Práctica 3

1) La semántica se refiere al significado/interpretación de las instrucciones en un programa.

2) Compilar significa traducir el código fuente de un ~~programa~~ lenguaje de alto nivel a un lenguaje de bajo nivel que puede ser comprendido y ejecutado x una computadora.

Los pasos para compilar un programa son divididos en 2 etapas.

- Etapa de análisis de código fuente: incluye ciertas etapas
  1) Análisis léxico: divide el cód. en tokens q' son componentes básicos como words, elimina comentarios y espacios en blanco.
  2) " sintáctico: verifica que los token coincidan con la sintaxis. y que cumpla con las reglas gramaticales.
  3) " semántico: cumpla con la semántica, tipos, asignaciones, uso adecuado de variables y llamadas a fx
- Etapa de síntesis: produce el cód. objeto. Se hace la optimizax, asignax de registros.

La semántica interviene en el análisis semántico, la importancia radica en que si no se cumplen las reglas semánticas, el programa no podrá ser ejecutado.

3) No, no es lo mismo. Diferencia)

	Intérprete	Compilador
Ejecución	Ejecuta el programa de entrada	Compila y ejecuta el cód. objeto.
Orden	Sigue orden lógico	Sigue órden físico.
Tiempo de ejecución	Lento	Rápido en el pto de vista de HW
Eficiencia	P/ c/ sentencia hace proc. de decodificax	Pasa x todas las sentencias
Espacio	- memoria.	
Errores.	Como uso el cód. fuente, es + fácil detectarlos.	Se pierde referencia p/ cód. fuente y cód. objeto.

4) • Error sintáctico: una instrux no sigue la sintáxis del lenguaje. Son detectados en el análisis semántico en la etapa de compilax.

ej: `if (num1 < num2 {` ↘ falta cierre de paréntesis.
`}`

• Error semántico: una instrux es gramaticalm correcta pero no tiene sentido en términos de lógica

ej: `int resultado = num1 / 0;` ↘ se divide x 0, no tiene sentido.

5) Sintácticos

a)
Linea 2: "var 5:integer", las variables deben comenzar con una letra
Linea 5: "for i:=5 to 10 do begin" la variable i nunca fue declarada.

Semántico.
Linea 4: "var a:char" se declara como char pero luego se le realiza una operación de suma
Linea 7: "a=a+1" la operación de asignar es ":="

b) Sintáctico.
Linea 0: el nombre de la clase no está declarado.
Linea 1: "arrayList" debe cambiarse a "ArrayList"
Linea 4: variable i no declarada.
Linea 4: i-- debería ser i++ porque nunca llegará a 11

Semántico.
Linea 3: result debería inicializarse como result="" porque sino no podré concatenar.
Linea 4: i-- debe ser i++
Linea 6: quiere asignar un valor cuando se esta obteniendo un valor de la lista.
En caso de querer asignar algo sería "listado.get(listado.size()-1)i numero>1)"

Lógico
Linea 8: quiere retornar un booleano cuando el valor de retorno es String.

c) Sintáctico
- Linea 11: cuadrado() no especifica valor de retorno
- Debe incluirse la función antes de la primera llamada o sea sobre main()

Semántico: Linea 11 cuadrado() no incluye apertura de llave. {
Linea 14: "numero_cuadrado == número * numero"; en vez de hacer una asignax debe ser ":="
Linea 14: "numero_cuadrado no se declaró el tipo.
Linea 11: no declara el tipo del parámetro número.
Linea 11: no se declara tipo de retorno.
Linea 12: numero se declara dos veces.

d) Sintácticos: Linea 2: a print le falta "()"
Linea 8: idem linea 2

Semánticos: Linea 7: variable N no declarada.
Linea 7: mod no corresponde a Python, debería ser %.
Linea 10: p/ elevar valor debe usarse "**" y no "^"

e) Semánticos Linea 1 "puts" en minúscula.
Linea 3: comilla de más en 'Juan'
Linea 4: edad no se definió previa
Linea 5: concatena variable sin antes parsearla.

Ejercicio 5 II

Pascal

```
procedure ordenar_arreglo(var arreglo:arreglo_de_caracteres;
    cant: integer);
var
  i:integer; ordenando:boolean;
  aux:char;
begin
  repeat
    ordenando:=true;
    for i:=1 to cant-1 do
      if ord(arreglo[i]>ord(arreglo[i+1])
        then begin
          aux:=arreglo[i];
          arreglo[i]:=arreglo[i+1];
          arreglo[i+1]:=aux;
          ordenado:=false
          end;
  until ordenado.
end;
```

Java

```
public void ordenar_arreglo (ArrayList
    <char> arreglo, int cant){

  int i;
  boolean ordenado;
  char aux;

do
  ordenado=false;
  for (i=1; i<cant; i++){
    if(ord(arreglo[i]>ord(arreglo[i+1]){
      aux=arreglo[i];
      arreglo[i]=arreglo[i+1];
      arreglo[i+1]=aux;
      ordenado=false;
    }
  }
while (!ordenado);
}
```

Ejercicio 6.

Self variable que se refiere al objeto actual o receptor del mensaje. Su semantica es la de una referencia al objeto actual en el que se encuentra el código en ese momento. Se puede usar p/ llamar a los métodos de instancia del obj. actual o p/ accede a variables de instancia.

nil: variable predefinida que se refiere a un objeto único que representa la ausencia de valor o la no existencia de valor un objeto. Es el equivalente a null en otros lenguajes. Se usa en Ruby para representar el resultado de una operación que no tiene valor de retorno válido.

Ejercicio 7

null indica explícitamente que una variable o propiedad no tiene un valor, o sea, se ha asignado intencionalmente el valor null

undefined: indica que aún a una variable o propiedad no tiene asignado un valor

Ejercicio 8

C. usada en un bucle for/while/do-while para salir del bucle antes de cumplir condix de finalizar, dentro de la estructura "switch" p/ salir de la estructura dsp de dada condix / al ejecutar la sentencia break, la ejecux del programa se fr mueve fuera del bucle y continúa c/ la instrux siguiente.

PHP ídem C
JavaScript
Ruby.

Ejercicio 9 El binding establece una relax e/ un identificador y su valor dentro del programa. La importancia radica en que permite al programa referirse a los valores mediante nombres significativos p/ el programador, c/ mayor facilidad y comprensión.

- Binding estático: hecho en la etapa de compilax. El identificador se asocia c/ una direx de memoria fija y no cambiará durante la ejecux.

```
int a=5;
int b=a+3;
```

- Binding dinámico: hecho en tiempo de ejecux. El identificador se asocia c/ una direx de memoria q' puede cambiar durante la ejecux.

```
x= "hola";
def imprimir();
    print(x);

x= "adiós"
```