Documentación de Patito

Resumen general

Patito es un pequeño traductor e intérprete para un lenguaje propio, pensado como ejercicio académico. A partir de un archivo con extensión .pato o .patito, el sistema construye un árbol sintáctico, valida tipos, genera cuádruplos intermedios y finalmente los ejecuta mediante una máquina virtual.

El flujo de trabajo típico es el siguiente:

- run.py muestra únicamente el AST.
- run_semantico.py valida el programa y genera cuádruplos.
- run_cuadruplos.py --run archivo ejecuta el código en la VM.

La sintaxis soportada incluye declaraciones de programa, variables, funciones, condicionales, ciclos mientras, llamadas, escritura de salida y retornos. Se acepta también el uso de llaves { } para agrupar bloques, con punto y coma opcional al cerrarlos.

Funcionamiento general

Patito está organizado como un pequeño compilador por etapas:

1. **Análisis léxico**, implementado en scanner.py.
2. **Análisis sintáctico**, definido en parser.py, que produce un AST compacto.
3. **Análisis semántico**, que realiza validación de tipos, manejo de variables y generación de cuádruplos.
4. **Máquina virtual**, encargada de ejecutar esos cuádruplos.
5. **Scripts de ejemplo**, que facilitan la ejecución y depuración del lenguaje.

Componentes principales

scanner.py

Define todos los tokens del lenguaje y las palabras reservadas. Incluye reglas para:

- Identificadores.
- Constantes enteras, flotantes y cadenas.
- Operadores aritméticos y relacionales.
- Paréntesis, llaves, punto y coma, comas y dos puntos.

Maneja comentarios de una línea y de un bloque, y se encarga de normalizar escapes de cadenas. La función build_lexer() devuelve el analizador léxico de PLY.

parser.py

Organiza la gramática y construye el AST. Maneja precedencia de operadores y soporta variantes de declaración de funciones (estilo “largo” o parecido a C).
Genera nodos para asignaciones, expresiones, condicionales, ciclos, impresiones, retornos y llamadas.

En caso de error de sintaxis, se reporta la línea y el token problemático.

semantico.py

Define tipos básicos (entero, flotante, string, bool y nula) y estructuras para manejar variables, funciones y memoria virtual.

Estructuras incluidas

- VarTable y VarInfo, para registrar tipo y dirección virtual.
- FuncDirectory y FuncInfo, que almacenan firma, parámetros, variables locales y punto de entrada.
- VirtualMemory, que asigna direcciones virtuales separadas por segmento (global, local, temporal y constantes).

También contiene el **cubo semántico**, usado para validar operaciones entre tipos.

SemanticAnalyzerMin

Realiza la validación semántica básica:

- Declaración correcta de variables y tipos.
- Que las expresiones tengan sentido.
- Que condicionales y ciclos usen booleanos.

- Compatibilidad de tipos en asignaciones.

QuadGenerator

Extiende el analizador semántico para producir cuádruplos.
Administra pilas de operadores y operandos, tabla de constantes, directorio de funciones y memorias virtuales.

Genera cuádruplos para:

- Operaciones aritméticas y relacionales.
- Impresiones.
- Asignaciones.
- Condicionales (GOTOF/GOTO).
- Ciclos mientras.
- Llamadas a funciones (ERA, PARAM, GOSUB).
- Retornos y fin de funciones.

Incluye parcheo de saltos y manejo de funciones definidas después de su uso.

vm.py

Implementa la máquina virtual que ejecuta el conjunto de cuádruplos.
Cuenta con memoria global, local, temporal y de constantes. Cada llamada a función crea un nuevo frame de activación, con sus propios espacios de memoria.

La VM ejecuta operaciones aritméticas y relacionales, controla saltos, maneja llamadas y retornos, y valida accesos válidos a memoria.

