آموزش زبان Lisp

گردآورندگان: سید عرفان فرهانیان، علی حسینی
استاد: مهدی عابدینی

مقدمه

- اولین بار در سال ۱۹۵۸توسط جان مک کارتی و دانشجویانش
- قدیمی ترین زبان بعد از فرترن
- طراحی جهت رشد و تکامل
- تعریف برنامه های لیسپ به عنوان ساختمان داده

امکانات

- اجازه تعریف اپراتورهای جدید
- برنامه نویسی پایین – بالا (down-top)
- هر لایه وظیفه مرتب سازی زبان برنامه سازی لایه بالایی را دارد
- از سال ۱۹۶۰ در حال تحول
- "شی گرایی پویا"

فرم (Form)

- شامل یک روش محاوره ای جلو-عقب (front-end) ==> سطح بالا (top level)
- > : برای نشان دادن پرامپت سطح بالا

```
> 1

1

>
```

اپراتورها

- جمع دو عدد در لیسپ :

```
> (+ 2 3)
5
```

- جمع بیش از ۲ عدد:

```
> (+ 2 3 4)
2 + 3 + 4
```

- روش نشانه گذاری Prefix

اپراتورها

- ابتدا و انتهای آن با پرانتز مشخص
- عبارات تو در تو:

```
> (/ (- 7 1) (- 4 2))
3
```

اپراتورها

- اکثر اپراتورها در لیسپ تابع هستند.
- روش قانون ارزیابی در common lisp
- اپراتور quote
- "انجام هیچ چیز" !!!
- **مثال:**

```
> (quote (+ 3 5))
(+3 5)
```

- quote = `

سمبول ها (Symbols) و لیست ها (Lists)

> 'Artichoke

ARTICHOKE

- صرف نظر از اینکه شما چگونه آنها را تایپ می کنید معمولا آنها به حروف بزرگ تبدیل می شوند.
- قبل از استفاده از یمبل ها استفاده از quote
- لیست ها در میان دو پرانتز برای نشان دادن هیچ یا تعداد زیادی عناصر. مثال:

> '(my 3 "Sons")

(MY 3 "Sons")

> '(the list (a b c) has 3 elements)

(THE LIST (A B C) HAS 3 ELEMENTS)

لیست ها (Lists)

- فراخوانی list برای ایجاد لیست ها

```
> (list 'my (+ 2 1) "Sons")
(MY 3 "Sons")
```

- نکته: quote قبل از لیست

```
> (list '(+ 2 1) (+ 2 1))
((+ 2 1) 3)
```

NIL

- برای نشان دادن لیست های خالی
- نشان دهنده تهی

> ( )
NIL
یا
>nil
NIL

- quote nil نداریم.

اپراتورهای لیست(List Operations)

- **cons.** : ساخت یا ایجاد یک لیست

> (cons 'a ' (b c d))

(A B C D)

- قابلیت ساخت لیست هایی توسط عناصر جدید برروی یک لیست خالی

> (cons 'a (cons ‘b nil))

(A B)

> (list'a'b)

(A B)

اپراتورهای لیست(List Operations)

- car و cdr : برای استخراج عناصر از داخل لیست ها
- car اولین عنصر لیست را برمی گرداند.
- cdr هر چیزی که بعد از عنصر اول باشد را برمی گرداند.

```
> (car '(a b c))
A
> (cdr '(a b c))
(B C)
```

- third برای بدست آوردن عنصر سوم

```
> (third '(a b c d))
C
```

درستی (Truth)

- سمبول t

```
>(listp ’(a b c ))
T
```

درست بودن : T

غلط بودن : NIL

دستور شرطی if

- معمولا ۳ آرگومان می گیرد: test, then, else
- به صورت یک تابع اجرا نمی شود.

```
> (if (listp '(a b c))
(+ 1 2)
(+ 5 6))
3
```

- اگر آرگومان سوم حذف شود به صورت پیش فرض NIL برمی گرداند.

```
> (if (listp 27)
(+ 2 3))
NIL
```

توابع (Function)

- با defun تعریف می شوند.
- دو یا سه آرگومان می گیرد.

- مثالی دیگر:

```
> (defun our-third (x) (car (cdr (cdr x))))
OUR-THIRD

> (defun sum-greater (x y z)
(> (+ x y) z))
SUM-GREATER
> (sum-greater 1 4 3)
T
```

ورودی و خروجی (Input and Output)

- عمومی ترین تابع خروجی در لیسپ : format
- دو یا تعداد بیشتری آرگومان می گیرد:

اولی نشان می دهد که خروجی کجا باید چاپ شود.

دومین آرگومان نشان دهنده قالب رشته می باشد.

```
> (format t "~A plus ~A equals ~A~%"2 3 (+ 2 3))
2 plus 3 equals 5
NIL
```

تابع read

- تابعی استاندارد برای ورودی
- از مکان پیشفرض می خواند.

```
(defun askem (string)
(format t "~A" string)
(read))
```

- خروجی:

```
> (askem "How old are you? ")
How old are you? 29
29
```

متغیرها (variables)

- let : اجازه تعریف متغیرهای جدید را می دهد(متغیرهای محلی).

```
> (let ((x 1) (y 2))
(+ x y))
3
```

متغیرها (variables)

- **متغیرهای سراری:** با استفاده از defparameter ایجاد می شوند.
- در هر جایی قابل دسترس می باشند. به جز در عباراتی که متغیر های محلی جدیدی با همین نام ایجاد شده باشد.
- برای جلوگیری از این اتفاق برای نشان دادن متغیرهای سراسری ابتدا و انتهای آن را * قرار می دهند.

```
> (defparameter *glob* 99)
*GLOB*
```

- برای نشان دادن یک متغیر سراسری یا محلی از boundp استفاده می شود.

```
> (boundp '*glob*)
T
```

انتساب (Assignment)

- از تابع setf برای انتساب استفاده می شود.

```
> (setf *glob* 98)
98
```

- می توان بیش از انتساب یک مقدار به متغیر انجام داد.

```
> (setf (car x) 'n)
N
>x
(N B C)
```

LAMBDA

- یک سمبول است.
- عبارت lambda شامل : یک سمبول lambda و دنبال آن لیستی از پارامترها بدنه که شامل صفر یا تعداد زیادی عبارت است می آید.
- Lambda می تواند به عنوان نام یک تابع نیز در نظر گرفته شود.
- اضافه کردن ’#’ به عبارت lambda

```
> (funcall #'(lambda (x) (+ x 100))
                1)

101
```

نوع ها (Types)

- لیسپ دارای یک رویکرد انعطاف پذیر غیر متداول به نوع ها دارد.
- Typep یک شی و یک نوع مشخص می گیرد و اگر عبارت درست باشد، مقدار درست را برمی گرداند.

```
> (typep 27 'integer)
T
```