0. Discernir con un ordenador: Programas

Los discursos son muy frecuentes en el día a día, refiriéndome a discurso como cualquier acción, por muy simple que esta sea, de conversación utilizando cualquier medio de comunicación ya sea oral u escrito. Cualquier narración, descripción, exposición, argumentación, instrucción, etc. en cualquier medio como la radio, el diálogo, la televisión, el periódico, la revista o el libro son ejemplos de discursos.

En nuestro caso, para entablar cualquier tipo de discurso que queramos, solamente se necesitaría un medio como el habla o el papel y bolígrafo y un idioma. Por ejemplo, para escribir esta guía de programación, hemos necesitado de un editor de documentos de mi ordenador y de lengua castellana.

Sin embargo, este ordenador no escribe esta guía por la magia de darle a las teclas correctas para formar oraciones ni las maqueta para que correspondan con palabras claves, título del capítulo, título de la sección, etc. de la nada, sino porque hay un **programa** que me permite realizar estas acciones.

Pero vayamos al principio, a la pregunta primera que nos debemos hacer. ¿Qué es un programa? Según la doctrina informática, se define como un **conjunto de instrucciones basadas en un lenguaje de programación** específico que un ordenador usa para **resolver un problema determinado**, es decir, una serie de pasos a seguir que se asemeja al **discurso instructivo** como cuando por ejemplo se quiere hacer una receta de un libro de cocinas.

Sin embargo, esto no es del todo cierto. Un programa no tiene por qué ser algo que simplemente siga una serie de pasos como si fuera una receta para hacer cosas muy específicas como calcular la raíz cuadrada, buscar un objeto de la lista o rellenar celdas de una base de datos. Un programa puede cumplir con un **propósito** más **general** como por ejemplo un editor de documentos, un juego de estrategia o un software de edición de multimedia. En estos programas, no existe un único rompecabezas o proceso a resolver, sino que este debe cumplir con su cometido resolviendo tantos rompecabezas como se precise.

Poniendo de ejemplo el juego de estrategia, debe reunir un montón de características como los escenarios, la historia, los niveles, los personajes, la jugabilidad, etc. Sus problemas concretos o rompecabezas como me gusta llamarlos ocupan un buen fragmento de código que en conjunto todos forman el programa. Necesitamos una parte de código que grafique en pantalla los escenarios, otra que configure los movimientos del personaje, otra que asigne la puntuación del personaje por enemigo derrotado, etc.

Ese **conjunto de instrucciones definido, ordenado y finito** que soluciona un problema, realizando **cómputos** o procesando **datos** determinados es lo que comúnmente denominamos **algoritmo**. Luego, el programa que gestiona el juego de estrategia debe gestionar distintos algoritmos para ejecutar correctamente todos los aspectos del juego.

Todavía no desarrollaremos el concepto de algoritmo, este se hará en el siguiente tema, pues me gustaría centrarme más en el mero funcionamiento de un ordenador.

Pregunta, ¿Cómo entabla el ordenador un discurso? Respuesta, por medio de programas. ¿Cómo los llega a comprender el ordenador? Vamos a abstraernos de ese proceso y nos lo vamos a imaginar como si el ordenador fuera un gólem del cual tú metes un trozo de pergamino en la boca con datos iniciales y obtienes el resultado final de la tarea que le encomendaste. El gólem tiene una peculiaridad y es que sólo es capaz de comprender 0's y 1's, es decir, sistema binario, pues es su lengua materna.

Por suerte, este gólem también sabe distintos **lenguajes de programación** y es capaz de traducir desde esos lenguajes al **código máquina** binario que conoce. Así mismo, para comprender básicamente cómo funciona la programación, no necesitamos saber cómo se traduce un programa de cualquier lenguaje a código máquina, de la misma forma que no necesitamos saber todo lo que hay dentro de un automóvil para conducir.

