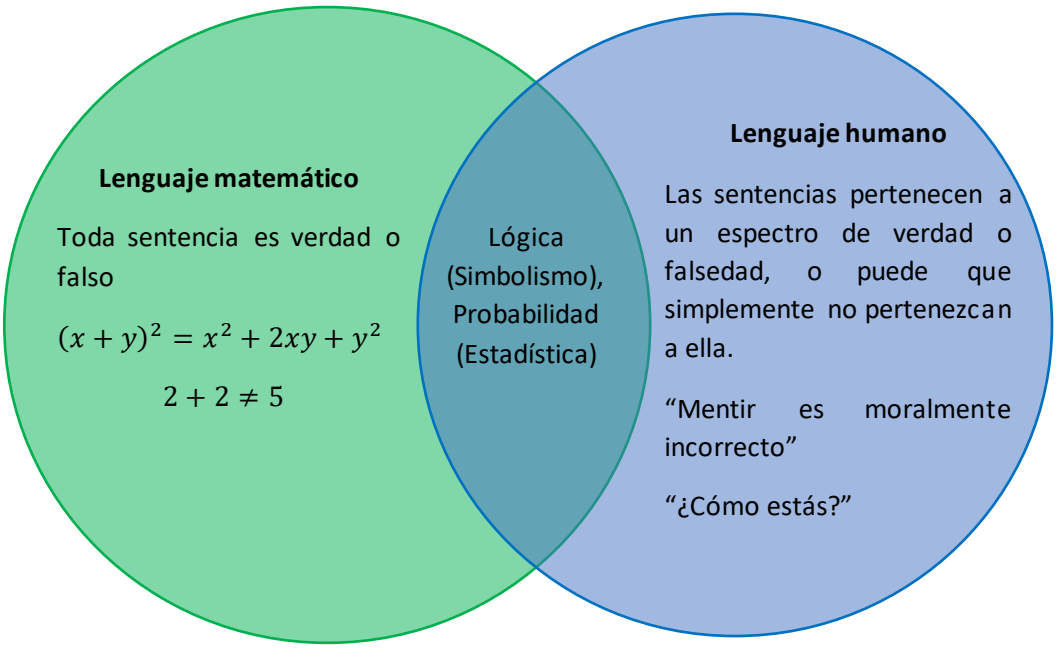
Interludios

Álgebra booleana

No sé si os habrá pasado esta situación en la cual están reflexionando sobre la vida, a tal punto que empezamos a cuestionar sobre los principios básicos de la realidad percibimos, y de repente se nos ocurren problemas retóricos como: ¿qué es matemática?, ¿qué es el lenguaje humano?

Por un lado, tenemos un sistema que nos permite describir el mundo objetivo (el fuego me causa dolor) o intersubjetivo (mentir es moralmente incorrecto) en donde sería incorrecto decir que toda sentencia es verdad o falso. Mientras que por el otro extremo del espectro tenemos un sistema que describe a un mundo universal (x + y ¿ y +x, en la cual toda sentencia es una verdad o falsedad). En la intersección del lenguaje matemático y humano nos encontramos con el lenguaje lógico o lenguaje probabilístico, la cual hereda las propiedades matemáticas de universalidad, y tiene mayor poder expresivo en el sentido de que puede trabajor más allá de números o variables, sino que también con signos que representan la realidad humana o argumentos (todo organismo vivo tiene ancestro, todo ser humano es un organismo, María es una ser humana, María tiene ancestro).



¿Qué tiene esto que ver con el tema de hoy? Casi nada. Si no eres alguien que está pasando por una crisis existencial y tienes obligaciones cotidianas como aprender a programar o cómo ser felices, entonces podemos dejar de un lado la diapositiva anterior. Hoy vamos a introducirnos de manera práctica el tema de álgebra booleana, y enfatizo de manera práctica debido a que si nos fuésemos de manera profunda conceptual este interludio sería innecesariamente largo.

