcc.Graph(id='grafico'), # Acá vamos a colocar un gráfico
    dcc.Slider(               # Debajo del gráfico vamos a colocar un slider
        id='slider',
        min=df['Año'].min(), # El valor mínimo del slider
        max=df['Año'].max(), # El valor máximo del slider
        value=df['Año'].min(), # El valor inicial del slider
        marks={str(año): str(año) for año in df['Año']}, # Etiquetas que va a mostrar el slider
        step=None            # El paso del slider
    )
])
```

```
# Definir el callback para actualizar el gráfico
@app.callback(
    Output('grafico', 'figure'), # Definir en qué objeto del layout colocar la salida
    [Input('slider', 'value')]) # Definir en qué objeto del layout está el elemento de entrada
def actualizar_grafico(año_seleccionado):
    df_filtrado = df[df['Año'] == año_seleccionado]
    fig = px.bar(df_filtrado, x='Año', y='Valor')
    return fig
```

```
# Ejecutar la aplicación
if __name__ == '__main__':
    app.run_server(debug=True) # Ejecutamos la aplicación y vamos a ver el enlace que
```

Dash app running on:

Ejemplo intermedio

```
# importamos pandas para leer el archivo csv
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder_u
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
...
3308	Zimbabwe	Africa	1987	62.351	9216418	706.157306
3309	Zimbabwe	Africa	1992	60.377	10704340	693.420786
3310	Zimbabwe	Africa	1997	46.809	11404948	792.449960
3311	Zimbabwe	Africa	2002	39.989	11926563	672.038623
3312	Zimbabwe	Africa	2007	43.487	12311143	469.709298

3313 rows × 6 columns

```
df.keys()
```

```
Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

```
import plotly.express as px # Importamos la librería de plotly para realizar los gráficos

# Estos son los componentes de Dash que vamos a utilizar más adelante
from dash import Dash, html, dcc, callback, Output, Input
```

```
# Creamos un objeto Dash, esta será la aplicación.
app = Dash()
```

```
# Vamos a definirle la organización del tablero que vamos a crear
app.layout = html.Div(
    [
        html.H1("Animated GDP and population over decades"),
        html.P("Select an animation:"),  
dcc.RadioItems(  
    id="selection",  
    options=[ "GDP - Scatter", "Population - Bar"],  
    value="GDP - Scatter",  
,  
    html.Iframe( # Agreguemos el mapa que hicimos al principio  
        srcDoc=open("map.html", "r").read(),  
        width="100%",  
        height="600px",  
,  
        dcc.Loading(dcc.Graph(id="graph"), type="cube"),  
    ]  
)
```

```

# El callback que va a actualizar la figura cuando se modifique el elemento de input.
@app.callback(
    Output("graph", "figure"), Input("selection", "value") # Definir los objetos de sa
)
def display_animated_graph(selection):
    animations = { # Definimos un diccionario que contiene para cada opción un tipo de
        "GDP - Scatter": px.scatter( # El primero es un scatter entonces le asignamos
            df,                                # Establecemos el nombre del dataframe
            x="gdpPercap",                      # Establecemos el eje x
            y="lifeExp",                        # Establecemos el eje y
            animation_frame="year",             # Establecemos el eje de animación
            animation_group="country",          # Establecemos el grupo de animación
            size="pop",                          # Establecemos el tamaño del punto proporciona
            color="continent",                 # Establecemos el color del punto proporcional
            hover_name="country",               # Establecemos el nombre que va a mostrar cada
            log_x=True,                         # Establecemos el eje x en escala logarítmica
            size_max=55,                        # Establecemos el tamaño máximo del punto
            range_x=[100, 100000],              # Establecemos el rango del eje x
            range_y=[25, 90],                  # Establecemos el rango del eje y
        ),
        "Population - Bar": px.bar( # El segundo es un bar entonces le asignamos a la
            df,                                # Establecemos el nombre del dataframe
            x="continent",                      # Establecemos el eje x
            y="pop",                            # Establecemos el eje y
            color="continent",                 # Establecemos el color del punto proporcional
            animation_frame="year",             # Establecemos el eje de animación
            animation_group="country",          # Establecemos el grupo de animación
            range_y=[0, 4000000000],            # Establecemos el rango del eje y
        ),
    }
    return animations[selection] # Retornamos el gráfico seleccionado que es el gráfi

# Run the app
if __name__ == '__main__':
    app.run(debug=True)

```

Dash app running on:

«Data cleaning» y «Filtros»

Contenido

- Tratamiento de valores Faltantes
- Ejercicio
- Operaciones con dataframes
- Retos

Para responder a preguntas a partir de los datos, pandas es una librería muy útil. Sin embargo, los datos no siempre vienen limpios y listos para ser analizados. En este notebook, veremos cómo limpiar datos y cómo filtrarlos para responder a preguntas específicas.

```
# Vamos a cargar un dataset de ejemplo.  
  
import pandas as pd  
  
df = pd.read_csv('data/titanic.csv')  
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A 211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticid
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0

891 rows × 12 columns

Revisa algunos de los métodos anteriores para revisar la data y explorar el tamaño de la información y las columnas.

TU CÓDIGO AQUÍ

Tratamiento de valores Faltantes

Un problema común en los datos es la presencia de valores faltantes. Estos valores pueden ser NaN, None, NaT, entre otros. Para tratar estos valores, podemos usar el método `fillna()` para reemplazar los valores faltantes por un valor específico. También podemos usar el método `dropna()` para eliminar las filas que contienen valores faltantes.

Con el método `isnull()`, podemos identificar los valores faltantes en un DataFrame. Este método devuelve un DataFrame booleano con True en las posiciones donde hay valores faltantes y False en las posiciones donde no hay valores faltantes.

Si usamos `fillna()` se reemplazarán los valores faltantes por el valor que le pasemos como argumento. Si usamos `dropna()` se eliminarán las filas que contienen valores faltantes.

```
df.fillna(' ')
df.head(10) # Note que al parecer el cambio no se ha realizado. ¿Por qué? ¿Qué podemos
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df_cleaned = df.fillna(' ')
df_cleaned.head(10) # Ahora sí se ha realizado el cambio.
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	-	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
9		10	1	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

```
df.fillna('-', inplace=True) # Esta es otra forma de tratar los valores nulos,
# solo que en esta le especificamos que queremos que sea '-'
```

```
# Otras formas de completar los valores nulos son:
# - fillna(0) para completar con ceros.
# - fillna('Desconocido') para completar con un string.
# - fillna(df['columna'].mean()) para completar con el promedio de la columna.
# - fillna(df['columna'].median()) para completar con la mediana de la columna.
# - fillna(df['columna'].mode()[0]) para completar con la moda de la columna.
# - fillna(method='ffill') para completar con el valor anterior.
# - fillna(method='bfill') para completar con el valor siguiente.
```

Ejercicio

Realiza la limpieza de solo una columna del dataframe y que los valores nulos se rellenen por el promedio de la columna.

```
# TU CÓDIGO AQUÍ
```

Operaciones con dataframes

Para realizar operaciones y filtrar información del dataframe o calcular nuevas columnas a partir de los datos ya existentes podemos utilizar funcionalidades especializadas de pandas. Vamos a revisar algunas cosas que podemos hacer con los dataframes.

Operaciones de indexación booleana

```
# Indexación booleana
# La indexación booleana es una técnica que nos permite seleccionar un subconjunto de
# que cumpla con cierta condición.

# Por ejemplo, si queremos seleccionar los pasajeros que son mujeres, podemos hacer lo
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0		1	0	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1		2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2		3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
# 1. revisemos los posibles valores que puede tomar la columna
df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

```
# 2. Como solo quiero seleccionar los que son mujeres hago lo siguiente:
result = df['Sex'] == 'female'

result # así, en result tenemos una serie de booleanos que nos indican si el pasajero
# Pero nosotros queremos es tener los registros de las mujeres, para eso hacemos lo si
```

```
0      False
1      True
2      True
3      True
4      False
...
886    False
887    True
888    True
889    False
890    False
Name: Sex, Length: 891, dtype: bool
```

```
df[result]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17543
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O ⁿ 310128
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	11383
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347740
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	23773
...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	23042
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

```
# O lo que es lo mismo:
df[df['Sex'] == 'female']
# Esta sintaxis se puede leer como: del dataframe df, vamos a seleccionar las filas de
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17543
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O ⁿ 310128
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	11383
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347740
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	23773
...
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	23042
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	75

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	3826
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W. 66

314 rows × 12 columns

Operaciones de indexación con operadores lógicos

```
# Pensemos ahora que queremos saber cuales fueron los pasajeros de la clase mayor a 1
df[df['Pclass'] > 1]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2113128
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O ⁿ 310.101802
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330.000000
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349.000000
...
884	885	0	3	Suthehall, Mr. Henry Jr	male	25.0	0	0	SOTON 39.000000
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	387.000000
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	21.000000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 100.000000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370.000000

675 rows × 12 columns

```
# Ahora extraigamos los pasajeros que son mujeres, sobrevivieron y mayores de 20 años.  
condicion = (df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20) # Note que pa  
# Esta not  
df[condicion]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monopeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

```
# modo PRO
df[(df['Sex']=='female') & (df['Survived']==1) & (df['Age']>20)]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113
...
866	867	1	2	Duran y More, Miss. Asuncion	female	27.0	1	0	SC/PA 2
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monopeny)	female	47.0	1	1	11
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P 3

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
879	880	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11
880	881	1	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230

144 rows × 12 columns

Operaciones entre columnas

Si quiero crear una columna nueva puedo simplemente decir `df['nueva_columna'] = df['columna1'] + df['columna2']` y se creará una nueva columna con la suma de los valores de las columnas 1 y 2.

Veamos como quedaría el código para realizar estas operaciones. en las que sumemos 10 a la columna Fare y luego ese resultado lo vamos a dividir entre la edad para cada uno de los pasajeros.

```
df['mi_nueva_columna'] = 10 + df['Fare'] # Podemos crear nuevas columnas a partir de :  
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

```
df['mi_otra_columna'] = df['mi_nueva_columna'] / df['Age'] # Podemos crear nuevas columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Ordenar los valores de un resultado

Si queremos ordenar los valores de un resultado, podemos usar el método `sort_values()`.

```
# Ordenemos los valores de una columna de forma ascendente, luego de sacar los registros
condicion = (df['Age'] > 20)
df[condicion].sort_values('Age', ascending=True)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
227	228	0	3	male	20.5	0	0	A/5 21173	7.2500
37	38	0	3	male	21.0	0	0	A./5. 2152	8.0500
56	57	1	2	female	21.0	0	0	C.A. 31026	10.5000
436	437	0	3	female	21.0	2	2	W./C. 6608	34.3750
106	107	1	3	female	21.0	0	0	343120	7.6500
...
116	117	0	3	male	70.5	0	0	370369	7.7500
96	97	0	1	male	71.0	0	0	PC 17754	34.6542
493	494	0	1	male	71.0	0	0	PC 17609	49.5042
851	852	0	3	male	74.0	0	0	347060	7.7750
630	631	1	1	male	80.0	0	0	27042	30.0000

535 rows × 13 columns

Operaciones sobre columnas

Si quieres sacar medidas estadísticas de una columna puedes usar los métodos `mean()`, `median()`, `std()`, `min()`, `max()`, `count()`, `sum()`, entre otros.

Aquellas columnas que tienen valores NaN no se incluyen en el cálculo de estas medidas. Si quieres incluir los valores NaN en el cálculo de estas medidas, puedes usar el argumento `skipna=False`.

```
df['Age'].sum() # Podemos hacer operaciones sobre las columnas.
```

```
np.float64(21205.17)
```

```
df.Age.mean() # También podemos hacerlo de esta forma, usando el nombre de la columna  
# pero si el nombre de la columna tiene espacios o caracteres especiales
```

```
np.float64(29.69911764705882)
```

Transformar textos de un dataframe

Ahora miremos otro ejemplo, que pasa si quiero separar de nombre el apellido y el nombre en dos columnas diferentes. Para esto podemos usar el método `str.split()` que nos permite separar un texto en base a un separador.

```
df['Apellido'] = df['Name'].str.split(',').str[0] # Podemos crear nuevas columnas a partir de una sola  
df['Nombre'] = df['Name'].str.split(',').str[1] # Podemos crear nuevas columnas a partir de una sola  
df.head(3)
```

	PassengerId	Survived	Pclass	Name		Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris		male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...		female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina		female	26.0	0	0	STON/O2. 3101282

Eliminar columnas de un dataframe

```
# Eliminar las columnas específicas de un DF
df.drop(columns=['mi_nueva_columna'], inplace=True) # Podemos eliminar columnas que no
df.drop(columns=['Name','mi_otra_columna'], inplace=True) # Podemos eliminar columnas
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Cambiar el nombre de las columnas

Ahora vamos a cambiar los nombres de las columnas al español. Para esto podemos usar el método `rename()` que nos permite cambiar el nombre de las columnas de un dataframe.

```
df.rename(columns={'Apellido':'Apellido_Pasajero', 'Nombre':'Nombre_Pasajero'}, inplace=True)
df.head(3)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250

Ver solo algunas columnas

Veamos como se puede seleccionar solo algunas columnas de un dataframe.

```
pedazo = df[['Nombre_Pasajero', 'Apellido_Pasajero', 'Survived', 'Pclass']]
pedazo.head(3)
```

	Nombre_Pasajero	Apellido_Pasajero	Survived	Pclass
0	Mr. Owen Harris	Braund	0	3
1	Mrs. John Bradley (Florence Briggs Thayer)	Cumings	1	1
2	Miss. Laina	Heikkinen	1	3

Funciones de agregación

Para realizar operaciones de agregación en un dataframe, podemos usar el método `agg()`. Este método nos permite aplicar una función de agregación a una o más columnas de un dataframe.

```
# Funciones de agregación
# Las funciones de agregación nos permiten realizar operaciones sobre un conjunto de columnas.

df['Age'].mean() # Por ejemplo, podemos calcular el promedio de la edad de los pasajeros.

df['Age'].max() # O la edad máxima.

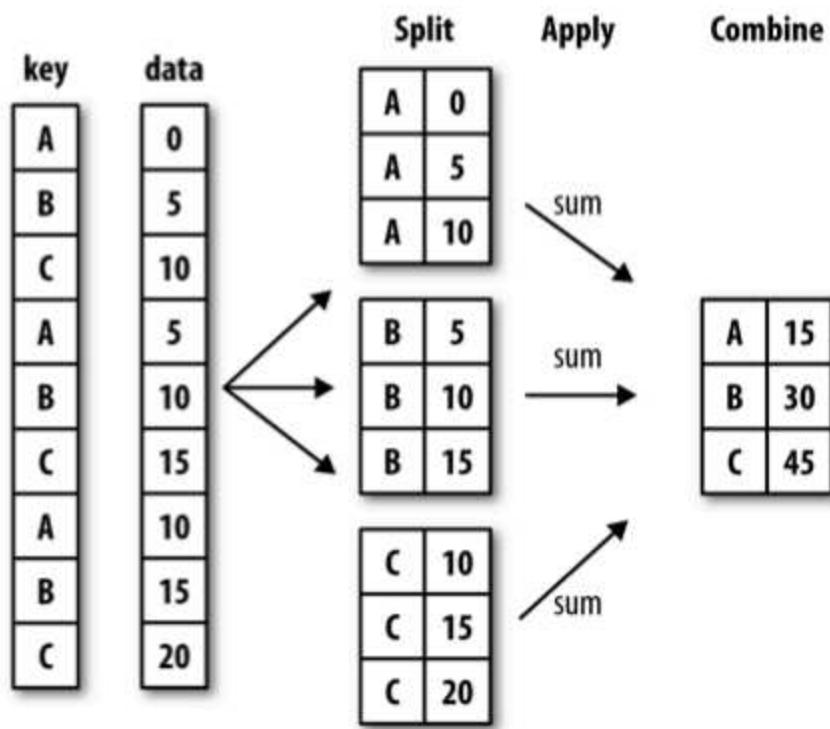
# otra sintaxis es usando el método agg

df['Age'].agg('mean') # Por ejemplo, podemos calcular el promedio de la edad de los pasajeros.
```

```
np.float64(29.69911764705882)
```

Funciones de agrupación

Para realizar operaciones de agrupación en un DataFrame, podemos usar el método `groupby()`. Este método divide el DataFrame en grupos según los valores de una o más columnas. Luego, podemos aplicar funciones de agregación a estos grupos.



```
# Vamos a agrupar nuestros datos por sobrevivientes o no y calcular el promedio de la edad para cada grupo
# Tradicionalmente lo haríamos así:

mean_1 = df[df['Survived']==1]['Age'].mean() # Promedio de la edad de los sobrevivientes
mean_0 = df[df['Survived']==0]['Age'].mean() # Promedio de la edad de los no sobrevivientes

print(mean_1)
print(mean_0)

## Entonces tendríamos que conocer cada una de las etiquetas para hacer los filtros, p
```

28.343689655172415
30.62617924528302

```
df.groupby('Survived')['Age'].sum() # Esto nos retorna una serie en la que saca el valor de la suma para cada grupo

# Esto se puede leer como: Agrupe los registros del dataframe df por la columna Survived
```

Survived
0 12985.50
1 8219.67
Name: Age, dtype: float64

```
df.groupby('Survived')['Age'].agg('sum') # Esto es igual a la anterior, solo que usamos agg()
df.groupby('Survived')['Age'].aggregate('sum') # Esto es igual a la anterior, solo que usamos aggregate()
```

```
Survived
0    12985.50
1     8219.67
Name: Age, dtype: float64
```

Unir varios dataframes usando merge

Esta es una opción muy útil cuando tenemos varios dataframes y queremos unirlos en uno solo. Para esto podemos usar el método `merge()` que nos permite unir dos dataframes en base a una o más columnas en común.

```
import pandas as pd

transactions = pd.read_csv('data/transactions.csv', index_col=0)
requests = pd.read_csv('data/requests.csv', index_col=0)
```

```
# Revisemos los dataframes obtenidos
transactions.head(2)
```

	sender	receiver	amount	sent_date
0	stein	smoyer	49.03	2018-01-24
1	holden4580	joshua.henry	34.64	2018-02-06

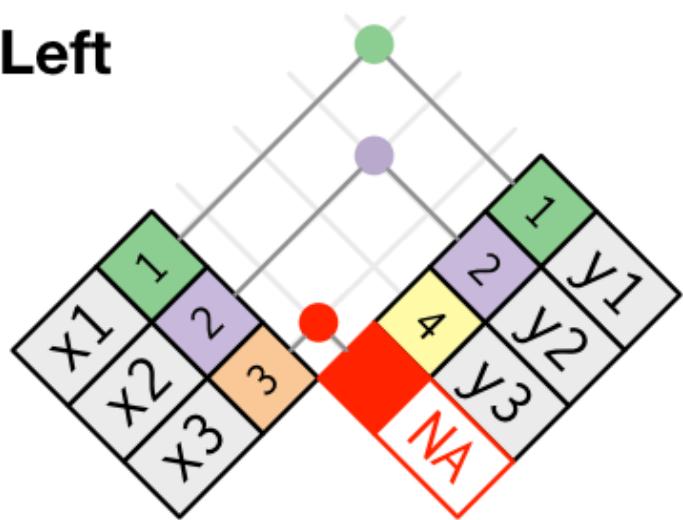
```
# Revisemos los dataframes obtenidos
requests.head(2)
```

	from_user	to_user	amount	request_date
0	chad.chen	paula7980	78.61	2018-02-12
1	kallen	lmoore	1.94	2018-02-23

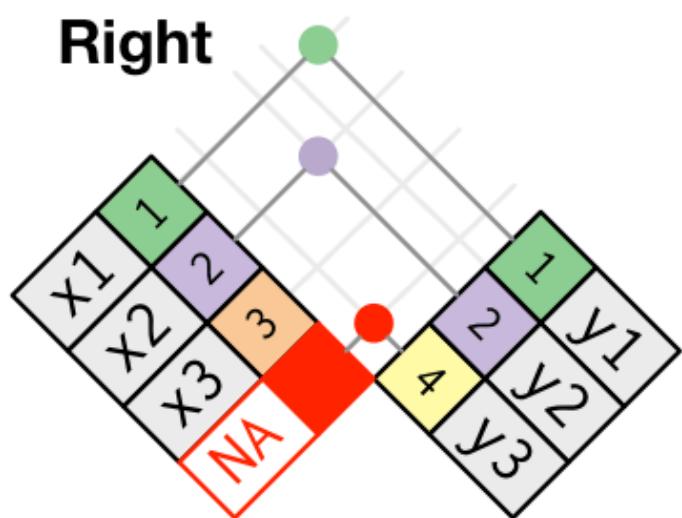
Me gustaría ver todas las solicitudes que tienen una transacción coincidente basada en los usuarios y el monto involucrado.

Para hacer esto, combinaremos ambos conjuntos de datos.

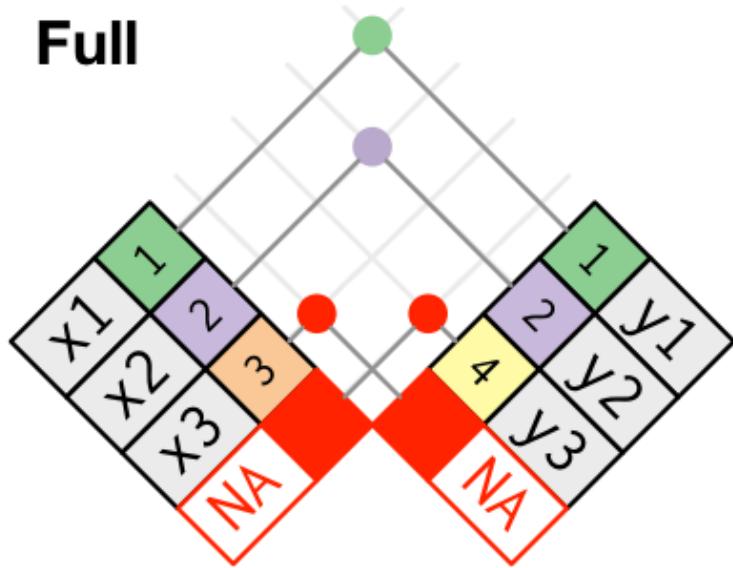
Crearemos un nuevo conjunto de datos utilizando el método DataFrame.merge.

Left

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right

key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

```
# Como estamos llamando a merge en el DataFrame `requests`, se considera el lado izquierdo
successful_requests = requests.merge(
    # Y `transactions` es el lado derecho
    transactions,
    # Así que ahora alineamos las columnas que harán que la unión tenga sentido. esto
    left_on=['from_user', 'to_user', 'amount'],
    right_on=['receiver', 'sender', 'amount']
)
# Veamos cómo nos fue
successful_requests.head()
```

	from_user	to_user	amount	request_date	sender	receiver
0	chad.chen	paula7980	78.61	2018-02-12	paula7980	chad.chen
1	kallen	lmoore	1.94	2018-02-23	lmoore	kallen
2	gregory.blackwell	rodriguez5768	30.57	2018-03-04	rodriguez5768	gregory.blackwell
3	kristina.miller	john.hardy	77.05	2018-03-12	john.hardy	kristina.miller
4	lacey8987	mcguire	54.09	2018-03-13	mcguire	lacey8987

Retos

Reto 1

Vamos a usar la data de sobrevivientes del Titanic. Vamos responder a los siguientes enunciados:

1. ¿Cuántos pasajeros sobrevivieron y cuántos no sobrevivieron?
2. ¿Cuántos pasajeros de cada clase sobrevivieron?
3. ¿Cuántos hombres y mujeres sobrevivieron?
4. Usando groupby(), calcula la edad promedio para cada sexo.
5. Calcula tasa promedio de supervivencia para todos los pasajeros.

6. Calcula este ratio de supervivencia para todos los pasajeros menores de 25 años (recuerda: filtrado/indexación booleana)

```
# Tu código aquí
```

Ejecución de código en múltiples lenguajes de programación

Contenido

- Ejecutar código en C
- Ejecutar código en C++
- Ejecutar código en R

Ejecutar código en C

El Hola Mundo

```
# Versión de GCC
!gcc --version
```

```
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
%%writefile welcome.c
#include <stdio.h> /* incluye biblioteca donde se define E/S */
int main( ){
    printf("Hola mundo en C");
    return 0;
}
```

Writing welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Hola mundo en C

Condicionales en C

```
# Uso de if, else, else if
%%writefile welcome.c

#include <stdio.h>
int main( ){
    int myNum = 10; // Is this a positive or negative number?

    if (myNum > 0) {
        printf("The value is a positive number.");
    } else if (myNum < 0) {
        printf("The value is a negative number.");
    } else {
        printf("The value is 0.");
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

The value is a positive number.

```
# Uso de switch
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int day = 4;

    switch (day) {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
            break;
        case 4:
            printf("Thursday");
            break;
        case 5:
            printf("Friday");
            break;
        case 6:
            printf("Saturday");
            break;
        case 7:
            printf("Sunday");
            break;
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Thursday

Ciclos en C

```
# Uso de While
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int i = 0;

    while (i < 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
./welcome
```

```
0
1
2
3
4
```

```
# Uso de do while
%%writefile welcome.c
#include <stdio.h>
int main( ){

    int i = 0;

    do {
        printf("%d\n", i);
        i++;
    }
    while (i < 5);

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3  
4
```

```
# Uso de for  
%%writefile welcome.c  
#include <stdio.h>  
int main( ){  
  
    int i;  
  
    for (i = 0; i < 5; i++) {  
        printf("%d\n", i);  
    }  
  
    return 0;  
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3  
4
```

```
# Uso de for anidado
%%writefile welcome.c
#include <stdio.h>
int main( ){

    int i, j;

    // Outer loop
    for (i = 1; i <= 2; ++i) {
        printf("Outer: %d\n", i); // Executes 2 times

        // Inner loop
        for (j = 1; j <= 3; ++j) {
            printf(" Inner: %d\n", j); // Executes 6 times (2 * 3)
        }
    }

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Outer: 1
    Inner: 1
    Inner: 2
    Inner: 3
Outer: 2
    Inner: 1
    Inner: 2
    Inner: 3
```

```
# Uso de break en un while
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int i = 0;

    while (i < 10) {
        if (i == 4) {
            break;
        }
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3
```

Funciones en C

```
# Crear una función y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction() {
    printf("I just got executed!\n");
}

int main( ){
    myFunction(); // Llamar a la función
    myFunction(); // Llamar a la función por segunda vez
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

I just got executed!
I just got executed!

```
# Crear una función con entrada de Múltiples parámetros y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction(char name[], int age) {
    printf("Hello %s. You are %d years old.\n", name, age);
}

int main() {
    myFunction("Liam", 3);
    myFunction("Jenny", 14);
    myFunction("Anja", 30);
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Hello Liam. You are 3 years old.  
Hello Jenny. You are 14 years old.  
Hello Anja. You are 30 years old.
```

```
# Crear una función que Retorna un resultado y utilizarla  
%%writefile welcome.c  
#include <stdio.h>  
  
int myFunction(int x) {  
    return 5 + x;  
}  
  
int main() {  
    printf("Result is: %d", myFunction(3));  
    return 0;  
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Result is: 8
```

Matrices en C

```
# Crear un Array 1D (Vector) y recorrerlo usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
    int myNumbers[] = {25, 50, 75, 100};
    int i;

    for (i = 0; i < 4; i++) {
        printf("%d\n", myNumbers[i]);
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
25
50
75
100
```

```
# Crear un Array 2D (Matriz) y recorrerla usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
    int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

    int i, j;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d\n", matrix[i][j]);
        }
    }
    return 0;
}
```

Overwriting welcome.c

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
1
4
2
3
6
8
```

Ejecutar código en C++

```
%%writefile welcome.cpp
#include <iostream>
int main()
{
    std::cout << "Hola mundo en C++";
    return 0;
}
```

Writing welcome.cpp

```
%%bash
g++ welcome.cpp -o welcome
```

```
%%bash
./welcome
```

```
Hola mundo en C++
```

Ejecutar código en R

```
# Instalar versión compatible  
!pip install rpy2==3.5.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu  
Collecting rpy2==3.5.1  
  Downloading rpy2-3.5.1.tar.gz (201 kB)  
           201.7/201.7 KB 15.2 MB/s eta 0:00:00  
?25h Preparing metadata (setup.py) ... ?25l?25hdone  
Requirement already satisfied: cffi>=1.10.0 in /usr/local/lib/python3.8/dist-packages  
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: pytz in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: tzlocal in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Building wheels for collected packages: rpy2  
  Building wheel for rpy2 (setup.py) ... ?25l?25hdone  
    Created wheel for rpy2: filename=rpy2-3.5.1-cp38-cp38-linux_x86_64.whl size=318371 s  
    Stored in directory: /root/.cache/pip/wheels/6b/40/7d/f63e87fd83e8b99ee837c8e3489081  
Successfully built rpy2  
Installing collected packages: rpy2  
  Attempting uninstall: rpy2  
    Found existing installation: rpy2 3.5.5  
    Uninstalling rpy2-3.5.5:  
      Successfully uninstalled rpy2-3.5.5  
Successfully installed rpy2-3.5.1
```

```
# activate R magic  
%load_ext rpy2.ipython
```

```
!R --version
```

```
R version 4.2.2 Patched (2022-11-10 r83330) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
https://www.gnu.org/licenses/.
```

```
%%R
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

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

Introducción

Contenido

- **Introducción**
- Ciencia de datos
- Aprendizaje automático
- Más recursos
- Como saber la versión de Python
- Como saber los recursos de GPU
- Como saber la versión de Linux
- Tutorial Python
- Conexión con Google drive
- Asignación automática de tipos de variables
- Gestión de la memoria
- Casteo de variables
- Operaciones Aritméticas
- Funciones útiles
- Numeros Aleatorios
- Libreria Math
- Operaciones con cadena de texto
- Tuplas (Datos inmutables)
- Listas (Datos no inmutables)
- Diccionarios
- Entrada y salida de datos
- Operadores relacionales
- Operadores lógicos
- Estructuras condicionadas
- Estructuras repetitivas

- Funciones
- Paso por valor y referencia
- Manejo de excepciones y errores.

Si ya conoces Colab, mira este video para aprender sobre las tablas interactivas, la vista histórica de código ejecutado y la paleta de comandos.



¿Qué es Colab?

Colab, o «Colaboratory», te permite escribir y ejecutar código de Python en tu navegador, con

- Una configuración lista para que empieces a programar
- Acceso gratuito a GPU
- Facilidad para compartir

Seas **estudiante, científico de datos o investigador de IA**, Colab facilita tu trabajo. Mira [este video introductorio sobre Colab](#) para obtener más información, o bien comienza a usarlo más abajo.

El documento que estás leyendo no es una página web estática, sino un entorno interactivo denominado **notebook de Colab**, que permite escribir y ejecutar código.

Por ejemplo, esta es una **celda de código** con una secuencia de comandos Python corta que calcula un valor, lo almacena en una variable y devuelve el resultado:

```
seconds_in_a_day = 24 * 60 * 60  
seconds_in_a_day
```

```
86400
```

```
type(seconds_in_a_day)
```

```
int
```

A fin de ejecutar el código en la celda anterior, haz clic en él para seleccionarlo y luego presiona el botón de reproducción ubicado a la izquierda del código o usa la combinación de teclas «Command/Ctrl + Intro». Para editar el código, solo haz clic en la celda y comienza a editar.

Las variables que defines en una celda pueden usarse en otras:

```
seconds_in_a_week = 7 * seconds_in_a_day  
seconds_in_a_week
```

```
604800
```

Los notebooks de Colab te permiten combinar **código ejecutable** y **texto enriquecido** en un único documento, junto con **imágenes**, **HTML**, **LaTeX** y mucho más. Los notebooks que crees en Colab se almacenan en tu cuenta de Google Drive. Puedes compartir fácilmente los notebooks de Colab con amigos o compañeros de trabajo para que realicen comentarios o los editen. Si quieres obtener más información, consulta la [Descripción general de Colab](#). Para crear un nuevo notebook de Colab, ve al menú Archivo que aparece más arriba o usa este vínculo: [crear un nuevo notebook de Colab](#).

Los notebooks de Colab son notebooks de Jupyter que aloja Colab. Para obtener más información sobre el proyecto Jupyter, visita [jupyter.org](#).

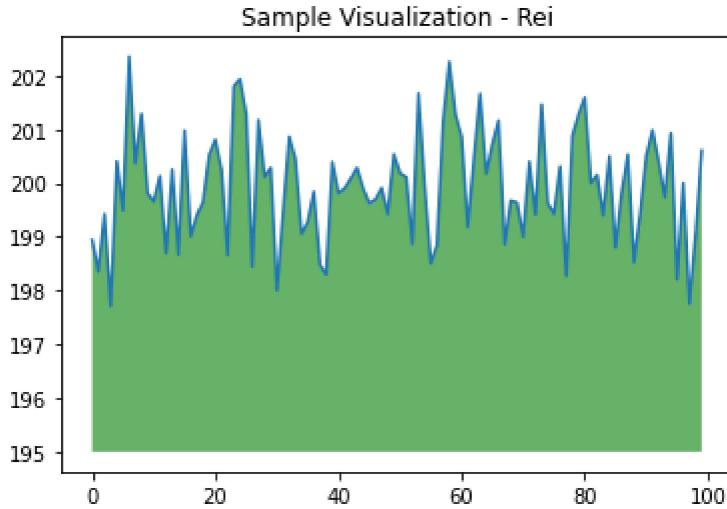
Con Colab, puedes aprovechar por completo las bibliotecas más populares de Python para analizar y visualizar datos. La celda de código que se incluye a continuación usa **NumPy** para generar algunos datos aleatorios y **matplotlib** para visualizarlos. Para editar el código, haz clic en la celda y comienza a editar.

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

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization - Rei")
plt.show()
```



Puedes importar datos propios a notebooks de Colab desde tu cuenta de Google Drive (incluso desde hojas de cálculos), GitHub y muchas otras fuentes. Para obtener más información acerca de la importación de datos y cómo puede usarse Colab para fines relacionados con la ciencia de datos, consulta los vínculos de [Cómo trabajar con datos](#).

Colab te permite importar un conjunto de datos de imágenes, entrenar un clasificador de imágenes en él y evaluar el modelo con solo [unas pocas líneas de código](#). Los notebooks de Colab ejecutan código en los servidores alojados en la nube de Google, lo que significa que puedes aprovechar al máximo el hardware de Google, incluidas las [GPU y TPU](#), independientemente de la potencia de tu máquina. Lo único que necesitas es un navegador.

Entre los usos que se la da a Colab en la comunidad de aprendizaje automático, se encuentran los siguientes:

- Introducción a TensorFlow
- Desarrollo y entrenamiento de redes neuronales
- Experimentación con TPU
- Diseminación de investigación de IA

- Creación de instructivos

Para ver notebooks de Colab de ejemplo que muestran los usos del aprendizaje automático, consulta los [ejemplos](#) que se incluyen a continuación.

Cómo trabajar con notebooks en Colab

- [Descripción general de Colaboratory](#)
- [Guía para usar Markdown](#)
- [Cómo importar bibliotecas y luego instalar dependencias](#)
- [Cómo guardar y cargar notebooks en GitHub](#)
- [Formularios interactivos](#)
- [Widgets interactivos](#)

Cómo trabajar con datos

- [Cómo cargar datos: Drive, Hojas de cálculo y Google Cloud Storage](#)
- [Gráficos: visualización de datos](#)
- [Cómo comenzar a usar BigQuery](#)

Curso intensivo de aprendizaje automático

Estos son algunos de los notebooks del curso de aprendizaje automático en línea de Google.

Para obtener más información, consulta el [sitio web del curso completo](#).

- [Introducción a Pandas DataFrame](#)
- [Regresión lineal con tf.keras usando datos sintéticos](#)

Uso de aceleración de hardware

- [TensorFlow con GPU](#)
- [TensorFlow con TPU](#)

Ejemplos destacados

- [NeMo Voice Swap](#): Utiliza el kit de herramientas de NeMo para IA conversacional de Nvidia si quieres cambiar una voz en un fragmento de audio por otra generada por computadora.
- [Reentrenamiento de un clasificador de imágenes](#): compila un modelo de Keras sobre un clasificador de imágenes previamente entrenado para distinguir flores.
- [Clasificación de texto](#): clasifica opiniones sobre películas de IMDB como *positivas* o *negativas*.
- [Transferencia de estilos](#): usa el aprendizaje profundo para transferir el estilo de una imagen a otra.
- [Codificador universal de oraciones en varios idiomas para preguntas y respuestas](#): usa un modelo de aprendizaje automático para responder preguntas del conjunto de datos SQuAD.
- [Interpolación de videos](#): predice lo que sucedió en un video entre el primer y el último fotograma.

```
#Versión de python  
!python --version
```

Python 3.8.10

```
# Recursos de GPU  
!nvidia-smi  
!/usr/local/cuda/bin/nvcc --version
```

Tue Feb 21 19:33:36 2023

```
+-----+
| NVIDIA-SMI 510.47.03      Driver Version: 510.47.03      CUDA Version: 11.6 |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|          Memory-Usage | GPU-Util  Compute M.  |
| |                               |               MIG M.   |
+-----+
| 0  Tesla T4           Off  | 00000000:00:04.0 Off |                0 |
| N/A   67C     P0    28W /  70W |          0MiB / 15360MiB |      0%      Default |
| |                               |                           N/A |
+-----+
+-----+
| Processes:
| GPU  GI  CI      PID  Type  Process name                  GPU Memory
|       ID  ID
+-----+
| No running processes found
+-----+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_Mar_8_18:18:20_PST_2022
Cuda compilation tools, release 11.6, V11.6.124
Build cuda_11.6.r11.6/compiler.31057947_0
```

```
# Versión de Ubuntu
!lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
```

<https://www.w3schools.com/python/default.asp>

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

En Python no es necesario declarar explícitamente el tipo de las variables. El intérprete trata de inferir el tipo de las variables según se usan. Igualmente, las operaciones tienen semántica distinta para distintos tipos de datos. Fíjate en el ejemplo siguiente.

```
a = 10
b = 3
c = 3.
d = "holaworld"
e = True
f = 1e3
g = [1,2,3] # La lista es mutable
h = (1,2,3) # La tupla es inmutable
i = 5j
j = 1+2j

print ("a=",type(a))
print ("b=",type(b))
print ("c=",type(c))
print ("d=",type(d))
print ("e=",type(e))
print ("f=",type(f))
print ("g=",type(g))
print ("h=",type(h))
print ("i=",type(i))
print ("j=",type(j))
```

```
a= <class 'int'>
b= <class 'int'>
c= <class 'float'>
d= <class 'str'>
e= <class 'bool'>
f= <class 'float'>
g= <class 'list'>
h= <class 'tuple'>
i= <class 'complex'>
j= <class 'complex'>
```

Tipo de datos BOOLEANO

El tipo de datos *boolean* (booleano) denota el rango de *valores lógicos* que consiste de dos elementos *true* y *false* y está definido de la siguiente manera:

$$\text{type boolean} = (\text{false}, \text{true})$$

Los siguientes *operadores* (operaciones) están definidos sobre argumentos de este tipo.

- AND: conjunción lógica,

$$\wedge : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- OR: disyunción lógica, inclusiva,

$$\vee : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- NOT: negación lógica,

$$\neg : \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

True and False , True or False, not True

(False, True, False)

Operadores relacionales

Todos los operadores relacionales producen un resultado de tipo *Booleano*. Los seis operadores relacionales básicos son,

$$=, \neq, <, \leq, \geq, >$$

Por ejemplo, de acuerdo a los valores que sean asignados a las variables x y y , el valor de la siguiente expresión es *false* o *true* :

$$(x + y \geq 0 \wedge x * y < 0) \vee \neg(x * x \geq 20).$$

Los cuatro operadores relacionales $<$, \leq , \geq , $>$ solo pueden ser aplicados a tipos de datos ordenados (escalares).

`2>1, True<False`

`(True, False)`

Tipo de datos INTEGER

EL tipo *integer* representa el conjunto de *números enteros*. Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: *integer* \times *integer* \rightarrow *integer*
- sustracción $-$: *integer* \times *integer* \rightarrow *integer*
- multiplicación $*$: *integer* \times *integer* \rightarrow *integer*
- división con truncamiento **div**: *integer* \times (*integer* $- \{0\}) \rightarrow *integer*
La operación *a div b* no está definida para $b = 0$.$
- residuo de la división entera **mod** : *integer* \times *integer*⁺ \rightarrow *integer*
La operación *a mod b* solo está definida para b un número entero positivo.

Profundizar en la representación de enteros en python:

- [https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To store integers%2C the computers,numbers to represent the integers.&text=By using 8 bits%2C you,8 – 1 %3D 255 integers.](https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To%20store,integers%2C%20the%20computers%2C%20numbers%20to%20represent%20the%20integers.&text=By%20using%208%20bits%2C,you,%208%20-%201%20=%20255%20integers.)

```
import numpy as np  
z=np.uint8(255)  
z
```

255

```
z,type(z),z nbytes
```

(255, numpy.uint8, 1)

```
x=np.int8(4)+np.int8(6)  
y=np.uint8(100)+np.uint8(27)
```

```
x, y, type(x), type(y)
```

(10, 127, numpy.int8, numpy.uint8)

Tipo de datos REAL

- Todo intervalo de números reales contiene infinitos números.
- Los números reales son densos.
- El tipo *real* no representa exactamente a los números reales.
- El tipo *real* es un conjunto finito de representantes de intervalos de números reales.
- El tipo *real* no es un tipo ordinal.
- Cursos de Sistemas Númericos, Análisis numérico.

Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: *real* \times *real* \rightarrow *real*
 - sustracción $-$: *real* \times *real* \rightarrow *real*
 - multiplicación $*$: *real* \times *real* \rightarrow *real*
 - división $/$: *real* \times (*real* $- \{0\}$) \rightarrow *real*
- La operación a/b no está definida para $b = 0$.

Char

Tipo de datos CHAR

El tipo *char* denota un *conjunto de caracteres*, finito y ordenado.

- 26 letras
- 10 dígitos
- caracteres especiales, como el carácter para espacio y otros símbolos de puntuación.

Funciones de transferencia y funciones predecesor y sucesor

Las funciones de transferencia del tipo *char* al tipo *integer* y viceversa, están definidas por:

- $ord(c)$ es el número entero, llamado ordinal, del carácter c en el conjunto de caracteres.
- $chr(i)$ es el carácter con número ordinal i .

Por tanto, se tienen las siguientes relaciones

$$chr(ord(c)) = c \quad \text{y} \quad ord(chr(i)) = i$$

y el ordenamiento del conjunto de caracteres está definido por:

$$c_1 < c_2 \text{ si y solamente si } ord(c_1) < ord(c_2)$$

Las funciones predecesor y sucesor están definidas respectivamente por:

$$pred(c) = chr(ord(c) - 1) \quad \text{y} \quad succ(c) = chr(ord(c) + 1)$$

```
ord("a"),ord("A"), ord("b"),ord("c"),ord("d")
```

```
(97, 65, 98, 99, 100)
```

```
'a'>'b'
```

```
False
```

```
'a'>'A'
```

```
True
```

```
chr(97),chr(98)
```

('a' , 'b')

Profundizar en la gestión de memoria en Python

- [!\[\]\(fbd40d5cb72f365b8aa2bed6680f8611_img.jpg\) customprogrammingsolutions/python-memory-usage](https://github.com/customprogrammingsolutions/python-memory-usage)
 - <https://codeblessu.com/python/size-of-data-types.html#:~:text=Use%20sys.%2C%20to%20store%20it%20in%20memory>
 - <https://www.lesinskis.com/images/CPythonMemoryStructurePosterJanisLesinskisAlyshalannetta.pdf>

```
from sys import getsizeof  
import sys as yi
```

help(sizeof)

Help on built-in function `getsizeof` in module `sys`:

```
getsizeof(...)  
    getsizeof(object [, default]) -> int
```

```
a= 100000000000000000000000000000000000000000000000000000000000000 el tamaño es= 40
b= 3 el tamaño es= 28
c= 3.0 el tamaño es= 24
d= holaWorld el tamaño es= 58
```

```
a = 5
b = np.int8(5)
c = 'reine'
d = np.array([1,2,3,4]).astype('int8')

print(type(a), getsizeof(a))
print(b.nbytes, type(b))
print(type(c), getsizeof(c))
print(d.nbytes, type(d))
```

```
<class 'int'> 28
1 <class 'numpy.int8'>
<class 'str'> 54
4 <class 'numpy.ndarray'>
```

```
x=np.int8(128)
```

`x, x nbytes, type(x)`

(-128, 1, numpy.int8)

$$y=40$$

`getsizeof(y), type(y)`

(28, int)

`type(x), sizeof(x)`

(int, 96)

¿Por qué el resultado de las dos últimas divisiones no es el mismo?

```
a=10  
b=3  
c=3.  
print ("a =",a,"; b =",b, " ; c =",c)  
print ("a/b =", a/b)  
print ("a/c =", a/c)  
print ("a//b =", a//b)  
print ("a//c =", a//c)
```

```
a = 10 ; b = 3 ; c = 3.0  
a/b = 3.333333333333335  
a/c = 3.333333333333335  
a//b = 3  
a//c = 3.0
```

```
Variable = """  
Esta es una cadena  
de varias líneas  
"""
```

```
print(Variable)
```

```
Esta es una cadena  
de varias líneas
```

```
type(Variable)
```

```
str
```

```
getsizeof(Variable)
```

```
110
```

```
x=np.array(['a','b','c'])
```

```
x.nbytes,type(x)
```

```
(12, numpy.ndarray)
```

```
x='2'
y='3'
z=x+y
print('z=',z)

x1=int(x)
y1=int(y)
z1=x1+y1
print('z1=',z1)
```

```
z= 23
z1= 5
```

```
print("Suma = ", 4+5)
print("Resta = ", 4-5)
print("Multiplicación = ", 4*5)
print("División = ", 4/5)
print("División entera = ", 10//3)
print("Residuo = ", 10%3)
print("Potencia = ", 2**3)
print("Raíz = ", 2**0.5)
```

```
Suma = 9
Resta = -1
Multiplicación = 20
División = 0.8
División entera = 3
Residuo = 1
Potencia = 8
Raíz = 1.4142135623730951
```

```
print("Redondear números= ",round(3.867483,3))
print("Valor Absoluto= ",abs(-4))
print("Convertir número a decimal= ",float('3.5'))
print("Convertir número a hexadecimal=", hex(5))
print("Convertir cadena o número decimal a entero=", int(6.9))
print("Convertir número a cadena de texto=", str(6)+str(6))
```

```
Redondear números= 3.867
Valor Absoluto= 4
Convertir número a decimal= 3.5
Convertir número a hexadecimal= 0x5
Convertir cadena o número decimal a entero= 6
Convertir número a cadena de texto= 66
```

```
import random
print('imprime un numero aleatorio entero=' ,random.randrange(3,10))
print('selecciona aleatoriamente=' ,random.choice(["uno", "dos", "tres"]))
print('imprime un numero aleatorio decimal entre 0 y 1=' ,random.random())
print('imprime una letra aleatoria=' ,random.choice("AEIOU"))
```

```
imprime un numero aleatorio entero= 9
selecciona aleatoriamente= dos
imprime un numero aleatorio decimal entre 0 y 1= 0.5008250519253324
imprime una letra aleatoria= U
```

```
print('imprime un numero aleatorio decimal entre 0 y 4=' ,int(4*random.random()))
```

```
imprime un numero aleatorio decimal entre 0 y 4= 3
```

```
import math
print("Factorial=",math.factorial(3))
print("PI=",math.pi)
print("E=",math.e)
print("seno=",math.sin(math.pi/6))
print("aseño=",math.degrees(math.asin(0.5)))
print("Logaritmo Natural=",math.log(math.exp(2)))
print("Logaritmo en base 10=",math.log10(100))
print("Logaritmo en base 2=",math.log2(8))
print("Potencia=",math.pow(3,2))
print("Raiz cuadrada=",math.sqrt(4))
```

```
Factorial= 6
PI= 3.141592653589793
E= 2.718281828459045
seno= 0.4999999999999994
aseño= 30.00000000000004
Logaritmo Natural= 2.0
Logaritmo en base 10= 2.0
Logaritmo en base 2= 3.0
Potencia= 9.0
Raiz cuadrada= 2.0
```

```
import numpy as np  
np.float(5/0)
```

```
<ipython-input-83-136236f1537d>:2: DeprecationWarning: `np.float` is a deprecated alias  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rele  
np.float(5/0)
```

```
-----  
ZeroDivisionError Traceback (most recent call last)  
<ipython-input-83-136236f1537d> in <module>  
      1 import numpy as np  
----> 2 np.float(5/0)  
  
ZeroDivisionError: division by zero
```

5/0

```
-----  
ZeroDivisionError Traceback (most recent call last)  
<ipython-input-82-0106664d39e8> in <module>  
----> 1 5/0  
  
ZeroDivisionError: division by zero
```

```
#Suma o concatenación  
x="Un divertido"+"programa"+ "de" + "radio"  
print(x)  
#Multiplicación de una cadena con un número  
y=3*"programas"  
print(y)  
#Obtener longitud de una cadena  
print('longitud de la cadena:',len(y))
```

Un divertidoprogamaderadio
programasprogramasprogramas
longitud de la cadena: 27

```
#Acceder a una posición de la cadena
cadena = "programA"
print('acceder a una posicion determinada:',cadena[0])
print('acceder a la última posición:',cadena[-1])

# Comillas simples
cadena = 'Texto entre comillas simples'
print (cadena)
print (type(cadena))

# Comillas dobles
cadena = "Texto entre comillas dobles"
print (cadena)
print (type(cadena))
```

```
acceder a una posicion determinada: p
acceder a la última posición: A
Texto entre comillas simples
<class 'str'>
Texto entre comillas dobles
<class 'str'>
```

cad='reinel'

cad[2:5]

'ine'

```
cadenaesc = 'Texto entre \n\n\n \t\t\tcomillas simples'
print (cadenaesc)
print (type(cadenaesc))
```

Texto entre

comillas simples
<class 'str'>

Las tuplas son más rápidas que las listas. Si define un conjunto **constante** de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.

Una tupla es una secuencia de items ordenada e inmutable.

Los items de una tupla pueden ser objetos de cualquier tipo.

Para especificar una tupla, lo hacemos con los elementos separados por comas dentro de **paréntesis**.

Una tupla con únicamente dos elementos es denominada par.

Para crear una tupla con un único elemento (singleton), se añade una coma al final de la expresión.

Para definir una tupla vacía, se emplean unos paréntesis vacíos.

```
tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
print(tupla)
print(type(tupla))
print(tupla[1:4])
print('cuantas veces hay un elemento=',tupla.count(25))
print('dice en que posicion esta el elemento=',tupla.index(15))
#tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.
```

```
('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
<class 'tuple'>
(15, 2.8, 'otro dato')
cuantas veces hay un elemento= 2
dice en que posicion esta el elemento= 1
```

```
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-6-1233f0186c5b> in <module>
      5 print('cuantas veces hay un elemento=',tupla.count(25))
      6 print('dice en que posicion esta el elemento=',tupla.index(15))
----> 7 tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.

TypeError: 'tuple' object does not support item assignment
```

```
tupla, type(tupla), tupla[1:-1],tupla.count(25),tupla.index(15)
```

```
(('cadena de texto', 15, 2.8, 'otro dato', 25, 25),
 tuple,
 (15, 2.8, 'otro dato', 25),
 2,
 1)
```

```
# Usando el tag: raises-exception, se evita que se detenga en compilación
#tupla[0]=2
```

Una lista es una secuencia ordenada de elementos **mutable**.

Los items de una lista pueden ser objetos de distintos tipos.

Para especificar una lista se indican los elementos separados por comas en el interior de **CORCHETES**.

Para denotar una lista vacía se emplean dos corchetes vacíos.

```
a = [1,2,3, "hola", [10, "nunca", 90], -32]
print ("Toda la lista= ",a)
print(type(a))
print ("Longitud de la lista= ", len(a))
print("Acceder a un rango de posiciones=",a[0:4])
print("Acceder a una posición específica= ",a[4])
print("Acceder a la última posición= ",a[-1])
```

```
Toda la lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
<class 'list'>
Longitud de la lista=  6
Acceder a un rango de posiciones= [1, 2, 3, 'hola']
Acceder a una posición específica=  [10, 'nunca', 90]
Acceder a la última posición= -32
```

```
print("Lista= ",a)
a.append("Nuevo último")
print("Lista con nuevo último= ",a)
a[1]='nuevo1'
print("Lista con nuevo elemento en las posición 1= ",a)
a.insert(2,'nuevo2')
print("Lista con nuevo elemento en las posición 2= ",a)
```

```
Lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
Lista con nuevo último=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
Lista con nuevo elemento en las posición 1=  [1, 'nuevo1', 3, 'hola', [10, 'nunca', 90]
Lista con nuevo elemento en las posición 2=  [1, 'nuevo1', 'nuevo2', 3, 'hola', [10, '
```

a,type(a),len(a)

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último'],
list,
8)
```

```
for i in range(len(a)):
    print (i, "-->", a[i])
```

```
0 --> 1
1 --> nuevo1
2 --> nuevo2
3 --> 3
4 --> hola
5 --> [10, 'nunca', 90]
6 --> -32
7 --> Nuevo último
```

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

```
1
nuevo1
nuevo2
3
hola
[10, 'nunca', 90]
-32
Nuevo último
```

```
print(a)
print (a[2:])
print (a[-3:])
print (a[4])
```

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
['nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
[[10, 'nunca', 90], -32, 'Nuevo último']
hola
```

```
a = [1,2,3,"hola", [10, "nunca", 90], -32]
print (a[4])
print (a[4][0])
print (a[4][1])
print (a[4][1][2:])
```

```
[10, 'nunca', 90]
10
nunca
nca
```

Los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados. En Python, un diccionario es una colección **no-ordenada** de valores que son accedidos a través de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos. Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior. No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves. La información almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario. Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

```
# Definir una variable diccionario
futbolistas = dict()

futbolistas = {
    1: "Casillas", 6: "Iniesta", 3: "Piqué",
    5: "Puyol",
    7: "Villa", 8: "Xavi Hernández",
    9: "Torres", 11: "Capdevila",
    14: "Xavi Alonso", 15: "Ramos",
    16: "Busquets"
}

# Recorrer el diccionario, imprimiendo clave - valor
print ("Vemos que los elementos no van \"ordenados\":")
for k, v in futbolistas.items():
    print ("{} --> {}".format(k,v))
```

Vemos que los elementos no van "ordenados":

```
1 --> Casillas
6 --> Iniesta
3 --> Piqué
5 --> Puyol
7 --> Villa
8 --> Xavi Hernández
9 --> Torres
11 --> Capdevila
14 --> Xavi Alonso
15 --> Ramos
16 --> Busquets
```

```
futbolistas.items(),futbolistas.keys(),futbolistas.values()
```

```
(dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa')]),  
 dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16]),  
 dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres'])
```

```
# Nº de elementos que tiene un diccionario  
numElemen = len(futbolistas)  
print ("\nEl número de futbolistas es de {}".format(numElemen))  
  
# Imprimir las claves que tiene un diccionario  
keys = futbolistas.keys();  
print ("\nLas claves de nuestro diccionario son : {}".format(keys))  
  
# Imprimir los valores que tiene un diccionario  
values = futbolistas.values();  
print ("\nLos valores de nuestro diccionario son : {}".format(values))
```

El número de futbolistas es de 11

Las claves de nuestro diccionario son : dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16])

Los valores de nuestro diccionario son : dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres', 'Busquets', 'Ramos', 'Capdevila', 'Alonso', 'Hernández', 'Torres'])

```
# Obtener el valor de un elemento dada su clave
elem = futbolistas.get(6)
print ("\nEl futbolista con clave 6 es {}".format(elem))

# Insertamos un elemento en el diccionario
## si la clave ya existe, el elemento NO se inserta
futbolistas.setdefault(10, 'Cesc')
print ("\nInsertamos el elemento con clave 10 y valor Cesc")
print ("Ahora el diccionario queda : {}".format(futbolistas))

# Añadimos, de otro modo, un elemento al diccionario
## si la clave ya existe, el valor se cambia por este nuevo
futbolistas[22] = 'Navas'
print ("\nAñadimos un nuevo elemento, ahora el diccionario queda: ")
print (futbolistas)
```

El futbolista con clave 6 es Iniesta

Insertamos el elemento con clave 10 y valor Cesc

Ahora el diccionario queda : {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7:

Añadimos un nuevo elemento, ahora el diccionario queda:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
# Eliminamos un elemento del diccionario dada su clave
futbolistas.pop(22)
print ("\nEliminamos el elemento con clave 22")
print ("Ahora el diccionario queda: {}".format(futbolistas))

# Hacemos una copia del diccionario
futbolistas_copia = futbolistas.copy()
print ("\nLa copia del diccionario tiene los valores:")
print (futbolistas_copia)

# Borramos los elementos de un diccionario
futbolistas_copia.clear()
print ("\nVaciamos el diccionario nuevo creado, ahora los valores en el: {}".format(fu
```

Eliminamos el elemento con clave 22

Ahora el diccionario queda: {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: '

La copia del diccionario tiene los valores:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

Vaciamos el diccionario nuevo creado, ahora los valores en el: {}

```
# Comprobamos si existe o no una clave en un diccionario
if 2 in futbolistas:
    print ("\nEl futbolista con la clave 2 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 2 NO existe en el diccionario.")

if 8 in futbolistas:
    print ("\nEl futbolista con la clave 8 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 8 NO existe en el diccionario.")
```

El futbolista con la clave 2 NO existe en el diccionario.

El futbolista con la clave 8 existe en el diccionario.

```
# Creamos un diccionario a partir de una lista de claves
keys = ['nombre', 'apellidos', 'edad']
datos_usuario = dict.fromkeys(keys, 'null')
print ("\nCreamos un diccionario a partir de la lista de claves:")
print (keys)
print ("y con valor 'null'")
print ("Así el diccionario queda: {}".format(datos_usuario))

# Creación de un diccionario
diccionario=dict({1:'rei',2:'pepe',3:'javier'})
print(diccionario)
```

Creamos un diccionario a partir de la lista de claves:

```
['nombre', 'apellidos', 'edad']
y con valor 'null'
Así el diccionario queda: {'nombre': 'null', 'apellidos': 'null', 'edad': 'null'}
{1: 'rei', 2: 'pepe', 3: 'javier'}
```

```
# Devolvemos los elementos del diccionario en una tupla
tupla = futbolistas.items()
print ("\nEl diccionario convertido en tupla queda así:")
print (tupla)

# Unimos dos diccionarios existentes
suplentes = {
    4:'Marchena', 12:'Valdes', 13:'Mata',
    17:'Arbeloa', 19:'Llorente', 20:'Javi Martinez',
    21:'Silva', 23:'Reina'
}

print ("\nUnimos el diccionario: ")
print (futbolistas)
futbolistas.update(suplentes) # Aquí hacemos la unión de los diccionarios
print ("con el diccionario:")
print (suplentes)
print ("siendo el resultado:")
print (futbolistas)
```

El diccionario convertido en tupla queda así:
dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa'),

Unimos el diccionario:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',
con el diccionario:
{4: 'Marchena', 12: 'Valdes', 13: 'Mata', 17: 'Arbeloa', 19: 'Llorente', 20: 'Javi Mar
siendo el resultado:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
#Entrada de datos
x=input('ingrese el dato 1: ');
y=input('ingrese el dato 2: ');
print('tipo de datos',type(x),type(y))
print('suma de datos sin casteo',x+y)
print('suma de datos con casteo',float(x)+float(y))
```

```
ingrese el dato 1: 5
ingrese el dato 2: 6
tipo de datos <class 'str'> <class 'str'>
suma de datos sin casteo 56
suma de datos con casteo 11.0
```

Operador	Descripción	Ejemplo
<code>==</code>	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
<code>!=</code>	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<code><</code>	¿es a menor que b?	<code>r = 5 < 3 # r es False</code>
<code>></code>	¿es a mayor que b?	<code>r = 5 > 3 # r es True</code>
<code><=</code>	¿es a menor o igual que b?	<code>r = 5 <= 5 # r es True</code>
<code>>=</code>	¿es a mayor o igual que b?	<code>r = 5 >= 3 # r es True</code>

```
a = 5
b = 5
a1 = 7
b1 = 3
c1 = 3

cadena1 = 'Hola'
cadena2 = 'Adios'

# igual
c = a == b
print (c)

cadenas = cadena1 == cadena2
print (cadenas)

# diferente
d = a1 != b
print (d)

cadena0 = cadena1 != cadena2
print (cadena0)

# mayor que
e = a1 > b1
print (e)

# menor que
f = b1 < a1
print (f)

# mayor o igual que
g = b1 >= c1
print (g)

# menor o igual que
h = b1 <= c1
print (h)
```

```
True
False
True
True
True
True
True
True
```

```
5==5
```

True

```
7!=9
```

True

```
cadena1 = 'Hola'
```

```
cadena1
```

'Hola'

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

```
x=3
y=7
if x==3 and y==7:
    print('si')
else:
    print('no')
```

si

```

if condicion1:
    instruccion1
    instruccion2
else:
    if condición2:
        instruccion3
        instruccion4
    else:
        instruccion5
        instruccion6
# fin del if
instruccion7
instruccion8
'
```

```

if 5>2:
    print('si')
    print('si2')
elif 1>3:
    print('sino si')
else:
    print('no')
    print('PorFuera')
print('PorFuera2')

```

si
si2
PorFuera2

Ejercicios

- Realice un programa que pida la edad de una persona en años. Si la edad es mayor o igual a 18, el programa debe imprimir la cadena: 'ADULTO'. Si la edad es menor a 18 se debe imprimir: 'MENOR DE EDAD'.
- Diseñe un algoritmo que determine si un número es o no es, par positivo.
- Diseñe un algoritmo que lea un número de tres cifras y determine si es igual al revés del número.

