

More on Functions

York University CSE 3401

Vida Movahedi

Overview

- Optional variables
- Variable number of arguments
- Keyword parameters
- Auxiliary parameters
- Function and closure

[ref.: Chap. 12- Wilensky]

Optional parameters

- Parameters following **&optional** in the formal parameter list of a function definition are optional.

```
(defun sayhi (&optional name)
  (princ "Hi ")
  (if name (princ name))
  (terpri))
```

```
> (sayhi)
```

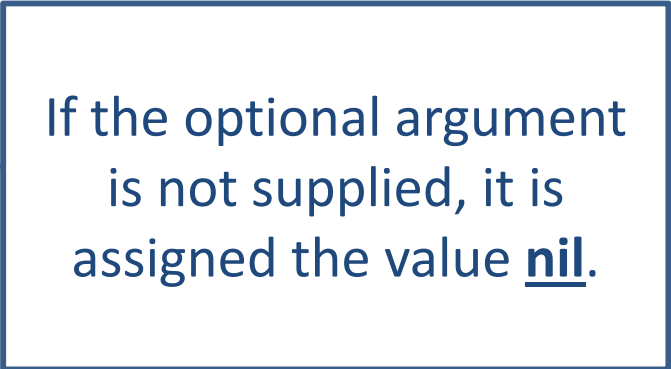
```
Hi
```

```
NIL
```

```
> (sayhi "Bob")
```

```
Hi Bob
```

```
NIL
```



If the optional argument is not supplied, it is assigned the value **nil**.

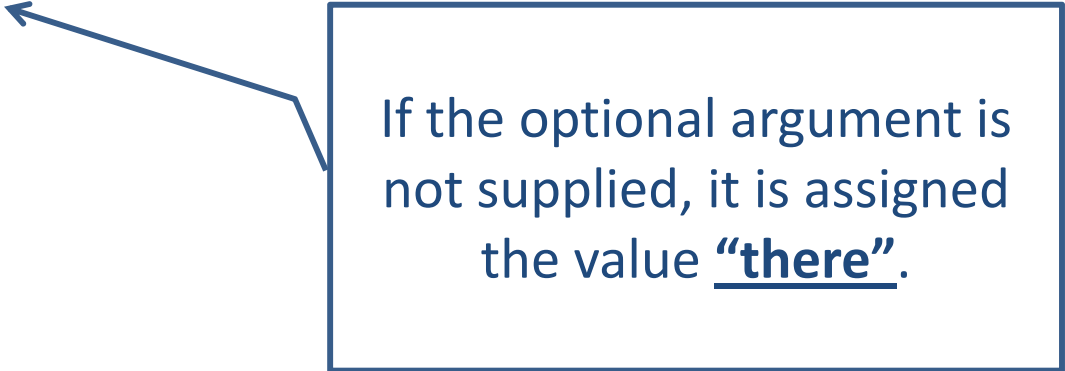
Optional parameters (cont.)

- It is also possible to define a default value for the optional argument

```
(defun sayhi (&optional (name "there"))  
  (princ "Hi ")  
  (princ name)  
  (terpri))
```

```
> (sayhi)  
Hi there  
NIL
```

```
> (sayhi "Bob")  
Hi Bob  
NIL
```



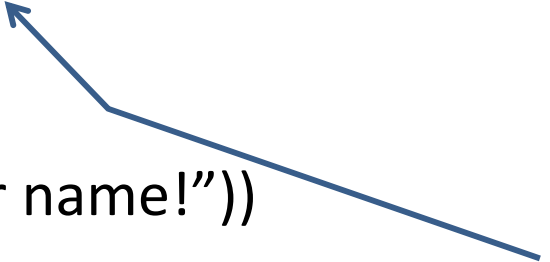
If the optional argument is not supplied, it is assigned the value "there".

Optional parameters(cont.)

- But we might need to know if the optional argument is supplied or not:

```
(defun sayhi (&optional (name "there" nameflag))  
  (princ "Hi ")  
  (princ name)  
  (if (null nameflag)  
      (print "Sorry I don't know your name!"))  
  (terpri))
```

```
> (sayhi)  
Hi there  
Sorry I don't know your name!  
NIL  
  
> (sayhi "Bob")  
Hi Bob  
NIL
```



If the optional argument is not supplied, it is assigned the value there, and nameflag will be set to nil.

