



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Laboratorio de datos

Python + Pandas

Primer Cuatrimestre 2025

Contenido

- + PythonTutor - Copias de Listas
- + Spyder - IDE
- + Diccionarios
- + Módulos
- + Manejo de archivos
- + Numpy
- + Pandas

PythonTutor

Python Tutor - <https://pythontutor.com/>

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
[known limitations](#)

```
1 a = 3
→ 2 b = 10
→ 3 c = a + b
4 a = 5
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First

< Prev

Next >

Last >>

Step 3 of 4

Frames

Objects

Global frame

a	3
b	10

Ejemplo

Escriban en pythontutor un código que:

- construya una lista a con los números del 1 al 5
- construya una lista b con las letras vocales
- construya una lista c que sea la concatenación de a y b

Ejemplo

Escriban en pythontutor un programa que:

- construya una lista a con los números del 1 al 5
- construya una lista b que sea igual a la lista a
- agregue un 6 a la lista a
- agregue un 6 a la lista b

¿Cómo es el estado de las variables al finalizar el programa?

Copias de Listas

Copias

Cuando creamos una lista igual a "a", al modificar una, se modifica la otra también.

```
a = [1,2,3,4,5]
b = a           # b = [1,2,3,4,5] igual que a
a.append(6)     # acá el 6 se agrega en ambas
b.append(6)     # acá el 6 se agrega en ambas
```

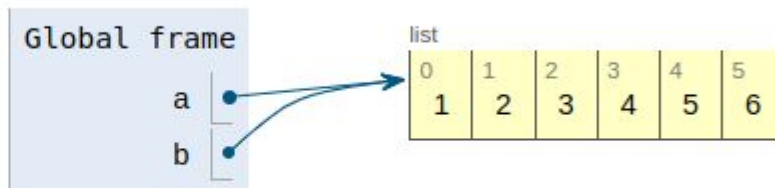
Python 3.6
[known limitations](#)

```
1 a = [1,2,3,4,5]
2 b = a
→ 3 a.append(6)
→ 4 b.append(6)
```

[Edit this code](#)

Frames

Objects



Copias

Las listas (y otros objetos) tienen un método para hacer copias. Cuando creamos una copia b de a, modificar una no tiene efecto sobre la otra.

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

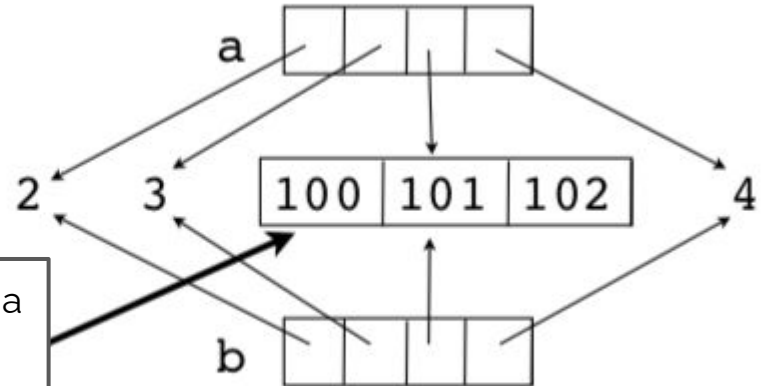
Ejemplo

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

```
a.append(5)
print(b)
```

```
a[2].append(102)
print(b)
```

Esta lista interna es la misma para a y b.



Deepcopy

```
import copy
```

```
a = [2,3,[100,101],4]
```

```
b = copy.deepcopy(a)
```

```
a.append(5)
```

```
print(b)
```

```
a[2].append(102)
```

```
print(b)
```



Entorno de desarrollo integrado (*IDE*) - Spyder

- + código abierto (open source)
- + multiplataforma (sirve en linux, ios, windows...)
- + es cómodo para escribir código, ejecutarlo, corregirlo, probarlo y utilizarlo, en el mismo entorno
- + el editor de texto remarca palabras clave del lenguaje
- + tiene atajos (shortcuts) útiles (y modificables a gusto de cada uno)

Entorno de desarrollo integrado (IDE) - Spyder

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `codigo_primeraclase.py` with the following content:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Feb 26 22:07:07 2023
5
6@author: mcerdeiro
7"""
8
9a = 3
10b = 10
11c = a + b
12a += 5
13
14
15
```

The Variable explorer panel on the right shows the current state of variables:

Name	Type	Size	Value
a	int	1	3
b	int	1	10

The IPython console at the bottom shows the execution of the code:

```
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: a = 3
In [2]: b = 10
In [3]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 15, Column: 1, Memory: 41 %.

Entorno de desarrollo integrado (IDE) - Spyder

The image shows the Spyder Python IDE interface with several components and annotations:

- Editor:** The main window for editing code. It contains a file named `codigo_primeraclase.py`. The code is a Python script with a shebang, encoding declaration, docstring, and variable assignments.
- Variable explorer:** A panel on the right that displays the current state of variables in the program. It shows a table with columns for Name, Type, Size, and Value.
- IPython console:** A panel at the bottom right for running code interactively. It shows the output of the code executed in the editor.
- Annotations:** Four callout boxes with arrows pointing to specific parts of the interface: "nombre del programa" points to the file name in the editor; "directorio actual de trabajo" points to the current working directory in the top bar; "explorador de variables (muestra el estado del programa)" points to the Variable explorer; "intérprete de python" points to the IPython console.

Code in the editor:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Feb 26 22:07:07 2023
5
6@author: mcerdeiro
7"""
8
9a = 3
10b = 10
11c = a + b
12a += 5
13
14
15
```

Variable explorer table:

Name	Type	Size	Value
a	int	1	3
b	int	1	10

IPython console output:

```
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: a = 3
In [2]: b = 10
In [3]:
```

Status bar: Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 15 Column: 1 Memory: 41 %

Diccionarios

Diccionarios

Los diccionarios son útiles si vamos a querer buscar rápidamente (por claves).

- Se construyen con llaves
- Cada entrada tiene una clave y un valor, separados por dos puntitos :
- Las entradas se separan con comas

```
{clave1: valor1, clave2: valor2, ... }
```

- Se acceden con corchetes indicando una clave
- Tanto las claves como los valores pueden ser de distintos tipos de objetos
- Las claves deben ser de tipo inmutable

Ejemplo

```
dias_engl = {'lunes': 'monday', 'martes': 'tuesday', 'miércoles': 'wednesday', 'jueves':  
'thursday'}
```

```
>>> dias_engl['lunes']  
'monday'
```

```
>>> dias_engl['viernes']  
Traceback (most recent call last):
```

```
  File "<ipython-input-33-ee0fa133453b>", line 1, in <module>  
    dias_engl['viernes']
```

```
KeyError: 'viernes'
```

```
>>> dias_engl['viernes'] = 'friday'      # agrego la entrada  
>>> dias_engl['viernes']  
'friday'
```

Ejemplo

También se pueden armar y modificar agregando entradas:

```
feriados = {} # Empezamos con un diccionario vacío
```

```
# Agregamos elementos
```

```
feriados[(1, 1)] = 'Año nuevo'
```

```
feriados[(1, 5)] = 'Día del trabajador'
```

```
feriados[(13, 9)] = 'Día del programador'
```

```
# Modifico una entrada
```

```
feriados[(13, 9)] = 'Día de la programadora'
```

Diccionarios

También se pueden armar a partir de una lista de tuplas (clave, valor).

```
>>> cuadrados = dict([(1,1), (2,4), (3,9), (4,16)])  
>>> cuadrados[2]  
4
```

La función **zip** genera tuplas a partir de dos listas:

```
>>> for t in zip([1,2,3,4],[1,4,9,16]):  
    print(t)  
(1, 1)  
(2, 4)  
(3, 9)  
(4, 15)
```

Es decir que podemos construir el diccionario a partir de dos listas (claves y valores);

```
>>> cuadrados = dict(zip([1,2,3,4],[1,4,9,16]))
```

Módulos

Módulos

Si bien Python tiene muchas funciones que se pueden usar directamente, hay muchas otras que están disponibles dentro de módulos.

Un **módulo** es una **colección de funciones** que alguien (o una comunidad) desarrollaron y empaquetaron para que estén disponibles para todo el mundo.

Para que las funciones estén disponibles para ser utilizadas en mi programa, tengo que usar la instrucción **import**.

Módulos

Si quiero generar números aleatorios, que están en el módulo random, tengo que escribir:

```
import random
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
```

```
random.seed(COMPLETAR con un NÚMERO)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
```

Módulos

Módulo math tiene funciones matemáticas.

```
import math
```

```
math.sqrt(2)
```

```
math.exp(x)
```

```
math.cos(x)
```

```
math.gcd(15, 12)
```

