Program Structure in Lisp

Lisp expressions are called: symbolic expressions or s-expressions

Lisp programs run either on an interpreter or as compiled code

Lisp uses prefix notation

The interpreter checks the source code in a repeated loop, which is also called the read-evaluate-print loop (REPL).

Graphical user interface, text, application, chat or text message

Description automatically generated

Syntax

Lisp programs are made up of 2 basic building blocks: atom and list

Graphical user interface, text, application

Description automatically generated

nil : kind of null

Defining Macro

Macros are often used in Common Lisp to extend the language

In fact, some of the predefined functions are actually macros, for instance, DOLIST

Users can define their own macros with defmacro keyword

Graphical user interface, text, application, chat or text message

Description automatically generated

Macro, runtime esnasında koda dönüşür

Variables: defvar, defparameter

You can declare a variable as a “dynamically scoped variable” with defvar and defparameter

Graphical user interface, application

Description automatically generated

Genelde variable isminin başına ve sonuna yıldız konulur ki fonksiyon parametreleriyle vs. çakışmasın.

defparameter 🡪 yeniden tanımlayabilirsin, koşulsuz çalışır  
defvar 🡪 yeniden tanımlayamazsın, a tanımlıysa hiçbir şey yapmaz

Variables: set, setf, setq

They are used for setting the values of symbols

Graphical user interface, text, application, Teams

Description automatically generated

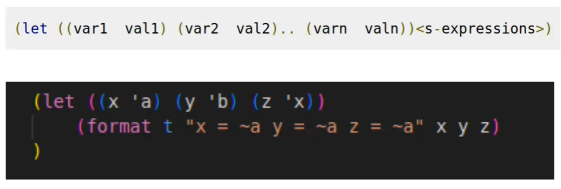
Sadece setq’yu kullansak yeter. Değişkenin değerini setq ile değiştirebiliriz. defvar ile tanımlasak bile. foo hiç tanımlanmamış olsa bile tanımlayıp değer atayabiliriz.

Graphical user interface, text

Description automatically generated

Local Variables: let, prog

There are two constructs – let and prog for creating local variables



Tek fark progta return edebiliyoruz.

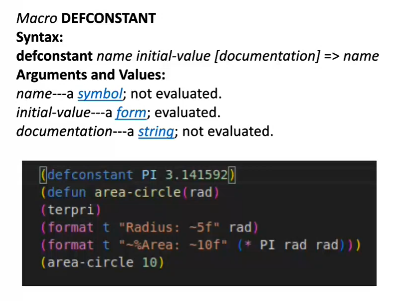
let ya da prog kullanabilirsin.

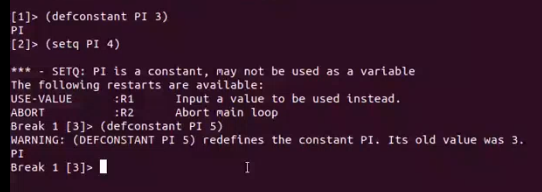
format : x = %d yaptığımız gibi stringi formatlamaya yarar. C’deki printf gibi düşünebilirsin

Constants

Constants are variables that never change their values during program execution. Constants are declared using the defconstant construct

defconstant kullandıktan sonra setq ile de değiştiremezsin ama tekrardan define edebilirsin.





warning verir ama çalışır.

Graphical user interface, text, application, email

Description automatically generatedOperators

Graphical user interface

Description automatically generated with medium confidenceDecisions: cond

test1 doğruysa action1 çalışır, değilse test2’ye bakar doğruysa action2 çalışır…

test2 doğru diyelim, action2 çalışır ve diğerlerine bakılmaz.

else yapmak için de “t” (true) deriz.

4 farklı decision operatörü vardır. En yaygını cond.

Decisions: if

Graphical user interface, text, chat or text message

Description automatically generated

doğruysa ilki çalışır, değilse ikincisi

Decisions: when

Graphical user interface, text, application, chat or text message

Description automatically generated

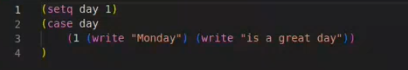
else kısmı yok, istediğin kadar şey yaptır

when yerine unless yazarsan tam tersi olur yani sadece condition yanlışsa satırlar çalışır

Decisions: case

Text

Description automatically generated



Birden fazla action kullanılabilir

format yerine write kullanılabilir

En sona “otherwise (format …” diyebilirsin, default gibi.

Functions

Text

Description automatically generatedThe macro named defun is used for defining functions

last expression neyse onu return eder

returnfrom keywordü ile istediğin değeri return edebilirsin.

Functions: lambda

Lisp allows you to write anonymous functions that are evaluated only when they are encountered in the program. These functions are called Lambda functions.

Graphical user interface, text, application

Description automatically generated

mavi parantezin tamamı bi fonksiyon return eder. a, b, c, x fonksiyonun parametreleridir.

Functions: mapcar

The mapcar function processes successive elements of one or more lists

The first argument of the mapcar function should be a function and the remaining arguments are the list(s) to which the function is applied

Graphical user interface

Description automatically generated

En sondaki 3 5 11 15 return edecek. Sonrası için uygulanamayacak çünkü ikinci listede eleman yok.

