Facultad de Ciencias, UNAM
Lenguajes de Programación
Tarea 1

Hernández Salinas Oscar
Rubí Rojas Tania Michelle

09 de octubre de 2020

1. Explica brevemente qué tipos de problemas puedes resolver con cada uno de los siguientes paradigmas y nombre un lenguaje perteneciente a cada uno.

   (a) Paradigma Estructurado.
   SOLUCIÓN: Con este paradigma se pueden abarcar diferentes problemas tales como aplicaciones de escritorio, aplicaciones cientificas, en particular simulaciones
   Y uno de los leguajes mas conocidos el cual ocupa este paradigma seria *C*

   (b) Paradigma Orientado a Objetos.
   SOLUCIÓN: Con este paradigma se pueden abarcar diferentes problemas tales como simulacion de eventos simultaneos, aplicaciones de escritorio, moviles y wep.
   Y uno de los leguajes mas conocidos el cual ocupa este paradigma seria *Java*

   (c) Paradigma Funcional.
   SOLUCIÓN: Con este paradigma se pueden abarcar diferentes problemas tales como representacion de expresiones simbolicas, ademas que tiene aplicaciones en la industria aeroespacial, de finanzas y algunas empresas de diseño de hardware.
   Y uno de los leguajes mas conocidos el cual ocupa este paradigma seria *LISP*

   (d) Paradigma Lógico.
   SOLUCIÓN: Con este paradigma se pueden abarcar diferentes problemas tales como desarrollo de aplicaciones de inteligencia artificial, prueba de teoremas de forma automática, construccion de sistemas expertos, procesamiento dl lenguaje natural
   Y uno de los leguajes mas conocidos el cual ocupa este paradigma seria *Prolog*

2. Usando el lenguaje de programación PROLOG, dar un ejemplo de cada uno de los siguientes conceptos y justificar. No es necesario que sean ejemplos demasiado elaborados.

   (a) Sintaxis
   SOLUCIÓN: La sintaxis es el conjunto de reglas que debemos seguir para que el compilador sea capaz de reconocer nuestro programa como válido. En Prolog, todas las cláusulas de Horn (hechos y reglas) deben terminar con un punto `.`
   Por ejemplo, si queremos definir como `gatos` (hechos) a `cuchis, kike` y `melisa`; y tener una regla que determine las posibles parejas de gatos, entonces tendríamos un código como el siguiente:

```
% Hechos
gato(cuchis).
gato(kike).
gato(melisa).

% Regla
pareja(X, Y) :- gato(X), gato(Y).

```

Donde cada una de nuestras cláusulas termina con un punto. Si llegáramos a omitir este último, entonces el programa no compilará porque el compilador no entenderá el código y mandará un mensaje de errror al programador.

(b) Semántica

SOLUCIÓN: La semántica es el significado que se le otorga a cada una de las sentencias escritas, de acuerdo a la sintaxis definida previamente en el lenguaje. En Prolog, dado el ejemplo del inciso anterior, la regla `pareja` obtendrá sus valores de acuerdo a su universo del discurso, el cual corresponde a los hechos sobre `gatos` que hemos definido anteriormente.

(c) Convenciones de programación (Idioms)

SOLUCIÓN: Son expresiones idiomáticas que son frecuentes en un lenguaje. En Prolog, se tiene la convención de poner cada subobjetivo de las reglas en una nueva línea. Esto mejora enormemente la legibilidad del código.

Por ejemplo,

```
            ord_union_all(N, Sets0, Union, Sets) :-
                A is N / 2,
                Z is N - A,
                ord_union_all(A, Sets0, X, Sets1),
                ord_union_all(Z, Sets1, Y, Sets),
                ord_union(X, Y, Union).

```

Así pues, no importa qué tan cortos sean algunos subobjetivos, es mejor usar una línea distinta para cada uno de ellos.