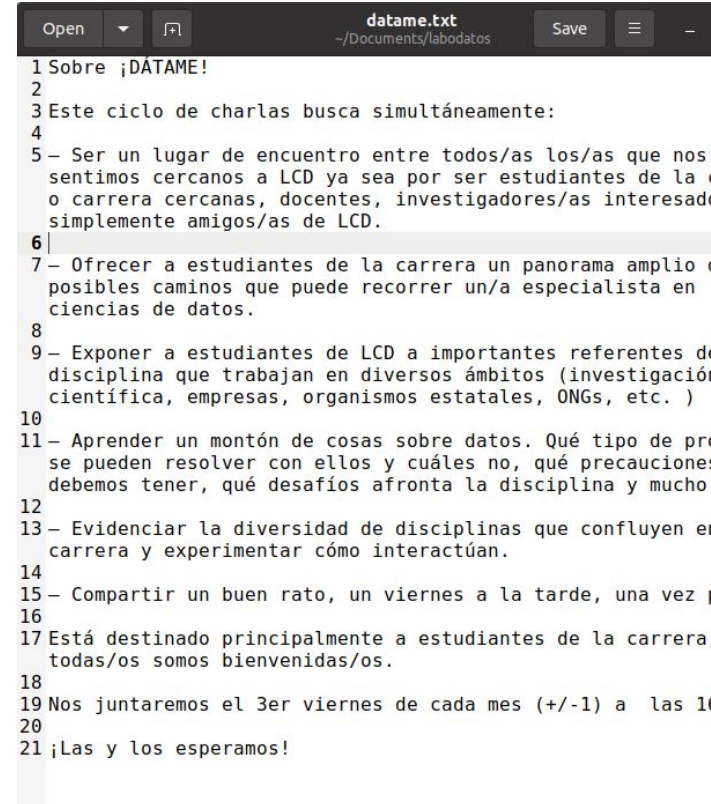

Archivos

Ejemplo

Tenemos un archivo de texto, que queremos actualizar.

Se trata del correo de difusión de las charlas Datame.

Queremos agregarle una línea al inicio: **“Este 2025 retomamos con nuestras ya clásicas charlas Datame”**, y también agregar un cierre al final: **“Sin presupuesto no hay universidad”**, y guardarlo con un nuevo nombre **“datame_2025.txt”**



```
1 Sobre ¡DATAME!
2
3 Este ciclo de charlas busca simultáneamente:
4
5 - Ser un lugar de encuentro entre todos/as los/as que nos
  sentimos cercanos a LCD ya sea por ser estudiantes de la
  o carrera cercanas, docentes, investigadores/as interesados
  simplemente amigos/as de LCD.
6
7 - Ofrecer a estudiantes de la carrera un panorama amplio
  posibles caminos que puede recorrer un/a especialista en
  ciencias de datos.
8
9 - Exponer a estudiantes de LCD a importantes referentes de
  disciplina que trabajan en diversos ámbitos (investigación
  científica, empresas, organismos estatales, ONGs, etc. )
10
11 - Aprender un montón de cosas sobre datos. Qué tipo de pro
  se pueden resolver con ellos y cuáles no, qué precauciones
  debemos tener, qué desafíos afronta la disciplina y mucho
12
13 - Evidenciar la diversidad de disciplinas que confluyen en
  carrera y experimentar cómo interactúan.
14
15 - Compartir un buen rato, un viernes a la tarde, una vez p
16
17 Está destinado principalmente a estudiantes de la carrera
  todas/os somos bienvenidas/os.
18
19 Nos juntaremos el 3er viernes de cada mes (+/-1) a las 1
20
21 ¡Las y los esperamos!
```

Ejemplo

```
Open  datame.txt  Save  ~/Documents/labodatos
1 Sobre ¡DATAME!
2
3 Este ciclo de charlas busca simultáneamente:
4
5 - Ser un lugar de encuentro entre todos/as los/as que nos
  sentimos cercanos a LCD ya sea por ser estudiantes de la carrera
  o carrera cercanas, docentes, investigadores/as interesados/as o
  simplemente amigos/as de LCD.
6
7 - Ofrecer a estudiantes de la carrera un panorama amplio de
  posibles caminos que puede recorrer un/a especialista en
  ciencias de datos.
8
9 - Exponer a estudiantes de LCD a importantes referentes de la
  disciplina que trabajan en diversos ámbitos (investigación
  científica, empresas, organismos estatales, ONGs, etc. )
10
11 - Aprender un montón de cosas sobre datos. Qué tipo de problemas
  se pueden resolver con ellos y cuáles no, qué precauciones
  debemos tener, qué desafíos afronta la disciplina y mucho más.
12
13 - Evidenciar la diversidad de disciplinas que confluyen en esta
  carrera y experimentar cómo interactúan.
14
15 - Compartir un buen rato, un viernes a la tarde, una vez por mes.
16
17 Está destinado principalmente a estudiantes de la carrera, pero
  todas/os somos bienvenidas/os.
18
19 Nos juntaremos el 3er viernes de cada mes (+/-1) a las 16hs.
20
21 ¡Las y los esperamos!
```