Escribe aquí tu código

```
#ciclo for
print("Ciclo for en un rango de datos: ")
for i in range(2,10):
    print(i)

print('Ciclo for en una colección de datos:')
x=[3,4,5,7,8,"Hola"]
for i in x:
    print("El elemento es= ",i)
for i in range(len(x)):
    print("El elemento de la posición",i,'es:',x[i])
```

Ciclo for en un rango de datos:

2
3
4
5
6
7
8
9

Ciclo for en una colección de datos:

El elemento es= 3
El elemento es= 4
El elemento es= 5
El elemento es= 7
El elemento es= 8
El elemento es= Hola
El elemento de la posición 0 es: 3
El elemento de la posición 1 es: 4
El elemento de la posición 2 es: 5
El elemento de la posición 3 es: 7
El elemento de la posición 4 es: 8
El elemento de la posición 5 es: Hola

```
print('Ciclo for en un diccionario:')
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
for nombre, color in frutas.items():
    print (nombre, "es de color", color)
for llave in frutas:
    print(llave, 'es de color', frutas[llave])
```

Ciclo for en un diccionario:

Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa

frutas

```
{'Fresa': 'roja',
'Limon': 'verde',
'Papaya': 'naranja',
'Manzana': 'amarilla',
'Guayaba': 'rosa'}
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
frutas
```

```
{'Fresa': 'roja',
'Limon': 'verde',
'Papaya': 'naranja',
'Manzana': 'amarilla',
'Guayaba': 'rosa'}
```

len(frutas)

5

```
frutas.get('Fresa')
```

'roja'

```
for i in range(10):
    print(i)
```

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

```
v=[ 'a' , 'b' , 'c' , 'd' ]

for i in range(len(v)):
    print(i,"-->",v[i])
```

```
0 --> a
1 --> b
2 --> c
3 --> d
```

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

```
a
b
c
d
```

```
#ciclo while
cont=0
while cont<10:
    print(cont)
    cont+=2
```

```
0  
2  
4  
6  
8
```

```
v=['a','b','c','d']
```

```
for i in range(len(v)):  
    print(i,v[i])
```

```
0 a  
1 b  
2 c  
3 d
```

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

```
a  
b  
c  
d
```

Ejercicio

- Realice un programa que lea las notas de un estudiante, despues devolver el promedio, la mejor y la peor nota.

```
# Escribe aquí tu código
```

Algoritmo Multiplicación 1

Multiplica los numeros naturales **x** y **w** **w**, sumando **x** veces el número número; el resultado lo asigna a la variable **z**

Nociones fundamentales

Multiplicar dos números naturales x y y , y denotar su producto por z

Paso 1: $z := 0;$

$u := x; \quad \rightsquigarrow x = 5$

Paso 2: **repetir las instrucciones**

$z := z + y; \quad \rightsquigarrow y = 13$

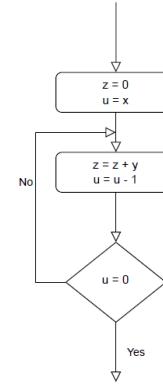
$u := u - 1;$

hasta que $u = 0$

Valor de las variables

Paso	z	u
1	0	5
2	13	4
2	26	3
2	39	2
2	52	1
2	65	0

Multiplicación de dos números naturales x y y



Pedimos introducir el primer número para multiplicar

```
x=int(input('Escribir un numero para multiplicar: '))
```

Escribir un numero para multiplicar: 6

Pedimos introducir el segundo número para multiplicar

```
w=int(input('Escribir otro numero para multiplicar: '))
```

Escribir otro numero para multiplicar: 5

Inicialización y asignación de variable

```
z=0
```

```
u=x
```

Realizamos la operación de multiplicación con sumas y estructuras repetitivas

```
while u>0:
    z=z+w
    u=u-1
```

Mostramos el resultado

```
print('El resultado de la multiplicacion es: ', z)
```

El resultado de la multiplicacion es: 30

Condensando todo el algoritmos en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir un numero para multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
paso1=1
z=0
u=x
paso2=2
print('Paso', ' ', 'Z', ' ', 'U')
print(paso1, ' ', z, ' ', u)
while u>0:
    z=z+w
    u=u-1
    print(paso2, ' ', z, ' ', u)
print('El resultado de la multiplicacion es: ', z)
```

Escribir un numero para multiplicar: 5
 Escribir otro numero para multiplicar: 6
 Paso Z U
 1 0 5
 2 6 4
 2 12 3
 2 18 2
 2 24 1
 2 30 0
 El resultado de la multiplicacion es: 30

Algoritmo División 1

Divide dos números naturales **x** y **w**, utilizando restas; el cociente es asignado a la variable **q** y el residuo a **r**.

Dividir el número natural x por el número natural y , y denotar por q el cociente y el residuo por r .

Teorema del cociente y del residuo:

Dado cualquier entero a y un entero positivo b existen dos enteros únicos q y r tales que:

$$a = q * b + r$$

donde $0 \leq r \leq b$. q se llama el cociente y r el residuo.

Notación: $q = a \text{ div } b$ y $r = a \text{ mod } b$

$$a = (a \text{ div } b) * b + (a \text{ mod } b)$$

División entera de x entre y : el cociente q es el número de veces que se puede restar (sustraer) el divisor y al dividendo x de tal forma que el resultado (cociente) sea no negativo.

Dividir el número natural x por el número natural y , y denotar por q el cociente y el residuo por r .

Paso 1: $q := 0;$

$$r := x; \quad \rightsquigarrow x = 100$$

Paso 2: mientras que $r \geq y$ repetir

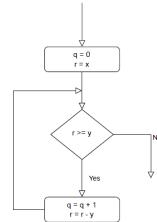
$$q := q + 1;$$

$$r := r - y; \quad \rightsquigarrow y = 15$$

Valor de las variables

Paso	q	r
1	0	100
2	1	85
2	2	70
2	3	55
2	4	40
2	5	25
2	6	10

División de dos números naturales x y y



Pedimos introducir el dividendo de la división

```
x=int(input('Escribir el dividendo: '))
```

Escribir el dividendo: 9

Pedimos introducir el divisor de la división

```
w=int(input('Escribir el divisor: '))
```

Escribir el divisor: 2

Inicialización y asignación de variable

```
q=0
r=x
```

Realizamos la operación de división con restas y estructuras repetitivas

```
while r>=w:
    q=q+1
    r=r-w
```

Mostramos el cociente y el residuo

```
print('El cociente de la división es: ',q,'\\n El residuo de la división es: ',r)
```

```
El cociente de la división es: 4
El residuo de la división es: 1
```

Condensando todo el algoritmo en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir el dividendo: '))
w=int(input('Escribir el divisor: '))
paso1=1
q=0
r=x
paso2=2
print('Paso', ' ', 'q', ' ', 'r')
print(paso1, ' ', q, ' ', r)
while r>=w:
    q=q+1
    r=r-w
    print(paso2, ' ', q, ' ', r)
print('El cociente de la división es: ',q, '\n El residuo de la división es: ',r)
```

```
Escribir el dividendo: 9
Escribir el divisor: 2
Paso   q   r
1     0   9
2     1   7
2     2   5
2     3   3
2     4   1
El cociente de la división es: 4
El residuo de la división es: 1
```

Python es un lenguaje **indentado**, no usa corchetes para delimitar el alcance de las estructuras de programación sino que se fija en los **cambios de indentación**.

No se declara el tipo de los argumentos de las funciones. La semántica de la implementación ha de estar preparada para funcionar con los tipos de datos que quieras.

```
import numpy as np
def funcion_1(a,b):
    r = a**2
    return r+b

def greatest(a,b):
    if a>b:
        return a
    else:
        return b
```

```
funcion_1 (10.4,2)
```

```
110.16000000000001
```

```
funcion_1 (10.4, np.array([2,4]))
```

```
array([110.16, 112.16])
```

```
funcion_1(np.array([1,5]),np.array([3,2]))
```

```
array([ 4, 27])
```

```
m1 = np.array([[3,4],[1,1]])  
m2 = np.array([[5,6],[0,0]])
```

```
funcion_1 (m1,m2)
```

```
array([[14, 22],  
       [ 1,  1]])
```

```
greatest(10,2)
```

```
10
```

Podemos definir valores por defecto para los argumentos de las funciones y llamarlas usando explícitamente el nombre de los argumentos. Además, las funciones pueden devolver varios valores.

```
def f_power(x, p=2):  
    return x**p
```

```
f_power(x=3)
```

9

```
f_power(p=4, x=3)
```

81

```
def f_power(x, p=2):  
    return x**p, x*p
```

```
r = f_power(p=4, x=3)  
print(r[0],r[1],"\n")
```

81 12

```
r1, r2 = f_power(p=4, x=3)  
print(r1)  
print(r2, "\n")
```

81

12

```
a,b = 10, np.array([10,4,-3])  
print(a)  
print(b)
```

10

[10 4 -3]

```
def f_power(x, p=2):  
    return x**p  
  
def f_powers(x, p1=2, p2=3):  
    return x**p1, x**p2
```

```
f_power(4)
```

```
16
```

```
f_power(4,3)
```

```
64
```

```
f_powers(4, p1=3)
```

```
(64, 64)
```

```
xp1, xp2 = f_powers(4, p2=4, p1=3)
print("power1",xp1, "power2", xp2)
```

```
power1 64 power2 256
```

Dependiendo del tipo de dato que enviamos a la **función**, podemos diferenciar dos comportamientos:

Paso por valor: Se crea una copia local de la variable dentro de la función.

Paso por referencia: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

Los tipos simples se pasan por valor: Enteros, flotantes, cadenas, lógicos...

Los tipos compuestos se pasan por referencia: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(x):
    x = 2*x
    return x
```

```
x=10
doblar_valor(x)
print(x,doblar_valor(x))
```

10 20

Para modificar los tipos simples podemos devolverlos modificados y reasignarlos:

```
x = 10
x = doblar_valor(x)
print(x)
```

20

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(y):
    for i,n in enumerate(y):
        y[i] = 2*y[i]
```

```
y=[1,2,3]
```

```
doblar_valores(y)
```

```
print(y)
```

[2, 4, 6]

Algoritmo multiplicación 2

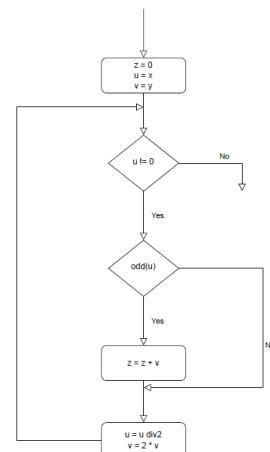
Otro algoritmo para la multiplicación de dos números naturales

Paso 1: $z := 0; u := x; v := y$; $\rightsquigarrow x = 9, y = 5$
 Paso 2: mientras que $u \neq 0$ repetir
 si $\text{odd}(u)$ entonces $z := z + v$;
 $u := u \text{ div } 2$;
 $v := 2 * v$

Valor de las variables			
Paso	z	u	v
1	0	9	5
2	5	4	10
2	5	2	20
2	5	1	40
2	45	0	80

$$9 = 2 * 2 * 2 + 1 \quad 2(2(2(5))) + 5 = 45$$

Otro algoritmo para la multiplicación de dos números naturales



```
# función para determinar el cociente
def funcionCociente(x,w):
```

```
    q=0
    r=x
    while r>=w:
        q=q+1
        r=r-w
    return q
```

```
# función para determinar el residuo
def funcionResiduo(x,w):
```

```
    q=0
    r=x
    while r>=w:
        q=q+1
        r=r-w
    return r
```

```
# función para determinar si un numero es par
def funcionPar(u):
```

```
    if funcionResiduo(u,2)==0:
        par=True
    else:
        par=False
    return par
```

```
def funcionMultiplicar(x,w):
    z=0
    u=x
    v=w
    while u!=0:
        if not funcionPar(u):
            z=z+v
        u=funcionCociente(u,2)
        v=2*v
    return z
```

```
x=int(input('Escribir un numero par multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
print('El resultado de la mutiplicación es: ', funcionMultiplicar(x,w))
```

Escribir un numero par multiplicar: 6
 Escribir otro numero para multiplicar: 5
 El resultado de la mutiplicación es: 30

Catching

```
int(3.2)
```

3

```
int("hola")
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-2-62025142aa33> in <cell line: 1>()
----> 1 int("hola")

ValueError: invalid literal for int() with base 10: 'hola'
```

Manejar cualquier tipo de error que pase en un bloque de código

```
def ejemplo_try_except(valor):
    try:
        resultado = 10 / valor
        print("El resultado es:", resultado)
    except Exception as e:
        print("Ha ocurrido un error:", e)

# Ejemplos de llamadas a la función
ejemplo_try_except(2)
ejemplo_try_except(0)
ejemplo_try_except('cadena')
```

El resultado es: 5.0
 Ha ocurrido un error: division by zero
 Ha ocurrido un error: unsupported operand type(s) for /: 'int' and 'str'

Manejando un tipo específico de error

```
def int_times_two(x):
    try:
        return(int(x)*2)
    except ValueError:
        return 0
```

int_times_two(2)

4

int_times_two(2.5)

4

int_times_two("hola")

0

Manejo de la sentencia finally para que, pase o no pase un error, siempre ejecute lo que se coloque en la sentencia finally

```
def division_segura(dividendo, divisor):
    try:
        resultado = dividendo / divisor
        return resultado
    except ZeroDivisionError:
        print("¡Error! No puedes dividir entre cero.")
        return None
    finally:
        print("Operación finalizada.")
```

```
dividendo = 10
divisor = 2

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

Operación finalizada.
El resultado de la división es: 5.0

```
dividendo = 10
divisor = 0

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

¡Error! No puedes dividir entre cero.
Operación finalizada.

Raising

Este tipo de sentencias se utilizan para generar errores personalizados dentro del código

```
def get_positive_integer_from_user():
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
    print ("thanks!")
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: 1
thanks!
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: -1
```

```
-----
ValueError                                                 Traceback (most recent call last)
<ipython-input-13-ea5635cae962> in <cell line: 1>()
----> 1 get_positive_integer_from_user()

<ipython-input-11-a7d1640efb92> in get_positive_integer_from_user()
      2     a = int(input("Enter a positive integer: "))
      3     if a <= 0:
----> 4         raise ValueError("That is not a positive number!")
      5     print ("thanks!")

ValueError: That is not a positive number!
```

Introducción a Pandas

Contenido

- Series de pandas
- DataFrames de pandas
- Cargar Fuentes externas de datos

Pandas es una biblioteca de Python fundamental para la manipulación y el análisis de datos, ampliamente utilizada en el ámbito de la ciencia de datos y la ingeniería de datos. Proporciona estructuras de datos fáciles de usar y de alto rendimiento, como DataFrames y Series, que permiten realizar operaciones complejas de limpieza, transformación, y análisis de datos con una sintaxis intuitiva y eficiente.

```
#  
# ¿Cómo importamos la librería?  
import pandas as pd
```

Series de pandas

Una pandas Series es una estructura de datos unidimensional en la biblioteca pandas de Python, similar a una columna en una tabla de datos o a un array en otros lenguajes de programación. Cada Series tiene un índice asociado, que son las etiquetas de las filas que permiten acceder a los elementos de la serie de manera rápida y eficiente. Los elementos de una Series pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

```
pd.Series([-2, -3, 0, 3, 2], dtype='int8')
```

↓

		Data
Index	0	-2
	1	-3
	2	0
	3	3
	4	2

dtype: int8

© w3resource.com

Lo que hace que las Series sean especialmente poderosas es su capacidad para manejar datos faltantes y realizar operaciones vectorizadas, lo que significa que puedes aplicar operaciones a todos los elementos de una Series de una sola vez, en lugar de tener que iterar sobre cada elemento individualmente. Además, las Series ofrecen una variedad de métodos y propiedades para realizar operaciones estadísticas, manipulación de datos, y transformaciones, lo que las convierte en una herramienta versátil y esencial para el análisis de datos en Python.

```
# Vamos a crear una serie a partir de un diccionario

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}
```

```
# Para crear una serie se debe partir de un objeto de tipo diccionario
balances = pd.Series(test_balance_data)
balances
```

```

pasan      20.00
treasure   20.18
ashley     1.05
craig      42.42
dtype: float64
```

```
# Acá tenemos una serie de pandas, que es una estructura de datos que se parece a un diccionario.  
# las etiquetas de las filas se obtienen  
balances.keys()  
# tal cual como el un diccionario.
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Alternativamente, se pueden obtener con el método index  
balances.index
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Si solo quieres los valores, puedes usar:  
balances.values
```

```
array([20. , 20.18, 1.05, 42.42])
```

```
# Ahora, creemos una serie a partir de una lista
```

```
unlabeled_balances = pd.Series([20.00, 20.18, 1.05, 42.42])  
unlabeled_balances # Note que las etiquetas de las filas son números enteros que van de 0 a 3
```

```
0    20.00  
1    20.18  
2     1.05  
3    42.42  
dtype: float64
```

```
# Si quieres personalizar esos index, puedes hacerlo de la siguiente manera  
labeled_balances = pd.Series(  
    [20.00, 20.18, 1.05, 42.42],  
    index=['pasan', 'treasure', 'ashley', 'craig'])  
labeled_balances
```

```
pasan      20.00
treasure   20.18
ashley     1.05
craig      42.42
dtype: float64
```

```
# Si pasas un escalar, recuerda que es un valor único,
# este se transmitirá a cada una de las claves especificadas en el argumento de palabras
pd.Series(20.0, index=["guil", "jay", "james", "ben", "nick"])
```

```
guil      20.0
jay       20.0
james    20.0
ben       20.0
nick     20.0
dtype: float64
```

Acceso a los elementos de una Serie

```
import pandas as pd

# Creamos una serie a partir de un diccionario que tiene nombres de personas como claves
test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

balances = pd.Series(test_balance_data)
```

Acceso por indexación como lista

```
# Una Serie es ordenada e indexada (como las listas de Python)
print(balances[0])
print(type(balances[0]))
```

```
20.0
<class 'numpy.float64'>
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:2: FutureWarning: print(balances[0])
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:3: FutureWarning: print(type(balances[0]))
```

```
# La forma correcta es:
balances.iloc[0]
```

```
np.float64(20.0)
```

```
# Si quieres obtener el ultimo balance
balances[-1]
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/3307448855.py:2: FutureWarning: balances[-1]
```

```
np.float64(42.42)
```

Acceso por la clave de índice

```
# Acceder a un elemento por su clave(index según pandas)
balances['pasan']
```

```
np.float64(20.0)
```

```
# Las series de pandas se comportan como diccionarios
for label,value in balances.items():
    print(label, value)
```

```
pasan 20.0
treasure 20.18
ashley 1.05
craig 42.42
```

```
# Acceder a un elemento usando su clave como si fuera un atributo
balances.ashley
```

```
np.float64(1.05)
```

```
# Acceder a los elementos explicitamente como lo menciona la documentación
print(balances.loc['ashley'])
print(balances.iloc[0])
```

```
1.05
20.0
```

```
# Acceder a una porción de la serie
balances[['treasure','craig']]
```

```
treasure    20.18
ashley      1.05
craig       42.42
dtype: float64
```

```
balances.iloc[0:3]
```

```
pasan      20.00
treasure   20.18
ashley     1.05
dtype: float64
```

Vectorización y broadcasting

se trata de la capacidad de pandas para realizar operaciones element-wise (elemento a elemento) en estructuras de datos con diferentes formas, alineando automáticamente las

dimensiones según sea necesario.

```
import pandas as pd

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

test_deposit_data = {
    'pasan': 20,
    'treasure': 10,
    'ashley': 100,
    'craig': 55
}

balances = pd.Series(test_balance_data)
deposits = pd.Series(test_deposit_data)
```

Vectorización

Aunque es posible recorrer cada elemento y aplicarlo a otro...

```
for label, value in deposits.items():
    print(f"Depositando {value} en la cuenta de {label}, que tiene {balances[label]}")
    balances[label] += value

balances
```

Depositando 20 en la cuenta de pasan, que tiene 20.0
 Depositando 10 en la cuenta de treasure, que tiene 20.18
 Depositando 100 en la cuenta de ashley, que tiene 1.05
 Depositando 55 en la cuenta de craig, que tiene 42.42

pasan	40.00
treasure	30.18
ashley	101.05
craig	97.42
dtype: float64	

...es importante recordar apoyarse en la vectorización y omitir los bucles por completo. La vectorización es más rápida y, como puedes ver, más fácil de leer y escribir.

```
# Se debe considerar que para esta operacion los indices deben coincidir, de lo contrario se genera un error
balances += deposits
balances
# agrega 10 a la cuenta de 'james' en el diccionario de depositos para y ejecuta nueva operacion
```

```
pasan      60.00
treasure    40.18
ashley     201.05
craig      152.42
dtype: float64
```

```
balances -= deposits
balances
```

```
pasan      40.00
treasure    30.18
ashley     101.05
craig      97.42
dtype: float64
```

Broadcasting

Esto lo que permite es sumar un escalar a cada uno de los elementos de la serie.

```
balances + 5
```

```
pasan      45.00
treasure    35.18
ashley     106.05
craig      102.42
dtype: float64
```

Hacer broadcast de una serie con otra serie es posible siempre y cuando ambas tengan el mismo índice. En caso de que no sea así, se devolverá NaN.

```
coupons = pd.Series(1, ['craig', 'ashley', 'james'])  
coupons
```

```
craig      1  
ashley     1  
james      1  
dtype: int64
```

```
# Sumemos los balances con los cupones, el resultado es una nueva serie  
balances + coupons
```

```
ashley    102.05  
craig     98.42  
james      NaN  
pasan      NaN  
treasure   NaN  
dtype: float64
```

```
balances
```

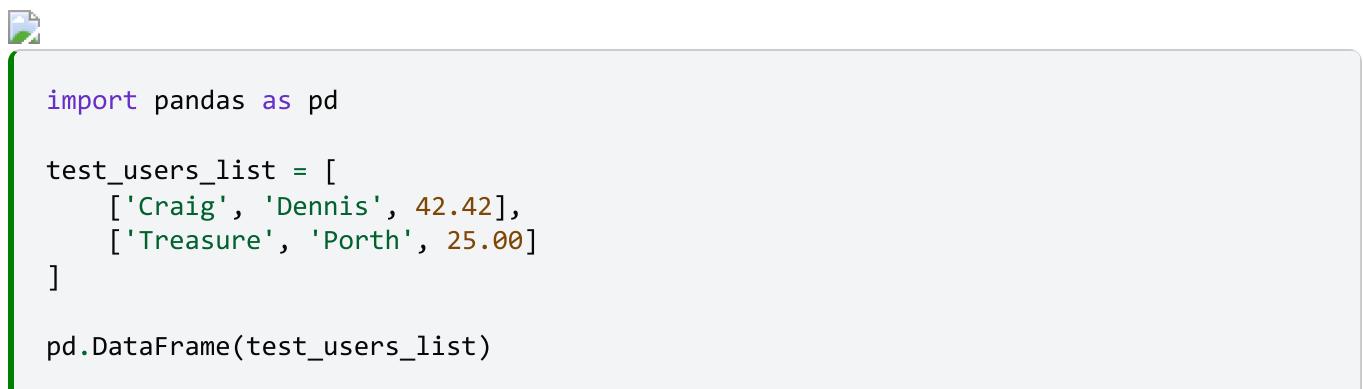
```
pasan     40.00  
treasure  30.18  
ashley    101.05  
craig     97.42  
dtype: float64
```

```
balances.add(coupons, fill_value=0) # Así se puede evitar el NaN y que no se pierda la
```

```
ashley    102.05  
craig     98.42  
james     1.00  
pasan     40.00  
treasure  30.18  
dtype: float64
```

DataFrames de pandas

Un DataFrame es una estructura de datos bidimensional en la biblioteca pandas de Python, similar a una tabla de datos o una hoja de cálculo en Excel. Cada DataFrame tiene un índice asociado, que son las etiquetas de las filas, y columnas, que son las etiquetas de las columnas. Los elementos de un DataFrame pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.



```
import pandas as pd

test_users_list = [
    ['Craig', 'Dennis', 42.42],
    ['Treasure', 'Porth', 25.00]
]

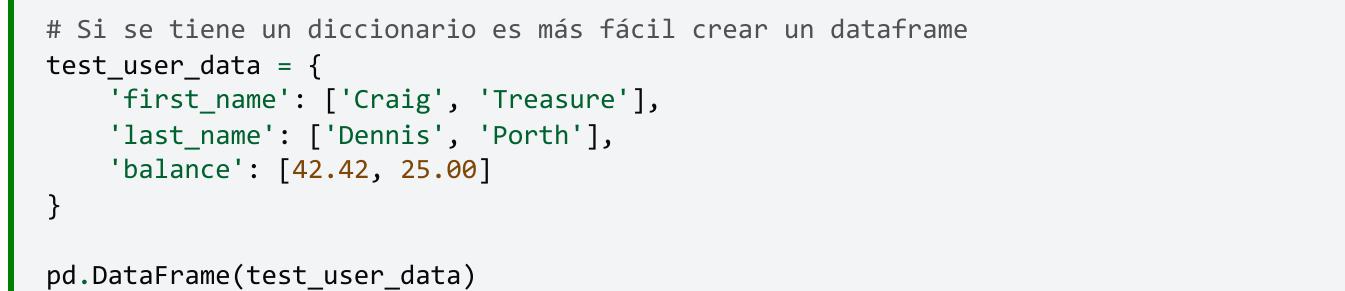
pd.DataFrame(test_users_list)
```

	0	1	2
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00



```
pd.DataFrame(test_users_list, index=['craigsdennis', 'treasure'],
             columns=['first_name', 'last_name', 'balance']) # Note que acá se especifica el índice y las columnas
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00



```
# Si se tiene un diccionario es más fácil crear un dataframe
test_user_data = {
    'first_name': ['Craig', 'Treasure'],
    'last_name': ['Dennis', 'Porth'],
    'balance': [42.42, 25.00]
}

pd.DataFrame(test_user_data)
```

	first_name	last_name	balance
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00

```
# Se puede especificar los indices de las filas
pd.DataFrame(test_user_data, index=['craigslist', 'treasure'])
```

	first_name	last_name	balance
craigslist	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00

Acceso a los elementos de un DataFrame

```
# Vamos a crear un dataframe con algunos datos de usuarios
import pandas as pd

test_user_data = { # Estos son los datos de los usuarios y el balance
    'first_name': ['Craig', 'Treasure', 'Ashley', 'Guil'],
    'last_name': ['Dennis', 'Porth', 'Boucher', 'Hernandez'],
    'balance': [42.42, 25.00, 2.02, 87.00]
}
test_user_names = ['craigslist', 'treasure', 'lindsay2000', 'guil'] # Supongamos que

users = pd.DataFrame(test_user_data, index=test_user_names)
users
```

	first_name	last_name	balance
craigslist	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00
lindsay2000	Ashley	Boucher	2.02
guil	Guil	Hernandez	87.00

Obtener una columna específica

Cada columna en un DataFrame es una Serie de pandas, por lo que puedes acceder a una columna específica utilizando la notación de corchetes y el nombre de la columna y el retorno es una serie.

```
balances = users['balance']  
balances
```

```
craigsdennis    42.42  
treasure        25.00  
lindsay2000     2.02  
guil            87.00  
Name: balance, dtype: float64
```

```
# La serie que retorna la celda anterior tiene una propiedad llamada name que retorna  
balances.name
```

```
'balance'
```

Puedes obtener una fila de un Dataframe usando la propiedad .loc[] y pasando el índice de la fila, o el número.

```
users.loc['guil']
```

```
first_name        Guil  
last_name        Hernandez  
balance          87.0  
Name: guil, dtype: object
```

```
users.iloc[3]
```

```
first_name        Guil  
last_name        Hernandez  
balance          87.0  
Name: guil, dtype: object
```

Recuperar un valor específico mediante encadenamiento

```
users['first_name']['craigdennis']
```

```
'Craig'
```

```
users.loc['craigdennis']['first_name']
```

```
'Craig'
```

```
users.loc['craigdennis', 'first_name']
```

```
'Craig'
```

```
users.at['craigdennis', 'first_name']
```

```
'Craig'
```

Usando las propiedades loc e iloc puedes dividir un DataFrame existente en uno nuevo.

En el ejemplo a continuación, usamos : en el eje de filas para seleccionar todas las filas, y especificamos qué columnas queremos recuperar usando una lista en el eje de columnas

```
# Esto se lee como:  
# En el dataframe users, selecciona todas las filas, y dame solo las columnas 'balance'  
users.loc[:, ['balance', 'last_name']]
```

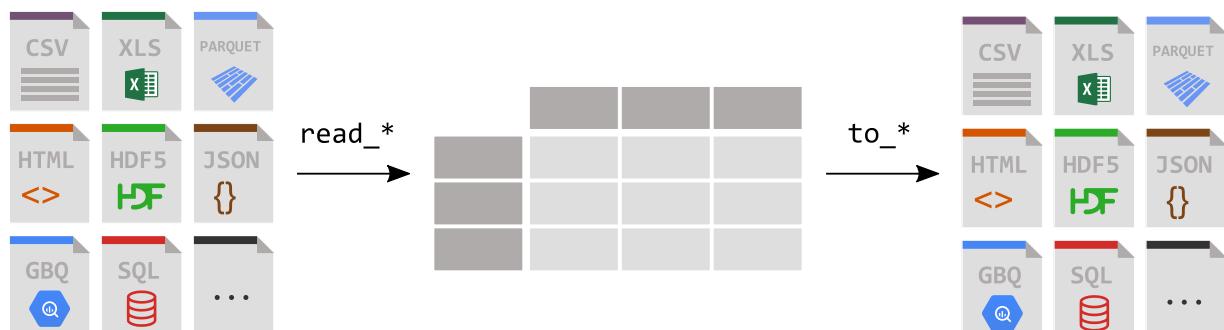
	balance	last_name
craigslist	42.42	Dennis
treasure	25.00	Porth
lindsay2000	2.02	Boucher
guil	87.00	Hernandez

```
# O lo puedes indexar por posición como en las listas
users.iloc[1:3, 1:]
```

	last_name	balance
treasure	Porth	25.00
lindsay2000	Boucher	2.02

Cargar Fuentes externas de datos

Naturalmente, cuando se trabaja con datos en la vida real, no siempre se tienen los datos en un DataFrame de pandas. A menudo, los datos se almacenan en archivos CSV, Excel, bases de datos SQL, o en la web. Afortunadamente, pandas proporciona una variedad de funciones para cargar datos desde diferentes fuentes y formatos, lo que facilita la importación y exportación de datos en Python.



La sintaxis general para cargar datos en un DataFrame

de pandas es la siguiente:

```
import pandas as pd
pd.read_<formato>('ruta/al/archivo')
```

La sintaxis general para guardar datos desde un DataFrame de pandas es la siguiente:

```
import pandas as pd
df.to_<formato>('ruta/al/archivo')
```

Los formatos más comunes para cargar y guardar datos en pandas son los siguientes:

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

```
# Vamos a importar los datos de un archivo CSV
# 1. Veamos lo que tiene el archivo users.csv
!head -n 5 data/users.csv # Este comando solo es una muestra del contenido del archivo
```

```
,first_name,last_name,email,email_verified,signup_date,referral_count,balance
aaron,Aaron,Davis,aaron6348@gmail.com,True,2018-08-31,6,18.14
acook,Anthony,Cook,cook@gmail.com,True,2018-05-12,2,55.45
adam.saunders,Adam,Saunders,adam@gmail.com,False,2018-05-29,3,72.12
adrian,Adrian,Fang,adrian.fang@teamtreehouse.com,True,2018-04-28,3,30.01
```

```
import pandas as pd

my_data = pd.read_csv('data/users.csv')
my_data # Note que hay una columna sin nombre, que es el índice y carga como Unnamed:
```

	Unnamed: 0	first_name	last_name	email	email_verified
0	aaron	Aaron	Davis	aaron6348@gmail.com	
1	acook	Anthony	Cook	cook@gmail.com	
2	adam.saunders	Adam	Saunders	adam@gmail.com	
3	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	
4	adrian.blair	Adrian	Blair	adrian9335@gmail.com	
...
470	wilson	Robert	Wilson	robert@yahoo.com	
471	wking	Wanda	King	wanda.king@holt.com	
472	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	
473	young	Jessica	Young	jessica4028@yahoo.com	
474	zachary.neal	Zachary	Neal	zneal@gmail.com	

475 rows × 8 columns

```
# Usemos los parámetros para cargar el archivo CSV
my_data = pd.read_csv('data/users.csv',index_col=0)
# Otros parametros que se pueden usar son:

# sep: el separador de los datos. Algunos archivos CSV usan ; en lugar de ,
# header: la fila que se usará como encabezado.
# names: una lista con los nombres de las columnas.
# skiprows: las filas que se deben omitir
# na_values: los valores que se deben tratar como NaN
# nrows: el número de filas que se deben leer
# encoding: la codificación de caracteres que se debe usar

# Existen otros parámetros que se pueden usar, pero estos son los más comunes.
# Vamos a ver otros parámetros a lo largo de los ejemplos.

my_data
```

	first_name	last_name	email	email_verified	
	aaron	Aaron	Davis	aaron6348@gmail.com	True
	acook	Anthony	Cook	cook@gmail.com	True
	adam.saunders	Adam	Saunders	adam@gmail.com	False
	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	True
	adrian.blair	Adrian	Blair	adrian9335@gmail.com	True

	wilson	Robert	Wilson	robert@yahoo.com	False
	wking	Wanda	King	wanda.king@holt.com	True
	wright3590	Jacqueline	Wright	jacqueline.wright@gonzalez.com	True
	young	Jessica	Young	jessica4028@yahoo.com	True
	zachary.neal	Zachary	Neal	zneal@gmail.com	True

475 rows × 7 columns

Exploraremos los datos que importamos

La forma más fácil si cargamos correctamente, podemos usar el método `.head()` para ver las primeras filas de un DataFrame, o `.tail()` para ver las últimas filas.

```
my_data.head() # Muestra las primeras 5 filas del dataframe
```

	first_name	last_name	email	email_verified	
	aaron	Aaron	Davis	aaron6348@gmail.com	True
	acook	Anthony	Cook	cook@gmail.com	True
	adam.saunders	Adam	Saunders	adam@gmail.com	False
	adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	True
	adrian.blair	Adrian	Blair	adrian9335@gmail.com	True

```
my_data.head(3)
```

	first_name	last_name	email	email_verified	signup
aaron	Aaron	Davis	aaron6348@gmail.com	True	2018-
acook	Anthony	Cook	cook@gmail.com	True	2018-
adam.saunders	Adam	Saunders	adam@gmail.com	False	2018-

```
# Revisemos la cantidad de muestras que tiene el dataframe  
len(my_data)
```

475

```
my_data.shape # Shape es un atributo que tiene todo DF (dataFrame) y retorna una tupla
```

(475, 7)

Explorando la información

Info

El método `.info()` proporciona una descripción concisa de un DataFrame, incluyendo el número de filas y columnas, el número de valores no nulos, el tipo de datos de cada columna, y la cantidad de memoria utilizada.

```
my_data.info() # Info es un método que retorna información sobre el dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 475 entries, aaron to zachary.neal
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   first_name    475 non-null    object  
 1   last_name     430 non-null    object  
 2   email         475 non-null    object  
 3   email_verified 475 non-null    bool    
 4   signup_date   475 non-null    object  
 5   referral_count 475 non-null    int64  
 6   balance       475 non-null    float64 
dtypes: bool(1), float64(1), int64(1), object(4)
memory usage: 26.4+ KB
```

Counts

El método `.count()` cuenta, por cada columna, cuantos valores no nulos hay.

```
my_data.count()
```

```
first_name    475
last_name     430
email         475
email_verified 475
signup_date   475
referral_count 475
balance       475
dtype: int64
```

La mayoría de nuestras columnas incluyen valores para cada fila, pero parece que `last_name` tiene algunos valores faltantes. Los datos faltantes aparecerán como `np.nan` – no es un número de NumPy – en esos registros.

Dtype

```
# Revisemos los tipos de datos de las columnas
my_data.dtypes
```

```
first_name      object
last_name       object
email           object
email_verified   bool
signup_date     object
referral_count  int64
balance         float64
dtype: object
```

a mayoría de los tipos de datos de estas columnas fueron inferidos o asumidos correctamente. Observa cómo email_verified es automáticamente bool, referral_count es un entero y balance un flotante. Esto es lo que hace pandas cuando usamos pd.read_csv.

Sin embargo, una cosa a tener en cuenta es que el campo signup_date es un objeto y no un datetime. Puedes convertir estos datos durante o después de la importación si lo necesitas, y haremos algunas de esas conversiones más adelante en este curso.

Describe

El método DataFrame.describe es una excelente manera de obtener una idea general de todos los datos numéricos en tu DataFrame. Notarás que solo se devuelven las columnas que tienen datos numéricos, mientras que las que tienen valores booleanos o texto, como email_verified y first_name, se omiten.

Verás muchas agregaciones diferentes.

```
my_data.describe()
```

	referral_count	balance
count	475.000000	475.000000
mean	3.429474	49.933263
std	2.281085	28.280448
min	0.000000	0.050000
25%	2.000000	25.305000
50%	3.000000	51.570000
75%	5.000000	74.480000
max	7.000000	99.900000

Otros métodos

```
# El promedio
my_data.balance.mean()
```

```
np.float64(49.933263157894736)
```

```
# La desviación estándar
my_data.balance.std()
```

```
np.float64(28.28044849675191)
```

```
# La suma de todo el balance
my_data.balance.sum()
```

```
np.float64(23718.3)
```

```
# El valor mínimo del balance
my_data.balance.min()
```

```
np.float64(0.05)
```

```
# El valor máximo del balance
my_data.balance.max()
```

```
np.float64(99.9)
```

```
# Obtener, de una columna en específico, el conteo de los valores únicos
my_data.email_verified.value_counts()
```

```
email_verified
True      389
False     86
Name: count, dtype: int64
```

Reordenar columnas

Con pandas puedes ordenar rápidamente las columnas de un DF.

Ordenemos el DataFrame para que el usuario con el saldo más alto esté en la parte superior. Por defecto, se asume el orden ascendente, pero puedes cambiar eso configurando el argumento de palabra clave ascending en False.

```
my_data.sort_values(by='balance', ascending=False).head()
```

	first_name	last_name	email	email_verified	signup_
twhite	Timothy	White	white5136@hotmail.com	True	2018-C
karen.snow	Karen	Snow	ksnow@yahoo.com	True	2018-C
king	Billy	King	billy.king@hotmail.com	True	2018-C
king3246	Brittney	King	brittney@yahoo.com	True	2018-C
crane203	Valerie	Crane	valerie7051@hotmail.com	True	2018-C

```
my_data.head()
```

	first_name	last_name	email	email_verified
aaron	Aaron	Davis	aaron6348@gmail.com	True
acook	Anthony	Cook	cook@gmail.com	True
adam.saunders	Adam	Saunders	adam@gmail.com	False
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	True
adrian.blair	Adrian	Blair	adrian9335@gmail.com	True

Notarás que la llamada a `sort_values` en realidad creó un nuevo DataFrame. Si deseas cambiar permanentemente el orden predeterminado (ordenado por índice), puedes pasar `True` como argumento al parámetro de palabra clave `inplace`.

```
# Ordenar primero por last_name y luego por first_name. Por defecto, np.nan aparece al final
my_data.sort_values(by=['last_name', 'first_name'], inplace=True)
# El orden ahora ha cambiado
my_data.head()
```

	first_name	last_name	email	email_verified	size
darlene.adams	Darlene	Adams	adams@hotmail.com	True	10
lauren	Lauren	Aguilar	lauren.aguilar@summers.com	False	9
daniel	Daniel	Allen	allen@hotmail.com	False	8
kallen	Kathy	Allen	kathy@hotmail.com	False	7
alvarado	Denise	Alvarado	alvarado@hotmail.com	True	6

```
# Si quieres ordenar por el índice, puedes usar sort_index
```

```
my_data.sort_index(inplace=True)
my_data.head()
```

	first_name	last_name	email	email_verified
aaron	Aaron	Davis	aaron6348@gmail.com	True
acook	Anthony	Cook	cook@gmail.com	True
adam.saunders	Adam	Saunders	adam@gmail.com	False
adrian	Adrian	Fang	adrian.fang@teamtreehouse.com	True
adrian.blair	Adrian	Blair	adrian9335@gmail.com	True

Más fuentes de información con pandas

Exploraremos la forma en la que se cargan diferentes fuentes de datos con pandas, una vez cargados en un dataframe, se pueden aplicar los métodos y propiedades que hemos visto anteriormente.

Cargar JSON

```
filepath = 'data/anscombe.json'

# Leer el archivo JSON
data = pd.read_json(filepath)

# Verificar los datos
data.head()
```

	Series	X	Y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33

```
data.to_json('data/mi_archivojson.json')
```

Cargar Excel

Carguemos el archivo data/datos_rrss.xlsx en un DataFrame de pandas.

Primero revisemos el contenido del archivo Excel.

```
# Para cargar los datos de un archivo Excel, se usa la función read_excel
# Debemos instalar la librería openpyxl

%pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.1.4-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et-xmlfile (from openpyxl)
  Using cached et_xmlfile-1.1.0-py3-none-any.whl.metadata (1.8 kB)
  Downloading openpyxl-3.1.4-py2.py3-none-any.whl (251 kB)
                                              251.4/251.4 kB 1.4 MB/s eta 0:00:00a 0:00:
?25hUsing cached et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.1.4
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

my_excel = pd.read_excel('data/datos_rrss.xlsx')
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Facebook	2449	True	2006
1	Twitter	339	False	2006
2	Instagram	1000	True	2010
3	YouTube	2000	False	2005
4	LinkedIn	663	False	2003

Esto por defecto carga la primera hoja del archivo Excel. Si deseas cargar una hoja específica, puedes hacerlo pasando el nombre de la hoja al argumento sheet_name.

```
my_excel = pd.read_excel('data/datos_rrss.xlsx', sheet_name='Hoja3')
my_excel.head() # Note que este no tiene encabezado de la información
```

	Facebook	2449	True	2006
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
# Ajustar el encabezado
my_excel = pd.read_excel('data/datos_rrss.xlsx', sheet_name='Hoja3')
my_excel.columns = ['Nombre', 'Cantidad', 'ES_FBK', 'Año'] # De esta forma asignamos los nombres
my_excel.head()
```

	Nombre	Cantidad	ES_FBK	Año
0	Twitter	339	False	2006
1	Instagram	1000	True	2010
2	YouTube	2000	False	2005
3	LinkedIn	663	False	2003
4	WhatsApp	1600	True	2009

```
my_excel.to_excel('data/mi_archivo_excel.xlsx', index=False) # Guardar el archivo sin indice
```

Cargar Directamente desde la web

Pandas también puede cargar datos directamente desde una URL. Esto es útil si los datos que deseas cargar están disponibles en la web y no necesitas descargarlos localmente.

Leer CSV desde una URL

```
# Ejemplo para cargar datos desde la web
import pandas as pd

url = 'https://raw.githubusercontent.com/datasets/investor-flow-of-funds-us/master/datasets/investor-flow-of-funds-us/master/data.csv'
data = pd.read_csv(url)
data.head()
```

	Date	Total Equity	Domestic Equity	World Equity	Hybrid	Total Bond	Taxable Bond	Municipal Bond	Total
0	2012-12-05	-7426	-6060	-1367	-74	5317	4210	1107	-2183
1	2012-12-12	-8783	-7520	-1263	123	1818	1598	219	-6842
2	2012-12-19	-5496	-5470	-26	-73	103	3472	-3369	-5466
3	2012-12-26	-4451	-4076	-375	550	2610	3333	-722	-1291
4	2013-01-02	-11156	-9622	-1533	-158	2383	2103	280	-8931

Leer HTML desde una URL

Para este necesitamos instalar la librería lxml, que es una librería de Python que permite trabajar con archivos XML y HTML.

```
%pip install lxml
```

Requirement already satisfied: lxml in ./venv/lib/python3.11/site-packages (5.2.2)
Note: you may need to restart the kernel to use updated packages.

```
import pandas as pd

# URL de la página web que contiene la tabla
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_inflation_rate'

# Leer todas las tablas en la página web
tables = pd.read_html(url)
tables
```

```
[      0                                         1
0 NaN  This article needs to be updated. Please help ..., \
          Country/Territory/Region/Group \
          Country/Territory/Region/Group
0                                         Aruba
1                                         Afghanistan
2                                         Angola
3                                         Albania
4                                         Andorra
...
245                                         ...
246                                         European Union
246                                         G20 (Group of Twenty)
247                                         G-77 (Group of 77)
248 OECD (Organisation for Economic Cooperation an...
249 Notes: WB: Consumer price index reflects chang...
```

```
                                         World Bank consumer price indices (in %) \
                                         2010
0                                         2.08
1                                         2.18
2                                         14.47
3                                         3.63
4                                         NaN
...
245                                         ...
246                                         1.53
246                                         NaN
247                                         NaN
248                                         1.81
249 Notes: WB: Consumer price index reflects chang...
```

```
                                         \
                                         2011
0                                         4.32
1                                         11.80
2                                         13.48
3                                         3.43
4                                         NaN
...
245                                         ...
246                                         3.29
246                                         NaN
247                                         NaN
248                                         3.37
249 Notes: WB: Consumer price index reflects chang...
```

```
                                         \
                                         2012
0                                         0.63
1                                         6.44
2                                         10.28
3                                         2.03
4                                         NaN
...
245                                         ...
246                                         2.66
246                                         NaN
247                                         NaN
```

```
248                                2.53
249 Notes: WB: Consumer price index reflects chang...
```

```
0                               2013
1                               -2.37
2                               7.39
3                               8.78
4                               1.94
..                             ...
245                                ...
246                                NaN
247                                NaN
248                                1.45
249 Notes: WB: Consumer price index reflects chang...
```

```
0                               2014
1                               0.42
2                               4.67
3                               7.28
4                               1.63
..                             ...
245                                ...
246                                NaN
247                                NaN
248                                0.62
249 Notes: WB: Consumer price index reflects chang...
```

```
0                               2015
1                               0.47
2                               -0.66
3                               9.35
4                               1.90
..                             ...
245                                ...
246                                NaN
247                                NaN
248                                0.34
249 Notes: WB: Consumer price index reflects chang...
```

```
0                               2016
1                               -0.93
2                               4.38
3                               30.70
4                               1.28
..                             ...
245                                ...
246                                NaN
247                                NaN
```

```
248          0.44
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0          2017
```

```
1          -1.03
```

```
2          4.98
```

```
3          29.84
```

```
4          1.99
```

```
..          NaN
```

```
245          ...
```

```
246          1.43
```

```
247          NaN
```

```
248          NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0          2018
```

```
1          3.63
```

```
2          0.63
```

```
3          19.63
```

```
4          2.03
```

```
..          NaN
```

```
245          ...
```

```
246          1.74
```

```
247          NaN
```

```
248          NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0          2019
```

```
1          4.26
```

```
2          2.30
```

```
3          17.08
```

```
4          1.41
```

```
..          NaN
```

```
245          ...
```

```
246          1.63
```

```
247          NaN
```

```
248          NaN
```

```
249 Notes: WB: Consumer price index reflects chang...
```

```
\
```

```
0          2020
```

```
1          NaN
```

```
2          NaN
```

```
3          22.27
```

```
4          1.62
```

```
..          NaN
```

```
245          ...
```

```
246          0.48
```

```
247          NaN
```

248 0.73

249 Notes: WB: Consumer price index reflects chang...

\

2021

0 NaN

1 NaN

2 25.75

3 2.04

4 NaN

.. ...

245 2.55

246 NaN

247 NaN

248 2.82

249 Notes: WB: Consumer price index reflects chang...

\

2022

0 NaN

1 NaN

2 NaN

3 6.73

4 NaN

.. ...

245 8.83

246 NaN

247 NaN

248 8.24

249 Notes: WB: Consumer price index reflects chang...

WB Consumer price index (2010=100) \
Index

0 109.5

1 149.9

2 583.7

3 131.8

4 NaN

.. ...

245 NaN

246 NaN

247 NaN

248 NaN

249 Notes: WB: Consumer price index reflects chang...

Year

0 2019

1 2019

2 2021

3 2022

4 NaN

.. ...

245 NaN

246 NaN

247 NaN

248

NaN

249 Notes: WB: Consumer price index reflects chang...

[250 rows x 16 columns],

	Country/Territory/Region/Group	\
	Country/Territory/Region/Group	
	Country/Territory/Region/Group	

0

Aruba

1

Afghanistan

2

Angola

3

Albania

4

Andorra

..

...

245

European Union

246

G20 (Group of Twenty)

247

G-77 (Group of 77)

248 OECD (Organisation for Economic Cooperation an...

249 Notes: WB: Consumer price index reflects chang...

	United Nations consumer price indices	\
	2010	

Unnamed: 1_level_2

0

2.08

1

NaN

2

2.15

3

3.61

4

14.48

..

...

245

1.69

246

2.54

247

5.15

248

1.74

249 Notes: WB: Consumer price index reflects chang...

\

2011

Unnamed: 2_level_2

0

4.38

1

12.17

2

4.70

3

3.44

4

13.48

..

...

245

2.74

246

3.75

247

6.87

248

2.79

249 Notes: WB: Consumer price index reflects chang...

\

2012

Unnamed: 3_level_2

0

0.57

1

-28.64

2

1.43

3

2.04

```

4          10.28
..
245        ...
246        2.53
247        2.77
248        5.45
248        2.13
249 Notes: WB: Consumer price index reflects chang...

```

\

	2013
0	Unnamed: 4_level_2
1	-2.37
2	-9.90
3	0.19
4	1.93
4	8.78
..	...
245	1.37
246	2.40
247	5.15
248	1.53

```
249 Notes: WB: Consumer price index reflects chang...
```

\

	2014
0	Unnamed: 5_level_2
1	0.42
2	4.67
2	-0.28
3	1.61
4	7.30
..	...
245	0.44
246	2.38
247	4.56
248	1.59

```
249 Notes: WB: Consumer price index reflects chang...
```

\

	2015
0	Unnamed: 6_level_2
1	0.47
2	-1.55
2	-0.94
3	1.87
4	9.16
..	...
245	0.17
246	1.72
247	4.08
248	0.53

```
249 Notes: WB: Consumer price index reflects chang...
```

\

	2016
0	Unnamed: 7_level_2

```

0          -0.93
1           2.12
2          -0.57
3           1.29
4          30.69
..
245         ...
246         0.22
247         2.08
247         5.03
248         1.00
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2017
Unnamed: 8_level_2
0          -1.03
1         -10.57
2           1.34
3           1.98
4          29.84
..
245         ...
246         1.56
246         2.34
247         3.98
248         2.07
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2018
Unnamed: 9_level_2
0           3.63
1           0.63
2           0.38
3           2.03
4          19.63
..
245         ...
246         1.81
246         2.72
247         4.59
248         2.37
249 Notes: WB: Consumer price index reflects chang...

```

\

```

2019
Unnamed: 10_level_2
0           3.94
1           2.30
2           0.75
3           1.41
4          17.08
..
245         ...
246         1.36
246         2.68
247         5.23
248         1.80
249 Notes: WB: Consumer price index reflects chang...

```

```
\_2020
    Unnamed: 11_level_2
0           -1.31
1            5.60
2           -0.47
3            1.62
4           22.28
..
245          ...
246          0.53
247          2.06
247          5.63
248          1.10
249 Notes: WB: Consumer price index reflects chang...
```

```
\_2021
    Unnamed: 12_level_2
0           0.73
1            5.13
2            1.78
3            2.04
4           25.77
..
245          ...
246          2.77
246          3.67
247          6.28
248          3.65
249 Notes: WB: Consumer price index reflects chang...
```

```
\_2022
    Unnamed: 13_level_2
0           5.59
1           13.71
2            3.04
3            6.73
4           21.36
..
245          ...
246          8.89
246          7.97
247          7.72
248          8.94
249 Notes: WB: Consumer price index reflects chang...
```

```
[250 rows x 14 columns],
      vteLists of countries by financial rankings \_
0                  Trade
1                  Investment
2                  Funds
3                  Budget and debt
4                  Income and taxes
5                  Bank rates
6                  Currency
7                  Other
```

8 Lists of countries by GDP rankings List of int...

```
vteLists of countries by financial rankings.1
0 Account balance % of GDP Exports by product me...
1 FDI received past FDI abroad GFI
2 Forex reserves Gold reserves Sovereign wealth ...
3 Government budget PPP % of GDP per capita Cred...
4 Tax rates Inheritance tax Tax revenue Wage ave...
5 Central bank interest rate Commercial bank pri...
6 Exchange rates to US$ Inflation rate
7 Financial Development Index Average annual lab...
8 Lists of countries by GDP rankings List of int... ]
```

```
# Imprimir el número total de tablas que se traen de la página web
print(f'Número total de tablas: {len(tables)}')
```

Número total de tablas: 4

```
# Revisemos las tablas que tenemos
tables[0].head()
```

	0	1
0	NaN	This article needs to be updated. Please help ...

```
tables[1].head()
```

	Country/Territory/Region/Group	World Bank consumer price indices (in %)							
	Country/Territory/Region/Group	2010	2011	2012	2013	2014	2015	2016	
0	Aruba	2.08	4.32	0.63	-2.37	0.42	0.47	-0.93	
1	Afghanistan	2.18	11.80	6.44	7.39	4.67	-0.66	4.38	
2	Angola	14.47	13.48	10.28	8.78	7.28	9.35	30.70	
3	Albania	3.63	3.43	2.03	1.94	1.63	1.90	1.28	
4	Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
tables[2].head()
```

	Country/Territory/Region/Group	United Nations consumer price indices			
	Country/Territory/Region/Group	2010	2011	2012	2013
	Country/Territory/Region/Group	Unnamed: 1_level_2	Unnamed: 2_level_2	Unnamed: 3_level_2	Unnamed: 4_level_2
0	Aruba	2.08	4.38	0.57	-2.37
1	Afghanistan	NaN	12.17	-28.64	-9.90
2	Angola	2.15	4.70	1.43	0.19
3	Albania	3.61	3.44	2.04	1.93
4	Andorra	14.48	13.48	10.28	8.78

```
tables[3].head()
```

	vteLists of countries by financial rankings	vteLists of countries by financial rankings.1
0	Trade	Account balance % of GDP Exports by product me...
1	Investment	FDI received past FDI abroad GFI
2	Funds	Forex reserves Gold reserves Sovereign wealth ...
3	Budget and debt	Government budget PPP % of GDP per capita Cred...
4	Income and taxes	Tax rates Inheritance tax Tax revenue Wage ave...

Vamos a leer otras tablas a partir de HTML

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota')
print(f'Total de tablas obtenidas: {len(table_MN)})
```

Total de tablas obtenidas: 28

Con 38 tablas, puede ser un desafío encontrar la que necesitas. Para facilitar la selección de la tabla, utiliza el parámetro match para seleccionar un subconjunto de tablas. Podemos usar el título "Resultados de las elecciones en carreras estatales" para seleccionar la tabla:

```
%pip install html5lib
```

```
Requirement already satisfied: html5lib in ./venv/lib/python3.11/site-packages (1.1)
Requirement already satisfied: six>=1.9 in ./venv/lib/python3.11/site-packages (from
Requirement already satisfied: webencodings in ./venv/lib/python3.11/site-packages (f
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
table_MN = pd.read_html('https://en.wikipedia.org/wiki/Minnesota', match='presidential')
len(table_MN)
```

1

```
df = table_MN[0]
df.head()
```

	Year	Republican		Democratic		Third party	
	Year	No.	%	No.	%	No.	%
0	2020	1484065	45.28%	1717077	52.40%	76029	2.32%
1	2016	1323232	44.93%	1367825	46.44%	254176	8.63%
2	2012	1320225	44.96%	1546167	52.65%	70169	2.39%
3	2008	1275409	43.82%	1573354	54.06%	61606	2.12%
4	2004	1346695	47.61%	1445014	51.09%	36678	1.30%

```
df.to_html('data/mi_tabla_descargada.html')
df.to_json('data/mi_tabla_descargada.json')
df.to_csv('data/mi_tabla_descargada.csv')
df.to_excel('data/mi_tabla_descargada.xlsx')
```

Cargar datos de una base de datos SQL (Mid-level)

Pandas permite cargar datos directamente desde una base de datos SQL o NoSQL, como SQLite, MySQL, PostgreSQL, MongoDB, entre otros. Para ello, necesitas instalar los controladores de la base de datos correspondiente, como sqlalchemy, psycopg2, pymysql, pymongo, entre otros.

SQL

```
%pip install sqlalchemy
```

```
Collecting sqlalchemy
```

```
  Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl.metadata (9.6 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in ./venv/lib/python3.11/site
  Downloading SQLAlchemy-2.0.31-cp311-cp311-macosx_11_0_arm64.whl (2.1 MB)

```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 5.4 MB/s eta 0:00:00a 0:00:010m
```

```
?25hInstalling collected packages: sqlalchemy
```

```
Successfully installed sqlalchemy-2.0.31
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd

from sqlalchemy import create_engine # Este es el método para crear la conexión con la base de datos

# Crear la conexión con la base de datos
engine = create_engine('sqlite:///data/mibase.db') # Para otros motores debes cambiar la string de conexión de la base de datos

# Leer los datos de una tabla específica. En este caso, la tabla rock_songs que es una tabla de la base de datos
data = pd.read_sql('SELECT * FROM rock_songs', engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Caught Up in You	.38 Special	1982.0	82
1	Hold On Loosely	.38 Special	1981.0	85
2	Rockin' Into the Night	.38 Special	1980.0	18
3	Art For Arts Sake	10cc	1975.0	1
4	Kryptonite	3 Doors Down	2000.0	13

```
data = pd.read_sql_query("SELECT * FROM rock_songs WHERE Artist='AC/DC' ", engine)
data.head()
```

	Song	Artist	Release_Year	PlayCount
0	Back In Black	AC/DC	1980.0	97
1	Big Gun	AC/DC	1993.0	6
2	Dirty Deeds Done Dirt Cheap	AC/DC	1976.0	85
3	For Those About To Rock	AC/DC	1981.0	46
4	Hard As A Rock	AC/DC	1995.0	1

Estas son formas de traer datos desde una base de datos, depende del tamaño de los datos, o del problema, si es conveniente hacer la query y traer el resultado o traer toda la tabla y trabajar con ella.

```
# Guardar el DataFrame en una tabla de la base de datos
data.to_sql('rock_song_filtered', con=engine, if_exists='replace', index=False)
```

20

[Imprimir en PDF](#)

Manejo de ficheros

Contenido

- Manejo de ficheros
- Ejercicio 1: Registro y Análisis de Asistencia
- Reto 2: Gestión de Inventario de Equipos
- Reto 1: Sistema de Gestión de Evaluaciones de Desempeño
- Reto 2: Sistema de Gestión de Proyectos de Infraestructura

El manejo de ficheros es una habilidad esencial en Python, especialmente cuando se trabaja con datos almacenados en archivos de texto, CSV, JSON y otros formatos. En esta clase, aprenderemos a leer y escribir archivos, así como a procesar y analizar los datos contenidos en ellos.

En diferentes aplicaciones es fundamental realizar manejo de archivos, tanto para el almacenamiento de información como para la carga de información de la aplicación general. Python ofrece un conjunto de funciones que hacen parte de la librería estándar y permiten la manipulación de archivos de texto plano. A continuación veremos el uso de las principales funciones.

Para abrir un archivo:

```
filepath = 'ruta_del_archivo'  
file = open('filepath/filename.txt', 'modo')
```

Como se puede observar un archivo puede abrirse en diferentes modos:



Para usar esta librería, en primer lugar crearemos un archivo desde cero, esto se puede hacer con los modos (x) o (w). **Verifique cual es la diferencia entre ambos**



```
# Descarguemos un archivo de prueba  
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 20:52:10-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPythc  
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185  
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44  
HTTP request sent, awaiting response... 200 OK  
Length: 41 [text/plain]  
Saving to: 'archivo.txt'  
  
archivo.txt      0%[                ]      0  --.-KB/s  
archivo.txt    100%[=====]      41  --.-KB/s    in 0s  
  
2024-06-12 20:52:10 (525 KB/s) - 'archivo.txt' saved [41/41]
```

```
# Abrir y leer un archivo de texto  
with open("archivo.txt", "r") as archivo:  
    contenido = archivo.read()  
    print(contenido)
```

Hola, este es el contenido de un archivo.

```
# Crear un archivo
folder='sample_data'
fileName = 'miArchivo'
fileExtension= 'txt'
filePath = folder + '/' + fileName + '.' + fileExtension
open(filePath, 'x')
```

```
<_io.TextIOWrapper name='sample_data/miArchivo.txt' mode='x' encoding='UTF-8'>
```

Podrá verificar que después de ejecutar las líneas anteriores se crea un archivo en la carpeta sample_data. A continuación abriremos el archivo en modo (w) para escribir algo en él.

```
file = open(filePath, 'w')
```

Ahora con el archivo abierto, podemos escribir la información que se contenga en otra estructura, por ejemplo una lista.

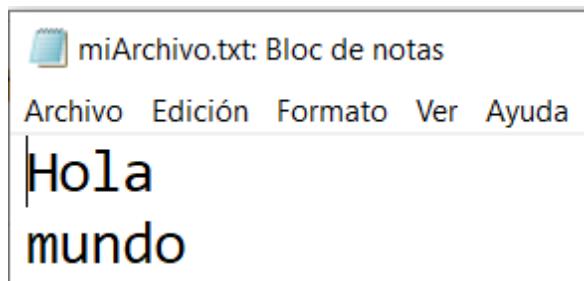
```
miLista = ["Hola","mundo"]
```

```
miLista = ['Hola','mundo']
for i in range(10):
    for palabra in miLista:
        file.write(palabra+'\t')
    file.write('\n')
```

Luego de procesar un archivo es necesario cerrarlo para que esté disponible para otras aplicaciones.

```
file.close()
```

En este momento haga el ejercicio de descargar el archivo y abrirlo desde su computador con un bloc de notas. Podrá ver algo como lo siguiente:



A partir de este archivo se pueden agregar nuevas líneas, para ello utilizaremos el modo (a), supongamos la siguiente tupla:

```
miTupla = ("Loren","ipsum","dolor","at","met")
```

```
miTupla = ('Loren','ipsum','dolor','at','met')
file = open(filePath,'a')
for palabra in miTupla:
    file.write(palabra+' ')
file.close()
```

```
with open(filePath, "w") as archivo:
    archivo.write("Hola, este es un archivo de texto.\n")
    archivo.write("Esta es una segunda línea.")
```

Ejemplo 1:

Escribe un programa que cree un archivo de texto, escriba varias líneas, y luego lo lea e imprima su contenido.