(d) Bibliotecas

SOLUCIÓN: Es el conjunto de funciones previamente definidas en un lenguaje, las cuales está disponibles para utilizarse por los programadores. En Prolog, tenemos la biblioteca `lists`, la cual nos proporciona predicados (reglas) básicos para la manipulación de listas. Por ejemplo, posee la regla

`append(?List1, ?List2, ?List1AndList2)`

cuyo significado es que `List1AndList2` es la concadenación de `List1` con `List2`.

En particular, Prolog tiene una buena documentación, aunque en la mayoría de los casos carece de ejemplos para mostrar cómo funciona cada una de las reglas que contienen las bibliotecas.

3. Dada la siguiente función, da una firma para la misma indicando el tipo de entrada de los parámetros, el tipo de la salida y asígnale un nombre mnemotécnico. Justifica tu respuesta.

```
(define (foo n 1)
   (cond
      [(zero? n) empty]
      [else (cons (car l) (foo (sub1 n) (cdr 1)))]))
```

SOLUCIÓN:

```
;;Función que recibe un numero n y una lista l y devuelve una lista con los primeros n elemento
de la lista pasada como parametro.
;;primeros-lista: number (listof a) -> (listof a)
```

4. Para los siguientes incisos, calcular el resultado de aplicar la función al parámetro recibido. Mostrar cada paso realizado hasta obtener el resultado final. Da tu propia implementación para ambas funciones.

(a) `(reverse '(1 7 2 9))`

(b) `(append '(m a n) (z a n a))`

(c) `(reverse (append '(m a n) (z a n a)))`

Solución: Las implementaciones propuestas para las funciones `reverse` y `append` son las siguientes

```
(define (append xs ys)
  (if (empty? xs)
      ys
      (cons (car xs) (append (cdr xs) ys))))


(define (reverse xs)
  (if (empty? xs)
      '()
      (append (reverse (cdr xs)) (list (car xs)))))
```

Por lo tanto,

(a) `(reverse '(1 7 2 9))`

```
= (append (reverse (cdr '(1 7 2 9))) (list (car '(1 7 2 9))))
= (append (reverse '(7 2 9)) (list 1))
= (append (append (reverse (cdr '(7 2 9))) (list (car '(7 2 9)))) '(1))
= (append (append (reverse '(2 9)) (list 7)) '(1))
= (append (append (append (reverse (cdr '(2 9)))
                          (list (car '(2 9)))) '(7)) '(1))
= (append (append (append (reverse '(9)) (list 2)) '(7)) '(1))
= (append (append (append (append (reverse (cdr '(9)))
                                  (list (car '(9)))) '(2)) '(7)) '(1))
= (append (append (append (append (reverse '()) (list 9)) '(2)) '(7)) '(1))
= (append (append (append (append '() '(9)) '(2)) '(7)) '(1))
= (append (append (append '(9) '(2)) '(7)) '(1))
= (append (append (cons (car '(9)) (append (cdr '(9)) '(2))) '(7)) '(1))
= (append (append (cons 9 (append '() '(2))) '(7)) '(1))
= (append (append (cons 9 '(2)) '(7)) '(1))
= (append (append '(9 2) '(7)) '(1))
= (append (cons (car '(9 2)) (append (cdr '(9 2)) '(7))) '(1))
= (append (cons 9 (append '(2) '(7))) '(1))
= (append (cons 9 (cons (car '(2)) (append (cdr '(2)) '(7)))) '(1))
= (append (cons 9 (cons 2 (append '() '(7)))) '(1))
= (append (cons 9 (cons 2 '(7))) '(1))
= (append (cons 9 '(2 7)) '(1))
= (append '(9 2 7) '(1))
= (cons (car '(9 2 7)) (append (cdr '(9 2 7)) '(1)))
= (cons 9 (append '(2 7) '(1)))
= (cons 9 (cons (car '(2 7)) (append (cdr '(2 7)) '(1))))
= (cons 9 (cons 2 (append '(7) '(1))))
= (cons 9 (cons 2 (cons (car '(7)) (append (cdr '(7)) '(1)))))
= (cons 9 (cons 2 (cons 7 (append '() '(1)))))
= (cons 9 (cons 2 (cons 7 '(1))))
= (cons 9 (cons 2 '(7 1)))
= (cons 9 '(2 7 1))
= '(9 2 7 1)
```

