# **Módulo III: Rudimentos de programación**

# Casi todos los lenguajes de programación comparten sus elementos básicos que hace que el aprendizaje de la programación sea cada vez más dinámico a medida que se va incorporando conocimiento. Sea el lenguaje que usted utilice, en todos estará programando secuencias de órdenes que harán interactuar elementos para lograr en última instancia el cumplimiento de un objetivo.

# Es muy común escuchar que un lenguaje se parece mucho a otro o que, sabiendo programar en uno, se pueda programar sin problemas en otro. Por esto mismo, antes de adentrarnos en las particularidades de los lenguajes que recorreremos en este libro, creemos importante repasar algunos términos con los que dejaremos firmes los conocimientos necesarios para entender los aspectos técnicos de los que hablaremos más tarde.

# Las similitudes van más allá de la simple homogeneidad visual que notará entre diferentes lenguajes. Es más bien una forma de pensar nueva que, cuando se acostumbre, podrá trasladar de un lenguaje a otro sin tanto costo de adaptación. En otras palabras, no es que sólo se verán parecido, sino que el parentesco tiene puntos en común más profundos. Es una forma lógica de pensamiento que verá que está muy distante de todas las complicaciones que pensó que la programación le traería. Todo lo contrario, la programación le abrirá las puertas, ya que si lo puede imaginar, lo podrá programar.

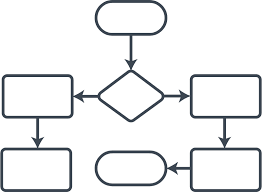
## 

## 3.1. ¿Qué es un algoritmo?

Esta es la definición más temida por quienes aún no se adentraron en el mundo de la programación. Veremos que este temor está totalmente infundado porque en el día a día de cualquier persona ejecutamos algoritmos de todo tipo. Y no es que nos referimos a aquellos profesionales altamente capacitados cuyo trabajo es incluso difícil de entender, si no que absolutamente toda persona que ejecuta un procedimiento de cualquier naturaleza está ejecutando implícitamente uno.

No hay una definición universal, pero en definitiva un algoritmo es, ni más ni menos, que un conjunto de instrucciones. Y si ahora está pensando que esta definición no tiene nada que ver con usted, sepa que las instrucciones de cómo usar el shampoo o cómo hacerse un café son simplemente uno de estos. Los manuales de uso de un aparato son también un excelentísimo ejemplo: “poner la perilla en la posición indicada y presionar el chispero hasta que el piloto quede encendido”. Además de indicarle cómo prender una estufa, también le está indicando una serie de órdenes, es decir, ¡un algoritmo!

Los diagramas de flujo son otro gran ejemplo de lo que es un algoritmo y seguramente los vio alguna vez en su vida. Permiten representar algunas de las características fundamentales que nuestros algoritmos de trading deben cumplir y muestran visualmente la estructura lógica que tendrán los desarrollos que hará al finalizar este libro. Vea la imagen que sigue y fíjese cómo se parte de un escenario inicial para luego dividir las órdenes siguientes según algún tipo de condición. Dependiendo de cuáles sean las condiciones de la situación, el algoritmo podría operar de distinta forma, o incluso no operar. Todo depende de cómo desarrolle su sistema y de la naturaleza del problema.



Cuando uno piensa de manera humanamente habitual, no se entera de la infinidad de elementos que se entrelazan en nuestras acciones y la rapidez con la que nuestro cerebro resuelve problemas. Ahora, cuando queremos programar la naturaleza de un problema y cómo solucionarlo, hay que ser meticulosamente detallista para decirle a la computadora exactamente cómo y cuándo hacerlo. Sin duda alguna el microprocesador de nuestra computadora es infinitas veces más rápido para resolver los problemas que programaremos, pero es tan exacto que debemos decirle precisamente lo que esperamos de él.

Las órdenes de nuestros algoritmos tienen que estar perfectamente definidas y no dejar lugar a ninguna ambigüedad para diseñar un sistema infalible. Imagine que programa su sistema de tal forma que “compra cuando una acción sube”, pero pierde totalmente de vista que la acción que está comprando está inmersa en una evidente tendencia bajista con más de un 50% de pérdidas en las ruedas anteriores. Si bien esta situación puede ser una gran oportunidad de compra, también puede ser tan peligroso como agarrar un cuchillo en plena caída. Se puede operar de la forma que uno quiera, pero hay que asegurarse que no hay lugar para operaciones incorrectas por definiciones demasiado laxas.

Otro punto importante es que las instrucciones tienen que estar ordenadas meticulosamente para que el orden secuencial tenga sentido. No es lo mismo “comprar luego analizar” que “analizar luego comprar”, aunque ambos casos contengan las mismas acciones. Si bien plasmado aquí parece bastante lógico, cuando un algoritmo comienza a tener una longitud considerable, muchas veces ocurre que el orden de las instrucciones se pierde un poco de vista, a pesar de tener un rol fundamental. Si bien uno en la vida atraviesa situaciones en la es que más fácil primero pedir perdón que preguntar, cuando de dinero se trata, le aseguro que será mucho mejor preguntar, repreguntar y volver a preguntar antes de ejecutar. En la bolsa los errores se pagan con dinero y, si bien ocurrirá indefectiblemente alguna vez, usted no está leyendo esto para perder capital.

## 3.2. ¿Qué son las APIs?

Una API (Application Programming Interface) es simplemente un mecanismo para conectar dos programas entre sí para el intercambio de información en general. Básicamente es como un puente entre dos softwares, para que ambos puedan comunicarse y entenderse. En el caso de las APIs que utilizaremos en este libro, son como un puente entre usted y su cuenta comitente para que pueda operar en la bolsa de valores directamente, de forma sencilla y sin la necesidad de pasar por una plataforma.

Los datos que se pueden obtener desde la mayoría de las APIs son una réplica exacta de los que se encuentran en la plataforma y se les ofrece a los clientes tradicionales. La única diferencia es que esta última cuenta con desarrollos gráficos que embellecen la experiencia de navegar por las diferentes secciones de la plataforma, mientras que las APIs, si bien cuentan con la misma información, se orientan a brindar la información pura y dura para agilizar la operatoria. Esto es justamente lo que se busca para la operatoria algorítmica. Las respuestas a las consultas que hagas por ellas se encontrarán en formato “JSON”, un formato de texto ligero para intercambio de datos.

En definitiva, con los servicios de APIs que veremos aquí, usted podrá obtener información de todos los instrumentos de Argentina, obtener información de los instrumentos de Estados Unidos, hacer seguimiento de su portafolio, verificar el estado de su cuenta comitente, confirmar cuánto saldo disponible tiene para operar, comprar y vender. Todo en tiempo real y sin pérdidas de tiempo. Lo que puede hacer con todo esto depende únicamente de su imaginación, ya que se podrían crear algoritmos propios de todo tipo.

Orientemos un poco nuestra creatividad. Supongamos que usted no es un hábil operador y simplemente hace alguna que otra compra o venta esporádica. Sin embargo, sabe que para limitar una posible pérdida quiere vender un instrumento cuando alcance un determinado nivel o, en el sentido contrario, quiere tomar ganancias y liquidar cuando un instrumento alcance la rentabilidad que esperaba. En ambos casos usted va a programar una venta. Entonces su algoritmo consistirá simplemente en dos pasos: primero, consultará la cotización del instrumento en cuestión durante la rueda de operatoria; segundo, ejecutará una venta sólo cuando el instrumento alcance el objetivo.

Veamos otra aplicación que se puede hacer con las APIs, ahora un poco más compleja. Supongamos que usted opera regularmente, su estrategia se fundamenta en indicadores técnicos y, ante la combinación de una determinada cantidad de señales, se ejecuta una compra o una venta. Con las APIs se solucionan estos problemas de forma automática. Simplemente va a consultar la información que necesita para calcular los indicadores técnicos (precio y volumen, por ejemplo) y va a replicar exactamente el mismo accionar que en el ejemplo anterior. Ante el cumplimiento de sus condiciones, va a programar ejecutar una compra o una venta, y así poder finalmente disfrutar de su tiempo libre mientras tu algoritmo opera por usted.

## 3.3. Estructuras básicas

Dentro de los paradigmas más básicos de programación, existe el concepto de “programación estructurada” con el que estará indefectiblemente en contacto ya que sus recursos son necesarios para desarrollar prácticamente todo algoritmo. Estos recursos, están expresados a modo de subrutinas, dentro de donde nuestros algoritmos trabajarán continuamente hasta cumplir la regla de finalización que nosotros le pongamos. Este paradigma afirma que todo código puede estar escrito con: sentencias secuenciales, rutinas de selección y rutinas de iteración. Aquí entraremos en detalle en las últimas dos (sentencias condicionales y bucles), ya que la primera no le representará dificultad entender ya que casi todos los lenguajes de programación son secuenciales, leen una línea por vez y no avanzan a la siguiente hasta haber terminado la anterior.

Anteriormente los scripts estaban compuestos por pequeños bloques de instrucciones cuya ejecución iba y venía de unos a otros sin respetar la secuencialidad del código. Esto hacía la lectura, interpretación y depuración demasiado engorrosa y se decía que los códigos eran un “spaghetti” por lo difícil de su interpretación. Este nueva forma de codear se originó para facilitar la lectura y permitir que los códigos puedan ser leídos de forma secuencial (como está leyendo este libro, primero el primer renglón y luego el segundo; primero el primer párrafo, luego el siguiente). Elimina la necesidad de hacer engorrosos seguimientos de saltos de línea que dificultan el entendimiento de la lógica general del algoritmo.

## 3.3.1. Estructuras de selección

Las primeras estructuras básicas que usted debe conocer son las sentencias condicionales, los famosos “if”. Estas sentencias bifurcan el orden de la ejecución según el cumplimiento o no de una condición impuesta. Entiéndase que la forma más sencilla es bifurcar el código en dos, pero uno podría abrir la lógica la cantidad de veces que se necesite. Veamos un ejemplo clarísimo: usted necesita salir a la calle y observa a través de la ventana para confirmar si llueve, ya que en ese caso debería llevar paraguas. Entonces su curso de acción sigue muy sencillamente según la veracidad o falsedad de esta simple condición: si es verdadero que llueve usted lleva paraguas, en caso contrario no lo hace.

Una vez más, la simpleza del ejemplo obliga a quien escribe a confesar que en la vida real de un programador la situación no es tan fácil. Este tipo de sentencias tienen un rol crucial en todos los tipos de código pero, particularmente en el algotrading, es fundamental su correcta expresión y colocación. Estas condiciones determinarán en última instancia la correcta detección y ejecución de una estrategia de trading. De nada valdría todo su esfuerzo si su algoritmo es incapaz de detectar una oportunidad de operatoria o si opera incorrectamente luego de su detección. El trabajo del programador al expresar la condición que bifurcará el accionar de su algoritmo es todo un arte, aquí exprimirá al máximo su imaginación y lógica para plasmar en una simple expresión todo un fenómeno financiero.

Una sentencia condicional está conformada habitualmente por tres partes. La primer parte es la que mencionamos recién: la sentencia de la condición. Esta parte como dijimos, es crucial para identificar correctamente el punto de inflexión que se necesita para dividir la naturaleza del problema en, al menos, las dos siguientes partes. Si bien en el lenguaje en el que solemos emplear en nuestras vidas resulta sencilla la expresión, cuando le estamos diciendo a la computadora qué hacer, puede representar otra complejidad. Para empezar, no estamos utilizando el lenguaje con el que solemos pensar diariamente y, por el otro lado, necesitamos una avanzada capacidad de abstracción y simplificación de la habilidad que tiene nuestro cerebro para solucionar todo tipo de problemas. Como dijimos, la computadora es más efectiva que nosotros, pero necesitamos adecuar nuestra interacción a su modo de interpretar infinitamente literal. Aquí no vale el “lo que yo quise decir en realidad era…”.

La segunda y tercer parte de las sentencias condicionales son las dos caras de la misma moneda. Una es la porción de código que se ejecutará cuando la condición resulte verdadera y la otra es la porción de código que se ejecutará cuando la condición sea falsa. Comúnmente se programa la condición en tono afirmativo, para que la porción que siga sea la positiva, pero uno podría declarar la condición en tono negativo para seguir codeando la parte negativa primero. Aquí entra su imaginación y su propio estilo. Quizás se maree un poco ahora, pero entienda que uno podría expresar la disyuntiva de llevar o no paraguas afirmando “hoy llueve” u “hoy no llueve”. Lo que sí debe entender desde este momento es que casi todos los lenguajes de programación estructuran las sentencias condicionales primero con la condición, luego con el código que se ejecuta cuando la condición es verdadera y por último con el código que se ejecuta cuando la condición es falsa.

Las sentencias condicionales están presentes en todos los algoritmos que vayan más allá de la simpleza de la secuencialidad de las instrucciones. Le brindan una determinada inteligencia a los scripts que programemos debido a la capacidad de identificar diferentes escenarios y las instrucciones claras (o al menos a eso apuntamos) que deben seguirse dependiendo del escenario con el que nos enfrentemos. Particularmente en los algoritmos de trading algorítmico, este tipo de sentencias es el recurso más importante a mí entender. A no ser que uno quiera programar un robot sangriento que sólo ejecute, uno quiere que al menos interprete información y ejecute la operación correspondiente. Uno no debe comprar, comprar y sólo comprar. En la bolsa a veces hay que comprar, otras vender y a veces simplemente esperar afuera sin posición.

## 3.3.2 Estructuras de iteración

Otro de los recursos de la programación estructurada que usted utilizará con absolutamente todos los lenguajes de programación que aprenda son las subrutinas de iteración. Aquí entraremos en detalle en los “for” y los “while”. Además de estos, quizás usted tenga a su disposición otros parecidos. Todos los problemas en donde necesite iterar podrán ser solucionados con cualquiera de estos dos recursos. Todo depende de cómo uno exprese la sentencia, del mismo modo que antes explicamos que se puede declarar una condición en tono positivo o en tono negativo. Lógicamente

Supongamos que tenemos un algoritmo perfectamente codeado, todas sus sentencias son claras, precisas y eliminan cualquier posibilidad de ambigüedad. Confirmamos que funciona como uno quiere y ahora queremos extender este fabuloso desarrollo más allá del instrumento financiero con el que lo desarrollamos, y deseamos que recorra todo un grupo de veinte activos diferentes. Las bondades de la programación estructurada nos ahorrarán el extenso trabajo que sería tener que multiplicar por veinte nuestro script para que el robot recorra todo el grupo. Con las rutinas de iteración simplificaremos el desafío. Imagínese si luego de multiplicar 20 veces su desarrollo se da cuenta que tiene que hacer una pequeña modificación. Perdería invalorable tiempo en hacer todas esas modificaciones y los códigos avanzados se volverían inentendibles a medida que se extiendan.

Volviendo a lo nuestro, tenemos los “for” y los “while”. Ambos siguen la estructura que pone en primer lugar una condición y luego una subrutina que iterarán hasta que la condición deje de cumplirse. Tan simple como eso. La diferencia entre uno y el otro es que, y dicho en español, el “for” itera la subrutina en cada uno de los elementos de un grupo que se defina en la condición, mientras que el “while” ejecuta iterativamente la subrutina hasta que la condición deje de ser verdadera. Entonces, verá que en el código dirá: “por cada elemento de un grupo, itere la siguiente rutina” o “mientras la condición se cumpla, itere la siguiente rutina”. Podríamos, por ejemplo, usar un “for” para decirle a nuestro algoritmo que itere una rutina por cada acción dentro de nuestro grupo, y podríamos también incluir un “while” para decirle que continúe sólo cuando el horario de la ejecución sea anterior al horario de cierre de mercado.

Cuando programamos es muy importante la parsimonia. Si podemos ir al grano sin mucho rodeo, mejor. No es para nada fácil retomar un trabajo unos días después de haber programado algo que da vueltas y vueltas sobre un asunto que podría haberse abordado de una forma más directa. Cuanto más directo sea el algoritmo, más veloz y más legible será para entenderlo luego. Por esto, a pesar que podamos intercambiar estos recursos como se nos dé la gana, siempre habrá alguno de los dos que se adapte mejor a la situación. Por esto, es muy recomendable familiarizarse con ambos a pesar que le guste más alguno de ellos.

# **Módulo V: Programación en Python**

Si ha llegado hasta este punto quiere decir que tiene todo lo necesario para formar parte del reducido grupo de jugadores que dominarán los mercados financieros del futuro cercano: los algotraders. Y esto no es sólo un decir, sino que realmente lo único que es necesario, es tener determinación y constancia. Lo demás se irá incorporando sobre la marcha. Incluso, si bien podría comenzar este largo camino desde infinidad de perspectivas, en este módulo haremos que usted lo haga dominando el lenguaje de programación más caliente del mercado actual. Le aseguramos que al finalizar este módulo e implementar los ejemplos que mostraremos, pasará de ser un simple usuario de planillas de cálculo a un hábil programador que podrá automatizar operaciones sencillas al principio, y cada vez más complejas con la práctica y esfuerzo que usted le dedique. Cambiará totalmente su percepción de este fascinante mundo del trading algorítmico y se preguntará por qué no empezó antes. Verá que la barrera de entrada a esta práctica no es tan hostil como pensaba y habiendo leído con atención los módulos anteriores, podrá marcar la diferencia en el mercado con las simples herramientas que se mostrarán aquí.

Aprendió qué es el trading algorítmico, parte de su historia y también aprendió cómo darle una vuelta de tuerca a su fiel y conocida planilla de cálculo que domina muy bien. Podría frenar ahí, especializarse en esa excelente herramienta y alejarse de esas pantallas negras con letras verdes que distan más de su realidad que una película de ciencia ficción. Con las herramientas que usted repasó en los módulos anteriores, verdaderamente podrá cohibir los impulsos emocionales que le hacen tomar malas decisiones financieras utilizando elegantemente lo que en la city se denomina algotrading. Pero si realmente desea que las probabilidades de que los tiburones del mercado no lo desplumen en los primeros pocos intentos, recomendamos fuertemente que se prepare con lo mejor.

El poder de cómputo de estas nuevas herramientas es crucial para reducir al máximo los tiempos muertos en las estrategias de trading. Los mercados financieros cambian diariamente en cuestión de segundos y, si nuestros cálculos no son instantáneos, podríamos perder oportunidades financieras únicas e irrepetibles. No se necesita ser un experto informático para lograrlo, sólo se necesita contar con una buena estrategia y saber cómo plasmarla en un algoritmo que la ejecute infaliblemente. A continuación, transitaremos por una guía que lo acompañará casi desde el botón de encendido de su computadora hasta sus primeras operaciones lanzadas a la bolsa de valores por su algoritmo.

Desde aquí, le recomiendo que vaya descargando las únicas dos cosas que necesitará para seguir y entender todas las líneas que vienen a continuación. En primer lugar, descargue Python desde la página oficial: www.python.org. Una vez allí descargue la última versión disponible. Es muy importante que al momento de la instalación tilde la opción de “Add Python to PATH”. Esto le ahorrará problemas a futuro.

Lo segundo que tendrá que hacer es descargarse el programa en donde programaremos. No lo haremos directamente en Python, sino a través de un IDE (luego veremos qué es esto). Por fines didácticos usaremos el programa “Spyder”, ya que considero que es simple de usar y además muy completo. Para descargarlo, diríjase a https://www.anaconda.com/distribution/ y descargue la última versión.

Si en estos pasos clave encuentra alguna complicación, porque no entiende algo (¡es lógico!) o porque quizás los links hayan cambiado desde el momento de la impresión del libro hasta su lectura, no se desanime. Como diré más adelante, hoy en día ante cualquier duda podemos aprovechar las bondades de nuestra época y buscar en internet lo que se nos ocurra. Busque el problema en donde está trabado o pegue en el buscador el error exacto que le está apareciendo y le aseguro que encontrará un tutorial que le ayudará a atravesar cualquier complicación. Saber cómo utilizar un buscador de internet es fundamental para el trabajo diario de cualquier programador, así que vaya practicando. Siempre es recomendable utilizar los buscadores utilizando el inglés porque es el idioma donde hay más contenido.

## 5.1. Porqué Python

Todos los lenguajes de programación tienen sus semejanzas. Incluso un programador profesional probablemente diría que son sintácticamente iguales. Todos tienen las estructuras “for”, “while” e “if” que hemos visto, por ejemplo. Como ocurre con los idiomas, a medida que se aprenden varios, comienzan a surgir puntos en común que hacen más fácil el proceso. Entonces, si bien todos podrían tener sintaxis relativamente parecida, el porqué de comenzar particularmente por éste lenguaje se encuentra en que tiene algunas características que podrían facilitar la entrada al mundo de la programación.

