Gestión de Tesorería con Python

Francisco Salas-Molina
David Pla-Santamaría

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Para seguir leyendo haga click aquí

Gestión de Tesorería con Python

Abril de 2017

Francisco Salas-Molina David Pla-Santamaria

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Colección Académica

Los contenidos de esta publicación han sido revisados por el Departamento de **Dpto. de Economía y Ciencias Sociales** de la Universitat Politécnica de València

Para referenciar esta publicación utilice la siguiente cita:
Salas Molina, Francisco; Pla Santamaria, David. (2017*). Gestión de Tesorería con Python*. Valencia: Universitat Politècnica de València

distribución: Telf.: 963 877 012 / www.lalibreria.upv.es / Ref.: 0606_03_01_01

Imprime: Byprint Percom, sl

ISBN: 978-84-9048-622-1
Impreso bajo demanda

Impreso en España

Índice general

Índice de figuras

Índice de tablas

Resumen

Este manual pretende servir de iniciación a la gestión de tesorería mediante el lenguaje de programación Python. Aunque su contenido está basado en los últimos avances en planificación financiera a corto plazo, el objetivo es básicamente instructivo y, por tanto, está pensado especialmente para ti, estudiante o profesional de las finanzas cuya inquietud por mejorar tus habilidades de gestión te lleva a explorar nuevos caminos de conocimiento.

En primer lugar, abordarás cuestiones introductorias a la gestión de tesorería desde un punto de vista cuantitativo. Para ello utilizarás como herramienta el lenguaje de programación de código abierto Python. Sin embargo, no es necesario que tengas ningún conocimiento previo sobre Python ya que este manual también persigue ayudarte a abrir una nueva ventana que probablemente estaba cerrada. Conocerás los elementos básicos del lenguaje para tratar y visualizar datos de cualquier dominio de aplicación. Aprenderás a realizar cálculos necesarios para aplicar los métodos cuantitativos más habituales.

En finanzas el tiempo lo es casi todo. Por ello, en el Capítulo 1 encontrarás una introducción a Python para finanzas que te permitirá manejar series temporales financieras. Verás lo sencillo que es importar datos, aplicar cualquier tratamiento que sea de tu interés sobre ellas y, finalmente, serás capaz de representarlas gráficamente. También podrás analizar cuáles son las principales propiedades estadísticas de cualquier variable financiera.

En el Capítulo 2, te ocuparás de la gestión de tesorería. Una vez que has conseguido llenar tu mochila de herramientas útiles, es hora de empezar a utilizarlas.

La gestión de tesorería es el área financiera en la que la automatización de todavía tiene un largo camino por recorrer. ¿Te apetece empezar a caminar? Si es así, visitarás algunos de los modelos de gestión de tesorería más utilizados. Y si te sientes con fuerzas, en el Capítulo 3 podrás diseñar tu propio modelo de tesorería con una, dos o cien cuentas bancarias para poder simular diferentes estrategias de planificación.

En el Capítulo 4, aprenderás a plantear el problema de gestión de tesorería desde un punto de vista de optimización. Para ello, podrás utilizar un módulo específico en Python para la obtención de los mejores planes de tesorería teniendo en cuenta tanto el coste como el riesgo de los planes.

Finalmente, en el Capítulo 5, te darás cuenta de lo interesante que es prever el futuro financiero. Planificar es prepararse para el futuro, y prever es prepararse para planificar. En multitud de ocasiones, la gran cantidad de datos que cada día generamos esconden patrones interesantes sobre los que nos podemos apoyar para tomar mejores decisiones. Te ayudaremos a descubrirlos.