(b) `(append '(m a n) (z a n a))`

```
= (cons (car '(m a n)) (append (cdr '(m a n)) '(z a n a)))
= (cons m (append '(a n) '(z a n a)))
= (cons m (cons (car '(a n)) (append (cdr '(a n)) '(z a n a))))
= (cons m (cons a (append '(n) '(z a n a))))
= (cons m (cons a (cons (car '(n)) (append (cdr '(n)) '(z a n a)))))
= (cons m (cons a (cons n (append '() '(z a n a)))))
= (cons m (cons a (cons n '(z a n a))))
= (cons m (cons a '(n z a n a)))
= (cons m '(a n z a n a))
= '(m a n z a n a)
```

(c) `(reverse (append '(m a n) (z a n a)))`

```
= (reverse (cons (car '(m a n)) (append (cdr '(m a n)) '(z a n a))))
= (reverse (cons m (append '(a n) '(z a n a))))
= (reverse (cons m (cons (car '(a n)) (append (cdr '(a n)) '(z a n a)))))
= (reverse (cons m (cons a (append '(n) '(z a n a)))))
= (reverse (cons m (cons a (cons (car '(n)) (append (cdr '(n)) '(z a n a))))))
= (reverse (cons m (cons a (cons n (append '() '(z a n a))))))
= (reverse (cons m (cons a (cons n '(z a n a)))))
= (reverse (cons m (cons a '(n z a n a))))
= (reverse (cons m '(a n z a n a)))
= (reverse '(m a n z a n a))
= (append (reverse (cdr '(m a n z a n a))) (list (car '(m a n z a n a))))
= (append (reverse '(a n z a n a)) (list m))
= (append (append (reverse (cdr '(a n z a n a))) (list (car '(a n z a n a)))) '(m))
= (append (append (reverse '(n z a n a)) (list a)) '(m))
= (append (append (append (reverse (cdr '(n z a n a))) (list (car '(n z a n a))))
                  '(a)) '(m))
= (append (append (append (reverse '(z a n a)) (list n)) '(a)) '(m))
= (append (append (append (append (reverse (cdr '(z a n a))) (list (car '(z a n a))))
                          '(n)) '(a)) '(m))
= (append (append (append (append (reverse '(a n a)) (list z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (reverse (cdr '(a n a))) (list (car '(a n a))))
                                  '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (reverse '(n a)) (list a)) '(z)) '(n))
                  '(a)) '(m))
= (append (append (append (append (append (append (reverse (cdr '(n a)))
                                                  (list (car '(n a))))
                                          '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (append (reverse '(a)) (list n))
                                          '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (append (append (reverse (cdr '(a)))
                                                          (list (car '(a))))
                                                  '(n)) '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (append (append (reverse '()) (list a))
                                                  '(n)) '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (append (append '() '(a))
                                                  '(n)) '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (append '(a) '(n)) '(a)) '(z)) '(n)) '(a)) '(m))
```

```
= (append (append (append (append (append (cons (car '(a)) (append (cdr '(a)) '(n)))
                                          '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (cons a (append '() '(n)))
                                          '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append (cons a '(n)) '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (append '(a n) '(a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons (car '(a n)) (append (cdr '(a n)) '(a)))
                                  '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons a (append '(n) '(a))) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons a (cons (car '(n)) (append (cdr '(n)) '(a))))
                                  '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons a (cons n (append '() '(a))))
                                  '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons a (cons n '(a))) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append (cons a '(n a)) '(z)) '(n)) '(a)) '(m))
= (append (append (append (append '(a n a) '(z)) '(n)) '(a)) '(m))