Ejemplo:

```
nombre_archivo = 'datame.txt'
```

```
nombre_archivo = 'datame.txt'
f = open(nombre_archivo, 'rt')
data = f.read()
f.close()
```

```
data
print(data)
```

Archivos

Frecuentemente vamos a utilizar una fuente de datos, que en muchos casos va a estar en un archivo. Tenemos que poder manejar archivos: leer, crear, modificar, guardar archivos de distintos tipos.

```
f = open(nombre_archivo, 'rt' )      # abrir para lectura ('r' de read, 't' de text)
data = f.read() # la variable data tiene el contenido del archivo
f.close()
data
print(data)
```

Archivos

```
with open(nombre_archivo, 'rt') as file:    # otra forma de abrir archivos
    data = file.read()
    # 'data' es una cadena con todo el texto en el archivo

data
print(data)
```

Para leer una archivo línea por línea, usá un ciclo for como éste:

```
with open(nombre_archivo, 'rt') as file:
    for line in file:
        # Procesar la línea
```

Python tiene dos modos de salida. En el primero, escribimos data en el intérprete y Python muestra la representación **cruda** de la cadena, incluyendo comillas y códigos de escape (\n). Cuando escribimos print(data), en cambio, se imprime la salida **formateada** de la cadena.

Escribir archivos

```
with open('datame.txt', 'rt') as file:  
    data = file.read()
```

```
data_nuevo = 'inicio del texto' + data  
data_nuevo = data_nuevo + 'cierre del texto'
```

```
datame = open("nuevonombre.txt","w") # write mode  
datame.write(data_nuevo)  
datame.close()
```

Archivos .csv

csv = comma separated values

Conjuntos de datos estructurados.

- son “planillas” guardadas como texto
- cada línea de texto es una fila de la planilla
- las comas separan las columnas

a,b,c

d,e,f

x,y,z

u,v,w

a	b	c
d	e	f
x	y	z
u	v	w

Archivos .csv

csv = comma separated values

Ejemplo:
nombre_archivo =
'cronograma_sugerido.csv'

Cuatrimestre	Asignatura	Correlatividad de Asignaturas
3	Álgebra I	CBC
3	Algoritmos y Estructuras de Datos I	CBC
4	Análisis I	CBC
4	Electiva de Introducción a las Ciencias Naturales	CBC
5	Análisis II	Análisis I
5	Álgebra Lineal Computacional	Álgebra I – Algoritmos y Estructuras de Datos I
5	Laboratorio de Datos	Algoritmos y Estructuras de Datos I
6	Análisis Avanzado	Análisis II, Álgebra I
6	Algoritmos y Estructuras de Datos II	Algoritmos y Estructuras de Datos I
7	Probabilidad	Análisis Avanzado
7	Algoritmos y Estructura de Datos III	Algoritmos y Estructuras de Datos II
8	Intr. a la Estadística y Ciencia de Datos	Lab de Datos, Probabilidad, Álgebra Lineal Computacional
8	Intr. a la Investigación Operativa y Optimización	Alg y Estruct de Datos III, Análisis II, Álgebra Lineal Computacional
8	Intr. al Modelado Continuo.	Análisis Avanzado, Álgebra Lineal Computacional, Alg y Estructura de Datos II

Archivos .csv

```
nombre_archivo = 'cronograma_sugerido.csv'
with open(nombre_archivo, 'rt') as file:
    for line in file:
        datos_linea = line.split(',')
        print(datos_linea[1])
```

¿Cómo podemos armar una lista con todas las materias del cronograma?

Módulo csv

Es útil para trabajar con archivos .csv

```
f = open(nombre_archivo)
filas = csv.reader(f)
for fila in filas:
    instrucciones
```

Acá `filas` es un iterador.

```
f.close()
```

Si la primera fila son encabezados, podemos guardarlos así:

```
f = open(nombre_archivo)
filas = csv.reader(f)
encabezado = next(filas) # un paso del iterador
for fila in filas:        # ahora el iterador sigue desde la segunda fila
    instrucciones

f.close()
```