Scripts incluidos

run.py

Construye el lexer y parser, procesa la entrada y muestra el AST generado.

run_semantico.py

Realiza el análisis sintáctico y semántico y genera cuádruplos, sin ejecutarlos.

run_cuadruplos.py

Genera cuádruplos e imprime:

- Variables globales.
- Funciones y sus firmas.
- Tabla de constantes.
- Lista completa de cuádruplos.

Con --run ejecuta el programa en la VM.

Programas de ejemplo

La carpeta ejemplos/ contiene pequeños programas de prueba: condicionales, ciclos, operaciones aritméticas y mensajes.

Diagrama y notas

En docs/diagramas.md hay un resumen visual del flujo de generación de cuádruplos y de la distribución de direcciones virtuales.

Tipos y restricciones

- Tipos permitidos: entero, flotante, string y bool (solo como resultado de comparaciones).
- No existen literales booleanos.
- Las variables deben declararse previamente y no pueden ser de tipo nula.
- Se permite la conversión entero → flotante.
- Las funciones pueden tener parámetros y valor de retorno.
- No hay arreglos ni estructuras compuestas.

Cuádruplos soportados

- Aritmética: +, -, *, /.
- Comparaciones: <, >, <=, >=, ==, !=.

- Asignación: =.
- Salida: PRINT.
- Control: GOTO, GOTOF.
- Funciones: ERA, PARAM, GOSUB, RET, ENDFUNC.

Memoria y direcciones virtuales

Los segmentos están organizados por tipo y por ámbito: global, temporal, constante y local. Cada segmento tiene bases distintas para cada tipo, y las direcciones se asignan de forma lineal.
Las constantes se internan: la misma constante utiliza la misma dirección.

Ejecución y pruebas

Tres formas principales de ejecución:

- Solo AST: python run.py archivo.pato.
- Semántico + cuádruplos: python run_semantico.py archivo.pato.
- Ejecución completa: python run_cuadruplos.py --run archivo.pato.

Extensión y depuración

Para extender el lenguaje pueden añadirse tokens, reglas de gramática, generación de cuádruplos y soporte en la VM.
Los archivos en ejemplos/ sirven como base para nuevas pruebas.

Limitaciones actuales

- Sin arreglos ni matrices.
- No existe lectura interactiva.
- Los booleanos sólo aparecen como resultado de comparaciones.
- Las operaciones numéricas dependen directamente de Python.

Guía simplificada

Patito toma un archivo .pato, tokeniza el contenido, construye el AST, valida el programa, genera cuádruplos y luego la VM los ejecuta uno a uno.
Cada cuádruplo representa una operación simple con hasta cuatro elementos.
Las direcciones virtuales sirven para organizar dónde se guarda cada valor en memoria.

Ejemplo de flujo interno

Un programa sencillo puede generar cuádruplos como:

- asignaciones a direcciones globales,
- operaciones intermedias que usan temporales,
- evaluaciones de condicionales con GOTOF,
- ciclos con saltos hacia atrás,
- y llamadas a funciones mediante ERA, GOSUB y RET.

Errores comunes

Patito detecta:

- variables no declaradas,
- tipos incompatibles,
- funciones duplicadas o no definidas,
- operaciones inválidas entre tipos.

Detalles técnicos adicionales

Léxico

Las expresiones regulares controlan identificadores, números, cadenas y operadores. Las cadenas admiten escapes. Se manejan correctamente saltos de línea para reportar errores.

Gramática y AST

Cada nodo del AST se representa como una tupla con un tag principal y sus componentes. La precedencia está definida para evitar ambigüedades en expresiones.

Semántica y memoria

El cubo semántico define los tipos permitidos para operaciones aritméticas y relacionales. La asignación de direcciones se separa por segmento y tipo.

Manejo de cuádruplos

La generación respeta el orden del programa: primero se registra un salto inicial, luego se procesan funciones, posteriormente el bloque principal, y finalmente se parchan los saltos de llamadas anticipadas.

Máquina virtual

Cada frame de función mantiene memoria local y temporal independiente. La VM interpreta cuádruplos en orden, gestionando saltos y accesos a memoria de forma controlada.