```
with open("mi_archivo.txt", "w") as archivo:
    archivo.write("Primera línea.\n")
    archivo.write("Segunda línea.\n")
    archivo.write("Tercera línea.\n")
```

```
# Leer el archivo de texto
with open("mi_archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

```
Primera línea.
Segunda línea.
Tercera línea.
```

Lectura de archivos

Ahora que ya tenemos un archivo creado y con contenido podemos abrirlo para leerlo.

```
file = open(filePath, 'r')
lista = list()
for linea in file:
    #print(type(linea))
    lista.append(linea)
    print(linea)

file.close()
print(lista)
```

Lecto - escritura de archivos

Existen modos de lectoescritura como w+, r+ y a+

```
## Lecto - escritura de archivos
f=open(filePath,"r+")
x=f.readlines()
print(x)
print(type(x))
f.write("\n Esto es otra línea")
f.close()
```

```
[ 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmundo\t\n', 'Hola\tmur  
<class 'list'>
```

Leyendo archivos JSON

Estos archivos se utilizan ampliamente en APIs de sitios web. En apariencia son similares a los diccionarios de Python, pues almacenan los datos a manera de clave y valor. Al igual que en el caso de CSV la mejor alternativa para leer estos archivos es mediante Pandas. En este caso utilizaremos el archivo **anscombe.json**

```
import json

with open('sample_data/anscombe.json') as json_file:
    data = json.load(json_file)
    print(data)
```

```
[{'Series': 'I', 'X': 10.0, 'Y': 8.04}, {'Series': 'I', 'X': 8.0, 'Y': 6.95}, {'Series': 'I', 'X': 13.0, 'Y': 7.58}, {'Series': 'I', 'X': 9.0, 'Y': 8.81}, {'Series': 'I', 'X': 11.0, 'Y': 8.33}, {'Series': 'I', 'X': 14.0, 'Y': 9.96}, {'Series': 'I', 'X': 6.0, 'Y': 7.04}, {'Series': 'I', 'X': 12.0, 'Y': 9.14}, {'Series': 'I', 'X': 7.0, 'Y': 8.04}, {'Series': 'I', 'X': 10.0, 'Y': 9.14}, {'Series': 'II', 'X': 8.08, 'Y': 9.14}, {'Series': 'II', 'X': 9.14, 'Y': 8.04}, {'Series': 'II', 'X': 7.82, 'Y': 6.95}, {"Series": "III", "X": 7.91, "Y": 7.58}, {"Series": "III", "X": 8.77, "Y": 8.81}, {"Series": "III", "X": 8.33, "Y": 8.33}, {"Series": "III", "X": 9.23, "Y": 9.96}, {"Series": "III", "X": 7.5, "Y": 7.04}, {"Series": "III", "X": 10.8, "Y": 9.14}, {"Series": "III", "X": 8.81, "Y": 8.04}, {"Series": "III", "X": 9.96, "Y": 9.14}, {"Series": "III", "X": 8.33, "Y": 7.58}, {"Series": "IV", "X": 8.81, "Y": 8.04}, {"Series": "IV", "X": 8.04, "Y": 8.81}, {"Series": "IV", "X": 8.33, "Y": 9.96}, {"Series": "IV", "X": 8.77, "Y": 7.58}, {"Series": "IV", "X": 9.23, "Y": 8.33}, {"Series": "IV", "X": 7.5, "Y": 9.14}, {"Series": "IV", "X": 10.8, "Y": 7.04}, {"Series": "IV", "X": 9.96, "Y": 8.81}, {"Series": "IV", "X": 8.81, "Y": 9.96}, {"Series": "IV", "X": 9.96, "Y": 7.58}], [{"x": 10.0, "y": 8.04}, {"x": 8.0, "y": 6.95}, {"x": 13.0, "y": 7.58}, {"x": 9.0, "y": 8.81}, {"x": 11.0, "y": 8.33}, {"x": 14.0, "y": 9.96}, {"x": 6.0, "y": 7.04}, {"x": 12.0, "y": 9.14}, {"x": 7.0, "y": 8.04}, {"x": 10.0, "y": 9.14}, {"x": 8.08, "y": 9.14}, {"x": 9.14, "y": 8.04}, {"x": 7.82, "y": 6.95}, {"x": 7.91, "y": 7.58}, {"x": 8.77, "y": 8.81}, {"x": 8.33, "y": 8.33}, {"x": 9.23, "y": 9.96}, {"x": 7.5, "y": 7.04}, {"x": 10.8, "y": 9.14}, {"x": 8.81, "y": 8.04}, {"x": 9.96, "y": 9.14}, {"x": 8.33, "y": 7.58}, {"x": 8.81, "y": 8.81}, {"x": 8.04, "y": 9.96}, {"x": 8.33, "y": 7.58}, {"x": 8.77, "y": 8.33}, {"x": 9.23, "y": 9.14}, {"x": 7.5, "y": 9.14}, {"x": 10.8, "y": 7.04}, {"x": 9.96, "y": 8.81}, {"x": 8.81, "y": 9.96}, {"x": 9.96, "y": 7.58}]]
```

Escribiendo archivos JSON

```
import json

# Escribir en un archivo JSON
datos = {
    "nombre": "Ana",
    "edad": 30,
    "ciudad": "Madrid"
}

with open("archivo.json", "w") as archivo:
    json.dump(datos, archivo, indent=4)
```

Escribiendo archivos csv

```
import csv

# Escribir en un archivo CSV
with open("archivo.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
```

Leyendo archivos csv

```
import csv

# Leer un archivo CSV
with open("archivo.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)
```

```
['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
```

Ejemplo:

Escribe un programa que cree un archivo CSV, escriba varias filas de datos, y luego lo lea e imprima su contenido.

```

import csv

# Escribir en un archivo CSV
with open("datos.csv", "w", newline='') as archivo:
    escritor_csv = csv.writer(archivo)
    escritor_csv.writerow(["Nombre", "Edad", "Ciudad"])
    escritor_csv.writerow(["Ana", 30, "Madrid"])
    escritor_csv.writerow(["Carlos", 25, "Barcelona"])
    escritor_csv.writerow(["Beatriz", 28, "Valencia"])

# Leer el archivo CSV
with open("datos.csv", "r") as archivo:
    lector_csv = csv.reader(archivo)
    for fila in lector_csv:
        print(fila)

```

```

['Nombre', 'Edad', 'Ciudad']
['Ana', '30', 'Madrid']
['Carlos', '25', 'Barcelona']
['Beatriz', '28', 'Valencia']

```

Ejemplo de análisis de datos desde un archivo CSV

Supongamos que tenemos un archivo CSV con datos de empleados y queremos calcular la edad promedio.

```

# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main

```

```
--2024-06-12 21:41:37-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 557 [text/plain]
```

```

Saving to: 'empleados.csv'

empleados.csv      0%[                                         ]      0  --.-KB/s
empleados.csv     100%[=====>]      557  --.-KB/s   in 0s

2024-06-12 21:41:37 (20.5 MB/s) - 'empleados.csv' saved [557/557]
```

```

import csv

# Leer datos de empleados y calcular la edad promedio
with open("empleados.csv", "r") as archivo:
    lector_csv = csv.DictReader(archivo)
    total_edad = 0
    contador = 0
    for fila in lector_csv:
        total_edad += int(fila["Edad"])
        contador += 1

    edad_promedio = total_edad / contador
    print(f"Edad promedio: {edad_promedio}")

```

Edad promedio: 30.133333333333333

Ejemplo de análisis de datos desde un archivo JSON

Supongamos que tenemos un archivo JSON con datos de varios productos y queremos calcular el precio total de todos los productos.

```

# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main

```

```

--2024-06-12 21:42:00-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPytho
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
HTTP request sent, awaiting response... 200 OK
Length: 1369 (1.3K) [text/plain]
Saving to: 'productos.json'

```

```

productos.json      0%[                                         ]      0  --.-KB/s
productos.json     100%[=====>]    1.34K  --.-KB/s    in 0s

2024-06-12 21:42:00 (64.5 MB/s) - 'productos.json' saved [1369/1369]

```

```
import json

# Leer datos de productos y calcular el precio total
with open("productos.json", "r") as archivo:
    productos = json.load(archivo)
    total_precio = sum(producto["precio"] for producto in productos)
    print(f"Precio total: {total_precio}")
```

Precio total: 370.0

Crea un programa para gestionar el registro de asistencia de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de asistencia en un archivo CSV.
2. Leer y mostrar los datos de asistencia desde el archivo.
3. Calcular y mostrar el porcentaje de asistencia de cada empleado.

```
# Descarguemos un archivo de prueba
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:08-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main/asistencia.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.108.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 704 [text/plain]
Saving to: ‘asistencia.csv’
```



```
asistencia.csv          0%[                                         ]      0  --.-KB/s
asistencia.csv          100%[=====]      704  --.-KB/s    in 0s

2024-06-12 21:47:08 (44.0 MB/s) - ‘asistencia.csv’ saved [704/704]
```

```
import csv

def agregar_asistencia(archivo, nombre, fecha, presente):
    with open(archivo, "a", newline='') as archivo_csv:
        escritor_csv = csv.writer(archivo_csv)
        escritor_csv.writerow([nombre, fecha, presente])
```

```
def leer_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        for fila in lector_csv:
            print(fila)
```

```
def calcular_asistencia(archivo):
    with open(archivo, "r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        empleados = {}
        for fila in lector_csv:
            nombre, _, presente = fila
            if nombre not in empleados:
                empleados[nombre] = {"asistencias": 0, "total": 0}
            empleados[nombre]["total"] += 1
            if presente == "True":
                empleados[nombre]["asistencias"] += 1

        for nombre, datos in empleados.items():
            porcentaje_asistencia = (datos["asistencias"] / datos["total"]) * 100
            print(f"{nombre}: {porcentaje_asistencia:.2f}% de asistencia")
```

```
# Agregar registros de asistencia
agregar_asistencia("asistencia.csv", "Juan", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-01", True)
agregar_asistencia("asistencia.csv", "Carlos", "2023-06-01", False)
agregar_asistencia("asistencia.csv", "Juan", "2023-06-02", False)
agregar_asistencia("asistencia.csv", "Ana", "2023-06-02", True)
```

```
# Leer y mostrar los registros de asistencia
leer_asistencia("asistencia.csv")
```

```
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
['Juan', '2023-06-01', 'True']
['Ana', '2023-06-01', 'True']
['Carlos', '2023-06-01', 'False']
['Juan', '2023-06-02', 'False']
['Ana', '2023-06-02', 'True']
```

```
# Calcular y mostrar el porcentaje de asistencia de cada empleado
calcular_asistencia("asistencia.csv")
```

```
Juan: 50.00% de asistencia  
Ana: 100.00% de asistencia  
Carlos: 0.00% de asistencia
```

Crea un programa para gestionar el inventario de equipos en una institución pública. El programa debe permitir:

1. Guardar los datos del inventario en un archivo JSON.
2. Leer y mostrar los datos del inventario desde el archivo.
3. Calcular y mostrar el valor total del inventario.

```
# Descarguemos un archivo de prueba  
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
--2024-06-12 21:47:42-- https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main/inventario.json  
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.108.134  
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 2982 (2.9K) [text/plain]  
Saving to: ‘inventario.json’  
  
inventario.json      0%[=====]      0  --.-KB/s  
inventario.json     100%[=====]    2.91K  --.-KB/s    in 0s  
  
2024-06-12 21:47:42 (16.1 MB/s) - ‘inventario.json’ saved [2982/2982]
```

```
import json  
  
def agregar_equipo(archivo, equipo):  
    try:  
        with open(archivo, "r") as archivo_json:  
            inventario = json.load(archivo_json)  
    except FileNotFoundError:  
        inventario = []  
  
    inventario.append(equipo)  
  
    with open(archivo, "w") as archivo_json:  
        json.dump(inventario, archivo_json, indent=4)
```

```
def leer_inventario(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        for equipo in inventario:
            print(equipo)
```

```
def calcular_valor_total(archivo):
    with open(archivo, "r") as archivo_json:
        inventario = json.load(archivo_json)
        valor_total = sum(equipo["valor"] for equipo in inventario)
        print(f"Valor total del inventario: {valor_total}")
```

```
# Agregar equipos al inventario
agregar_equipo("inventario.json", {"nombre": "Computadora", "marca": "Dell", "modelo": "Inspiron", "año": 2021, "valor": 200}
agregar_equipo("inventario.json", {"nombre": "Impresora", "marca": "HP", "modelo": "LaserJet", "año": 2019, "valor": 200}
agregar_equipo("inventario.json", {"nombre": "Escáner", "marca": "Epson", "modelo": "Perfection", "año": 2020, "valor": 150}
```

```
# Leer y mostrar los datos del inventario
leer_inventario("inventario.json")
```

```
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor': 200}
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 150}
{'nombre': 'Computadora', 'marca': 'Dell', 'modelo': 'Inspiron', 'año': 2021, 'valor': 200}
{'nombre': 'Impresora', 'marca': 'HP', 'modelo': 'LaserJet', 'año': 2019, 'valor': 200}
{'nombre': 'Escáner', 'marca': 'Epson', 'modelo': 'Perfection', 'año': 2020, 'valor': 150}
```

```
calcular_valor_total("inventario.json")
```

Valor total del inventario: 2300

Crea un programa que gestione las evaluaciones de desempeño de empleados en una institución pública. El programa debe permitir:

1. Guardar los datos de las evaluaciones en un archivo JSON.
2. Leer y mostrar los datos de las evaluaciones desde el archivo.
3. Calcular y mostrar el puntaje promedio de desempeño por departamento.

Requisitos:

- Cada evaluación debe contener el nombre del empleado, departamento, fecha de evaluación y puntaje.
- El programa debe ser capaz de agregar nuevas evaluaciones, actualizar puntajes de evaluaciones existentes y eliminar evaluaciones.
- Debe calcular el puntaje promedio de desempeño por departamento y mostrar un resumen de todos los empleados y sus evaluaciones.

```
# Descarguemos un archivo de prueba  
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Crea un programa que gestione los proyectos de infraestructura en una institución pública. El programa debe permitir:

1. Guardar los datos de los proyectos en un archivo CSV.
2. Leer y mostrar los datos de los proyectos desde el archivo.
3. Calcular y mostrar el presupuesto total de los proyectos por estado (planificado, en curso, completado).

Requisitos:

- Cada proyecto debe contener el nombre del proyecto, descripción, fecha de inicio, fecha de fin, estado y presupuesto.
- El programa debe ser capaz de agregar nuevos proyectos, actualizar el estado y presupuesto de proyectos existentes y eliminar proyectos.
- Debe calcular el presupuesto total por estado y mostrar un resumen de todos los proyectos.

```
# Descarguemos un archivo de prueba  
!wget https://raw.githubusercontent.com/BioAITeamLearning/IntroPython_2024_01_UAI/main
```

```
# Tu código va acá
```

Matrices y vectores

Contenido

- Matrices y vectores
- Numpy
- Generación de matrices y vectores
- Operaciones vectorizadas
- Más operaciones vectorizadas
- Expresiones compactas / Comprehensions
- Matplotlib
- Imágenes

Buscar un elemento en un arreglo

Dado un arreglo con componentes $A[1], \dots, A[n]$ y un valor x , el siguiente algoritmo retorna en la variable q el valor *false* si no existe k tal que $A[k] = x$; en caso contrario, retorna en la variable q el valor *true* y en la variable i el valor k .

$\{n > 0\}$
 $i : 0..n; q : boolean;$
 $A : array [1..n] of T;$
 $\{\text{asignarle valores a las componentes de } A\}$

Paso 1: $i := 0;$
Paso 2: **repetir**
 $i := i + 1;$
 $q := A[i] = x$
 hasta que $q \vee (i = n)$

```
import numpy as np
```

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
q=True
n=len(A)
while q and (i!=n) :
    q=not(A[i]==x)
    i=i+1
if i==n and q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',i-1,'El valor es:',A[i-1])
```

la posición es: 4 El valor es: e

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
n=len(A)
while i!=n:
    if A[i]==x:
        break
    i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 4 El valor es: e

Otra forma de buscar un elemento en un arreglo: (centinela)

Permite simplificar la condición de terminación del anterior algoritmo.

- Se le adiciona una componente al arreglo A .
- A la nueva componente se le asigna el valor x , que actua como **centinela** para la finalización de la búsqueda.

$$\{c = n + 1, n > 0\}$$

$$i : 0..c;$$

$$A : \text{array}[1..c] \text{ of } T;$$

Paso 1: $i := 0;$
 $A[c] := x;$

Paso 2: **repetir**
 $i := i + 1;$
hasta que $A[i] = x;$

```
A=np.array(['a','b','c','d','e'])
x='c'
n=len(A)
A=np.append(A,x)

i=0
while not(A[i]==x):
    i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 2 El valor es: c

Buscar un elemento en un arreglo ordenado

En este caso las componentes del arreglo están ordenadas, por ejemplo en orden creciente, es decir $A[i] < A[j]$ para todo $i < j$.

El siguiente algoritmo tiene en cuenta que el arreglo está ordenado.

```

i, j, k : integer; q : boolean;
{n > 0}
A : array [1..n] of T;
Paso 1: i := 1; j := n; q := false
Paso 2: repetir
          k := (i + j) div 2;
          si A[k] = x entonces q := true sino
              si A[k] < x entonces i = k + 1 sino j := k - 1;
hasta que q  $\vee (i > j)$ 
```

Busqueda binaria

```

import numpy as np
A=np.array([2,3,5,6,8,10,11])
x=11
i=0
j=len(A)
q=True
while q and (i<=j):
    k=(i+j)//2
    if A[k]==x:
        q=False
    else:
        if A[k]<x:
            i=k+1
        else:
            j=k-1
if q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',k,'El valor es:',A[k])
```

la posición es: 6 El valor es: 11

Producto escalar o producto interno

Dados las sucesiones de números (x_1, \dots, x_n) y (y_1, \dots, y_n) calcular el producto escalar

$$s = x_1 * y_1 + x_2 * y_2 + \dots + x_n * y_n = \sum_{i=1}^n x_i * y_i$$

Teniendo en cuenta la definición de esta suma en términos de la relación de recurrencia:

$$s_i = s_{i-1} + x_i * y_i, \quad s_0 = 0$$

se tiene el siguiente programa,

```
s : real; i : integer;
x, y : array [1..n] of real;
Paso 1: s := 0; i := 0;
Paso 2: repetir
          i := i + 1;
          s := s + x[i] * y[i];
      hasta que i = n;
```



```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
while i!=n:
    s=s+v1[i]*v2[i]
    i+=1
print('el producto escalar es: ',s)
```

el producto escalar es: 60

Sentencia for

Otro algoritmo para hallar el producto escalar de dos sucesiones de números (x_1, \dots, x_n) y (y_1, \dots, y_n) es:

```
s : real; i : 1..n;  
x, y : array [1..n] of real;  
s := 0;  
para i := 1 hasta n hacer  
  s := s + x[i] * y[i];
```

```
v1=[1,2,3,4]  
v2=[4,5,6,7]  
n=len(v1)  
s=0  
i=0  
for i in range(n):  
    s=s+v1[i]*v2[i]  
print('el producto escalar es: ',s)
```

```
el producto escalar es: 60
```

Encontrar el elemento máximo

Encontrar en un arreglo A el índice j tal que $x_j = \max(x_m, \dots, x_n)$.

$j, k : m..n;$
 $x : \text{array } [m..n] \text{ of } T;$
Paso 1: $j := m;$
Paso 2: **para** $k := m + 1$ **hasta** n **hacer**
 si $x[k] > x[j]$ **entonces** $j := k$

```
A=[1,2,4,5,3,6,7]
j=0
k=0
for k in range(1,len(A)):
    if A[k]>A[j]:
        j=k
print('el elemento maximo es:',A[j], 'en la posición: ',j)
```

```
el elemento maximo es: 7 en la posición:  6
```

Sentencias for anidadas

Ejemplo:

Dado un entero positivo n , calcular la suma $1^1 + 2^2 + \cdots + n^n$.

```

x, i : 1..n;
s, potencia : integer;
s := 0;
para x := 1 hasta n hacer
    potencia := 1;
    para i := 1 hasta x hacer
        potencia = potencia * x;
    s := s + potencia;

```

```

n=3
s=0
for x in range(1,n+1):
    potencia=1
    for i in range(1,x+1):
        potencia=potencia*x
    s=s+potencia
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

```

n=3
s=0
for x in range(1,n+1):
    s=s+x**x
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

Algoritmo para el ordenamiento de un arreglo

```

 $h, j, k : 1..n;$ 
 $x : \text{array } [1..n] \text{ of } T;$ 
 $u : T;$ 
para  $h := 1$  hasta  $n - 1$  hacer
     $j := h;$ 
    para  $k := h + 1$  hasta  $n$  hacer
        si  $x[k] > x[j]$  entonces  $j := k$ 
         $u := x[h];$ 
         $x[h] := x[j]$ 
         $x[j] := u$ 

```

Complejidad del algoritmo:

$$(n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n}{2}(n - 1).$$

```

x=np.random.randint(200, size=100)
h=0
for h in range(len(x)):
    j=h
    for k in range(h+1,len(x)):
        if x[k]>x[j]:
            j=k
    u=x[h]
    x[h]=x[j]
    x[j]=u
print(x)

```

198	194	192	187	181	180	179	179	178	177	175	175	172	172	171	168	166	164	163
161	161	160	158	156	154	152	147	146	146	146	146	142	140	135	135	130	128	126
124	120	119	118	118	116	112	110	107	102	101	101	100	100	95	89	88	86	
84	83	82	82	82	81	81	80	77	76	73	73	72	71	69	68	67	66	
61	61	57	55	49	45	42	41	41	40	39	36	35	35	29	23	22	21	
18	17	15	13	12	12	10	8	6	0									

Con la librería `numpy` se trabaja con matrices de forma natural. Fíjate cómo se declaran y cómo descubrimos sus dimensiones.

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

```
import numpy as np

a = np.array([[1,2,3,4,5],
              [5,4,3,2,1],
              [9,8,7,6,5],
              [7,6,5,6,7],
              [2,2,2,3,3],
              [4,3,4,3,4],
              [5,1,1,4,1]]).astype('int32')

print(a, type(a), a nbytes, "\n")
print("a shape", a.shape, "\n")
print("a rows", a.shape[0], "\n")
print("a cols", a.shape[1])
```

```
[[1 2 3 4 5]
 [5 4 3 2 1]
 [9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]
 [4 3 4 3 4]
 [5 1 1 4 1]] <class 'numpy.ndarray'> 140

a shape (7, 5)

a rows 7

a cols 5
```

```
a[0,0]=100
a
```

```
array([[100,    2,    3,    4,    5],
       [  5,    4,    3,    2,    1],
       [  9,    8,    7,    6,    5],
       [  7,    6,    5,    6,    7],
       [  2,    2,    2,    3,    3],
       [  4,    3,    4,    3,    4],
       [  5,    1,    1,    4,    1]], dtype=int32)
```

```
a nbytes
```

140

```
v = np.array([2,3,4,5,6,7,3,12])
print("v shape", v.shape)
print("v elems", v.shape[0])
print(v, type(v))
```

```
v shape (8,)
v elems 8
[ 2  3  4  5  6  7  3 12] <class 'numpy.ndarray'>
```

```
v[3:-1]
```

```
array([5, 6, 7, 3])
```

```
np.min(v), np.max(v)
```

```
(2, 12)
```

Con la notación de índices accedemos a columnas o filas enteras, rangos de columnas o filas, elementos individuales o porciones de una matriz o un vector.

```
print("una fila      ", a[2])
print("una fila      ", a[2,:])
print("una columna    ", a[:,2])
print("un elemento    ", a[2,2])
print("varias filas   \n", a[2:5])
print("varias columnas\n", a[:,1:3])
print("una porcion    \n", a[2:5,1:3])
```

```

una fila      [9 8 7 6 5]
una fila      [9 8 7 6 5]
una columna    [3 3 7 5 2 4 1]
un elemento    7
varias filas
[[9 8 7 6 5]
[7 6 5 6 7]
[2 2 2 3 3]]
varias columnas
[[2 3]
[4 3]
[8 7]
[6 5]
[2 2]
[3 4]
[1 1]]
una porcion
[[8 7]
[6 5]
[2 2]]

```

Muchas funciones de la librería `numpy` operan sobre una matriz completa, o de forma separada por columnas o filas según el valor del argumento `axis`.

```

print(a)
print("suma total", np.sum(a))
print("suma eje 0", np.sum(a, axis=0))
print("suma eje 1", np.sum(a, axis=1))
print("promedio total", np.mean(a))
print("promedio eje 0", np.mean(a, axis=0))
print("promedio eje 1", np.mean(a, axis=1))

```

```

[[100  2   3   4   5]
 [ 5   4   3   2   1]
 [ 9   8   7   6   5]
 [ 7   6   5   6   7]
 [ 2   2   2   3   3]
 [ 4   3   4   3   4]
 [ 5   1   1   4   1]]
suma total 237
suma eje 0 [132  26  25  28  26]
suma eje 1 [114  15  35  31  12  18  12]
promedio total 6.771428571428571
promedio eje 0 [18.85714286  3.71428571  3.57142857  4.           3.71428571]
promedio eje 1 [22.8  3.    7.    6.2  2.4  3.6  2.4]

```

Las matrices en Python pueden tener un número arbitrario de dimensiones y podemos acceder a submatrices en la dirección o dimensión que queramos.

```
z = np.random.randint(-20,20, size=(5,5))
print(z)
```

```
[[ -19    4   -13  -19   13]
 [ 19     6   -9   -9  -16]
 [-11   12   -6  -20   16]
 [ -4   -1   -4   13    0]
 [ -19   13   -1    9    8]]
```

```
np.sum(z, axis=1)
```

```
array([-34,   -9,   -9,    4,   10])
```

```
#m = np.random.randint(10, size=(3,3,3))
print("Matrix 3D completa\n", m)
print("-----\n", m[0,:,:])
print("-----\n", m[1,:,:])
print("-----\n", m[:,0,:])
print("-----\n", m[:,0,1])
print("-----\n", np.mean(m, axis=1))
```

```
Matrix 3D completa
```

```
[[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```
[[8 6 9]
 [0 3 0]
 [0 9 2]]]
```

```
-----
```

```
[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```
-----
```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```
-----
```

```
[[7 0 8]
 [4 7 9]
 [8 6 9]]]
```

```
-----
```

```
[0 7 6]
```

```
-----
```

```
[[4.66666667 1.33333333 7.
 [2.          4.66666667 8.
 [2.66666667 6.          3.66666667]]]
```

`m.shape`

(3, 3, 3)

`m`

```
print("matrix identidad\n", np.eye(3))
print("vector de ceros", np.zeros(4))
print("matriz de ceros\n", np.zeros((3,2)))
print("matriz de unos\n", np.ones((2,3)))
print("vector rango", np.arange(10))
print("vector rango", np.arange(5,10))
print("vector espacio lineal", np.linspace(2,12,11))
print("matriz aleatoria según distribución uniforme [0,1]\n", np.random.random(size=(3
print("vector aleatorio de enteros entre 0 y 5", np.random.randint(5, size=10))
```

```

matrix identidad
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
vector de ceros [0. 0. 0. 0.]
matriz de ceros
[[0. 0.]
 [0. 0.]
 [0. 0.]]
matriz de unos
[[1. 1. 1.]
 [1. 1. 1.]]
vector rango [0 1 2 3 4 5 6 7 8 9]
vector rango [5 6 7 8 9]
vector espacio lineal [ 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.]
matriz aleatoria según distribución uniforme [0,1]
[[0.57996022 0.65361661 0.99420518 0.12111156 0.26731741]
 [0.206147 0.25575192 0.66588831 0.55022348 0.76820253]
 [0.51384056 0.63172996 0.50836505 0.41256526 0.66835632]]
vector aleatorio de enteros entre 0 y 5 [0 3 3 0 3 3 2 1 2 1]

```

```

import numpy as np
v = np.array([10,12,13,15,20])
print(v)
print(v+1)
print(v*2)

a = np.random.randint(100, size=5)
print(a)

print(v.dot(a))

```

```

[10 12 13 15 20]
[11 13 14 16 21]
[20 24 26 30 40]
[60 83 36 31 82]
4169

```

```

v1=np.array([2,-1,1])
v2=np.array([-3,1,1])
print(v1.dot(v2))
print(v1*v2)
print(np.cross(v2,v1))
print(np.cross(v1,v2).dot(v1))
print(np.cross(v1,v2).dot(v2))

```

```
-6  
[-6 -1  1]  
[2 5 1]  
0  
0
```

```
m1=np.array([[1,2],[3,4]])
```

```
m2=np.array([[4,8],[9,1]])
```

```
m1*m2
```

```
array([[ 4, 16],  
       [27,  4]])
```

```
m1.dot(m2)
```

```
array([[22, 10],  
       [48, 28]])
```

```
a = np.array([[1,2,3],[4,5,6]])  
b = np.array([[6,5,4],[3,2,1]])  
c = np.array([[1,2],[4,5]])  
print(a)  
print("--")  
print(a.T)  
print("--")  
print(b)  
print("--")  
print(a.T.dot(a))  
print("--")  
print(a*b)  
print("--")  
x=np.array([[2,3],[4,5]])  
z=np.array([[5,6,7],[2,3,1]])  
print(x.dot(z))
```

```
[[1 2 3]
 [4 5 6]]
 --
 [[1 4]
 [2 5]
 [3 6]]
 --
 [[6 5 4]
 [3 2 1]]
 --
 [[17 22 27]
 [22 29 36]
 [27 36 45]]
 --
 [[ 6 10 12]
 [12 10  6]]
 --
 [[16 21 17]
 [30 39 33]]
```

Las operaciones vectorizadas también funcionan con expresiones *booleanas*. Fíjate cómo se indexa un vector con una expresión booleana para seleccionar un conjunto de elementos.

```
a = np.array([1,8,4,10,-4,5])
print("posiciones en a >4:", a>4)
print("elementos de a >4:",a[a>4])
```

```
posiciones en a >4: [False  True False  True False  True]
elementos de a >4: [ 8 10  5]
```

```
a = np.random.randint(100, size=(6,10))
a
```

```
array([[14, 19,  5, 87, 99, 92, 31, 63, 18, 44],
 [81, 26, 65, 86,  2, 24, 99, 79, 71, 37],
 [42, 87,  8, 92, 63, 29, 31, 85, 75, 93],
 [59, 58, 97, 10, 12, 99, 86, 99, 28, 56],
 [76, 76, 75, 56, 49, 33, 51, 58, 41,  4],
 [35, 55, 41,  7, 42, 85, 53, 14, 22, 51]])
```

```
np.mean(a, axis=1)
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
%timeit np.mean(a, axis=1)
```

```
10.8 µs ± 2.4 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
43.7 µs ± 925 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Pregúntate siempre si puedes vectorizar algo y hazlo. Por ejemplo, Para dos matrices (**grandes**).

```
a = np.random.randint(100, size=(1000,100))
b = np.random.randint(200, size=(1000,100))
```

```
# El número de elementos que son iguales
np.mean(a==b)
```

```
0.00512
```

```
# El promedio de los elementos de a que son mayores a su correspondiente posición en b
np.mean(a[a>b])
```

```
66.36604323536555
```

```
# El promedio de los elementos de b que son mayores a su correspondiente posición en a
np.mean(b[b>a])
```

```
122.37378653758725
```

```
# Con matrices más pequeñas
a = np.random.randint(100, size=(10))
b = np.random.randint(200, size=(10))
print (a)
print (b)
```

```
[73 19 71 47 95 38 48 59 74 61]
[ 68  94  69 142 167  40 190  70  33  95]
```

```
# el elemento en b correspondiente a la posición del elemento más grande en a
b[np.argmax(a)]
```

```
167
```

Broadcasting

Normalmente, Numpy necesita que las dimensiones de las matrices coincidan cuando se hacen operaciones con ellas

```
a = np.random.randint(100, size=(3,5))
b = np.random.randint(10, size=(3,4))
print (a)
print (b)
a + b
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86  0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```

-----
ValueError                                Traceback (most recent call last)
<ipython-input-14-860cc8167430> in <cell line: 5>()
      3 print (a)
      4 print (b)
----> 5 a + b

```

`ValueError: operands could not be broadcast together with shapes (3,5) (3,4)`

Pero numoy intentan expandir las operaciones si alguna de las dimensiones coincide

a

```
array([[83, 58, 25, 73, 33],
       [91, 73, 70, 60, 18],
       [20, 86, 0, 23, 62]])
```

a*10

```
array([[830, 580, 250, 730, 330],
       [910, 730, 700, 600, 180],
       [200, 860, 0, 230, 620]])
```

Veamos como funciona el reshape en la siguiente operación.
`a + b[:,1].reshape(-1,1)`

```
array([[85, 60, 27, 75, 35],
       [97, 79, 76, 66, 24],
       [29, 95, 9, 32, 71]])
```

`b[:,1].reshape(-1,1)`

```
array([[2],
       [6],
       [9]])
```

`b[:,1]`

```
array([2, 6, 9])
```

```
a + b[:,1]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-20-d148e74cc6fe> in <cell line: 1>()
----> 1 a + b[:,1]
```

ValueError: operands could not be broadcast together with shapes (3,5) (3,)

```
# Observa la forma de las filas
a + b.flatten()[:a.shape[1]]
```

```
array([[90, 60, 29, 73, 36],
       [98, 75, 74, 60, 21],
       [27, 88, 4, 23, 65]])
```

```
print (a)
print (b)
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86 0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```
b.flatten()
```

```
array([7, 2, 4, 0, 3, 6, 7, 1, 7, 9, 5, 2])
```

```
b.flatten()[:a.shape[1]]
```

```
array([7, 2, 4, 0, 3])
```

Fíjate cómo las siguientes expresiones son equivalentes:

```
a=15
if a > 10:
    s = "mayor que 10"
else:
    s = "menor que 10"

print(s)
```

mayor que 10

```
a = 15
s = "mayor que 10" if a > 10 else "menor que 10"
print(s)
```

mayor que 10

```
l=[]
for i in range(5):
    l.append(i)
print(l)
```

[0, 1, 2, 3, 4]

```
l=[i for i in range(5)]
print(l)
```

[0, 1, 2, 3, 4]

```
a = [10, -4, 20, 5]

#o = ["10A", "-4B", "20A", "5A"]

o = []
for i in a:
    if i<0:
        o.append(str(i)+"B")
    else:
        o.append(str(i)+"A")

print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
a = [10, -4, 20, 5]
def convert(x):
    return str(x)+"B" if x<0 else str(x)+"A"

o = [convert(i) for i in a]
print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
r = []
for i in range(10):
    r.append("el numero "+str(i))
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
r = ["el numero "+str(i) for i in range(10)]
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla', 'Guayaba':'rosa'}
for nombre, color in frutas.items():
    print (nombre, "es de color", color)

print()
r = [nombre+" es de color "+color for nombre, color in frutas.items()]
r
```

```
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
```

```
[ 'Fresa es de color roja',
  'Limon es de color verde',
  'Papaya es de color naranja',
  'Manzana es de color amarilla',
  'Guayaba es de color rosa']
```

Ejercicio

Construir una matriz aleatoria de cuatro letras (A,T,C,G) de tamaño 100x1000 luego cambiar las letras por las siguientes codificaciones:

Codificación 1: -2,-1,1,2

Codificación 2: 0,1,2,3

Codificación 3: 0001,0010,0100,1000

```
m=np.array([[ 'A', 'T'], ['C', 'G']])
```

m

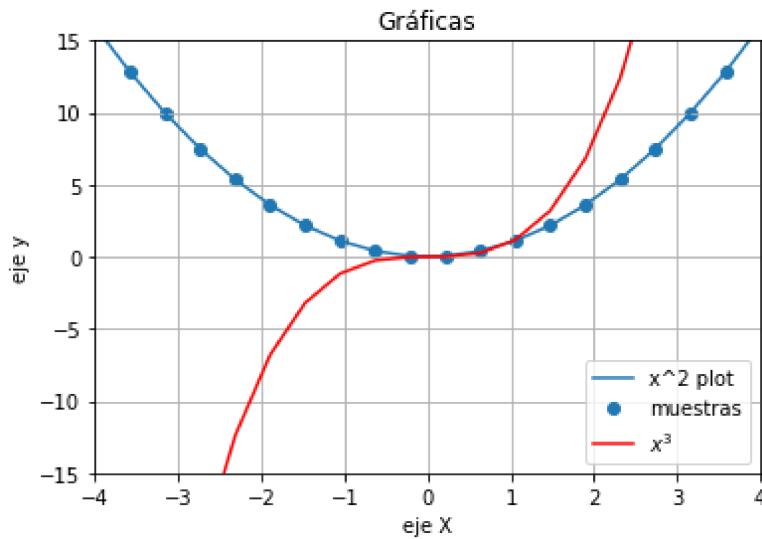
```
array([[ 'A', 'T'],
       ['C', 'G']], dtype='<U1')
```

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

x = np.linspace(-4,4,20)# El vector de tabulación
y=x**2
z=x**3

plt.plot(x, y, label="x^2 plot")
plt.scatter(x, y, label="muestras")
plt.plot(x, z, label="$x^3$", color="red")
plt.xlim([-4,4])
plt.ylim([-15, 15])
plt.legend()
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Gráficas')
```

Text(0.5, 1.0, 'Gráficas')

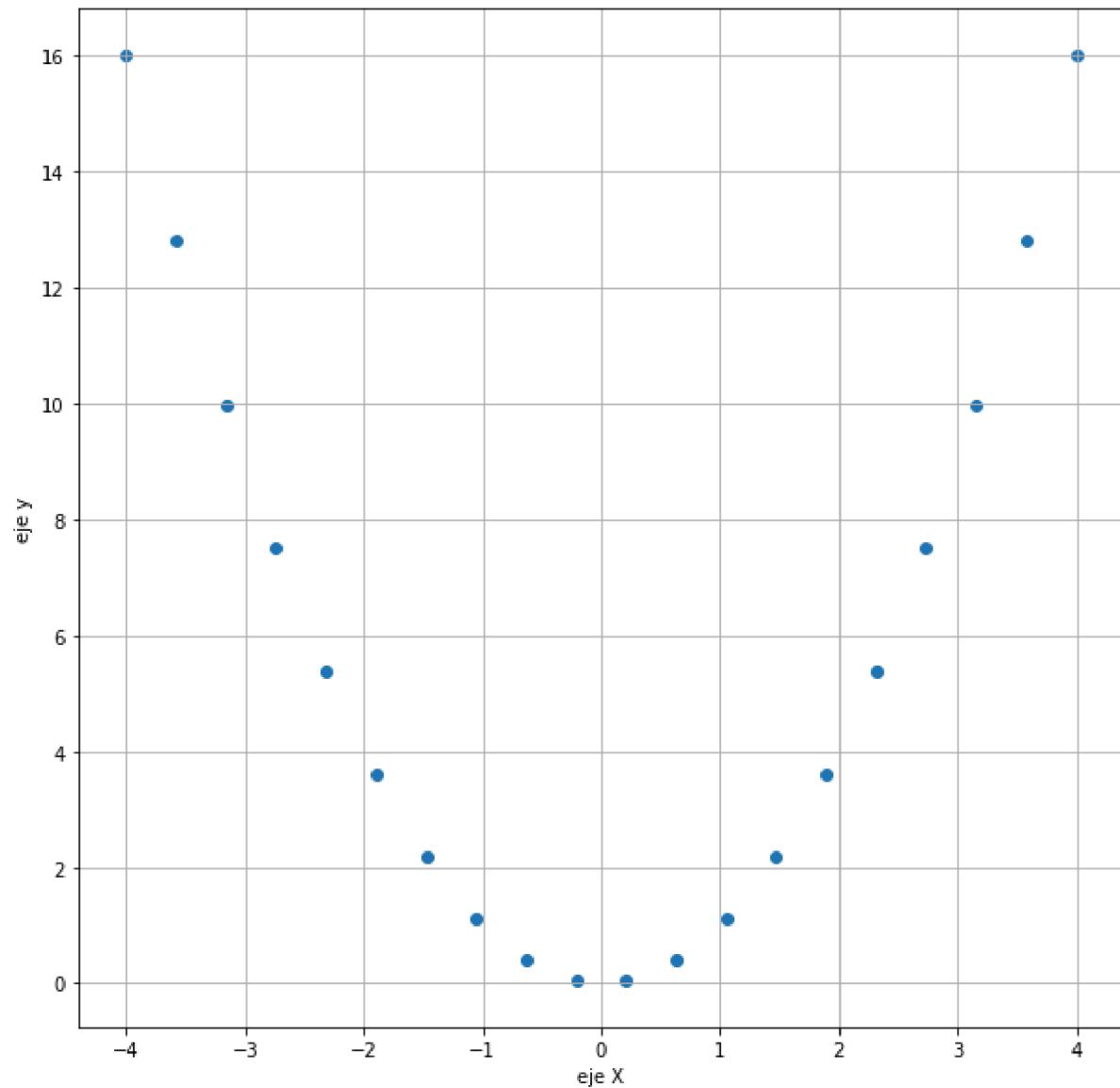


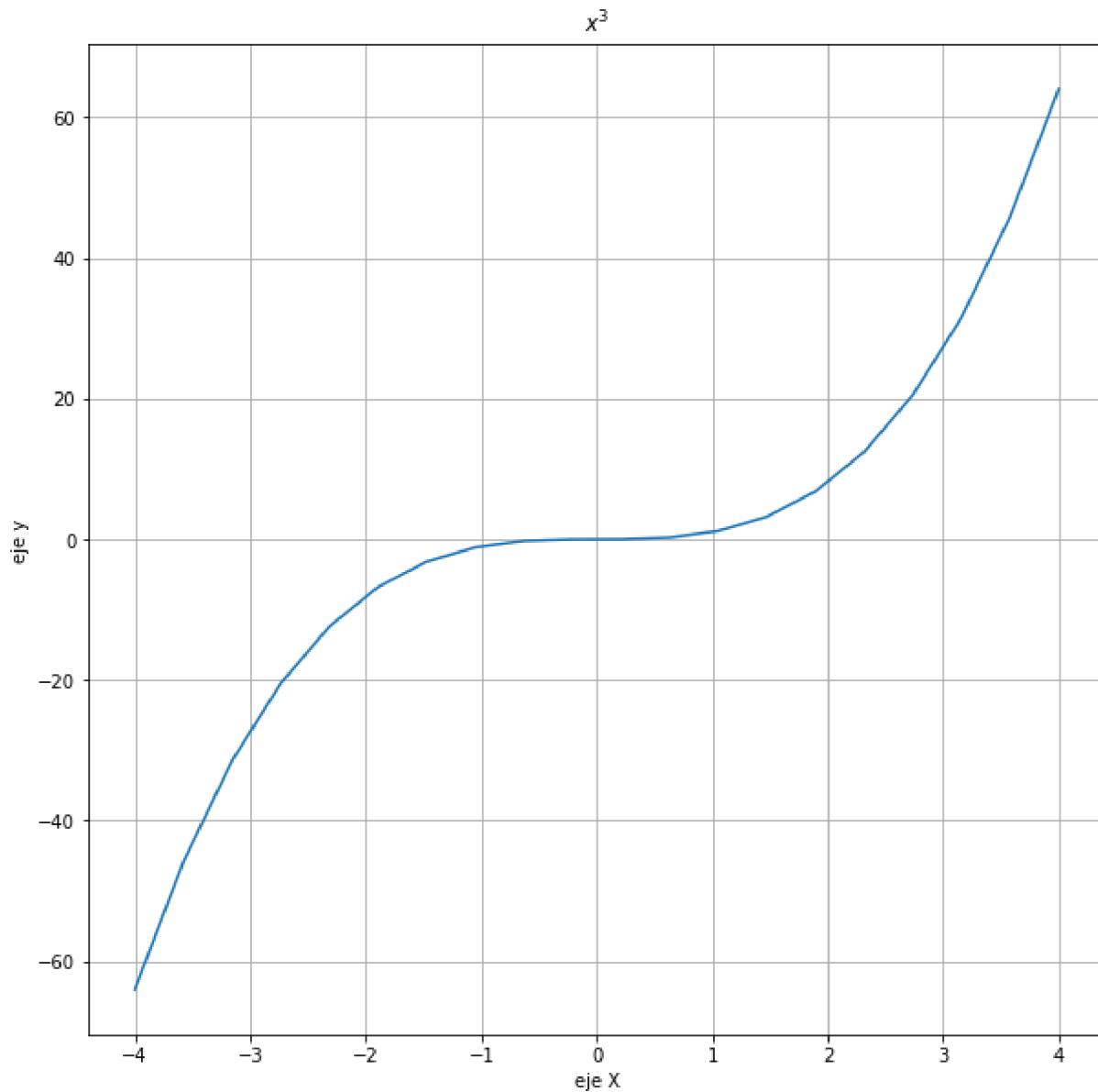
```
x = np.linspace(-4,4,20)
y=x**2

plt.figure(figsize=(10,10))
plt.scatter(x, y)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Parabola')

plt.figure(figsize=(10,10))
plt.plot(x, x**3)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('$x^3$')
```

Text(0.5, 1.0, '\$x^3\$')

Parabola

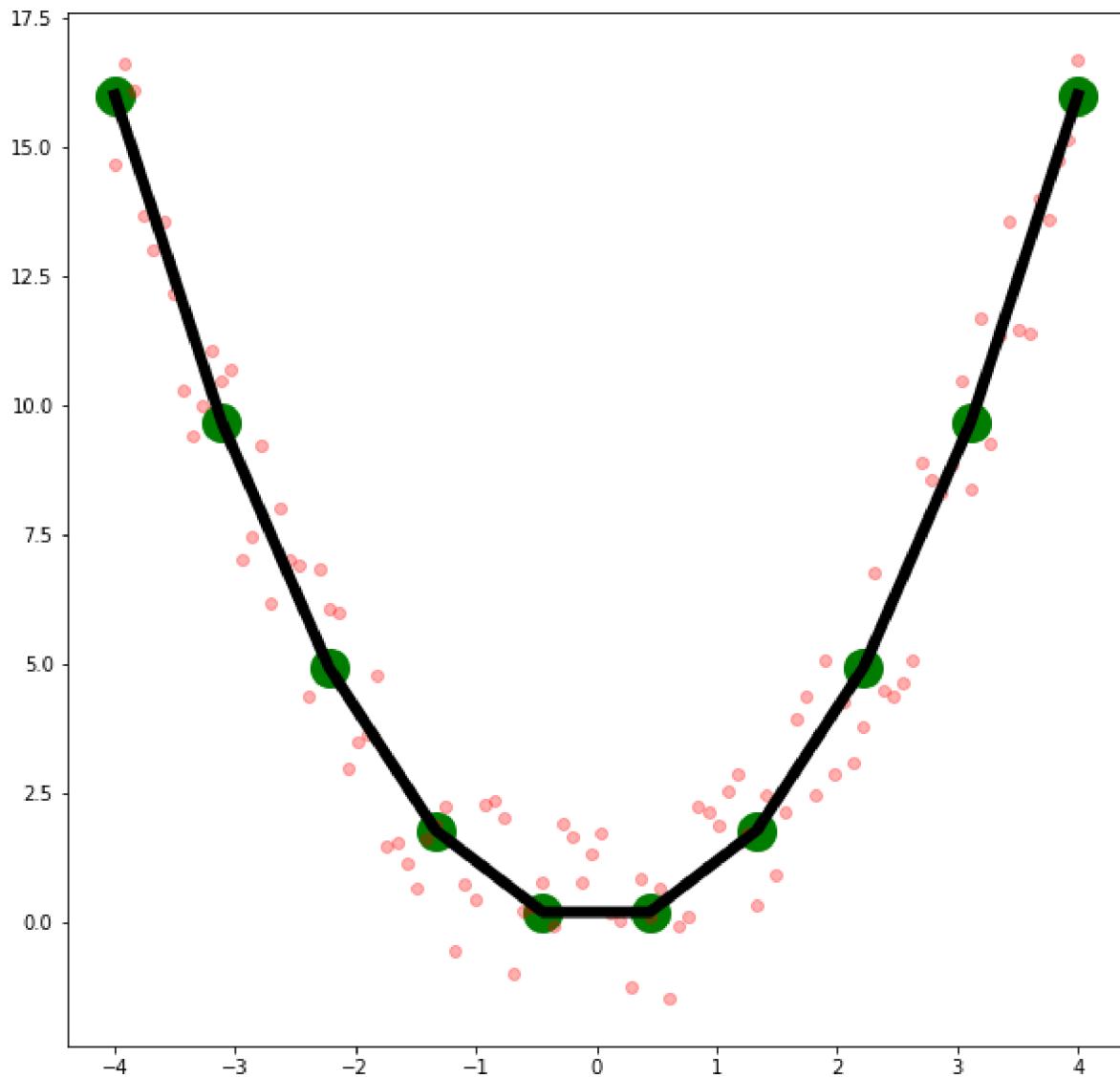


```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.figure(figsize=(10,10))
x = np.linspace(-4,4,10)

plt.plot(x, x**2, color="black", linewidth=6)
plt.scatter(x, x**2, c="green", s=400)

x_r = np.linspace(-4,4,100)
x_ruido = x_r**2 + (np.random.random(x_r.shape)-0.5)*4
plt.scatter(x_r,x_ruido, c="red", alpha=0.3)
```

```
<matplotlib.collections.PathCollection at 0x7ff4845f44f0>
```



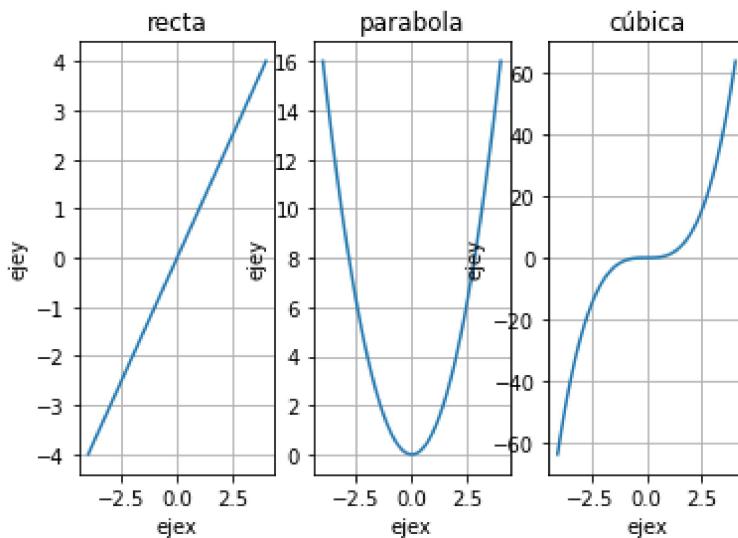
```

x=np.linspace(-4,4,100)
y=x
z=x**2
w=x**3
#plt.figure()
plt.subplot(1,3,1)
plt.plot(x,y)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('recta')
plt.grid()

#plt.figure()
plt.subplot(1,3,2)
plt.plot(x,z)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('parabola')
plt.grid()

#plt.figure()
plt.subplot(1,3,3)
plt.plot(x,w)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('cúbica')
plt.grid()

```



```

from google.colab import drive
drive.mount('/content/drive')

```

Mounted at /content/drive

```
#Rei
#path_data = "/content/drive/MyDrive/Machine Learning 2023-1 UManizales Maestría/Clase
#Arteaga
path_data = "/content/drive/MyDrive/Clases 2023-01/Programación Concurrente y Distribu

path_mesa = path_data+"/Mesa.jpg"
path_chestxray = path_data+"/ChestXRay.jpg"
```

```
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

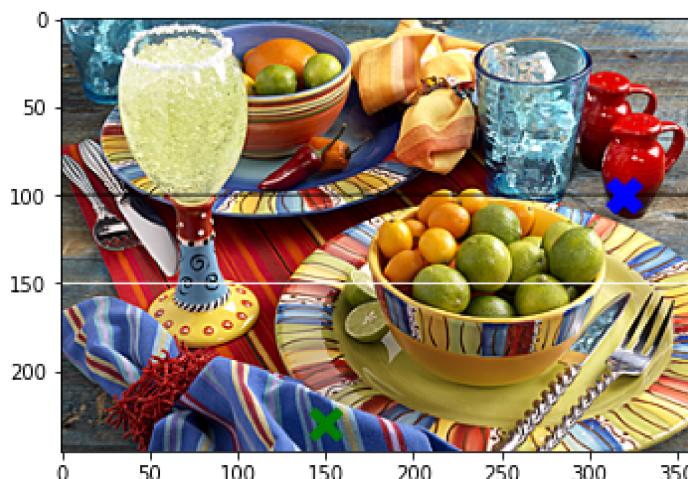
#Arteaga
img = io.imread(path_mesa)
img2 = io.imread(path_chestxray)

print("dimensiones", img.shape, "max", np.max(img), "min", np.min(img), "tipo", type(img))

plt.scatter(320,100, marker="x", s=200, linewidth=7, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print("pixel at blue marker ", img[100,320,:])
print("pixel at green marker", img[230,150,:])
img[100,:,:]=0
img[150,:,:]=255
#plt.grid() # remove gridlines
plt.imshow(img)
```

```
dimensiones (246, 360, 3) max 255 min 0 tipo <class 'numpy.ndarray'>
pixel at blue marker [75 0 5]
pixel at green marker [ 24  48 112]
```

<matplotlib.image.AxesImage at 0x7ff45c6ccf10>



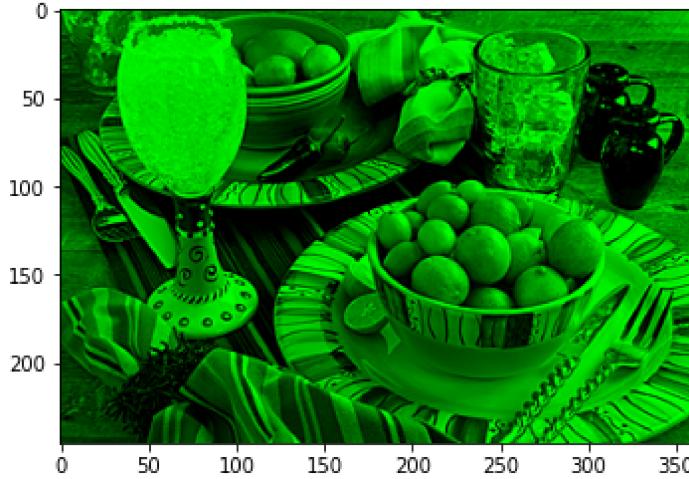
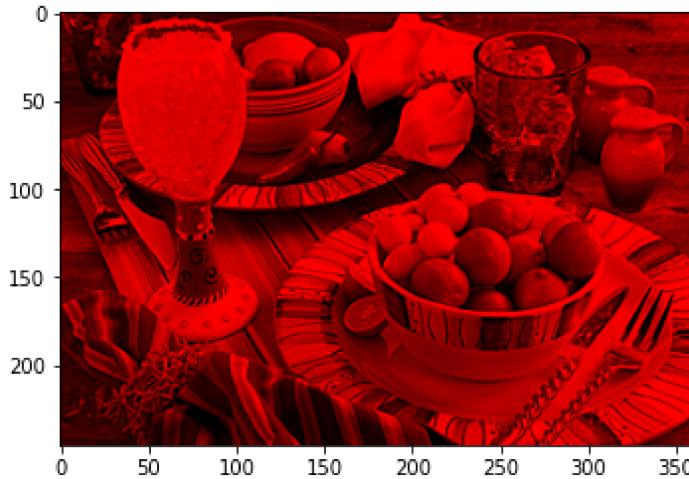
```
array([[[ 90, 108, 132],  
       [224, 254, 254],  
       [ 64, 110, 143],  
       ...,  
       [173, 188, 191],  
       [144, 162, 164],  
       [172, 189, 196]],  
  
      [[ 34,  61,  80],  
       [212, 223, 241],  
       [156, 201, 230],  
       ...,  
       [173, 192, 198],  
       [173, 193, 200],  
       [158, 187, 191]],  
  
      [[ 27,  57,  83],  
       [134, 141, 157],  
       [172, 231, 247],  
       ...,  
       [195, 205, 207],  
       [188, 193, 197],  
       [175, 186, 190]],  
  
      ...,  
  
      [[ 44,  52,  55],  
       [ 36,  45,  52],  
       [ 38,  43,  46],  
       ...,  
       [119, 113,  97],  
       [120, 122, 109],  
       [122, 125, 130]],  
  
      [[ 57,  47,  38],  
       [ 54,  46,  43],  
       [ 55,  52,  47],  
       ...,  
       [109,  99,  87],  
       [113, 110,  91],  
       [105, 108, 101]],  
  
      [[ 58,  56,  57],  
       [ 69,  70,  65],  
       [ 61,  66,  69],  
       ...,  
       [139, 133, 119],  
       [134, 119,  98],  
       [132, 123, 108]]], dtype=uint8)
```

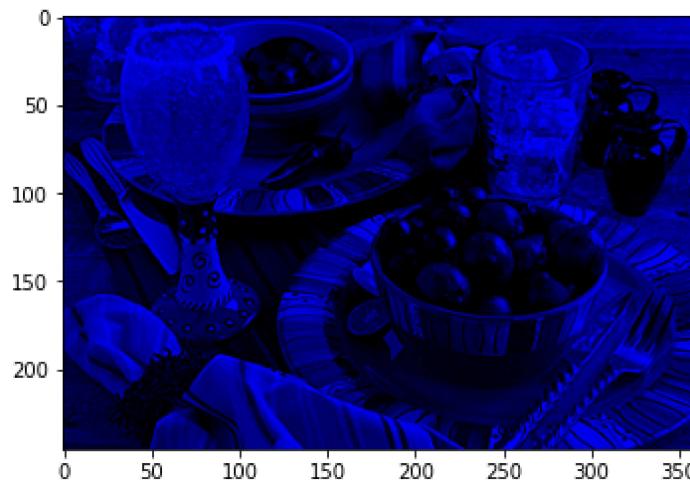
```
img = io.imread(path_mesa)
img[:, :, 1]=0
img[:, :, 2]=0
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 2]=0
plt.figure()
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 1]=0
plt.figure()
plt.imshow(img)
```

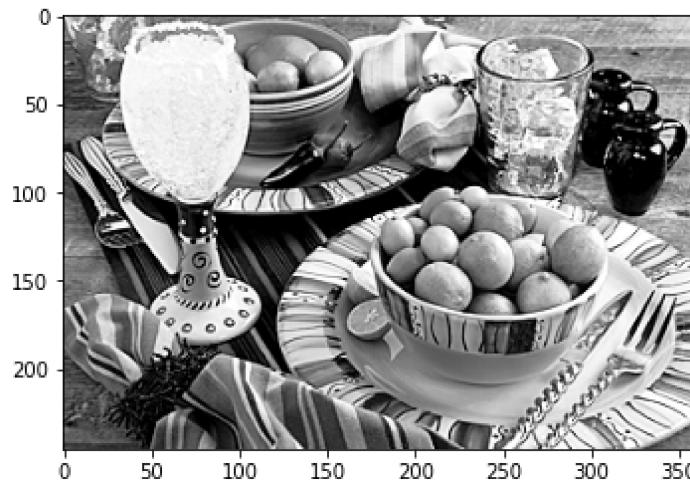
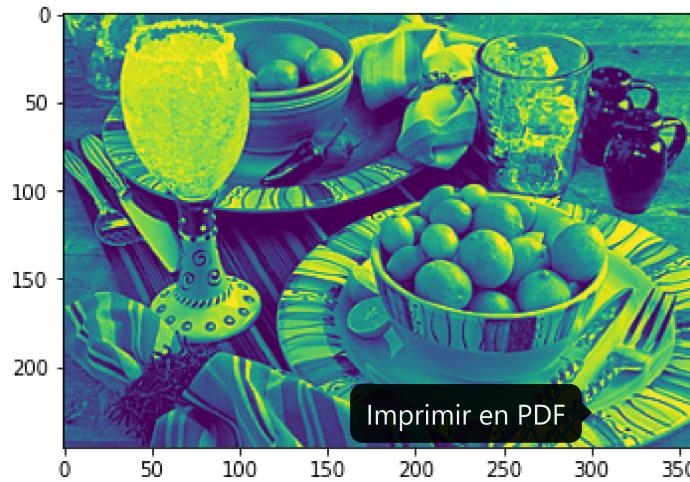
<matplotlib.image.AxesImage at 0x7ff45c4e0310>





```
img = io.imread(path_mesa)
plt.imshow(img[:, :, 1])
plt.figure()
plt.imshow(img[:, :, 1], cmap = plt.cm.Greys_r)
```

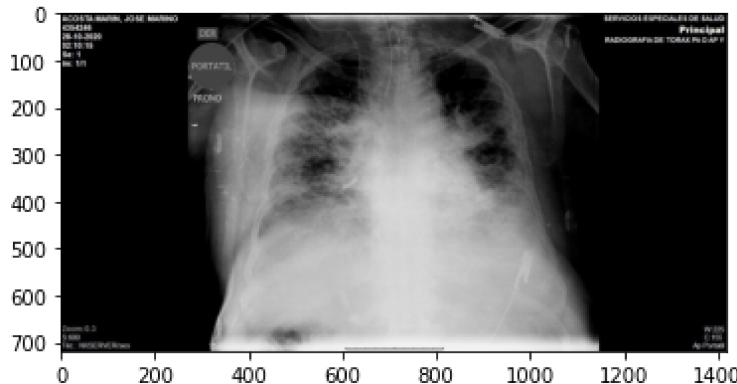
<matplotlib.image.AxesImage at 0x7ff45c3c1700>



```
# Otra imagen
print(img2.shape)
plt.imshow(img2)
```

(718, 1418, 3)

<matplotlib.image.AxesImage at 0x7ff45c3cd940>

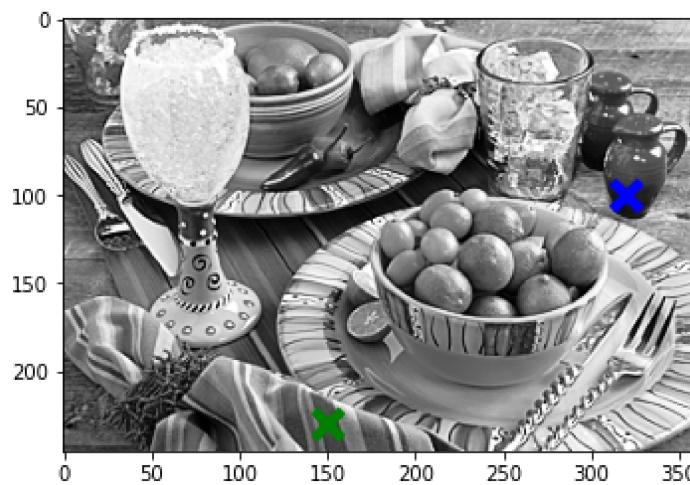


La manipulación de imágenes se *reduce* a realizar cálculos sobre los valores de luminosidad de cada pixel. Por ejemplo, obtenemos una versión en escala de grises promediando los valores RGB de cada pixel. Esto se hace de manera natural con la función `np.mean` y el argumento `axis` adecuado.

```
gimg = np.mean(img, axis=2)
print("dimensiones", gimg.shape, "max", np.max(gimg), "min", np.min(gimg))
plt.scatter(320,100, marker="x", s=200, linewidth=5, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print(img[100,320,:])
print("pixel at blue marker ", gimg[100,320])
print("pixel at green marker", gimg[230,150])
# plt.grid() # remove gridlines
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

dimensiones (246, 360) max 255.0 min 0.0
[75 0 5]
pixel at blue marker 26.666666666666668
pixel at green marker 61.333333333333336

<matplotlib.image.AxesImage at 0x7ff45c182640>



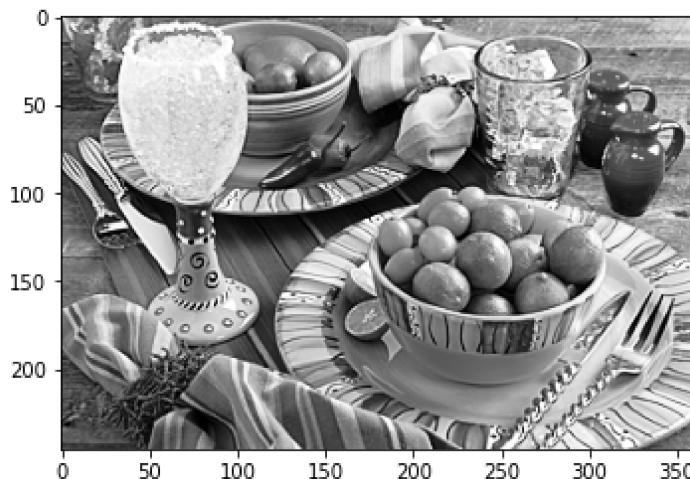
Ejercicio

También podemos acceder a porciones (**parches**) de la imagen usando la notación natural de matrices de Python. Si además, reducimos la luminosidad de cada pixel a la mitad lo que hacemos es oscurecer la imagen.

1. Cargar nuevamente la imagen de la mesa
2. Visualizarla
3. Obtener una nueva imagen promediando los 3 canales RGB
4. Visualizarla
5. Extraer el jarrón de la posición 50:100 y 300:350 (ver imagen anterior)
6. Visualizarla
7. Dividir todos los pixeles del parche extraído (item 5) entre 2
8. Compara los parches (visualiza usando los parámetros: `cmap=>gray<`, `vmin=0`, `vmax=255`), el original y el de división de cada pixel entre 2 ¿Qué observa?