= (append (append (append (cons (car '(a n a)) (append (cdr '(a n a)) '(z)))
                          '(n)) '(a)) '(m))
= (append (append (append (cons a (append '(n a) '(z))) '(n)) '(a)) '(m))
= (append (append (append (cons a (cons (car '(n a)) (append (cdr '(n a)) '(z))))
                          '(n)) '(a)) '(m))
= (append (append (append (cons a (cons n (append '(a) '(z)))) '(n)) '(a)) '(m))
= (append (append (append (cons a (cons n (cons (car '(a)) (append (cdr '(a)) '(z)))))
                          '(n)) '(a)) '(m))
= (append (append (append (cons a (cons n (cons a (append '() '(z)))))
                          '(n)) '(a)) '(m))
= (append (append (append (cons a (cons n (cons a '(z)))) '(n)) '(a)) '(m))
= (append (append (append (cons a (cons n '(a z))) '(n)) '(a)) '(m))
= (append (append (append (cons a '(n a z)) '(n)) '(a)) '(m))
= (append (append (append '(a n a z) '(n)) '(a)) '(m))
= (append (append (cons (car '(a n a z)) (append (cdr '(a n a z)) '(n))) '(a)) '(m))
= (append (append (cons a (append '(n a z) '(n))) '(a)) '(m))
= (append (append (cons a (cons (car '(n a z)) (append (cdr '(n a z)) '(n))))
                  '(a)) '(m))
= (append (append (cons a (cons n (append '(a z) '(n)))) '(a)) '(m))
= (append (append (cons a (cons n (cons (car '(a z)) (append (cdr '(a z)) '(n)))))
                  '(a)) '(m))
= (append (append (cons a (cons n (cons a (append '(z) '(n))))) '(a)) '(m))
= (append (append (cons a (cons n (cons a (cons (car '(z)) (append (cdr '(z)) '(n))))))
                  '(a)) '(m))
= (append (append (cons a (cons n (cons a (cons z (append '() '(n)))))) '(a)) '(m))
= (append (append (cons a (cons n (cons a (cons z '(n))))) '(a)) '(m))
= (append (append (cons a (cons n (cons a '(z n)))) '(a)) '(m))
= (append (append (cons a (cons n '(a z n))) '(a)) '(m))
```

```
= (append (append (cons a '(n a z n)) '(a)) '(m))
= (append (append '(a n a z n) '(a)) '(m))
= (append (cons (car '(a n a z n)) (append (cdr '(a n a z n)) '(a))) '(m))
= (append (cons a (append '(n a z n) '(a))) '(m))
= (append (cons a (cons (car '(n a z n)) (append (cdr '(n a z n)) '(a)))) '(m))
= (append (cons a (cons n (append '(a z n) '(a)))) '(m))
= (append (cons a (cons n (cons (car '(a z n)) (append (cdr '(a z n)) '(a))))) '(m))
= (append (cons a (cons n (cons a (append '(z n) '(a))))) '(m))
= (append (cons a (cons n (cons a (cons (car '(z n)) (append (cdr '(z n)) '(a)))))) '(m))
= (append (cons a (cons n (cons a (cons z (append '(n) '(a)))))) '(m))
= (append (cons a (cons n (cons a (cons z (cons (car '(n)) (append (cdr '(n)) '(a))))))) (m))
= (append (cons a (cons n (cons a (cons z (cons n (append '() '(a))))))) '(m))
= (append (cons a (cons n (cons a (cons z (cons n '(a)))))) '(m))
= (append (cons a (cons n (cons a (cons z '(n a))))) '(m))
= (append (cons a (cons n (cons a '(z n a)))) '(m))
= (append (cons a (cons n '(a z n a))) '(m))
= (append (cons a '(n a z n a)) '(m))
= (append '(a n a z n a) '(m))
= (cons (car '(a n a z n a)) (append (cdr '(a n a z n a)) '(m)))
= (cons a (append '(n a z n a) '(m)))
= (cons a (cons (car '(n a z n a)) (append (cdr '(n a z n a)) '(m))))
= (cons a (cons n (append '(a z n a) '(m))))
= (cons a (cons n (cons (car '(a z n a)) (append (cdr '(a z n a)) '(m)))))
= (cons a (cons n (cons a (append '(z n a) '(m)))))
= (cons a (cons n (cons a (cons (car '(z n a)) (append (cdr '(z n a)) '(m))))))
= (cons a (cons n (cons a (cons z (append '(n a) '(m))))))
= (cons a (cons n (cons a (cons z (cons (car '(n a)) (append (cdr '(n a)) '(m)))))))
= (cons a (cons n (cons a (cons z (cons n (append '(a) '(m)))))))
= (cons a (cons n (cons a (cons z (cons n (cons (car '(a)) (append (cdr '(a)) '(m))))))))
= (cons a (cons n (cons a (cons z (cons n (cons a (append '() '(m))))))))
= (cons a (cons n (cons a (cons z (cons n (cons a '(m)))))))
= (cons a (cons n (cons a (cons z (cons n '(a m))))))
= (cons a (cons n (cons a (cons z '(n a m)))))
= (cons a (cons n (cons a '(z n a m))))
= (cons a (cons n '(a z n a m)))
= (cons a '(n a z n a m))
= '(a n a z n a m)
```