Örneğin en alttaki + nın başına tırnak koymazsak evaluate edilmiyor. Mesela ‘(1 3 5 7 9 11 13) yerine tırnak koymasaydık hata verirdi. Çünkü 1’i fonksiyon olarak algılayacaktı ve 3 5 7 9 11 13 elemanlarını da fonksiyonun parametreleri olarak algılayacaktı.

Text

Description automatically generated

Örneğin ‘1+ yerine 1+ yazarsak 1+’yı hiç evaluate etme, direkt data olarak return et diyoruz.

1+ 🡪 Lisp’te tanımlı fonksiyon

Eğer bir şeyin evaluate edilmesini istiyorsan başına tırnak koymalısın.

Text

Description automatically generated with low confidence

#’print 🡪 go and return the function named print

Text

Description automatically generated

Örneğin fonksiyona parametre olarak liste göndereceksin, listelerin evaluate edilmesini istemediğin için başına tırnak koyarsın. Liste zaten evaluate edilecek bir şey değil, tırnak koymazsan hata verir.

Örneğin listenin ilk elemanı çarpı olabilirdi. Öyle bir fonksiyon var ama bizim niyetimiz o değil o yüzden liste istiyorsan tırnak koymaya dikkat et.

mapcar, fonksiyon ve liste(ler) alır. Fonksiyonu listenin elemanlarına uygular. İlk parametre fonksiyon, sonraki parametre(ler) liste.

Verdiğimiz fonksiyon tek parametre alan bir fonksiyonsa mapcara tek liste veriyoruz. mapcar tek tek o fonksiyonu uygular ve sonuçlarını liste olarak return eder.

Lists

Lists are the most important and the primary composite data structure in Lisp

A picture containing diagram

Description automatically generated

List Functions: car, cdr, cons

car 🡪 takes a list as argument, and returns its first element

cdr 🡪 takes a list as argument, and returns a list without the first element

cons 🡪 takes 2 arguments, an element and a list, and returns a list with the element inserted at the first place

Graphical user interface, text

Description automatically generated

(cdr ‘(1)) 🡪 tek eleman 1 olduğu için NIL döndürür

Listenin car’ını cons’a ilk parametre olarak verirsek, cdr’ını da cons’a ikinci parametre olarak verirsek listenin kendisini return eder.

Text

Description automatically generated

Case Study

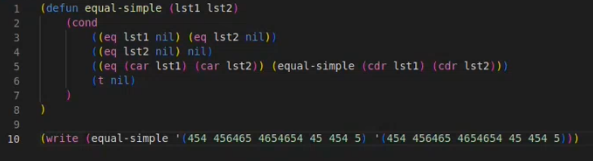
Implementing member function

* Lisp’te member diye bir fonksiyon var. Liste ve eleman alır. Eleman listede mi diye bakar.
* Text

  Description automatically generated
* Olmayan bir şey için NIL, olan bir şey için true return eder.
* = fonksiyonu sadece 2 tane sayı kıyaslar, liste kıyaslamak için eq kullanırız
* Text

  Description automatically generated
* Listenin elemanı sayı olmayabilir, bu daha sağlıklı
* = ile NIL de kıyaslanmaz
* boş listenin car’ı 🡪 NIL
* NIL’in car’ı 🡪 NIL

Checking two simple (no nested list) lists are equal

* list bir keyword/fonksiyon olduğu için liste isimlerimizi lst yapıyoruz.
* 

Checking two complex lists are equal

Implementing union and difference of two list

*BÜTÜN FONKSİYONLAR RECURSIVE YAZILMALI*

(format t “hello world ~%”) 🡪 t: terminal , ~%: new line

(print “what’s your name”) 🡪 output to screen

(defvar \*name\* (read)) 🡪 global variables are surrounded by asterisks but this is not necessary

(defun hello-you (nameValue)  
 (format t “Hello ~a! ~%” nameValue)  
)

(setq \*print-case\* :capitalize) 🡪 you can use also “upcase, downcase”

(hello-you \*name\*)

~a 🡪 shows the value  
~s 🡪 shows quotes around the value  
~d 🡪 shows number  
~10a 🡪 sağa 10 boşluk ekler  
~10@a 🡪 sola 10 boşluk ekler

nil is at the end of the list and represent the end of it.

Change variable:

* (defvar \*number\* 0)
* (setf \*number\* 6)

eq 🡪 verdiğin 2 değer eşit mi bakar

(defvar \*num\* 2)  
(defvar \*num-2\* 2)  
(defvar \*num-3\* 2)  
(if (= \*num\* 2)  
 (progn 🡪 to put multiple statements  
 (set \*num-2\* (\* \*num-2\* 2))  
 (set \*num-3\* (\* \*num-3\* 2))  
 )  
 (format t “Not equal to 2 ~%)

)

(terpri) 🡪 new line koyar

Looping

(loop for x from 1 to 10  
 do (print x)  
)

(setq x 1)  
(loop  
 (format t “~d ~%” x)  
 (setq x (+ x 1))  
 (when (> x 10) (return x))  
)

(loop for x in ‘(Peter Paul Mary) do  
 (format t “~s ~%” x)  
)

(loop for y from 1 to 100 do  
 (print y)  
)

(dotimes (y 12)  
 (print y)  
)