```
# Escribe tu código aquí
gimg = np.mean(img, axis=2)
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

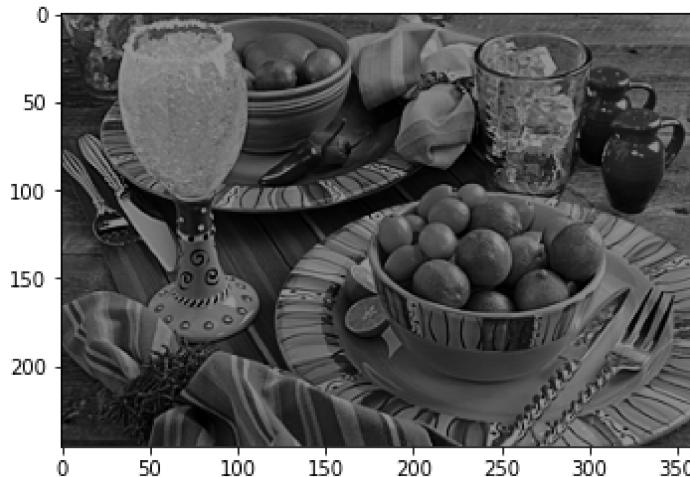
```
<matplotlib.image.AxesImage at 0x7ff45c160760>
```



```
gimg2=gimg/2
```

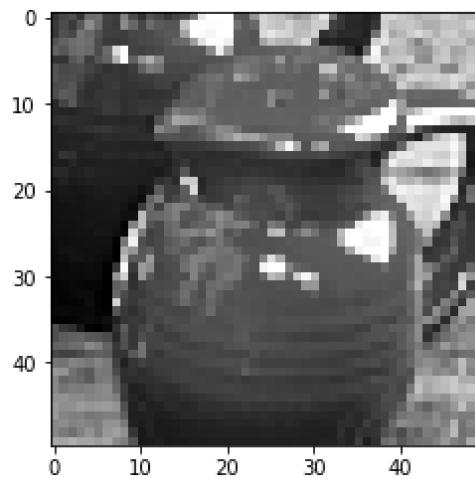
```
plt.imshow(gimg2, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff45c0a1b80>
```



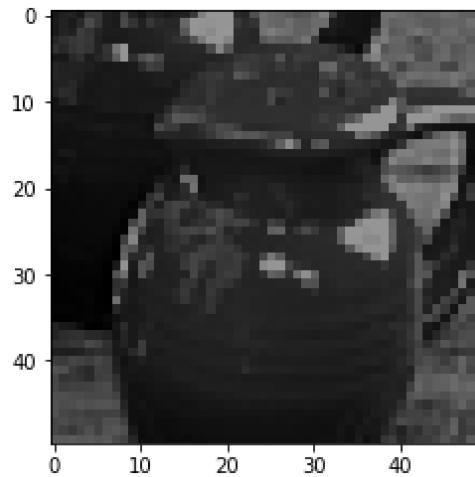
```
jarron=gimg[50:100 , 300:350 ]  
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ecbee80>
```



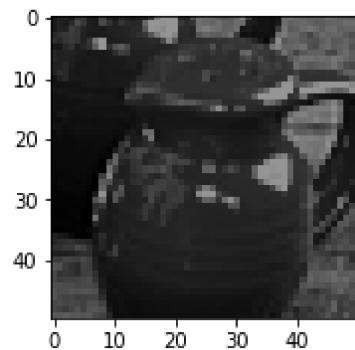
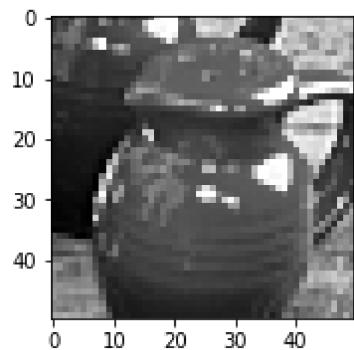
```
jarronOscurito=jarron/2  
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ec2b160>
```



```
plt.subplot(1,2,1)  
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)  
plt.subplot(1,2,2)  
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44e835fa0>
```



Método de euler para ecuaciones diferenciales.

Contenido

- Método de euler para ecuaciones diferenciales.
- Actividad:

El método de Euler es un procedimiento numérico simple para resolver ecuaciones diferenciales ordinarias (EDOs) con un valor inicial dado. Es un método iterativo que comienza en un punto conocido y avanza paso a paso hasta llegar al punto deseado, utilizando la pendiente de la solución (es decir, la derivada) para avanzar en cada paso.

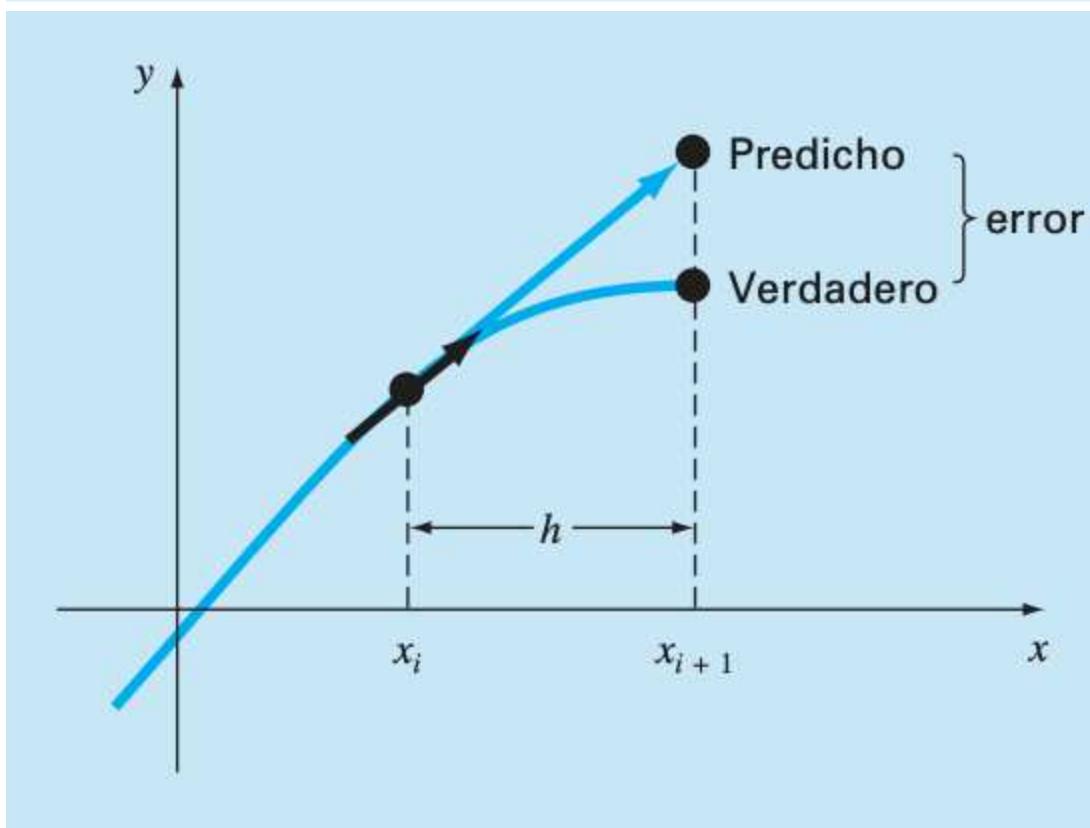
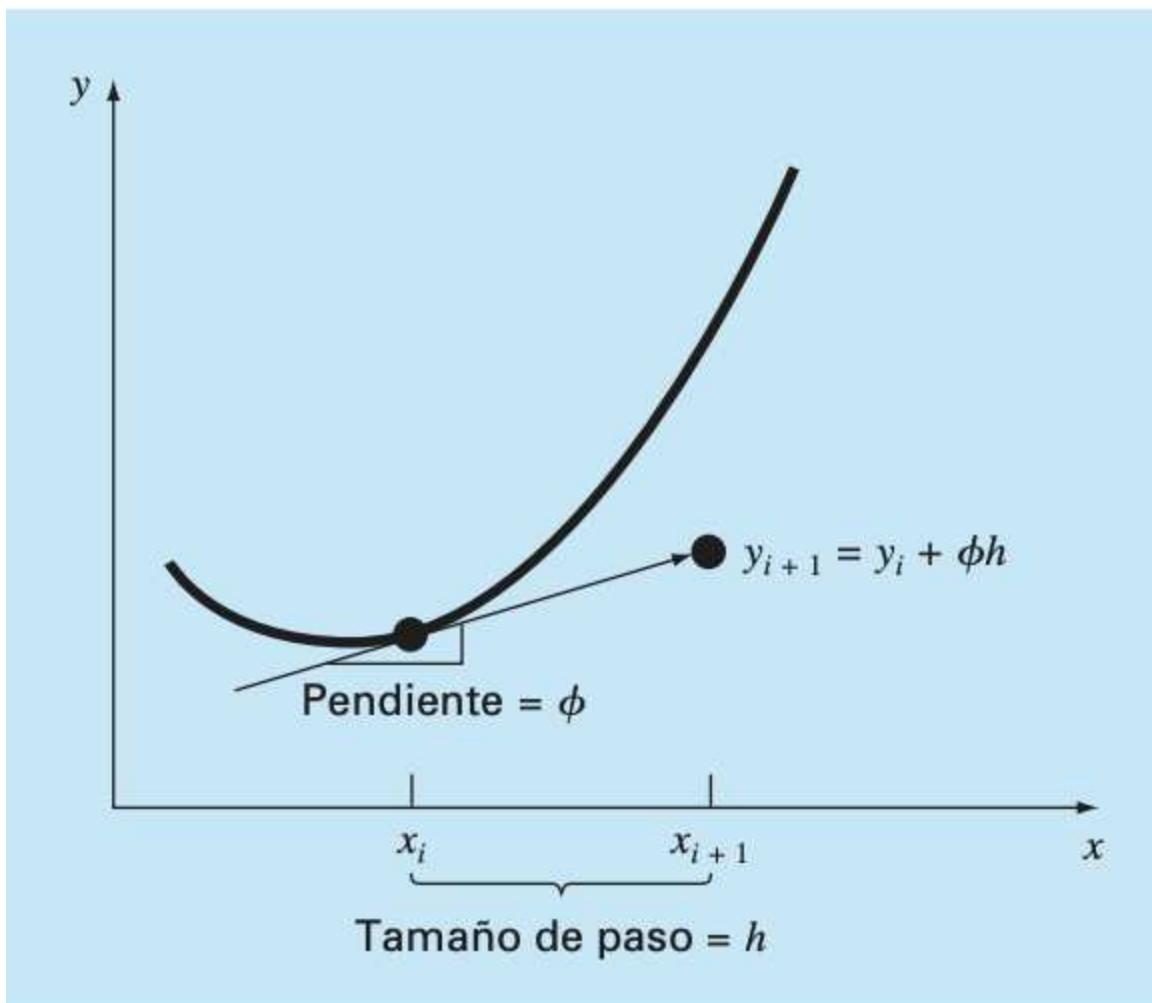
En este capítulo se enfoca la solución de ecuaciones diferenciales ordinarias de la forma:

$$\frac{dy}{dx} = f(x, y)$$

en términos matemáticos calculabamos el siguiente de la forma:

$$y_{i+1} = y_i + mh$$

Donde m es la pendiente estimada que se usa para extrapolar desde un valor anterior y_i a un nuevo valor y_{i+1} en una distancia h . Este valor se usa paso a paso para calcular un valor posterior y trazar la trayectoria de la solución. Normalmente, la pendiente representa la derivada de la función en el punto y_i



Algoritmo de Euler para la integración

1. Hacer $y_{i+1} = y_i + h f'(x_i, y_i)$
2. Hacer $x_i = x_{i+1}$

Ejercicio:

Hallar la integral geométrica de la función $y' = 2t$ con una condición inicial $y(0)=0$ y un tiempo de integración de 0 hasta 0.6 y un paso de integración de 0.1

```
# Definimos la función
import sympy as sp
import numpy as np
import pandas as pd

t = sp.symbols('t')
dy = 2*t

y0 = 0 # Condición inicial
ti = 0 # Tiempo inicial
h = 0.1 # paso de integración
tf = 0.6 # Tiempo final

# Definimos los vectores donde almacenaremos el resultado.
x = np.arange(ti, tf + h, h)
y = np.empty_like(x)
y[0] = y0
print(x)
print(y)
```

```
[0. 0.1 0.2 0.3 0.4 0.5 0.6]
[0.00000000e+000 1.68821824e+195 2.51313104e+180 4.44032494e+252
 5.40026746e-310 0.00000000e+000 0.00000000e+000]
```

```
columnas = ['it','t','yi','yi+1']
tabla = pd.DataFrame(columns=columnas)
for i in range(0,len(y)-1):
    y[i+1] = y[i] + h * dy.subs({t:x[i]})
    nueva_fila = pd.DataFrame(data={'it':[i+1], 't':[round(x[i],2)], 'yi':[round(y[i],2)]})
    tabla = pd.concat([tabla,nueva_fila], ignore_index=True)

tabla.head()
```

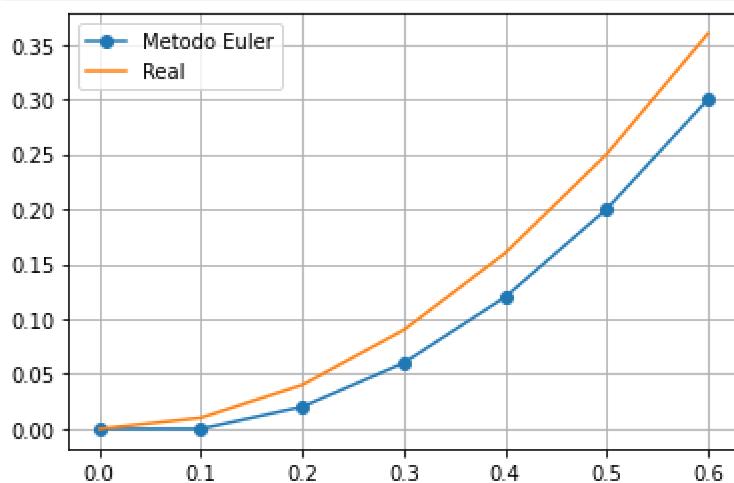
```
C:\Users\ricar\AppData\Local\Temp\ipykernel_9580\3804453467.py:6: FutureWarning: The b
    tabla = pd.concat([tabla,nueva_fila], ignore_index=True)
```

	it	t	yi	yi+1
0	1	0.0	0.00	0.00
1	2	0.1	0.00	0.02
2	3	0.2	0.02	0.06
3	4	0.3	0.06	0.12
4	5	0.4	0.12	0.20

```
import matplotlib.pyplot as plt
plt.figure()
fig, ax = plt.subplots()
ax.grid()
ax.plot(x,y, marker='o', label ="Metodo Euler")
ax.plot(x, x**2, label="Real")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x1f5bb1e7730>
```

```
<Figure size 432x288 with 0 Axes>
```

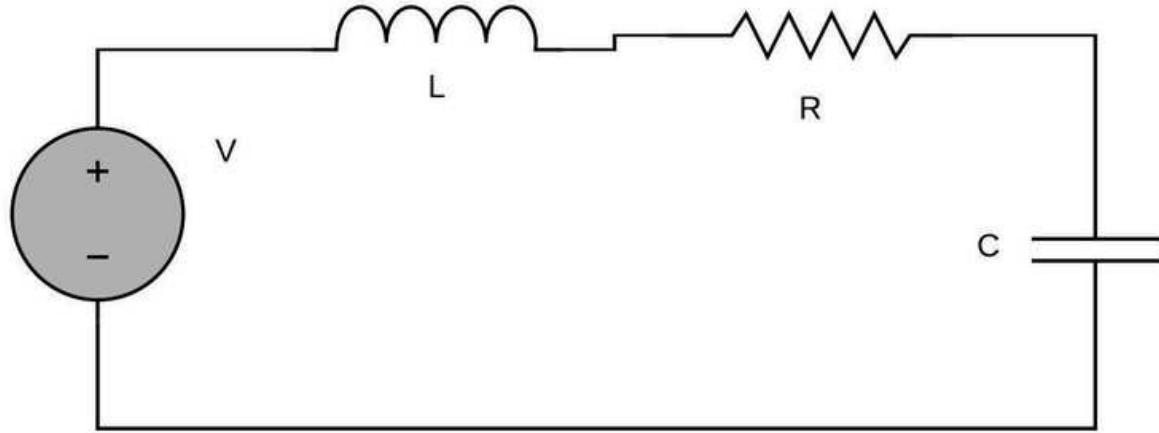


Circuito RLC: Análisis analítico.

Vamos a deducir el modelo del siguiente circuito:

$$\frac{di}{dt} = -\frac{R}{L}i(t) - \frac{1}{L}v(t) + \frac{1}{L}vi$$

$$\frac{dv}{dt} = \frac{1}{C}i(t)$$



Definamos las ecuaciones del sistema en [sympy](#)

```
from sympy import *

t, vi, L, R, C = symbols("t vi L R C")
C1,C2 = symbols("C1 C2")
v = Function("v")(t)
i = Function("i")(t)

didt=i.diff(t)
dvdt=v.diff(t)
expr1 = Eq(didt, (1/L)*vi-(R/L)*i-(1/L)*v)
expr1
```

$$\frac{d}{dt}i(t) = -\frac{Ri(t)}{L} + \frac{vi}{L} - \frac{v(t)}{L}$$

```
# Asumimos valores de R,L y C = 1

expr1=expr1.subs(L, 1).subs(R,1).subs(C,1).subs(vi,1)
expr1
```

$$\frac{d}{dt}i(t) = -i(t) - v(t) + 1$$

```
expr2 = Eq(dvdt, (1/C)*i)
expr2
```

$$\frac{d}{dt}v(t) = \frac{i(t)}{C}$$

```
# Asumimos valores de R,L y C = 1

expr2=expr2.subs(L,1).subs(R,1).subs(C,1).subs(vi,1)
expr2
```

$$\frac{d}{dt}v(t) = i(t)$$

```
init_printing(use_latex=True)
s=dsolve([expr1,expr2])
s
```

$$[i(t) = -(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2), v(t) = C_1e^{-t/2}\cos(\sqrt{3}t/2) - C_2e^{-t/2}\sin(\sqrt{3}t/2) + \sin^2(\sqrt{3}t/2) + \cos^2(\sqrt{3}t/2)]$$

```
# Vamos a hallar los valores de las ctes C1 y C2
eq1 = Eq(s[0].rhs.subs({t:0}).evalf(), 0.1)
eq2 = Eq(s[1].rhs.subs({t:0}).evalf(), 0.1)
sol = solve([eq1,eq2], [C1,C2])
print(sol)
```

$$\{C1: -0.900000000000000, C2: 0.404145188432738\}$$

```
eq1
```

$$-0.5C_1 - 0.866025403784439C_2 = 0.1$$

```
s[0].rhs
```

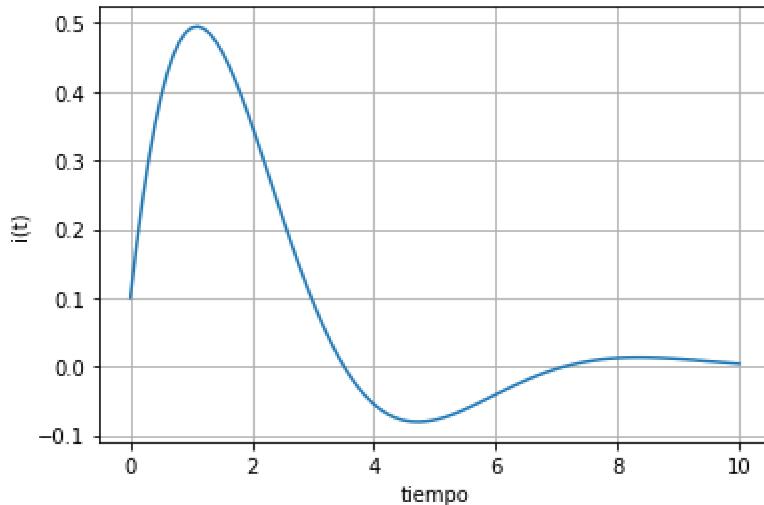
$$-(C_1/2 + \sqrt{3}C_2/2)e^{-t/2}\cos(\sqrt{3}t/2) - (\sqrt{3}C_1/2 - C_2/2)e^{-t/2}\sin(\sqrt{3}t/2)$$

```
## Generemos dos funciones que permitan reemplazar los valores de t dados para graficar

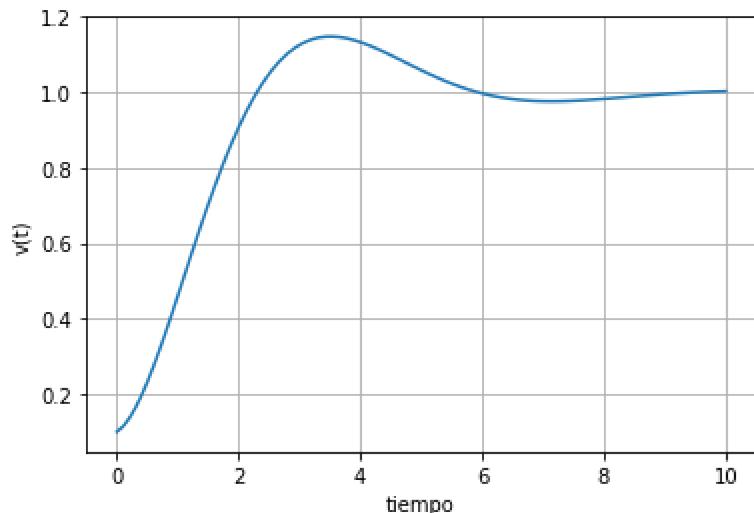
e1 = lambdify(t,s[0].rhs.subs(sol), 'numpy')
e2 = lambdify(t,s[1].rhs.subs(sol), 'numpy')
```

```
t_vals = np.linspace(0,10,100)
```

```
plt.plot(t_vals, e1(t_vals))
plt.xlabel('tiempo')
plt.ylabel('i(t)')
plt.grid(True)
```



```
plt.plot(t_vals, e2(t_vals))
plt.xlabel('tiempo')
plt.ylabel('v(t)')
plt.grid(True)
```



las gráficas para la corriente $i(t)$ y el voltaje $v(t)$ en función del tiempo para el circuito RLC analizado. Como puedes ver, ambas funciones muestran un comportamiento oscilatorio amortiguado, que es típico en este tipo de circuitos.

```

def euler_method_system(funcs, y0, t_range, dt):
    """
    Método de Euler para un sistema de ecuaciones diferenciales.

    :param funcs: Lista de funciones que representan el sistema de ecuaciones diferenciales.
    :param y0: Condiciones iniciales para cada variable en el sistema.
    :param t_range: Rango de tiempo como una tupla (inicio, fin).
    :param dt: Paso de tiempo.
    :return: Tupla de listas (tiempos, valores de cada variable).
    """
    t_values = np.arange(t_range[0], t_range[1], dt)
    y_values = [np.array(y0)]

    for t in t_values[:-1]:
        y_current = y_values[-1]
        y_next = y_current + dt * np.array([f(*y_current, t) for f in funcs])
        y_values.append(y_next)

    return t_values, np.array(y_values).T

# Funciones para el sistema de ecuaciones diferenciales
def func_i(i, v, t):
    return 1 - i - v

def func_v(i, v, t):
    return i

# Paso de tiempo para el Método de Euler
dt = 0.1

# Condiciones iniciales y rango de tiempo
initial_i = 0.1
initial_v = 0.1
time_range = (0, 10)

# Condiciones iniciales para i y v
initial_conditions = [initial_i, initial_v]

# Solución aproximada para i(t) y v(t) usando el Método de Euler para sistemas
t_values_euler, [i_values_euler_system, v_values_euler_system] = euler_method_system(
    [func_i, func_v], initial_conditions, time_range, dt
)

# Graficando soluciones analíticas y aproximadas
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(t_vals, e1(t_vals), label='Analítico i(t)')
plt.plot(t_values_euler, i_values_euler_system, label='Aproximado i(t)', linestyle='--')
plt.xlabel('Tiempo')
plt.ylabel('Corriente (i(t))')
plt.title('Corriente en función del tiempo')
plt.grid(True)
plt.legend()

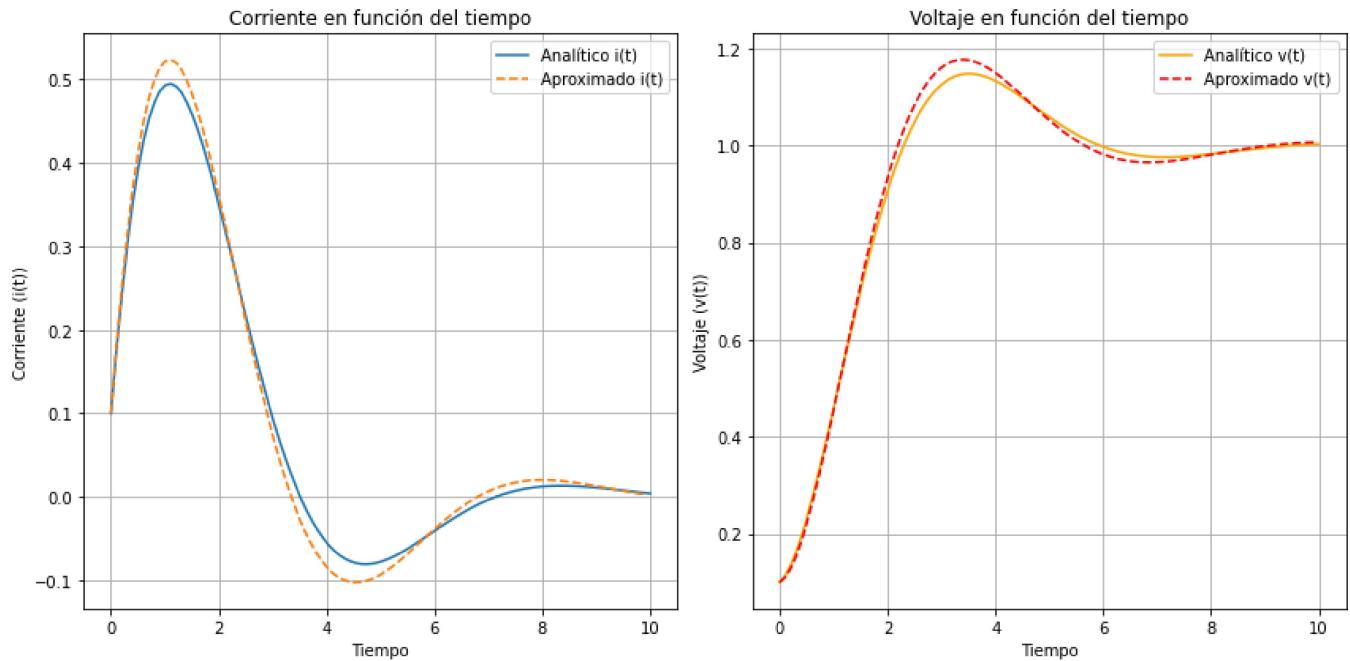
```

```

plt.subplot(1, 2, 1)
plt.plot(t_vals, e2(t_vals), label='Analítico v(t)', color='orange')
plt.plot(t_values_euler, v_values_euler_system, label='Aproximado v(t)', linestyle='--')
plt.xlabel('Tiempo')
plt.ylabel('Voltaje (v(t))')
plt.title('Voltaje en función del tiempo')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

```



Ajustar el código para que, dado un sistema de ecuaciones diferenciales, pueda calcular la solución del mismo dadas unas condiciones iniciales y un paso con el método de euler.

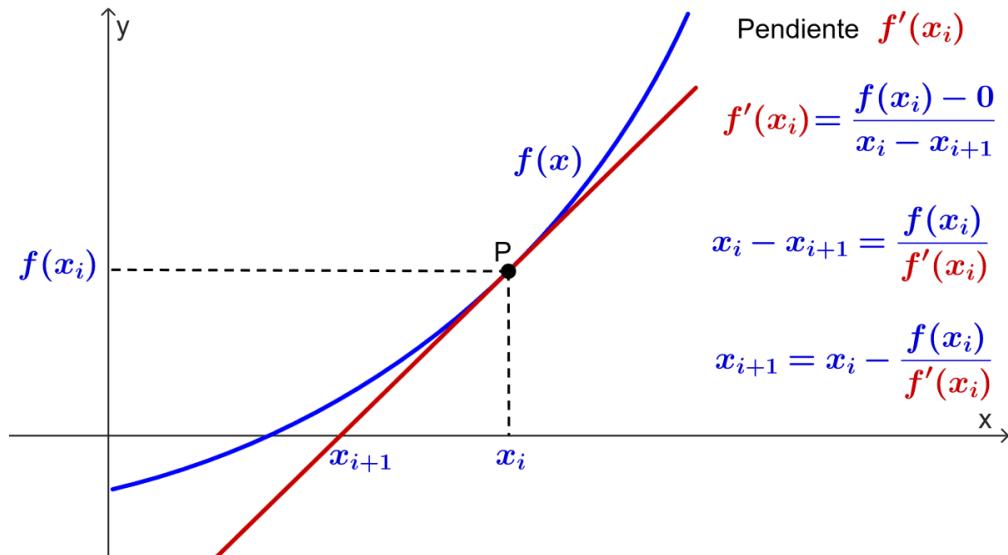
Método de newton-raphson

Contenido

- Algoritmo de Newton Raphson
- Veamos un ejemplo práctico
- Ejercicio:

Es el método más usado y trabaja como referencia la pendiente (m) de la recta tangente en una raíz x_i incial para hallar el x_{i+1} .

Este método tiende a acercarse muy rápido a la raíz. Pero también puede tender a la divergencia rápidamente.



$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

$$f'(x_i)[x_i - x_{i+1}] = f(x_i)$$

$$f'(x_i)(x_i) - f'(x_i)(x_{i+1}) = f(x_i)$$

$$f'(x_i)(x_{i+1}) = f'(x_i)(x_i) - f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Algoritmo de Newton Raphson

1. Se calcula el siguiente x_{i+1} con la formula.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

2. Se repite el paso 1 hasta convergencia.

Error relativo

$$e_r = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right|$$

Tolerancia

Precisión que se requiere en el método!!

Veamos un ejemplo práctico

Hallar la raíz de $f(x) = e^{-x} - x$ con el método de Newton Raphson y una condición inicial $x_i = 0$ y $tol = 1\%$

Solución

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

```
# Tomamos el intervalo inicial y grafiquemos la función dentro de ese valor

x = sp.symbols('x')

y = sp.E**(-x)-x
print("La función es: ",y)

xi = 0

print(f"El punto inicial es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi : f({xi})={fxi}")
```

La función es: $-x + \exp(-x)$
 El punto inicial es: 0
 la función evaluada en xi : $f(0)=1$

```
# @title Gráfica incial de la función y $x_i$ dado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

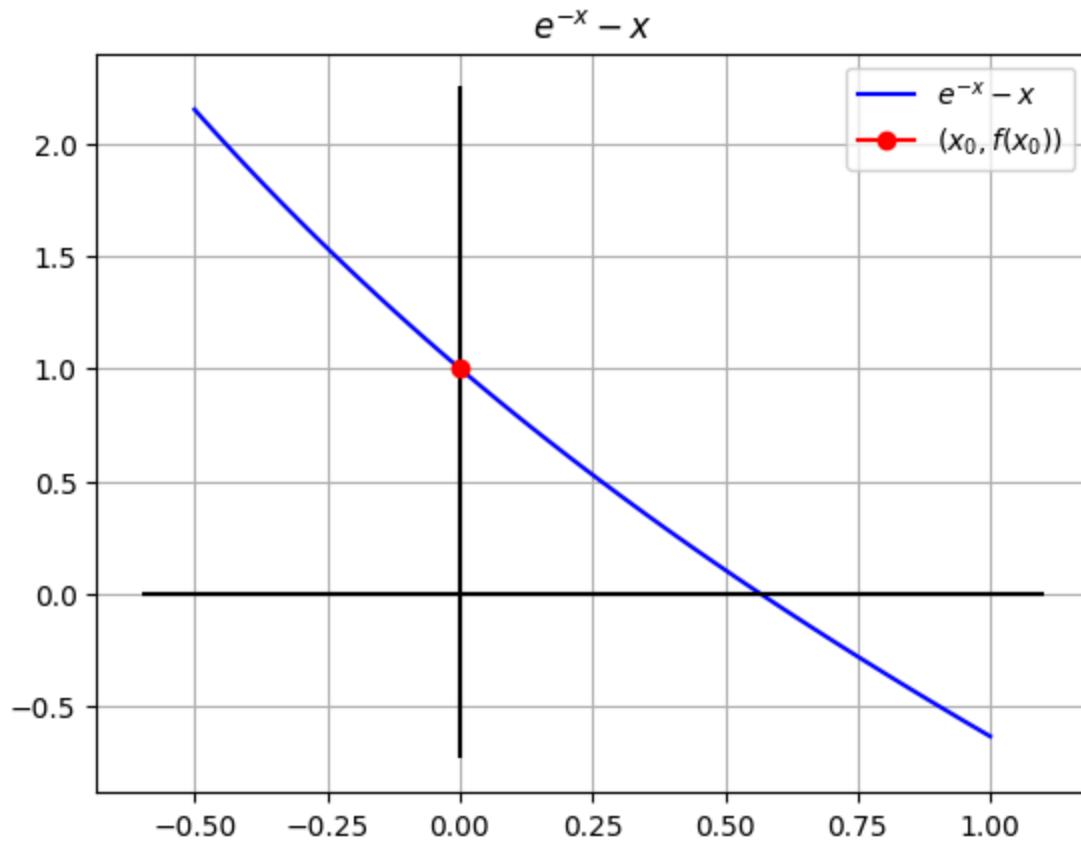
## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



Formula para el cálculo del x_{i+1} en el ejercicio dado

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{e^{-x} - x}{-e^{-x} - 1}$$

```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

La derivada de la función $f(x)=-x + \exp(-x)$ -> $f'(x)= -1 - \exp(-x)$
 La derivada evaluada en xi $f'(xi)= -2$

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

La raíz supuesta está en x=0.5000

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

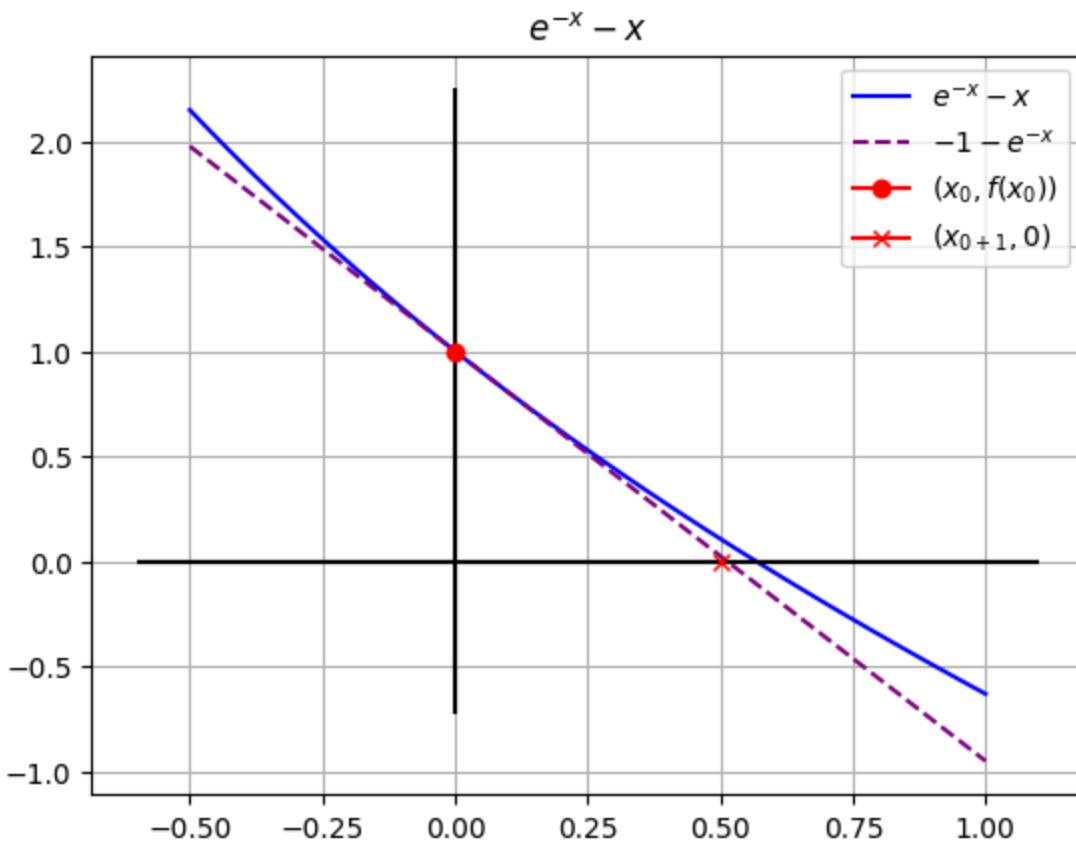
## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_0,f(x_0))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_{0+1},0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

El error del cálculo es: 100.0%

```
# Generemos la tabla de iteraciones
# @title Tabla de iteraciones
import pandas as pd

columnas = ['xi','Xi+1','f(xi)','f'(Xi)', 'er(%)']

primer_iter = {'xi':[xi], 'Xi+1':[xs], 'f(xi)':[fxi], "f'(Xi)": [dfxi], 'er(%)':[round(er,1)]}

tabla = pd.DataFrame(data=primer_iter, columns=columnas)
tabla.head(1)
```

	xi	Xi+1	f(xi)	f'(Xi)	er(%)
0	0	0.5000	1	-2	100.0

Sigamos iterando...

```
# Nuestro x_i+1 ahora se convierte en nuestro x_i
xi = xs
```

```
print(f"El punto inicial ahora es: {xi} ")

fxi = round(y.subs({x: xi}), 4)
print(f"la función evaluada en xi ahora es: f({xi})={fxi}")
```

El punto inicial ahora es: 0.5000
la función evaluada en xi ahora es: f(0.5000)=0.1065

```
# @title Gráfica incial de la función y $x_i$ para la iteración 2
import numpy as np
plt.figure()

r = np.linspace(-0.5, 1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

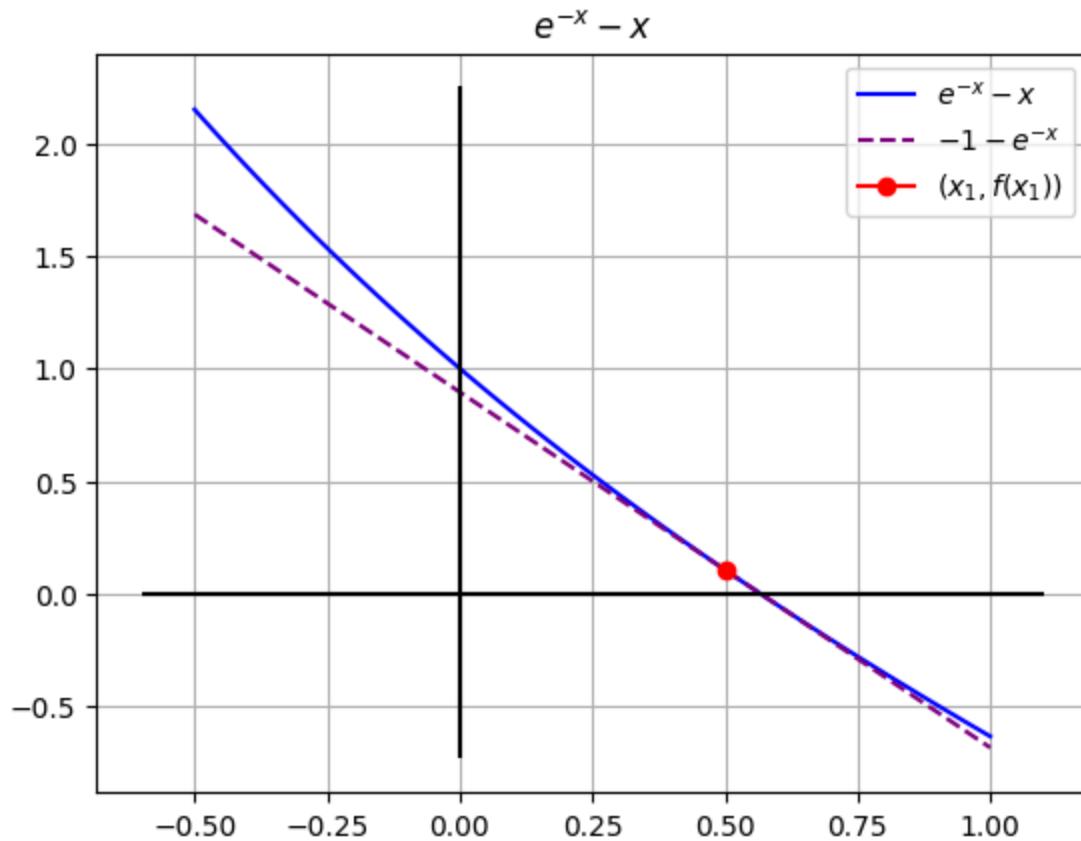
h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial
ax.plot([xi],[fxi], color='red', marker='o', label='${x_1,f(x_1)}$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
# Calculamos la derivada de la función
dy = y.diff()
print(f"La derivada de la función f(x)={y} -> \t f'(x)= {dy}")

# Evaluemos la derivada en x_i

dfxi = round(dy.subs({x:xi}),4)
print(f"La derivada evaluada en xi f'(xi)= {dfxi}")
```

La derivada de la función $f(x)=-x + \exp(-x)$ -> $f'(x)= -1 - \exp(-x)$
 La derivada evaluada en xi $f'(xi)= -1.605$

```
# Calculemos el x_i+1 (siguiente)

xs = round(xi - ((fxi)/(dfxi)),4)
print(f"La raíz supuesta está en x={xs}")
```

La raíz supuesta está en $x=0.5664$

```
# @title Gráfica función con el $x_i$ y $x_{i+1}$ calculado
import numpy as np
plt.figure()

r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

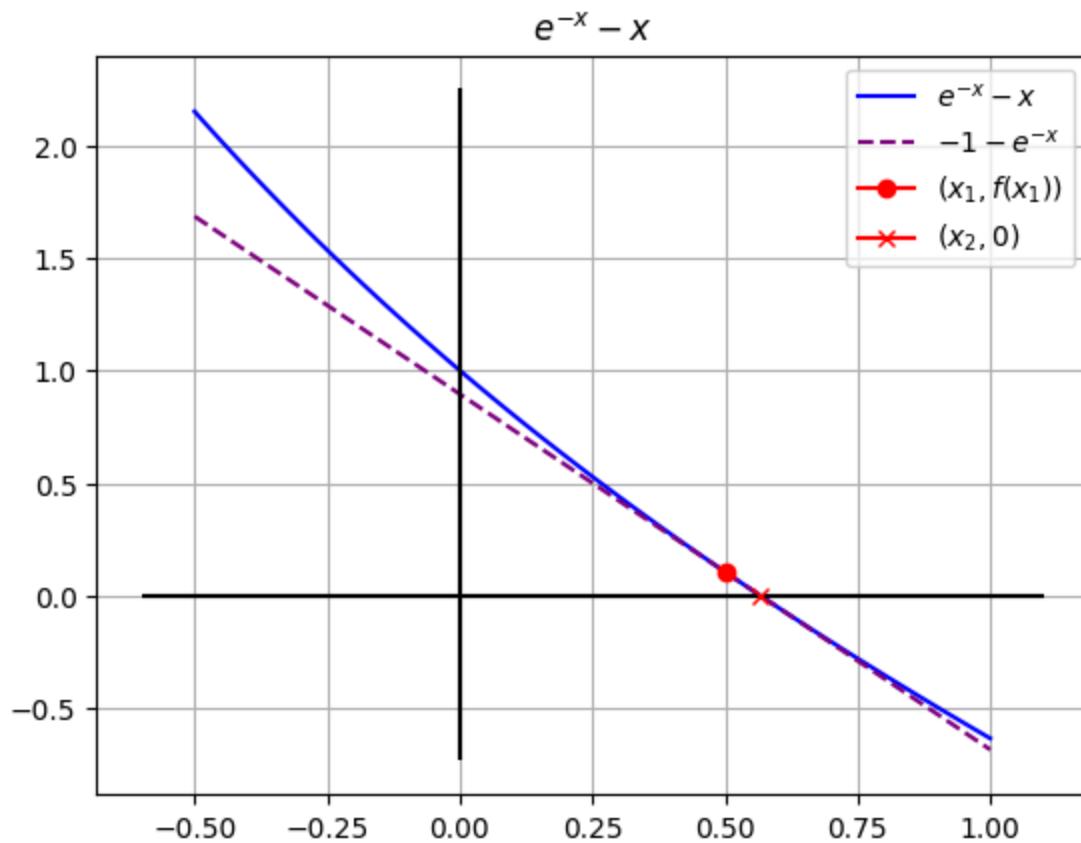
## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

## Punto inicial

ax.plot([xi],[fxi], color='red', marker='o', label='$(x_1,f(x_1))$')
ax.plot([xs],[0], color='red', marker='x', label='$(x_2,0)$')

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()
```

<Figure size 640x480 with 0 Axes>



```
# Calculemos el error
er = np.abs((xs-xi)/(xs)) * 100
print(f"El error del cálculo es: {round(er,1)}%")
```

El error del cálculo es: 11.7%

```
# @title Actualizamos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
tabla = tabla.append(nueva_fila, ignore_index=True)
tabla.head()
```

```
<ipython-input-141-30e46d477870>:3: FutureWarning: The frame.append method is deprecated
tabla = tabla.append(nueva_fila, ignore_index=True)
```

	xi	$Xi+1$	$f(xi)$	$f'(Xi)$	$er(%)$
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72

Ahora veamos el algoritmo completo

```

# @title iteremos hasta que el error sea de 1%
from IPython.display import HTML, display, clear_output
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import sympy as sp
import pandas as pd

x = sp.symbols('x')

y = sp.E**(-x)-x
xi = 0

print("La función es: ",y)

columnas = ['xi','Xi+1','f(xi)','f'(Xi)','er(%)']
tabla = pd.DataFrame(columns=columnas)

tol = 1

er = tol+1
it = 1

ims = []

while er > tol:
    fxi = round(y.subs({x: xi}), 4)

    #####
    plt.figure()
    fig, ax = plt.subplots()

    r = np.linspace(-0.5,1, 100)
    fx = [y.subs({x:x_i}) for x_i in r]

    ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

    ## Plano cartesiano (Ejes)
    ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
    ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')

    ## Punto inicial

    ax.plot([xi],[fxi], color='red', marker='o', label=f'$(x_{it}), f(x_{it})) = ({xi},{fxi})$')

    ## Calculemos la derivada
    dy = y.diff()
    ## Evaluemos la derivada en x_i
    dfxi = round(dy.subs({x:xi}),4)

    ## Calculemos el x_i+1 (siguiente)
    xs = round(xi - ((fxi)/(dfxi)),4)
    ax.plot([xs],[0], color='red', marker='x', label=f'$(x_{it+1},0) = ({xs},0)$' )

    ims.append(ax)

    tol = 10**-it
    er = abs(fxi - xs)
    it += 1
    xi = xs

```

```
## Pintar la recta tangente
h=0.1
dfx = (y.subs({x:xi+h})-y.subs({x:xi}))/h # derivative
tan = y.subs({x:xi})+dfx*(r-xi) # tangent
ax.plot(r,tan,color='purple',linestyle='--',label=f"${sp.latex(dy)}$")

ax.set_title("$e^{-x} - x$")
ax.grid()
ax.legend()
plt.show()

## Calculo del error
er = np.abs((xs-xi)/(xs)) * 100

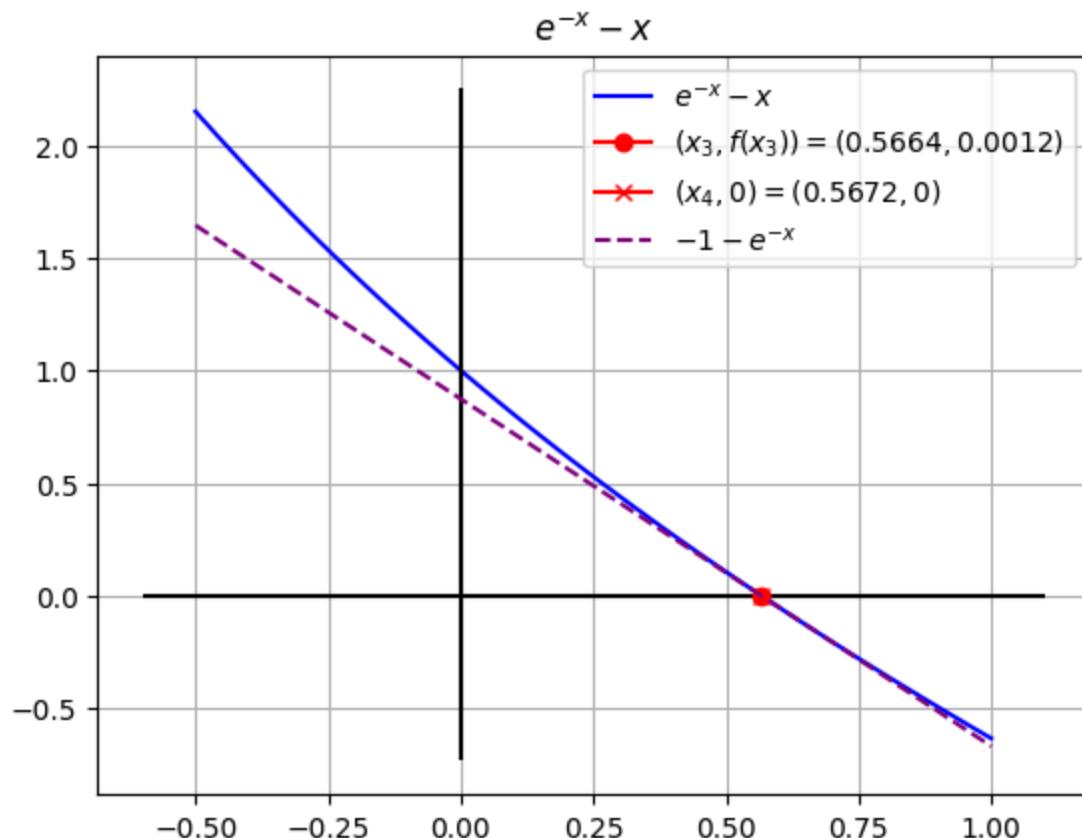
## Actualicemos la tabla de iteraciones
nueva_fila = {'xi':xi,'Xi+1':xs,'f(xi)':fxi,"f'(Xi)":dfxi,'er(%)':round(er,4)}
nueva_fila = pd.DataFrame([nueva_fila])
tabla = pd.concat([tabla, nueva_fila], ignore_index=True)

display(HTML(tabla.head(it).to_html()))

#Actualizamos la iteración y el valor de x actual (xi)
it += 1
xi = xs

print("")
input()
clear_output(wait=True)
```

<Figure size 640x480 with 0 Axes>



	x_i	X_{i+1}	$f(x_i)$	$f'(X_i)$	er(%)
0	0	0.5000	1	-2	100.0
1	0.5000	0.5664	0.1065	-1.605	11.72
2	0.5664	0.5672	0.0012	-1.568	0.1412

```

# @title Grafico dinámico (movie del método)
# Librerias necesarias para realizar el gráfico
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from IPython.display import HTML;
import sympy as sp
rc('animation', html='html5');

x = sp.symbols('x')
y = sp.E**(-x)-x
dy = y.diff()
r = np.linspace(-0.5,1, 100)
fx = [y.subs({x:x_i}) for x_i in r]

fig, ax = plt.subplots()
ax.plot(r,fx,color='blue',label="$e^{-x} - x$")

## Plano cartesiano (Ejes)
ax.vlines(x=0,ymin=min(fx)-0.1,ymax=max(fx)+0.1,color='k')
ax.hlines(y=0,xmin=min(r)-0.1,xmax=max(r)+0.1,color='k')
ax.set_title("$e^{-x} - x$")
ax.grid()

# Se definen los atributos que debe tener la linea o en este caso el punto que se va a dibujar
linea1, = ax.plot([],[],'o',color = 'r', label = '')
linea2, = ax.plot([],[],'x',color = 'r', label = '')
linea3, = ax.plot([],[],'--',color = 'purple', label = '')

frames = len(tabla)

def graficar(i):

    xg = tabla['xi'].to_list()
    yg = tabla['f(xi)'].to_list()

    xs = tabla['Xi+1'].to_list()
    ys = tabla["f'(Xi)"].to_list()

    linea1.set_data(xg[i],yg[i])
    linea1.set_label(f"${x_{i}}, f(x_{i})= ({xg[i]},{yg[i]})$')

    linea1.set_data(xs[i],0)
    linea2.set_label(f"${x_{i}}, 0) = ({xs[i]},0)$')

    ## Pintar la recta tangente
    h=0.1
    dfx = (y.subs({x:xg[i]+h})-y.subs({x:xg[i]}))/h # derivative
    tan = y.subs({x:xg[i]})+dfx*(r-xg[i]) # tangent
    linea3.set_data(r,tan)
    linea3.set_label(f"${sp.latex(dy)}$")

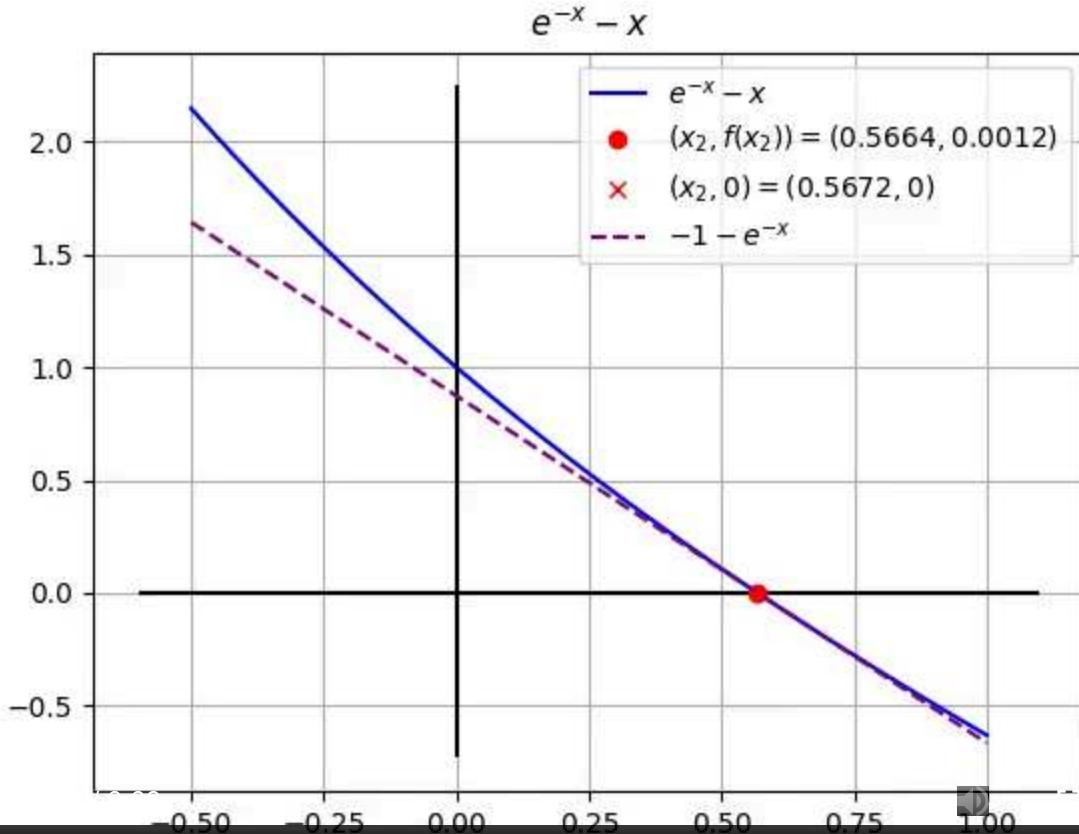
    ax.legend()
    return (linea1,linea2,linea3,)


```

```
plt.close()
```

```
# Ejecutamos la animación para que se genere y quede en loop mostrando su resultado.
anim = animation.FuncAnimation(fig, graficar, frames=frames, interval=1000, repeat=False)
anim
```

```
<ipython-input-150-276efacf074c>:40: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xg[i],yg[i])
<ipython-input-150-276efacf074c>:44: MatplotlibDeprecationWarning: Setting data with a
  linea1.set_data(xs[i],0)
```



Ejercicio:

Calcular la raíz de $f(x) = \sin(x) - x$ en el punto $x_i = 0.75$ con una $tol <= 1\%$

Implementar una función que me permita dinámicamente modificar la función (pedirla por consola), la tolerancia y el punto inicial, además de la gráfica y que retorne la raíz, el error, la tabla de iteraciones y la gráfica dinámica de como itera el método.

TIP: Indagar sobre los formularios con entradas en COLAB.

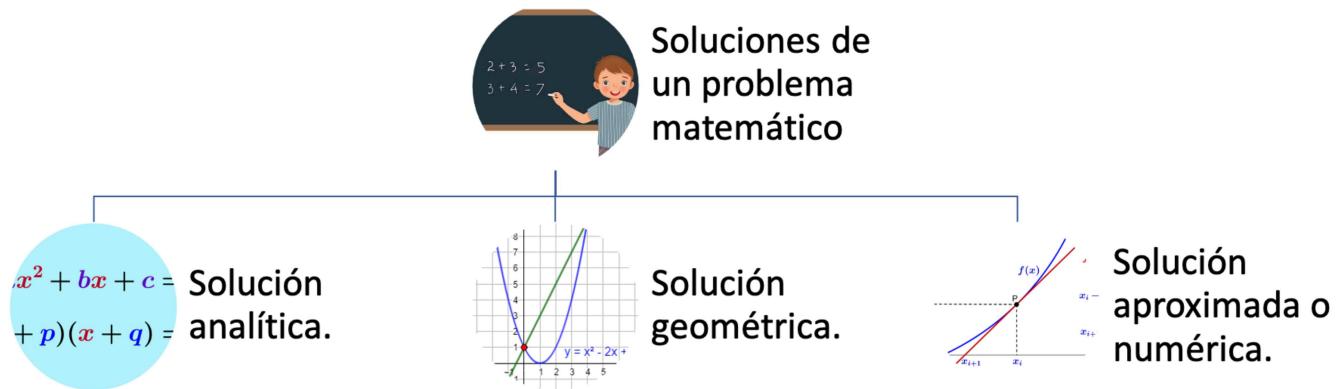
NOTA: Modificar el código del método para que se pase por consola la ecuación y se realice el método con esta entrada, mostrando la ecuación en el título y en los legend de forma dinámica.

Métodos numéricos - Cálculo de raíces

Contenido

- Solución de un problema matemático.
- Raíces de funciones

Solución de un problema matemático.



Raíces de funciones

Una raíz se define como una función igualada a cero o el intercepto de una curva con el eje X

$$f(x) = 0$$



💡 Encontremos las raíces de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

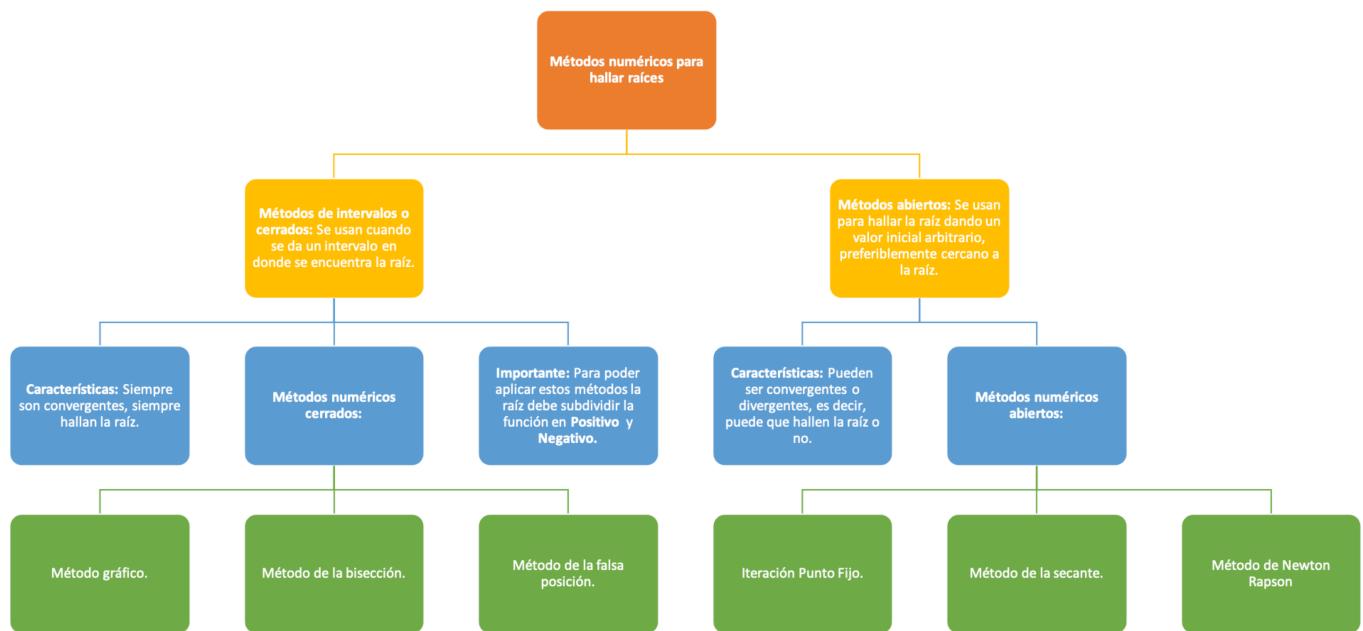
$$f(x) = x^2 + 5x + 6$$

💡 Encontremos las raíces de la siguiente función...

