1) La semántica define el significado de los símbolos, palabras y frases de un lenguaje.

2)

a) Traducir el código fuente a un código de máquina, en el proceso también puede haber optimizaciones.

b) Tiene 2 partes, análisis y síntesis:

Análisis:

Análisis léxico (Scanner): análisis a nivel de palabra. Genera los tokens, dividiendo el código en elementos constitutivos y asignándoles el tipo. Además, genera errores si la entrada no coincide con ninguna regla léxica.

Análisis sintáctico (Parser): analiza las sentencias y, con gramáticas formales y sus respectivos árboles sintácticos, determina las estructuras, declaraciones, expresiones, etc.

Análisis semántico (semántica estática): Analiza las estructuras sintácticas obtenidas. Se realiza la comprobación de tipos y de nombres.

Síntesis: se crea el ejecutable creando el código máquina. Opcionalmente se hace una optimización del código.

c) La semántica entra en el análisis de la semántica estática, y determina el significado de las estructuras sintácticas que obtuvimos en los pasos anteriores.

3) No, no es lo mismo. La interpretación, cuando se ejecuta un programa, sigue el orden lógico y va leyendo, analizando, decodificando y ejecutando línea por línea con un subprograma. En cambio, la compilación crea otro archivo con el código pasado a código máquina, y ese pasaje se realiza siguiendo el orden físico. Una vez generado el código máquina se lo ejecuta.

Ventajas intérprete:

* Ocupa menos espacio porque no requiere crear otro archivo traducido.
* En la detección de errores, las sentencias del código fuente se pueden relacionar con la que se está ejecutando.

Ventajas compilador:

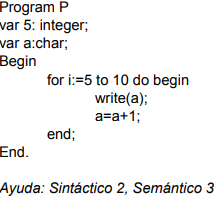
* Al no seguir un orden lógico, es más eficiente a la hora de evaluar loops.
* Más rápido en la ejecución.
* No tiene que decodificar cada vez que se ejecuta.

4) Un error sintáctico depende de la estructura de la sentencia que estamos analizando, en cambio, un error semántico depende del cumplimiento de una forma válida, es decir, de reglas que definen el significado de aquellas sentencias que cumplen con la estructura sintáctica.

En el lenguaje C:

La sentencia int a := 10 tendría un error sintáctico, porque la asignación se realiza con = , no con :=

La sentencia int a = ‘hola’ sería un error semántico, porque si bien la estructura sintáctica se cumple, no podemos asignarle una cadena de texto a una variable de tipo integer.

5) Pascal:

1

2

3

4

5

6

7

8

9

Líneas:

1. Error sintáctico, la línea no termina con “;”

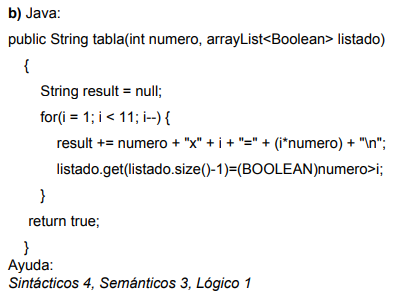
2. Error sintáctico, no cumple con la regla de definición de variables, porque 5 no es una posible variable

3. Error sintáctico, var ya fue escrito antes, no lo puede volver a escribir.

5. Error semántico, la variable índice “i” no está definida.

6. Error semántico, intenta imprimir una variable a la que no se le asignó un valor.

7. Error sintáctico y semántico, intenta hacer una asignación con “=” en vez de “:=”. Semántico porque intenta sumar “a” con 1, cuando el valor de a no está definido.



1

2

3

4

5

6

7

8

9

Líneas:

1. Error semántico: arrayList no existe, es “ArrayList”, clase no encontrada.

4. Error lógico: crea un bucle infinito

4. Error semántico: no encuentra la variable utilizada

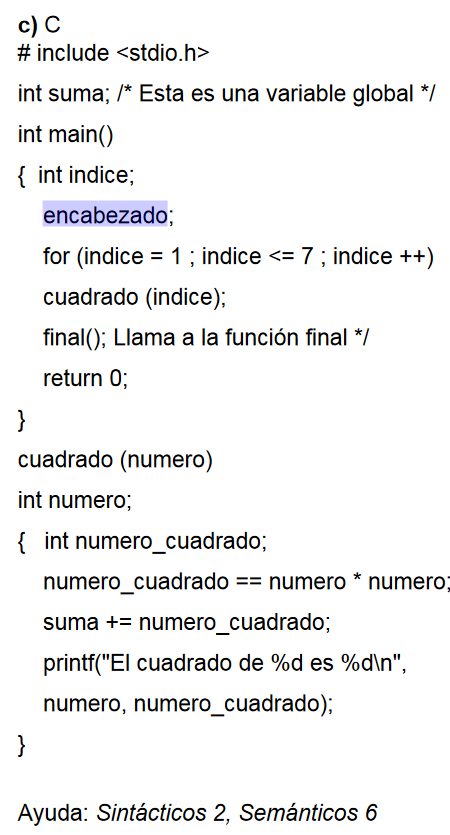
6. Error semántico: no se le puede asignar a un valor.