Francisco Salas-Molina
francisco.salas.molina@gmail.com

David Pla-Santamaria
dplasan@upv.es

Capítulo 1

Introducción a Python para finanzas

En este capítulo conocerás qué es Python en la sección 1.1 y cómo te puede ayudar a analizar y visualizar datos financieros. En la sección sección 1.2 aprenderás a importar, analizar y realizar operaciones básicas de tratamiento de datos financieros. Los datos hablan si se les pregunta adecuadamente. Así que cuanto más conozcas sobre tus datos, mayor provecho podrás obtener de ellos. Finalmente, en la sección 1.3 encontrarás las instrucciones necesarias para visualizar los datos de manera que te resulte más fácil comprenderlos y presentarlos a los demás.

1.1 Todo lo que debes saber sobre Python

Antes de empezar a construir necesitamos un entorno de trabajo. Nuestro laboratorio serán los Notebooks de Python que te permiten hacer un gran número de cosas como ejecutar código, añadir texto, presentar gráficas o incrustar imágenes, en un entorno tan conocido como es tu navegador habitual. Lo más recomendable es que primero descargues e instales la última versión de Python, a través de alguna plataforma de distribución como:

`https://www.continuum.io/`

Tranquilo no te va a costar nada y si tienes algún problema en la instalación siempre puedes acudir a los numerosos tutoriales que hay en la red. Una vez tengas instalado Python, ya puedes ejecutar *jupyter notebook* desde la interfaz de consola PowerShell en Windows. Si no utilizas Windows, ya sabrás cómo hacerlo. Por el contrario, si primero quieres probar de qué va esto de los Notebooks sin instalar nada en tu equipo, puedes trabajar con ellos online en:

`https://jupyter.org/`

¿Todo en orden? ¿Has conseguido abrir tu primer Notebook de Python en tu navegador? ¿Eres capaz de ver el típico menú *File-Edit-View* y un In [ ] : con una celda en blanco a su derecha como el de la Figura 1.1? Vamos a pensar que sí. Es hora de dar nuestro primer paso. Escribe lo siguiente en la celda: $2 + 2$ y pulsa simultáneamente las teclas *Shift+Enter*. ¿El resultado a la derecha de Out [1] : es 4? Enhorabuena, ya eres un programador@ de Python.

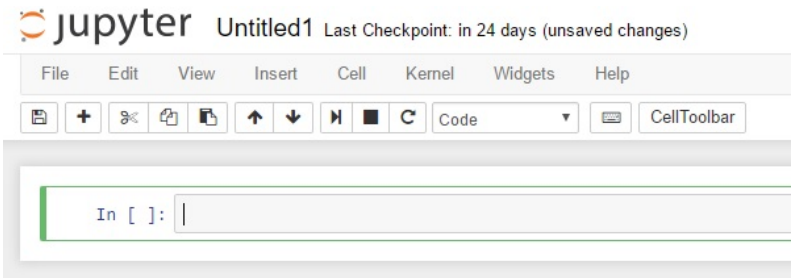

Figura 1.1: Una celda de un Notebook esperándote.

En los Notebooks de Python, hay cuatro tipos de celdas aunque probablemente trabajarás sólo con dos de ellas:

- *Code*: Las celdas de tipo *Code* son las celdas que aparecen por defecto y se utilizan para insertar código ejecutable. Cuando se ejecutan (pulsando

Shift+Enter), Python evalúa la expresión introducida y devuelve un resultado o un mensaje de error si hay algo que no ha funcionado bien. Si queremos añadir un comentario en una celda de código, podemos utilizar el carácter especial '#'.

- *Markdown*: Este tipo de celdas se utilizan para insertar texto con formato que puedes utilizar para explicar el código, para insertar fórmulas, enlaces a páginas web, imágenes o incluso vídeo.
- *Raw NBConvert*: En estas celdas puedes escribir cualquier expresión que no se evalúa nunca. ¿Te cuento un secreto? No las he utilizado nunca.
- *Heading*: Se utilizan para escribir títulos cosa que también se puede hacer mediante una celda de tipo *Markdown*

