Clips

1] Function

1-تعريف دالة لا ترجع بقيمة، اى لا تأخذ parameter:

(deffunction اسم الدالة ( )

.....)

2-تعريف دالة لا ترجع بقيمة، و تأخذ parameter:

(deffunction اسم الدالة (?parameters)

.....)

3-تعريف دالة ترجع بقيمة، و تأخذ parameter:

(deffunction اسم الدالة (?parameters)

.....

return القيمة التى ترجع

Or return القيمة التى ترجع بدون كلمة

)

EX1:- Write a function to check if the number is positive, negative, or zero, or non-numeric.

```
(deffunction ChkNum(?n)
   (if (eq(number ?n) FALSE) then "non-numeric"
      else
   (if (= ?n 0) then Zero
      else
   (if (> ?n 0) then Positive
                else Negative)
   )
)
```

EX2:- Write a function to compute a factorial.

```
(deffunction factorial(?v)
   (if (= ?v 0) then 1
      else
   (* ?v (factorial (- ?v 1)))
   )
)
```

❑ loop-for-count

(loop-for-count عدد مرات التكرار do

……)

EX1: (loop-for-count 3 (printout t "Hello" crlf))

EX2: (loop-for-count (?c 2 6) do (printout t ?c crlf))← 6 : 2 بتطبع الأرقام من

❑ progn$

- دالة تعمل مع المتغيرات المتعددة القيم، بتعدى على العناصر عنصر بعنصر.

(progn$?i $?c)

- ?i (المتغير اللى بسجل فيه قيمة كل عنصر فى المتغير المتعدد)

EX: Write the following facts and then print a student name and score.

```
(deffacts  fct
(stu  ahmed  10  40  30  10  20)
(stu  mohamed  5  4  30  30  20)
(stu  mona  30  40  70  14  40) )
(defrule calctotal
(stu  ?n  $?d)
=>
```

```
(bind  ?sum  0)
(progn$  ?i  $?d)
(bind  ?sum (+  ?sum  ?i))
(printout  t "student name:"  ?n  "score"  ?sum  crlf)
 )
```

❑ Switch

```
(switch  ?اسم المتغير
(case 1  then  ناتج الحالة)
(case 2  then  ناتج الحالة)
(case 3  then  ناتج الحالة)
(default   النتيجة بحيث لو مفيش اى حالة تحققت) )
```

```
Ex: (deffunction number(?n)
      (switch ?n
            (case 1 then (printout t "one" crlf))
            (case 2 then (printout t "two" crlf))
            (case 3 then (printout t "three" crlf))
            (case 4 then (printout t "four" crlf))
      (default (printout t "please insert number less than four" crlf))
)
```

2] Predicate Function

1) numberp → تختبر هل العنصر المُدخل رقم ام لا

EX: (numberp 8) → True

2) floatp → تختبر هل الرقم المُدخل عشري ام لا

EX: (floatp 6.3) → True

3) integerp → تختبر هل الرقم المُدخل صحيح ام لا

EX: (integerp 7) → True

4) lexemep → تختبر هل القيمة المُدخلة symbol or string

EX: (lexemep ali) → True

5) stringp → تختبر هل القيمة المُدخلة string

EX: (stringp "ahmed") → True

6) symbolp → تختبر هل القيمة المُدخلة symbol

EX: (symbolp "ahmed") → True

7) evenp → تختبر هل الرقم المُدخل زوجى ام لا

EX: (evenp 4) → True

8) oddp → تختبر هل الرقم المُدخل فردى ام لا

EX: (oddp 3) → True

9) multifieldp & sequencep → تختبر هل القيمة المُدخلة multivalue

EX: (multifieldp (create$ a b c d)) → True

10) eq → تختبر هل الرقمين المُدخلين متساويين ام لا

EX: (eq 3 3 3) → True

11) neq → تختبر هل الرقمين المُدخلين غير متساويين ام لا

EX: (neq A a) → True

3] Multi-Field Function

1) create$ → multivalues بتنشأ متغير نوعه

EX: (create$ a b c d e)

2) nth$ → multivalues بتجيب قيمة العنصر اللى انا بحدد رقمه من المتغير ال

- Syntax: (nth$ index ($?Variable))

EX: (nth$ 2 (create$ Red Green Blue)

3) member$ → multivalues بتجيب رقم العنصر اللى انا بحدده من المتغير ال

- Syntax: (member$ القيمة multivalue)

EX: (member$ Green (create$ Red Green Blue)

4) subsetp$ → الثانى multivalue الأول جزء من ال multivalue بيختبر هل ال

- Syntax: (subsetp$ $? المتغير الأول $? المتغير الثانى)

EX: (subsetp$ (create$ 4 3 7) (create$ 1 3 4 5 7 8)) → Res: True

5) delete$ → multivalue بتمسح من ال

- Syntax: (delete$ $? المتغير البداية النهاية)

EX: (delete$ (create$ a b c d e f) 3 4) → Res: (a b e f)

- لو عاوز احذف عنصر واحد هنخلى البداية زى النهاية.

