Laboratorio de Datos

Primer cuatrimestre de 2023

Introducción a Python - parte 3

Contenido

- Repaso
- Numpy
- Pandas
- Ejercicios

Repaso

Repaso

+ Generala
+ Datame
+ Cronograma

Generala

Escribir una función generala_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].

Datame

Escribir un programa que recorra las líneas del archivo ‘datame.txt’ e imprima solamente las líneas que contienen la palabra ‘estudiante’.

Cronograma

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.

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.
Debe funcionar así:

```
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'}]
```

Numpy

Numpy (Numerical Python)

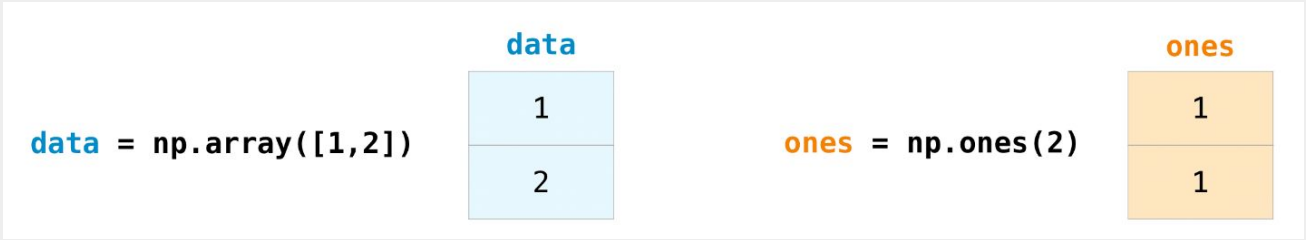
- ➔ Colección de módulos de código abierto que tiene aplicaciones en casi todos los campos de las ciencias y de la ingeniería.
- ➔ Estándar para trabajar con datos numéricos en Python.
- ➔ Muchas otras bibliotecas de Python (Pandas, SciPy, Matplotlib, scikit-learn, scikit-image, etc) usan numpy.
- ➔ 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))
```

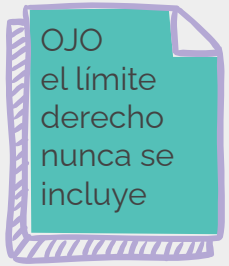


Numpy (Numerical Python)

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.])
```

Ejercicio

Generá un vector que tenga los números impares entre el 1 y el 19 inclusive usando arange().

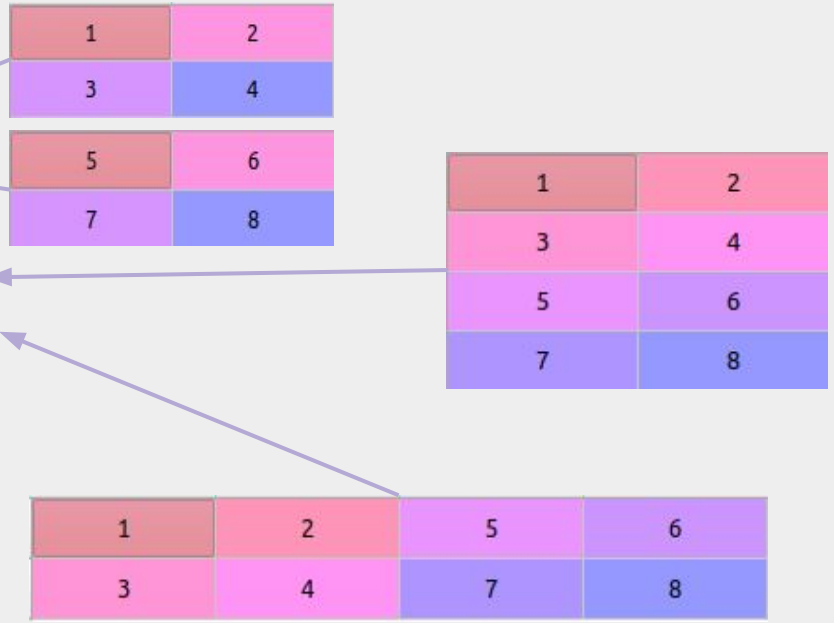
Repetí el ejercicio usando linspace(). ¿Qué diferencia hay en el resultado?