Analiticamente ¿cuál es la solución?, ¿qué multiplicidad tiene la función?

$$f(x) = x^2 + 2x + 1$$

Métodos numéricos para hallar raíces



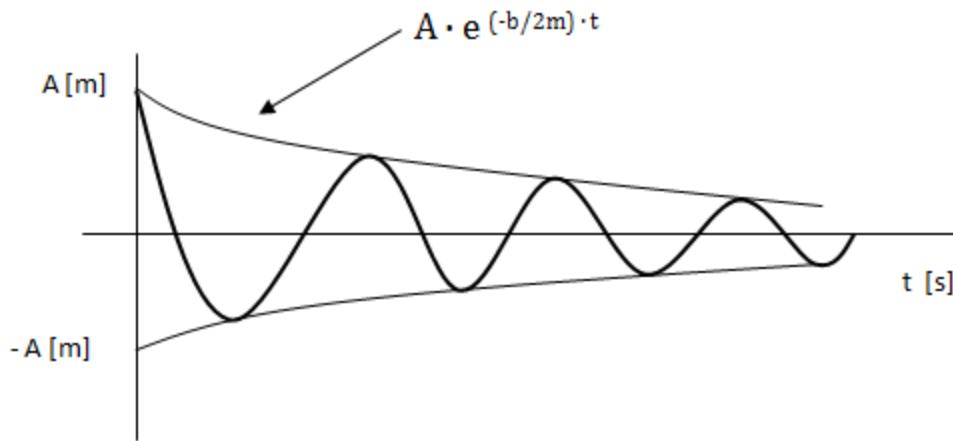
Método gráfico:

El método gráfico sirve como método inicial para conocer dónde están las raíces de forma aproximada.

Nota

Hallar una raíz geométricamente consiste en hallar el punto de cruce con el eje X

Encontremos las raíces de la siguiente función...



Este ejemplo tiene 8 raíces de multiplicidad 1

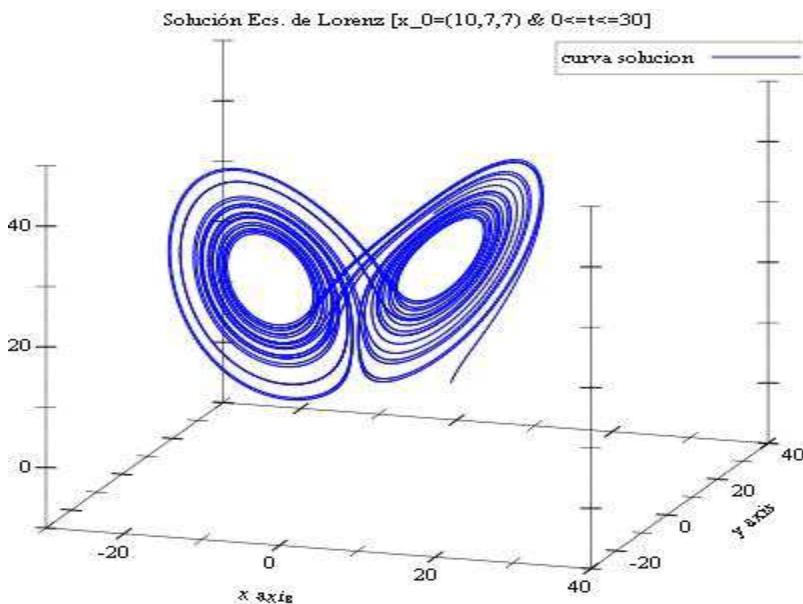
Métodos Numéricos: Ecuaciones Diferenciales Ordinarias

Contenido

- Métodos para solución de ecuaciones diferenciales.
- Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

Métodos para solución de ecuaciones diferenciales.

Las ecuaciones diferenciales son una parte fundamental de las matemáticas aplicadas y la ingeniería, proporcionando una poderosa herramienta para modelar y entender fenómenos que varían con el tiempo o el espacio. Desde el movimiento de los planetas hasta el flujo de corriente en circuitos eléctricos, las ecuaciones diferenciales juegan un papel crucial en describir cómo cambian las cosas en el mundo real.



Sin embargo, la mayoría de las ecuaciones diferenciales no pueden resolverse con métodos analíticos, especialmente cuando se trata de modelos complejos o sistemas no lineales. Aquí es donde los métodos numéricos entran en juego. Estos métodos proporcionan aproximaciones prácticas para resolver ecuaciones diferenciales, permitiendo a los científicos e ingenieros obtener soluciones útiles donde las técnicas analíticas fallan o son demasiado complicadas para aplicarse.

Aplicaciones de los Métodos Numéricos en Ecuaciones Diferenciales

- 1. Ingeniería y Física:** En el diseño de sistemas mecánicos, electrónicos y de control, las ecuaciones diferenciales modelan el comportamiento de componentes y sistemas. Por ejemplo, en la ingeniería eléctrica, los circuitos RLC, que combinan resistencias, inductancias y capacitancias, se analizan mediante ecuaciones diferenciales.
- 2. Meteorología y Modelado Climático:** Los modelos climáticos y meteorológicos utilizan ecuaciones diferenciales para predecir el clima y entender los patrones climáticos. Estos modelos son intrincadamente complejos y requieren el uso de métodos numéricos para su resolución.
- 3. Biología y Medicina:** En la farmacocinética, las ecuaciones diferenciales describen cómo los medicamentos se distribuyen y metabolizan en el cuerpo. También son fundamentales en la modelización de la dinámica de las poblaciones y la propagación de enfermedades.
- 4. Economía y Finanzas:** En la modelización financiera, las ecuaciones diferenciales se utilizan para entender la dinámica de los mercados y para la valoración de opciones y otros instrumentos financieros.
- 5. Ciencias de la Computación y la Inteligencia Artificial:** Los algoritmos de aprendizaje automático y redes neuronales a menudo implican resolver ecuaciones diferenciales para optimizar y entrenar modelos.

Unidad 1: Clusters en computación paralela/distribuida

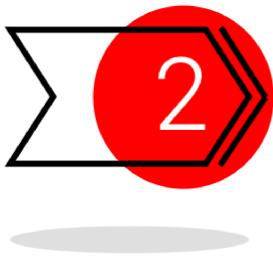
Contenido

- Contenido de la unidad
- Introducción al HPC
- ¿Dónde corre lo que se va a programar en el curso?
- CPU & GPU & TPU
- CPU & GPU
- Actualidad del software para Big Data
- Aplicaciones del HPC

Contenido de la unidad



Introducción a
entornos
HPC/HTC



Colas y
calendarizadores



Computación
Cloud



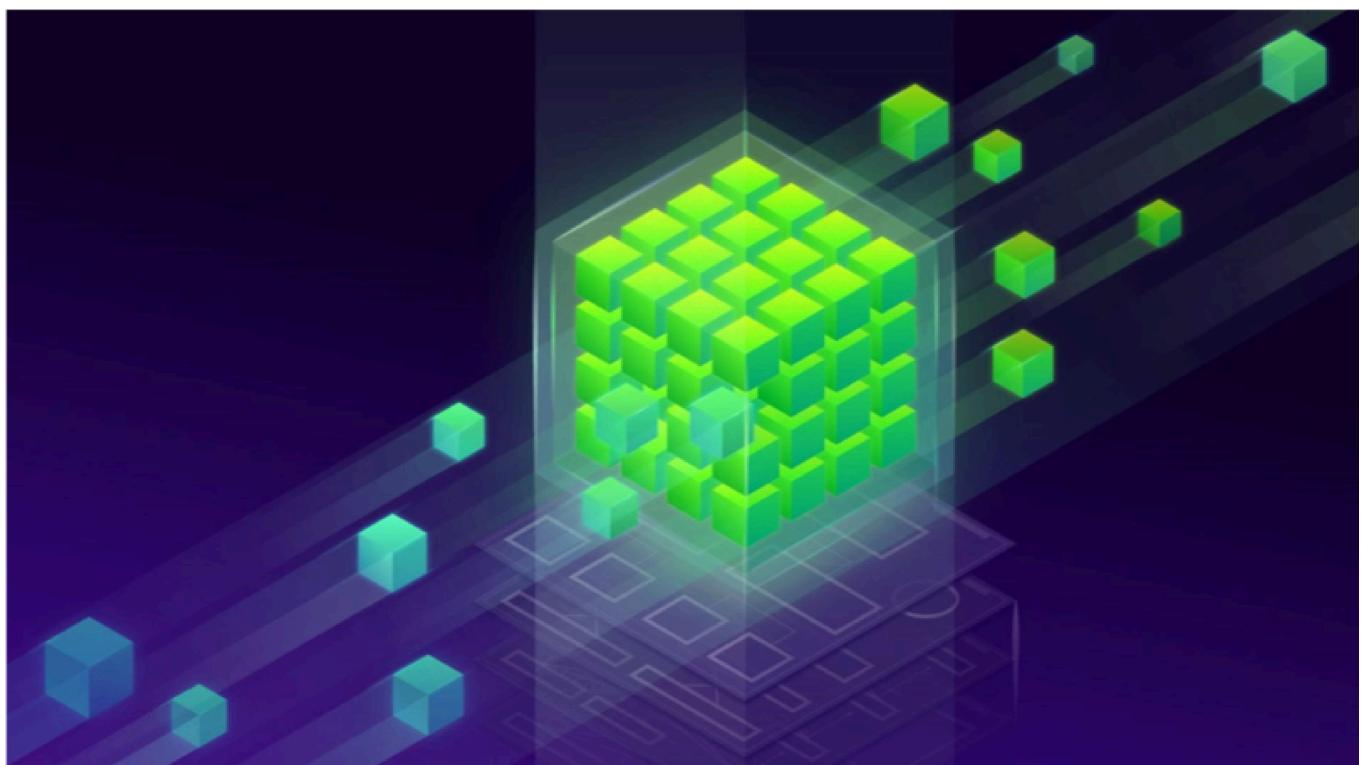
Repaso de
Python y C para
data science

Introducción al HPC

HPC: Cualquier técnica computacional que soluciona un problema grande de forma más rápida que usando posiblemente sistemas simples.

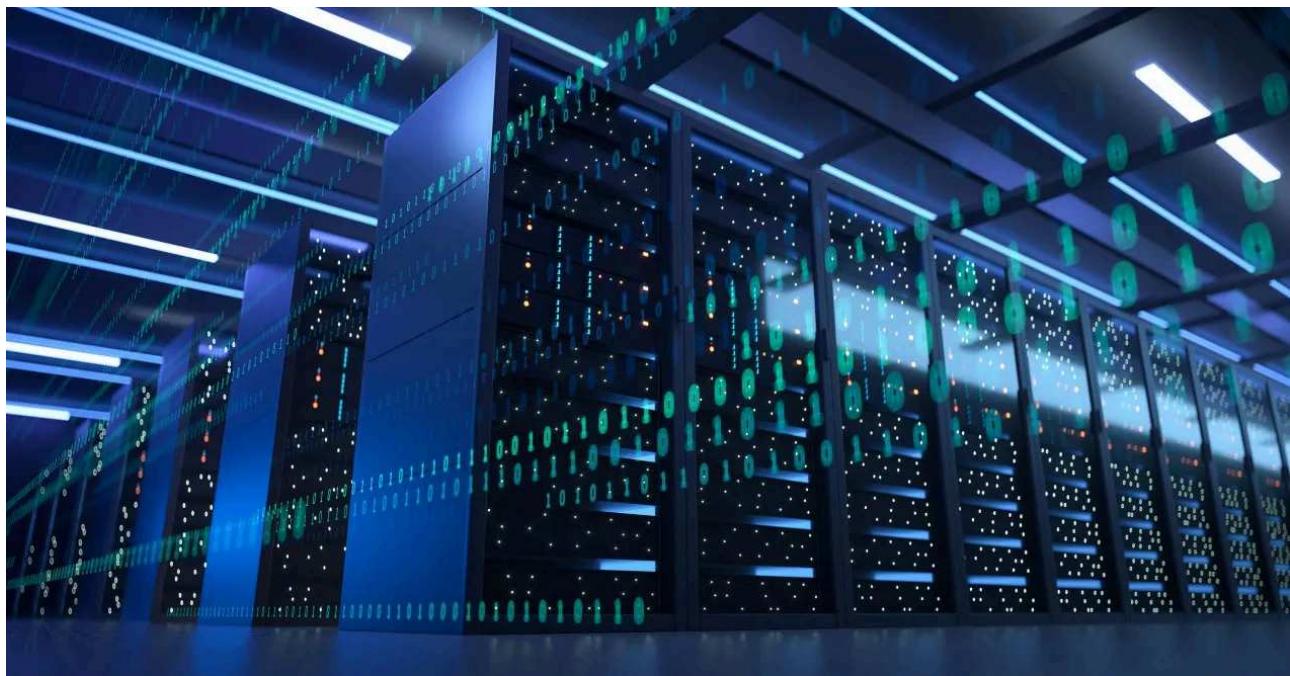
HPC ha tenido gran impacto sobre todas las áreas de ciencias computacionales e ingeniería en la academia, gobierno e industria.

Muchos problemas han sido solucionados con técnicas de **HPC** que eran imposibles de solucionar con estaciones de trabajo individuales o computadores personales.



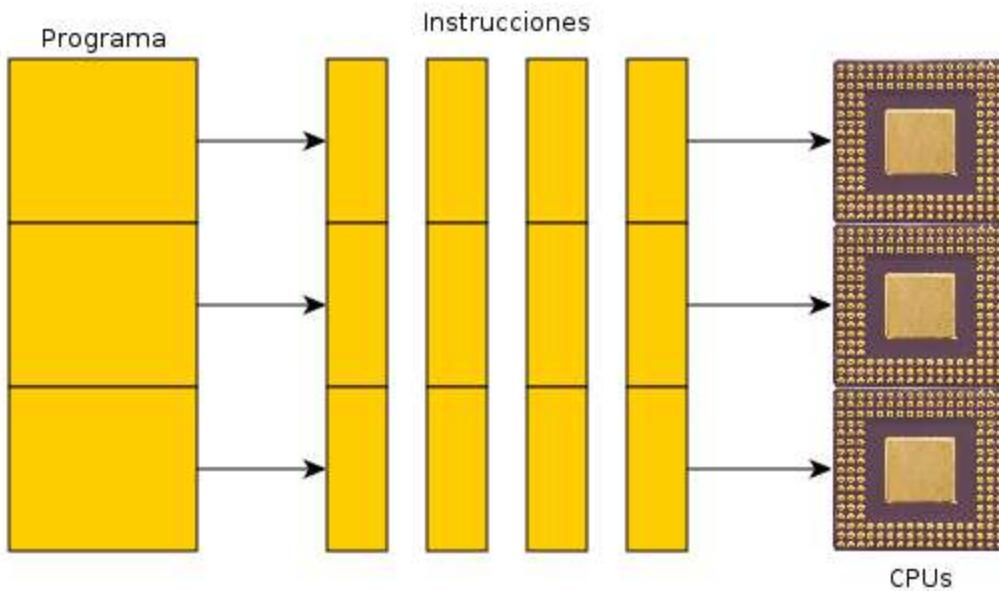
Procesadores de alto rendimiento





Computación paralela

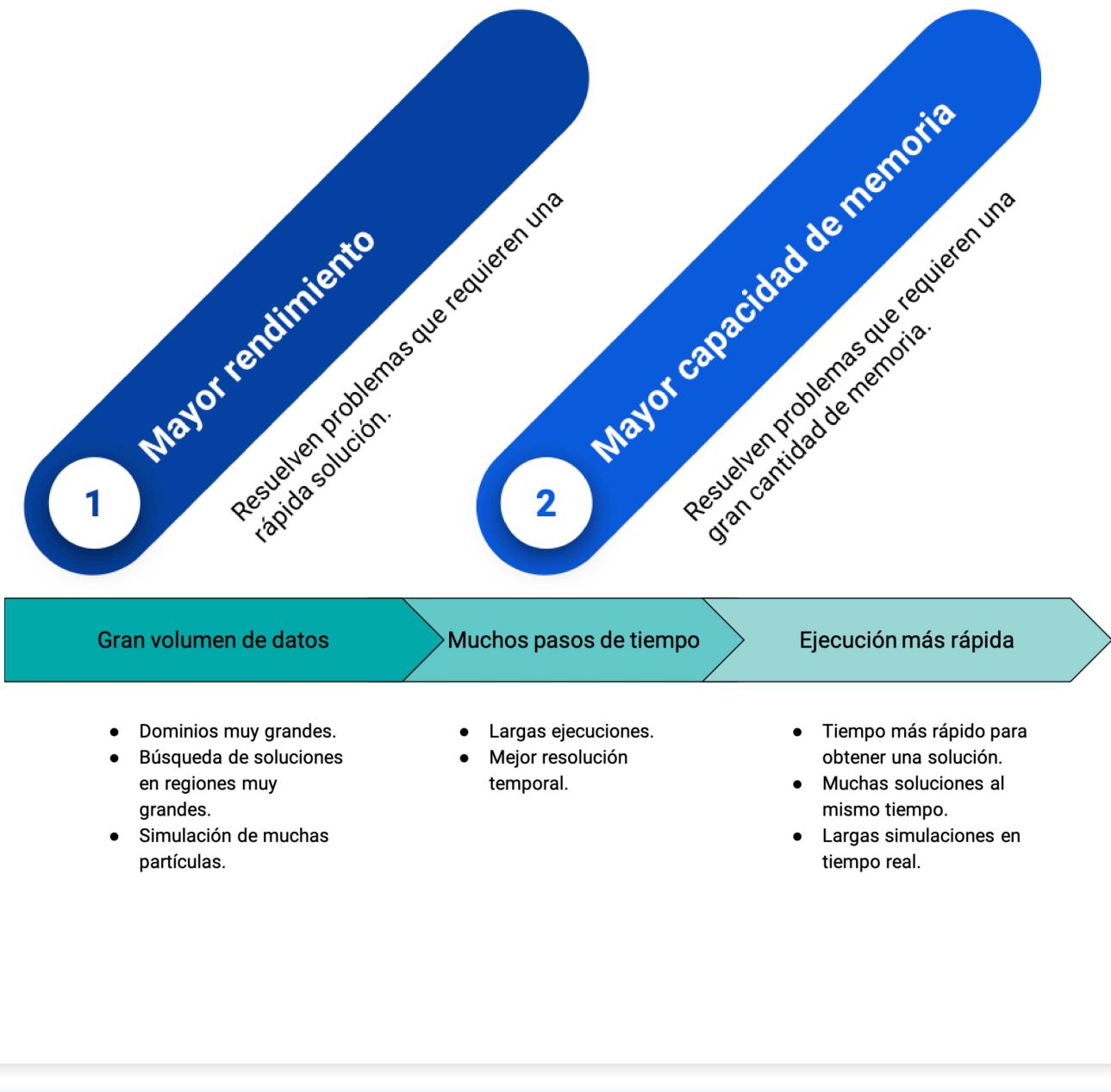
Sistemas simples con varios procesadores trabajando sobre el mismo problema



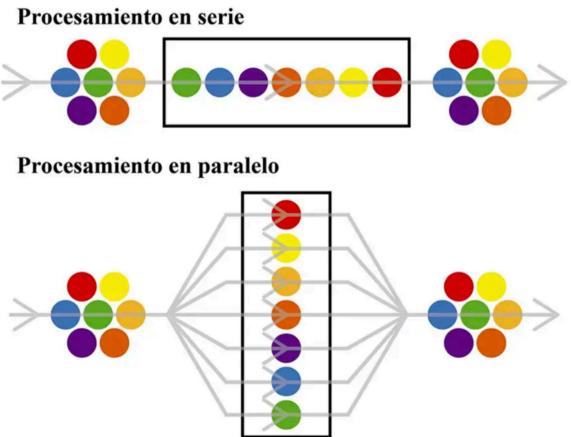
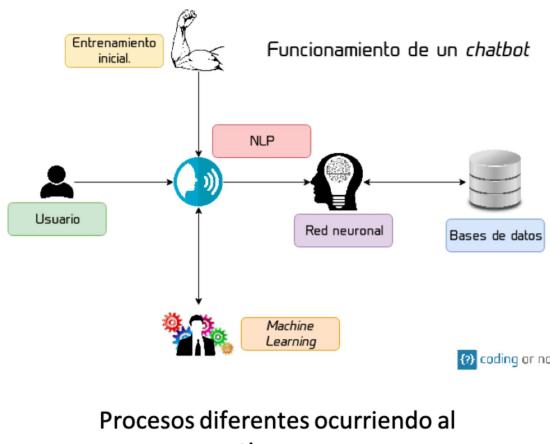
Computación paralela Vs Computador paralelo

- Computación Paralela: el uso de múltiples computadores o procesadores trabajando en conjunto sobre una tarea común
- Computador Paralelo: un computador que contiene múltiples procesadores:

- Cada procesador trabaja sobre una sección del problema
- Los procesadores permiten intercambio de información con otros procesadores



Computación/programación Concurrente



Procesos diferentes ocurriendo al tiempo

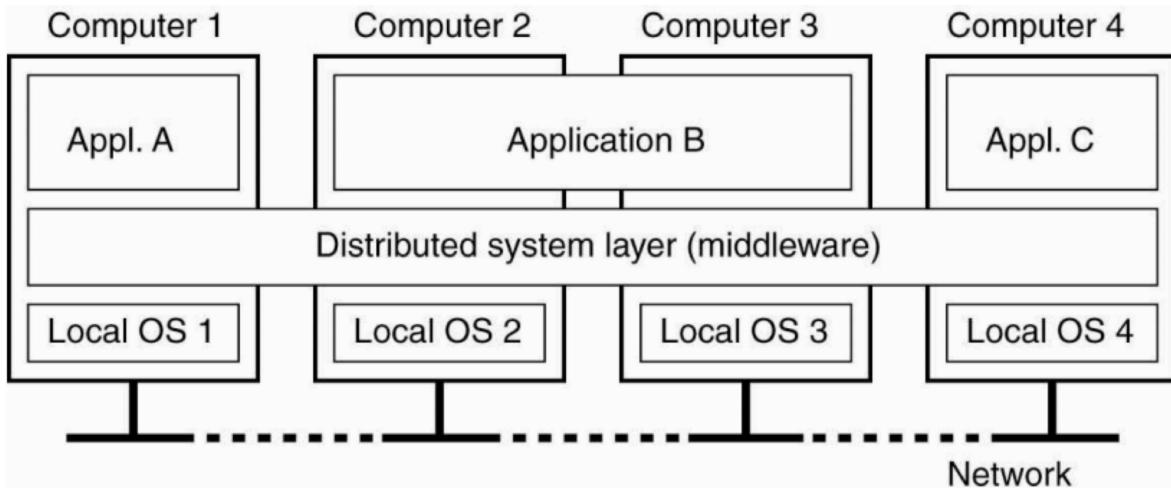
La programación concurrente se enfoca en manejar múltiples tareas al mismo tiempo (ya sea simultáneamente o dando la ilusión de simultaneidad), la programación paralela se enfoca en la división y ejecución simultánea de tareas para resolver un problema más grande en menos tiempo.

Computación Distribuida

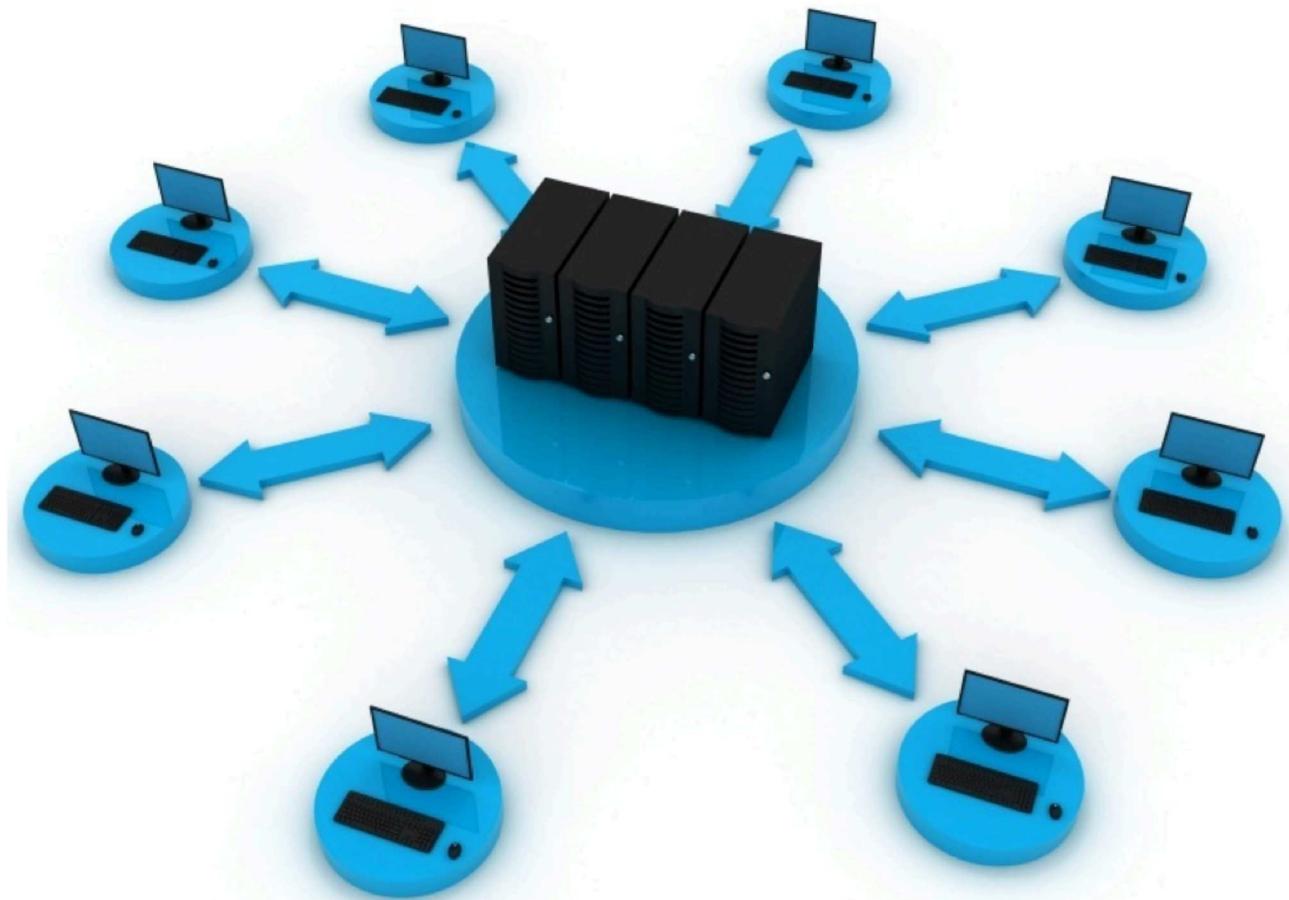
Varios sistemas acoplados por un secuenciador de trabajo sobre problemas relacionados



Sistemas distribuidos



Computación grid



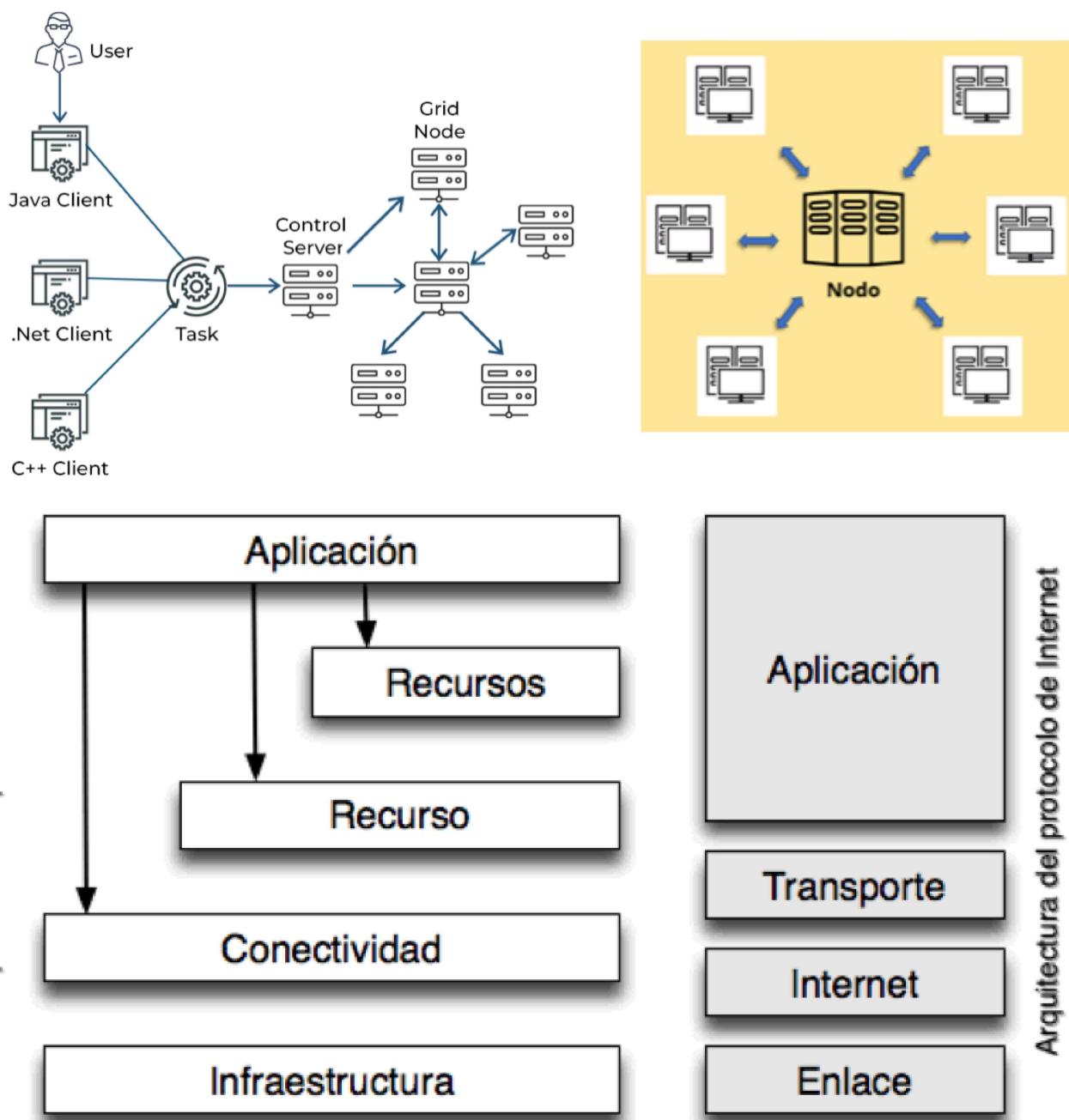
Varios sistemas acoplados por software y redes para trabajar en conjunto en problemas simples o en problemas relacionados.

Unión de Clusters de computadores, recursos computacionales, datos, grupos de investigación, científicos, etc., distribuidos geográficamente y conectados mediante redes WAN

Funcionamiento de la computación grid

Middleware para comunicación transparente y explotación de recursos.

- El objetivo final es usar recursos remotos
- Se requieren conexiones de redes rápidas
- El grid busca el uso eficiente de los recursos
- Es esencial la seguridad centrada en: políticas de acceso, autenticación y autorización.
- Es importante estandarizar las aplicaciones grid.



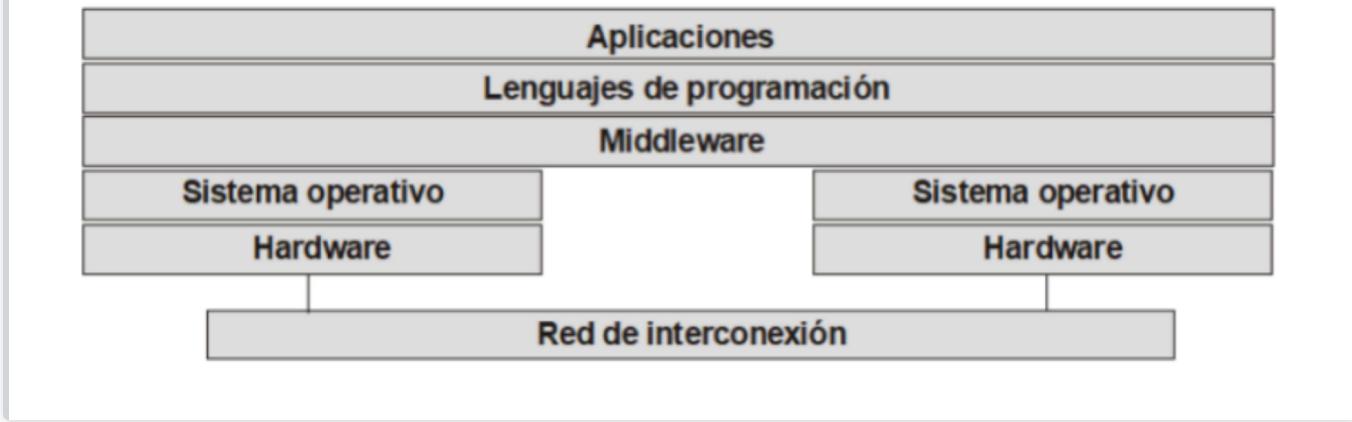
Arquitectura Grid

- Capa de aplicación
- Capa de middleware
- Capa de recursos
- Capa de red

Middleware

El auténtico cerebro del grid, se encarga de las siguientes funciones:

- Encontrar lugar conveniente para ejecutar la tarea solicitada por el usuario.
- Optimizar el uso de recursos que se encuentren dispersos.
- Organizar el acceso eficiente a los datos.
- Autenticar los diferentes elementos.
- Ejecutar tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar automáticamente la recuperación frente a fallos.
- Notificar el fallo, culminación de la ejecución de una tarea y sus resultados.

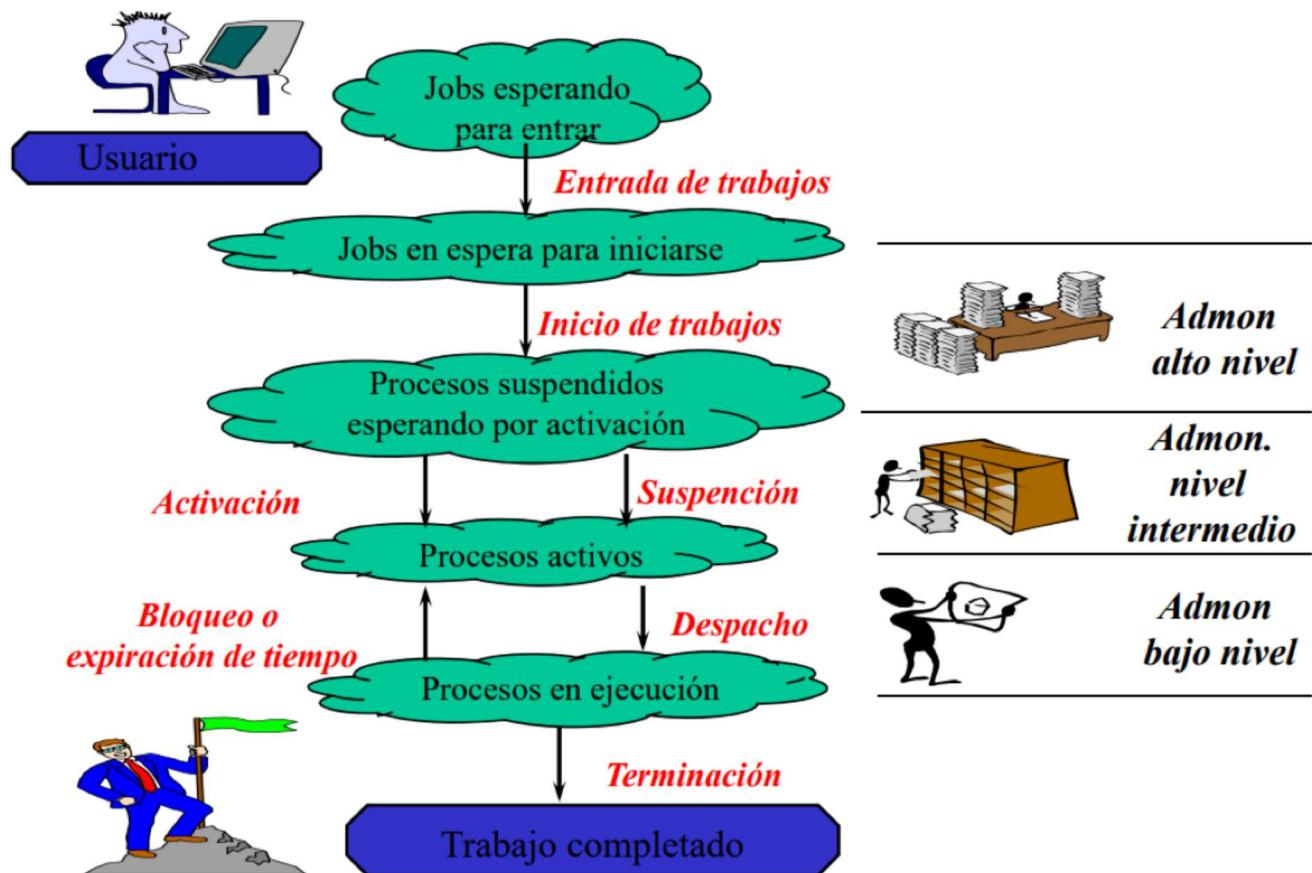


¿Dónde corre lo que se va a programar en el curso?

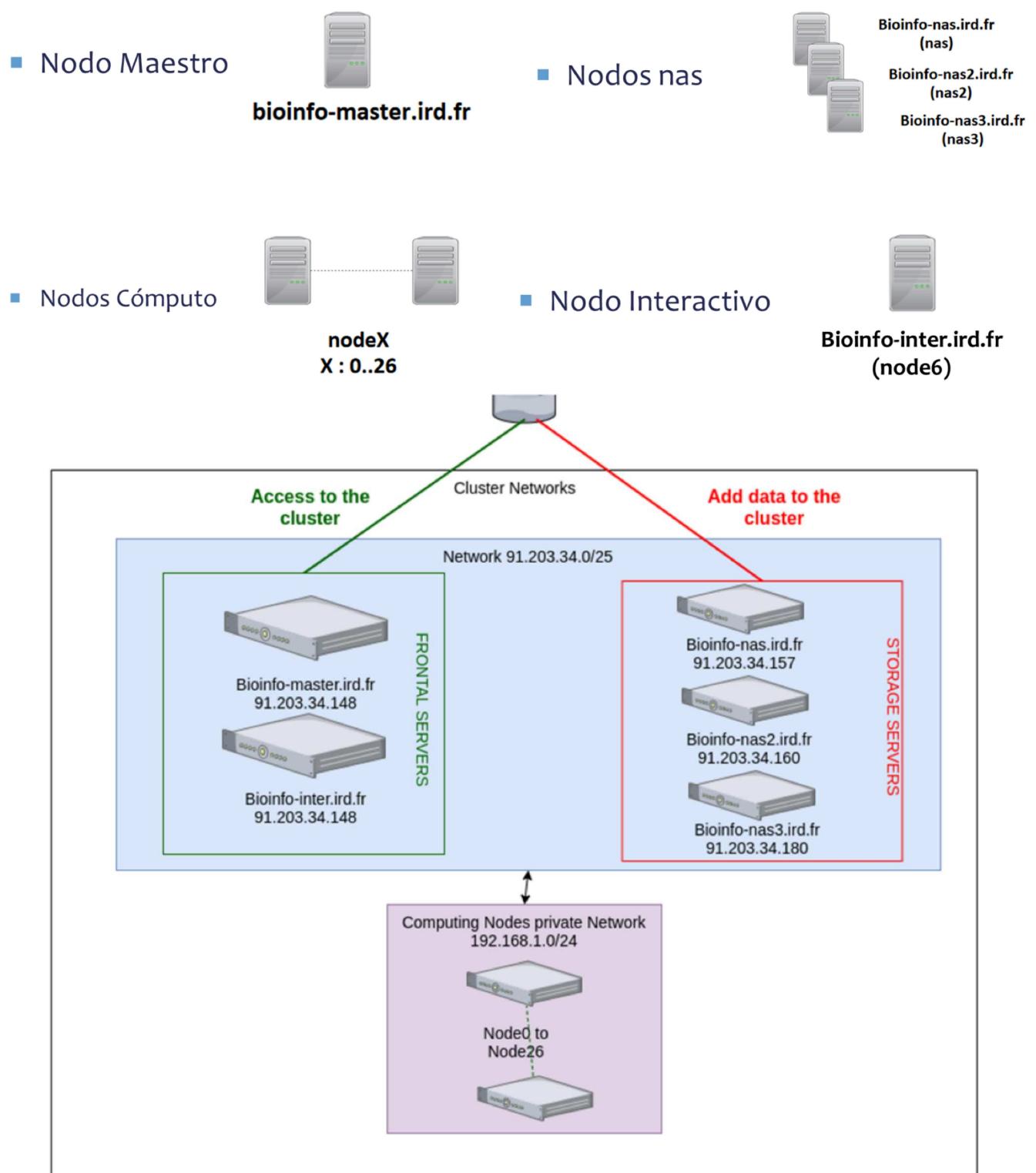
Infraestructura: Cluster



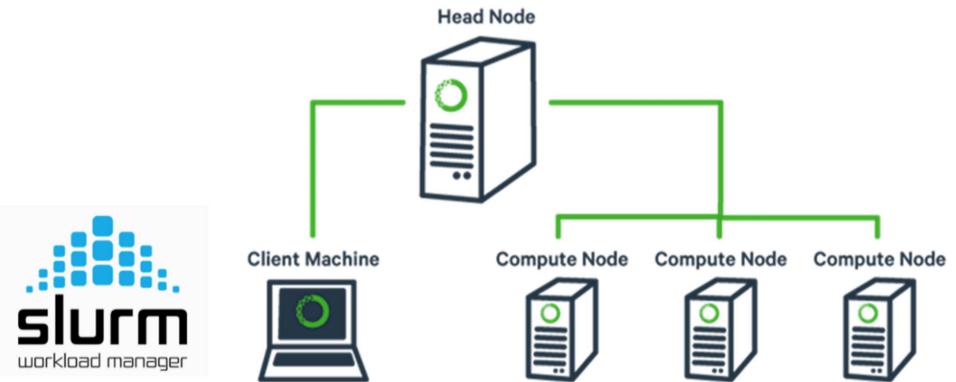
Colas y calendarizadores



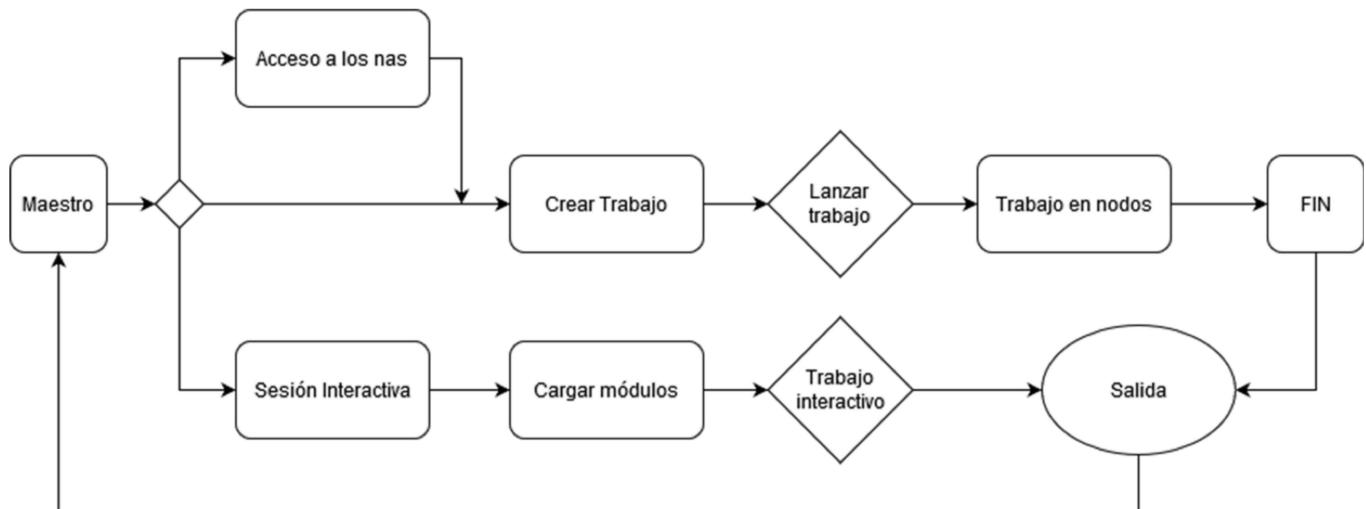
- Administrar recursos del Clúster
- Priorizar trabajos
- Limitar capacidad de cómputo
- Administrar usuarios



Algunos calendarizadores



Flujo de trabajo con calendarizadores

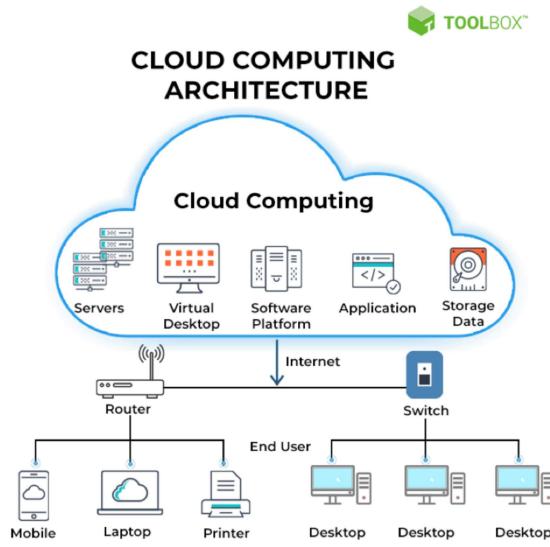


Así se ve un job para calendarizador

```
#!/bin/bash
## Define the job name
#SBATCH --job-name=test
## Define the output file
#SBATCH --output=res.txt
## Define the number of tasks
#SBATCH --ntasks=1
```

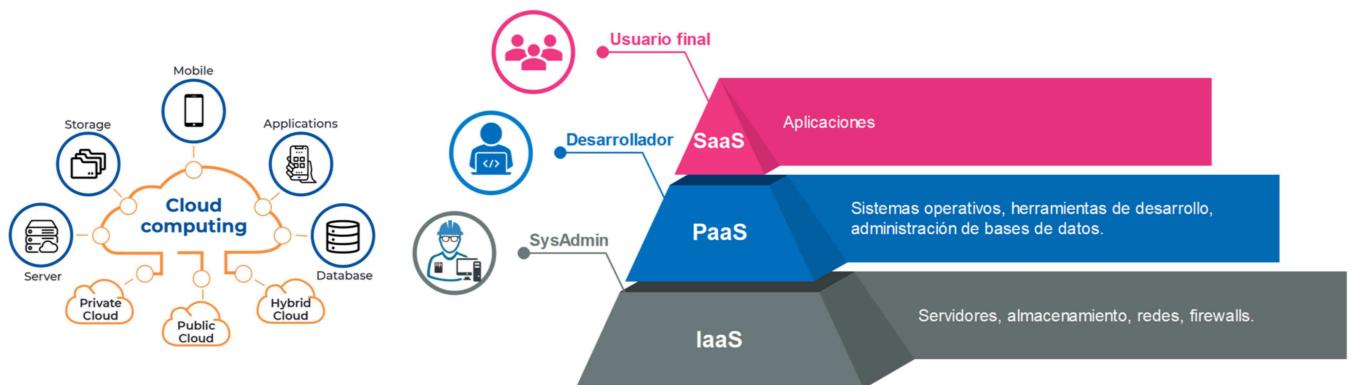


Infraestructura: Cloud

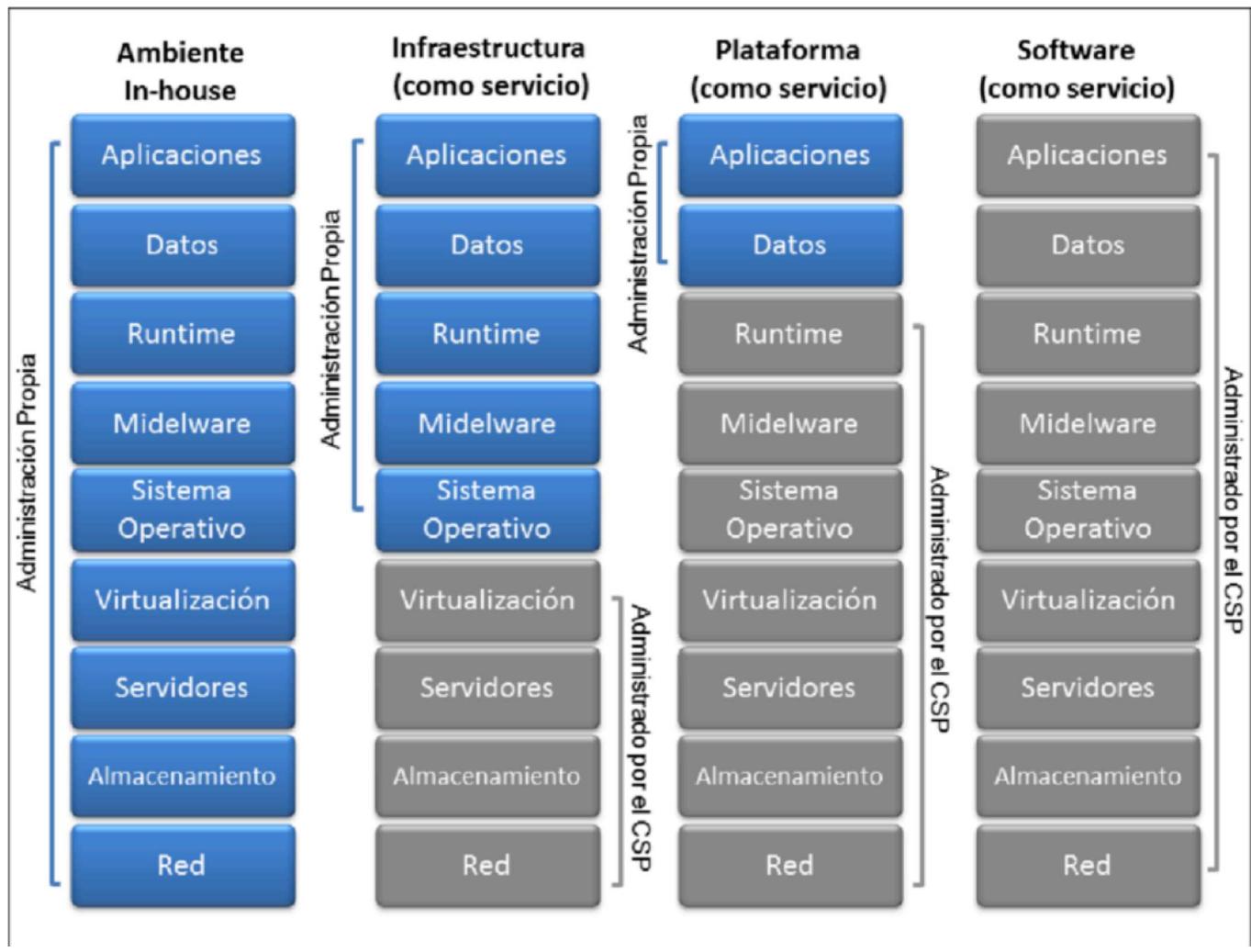


- Externalizar TI a proveedores de servicio
- Presentar transparencia al usuario final
- Asignación dinámica de recursos bajo demanda (Sin compartir) – Pago asociado
- Virtualización

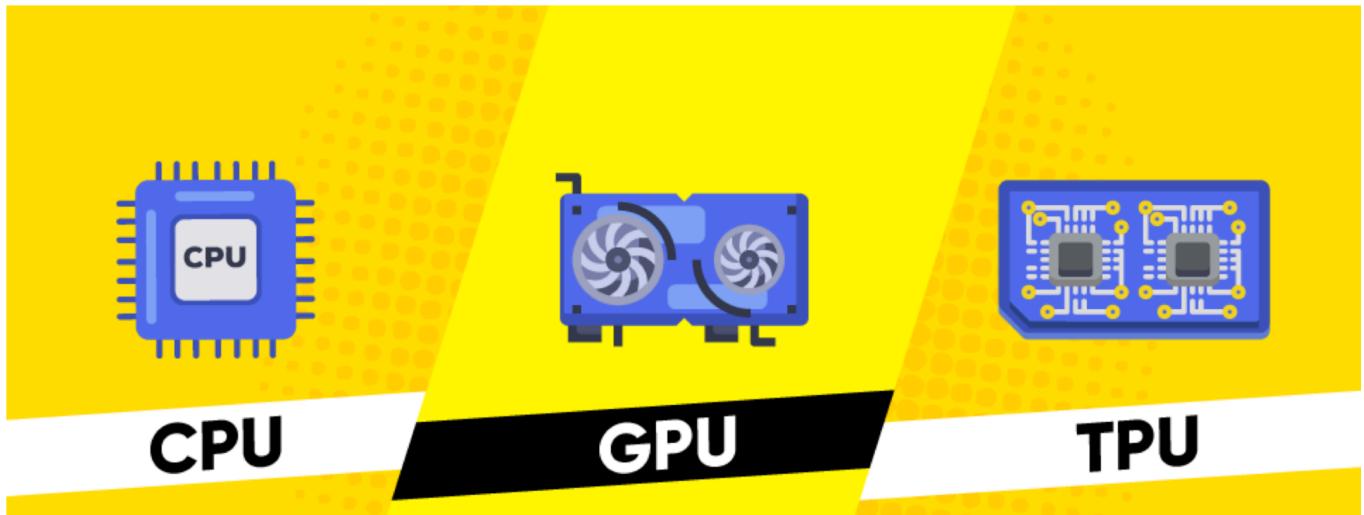
Computación cloud

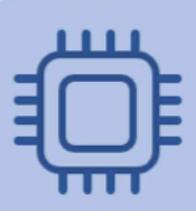


Infraestructura tradicional Vs Cloud



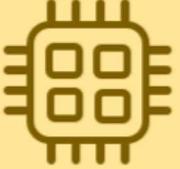
CPU & GPU & TPU





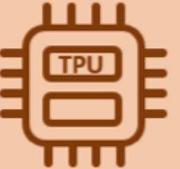
CPU

- Small models
- Small datasets
- Useful for design space exploration



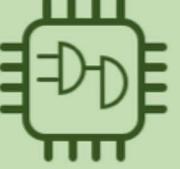
GPU

- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL



TPU

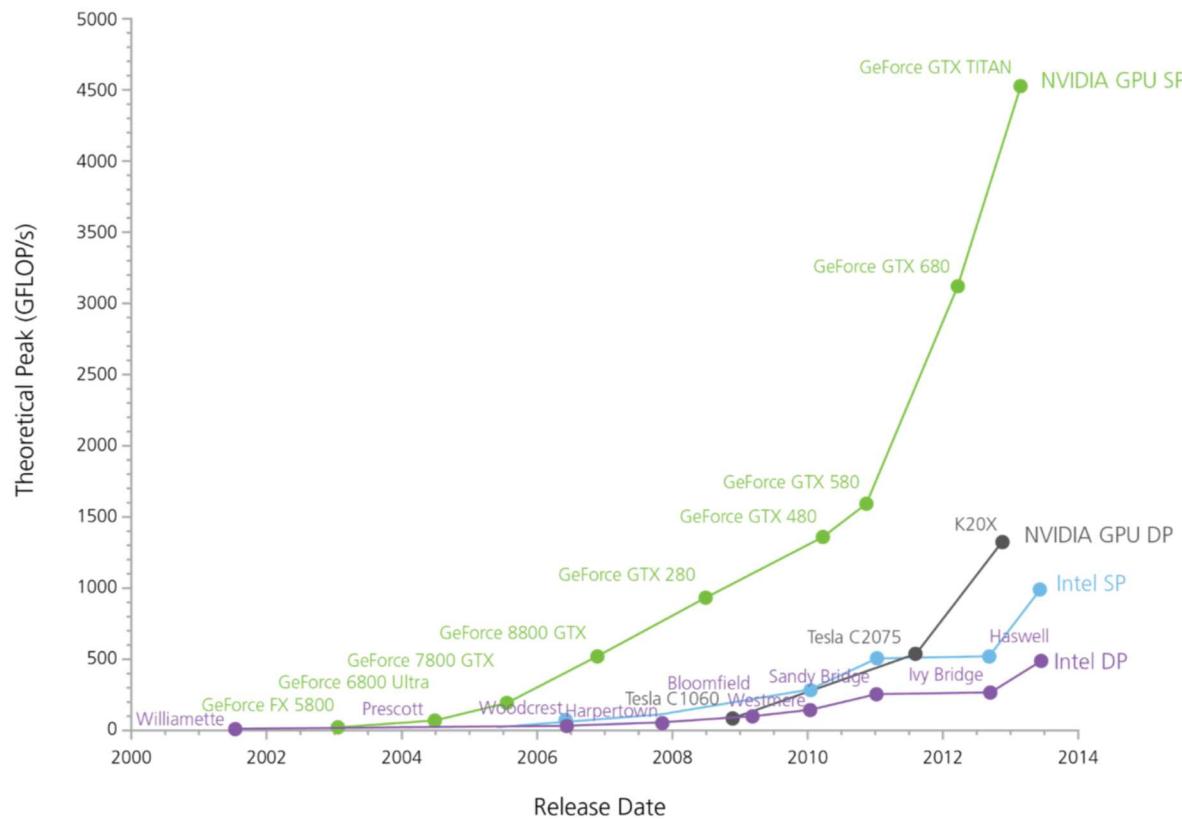
- Matrix computations
- Dense vector processing
- No custom TensorFlow operations



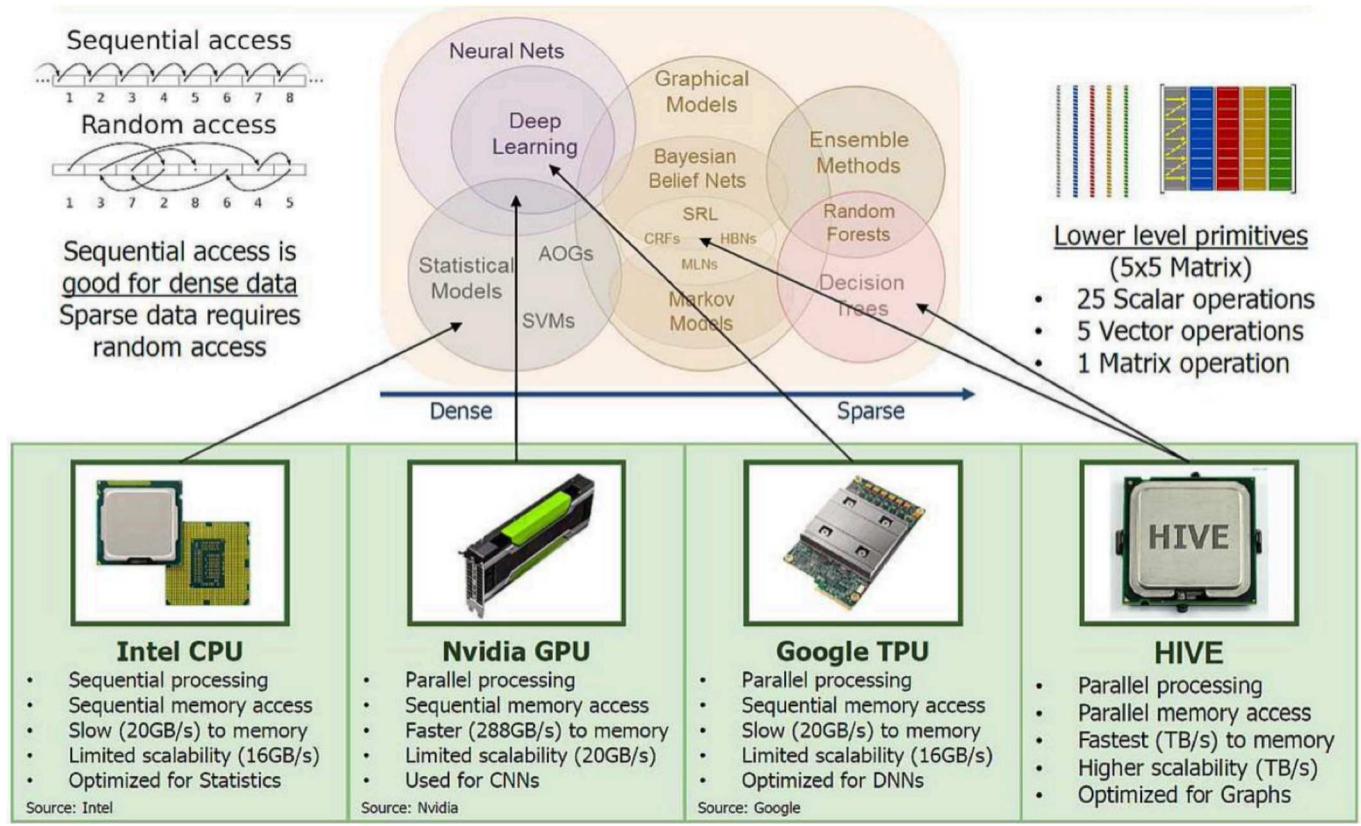
FPGA

- Large datasets, models
- Compute intensive applications
- High performance, high perf./cost ratio

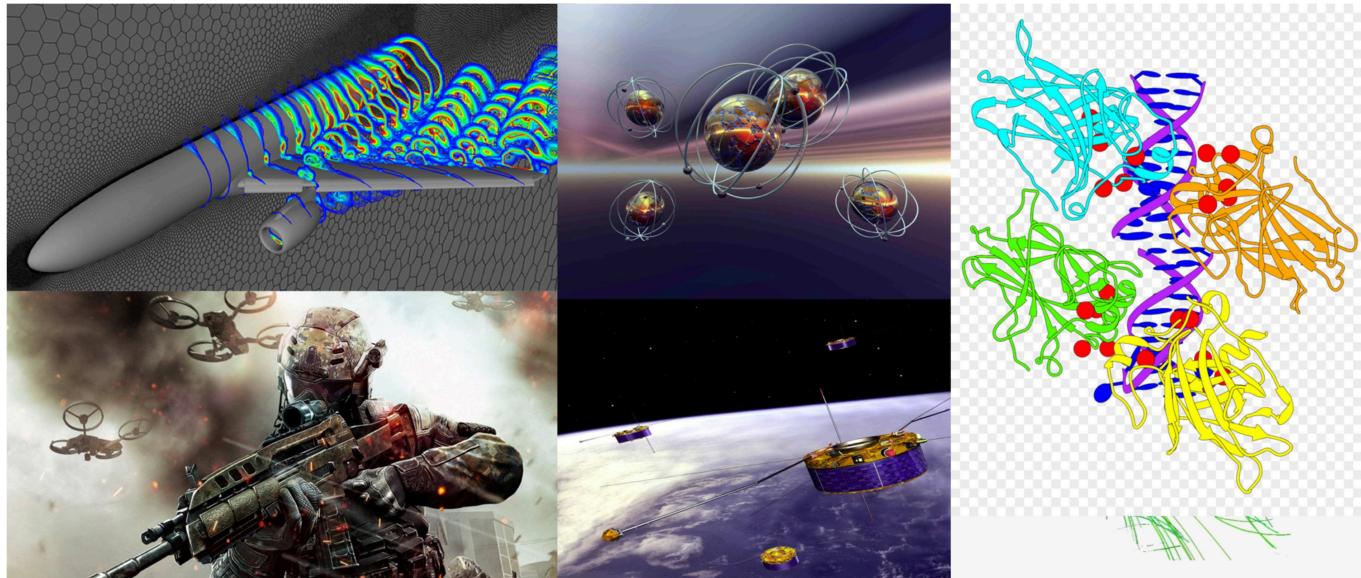
CPU & GPU



Actualidad del software para Big Data



Aplicaciones del HPC

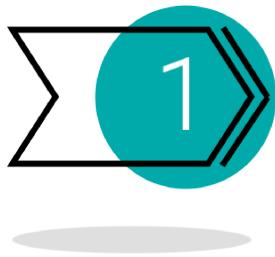


Unidad 1: Clusters en computación paralela/distribuida

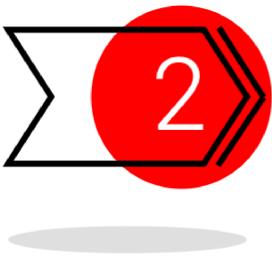
Contenido

- Contenido de la unidad
- Introducción al HPC
- ¿Dónde corre lo que se va a programar en el curso?
- CPU & GPU & TPU
- CPU & GPU
- Actualidad del software para Big Data
- Aplicaciones del HPC

Contenido de la unidad



Introducción a
entornos
HPC/HTC



Colas y
calendarizadores



Computación
Cloud



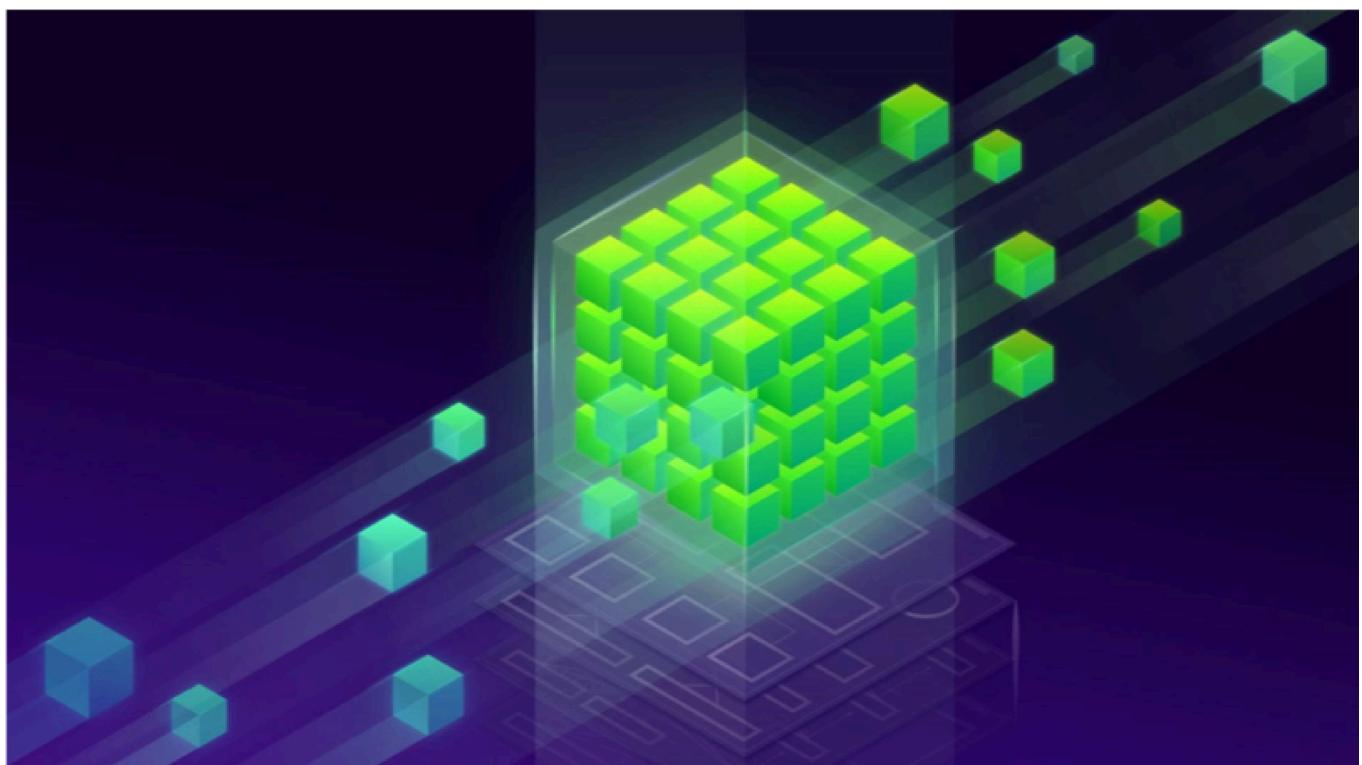
Repaso de
Python y C para
data science

Introducción al HPC

HPC: Cualquier técnica computacional que soluciona un problema grande de forma más rápida que usando posiblemente sistemas simples.