Ejercicios

- + Escribir una función `general_tirar()` que simule una tirada de dados para el juego de la generala. Es decir, debe devolver una lista aleatoria de 5 valores de dados. Por ejemplo `[2, 3, 2, 1, 6]`.
- + *Opcional: Agregar al ejercicio `general_tirar()` que además imprima en pantalla si salió poker, full, generala, escalera o ninguna de las anteriores. Por ejemplo, si sale `2, 1, 1, 2, 2` debe devolver `[2, 1, 1, 2, 2]` e imprimir en pantalla `Full`.
- + Escribir un programa que recorra las líneas del archivo `'datame.txt'` e imprima solamente las líneas que contienen la palabra `'estudiante'`.
- + Utilizando el archivo `cronograma_sugerido`, armar una lista de las materias del cronograma, llamada `"lista_materias"`.
- + Luego, definir una función `"cuantas_materias(n)"` que, dado un número de cuatrimestre (n entre 3 y 8), devuelva la cantidad de materias a cursar en ese cuatrimestre. Por ejemplo: `cuantas_materias(5)` debe devolver 3.
- + Definir una función `materias_cuatrimestre(nombre_archivo, n)` que recorra el archivo indicado, conteniendo información de un cronograma sugerido de cursada, y devuelva una lista de diccionarios con la información de las materias sugeridas para cursar el n-ésimo cuatrimestre.

```
materias_cuatrimestre('cronograma_sugerido.csv', 3):  
  
[{'Cuatrimestre': '3',  
  'Asignatura': 'Álgebra I',  
  'Correlatividad de Asignaturas': 'CBC'},  
 {'Cuatrimestre': '3',  
  'Asignatura': 'Algoritmos y Estructuras de Datos I',  
  'Correlatividad de Asignaturas': 'CBC'}]
```

Ejemplo

Definimos `registros(nombre_archivo)` que recorre el archivo indicado, conteniendo información de un cronograma sugerido de cursada, y devuelve la información como una lista de diccionarios. Las claves de los diccionarios son las columnas del archivo, y los valores son las entradas de cada fila para esa columna.

```
def registros(nombre_archivo):  
    lista = []  
    with open(nombre_archivo, 'rt') as f:  
        filas = csv.reader(f)  
        encabezado = next(filas)  
        for fila in filas:  
            registro = dict(zip(encabezado, fila)) # armo el diccionario de cada fila  
            lista.append(registro)                # lo agrego a la lista  
    return lista
```

Numpy

Numpy (Numerical Python)

- Colección de módulos de código abierto que tiene aplicaciones en casi todos los campos de las ciencias y de la ingeniería.
- Estándar para trabajar con datos numéricos en Python.
- Muchas otras bibliotecas de Python (Pandas, SciPy, Matplotlib, scikit-learn, scikit-image, etc) usan numpy.
- Objetos: matrices multidimensionales por medio del tipo **ndarray** (un objeto n-dimensional homogéneo, es decir, con todas sus entradas del mismo tipo)
- Métodos para operar **eficientemente** sobre las mismas.

Se lo suele importar así:

```
import numpy as np
```

Numpy (Numerical Python)

```
import numpy as np
```

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

```
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) # 2 dimensiones
```

```
print(a[0])
```

```
print(b[0])
```

```
print(b[2][3])
```

```
print(b[2,3])
```

```
np.zeros(2) # matriz de ceros del tamaño indicado
```

```
np.zeros((2,3))
```

data = np.array([1,2])

data

1
2

ones = np.ones(2)

ones

1
1

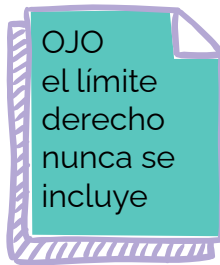
Numpy (Numerical Python)

También podés crear vectores a partir de un rango de valores:

```
np.arange(4) # array([0, 1, 2, 3])
```

También un vector que contiene elementos equiespaciados, especificando el primer número, el límite, y el paso.

```
np.arange(2, 9, 2) # array([2, 4, 6, 8])
```



También podés usar `np.linspace()` para crear un vector de valores equiespaciados especificando el primer número, el último número, y la cantidad de elementos:

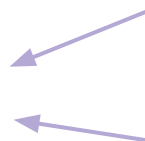
```
np.linspace(0, 10, num=5) # array([0., 2.5, 5., 7.5, 10.])
```


Ejemplos

```
a = np.array([1, 2, 3, 4])  
b = np.array([5, 6, 7, 8])  
np.concatenate((a, b))
```

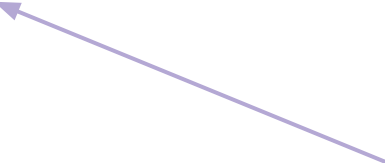
```
x = np.array([[1, 2], [3, 4]])  
y = np.array([[5, 6], [7, 8]])
```

```
z = np.concatenate((x, y), axis = 0)  
z = np.concatenate((x, y), axis = 1)
```