6. Error semántico: BOOLEAN no es un tipo válido

6. Error lógico: Si listado.size() fuera 0 y se le resta uno se iría del limite.

6. Error semántico: Si BOOLEAN estuviera bien escrito, intentaría convertir la variable “numero” a boolean, lo cual sería un problema de tipos incompatibles.

8. Error semántico: El tipo que devuelve es diferente al de la función



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

5: error semántico, “encabezado” no existe

7: semántico, llama una función que no existe

8: semántico por función final() inexistente y sintáctico por no abrir comentario

11: semántico, invoca a una función inexistente y sintáctico por que falta “;” o abrir “{” si es método y agregar el void

//esto pienso yo

12: semántico, no se puede crear una variable con el nombre que se pasa por parámetro

//esto pusieron los chicos

13: error sintáctico abre llave sin ninguna sentencia que requiera llaves

(sacamos línea 13)

14: semántico, usa variable que se declaró en una línea con error

15: semántico, usa variable que se declaró en una línea con error

18: sintáctico, cierra llave de la puta nada

Semántico= 6

Sintáctico=4

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

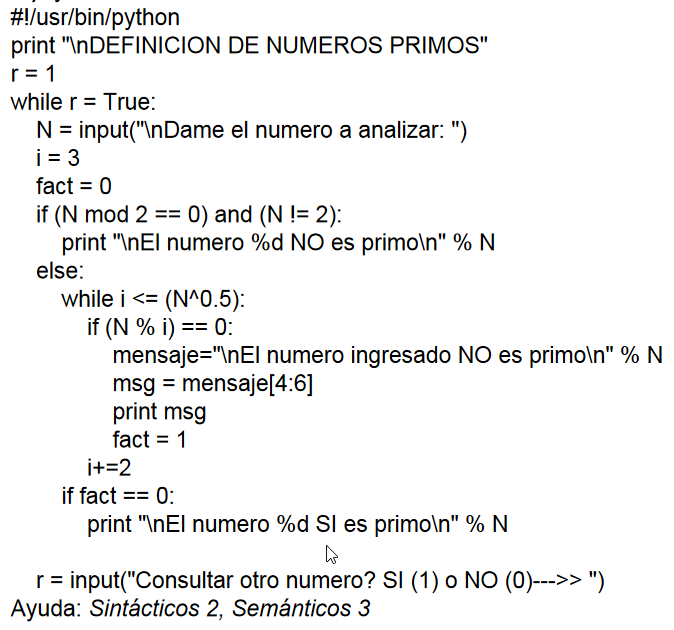
17

18

19

20

21

4: sintactico, debería ser == (además creo que debería ser = 1 porque sino no entra)

8: sintactico, no existe el operador mod, es %

11: semántico, el ^no admite float (creo que el ^ es \*\* en python, no sería esto sintáctico?)

13: semántico, no hay nada para % N (esto no se por que es un error, no entiendo si no anda N o como)

Semánticos = 2

Sintácticos = 2

1

2

3

4

5

6

7

8

9

10

11

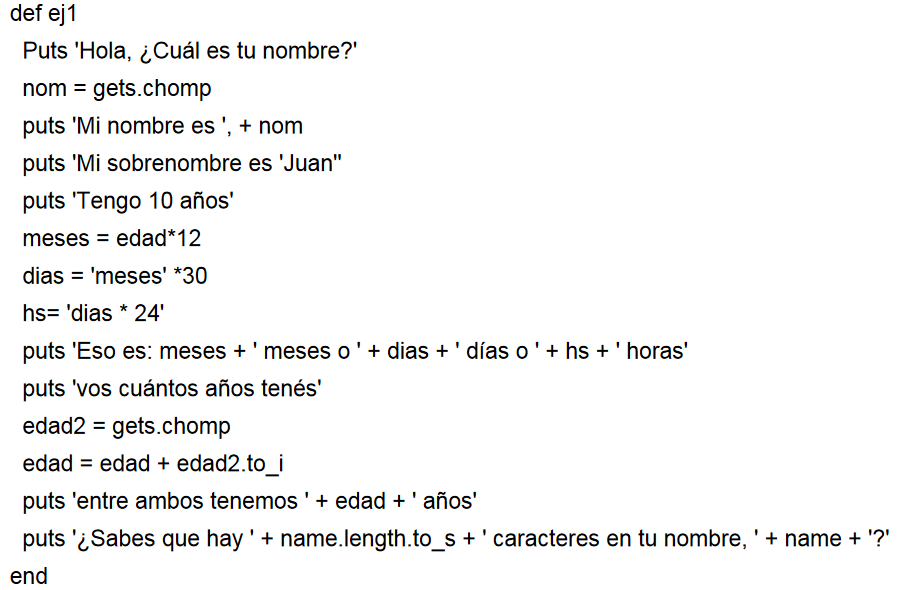
12

13

14

15

16



2: semántico, busca el método Puts y no lo encuentra

5: sintactico, cierra las comillas por lo que espera un concatenador, y recibe un método