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)
```

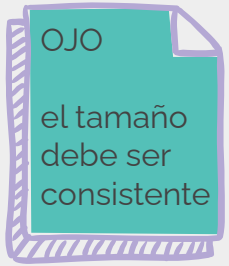


Ejemplos

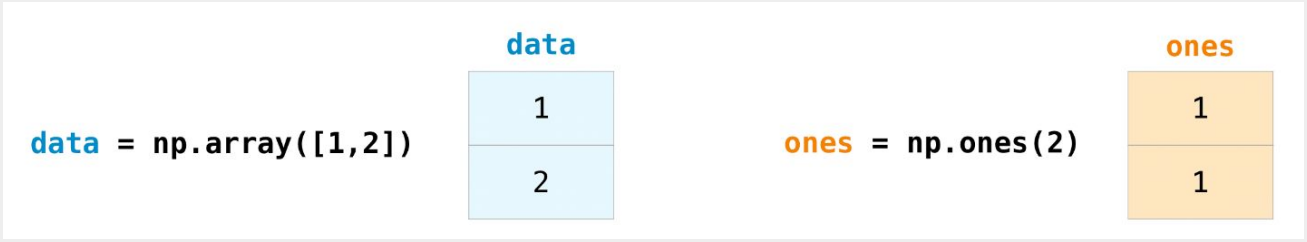
Un ejemplo de array de 3 dimensiones.

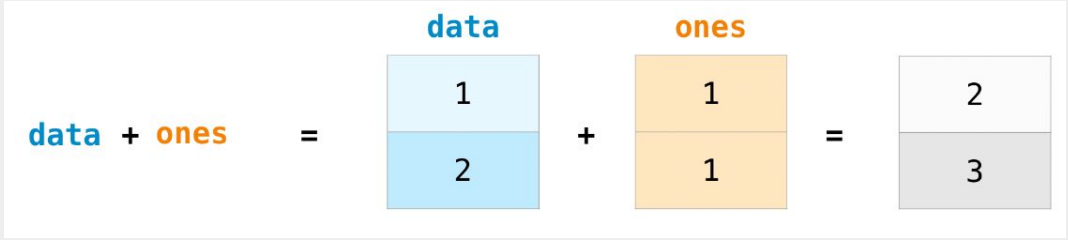
```
array_ejemplo = np.array([[[0, 1, 2, 3],
                           [4, 5, 6, 7]],
                           [[3, 8, 10, -1],
                           [0, 1, 1, 0]],
                           [[3 ,3 ,3, 3],
                           [5, 5, 5, 5]]])
```

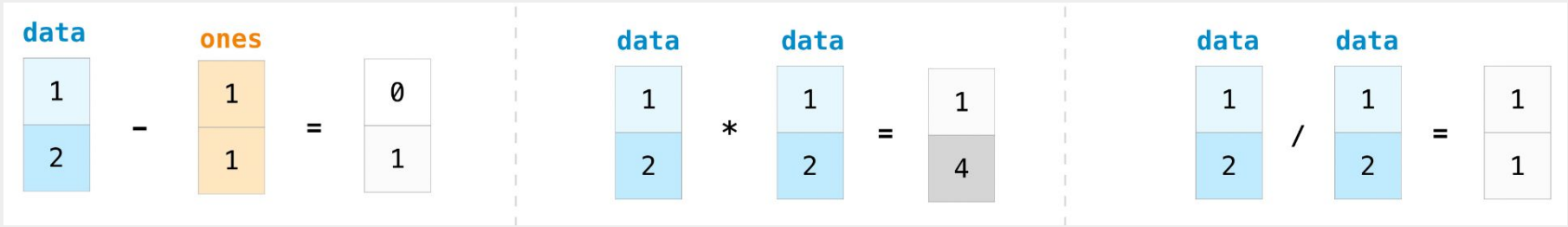
```
array_ejemplo.ndim # cantidad de dimensiones - 3
array_ejemplo.shape # cantidad de elementos en cada eje (3,2,4)
array_ejemplo.size # total de entradas 3*2*4