De manera breve, álgebra booleana permite el estudio sistemático del pensamiento lógico. En el contexto de programación, es importante saber cómo definir variables y expresiones booleanas ya que esto será relevante para resolver problemas computacionales. Como veremos pronto, las expresiones booleanas nos sirven para definir estructuras de control y bucles ya que actuarán como semáforos que determinan qué bloque de código será ejectuado en qué circunstancias

1. Toda variable y expresión booleana es evaluada ya sea a verdadero y falso. Los operadores que usan son <, >, not, ==, /=
    a. Atención, == no es lo mismo que =
2. Podemos unir dos expresiones booleanas con los operadores AND y OR.
    a. AND es como un puente unido
    b. OR es como un puente paralelo

Referencias:

Entrevistas y Podcast del Profesor de Lingüística Noam Chomsky en Google y con Lex Fridman respectivamente https://lexfridman.com/noam-chomsky

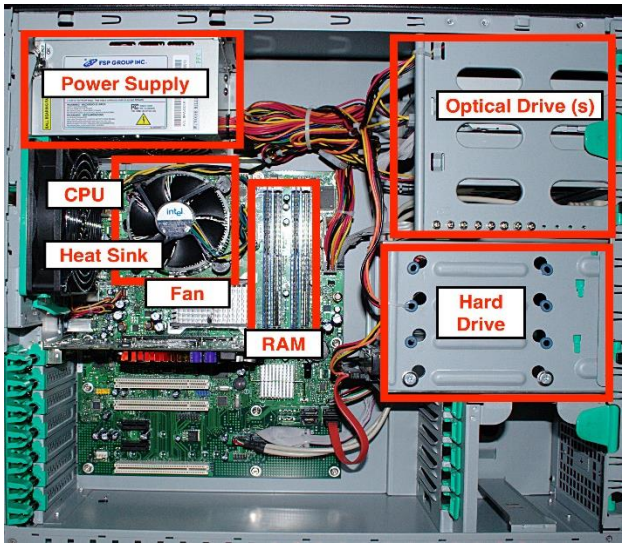
```
boolean miPrimeraVariableBool = true;
        boolean miSegundaVariableBool = false;
        boolean negacion = !true;    // voltear el valor
como una moneda
        boolean negacion2 = !false;  // true
        boolean desigualdad = 5 > 3; // true
        boolean desigualdad2 = !(-5 > 3);   // true
         boolean igualdad = 2 + 2 == 4; // = significa
definir,
                                       // == significa
igual que
         boolean igualdad2 = 2*5 - 9*7 != 9*7 - 2*5; //
true
        // CONJUNCIÓN

        // DISYUNCIÓN
```

Pila y montículo: punteros y referencias a la memoria

Muy buenos días a todos.

En el presente interludio vamos a introducirnos a la pila y el montículo.

Si nuestra computadora es análogo a un cuerpo humano y empezamos a observar sus órganos interiores, veremos la siguiente esquematización. Podemos ver un ventilador debido a que las computadoras no sudan.



https://www.youtube.com/watch?v=-uleG_Vecis

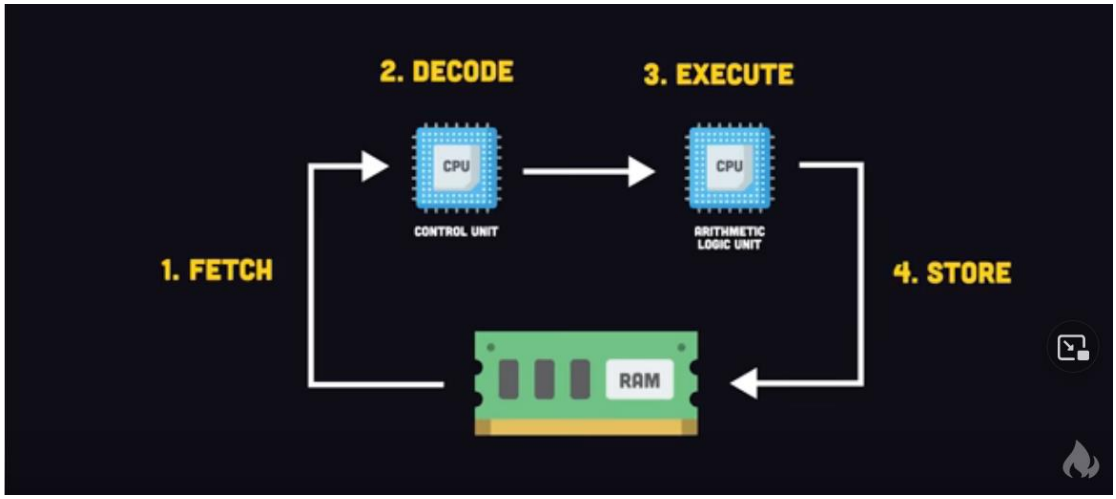
Interludio: Punteros y referencias a la memoria

Buenas a todos.