Another example

- Get the n^{th} element of a list; if n not supplied, get the first one

```
> (defun gete (lst &optional (n 1))  
      (do ((tlst lst (cdr tlst))  
          (i (1- n) (1- i)))  
          ((zerop i) (car tlst))))
```

GETE

```
> (gete '(1 2 3) 1)
```

1

```
> (gete '(1 2 3) 3)
```

3

```
> (gete '(1 2 3))
```

1

lst: required parameter
n: optional parameter
with a default value of 1

Parameter designators

- Symbols such as **&optional** are called parameter designators.
- They can be used in lambda expressions in general.
- Note that if you have more than one optional argument, they will all be listed after the **&optional** symbol, e.g.
(p1 p2 **&optional** p3 (p4 p4-default) (p5 p5-default p5-flag))

Variable number of arguments

- **&rest** is a parameter designator allowing for variable number of supplied arguments
- Always followed by a single parameter, called the **rest parameter**
- Any arguments not assigned to required or optional parameters will be put in a list and assigned to the rest parameter

(p1 p2 **&optional** p3 (p4 p4def p4flg) **&rest** p5)

Assigned arguments

- Assume we define a function as

```
(defun f1 (p1 p2  
          &optional p3 (p4 p4def p4flg)  
          &rest p5)  
  ...)
```

The value of a, b, etc will be assigned to the parameters

	p1	p2	p3	p4	p4flg	p5
(f1 a)	a	<i>Error!</i>				
(f1 a b)	a	b	nil	p4def	nil	nil
(f1 a b c)	a	b	c	p4def	nil	nil
(f1 a b c d)	a	b	c	d	t	nil
(f1 a b c d e)	a	b	c	d	t	(e)
(f1 a b c d e f)	a	b	c	d	t	(e f)

Example

```
(defun sayhi (&optional (name "there" nameflag) &rest others)
  (princ "Hi ")
  (princ name)
  (if (null nameflag)
      (print "Sorry I don't know your name!"))
  (do ((tlst others (cdr tlst)))
      ((null tlst) )
      (princ " and ") (princ (car tlst)))
  (terpri))
```

```
> (sayhi "Adam" "Bob" "Christina")
Hi Adam and Bob and Christina
NIL
```

```
> (sayhi)
Hi there
Sorry I don't know your name
NIL
```

Keyword parameters

- What if we want to supply an optional parameter, but skip the previous ones?
 - We need keywords to specify which one we are supplying
 - **&key** followed by a set of parameters
 - The parameters following &key are keywords used as **:keyword** when supplying arguments
 - The parameters set to nil if not supplied

```
> (defun setlocation (p &key x y)
  (if x (setf (get p 'xloc) x))
  (if y (setf (get p 'yloc) y)))
```

```
> (setlocation 'p1 :y 20 :x 10)
20
> (symbol-plist 'p1)
(YLOC 20 XLOC 10)
```

The order of supplied keyword parameters is not important.

Keyword parameters (cont.)

- We can specify default values and supply flags for keyword parameters (similar to optional parameters)

```
(defun fkey (&key k1 (k2 k2def k2flg)) ....
```

- We can also specify a different keyword name for a keyword parameter

```
> (defun setlocation (p &key ((:x xval) -1) ((:y yval) -1)  
  (setf (get p 'xloc) xval)  
  (setf (get p 'yloc) yval))
```

```
> (setlocation 'p2 :y 30)  
30  
> (symbol-plist 'p2)  
(YLOC 30 XLOC -1)
```

The keyword parameters are
NOT required.
They are optional.

Auxiliary parameters

- Parameters following **&aux** are parameters never supplied in a function call,
But are local variables in a function definition.
Default values can be defined for them.
- Examples: Add integers less than n
 - > (defun sumupto (n **&aux** (sum 0))
 (dotimes (i n sum) (setq sum (+ i sum))))
 - > (sumupto 2)
1
 - > (sumupto 5)
10
 - > **sum**
Error: variable sum has no value!

Assigned arguments

- Assume we define a function as

```
(defun f2 (p1 &optional p2 &rest p3
           &key (p4 p4def p4flg)
           &aux (p5 p5def)) ...)
```

The value of a, b,
etc will be assigned
to the parameters

	p1	p2	p3	p4	P4flg	P5
(f2 a)	a	nil	nil	p4def	Nil	P5def
(f2 a b)	a	b	nil	p4def	Nil	P5def
(f2 a b c)	a	b	Error! Must provide p4 in pairs			
(f2 a b :p4 c)	a	b	(:p4 c)	c	T	P5def
(f2 a b :p4 c d)	a	b	Error! Must provide pairs! (and only :p4 keyword is accepted)			
(f2 a b :p4 c :p4 d)	a	b	(:p4 c :p4 d)	C	T	p5def

Function and #'

- It is good programming practice to use **function** instead of **quote** when quoting functions.
- And it can be abbreviated as **#'** instead of **'**
- Example:
Instead of:

```
> (mapcar 'sqrt '(1 4 9 16))  
(1 2 3 4)
```

Use:

```
> (mapcar #'sqrt '(1 4 9 16))  
(1 2 3 4)
```

Function and closure

- Function also creates a closure- a snapshot of a function saving its free variables.

```
> (defun seq-generator (n)
  (function (lambda () (setq n (1+ n)))))
```

```
> (setq seqgen1 (seq-generator 0))
```

```
 #(FUNCTION :....
```

```
> (funcall seqgen1)
```

```
1
```

```
> (funcall seqgen1)
```

```
2
```

```
> (setq seqgen2 (seq-generator 10))
```

```
 #(FUNCTION : ...
```

```
> (funcall seqgen2)
```

```
11
```

```
> (funcall seqgen1)
```

```
3
```

```
> (funcall seqgen2)
```

```
12
```