Herramientas de IA Utilizadas

- ¿qué herramientas usaste?

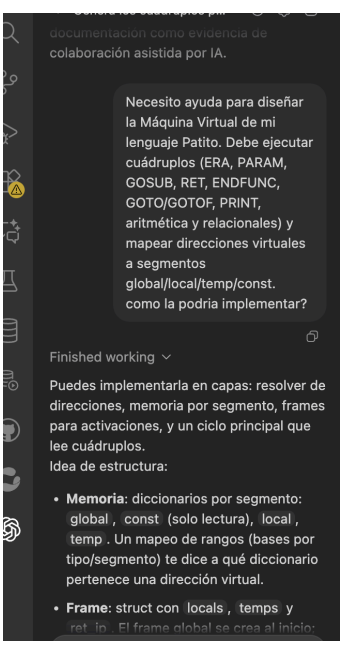
CODEX de OpenAI

- ¿qué partes del proyecto se vieron impactadas por este apoyo?

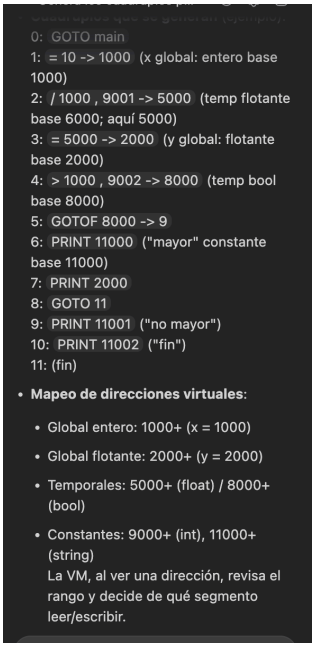
La máquina virtual, la memoria virtual y el generador de cuádruplos.

- ¿cómo o con qué entradas (prompts) solicitaste apoyo?

Prompts para la VM:



Prompts para cuadruplos y memoria:



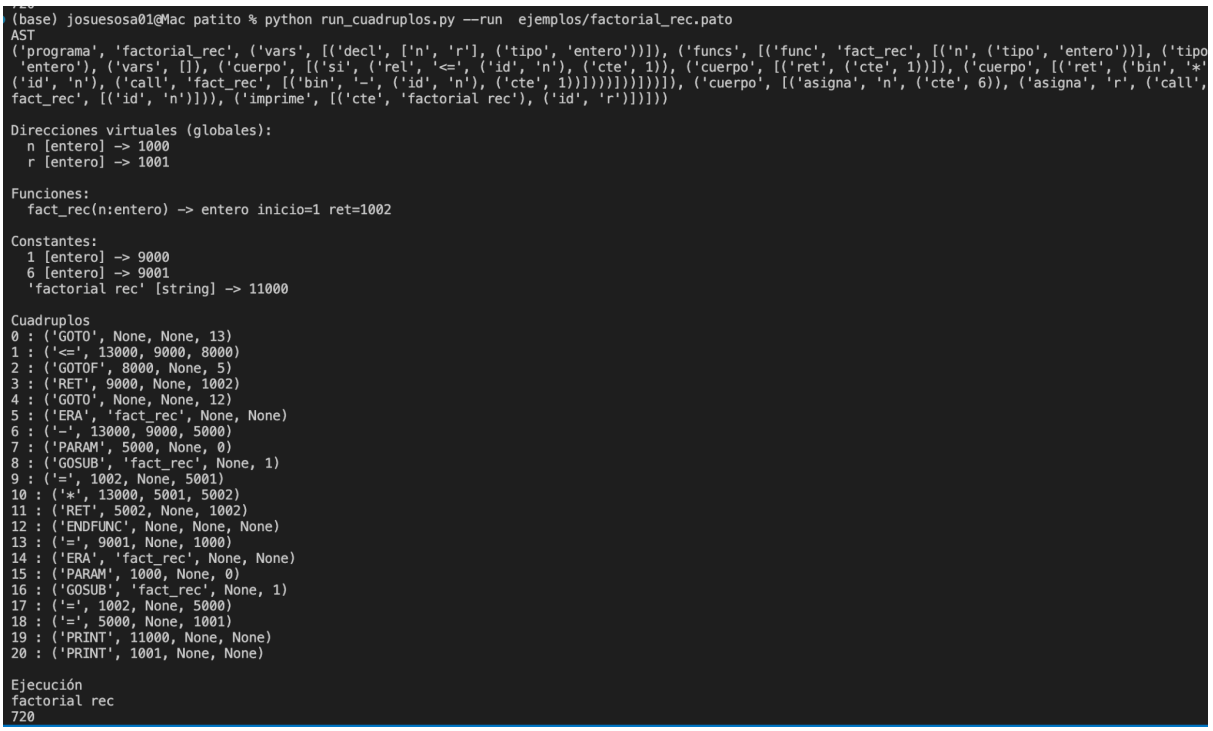
Test Cases:

1er Test:

```
programa factorial_rec;
vars:
  n, r : entero;

func fact_rec(n : entero) : entero
  si (n <= 1)
    ret 1;
  sino
    ret n * fact_rec(n - 1);
finf

inicio
  n = 6;
  r = fact_rec(n);
  escribe("factorial rec", r);
fin
```



```
(base) josuesosa01@Mac patito % python run_cuadruplos.py --run  ejemplos/factorial_rec.pato
AST
('programa', 'factorial_rec', ('vars', [('decl', ['n', 'r'], ('tipo', 'entero'))]), ('funcs', [('func', 'fact_rec', [('n', ('tipo', 'entero'))], ('tipo
 'entero'), ('vars', []), ('cuerpo', [('si', ('rel', '<=', ('id', 'n'), ('cte', 1)), ('cuerpo', [('ret', ('cte', 1))]), ('cuerpo', [('ret', ('bin', '*'
('id', 'n'), ('call', 'fact_rec', [('bin', '-', ('id', 'n'), ('cte', 1))])))])))])]), ('cuerpo', [('asigna', 'n', ('cte', 6)), ('asigna', 'r', ('call',
fact_rec', [('id', 'n')])), ('imprime', [('cte', 'factorial rec'), ('id', 'r')])]))

Direcciones virtuales (globales):
  n [entero] -> 1000
  r [entero] -> 1001

Funciones:
  fact_rec(n:entero) -> entero inicio=1 ret=1002

Constantes:
  1 [entero] -> 9000
  6 [entero] -> 9001
  'factorial rec' [string] -> 11000

Cuadruplos
0 : ('GOTO', None, None, 13)
1 : ('<=', 13000, 9000, 8000)
2 : ('GOTOF', 8000, None, 5)
3 : ('RET', 9000, None, 1002)
4 : ('GOTO', None, None, 12)
5 : ('ERA', 'fact_rec', None, None)
6 : ('-', 13000, 9000, 5000)
7 : ('PARAM', 5000, None, 0)
8 : ('GOSUB', 'fact_rec', None, 1)
9 : ('=', 1002, None, 5001)
10 : ('*', 13000, 5001, 5002)
11 : ('RET', 5002, None, 1002)
12 : ('ENDFUNC', None, None, None)
13 : ('=', 9001, None, 1000)
14 : ('ERA', 'fact_rec', None, None)
15 : ('PARAM', 1000, None, 0)
16 : ('GOSUB', 'fact_rec', None, 1)
17 : ('=', 1002, None, 5000)
18 : ('=', 5000, None, 1001)
19 : ('PRINT', 11000, None, None)
20 : ('PRINT', 1001, None, None)

Ejecución
factorial rec
720
```