array_ejemplo.reshape((12,2)) # modifico la forma
array_ejemplo.reshape((4,6))
array_ejemplo.reshape((3,-1)) # 3 por lo que corresponda
```



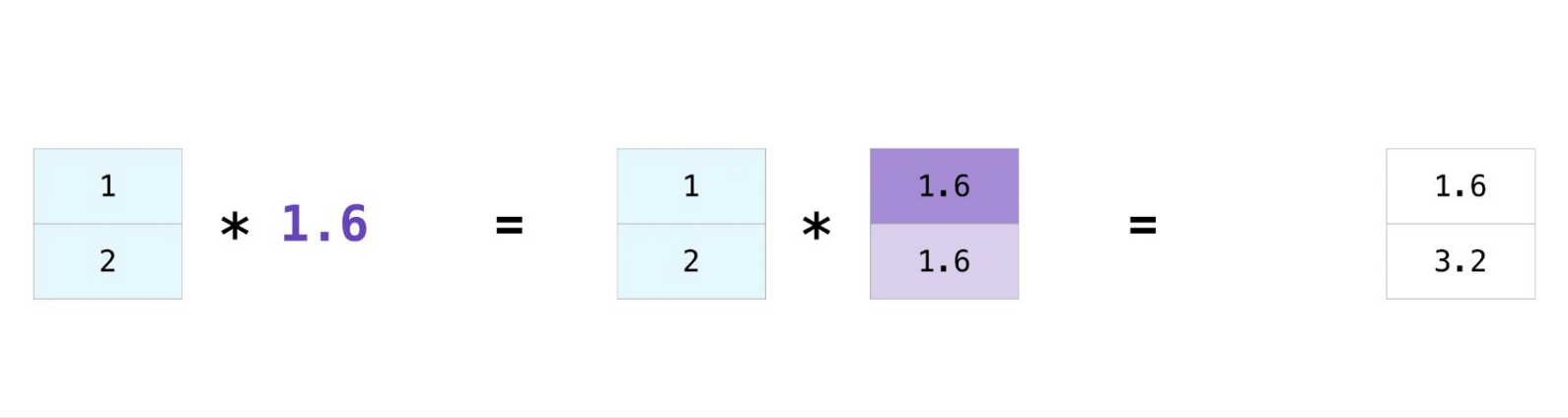
Operaciones

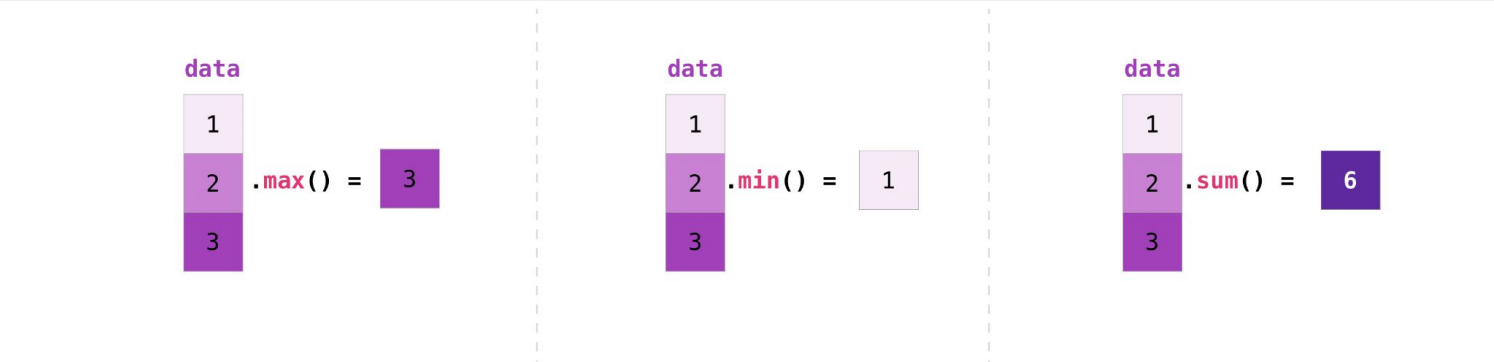