1	2
3	4
5	6
7	8

1	2
3	4
5	6
7	8



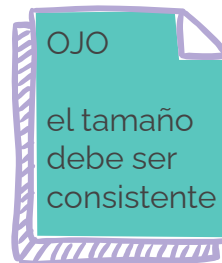
1	2	5	6
3	4	7	8

Ejemplos

Un ejemplo de array de 3 dimensiones.

```
array_ejemplo = np.array([[[0, 1, 2, 3],  
                             [4, 5, 6, 7]],  
                           [[3, 8, 10, -1],  
                             [0, 1, 1, 0]],  
                           [[3, 3, 3, 3],  
                             [5, 5, 5, 5]]])
```

```
array_ejemplo.ndim # cantidad de dimensiones - 3  
array_ejemplo.shape # cantidad de elementos en cada eje  
(3,2,4)  
array_ejemplo.size # total de entradas 3*2*4  
  
array_ejemplo.reshape((12,2)) # modifico la forma  
array_ejemplo.reshape((4,6))  
array_ejemplo.reshape((3,-1)) # 3 por lo que corresponda
```



Operaciones

`data = np.array([1,2])`

data

1
2

`ones = np.ones(2)`

ones

1
1

data + ones

=

data

1
2

+

ones

1
1

=

2
3

data

1
2

-

ones

1
1

=

0
1

data

1
2

*

data

1
2

=

1
4

data

1
2

/

data

1
2

=

1
1

Operaciones

1
2

 * 1.6 =

1
2

 *

1.6
1.6

 =

1.6
3.2

data

1
2
3

`.max()` =

3

data

1
2
3

`.min()` =

1

data

1
2
3

`.sum()` =

6

Operaciones

data

	0	1
0	1	2
1	3	4
2	5	6

data[0,1]

	0	1
0	1	2
1	3	4
2	5	6

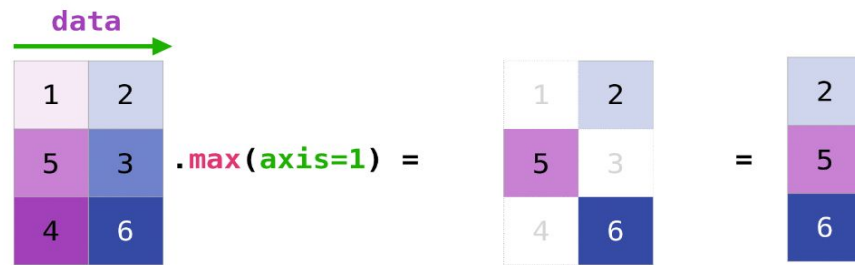
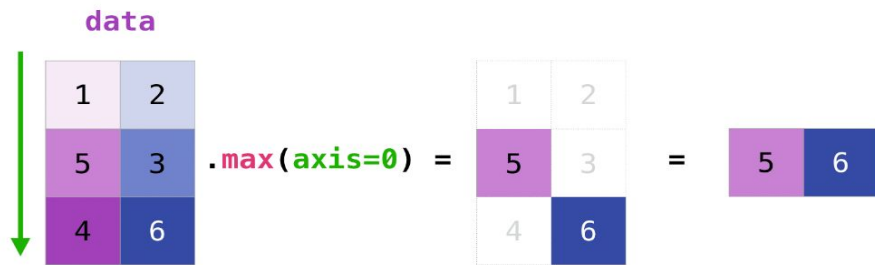
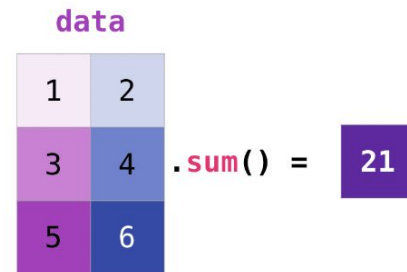
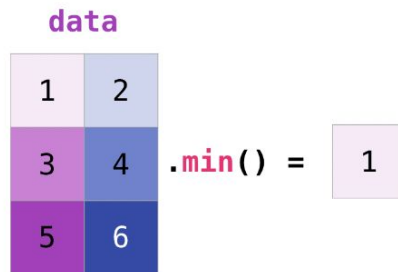
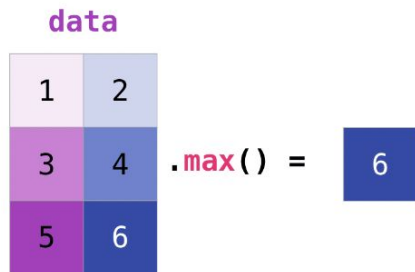
data[1:3]

	0	1
0	1	2
1	3	4
2	5	6

data[0:2,0]

	0	1
0	1	2
1	3	4
2	5	6

Operaciones



Ejercicio

Definir una función `pisar_elemento(M,e)` que tome una matriz de enteros `M` y un entero `e` y devuelva una matriz similar a `M` donde las entradas coincidentes con `e` fueron cambiadas por `-1`.