Ese paso llamado **compilación** lo percibiremos a partir de ahora como una **caja negra** donde asumiremos que ese proceso es autónomo y que esta caja recibe datos de entrada (input) y envía datos de salida (output), o también simplemente transforman el dato (in&output)

La ventaja de utilizar un lenguaje de programación y no el código máquina, es que podemos conocer a la perfección lo que realiza el programa leyendo únicamente su código, ya que **el lenguaje de programación se asemeja al lenguaje natural** y es totalmente inteligible. Veamos el siguiente programa escrito en lenguaje Pascal:

`program de_24h_a_12h (INPUT, OUTPUT);`	Título del programa
`{Este programa lee un número entero que simboliza una hora en formato 24h como 0835 o 2050 e imprime en pantalla su correspondiente en formato 12h}`	Descripción o especificación del programa
`var` `Hora_24h, Horas, Minutos: Integer;`	Declaración de variables

`begin`
`Write ('Escriba la hora en formato 24h: ');`
`Read (Hora_24h);`
`Horas := Hora_24h div 100;`
`Minutos := Hora_24h mod 100;`
`Write ('La hora en formato 12h es: ');`
`if (Horas = 0) then`
`  Write ('12:', Minutos, ' a.m')`
`else if (Horas > 0) and (Horas < 12) then`
`  Write (Horas, ':', Minutos, ' a.m')`
`else if (Horas = 12) then`
`  Write ('12:', Minutos, ' p.m')`
`else`
`  Write ((Horas – 12), ':', Minutos, ' p.m');`
`end.`
	Instrucciones del programa o algoritmo

Como podemos observar, no necesitamos apenas conocer las reglas sintácticas ni semánticas del lenguaje de programación para comprender lo que hace el programa, sólo se necesita un **conocimiento mínimo de inglés, de matemáticas y de lógica**. Expresemos lo que hace el programa con nuestras palabras:

<<En primer lugar, aparece una cabecera del programa, que indica el **título** de este, y se titula "de_24h_a_12h", lo que nos da una idea de lo que hace el programa. Además, en este caso se nos indican dos etiquetas llamadas INPUT y OUTPUT. Esto permite al programa tanto **leer del teclado** los datos solicitados como **imprimir**los **en pantalla**. La **habilitación de la E/S** depende del lenguaje de programación, cada uno tendrá su manera peculiar.

Tras el título del programa, nos aparece una **breve descripción** de lo que debe realizar, lo que también se denomina como **especificación**. Esto no es obligatorio, pero nos sirve de gran ayuda a la hora de compartir programas con otros desarrolladores con tal de que se comprenda cómo nosotros lo hemos realizado. Esta descripción aparece a modo de **comentario**, y **el ordenador hará caso omiso** de este.

A continuación, se **declaran las variables** que necesite el programa. Las variables son los **espacios de memoria** principal del ordenador que se reservan para una cantidad de **información** conocida o desconocida de un **tipo de dato** en concreto y modificable a lo largo del desarrollo del programa. Así mismo, necesitamos almacenar en nuestro programa tres datos de tipo entero: el entero que simboliza la hora en formato 24h (p.ej. 1615) y otros dos para indicar las horas y minutos que han pasado (siendo Horas=16 y Minutos=15)

Por último, se ejecuta el **algoritmo** necesario para obtener la hora en su correspondiente formato de 12h. Para ello, se nos pide introducir el valor de Hora_24h por el teclado, un entero. Una vez hecho esto, el algoritmo asignará a Horas sus dos primeras cifras y a Minutos, sus dos últimas. En este ejemplo (1615), como Horas es mayor que 12, ejecutará la línea de código del bloque IF correspondiente, restando 12 a Horas, y señalizando debidamente la hora con la marca "p.m", mostrándose en la pantalla "4:15 p.m">>

La estructura del discurso que sigue nuestro programa es muy similar a la de cualquier texto escrito, ya sea narrativo, expositivo, argumentativo, etc.

Programa	Título	Especificación	Declaración	Algoritmo
Instrucción		Introducción	Listado de inventario	Procedimiento
Narración		Planteamiento	Nudo	Desenlace
Descripción		Introducción	Desarrollo	Conclusión
Argumentación		Tesis	Cuerpo	Conclusión
Exposición		Presentación	Desarrollo	Conclusión

Sin embargo, la principal diferencia entre los textos no instructivos y el programa radica en la localización de su parte nuclear. Mientras que en los textos no instructivos la parte nuclear se encuentra en medio del texto, y el final es una conclusión extraída de los acontecimientos, hechos o argumentos de dicha parte; en el programa y en el texto instructivo, antes de desarrollar su núcleo, se procede a hacer un listado del inventario que se necesita para realizar correctamente la tarea, siendo por una parte una lista de ingredientes y utensilios para una receta de un libro de cocina y por la otra parte la declaración de las variables que se necesiten para la correcta ejecución del programa. Luego, la parte principal del programa es el algoritmo responsable de ejecutar la tarea, el cual se desarrolla tras declarar todas las variables que el programa necesita, de forma similar a cuando se escriben los pasos que debe seguir una receta de cocina después de realizar la lista de ingredientes y utensilios necesarios. Ambos funcionan no sólo de desarrollo, sino también de conclusión.