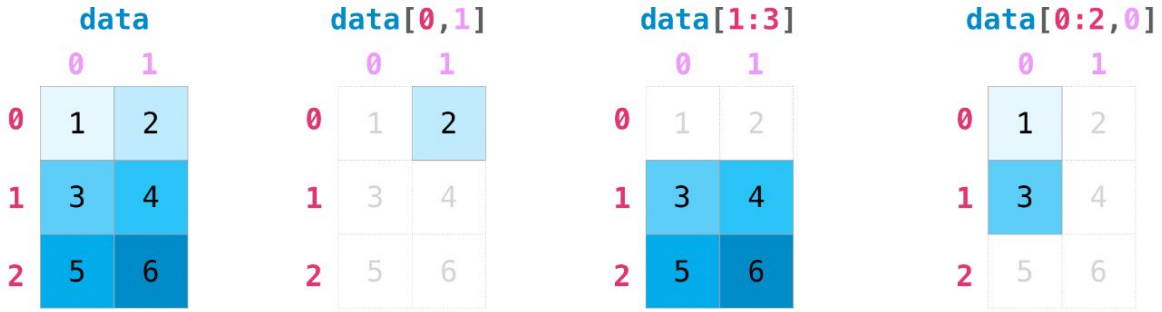


Operaciones

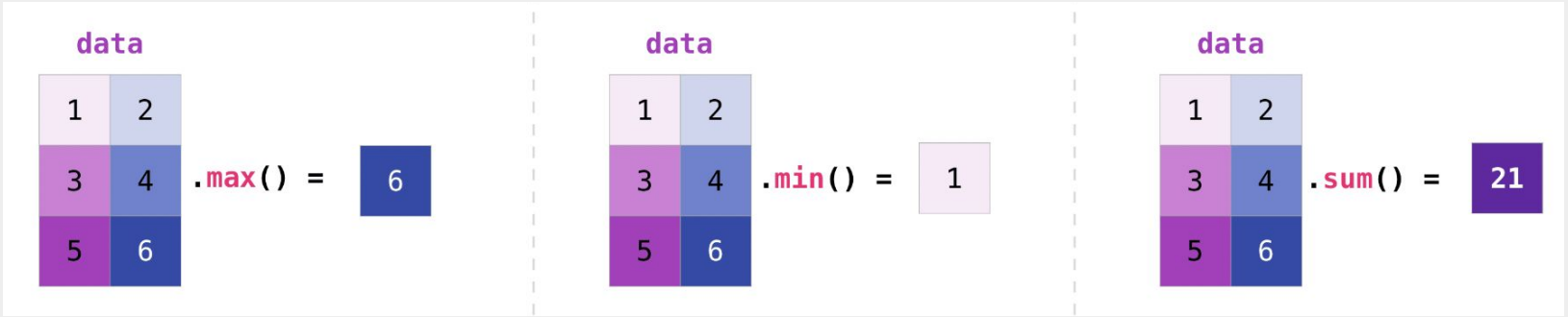


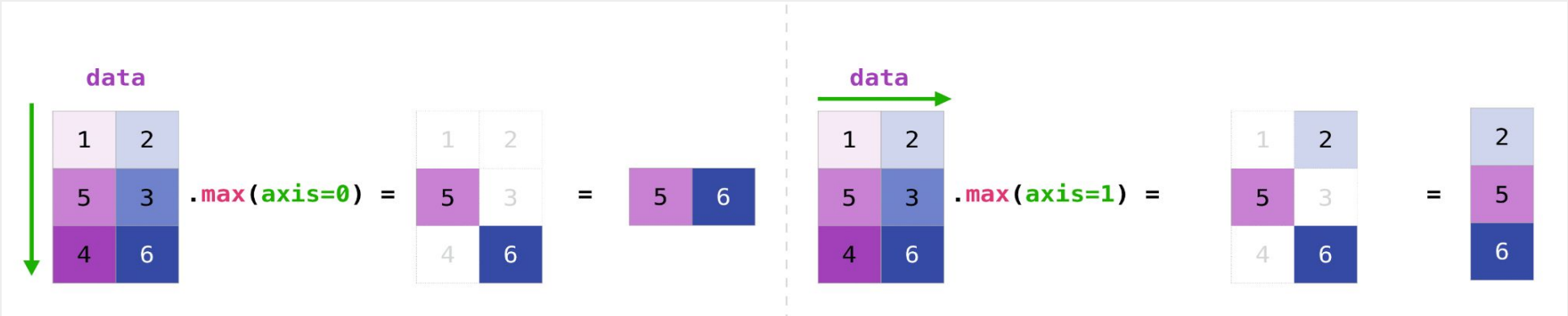


Operaciones



Operaciones





Ejercicios

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
```

