Semántica

1. ¿Qué es la semántica?

Es un conjunto de reglas para dar significado a los programas sintácticamente válidos. Describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático. Esto implica que para que un programa sea semánticamente válido debe serlo también sintácticamente. No así, el que sea sintácticamente válido implica necesariamente también lo sea semánticamente.

2. ¿Cuál es la utilidad de definir y conocer la semántica de un lenguaje? ¿Quiénes se benefician?

La definición de la semántica de un lenguaje de programación proporcionan mecanismos para que una persona o una computadora pueda decir, siendo el programa ya válido, lo que significa. Esto, a primera vista, implicaría que solo el programador se beneficia de esto. Lo cierto es que también estas reglas sirven para el desarrollo de compiladores e intérpretes, por lo que, si no se hiciera tanta énfasis como se hace, el desarrollo de éstos no sería posible.

3. ¿Qué tipos de semánticas existen?

Hay dos tipos:

- **Semántica estática**: no está relacionado con el significado del programa, está relacionado con las formas validas. Se la llama así porque el análisis para el chequeo puede hacerse durante la compilación o el interpretado. Generalmente las gramáticas sensibles al contexto resuelven los aspectos de la semántica estática.
- **Semántica dinámica**: es la que describe el efecto de ejecutar las diferentes construcciones en el lenguaje de programación. Su efecto se describe durante la ejecución del programa. Los programas solo se pueden ejecutar si son correctos para la sintáxis y para la semántica estática.

4. ¿Cómo se describe la semántica?

Algunas soluciones son:

- **Semántica axiomática**: considera al programa como una máquina de estados. Los constructores de un lenguajes de programación se formalizan describiendo como su ejecución provoca un cambio de estado.
- **Semántica denotacional**: cada frase en el lenguaje es interpretada como una denotación. Tales denotaciones a menudo son objetos matemáticos que habitan espacios matemáticos, pero no es un requerimiento que éstas deban serlo. Como una necesidad práctica, las denotaciones se describen usando alguna forma de notación matemática, la cual en turno puede ser formalizada como un metalenguaje denotativo.
- **Semántica operacional**: el significado de un programa se describe mediante otro lenguaje de bajo nivel implementado sobre una máquina abstracta. Los cambios que se producen en el estado de la máquina cuando se ejecuta una sentencia del lenguaje de programación definen su significado

5. ¿Cómo se procesa un lenguaje?

Por defecto, todas las maquinas entienden un lenguaje básico que es el ensamblador. En los inicios de la informática, ésta era la forma en que se programaba. Con los lenguajes de programación, aparece la abstracción del lenguaje ensamblador que permite que la programación se simplifique y no se pierda tiempo con el lenguaje de máquina. Pero para que esto sea posible es necesario un traductor de lenguaje de programacion x(como puede ser C o java) a ensamblador puro. Este traductor es lo que llamamos compilador o intérprete, según el lenguaje.

6. ¿Qué es un intérprete?

El intérprete lee, analiza, decodifica y ejecuta una a una las sentencias de un programa escrito en un lenguaje de programación. Algunos ejemplos son Lisp, Smalltalk, Basic, Python, etc. Por cada posible acción hay un subprograma que ejecuta esa acción. La interpretación se realiza llamando a estos subprogramas en la secuencia adecuada. Un intérprete ejecuta repetidamente la siguiente secuencia de acciones:

- Obtiene la próxima sentencia.
- Determina la acción a ejecutar.
- Ejecuta la acción.

7. ¿Qué es la compilación?

En la compilación, los programas escritos en un lenguaje de alto nivel se traducen a una versión en lenguaje de máquina antes de ser ejecutados. Si bien tienen varias etapas, por lo general se pueden resumir en éstas:

- Análisis
    - **Análisis léxico(Scanner)**: es el que lleva mas tiempo ya que hace el análisis a nivel de palabra. Divide el programa es sus elementos constitutivos: identificadores, delimitadores, símbolos especiales, números, palabras clave, delimitadores, comentarios, etc. Analiza el tipo de cada token, filtra comentarios y separadores(espacios en blanco, tabulaciones, etc.), genera errores si la entrada no coincide con ninguna categoría léxica, convierte a representación interna los números en punto fijo o punto flotante, pone los identificadores en la tabla de símbolos y reemplaza cada símbolo por su entrada en la tabla. El resultado de éste paso será el descubrimiento de los items léxicos o tokens.
    - **Análisis sintáctico(Parser)**: el análisis se realiza a nivel de sentencia. Se identifican las estructuras, sentencias, declaraciones, expresiones, etc. ayudándose con los tokens. El analizador sintáctico se alterna con el análisis semántico. Usualmente se utilizan técnicas basadas en gramáticas formales. Aplica una gramática para construir el árbol sintáctico del programa.
    - **Análisis semántico(Semántica estática)**: es la fase más importante. Las estructuras sintácticas reconocidas por el analizador sintáctico son procesadas y la estructura del código ejecutable toma forma. Se realiza la comprobación de tipos; se agrega la información implícita (variables no declaradas); se agrega a la tabla de símbolos los descriptores de tipos, etc. a la vez que se hacen consultas para realizar comprobaciones y se hacen las comprobaciones de nombres. Es el nexo entre el análisis y la síntesis.
    - **Generación de código intermedio**: debe de ser fácil de producir y traducir al programa objeto.
- Síntesis
    - **Optimización del código**: en esta etapa se optimiza el programa objeto generado.
    - **Generación del código**: En esta etapa se construye el programa ejecutable. Se genera el código necesario y si hay traducción separada de módulos, es en esta etapa cuando se linkedita.

8. Intérprete vs compilador

9.

	Intérprete	**Compilador**
¿Cómo se ejecuta?	Ejecuta el programa de entrada directamente.	Produce un programa equivalente en lenguaje objeto.
¿Qué orden ejecuta?	Sigue el orden lógico de ejecución.	Sigue el orden físico de las sentencias.

9.

	Intérprete	Compilador
Tiempo de ejecución	Por cada sentencia se realiza el proceso de decodificación para determinar las operaciones a ejecutar y sus operandos. Si la sentencia está en un proceso iterativo, se realizará la tarea tantas veces como sea requerido. Esto podría afectar la velocidad del proceso	No repetir lazos, se decodifica una sola vez.
Eficiencia	Más lento en ejecución.	Más rápido desde el punto de vista del hardware.
Espacio ocupado	Ocupa menos espacio, cada sentencia se deja en la forma original.	Una sentencia puede ocupar cientos de sentencias de máquina.
Detección de errores	Las sentencias del código fuente pueden ser relacionadas directamente con la que se esta ejecutando.	Cualquier referencia al código fuente se pierde en el código objeto.

¿Cuál es mejor?

Los compiladores e intérpretes tienen muchas diferencias entre sí, ventajas y desventajas el uno con respecto al otro. El truco está en hacer una combinación de ambas técnicas.

Algunos ambientes de programación contienen las dos versiones (interpretación y compilación), lo que permite utilizar el intérprete en la etapa de desarrollo, facilitando el diagnóstico de errores, y luego de que el programa haya sido validado es compilado para generar código mas eficiente.
Otra forma es la traducción a un código intermedio que luego se interpretará. En este caso, se genera código portable, es decir, código fácil de transferir a diferentes máquinas.