Características avanzadas del lenguaje

Concurrencia, Multithreading

- Soporte en los dialectos
- Ejemplo: Common Lisp - Bordeaux Library (https://github.com/sionescu/bordeaux-threads)

Manejo de errores

- Manejo de errores a través de condiciones
- macro handler-case

```
;; Prueba variable local fuera de ámbito
(format t "Test 5 - Variable local fuera de ámbito~%")
(format t "~%")
(ejecutar-oz '((local ((a))
                      (in
                        (a = 34)
                        (Browse a)
                      end))))
;; Ejecuta la expresión local que debería imprimir 50
(handler-case
    (ejecutar-oz '((Browse a)))
    ;; Intenta acceder a la variable local 'a' fuera de su ámbito
  (error (c)
    (format t "Error: variable local fuera de ámbito.~%")))
    ;; Maneja el error y emite un mensaje si se intenta acceder a la variable local fuera de su ámbito
(format-test)
```

Garbage Collector

- Para liberar la memoria que ya no se utiliza
- Lisp es pionero
- Ejemplo: ¿Qué pasa con la variable al salir del contexto?

```
(let ((x (list 1 2 3 4 5)))
(print x))
```

Homoiconicidad

- El código es un tipo de dato
- En lisp todo es una lista, el código también
- Más flexibilidad en ejecución, da pie a Macros

Macros

- Función que opera con código
- Diferencia con función: no evalúa, sino que expande
- Ejemplo: ¿Cómo se expande esta macro?

```
(defmacro unless-print (condition &body body)
  `(if (not ,condition)
       (progn ,@body)))

(let ((x 5))
  (unless-print (> x 10)
    (format t "x no es mayor que 10. x es: ~d~%" x)))
```

Listas (avanzado)

- Comandos para el manejo de listas: car, cdr, cadr, cons, list, reverse
- Aplicación en nuestro proyecto: Ejemplo de manejo de campos de una lista

```
;; Ejecuta una expresion sobre un entorno dado.
(defun ejecutar-expresion (expr env)
    (case (first expr)
        ;; Evalúa el tipo de la primera expresión.

        (Browse (browse (second expr) env))
        ;; Si es 'Browse', llama a la función browse con el segundo elemento de la expresión.

        (declare (declare-var (second expr) env))
        ;; Si es 'declare', llama a la función declare-var con el segundo y tercer elemento de la expresión.

        (local (case (first (third expr)) ;; chequeo que este el in
                (in (case (first (last (third expr))) ;; chequeo que lo ultimo sea end
                    (end (local-vars (second expr) (third expr) env))
                    (t (error "Error: Missing 'end' in introduction of variable(s)"))
                ))
                (t (error "Error: Missing 'in' in introduction of variable(s)"))
            )
        ) ; acá 'local'
        ;; Si es 'local', llama a la función local-vars con el segundo y tercer elemento de la expresión.
```

Recursividad

- Lista, estructura de tipo recursiva (registros)
- Operaciones en el proyecto: implementación recursiva

```
(defun math-evaluation (args)
  (print args)
  (case (second args)
    (+ (+ (first args) (math-evaluation (cdr (cdr args)))))
    (- (- (first args) (math-evaluation (cdr (cdr args)))))
    (* (math-evaluation (cons (* (* (first args) (third args))) (cddr (cdr args)))))
    (/ (math-evaluation (cons (/ (/ (first args) (third args))) (cddr (cdr args)))))

    (t (first args))
  )
)
```

Comparación otros lenguajes similares

Clojure

- Dialecto de Lisp que corre en la JVM
- Filosofía “code-as-data”
- Manejo de multi-threading para mutabilidad
- https://clojure.org/

CommonLisp

- Multiparadigma: Funcional y Orientado a Objetos
- Tipado dinámico
- Macros
- https://lisp-lang.org/

```
(reduce #'-
        (reverse (list 1 2 3)))
⇒ 0

(mapcar #'string-downcase
        (list "Hello" "world!"))
=> ("hello" "world!")
```

```
(defclass book ()
  ((title :reader book-title
          :initarg :title)
   (author :reader book-author
           :initarg :author))
  (:documentation "Describes a book."))

(make-instance 'book
               :title "ANSI Common Lisp"
               :author "Paul Graham")
```

Scheme

- Basado en su simplicidad
- Conjunto mínimo de herramientas
- Tipado dinámico, homoiconicidad, macros
- https://www.scheme.org/

