

Sintaxis del compilador Ruby

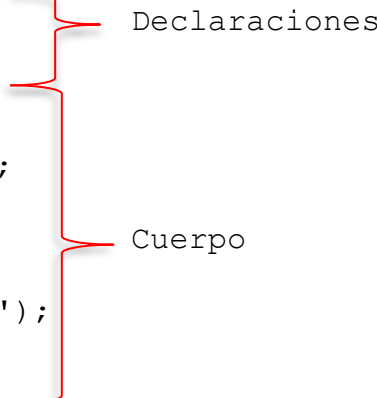
➤ Estructura general del programa.

La estructura de un programa en Ruby comienza por las declaraciones seguido del cuerpo principal.

DECLARACIONES () CUERPO ()

Ejemplo:

```
var Itver: string;
const PI=3.1416;
class carro
  def carro
    print('0 kilometros nuevo ');
  end;
...
  def ccarro
    print('1000 kilometros viejo');
  end;
end;
```



✓ Declaraciones.

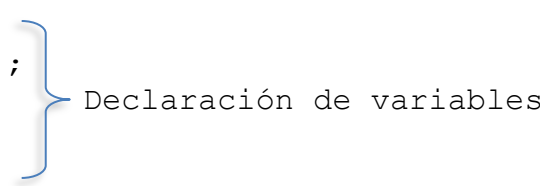
Las declaraciones pueden ser de variables, constantes, métodos o clases, un programa puede tener de 0 a n variables declaradas.

Se finaliza la declaración con el delimitador (;)

DECLARACIONES () → (D_VAR () | D_CONST () | D_METHOD () | D_CLASS ()) *

Ejemplo:

```
var
  arreglo: array[uno..dos];
  arreglo1: array['a'..'c'];
  vall: integer;
  var2: real;
  Itver: string;
```



PI=3.1416;  Declaración de constante

✓ Declaración de variables.

Comienza con la palabra reservada *var* seguido del identificador es decir, el nombre que va a llevar la variable seguido de 2 puntos (:), después el tipo de dato que va a ser la variable, finalizando con punto y coma (;) como delimitador.



Para declarar diferentes variables a la vez del mismo tipo de dato, se separan los identificadores con coma (,).

Si se va a declarar un arreglo se debe de seguir las instrucciones de cómo se declara un arreglo.

D_VAR() →

<pvar> ((<ident>|<variable_instancia>|<variable_local>|<variable_global>) ((<coma><ident>)*)
<dospuntos> (DGARRAY() | TIPO_DAT()) <puncoma>)

Ejemplo:

```
var  
cuidad, pais, nombre: string;  declaración de diferentes  
variables del mismo tipo  
arreglo: array['b'..'g'];  declaracion de un variable tipo arreglo
```

✓ Declaración de arreglos

Empieza con la palabra reservada *array* y entre corchetes se pone identificador, entero o carácter seguido de dos puntos y terminado con el mismo tipo de dato que se había puesto al inicio, después de cerrar los corchetes se pone el delimitador.

GARRAY() → ((<arreglo_uni>|<arreglo_bid>)+)<puncoma>
 |(<variable_instancia>((<formato>)+|(<formato_sub>)+)<puncoma>
 >)|(<variable_local>((<formato>)+|(<formato_sub>)+)<puncoma>)
 |(<variable_global>((<formato>)+|(<formato_sub>)+)<puncoma>)

Ejemplo:

arreglo1: array[1..100]; → arreglo tipo integer
 arreglo2: array[uno..dos]; → arreglo tipo string
 arreglo3: array['a'..'c']; → arreglo tipo char

✓ Declaración de constantes

Empieza con la palabra reservada *const* seguido del identificador seguido del signo de asignación después de un valor o una operación aritmética terminando con el delimitador.

D_CONST() →
 (<letramayus>)+<asignacion>(<caracter>|<cad>|<entero>|<decim>
 |<arreglo_uni>|<arreglo_bid>)<puncoma>
 |<constante><puncoma>

Ejemplo:


PI = 3.1416; → Declaración de una constante.


✓ Estructura de un método

Empieza con la palabra reservada *def* seguido el nombre del método, y si tiene de los parámetros, después sigue las instrucciones que es el cuerpo del método y finalizando con la palabra reservada *end* y el delimitador.