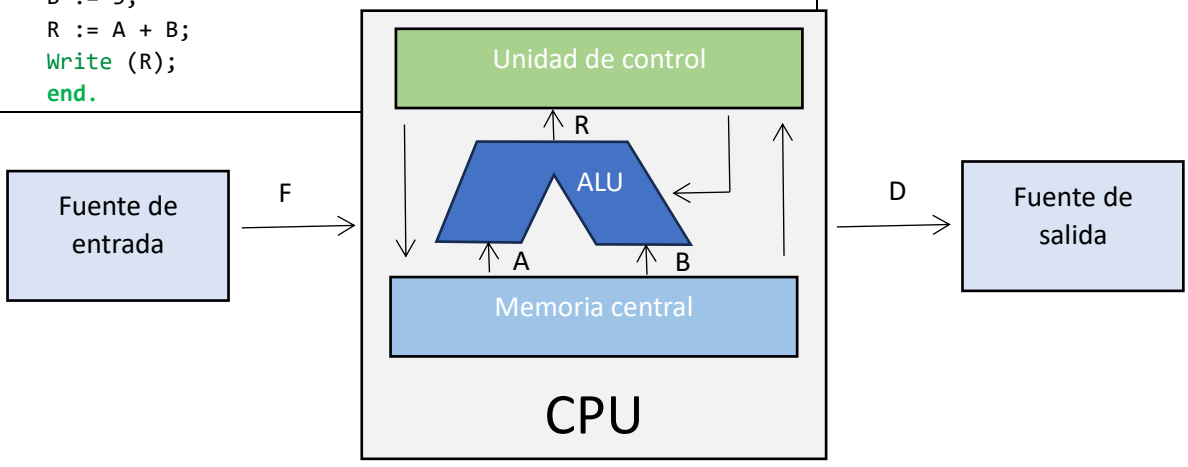
Por otra parte, todos los textos se caracterizan por su introducción, que explica siempre el propósito inicial del texto, ya sea la presentación de la descripción de un lugar, la tesis que se desea defender, un breve precepto de los hechos de una historia o la breve descripción de lo que realiza un programa.

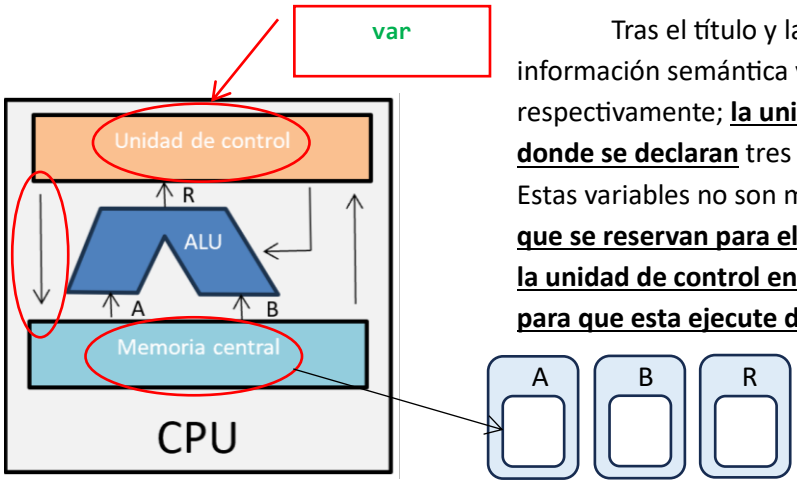
Para terminar este tema, me gustaría explicar un poco más detalladamente cómo el ordenador es capaz de ejecutar un programa a través de los distintos componentes del ordenador. Para ello, asumimos que nuestro procesador sigue la **arquitectura de Von Neumann**.