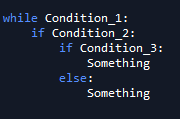
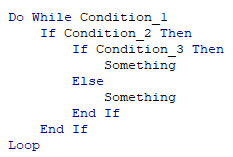
### 5.1.1. Sintaxis fácil de entender

Python, en primer lugar, posee una escritura que se asemeja al inglés escrito y permite entender desde el comienzo qué están queriendo indicar las líneas que se ven programadas. Aunque usted no tenga un inglés londinense, es muy probable que al encontrarse con los términos que se revisarán en este módulo entienda perfectamente el sentido que le queremos dar a nuestro código y pueda reproducir las estructuras de comandos para solucionar los desafíos que se plantee.

Ante la necesidad de entender un script, no debe existir peor sensación que la de no entender ni una sola palabra de lo que está leyendo o no saber si se comienza a leer desde arriba o por el centro. Aunque parezca descabellado, cuando se intenta descifrar un código de HTML de una página web, es muy común querer obtener los atributos de un elemento que se ubica bien por arriba de la página web que se está desarrollando, pero cuando se va a ver el código el elemento en cuestión está mucho más al medio.

Dentro del mundillo de los programadores (al cual pertenecerá al terminar este libro) es muy común escuchar que alguien ha pasado las últimas horas tratando de debuggear un script, o sea arreglando un programa. Como hemos dicho, a diferencia de los idiomas que hablamos, los lenguajes de programación poseen estructuras inmutables que se ejecutan satisfactoriamente únicamente con la sintaxis correcta, o al menos ejecutan lo que realmente se pretende. Esto hace que las reglas y convenciones de programación de cada lenguaje cumplan un rol bastante importante en lo que respecta a evitar errores involuntarios.

Hemos visto en el módulo anterior que VBA utiliza una estructura “begin…end” que obliga recordar cuándo debe finalizar una función o detener un proceso. Esto hace que muy probablemente cometa errores al programar un script de corrido y que deba volver a revisarlo al menos una vez. Intuitivamente, la razón parece que tiene sentido: habitualmente no solemos dar aviso del final de nuestras frases u oraciones, simplemente las terminamos. Entonces, a no ser que esté acostumbrado a manejar una radio militar y decir “cambio y fuera”, lo más común es que se olvide de respetar alguna regla y pierda tiempo en debuggear su algoritmo.

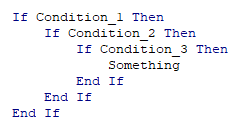


Fíjese la diferencia entre VBA y Python. Note cuántas líneas menos se necesitan incluir. Por otro lado, existen lenguajes que requieren el continuo uso de llaves para encapsular sus funciones o métodos (estos dos términos se refieren casi a lo mismo pero, como aclararemos más adelante, los últimos están ligados a objetos pero las primeras no). La necesidad de expresar el final de una declaración puede ser tan favorable para el ordenamiento de toda la estructura del script como tedioso y problemático para el programador. Mucho más si no es experto. Ni mencionar la necesidad o la simple costumbre de declarar un punto y coma al final de cada una de las líneas que componen su programa. Imagínese que en un script de ochocientas o novecientas líneas, algo muy común, se haya olvidado de poner un punto y coma en algún lugar. No solo es peligroso para el programador sino también poco favorable para la parsimonia visual de los algoritmos que programará.



Para declarar bloques de código en Python, por ejemplo para crear una nueva función, simplemente se utiliza una sangría o indentación. Teóricamente una indentación es una misma cantidad de espacios al principio de cada línea para diferenciar un bloque de código. Aunque esa cantidad pueda ser cualquiera, se recomienda usar siempre cuatro espacios o simplemente una tabulación, siendo esta última la más usada por su comodidad (no es placentero contar cuatro barras espaciadoras con cada nueva línea, ni para usted ni para sus compañeros de piso). Tampoco se necesita introducir ningún signo al final de cada renglón, simplemente crea uno nuevo y el mismo lenguaje lo entenderá. Incluso la mayoría de softwares lo posicionará en el mismo nivel de indentación de la línea que acaba de terminar para continuar con su bloque de código.

Al finalizar cada declaración no es necesario declarar su final, simplemente se quita la indentación y se vuelve a la posición inicial. Este simple detalle, aunque no parezca, soluciona muchísimo la tarea de programar un algoritmo. Una sentencia condicional múltiple en Python se ahorra la necesidad de contar cuántos “if” ha programado para asegurarse de haber declarado la misma cantidad de “end if”, sumado a que estén en su posición correcta del script. En el ejemplo que continúa verá la diferencia y tenga en cuenta que la indentación en VBA no es obligatoria, sólo que a quién escribe le parece visualmente más ordenado. Uno podría programar todo sin indentación, pero podría complicar aún más el problema que se menciona.



Esto tampoco significa que programar en este lenguaje sea pan comido. Usted tendrá errores, como los tienen todos los programadores. Nadie se salva de los bugs. Pero sin duda alguna los problemas de indentación y finalizar sentencias serán muchos menos y serán detectados más fácilmente. Incluso llegará a la conclusión, o al menos eso esperamos, que sus algoritmos tendrán cierta belleza visual. Seguramente la simplicidad y parsimonia de la indentación tendrá gran importancia en ello.

### 5.1.2. Lenguaje gratis y de código abierto

La puerta de entrada al fascinante mundo de la programación, en el caso de Python, está completamente abierta ya que no requiere invertir ni un centavo. Imagínese que muere de ganas por aprender a usar un nuevo software, pero se da cuenta que para hacerlo tiene que comprar un costoso paquete de programas y complementos. Ni siquiera sabe si eso es lo suyo y ya tiene que desembolsar grandes sumas de dinero. En muchos casos, los softwares especializados que se utilizan en las altas esferas del mundo financiero son extremadamente costosos. Sin embargo, no hay nada que no pueda hacer completamente a sus gustos y medidas con el poder de Python.

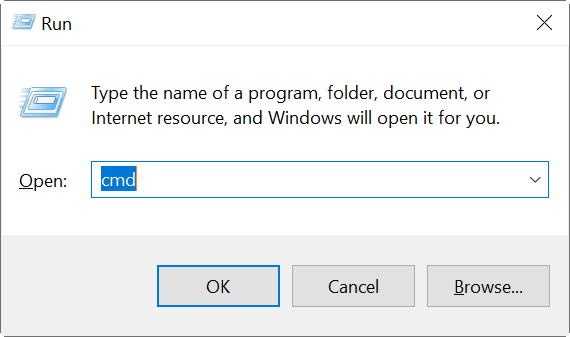
Lo fascinante de este lenguaje es que no sólo no tendrá que pagar ni un solo centavo por el software en sí, sino que tampoco tendrá que preocuparse por invertir en paquetes, complementos ni librerías. Estas últimas son muy importantes y veremos al final de este inciso cómo descargarlas gratuitamente. En esta comunidad, en reglas generales, es todo completamente gratis. Esto hace que no existan barreras de entrada ni contratiempos monetarios en el recorrido de aprendizaje. Ni siquiera se venden los compiladores profesionales que utilizaremos en este módulo. En su amplia mayoría, los recursos necesarios para emprender proyectos de todo tipo son completamente gratuitos.

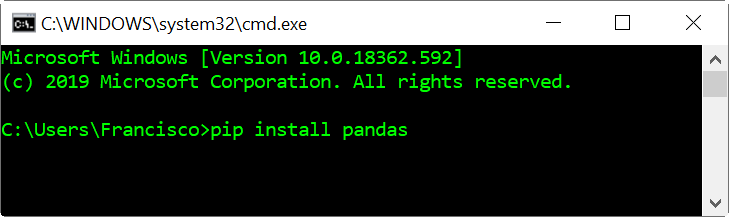
Existen librerías que proveen modelos estadísticos para implementarlos fácilmente sin tener que programarlos desde cero, otras que facilitan los cálculos matemáticos complejos y mejoran evidentemente el rendimiento computacional, también hay algunas que ayudan a crear interfaces gráficas y otras que permiten desarrollar cualquier cosa que uno quiera hacer con la computadora. Es realmente impresionante todo lo que se puede hacer sabiendo de programación, y con Python es posible de forma gratuita y de manera relativamente sencilla.

Como si esto fuera poco, el código de todo todas las librerías mencionadas anteriormente está completamente abierto. Esto significa que en caso de encontrar una librería que resulte útil o simplemente interesante, uno podría investigar cómo está programada, entrar hasta las líneas de código más recónditas y aprender cómo interactúan todos los segmentos internos. No sólo estaría leyendo programación avanzada de manera implícita, sino también podría aprender a replicar las líneas de código que crea conveniente, para solucionar o mejorar cualquier cosa que usted esté desarrollando. Uno podría tomar una sección de código ya programado por otro desarrollador, adaptarla a la naturaleza propia de un nuevo problema y hallar una solución innovadora para un desafío, por ejemplo, de automatización de procesos manuales. También se podría investigar cómo responde la librería a una determinada situación, para adaptar el código original, o quizás, para enviar una recomendación al desarrollador original.

Me ha pasado que al usar una librería de cálculo avanzado muy conocida, mi algoritmo devolvía un resultado inesperado. Yo sabía qué resultado tenía que aparecer, pero al procesar todo el cómputo, Python me devolvía un número que no coincidía con mi cálculo manual y ni siquiera con el modelo preliminar hecho en Excel. Fue ahí donde aproveché la apertura del código original y descubrí que la información viajaba desde una sección de la librería hacia otra aplicando redondeos en cada uno de los viajes. Eran cuestiones prácticamente imperceptibles, pero en este caso los decimales eran fundamentales ya que significaban una variación sustancial en el valor del derivado financiero que estaba valuando. Gracias a que la librería tenía código abierto no fue necesario modificar nada del algoritmo en general, sólo se tuvo que agrandar la cantidad de decimales que la librería tomaba por defecto. De esta forma, el cálculo terminó siendo tan preciso como se esperaba y se pudo implementar una valuación de derivados de alto nivel.

Para descargar las librerías de forma totalmente gratuita debemos ingresar a la línea de comando de nuestra computadora. Si tiene sistema operativo Windows presione en conjunto las teclas Ctrl + R, luego escriba “cmd” y presione la tecla enter. Por último, en la ventana que se le abrirá, deberá ingresar el comando “pip install” seguido del nombre de la librería que quiera instalar. Por ejemplo, una que usará indudablemente es una que se llama “pandas” y sirve para el manejo de datos.





Si no tiene sistema operativo Windows o si el mensaje de la última pantalla no es satisfactorio, le recomendaré que busque en su navegador de internet la solución específica para el problema que está atravesando. Esto no es sólo por el hecho que no podríamos brindar en un pequeño libro una solución exhaustiva para todos los problemas que enfrentará, sino también porque Python tiene otra gran ventaja y la expresaremos a continuación.

### 3.1.3. Gran comunidad

Como ocurre en la actualidad con prácticamente cualquier tema, lo más probable es que al toparse con alguna duda o inquietud, lo primero que haga es buscar ayuda en su buscador de internet. Como digo habitualmente cuando me preguntan cómo aprendí a programar, creo que la programación es un 80% de búsquedas en internet y el 20% restante es prueba y error. Entonces, para nuestra suerte, tener una comunidad de expertos que recorren todos los temas habidos y por haber de nuestro lenguaje, es fundamental para atravesar ese primer 80% necesario para aprender.

En el transcurso de mi experiencia, he compartido con la comunidad material de todo tipo. No es que existe soporte únicamente para los temas sencillos o triviales y después estará solo enfrentándose con los problemas más complejos. Realmente existe información de absolutamente todas las librerías y todos los métodos. Hay especialistas que hace trabajos impresionantes analizando dos librerías que cumplen el mismo objetivo para comparar sus ventajas y sus desventajas. Hay gente también que comparte herramientas que no son las más conocidas, pero terminan siendo las más eficientes para casos particulares o que pueden trabajar con formatos de datos que las otras no permiten. Básicamente, si no existe información sobre un tema en los principales foros, ese tema directamente no existe.

Existirán veces en que la comunidad ya tenga muchos artículos relacionados con su problema y no tenga que hacer grandes esfuerzos para reproducir la solución en su script. Pero, si bien tendrá la suerte a veces de limitar su creatividad resolutiva a un simple copiar y pegar desde un foro, hay otras en que la comunidad brindará simplemente una orientación para poner en juego el otro 20% restante del proceso de aprendizaje. En ese momento será crucial recurrir a la documentación oficial de las herramientas que está utilizando y leer información de todos los métodos de la librería que se intenta aplicar. Llegado a este punto, su tarea se volverá un “probar y corregir” constante hasta que su programa compile bien y haga lo que usted está intentando hacer.

Es ahí donde podrá devolver a la comunidad la enorme ayuda que le aseguro que le dará y hacer públicos los avances que usted pudo lograr por cuenta propia para contribuir con gente igual que usted que podría estar luchando con una línea de código que no le está funcionando. Me ha pasado de programar varios scripts por separado y confirmar que individualmente funcionan todos a la perfección, pero a la hora de ensamblarlos para transformarlos en un programa integral detectar que hay una pequeñísima sección que no funciona como esperaba. En este punto, los aportes de otros programadores, que quizás no están resolviendo un problema de la misma naturaleza pero están usando las mismas herramientas, son de crucial importancia para nutrir a la comunidad de información útil que derive finalmente en soluciones.

No es simplemente ayudar y ser ayudado, es más bien sumarse a una especie de construcción colectiva. Cuando programé mi primer optimizador de cartera de inversión vía una maximización de rentabilidad esperada, lo hice público en un portal en internet que los programadores usan mucho para compartir sus logros personales y mostrar una especie de habilidades al estilo de un currículum. Al poco tiempo noté que una persona del otro lado del mundo usó mi desarrollo para incluirlo en un problema matemático que no tenía nada que ver con mi problema financiero, pero como le servía mi maximizador matemático, lo terminó usando. Lo mejor de todo, es que este programador le incluyó una nueva sección que incorporaba restricciones lineales a todo el sistema y permitía hacer optimizaciones más complejas. Y así fue como darle gratis a la comunidad un desarrollo que tanto esfuerzo me costó, terminó siendo una nueva ayuda para mi aprendizaje personal. Mi optimizador de portafolio, tan simple que era, terminó incluyendo restricciones que me permitían decirle a mi algoritmo que no se sobrepase un determinado porcentaje de acciones y otro porcentaje de bonos, y así transformarlo en un mejor recurso financiero.

5.1.4. Multiparadigma

Como ocurre con los idiomas, en los lenguajes de programación existen paradigmas que delinean la estructura básica de las expresiones y determinan la manera en la que se ensamblan unas con las otras. Cuando uno sabe hablar español, es natural sentir que resulta mucho más fácil aprender a hablar italiano o portugués, que empezar a estudiar alemán. Esto no ocurre simplemente porque el alemán es difícil, sino más bien porque nuestro idioma materno proviene de la misma familia de lenguas que el italiano, el portugués, el francés y unos cuantos más. Hubiese sido totalmente al revés si naturalmente hablásemos alemán. Veríamos el español como un idioma sumamente complicado y quizás preferiríamos aprender neerlandés o noruego.

Este libro no pretende profundizar en particularidades lingüísticas, pero para entender lo anterior un poco mejor note simplemente la diferencia en la posición de los adjetivos en los diferentes idiomas. En las lenguas romances comúnmente uno ubica el adjetivo detrás del sustantivo. Así es como uno dice “manzana roja”, “pomme rouge” y “mela rossa” en español, francés e italiano respectivamente. Ocurre exactamente lo opuesto con las lenguas germánicas, porque uno debe poner el adjetivo antes del sustantivo. Por eso mismo, si sabemos que la misma fruta se dice “apfel”, “appel” y “eple” en alemán, neerlandés y noruego respectivamente, entenderíamos que para describir su color deberíamos colocar el adjetivo antes y decir “roter apfel”, “rode appel” y “rødt eple”. Qué difícil, ¿no?

Este detalle, aunque nunca lo hubiese pensado es bastante importante en el mundo de la programación. A diferencia de los idiomas, en donde de una u otra manera uno podría expresar la misma idea, con los lenguajes de programación el paradigma que utiliza podría representar una ventaja o una desventaja según cuál sea el objetivo del desarrollador. Por esto mismo es muy común escuchar que el primer paso para aprender a programar es saber qué se quiere hacer, ya que de esto dependerá qué lenguaje es el indicado para el proyecto. Si uno quiere construir una estructura de elementos que interactúen entre sí y compartan atributos, no sería correcto utilizar un lenguaje de programación con paradigma “imperativo”, el cual espera que se le indiquen secuencias de comandos que ordenan acciones a la computadora. Lo correcto sería utilizar un lenguaje que maneje el paradigma de “orientación a objetos”, el cual trabaja con entidades que representan elementos del problema a resolver, tienen atributos y comportamientos.

Hay muchos paradigmas de programación. Además de los mencionados anteriormente, dentro del grupo de los más usados, existe el declarativo, el lógico y el funcional. Cada uno de ellos no es mejor ni peor que el resto, sino que cada paradigma es ideal para la resolución de un conjunto de problemas particular y por ello es tan importante saber qué queremos desarrollar de antemano. De nada nos serviría aprender un lenguaje que no esté diseñado para cumplir el desafío que nos estamos planteando. Otra alternativa, sino, es directamente aprender un lenguaje que pueda interpretar varios paradigmas al mismo tiempo, como lo hace Python.

Un lenguaje multiparadigma es aquel que tiene la posibilidad de soportar muchos paradigmas según cuál sea el objetivo principal el desarrollo. De esta forma, Python permite a los programadores utilizar la mejor herramienta para cada trabajo. Por ejemplo podemos tener códigos del paradigma funcional, con restricciones u orientado a objeto, dependiendo de cuál función está siendo evaluada, qué información es usada o qué manipulación de estado es necesaria. Aprender y emplear un lenguaje multiparadigma es muy valioso ya que estarán a nuestro alcance, las herramientas más destacadas de cada tipo de paradigma, para usarlas de la mejor manera ante los diversos problemas que debemos tratar de solucionar, y principalmente lograrlo de la forma más eficiente posible.

### 5.1.5. Lenguaje interpretado

El objetivo final de todos los lenguajes de programación es enviarle una serie de instrucciones a la computadora para que esta las ejecute. La cuestión está en que la máquina no entiende todos los lenguajes que existen, de hecho, sólo entiende uno: el binario. Es por eso que tanto los lenguajes “compilados” como los “interpretados”, en última instancia, convierten el código que los programadores escriben en un conjunto de unos y ceros para que la computadora pueda entender qué se le está ordenando.

La principal diferencia entre un lenguaje compilado y uno interpretado es que el lenguaje compilado requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código que escribes a lenguaje de máquina. Un lenguaje interpretado, por otro lado, es convertido a lenguaje de máquina a medida que es ejecutado. Generalmente a los lenguajes interpretados se los conoce como “de alto nivel”, pero no porque sean más avanzados, sino porque están más alejados del lenguaje binario que es el de más abstracto de todos.

Lo bueno que tiene Python es que al ser un lenguaje interpretado, hace que los tiempos de desarrollo en general sean más cortos y se facilite la vida de los programadores. ¡Justamente es lo que estamos buscando en este período de aprendizaje! A diferencia de los lenguajes interpretados, con los compilados es necesario realizar un proceso de compilación cada vez que se hace hasta el más mínimo cambio del código. Imagínese tener que compilar infinidad de veces el mismo código hasta poder dar en el clavo y descubrir cómo se usa la librería que está queriendo implementar. Resultaría un proceso muy tedioso.

Otra de las ventajas que tiene Python, como otros lenguajes interpretados, es que es posible ejecutarlo desde cualquier sistema operativo. No importa si usted tiene una computadora con Windows de Microsoft, Mac OS de Apple o si tiene una con Linux. En todas podrá utilizar sin problemas este lenguaje. Cuando se utiliza un lenguaje compilado, es necesario crear un ejecutable diferente para cada uno de los sistemas operativos en donde se vaya a utilizar el programa. Un ejecutable creado para Linux no va a servir en Windows. Imagine las complicaciones que podría traer si su compañero de equipo tiene un sistema operativo diferente al suyo.

De igual forma, no todo está a favor de los lenguajes interpretados. Dado que los lenguajes compilados ya están en código de máquina cuando se ejecutan, son un poco más rápidos que los interpretados. Aunque hoy en día los procesadores son tan avanzados que esta desventaja es prácticamente imperceptible. Además de la velocidad, otra desventaja de un lenguaje interpretado es que, para ser ejecutado, hay que tener instalado el interpretador. Esto no es necesario en un lenguaje compilado ya que es convertido directamente a lenguaje de máquina.

5.1.6 Tipado fuerte y dinámico

El tipado se refiere a cómo declaramos los tipos de variables. Uno podría declarar a la variable “X” como un entero, un flotante, una cadena de texto o como otras cosas. En algunos lenguajes, no es necesario declarar cuál es el tipo de las variables, ya que este lo entiende solo. Es aquí, en esta cuestión que parece tan pequeña, donde se genera una gran distinción entre lenguajes de programación. No sólo por la necesidad de declarar el tipo de nuestras variables explícitamente, sino también por la capacidad de hacer algunas operaciones entre distintas variables.

