**8. Fechas, Carpetas y Pandas**

En esta clase introducimos el módulo datetime para manejar datos relacionados con el tiempo (Sección 2) y un par de funciones del módulo os para leer directorios, procesar archivos y realizar algunas tareas relacionadas con el sistema operativo (Sección 3). Luego te proponemos integrar esto para hacer un script que corra desde línea de comandos y te permita ordenar los archivos de cierto tipo (Sección 4).

En la segunda mitad introducimos el módulo Pandas y el tipo DataFrame así como algunos de sus métodos elementales. Usamos pandas para analizar dos datasets de Arbolado Porteño graficando algunos de sus datos. En esta parte tenés que comparar caracterísiticas de árboles que crecen en los parques con otros que crecen en las veredas (Sección 5 y ejercicio de revisión por pares).

En la última parte te proponemos análizar ondas de mareas en el Río de la Plata como excusa para introducir el procesamiento de series temporales. Nos metemos un poco en temas específicos con una última parte optativa que incluye un breve análsis de la transformada de Fourier para medir el tiempo que tarda esta onda de marea en trasladarse de un puerto a otro.

* [8.1 Introducción](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/01_Intro.md)
* [8.2 Manejo de fechas y horas](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/02_Fechas.md)
* [8.3 Manejo de archivos y carpetas](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/03_Archivos_y_Directorios.md)
* [8.4 Ordenar archivos en Python](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/04_Ordenando_archivos.md)
* [8.5 Introducción a Pandas](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/05_Pandas.md)
* [8.6 Series temporales](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/06_Series_Temporales.md)
* [8.7 Cierre de la clase](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/07_Cierre.md)

**8.1 Introducción**

En [este video](https://youtu.be/bZZVMLD2GDs) damos una breve introducción a los temas de la clase. Luego, en cada sección vas a encontrar también un video corto con mayor nivel de detalle sobre los temas de la sección. En estos videos usamos el código de [este archivo](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/clase8.py).

¡Que disfrutes la clase!

**8.2 Manejo de fechas y horas**

Esta sección tiene un breve [video introductorio](https://youtu.be/exTngw8oOp0) sobre el módulo datetime.

**El módulo datetime**

A continuación introducimos el módulo datetime que permite trabajar con fechas y horas. Este módulo define un nuevo tipo de objeto: datetime (sí, con el mismo nombre del módulo), que permite representar un instante temporal (fecha y hora). También define objetos de tipo date para representar sólo una fecha y de tipo time para guardar y trabajar con horarios. Finalmente, en esta breve introducción al módulo datetime mencionamos el tipo timedelta que se usa para representar diferencias entre instantes de tiempos, es decir, duraciones y trabajar con ellas.

**Ejemplo: Obtener fecha y hora actuales**

>>> import datetime

>>> fecha\_hora = datetime.datetime.now()

>>> print(fecha\_hora)

2020-09-24 10:03:18.636670

Lo que hicimos fue importar el módulo **datetime** y usar el método now() de la clase datetime del módulo (con el mismo nombre) para crear el objeto fecha\_hora que va a contener la fecha y la hora actuales.

**Ejemplo: Obtener fecha actual**

Análogamente, podemos obtener solo la fecha:

>>> fecha = datetime.date.today()

>>> print(fecha)

2020-09-24

Acá usamos el método today() de la clase date para obtener la fecha actual.

**¿Qué hay dentro del módulo datetime?**

En Python podemos usar la función dir() para obtener una lista de todos los atributos de un módulo.

>>> print(dir(datetime))

['MAXYEAR', 'MINYEAR', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_',

'\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_divide\_and\_round',

'date', 'datetime', 'datetime\_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']

Nos vamos a concentrar en lo más usado de datetime:

* date
* time
* datetime
* timedelta

**La clase datetime.date**

Podés generar objetos de tipo fecha con la clase date. Un objeto de esta clase representa una fecha (año, mes, día).

**Ejemplo: Un objeto para representar una fecha**

>>> d = datetime.date(2019, 4, 13)

>>> print(d)

2019-04-13

El comando date() de este ejemplo construye un objeto de tipo date. Este *constructor* toma tres argumentos: año, mes y día.

La variable d es un objeto de tipo date (es decir, representa una fecha).

También podríamos importar directamente la clase date del módulo datetime:

>>> from datetime import date

>>>

>>> d = date(2019, 4, 13)

>>> print(d)

2019-04-13

**Ejemplo: Obtener la fecha a partir de un timestamp**

En los sistemas operativos derivados de Unix (Mac OS X, Linux, etc.) se toma como medida de tiempo el número de segundos transcurridos desde el primero de enero de 1970 a las 0 horas UTC hasta el momento a representar. Se lo conoce como Unix timestamp. Podés convertir un timestamp a fecha usando el método fromtimestamp().

>>> from datetime import date

>>>

>>> timestamp = date.fromtimestamp(1326244364)

>>> print('Fecha =', timestamp)

Fecha = 2012-01-10

Esto es importante porque las fechas de modificación de los archivos usan timestamps por ejemplo.

**Ejemplo: Obtener el año, el mes y el día por separado.**

Así podés obtener el año, el mes, el día y el día de la semana:

from datetime import date

hoy = date.today()

print('Año actual:', hoy.year)

print('Mes actual:', hoy.month)

print('Día actual:', hoy.day)

print('Día de la semana:', hoy.weekday()) # va de 0 a 6 empezando en lunes

**La clase datetime.time**

Un objeto de la clase time representa la hora local (de como este configurada tu computadora). No nos vamos a meter en esta clase con los [husos horarios](https://es.wikipedia.org/wiki/Huso_horario) (conocido también como timezones), pero si vas a usar datos provistos por otres, es importante que sepas si está expresado en tu hora local, en la hora local de otro lugar o en [UTC](https://es.wikipedia.org/wiki/Tiempo_universal_coordinado).

**Ejemplo: Representar la hora con un objeto time**

La clase time se usa para representar horarios. A continuación damos algunos ejemplos de constructores de esta clase (un constructor es una forma de construir un objeto de una clase dada, una forma de inicializarlo, digamos).

>>> from datetime import time

>>>

>>> a = time() # time(hour = 0, minute = 0, second = 0)

>>> print('a =', a)

a = 00:00:00

>>> b = time(11, 34, 56)

>>> print('b =', b)

b = 11:34:56

>>> c = time(hour = 11, minute = 34, second = 56)

>>> print('c =', c)

c = 11:34:56

>>> d = time(11, 34, 56, 234566) # time(hour, minute, second, microsecond)

>>> print('d =', d)

d = 11:34:56.234566

**Ejemplo: Obtener horas, minutos, segundos y micro-segundos**

Una vez que creaste un objeto time, podés extraer sus atributos así:

from datetime import time

a = time(11, 34, 56)

print('hour =', a.hour)

print('minute =', a.minute)

print('second =', a.second)

print('microsecond =', a.microsecond)

Como no le pasaste ningún valor para el argumento microsecond, éste va a tomar el valor predeterminado, que es 0.

**La clase datetime.datetime**

Como ya mencionamos, el módulo datetime tiene una clase con su mismo nombre que permite almacenar información de fecha y hora en un solo objeto.

**Ejemplo: Objeto datetime**

>>> from datetime import datetime

>>> # datetime(year, month, day)

>>> a = datetime(2020, 4, 21)

>>> print(a)

2020-04-21 00:00:00

>>> # datetime(year, month, day, hour, minute, second, microsecond)

>>> b = datetime(2021, 4, 21, 6, 53, 31, 342260)

>>> print(b)

2021-04-21 06:53:31.342260

Los primeros tres argumentos, year, month y day del constructor datetime() son obligatorios. Los otros tienen a 0 como valor por omisión.

**Ejemplo: Obtener año, mes, día, hora, minutos, timestamp de un datetime**

El siguiente código genera un objeto datetime con valores pasados por parámetro y luego imprime la información.

En particular, muestra cómo convertir una fecha a timestamp. En general los timestamps son enteros y no tienen en cuenta las décimas de segundos.

from datetime import datetime

a = datetime(2021, 4, 21, 6, 53, 31, 342260)

print('año =', a.year)

print('mes =', a.month)

print('día =', a.day)

print('hora =', a.hour)

print('minuto =', a.minute)

print('timestamp =', a.timestamp())

**La clase datetime.timedelta**

Un objeto timedelta representa una duración, es decir, la diferencia entre dos instantes de tiempo.

**Ejemplo: Diferencia entre fechas y horarios**

>>> from datetime import datetime, date

>>> t1 = date(year = 2021, month = 4, day = 21)

>>> t2 = date(year = 2020, month = 8, day = 23)

>>> t3 = t1 - t2

>>> print(t3)

241 days, 0:00:00

>>> t4 = datetime(year = 2020, month = 7, day = 12, hour = 7, minute = 9, second = 33)

>>> t5 = datetime(year = 2021, month = 6, day = 10, hour = 5, minute = 55, second = 13)

>>> t6 = t4 - t5

>>> print(t6)

-333 days, 1:14:20

>>> print('tipo de t3 =', type(t3))

tipo de t3 = <class 'datetime.timedelta'>

>>> print('tipo de t6 =', type(t6))

tipo de t6 = <class 'datetime.timedelta'>

Observá que t3 y t6 son de tipo <class 'datetime.timedelta'>.