D_METHOD() → <pdef>((<letraminus>)+|<ident>)((PARAMET())?)
 ((INSTRUCCIONES())*)
 <pend><puncoma>


Ejemplo:

```
def metodo  Método sin parámetros
  conta=0;
  print 'Hola ruby';
  puts(conta);
end;
```

```
def metodo2 (val5: string)  Método con parámetros
  resultado = val5;
end;
```


✓ Parámetros de un método

Los parámetros de un método se ponen después de poner el nombre del método, el parámetro va entre paréntesis, un método puede aceptar más de un parámetro, en ese caso se separan con punto y coma; los parámetros pueden ser variables sencillas o arreglos.

PARAMET() 

<parenizq>(((<caracter>|<cad>|<entero>|<decim>|<llamado>|<variable_instancia>|<variable_local>|<variable_global>|<pfalse>|<ptrue>|<constante>) (<coma>?) +) | (((<ident>|<arreglo_uni>|<arreglo_bid>) (<coma>?) +))<parender>

Ejemplo:

```
def metodo3(perro: string)  Con un parámetro tipo string.
  print('sale al parque ' + perro);
end;
```

✓ Estructura de clases

Empieza con la palabra reservada *class* después sigue el nombre de la clase, dentro del cuerpo del método se puede tener método e instrucciones, y termina con la palabra reservada *end* y el delimitador.

```
D_CLASS() →  
<pclass>(((<letramayus>)+(<letraminus>)*)|<ident>)  
          (D_METHOD())+  
          ((OPERACIONES())*)  
<pend><puncoma>
```

Ejemplo:

```
class Honesta  
  def tarifatren(edad: real)  
    ...  
  end ;  
  def valornuevo(n: real)  
    ...  
  end;  
end;
```

✓ Cuerpo del programa

Se compone del conjunto de instrucciones:

```
CUERPO() → INSTRUCCIONES()
```

Ejemplo:

Mensajes() | Asigna() | Ciclos()

✓ Estructura de la salida a pantalla


Se puede utilizar la palabra reservada *puts* o *print* entre paréntesis o no ponemos aquello que queremos que se imprima y termina con el delimitador punto y coma.

MENSAJE () →

```
((<pputs>|<pprint>)((<parenizq>(<ident>|<cad>|<caracter>|<entero>|<decim>|<arreglo_uni>|<arreglo_bid>|<llamado>|<variable_instancia>|<variable_local>|<variable_global>|<pfalse>|<ptrue>|<constante>)(OPARI()(<ident>|<entero>|<decim>|<arreglo_uni>|<arreglo_bid>|<llamado>|<variable_instancia>|<variable_local>|<variable_global>|<pfalse>|<ptrue>|<constante>))*<parender>)|((<ident>|<cad>|<caracter>|<entero>|<decim>|<arreglo_uni>|<arreglo_bid>|<llamado>|<variable_instancia>|<variable_local>|<variable_global>|<pfalse>|<ptrue>|<constante>)(OPARI()(<ident>|<entero>|<decim>|<arreglo_uni>|<arreglo_bid>|<llamado>|<variable_instancia>|<variable_local>|<variable_global>|<pfalse>|<ptrue>|<constante>))*<parender>))*<puncoma>
```

Ejemplo:

puts("una cadena");  Salida en pantalla utilizando la palabra reservada puts

print (5*6+67);  Salida en pantalla utilizando la palabra reservada print

✓ Estructura de una asignación

Una estructura de asignación empieza con el nombre del identificador seguido del símbolo de asignación (=) y de ahí el valor que desea que sea asignado.

ASIGNA () →

```
<ident>(DGARRAY())?<asignacion>(((VALOR() (OPARI() VALOR()) *)<puncoma>)|((<parenizq>VALOR() (OPARI() VALOR()) *<parender>)<puncoma>))
```

Ejemplo:

```
contaduría_total= 58*45 (índice_total / utilidad_neta);
```

✓ Tipos de ciclos

Los ciclos que maneja Ruby son: WHILE, IF-ELSE, CASE y FOR:

CICLOS () → GWHILE () | GIFEND () | GCASE () | GFOR ()

Más adelante veremos ejemplos de cada ciclo en específico.