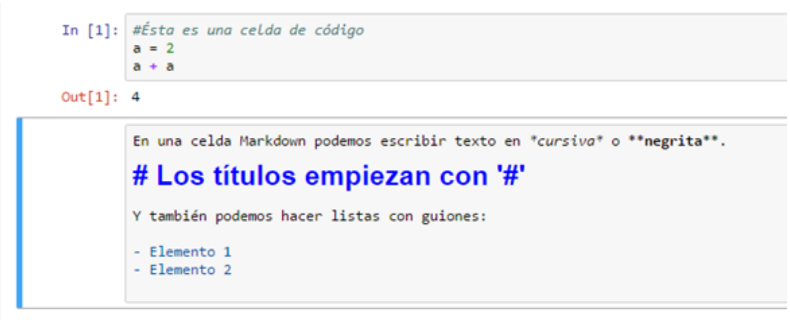

Figura 1.2: Tipos de celdas en un Notebook.

Pulsando simultáneamente *Shift+Enter*, la celda actual se ejecuta y pasa a la siguiente. Si no hay siguiente, crea una celda nueva. Pero hay otras combinaciones de teclas que te resultarán de utilidad cuando empieces a trabajas con Notebooks:

- *Alt+Enter*: inserta una nueva celda después de la actual (el signo + denota pulsar las dos teclas simultáneamente).
- *Esc* $\rightarrow$ *m*: convierte la celda actual en una celda de tipo *Markdown* (el signo $\rightarrow$ denota pulsar primero una y luego la otra).
- *Esc* $\rightarrow$ *a*: inserta una celda arriba de la actual.
- *Esc* $\rightarrow$ *b*: inserta una celda abajo de la actual.
- *Esc* $\rightarrow$ *d* $\rightarrow$ *d*: elimina la celda actual.

- *Tab*: permite mostrar los atributos de un objeto para no tener que aprenderlos de memoria.
- *Shift+Tab*: muestra en pantalla la ayuda sobre cualquier función.

Si las palabras objeto y función no te resultan familiares, no te preocupes. Veremos de qué se trata más adelante. Otro de los aspectos importantes que debes conocer sobre Python es que muchas de las funcionalidades están almacenadas en librerías que hay que importar para poder utilizarse en un Notebook. Puede parecer una tarea pesada pero tiene su lógica si tenemos en cuenta que hay centenares de librerías que probablemente no utilizaremos nunca.

> *Una **librería** es una colección de funciones implementadas en un lenguaje de programación que permite su reutilización a través de una interfaz determinada.*

Algunas de las más importantes son: *Numpy* que contiene las herramientas básicas de cálculo y análisis de datos almacenados en vectores y matrices; *Matplotlib* que permite la visualización de datos mediante gráficos; y *Pandas* que dispone de las estructuras de datos necesarias para la manipulación y visualización avanzada de datos como las series temporales. Entenderás mejor cómo se hace mediante el siguiente ejemplo. Si ejecutamos la línea 1 para calcular e^1 sin importar la librería *Numpy*, verás un mensaje de error. Para hacerlo tienes tres opciones: importar sólo la función *exp*; importar todas las funciones de la librería sin asignarle una etiqueta; importar toda la librería y asignarle un nombre. La primera de las opciones te permite importar sólo aquello que vas a necesitar y las otras dos son similares aunque es recomendable añadir un nombre para saber de qué librería procede la función que estás utilizando.

Ejemplo 1.1.1 *Importación de librerías.*

```
1 print(exp(1))              #Devuelve un error
2 from numpy import exp      #Importa sólo función
3 print(exp(1))
4 from numpy import *        #Importa todo sin nombre
5 print(exp(1))
6 import numpy as np         #Importa todo con nombre
7 np.exp(1)                  #Necesario indicar nombre
```

En el ejemplo anterior, has utilizado la función *print* para presentar el resultado de la función $exp(1)$ por pantalla. Cuando se ejecuta una celda con varias líneas, el Notebook sólo presenta el resultado de una instrucción de tipo mostrar variable cómo la de la línea 7 si esta es la última línea de la celda. Para presentarlas todas es necesario utilizar la función *print*. De nuevo aparece la palabra función. Es hora de ver qué son y qué pueden hacer por ti.