**Ejemplo: Diferencia entre objetos timedelta**

>>> from datetime import timedelta

>>> t1 = timedelta(weeks = 1, days = 2, hours = 1, seconds = 33)

>>> t2 = timedelta(days = 6, hours = 11, minutes = 4, seconds = 54)

>>> t3 = t1 - t2

>>> print('t3 =', t3)

2 days, 13:55:39

t3 también es de tipo <class 'datetime.timedelta'>.

**Ejemplo: Imprimir objetos timedelta negativos**

>>> from datetime import timedelta

>>> t1 = timedelta(seconds = 21)

>>> t2 = timedelta(seconds = 55)

>>> t3 = t1 - t2

>>> print(t3)

-1 day, 23:59:26

>>> print(abs(t3))

0:00:34

**Ejemplo: Duración en segundos**

Podés obtener el tiempo medido en segundos usando el método total\_seconds().

>>> from datetime import timedelta

>>> t = timedelta(days = 1, hours = 2, seconds = 30, microseconds = 100000)

>>> print('segundos totales =', t.total\_seconds())

segundos totales = 93630.1

También podés sumar fechas y horarios usando el operador +. También podés multiplicar o dividir un objeto timedelta por números enteros o floats.

**Formato para fechas y horas**

Hay diversas formas de representar el tiempo, que varían según el lugar, la organización, etc. Por ejemplo, en Argentina solemos usar dd/mm/yyyy, mientras que en las culturas anglosajonas es más común usar mm/dd/yyyy.

En Python tenemos los métodos strftime() y strptime() para manejar esto.

**Python strftime() - convertir un objeto datetime a string**

El método strftime() está definido en las clases date, datetime y time. Este método crea una cadena con formato a partir estos objetos.

**Ejemplo: Formato de fecha usando strftime()**

>>> from datetime import datetime

>>> now = datetime.now()

>>> t = now.strftime('%H:%M:%S')

>>> print('hora:', t)

hora: 14:40:06

>>> s1 = now.strftime('%m/%d/%Y, %H:%M:%S')

>>> # en formato mm/dd/YY H:M:S

>>> print('s1:', s1)

s1: 09/24/2020, 14:40:06

>>> s2 = now.strftime('%d/%m/%Y, %H:%M:%S')

>>> # en formato dd/mm/YY H:M:S

>>> print('s2:', s2)

s2: 24/09/2020, 14:40:06

Acá, %Y, %m, %d, %H etc. son códigos de formato. El método strftime()toma uno o más códigos de formato y devuelve la cadena con formato basado en esos códigos.

En el programa de arriba, t, s1 y s2 son cadenas. Y los códigos de formato son:

* %Y - año [0001,..., 2018, 2019,..., 9999]
* %m - mes [01, 02, ..., 11, 12]
* %d - día [01, 02, ..., 30, 31]
* %H - hora [00, 01, ..., 22, 23
* %M - minuto [00, 01, ..., 58, 59]
* %S - segundo [00, 01, ..., 58, 59]

Para aprender más sobre strftime() visitá [la documentación](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior).

**Python strptime() - convertir una cadena a un objeto datetime**

El método strptime() crea un objeto datetime a partir de una cadena.

**Ejemplo: strptime()**

>>> from datetime import datetime

>>> cadena\_con\_fecha= '21 September, 2021'

>>> print('date\_string =', cadena\_con\_fecha)

date\_string = 21 September, 2021

>>> date\_object = datetime.strptime(cadena\_con\_fecha, '%d %B, %Y')

>>> print('date\_object =', date\_object)

date\_object = 2021-09-21 00:00:00

El método strptime() toma dos argumentos:

* una cadena que representa una fecha y hora
* un código de formato correspondiente al primer argumento

Los códigos de formato %d, %B, %Y significan day, month (full name) y year respectivamente.

Visitá [la documentación](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior) para más detalles.

**Ejercicios:**

**Ejercicio 8.1: Segundos vividos**

Escribí una función a la que le pasás tu fecha de nacimiento como cadena en formato 'dd/mm/AAAA' (día, mes, año con 2, 2 y 4 dígitos, separados con barras normales) y te devuelve la cantidad de segundos que viviste (asumiendo que naciste a las 00:00hs de tu fecha de nacimiento).

Guardá este código en el archivo vida.py.

**Ejercicio 8.2: Cuánto falta**

Un conocido canal Argentino tiene por costumbre anunciar la cantidad de días que faltan para la próxima primavera.

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/cronica.jpg)

Escribí un programa que asista a los técnicos del canal indicándoles, al correr el programa el número que deben poner en la placa.

**Ejercicio 8.3: Fecha de reincorporación**

Si tenés una licencia por xaternidad que empieza el 26 de septiembre de 2020 y dura 200 días, ¿qué día te reincorporás al trabajo?

**Ejercicio 8.4: Días hábiles**

Escribí una función dias\_habiles(inicio, fin, feriados) que calcule los días hábiles entre dos fechas dadas. La función debe tener como argumentos el día inicial, el día final, y una lista con las fechas correspondientes a los feriados que haya en ese lapso, y debe devolver una lista con las fechas de días hábiles del período, incluyendo la fecha inicial y la fecha final indicadas. Las fechas de entrada y salida deben manejarse en formato de texto.

Consideramos día hábil a un día que no es feriado ni sábado ni domingo.

Probala entre el 20 de septiembre de 2020 y el 10 de octubre del mismo año, sabiendo que no hubo feriados en el medio. Probala entre ese día y fin del 2020 considerando los siguientes feriados de Argentina:

feriados = ['12/10/2020', '23/11/2020', '7/12/2020', '8/12/2020', '25/12/2020']

**8.2 Manejo de fechas y horas**

Esta sección tiene un breve [video introductorio](https://youtu.be/exTngw8oOp0) sobre el módulo datetime.

**El módulo datetime**

A continuación introducimos el módulo datetime que permite trabajar con fechas y horas. Este módulo define un nuevo tipo de objeto: datetime (sí, con el mismo nombre del módulo), que permite representar un instante temporal (fecha y hora). También define objetos de tipo date para representar sólo una fecha y de tipo time para guardar y trabajar con horarios. Finalmente, en esta breve introducción al módulo datetime mencionamos el tipo timedelta que se usa para representar diferencias entre instantes de tiempos, es decir, duraciones y trabajar con ellas.

**Ejemplo: Obtener fecha y hora actuales**

>>> import datetime

>>> fecha\_hora = datetime.datetime.now()

>>> print(fecha\_hora)

2020-09-24 10:03:18.636670

Lo que hicimos fue importar el módulo **datetime** y usar el método now() de la clase datetime del módulo (con el mismo nombre) para crear el objeto fecha\_hora que va a contener la fecha y la hora actuales.

**Ejemplo: Obtener fecha actual**

Análogamente, podemos obtener solo la fecha:

>>> fecha = datetime.date.today()

>>> print(fecha)

2020-09-24

Acá usamos el método today() de la clase date para obtener la fecha actual.

**¿Qué hay dentro del módulo datetime?**

En Python podemos usar la función dir() para obtener una lista de todos los atributos de un módulo.

>>> print(dir(datetime))

['MAXYEAR', 'MINYEAR', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_',

'\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_divide\_and\_round',

'date', 'datetime', 'datetime\_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']

Nos vamos a concentrar en lo más usado de datetime:

* date
* time
* datetime
* timedelta

**La clase datetime.date**

Podés generar objetos de tipo fecha con la clase date. Un objeto de esta clase representa una fecha (año, mes, día).

**Ejemplo: Un objeto para representar una fecha**

>>> d = datetime.date(2019, 4, 13)

>>> print(d)

2019-04-13

El comando date() de este ejemplo construye un objeto de tipo date. Este *constructor* toma tres argumentos: año, mes y día.

La variable d es un objeto de tipo date (es decir, representa una fecha).

También podríamos importar directamente la clase date del módulo datetime:

>>> from datetime import date

>>>

>>> d = date(2019, 4, 13)

>>> print(d)

2019-04-13

**Ejemplo: Obtener la fecha a partir de un timestamp**

En los sistemas operativos derivados de Unix (Mac OS X, Linux, etc.) se toma como medida de tiempo el número de segundos transcurridos desde el primero de enero de 1970 a las 0 horas UTC hasta el momento a representar. Se lo conoce como Unix timestamp. Podés convertir un timestamp a fecha usando el método fromtimestamp().

>>> from datetime import date

>>>

>>> timestamp = date.fromtimestamp(1326244364)

>>> print('Fecha =', timestamp)

Fecha = 2012-01-10

Esto es importante porque las fechas de modificación de los archivos usan timestamps por ejemplo.

**Ejemplo: Obtener el año, el mes y el día por separado.**

Así podés obtener el año, el mes, el día y el día de la semana:

from datetime import date

hoy = date.today()

print('Año actual:', hoy.year)

print('Mes actual:', hoy.month)

print('Día actual:', hoy.day)

print('Día de la semana:', hoy.weekday()) # va de 0 a 6 empezando en lunes

**La clase datetime.time**

Un objeto de la clase time representa la hora local (de como este configurada tu computadora). No nos vamos a meter en esta clase con los [husos horarios](https://es.wikipedia.org/wiki/Huso_horario) (conocido también como timezones), pero si vas a usar datos provistos por otres, es importante que sepas si está expresado en tu hora local, en la hora local de otro lugar o en [UTC](https://es.wikipedia.org/wiki/Tiempo_universal_coordinado).