7: semántico, no encuentra la variable edad

10: 4 sintacticos y 1 semantico…. Por ahora

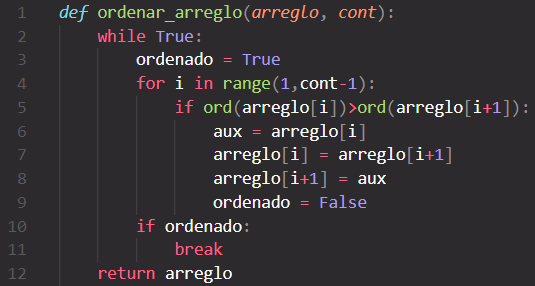
14: semántico, intenta sumar strings y integers

15: semántico, busca la variable name y no la encuentra

Semántico = 4

Sintactico = ~985

5)



En la definición no se pasan argumentos por referencia o por valor como en pascal, ya que son todos por valor si o si, por lo que se usa un return del argumento que hay que devolver

Python a diferencia de pascal no requiere una sección de definición de variables (gracias a ser interpretado) las variables se asignan dinámicamente

Como en Python no existe el do while o repeat until, utilizamos un while true con un break cuando se cumpla la condición pedida, de esta manera siempre se ingresa al while aunque sea una vez y con el break se corta el loop

En el for Python necesita un rango o una variable indexeable, por lo que ponemos un rango desde 0 hasta el valor indicado con la función range, que retorna una lista de números enteros entre las cotas pasadas

Desde la linea 5 a la 9 las únicas diferencias son sintácticas (=, :=) y los “;” ,

Luego con un if se simula el funcionamiento del do while

Y se retorna el valor

6) ruby

self: en cada momento de la ejecucion del programa hay un único self, ya que es una variable especial que siempre va a tomar como valor el objeto dueño del codigo que actualmente se está ejecutando. Al inicio del programa en el “main” self es una instancia de la clase Object de ruby.

nil: es un valor que expresa la falta de un objeto (hace referencia la clase abstracta NilClass), se usa para marcar las variables que el garbaje collector tiene que destruir

7)

null: es un valor vacío, que tiene que ser asignado a las variables

undefined: es el valor por defecto que se pone a variables que se declaran, pero no se les define (todavía) un valor

8)

C: tiene dos usos, si el break se encuentra en un ciclo, éste se termina y el control de la ejecucion pasa a la siguiente instrucción después del loop. Su otro uso es para terminar los “case” en un switch

PHP: igual que en C vamos

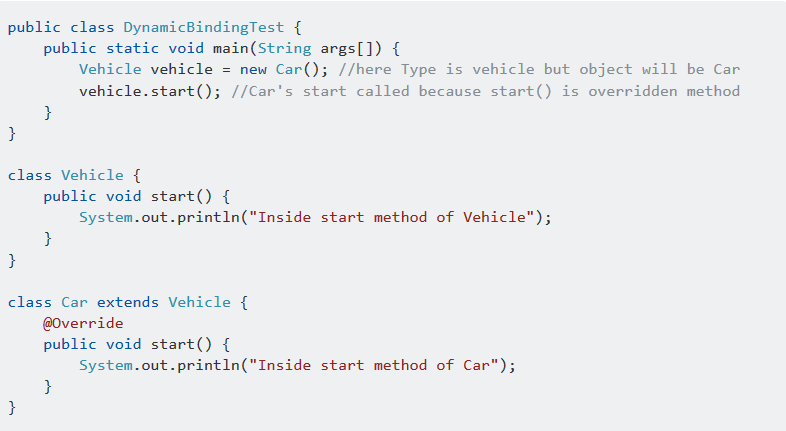
Ruby: igual que en C pero sin lo del switch

Javascript: igual que en C pero también se puede usar para terminar cualquier bloque de codigo pasándoselo, ej “break loop1”;

9) la ligadura o enlace se le llama al proceso por el cual a un elemento se le asigna su tipo o propiedades, el enlace dinámico se hace en tiempo de ejecución mientras que el estático se hace en tiempo de compilación



En este ejemplo al invocar el método “sort” se ejecuta el primer método (y no el sobreescrito) porque “c” es de tipo Collection, ya que se enlazo en tiempo de compilación, entonces si bien el objeto c es un HashSet el tipo de la variable sigue siendo Collection por lo que entra al método que recibe un parámetro de tipo Collection



huggi

En este otro ejemplo, se crea una variable “vehicle” de tipo “Vehicle” (asignado en compilación) pero al ejecutar el método “start”, como se está en tiempo de ejecucion ya se “sabe” que el objeto en realidad es un “Car” por mas que su tipo sea “Vechicle” por lo que se ejecuta el método “start” de la clase “Car” en lugar del método “start” de la clase “Vechicle”

alejo

En este otro ejemplo, se crea una variable “vehicle” de tipo “Vehicle” (asignado en compilación), pero como el método “start” se evalua en tiempo de ejecución, la variable “vehicle” tiene asignada la clase “Car”, por lo que el método que se ejecuta es el definido en esta.