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 es 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 preguntándose “¿hoy llueve?” o “¿hoy no llueve?”. Lo que si 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 son 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 puede comprar, comprar y sólo comprar. En la bolsa a veces hay que comprar, otras vender y a veces simplemente esperar afuera sin posición.

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. Si bien aquí entraré en detalle en los “for” y los “while”, quizás usted tenga a su disposición otros parecidos. De todas formas le aseguro que todos los problemas que los requieran podrán ser expresados de tal forma que sea lo mismo utilizar uno u el otro, 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.

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 de la acción con la que lo imaginamos y deseamos que recorra todo un grupo de veinte acciones. 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, ya que con las rutinas iterativas 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, dicho en código, podríamos decir “por cada elemento de un grupo, itere la siguiente rutina” o “mientras la condición se cumpla, itere la siguiente rutina”. Podríamos 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. En estas líneas intento expresar de la forma más clara lo que intento explicar, pero podría utilizar un vocabulario más difícil si hiciese el esfuerzo y podría decir los mismo pero diferente. De la misma forma, podría invertir los ejemplos anteriores para invertir las estructuras y mostrar que se pueden intercambiar. Pero de poco vale la pena dificultar la tarea. Mejor expresar de la forma en que mejor uno se entienda. Pero sepa que se podría hacer con un poco de esfuerzo.