2do Test:

```
programa fibo_iter;
vars:
  n, r : entero;

func fib_iter(n : entero) : entero
vars:
  a, b, i, next : entero;
  si (n <= 0)
    ret 0;
  sino
    si (n == 1)
      ret 1;
    sino
      a = 0;
      b = 1;
      i = 2;
      mientras (i <= n) haz
        next = a + b;
        a = b;
        b = next;
```

```
        i = i + 1;
      ret b;
finf

inicio
  n = 7;
  r = fib_iter(n);
  escribe("fibo iter", r);
fin
```

```
(base) josuesosa01@Mac patito % python run_cuadruplos.py --run  ejemplos/fibo_iter.pato
AST
('programa', 'fibo_iter', ('vars', [('decl', ['n', 'r'], ('tipo', 'entero'))]), ('funcs', [('func', 'fib_iter', [('n', ('tipo', 'entero'))], ('tipo', '
tero'), ('vars', [('decl', ['a', 'b', 'i', 'next'], ('tipo', 'entero'))]), ('cuerpo', [('si', ('rel', '<=', ('id', 'n'), ('cte', 0)), ('cuerpo', [('ret
 ('cte', 0))]), ('cuerpo', [('si', ('rel', '==', ('id', 'n'), ('cte', 1)), ('cuerpo', [('ret', ('cte', 1))]), ('cuerpo', [('asigna', 'a', ('cte', 0)),
asigna', 'b', ('cte', 1)), ('asigna', 'i', ('cte', 2)), ('mientras', ('rel', '<=', ('id', 'i'), ('id', 'n')), ('cuerpo', [('asigna', 'next', ('bin', '+
 ('id', 'a'), ('id', 'b'))), ('asigna', 'a', ('id', 'b')), ('asigna', 'b', ('id', 'next')), ('asigna', 'i', ('bin', '+', ('id', 'i'), ('cte', 1))), ('r
', ('id', 'b'))])])])])])]), ('cuerpo', [('asigna', 'n', ('cte', 7)), ('asigna', 'r', ('call', 'fib_iter', [('id', 'n')])), ('imprime', [('cte', 'fib
iter'), ('id', 'r')])]))

Direcciones virtuales (globales):
  n [entero] -> 1000
  r [entero] -> 1001

Funciones:
  fib_iter(n:entero) -> entero inicio=1 ret=1002

Constantes:
  0 [entero] -> 9000
  1 [entero] -> 9001
  2 [entero] -> 9002
  7 [entero] -> 9003
  'fibo iter' [string] -> 11000

Cuadruplos
0 : ('GOTO', None, None, 23)
1 : ('<=', 13000, 9000, 8000)
2 : ('GOTOF', 8000, None, 5)
3 : ('RET', 9000, None, 1002)
4 : ('GOTO', None, None, 22)
5 : ('==', 13000, 9001, 8001)
6 : ('GOTOF', 8001, None, 9)
7 : ('RET', 9001, None, 1002)
8 : ('GOTO', None, None, 22)
9 : ('=', 9000, None, 13001)
10 : ('=', 9001, None, 13002)
11 : ('=', 9002, None, 13003)
12 : ('<=', 13003, 13000, 8002)
13 : ('GOTOF', 8002, None, 22)
14 : ('+', 13001, 13002, 5000)
15 : ('=', 5000, None, 13004)
16 : ('=', 13002, None, 13001)
17 : ('=', 13004, None, 13002)
18 : ('+', 13003, 9001, 5001)
19 : ('=', 5001, None, 13003)
```

```
18 : ('+', 13003, 9001, 5001)
19 : ('=', 5001, None, 13003)
20 : ('RET', 13002, None, 1002)
21 : ('GOTO', None, None, 12)
22 : ('ENDFUNC', None, None, None)
23 : ('=', 9003, None, 1000)
24 : ('ERA', 'fib_iter', None, None)
25 : ('PARAM', 1000, None, 0)
26 : ('GOSUB', 'fib_iter', None, 1)
27 : ('=', 1002, None, 5000)
28 : ('=', 5000, None, 1001)
29 : ('PRINT', 11000, None, None)
30 : ('PRINT', 1001, None, None)

Ejecución
fibo iter
```