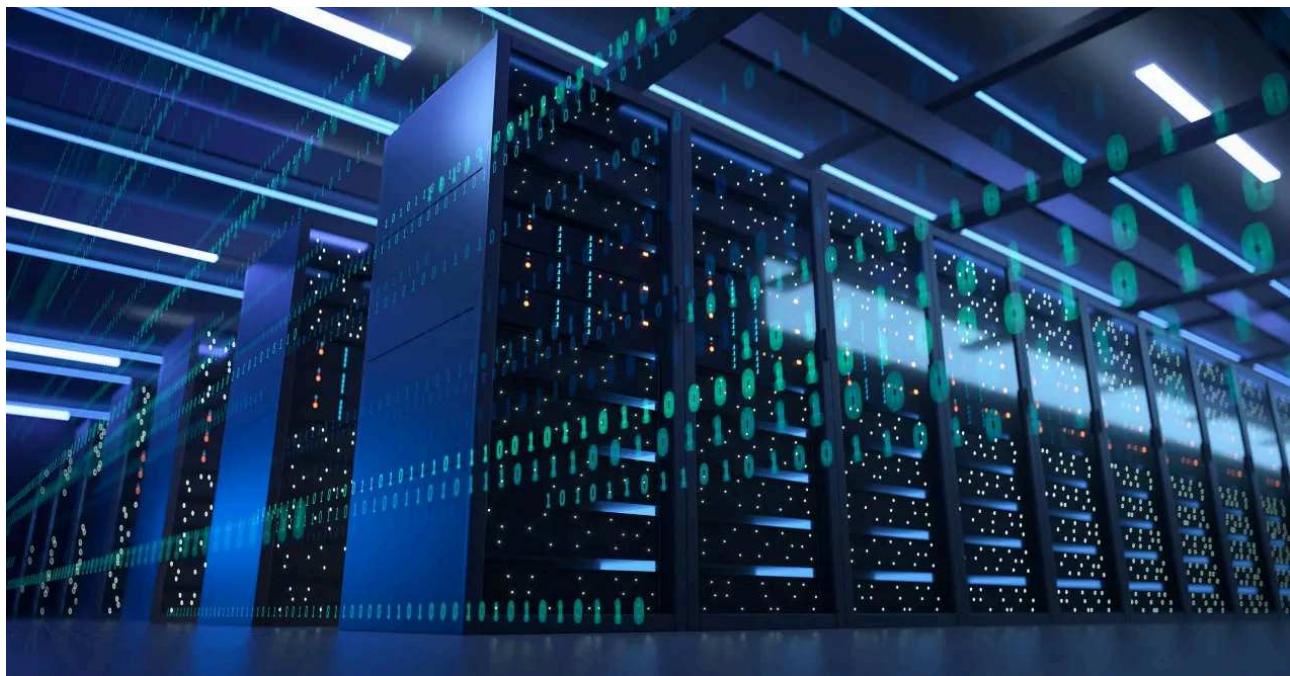
HPC ha tenido gran impacto sobre todas las áreas de ciencias computacionales e ingeniería en la academia, gobierno e industria.

Muchos problemas han sido solucionados con técnicas de **HPC** que eran imposibles de solucionar con estaciones de trabajo individuales o computadores personales.



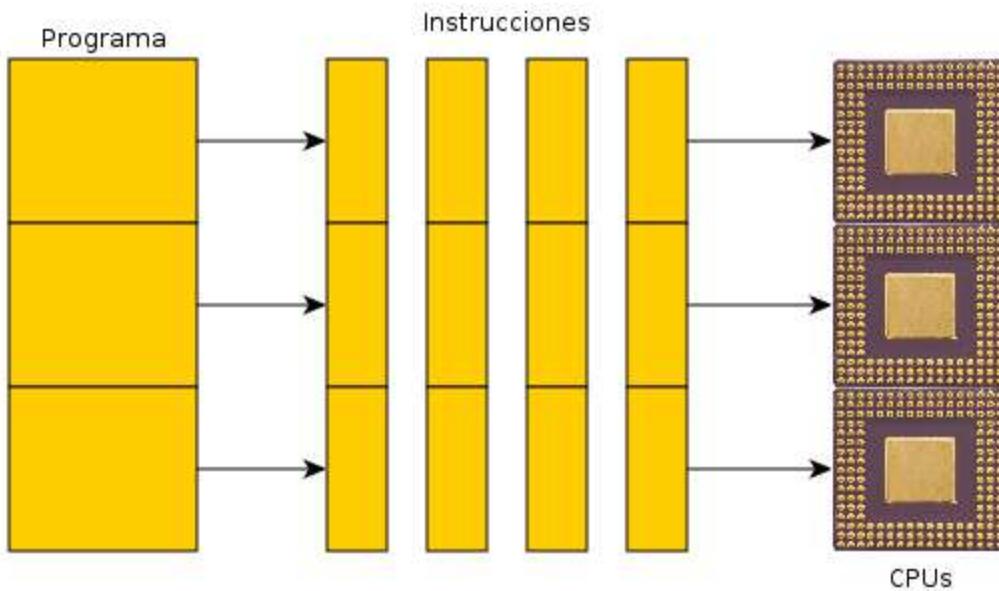
Procesadores de alto rendimiento





Computación paralela

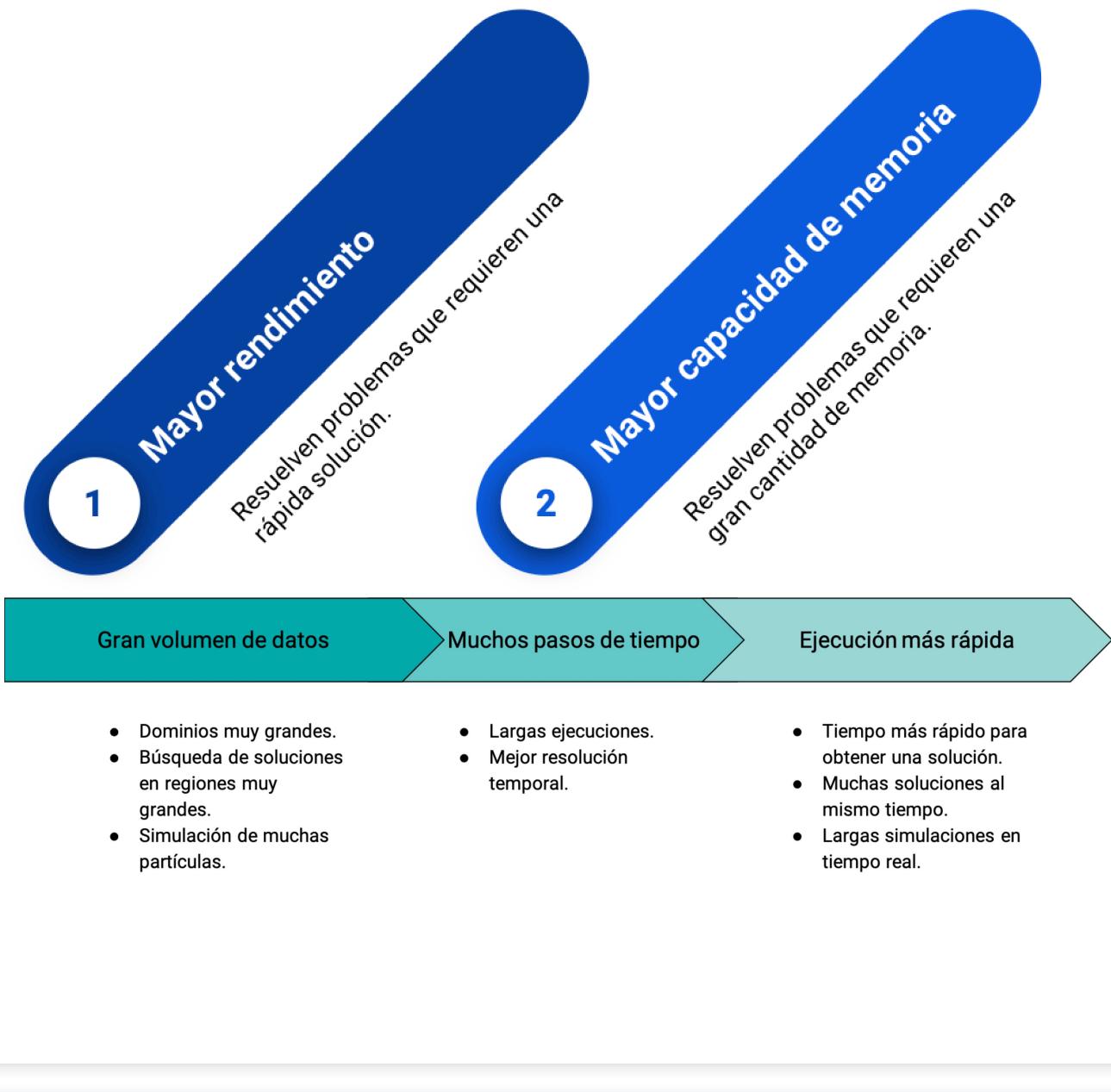
Sistemas simples con varios procesadores trabajando sobre el mismo problema



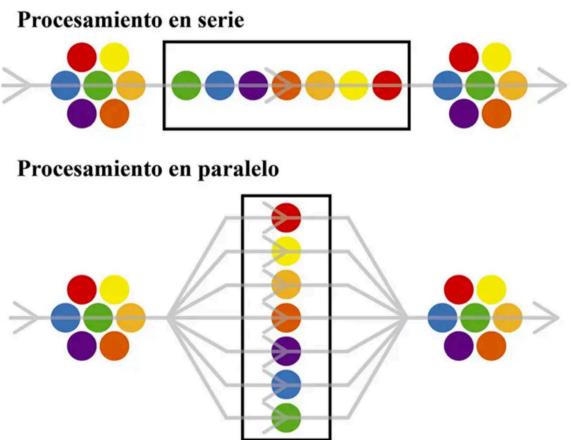
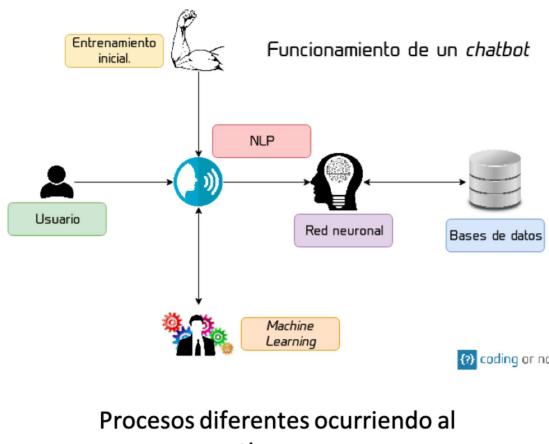
Computación paralela Vs Computador paralelo

- Computación Paralela: el uso de múltiples computadores o procesadores trabajando en conjunto sobre una tarea común
- Computador Paralelo: un computador que contiene múltiples procesadores:

- Cada procesador trabaja sobre una sección del problema
- Los procesadores permiten intercambio de información con otros procesadores



Computación/programación Concurrente



Procesos diferentes ocurriendo al tiempo

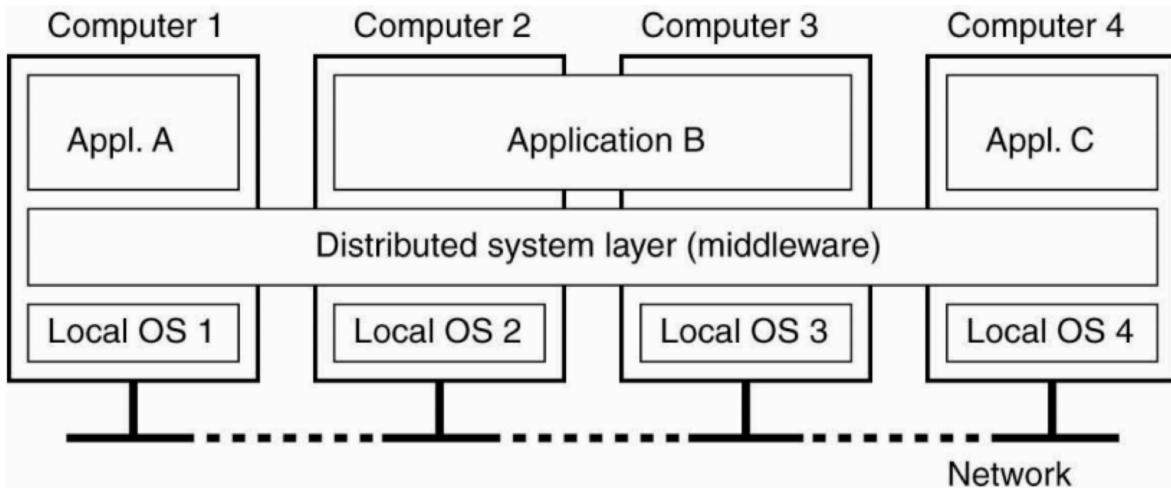
La programación concurrente se enfoca en manejar múltiples tareas al mismo tiempo (ya sea simultáneamente o dando la ilusión de simultaneidad), la programación paralela se enfoca en la división y ejecución simultánea de tareas para resolver un problema más grande en menos tiempo.

Computación Distribuida

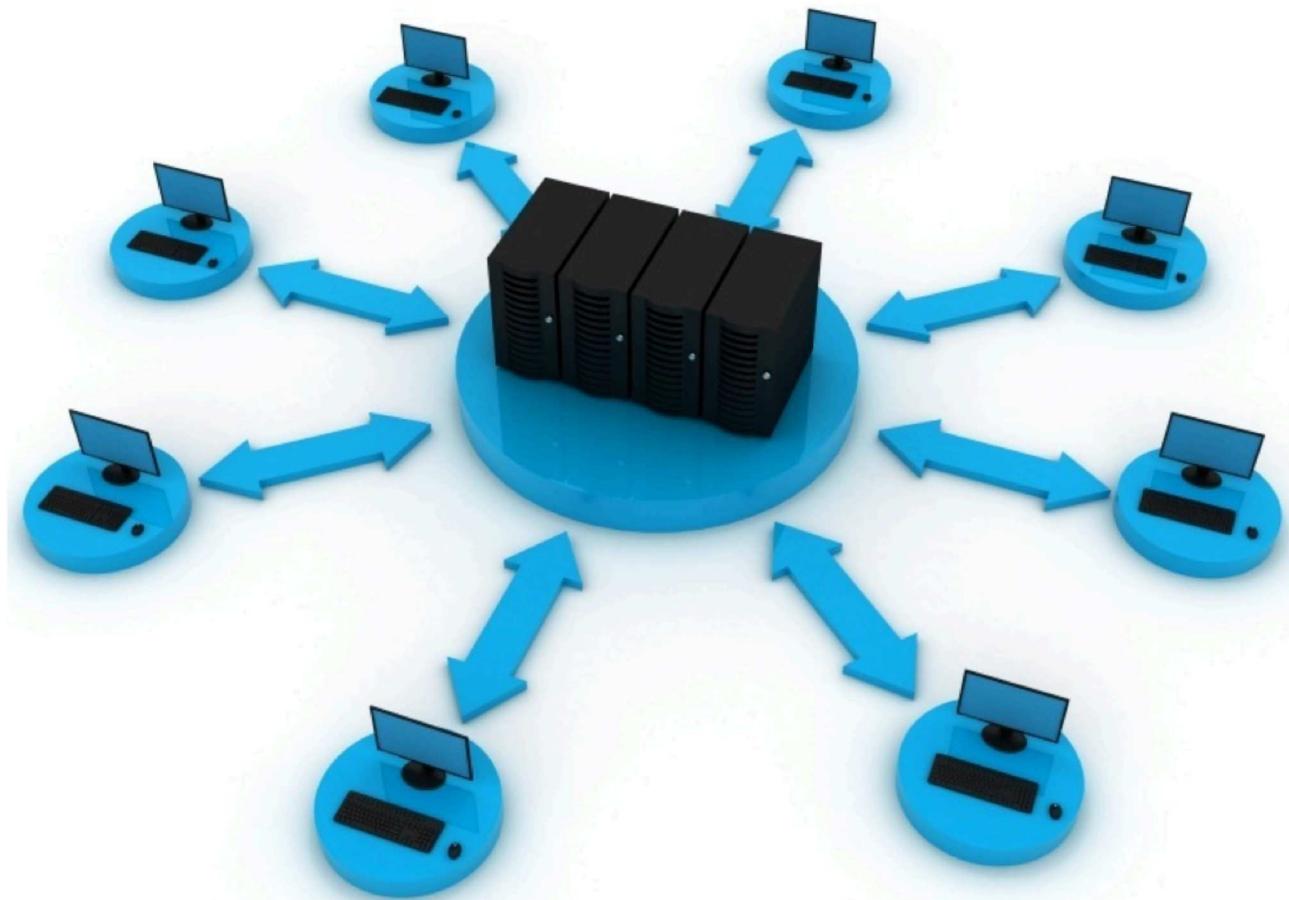
Varios sistemas acoplados por un secuenciador de trabajo sobre problemas relacionados



Sistemas distribuidos



Computación grid



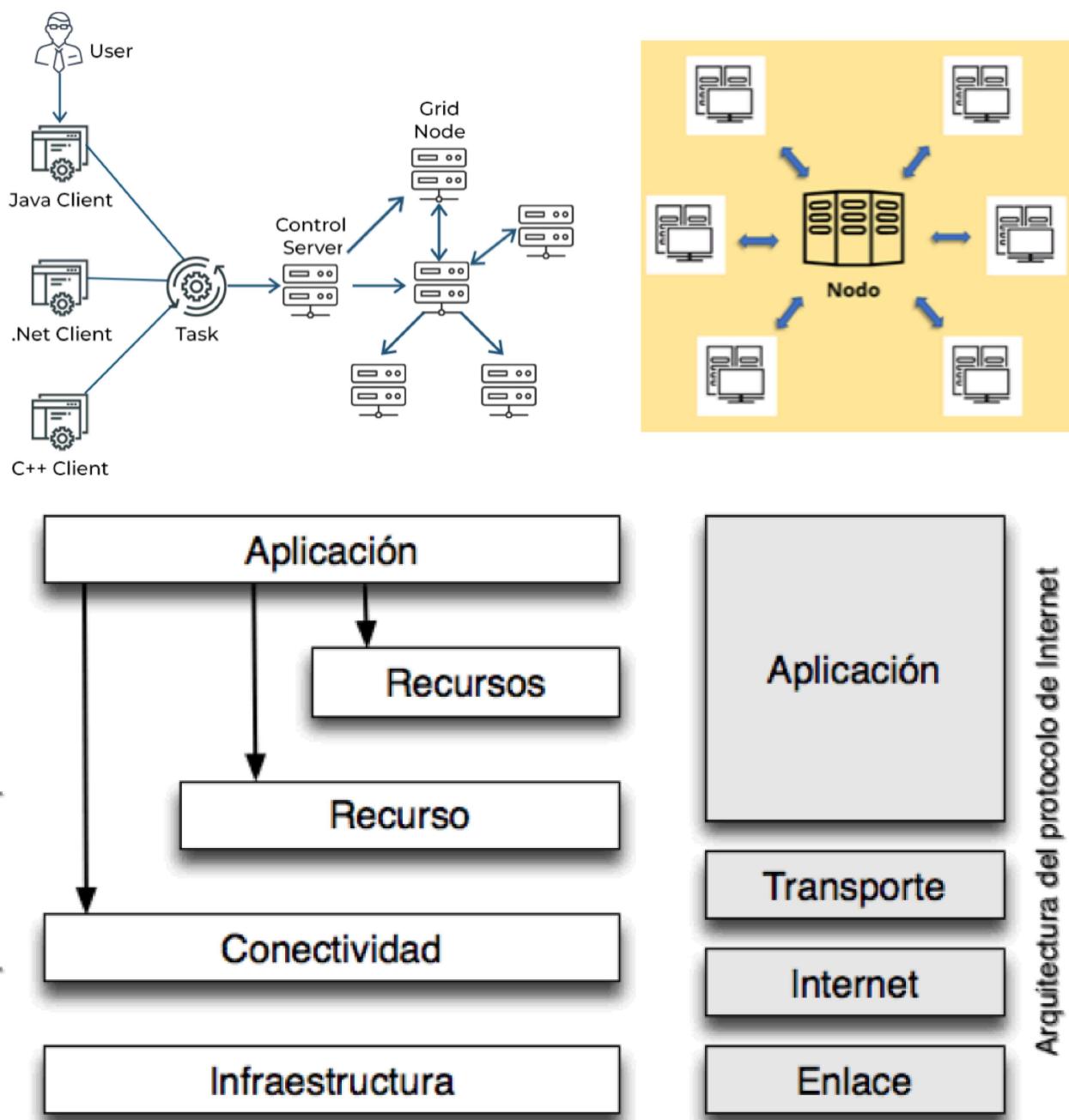
Varios sistemas acoplados por software y redes para trabajar en conjunto en problemas simples o en problemas relacionados.

Unión de Clusters de computadores, recursos computacionales, datos, grupos de investigación, científicos, etc., distribuidos geográficamente y conectados mediante redes WAN

Funcionamiento de la computación grid

Middleware para comunicación transparente y explotación de recursos.

- El objetivo final es usar recursos remotos
- Se requieren conexiones de redes rápidas
- El grid busca el uso eficiente de los recursos
- Es esencial la seguridad centrada en: políticas de acceso, autenticación y autorización.
- Es importante estandarizar las aplicaciones grid.



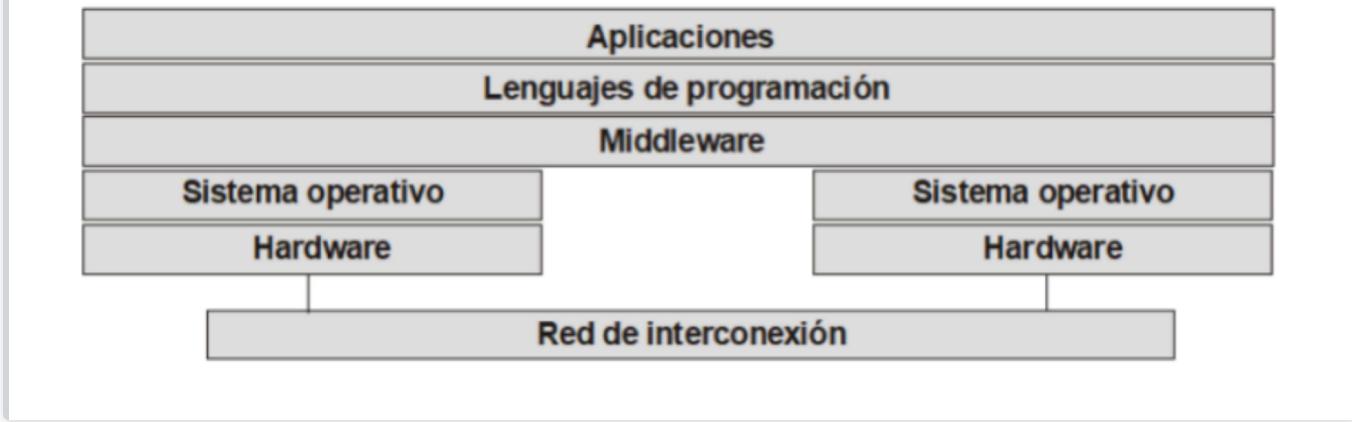
Arquitectura Grid

- Capa de aplicación
- Capa de middleware
- Capa de recursos
- Capa de red

Middleware

El auténtico cerebro del grid, se encarga de las siguientes funciones:

- Encontrar lugar conveniente para ejecutar la tarea solicitada por el usuario.
- Optimizar el uso de recursos que se encuentren dispersos.
- Organizar el acceso eficiente a los datos.
- Autenticar los diferentes elementos.
- Ejecutar tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar automáticamente la recuperación frente a fallos.
- Notificar el fallo, culminación de la ejecución de una tarea y sus resultados.

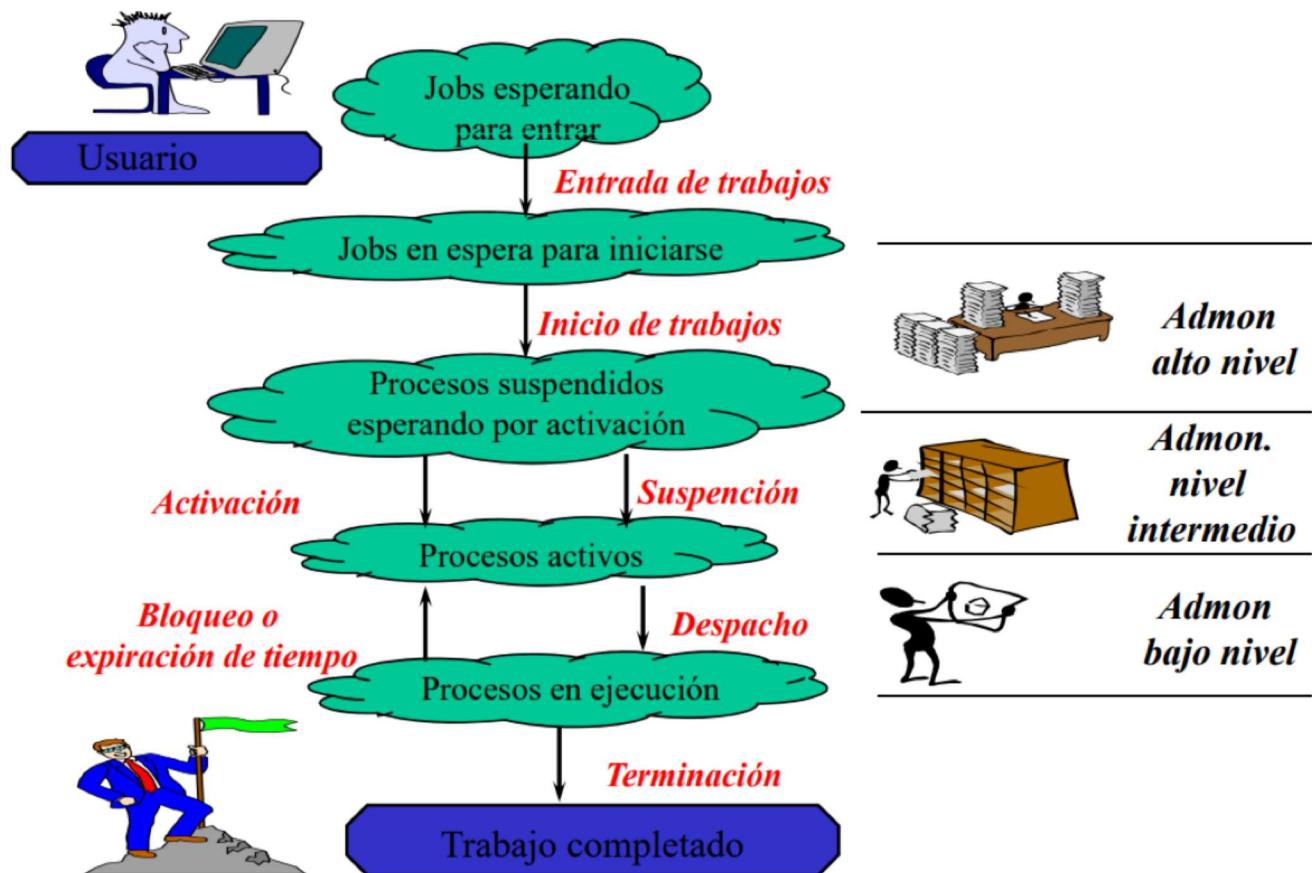


¿Dónde corre lo que se va a programar en el curso?

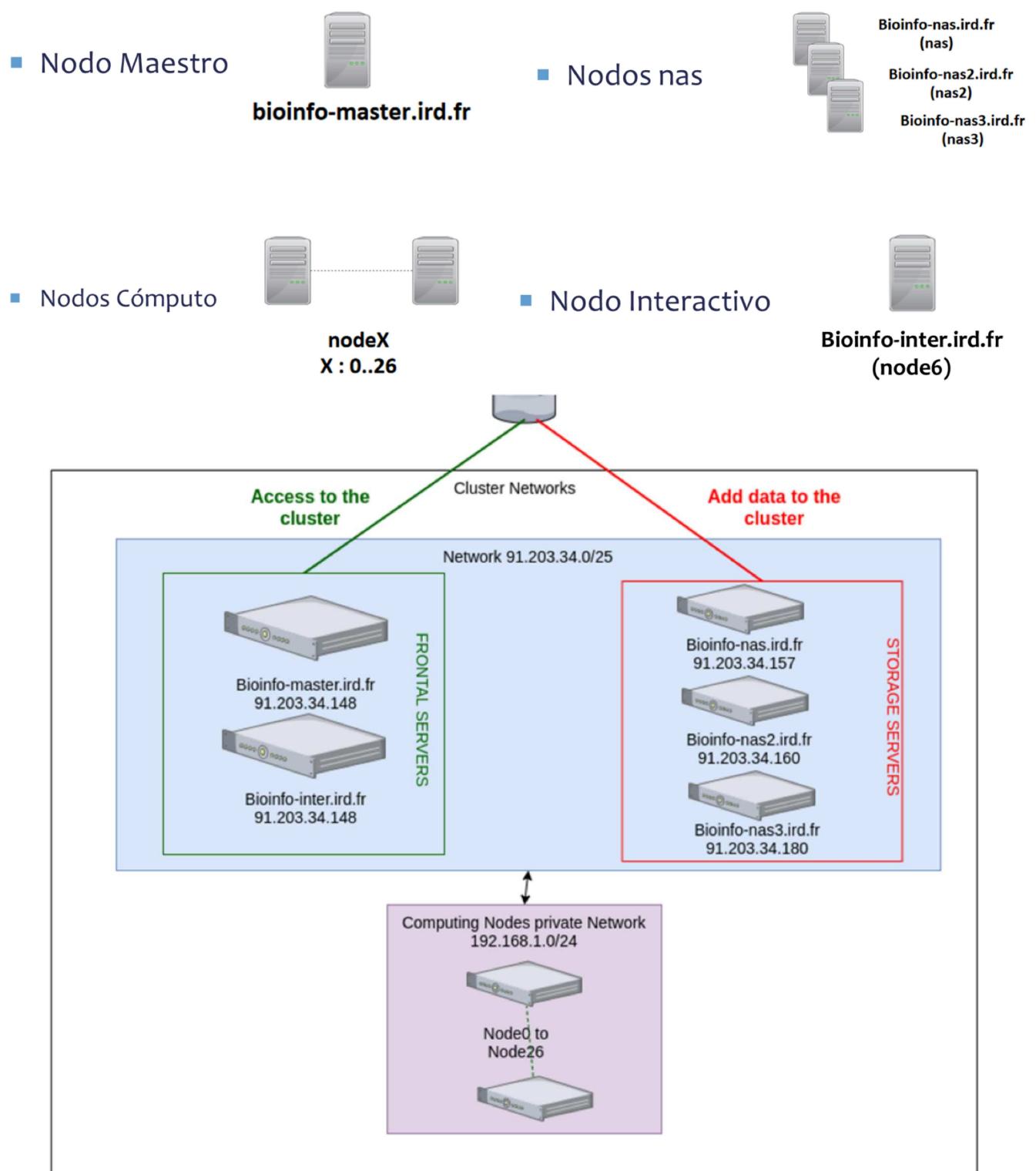
Infraestructura: Cluster



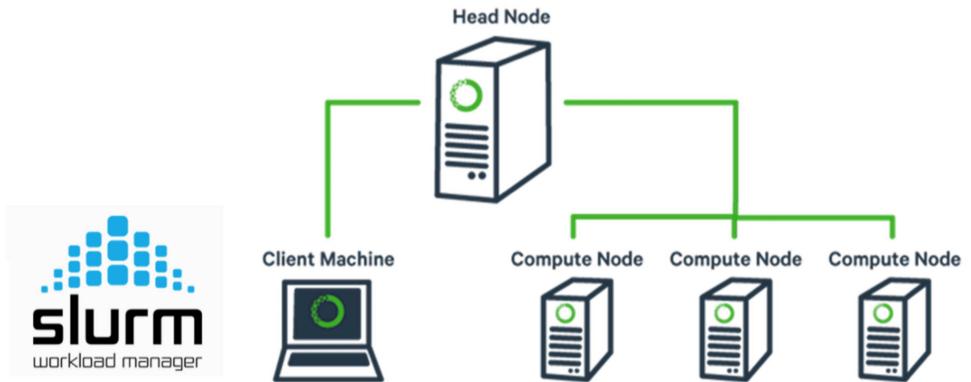
Colas y calendarizadores



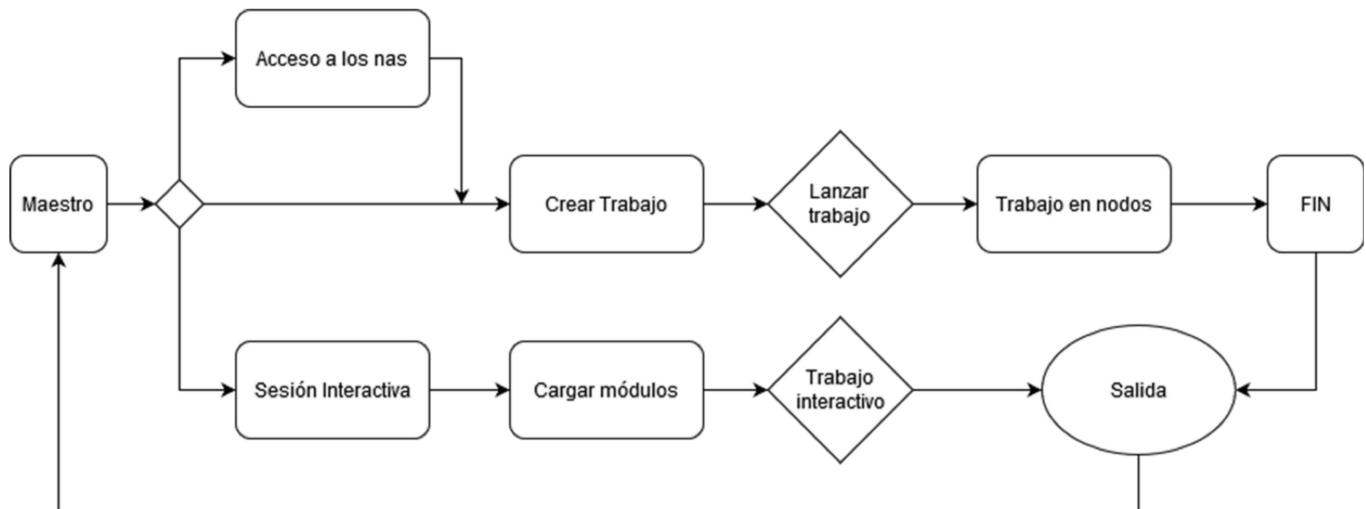
- Administrar recursos del Clúster
- Priorizar trabajos
- Limitar capacidad de cómputo
- Administrar usuarios



Algunos calendarizadores



Flujo de trabajo con calendarizadores

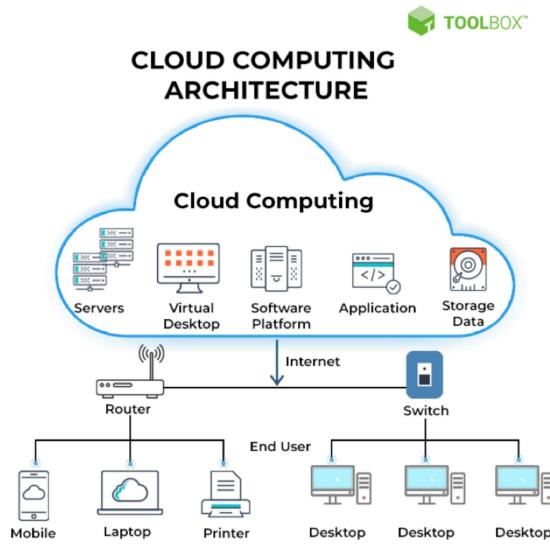


Así se ve un job para calendarizador

```
#!/bin/bash
## Define the job name
#SBATCH --job-name=test
## Define the output file
#SBATCH --output=res.txt
## Define the number of tasks
#SBATCH --ntasks=1
```

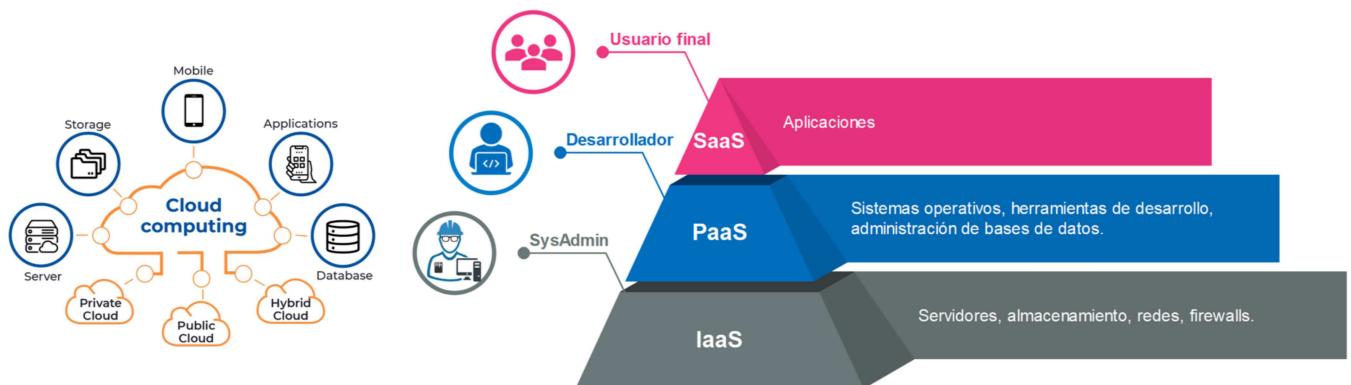


Infraestructura: Cloud

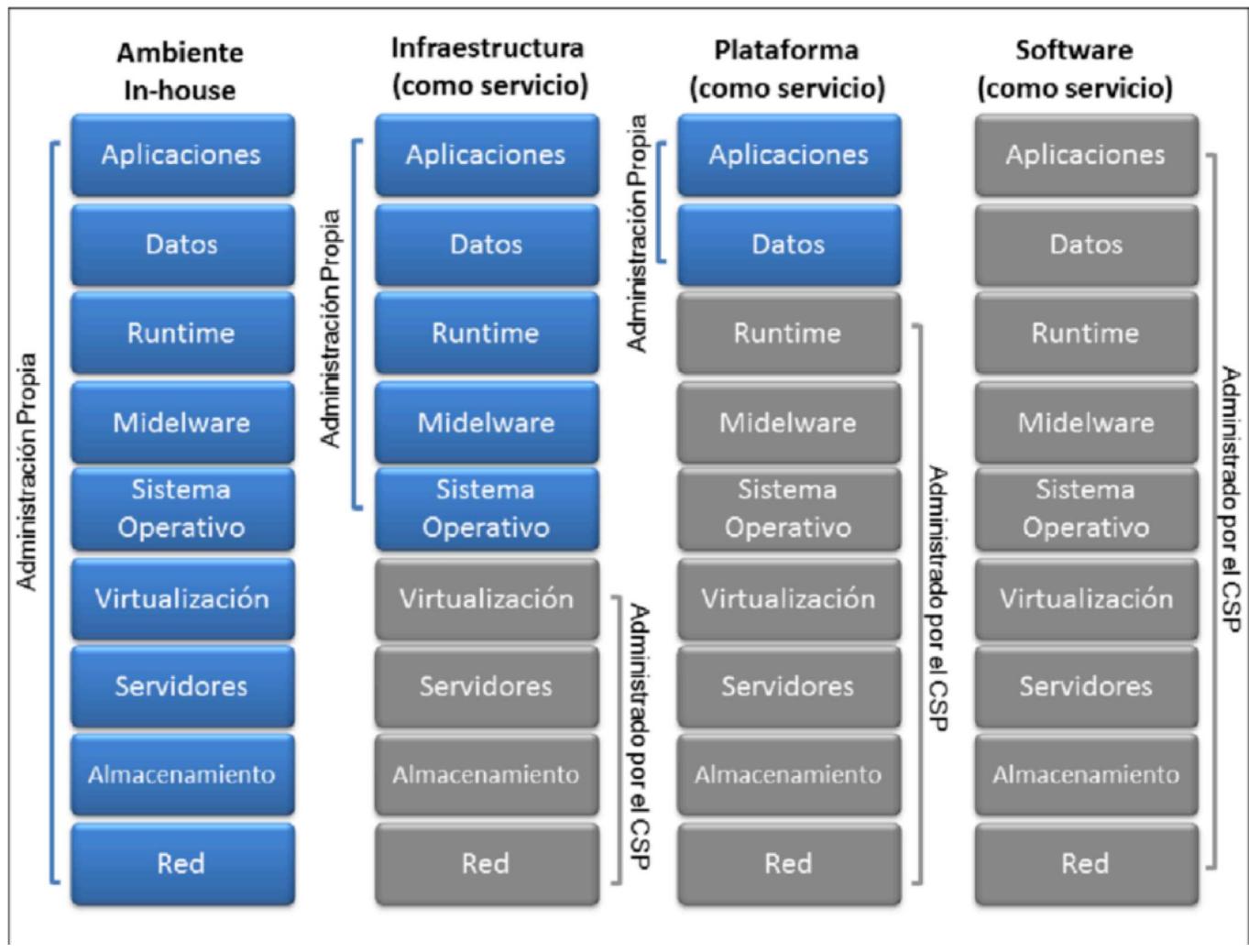


- Externalizar TI a proveedores de servicio
- Presentar transparencia al usuario final
- Asignación dinámica de recursos bajo demanda (Sin compartir) – Pago asociado
- Virtualización

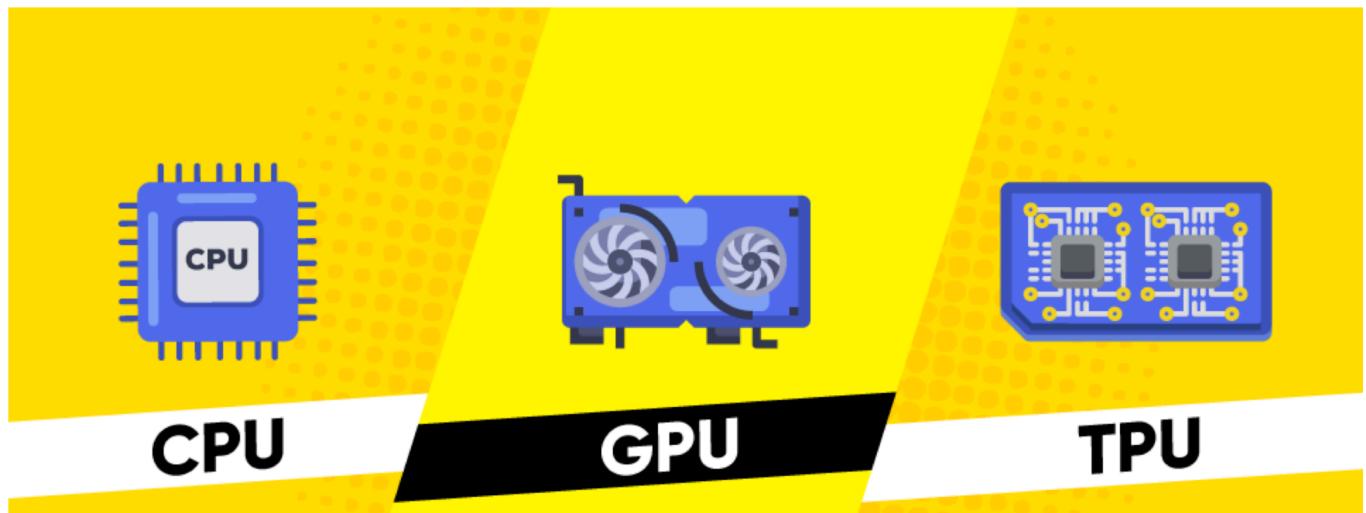
Computación cloud

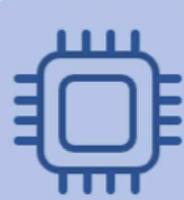


Infraestructura tradicional Vs Cloud



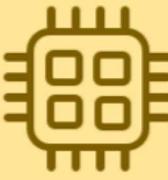
CPU & GPU & TPU





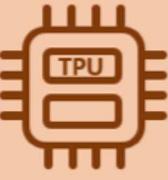
CPU

- Small models
- Small datasets
- Useful for design space exploration



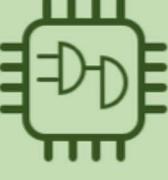
GPU

- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL



TPU

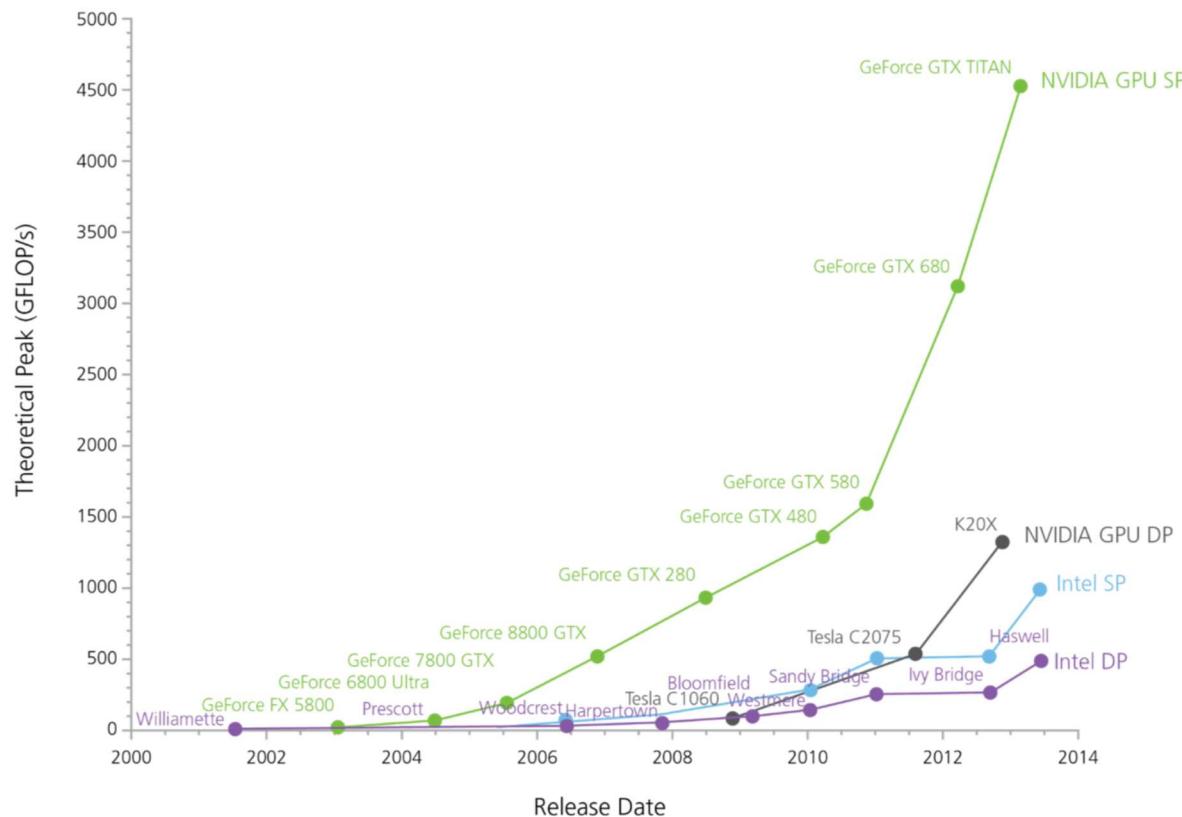
- Matrix computations
- Dense vector processing
- No custom TensorFlow operations



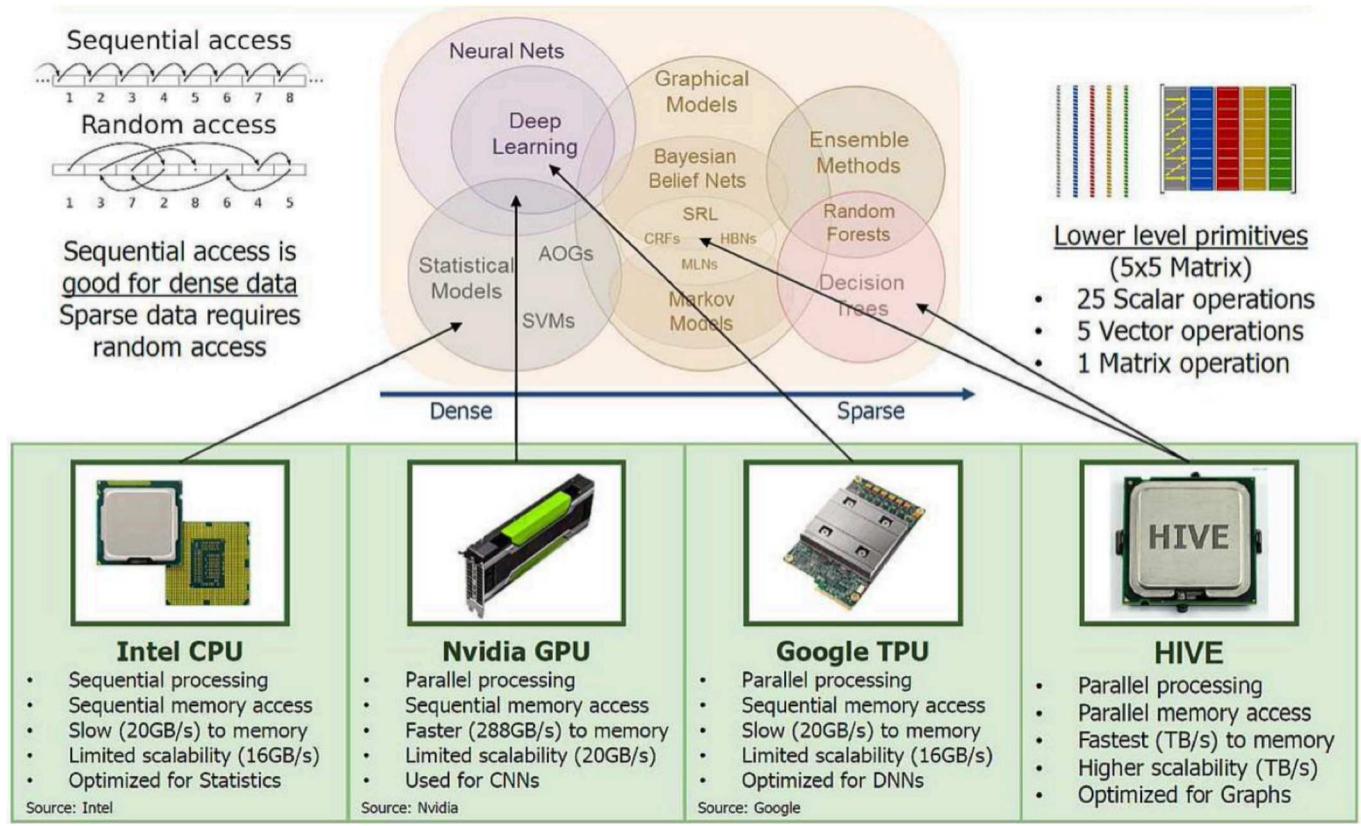
FPGA

- Large datasets, models
- Compute intensive applications
- High performance, high perf./cost ratio

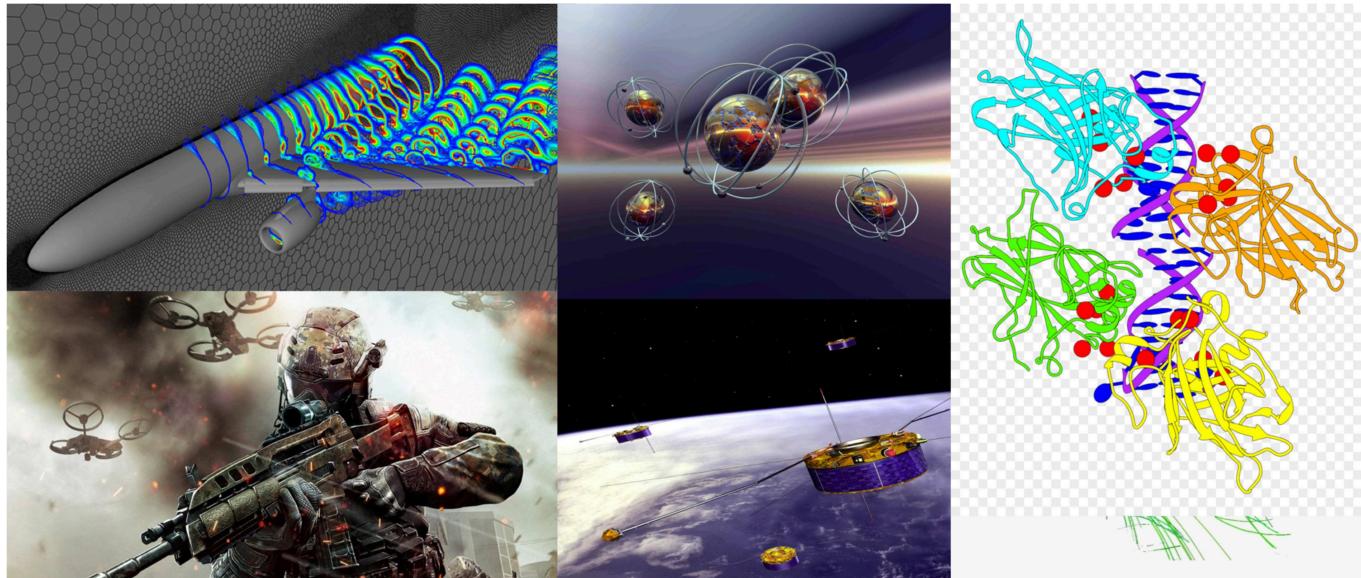
CPU & GPU



Actualidad del software para Big Data



Aplicaciones del HPC



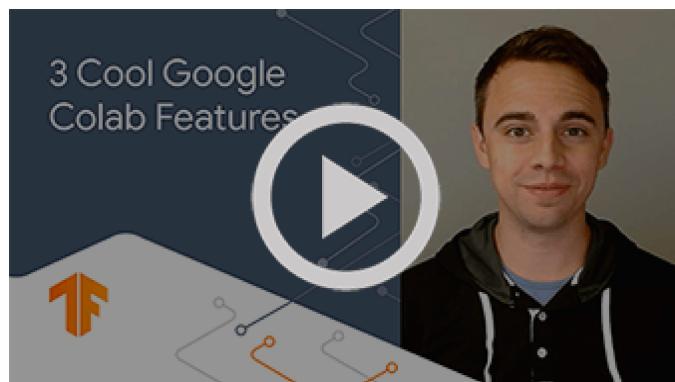
Introducción

Contenido

- **Introducción**
- Ciencia de datos
- Aprendizaje automático
- Más recursos
- Como saber la versión de Python
- Como saber los recursos de GPU
- Como saber la versión de Linux
- Tutorial Python
- Conexión con Google drive
- Asignación automática de tipos de variables
- Gestión de la memoria
- Casteo de variables
- Operaciones Aritméticas
- Funciones útiles
- Numeros Aleatorios
- Libreria Math
- Operaciones con cadena de texto
- Tuplas (Datos inmutables)
- Listas (Datos no inmutables)
- Diccionarios
- Entrada y salida de datos
- Operadores relacionales
- Operadores lógicos
- Estructuras condicionadas
- Estructuras repetitivas

- Funciones
- Paso por valor y referencia
- Manejo de excepciones y errores.

Si ya conoces Colab, mira este video para aprender sobre las tablas interactivas, la vista histórica de código ejecutado y la paleta de comandos.