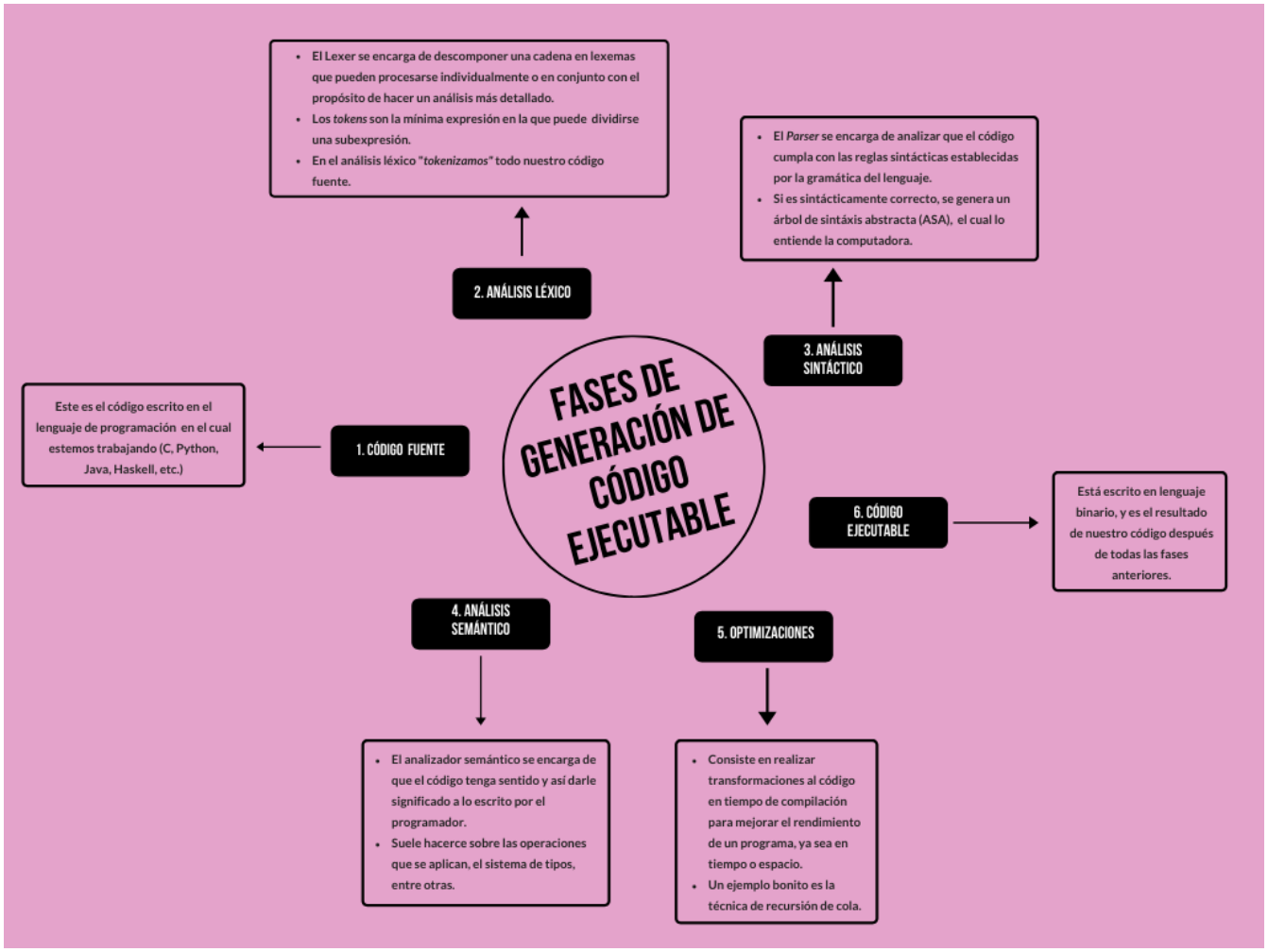
5. Da una tabla donde expliques las principales diferencias entre un compilador y un intérprete.