Pandas

```
import pandas as pd

import os

archivo = 'arbolado-en-espacios-verdes.csv'

fname = os.path.join(directorio,archivo)

df = pd.read_csv(fname)
```

La variable df es de tipo DataFrame y contiene todos los datos del archivo csv estructurados adecuadamente.

Con df.head() podés ver las primeras líneas de datos. Si a head le pasás un número como parámetro podés seleccionar cuántas líneas querés ver. Análogamente con df.tail(n) verás las últimas n líneas de datos.

Pandas

```
>>> df.head() # primeras líneas

   long       lat      id_arbol ...          origen       coord_x       coord_y

0 -58.477564 -34.645015    1  ...           Exótico  98692.305719  98253.300738

1 -58.477559 -34.645047    2  ...           Exótico  98692.751564  98249.733979

2 -58.477551 -34.645091    3  ...           Exótico  98693.494639  98244.829684

3 -58.478129 -34.644567    4  ...  Nativo/Autóctono  98640.439091  98302.938142

4 -58.478121 -34.644598    5  ...  Nativo/Autóctono  98641.182166  98299.519997
```

Pandas

```
>>> df.columns
Index(['long', 'lat', 'id_arbol', 'altura_tot', 'diametro', 'inclinacio',
       'id_especie', 'nombre_com', 'nombre_cie', 'tipo_folla', 'espacio_ve',
       'ubicacion', 'nombre_fam', 'nombre_gen', 'origen', 'coord_x',
       'coord_y'],
      dtype='object')
>>> df.index
RangeIndex(start=0, stop=51502, step=1)
```

Pandas

```
>>> df[['altura_tot', 'diametro', 'inclinacio']].describe()
         altura_tot      diametro    inclinacio
count  51502.000000  51502.000000  51502.000000
mean      12.167100     39.395616      3.472215
std        7.640309     31.171205      7.039495
min        0.000000      1.000000      0.000000
25%        6.000000     18.000000      0.000000
50%       11.000000     32.000000      0.000000
75%       18.000000     54.000000      5.000000
max       54.000000    500.000000     90.000000
```

Filtros

```
>>> df['nombre_com'] == 'Ombú'
0        False
1        False
2        False
3         True
...
>>> (df['nombre_com'] == 'Ombú').sum()
590
df['nombre_com'].unique()   # una vez cada nombre
```

Filtros

```
cant_ejemplares = df['nombre_com'].value_counts()

cant_ejemplares.head(10)  # tabla con los 10 nombres más frecuentes
```

Filtros

```
>>> df_jacarandas = df[df['nombre_com'] == 'Jacarandá']

>>> cols = ['altura_tot', 'diametro', 'inclinacio']

>>> df_jacarandas = df_jacarandas[cols]

>>> df_jacarandas.tail()

       altura_tot  diametro  inclinacio

51104           7        97           4

51172           8        28           8

51180           2        30           0

51207           3        10           0

51375          17        40          20
```

Filtros

Si vas a querer modificar df_jacarandas es conveniente crear una copia de los datos de df en lugar de simplemente una vista. Esto se puede hacer con el método copy() como en el siguiente ejemplo.

```
df_jacarandas = df[df['nombre_com'] == 'Jacarandá'][cols].copy()
```

Filtros por índice y por posición

El índice de df no tiene una semántica interesante. Veamos, en cambio, que la serie que generamos con cant_ejemplares = df['nombre_com'].value_counts() sí lo tiene:

```
>>> cant_ejemplares.index

Index(['Eucalipto', 'Tipa blanca', 'Jacarandá', 'Palo borracho rosado',

       'Casuarina', 'Fresno americano', 'Plátano', 'Ciprés', 'Ceibo', 'Pindó',

       ...

       'Naranjo dulce', 'Peltophorum', 'Ligustrina de California',

       'Afrocarpus', 'Caranday', 'Esterculea', 'Boj cepillo', 'Sesbania',

       'Ligustrum', 'Árbol del humo'],

      dtype='object', length=337)
```

cant_ejemplares es una serie.
Tiene los nombres de las especies como índice y sus respectivas cantidades como dato asociado.

Para acceder por número de posición usá iloc, como se muestra a continuación.

```
>>> df.loc[165]
long                                          -58.4684
lat                                           -34.6648
id_arbol                                           166
altura_tot                                           5
diametro                                            10
inclinacio                                           0
id_especie                                          11
nombre_com                                   Jacarandá
nombre_cie                          Jacarandá mimosifolia
tipo_folla                  Árbol Latifoliado Caducifolio
```

Podemos acceder a una fila de un DataFarme o una Serie tanto a través de su posición como a través de su índice. Para acceder con el índice usá loc[] como en los siguientes ejemplos:

```
>>> df_jacarandas.iloc[0]

altura_tot     5

diametro      10

inclinacio     0

Name: 165, dtype: int64
```