> *Una* ***función*** *es un conjunto reutilizable de código que recibe uno o varios argumentos de entrada, realiza una serie de acciones, y devuelve uno o varios resultados.*

En Python, hay funciones previamente definidas como *exp*, que aceptan un argumento como 1 y que devuelven un resultado, en este caso, 2.718. Pero también puedes definir tus propias funciones. Por ejemplo, imagina que tienes una serie de flujos de caja almacenados en una lista llamada f y quieres calcular la media de estos valores. ¿Podrías diseñar una función que hiciera esto por ti? El Ejemplo 1.1.2 muestra cómo podría hacerse. Como verás la definición de una función en Python comienza por la palabra *def* seguida del nombre de la función que incluye entre paréntesis los argumentos de la función acabando siempre con dos puntos (:). En nuestro ejemplo el nombre que hemos elegido es *media* y el argumento es *lista* cuyo nombre también elegimos nosotros. La definición explícita del código de la función, es decir, las acciones que se realizan, empiezan después de los dos puntos y tras una tabulación. Python utiliza los dos puntos y las tabulaciones para delimitar las distintas partes del código.

Ejemplo 1.1.2 *Definición de la función media.*

```
def media(lista):
    suma = 0
    cuenta = 0
    for elemento in lista:
        suma = suma + elemento
        cuenta = cuenta + 1
    return(suma/cuenta)
```

La definición de nuestra función *media* incluye la inicialización de dos variables auxiliares *suma* y *cuenta* que se utilizan para is sumando y contando los elementos de la lista. ¿Y cómo se recorre la lista? Mediante un bucle *for*. No te asustes, es sencillo. Un bucle *for* es una instrucción especial presente en todos los lenguajes de programación que permite realizar la misma serie de instrucciones un número determinado de veces que viene dado normalmente por un contador. En nuestro ejemplo, por la posición de los elementos de la lista. De la misma manera que con la función, las instrucciones a realizar de manera iterativa empiezan tras los dos puntos y una tabulación. Por ejemplo, para calcular la media de los valores almacenados en una lista, sumamos el valor del elemento en cada iteración a la suma parcial de todos los elementos anteriores. Además, utilizamos la variable auxiliar *cuenta* para contar los elementos que hemos sumado. En ambos casos, lo hacemos mediante una instrucción de asignación que incluye el valor anterior de la variable. En otras palabras, leemos el valor de la variable *suma*, lo sumamos al valor del *elemento* de la lista, y el resultado lo asignamos de nuevo a la variable *suma*, actualizando su valor.

Finalmente, la instrucción *return()* incluye entre paréntesis el resultado que debe devolver la función cuando sea llamada. Esta instrucción marca también el fin de la definición de la función. Para comprobar si has definido correctamente la función, haz una llamada a la función tal como se indica en el Ejemplo 1.1.3. Para ello, primero crea una lista de valores f, escribiendo los valores entre corchetes y separados por comas, y luego escribe el nombre de la función incluyendo como argumento la lista f que acabamos de crear. Otra función muy útil de una lista es *append* que te permite añadir elementos al final de una lista. Prueba a añadir un nuevo elemento y vuelve a llamar a la función media. Seguramente el resultado será diferente.

Ejemplo 1.1.3 *Llamando a la función media.*

```
f = [1,2,3,-1,-2]
print(media(f))  # Resultado 0.6
f.append(5)      # Añade un nuevo elemento
print(media(f))  # Resultado 1.33
```

Las listas son una de las estructuras de datos más utilizadas en Python, pero hay más, por ejemplo, los vectores, las matrices, las tuplas, los rangos, los diccionarios, las series, los *data frames*. Más adelante, verás qué ventajas tiene organizar los datos de una determinada manera. De momento, con saber que existen es suficiente.