Cuando no indicamos el tipo de variable que estamos declarando, se dice que el lenguaje es de “tipado débil”. Estos lenguajes nos permiten hacer operaciones aritméticas extrañas que podrían no tener sentido para nosotros, pero al no declarar el tipo, nuestro programa ejecuta literalmente lo que tenemos codeado. Un ejemplo claro sería la típica operación “5” + 5. La primera variable, aunque el ojo no entrenado aún no se dé cuenta, es una cadena de texto. Si bien refleja claramente el número cinco, el hecho de estar entre comillas hace que la computadora la interprete como la cifra cinco expresada en texto. Por esto mismo, lenguajes de tipado débil como Javascript no emitirán ninguna alerta, computarán literalmente la orden y emitirán “55” (concatena ambas variables en un resultado).

Imagínese que si está desarrollando un complejo algoritmo matemático, no sería muy profesional que su programa arroje que “5” + 5 = “55”. Quien no se dé cuenta que hay un problema de tipos, pensará que usted no está capacitado ni para usar una calculadora. Y aunque parezca algo obvio en estas líneas, le aseguro que muchas veces es realmente difícil detectar que el problema viene por este pequeño detalle. Los más probable es que primero pierda tiempo buscando en qué parte del código exista algún error matemático mucho antes de verificar los tipos de las variables.

También hay que destacar que Python es un lenguaje de programación que tiene tipado dinámico. A diferencia de otros lenguajes, como Java o C++, en este caso no es necesario declarar el tipo específico de cada variable al inicio del script, sino que se puede ir declarando en el mismo tiempo de ejecución. Esto le da cierta flexibilidad a nuestros desarrollos porque nos permite aplicarle cualquier valor a nuestras variables sin tener que preocuparnos por generar otras nuevas o por tener que declararlas nuevamente al principio.

Suponga que tiene un largo script ya desarrollado y necesita declarar una variable temporal. La variable no sirve más que para almacenar una información temporalmente hasta que sea usada. La complicación comienza cuando, por alguna razón, necesita cambiar el tipo de la variable. Podría haber empezado siendo un número, pero ahora necesita que sea texto. Un programador profesional de un lenguaje estáticamente tipado soluciona este problema, ni más ni menos, que generando una variable nueva y perjudicando la parsimonia de todo el código. Son pequeños detalles, pero le aseguro que cuantas menos variables recordar, mejor.

Le puedo dar un ejemplo clarísimo que me ocurre todos los días cuando ejecuto mis algoritmos de trading. Para mejorar la rapidez de ejecución, suelo fragmentar un gran algoritmo en pequeñas partes. Esto también ayuda a la reutilización del código para otros fines. Es como tener pequeñas piezas de Lego que podrían ser utilizadas de distintas formas. Ocurre mucho que, para poder comunicar estos distintos fragmentos, es necesario crear variables que enviarán y recibirán los mensajes entre cada par de segmentos. El problema se origina cuando la variable envía un mensaje numérico (precio de la operación, por ejemplo), pero recibe un texto (“Ejecutada”, por ejemplo). Gracias a esta gran característica, Python es ideal para sortear este problema sin complicaciones.

## 5.2. Entorno de desarrollo integrado (IDE)

Los lenguajes de programación por sí solos suelen ser poco atractivos, por lo que nosotros programaremos todo en el programa Spyder que descargamos antes. Pruebe abrir Python directamente para comprobarlo usted mismo. Se le abrirá una ventana igual a una consola de comandos de Windows, con formato arcaico y sin ninguna cosmética que lo anime a continuar. Puede ser que se sienta dentro de la Matrix y quizás lo anime fanfarronear con sus amigos diciendo que se está volviendo una especie de hacker con este libro.

Nadie programa directamente en Python, el software está diseñado para otra cosa. El objetivo es más que nada ejecutar todas las cosas complejas que desarrollaremos, pero dedica poca atención a la plataforma gráfica destinada a que los usuarios se sientan cómodos y felices utilizándola. Es aquí donde entran los IDEs (entornos de desarrollo integrados, por sus siglas en inglés). Estos softwares facilitan la tarea del programador ofreciéndole, en un sólo lugar, todas las herramientas necesarias para encarar un desarrollo y además otorgan un marco de trabajo mucho más amigable que la mayoría de lenguajes de programación.

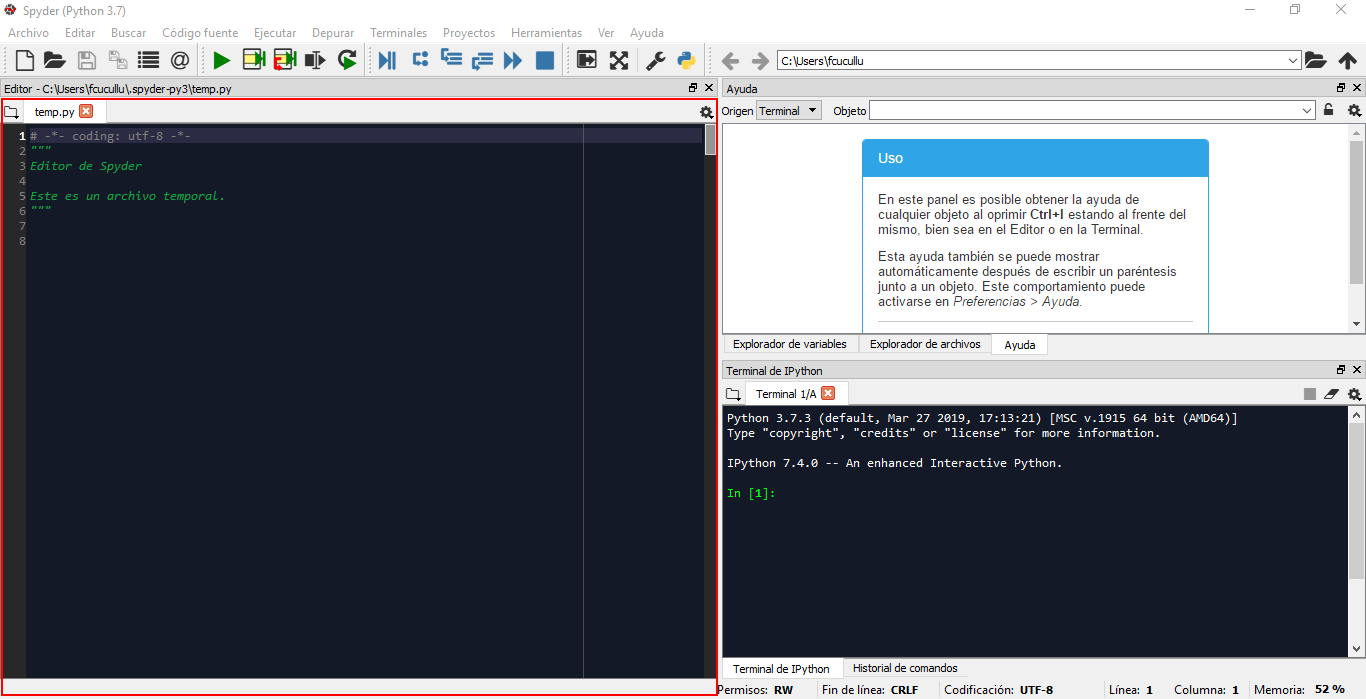
Para entender un poco mejor esta idea, piense que ocurre lo mismo cuando uno hace largos viajes en auto. Usted puede jactarse de tener un excelente último modelo, cómo lo hará cuando programe en Python, pero cuando utiliza su auto necesita cubrir muchas otras necesidades. Si comienza a hacer calor, recurre a una cómoda perilla y enciende el aire acondicionado. Si desea relajarse escuchando música, basta con encender la radio o conectar algún dispositivo al reproductor. Uno da por obvias todas estas cosas, pero cuando uno elige un auto, lo hace considerando muchas cosas además de únicamente prestar atención al motor.

Al igual que un conductor, un programador utiliza muchas herramientas durante el proceso de construcción de un algoritmo. Gracias a los IDEs uno se puede ahorrar el gran trabajo de buscar, descargar y conectar todas las herramientas. La tarea de unificación podría no ser nada fácil, además de tediosa. Imagínese pasando sus desarrollos de una herramienta a otra, tener que adaptarlos para finalmente testear todo el desarrollo, quizás, en otra aparte. Sería algo tan problemático como tener que acordarse de llevar el ventilador especial para su auto o tener que conseguir un reproductor de música que pueda ser adaptado al modelo, marca y año de fabricación de su vehículo. Impracticable. Por eso veamos a continuación qué nos ofrece comúnmente un IDE y particularmente el Spyder.

5.2.1 Editor de código

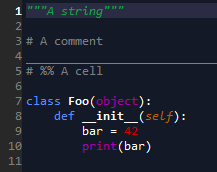
El editor será el lugar en donde estemos la mayor parte del tiempo. Es aquí en donde podremos escribir largos bloques de código para formar nuestros algoritmos. Si han probado abrir Python directamente como fue sugerido anteriormente, habrán visto que al apretar la tecla “enter” luego de cada línea el programa directamente lo ejecuta. Pero si uno quisiese tener varias líneas a la vista sin ejecutar, por ejemplo para visualizar toda la estructura de su algoritmo, sería difícil hacerlo desde allí. En general uno va construyendo de a poco todo el algoritmo, modificando cosas de aquí y de allá. Rara vez uno escribe todo un script de una sola vez.

La configuración inicial de Spyder ubicará al editor en la parte izquierda de su pantalla. Se resaltará una pequeña solapa por encima del campo disponible para escribir que indica cómo se llama el archivo que usted está editando. En la imagen verá que en el ejemplo el archivo se llama “temp.py” porque es simplemente un archivo temporal de prueba, pero al momento de guardarlo se cambiará al nombre que usted ponga. Todos los archivos ejecutables de Python tendrán la extensión “.py”. Algo a aclarar en este momento es que mi preferencia personal es utilizar el programa con sus colores oscuros, pero esto es completamente preferencia del usuario. Para modificar los colores podría ir a “Herramientas” -> “Preferencias” -> “Coloreado de Sintaxis” y elegir la que más le guste.

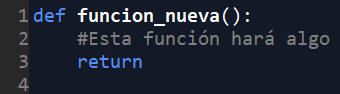
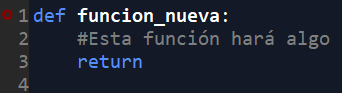
He conocido programadores que para que no se le ejecute cada línea que ingresaban y para construir grandes scripts programaban directamente en un bloc de notas. Si bien un hábil programador lo puede hacer sin ningún problema, le aseguro que programar en el editor de su IDE lo ayudará en infinidad de sentidos. Esta sección del programa está diseñada para volcar todas las ideas descabelladas que usted tenga para luego darles forma. Podría cortar y pegar diferentes bloques de texto, como también cambiar el orden de su algoritmo. Poner al final lo que antes estaba al principio, renombrar funciones o lo que sea necesario para construir, ordenar y optimizar su código.

Spyder nos ofrece un editor que es capaz de interpretar múltiples lenguajes. Aunque nosotros estemos aprendiendo Python, si el día de mañana necesita abrir un código HTML, por ejemplo, lo podría hacer tranquilamente en este software y disfrutar de las bondades que repasaremos. De hecho, me ha tocado construir un código HTML y lo hice directamente desde este programa. En aquella ocasión, creé una pequeña web con código HTML que era a su vez ejecutada sobre un servidor basado en Python. Lo importante de esta anécdota es que cuando aprenda a usar este u otro IDE, podrá construir scripts en varios lenguajes. No todos los IDEs están especialmente preparados para todos los lenguajes, pero en general le darán las herramientas básicas.

Lo que notará al instante cuando empiece a escribir aquí es que el software detecta automáticamente elementos claves a medida que uno va escribiendo y los resalta con colores específicos para facilitar toda la lectura. Si ha visto los distintos colores de sintaxis que ofrece Spyder, habrá visto que el programa le otorga un color específico según cual es el estilo que nosotros escojamos (también puede ser modificado individualmente). De esta forma, todas las funciones que definamos tendrán un color azul, los números serán rojos, los comentarios serán grises, y así sucesivamente según nuestras preferencias. Esto que quizás parezca estéticamente irrelevante no es simplemente para colorear nuestro código, sino para facilitar la orientación visual y saber dónde está cada elemento de nuestro desarrollo.



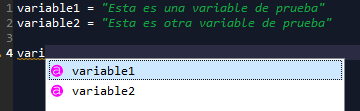
Si bien la parte visual es muy importante, la característica que más ayuda a programar es que Spyder hace un control en tiempo real de lo que vamos escribiendo y nos ayuda a corregir los pequeños errores que pueden ir ocurriendo (si es que no lo soluciona sólo). Un ejemplo clarísimo, y que le aseguro que le ocurrirá al principio, es olvidar los dos puntos al finalizar la definición de una función o de alguna estructura básica (lo veremos en el siguiente apartado). Spyder se da cuenta que olvidó algo, le llama la atención con una señal e incluso le da un mensaje específico del error. Básicamente hace el trabajo duro de debuggear el código por usted y lo hace automáticamente.



Vea cómo un pequeño detalle marca la diferencia. Por ahora no entraremos en detalles porque profundizaremos en el apartado que sigue, pero para que vaya entendiendo sepa que aquí estamos definiendo una función nueva que hará alguna cosa que nosotros definamos. Fíjese cómo aparece una señal de alerta al olvidar los paréntesis y los dos puntos. De hecho, al crear este ejemplo el programa automáticamente me autocompletaba los signos faltantes, pero lo forcé para plasmar la idea. En caso que no se entienda bien el motivo de la alerta, uno puede posicionar el puntero de la computadora sobre la señal para que aparezca un cuadro de texto con más detalles del inconveniente.

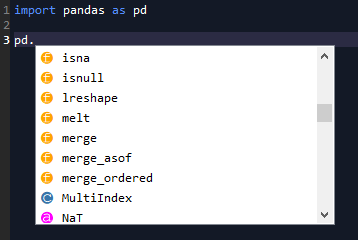
Además de analizar en tiempo real que la sintaxis esté correcta, Spyder nos ayuda a completar cada línea que nosotros escribamos a través de una herramienta llamada “completado a demanda”. Esto es básicamente un predictor que nos ofrece todas las opciones de texto que puedan seguir a lo que nosotros hayamos escrito hasta el momento y también nos ayuda a completar los métodos que se le pueda aplicar al objeto que acabamos de declarar. En ambos casos esta herramienta se acciona al presionar la tecla de tabulación (bien a la izquierda de su teclado verá la tecla con dos flechas).

La primera función de esta herramienta no es más que el predictor de escritura que usted utiliza habitualmente para escribir un mensaje de texto. Suponga que algunas líneas atrás usted declaró una variable con el nombre de “variable1” y luego una con el de “variable2”. Si luego usted escribe simplemente “vari”, por ejemplo, y presiona la tecla de tabulación, la herramienta le propondrá las dos variables que declaró antes para completar lo que está escribiendo. Esto ayuda mucho cuando tiene muchas variables con nombres parecidos o, simplemente sus variables tienen nombres largos y le da pereza escribirlos enteros.



Es la segunda función de esta herramienta la que considero la más importante. Suponga que usted está utilizando una librería nueva cuyos métodos todavía no maneja del todo bien. Recuerde que los métodos son las “funciones” que se le aplican a un objeto. En este caso, es muy común que no sepa qué método utilizar para un determinado fin o que no recuerda exactamente cómo se llamaba pero con un vistazo podría recordarlo. En este tipo de situaciones lo que hace habitualmente un programador es llamar la librería, poner un punto para indicar que está por utilizar un método y ahí apretar la tecla de tabulación para que Spyder le recomiende todos los disponibles y usted pueda elegir el adecuado.

A mí me suele ocurrir que para hacer Data Science necesito utilizar diariamente la librería “pandas” que mencionamos anteriormente. Tiene infinidad de métodos y a veces se puede olvidar cómo se llaman todos. Por esto mismo a veces llamo la librería, agrego un punto para indicarle que estoy a punto de llamar un método y activo el autocompletado a demanda para que me ayude a concluir el comando.



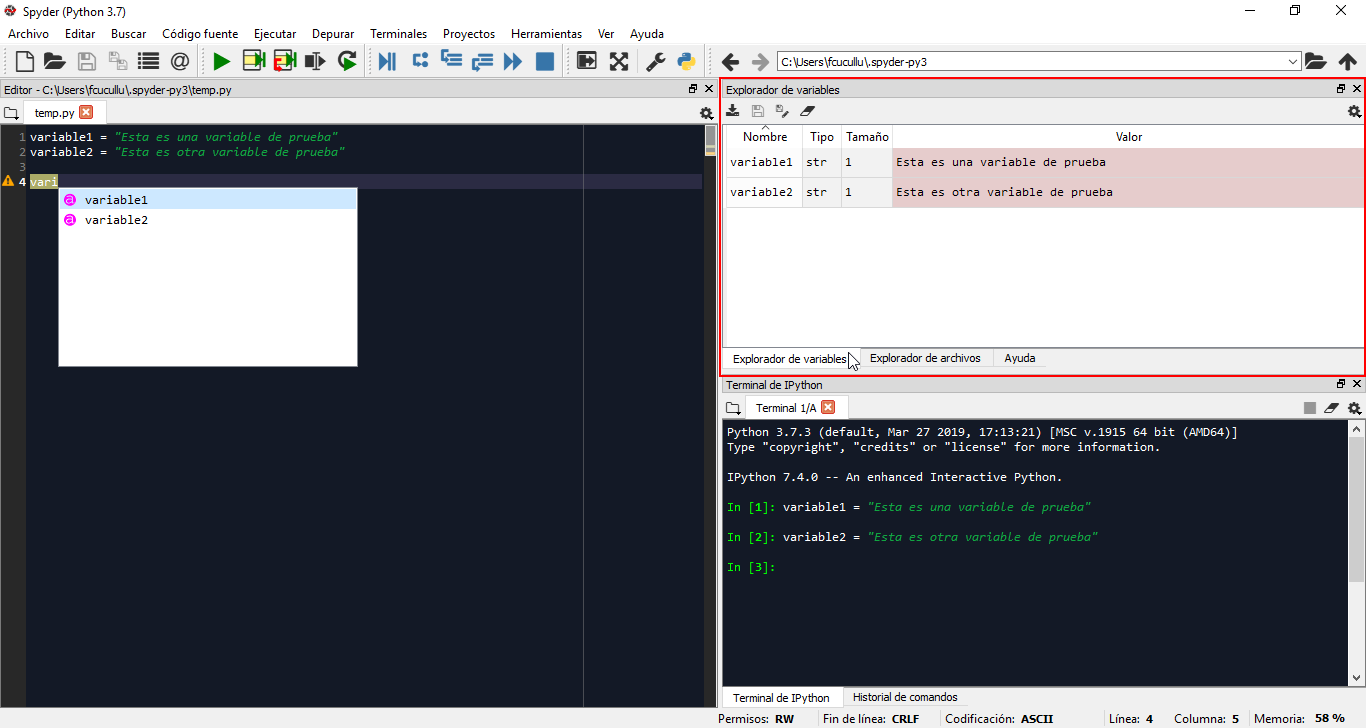
En la primer línea importo la librería “pandas” que se utiliza muy a menudo para manejar grandes volúmenes de información y armar tablas. Vean que para no tener que escribir el nombre completo de la librería, le pongo un nombre nuevo para reducir su longitud y que sea más cómodo. De hecho, llamar a esta librería simplemente como “pd” es lo que más verá en cualquier script de Python. Luego, llamo la librería a través de su nuevo nombre, le agrego un punto para indicarle al IDE que estoy por declarar un método y presiono la tecla de tabulación para que me ayude a completar la acción. Fíjese todos los métodos que me recomienda Spyder en una lista que contiene muchos más métodos que los que se ven en la imagen.

5.2.2 Explorador de variables

Otra de las ventajas de trabajar desde un IDE es que nos permite tener a la vista varias secciones que le serán como un panel de control. Si bien se podrían utilizar otras herramientas diferentes en el espacio que Spyder dedica para estos fines, recomiendo fuertemente tener el explorador de variables siempre a la vista, o al menos fácilmente accesible, por la gran utilidad que se explicará a continuación y para evitar todos los errores de tipos y formatos que vimos en las secciones anteriores.

El programa suele ubicarlo en la esquina superior derecha y para visualizarlo hay que hacer click sobre la pequeña solapa que indica su nombre ya que por defecto Spyder muestra la sección de ayuda que, con la práctica y este libro, dejará de usar. Recuerde que la disposición de todas las secciones, tamaños y colores es totalmente personalizable. En esta imagen se muestra cómo viene por defecto para que sepa ubicarlo sin problemas, pero lo más probable es que con el tiempo termine modificando esta distribución a sus propias preferencias, agregue secciones que le sean útiles y quite las que no.