**Ejemplo: Representar la hora con un objeto time**

La clase time se usa para representar horarios. A continuación damos algunos ejemplos de constructores de esta clase (un constructor es una forma de construir un objeto de una clase dada, una forma de inicializarlo, digamos).

>>> from datetime import time

>>>

>>> a = time() # time(hour = 0, minute = 0, second = 0)

>>> print('a =', a)

a = 00:00:00

>>> b = time(11, 34, 56)

>>> print('b =', b)

b = 11:34:56

>>> c = time(hour = 11, minute = 34, second = 56)

>>> print('c =', c)

c = 11:34:56

>>> d = time(11, 34, 56, 234566) # time(hour, minute, second, microsecond)

>>> print('d =', d)

d = 11:34:56.234566

**Ejemplo: Obtener horas, minutos, segundos y micro-segundos**

Una vez que creaste un objeto time, podés extraer sus atributos así:

from datetime import time

a = time(11, 34, 56)

print('hour =', a.hour)

print('minute =', a.minute)

print('second =', a.second)

print('microsecond =', a.microsecond)

Como no le pasaste ningún valor para el argumento microsecond, éste va a tomar el valor predeterminado, que es 0.

**La clase datetime.datetime**

Como ya mencionamos, el módulo datetime tiene una clase con su mismo nombre que permite almacenar información de fecha y hora en un solo objeto.

**Ejemplo: Objeto datetime**

>>> from datetime import datetime

>>> # datetime(year, month, day)

>>> a = datetime(2020, 4, 21)

>>> print(a)

2020-04-21 00:00:00

>>> # datetime(year, month, day, hour, minute, second, microsecond)

>>> b = datetime(2021, 4, 21, 6, 53, 31, 342260)

>>> print(b)

2021-04-21 06:53:31.342260

Los primeros tres argumentos, year, month y day del constructor datetime() son obligatorios. Los otros tienen a 0 como valor por omisión.

**Ejemplo: Obtener año, mes, día, hora, minutos, timestamp de un datetime**

El siguiente código genera un objeto datetime con valores pasados por parámetro y luego imprime la información.

En particular, muestra cómo convertir una fecha a timestamp. En general los timestamps son enteros y no tienen en cuenta las décimas de segundos.

from datetime import datetime

a = datetime(2021, 4, 21, 6, 53, 31, 342260)

print('año =', a.year)

print('mes =', a.month)

print('día =', a.day)

print('hora =', a.hour)

print('minuto =', a.minute)

print('timestamp =', a.timestamp())

**La clase datetime.timedelta**

Un objeto timedelta representa una duración, es decir, la diferencia entre dos instantes de tiempo.

**Ejemplo: Diferencia entre fechas y horarios**

>>> from datetime import datetime, date

>>> t1 = date(year = 2021, month = 4, day = 21)

>>> t2 = date(year = 2020, month = 8, day = 23)

>>> t3 = t1 - t2

>>> print(t3)

241 days, 0:00:00

>>> t4 = datetime(year = 2020, month = 7, day = 12, hour = 7, minute = 9, second = 33)

>>> t5 = datetime(year = 2021, month = 6, day = 10, hour = 5, minute = 55, second = 13)

>>> t6 = t4 - t5

>>> print(t6)

-333 days, 1:14:20

>>> print('tipo de t3 =', type(t3))

tipo de t3 = <class 'datetime.timedelta'>

>>> print('tipo de t6 =', type(t6))

tipo de t6 = <class 'datetime.timedelta'>

Observá que t3 y t6 son de tipo <class 'datetime.timedelta'>.

**Ejemplo: Diferencia entre objetos timedelta**

>>> from datetime import timedelta

>>> t1 = timedelta(weeks = 1, days = 2, hours = 1, seconds = 33)

>>> t2 = timedelta(days = 6, hours = 11, minutes = 4, seconds = 54)

>>> t3 = t1 - t2

>>> print('t3 =', t3)

2 days, 13:55:39

t3 también es de tipo <class 'datetime.timedelta'>.

**Ejemplo: Imprimir objetos timedelta negativos**

>>> from datetime import timedelta

>>> t1 = timedelta(seconds = 21)

>>> t2 = timedelta(seconds = 55)

>>> t3 = t1 - t2

>>> print(t3)

-1 day, 23:59:26

>>> print(abs(t3))

0:00:34

**Ejemplo: Duración en segundos**

Podés obtener el tiempo medido en segundos usando el método total\_seconds().

>>> from datetime import timedelta

>>> t = timedelta(days = 1, hours = 2, seconds = 30, microseconds = 100000)

>>> print('segundos totales =', t.total\_seconds())

segundos totales = 93630.1

También podés sumar fechas y horarios usando el operador +. También podés multiplicar o dividir un objeto timedelta por números enteros o floats.

**Formato para fechas y horas**

Hay diversas formas de representar el tiempo, que varían según el lugar, la organización, etc. Por ejemplo, en Argentina solemos usar dd/mm/yyyy, mientras que en las culturas anglosajonas es más común usar mm/dd/yyyy.

En Python tenemos los métodos strftime() y strptime() para manejar esto.

**Python strftime() - convertir un objeto datetime a string**

El método strftime() está definido en las clases date, datetime y time. Este método crea una cadena con formato a partir estos objetos.

**Ejemplo: Formato de fecha usando strftime()**

>>> from datetime import datetime

>>> now = datetime.now()

>>> t = now.strftime('%H:%M:%S')

>>> print('hora:', t)

hora: 14:40:06

>>> s1 = now.strftime('%m/%d/%Y, %H:%M:%S')

>>> # en formato mm/dd/YY H:M:S

>>> print('s1:', s1)

s1: 09/24/2020, 14:40:06

>>> s2 = now.strftime('%d/%m/%Y, %H:%M:%S')

>>> # en formato dd/mm/YY H:M:S

>>> print('s2:', s2)

s2: 24/09/2020, 14:40:06

Acá, %Y, %m, %d, %H etc. son códigos de formato. El método strftime()toma uno o más códigos de formato y devuelve la cadena con formato basado en esos códigos.

En el programa de arriba, t, s1 y s2 son cadenas. Y los códigos de formato son:

* %Y - año [0001,..., 2018, 2019,..., 9999]
* %m - mes [01, 02, ..., 11, 12]
* %d - día [01, 02, ..., 30, 31]
* %H - hora [00, 01, ..., 22, 23
* %M - minuto [00, 01, ..., 58, 59]
* %S - segundo [00, 01, ..., 58, 59]

Para aprender más sobre strftime() visitá [la documentación](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior).

**Python strptime() - convertir una cadena a un objeto datetime**

El método strptime() crea un objeto datetime a partir de una cadena.

**Ejemplo: strptime()**

>>> from datetime import datetime

>>> cadena\_con\_fecha= '21 September, 2021'

>>> print('date\_string =', cadena\_con\_fecha)

date\_string = 21 September, 2021

>>> date\_object = datetime.strptime(cadena\_con\_fecha, '%d %B, %Y')

>>> print('date\_object =', date\_object)

date\_object = 2021-09-21 00:00:00

El método strptime() toma dos argumentos:

* una cadena que representa una fecha y hora
* un código de formato correspondiente al primer argumento

Los códigos de formato %d, %B, %Y significan day, month (full name) y year respectivamente.

Visitá [la documentación](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior) para más detalles.

**Ejercicios:**

**Ejercicio 8.1: Segundos vividos**

Escribí una función a la que le pasás tu fecha de nacimiento como cadena en formato 'dd/mm/AAAA' (día, mes, año con 2, 2 y 4 dígitos, separados con barras normales) y te devuelve la cantidad de segundos que viviste (asumiendo que naciste a las 00:00hs de tu fecha de nacimiento).

Guardá este código en el archivo vida.py.

**Ejercicio 8.2: Cuánto falta**

Un conocido canal Argentino tiene por costumbre anunciar la cantidad de días que faltan para la próxima primavera.

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/cronica.jpg)

Escribí un programa que asista a los técnicos del canal indicándoles, al correr el programa el número que deben poner en la placa.

**Ejercicio 8.3: Fecha de reincorporación**

Si tenés una licencia por xaternidad que empieza el 26 de septiembre de 2020 y dura 200 días, ¿qué día te reincorporás al trabajo?

**Ejercicio 8.4: Días hábiles**

Escribí una función dias\_habiles(inicio, fin, feriados) que calcule los días hábiles entre dos fechas dadas. La función debe tener como argumentos el día inicial, el día final, y una lista con las fechas correspondientes a los feriados que haya en ese lapso, y debe devolver una lista con las fechas de días hábiles del período, incluyendo la fecha inicial y la fecha final indicadas. Las fechas de entrada y salida deben manejarse en formato de texto.

Consideramos día hábil a un día que no es feriado ni sábado ni domingo.

Probala entre el 20 de septiembre de 2020 y el 10 de octubre del mismo año, sabiendo que no hubo feriados en el medio. Probala entre ese día y fin del 2020 considerando los siguientes feriados de Argentina:

feriados = ['12/10/2020', '23/11/2020', '7/12/2020', '8/12/2020', '25/12/2020']

**8.4 Ordenar archivos en Python**

En esta sección vamos a integrar las últimas dos secciones con lo que veníamos viendo antes del parcial. La idea es que descomprimas [este archivo](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/ordenar.zip) en tu carpeta ../Data/ y escribas un script que trabaje con estos archivos.

Esta sección tiene un ejercicio para entregar y luego otro más complejo que es optativo.