En este corto interludio quisiera aclararte de manera más profunda la diferencia entre variables primitivos y referenciales. Recordemos que la distinción entre ambos aplica para varios lenguajes de programación, aunque en Python no es explícito definir el tipo de variable como en Java.

Entonces, para empezar quisiera recordar qué es lo que ocurre cuando escribimos algunas líneas de código. Cuando presionamos ejecutar, en realidad lo que está pasando es que estamos compilando el código debido a que la computadora no entiende el lenguaje Python o Java. Por ende, un programa detrás de cámaras nos ayuda a traducir el código desde un lenguaje programación a lenguaje primario.

Diapositiva adicional:



Dentro de la RAM, tenemos la siguiente representación ABSTRACTA:

Según (Patterson & Hennessy, 2007) en realidad el proceso es más sutil. Hace bastante tiempo, cuando Python y Java no existían, en realidad el lenguaje que los programadores usaban se llamaba lenguaje ensamblador. La sintaxis era más complicado, por ejemplo, una suma entre dos variables era la siguiente:

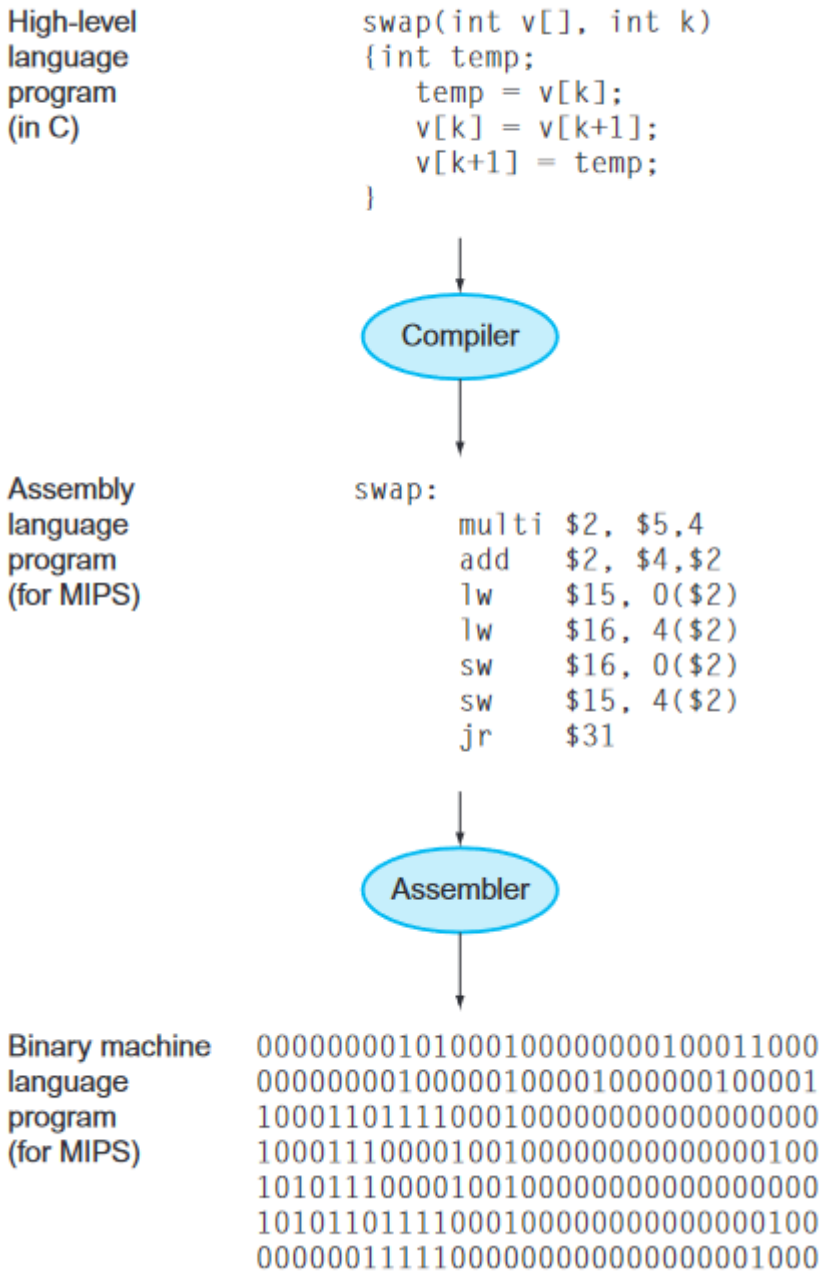
```
add A,B
```

Una función se vería de esta manera:

```
swap:
        multi $2, $5,4
        add   $2, $4,$2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

En ese entonces, el ensamblador (disculpa la redundancia) era el programa que estaba encargado de traducir el lenguaje a lenguaje binario. Debido a la dificultad para un programador escribir código de esta manera, otros lenguajes, llamadas lenguajes de programación de nivel