Los algoritmos suelen contener mucho más que simples órdenes secuenciales. En su mayoría, los scripts que programe tendrán infinidad de comandos que impliquen la manipulación de variables. Debido a esto, tener un panel que le permita visualizar sus características ayudará en gran medida para entender qué acciones son adecuadas según su naturaleza original y si el resultado de las acciones que ejecute sobre ellas concluyen en el resultado deseado.

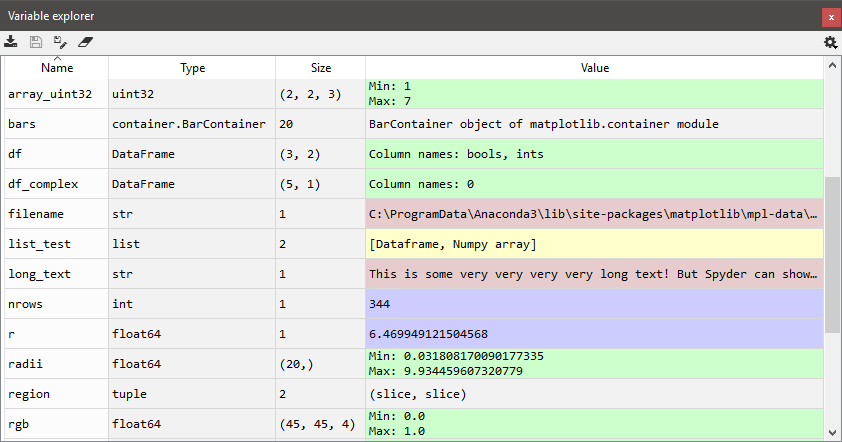


Ocurre muy seguido que uno intenta aplicarle una determinada transformación a una variable pero esta no tiene las características necesarias para soportar esta acción. En el mejor de los casos el mismo script reportará un error. Pero podría pasar que la orden no necesariamente dispare una advertencia, ejecute la acción incorrectamente y el script termine con un resultado erróneo. Tomar decisiones en base a información incorrecta es lo peor que podría pasarle a un algotrader.

El explorador de variables también facilita el trabajo de programación porque permite visualizar características clave de las variables que estamos manipulando. Nos dice qué variables ya existen. Si no están en esta sección, es porque se ha olvidado de declararlas. También permite a ver a simple vista el tipo de la variable. Al ver el explorador, se puede asegurar que las variables introducidas en una operación matemática son números, que lo que vamos a imprimir será un mensaje de texto o que la variable a la que le aplicamos índices y columnas sea realmente una tabla.

Cuando se manejan grandes volúmenes de información es muy común contar tablas de datos. El algotrading no es la excepción. Un ejemplo muy claro será cuando empiece a manipular cotizaciones de instrumentos financieros. Necesitará tener a mano, por ejemplo, tickers, precios de apertura y precios de cierre. En este ejemplo sencillo, necesitará construir una tabla de muchas filas y de al menos tres columnas. Aquí usará otra vez el explorador de variables para confirmar que lo que está haciendo es correcto, ya que una vez construida la tabla, podrá confirmar que sus dimensiones (columna “Size” del explorador) son las esperadas.

Por último, muchas veces resulta importante analizar visualmente la información que contienen las variables. Ocurre casi siempre que nuestra información cuenta con pequeños baches como nulos, datos erróneos, outliers o muchas otras cosas que pueden impedir procesar la información correctamente. Para estos casos, se puede ver la información en la columna “Value” del explorador. Ahí mismo tendrá una vista previa de la información contenida en la variable.

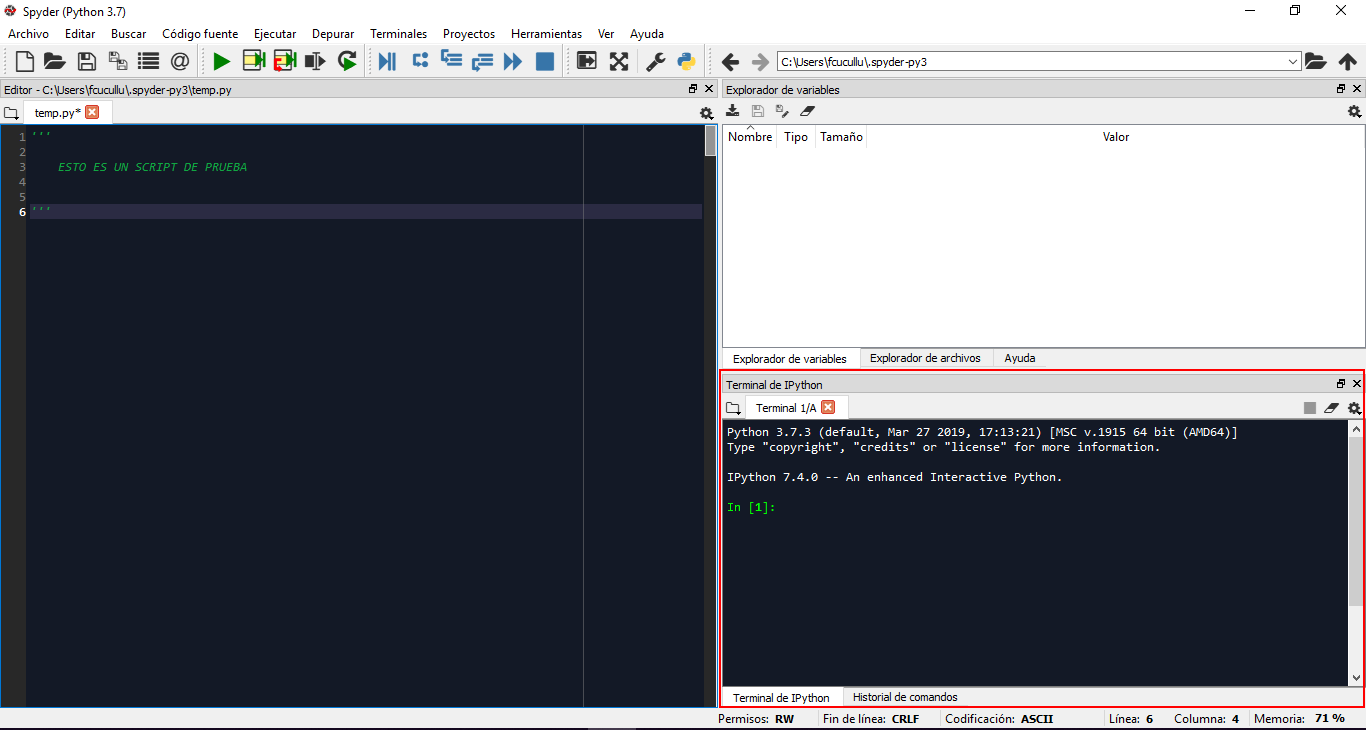


En caso que necesite más detalle, basta con darle doble click a la variable en cuestión y se le abrirá una ventana con la tabla completa. Puede reordenar la información de forma ascendente o descendente para investigar y buscar datos que le puedan complicar la operación que quiere ejecutar. Incluso, en caso de querer modificar un dato en particular podría hacerlo directamente desde allí dándole doble click a la celda que quiere modificar.

Pruebe haciendo click derecho sobre sus variables. Verá que el explorador de variables le facilitará el trabajo de hacer análisis de datos. Cuando el tipo de variable lo permite, al hacer click derecho le da al usuario la posibilidad de hacer gráficos, histogramas, editar, duplicar y muchas otras cosas. Cuando uno hace algotrading, suele ser útil hacer un análisis exploratorio de los datos con los que se cuenta para confirmar que, en primer lugar son correctos, pero también para descubrir cuál es la naturaleza de los datos con los que se está enfrentando. En muchas ocasiones podrá identificar patrones, distribuciones y particularidades que le permitan tomar buenas decisiones.

5.2.3 Terminal de IPython

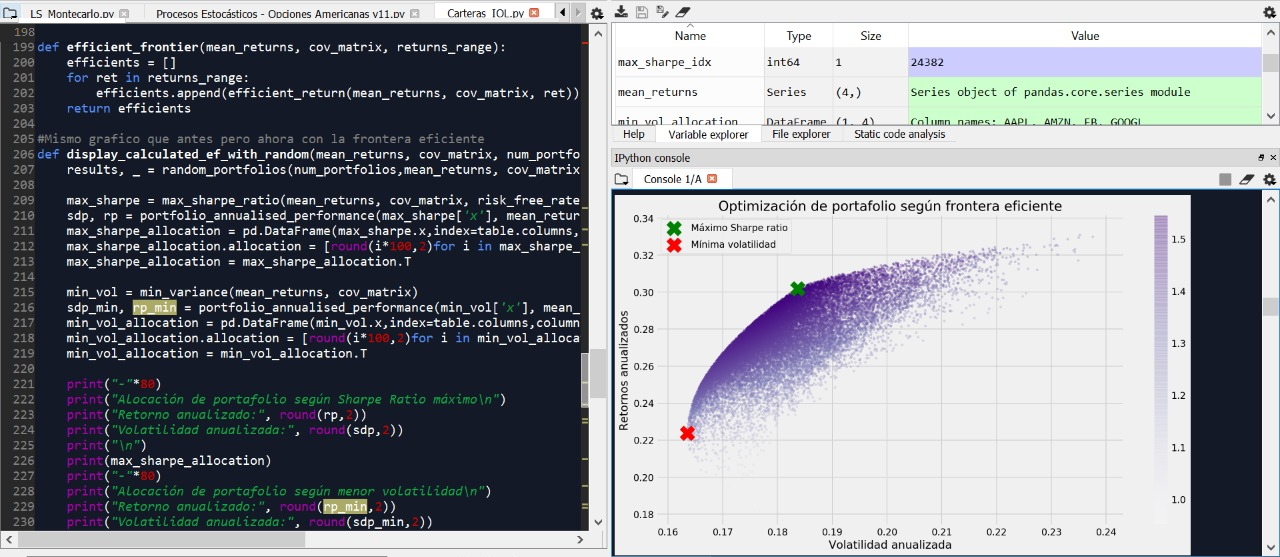
Sin duda alguna las secciones del Spyder anteriormente mencionadas son fundamentales para el día a día del programador, ya que como hemos visto, facilitan el desarrollo en gran medida. Sin embargo, es aquí, en la terminal de IPython, en donde ocurre toda la magia. Si bien esta última sección es muy similar a la consola de comandos que se le abrió cuando intentó utilizar Python por fuera del IDE, verá que presenta algunas diferencias que, al igual que las anteriores, facilitará todo el proceso de programación.



El objetivo principal de esta sección es, ni más ni menos que, ejecutar. En esta sección se correrá todo el código que nosotros programemos. Por aquí pasarán los scripts que se desarrollen en el editor de texto, pero también podrían ejecutarse aquellos comandos que escribamos directamente en esta terminal. Piense que esta sección es como una pequeña terminal de Python introducida dentro del Spyder. De todas formas, se le añaden algunas características extra para transformarla en una terminal “interactiva” (por eso la inicial “I” del nombre).

La terminal de IPython tendrá todas las bondades visuales que repasamos en la sección del editor de texto. Cuenta con la misma sintaxis coloreado según nuestras preferencias lo que nos permitirá distinguir cuando estamos ejecutando distintos tipos de información, cuenta con el autocompletado a demanda que tanto sirve al desarrollar y también nos permitirá visualizar todo tipo de gráficos para poder analizar la información correctamente.

A continuación vemos una imagen extraída del optimizador de portafolio que pude mejorar gracias al aporte de la comunidad. Esto es sólo un ejemplo de los gráficos que podríamos diseñar con esta herramienta. Lo fabuloso de saber programar es que todo está a nuestro alcance. Podríamos cambiar los colores, el formato, el estilo de gráfico, los ejes, sus escalas y más. Podremos adaptar nuestro análisis enteramente a nuestras especificaciones.



Es también en esta terminal donde podremos sacar provecho del código abierto de Python. Cuando tenga algún error, no sólo se le indicará detalladamente lo que derivó en la falla, sino que también podrá hacer click en la sección específica del código que falló y podrá sumergirse en los detalles más recónditos de la librería que está usando para investigar cuál fue la razón específica. He tenido la oportunidad de modificar el código original para adaptarlo a mi necesidad específica, pero no es lo más habitual.

Comúnmente cuando aparece un error, uno se limita simplemente a seguir el camino que aparecerá impreso en la terminal hasta llegar al punto exacto en donde falló su programa. En programación, este camino de llamadas entre una sección de la librería a otra se denomina “*tracebac*k” y en general se ordena de tal forma que el error más reciente está al final. De esta forma se puede ir siguiendo el camino que hizo todo el código hasta llegar a la parte en donde falló. Así, uno generalmente lee en el traceback que su script llamó a una librería, luego esta llamó a otra, que finalmente llamó a un método y que finalmente este método falló por una causa que está aclarada perfectamente en el mensaje del error.

Otra característica que hay que destacar de las consolas de IPython, es que permiten la ejecución de aplicaciones en paralelo. Esto permite desarrollar y ejecutar varias consolas en simultáneo, lo que permite correr varios scripts diferentes al mismo tiempo, interrumpir los procesos que se crea necesario y hasta reiniciar o terminar una consola sin afectar el normal funcionamiento de las otras. Esta es una buena forma de testear procesos menores sin entorpecer procesos primarios.

Para hacer esto, usted puede hacer click derecho en la solapa de la terminal actual y verá que aparece una opción que le permite abrir una terminal de IPython nueva o también podría utilizar el atajo Ctrl+T para hacerlo más rápido. Un detalle que notará es que el explorador de variables cambiará según cuál sea la terminal que tenga seleccionada. Esto es debido a que cada una de ellas es un entorno separado en donde puede haber variables diferentes, entonces depende de cuál sea su terminal seleccionada, el explorador le mostrará únicamente las variables declaradas en el entorno en cuestión.

## 5.3. Recursos básicos

5.3.2 Programación orientada a objetos

Antes de comenzar deberíamos recorrer al menos cinco términos que utilizaremos en las líneas que siguen, que hemos utilizado en las líneas anteriores sin entrar en demasiado detalle. También, deberíamos entender que una particularidad de Python es que sus estructuras y elementos se pueden programar pensando en objetos. Los términos que veremos son: “clase”, “método”, “función”, “variable” y “atributo”.

Una “clase”, es un conjunto de elementos, u objetos como veremos más adelante, que comparten ciertas características. Por ejemplo, uno podría tener un descapotable, un deportivo o un cupé, pero todos ellos podrían clasificarse dentro de la clase “autos”. En este caso, cada modelo de auto será una “variable” (¡que también son objetos!). Todos estos autos comparten algunos “atributos” como tener ruedas, puertas, luces o cualquier otra cosa que nosotros declaremos como característica compartida. Al mismo tiempo, todos los autos comparten acciones que pueden ejecutarse sobre ellos, es decir que comparten “métodos”. Por ejemplo, todos se pueden prender, apagar, acelerar o frenar.



En este sencillo ejemplo podemos ver cómo agrupamos las puertas y las ruedas en una clase. Esto es debido a que, si bien aquí le incluimos sólo un atributo a cada una, en realidad podríamos haber incluido variedad de cosas que todas podrían compartir (tamaño, color, materiales, accesorios, etc.). Luego creamos una clase llamada “auto” que toma como atributos las clases anteriores y hereda los atributos de estas subclases. Esta es la magia de la programación orientada a objetos. Primero definimos objetos pequeños para luego ensamblarlos en uno más grande sin necesidad de especificar este último detalladamente porque sus características ya fueron expresadas en sus partes.

La diferencia fundamental entre un “método” y una “función” es que el primero está relacionado a un objeto, mientras que la segunda no. En este ejemplo, todas las acciones que estamos declarando están relacionadas al objeto “auto” y por eso todos son métodos. Pero podría ocurrir que queramos resumir un proceso secuencial muy grande en una función para facilitar su ejecución cuando sea necesario y así evitar tener que ejecutar manualmente cada uno de sus pasos intermedios. En este caso, lo que estamos declarando será una “función”, porque no está relacionada a un objeto, sino simplemente a una secuencia de órdenes para cumplir un determinado proceso.

Volvamos a ver la imagen. Vemos que la declaración de las clases es fácilmente reconocible por que comienzan con el comando “class” y le sigue el nombre de la clase con unos paréntesis. Aquí deberíamos hacer un comentario no menor: las buenas prácticas de programación sugieren nombrar a las clases con mayúscula inicial. Esto es porque Python detecta la diferencia entre “clase” y “Clase”, por lo que resulta ordenado y prolijo separar las clases de variables normales. Ahora sabemos que cuando vemos una variable con mayúscula inicial, estamos en presencia de una clase.

Inmediatamente después de cada declaración de una clase, observe que aparece un comando “def”. Esto lo que significa es que estamos en presencia de un método o función. Aquí, al estar declarando un objeto, será un método. En este caso particular, está combinado con el nombre “\_\_init\_\_” (son dos guiones bajos de cada lado). Este método en particular lo que hace es disparar una acción tan pronto como la clase es iniciada y ejecutarla utilizando los valores de entrada, o inputs, que están entre paréntesis. Vea que también utilizamos el comando “self” (algo como “sí mismo”, en inglés) para indicar que uno de los inputs será el mismo objeto que se está iniciando. Estas funciones lo que en definitiva están haciendo es asignarle un “atributo” a la “variables”, y se hacen con el comando \_\_init\_\_ porque estas características son fundamentales para el objeto y no deben pasarse por alto. Si no tuviésemos un método \_\_init\_\_ le estaríamos dando margen a nuestro auto de no tener cantidad de puertas o no tener estilo de ruedas. Note por último que para referenciar a un atributo de una variable se hace de la siguiente forma: variable.atributo (nombre de la variable + punto + nombre del atributo).

Permítame repasar una vez más el párrafo anterior, pero ahora con lenguaje corriente. Tomemos el ejemplo de la clase “Ruedas”. Con el trabalenguas que acaba de leer, ya debería comenzar a entender la programación orientada a objetos, pero repasemos juntos cómo piensa la computadora. Si bien en las primeras tres líneas de la imagen se declara la clase, la verdadera magia arranca un poco más abajo cuando aparece “ruedas = Ruedas(‘cinco’)”. Ajústese los cinturones y arranquemos.

La computadora detecta que le ingresaron una variable nueva, “ruedas”. A esta nueva variable, ya que está acompañada por un signo “=”, se le está asignando un valor. El valor de esta variable aparece con mayúscula (¡es una clase!) y viene acompañado con un input entre comillas. El hecho que venga con comillas significa que “cinco” no es otra variable, sino que es una cadena de caracteres, o “string”, y debe asignarse literalmente como texto. La computadora cuando le quiere asignar a “ruedas” el valor de “Ruedas(‘cinco’)”, lee todo el script buscando dónde se declaró una clase con ese nombre y lo encuentra en las primeras tres líneas. Ve que tan pronto como se crea una “Ruedas”, tiene que ejecutar un método, así que lo ejecuta inmediatamente a continuación. Este método lo que termina haciendo es asignarle un atributo al objeto que se está iniciando, es decir “ruedas” (sin mayúscula, ojo). El atributo que le asignará no lo inventa la computadora, sino que toma los valores de entrada que le otorgaron: “cinco”. Y en definitiva hace que “ruedas.cantidad” sea igual a “cinco”. Hermoso, ¿no?

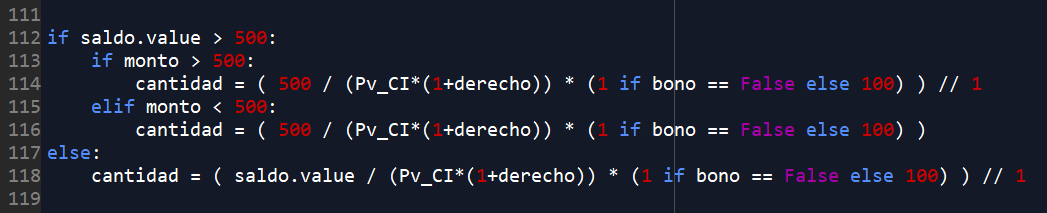
El resto de la imagen funciona más o menos parecido, pero antes de pasar al siguiente apartado, veamos el último detalle. Vea que hay una función llamada “print”. Esta es una de las que vienen por defecto con Python y no hay necesidad de declararla con un “def”. Su objetivo final es el de “printear” o imprimir en la consola lo que se introduzca entre paréntesis. Muchas veces se usa para verificar que lo que se está haciendo es correcto y hacer que nuestro script muestre los atributos de las variables, por ejemplo. En este caso le estamos pidiendo que imprima una oración que tiene unos espacios en blanco identificados con llaves. Analice cómo es la estructura de esta función y vea cómo se utilizan las llaves, los paréntesis y las comillas. Es esta línea la responsable que al final de todo el script se imprima la confirmación de que todo el código se cumplió satisfactoriamente y con sentido.