**Ejercicio 8.5: Recorrer el árbol de archivos**

Escribí un programa que dado un directorio, imprima en pantalla los nombres de todos los archivos .png que se encuentren en *algún* subdirectorio del él.

*Observación:* Al final, tu script debería poder ejecutarse desde la línea de comandos recibiendo como parámetro el directorio a leer original. En la [Sección 7.3](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/07_Plt_Especificacion_y_Documentacion/03_Modulo_principal.md#modelo-de-script-con-par%C3%A1metros) dimos un modelo de script que te puede servir.

Guardá el script resultante en un archivo listar\_imgs.py.

**Ejercicio 8.6: Ordenar el árbol de archivos (optativo)**

Escribí un programa que te permita ordenar las imágenes PNG de esta carpeta. Guardalo en un archivo ordenar\_imgs.py.

1. Creá un nuevo directorio ../Data/imgs\_procesadas/.
2. Usá os.walk() para recorrer los archivos en la carpeta ../Data/ordenar/ (y sus subcarpetas).
3. Cuando encuentres archivos con extensión png los vas a *procesar*. En este caso *procesar* significa:
   * Leer la fecha que se encuentra codificada en los últimos 8 caracteres de su nombre en el formato AAAAMMDD (año en 4 dígitos, mes en 2 y día en 2).
   * Usar la fecha obtenida para setear la fecha de última modificación (y de último acceso si no usás Windows).
   * Cambiarle el nombre al archivo para que no tenga más esos dígitos (ni el guión bajo).
   * Mover el archivo a la carpeta ../Data/imgs\_procesadas/.
4. Los archivos que no son png no los modifiques.
5. Borrá todas las subcarpetas de ../Data/ordenar/ que hayan quedado vacías.

*Observación:* Al final, tu script debería poder ejecutarse desde la línea de comandos recibiendo como parámetro el directorio a leer original y un directorio destino (que debería ser creado si no existe).

*Observación:* Este tipo de tareas se repite con mucha frecuencia. Tener la capacidad de automatizarlas mediante un script de Python te puede ahorrar muchísimo tiempo.

**Algunos puntos importantes:**

* Te recomendamos que modularices el procesamiento de los archivos png. Podés, por ejemplo, escribir una función que manipule strings (que tome el nombre de un archivo y devuelva la fecha y el nombre corregido) y otra función que precese cada archivo (que use la función anterior para renombrar, mover y modificar la fecha de cada archivo). La modularización del código es clave para que otras personas lo puedan entender y que sea sencillo de mantener.
* Usá docstrings y comentarios en tu código de manera que sea legible.

**8.5 Introducción a Pandas**

La biblioteca Pandas es una extensión de NumPy para manipulación y análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas de datos (numéricos y de otros tipos) y series temporales. Se distribuye como software libre.

Ésta es una breve introducción a [Pandas](https://pandas.pydata.org/docs/getting_started/index.html). Para información más completa, te recomendamos consultar [la documentación oficial](https://pandas.pydata.org/docs/user_guide/10min.html).

Esta biblioteca tiene dos tipos de datos fundamentales: los DataFrames que almacenan tablas de datos y las series que contienen secuencias de datos.

Esta sección tiene un breve [video introductorio](https://youtu.be/aQhssTeFnYY) sobre las posibilidades que ofrece la biblioteca Pandas.

**Lectura de datos**

Pandas permite leer diversos formatos de tablas de datos directamente. Probá el siguiente código, para leer un archivo CSV:

import pandas as pd

import os

directorio = '../Data'

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.

>>> df.head()

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

Usando df.columns pandas te va a devolver un índice con los nombres de las columnas del DataFrame. Recordá que en la [Sección 3.6](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/03_Datos/06_Arboles1.md#descripci%C3%B3n-de-la-base) describimos la base de datos. A su vez, df.index te mostrará el índice. En este caso el índice es numérico y se corresponde con el número de la línea leida del archivo. En principio no es muy interesante para analizar cuestiones de árboles, simplemente tenemos las filas numeradas. Veremos otros ejemplos donde el índice puede contener información vital (una categoría, un timestamp, etc).

>>> 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)

Otra herramienta útil para inspeccionar los datos recién levantados es describe(). Para ver mejor una parte, podemos seleccionar algunas columnas de interés antes de pedirle la descripción.

>>> 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

**Selección**

Una de las operaciones primitivas más importantes es la selección de fragmentos de las tablas de datos, ya sean filas, columnas o rangos de filas y columnas.

Por ejemplo con df['nombre\_com'] veremos la columna (que es una serie) de nombres comunes de los árboles en la base. Podemos usar unique para ver una vez cada nombre:

>>> df['nombre\_com'].unique()

array(['Washingtonia (Palmera washingtonia)', 'Ombú', 'Catalpa', 'Ceibo',

'Brachichiton (Árbol botella, Brachichito)', 'Álamo plateado',

'Acacia de constantinopla', 'Acacia', 'Roble sedoso (Grevillea)',

...

'Jazmín del Paraguay', 'Plumerillo rojo', 'Árbol fuccia',

'Canela de venado', 'Boj cepillo', 'Caranday'], dtype=object)

Podemos preguntar cuáles se llaman de cierta manera ('Ombú' en este caso), como hacíamos con los ndarrays en numpy:

>>> df['nombre\_com'] == 'Ombú'

0 False

1 False

2 False

3 True

...

Observá que esto generó una serie. Podemos sumar los True de esta serie para contar la cantidad de Ombús:

>>> (df['nombre\_com'] == 'Ombú').sum()

590

Si queremos hacer lo mismo para otras especies podemos usar value\_counts()

>>> cant\_ejemplares = df['nombre\_com'].value\_counts()

>>> cant\_ejemplares.head(10)

Eucalipto 4112

Tipa blanca 4031

Jacarandá 3255

Palo borracho rosado 3150

Casuarina 2719

Fresno americano 2166

Plátano 1556

Ciprés 1467

Ceibo 1149

Pindó 1068

Name: nombre\_com, dtype: int64

De esta forma obtenemos, en orden decreciente, los nombres comunes y las cantidades de las especies más frecuentes en la base de datos.

**Filtros booleanos**

La serie booleana que obtuvimos con df['nombre\_com'] == 'Ombú' puede usarse para seleccionar esas filas del DataFrame. Probemos con Jacarandá:

>>> df\_jacarandas = df[df['nombre\_com'] == 'Jacarandá']

Análogamente, podemos seleccionar algunas columnas de interés y generar vistas (ojo, en estos casos no estamos copiando la información):

>>> 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

>>> df\_jacarandas.describe()

altura\_tot diametro inclinacio

count 3255.000000 3255.000000 3255.000000

mean 10.369585 28.804301 6.549923

std 5.905744 19.166388 8.459921

min 1.000000 1.000000 0.000000

25% 6.000000 14.000000 0.000000

50% 10.000000 25.000000 4.000000

75% 15.000000 41.000000 10.000000

max 49.000000 159.000000 70.000000

Observá que cuando le pedimos los últimos datos de df\_jacarandas nos mostró los últimos 5 jacarandás de la base de datos, respetando los números de índice de la tabla original (..., 51207, 51375).

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()

**Scatterplots**

Pandas también permite [hacer gráficos bonitos](https://pandas.pydata.org/docs/user_guide/visualization.html). Es realmente sencillo:

df\_jacarandas.plot.scatter(x = 'diametro', y = 'altura\_tot')

Hay otro módulo para hacer gráficos que interactúa muy bien con pandas y se llama [Seaborn](https://seaborn.pydata.org/). Está basado en matplotlib, y ofrece una interfaz de alto nivel para realizar gráficos estadísticos atractivos e informativos. En criollo: "usar pandas para manejar los datos y seaborn para visualizarlos, es la posta".

Fijate que seaborn entiende los DataFrames y las columnas y su sintaxis es muy similar a la de pandas:

import seaborn as sns

sns.scatterplot(data = df\_jacarandas, x = 'diametro', y = 'altura\_tot')

**Filtros por índice y por posición**

Como ya mencionamos, 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 (es como un DataFrame de una sola columna). Tiene los nombres de las especies como índice y sus respectivas cantidades como dato asociado.

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.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

espacio\_ve INDOAMERICANO

ubicacion LACARRA, Av. - ESCALADA, Av. - CASTAÑARES, Av....

nombre\_fam Bignoniáceas

nombre\_gen Jacarandá

origen Nativo/Autóctono

coord\_x 99534.3

coord\_y 96061.8

Name: 165, dtype: object

>>> cant\_ejemplares.loc['Eucalipto']

4112

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

>>> 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'). Fijate que siempre vienen acompañados del índice.

**Selección de una columna**

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

**Series temporales en Pandas**

Pandas tiene un gran potencial para el manejo de series temporales. Es muy sencillo crear índices con fechas y frecuencias seleccionadas.