Este diseño es bastante más simplificado de lo que sería la arquitectura del computador en realidad, pero lo suficiente como para comprender el mecanismo de la programación. Pondremos como ejemplo otro programa en lenguaje Pascal.

```
program Suma_5 (INPUT, OUTPUT);

{Este programa lee un número entero A por el teclado,
al cual se le suma la variable B, a la que se le asigna
el valor 5. El resultado R de la suma se imprimirá en
pantalla}

var
A, B, R: Integer;

begin
Read (A);
B := 5;
R := A + B;
Write (R);
end.
```

Con este programa y el esquema del **CPU**, del inglés, **Unidad Central de Procesamiento** explicaremos cómo el ordenador es capaz de **leer, almacenar, procesar y escribir datos** a través de sus distintos componentes.

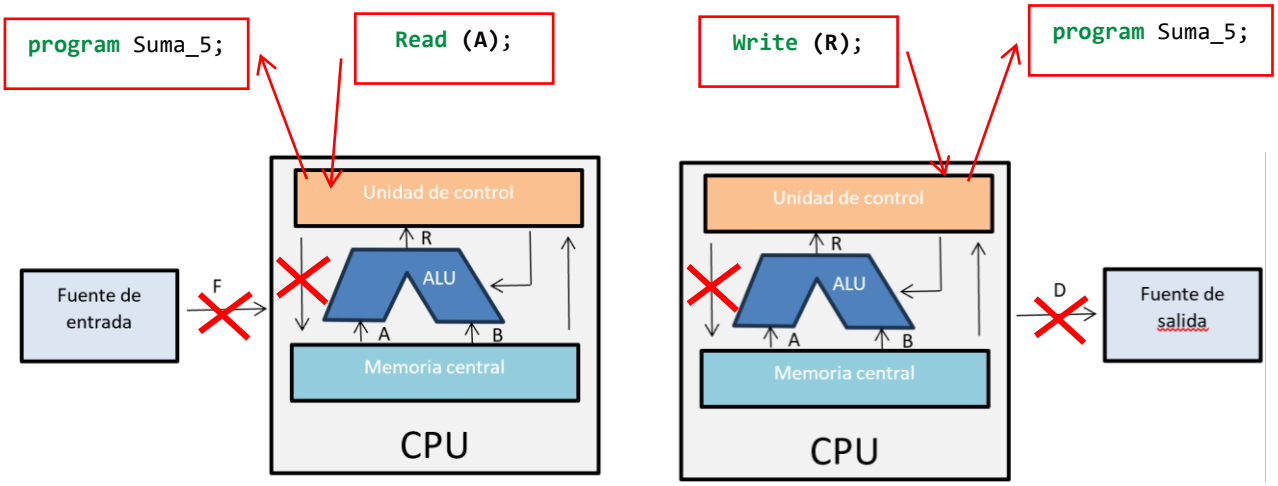




Tras el título y la especificación, que es más bien información semántica y descriptiva para el desarrollador, respectivamente; **la unidad de control recibe la instrucción donde se declaran** tres **variables**, A, B y R de tipo entero. Estas variables no son más que **huecos de memoria vacíos que se reservan para el almacenamiento** de un entero. Así, **la unidad de control envía una señal a la memoria central para que esta ejecute dicha acción**.

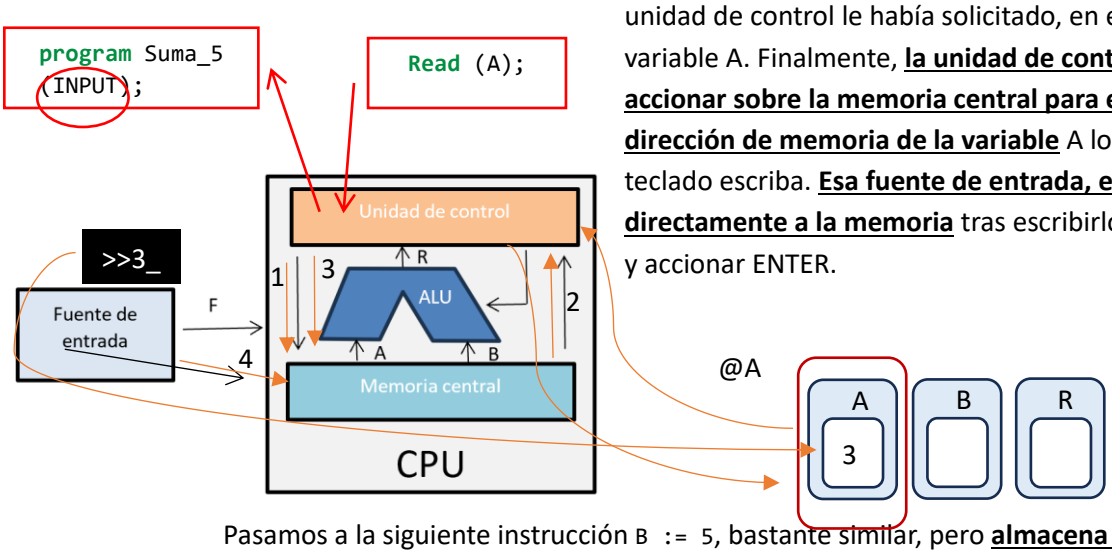
A continuación, se procede a ejecutar el algoritmo del programa. En este caso, debido a la sintaxis del lenguaje, la unidad de control lo procesa porque recibe la palabra reservada `begin`, la cual indica el inicio del algoritmo.