✓ Estructura del ciclo CASE

Empieza con la palabra reservada `case`, después le sigue el signo de asignación, seguido de la palabra reservada `when` seguida de una condición y/u operaciones, continúa `then` operaciones, asignaciones, valores, mensajes, etc. y, ésta rutina desde la palabra `when` se puede repetir, y finalizado se pone la palabra reservada `end` seguida del delimitador `(;)` y `puts leap`; al final de toda la instrucción `CASE`.

```
GCASE() → <pleap><asignacion><pcase>
(( <pwhen>(OPERACIONES()) <pthen>(OPERACIONES()) <puncoma>)+)
(( <pelse>(OPERACIONES()) <puncoma>)+) ?)
<pend><puncoma>
<pputs><pleap><puncoma>
```

Ejemplo:

```
leap = case
  when (r<7) then true;
  when (4>56) then false;
end;
puts leap;
```

✓ Estructura del ciclo WHILE

Empieza con la palabra reservada *while*, seguida de una condición a evaluar, después sigue el conjunto de instrucciones que componen el cuerpo del ciclo, se finaliza con la palabra reservada *end* y punto y coma como delimitador.

```
GWHILE () → <pwhile> CONDICION()  
              ((INSTRUCCIONES()) +)  
              <pend><puncoma>
```

Ejemplo:

```
while (caso_1 < caso_2)  
    puts("el caso_1 es mayor a el caso_2");  
end;
```

✓ Estructura de una condición

Una condición se compone de un valor o identificador el cual se compara con otro valor o identificador; puede realizarse varias comparaciones anidadas y los operadores usados para dichas comparaciones son los relacionales y/o lógicos.

```
CONDICION () →  
(VALOR () (OPREL () VALOR ()) ?) | (<parenizq>VALOR () (OPREL () VALOR ())  
?<parender> (OPELOG () <parenizq>VALOR () (OPREL () VALOR ()) ?  
<parender> *)
```

Ejemplo:

```
(capital < utilidad_neta)  
  
usuario > (c_1 < destinacion)
```


✓ Operadores relacionales

Con estos realizamos las operaciones de igualdades desigualdades, etc., en sí las operaciones relacionales que se deseen.

OPREL () →

<Igual> | <Desigualdad> | <MayQue> | <MenQue> | <MayIgual> | <MenIgual>
| <Pertenencia>

Ejemplo:

area >= área_pequena

✓ Operadores lógicos

Para realizar operaciones lógicas, tenemos estos operandores:

OPELOG () → <Not> | <And> | <Or> | <XOr>

Ejemplo:

expresion_1 and expresion _2

✓ Operadores aritméticos

Para realizar operaciones aritméticas como suma, resta y todas las básicas:

OPARI () → <Suma> | <Resta> | <Multip> | <Division> | <Mod>

Ejemplo:

ident1 + 5 – ident2 – 3/cont%2;

✓ Estructura del IF-END

Empieza con la palabra reservada *if* continua con una condición la cual es evaluada, le sigue el conjunto de instrucciones que componen el cuerpo de la condición y termina con la palabra reservada *end* y el delimitador; después de eso si se desea programar una instrucción *else*, escribimos la palabra reservada *else* y el conjunto de instrucciones que lo componen, más adelante se muestra la sintaxis de la instrucción *ELSE*.

```
GIFEND() → <pif> CONDICION()  
          ( ( INSTRUCCIONES() ) + )  
          <pend><puncoma>  
          ( ( GELSE() ) * )
```

✓ Estructura del ELSE

La estructura del identificador *else* es igual que la del *if*, un conjunto de instrucciones o condiciones seguidas de la palabra reservada *end* y el delimitador.

```
GELSE() → <pelse>  
          ( INSTRUCCIONES() ) +  
          <pend><puncoma>
```

Ejemplo:

```
if (edad > 18) → estructura if  
    puts("es mayor de edad"); → cuerpo del if  
else → estructura else  
    puts("no es mayor de edad"); → cuerpo del else  
end;
```

✓ Estructura FOR

GFOR() → <pfor><ident><Pertenencia>(<formato>|<formato_sub>)
(INSTRUCCIONES()) +
<pend><puncoma>