5.3.2 Estructuras condicionales

Existen situaciones en donde debemos abrir la secuencialidad de nuestro algoritmo en dos o más caminos diferentes para adaptarlo a la naturaleza de nuestro problema. Aquí es donde utilizaremos las estructuras condicionales para determinar si una condición se cumple o no y, en base a eso, adaptar los siguientes pasos de nuestro algoritmo según cuál sea el caso que se está enfrentando. Hay que recordar que, particularmente en el algotrading, es fundamental expresar la condición correctamente y ubicar la bifurcación en el lugar indicado. Nuestro algoritmo estará tratando de interpretar un fenómeno financiero complejo a través de la abstracción de nuestro código. Es todo un arte expresar la condición que bifurcará el accionar de un algoritmo. El programador exprimirá aquí al máximo su imaginación y su lógica para abstraer un fenómeno financiero en una simple expresión de código.

Las sentencias condicionales están presentes en todos los algoritmos que vayan más allá de la simpleza de la secuencialidad de las instrucciones. Le brindan una determinada inteligencia a los scripts que programemos debido a la capacidad de identificar diferentes escenarios y las instrucciones claras que deben seguirse dependiendo del escenario con el que nos enfrentemos. Particularmente en los algoritmos de trading algorítmico, este tipo de sentencias es uno de los recursos más importante. A no ser que uno quiera programar un robot que sólo ejecute, uno quiere que al menos interprete información y ejecute la operación correspondiente. Como se dijo unas cuantas líneas atrás, uno no puede comprar, comprar y sólo comprar. En la bolsa a veces hay que comprar, otras vender y a veces simplemente esperar afuera sin posición.

Veamos un ejemplo para analizar los componentes de estas estructuras y cuáles son las características particularmente en Python. Seguro que ya está cansado de los ejemplos fáciles que estuvimos viendo hasta ahora, así que analicemos un caso un poco más difícil. A continuación veremos un pequeño extracto de una estructura condicional de un algoritmo de trading real mucho más extenso. De hecho, la mayoría de las estructuras condicionales tienen una longitud que hacen inviable plasmarlas en una sola página. Limitémonos a entender la estructura en general y no se pierda en los pequeños detalles. Es sólo un ejemplo ilustrativo.



Una sentencia condicional está conformada habitualmente por tres partes. La primer parte es la sentencia de la condición y, como dijimos, es crucial para identificar correctamente el punto de inflexión que se necesita para dividir la naturaleza del problema en, al menos, las dos siguientes partes. En la imagen, la expresión de la condición está formada por el comando “if” seguido de una declaración que podría ser únicamente verdadera o falsa y se finaliza con dos puntos. Aquí, le estamos preguntando a la computadora si el atributo “value” de la variable “saldo” es mayor que 500. Note que las condiciones suelen utilizar los signos “<”, “>” o “==”. Se usan dos iguales porque significan “igual a”. Si se usara uno sólo se estaría asignando un valor a una variable, y en este momento eso no está permitido. Recuerde que la computadora espera un valor verdadero o falso.

La segunda parte, es la porción de código que se ejecutará cuando la condición resulte verdadera. En Python, esta parte está claramente identificada por la indentación. Recuerde que este lenguaje utiliza únicamente este recurso para identificar subprocesos dentro de uno mayor. Entonces, luego de los dos puntos de la condición, se debe introducir en el siguiente renglón una indentación para expresarle a la computadora que comienza el bloque de código que deberá ejecutarse cuando la sentencia condicional resulte verdadera. No se pierda en los detalles de la imagen porque, como usted lo pidió, este ejemplo es un poco complejo para entenderlo a la primera vez. Sólo recuerde que esta segunda parte son las siguientes cuatro líneas luego de la condición. Al finalizar con la siguiente parte, volveremos hacia atrás para despejar sus dudas sobre estos detalles, así que no se preocupe.

La tercera y última parte, es la porción de código que se ejecutará cuando la condición sea falsa pero no siempre hay necesidad de declararla. ¿Cómo? Si, puede omitirse. Si usted no tiene la necesidad de indicarle a la computadora qué hacer cuando la condición es falsa y podría directamente omitir esta parte del código, pero debería asegurarse que su estructura condicional refleje totalmente el fenómeno en cuestión y no deje lugar a ambigüedades. Entonces, luego del bloque de código que se ejecuta cuando la condición es positiva, el programador debería retroceder una indentación y continuar. Si no desea declarar la tercera parte, debería continuar normalmente programando la secuencialidad de su algoritmo. Pero si desea hacerlo, debe declarar explícitamente alguno de los siguientes dos comandos: “else” o “elif”.

El comando “else” se utiliza para captar todos aquellos casos en los que la condición sea falsa, sin incurrir en ninguna otra prueba lógica. Directamente capta todas las situaciones que escapan de la condición y ejecuta. Para utilizarla, vea en la imagen que se expresa y luego se introducen dos puntos. Las líneas siguientes deben tener una indentación para indicarle a la computadora que está dentro de esta tercera parte de la expresión condicional. Por otro lado, el comando “elif” se utiliza cuando se quiere volver a filtrar a través de otra expresión condicional. Si se utiliza este recurso, luego de declararlo debe incluirse una expresión verdadera o falsa y finalizarlo con dos puntos. Las líneas siguientes deben tener indentación, como siempre. Algunos programadores no usan este recurso y lo reemplazan por una estructura condicional sin la última parte.

Repasemos brevemente la imagen antes de dar el último remate de complicación al asunto. Se tiene una variable “saldo” que tiene un atributo “value” y se quiere filtrar las diferentes situaciones según el valor de este último. Además de esa variable se tiene otras dos que no habíamos resaltado antes: “monto” y “cantidad”. La computadora reconoce una estructura condicional y piensa como sigue. “Si el valor de ‘saldo.value’ es mayor a 500, entonces analizo el valor de ‘monto’. Si ‘monto’ es mayor a 500 le asigno a ‘cantidad’ un determinado valor, pero si ‘monto’ es menor a 500 le asigno a ‘cantidad’ otro determinado valor. Pero si, y volviendo al principio, el valor de ‘saldo.value’ es menor a 500, le asigno a ‘cantidad’ otro valor diferente”. Rebuscado, ¿no? Pero de esta forma el programador captó la naturaleza de su problema financiero.

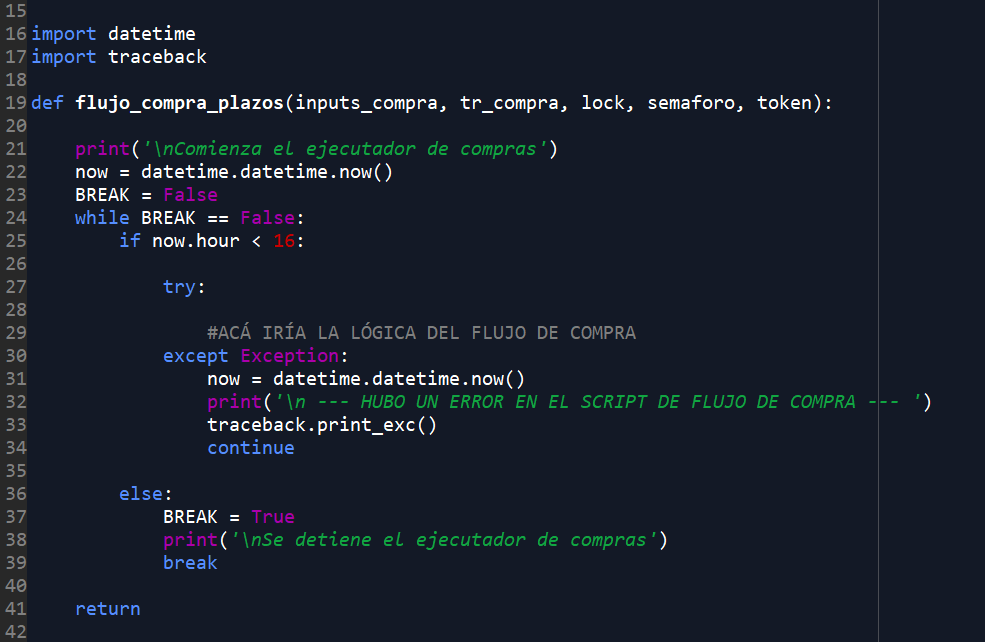
Ahora rematemos como prometimos atrás. Vea que el valor que se le asigna a “cantidad” en cada caso tiene dentro otra especie de estructura condicional extraña. Esto no es ni más ni menos que un tipo especial de estructura en donde las tres partes están resumidas en una sola línea. Esto comúnmente se usa para minimizar la cantidad de líneas que se necesitan para el caso. Imaginen que dentro de la estructura condicional anidada que vimos recién tengamos que incluir otra más. La lógica se hubiese vuelto imposible de seguir. Entonces, cómo es una expresión relativamente corta, el programador la introdujo en una sola línea. En este caso especial, si bien funciona exactamente igual, se expresa primero el valor en caso que la condición sea verdadera, luego la condición y por último el valor en caso que la condición sea falsa. Para leerlo, tengamos en cuenta que entra en juego una cuarta variable llamada “bono” y que parece tener valores verdaderos o falsos únicamente. La computadora ve esta expresión y lee “usar 1 si bono es falso, en otro caso usar 100”. Parece mucho más fácil de recordar este tipo de expresiones, lo único que no debe olvidar es que el orden de las tres partes cambia un poco.

5.3.2 Estructuras de iteración

Como habíamos expresado anteriormente, este recurso de la programación estructurada nos facilita el trabajo de repetir una tarea varias veces sin tener que codear la secuencia entera. Sólo con declarar cual es la subrutina a repetir y definir las condiciones de su repetición, podemos simplificar nuestros algoritmos en gran medida conservando la parsimonia del script. Cuando programamos es muy importante la parsimonia. Si podemos ir al grano sin mucho rodeo, mejor. No es para nada fácil retomar un trabajo unos días después de haber programado algo que da vueltas y vueltas sobre un asunto que podría haberse abordado de una forma más directa. Cuanto más directo sea el algoritmo, más legible será para entenderlo luego. Pero sobre todas las cosas, cuanto más parsimonioso sea, más veloz será. Esto último es fundamental para la operatoria computarizada en la bolsa.

Recuerde de los módulos anteriores que podemos declarar nuestras subrutinas de iteración con básicamente dos recursos: los “for” y los “while”. Si bien pueden ser intercambiables, en general es más fácil expresar la naturaleza del problema con alguno en particular según cuál sea la situación. Mientras que el “for” repetirá la subrutina una determinada cantidad de veces, el “while” repetirá la subrutina indefinidamente hasta que se cumpla la condición de corte.

Una situación habitual en el mundo del algotrading es la necesidad de introducir una condición que detecta qué hora es, para detener los robots cuando el mercado cierra. Parece innecesario, pero si no se introducen controles como estos, puede ocurrir que nuestros robots sigan escaneando el mercado cuando el mercado está cerrado. Podría ocurrir que una persona introduzca una orden para la rueda siguiente con un precio significativamente diferente al último operado y que provoque que nuestro algoritmo accione su operar. Si esto ocurriese, nuestro robot lanzaría una orden que podría estar firme hasta la rueda siguiente y termine ejecutándose en una situación desfavorable. Por esto mismo, veamos cómo un algotrader solucionaría este problema con un “while”:



Tenemos muchas cosas nuevas para analizar en esta imagen. Empecemos con las dos librerías que el algotrader descarga al principio de su script. Recuerde que para instalarlas, en caso que no las tenga, debe ingresar a la línea de comando (Ctrl+R) e introducir “pip install datetime” por ejemplo. La librería datetime sirve para manipular datos con el formato de fecha y hora. El programador la usa para crear una variable llamada “now” que almacena la fecha y la hora exacta en las que se ejecuta la línea 22 o la línea 31. Por otro lado, la librería traceback se utiliza para pedirle a nuestro algoritmo que, con el método “print\_exc()” de la línea 33, nos imprima una descripción detallada de cuál fue el motivo de algún error que haya ocurrido. Otro recurso nuevo es la estructura try-except. Ocurre que, cuando Python se topa con un error, el script entero se detiene para que el programador lo solucione. En este caso, el programador lo que hace es decirle exactamente a Python qué quiere que haga cuando se tope con un error. Básicamente está manipulando a su gusto cómo reacciona el software ante un imprevisto, ya que lo que hace por defecto no le satisface.

Habiendo visto estos recursos nuevos, entendamos qué está haciendo el algotrader con esta estructura de iteración. A primera vista podemos ver que en la indentación principal está declarando el comando “def”, lo que indica que está estructurando una nueva función. No está afectando a ningún objeto, sino que aglutinará una serie de instrucciones secuenciales. Vea que luego del nombre de la función, se declaran entre paréntesis los inputs que requerirá para su ejecución. Luego de cerrar el paréntesis de los inputs, inmediatamente le siguen dos puntos y comienza un bloque indentado de código que conformará el contenido de este nuevo proceso.

Lo primero que hace esta nueva función es imprimir un mensaje que seguramente mostraría en su panel de control y le indicaría que el proceso en cuestión ha sido puesto en marcha. Cuando se manipulan varios procesos al mismo tiempo, es posible que alguno de ellos falle o encuentra una complicación, por lo que siempre es útil pedirle al algoritmo que nos dé una señal cuando haya pasado por una zona clave de su ejecución.

Inmediatamente después de avisarle al operador que se ha comenzado satisfactoriamente con el proceso, se declaran dos variables que tendrán un rol clave. La primera es la variable “now” de la que se habló anteriormente, y la segunda es la variable “BREAK” que tendrá únicamente el valor “True” o “False” (este tipo de variables se llama “boolean”). El programador le asigna en un principio un valor falso.

Ahora comienza la parte interesante. A partir de aquí comienza nuestra rutina de iteración que se ejecutará indefinidamente hasta que la condición de corte se cumpla. Esta condición hará que la ejecución continúe siempre y cuando BREAK tenga un valor falso. En otras palabras, la rutina se ejecutara mientras (while, en inglés) BREAK sea igual a “False”. Observe con atención que al finalizar con la condición de corte, se deben incluir dos puntos y aplicar una nueva indentación para avisarle a Python que el contenido de la rutina será el que sigue.

Una vez dentro del while, el programador incluye una estructura condicional como la que vimos anteriormente. El criterio de bifurcación será en base a cuál sea el atributo “hour” de la variable “now”. Note cómo con un poco de inglés podría entender perfectamente qué está ocurriendo aquí. Recuerde que la variable “now” contenía la fecha y hora. Entonces lo que se le está pidiendo es el valor únicamente de la hora, omitiendo el día, el mes, el año, los minutos y los segundos. La razón es simple: el mercado en donde el programador está operando cierra a las cuatro de la tarde.

Comencemos por la sección negativa de la estructura condicional. Vea que cuando la hora de “now” no es menor a 16, el algoritmo salta hacia la línea 36 para asignarle el valor “True” a “BREAK”. Acto seguido, se imprime un mensaje para indicarle al operador que se está deteniendo el proceso. Esto le servirá para poder confirmar visualmente que los procesos que quiere que se detengan realmente lo han hecho. Y por último, el programador incluye un nuevo comando: “break”. Este comando es de uso exclusivo de las rutinas de iteración y lo que hace es detener forzosamente la repetición. Este comando se utiliza mucho cuando se desea detener la rutina antes que termine una iteración completa. En este caso se incluye a modo ilustrativo, ya que al cambiarle el valor a la variable “BREAK” automáticamente el proceso se iba a detener ya que la condición de corte se dejó de cumplir.

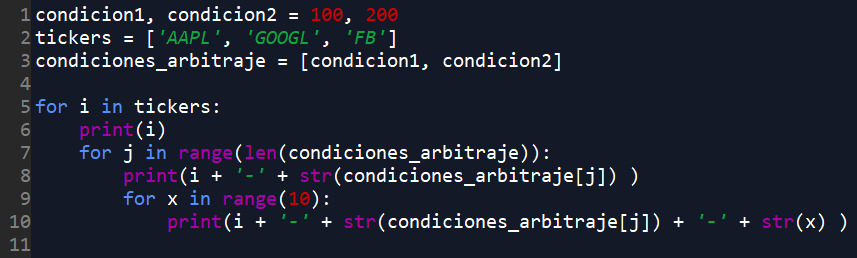
Pasemos ahora a la sección positiva de la estructura condicional. En caso que la hora del proceso sea inferior que las cuatro de la tarde, la secuencia atravesaría una estructura try-except. En este ejemplo, vea que la primer parte está resumida en un comentario que indica dónde estaría la lógica de la ejecución, pero por fines didácticos no valía la pena incluirla, por su complicación y, sobre todo, por su extensión. En caso que la ejecución de la operación tenga algún problema, la rutina de iteración y el script en general no se detendrán. Esto es gracias a la estructura try-except que captura todos los errores. En este caso, el programador le pide que en caso que ocurra algún error, el algoritmo le imprima un mensaje en su panel de control indicándole lo ocurrido. Pero no sólo le imprimirá el mensaje que se ve escrito en la imagen, sino que gracias al método print\_exc() de la librería traceback, también se imprimirá un informe detallado de lo ocurrido para indicarle al operador si la operatoria puede continuar o si es necesario detener el algoritmo anticipadamente.

Observe que luego del informe, el programador incluyo otro nuevo comando: “continue”. Este, junto con “break” son de uso exclusivo de las rutinas de iteración. Pero a diferencia del anterior, este le pide al algoritmo que continúe con siguiente iteración de la rutina y que no se detenga con la iteración problemática.

Por último, y sumamente importante, vea al final de la imagen el comando “return”. Este comando se utiliza estrictamente cuando se están desarrollando métodos o funciones y su objetivo es avisarle al sistema que se está finalizando con el proceso y se desea que retorne. Ocurre muchas veces en las que se desea que la función devuelva un valor para luego ser asignado a una variable, por lo que el comando estará acompañado por el valor a retornar. En este ejemplo en particular, no es necesario acompañarlo con nada porque el programador no desea que la función retorne algún valor, dada la naturaleza de su problema. Pero, aunque no es necesario, es una buena práctica introducirlo de todas formas. Verá en algunas imágenes que siguen que el comando está acompañado, luego de un espacio, de una porción adicional de código. Sepa que en esa situación, la función retornará ese valor hacia afuera del bloque de indentación. Si no hay un return, todo lo que se procese dentro de la función queda dentro de la función. Por ello, si se desea utilizar algún resultado de la misma, es necesario incluir un return y acompañarlo del valor deseado.

Sería una situación completamente diferente si en vez de tener una condición de corte bien definida, tengamos la necesidad de recorrer todos los elementos de un conjunto, por ejemplo. En ese caso, sería mucho más fácil recurrir a un “for”. Este tipo de estructuras, tienen la ventaja de tener una condición de corte que no está explícitamente declarada, sino que dependerá de la composición del conjunto que queramos recorrer. En algunos casos, podría ocurrir que tengamos un objeto con componentes variables. A veces tiene más, otras veces tiene menos. No sería lógico utilizar “while” en este caso, ya que si queremos recorrer cada uno de sus elementos, deberíamos contemplar todos los posibles casos del problema, y como su composición es variable, declarar una sentencia explícita nos podría dirigir hacia una descripción incorrecta de su naturaleza.

Veamos a continuación un ejemplo en el que, por fines didácticos, se ensambla una estructura de iteración anidada, en el que varias subrutinas están embebidas en otras. Aunque parezca forzado, en la práctica se topará regularmente con este tipo de situaciones. Recorramos línea por línea para incorporar algunos elementos nuevos y entender cuál será el resultado de este pequeño pero intrincado script.



En la imagen puede observarse que, antes de comenzar con las rutinas de iteración, el programador declara en las primeras tres líneas algunos elementos que utilizará posteriormente. En la línea número uno se declara dos variables al mismo tiempo, una al lado de la otra. Si bien en este caso el ejemplo se limita a asignarle un simple número, usted en la práctica podría introducir en este caso varias estrategias de trading. En última instancia, verá que gracias a las rutinas de iteración, podrá pedirle a su robot que investigue si existe oportunidad de hacer dinero con varias estrategias diferentes que irá probando secuencialmente.

En la segunda y tercera línea el programador vuelve a declarar variables nuevas, pero en estos casos lo hace de una forma que no habíamos visto antes. Fíjese que al declarar “tickers” y “condiciones\_arbitraje”, lo hace agrupando varios elementos entre corchetes. En Python existen básicamente tres formas de agrupar variables: listas, tuplas y diccionarios. En este caso, el programador agrupa varios elementos utilizando una lista, pero podría haberlo hecho con paréntesis o con llaves para estructurarlos en una tupla o un diccionario, respectivamente. Cada una de estas herramientas tiene sus características diferenciales y las irá conociendo a medida que avance con su experiencia de programación. Por ahora sepa que cuando hay varios objetos agrupados entre corchetes y separados entre comas, quiere decir que se los está introduciendo en una lista. Detalle importante: observe y recuerde que la variable “tickers” está conformada por strings (valores de texto), mientras que “condiciones\_arbitraje” está conforma por valores numéricos.