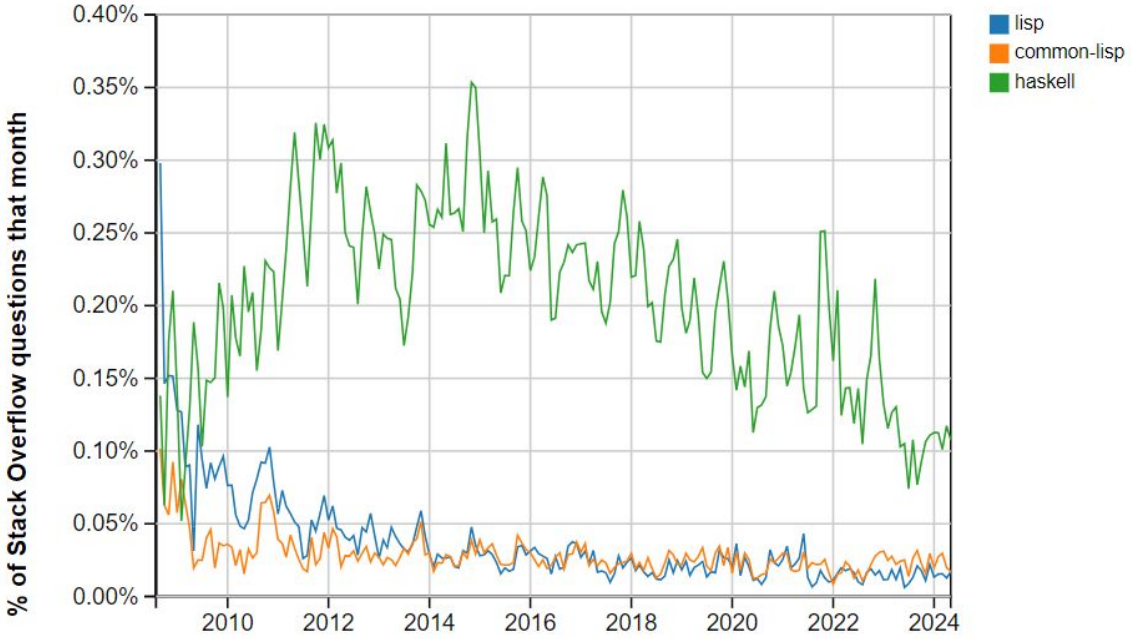
Haskell

- Puramente funcional
- Orientado a investigación
- Inmutabilidad, funciones sin side-effects, evaluación perezosa

Estadísticas

Stackoverflow Tag Trends

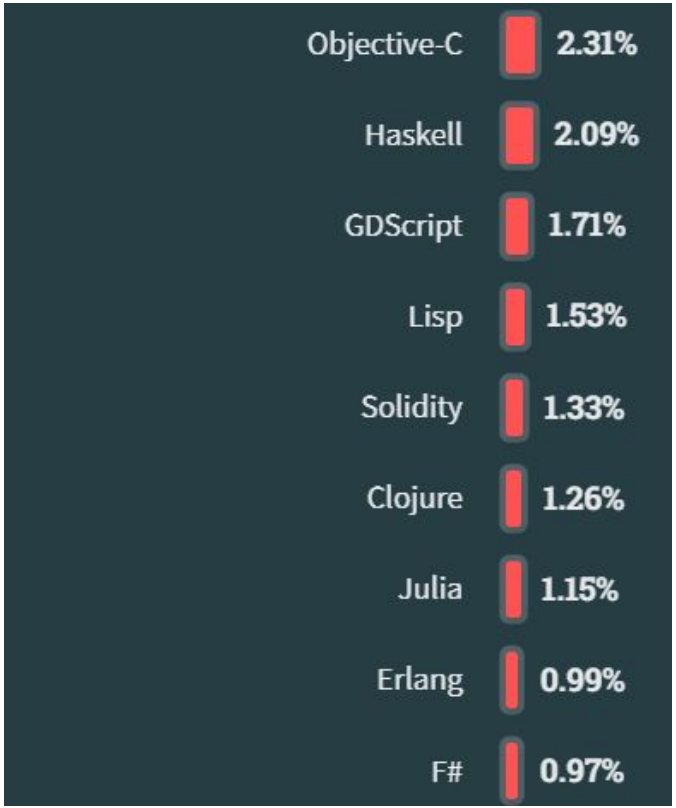
Un medidor de popularidad basado en los tags de consultas de StackOverflow a lo largo del tiempo



https://trends.stackoverflow.co/?tags=lisp,common-lisp,haskell

Stackoverflow Developer Survey

Tanto Lisp como sus derivados no tienen una gran popularidad en el público general



https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages

Tiobe Index

Índice de popularidad histórica.Gran popularidad en 1989, baja desde entonces.

Programming Language	2024	2019	2014	2009	2004	1999	1994	1989
Python	1	3	8	6	8	26	23	-
C	2	2	1	2	2	1	1	1
C++	3	4	4	3	3	2	2	2
Java	4	1	2	1	1	15	-	-
C#	5	6	5	7	7	21	-	-
JavaScript	6	7	9	9	9	19	-	-
Visual Basic	7	19	-	-	-	-	-	-
SQL	8	9	-	-	92	-	-	-
Go	9	17	35	-	-	-	-	-
PHP	10	8	6	5	6	-	-	-
Objective-C	32	10	3	34	42	-	-	-
Lisp	35	32	14	21	14	13	6	3
(Visual) Basic	-	-	7	4	5	3	3	7

https://www.tiobe.com/tiobe-index/