   Solución:

Interprete	Compilador
Ejecuta directamente las instrucciones escritas en un lenguaje de programación	Transforma el código fuente de un programa a su equivalente en otro lenguaje de mas bajo nivel
El código se traduce durante el tiempo de ejecución	El código se traduce antes de ejecutar
Se traduce linea por linea	Se traduce siempre todo el código
Manda errores después de cada linea	Manda los errores en conjunto, después de la compilación
Velocidad de traducción alta	Velocidad de traducción baja
Eficiencia de traducción baja	Eficiencia de traducción alta
Coste de desarrollo bajo	Coste de desarrollo alto

6. Dibuja un mapa mental que muestre las fases de generación de código ejecutable, sus principales características y elementos involucrados.

   Solución: Las fases de generación de código ejecutable son aquellas que analzan, modifican y optimizan el código para que la computadora lo pueda entender. El siguiente mapa mental describe el órden de estas fases.



7. Dadas las siguientes expresiones de AE en sintaxis concreta, da su respectiva representación en sintaxis abstracta por medio de los Árboles de Sintaxos Abstracta correspondientes. En caso de no poder generar el árbol, justificar.

   (a) `{+ 18 {- 15 {+ 40 5}}}`

Solución: Tenemos que

```
(parse '{+ 18 {- 15 {+ 40 5}}}) = (add (parse '18) (parse '{- 15 {+ 40 5}}))
                                = (add (num 18) (sub (parse '15) (parse '{+ 40 5})))
                                = (add (num 18) (sub (num 15) (add (parse '40) (parse '5))))
                                = (add (num 18) (sub (num 15) (add (num 40) (num 5))))
```

Por lo tanto, su representación en sintaxis abstracta es

```
(add (num 18) (sub (num 15) (add (num 40) (num 5))))
```

(b) `{+ {- 15 {+ 40}}}`

Solución: No es posible generar el árbol ya que el operador `+` es binario, y en las dos ocasiones en que es utiliza, falta alguna expresión de AE en el lado izquierdo. Así, al momento de querer aplicar el `parser`, este hecho hará que falle nuestra función y no podamos generar el árbol.

References

[1] Coding Guidelines for Prolog
`https://www.cs.unipr.it/~bagnara/Papers/PDF/TPLP12.pdf`

[2] A.21 library(lists): List Manipulation
`https://www.swi-prolog.org/pldoc/man?section=lists`

[3] Sintaxis de Prolog
`http://gpd.sip.ucm.es/jaime/pl/prolog.pdf`

[4] Sintáxis y semántica de Prolog
`https://es.qaz.wiki/wiki/Prolog_syntax_and_semantics#Semantics`

[5] Nota de clase 1: Conceptos generales
`https://drive.google.com/file/d/12JEvZQh8F8UGC7Jep3IUZSchagBBa-Yd/view`

[6] The Difference Between a Compiler and an Interpreter
`https://tomassetti.me/difference-between-compiler-interpreter/`

[7] Compilador e intérprete: definición y diferencias
`https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/compilador-e-interprete/`