A partir de la línea cinco comienzan las rutinas de iteración. El programador comienza declarando un primer “for”. En este, se le pide a Python que, por cada elemento (al que se lo nombra “i”) de la variable “tickers” se repita lo que continua. Se introducen dos puntos y se aplica una indentación para indicar que comienza la rutina. En primer lugar, la rutina debe imprimir el valor “i” en cada iteración para luego introducirse a una segunda rutina de iteración. La lógica de este segundo “for” es igual que la anterior: por cada elemento de un objeto, ejecute la subrutina. Pero vea que el objeto que se está recorriendo en esta segunda rutina tiene una forma diferente. Pasemos a ver entonces, qué significan estos nuevos dos comandos.

Vayamos de adentro hacia afuera. El primer comando que analizará Python es “len” y no es ni más ni menos que la longitud del objeto al que se le está aplicando. En este caso, la longitud de la variable será igual a 2, ya que esta contiene los dos elementos que se le asigno en la tercera línea. El segundo comando que analizará Python es “range”, el cual tiene como objetivo expresar un rango de números. Aquí, el sistema entenderá que se quiere recorrer un rango que va a ir desde el 0 hasta el 2. Si bien parece muy rebuscado o innecesario, existen muchas ocasiones en las que no se quiere tener el elemento en sí del objeto que se recorre, como se hizo en la rutina anterior. En este caso, lo que el sistema tomará en cada iteración será su índice, su ubicación dentro del objeto. Un ejemplo claro es cuando se tiene una tabla de más de una columna y no se desea tener el valor de la primera, sino de alguna otra. Entonces se le pide a Python que reconozca qué tan larga es la tabla con el comando “len”, para luego recorrerla a través de sus índices y no específicamente los elementos que la componen.

Recapitulando, la segunda rutina entonces recorrerá los índices de la variable que se recorre y los llamará “j”. Como la longitud es de 2 elementos, se generará un rango que vaya del 0 al 2. Por esto mismo, los valores que tomará “j” serán: 0 y 1. Lógicamente, se preguntará dónde quedó el 2, y esta es otra de las razones por las que se expuso un ejemplo como este. Dos cosas muy muy importantes a tener en cuenta es que, en primer lugar Python comienza a contar desde cero, y en segundo lugar es que los rangos incluyen el límite inferior pero excluyen el superior. De esta forma, cuando usted tenga una tabla, el primerísimo elemento no tendrá el índice número uno, sino que tendrá el número cero. Raro, ¿no? Debido a esto, los rango excluyen el límite superior ya que, si fuesen a buscar el elemento con índice 2 aparecería un error porque el objeto no tiene tres elementos (el tercer elemento llevaría el índice numero 2).

Habiendo entendido lo anterior, observe cómo en cada iteración de la segunda rutina se pide en la línea 8 que el algoritmo imprima algo que está conformado por “i” más el elemento ubicado en el índice “j” de la variable “condiciones\_arbitraje”. Vayamos paso a paso. El elemento “i” no presenta complicaciones de entendimiento, será alguno de los componentes de la variable “tickers” según cuál sea la iteración de la primer rutina: AAPL, GOOGL o FB. Ahora, en la segunda parte las cosas se pueden complicar un poco más. Recuerde que “j” podría ser 0 o 1, ya que la “len” era igual a 2 y “range” excluye su límite superior (el número 2). Observe entonces que en esta línea, se expresa “condiciones\_arbitraje[j]”, lo que quiere decir que se pide el elemento de la lista ubicado en el índice “j”.

Cuando vamos a imprimir estos dos elementos, nos damos cuenta que el primero es un valor de texto, pero el segundo es un número. Como hay una operación de sumatoria entre ambos, no puede ocurrir que tengan un formato diferente porque Python es un lenguaje de tipado fuerte, ¿lo recuerda? Por esto, con el nuevo método “str()” lo convertimos en string y todos quedan con formato de texto, listos para imprimirse en conjunto.

Por último, el algoritmo ingresa a una tercera subrutina que navegará por un rango constituido por números del 0 al 10 (pero recuerde que se excluye el límite superior, entonces llegará al 9) e imprimirá un texto que incluirá la variable de iteración de cada una de las tres rutinas: “i”, “j” y “x”. Observe que “x” también tiene un valor numérico, por lo que es necesario aplicarle el método “str()” para transformarla en texto. En última instancia, esta combinación de rutinas anidadas imprimirá un mensaje con el formato “i-j-x” para cada iteración.

Imagino que hasta aquí parece todo un trabalenguas, pero créame que al implementarlo entenderá todo. Esto es simplemente una exacerbación de las complicaciones con las que usted se topará aglutinadas en unas pocas líneas. Pero en la práctica, no sólo manejará los recursos a la perfección, sino que no tendrá que enfrentarlas todas al mismo tiempo y utilizará parte de estos métodos de navegación según lo que usted necesite en cada caso. Vea que las bondades de incluir prints en cada paso del algoritmo le permite verificar visualmente a través de la terminal de IPython qué está ocurriendo en realidad. Trate de implementar este ejemplo en su computadora y vea que le devuelve este ejercicio. Estoy seguro que cuando vea el resultado con sus propios ojos entenderá mejor todo el rompecabezas que acabamos de recorrer. Recuerde que cada vez que tenga dudas con alguna variable o si su algoritmo está ensamblado correctamente, puede ayudarse a sí mismo imprimiendo lo que fuera necesario en las situaciones que vea complicadas. De esta forma, confirmará visualmente que el valor que tu robot está procesando es realmente el que usted quiere.

## 5.4. Hacia nuestro primer algoritmo

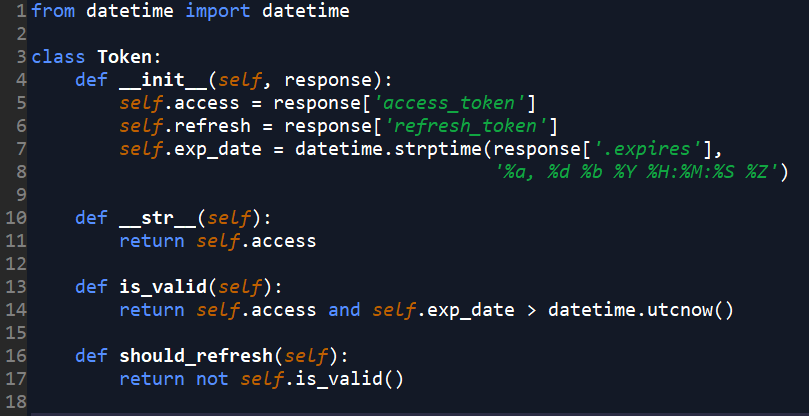
A partir de aquí nos adentraremos en el maravilloso desafío de interactuar con su cuenta comitente utilizando Python. Para ello, utilizaremos las APIs de InvertirOnline en todos los ejemplos y se le otorgará acceso exclusivo a los desarrollos reales que los operadores cuantitativos han utilizado para operar en la bolsa argentina. En este apartado, se recorrerán todas herramientas necesarias para que usted, al terminar de leer este libro, pueda aventurarse a hacer sus propios desarrollos y modificar los existentes para adaptarlos a sus preferencias personales. En caso que necesite ayuda adicional, sepa que toda la información necesaria para utilizar las diversas funcionalidades de las APIs estará en: [https://api.invertironline.com](https://api.invertironline.com/).

5.4.1 Recursos preliminares

Demos nuestro primer paso para bajar todo este nuevo conocimiento a tierra. Pero antes, pida acceso a las APIs de InvertirOnline para poder seguir todos los ejemplos que mostraremos a continuación. Con este servicio podrá darle vida a su algoritmo utilizando los servicios y la capacidad de cómputo de IOL. También puede usar otras APIs y otros brokers, pero en ese caso tendrá que utilizar los ejemplos que siguen sólo a modo ilustrativo y adentrarse a investigar las particularidades de los servicios de otras compañías.

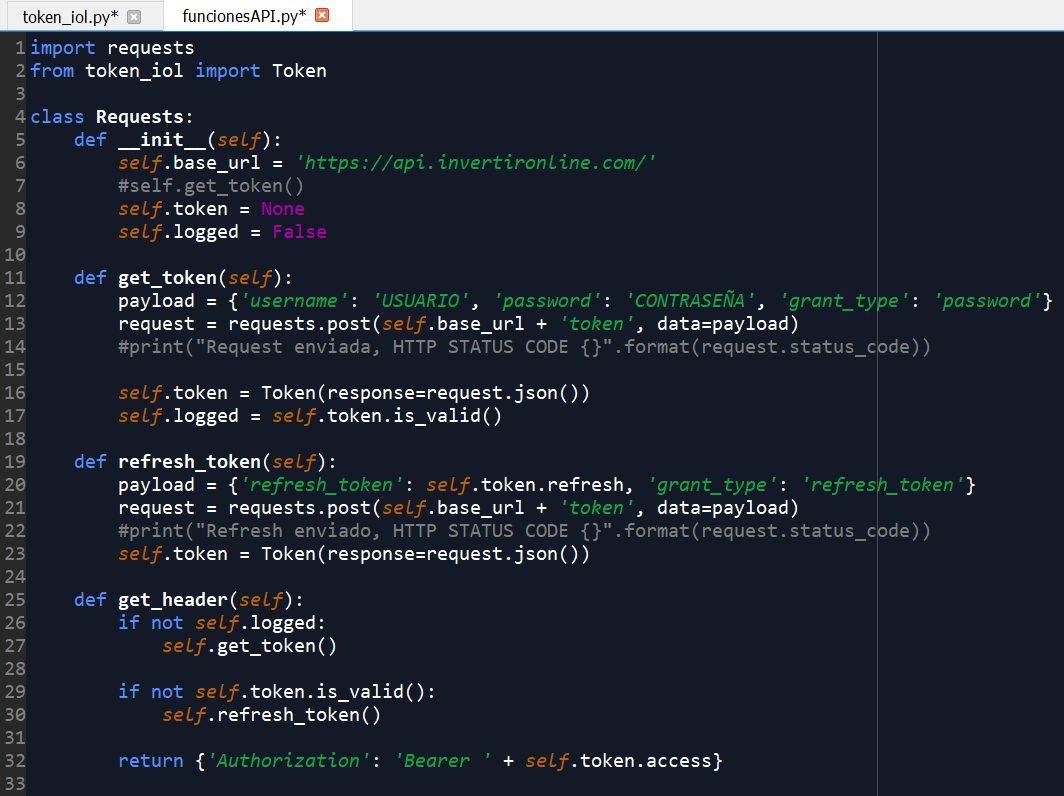
Una vez que tenga acceso, creemos juntos lo primero que vamos a necesitar para operar en bolsa: el *token*, un método de autenticación. Los tokens son simples strings que se envían al servidor para autentificar la identidad del usuario. Exactamente igual que la tarjeta de coordenada que utiliza con su banco para hacer una transferencia. En todas las herramientas, al menos las confiables, se le pedirá que ingrese este tipo de autentificación para confirmar cualquier tipo de operación antes de siquiera poder mover un centavo. Puede parecer un paso engorroso, pero si le piden esto, quiere decir que su dinero está en buenas manos.

Al menos en sitio de InvertirOnline, existen dos tipos de tokens: *bearer token* y *refresh tokens*. El bearer token se debe presentar en cada request (o pedido a la API) para poder acceder a la información solicitada. Este token es válido por 15 minutos desde que se lo solicita por primera vez y luego debe ser renovado. Esto es lógicamente por cuestiones de seguridad. Sería altamente inseguro que el token tenga una vida útil indefinida, y más sabiendo que una vez que tenga programado lo que le mostraremos a continuación, en realidad puede renovarlo las veces que usted desee. Para renovarlo, se utiliza simplemente el refresh token. Si quiere profundizar en este tema en particular, le recomendamos leer nuestro tutorial de autenticación que ofrecemos en<https://api.invertironline.com/Help/Autenticacion>. De todas formas con el código que le facilitamos a continuación lo tendrá todo listo. El link es sólo para los más curiosos.



En la imagen de arriba se ve todo el desarrollo de autentificación que es necesario para operar. Fíjese que creamos la clase llamada “Token” que tiene cuatro métodos definidos por el comando “def”. El primer método de la clase es muy importante, ya que es el que declara sus tres atributos en base al input “response” que veremos más tarde. Los tres atributos son: el token de acceso (o bearer), el refresh token y la fecha de expiración. Vea cómo se le indica al algoritmo con qué formato viene la fecha en la octava línea. Siempre las fechas le darán dolor de cabeza, es un mal que podrá compartir con todos los programadores que conozca. En este caso se le informa al robot que recibirá un texto que contenderá consecutivamente día de la semana, día del mes, mes, año, hora, minuto, segundo y zona horaria. Cuando trabaje con fechas y horarios, indudablemente tendrá que ayudarse de su buscador de internet porque, insisto, nadie es experto en este tema de las fechas. Siempre trae complicaciones y por eso en este libro se lo facilitamos.

Si bien los otros métodos también son importantes, no vale la pena entrar en tanto detalle. De igual forma, para la ansiedad de lo más curiosos, sepa que el segundo es para que al imprimir el token aparezca el access token. Esto es debido a que se le estaría pidiendo a Python que imprima una clase entera y no daría un buen resultado, entonces se declara este método para avisarle qué es lo que se quiere en esos casos. Los otros dos métodos son para verificar si los tokens que se tienen son válidos o si es necesaria una actualización. Demos un paso más.



Ya teníamos una clase que agrupaba los tokens para la autentificación. A ese archivo lo nombré “token\_iol.py” y lo guardé en la misma carpeta donde seguiremos trabajando. Ahora crearemos otro archivo llamado “funcionesAPI.py” en la misma carpeta y, como muestra el ejemplo, tendrá una nueva clase que englobará todas nuestras requests a la API. Vea cómo se importa la clase Token del archivo que creamos antes y también se importa una nueva librería llamada “requests” que nos permitirá a interactuar con internet. No confunda la clase Requests que crearemos a continuación con esta librería. Recuerde que las mayúsculas en Python importan mucho. Si nunca utilizó esta librería, como es de esperar, acuérdese de instalarla con “pip install requests” desde la línea de comando (Ctrl+R y luego “cmd”).

Observe que el primer método declara los atributos de esta nueva clase. Declara la URL que tendrán como base todas las requests, al que sólo habrá que agregarle el *endpoint* correspondiente para cada consulta (comprar, vender, consulta de saldos, etc). Es decir que cuando queramos operar, todas nuestras operaciones irán a parar a una página de internet que compartirá la primera parte que estamos declarando, pero lo que le sigue diferirá según cuál es la operación que estemos queriendo ejecutar. Aquí es muy muy importante aclarar que esta URL base direcciona todas nuestras operaciones a su cuenta real. Si, su cuenta real con dinero de verdad. Recomendamos enfáticamente no utilizar esta URL base hasta estar completamente seguros de lo que se está haciendo, hasta entonces, utilizar el servicio se testeo que ofrece InvertirOnline. Además de la URL base, se le asigna a este nuevo objeto el token y el estatus de logueo. Inicialmente el token estará vacío y el logueo será falso por cuestiones de seguridad. Pero de igual forma, se le asignan al objeto estos atributos desde el primer momento porque son fundamentales luego.

El segundo método que se declara en la imagen será el que pida el token que se usará en cada operación. Recorramos línea por línea para entenderlo. Observe cómo en la primer línea se declara la variable “payload”. Esta variable será la que contendrá la información útil a la hora de ejecutar el request. Fíjese cómo está estructurada, vea las llaves. ¿Recuerda qué indica tener información entre llaves? Exactamente, es un diccionario. Los diccionarios se declaran de una forma especial. Al igual que los diccionarios que ya conoce, estos necesitan un valor “key” seguido de dos puntos y un valor “value”. Es decir que necesitan estar estructurados como en forma de definición, que una clave indique a través de los dos puntos cuál es su definición. En este caso vea que se le están pidiendo sus credenciales, por lo que recuerde que en donde están las mayúsculas usted debe ingresar su usuario y contraseña reales.

Luego de ingresar sus credenciales, viene la línea más importante del método: realizar la request propiamente dicha. Se crea una variable llamada “request” que almacenará lo que responda el comunicado de la API. Vea que, luego del símbolo de igual, se utiliza por primera vez la librería “requests” que importamos más arriba. Tendrá que afilar el ojo para darse cuenta que la variable y la librería son diferentes, vea que una finaliza en S y la otra no. Particularmente se utiliza el método “post” de la librería, ya que se está enviando información. Si se hubiese querido pedir información, se utilizaría el método “get”. Para hacer estos posts son necesarios básicamente dos elementos: el destino (que en este caso es la URL base seguido del endpoint “token”) y el payload. Si falta alguno de estos dos requisitos, el valor de la variable “request” será algún error del tipo 400. Los errores 400 son los pedidos mal hechos, los errores 500 son pedidos bien hechos pero que tienen algún problema del lado del proveedor del servicio, y los mensajes con códigos 200 son los que se ejecutaron satisfactoriamente.

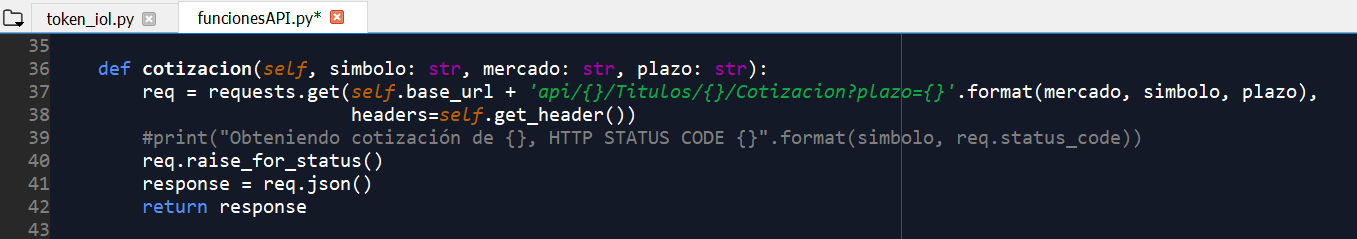
Una vez que se ejecuta el post, en las últimas dos líneas de este método se le asigna un valor nuevo a dos atributos del objeto Requests (self, él mismo). Vea que al atributo “token” se le asigna el objeto “Token” que habíamos definido en el primer archivo que creamos juntos. ¿Recuerda que había un input “response”? Aquí vemos que ese input es ni más ni menos que la respuesta que se obtiene de nuestro post. Lo que ocurría es que la respuesta de este post tiene los dos tokens diferentes y su fecha de expiración. Entonces es conveniente trabajar con un objeto que disponga de las tres características bien ordenadas. Por último, se le cambia el estatus del logueo a True en caso que la respuesta pase el control establecido. ¿Cuál es este control? Bueno, vea que al atributo “logged” (que antes era False) se le asigna el valor de sale de ir al objeto self (que en este caso es Requests), ver su atributo “token” (que le acabamos de asignar el objeto Token del archivo que hicimos al principio) y por último el método “is\_valid()”. Como este método es de la clase Token, para verificar qué ocurre en última instancia, tendrá que investigar el primer archivo que hicimos.

5.4.2 Consulta de información

Antes de comenzar con la operatoria propiamente dicha debemos aprender cómo consultar, descargar y manipular la información del mercado, materia prima del algotrading. Con los ejemplos que vimos anteriormente, tendremos una excelente base para continuar construyendo nuestras herramientas. Será importante recordar cómo navegar dentro de un objeto utilizando los índices que vimos durante nuestro repaso de las rutinas de iteración. Además, antes de continuar, sugerimos haber entendido el apartado anterior y tenerlo codeado en su computadora por que seguiremos alimentando los mismos archivos.

En toda esta sección veremos procedimientos de consulta. Recuerde que la librería requests tiene dos métodos principales: get y post. Por esto mismo, esté atento al método que verá en todos los ejemplos ya que al estar siempre solicitando información, se repetirá siempre el mismo método.

Veamos en primer lugar la solicitud más elemental: consultar la cotización de un activo. Observe cómo continuamos programando en el mismo archivo “funcionesAPI.py” que vimos en el apartado anterior. Tiene sentido seguir incluyendo todos estos desarrollos en el mismo archivo porque podría decirse que son funciones para utilizar las APIs y, no sólo eso, también tiene sentido seguir incluyéndolos en la clase Requests() porque todos serán pedidos al servidor.