Por ejemplo si `M = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])` y `e = 2`, entonces la función debe devolver la matriz `np.array([[0, 1, -1, 3], [4, 5, 6, 7]])`

Pandas

Pandas

- + Pandas es una extensión de NumPy para manipulación y análisis de datos.
- + Ofrece estructuras de datos y operaciones para manipular tablas de datos (numéricos y de otros tipos) y series temporales.
- + Tipos de datos fundamentales: **DataFrames** que almacenan tablas de datos y las **Series** que contienen secuencias de datos.

```
import pandas as pd
```

Crear un Data Frame

Ejemplo: Queremos armar una tabla con los datos de lxs estudiantes de un curso.

lu	nombre	apellido
78/23	Antonio	Restrepo
449/22	Brenda	Saenz
111/24	Camilo	Torres
1/21	David	Urondo

Crear un Data Frame con un diccionario

```
d = {'nombre':['Antonio', 'Brenda', 'Camilo', 'David'], 'apellido': ['Restrepo', 'Saenz', 'Torres', 'Urondo'], 'lu': ['78/23', '449/22', '111/24', '1/21']}
```

```
df = pd.DataFrame(data = d) # creamos un df a partir de un diccionario
```

```
df.set_index('lu', inplace = True) # seteamos una columna como index
```

lu	nombre	apellido
78/23	Antonio	Restrepo
449/22	Brenda	Saenz
111/24	Camilo	Torres
1/21	David	Urondo

Crear un Data Frame *con un array*

Si tenemos datos numéricos en un array M:

```
M = np.array([[11, 1, -5, 3],[10, 5, 6, 7],[3, 8, 10, -1]])
```

```
df2 = pd.DataFrame(data = d) # creamos un df a partir de un array
```

```
df2 = pd.DataFrame(M, columns = ['a', 'b', 'c', 'd'], index = ['v1', 'v2', 'v3'])
```

Index	a	b	c	d
v1	11	1	-5	3
v2	10	5	6	7
v3	3	8	10	-1

Cargar un Data Frame *desde un archivo*

```
fname = 'directorio/cronograma_sugerido.csv'
```

```
df = pd.read_csv(fname)
```

Index	Cuatrimestre	Asignatura	Correlatividad de Asignaturas
0	3	Álgebra I	CBC
1	3	Algoritmos y Estructuras de Datos I	CBC
2	4	Análisis I	CBC
3	4	Electiva de Introducción a las Ciencias Naturales	CBC
4	5	Análisis II	Análisis I
5	5	Álgebra Lineal Computacional	Álgebra I - Algoritmos y Estructuras de Datos I
6	5	Laboratorio de Datos	Algoritmos y Estructuras de Datos I
7	6	Análisis Avanzado	Análisis II, Álgebra I
8	6	Algoritmos y Estructuras de Datos II	Algoritmos y Estructuras de Datos I
9	7	Probabilidad	Análisis Avanzado
10	7	Algoritmos y Estructura de Datos III	Algoritmos y Estructuras de Datos II
11	8	Intr. a la Estadística y Ciencia de Datos	Lab de Datos, Probabilidad, Álgebra Lineal Computacional
12	8	Intr. a la Investigación Operativa y Optimización	Alg y Estruct de Datos III, Análisis II, Álgebra Lineal Computacional

Primeras exploraciones

```
df.head()           # primeras 5 líneas
df.tail()           # últimas 5
df.info()           # info del df
df.dtypes           # tipos de dato
df.columns          # columnas
df.index            # índice (id de filas, pueden no ser int)
df.describe()       # una descripción
df[columnas]        # selecciono algunas columnas (una lista) por nombre
df[columna]         # solo una columna (sin lista) da una Serie
df.iloc[i]          # acceso a la fila i-ésima
df.iloc[2:6]        # filas 2 a 5
df.loc[index_6]      # acceso a fila por el index
df.loc[index_5, col2] # acceso a fila Y columna con index y nombre de col
df.sample()          # muestra una fila random
df.sample(n = 3)     # muestra n filas random
```

Ejemplo

Prueben con el siguiente dataframe.

Código en el campus - `pandas_script1.py`

Index	nombre	apellido	nota1	nota2	aprueba
78/23	Antonio	Restrepo	9	10	True
449/22	Brenda	Saenz	7	6	True
111/24	Camila	Torres	7	7	True
1/21	David	Urondo	4	8	False
201/06	Esteban	Valdes	3	5	False
47/20	Felicitas	Wainstein	nan	nan	nan

Ejemplo

¿QUÉ COSAS PODRÍAMOS QUERER HACER CON ESTE DATAFRAME?