superior (Python y Java) fueron creados. Estos lenguajes de nivel superior tienen un sintaxis más fácil de aprender. Por ejemplo, para sumar usamos un símbolo +. El compilador, un programa más avanzado que el ensemblador fue creado, el cual traduce el lenguaje de programación de nivel superior a lenguaje binario entendido por la máquina. Por ende, según (Patterson & Hennessy, 2007), el proceso completo de cómo el compilador funciona es el siguiente:

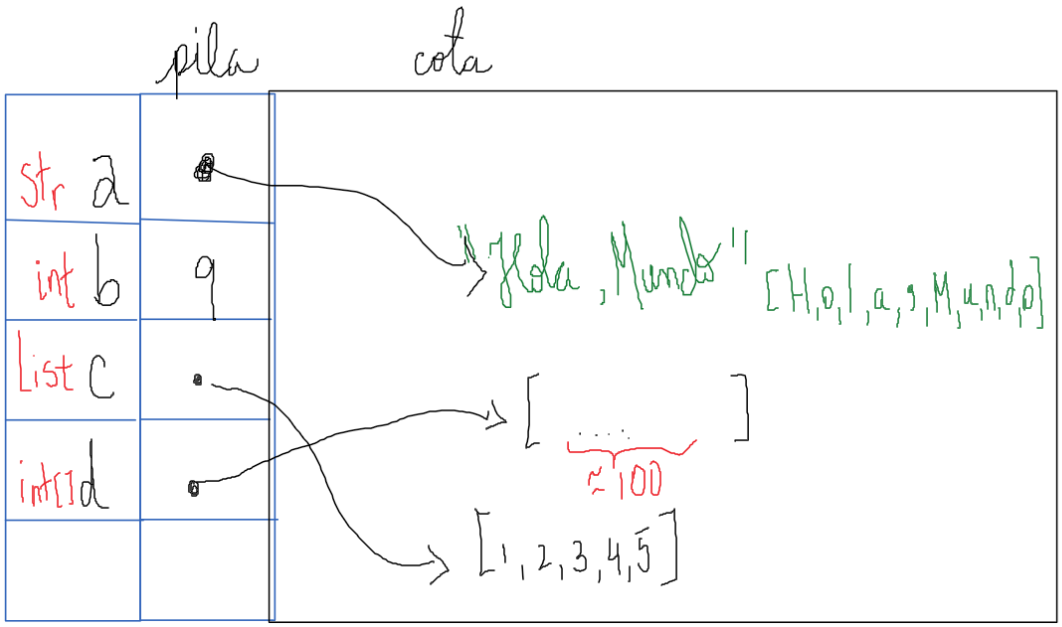
Observemos que lo que el compilador hace



Claro que este diagrama tampoco describe precisamente el flujo de compilación para todos lenguajes de programación. Puede que ya no exista el ensamblador, por eso en algunos diagramas solo aparece el compilador.

Cuando la máquina esté ejecutando las instrucciones, esta estará alocando variables en la memoria. Nosotros podemos suponer que la memoria RAM trabaja con el disco duro con el fin de alocar variables de un lado a otro. Esto es porque recordemos qué son instrucciones fundamentalmente: es simplemente manipulación de variables (Recuerda del prólogo).

```java
String a = "Hola, Mundo.";
int b = 19;
int[] c = new int[] {1,2,3,4,5};
List <Integer> d = new ArrayList<Integer>();
```



La asignación de variables ocurre en la pila y cola. Los que son alocados en la pila son las variables primitivas, mientras los que son alocadas a la cola son referenciales. Si estás sumando dos variables con el fin de referirte a ella con una tercera variable, entonces esta asignación y reasignación ocurre en la pila.