>>> pd.date\_range('20200923', periods = 7)

DatetimeIndex(['2020-09-23', '2020-09-24', '2020-09-25', '2020-09-26',

'2020-09-27', '2020-09-28', '2020-09-29'],

dtype='datetime64[ns]', freq='D')

>>> pd.date\_range('20200923 14:00', periods = 7)

DatetimeIndex(['2020-09-23 14:00:00', '2020-09-24 14:00:00',

'2020-09-25 14:00:00', '2020-09-26 14:00:00',

'2020-09-27 14:00:00', '2020-09-28 14:00:00',

'2020-09-29 14:00:00'],

dtype='datetime64[ns]', freq='D')

>>> pd.date\_range('20200923 14:00', periods = 6, freq = 'H')

DatetimeIndex(['2020-09-23 14:00:00', '2020-09-23 15:00:00',

'2020-09-23 16:00:00', '2020-09-23 17:00:00',

'2020-09-23 18:00:00', '2020-09-23 19:00:00'],

dtype='datetime64[ns]', freq='H')

Luego, podés usar esos índices junto con datos para armar series temporales o DataFrames:

>>> pd.Series([1, 2, 3, 4, 5, 6], index = pd.date\_range('20200923 14:00', periods = 6, freq = 'H'))

2020-09-23 14:00:00 1

2020-09-23 15:00:00 2

2020-09-23 16:00:00 3

2020-09-23 17:00:00 4

2020-09-23 18:00:00 5

2020-09-23 19:00:00 6

Freq: H, dtype: int64

**Caminatas al azar**

Volviendo al tema de las caminatas al azar, podemos hacer una caminata de dos horas dando un paso por minuto a partir del comienzo de esta clase con el siguiente comando:

import numpy as np

idx = pd.date\_range('20200923 14:00', periods = 120, freq = 'min')

s1 = pd.Series(np.random.randint(-1,2,120), index = idx)

s2 = s1.cumsum()

Observá que estamos usando random del módulo numpy, no de random. La función np.random.randint(-1,2,120) genera un array de longitud 120 con valores -1, 0, 1 (no incluye extremo derecho del rango de valores).

Podemos ver el gráfico sencillamente:

s2.plot()

O usar una [media móvil](https://es.wikipedia.org/wiki/Media_m%C3%B3vil) (rolling mean) para suavizar los datos:

w = 5 # ancho en minutos de la ventana

s3 = s2.rolling(w).mean()

s3.plot()

Podés ver ambas curvas en un mismo gráfico para ver más claramente el efecto del suavizado:

df\_series\_23 = pd.DataFrame([s2, s3]).T # armo un dataframe con ambas series

df\_series\_23.plot()

Fijate que los datos de la curva suavizada empiezan más tarde, porque al principio no hay datos sobre los cuales hacer promedio. El parámetro min\_periods = 1 del método rolling te permite controlar esto. Probalo.

**Ejemplo: 12 personas caminando 8 horas**

En el siguiente ejemplo creamos un índice que contenga un elemento por minuto a partir del comienzo de la clase y durante 8 horas. Armamos también una lista de nombres.

horas = 8

idx = pd.date\_range('20200923 14:00', periods = horas\*60, freq = 'min')

nombres = ['Pedro', 'Santiago', 'Juan', 'Andrés','Bartolomé','Tiago','Isca','Tadeo','Mateo','Felipe','Simón','Tomás']

Luego usamos el módulo random de numpy para generar pasos para cada persona a cada minuto. Los acumulamos con cumsum y los acomodamos en un DataFrame, usando el índice generado antes y poniéndoles nombres adecuados a cada columna:

df\_walks = pd.DataFrame(np.random.randint(-1,2,[horas\*60,12]).cumsum(axis=0), index = idx, columns = nombres)

df\_walks.plot()

Ahora suavizamos los datos, usando min\_periods para no perder los datos de los extremos.

w = 45

df\_walk\_suav = df\_walks.rolling(w, min\_periods = 1).mean() # datos suavizados

nsuav = ['S\_' + n for n in nombres]

df\_walk\_suav.columns = nsuav # cambio el nombre de las columnas

# para los datos suavizados

df\_walk\_suav.plot()

**Guardando datos**

Guardar una serie o un DataFrame en el disco es algo realmente sencillo. Probá, por ejemplo, el efecto del comando df\_walk\_suav.to\_csv('caminata\_apostolica.csv').

**Incorporando el Arbolado lineal**

**Ejercicio 8.7: Lectura y selección**

Vamos a trabajar ahora con el archivo ['arbolado-publico-lineal-2017-2018.csv'](https://data.buenosaires.gob.ar/dataset/arbolado-publico-lineal). Descargalo y guardalo en tu directorio '../Data/'.

Levantalo y armá un DataFrame df\_lineal 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)]

**Ejercicio 8.8: Boxplots**

El siguiente comando realiza un [boxplot](https://es.wikipedia.org/wiki/Diagrama_de_caja) de los diámetros de los árboles agrupados por especie.

df\_lineal\_seleccion.boxplot('diametro\_altura\_pecho', by = 'nombre\_cientifico')

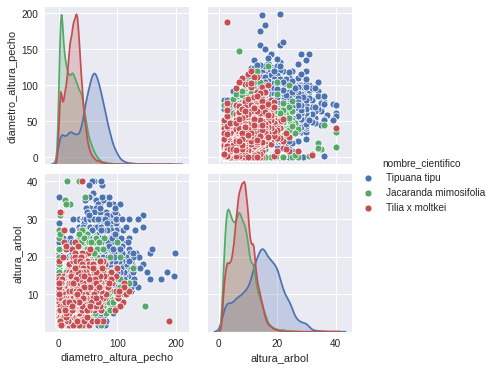
Realizá un gráfico similar pero de los altos en lugar de los diámetros de los árboles.

**Ejemplo de pairplot**

Otro gráfico interesante que resume muy bien la información es el *pairplot* de seaborn que es una grilla cuadrada de subplots.

Probá el siguiente código:

sns.pairplot(data = df\_lineal\_seleccion[cols\_sel], hue = 'nombre\_cientifico')

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure200027.png)

El gráfico va a tener una fila (y columna) por cada variable numérica en el DataFrame pasado como data. En la diagonal del gŕafico, va a haber kdeplots (kernel density estimation plots, una versión suavizada de los histogramas) y fuera de la diagonal scatterplots combinando todos los pares de variables (cada combinación aparece dos veces, una sobre y otra debajo de la diagonal).

El hue selecciona la variable categórica a usar para distinguir subgrupos y asociarles colores. En la diagonal de este ejemplo (y en los scatterplots también) se ve por ejemplo que las Tipas suelen ser más anchas y más altas que los Tilos y los Jacarandás.

Pregunta: ¿Por qué el ancho\_acera no tiene lugar en el gráfico?

*Te recomendamos pegarle una mirada a*[*esta página*](http://seaborn.pydata.org/introduction.html)*donde vas a poder ver un poco más sobre el potencial de seaborn.*

**Ejercicio 8.9: Comparando especies en parques y en veredas**

Al comienzo de la materia estuvimos trabajando con el dataset de árboles en parques. Ahora estuvimos analizando otro dataset: el de árboles en veredas. Ahora queremos estudiar si hay diferencias entre los ejemplares de una misma especie según si crecen en un sitio o en otro. Queremos hacer un boxplot del diámetro a la altura del pecho para las Tipas (su nombre científico es *tipuana tipu*), que crecen en ambos tipos de ambiente. Para eso tendremos que juntar datos de dos bases de datos diferentes.

Nos vamos en meter en un lío. 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.

En este ejercicio te proponemos los siguientes pasos para comparar los diámetros a la altura del pecho de las tipas en ambos tipos de entornos. Guardá este trabajo en un archivo arbolado\_parques\_veredas.py.

1. Abrí ambos datasets a los que llamaremos df\_parques y df\_veredas.
2. 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](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html).
3. 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'.
4. 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.
5. Creá un boxplot para los diámetros a la altura del pecho de la tipas distinguiendo los ambientes (boxplot('diametro\_altura\_pecho',by = 'ambiente')).
6. Repetí para alturas.
7. ¿Qué tendrías que cambiar para repetir el análisis para otras especies? ¿Convendría definir una función?

**8.6 Series temporales**

Para esta Sección contamos con el valioso aporte de [Octavio Bruzzone](https://inta.gob.ar/personas/bruzzone.octavio). Octavio da dos cursos de posgrado excelentes sobre Series Temporales en Python. Uno se enfoca en los análisis en el dominio del tiempo y el otro en el dominio de las frecuencias. Generosamente nos compartió algunas ideas para este trabajo práctico.

Esta sección tiene un breve [video introductorio](https://youtu.be/fLdVsU12sR8) sobre los datos y lo que hay que hacer en este TP.

**Análisis y visualización de series temporales.**

En este práctico vamos a visualizar y analizar datos de mareas en el Río de la Plata. Tiene una primera parte que esperamos que todes hagan y una segunda parte, más larga y compleja, optativa. Trabajá en el archivo mareas\_fft.py.

Para comenzar, copiate [el archivo](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/OBS_SHN_SF-BA.csv) con datos de mareas en los puertos de San Fernando y Buenos Aires a tu carpeta Data

**Lectura de archivos temporales**

import pandas as pd

df = pd.read\_csv('../Data/OBS\_SHN\_SF-BA.csv')

Observá los datos:

>>> df.head()

Time H\_SF H\_BA

0 2011-01-01 00:00:00 NaN 92.0

1 2011-01-01 01:00:00 NaN 110.0

2 2011-01-01 02:00:00 NaN 124.0

3 2011-01-01 03:00:00 NaN 132.0

4 2011-01-01 04:00:00 NaN 136.0

>>> df.index

RangeIndex(start=0, stop=35064, step=1)

Este archivo tiene alturas del agua en el puerto de San Fernando (columna H\_SF) y en el puerto de Buenos Aires (columna H\_BA) medidas en centímetros. Tiene un dato por hora (columna Time) durante cuatro años. En los primeros registros se observa algo muy frecuente con este tipo de archivos: tiene muchos datos faltantes.