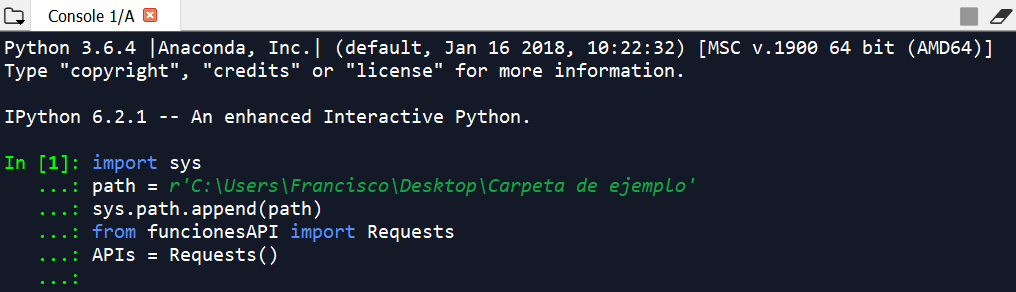


Aquí tenemos un nuevo método de la clase mencionada que requiere como input tres cosas importantes: saber qué activo se quiere consultar, en qué mercado y a qué plazo. Podría ocurrir que el activo local y su ADR tengan el mismo ticker, por lo que precisar qué mercado se requiere es bastante importante. Lo mismo ocurre con el plazo. No es lo mismo operar en contado inmediato que en plazo normal.

El paso siguiente es declarar una variable “req” a la que se le asignará la respuesta de una request del tipo “get” al servidor. Esta solicitud está direccionada a una URL especial destinada únicamente a estos fines. Vea que se ensambla la URL base más un endpoint especial para la consulta de cotizaciones. Recuerde de cuando aprendimos armar clases con el ejemplo de los autos que el método “format” introduce el valor de las variables dentro de un string en los lugares donde se introducen las llaves. Esto es porque las cadenas de texto suelen ser fijas, pero las variables tienen valores que pueden cambiar. Entonces se soluciona con esta práctica. Por último se utiliza en la misma request el método get\_headers() que en definitiva brindará el token actualizado. Repase esto último en la línea 24 del mismo archivo.

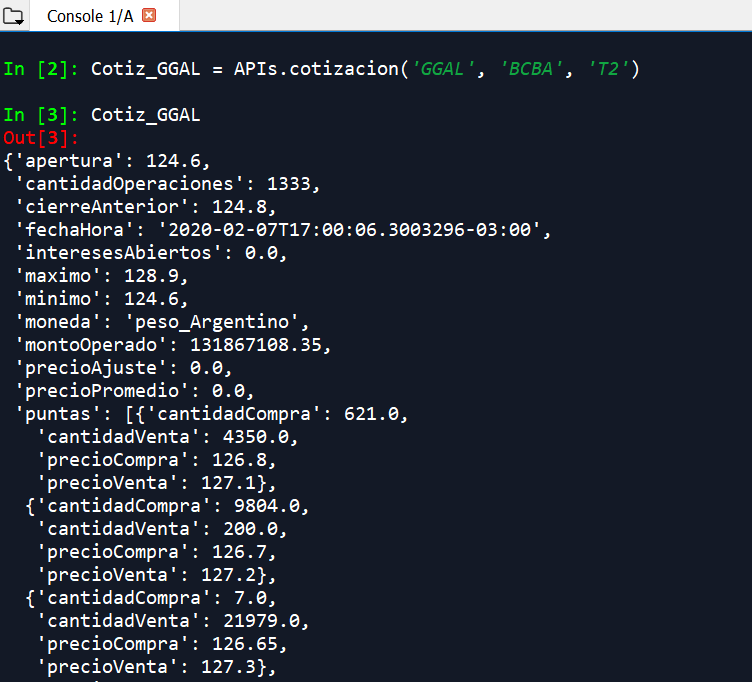
Las líneas 39 y 40 son para controles de errores. La primera está comentada, es decir que Python no la ejecutará porque entiende que es un comentario del programador. Si quiere ver qué hace, puede quitarle el numeral del principio y verá qué código HTTP devuelve. Si es 200, indicará que está todo bien. La otra línea dará algún tipo de respuesta sólo cuando ocurra un error. Si el código es 200 será exactamente como si no estuviese.

Al final del método, se crea una nueva variable “response” que será justamente la respuesta de la variable “req”. Pero acuérdese que las APIs en general devuelven información encriptada en un formato llamado “JSON”, como lo habíamos visto en el capítulo de rudimentos de programación. Por esto mismo, se le aplica el método correspondiente para desencriptar la variable “req” y finalmente asignar el valor a “response”. Al final, se pide que la consulta de cotizaciones retorne el valor de esta variable, lo que le dará la posibilidad de asignarle este valor a otro objeto como muestra las siguientes imágenes.



Esta primera imagen es el puntapié inicial para nuestra operatoria. Dejamos de trabajar en el editor de texto del Spyder y pasamos a la consola de IPython. En caso que esté trabajando en la misma carpeta en donde están todos los archivos, las primeras tres líneas no serán necesarias. Pero si está en otro directorio de su computadora, estas le dirán exactamente a Python en dónde buscar los recursos que está importando. Sino, le dará un error porque el sistema no encuentra lo que quiere utilizar.

Luego, se le pide a Python que, del archivo “funcionesAPI.py” que estuvimos trabajando, únicamente importe la clase Requests. Se crea una variable “APIs” y se le asigna un nuevo objeto. Los paréntesis son para indicarle que se desea asignarle el objeto inicializado, y no simplemente asignarle la clase. Una vez hecho esto, todos nuestros métodos estarán embebidos en esta nueva variable y podremos hacer uso, de una vez por todas, de los desarrollos que estuvimos viendo anteriormente. Para hacerlo, tendremos que escribir el nombre de esta nueva variable, agregarle un punto y ahí la herramienta de autocompletado a demanda nos ayudará a aplicarle el método que queramos.

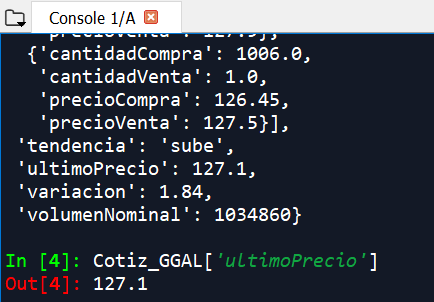


Vea como en la imagen creamos una nueva variable para almacenar la cotización de un instrumento. Si no hubiésemos hecho esto, la cotización no se habría guardado en ningún lado, lo que haría imposible su comparación. De hecho, si tenemos que usar la cotización para varios cálculos diferentes, no sería eficiente pedir la información muchas veces ya que haría muy lento nuestro código. Es mejor pedirla una sola vez, almacenarla en alguna variable como hacemos aquí y luego usar esa variable la cantidad de veces que necesitemos. Todo esto sin tener que recurrir al a consulta de la API en repetidas ocasiones, ya que la comunicación con el servidor requiere tiempo muy valioso para la operatoria algorítmica.

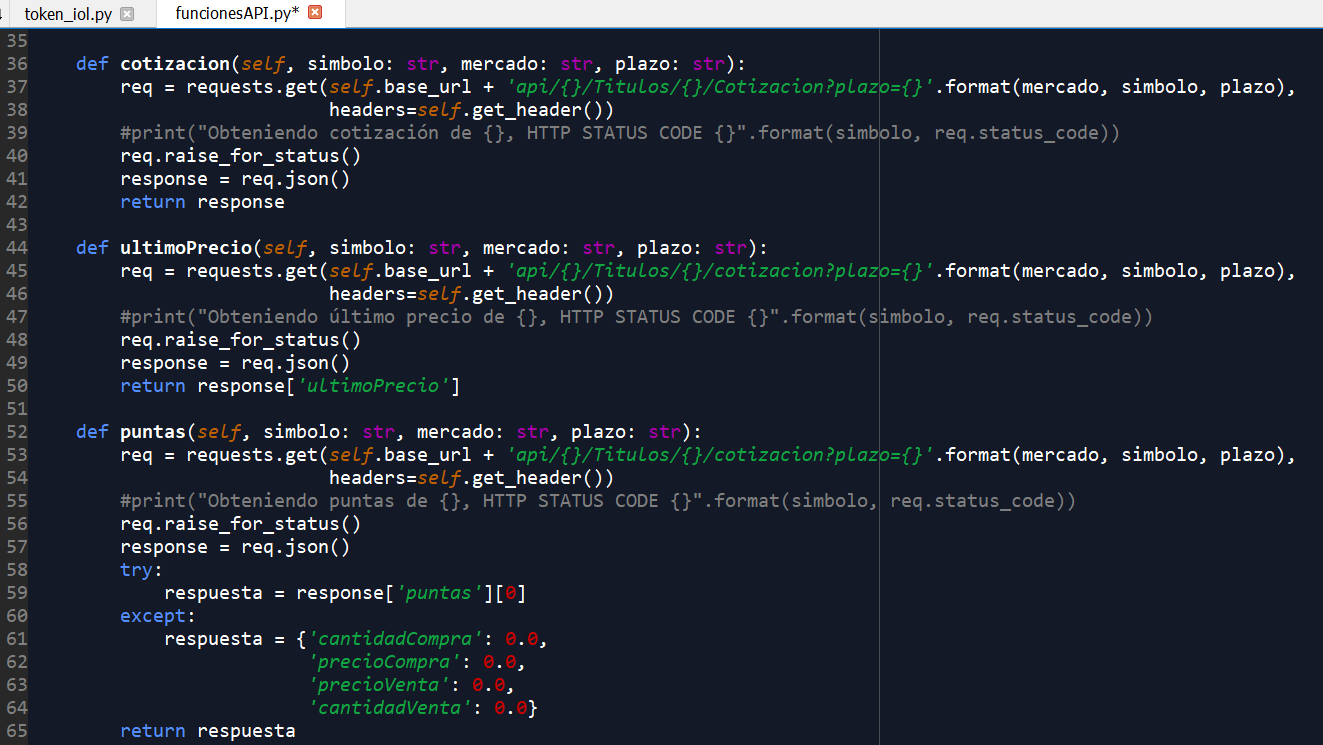
Cuando hacemos la consulta a la API, note que estamos introduciendo todos los inputs en formato de texto porque serán introducidos el endpoint de consulta de cotizaciones que es justamente un texto en forma de URL. Tanto las comillas simples como las dobles son igualmente válidas. Puede resultar útil también aclarar que BCBA corresponde a Bolsa de Comercio de Buenos Aires y que T2 corresponde al plazo de liquidación “T+2”, es decir 48hrs. Si se desea consultar el plazo de contado inmediato, introduzca un T0 en su lugar.

En la línea número 3 se pide conocer el valor de esta variable de cotizaciones que hemos creado gracias a nuestra primera consulta con la API. Lo primero que se puede observar es que la respuesta está estructurada como un diccionario ya que toda información está contenida entre llaves. Todos los JSON están estructurados de esta forma. Lo segundo que puede observarse es que la información que nos devuelve la API es demasiado extensa para lo que nosotros estamos queriendo. Dado este problema, tendremos que adentrarnos en la manipulación de esta información.

El servidor nos devuelve información sumamente útil. Tenemos el precio de apertura, el máximo, el mínimo, las puntas, el volumen y mucho más. Pero nosotros en este ejemplo queremos únicamente la cotización del activo. Para obtenerlo tenemos dos opciones: manipular la respuesta de nuestro método como está o reformular el método para que nos otorgue únicamente la información que queremos. Si bien aquí usted podría optar por cualquiera de las dos soluciones porque son igualmente válidas. Sepa igual que, la primera le demandará manipulación de la respuesta cada vez que utilice el método, mientras que la segunda le filtrará el dato automáticamente todas las veces.



Si opta por la primera opción, debería hacer algo como la imagen de arriba. Ya teníamos la variable entera, asi que filtramos la información por el “key value” que queramos. Observe que para filtrar diccionarios de esta forma hace falta introducir corchetes y el key value entre comillas. El resultado será finalmente la cotización del instrumento. Pero recuerde que tendrá que introducir los corchetes y las comillas cada vez que lo necesite. Si esto le parece demasiado engorroso, como a mí, debería hacer simplemente lo que sigue.



Volvemos a nuestro archivo de fucionesAPI.py y copiamos exactamente el método “cotizacion” como lo teníamos antes y lo pegamos abajo. Lo renombramos con un nombre nuevo para diferenciarlo del anterior y, en la última línea, filtramos la respuesta desde aquí mismo. El método será exactamente igual que el anterior, sólo que el filtrado estará desde el origen y no del lado del programador cada vez que lo utiliza.

En la misma imagen se muestra un tercer método que filtra información importante desde el origen. La metodología fue idéntica. Se copia y pega el método original, pero se filtra desde aquí la información que queremos. En este caso: las primeras puntas. Observe que el programador filtra la información con el key value “puntas”, pero en este caso el filtro es insuficiente, porque sigue habiendo demasiada información. Para este caso en particular, el operador necesitaba únicamente la primera punta, y por eso se le agrega un corchete adicional con el índice del dato que se desea. El número cero será entonces, las primeras órdenes de la caja de puntas.

El programador detectó que existían ocasiones en donde esta consulta traía errores. Particularmente, cuando el activo financiero no estaba cotizando no tenía puntas, como es de esperar. Si no rescataba esta excepción, su script se detendría inesperadamente a causa de un error por consultar un key value inexistente. Por esto mismo, introduce una estructura try-except para enmendar este problema, dándole a la excepción un formato similar a lo que hubiese recibido sin error. La estandarización de la información es muy importante. Los sistemas de trading sabrán cómo trabajar de una única manera, mejor darles siempre la información como la esperan.

Para finalizar con esta sección de consulta de información, veamos cómo consultar un histórico de datos. Este tipo de información tiene especial relevancia en el armado de sistemas de trading, ya que no posibilitaría calcular los factores de nuestro sistema usando la información histórica de los instrumentos financieros que vayamos a utilizar. Tiene especial utilidad para armar los backtesting que veremos más delante del libro.

Si bien el comportamiento de los instrumentos en el pasado no garantiza bajo ningún punto el comportamiento futuro, el análisis histórico de la información nos permite dar una idea de cómo se comporta el activo, calcular su volatilidad, su tendencia y muchas otras cosas que sirven para tomar decisiones basadas en información. Los activos financieros son variables estocásticas por definición. No puede garantizarse en ningún momento cuál será su comportamiento el día de mañana pero, al menos, puede estimarse con un cierto nivel de confianza estadística.

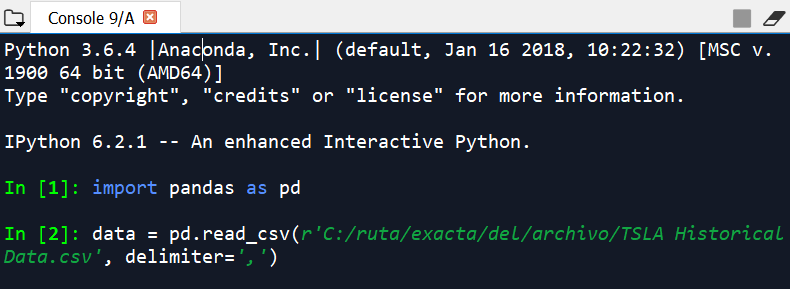


Siguiendo en nuestro archivo de funcionesAPI.py, agregamos dos métodos más. El primero será el que utilizaremos para obtener series diarias, mientras que el segundo será destinado para series intradiarias. Los inputs serán: el instrumento que quiere consultarse, la fecha de origen y la fecha final. Todos deben ser introducidos entre comillas para que sean cadenas de texto. Adicionalmente, es muy importante respetar el formato “dd-mm-aaaa” en las fechas para que el sistema interprete correctamente el intervalo que está solicitando. Si ingresa las fechas de cualquier otra forma, el sistema le arrojará un error 400 o 500. Recuerde que las fechas siempre presentan complicaciones, así que este último detalle resulta de especial relevancia.

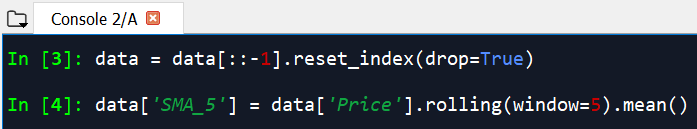
Estos dos métodos son tienen exactamente las mismas líneas que los que vimos anteriormente, pero llegando al final de su declaración vemos que el programador incluyó una pequeña variación a lo que aprendimos hasta ahora. Para empezar, en ambos casos, inicia una lista vacía con el nombre “precios”. Esto es para darle un destino a la rutina de iteración que incluirá en las líneas siguientes.

Ocurre que este tipo de consultas, devuelven la misma información que nuestra primera consulta de cotizaciones, pero para cada uno de los días que se consultan. Resulta que para calcular una simple media móvil, la respuesta presenta demasiada información. Por esto, el programador ideó un “for” que navega por cada elemento de la respuesta a través de sus índices, extrae la información que le interesa y luego la agrega a la lista vacía. En este ejemplo, el algoritmo recogerá únicamente la fecha de la observación y el precio de cierre de cada rueda. En caso que usted necesite almacenar otra información adicional, deberá modificar este script de ejemplo para adaptarlo a sus preferencias. Esto lo podrá hacer reemplazando alguno de los dos valores que tiene el ejemplo actualmente o simplemente agregar una coma después del último elemento e incluir el dato adicional que usted necesite.

En caso que usted ya disponga la información y no quiera utilizar esto último, sepa que también puede leer cualquier tipo de base de datos con el poder de Python. Por ejemplo, suponga que se ha descargado la base histórica y la tiene en formato “.csv”. Lo único que tendría que hacer es seguir los siguientes dos pasos:



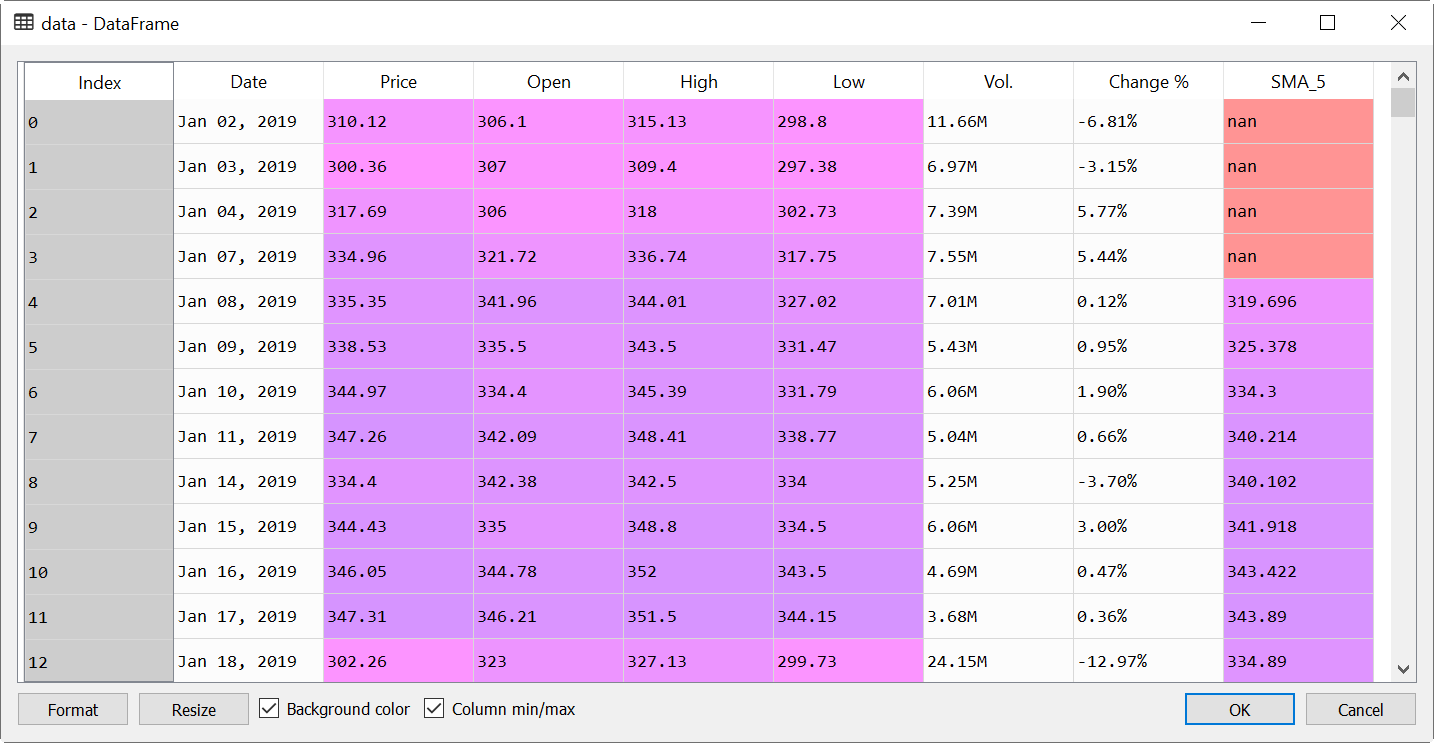
Con estos dos simples pasos, tendrá la información en una tabla de pandas que será sumamente fácil de manipular. Este es el formato más utilizado en Python cuando se trabaja con grandes volúmenes de información. La librería pandas incluso tiene infinidad de métodos que le ayudarán a hacer prácticamente todo lo que necesite hacer con la información. Por ejemplo, si quiere computar una media móvil lo hace simplemente con estas dos líneas:



La primera es para dar vuelta la tabla, ya que originalmente los datos más nuevos estaban arriba y para los cálculos es conveniente tenerlos al final. Está compuesta por dos partes. La parte anterior al punto, significa que se quiere la toda la información pero invertida. La parte que le sigue al punto es para resetear los índices, porque sino estarían invertidos y el “drop=True” es para descartar los índices originales ya que por defecto se conservan. En definitiva se le pide a Python que “data” sea igual a ella misma pero invertida y con nuevos índices.