Index	nombre	apellido	nota1	nota2	aprueba
78/23	Antonio	Restrepo	9	10	True
449/22	Brenda	Saenz	7	6	True
111/24	Camila	Torres	7	7	True
1/21	David	Urondo	4	8	False
201/06	Esteban	Valdes	3	5	False
47/20	Felicitas	Wainstein	nan	nan	nan

- ubicar valores nan
- sacar filas con valores nan
- ordenar por alguna columna
- calcular promedio de notas
- modificar una entrada
- agregar una fila o columna
- iterar sobre las filas para hacer algún cálculo
- considerar el conjunto de quienes tienen nota 7 o más
- ...



Ejercicio - Consigna 1

Index	nombre	apellido	nota1	nota2	aprueba
78/23	Antonio	Restrepo	9	10	True
449/22	Brenda	Saenz	7	6	True
111/24	Camila	Torres	7	7	True
1/21	David	Urondo	4	8	False
201/06	Esteban	Valdes	3	5	False
47/20	Felicitas	Wainstein	nan	nan	nan

1. mostrar sólo las columnas nombre y apellido
2. mostrar sólo la fila de libreta 449/22
3. mostrar las filas 2 a 4
4. mostrar el nombre de lx estudiante de libreta 201/06
5. armar una tabla notas parcial con libretas y notas del primer examen

Ejercicio - Consigna 2

Index	nombre	apellido	nota1	nota2	aprueba
78/23	Antonio	Restrepo	9	10	True
449/22	Brenda	Saenz	7	6	True
111/24	Camila	Torres	7	7	True
1/21	David	Urondo	4		
201/06	Esteban	Valdes	3		
47/20	Felicitas	Wainstein	na		

1. completar los NULL de la columna "aprueba" con False
2. corregir el nombre de David, en realidad es Daniel
3. calcular los promedios de las notas de cada estudiante
4. responder: aprueban todxs?
5. eliminar la columna "aprueba"
6. agregar 2 estudiantes a la lista (inventando sus datos)

Código en el campus -
`pandas_script2.py`

Filtros

Si queremos ver sólo las filas que satisfacen determinada condición, utilizamos un filtro basado en dicha condición.

```
df['nota1']>=7
```

nos da una serie booleana, que indica donde se cumple la condición

el index de esta serie es el del df

```
(df['nota1']>=7).sum()
```

cuenta los True

```
df[df['nota1']>=7]
```

nos da el sub-dataframe donde se cumple la condición

```
df[(df['nota1']>=7) & (df['nota2']>=7)]
```

podemos usar doble filtro, & es la conjunción

```
df[df['nota1']== 7]
```

```
df[df['nota1'].isin([7,4])]
```

```
df[(df['nota2'] <=7) & df['aprueba']]
```

```
df[(df['nota2'] <=7) | df['aprueba']]
```

| es la disyunción ("o")

Ejercicio - Consigna 3

Index	nombre	apellido	nota1	nota2	aprueba
78/23	Antonio	Restrepo	9	10	True
449/22	Brenda	Saenz	7		
111/24	Camila	Torres	7		
1/21	David	Urondo	4		
201/06	Esteban	Valdes	3		
47/20	Felicitas	Wainstein	na		

1. ver quiénes sacaron 6 o más en la nota 1
2. contar cuántxs estudiantes sacaron 6 o más
3. armar un dataframe con quienes sacaron 6 o más
4. armar un dataframe con quienes sacaron 6 o más en la nota 1 y 7 o más en la nota 2
5. armar un dataframe con quienes sacaron exactamente 7 en la nota 1
6. armar un dataframe con quienes aprobaron y sacaron 7 o menos en la nota 2
7. armar un dataframe con quienes sacaron menos que 6 en alguna instancia
8. armar un dataframe omitiendo las filas donde no hay nota 1 ni nota 2
9. contar cuántos estudiantes sacaron cada valor para la nota1
10. poner un 0 en las notas 1 que sean mayores a 4

Prueben con los datos de árboles.

Datos de árboles en parques de la Ciudad de Buenos Aires.

<https://data.buenosaires.gob.ar/dataset/arbolado-espacios-verdes>

```
import pandas as pd
archivo = 'arbolado-en-espacios-verdes.csv'
df = pd.read_csv(fname, index_col = 2)
```

Ejercicios:

1. Armar un dataframe que contenga las filas de Jacarandás y otro con los Palos Borrachos.
2. Calcular para cada especie seleccionada:
 - a. Cantidad de árboles, altura máxima, mínima y promedio, diámetro máximo, mínimo y promedio.
 - b. Definir una función cantidad_arboles(parque) que, dado el nombre de un parque, calcule la cantidad de árboles que tiene.
3. Definir una función cantidad_nativos (parque) que calcule la cantidad de árboles nativos de dicho parque.