En primer lugar, tenemos la instrucción `Read (A)`, que lee el entero por el teclado y lo almacena en la variable A. Importante saber bien estas dos cosas relacionadas con la E/S. En primer lugar, **la unidad de control no enviará una señal a la memoria central** (que es la que permite almacenar el dato en la variable) **si no se ha habilitado previamente la fuente de entrada o de salida** (en este caso, mediante la etiqueta INPUT u OUTPUT), resultando en un **error de compilación**.



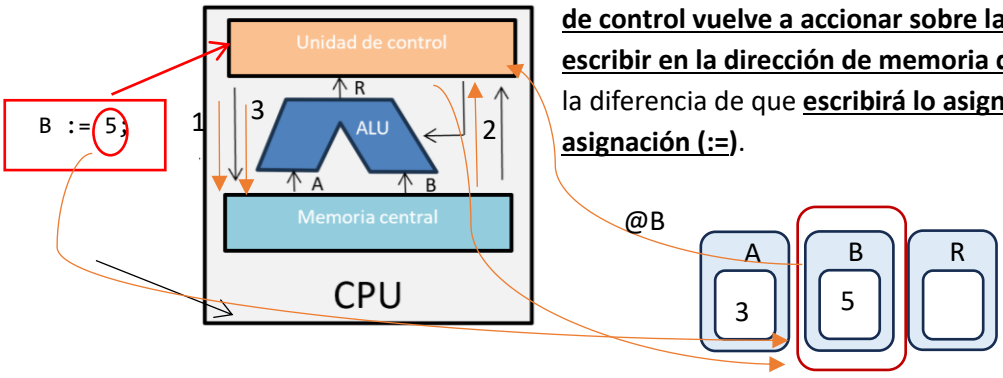
Lo segundo, **al introducir un dato por el teclado, hay que asegurarse de que lo que se introduzca pertenece al tipo de dato que se solicita**. Así mismo, si se pide un entero no se pueden introducir ni números decimales, ni letras, ni oraciones ni nada distinto a un número entero. Si no se hiciera, provocaría un **error de compilación** por la no correspondencia de lo que se recibe y lo que se desea almacenar.

Prosiguiendo con la instrucción de **lectura de datos, si la etiqueta INPUT está habilitada**, la unidad de control llamará a la memoria central y le advertirá de que va a recibir un dato por el teclado, y por tanto **el bus de entrada de datos se habilitará**. Posteriormente, **la memoria manda a la unidad de control la dirección de memoria donde se almacena la variable** que la unidad de control le había solicitado, en este caso, la variable A. Finalmente, **la unidad de control vuelve a accionar sobre la memoria central para escribir en la dirección de memoria de la variable** A lo que el teclado escriba. **Esa fuente de entrada, enviará el dato directamente a la memoria** tras escribirlo en el teclado y accionar ENTER.



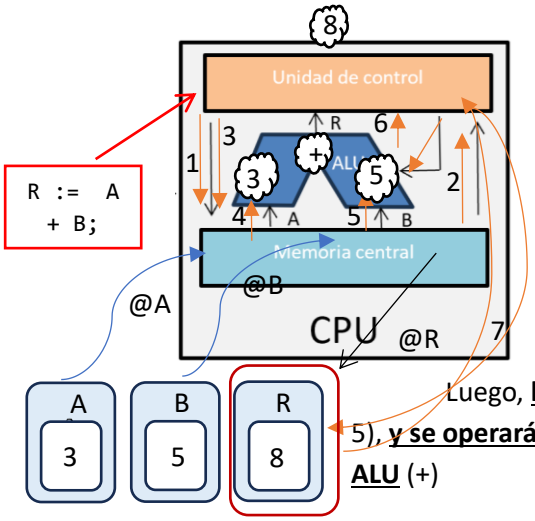
Pasamos a la siguiente instrucción `B := 5`, bastante similar, pero **almacena el dato** directamente dada una instrucción que asigna el valor 5 a la variable B, **por acción del código en sí** y no por fuentes externas al procesador.