La segunda línea es para calcular la media móvil finalmente. Antes del igual, se le pide a Python que la columna “SMA\_5” (como no existía, la creará) sea igual al promedio de los datos comprendidos en una ventana de 5 días de la columna “Price”. Sencillo, ¿no?

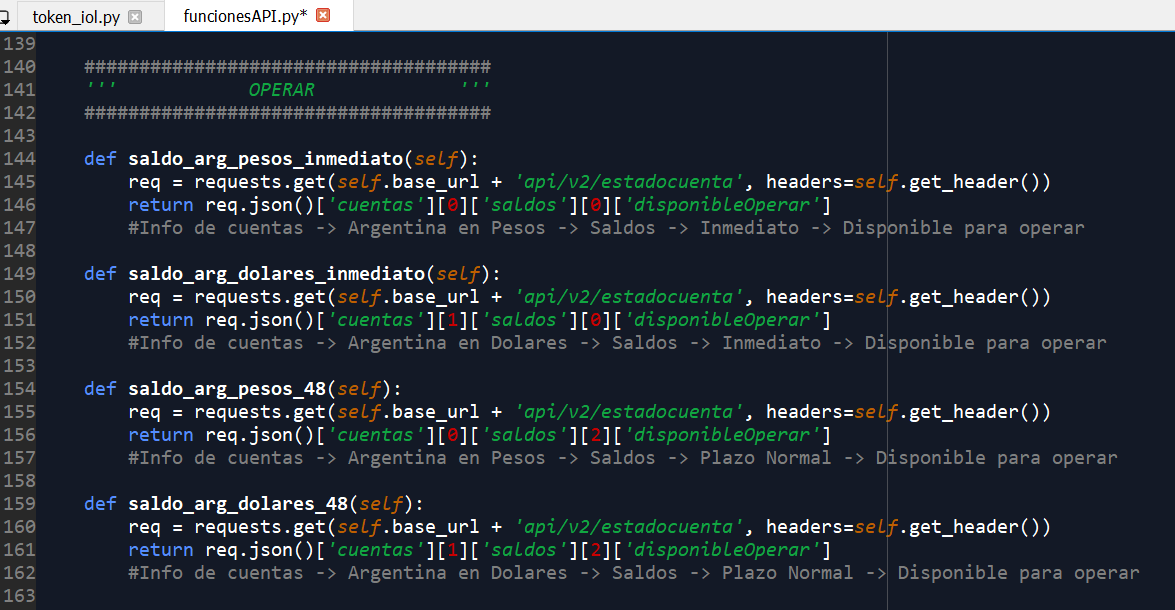
En caso que quiera ver cómo se leyó la información o si los cálculos que estuvo haciendo están bien, puede ir al explorador de variables de Spyder y ahí verá que hay una nueva variable llamada “data”. Hágale doble click y verá una tabla como la que sigue:



5.4.3 Primeras operaciones

Ha llegado el momento en que empezaremos a programar cosas que utilicen el saldo real de su cuenta. Recuerde que en estas ilustraciones veremos desarrollos reales que los operadores profesionales utilizan día a día, pero usted debería atravesar primero una parte de testeo de su sistema. Más adelante verá cómo puede hacer un backtest, pero es sumamente importante que, a diferencia de todos los ejemplos anteriores, esté completamente seguro que quiere operar con los recursos que presentaremos a continuación. Siga paso a paso todas las indicaciones de esta sección, pero no envíe órdenes hasta que haya probado que su sistema está correctamente diseñado y, en términos generales, le haga ganar dinero.

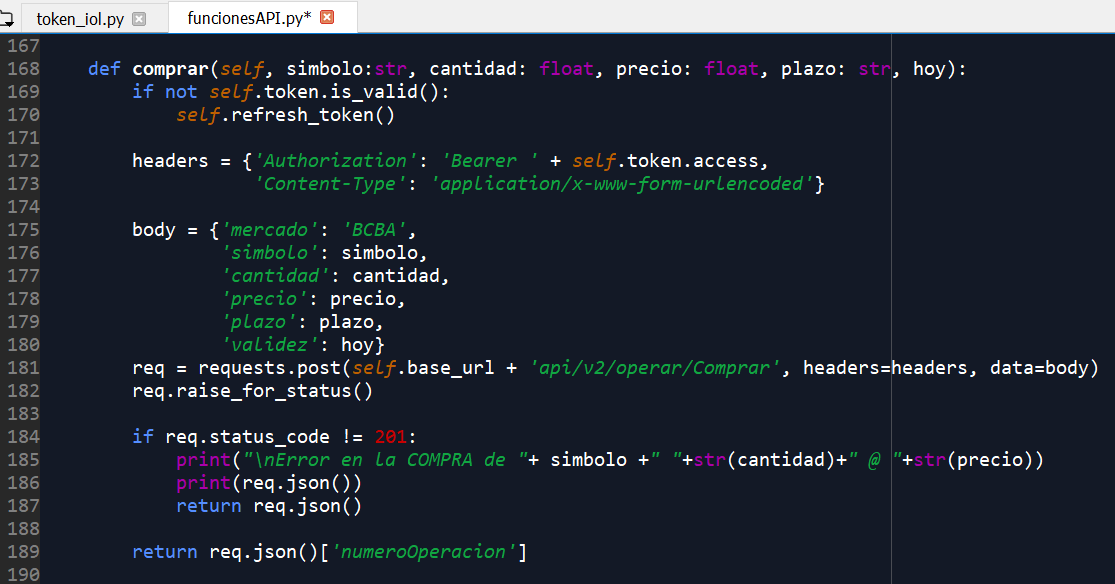
En primer lugar, veamos cómo puede consultar el saldo de su cuenta. Esta información le servirá para saber de antemano cuánto dinero tiene para operar. La respuesta que le brinde el servidor a esta consulta tiene que ser su límite de operatoria. Podría ocurrir que encuentra una oportunidad de operatoria pero sobre la marcha se da cuenta que no tiene dinero suficiente para alcanzar una escala suficientemente grande y termina teniendo un resultado negativo porque las comisiones licuaron toda la ganancia. Por eso, siempre debe tener bien en claro cuánto dinero tiene su robot para operar.



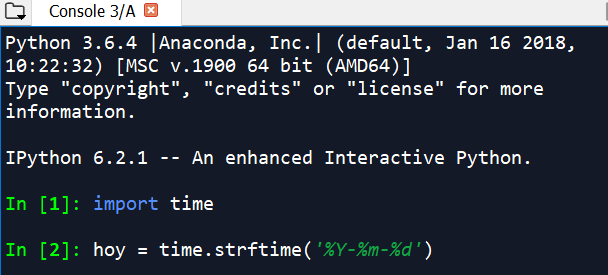
Seguimos en nuestro archivo funcionesAPI.py, pero fíjese que el programador incluyó un gran cartel en el script para tener bien presente que los métodos que vengan de aquí en más tienen incidencia real en los saldos.

En la imagen se presentan cuatro nuevos métodos que le permitirán consultar los saldos más relevantes de su cuenta comitente. Vea que en todos se incluyó un filtro en la respuesta, ya que este request devuelve demasiada información. En cada uno se incluyó un comentario al final para recordarle al programador cuál es la lógica de los filtros. Es altamente recomendable introducir comentarios habitualmente para ayudarse con recordatorios. Como puede observarse, la respuesta tiene una sección que se llama “cuentas”. En el índice 0 están las cuentas en pesos, mientras que en el índice número 1 están las cuentas en dólares. Se ve que en esta sección hay diversidad de información adicional, pero se filtra únicamente la parte de “saldos”. A continuación parece estar el plazo que se desea consultar, los datos ubicados en el índice 0 son los saldos en contado inmediato y los datos ubicados en el índice 2 son los saldos en el plazo normal de 48 horas. Por último, hay que filtrar qué tipo de saldo se desea obtener: el saldo inicial, el comprometido o el disponible para operar. El programador dejó fijo que siempre quiere el saldo disponible para operar.

Pasemos a la mejor parte. Veamos cómo tienen programadas las operaciones de compra y de venta los operadores cuantitativos profesionales. Recuerde que, llegado el caso que necesite más información o desee ampliar estos contenidos puede seguir la documentación disponible en [https://api.invertironline.com](https://api.invertironline.com/).



Este nuevo método tiene todos los atributos que necesita una compra normal: el símbolo de la orden, la cantidad, el precio límite que se desea pagar, el plazo en el que se quiere operar y el plazo de validez de la orden que en este caso será sólo por hoy. Este último input debe tener siempre el formato “aaaa-mm-dd”. Si desea puede incluir las siguientes dos líneas al principio de su script para nunca fallar:



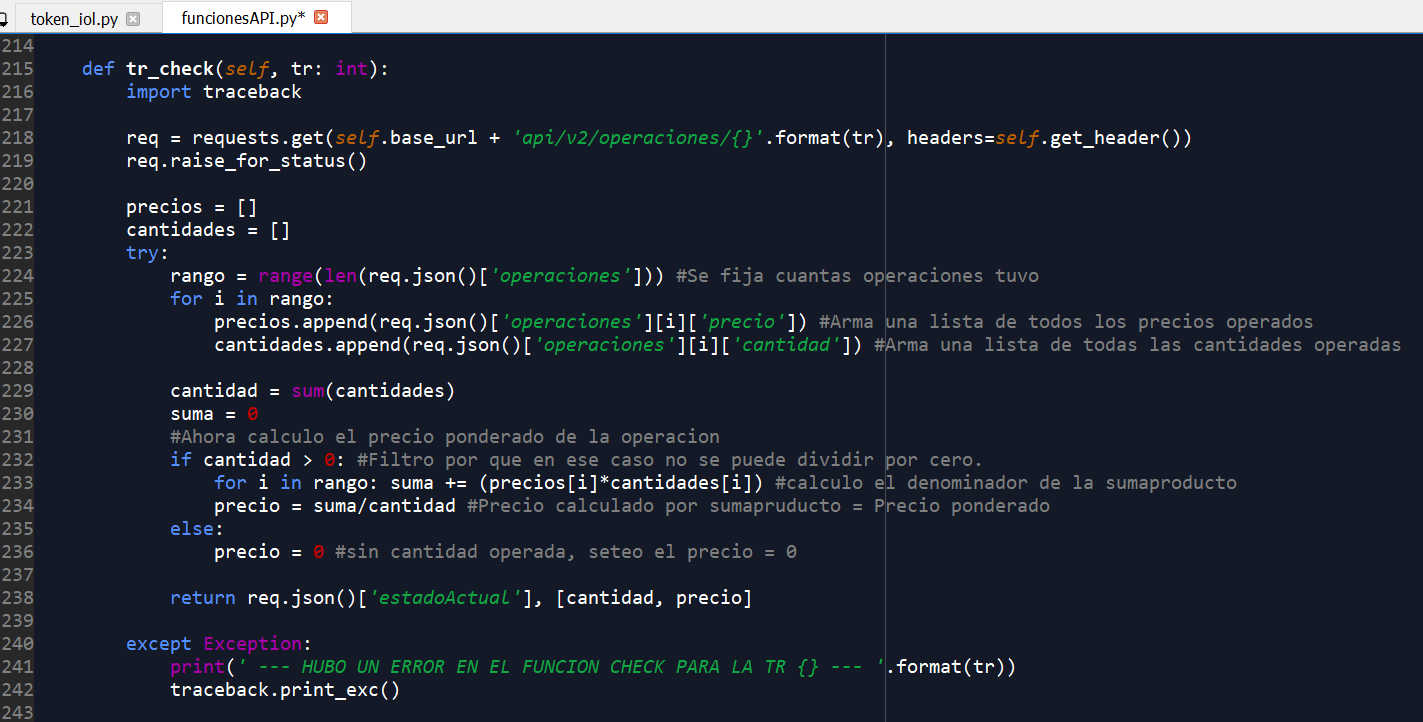
Fíjese que el método tiene unos *headers* al principio, que serán siempre los mismos y luego tiene un *body* que es igual al que se indica en la documentación. Una vez que se tienen estos dos elementos, se puede hacer la request a la API con el método “post” para enviar la información. Luego el operador armó un manejo de excepciones para que cuando exista alguna excepción, el sistema le diga que hubo un error en la compra que acaba de enviar. Por último, la API devuelve un JSON con mucha información pero el operador programó que sólo se devuelva el número de operación para saber qué transacción generó. Las dos primeras líneas del método sirven para asegurarse de tener siempre un token válido.

La forma de programar es muy personal, así que crea que esta será la única forma de ejecutar tus compras. Recuerde siempre enviar sus órdenes con los parámetros correctos y al endpoint correspondiente, sino la API le devolverá un error 400. Todo lo demás, como el manejo de excepciones, será completamente personal.

La operación de venta es prácticamente igual, sólo que el endpoint cambia al que se usa exclusivamente para operaciones de venta:



Antes de finalizar con este módulo, nos gustaría también otorgarle dos métodos adicionales que le servirán como control y contingencia. Como habíamos dicho hace muchas líneas atrás, es sumamente importante desarrollar un algoritmo que contemple todos los casos posibles. De hecho, desde mi punto de vista es más importante contar con mecanismos de seguridad que con mecanismos de ejecución. Si bien puede haber leído varias secciones de este libro para tener acceso al código que acabamos de repasar, esperamos que le preste la misma atención a los dos métodos que siguen a continuación. El primero será para verificar el estado de una operación, la cantidad operada y calculará el precio ponderado de compra. Será realmente útil para llevar una contabilidad activa y bien detallada dentro de su mismo algoritmo, para que este pueda ser autosuficiente no solo en la operatoria, sino también en la contabilidad. El segundo es un método de emergencia para cancelar una operación que no se haya ejecutado. A veces es prudente cancelar las operaciones que están mucho tiempo en el mercado. La situación pudo haber cambiado rotundamente de un momento al otro, quizás su estrategia deja de ser rentable y necesitará cancelar operaciones luego de algún tiempo prudencial.



¿Recuerda que el operador sólo quería el número de operación en los métodos de ejecución? Bueno, el motivo es que su algoritmo controlaba de forma automática el estado de las operaciones que iba generando a través de este primer método de contingencia. Fíjese que el único input necesario es el número de transacción.

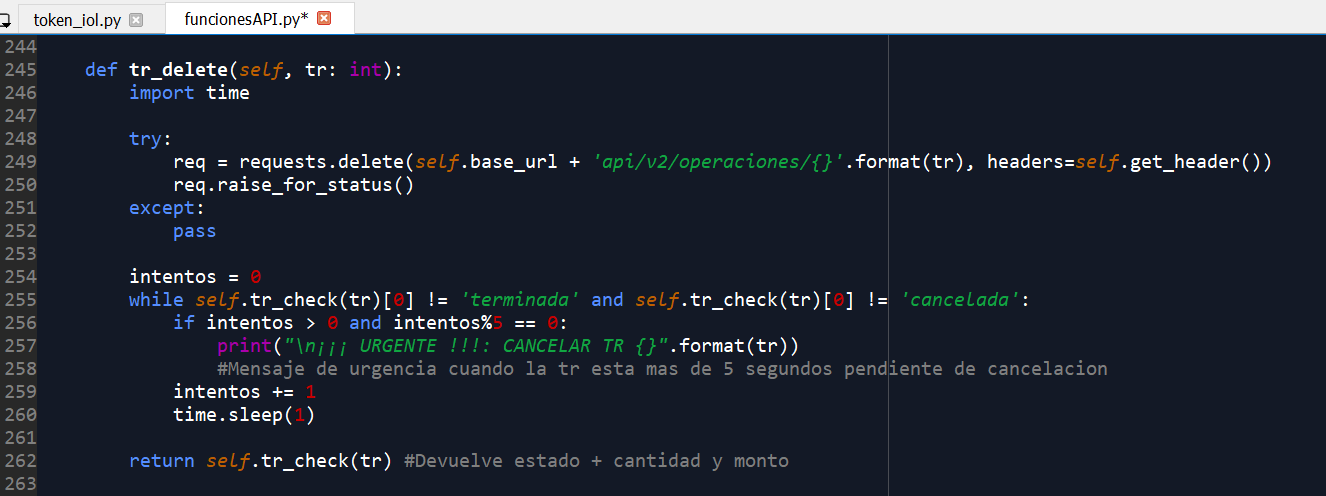
Luego de autentificarse y enviar el request al servidor, el algoritmo genera dos listas vacías para almacenar la información que se calculará a continuación. Introduce toda la lógica en una estructura try-except para que su robot no se detenga en caso de errores, que sólo le imprima un mensaje de alerta en su panel de control.

Observe qué fácil es seguir toda la secuencia de instrucciones gracias a los comentarios que el operador dejó en el script a medida que lo iba desarrollando. Lo primero que hace es detectar cuántas operaciones ejecutó nuestra orden. Recuerde que puede ser que su orden no sea ejecutada en una sola parte y que puede ser a diferentes precios. Una vez que se sabe cuántas operaciones están involucradas en nuestra transacción, el algoritmo comienza una rutina de iteración alimentando las dos listas vacías que había creado al principio. Si no hubo operaciones, la cantidad de títulos operados lógicamente también lo será, por eso al continuar con la estructura condicional se establece que sólo se continuará si la cantidad de títulos operados es positiva.

Acto seguido, se pasa a calcular una suma producto de los títulos operados por sus respectivos precios. Para ello, se creó una variable nueva llamada “suma” con un valor inicial de cero y luego se recorrió cada una de las operaciones sumándole progresivamente la multiplicación entre la cantidad de títulos por el precio correspondiente. Al fin y al cabo, si se divide esta nueva variable suma (que será en última instancia el valor monetario de cada operación) por la cantidad total de títulos operados, se obtendrá el precio ponderado de la operación.

Este método tendrá como salida final dos valores. El primero será un string y el segundo será una lista de dos elementos. El string será el estado de la operación. Los más comunes e importantes son: iniciada, en proceso, ejecutada o cancelada. El segundo elemento, la lista, estará conformada por la cantidad de títulos ejecutados y el precio ponderado de la operación hasta ese momento.

Para finalizar el módulo, veamos el freno de mano de su robot: la cancelación de una operación. Este método en particular no va a usar ni un “get” ni un “post”. A diferencia de todo lo que vimos hasta ahora, en este caso se utilizará el método “delete” de la librería requests.



El único input que necesita, al igual que el método de verificación de estado, es el número de transacción que se está analizando. Utiliza una librería llamada “time” que nunca habíamos visto y se la utilizará para pedirle más adelanta al algoritmo que espere antes de continuar. Ocurre que el algoritmo tiene una velocidad de cómputo demasiado elevada y a veces hay que pedirle que descanse para darle tiempo suficiente a otros sistemas para que reaccionen. Vea que se introduce una rutina de iteración que se repetirá indefinidamente hasta que el estado de la operación sea igual a “terminada” o “cancelada”. Recuerde que el método de verificación devolvía una respuesta doble, por eso aparece el cero entre corchetes, para indicarle que sólo queremos el estado. También se observa que el programador utiliza un nuevo símbolo lógico: “!=”. Lo que interpreta Python de esto es que debe continuar iterando siempre y cuando el estado sea diferente de “terminada” y “cancelada”. Si el estado es igual a alguno de estos, la iteración se detendrá y el método devolverá lo mismo que el de verificación de estado.

Con todos estos nuevos recursos, ya está listo para programar su primer algoritmo. Posee las herramientas de consulta de información, los métodos de ejecución y hasta posee algunos procedimientos de contingencia para darle seguridad y completitud a su robot. Sin embargo, es necesario hacer muchas pruebas antes de poner un algoritmo en marcha. Al desarrollar una lógica de operatoria, no debe lanzarse a probar con dinero real, sino que primero debe probar infinidad de variaciones para asegurarse de ganar dinero. No sólo para confirmar que su robot interpreta correctamente la naturaleza de su problema sin ambigüedades, sino también para asegurarse que la lógica de su operatoria genera dinero en la bolsa. Una cosa no garantiza a la otra. Podría tener un robot completo pero que pierda dinero o ganar dinero momentáneamente hasta que el mercado le hace dar cuenta que se le escapó una situación especial que su robot no contempló.