Observá que esto nos devuelve los datos de la primera fila de df_jacarandas que corresponde al índice 165 (lo dice en la última línea). También podemos acceder a rebanadas (slices) usando iloc:

```
>>> cant_ejemplares.iloc[0:3]
Eucalipto      4112
Tipa blanca    4031
Jacarandá      3255
  Name: nombre_com, dtype: int64
```

Por otra parte, podemos seleccionar tanto filas como columnas, si separamos con comas las respectivas selecciones:

```
>>> df_jacarandas.iloc[-5:,2]
51104     4
51172     8
51180     0
51207     0
51375    20
Name: inclinacio, dtype: int64
```

Esto nos devuelve los datos correspondientes a las últimas 5 filas y a la tercera columna ('inclinacio').

Siempre vienen acompañados del índice.

Si queremos seleccionar una sola columna podemos especificarla por medio de su nombre. Recordemos que al tomar una sola columna obtenemos una serie en lugar de un DataFrame:

```
>>> df_jacarandas_diam = df_jacarandas['diametro']

>>> type(df_jacarandas)

pandas.core.frame.DataFrame

>>> type(df_jacarandas_diam)

pandas.core.series.Series
```

Ejercicios

Ejercicios

Utilizar el dataset de arbolado porteño en parques.

Cargar en un dataframe data_arboles_parques la información del archivo csv.

Armar un dataframe que contenga las filas de Jacarandás y otro con los Palos Borrachos.

Calcular para cada especie seleccionada:

Cantidad de árboles, altura máxima, mínima y promedio, diámetro máximo, mínimo y promedio.

Definir una función cantidad_arboles(parque) que, dado el nombre de un parque, calcule la cantidad de árboles que tiene.

Definir una función cantidad_nativos (parque) que calcule la cantidad de árboles nativos.

Ejercicios

Vamos a trabajar ahora con el archivo 'arbolado-publico-lineal-2017-2018.csv'.

Levantalo y armá un DataFrame data_arboles_veredas que tenga solamente las siguiente columnas:

```
cols_sel = ['nombre_cientifico', 'ancho_acera', 'diametro_altura_pecho', 'altura_arbol']
```

Imprimí las diez especies más frecuentes con sus respectivas cantidades.

Trabajaremos con las siguientes especies seleccionadas:

```
especies_seleccionadas = ['Tilia x moltkei', 'Jacaranda mimosifolia', 'Tipuana tipu']
```

Una forma de seleccionarlas es la siguiente:

```
df_lineal_seleccion = df_lineal[df_lineal['nombre_cientifico'].isin(especies_seleccionadas)
```

Ejercicios

Queremos estudiar si hay diferencias entre los ejemplares de una misma especie según si crecen en un sitio o en otro. Para eso tendremos que juntar datos de dos bases de datos diferentes.

El GCBA usa en un dataset 'altura_tot', 'diametro' y 'nombre_cie' para las alturas, diámetros y nombres científicos de los ejemplares, y en el otro dataset usa 'altura_arbol', 'diametro_altura_pecho' y 'nombre_cientifico' para los mismos datos.

Es más, los nombres científicos varían de un dataset al otro. 'Tipuana Tipu' se transforma en 'Tipuana tipu' y 'Jacarandá mimosifolia' en 'Jacaranda mimosifolia'. Obviamente son cambios menores pero suficientes para desalentar al usuarie desprevenide.

Te proponemos los siguientes pasos para comparar los diámetros a la altura del pecho de las tipas en ambos tipos de entornos.

Ejercicios

- Para cada dataset armate otro seleccionando solamente las filas correspondientes a las tipas (llamalos df_tipas_parques y df_tipas_veredas, respectivamente) y las columnas correspondientes al diametro a la altura del pecho y alturas. Hacelo como copias (usando .copy() como hicimos más arriba) para poder trabajar en estos nuevos dataframes sin modificar los dataframes grandes originales. Renombrá las columnas que muestran la altura y el diámetro a la altura del pecho para que se llamen igual en ambos dataframes, para ello explorá el comando rename.
- Agregale a cada dataframe (df_tipas_parques y df_tipas_veredas) una columna llamada 'ambiente' que en un caso valga siempre 'parque' y en el otro caso 'vereda'.
- Juntá ambos datasets con el comando df_tipas = pd.concat([df_tipas_veredas, df_tipas_parques]). De esta forma tenemos en un mismo dataframe la información de las tipas distinguidas por ambiente.