El índice de un dataframe nos da información de su estructura. En este caso, está representando el número de línea del archivo que leímos. Pero un índice puede aportarnos más información relevante para nuestro problema, por lo que la propuesta es que el índice debería ser el instante en le que se tomó cada muestra ('Time').

Para esto tenemos que decirle a la función read\_csv dos cosas:

* por un lado que use la columna 'Time' como índice (index\_col = ['Time']) y
* por el otro que la interprete como un timestamp (parse\_dates = True).

df = pd.read\_csv('../Data/OBS\_SHN\_SF-BA.csv', index\_col=['Time'], parse\_dates=True)

Observá la diferencia:

>>> df.head()

H\_SF H\_BA

Time

2011-01-01 00:00:00 NaN 92.0

2011-01-01 01:00:00 NaN 110.0

2011-01-01 02:00:00 NaN 124.0

2011-01-01 03:00:00 NaN 132.0

2011-01-01 04:00:00 NaN 136.0

>>> df.index

DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 01:00:00',

...

'2014-12-31 22:00:00', '2014-12-31 23:00:00'],

dtype='datetime64[ns]', name='Time', length=35064, freq=None)

Que el índice sea temporal nos da una versatilidad genial para trabajar con estos datos. Probá por ejemplo los siguientes comandos:

>>> df['1-18-2014 9:00':'1-18-2014 18:00']

H\_SF H\_BA

Time

2014-01-18 09:00:00 85.0 67.0

2014-01-18 10:00:00 79.0 60.0

2014-01-18 11:00:00 73.0 49.0

2014-01-18 12:00:00 65.0 43.0

2014-01-18 13:00:00 59.0 36.0

2014-01-18 14:00:00 53.0 29.0

2014-01-18 15:00:00 48.0 22.0

2014-01-18 16:00:00 42.0 18.0

2014-01-18 17:00:00 36.0 33.0

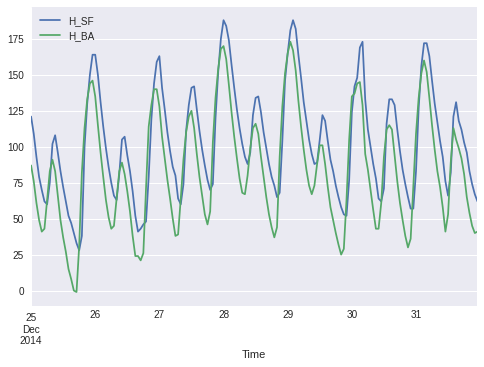
2014-01-18 18:00:00 40.0 67.0

Probá también df['2-19-2014'] (observá que el formato de fechas que se usa es el de EEUU), y df['12-25-2014':].

**Ondas de marea en el Río de la Plata**

Grafiquemos estos últimos datos:

df['12-25-2014':].plot()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175144.png)

Acá se ven tres fenómenos interesantes:

* Hay 14 picos en 7 días, esto corresponde a la frecuencia *semidiurna* de las mareas. Cada poco más de 12hs tenemos un ciclo con pleamar y bajamar. Dos ciclos por día.
* Por otra parte, se ve que las mareas en San Fernando están retrasadas respecto a las de Buenos Aires. Esto se debe a que las ondas de marea vienen del océano atlántico y se propagan por el estuario del río de la Plata, pasando primero por Buenos Aires y llegando luego, con retraso, a San Fernando. En ciertas condiciones esta onda de mareas puede llegar a la ciudad de Rosario, aunque se va atenuando en su viaje desde el atlántico.
* Finalmente, hay una marcada diferencia entre la altura registrada en San Fernando y la de Buenos Aires. Esto se debe a que las dos escalas, a partir de las que se registran los datos, tienen ceros que no están nivelados.

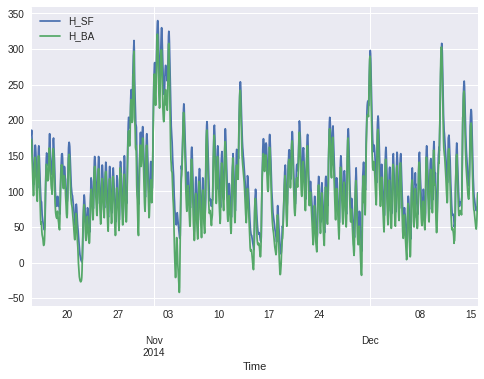
En este práctico nos proponemos estudiar la propagación de esta *onda de marea* que es generada por la atracción gravitacional que ejercen la luna y el sol sobre el agua. Vamos a usar una transformada de Fourier que nos permite estudiar las frecuencias predominantes en la serie de alturas. Las mareas se verán claramente porque estos efectos astronómicos son regulares y tienen frecuencias invariantes.

**Vientos y ondas de tormenta en el Río de la Plata**

Si miramos un gráfico un poco más extendido en el tiempo vamos a ver que las alturas no solo fluctúan con las mareas semidiurnas sino que la componente meteorológica (vientos principalmente, que generan *ondas de tormenta*) modifica las alturas de manera muy considerable.

El siguiente comando genera un gráfico entre el 15 de octubre de 2014 y el 15 de diciembre del mismo año.

df['10-15-2014':'12-15-2014'].plot()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175349.png)

En ese gráfico se puede observar cómo una sudestada a principios de noviembre elevó el nivel del estuario más de un metro durante casi tres días. Las dos fotos que siguen son justamente de esa sudestada. Fueron tomadas el primero de noviembre por Gustavo Castaing.

[](https://camo.githubusercontent.com/5cb0be4eaedab6e33ee68154db174143b3ccd674eb917f2a4162d91364bd54fb/68747470733a2f2f696d616765732e636c6172696e2e636f6d2f323031342f31312f30312f72314d51666f693337675f31323536783632302e6a7067)

Así como el viento del sudeste empuja el agua del mar hacia adentro del estuario y genera crecidas, los vientos del norte o el oeste también impulsan desplazamientos del agua del estuario, en este caso generando bajantes. En las siguientes dos fotos puede verse una bajante capturada por Juan Pablo Martínez Bigozzi el 19 de junio del 2019.

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/bajante19062019.jpg) [](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/bajante219062019.jpg)

La transformada de Fourier no resultará muy útil para ver estas *ondas de tormenta*. Como carecen de regularidad, no aparecerán claramente en el espectro de frecuencias.

**Ejercicio 8.10:**

Trabajemos con una copia de este fragmento:

dh = df['12-25-2014':].copy()

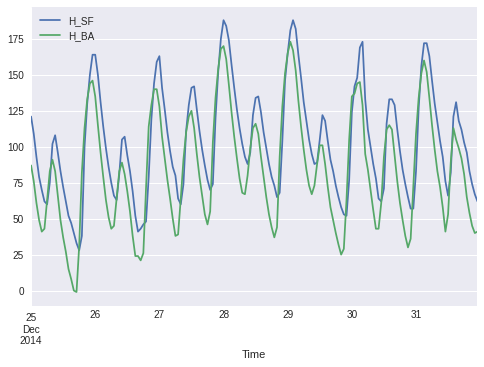
Podemos desplazar (shift en inglés) una Serie de Pandas usando el método ds.shift(pasos). Además, podemos subir o bajar su gráfico sumando una constante a todas las mediciones ds + cte.

Finalmente podemos unir dos series en un DataFrame de manera muy simple, para poder graficarlas juntas. Si concatenamos estas operaciones obtenemos algo así:

delta\_t = 0 # tiempo que tarda la marea entre ambos puertos

delta\_h = 0 # diferencia de los ceros de escala entre ambos puertos

pd.DataFrame([dh['H\_SF'].shift(delta\_t) - delta\_h, dh['H\_BA']]).T.plot()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175447.png)

Buscá los valores de delta\_t (es un número entero, son pasos) y delta\_h (puede tener decimales, es un float) que hacen que los dos gráficos se vean lo más similares posible.

Guardá tu código en el archivo mareas\_a\_mano.py para entregar.

**Correlación con desplazamientos**

El siguiente código calcula y grafica el [coeficiente de correlación *r* de Pearson](https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Pearson) entre la serie de alturas en Buenos Aires y la de San Fernando desplazada temporalmente.

import numpy as np

import pandas as pd

from scipy.stats import pearsonr

import matplotlib.pyplot as plt

# Levanto las dos series

df=pd.read\_csv('../Data/OBS\_SHN\_SF-BA.csv',index\_col=['Time'],parse\_dates=True)

# Me quedo con un fregmento

dh=df['10-01-2014':].copy()

# Selecciono los intervalos que voy a usar para desplazar SF

shifts = np.arange(-12,13)

# Genero un vector donde guardar los coeficientes de correlacion para cada deslpazamiento

corrs = np.zeros(shifts.shape)

for i, sh in enumerate(shifts):

#guardo el coeficiente de correlación r entre de SF desplazada con BA original.

corrs[i] = pearsonr(dh['H\_SF'].shift(sh)[12:-12],dh['H\_BA'][12:-12])[0]

# ploteo los resultados

plt.plot(shifts, corrs)

**Ejercicio 8.11: Interpolación**

Este ejemplo muestra una manera de interplolar la serie de manera de poder usar desplazamientos menores a una hora.