¿Qué es Colab?

Colab, o «Colaboratory», te permite escribir y ejecutar código de Python en tu navegador, con

- Una configuración lista para que empieces a programar
- Acceso gratuito a GPU
- Facilidad para compartir

Seas **estudiante, científico de datos o investigador de IA**, Colab facilita tu trabajo. Mira [este video introductorio sobre Colab](#) para obtener más información, o bien comienza a usarlo más abajo.

El documento que estás leyendo no es una página web estática, sino un entorno interactivo denominado **notebook de Colab**, que permite escribir y ejecutar código.

Por ejemplo, esta es una **celda de código** con una secuencia de comandos Python corta que calcula un valor, lo almacena en una variable y devuelve el resultado:

```
seconds_in_a_day = 24 * 60 * 60  
seconds_in_a_day
```

```
86400
```

```
type(seconds_in_a_day)
```

```
int
```

A fin de ejecutar el código en la celda anterior, haz clic en él para seleccionarlo y luego presiona el botón de reproducción ubicado a la izquierda del código o usa la combinación de teclas «Command/Ctrl + Intro». Para editar el código, solo haz clic en la celda y comienza a editar.

Las variables que defines en una celda pueden usarse en otras:

```
seconds_in_a_week = 7 * seconds_in_a_day  
seconds_in_a_week
```

```
604800
```

Los notebooks de Colab te permiten combinar **código ejecutable** y **texto enriquecido** en un único documento, junto con **imágenes**, **HTML**, **LaTeX** y mucho más. Los notebooks que crees en Colab se almacenan en tu cuenta de Google Drive. Puedes compartir fácilmente los notebooks de Colab con amigos o compañeros de trabajo para que realicen comentarios o los editen. Si quieres obtener más información, consulta la [Descripción general de Colab](#). Para crear un nuevo notebook de Colab, ve al menú Archivo que aparece más arriba o usa este vínculo: [crear un nuevo notebook de Colab](#).

Los notebooks de Colab son notebooks de Jupyter que aloja Colab. Para obtener más información sobre el proyecto Jupyter, visita [jupyter.org](#).

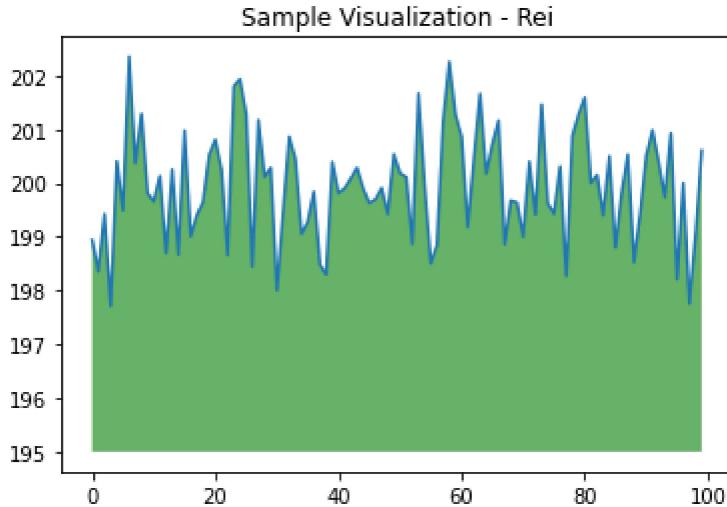
Con Colab, puedes aprovechar por completo las bibliotecas más populares de Python para analizar y visualizar datos. La celda de código que se incluye a continuación usa **NumPy** para generar algunos datos aleatorios y **matplotlib** para visualizarlos. Para editar el código, haz clic en la celda y comienza a editar.

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

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization - Rei")
plt.show()
```



Puedes importar datos propios a notebooks de Colab desde tu cuenta de Google Drive (incluso desde hojas de cálculos), GitHub y muchas otras fuentes. Para obtener más información acerca de la importación de datos y cómo puede usarse Colab para fines relacionados con la ciencia de datos, consulta los vínculos de [Cómo trabajar con datos](#).

Colab te permite importar un conjunto de datos de imágenes, entrenar un clasificador de imágenes en él y evaluar el modelo con solo [unas pocas líneas de código](#). Los notebooks de Colab ejecutan código en los servidores alojados en la nube de Google, lo que significa que puedes aprovechar al máximo el hardware de Google, incluidas las [GPU y TPU](#), independientemente de la potencia de tu máquina. Lo único que necesitas es un navegador.

Entre los usos que se la da a Colab en la comunidad de aprendizaje automático, se encuentran los siguientes:

- Introducción a TensorFlow
- Desarrollo y entrenamiento de redes neuronales
- Experimentación con TPU
- Diseminación de investigación de IA

- Creación de instructivos

Para ver notebooks de Colab de ejemplo que muestran los usos del aprendizaje automático, consulta los [ejemplos](#) que se incluyen a continuación.

Cómo trabajar con notebooks en Colab

- [Descripción general de Colaboratory](#)
- [Guía para usar Markdown](#)
- [Cómo importar bibliotecas y luego instalar dependencias](#)
- [Cómo guardar y cargar notebooks en GitHub](#)
- [Formularios interactivos](#)
- [Widgets interactivos](#)

Cómo trabajar con datos

- [Cómo cargar datos: Drive, Hojas de cálculo y Google Cloud Storage](#)
- [Gráficos: visualización de datos](#)
- [Cómo comenzar a usar BigQuery](#)

Curso intensivo de aprendizaje automático

Estos son algunos de los notebooks del curso de aprendizaje automático en línea de Google.

Para obtener más información, consulta el [sitio web del curso completo](#).

- [Introducción a Pandas DataFrame](#)
- [Regresión lineal con tf.keras usando datos sintéticos](#)

Uso de aceleración de hardware

- [TensorFlow con GPU](#)
- [TensorFlow con TPU](#)

Ejemplos destacados

- [NeMo Voice Swap](#): Utiliza el kit de herramientas de NeMo para IA conversacional de Nvidia si quieres cambiar una voz en un fragmento de audio por otra generada por computadora.
- [Reentrenamiento de un clasificador de imágenes](#): compila un modelo de Keras sobre un clasificador de imágenes previamente entrenado para distinguir flores.
- [Clasificación de texto](#): clasifica opiniones sobre películas de IMDB como *positivas* o *negativas*.
- [Transferencia de estilos](#): usa el aprendizaje profundo para transferir el estilo de una imagen a otra.
- [Codificador universal de oraciones en varios idiomas para preguntas y respuestas](#): usa un modelo de aprendizaje automático para responder preguntas del conjunto de datos SQuAD.
- [Interpolación de videos](#): predice lo que sucedió en un video entre el primer y el último fotograma.

```
#Versión de python  
!python --version
```

Python 3.8.10

```
# Recursos de GPU  
!nvidia-smi  
!/usr/local/cuda/bin/nvcc --version
```

Tue Feb 21 19:33:36 2023

```
+-----+
| NVIDIA-SMI 510.47.03      Driver Version: 510.47.03      CUDA Version: 11.6 |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|          Memory-Usage | GPU-Util  Compute M.  |
| |                               |               MIG M.   |
+-----+
| 0  Tesla T4           Off  | 00000000:00:04.0 Off |                0 |
| N/A   67C     P0    28W /  70W |          0MiB / 15360MiB |      0%      Default |
| |                               |                           N/A |
+-----+
+-----+
| Processes:
| GPU  GI  CI      PID  Type  Process name                  GPU Memory
|       ID  ID
+-----+
| No running processes found
+-----+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_Mar_8_18:18:20_PST_2022
Cuda compilation tools, release 11.6, V11.6.124
Build cuda_11.6.r11.6/compiler.31057947_0
```

```
# Versión de Ubuntu
!lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.5 LTS
Release:        20.04
Codename:       focal
```

<https://www.w3schools.com/python/default.asp>

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

En Python no es necesario declarar explícitamente el tipo de las variables. El intérprete trata de inferir el tipo de las variables según se usan. Igualmente, las operaciones tienen semántica distinta para distintos tipos de datos. Fíjate en el ejemplo siguiente.

```
a = 10
b = 3
c = 3.
d = "holoWorld"
e = True
f = 1e3
g = [1,2,3] # La lista es mutable
h = (1,2,3) # La tupla es inmutable
i = 5j
j = 1+2j

print ("a=",type(a))
print ("b=",type(b))
print ("c=",type(c))
print ("d=",type(d))
print ("e=",type(e))
print ("f=",type(f))
print ("g=",type(g))
print ("h=",type(h))
print ("i=",type(i))
print ("j=",type(j))
```

```
a= <class 'int'>
b= <class 'int'>
c= <class 'float'>
d= <class 'str'>
e= <class 'bool'>
f= <class 'float'>
g= <class 'list'>
h= <class 'tuple'>
i= <class 'complex'>
j= <class 'complex'>
```

Tipo de datos BOOLEANO

El tipo de datos *boolean* (booleano) denota el rango de *valores lógicos* que consiste de dos elementos *true* y *false* y está definido de la siguiente manera:

$$\text{type boolean} = (\text{false}, \text{true})$$

Los siguientes *operadores* (operaciones) están definidos sobre argumentos de este tipo.

- AND: conjunción lógica,

$$\wedge : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- OR: disyunción lógica, inclusiva,

$$\vee : \{\text{false}, \text{true}\} \times \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

- NOT: negación lógica,

$$\neg : \{\text{false}, \text{true}\} \rightarrow \{\text{false}, \text{true}\}$$

True and False , True or False, not True

(False, True, False)

Operadores relacionales

Todos los operadores relacionales producen un resultado de tipo *Booleano*. Los seis operadores relacionales básicos son,

$$=, \neq, <, \leq, \geq, >$$

Por ejemplo, de acuerdo a los valores que sean asignados a las variables x y y , el valor de la siguiente expresión es *false* o *true* :

$$(x + y \geq 0 \wedge x * y < 0) \vee \neg(x * x \geq 20).$$

Los cuatro operadores relacionales $<$, \leq , \geq , $>$ solo pueden ser aplicados a tipos de datos ordenados (escalares).

`2>1, True<False`

(True, False)

Tipo de datos INTEGER

EL tipo *integer* representa el conjunto de *números enteros*. Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: *integer* \times *integer* \rightarrow *integer*
- sustracción $-$: *integer* \times *integer* \rightarrow *integer*
- multiplicación $*$: *integer* \times *integer* \rightarrow *integer*
- división con truncamiento **div**: *integer* \times (*integer* $- \{0\}) \rightarrow *integer*
La operación *a div b* no está definida para $b = 0$.$
- residuo de la división entera **mod** : *integer* \times *integer*⁺ \rightarrow *integer*
La operación *a mod b* solo está definida para b un número entero positivo.

Profundizar en la representación de enteros en python:

- [https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To store integers%2C the computers,numbers to represent the integers.&text=By using 8 bits%2C you,8 – 1 %3D 255 integers.](https://www.pythontutorial.net/advanced-python/python-integers/#:~:text=To%20store,integers%2C%20the%20computers%2C%20numbers%20to%20represent%20the%20integers.&text=By%20using%208%20bits%2C,you,%208%20-%201%20=%20255%20integers.)

```
import numpy as np  
z=np.uint8(255)  
z
```

255

```
z,type(z),z nbytes
```

(255, numpy.uint8, 1)

```
x=np.int8(4)+np.int8(6)  
y=np.uint8(100)+np.uint8(27)
```

```
x, y, type(x), type(y)
```

(10, 127, numpy.int8, numpy.uint8)

Tipo de datos REAL

- Todo intervalo de números reales contiene infinitos números.
- Los números reales son densos.
- El tipo *real* no representa exactamente a los números reales.
- El tipo *real* es un conjunto finito de representantes de intervalos de números reales.
- El tipo *real* no es un tipo ordinal.
- Cursos de Sistemas Númericos, Análisis numérico.

Los siguientes operadores están definidos para este tipo de datos.

- adición $+$: *real* \times *real* \rightarrow *real*
 - sustracción $-$: *real* \times *real* \rightarrow *real*
 - multiplicación $*$: *real* \times *real* \rightarrow *real*
 - división $/$: *real* \times (*real* $- \{0\}$) \rightarrow *real*
- La operación a/b no está definida para $b = 0$.

Char

Tipo de datos CHAR

El tipo *char* denota un *conjunto de caracteres*, finito y ordenado.

- 26 letras
- 10 dígitos
- caracteres especiales, como el carácter para espacio y otros símbolos de puntuación.

Funciones de transferencia y funciones predecesor y sucesor

Las funciones de transferencia del tipo *char* al tipo *integer* y viceversa, están definidas por:

- $ord(c)$ es el número entero, llamado ordinal, del carácter c en el conjunto de caracteres.
- $chr(i)$ es el carácter con número ordinal i .

Por tanto, se tienen las siguientes relaciones

$$chr(ord(c)) = c \quad \text{y} \quad ord(chr(i)) = i$$

y el ordenamiento del conjunto de caracteres está definido por:

$$c_1 < c_2 \text{ si y solamente si } ord(c_1) < ord(c_2)$$

Las funciones predecesor y sucesor están definidas respectivamente por:

$$pred(c) = chr(ord(c) - 1) \quad \text{y} \quad succ(c) = chr(ord(c) + 1)$$

```
ord("a"),ord("A"), ord("b"),ord("c"),ord("d")
```

```
(97, 65, 98, 99, 100)
```

```
'a'>'b'
```

```
False
```

```
'a'>'A'
```

```
True
```

```
chr(97),chr(98)
```

```
('a', 'b')
```

Profundizar en la gestión de memoria en Python

- [customprogrammingsolutions/python-memory-usage](https://customprogrammingsolutions.com/python-memory-usage)
- [https://codeblessu.com/python/size-of-data-types.html#:~:text=Use sys.,to store it in memory](https://codeblessu.com/python/size-of-data-types.html#:~:text=Use%20sys.%20to%20store%20it%20in%20memory)
- <https://www.lesinskis.com/images/CPythonMemoryStructurePosterJanisLesinskisAlyshalannetta.pdf>

```
from sys import getsizeof
import sys as yi
```

```
help(getsizeof)
```

Help on built-in function getsizeof in module sys:

```
getsizeof(...)
    getsizeof(object [, default]) -> int

    Return the size of object in bytes.
```

```
a=1000000000000000000000000000000000000000000000000000000000000000
print ("a=",a,'el tamaño es=', getsizeof(a))
print ("b=", b, 'el tamaño es=',getsizeof(b))
print ("c=",c, 'el tamaño es=',getsizeof(c))
print ("d=",d, 'el tamaño es=',getsizeof(d))
```

```
a= 1000000000000000000000000000000000000000000000000000000000000000 el tamaño es= 40
b= 3 el tamaño es= 28
c= 3.0 el tamaño es= 24
d= holaWorld el tamaño es= 58
```

```
a = 5
b = np.int8(5)
c = 'reine'
d = np.array([1,2,3,4]).astype('int8')

print(type(a), getsizeof(a))
print(b.nbytes, type(b))
print(type(c), getsizeof(c))
print(d.nbytes, type(d))
```

```
<class 'int'> 28
1 <class 'numpy.int8'>
<class 'str'> 54
4 <class 'numpy.ndarray'>
```

```
x=np.int8(128)
```

`x, x nbytes, type(x)`

(-128, 1, numpy.int8)

$$y=40$$

`getsizeof(y), type(y)`

(28, int)

`type(x).getsizeof(x)`

(int, 96)

¿Por qué el resultado de las dos últimas divisiones no es el mismo?

```
a=10  
b=3  
c=3.  
print ("a =",a,"; b =",b, " ; c =",c)  
print ("a/b =", a/b)  
print ("a/c =", a/c)  
print ("a//b =", a//b)  
print ("a//c =", a//c)
```

```
a = 10 ; b = 3 ; c = 3.0  
a/b = 3.333333333333335  
a/c = 3.333333333333335  
a//b = 3  
a//c = 3.0
```

```
Variable = """  
Esta es una cadena  
de varias líneas  
"""
```

```
print(Variable)
```

```
Esta es una cadena  
de varias líneas
```

```
type(Variable)
```

```
str
```

```
getsizeof(Variable)
```

```
110
```

```
x=np.array(['a','b','c'])
```

```
x.nbytes,type(x)
```

```
(12, numpy.ndarray)
```

```
x='2'
y='3'
z=x+y
print('z=',z)

x1=int(x)
y1=int(y)
z1=x1+y1
print('z1=',z1)
```

```
z= 23
z1= 5
```

```
print("Suma = ", 4+5)
print("Resta = ", 4-5)
print("Multiplicación = ", 4*5)
print("División = ", 4/5)
print("División entera = ", 10//3)
print("Residuo = ", 10%3)
print("Potencia = ", 2**3)
print("Raíz = ", 2**0.5)
```

```
Suma = 9
Resta = -1
Multiplicación = 20
División = 0.8
División entera = 3
Residuo = 1
Potencia = 8
Raíz = 1.4142135623730951
```

```
print("Redondear números= ",round(3.867483,3))
print("Valor Absoluto= ",abs(-4))
print("Convertir número a decimal= ",float('3.5'))
print("Convertir número a hexadecimal=", hex(5))
print("Convertir cadena o número decimal a entero=", int(6.9))
print("Convertir número a cadena de texto=", str(6)+str(6))
```

```
Redondear números= 3.867
Valor Absoluto= 4
Convertir número a decimal= 3.5
Convertir número a hexadecimal= 0x5
Convertir cadena o número decimal a entero= 6
Convertir número a cadena de texto= 66
```

```
import random
print('imprime un numero aleatorio entero=' ,random.randrange(3,10))
print('selecciona aleatoriamente=' ,random.choice(["uno", "dos", "tres"]))
print('imprime un numero aleatorio decimal entre 0 y 1=' ,random.random())
print('imprime una letra aleatoria=' ,random.choice("AEIOU"))
```

```
imprime un numero aleatorio entero= 9
selecciona aleatoriamente= dos
imprime un numero aleatorio decimal entre 0 y 1= 0.5008250519253324
imprime una letra aleatoria= U
```

```
print('imprime un numero aleatorio decimal entre 0 y 4=' ,int(4*random.random()))
```

```
imprime un numero aleatorio decimal entre 0 y 4= 3
```

```
import math
print("Factorial=",math.factorial(3))
print("PI=",math.pi)
print("E=",math.e)
print("seno=",math.sin(math.pi/6))
print("aseño=",math.degrees(math.asin(0.5)))
print("Logaritmo Natural=",math.log(math.exp(2)))
print("Logaritmo en base 10=",math.log10(100))
print("Logaritmo en base 2=",math.log2(8))
print("Potencia=",math.pow(3,2))
print("Raiz cuadrada=",math.sqrt(4))
```

```
Factorial= 6
PI= 3.141592653589793
E= 2.718281828459045
seno= 0.4999999999999994
aseño= 30.00000000000004
Logaritmo Natural= 2.0
Logaritmo en base 10= 2.0
Logaritmo en base 2= 3.0
Potencia= 9.0
Raiz cuadrada= 2.0
```

```
import numpy as np  
np.float(5/0)
```

```
<ipython-input-83-136236f1537d>:2: DeprecationWarning: `np.float` is a deprecated alias  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/rele  
np.float(5/0)
```

```
-----  
ZeroDivisionError Traceback (most recent call last)  
<ipython-input-83-136236f1537d> in <module>  
      1 import numpy as np  
----> 2 np.float(5/0)  
  
ZeroDivisionError: division by zero
```

5/0

```
-----  
ZeroDivisionError Traceback (most recent call last)  
<ipython-input-82-0106664d39e8> in <module>  
----> 1 5/0  
  
ZeroDivisionError: division by zero
```

```
#Suma o concatenación  
x="Un divertido"+"programa"+ "de" + "radio"  
print(x)  
#Multiplicación de una cadena con un número  
y=3*"programas"  
print(y)  
#Obtener longitud de una cadena  
print('longitud de la cadena:',len(y))
```

Un divertidoprogamaderadio
programasprogramasprogramas
longitud de la cadena: 27

```
#Acceder a una posición de la cadena
cadena = "programA"
print('acceder a una posicion determinada:',cadena[0])
print('acceder a la última posición:',cadena[-1])

# Comillas simples
cadena = 'Texto entre comillas simples'
print (cadena)
print (type(cadena))

# Comillas dobles
cadena = "Texto entre comillas dobles"
print (cadena)
print (type(cadena))
```

```
acceder a una posicion determinada: p
acceder a la última posición: A
Texto entre comillas simples
<class 'str'>
Texto entre comillas dobles
<class 'str'>
```

cad='reinel'

cad[2:5]

'ine'

```
cadenaesc = 'Texto entre \n\n\n \t\t\tcomillas simples'
print (cadenaesc)
print (type(cadenaesc))
```

Texto entre

comillas simples
<class 'str'>

Las tuplas son más rápidas que las listas. Si define un conjunto **constante** de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.

Una tupla es una secuencia de items ordenada e inmutable.

Los items de una tupla pueden ser objetos de cualquier tipo.

Para especificar una tupla, lo hacemos con los elementos separados por comas dentro de **paréntesis**.

Una tupla con únicamente dos elementos es denominada par.

Para crear una tupla con un único elemento (singleton), se añade una coma al final de la expresión.

Para definir una tupla vacía, se emplean unos paréntesis vacíos.

```
tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
print(tupla)
print(type(tupla))
print(tupla[1:4])
print('cuantas veces hay un elemento=',tupla.count(25))
print('dice en que posicion esta el elemento=',tupla.index(15))
#tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.
```

```
('cadena de texto', 15, 2.8, 'otro dato', 25, 25)
<class 'tuple'>
(15, 2.8, 'otro dato')
cuantas veces hay un elemento= 2
dice en que posicion esta el elemento= 1
```

```
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-6-1233f0186c5b> in <module>
      5 print('cuantas veces hay un elemento=',tupla.count(25))
      6 print('dice en que posicion esta el elemento=',tupla.index(15))
----> 7 tupla[0]=2# dado que es una TUPLA no se puede agregar ni borrar información.

TypeError: 'tuple' object does not support item assignment
```

```
tupla, type(tupla), tupla[1:-1],tupla.count(25),tupla.index(15)
```

```
(('cadena de texto', 15, 2.8, 'otro dato', 25, 25),
 tuple,
 (15, 2.8, 'otro dato', 25),
 2,
 1)
```

```
# Usando el tag: raises-exception, se evita que se detenga en compilación
#tupla[0]=2
```

Una lista es una secuencia ordenada de elementos **mutable**.

Los items de una lista pueden ser objetos de distintos tipos.

Para especificar una lista se indican los elementos separados por comas en el interior de **CORCHETES**.

Para denotar una lista vacía se emplean dos corchetes vacíos.

```
a = [1,2,3, "hola", [10, "nunca", 90], -32]
print ("Toda la lista= ",a)
print(type(a))
print ("Longitud de la lista= ", len(a))
print("Acceder a un rango de posiciones=",a[0:4])
print("Acceder a una posición específica= ",a[4])
print("Acceder a la última posición= ",a[-1])
```

```
Toda la lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
<class 'list'>
Longitud de la lista=  6
Acceder a un rango de posiciones= [1, 2, 3, 'hola']
Acceder a una posición específica=  [10, 'nunca', 90]
Acceder a la última posición= -32
```

```
print("Lista= ",a)
a.append("Nuevo último")
print("Lista con nuevo último= ",a)
a[1]='nuevo1'
print("Lista con nuevo elemento en las posición 1= ",a)
a.insert(2,'nuevo2')
print("Lista con nuevo elemento en las posición 2= ",a)
```

```
Lista=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32]
Lista con nuevo último=  [1, 2, 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
Lista con nuevo elemento en las posición 1=  [1, 'nuevo1', 3, 'hola', [10, 'nunca', 90]
Lista con nuevo elemento en las posición 2=  [1, 'nuevo1', 'nuevo2', 3, 'hola', [10, '
```

a,type(a),len(a)

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último'],
list,
8)
```

```
for i in range(len(a)):
    print (i, "-->", a[i])
```

```
0 --> 1
1 --> nuevo1
2 --> nuevo2
3 --> 3
4 --> hola
5 --> [10, 'nunca', 90]
6 --> -32
7 --> Nuevo último
```

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

```
1
nuevo1
nuevo2
3
hola
[10, 'nunca', 90]
-32
Nuevo último
```

```
print(a)
print (a[2:])
print (a[-3:])
print (a[4])
```

```
[1, 'nuevo1', 'nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
['nuevo2', 3, 'hola', [10, 'nunca', 90], -32, 'Nuevo último']
[[10, 'nunca', 90], -32, 'Nuevo último']
hola
```

```
a = [1,2,3,"hola", [10, "nunca", 90], -32]
print (a[4])
print (a[4][0])
print (a[4][1])
print (a[4][1][2:])
```

```
[10, 'nunca', 90]
10
nunca
nca
```

Los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados. En Python, un diccionario es una colección **no-ordenada** de valores que son accedidos a través de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos. Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior. No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves. La información almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario. Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

```
# Definir una variable diccionario
futbolistas = dict()

futbolistas = {
    1: "Casillas", 6: "Iniesta", 3: "Piqué",
    5: "Puyol",
    7: "Villa", 8: "Xavi Hernández",
    9: "Torres", 11: "Capdevila",
    14: "Xavi Alonso", 15: "Ramos",
    16: "Busquets"
}

# Recorrer el diccionario, imprimiendo clave - valor
print ("Vemos que los elementos no van \"ordenados\":")
for k, v in futbolistas.items():
    print ("{} --> {}".format(k,v))
```

Vemos que los elementos no van "ordenados":

```
1 --> Casillas
6 --> Iniesta
3 --> Piqué
5 --> Puyol
7 --> Villa
8 --> Xavi Hernández
9 --> Torres
11 --> Capdevila
14 --> Xavi Alonso
15 --> Ramos
16 --> Busquets
```

```
futbolistas.items(),futbolistas.keys(),futbolistas.values()
```

```
(dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa')]),  
 dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16]),  
 dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres'])
```

```
# Nº de elementos que tiene un diccionario  
numElemen = len(futbolistas)  
print ("\nEl número de futbolistas es de {}".format(numElemen))  
  
# Imprimir las claves que tiene un diccionario  
keys = futbolistas.keys();  
print ("\nLas claves de nuestro diccionario son : {}".format(keys))  
  
# Imprimir los valores que tiene un diccionario  
values = futbolistas.values();  
print ("\nLos valores de nuestro diccionario son : {}".format(values))
```

El número de futbolistas es de 11

Las claves de nuestro diccionario son : dict_keys([1, 6, 3, 5, 7, 8, 9, 11, 14, 15, 16])

Los valores de nuestro diccionario son : dict_values(['Casillas', 'Iniesta', 'Piqué', 'Puyol', 'Villa', 'Xavi Hernández', 'Torres', 'Capdevila', 'Ramos', 'Alonso', 'Hernández', 'Busquets'])

```
# Obtener el valor de un elemento dada su clave
elem = futbolistas.get(6)
print ("\nEl futbolista con clave 6 es {}".format(elem))

# Insertamos un elemento en el diccionario
## si la clave ya existe, el elemento NO se inserta
futbolistas.setdefault(10, 'Cesc')
print ("\nInsertamos el elemento con clave 10 y valor Cesc")
print ("Ahora el diccionario queda : {}".format(futbolistas))

# Añadimos, de otro modo, un elemento al diccionario
## si la clave ya existe, el valor se cambia por este nuevo
futbolistas[22] = 'Navas'
print ("\nAñadimos un nuevo elemento, ahora el diccionario queda: ")
print (futbolistas)
```

El futbolista con clave 6 es Iniesta

Insertamos el elemento con clave 10 y valor Cesc

Ahora el diccionario queda : {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7:

Añadimos un nuevo elemento, ahora el diccionario queda:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
# Eliminamos un elemento del diccionario dada su clave
futbolistas.pop(22)
print ("\nEliminamos el elemento con clave 22")
print ("Ahora el diccionario queda: {}".format(futbolistas))

# Hacemos una copia del diccionario
futbolistas_copia = futbolistas.copy()
print ("\nLa copia del diccionario tiene los valores:")
print (futbolistas_copia)

# Borramos los elementos de un diccionario
futbolistas_copia.clear()
print ("\nVaciamos el diccionario nuevo creado, ahora los valores en el: {}".format(fu
```

Eliminamos el elemento con clave 22

Ahora el diccionario queda: {1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: '

La copia del diccionario tiene los valores:

{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

Vaciamos el diccionario nuevo creado, ahora los valores en el: {}

```
# Comprobamos si existe o no una clave en un diccionario
if 2 in futbolistas:
    print ("\nEl futbolista con la clave 2 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 2 NO existe en el diccionario.")

if 8 in futbolistas:
    print ("\nEl futbolista con la clave 8 existe en el diccionario.")
else:
    print ("\nEl futbolista con la clave 8 NO existe en el diccionario.")
```

El futbolista con la clave 2 NO existe en el diccionario.

El futbolista con la clave 8 existe en el diccionario.

```
# Creamos un diccionario a partir de una lista de claves
keys = ['nombre', 'apellidos', 'edad']
datos_usuario = dict.fromkeys(keys, 'null')
print ("\nCreamos un diccionario a partir de la lista de claves:")
print (keys)
print ("y con valor 'null'")
print ("Así el diccionario queda: {}".format(datos_usuario))

# Creación de un diccionario
diccionario=dict({1:'rei',2:'pepe',3:'javier'})
print(diccionario)
```

Creamos un diccionario a partir de la lista de claves:

```
['nombre', 'apellidos', 'edad']
y con valor 'null'
Así el diccionario queda: {'nombre': 'null', 'apellidos': 'null', 'edad': 'null'}
{1: 'rei', 2: 'pepe', 3: 'javier'}
```

```
# Devolvemos los elementos del diccionario en una tupla
tupla = futbolistas.items()
print ("\nEl diccionario convertido en tupla queda así:")
print (tupla)

# Unimos dos diccionarios existentes
suplentes = {
    4:'Marchena', 12:'Valdes', 13:'Mata',
    17:'Arbeloa', 19:'Llorente', 20:'Javi Martinez',
    21:'Silva', 23:'Reina'
}

print ("\nUnimos el diccionario: ")
print (futbolistas)
futbolistas.update(suplentes) # Aquí hacemos la unión de los diccionarios
print ("con el diccionario:")
print (suplentes)
print ("siendo el resultado:")
print (futbolistas)
```

El diccionario convertido en tupla queda así:
dict_items([(1, 'Casillas'), (6, 'Iniesta'), (3, 'Piqué'), (5, 'Puyol'), (7, 'Villa'),

Unimos el diccionario:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',
con el diccionario:
{4: 'Marchena', 12: 'Valdes', 13: 'Mata', 17: 'Arbeloa', 19: 'Llorente', 20: 'Javi Mar
siendo el resultado:
{1: 'Casillas', 6: 'Iniesta', 3: 'Piqué', 5: 'Puyol', 7: 'Villa', 8: 'Xavi Hernández',

```
#Entrada de datos
x=input('ingrese el dato 1: ');
y=input('ingrese el dato 2: ');
print('tipo de datos',type(x),type(y))
print('suma de datos sin casteo',x+y)
print('suma de datos con casteo',float(x)+float(y))
```

```
ingrese el dato 1: 5
ingrese el dato 2: 6
tipo de datos <class 'str'> <class 'str'>
suma de datos sin casteo 56
suma de datos con casteo 11.0
```

Operador	Descripción	Ejemplo
<code>==</code>	¿son iguales a y b?	<code>r = 5 == 3 # r es False</code>
<code>!=</code>	¿son distintos a y b?	<code>r = 5 != 3 # r es True</code>
<code><</code>	¿es a menor que b?	<code>r = 5 < 3 # r es False</code>
<code>></code>	¿es a mayor que b?	<code>r = 5 > 3 # r es True</code>
<code><=</code>	¿es a menor o igual que b?	<code>r = 5 <= 5 # r es True</code>
<code>>=</code>	¿es a mayor o igual que b?	<code>r = 5 >= 3 # r es True</code>

```
a = 5
b = 5
a1 = 7
b1 = 3
c1 = 3

cadena1 = 'Hola'
cadena2 = 'Adios'

# igual
c = a == b
print (c)

cadenas = cadena1 == cadena2
print (cadenas)

# diferente
d = a1 != b
print (d)

cadena0 = cadena1 != cadena2
print (cadena0)

# mayor que
e = a1 > b1
print (e)

# menor que
f = b1 < a1
print (f)

# mayor o igual que
g = b1 >= c1
print (g)

# menor o igual que
h = b1 <= c1
print (h)
```

```
True
False
True
True
True
True
True
True
```

```
5==5
```

True

```
7!=9
```

True

```
cadena1 = 'Hola'
```

```
cadena1
```

'Hola'

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	r = True and False # r es False
or	¿se cumple a o b?	r = True or False # r es True
not	No a	r = not True # r es False

```
x=3  
y=7  
if x==3 and y==7:  
    print('si')  
else:  
    print('no')
```

si

```
if condicion1:  
    instruccion1  
    instruccion2  
else:  
    if condición2:  
        instruccion3  
        instruccion4  
    else:  
        instruccion5  
        instruccion6  
# fin del if  
instruccion7  
instruccion8
```

```
if 5>2:  
    print('si')  
    print('si2')  
elif 1>3:  
    print('sino si')  
else:  
    print('no')  
    print('PorFuera')  
print('PorFuera2')
```

si
si2
PorFuera2

Ejercicios

1. Realice un programa que pida la edad de una persona en años. Si la edad es mayor o igual a 18, el programa debe imprimir la cadena: 'ADULTO'. Si la edad es menor a 18 se debe imprimir: 'MENOR DE EDAD'.
2. Diseñe un algoritmo que determine si un número es o no es, par positivo.
3. Diseñe un algoritmo que lea un número de tres cifras y determine si es igual al revés del número.

Escribe aquí tu código

```
#ciclo for
print("Ciclo for en un rango de datos: ")
for i in range(2,10):
    print(i)

print('Ciclo for en una colección de datos:')
x=[3,4,5,7,8,"Hola"]
for i in x:
    print("El elemento es= ",i)
for i in range(len(x)):
    print("El elemento de la posición",i,'es:',x[i])
```

Ciclo for en un rango de datos:

2
3
4
5
6
7
8
9

Ciclo for en una colección de datos:

El elemento es= 3
El elemento es= 4
El elemento es= 5
El elemento es= 7
El elemento es= 8
El elemento es= Hola
El elemento de la posición 0 es: 3
El elemento de la posición 1 es: 4
El elemento de la posición 2 es: 5
El elemento de la posición 3 es: 7
El elemento de la posición 4 es: 8
El elemento de la posición 5 es: Hola

```
print('Ciclo for en un diccionario:')
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',
for nombre, color in frutas.items():
    print (nombre, "es de color", color)
for llave in frutas:
    print(llave, 'es de color', frutas[llave])
```

```
Ciclo for en un diccionario:  
Fresa es de color roja  
Limon es de color verde  
Papaya es de color naranja  
Manzana es de color amarilla  
Guayaba es de color rosa  
Fresa es de color roja  
Limon es de color verde  
Papaya es de color naranja  
Manzana es de color amarilla  
Guayaba es de color rosa
```

```
frutas
```

```
{'Fresa': 'roja',  
 'Limon': 'verde',  
 'Papaya': 'naranja',  
 'Manzana': 'amarilla',  
 'Guayaba': 'rosa'}
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla',  
frutas
```

```
{'Fresa': 'roja',  
 'Limon': 'verde',  
 'Papaya': 'naranja',  
 'Manzana': 'amarilla',  
 'Guayaba': 'rosa'}
```

```
len(frutas)
```

```
5
```

```
frutas.get('Fresa')
```

```
'roja'
```

```
for i in range(10):
    print(i)
```

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

```
v=[ 'a' , 'b' , 'c' , 'd' ]

for i in range(len(v)):
    print(i,"-->",v[i])
```

```
0 --> a
1 --> b
2 --> c
3 --> d
```

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

```
a
b
c
d
```

```
#ciclo while
cont=0
while cont<10:
    print(cont)
    cont+=2
```

```
0  
2  
4  
6  
8
```

```
v=['a','b','c','d']
```

```
for i in range(len(v)):  
    print(i,v[i])
```

```
0 a  
1 b  
2 c  
3 d
```

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

```
a  
b  
c  
d
```

Ejercicio

- Realice un programa que lea las notas de un estudiante, despues devolver el promedio, la mejor y la peor nota.

```
# Escribe aquí tu código
```

Algoritmo Multiplicación 1

Multiplica los numeros naturales **x** y **w** **w**, sumando **x** veces el número número; el resultado lo asigna a la variable **z**

Nociones fundamentales

Multiplicar dos números naturales x y y , y denotar su producto por z

Paso 1: $z := 0;$

$u := x; \quad \rightsquigarrow x = 5$

Paso 2: **repetir las instrucciones**

$z := z + y; \quad \rightsquigarrow y = 13$

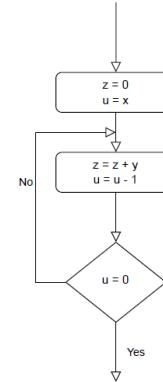
$u := u - 1;$

hasta que $u = 0$

Valor de las variables

Paso	z	u
1	0	5
2	13	4
2	26	3
2	39	2
2	52	1
2	65	0

Multiplicación de dos números naturales x y y



Pedimos introducir el primer número para multiplicar

```
x=int(input('Escribir un numero para multiplicar: '))
```

Escribir un numero para multiplicar: 6

Pedimos introducir el segundo número para multiplicar

```
w=int(input('Escribir otro numero para multiplicar: '))
```

Escribir otro numero para multiplicar: 5

Inicialización y asignación de variable

```
z=0
```

```
u=x
```

Realizamos la operación de multiplicación con sumas y estructuras repetitivas

```
while u>0:
    z=z+w
    u=u-1
```

Mostramos el resultado

```
print('El resultado de la multiplicacion es: ', z)
```

El resultado de la multiplicacion es: 30

Condensando todo el algoritmos en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir un numero para multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
paso1=1
z=0
u=x
paso2=2
print('Paso', ' ', 'Z', ' ', 'U')
print(paso1, ' ', z, ' ', u)
while u>0:
    z=z+w
    u=u-1
    print(paso2, ' ', z, ' ', u)
print('El resultado de la multiplicacion es: ', z)
```

Escribir un numero para multiplicar: 5
 Escribir otro numero para multiplicar: 6
 Paso Z U
 1 0 5
 2 6 4
 2 12 3
 2 18 2
 2 24 1
 2 30 0
 El resultado de la multiplicacion es: 30

Algoritmo División 1

Divide dos números naturales **x** y **w**, utilizando restas; el cociente es asignado a la variable **q** y el residuo a **r**.

Dividir el número natural x por el número natural y , y denotar por q el cociente y el residuo por r .

Teorema del cociente y del residuo:

Dado cualquier entero a y un entero positivo b existen dos enteros únicos q y r tales que:

$$a = q * b + r$$

donde $0 \leq r \leq b$. q se llama el cociente y r el residuo.

Notación: $q = a \text{ div } b$ y $r = a \text{ mod } b$

$$a = (a \text{ div } b) * b + (a \text{ mod } b)$$

División entera de x entre y : el cociente q es el número de veces que se puede restar (sustraer) el divisor y al dividendo x de tal forma que el resultado (cociente) sea no negativo.

Dividir el número natural x por el número natural y , y denotar por q el cociente y el residuo por r .

Paso 1: $q := 0;$

$$r := x; \quad \rightsquigarrow x = 100$$

Paso 2: mientras que $r \geq y$ repetir

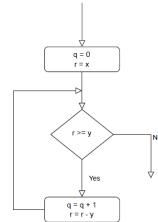
$$q := q + 1;$$

$$r := r - y; \quad \rightsquigarrow y = 15$$

Valor de las variables

Paso	q	r
1	0	100
2	1	85
2	2	70
2	3	55
2	4	40
2	5	25
2	6	10

División de dos números naturales x y y



Pedimos introducir el dividendo de la división

```
x=int(input('Escribir el dividendo: '))
```

Escribir el dividendo: 9

Pedimos introducir el divisor de la división

```
w=int(input('Escribir el divisor: '))
```

Escribir el divisor: 2

Inicialización y asignación de variable

```
q=0
r=x
```

Realizamos la operación de división con restas y estructuras repetitivas

```
while r>=w:
    q=q+1
    r=r-w
```

Mostramos el cociente y el residuo

```
print('El cociente de la división es: ',q,'\\n El residuo de la división es: ',r)
```

```
El cociente de la división es: 4
El residuo de la división es: 1
```

Condensando todo el algoritmo en una sola celda y mostrando prueba de escritorio

```
x=int(input('Escribir el dividendo: '))
w=int(input('Escribir el divisor: '))
paso1=1
q=0
r=x
paso2=2
print('Paso', ' ', 'q', ' ', 'r')
print(paso1, ' ', q, ' ', r)
while r>=w:
    q=q+1
    r=r-w
    print(paso2, ' ', q, ' ', r)
print('El cociente de la división es: ',q, '\n El residuo de la división es: ',r)
```

```
Escribir el dividendo: 9
Escribir el divisor: 2
Paso   q   r
1     0   9
2     1   7
2     2   5
2     3   3
2     4   1
El cociente de la división es: 4
El residuo de la división es: 1
```

Python es un lenguaje **indentado**, no usa corchetes para delimitar el alcance de las estructuras de programación sino que se fija en los **cambios de indentación**.

No se declara el tipo de los argumentos de las funciones. La semántica de la implementación ha de estar preparada para funcionar con los tipos de datos que quieras.

```
import numpy as np
def funcion_1(a,b):
    r = a**2
    return r+b

def greatest(a,b):
    if a>b:
        return a
    else:
        return b
```

```
funcion_1 (10.4,2)
```

```
110.16000000000001
```

```
funcion_1 (10.4, np.array([2,4]))
```

```
array([110.16, 112.16])
```

```
funcion_1(np.array([1,5]),np.array([3,2]))
```

```
array([ 4, 27])
```

```
m1 = np.array([[3,4],[1,1]])  
m2 = np.array([[5,6],[0,0]])
```

```
funcion_1 (m1,m2)
```

```
array([[14, 22],  
       [ 1,  1]])
```

```
greatest(10,2)
```

```
10
```

Podemos definir valores por defecto para los argumentos de las funciones y llamarlas usando explícitamente el nombre de los argumentos. Además, las funciones pueden devolver varios valores.

```
def f_power(x, p=2):  
    return x**p
```

```
f_power(x=3)
```

9

```
f_power(p=4, x=3)
```

81

```
def f_power(x, p=2):  
    return x**p, x*p
```

```
r = f_power(p=4, x=3)  
print(r[0],r[1],"\n")
```

81 12

```
r1, r2 = f_power(p=4, x=3)  
print(r1)  
print(r2, "\n")
```

81

12

```
a,b = 10, np.array([10,4,-3])  
print(a)  
print(b)
```

10

[10 4 -3]

```
def f_power(x, p=2):  
    return x**p  
  
def f_powers(x, p1=2, p2=3):  
    return x**p1, x**p2
```

```
f_power(4)
```

```
16
```

```
f_power(4,3)
```

```
64
```

```
f_powers(4, p1=3)
```

```
(64, 64)
```

```
xp1, xp2 = f_powers(4, p2=4, p1=3)
print("power1",xp1, "power2", xp2)
```

```
power1 64 power2 256
```

Dependiendo del tipo de dato que enviamos a la **función**, podemos diferenciar dos comportamientos:

Paso por valor: Se crea una copia local de la variable dentro de la función.

Paso por referencia: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

Los tipos simples se pasan por valor: Enteros, flotantes, cadenas, lógicos...

Los tipos compuestos se pasan por referencia: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

```
def doblar_valor(x):
    x = 2*x
    return x
```

```
x=10
doblar_valor(x)
print(x,doblar_valor(x))
```

10 20

Para modificar los tipos simples podemos devolverlos modificados y reasignarlos:

```
x = 10
x = doblar_valor(x)
print(x)
```

20

Ejemplo de paso por referencia

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

```
def doblar_valores(y):
    for i,n in enumerate(y):
        y[i] = 2*y[i]
```

```
y=[1,2,3]
```

```
doblar_valores(y)
```

```
print(y)
```

[2, 4, 6]

Algoritmo multiplicación 2

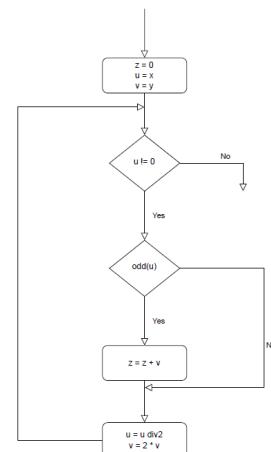
Otro algoritmo para la multiplicación de dos números naturales

Paso 1: $z := 0; u := x; v := y;$ $\rightsquigarrow x = 9, y = 5$
 Paso 2: mientras que $u \neq 0$ repetir
 si $\text{odd}(u)$ entonces $z := z + v;$
 $u := u \text{ div } 2;$
 $v := 2 * v$

Valor de las variables			
Paso	z	u	v
1	0	9	5
2	5	4	10
2	5	2	20
2	5	1	40
2	45	0	80

$$9 = 2 * 2 * 2 + 1 \quad 2(2(2(5))) + 5 = 45$$

Otro algoritmo para la multiplicación de dos números naturales



```
# función para determinar el cociente
def funcionCociente(x,w):
```

```
    q=0
    r=x
    while r>=w:
        q=q+1
        r=r-w
    return q
```

```
# función para determinar el residuo
def funcionResiduo(x,w):
```

```
    q=0
    r=x
    while r>=w:
        q=q+1
        r=r-w
    return r
```

```
# función para determinar si un numero es par
def funcionPar(u):
```

```
    if funcionResiduo(u,2)==0:
        par=True
    else:
        par=False
    return par
```

```
def funcionMultiplicar(x,w):
    z=0
    u=x
    v=w
    while u!=0:
        if not funcionPar(u):
            z=z+v
        u=funcionCociente(u,2)
        v=2*v
    return z
```

```
x=int(input('Escribir un numero par multiplicar: '))
w=int(input('Escribir otro numero para multiplicar: '))
print('El resultado de la mutiplicación es: ', funcionMultiplicar(x,w))
```

Escribir un numero par multiplicar: 6
 Escribir otro numero para multiplicar: 5
 El resultado de la mutiplicación es: 30

Catching

```
int(3.2)
```

3

```
int("hola")
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-2-62025142aa33> in <cell line: 1>()
----> 1 int("hola")

ValueError: invalid literal for int() with base 10: 'hola'
```

Manejar cualquier tipo de error que pase en un bloque de código

```
def ejemplo_try_except(valor):
    try:
        resultado = 10 / valor
        print("El resultado es:", resultado)
    except Exception as e:
        print("Ha ocurrido un error:", e)

# Ejemplos de llamadas a la función
ejemplo_try_except(2)
ejemplo_try_except(0)
ejemplo_try_except('cadena')
```

El resultado es: 5.0
 Ha ocurrido un error: division by zero
 Ha ocurrido un error: unsupported operand type(s) for /: 'int' and 'str'

Manejando un tipo específico de error

```
def int_times_two(x):
    try:
        return(int(x)*2)
    except ValueError:
        return 0
```

int_times_two(2)

4

int_times_two(2.5)

4

int_times_two("hola")

0

Manejo de la sentencia finally para que, pase o no pase un error, siempre ejecute lo que se coloque en la sentencia finally

```
def division_segura(dividendo, divisor):
    try:
        resultado = dividendo / divisor
        return resultado
    except ZeroDivisionError:
        print("¡Error! No puedes dividir entre cero.")
        return None
    finally:
        print("Operación finalizada.")
```

```
dividendo = 10
divisor = 2

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

Operación finalizada.
El resultado de la división es: 5.0

```
dividendo = 10
divisor = 0

resultado_division = division_segura(dividendo, divisor)

if resultado_division is not None:
    print("El resultado de la división es:", resultado_division)
```

¡Error! No puedes dividir entre cero.
Operación finalizada.

Raising

Este tipo de sentencias se utilizan para generar errores personalizados dentro del código

```
def get_positive_integer_from_user():
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
    print ("thanks!")
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: 1
thanks!
```

```
get_positive_integer_from_user()
```

```
Enter a positive integer: -1
```

```
-----
ValueError                                                 Traceback (most recent call last)
<ipython-input-13-ea5635cae962> in <cell line: 1>()
----> 1 get_positive_integer_from_user()

<ipython-input-11-a7d1640efb92> in get_positive_integer_from_user()
      2     a = int(input("Enter a positive integer: "))
      3     if a <= 0:
----> 4         raise ValueError("That is not a positive number!")
      5     print ("thanks!")

ValueError: That is not a positive number!
```

Ejecución de código en múltiples lenguajes de programación

Contenido

- Ejecutar código en C
- Ejecutar código en C++
- Ejecutar código en R

Ejecutar código en C

El Hola Mundo

```
# Versión de GCC
!gcc --version
```

```
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
%%writefile welcome.c
#include <stdio.h> /* incluye biblioteca donde se define E/S */
int main( ){
    printf("Hola mundo en C");
    return 0;
}
```

Writing welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Hola mundo en C

Condicionales en C

```
# Uso de if, else, else if
%%writefile welcome.c

#include <stdio.h>
int main( ){
    int myNum = 10; // Is this a positive or negative number?

    if (myNum > 0) {
        printf("The value is a positive number.");
    } else if (myNum < 0) {
        printf("The value is a negative number.");
    } else {
        printf("The value is 0.");
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

The value is a positive number.

```
# Uso de switch
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int day = 4;

    switch (day) {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
            break;
        case 4:
            printf("Thursday");
            break;
        case 5:
            printf("Friday");
            break;
        case 6:
            printf("Saturday");
            break;
        case 7:
            printf("Sunday");
            break;
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

Thursday

Ciclos en C

```
# Uso de While
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int i = 0;

    while (i < 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
./welcome
```

```
0
1
2
3
4
```

```
# Uso de do while
%%writefile welcome.c
#include <stdio.h>
int main( ){

    int i = 0;

    do {
        printf("%d\n", i);
        i++;
    }
    while (i < 5);

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3  
4
```

```
# Uso de for  
%%writefile welcome.c  
#include <stdio.h>  
int main( ){  
  
    int i;  
  
    for (i = 0; i < 5; i++) {  
        printf("%d\n", i);  
    }  
  
    return 0;  
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3  
4
```

```
# Uso de for anidado
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int i, j;

    // Outer loop
    for (i = 1; i <= 2; ++i) {
        printf("Outer: %d\n", i); // Executes 2 times

        // Inner loop
        for (j = 1; j <= 3; ++j) {
            printf(" Inner: %d\n", j); // Executes 6 times (2 * 3)
        }
    }

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Outer: 1
    Inner: 1
    Inner: 2
    Inner: 3
Outer: 2
    Inner: 1
    Inner: 2
    Inner: 3
```

```
# Uso de break en un while
%%writefile welcome.c
#include <stdio.h>
int main( ){
    int i = 0;

    while (i < 10) {
        if (i == 4) {
            break;
        }
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
0  
1  
2  
3
```

Funciones en C

```
# Crear una función y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction() {
    printf("I just got executed!\n");
}

int main( ){
    myFunction(); // Llamar a la función
    myFunction(); // Llamar a la función por segunda vez
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

I just got executed!
I just got executed!

```
# Crear una función con entrada de Múltiples parámetros y utilizarla
%%writefile welcome.c
#include <stdio.h>

void myFunction(char name[], int age) {
    printf("Hello %s. You are %d years old.\n", name, age);
}

int main() {
    myFunction("Liam", 3);
    myFunction("Jenny", 14);
    myFunction("Anja", 30);
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Hello Liam. You are 3 years old.  
Hello Jenny. You are 14 years old.  
Hello Anja. You are 30 years old.
```

```
# Crear una función que Retorna un resultado y utilizarla  
%%writefile welcome.c  
#include <stdio.h>  
  
int myFunction(int x) {  
    return 5 + x;  
}  
  
int main() {  
    printf("Result is: %d", myFunction(3));  
    return 0;  
}
```

```
Overwriting welcome.c
```

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
Result is: 8
```

Matrices en C

```
# Crear un Array 1D (Vector) y recorrerlo usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
    int myNumbers[] = {25, 50, 75, 100};
    int i;

    for (i = 0; i < 4; i++) {
        printf("%d\n", myNumbers[i]);
    }
    return 0;
}
```

Overwriting welcome.c

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
25
50
75
100
```

```
# Crear un Array 2D (Matriz) y recorrerla usando un for
%%writefile welcome.c
#include <stdio.h>

int main() {
    int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

    int i, j;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d\n", matrix[i][j]);
        }
    }
    return 0;
}
```

Overwriting welcome.c

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

```
!gcc welcome.c -o welcome
```

```
!./welcome
```

```
1
4
2
3
6
8
```

Ejecutar código en C++

```
%%writefile welcome.cpp
#include <iostream>
int main()
{
    std::cout << "Hola mundo en C++";
    return 0;
}
```

Writing welcome.cpp

```
%%bash
g++ welcome.cpp -o welcome
```

```
%%bash
./welcome
```

```
Hola mundo en C++
```

Ejecutar código en R

```
# Instalar versión compatible  
!pip install rpy2==3.5.1
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pu  
Collecting rpy2==3.5.1  
  Downloading rpy2-3.5.1.tar.gz (201 kB)  
           201.7/201.7 KB 15.2 MB/s eta 0:00:00  
?25h Preparing metadata (setup.py) ... ?25l?25hdone  
Requirement already satisfied: cffi>=1.10.0 in /usr/local/lib/python3.8/dist-packages  
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: pytz in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: tzlocal in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from rpy2==3.5.1)  
Building wheels for collected packages: rpy2  
  Building wheel for rpy2 (setup.py) ... ?25l?25hdone  
    Created wheel for rpy2: filename=rpy2-3.5.1-cp38-cp38-linux_x86_64.whl size=318371 s  
    Stored in directory: /root/.cache/pip/wheels/6b/40/7d/f63e87fd83e8b99ee837c8e3489081  
Successfully built rpy2  
Installing collected packages: rpy2  
  Attempting uninstall: rpy2  
    Found existing installation: rpy2 3.5.5  
    Uninstalling rpy2-3.5.5:  
      Successfully uninstalled rpy2-3.5.5  
Successfully installed rpy2-3.5.1
```

```
# activate R magic  
%load_ext rpy2.ipython
```

```
!R --version
```

```
R version 4.2.2 Patched (2022-11-10 r83330) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
https://www.gnu.org/licenses/.
```

```
%%R
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

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

Matrices y vectores

Contenido

- Matrices y vectores
- Numpy
- Generación de matrices y vectores
- Operaciones vectorizadas
- Más operaciones vectorizadas
- Expresiones compactas / Comprehensions
- Matplotlib
- Imágenes

Buscar un elemento en un arreglo

Dado un arreglo con componentes $A[1], \dots, A[n]$ y un valor x , el siguiente algoritmo retorna en la variable q el valor *false* si no existe k tal que $A[k] = x$; en caso contrario, retorna en la variable q el valor *true* y en la variable i el valor k .

$\{n > 0\}$
 $i : 0..n; q : boolean;$
 $A : array [1..n] of T;$
 $\{\text{asignarle valores a las componentes de } A\}$

Paso 1: $i := 0;$
Paso 2: **repetir**
 $i := i + 1;$
 $q := A[i] = x$
 hasta que $q \vee (i = n)$

```
import numpy as np
```

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
q=True
n=len(A)
while q and (i!=n) :
    q=not(A[i]==x)
    i=i+1
if i==n and q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',i-1,'El valor es:',A[i-1])
```

la posición es: 4 El valor es: e

```
A=np.array(['a','b','c','d','e'])
x='e'
i=0
n=len(A)
while i!=n:
    if A[i]==x:
        break
    i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 4 El valor es: e

Otra forma de buscar un elemento en un arreglo: (centinela)

Permite simplificar la condición de terminación del anterior algoritmo.

- Se le adiciona una componente al arreglo A .
- A la nueva componente se le asigna el valor x , que actua como **centinela** para la finalización de la búsqueda.

$$\{c = n + 1, n > 0\}$$

$$i : 0..c;$$

$$A : \text{array}[1..c] \text{ of } T;$$

Paso 1: $i := 0;$
 $A[c] := x;$

Paso 2: **repetir**
 $i := i + 1;$
hasta que $A[i] = x;$

```
A=np.array(['a','b','c','d','e'])
x='c'
n=len(A)
A=np.append(A,x)

i=0
while not(A[i]==x):
    i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

la posición es: 2 El valor es: c

Buscar un elemento en un arreglo ordenado

En este caso las componentes del arreglo están ordenadas, por ejemplo en orden creciente, es decir $A[i] < A[j]$ para todo $i < j$.

El siguiente algoritmo tiene en cuenta que el arreglo está ordenado.

```

i, j, k : integer; q : boolean;
{n > 0}
A : array [1..n] of T;
Paso 1: i := 1; j := n; q := false
Paso 2: repetir
          k := (i + j) div 2;
          si A[k] = x entonces q := true sino
              si A[k] < x entonces i = k + 1 sino j := k - 1;
hasta que q  $\vee (i > j)$ 
```

Busqueda binaria

```

import numpy as np
A=np.array([2,3,5,6,8,10,11])
x=11
i=0
j=len(A)
q=True
while q and (i<=j):
    k=(i+j)//2
    if A[k]==x:
        q=False
    else:
        if A[k]<x:
            i=k+1
        else:
            j=k-1
if q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',k,'El valor es:',A[k])
```

la posición es: 6 El valor es: 11

Producto escalar o producto interno

Dados las sucesiones de números (x_1, \dots, x_n) y (y_1, \dots, y_n) calcular el producto escalar

$$s = x_1 * y_1 + x_2 * y_2 + \dots + x_n * y_n = \sum_{i=1}^n x_i * y_i$$

Teniendo en cuenta la definición de esta suma en términos de la relación de recurrencia:

$$s_i = s_{i-1} + x_i * y_i, \quad s_0 = 0$$

se tiene el siguiente programa,

```
s : real; i : integer;
x, y : array [1..n] of real;
Paso 1: s := 0; i := 0;
Paso 2: repetir
          i := i + 1;
          s := s + x[i] * y[i];
      hasta que i = n;
```



```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
while i!=n:
    s=s+v1[i]*v2[i]
    i+=1
print('el producto escalar es: ',s)
```

el producto escalar es: 60

Sentencia for

Otro algoritmo para hallar el producto escalar de dos sucesiones de números (x_1, \dots, x_n) y (y_1, \dots, y_n) es:

```
s : real; i : 1..n;  
x, y : array [1..n] of real;  
s := 0;  
para i := 1 hasta n hacer  
  s := s + x[i] * y[i];
```

```
v1=[1,2,3,4]  
v2=[4,5,6,7]  
n=len(v1)  
s=0  
i=0  
for i in range(n):  
    s=s+v1[i]*v2[i]  
print('el producto escalar es: ',s)
```

```
el producto escalar es: 60
```

Encontrar el elemento máximo

Encontrar en un arreglo A el índice j tal que $x_j = \max(x_m, \dots, x_n)$.

$j, k : m..n;$
 $x : \text{array } [m..n] \text{ of } T;$
Paso 1: $j := m;$
Paso 2: **para** $k := m + 1$ **hasta** n **hacer**
 si $x[k] > x[j]$ **entonces** $j := k$

```
A=[1,2,4,5,3,6,7]
j=0
k=0
for k in range(1,len(A)):
    if A[k]>A[j]:
        j=k
print('el elemento maximo es:',A[j], 'en la posición: ',j)
```

```
el elemento maximo es: 7 en la posición:  6
```

Sentencias for anidadas

Ejemplo:

Dado un entero positivo n , calcular la suma $1^1 + 2^2 + \dots + n^n$.

```

x, i : 1..n;
s, potencia : integer;
s := 0;
para x := 1 hasta n hacer
    potencia := 1;
    para i := 1 hasta x hacer
        potencia = potencia * x;
    s := s + potencia;

```

```

n=3
s=0
for x in range(1,n+1):
    potencia=1
    for i in range(1,x+1):
        potencia=potencia*x
    s=s+potencia
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

```

n=3
s=0
for x in range(1,n+1):
    s=s+x**x
print('el resultado de la suma es: ',s)

```

el resultado de la suma es: 32

Algoritmo para el ordenamiento de un arreglo

```

 $h, j, k : 1..n;$ 
 $x : \text{array } [1..n] \text{ of } T;$ 
 $u : T;$ 
para  $h := 1$  hasta  $n - 1$  hacer
     $j := h;$ 
    para  $k := h + 1$  hasta  $n$  hacer
        si  $x[k] > x[j]$  entonces  $j := k$ 
         $u := x[h];$ 
         $x[h] := x[j]$ 
         $x[j] := u$ 

```

Complejidad del algoritmo:

$$(n - 1) + (n - 2) + \dots + 2 + 1 = \frac{n}{2}(n - 1).$$

```

x=np.random.randint(200, size=100)
h=0
for h in range(len(x)):
    j=h
    for k in range(h+1,len(x)):
        if x[k]>x[j]:
            j=k
    u=x[h]
    x[h]=x[j]
    x[j]=u
print(x)

```

198	194	192	187	181	180	179	179	178	177	175	175	172	172	171	168	166	164	163
161	161	160	158	156	154	152	147	146	146	146	146	142	140	135	135	130	128	126
124	120	119	118	118	116	112	110	107	102	101	101	100	100	95	89	88	86	
84	83	82	82	82	81	81	80	77	76	73	73	72	71	69	68	67	66	
61	61	57	55	49	45	42	41	41	40	39	36	35	35	29	23	22	21	
18	17	15	13	12	12	10	8	6	0									

Con la librería `numpy` se trabaja con matrices de forma natural. Fíjate cómo se declaran y cómo descubrimos sus dimensiones.

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

```
import numpy as np

a = np.array([[1,2,3,4,5],
              [5,4,3,2,1],
              [9,8,7,6,5],
              [7,6,5,6,7],
              [2,2,2,3,3],
              [4,3,4,3,4],
              [5,1,1,4,1]]).astype('int32')

print(a, type(a), a nbytes, "\n")
print("a shape", a.shape, "\n")
print("a rows", a.shape[0], "\n")
print("a cols", a.shape[1])
```

```
[[1 2 3 4 5]
 [5 4 3 2 1]
 [9 8 7 6 5]
 [7 6 5 6 7]
 [2 2 2 3 3]
 [4 3 4 3 4]
 [5 1 1 4 1]] <class 'numpy.ndarray'> 140

a shape (7, 5)

a rows 7

a cols 5
```

```
a[0,0]=100
a
```

```
array([[100,    2,    3,    4,    5],
       [  5,    4,    3,    2,    1],
       [  9,    8,    7,    6,    5],
       [  7,    6,    5,    6,    7],
       [  2,    2,    2,    3,    3],
       [  4,    3,    4,    3,    4],
       [  5,    1,    1,    4,    1]], dtype=int32)
```

```
a nbytes
```

140

```
v = np.array([2,3,4,5,6,7,3,12])
print("v shape", v.shape)
print("v elems", v.shape[0])
print(v, type(v))
```

```
v shape (8,)
v elems 8
[ 2  3  4  5  6  7  3 12] <class 'numpy.ndarray'>
```

```
v[3:-1]
```

```
array([5, 6, 7, 3])
```

```
np.min(v), np.max(v)
```

```
(2, 12)
```

Con la notación de índices accedemos a columnas o filas enteras, rangos de columnas o filas, elementos individuales o porciones de una matriz o un vector.