```java
int miNum1 = 8;
int miNum2 = 9;
int miNum3 = miNum1 + miNum2;
```

Puedes tratar de generalizar esto con las variables definidas en tus funciones, sean estas primitivas o referenciales.

Dentro de la RAM, tenemos la siguiente representación ABSTRACTA:

LA PILA Y LA COLA NO SON HARDWARE

PILA: Variables primitivas

COLA: Variables referenciales

En la pila están guardadas los punteros de variables referenciales

```java
int[] miArreglo = new int[] {1,2,3,4,5};
```

Diferentes variables pueden apuntar a una misma variable referencial

```java
String miCadena = "Hola, Mundo";
String miCadenaSobra = miCadena;
```

Valores sin punteros son reciclados

Intenta "$Hola, Mundo$" == "$Hola, Mundo$" en Java

StackOverflow

Desafío personal: optimizar el uso de memoria

Cabe recalcar aquí una posible confusión que yo cometí. En primer lugar existen otras pilas y colas que son estructuras de datos, o sea, son variables referenciales (otras versiones de listas), sin embargo, aquí no nos estamos refiriéndonos a ellas. La pila y cola aquí tampoco se refiere a hardware de la máquina. Es simplemente una representación de la memoria.

Algunos de los errores que ocurren en java cuando imprimes un arreglo es que en realidad estás imprimiendo su ubicación en la memoria.

```java
int[] test1 = {2, 3, 4, 5, 2};
System.out.println(test1);
```

[I@279f2327

Además, debido a esta manera de manejar memoria, podemos aprender de manera más detallada la diferencia entre lista y arreglo. Una lista asigna un tamaño significantemente grande (creo que ~100 celdas) cuando lo inicializas. No tiene tamaño infinito. Por eso puedes agregar elementos usando la función predefinida .append(). El arreglo asigna memoria de manera determinada por el usuario.

En VS Code, si quieres puedes descargarte una herramienta llamada Binary Viewer para supuestamente ver el lenguaje binario de un archivo Java compilado:

```
00000080        0a 00 10 00 11 07 00 12                    0c 00 13 00 14 01
00000090        6a 61 76 61 2f 69 6f 2f                    50 72 69 6e 74 53
000000a0        65 61 6d 01 00 05 70 72                    69 6e 74 01 00 04
000000b0        29 56 07 00 16 01 00 15                    70 72 6f 79 65 63
000000c0        73 4a 61 76 61 2f 66 75                    6e 63 69 6f 6e 01
000000d0        43 6f 64 65 01 00 0f 4c                    69 6e 65 4e 75 6d
```

Claro, hoy no hemos explicado qué ocurre detrás de las operaciones con variables: suma, resta, multiplicación, división, exponenciación, raíz cuadrada, chequeo de igualdad, mayor que, menor que. Sin embargo, para ser melodramático, te puedo decir lo siguiente

En resumen:

La moraleja de este interludio es que tratemos de *saber y entender.*

Las variables primitivas y referenciales residen en la pila y cola de la memoria durante asignación de variables y ejecución de una función después de haber compilado.

Ejercicios:

¿Qué crees que va a ocurrir cuando una variable referencial es inaccesibl?

¿Qué crees que ocurre con los comentarios?

Yo no soy un experto en la arquitectura básica de la computadora, por ende, si te itneresa saber más o menos cómo funciona la computadora de un punto principal, te recomiendo leer Computer Organization And Design The Hardware/Software. Aquí habrán temas interesantes que conciernen al compilador, qué es el CPU, RAM,

Introducción a la Paradoja de Russell y el Teorema de la incompletitud de Gödel

Esto talvez suene cliché, pero si el genio de la lámpara apareciese a tu frente y te ofreciera 3 deseos, ¿qué deseos pedirás? Amor,

Objetivos

Al final de este interludio, espero que familiarices con el concepto e importancia de recursión a través de la Paradoja de Russell y Teorema de Incompletitud de Gödel.

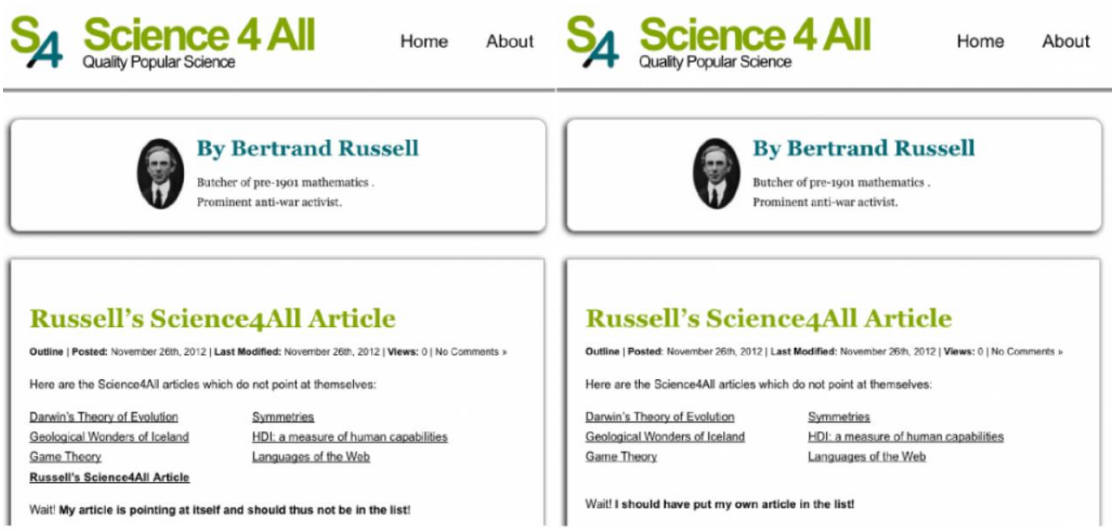
A principio del siglo XX, una serie de descubrimientos hicieron temblar al mundo matemático.

Bertrand Russell era un matemático quien estudiaba sets (una rama matemática). Un set es simplemente un agrupamiento de números según alguna regla especificada. Yo puedo construir un set de números pares, un set de número primos o un set de números de dos dígitos que sumen 8.

¡Activa el pensamiento recursivo! Es posible crear sets de sets. Por ejemplo, el set de los sets que contienen el número 2. Este set tendrá el set de primos, el set de pares, el set cuyos dígitos sumen 8.

Ahora, ¿qué pasa si te pregunto: genera el set que contengan los sets que no se contengan a sí mismos?

Si este ejemplo es un poco críptico, puedes pensar en siguiente ejemplo de Le Nguyen.



Lê Nguyên Hoang http://www.science4all.org/article/self-reference/

Aquí la analogía es que la página web es un set, mientras que los enlaces son los números que están dentro de este set (página) de acuerdo a ciertas reglas.

¡Qué trágico!

La paradoja anterior es como un mosquito; se ha vuelto omnipresente.

David Hilbert es un matemático quien tenía la ambición de usar lógica matemática para axiomatizar toda la realidad. *La montaña Chimborazo está cubierta de nieve. Los números 3, 4 y 5 forman una terna Pitagórica. Mi mascota tiene 4 patas.*

¡Activa el pensamiento recursivo! "Esta oración tiene 5 palabras"

Toda expresión lógica tiene que ser evaluada a verdadero o falso.

¿Pudo o no Hilbert cumplir con su sueño de axiomatizar toda la realidad a través de lógica?

No. Según el matemático Godel, todo sistema que consiste de axiomas lógicas no podrá representar toda la realidad, debido a que habrán paradojas.

Considera la expresión lógica: "Esta expresión será evaluada a falsa".

Obviamente el teorema en sí fue estudiada, formulada y comprobada de manera más rigurosa. Brevemente, Godel no ofreció el ejemplo anterior. Toda expresión lógica podía ser transformado/mapeado a un número único. Esto es análogo a la función de *hashCode()* aplicada en cadenas. Pero, Godel, siguiendo estas rigurosas reglas, formuló la expresión: "Esta expresión lógica no tiene su número Godel único".

https://www.youtube.com/watch?v=HeQX2HjkcNo Te

Teorema de Incompletitud de Gödel

La clave del pensamiento recursivo en estos dos casos es que en vez de pensar sobre el contenido de un sistema de reglas, en vez empiezas a referirte al sistema en sí.

Las mentiras de pinocho, completar misiones

¿Cuál es la moraleja de la historia? Los autor@s de estas caricaturas tienen que aprender un poco de pensamiento recursivo.

¿Qué le desearías? Bueno, aparte de desear deseos infinitos, podría intentar desear “un cumplas este deseo”