# Cada cuarto de hora

df=pd.read\_csv('../Data/OBS\_SHN\_SF-BA.csv',index\_col=['Time'],parse\_dates=True)

dh =df['10-01-2014':].copy() #ultimo trimestre

freq\_horaria = 4 # 4 para 15min, 60 para 1min

cant\_horas = 24

N = cant\_horas \* freq\_horaria

#resampleo cada tantos minutos

dh = dh.resample(f'{int(60/freq\_horaria)}T').mean()

#rellenos los NaNs suavemente

dh =dh.interpolate(method='quadratic')

# genero vector de desplazamientos (enteros)

ishifts = np.arange(-N,N+1)

# y su desplamiento horario asociado

shifts=ishifts/freq\_horaria

# finalmente calculo las correlaciones correspondientes

corrs = np.zeros(shifts.shape)

for i, sh in enumerate(ishifts):

corrs[i] = pearsonr(dh['H\_SF'].shift(sh)[N:-N],dh['H\_BA'][N:-N])[0]

# y grafico

plt.plot(shifts, corrs)

El comando np.argmax(corrs) se puede usar para devolver la coordenada de la máxima correlación. Esa posición del vector shifts indicará a cuántas horas corresponde ese desplazamiento. Si lo corremos con esta interpolación cada 15 minutos propuesta en el código, obtendremos que la onda tarda una hora en llegar a San Fernando. Usando una interpolación más fina, ¿podés estimar el desfazaje en minutos? A nosotros nos dio 57 minutos.

**Parte optativa**

En lo que sigue vamos a usar herramientas matemáticas para hacer un análisis similar al que hicimos recién de manera *artesanal*. Para una onda sinusoidal, el desplazamiento horizontal corresponde a una diferencia de fase y el desplazamiento vertical es simplemente una constante aditiva. Vamos a descomponer la serie de alturas observadas del agua por medio de la transformada de Fourier.

Las ondas de marea se mueven lentamente y tardan cierto tiempo en llegar de un puerto a otro. En lo que sigue vamos a ver cómo calcular el tiempo que le toma a esta onda en desplazarse de Buenos Aires hasta San Fernando.

**Lo que sigue es optativo**.

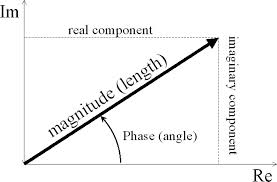
**Análisis por medio de transformadas de Fourier**

La transformada de Fourier descompone una señal en una suma de senos y cosenos (sinusoides) con diferentes frecuencias y amplitudes.

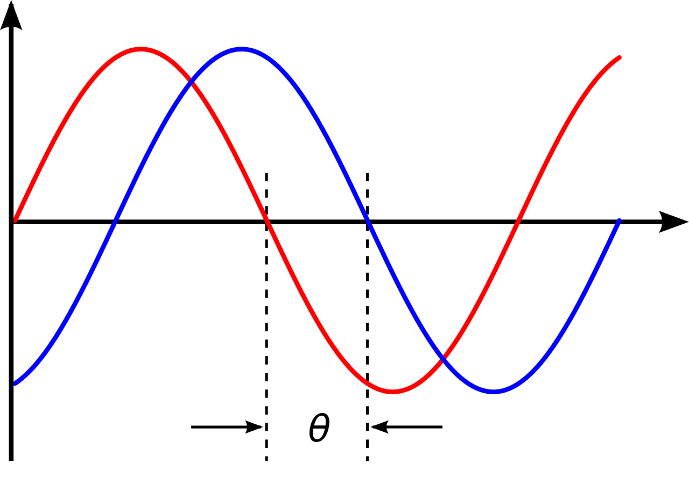
Esta animación ilustra gráficamente el proceso de la transformada de Fourier.

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/cuadrada.gif)

La transformada da, para cada frecuencia, un número complejo a + bi que puede pensarse como un vector (a,b) en el plano. La parte real va a multiplicar un coseno con la frecuencia indicada y la parte imaginaria un seno con la misma frecuencia. La magnitud (o amplitud, o potencia) de la señal en esa frecuencia se corresponde con el largo del vector (a, b).

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/vectorial.jpg)

La fase (o desplazamiento del máximo respecto del origen de las coordenadas), se corresponde con ángulo que forma este vector (a, b) con el semieje de los reales positivos.

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/phase_shift.png)

Aquí, la variable theta (θ) representa el desplazamiento de fase de la curva azul (respecto a la roja que tiene desplazamiento nulo). Esta *fase* suele medirse en radianes, correspondiendo 2\*π a un ciclo completo de desfasaje.

Vamos a aplicar estas herramientas al análisis de la propagación de la onda de marea por el estuario del plata.

**Preparación de módulos y datos**

Vamos a usar los siguientes módulos:

from scipy import signal # para procesar señales

import numpy as np

import matplotlib.pyplot as plt

Seleccionemos las dos series como vectores de numpy (con el método to\_numpy()).

inicio = '2014-01'

fin = '2014-06'

alturas\_sf = df[inicio:fin]['H\_SF'].to\_numpy()

alturas\_ba = df[inicio:fin]['H\_BA'].to\_numpy()

Primero definamos una función que calcule la transformada de Fourier para datos horarios y considerando como unidad de frecuencia los días (descartamos la mitad de los coeficientes de la transformada porque los datos son reales y no complejos). Podés tomarla como una caja negra por ahora...

def calcular\_fft(y, freq\_sampleo = 24.0):

'''y debe ser un vector con números reales

representando datos de una serie temporal.

freq\_sampleo está seteado para considerar 24 datos por unidad.

Devuelve dos vectores, uno de frecuencias

y otro con la transformada propiamente.

La transformada contiene los valores complejos

que se corresponden con respectivas frecuencias.'''

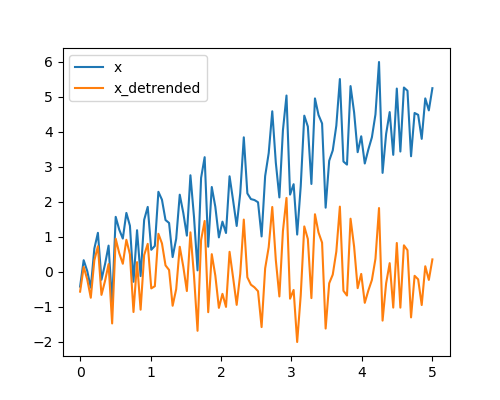
N = len(y)

freq = np.fft.fftfreq(N, d = 1/freq\_sampleo)[:N//2]

tran = (np.fft.fft(y)/N)[:N//2]

return freq, tran

Para poder analizar una onda por medio de su transformada de Fourier, es necesario que la onda sea periódica. Puede pasar que no sea el caso y que una onda tenga tendencia lineal, en ese caso podríamos usar la función scipy.signal.detrend().

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/detrend.png)

En nuestro caso supondremos que la marea media se mantuvo estable a lo largo del período de estudio, así que no tenemos que hacerle este procesamiento intermedio.

**Espectro de potencia y de ángulos para San Fernando**

Primero calculamos la transformada de las alturas de San Fernando.

freq\_sf, fft\_sf = calcular\_fft(alturas\_sf)

Si quisiéramos graficar freq\_sf contra fft\_sf no podríamos ver mucho ya que fft\_sf contiene números complejos.

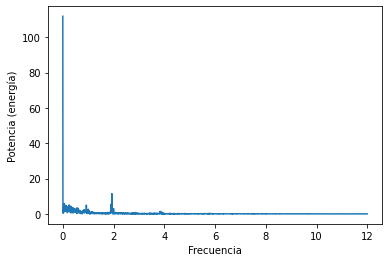
La potencia (o amplitud) para cada frecuencia se calcula como el módulo del número complejo correspondiente (para la frecuencia freq\_sf[i] y la potencia es abs(fft\_sf[i])). Al graficar esto podemos ver la amplitud de los sinusoides para cada frecuencia. Este gráfico se llama el *espectro de potencias* de la onda original.

plt.plot(freq\_sf, np.abs(fft\_sf))

plt.xlabel("Frecuencia")

plt.ylabel("Potencia (energía)")

plt.show()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175811_.png)

A simple vista se observan dos picos, uno en frecuencia 0 (constante relacionada con el cero de escala) y otro pico cercano a la frecuencia 2 (frecuencia semidiurna) que está relacionado con la onda de mareas.

El pico en la primera posición efectivamente se corresponde con la frecuencia 0 y su amplitud es:

>>> freq\_sf[0]

0.0

>>> np.abs(fft\_sf[0])

111.83

A partir de esto podemos decir que las alturas del río en San Fernando durante este período oscilan alrededor de los 111.8 cm de altura.

Para analizar precisamente el pico semidiurno podemos usar find\_peaks que proveescipy.signal para evitar hacerlo a ojo. Vamos a pedir aquellos picos que tengan al menos cierta diferencia con su entorno (prominencia), un buen valor para esto es el 8. Podés probar otros valores y observar el resultado.

>>> print(signal.find\_peaks(np.abs(fft\_sf), prominence = 8))

(array([350]), {'prominences': array([11.4554514]), 'left\_bases': array([307]), 'right\_bases': array([2109])})

Esta respuesta nos indica que hay un pico con la prominencia solicitada (al menos 8), que tiene un magnitud de 11.45 y que corresponde a la posición 350 del vector.