Los pasos que se ejecutan son exactamente iguales salvo el último, en el que **la unidad de control vuelve a accionar sobre la memoria central para escribir en la dirección de memoria de la variable** B, pero con la diferencia de que **escribirá lo asignado en la instrucción de asignación (:=)**.



A continuación, ejecutamos la siguiente instrucción `R := A + B`. En esta línea de código, se deben sumar los valores almacenados en las variables A y B y almacenar dicho resultado en la variable R.

Para ello se hará uso de la **ALU**, del inglés, **Unidad Aritmético-Lógica**. La ALU es un componente esencial para el **proceso de datos** y la ejecución correcta de las operaciones sobre los datos.



Como de costumbre, en primer lugar, **la unidad de control lanza una señal a la memoria central** y se le advierte de la operación que se desea ejecutar.

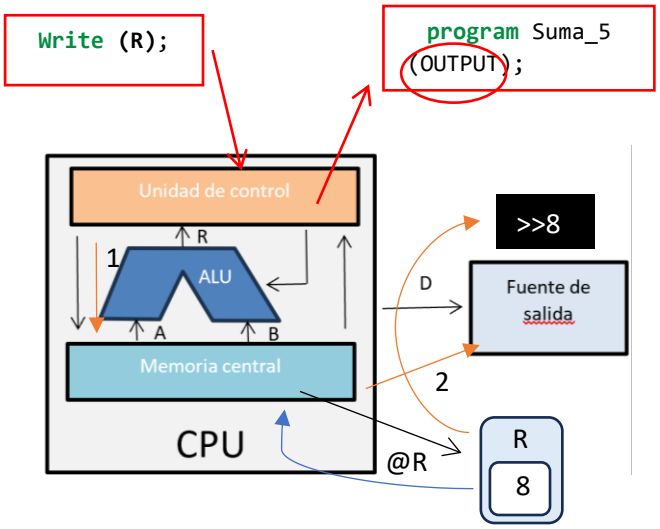
La memoria envía a la unidad de control la dirección de memoria de la variable de destino (R) **y busca las direcciones de memoria de las variables fuente** (A y B), es decir, los operandos.

Luego, **la memoria envía los valores de los operandos a la ALU** (3 y 5), **y se operarán según el operando que la unidad de control envíe a la ALU** (+)

Finalmente, **la ALU opera la operación** (3 + 5) **y emite un resultado** (8). **Este resultado se escribirá en la dirección de memoria de la variable destino** (R) una vez la unidad de control ejerza una señal sobre la memoria.

Y en el paso final, ejecutaremos la instrucción `Write (R)`, la cual **escribirá el dato** mostrándolo en la pantalla.

Si la etiqueta OUTPUT está habilitada, la unidad de control llamará a la memoria central y le advertirá de que uno de los datos que tiene guardados va a imprimirse en pantalla, y por tanto, **se habilitará el bus de salida de datos.**



La memoria buscará la dirección donde se almacenó la variable solicitada por la unidad de control (R) **e imprimirá su valor en pantalla** (8).

Esta breve explicación del funcionamiento del procesador nos da una breve idea de lo que se realiza en la caja negra, pero no podemos excedernos de confianza, ya que sólo estamos viendo su superficie. Sin embargo, nos da una idea muy aproximada a la hora de ejecutar un programa del porqué de los errores que podemos tener en nuestro código, ¡y habrá para un rato!

Y es que cuando programamos, muchas veces pensamos como un ser humano, cuando lo suyo sería pensar como un ordenador, por ello es importante saber un poco acerca del procesador, el cerebro del ordenador, puesto la lógica que a nuestro parecer parecería algo muy obvio, el ordenador necesitaría otra manera de comprender el mismo concepto, nuestros cerebros no están hechos del mismo material.