> *Una **estructura de datos** es una forma determinada de organizar los datos para poder ser utilizados de forma eficiente.*

Probablemente estarás pensando que alguien habrá necesitado antes definir una función tan común como la media y que lo más lógico sería reutilizar esa función más que crear una nueva. Estás en lo cierto. Es más, el concepto de reutilización de código está en la esencia de un lenguaje abierto como Python. Para ello, debemos importar las librerías que contienen las funciones que nos interesan tal como se explicó más arriba. Una búsqueda rápida en la red te dará información sobre qué librerías te pueden resultar de ayuda. La librería de Python que contiene la mayoría de las funciones para cálculo científico es Numpy, abreviación de *Numerical Python* que incluye un buen número de utilidades que te pueden resultar de gran ayuda en algún momento:

- vectores, matrices y otras estructuras de datos multidimensionales;
- funciones algebraicas, por ejemplo, para resolver sistemas de ecuaciones lineales;
- generación de números aleatorios;

Si quieres averiguar más sobre *Numpy*, un buen sitio donde empezar a buscar es McKinney (2012). Para cumplir con los objetivos marcados de este manual, al menos debes conocer cómo se crea un vector y una matriz, cómo se indexan los elementos de vectores y matrices, y cómo realizar las operaciones básicas entre ellos para obtener la información que te interesa. Empecemos por crear

un vector a partir de la lista f del ejemplo anterior. Para importar *Numpy*, utilizaremos la convención habitual *import numpy as np*. Una vez creado, puedes utilizar funciones básicas de Python como *len* para presentar la longitud del vector, o funciones de *Numpy* para calcular la media y la desviación típica de los valores del vector.

Ejemplo 1.1.4 *Crea un vector y presenta sus características básicas.*

```
import numpy as np
vector = np.array(f)    # Equivalente a vector = np.array([1,2,3,-1,-2])
print(len(vector))      # Presenta la longitud del vector
print(sum(vector))      # Presenta la suma de los elementos del vector
print(np.mean(vector)) # Calcula la media
print(np.std(vector))  # Calcula la desviación típica
```

¿Y qué ocurre si queremos acceder al primer elemento de nuestro *vector*? Lo primero que debes tener en cuenta es que las operaciones de indexación en Python empiezan con el cero, no con el uno. Lo segundo es que para acceder a un determinado elemento de un vector debes escribir el nombre del vector seguido del índice entre corchetes. Si lo que pretendes es extraer los elementos del vector en un determinado rango de índices debes utilizar el operador (:) tal como se indica en el siguiente ejemplo:

Ejemplo 1.1.5 *Indexación de los elementos de un vector.*

```
print(f[1])     # Presenta el segundo elemento: 2
print(f[0])     # Presenta el primer elemento: 1
print(f[0:2])   # Presenta desde el primer elemento al segundo: [1, 2]
print(f[-1])    # Presenta el último elemento: -2
print(f[1:])    # Presenta desde el segundo al último: [2, 3, -1, -2]
print(f[-2:])   # Presenta los dos últimos: [-1, -2]
f[-1] = 0       # Asigna un nuevo valor al último elemento
print(f[-1])    # Presenta el último elemento: 0
```

La forma más sencilla de crear una matriz con unas dimensiones determinadas es la indicada en el Ejemplo 1.1.6. Sin embargo, imagina que tienes que controlar un conjunto de 3 cuentas bancarias y que quieres simular el efecto que tendrá en los próximos 4 días un flujo de caja aleatorio entero entre un valor mínimo de -3 y máximo de 5. Si buscas un poco en la red verás que la función de *Numpy* que permite generar valores aleatorios enteros entre un valor mínimo y un valor máximo es *random.randint*. Utilizando esta función puedes crear un vector de 12 valores aleatorios que posteriormente puedes transformar en la matriz M mediante la función *reshape* que recibe como argumento una

Para seguir leyendo haga click aquí