>>> freq\_sf[350]

1.93

La frecuencia relacionada con esa posición es cercana a dos, como ya habíamos observado en el gráfico (dos ciclos por día). Podemos distinguir los picos agregando un punto rojo y mirando más de cerca el área de interés:

plt.plot(freq\_sf, np.abs(fft\_sf))

plt.xlabel("Frecuencia")

plt.ylabel("Potencia (energía)")

plt.xlim(0,4)

plt.ylim(0,20)

# me quedo solo con el último pico

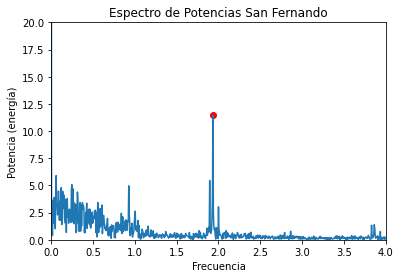
pico\_sf = signal.find\_peaks(np.abs(fft\_sf), prominence = 8)[0][-1]

# es el pico a analizar, el de la onda de mareas

# marco ese pico con un circulito rojo

plt.scatter(freq\_sf[pico\_sf], np.abs(fft\_sf)[pico\_sf], facecolor = 'r')

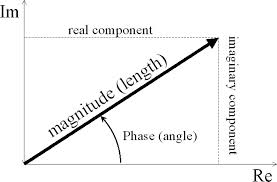
plt.show()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175902_.png)

Estos gráficos permiten interpretar que si descomponemos la curva de alturas en San Fernando como suma de sinusoidales, el sinusoide con frecuencia 1.93 tiene una magnitud considerablemente llamativa. No es casualidad que este sea un punto distinguido: se trata de la frecuencia de las mareas lunares.

Ahora viene la parte un poco más sutíl: **el análisis de las fases**. Si conocemos la fase de estas componentes en dos puertos distintos, podremos estimar el tiempo que tarda en desplazarse la marea de uno a otro.

Para calcular la fase (entre -π y π) de dicha componente (la que ubicamos en la posición 350) en el puerto de San Fernando, podemos simplemente usar np.angle() y pasarle el número complejo en cuestión:

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/vectorial.jpg)

>>> ang\_sf = np.angle(fft\_sf)[pico\_sf]

>>> print(ang\_sf)

1.4849

Obtenemos un valor cercano a pi/2. Recordemos que 2*pi corresponde a un desfasaje de un ciclo completo de la curva. Como nuestra curva de estudio tiene una frecuencia diaria ligeramente inferior a 2 (freq\_sf[350]~1.93), 2*pi corresponde a 24/1.93 horas ~ 12.44 horas. Por lo tanto la fase obtenida con angSF[350] corresponde a un retardo de

>>> ang\_sf \* 24 / (2 \* np.pi \* freq\_sf[350])

2.93

Es decir, este sinusoide está desfasado poco menos de 3hs respecto al seno *neutro*.

**Espectro de potencia y de ángulos para Buenos Aires**

Repitamos velozmente el procedimiento para el puerto de Buenos Aires y analicemos las diferencias.

freq\_ba, fft\_ba = calcular\_fft(alturas\_ba)

plt.plot(freq\_ba, np.abs(fft\_ba))

plt.xlabel("Frecuencia")

plt.ylabel("Potencia (energía)")

plt.xlim(0,4)

plt.ylim(0,20)

# me quedo solo con el último pico

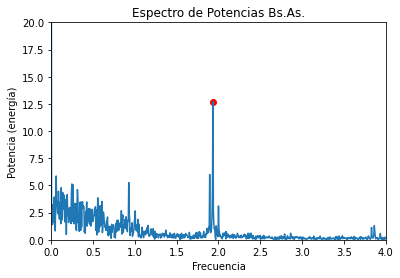
pico\_ba = signal.find\_peaks(np.abs(fft\_ba), prominence = 8)[0][-1]

#se grafican los picos como circulitos rojos

plt.scatter(freq\_ba[pico\_ba], np.abs(fft\_ba)[pico\_ba], facecolor='r')

plt.title("Espectro de Potencias Bs.As.")

plt.show()

[](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/Figure175932_.png)

Si buscamos la constante alrededor de la que oscilan las mareas según el nivel del puerto de Buenos Aires obtenemos:

>>> np.abs(fft\_ba[0])

88.21

Con este resultado es sencillo obtener una estimación para la diferencia de alturas de los ceros de escala entre ambos puertos.

*Pregunta 1:* ¿Cuál es la diferencia de altura media entre los puertos obtenida de esta forma?

*Pregunta 2:* ¿De qué otra forma se puede estimar el valor medio de un puerto? ¿Cuánto da la diferencia con este otro método?

Por otra parte, si observamos que el espectro de potencia vemos que los picos en ambos puertos son súmamente similares.

>>> print(signal.find\_peaks(np.abs(fft\_ba), prominence=8))

(array([350]), {'prominences': array([12.67228046]), 'left\_bases': array([279]), 'right\_bases': array([1000])})

Las mareas de Buenos Aires tiene una componente de máxima amplitud en la frecuencia 1.93 (misma que San Fernando) y con una magnitud de 12.67 (bastante similar a la magnitud correspondiente en San Fernando). Resta estudiar la fase de la curva de los datos de df\_ba en esta frecuencia para poder determinar con precisión la diferencia de fase entre ambos puertos para ondas de marea. Primero calculamos el ángulo de la componente correspondiente y luego lo convertimos en horas usando el factor ang2h:

>>> ang\_ba = np.angle(fft\_ba)[pico\_ba]

>>> print(ang\_ba)

1.96

>>> freq = freq\_ba[pico\_ba]

>>> ang2h = 24 / (2\*np.pi\*freq)

>>> ang\_ba \* ang2h

3.8786004708135566

Por lo tanto, el retardo de la onda de mareas puede calcularse usando

(ang\_ba - ang\_sf) \* ang2h

**Ejercicio 8.12: Desfasajes**

En la Pregunta 1 estimaste el desfasaje vertical entre los ceros de escala de los puertos analizados. Ahora tenés que estimar el desfasaje temporal de las ondas de marea entre ambos puertos. ¿A cuántos minutos corresponde aproximadamente el tiempo que tarda la onda de mareas en llegar del puerto de Buenos Aires al de San Fernando?

Usá estos datos para volver a hacer el gráfico del [Ejercicio 8.10](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/06_Series_Temporales.md#ejercicio-810) (vas a tener que redondear a horas enteras el delay temporal).

**Un poco más avanzados:**

**Ejercicio 8.13: Otros puertos**

Usando el [archivo con datos del Puerto de Zárate](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/OBS_Zarate_2013A.csv), estimá el tiempo (expresado en horas y minutos) que le toma a la onda de marea llegar de Buenos Aires a Zárate.

Obviamente la onda llega atenuada a Zárate. ¿Cómo se refleja esta atenuación en la transformada? ¿Podés cuantificar esta atenuación?

Guardá lo que hayas hecho hasta acá en el archivo mareas\_fft.py para entregar.

**Ejercicio 8.14: Otros períodos**

El primer análisis se realizó con el primer semestre del 2014 ya que no tiene ni datos faltantes ni outliers. Este ejercicio es una invitación a explorar estos problemas tan frecuentes.

* ¿Se puede comparar Zárate con San Fernando usando todos los datso de Zárate? ¿Cómo se comporta San Fernando en esas fechas?
* ¿Se pueden usar las series completas de BA y SF para calcular el desfasaje de la onda de mareas? ¿Qué son las alturas negativas? ¿Tienen sentido?

La siguiente función completa datos faltantes y corrige pequeños problemas en los índices. Es un poco brutal tratar así un DataFrame: es conveniente mirar los datos antes de completar faltantes. Lo dejamos como puntero a diferentes métodos muy útiles para la limpieza de series.

def reparar(df):

df = df.interpolate()

df = df.resample('H').mean()

df = df.fillna(method = 'ffill')

return df

**8.7 Cierre de la clase**

Para cerrar esta clase te pedimos dos cosas:

* Que recopiles las soluciones de los siguientes ejercicios:
  1. El archivo vida.py del [Ejercicio 8.1](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/02_Fechas.md#ejercicio-81-segundos-vividos).
  2. El archivo listar\_imgs.py del [Ejercicio 8.5](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/04_Ordenando_archivos.md#ejercicio-85-recorrer-el-%C3%A1rbol-de-archivos).
  3. El archivo arbolado\_parques\_veredas.py del [Ejercicio 8.9](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/05_Pandas.md#ejercicio-89-comparando-especies-en-parques-y-en-veredas).
  4. El archivo mareas\_a\_mano.py del [Ejercicio 8.10](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/06_Series_Temporales.md#ejercicio-810).

Te proponemos además dos optativos:

* 1. El archivo ordenar\_imgs.py del [Ejercicio 8.6](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/04_Ordenando_archivos.md#ejercicio-86-ordenar-el-%C3%A1rbol-de-archivos-optativo).
  2. El archivo mareas\_fft.py del [Ejercicio 8.13](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/06_Series_Temporales.md#ejercicio-813-otros-puertos).
* Que completes [este formulario](https://docs.google.com/forms/d/e/1FAIpQLScrVpEMe6Vmhwz5tVOBOxQ3ML5EkF851RFL7Zeyejs0cK_9gg/viewform) usando tu dirección de mail como identificación. Al terminar vas a obtener un link para enviarnos tus ejercicios.

La corrección de pares de esta semana será con el archivo arbolado\_parques\_veredas.py del [Ejercicio 8.9](https://github.com/python-unsam/Programacion_en_Python_UNSAM/blob/master/Notas/08_Fechas_Carpetas_y_Pandas/05_Pandas.md#ejercicio-89-comparando-especies-en-parques-y-en-veredas).

¡Gracias!