```
print("una fila      ", a[2])
print("una fila      ", a[2,:])
print("una columna    ", a[:,2])
print("un elemento    ", a[2,2])
print("varias filas   \n", a[2:5])
print("varias columnas\n", a[:,1:3])
print("una porcion    \n", a[2:5,1:3])
```

```

una fila      [9 8 7 6 5]
una fila      [9 8 7 6 5]
una columna    [3 3 7 5 2 4 1]
un elemento    7
varias filas
[[9 8 7 6 5]
[7 6 5 6 7]
[2 2 2 3 3]]
varias columnas
[[2 3]
[4 3]
[8 7]
[6 5]
[2 2]
[3 4]
[1 1]]
una porcion
[[8 7]
[6 5]
[2 2]]

```

Muchas funciones de la librería `numpy` operan sobre una matriz completa, o de forma separada por columnas o filas según el valor del argumento `axis`.

```

print(a)
print("suma total", np.sum(a))
print("suma eje 0", np.sum(a, axis=0))
print("suma eje 1", np.sum(a, axis=1))
print("promedio total", np.mean(a))
print("promedio eje 0", np.mean(a, axis=0))
print("promedio eje 1", np.mean(a, axis=1))

```

```

[[100  2   3   4   5]
 [ 5   4   3   2   1]
 [ 9   8   7   6   5]
 [ 7   6   5   6   7]
 [ 2   2   2   3   3]
 [ 4   3   4   3   4]
 [ 5   1   1   4   1]]
suma total 237
suma eje 0 [132  26  25  28  26]
suma eje 1 [114  15  35  31  12  18  12]
promedio total 6.771428571428571
promedio eje 0 [18.85714286  3.71428571  3.57142857  4.          3.71428571]
promedio eje 1 [22.8  3.    7.    6.2  2.4  3.6  2.4]

```

Las matrices en Python pueden tener un número arbitrario de dimensiones y podemos acceder a submatrices en la dirección o dimensión que queramos.

```
z = np.random.randint(-20,20, size=(5,5))
print(z)
```

```
[[ -19   4  -13 -19  13]
 [ 19   6  -9  -9 -16]
 [-11  12  -6 -20  16]
 [ -4  -1  -4  13  0]
 [ -19  13  -1   9  8]]
```

```
np.sum(z, axis=1)
```

```
array([-34,   -9,   -9,    4,   10])
```

```
#m = np.random.randint(10, size=(3,3,3))
print("Matrix 3D completa\n", m)
print("-----\n", m[0,:,:])
print("-----\n", m[1,:,:])
print("-----\n", m[:,0,:])
print("-----\n", m[:,0,1])
print("-----\n", np.mean(m, axis=1))
```

```
Matrix 3D completa
```

```
[[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```
[[8 6 9]
 [0 3 0]
 [0 9 2]]]
```

```
-----
```

```
[[7 0 8]
 [4 0 4]
 [3 4 9]]]
```

```
-----
```

```
[[4 7 9]
 [2 2 6]
 [0 5 9]]]
```

```
-----
```

```
[[7 0 8]
 [4 7 9]
 [8 6 9]]]
```

```
-----
```

```
[0 7 6]
```

```
-----
```

```
[[4.66666667 1.33333333 7.
 [2.          4.66666667 8.
 [2.66666667 6.          3.66666667]]]
```

`m.shape`

(3, 3, 3)

`m`

```
print("matrix identidad\n", np.eye(3))
print("vector de ceros", np.zeros(4))
print("matriz de ceros\n", np.zeros((3,2)))
print("matriz de unos\n", np.ones((2,3)))
print("vector rango", np.arange(10))
print("vector rango", np.arange(5,10))
print("vector espacio lineal", np.linspace(2,12,11))
print("matriz aleatoria según distribución uniforme [0,1]\n", np.random.random(size=(3
print("vector aleatorio de enteros entre 0 y 5", np.random.randint(5, size=10))
```

```

matrix identidad
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
vector de ceros [0. 0. 0. 0.]
matriz de ceros
[[0. 0.]
 [0. 0.]
 [0. 0.]]
matriz de unos
[[1. 1. 1.]
 [1. 1. 1.]]
vector rango [0 1 2 3 4 5 6 7 8 9]
vector rango [5 6 7 8 9]
vector espacio lineal [ 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.]
matriz aleatoria según distribución uniforme [0,1]
[[0.57996022 0.65361661 0.99420518 0.12111156 0.26731741]
 [0.206147 0.25575192 0.66588831 0.55022348 0.76820253]
 [0.51384056 0.63172996 0.50836505 0.41256526 0.66835632]]
vector aleatorio de enteros entre 0 y 5 [0 3 3 0 3 3 2 1 2 1]

```

```

import numpy as np
v = np.array([10,12,13,15,20])
print(v)
print(v+1)
print(v*2)

a = np.random.randint(100, size=5)
print(a)

print(v.dot(a))

```

```

[10 12 13 15 20]
[11 13 14 16 21]
[20 24 26 30 40]
[60 83 36 31 82]
4169

```

```

v1=np.array([2,-1,1])
v2=np.array([-3,1,1])
print(v1.dot(v2))
print(v1*v2)
print(np.cross(v2,v1))
print(np.cross(v1,v2).dot(v1))
print(np.cross(v1,v2).dot(v2))

```

```
-6  
[-6 -1  1]  
[2 5 1]  
0  
0
```

```
m1=np.array([[1,2],[3,4]])
```

```
m2=np.array([[4,8],[9,1]])
```

```
m1*m2
```

```
array([[ 4, 16],  
       [27,  4]])
```

```
m1.dot(m2)
```

```
array([[22, 10],  
       [48, 28]])
```

```
a = np.array([[1,2,3],[4,5,6]])  
b = np.array([[6,5,4],[3,2,1]])  
c = np.array([[1,2],[4,5]])  
print(a)  
print("--")  
print(a.T)  
print("--")  
print(b)  
print("--")  
print(a.T.dot(a))  
print("--")  
print(a*b)  
print("--")  
x=np.array([[2,3],[4,5]])  
z=np.array([[5,6,7],[2,3,1]])  
print(x.dot(z))
```

```
[[1 2 3]
 [4 5 6]]
 --
 [[1 4]
 [2 5]
 [3 6]]
 --
 [[6 5 4]
 [3 2 1]]
 --
 [[17 22 27]
 [22 29 36]
 [27 36 45]]
 --
 [[ 6 10 12]
 [12 10  6]]
 --
 [[16 21 17]
 [30 39 33]]
```

Las operaciones vectorizadas también funcionan con expresiones *booleanas*. Fíjate cómo se indexa un vector con una expresión booleana para seleccionar un conjunto de elementos.

```
a = np.array([1,8,4,10,-4,5])
print("posiciones en a >4:", a>4)
print("elementos de a >4:",a[a>4])
```

```
posiciones en a >4: [False  True False  True False  True]
elementos de a >4: [ 8 10  5]
```

```
a = np.random.randint(100, size=(6,10))
a
```

```
array([[14, 19,  5, 87, 99, 92, 31, 63, 18, 44],
 [81, 26, 65, 86,  2, 24, 99, 79, 71, 37],
 [42, 87,  8, 92, 63, 29, 31, 85, 75, 93],
 [59, 58, 97, 10, 12, 99, 86, 99, 28, 56],
 [76, 76, 75, 56, 49, 33, 51, 58, 41,  4],
 [35, 55, 41,  7, 42, 85, 53, 14, 22, 51]])
```

```
np.mean(a, axis=1)
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
array([47.2, 57. , 60.5, 60.4, 51.9, 40.5])
```

```
%timeit np.mean(a, axis=1)
```

```
10.8 µs ± 2.4 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit np.array([np.mean(a[i,:]) for i in range(a.shape[0])])
```

```
43.7 µs ± 925 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Pregúntate siempre si puedes vectorizar algo y hazlo. Por ejemplo, Para dos matrices (**grandes**).

```
a = np.random.randint(100, size=(1000,100))
b = np.random.randint(200, size=(1000,100))
```

```
# El número de elementos que son iguales
np.mean(a==b)
```

```
0.00512
```

```
# El promedio de los elementos de a que son mayores a su correspondiente posición en b
np.mean(a[a>b])
```

```
66.36604323536555
```

```
# El promedio de los elementos de b que son mayores a su correspondiente posición en a
np.mean(b[b>a])
```

```
122.37378653758725
```

```
# Con matrices más pequeñas
a = np.random.randint(100, size=(10))
b = np.random.randint(200, size=(10))
print (a)
print (b)
```

```
[73 19 71 47 95 38 48 59 74 61]
[ 68  94  69 142 167  40 190  70  33  95]
```

```
# el elemento en b correspondiente a la posición del elemento más grande en a
b[np.argmax(a)]
```

```
167
```

Broadcasting

Normalmente, Numpy necesita que las dimensiones de las matrices coincidan cuando se hacen operaciones con ellas

```
a = np.random.randint(100, size=(3,5))
b = np.random.randint(10, size=(3,4))
print (a)
print (b)
a + b
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86  0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```

-----
ValueError                                Traceback (most recent call last)
<ipython-input-14-860cc8167430> in <cell line: 5>()
      3 print (a)
      4 print (b)
----> 5 a + b

ValueError: operands could not be broadcast together with shapes (3,5) (3,4)

```

Pero numoy intentan expandir las operaciones si alguna de las dimensiones coincide

```
a
```

```
array([[83, 58, 25, 73, 33],
       [91, 73, 70, 60, 18],
       [20, 86, 0, 23, 62]])
```

```
a*10
```

```
array([[830, 580, 250, 730, 330],
       [910, 730, 700, 600, 180],
       [200, 860, 0, 230, 620]])
```

```
# Veamos como funciona el reshape en la siguiente operación.
a + b[:,1].reshape(-1,1)
```

```
array([[85, 60, 27, 75, 35],
       [97, 79, 76, 66, 24],
       [29, 95, 9, 32, 71]])
```

```
b[:,1].reshape(-1,1)
```

```
array([[2],
       [6],
       [9]])
```

```
b[:,1]
```

```
array([2, 6, 9])
```

```
a + b[:,1]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-20-d148e74cc6fe> in <cell line: 1>()
----> 1 a + b[:,1]
```

ValueError: operands could not be broadcast together with shapes (3,5) (3,)

```
# Observa la forma de las filas
a + b.flatten()[:a.shape[1]]
```

```
array([[90, 60, 29, 73, 36],
       [98, 75, 74, 60, 21],
       [27, 88, 4, 23, 65]])
```

```
print (a)
print (b)
```

```
[[83 58 25 73 33]
 [91 73 70 60 18]
 [20 86 0 23 62]]
 [[7 2 4 0]
 [3 6 7 1]
 [7 9 5 2]]
```

```
b.flatten()
```

```
array([7, 2, 4, 0, 3, 6, 7, 1, 7, 9, 5, 2])
```

```
b.flatten()[:a.shape[1]]
```

```
array([7, 2, 4, 0, 3])
```

Fíjate cómo las siguientes expresiones son equivalentes:

```
a=15
if a > 10:
    s = "mayor que 10"
else:
    s = "menor que 10"

print(s)
```

mayor que 10

```
a = 15
s = "mayor que 10" if a > 10 else "menor que 10"
print(s)
```

mayor que 10

```
l=[]
for i in range(5):
    l.append(i)
print(l)
```

[0, 1, 2, 3, 4]

```
l=[i for i in range(5)]
print(l)
```

[0, 1, 2, 3, 4]

```
a = [10, -4, 20, 5]

#o = ["10A", "-4B", "20A", "5A"]

o = []
for i in a:
    if i<0:
        o.append(str(i)+"B")
    else:
        o.append(str(i)+"A")

print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
a = [10, -4, 20, 5]
def convert(x):
    return str(x)+"B" if x<0 else str(x)+"A"

o = [convert(i) for i in a]
print(o)
```

```
['10A', '-4B', '20A', '5A']
```

```
r = []
for i in range(10):
    r.append("el numero "+str(i))
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
r = ["el numero "+str(i) for i in range(10)]
print(r)
```

```
['el numero 0', 'el numero 1', 'el numero 2', 'el numero 3', 'el numero 4', 'el numero
```

```
frutas = {'Fresa':'roja', 'Limon':'verde', 'Papaya':'naranja', 'Manzana':'amarilla', 'Guayaba':'rosa'}
for nombre, color in frutas.items():
    print (nombre, "es de color", color)

print()
r = [nombre+" es de color "+color for nombre, color in frutas.items()]
r
```

```
Fresa es de color roja
Limon es de color verde
Papaya es de color naranja
Manzana es de color amarilla
Guayaba es de color rosa
```

```
[ 'Fresa es de color roja',
  'Limon es de color verde',
  'Papaya es de color naranja',
  'Manzana es de color amarilla',
  'Guayaba es de color rosa']
```

Ejercicio

Construir una matriz aleatoria de cuatro letras (A,T,C,G) de tamaño 100x1000 luego cambiar las letras por las siguientes codificaciones:

Codificación 1: -2,-1,1,2

Codificación 2: 0,1,2,3

Codificación 3: 0001,0010,0100,1000

```
m=np.array([[ 'A', 'T'], ['C', 'G']])
```

m

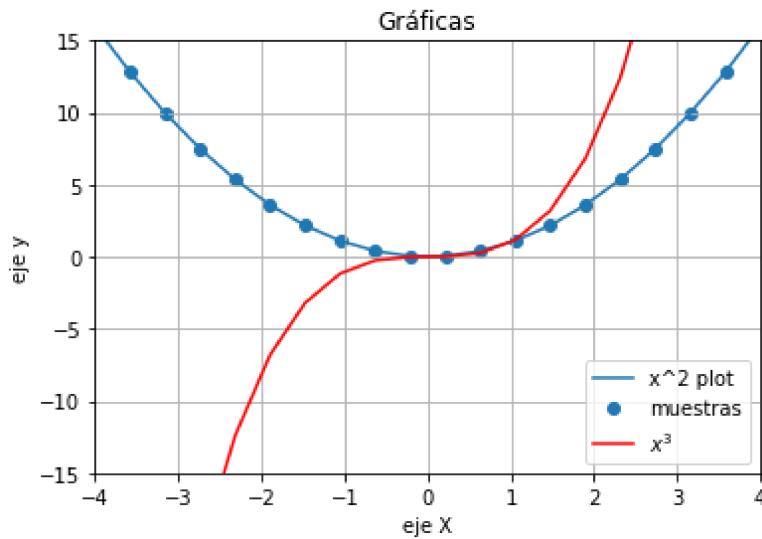
```
array([[ 'A', 'T'],
       ['C', 'G']], dtype='<U1')
```

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

x = np.linspace(-4,4,20)# El vector de tabulación
y=x**2
z=x**3

plt.plot(x, y, label="x^2 plot")
plt.scatter(x, y, label="muestras")
plt.plot(x, z, label="$x^3$", color="red")
plt.xlim([-4,4])
plt.ylim([-15, 15])
plt.legend()
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Gráficas')
```

Text(0.5, 1.0, 'Gráficas')

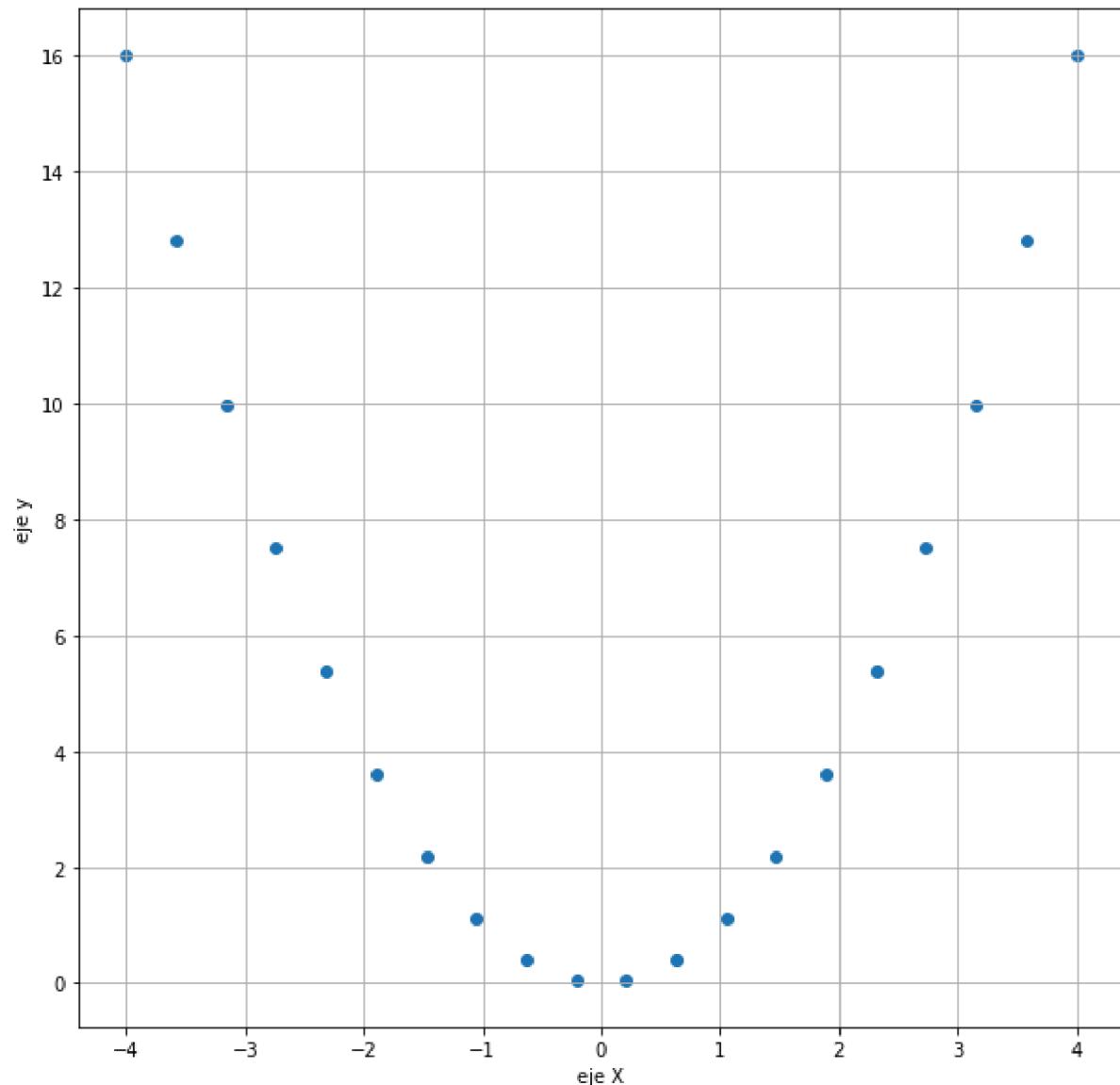


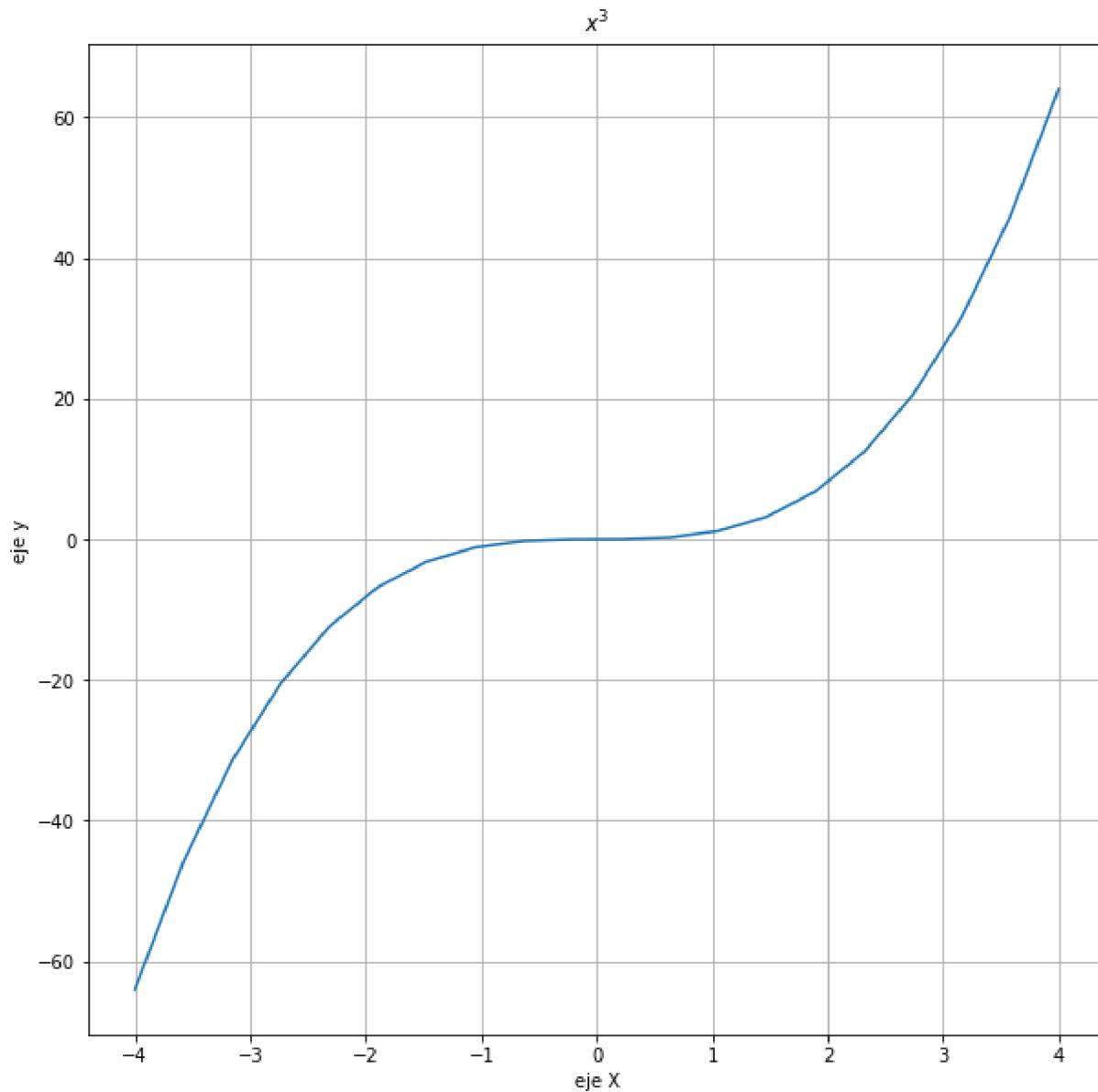
```
x = np.linspace(-4,4,20)
y=x**2

plt.figure(figsize=(10,10))
plt.scatter(x, y)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Parabola')

plt.figure(figsize=(10,10))
plt.plot(x, x**3)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('$x^3$')
```

Text(0.5, 1.0, '\$x^3\$')

Parabola

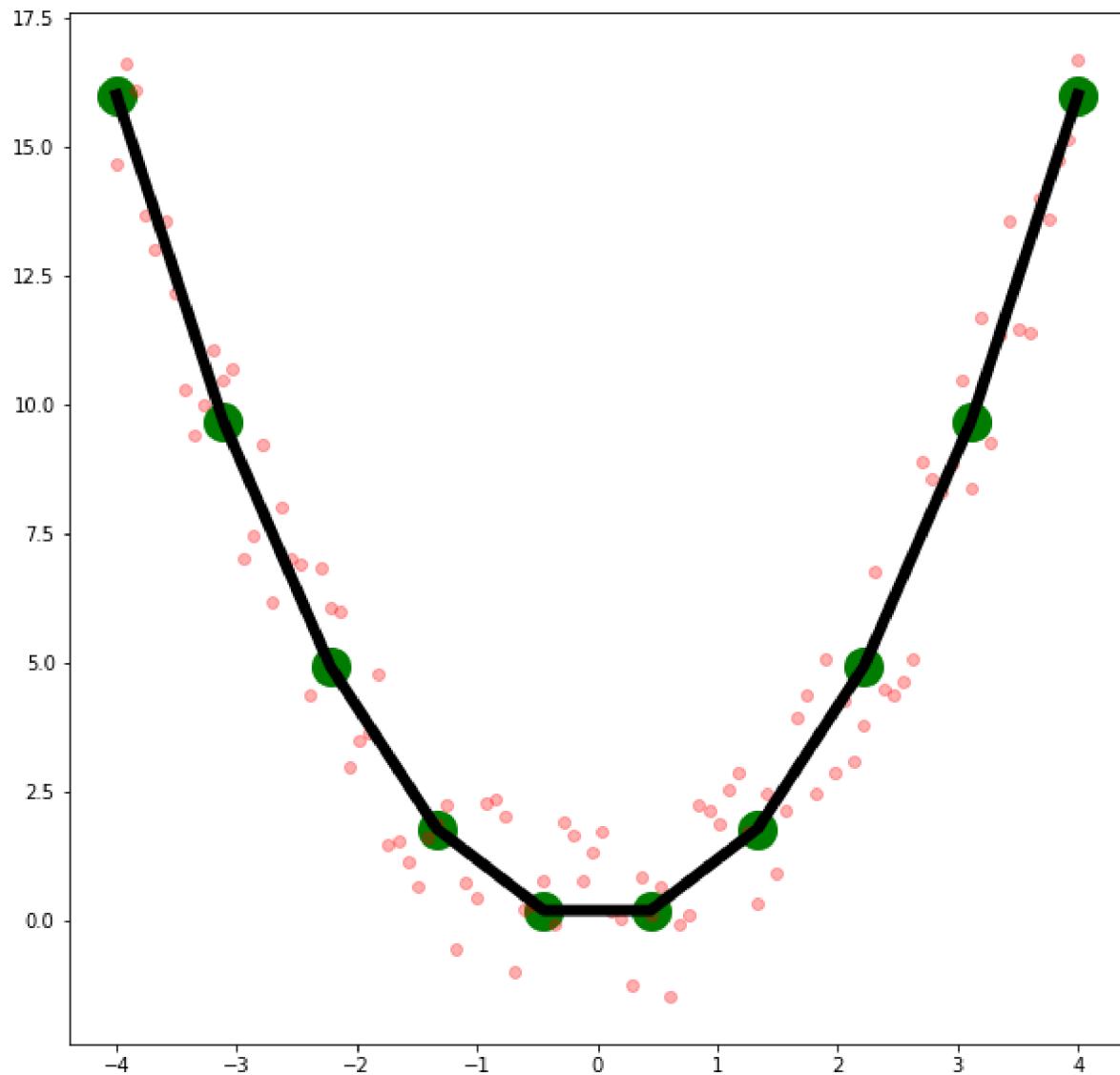


```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.figure(figsize=(10,10))
x = np.linspace(-4,4,10)

plt.plot(x, x**2, color="black", linewidth=6)
plt.scatter(x, x**2, c="green", s=400)

x_r = np.linspace(-4,4,100)
x_ruido = x_r**2 + (np.random.random(x_r.shape)-0.5)*4
plt.scatter(x_r,x_ruido, c="red", alpha=0.3)
```

```
<matplotlib.collections.PathCollection at 0x7ff4845f44f0>
```



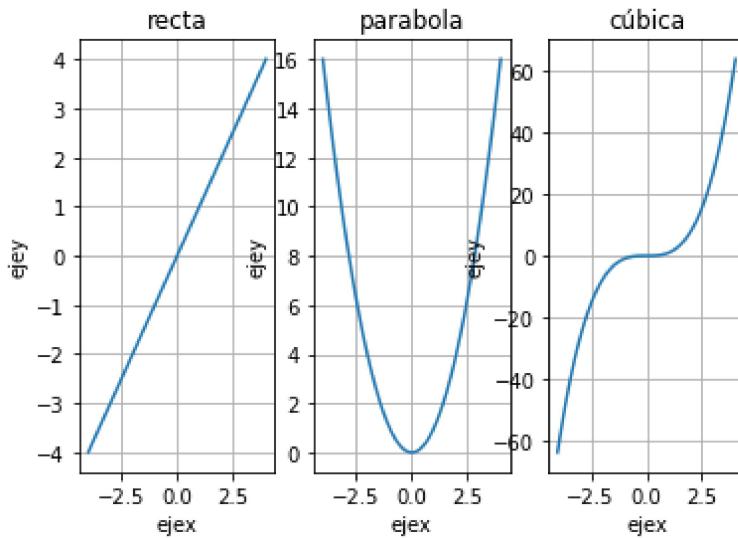
```

x=np.linspace(-4,4,100)
y=x
z=x**2
w=x**3
#plt.figure()
plt.subplot(1,3,1)
plt.plot(x,y)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('recta')
plt.grid()

#plt.figure()
plt.subplot(1,3,2)
plt.plot(x,z)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('parabola')
plt.grid()

#plt.figure()
plt.subplot(1,3,3)
plt.plot(x,w)
plt.xlabel('ejex')
plt.ylabel('ejey')
plt.title('cúbica')
plt.grid()

```



```

from google.colab import drive
drive.mount('/content/drive')

```

Mounted at /content/drive

```
#Rei
#path_data = "/content/drive/MyDrive/Machine Learning 2023-1 UManizales Maestría/Clase
#Arteaga
path_data = "/content/drive/MyDrive/Clases 2023-01/Programación Concurrente y Distribu

path_mesa = path_data+"/Mesa.jpg"
path_chestxray = path_data+"/ChestXRay.jpg"
```

```
from skimage import io
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

#Arteaga
img = io.imread(path_mesa)
img2 = io.imread(path_chestxray)

print("dimensiones", img.shape, "max", np.max(img), "min", np.min(img), "tipo", type(img))

plt.scatter(320,100, marker="x", s=200, linewidth=7, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print("pixel at blue marker ", img[100,320,:])
print("pixel at green marker", img[230,150,:])
img[100,:,:]=0
img[150,:,:]=255
#plt.grid() # remove gridlines
plt.imshow(img)
```

```
dimensiones (246, 360, 3) max 255 min 0 tipo <class 'numpy.ndarray'>
pixel at blue marker [75 0 5]
pixel at green marker [ 24  48 112]
```

<matplotlib.image.AxesImage at 0x7ff45c6ccf10>



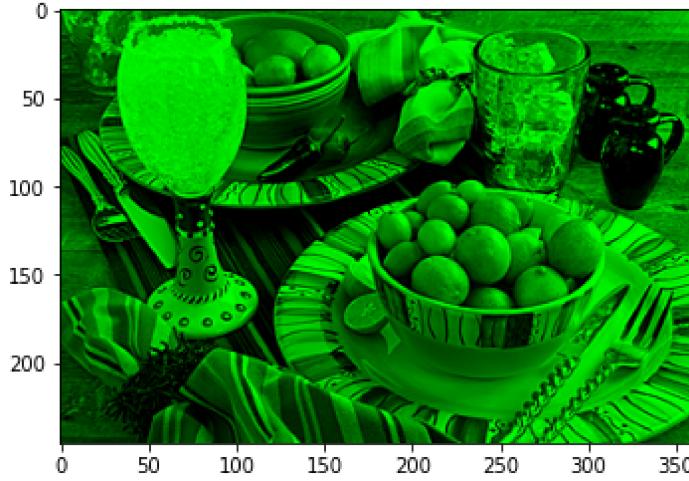
```
array([[[ 90, 108, 132],  
       [224, 254, 254],  
       [ 64, 110, 143],  
       ...,  
       [173, 188, 191],  
       [144, 162, 164],  
       [172, 189, 196]],  
  
      [[ 34,  61,  80],  
       [212, 223, 241],  
       [156, 201, 230],  
       ...,  
       [173, 192, 198],  
       [173, 193, 200],  
       [158, 187, 191]],  
  
      [[ 27,  57,  83],  
       [134, 141, 157],  
       [172, 231, 247],  
       ...,  
       [195, 205, 207],  
       [188, 193, 197],  
       [175, 186, 190]],  
  
      ...,  
  
      [[ 44,  52,  55],  
       [ 36,  45,  52],  
       [ 38,  43,  46],  
       ...,  
       [119, 113,  97],  
       [120, 122, 109],  
       [122, 125, 130]],  
  
      [[ 57,  47,  38],  
       [ 54,  46,  43],  
       [ 55,  52,  47],  
       ...,  
       [109,  99,  87],  
       [113, 110,  91],  
       [105, 108, 101]],  
  
      [[ 58,  56,  57],  
       [ 69,  70,  65],  
       [ 61,  66,  69],  
       ...,  
       [139, 133, 119],  
       [134, 119,  98],  
       [132, 123, 108]]], dtype=uint8)
```

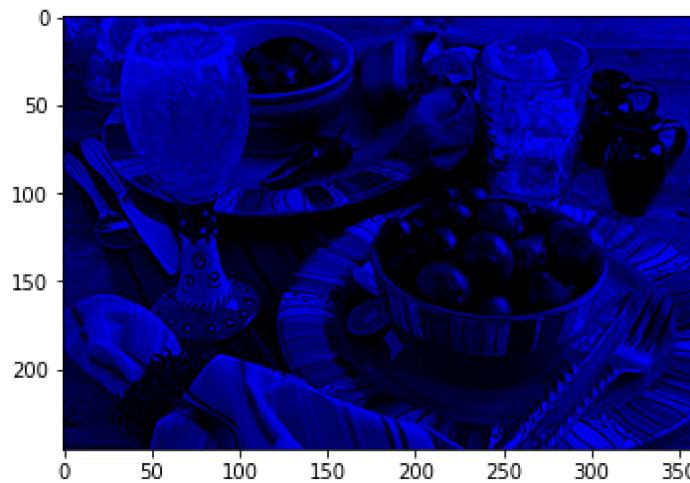
```
img = io.imread(path_mesa)
img[:, :, 1]=0
img[:, :, 2]=0
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 2]=0
plt.figure()
plt.imshow(img)

img = io.imread(path_mesa)
img[:, :, 0]=0
img[:, :, 1]=0
plt.figure()
plt.imshow(img)
```

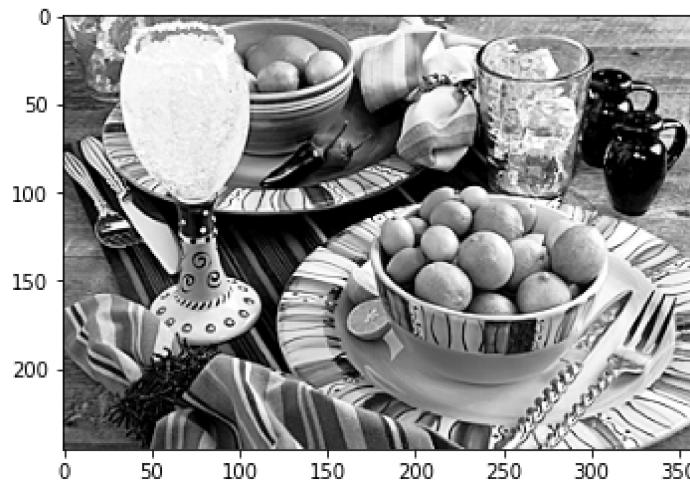
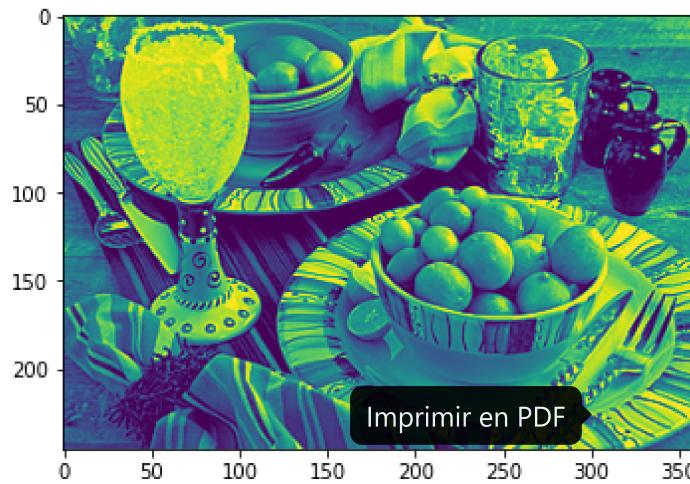
<matplotlib.image.AxesImage at 0x7ff45c4e0310>





```
img = io.imread(path_mesa)
plt.imshow(img[:, :, 1])
plt.figure()
plt.imshow(img[:, :, 1], cmap = plt.cm.Greys_r)
```

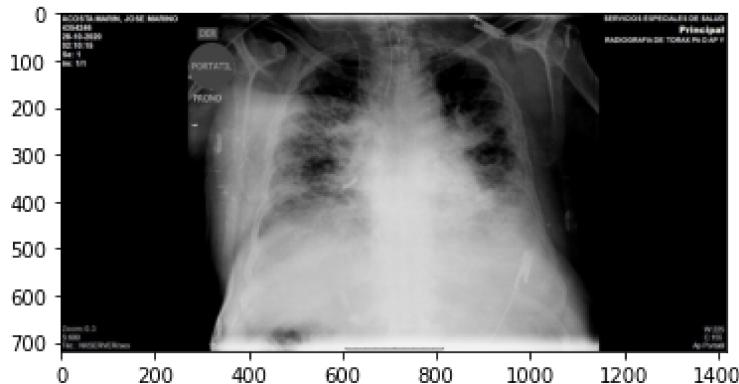
<matplotlib.image.AxesImage at 0x7ff45c3c1700>



```
# Otra imagen
print(img2.shape)
plt.imshow(img2)
```

(718, 1418, 3)

<matplotlib.image.AxesImage at 0x7ff45c3cd940>

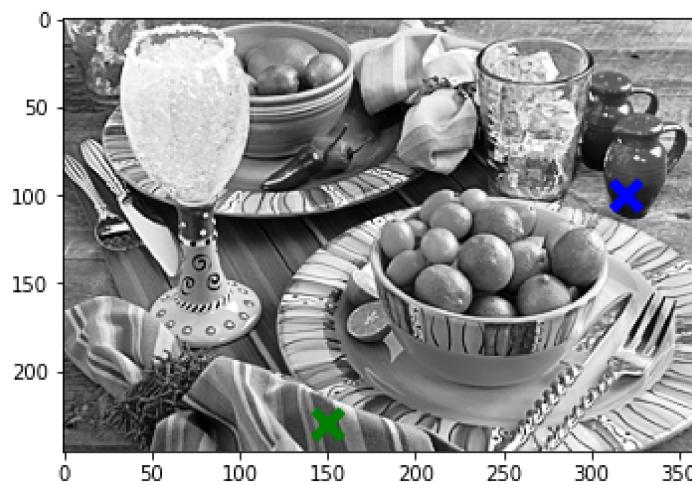


La manipulación de imágenes se *reduce* a realizar cálculos sobre los valores de luminosidad de cada pixel. Por ejemplo, obtenemos una versión en escala de grises promediando los valores RGB de cada pixel. Esto se hace de manera natural con la función `np.mean` y el argumento `axis` adecuado.

```
gimg = np.mean(img, axis=2)
print("dimensiones", gimg.shape, "max", np.max(gimg), "min", np.min(gimg))
plt.scatter(320,100, marker="x", s=200, linewidth=5, c="b")
plt.scatter(150,230, marker="x", s=200, linewidth=5, c="g")
print(img[100,320,:])
print("pixel at blue marker ", gimg[100,320])
print("pixel at green marker", gimg[230,150])
# plt.grid() # remove gridlines
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

dimensiones (246, 360) max 255.0 min 0.0
[75 0 5]
pixel at blue marker 26.666666666666668
pixel at green marker 61.333333333333336

<matplotlib.image.AxesImage at 0x7ff45c182640>



Ejercicio

También podemos acceder a porciones (**parches**) de la imagen usando la notación natural de matrices de Python. Si además, reducimos la luminosidad de cada pixel a la mitad lo que hacemos es oscurecer la imagen.

1. Cargar nuevamente la imagen de la mesa
2. Visualizarla
3. Obtener una nueva imagen promediando los 3 canales RGB
4. Visualizarla
5. Extraer el jarrón de la posición 50:100 y 300:350 (ver imagen anterior)
6. Visualizarla
7. Dividir todos los pixeles del parche extraído (item 5) entre 2
8. Compara los parches (visualiza usando los parámetros: `cmap=>gray<`, `vmin=0`, `vmax=255`), el original y el de división de cada pixel entre 2 ¿Qué observa?

```
# Escribe tu código aquí
gimg = np.mean(img, axis=2)
plt.imshow(gimg, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

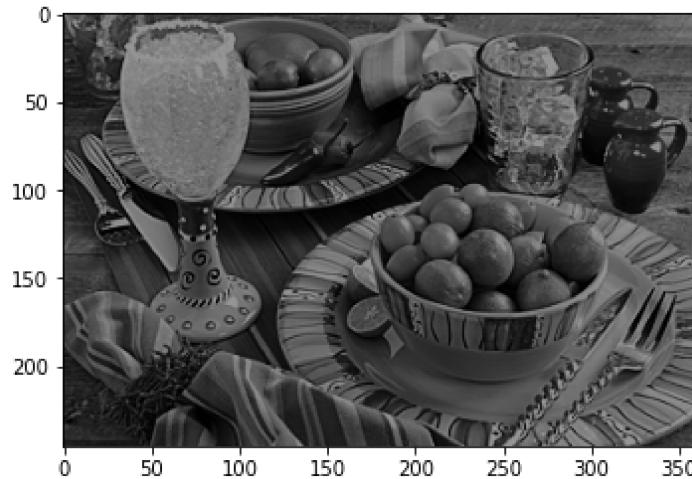
```
<matplotlib.image.AxesImage at 0x7ff45c160760>
```



```
gimg2=gimg/2
```

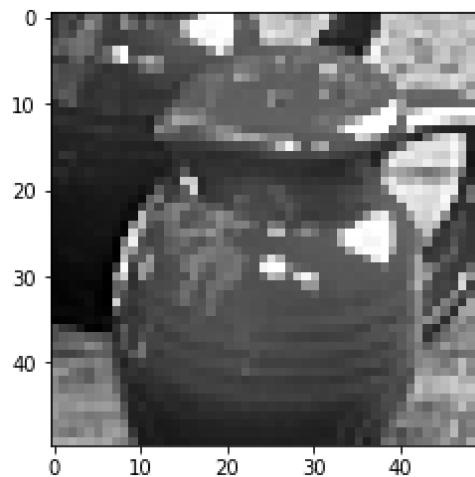
```
plt.imshow(gimg2, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff45c0a1b80>
```



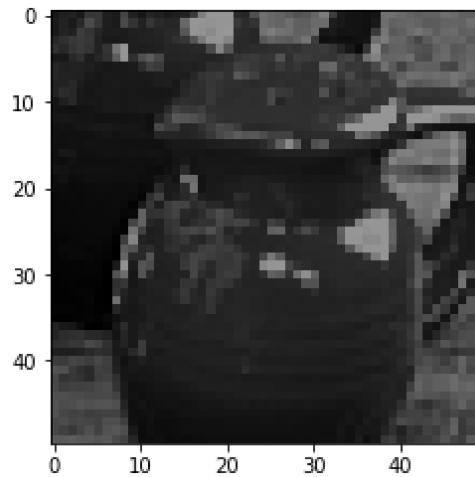
```
jarron=gimg[50:100 , 300:350 ]  
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ecbee80>
```



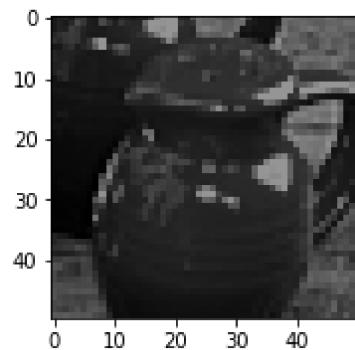
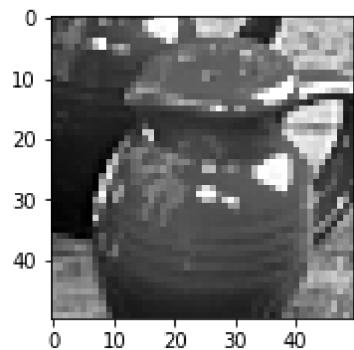
```
jarronOscurito=jarron/2  
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44ec2b160>
```



```
plt.subplot(1,2,1)  
plt.imshow(jarron, cmap = plt.cm.Greys_r, vmin=0, vmax=255)  
plt.subplot(1,2,2)  
plt.imshow(jarronOscurito, cmap = plt.cm.Greys_r, vmin=0, vmax=255)
```

```
<matplotlib.image.AxesImage at 0x7ff44e835fa0>
```



Introducción a Pandas

Contenido

- Series de pandas
- DataFrames de pandas
- Cargar Fuentes externas de datos

Pandas es una biblioteca de Python fundamental para la manipulación y el análisis de datos, ampliamente utilizada en el ámbito de la ciencia de datos y la ingeniería de datos. Proporciona estructuras de datos fáciles de usar y de alto rendimiento, como DataFrames y Series, que permiten realizar operaciones complejas de limpieza, transformación, y análisis de datos con una sintaxis intuitiva y eficiente.

```
#  
# ¿Cómo importamos la librería?  
import pandas as pd
```

Series de pandas

Una pandas Series es una estructura de datos unidimensional en la biblioteca pandas de Python, similar a una columna en una tabla de datos o a un array en otros lenguajes de programación. Cada Series tiene un índice asociado, que son las etiquetas de las filas que permiten acceder a los elementos de la serie de manera rápida y eficiente. Los elementos de una Series pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

```
pd.Series([-2, -3, 0, 3, 2], dtype='int8')
```

↓

		Data
Index	0	-2
	1	-3
	2	0
	3	3
	4	2

dtype: int8

© w3resource.com

Lo que hace que las Series sean especialmente poderosas es su capacidad para manejar datos faltantes y realizar operaciones vectorizadas, lo que significa que puedes aplicar operaciones a todos los elementos de una Series de una sola vez, en lugar de tener que iterar sobre cada elemento individualmente. Además, las Series ofrecen una variedad de métodos y propiedades para realizar operaciones estadísticas, manipulación de datos, y transformaciones, lo que las convierte en una herramienta versátil y esencial para el análisis de datos en Python.

```
# Vamos a crear una serie a partir de un diccionario

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}
```

```
# Para crear una serie se debe partir de un objeto de tipo diccionario
balances = pd.Series(test_balance_data)
balances
```

```

pasan      20.00
treasure   20.18
ashley     1.05
craig      42.42
dtype: float64
```

```
# Acá tenemos una serie de pandas, que es una estructura de datos que se parece a un diccionario.  
# las etiquetas de las filas se obtienen  
balances.keys()  
# tal cual como el un diccionario.
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Alternativamente, se pueden obtener con el método index  
balances.index
```

```
Index(['pasan', 'treasure', 'ashley', 'craig'], dtype='object')
```

```
# Si solo quieres los valores, puedes usar:  
balances.values
```

```
array([20. , 20.18, 1.05, 42.42])
```

```
# Ahora, creemos una serie a partir de una lista
```

```
unlabeled_balances = pd.Series([20.00, 20.18, 1.05, 42.42])  
unlabeled_balances # Note que las etiquetas de las filas son números enteros que van de 0 a 3
```

```
0    20.00  
1    20.18  
2     1.05  
3    42.42  
dtype: float64
```

```
# Si quieres personalizar esos index, puedes hacerlo de la siguiente manera  
labeled_balances = pd.Series(  
    [20.00, 20.18, 1.05, 42.42],  
    index=['pasan', 'treasure', 'ashley', 'craig'])  
labeled_balances
```

```
pasan      20.00
treasure   20.18
ashley     1.05
craig      42.42
dtype: float64
```

```
# Si pasas un escalar, recuerda que es un valor único,
# este se transmitirá a cada una de las claves especificadas en el argumento de palabras
pd.Series(20.0, index=["guil", "jay", "james", "ben", "nick"])
```

```
guil      20.0
jay       20.0
james    20.0
ben       20.0
nick     20.0
dtype: float64
```

Acceso a los elementos de una Serie

```
import pandas as pd

# Creamos una serie a partir de un diccionario que tiene nombres de personas como claves
test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

balances = pd.Series(test_balance_data)
```

Acceso por indexación como lista

```
# Una Serie es ordenada e indexada (como las listas de Python)
print(balances[0])
print(type(balances[0]))
```

```
20.0
<class 'numpy.float64'>
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:2: FutureWarning: print(balances[0])
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/363276410.py:3: FutureWarning: print(type(balances[0]))
```

```
# La forma correcta es:
balances.iloc[0]
```

```
np.float64(20.0)
```

```
# Si quieres obtener el ultimo balance
balances[-1]
```

```
/var/folders/yj/2_4_0_ln42j05fvg3t6db0pc0000gn/T/ipykernel_46975/3307448855.py:2: FutureWarning: balances[-1]
```

```
np.float64(42.42)
```

Acceso por la clave de índice

```
# Acceder a un elemento por su clave(index según pandas)
balances['pasan']
```

```
np.float64(20.0)
```

```
# Las series de pandas se comportan como diccionarios

for label,value in balances.items():
    print(label, value)
```

```
pasan 20.0
treasure 20.18
ashley 1.05
craig 42.42
```

```
# Acceder a un elemento usando su clave como si fuera un atributo
balances.ashley
```

```
np.float64(1.05)
```

```
# Acceder a los elementos explicitamente como lo menciona la documentación
print(balances.loc['ashley'])
print(balances.iloc[0])
```

```
1.05
20.0
```

```
# Acceder a una porción de la serie
balances[['treasure':'craig']]
```

```
treasure    20.18
ashley      1.05
craig       42.42
dtype: float64
```

```
balances.iloc[0:3]
```

```
pasan      20.00
treasure   20.18
ashley     1.05
dtype: float64
```

Vectorización y broadcasting

se trata de la capacidad de pandas para realizar operaciones element-wise (elemento a elemento) en estructuras de datos con diferentes formas, alineando automáticamente las

dimensiones según sea necesario.

```
import pandas as pd

test_balance_data = {
    'pasan': 20.00,
    'treasure': 20.18,
    'ashley': 1.05,
    'craig': 42.42,
}

test_deposit_data = {
    'pasan': 20,
    'treasure': 10,
    'ashley': 100,
    'craig': 55
}

balances = pd.Series(test_balance_data)
deposits = pd.Series(test_deposit_data)
```

Vectorización

Aunque es posible recorrer cada elemento y aplicarlo a otro...

```
for label, value in deposits.items():
    print(f"Depositando {value} en la cuenta de {label}, que tiene {balances[label]}")
    balances[label] += value

balances
```

Depositando 20 en la cuenta de pasan, que tiene 20.0
 Depositando 10 en la cuenta de treasure, que tiene 20.18
 Depositando 100 en la cuenta de ashley, que tiene 1.05
 Depositando 55 en la cuenta de craig, que tiene 42.42

pasan	40.00
treasure	30.18
ashley	101.05
craig	97.42
dtype: float64	

...es importante recordar apoyarse en la vectorización y omitir los bucles por completo. La vectorización es más rápida y, como puedes ver, más fácil de leer y escribir.

```
# Se debe considerar que para esta operacion los indices deben coincidir, de lo contrario se genera un error
balances += deposits
balances
# agrega 10 a la cuenta de 'james' en el diccionario de depositos para y ejecuta nueva operacion
```

```
pasan      60.00
treasure    40.18
ashley     201.05
craig      152.42
dtype: float64
```

```
balances -= deposits
balances
```

```
pasan      40.00
treasure    30.18
ashley     101.05
craig      97.42
dtype: float64
```

Broadcasting

Esto lo que permite es sumar un escalar a cada uno de los elementos de la serie.

```
balances + 5
```

```
pasan      45.00
treasure    35.18
ashley     106.05
craig      102.42
dtype: float64
```

Hacer broadcast de una serie con otra serie es posible siempre y cuando ambas tengan el mismo índice. En caso de que no sea así, se devolverá NaN.

```
coupons = pd.Series(1, ['craig', 'ashley', 'james'])  
coupons
```

```
craig      1  
ashley     1  
james      1  
dtype: int64
```

```
# Sumemos los balances con los cupones, el resultado es una nueva serie  
balances + coupons
```

```
ashley      102.05  
craig       98.42  
james        NaN  
pasan        NaN  
treasure     NaN  
dtype: float64
```

```
balances
```

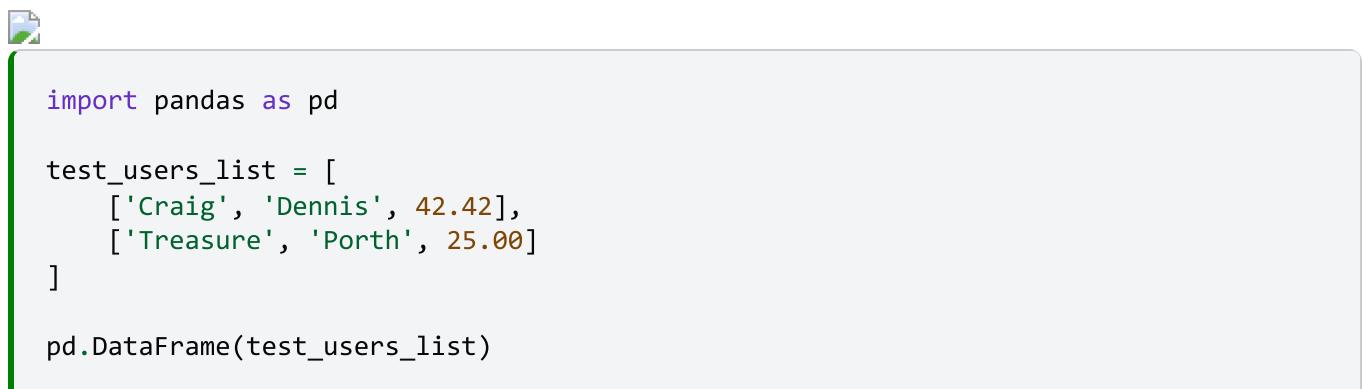
```
pasan      40.00  
treasure   30.18  
ashley     101.05  
craig      97.42  
dtype: float64
```

```
balances.add(coupons, fill_value=0) # Así se puede evitar el NaN y que no se pierda la
```

```
ashley      102.05  
craig       98.42  
james       1.00  
pasan      40.00  
treasure    30.18  
dtype: float64
```

DataFrames de pandas

Un DataFrame es una estructura de datos bidimensional en la biblioteca pandas de Python, similar a una tabla de datos o una hoja de cálculo en Excel. Cada DataFrame tiene un índice asociado, que son las etiquetas de las filas, y columnas, que son las etiquetas de las columnas. Los elementos de un DataFrame pueden ser de diferentes tipos, como enteros, flotantes, cadenas, entre otros.

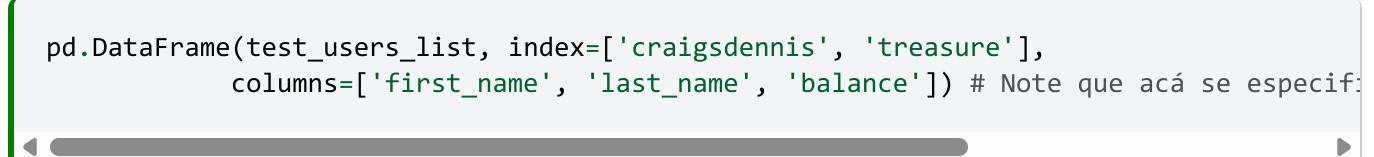


```
import pandas as pd

test_users_list = [
    ['Craig', 'Dennis', 42.42],
    ['Treasure', 'Porth', 25.00]
]

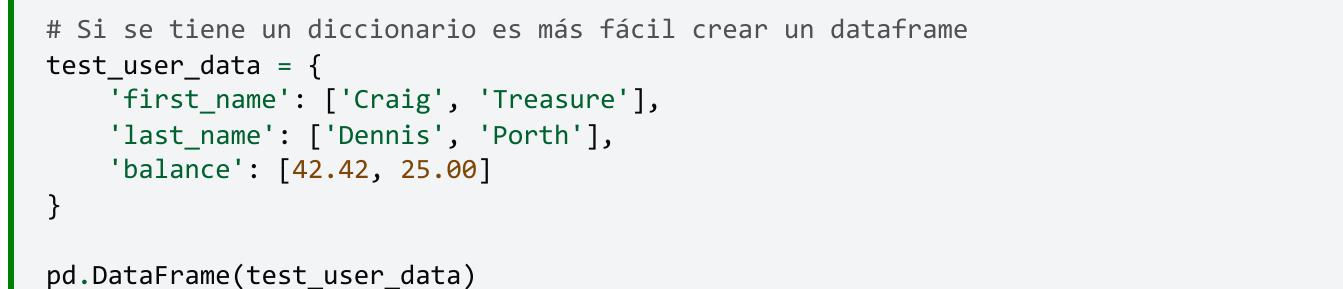
pd.DataFrame(test_users_list)
```

	0	1	2
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00



```
pd.DataFrame(test_users_list, index=['craigsdennis', 'treasure'],
             columns=['first_name', 'last_name', 'balance']) # Note que acá se especifica el índice y las columnas
```

	first_name	last_name	balance
craigsdennis	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00



```
# Si se tiene un diccionario es más fácil crear un dataframe
test_user_data = {
    'first_name': ['Craig', 'Treasure'],
    'last_name': ['Dennis', 'Porth'],
    'balance': [42.42, 25.00]
}

pd.DataFrame(test_user_data)
```

	first_name	last_name	balance
0	Craig	Dennis	42.42
1	Treasure	Porth	25.00

```
# Se puede especificar los indices de las filas
pd.DataFrame(test_user_data, index=['craigslist', 'treasure'])
```

	first_name	last_name	balance
craigslist	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00

Acceso a los elementos de un DataFrame

```
# Vamos a crear un dataframe con algunos datos de usuarios
import pandas as pd

test_user_data = { # Estos son los datos de los usuarios y el balance
    'first_name': ['Craig', 'Treasure', 'Ashley', 'Guil'],
    'last_name': ['Dennis', 'Porth', 'Boucher', 'Hernandez'],
    'balance': [42.42, 25.00, 2.02, 87.00]
}
test_user_names = ['craigslist', 'treasure', 'lindsay2000', 'guil'] # Supongamos que

users = pd.DataFrame(test_user_data, index=test_user_names)
users
```

	first_name	last_name	balance
craigslist	Craig	Dennis	42.42
treasure	Treasure	Porth	25.00
lindsay2000	Ashley	Boucher	2.02
guil	Guil	Hernandez	87.00

Obtener una columna específica

Cada columna en un DataFrame es una Serie de pandas, por lo que puedes acceder a una columna específica utilizando la notación de corchetes y el nombre de la columna y el retorno es una serie.

```
balances = users['balance']  
balances
```

```
craigsdennis    42.42  
treasure        25.00  
lindsay2000     2.02  
guil            87.00  
Name: balance, dtype: float64
```

```
# La serie que retorna la celda anterior tiene una propiedad llamada name que retorna  
balances.name
```

```
'balance'
```

Puedes obtener una fila de un Dataframe usando la propiedad .loc[] y pasando el índice de la fila, o el número.

```
users.loc['guil']
```

```
first_name        Guil  
last_name        Hernandez  
balance          87.0  
Name: guil, dtype: object
```

```
users.iloc[3]
```

```
first_name        Guil  
last_name        Hernandez  
balance          87.0  
Name: guil, dtype: object
```

Recuperar un valor específico mediante encadenamiento

```
users['first_name']['craigdennis']
```

```
'Craig'
```

```
users.loc['craigdennis']['first_name']
```

```
'Craig'
```

```
users.loc['craigdennis', 'first_name']
```

```
'Craig'
```

```
users.at['craigdennis', 'first_name']
```

```
'Craig'
```

Usando las propiedades loc e iloc puedes dividir un DataFrame existente en uno nuevo.

En el ejemplo a continuación, usamos : en el eje de filas para seleccionar todas las filas, y especificamos qué columnas queremos recuperar usando una lista en el eje de columnas

```
# Esto se lee como:  
# En el dataframe users, selecciona todas las filas, y dame solo las columnas 'balance'  
users.loc[:, ['balance', 'last_name']]
```

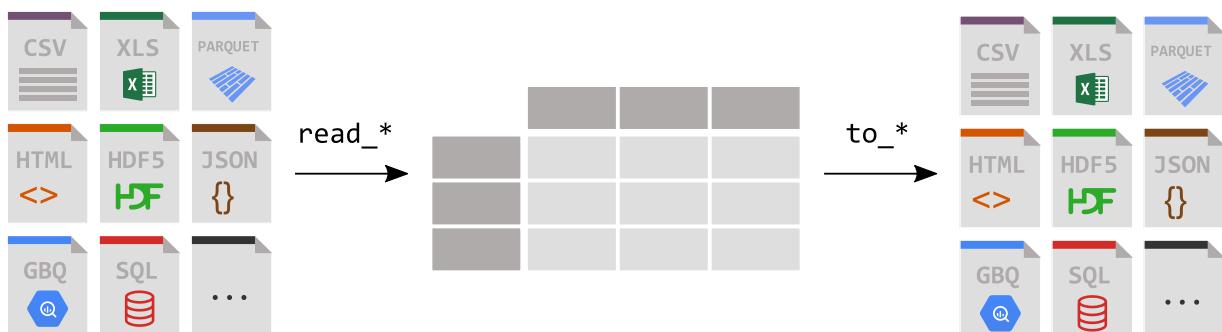
	balance	last_name
craigslist	42.42	Dennis
treasure	25.00	Porth
lindsay2000	2.02	Boucher
guil	87.00	Hernandez

```
# O lo puedes indexar por posición como en las listas
users.iloc[1:3, 1:]
```

	last_name	balance
treasure	Porth	25.00
lindsay2000	Boucher	2.02

Cargar Fuentes externas de datos

Naturalmente, cuando se trabaja con datos en la vida real, no siempre se tienen los datos en un DataFrame de pandas. A menudo, los datos se almacenan en archivos CSV, Excel, bases de datos SQL, o en la web. Afortunadamente, pandas proporciona una variedad de funciones para cargar datos desde diferentes fuentes y formatos, lo que facilita la importación y exportación de datos en Python.



La sintaxis general para cargar datos en un DataFrame