6) explode$ → multivalues لمتغير string بتحول ال

EX: (explode$ "mohamed ali ahmed") → Res: mohamed ali ahmed

7) implode$ → string J multivalue بتحول ال

EX: (implode$ (create$ mohamed ali ahmed) → Res: "mohamed ali ahmed"

8) subseq$ → multivalues من المتغير ال range بيطلعلى ال

- Syntax: (subseq$ $? المتغير اللى هاخد منه البداية النهاية)

EX: (subseq$ (create$ a b c d e f) 2 5) → Res: (b c d e)

9) replace$ → بيبدل عنصر بمجموعة عناصر او عنصر واحد

- Syntax: (replace$ $? المتغير البداية النهاية القيمة الجديدة)

EX: (replace$ (create$ a b c d e f) 2 3 xyz) → Res: (a xyz d e f)

10) insert$ → بضيف عنصر او مجموعة عناصر فى ال index اللى بحدده

- **Syntax: (insert$ $?المتغير index القيمة اللى هضيفها)**

EX: (insert$ (create$ a d e) 2 b c) → Res: (a b c d e)

11) first$ → بتجيب اول عنصر فى ال multifield

EX: (first$ (create$ a b c)) → Res: (a)

EX: (first$ (create$)) → Res: ( )

12) rest$ → بتجيب آخر عنصر فى ال multifield

EX: (rest$ (create$ a b c d)) → Res: (b c d)

EX: (rest$ (create$ a)) → Res: ( )

EX1:- Write the following fact and then write a rule which test if the first fact equal to the second fact.

```
(deffact  f1
        (fct1 a b c d e)
        (fct2 b a c e d)
 )
(defrule r1
        (fct1 $?p1)
        (fct2 $?p2)
(test (subsetp $?p  $?p))
(test (eq (length$ $?p2) (length$ $?p1)))
  =>
(printout t $?p1  $?p2 crlf)
 )
```

EX2:- Write the following facts and then replace every value equal -1 with "ok"

```
(deffacts data
        (stu ahmed 10 -1 20 -1 20 10 -1 5)
)
(defrule r1
   ?f<- (stu ?n $?scr)
(test (neq (member$ -1 $?scr) FALSE))
=>
(progn$ (?i $?scr) do
   (if (= ?i -1) then
(bind $?scr (replce$ $?scr ?i-index ?i-index ok))
   )
)
(retract ?f)
(assert (stu ?n $?scr))
)
```

4] String Function

1) str-length or length → بتحسب طول النص مع المسافات ايضاً

- Syntax: (str-length "النص")

EX: (str-length "abc d") → Res: 5

2) str-cat → بتدمج نص مع نص آخر بدون مسافات

- Syntax: (str-cat "النص" "النص الجديد اللى هيضاف")

EX: (str-cat "micro" computer) → Res: "microcomputer"

3) sym-cat → بتدمج symbol مع symbol

EX: (sym-cat a bc) → Res: abc

4) sub-string → بتاخد جزء من النص انا بكون محدد بدايته ونهايته مع وجود المسافات

- Syntax: (sub-string البداية النهاية "النص")

EX: (sub-string 10 17 "personal computer") → Res: "computer"

5) str-index → بتجيب الـ index بتاع string

- Syntax: (str-index "النص اللى عاوز احدد مكانه" "النص اللى بدور فيه")

EX: (str-index "computer" "personal computer") → Res: 10

6) upcase → بتحول الحروف لـ capital

EX: (upcase "computer") → Res: "COMPUTER"

7) lowcase → بتحول الحروف لـ small

EX: (lowcase HELLO) → Res: hello

8) str-compar → بتقارن ما بين نصين من حيث الـ ASCII code

- لو النصين متساويين الناتج هيكون 0
- لو النص الأول اكبر الناتج هيكون 1
- لو النص الثانى اكبر الناتج هيكون -1

EX: (str-compare "a" "b") → Res: -1

9) eval → command promot بتحول النص لـ

```
EX: (eval "(* 10 2)") → Res: 20
EX: (eval "(create$ a b c)") → Res: (a b c)
```

10) build

- بتاخد الـ construct على انه string وترجع بـ True لو اتنفذت او بـ False لو مش اتنفذت.

```
EX: (build "defrule myRule"
           =>
           (assert (ok)))"
)
```

- لو عاوز اشوف الـ rule دى فعلاً موجوده ام لا بنكتب :

```
> (rules)
```

- Res : myRule

5] Classes

① Define class

(defclass classاسم (is-a USER)

اسم الكلاس او اللى وارث منه ← لو مش وارث من حاجة

(slot اسم اول متغير)

(slot اسم ثانى متغير)

)

EX1:- (defclass A (is-a USER)

(slot a)

(slot b)

)

EX2:- (defclass B (is-a A)

(slot x)

(slot y)

)

الكلاس ده فيه قيم x,y والـ a,b اللى ورثهم من الـ A .

- لمعرفة الـ classes الوارثة من كل class .

From Browse → defclass manager →

② Make-instance

- عشان انشأ object من الـ class

(make-instance [instance الـ اسم] OF class_name

(اسم المتغير القيمة)

(اسم المتغير القيمة)

)

```
EX1: (defclass  x (is-a  USER)
          (slot  a)
          (slot  b)
      )
(make-instance  [x1]  OF  x  (a 3) (b 4))
```

- ممكن مكتبش اسم لـ instance فى الحالة دى هيحصل generate باسم من عنده.
- لو مخصصتش قيمة لـ variable فالكلاس هنا هيبقى بـ nil.
- عشان نشوف الـ instance اللى عندى : windows → instances

```
EX2: (defclass  x (is-a  USER)
        (slot  n1)
        (slot  n2)
      )
(defrule  start
 =>
(make-instance  [x1]  OF  x  (n1 20) (n2 3))
(make-instance  [x2]  OF  x  (n1 40) (n2 5))
(make-instance  [x3]  OF  x  (n1 10) (n2 4))
)
(defrule  R1
  (object  (is-a  x) (n1 ?a) (n2 ?b))
   =>
(printout t  ?a "*"  ?b "=" (* ?a ?b) crlf)
)
(defrule  R2
  ?i <- (object  (is-a  x))
   =>
(printout t ?i  crlf)
)
```

```
EX3: (defclass  x (is-a  USER)
         (slot  a)
         (slot  b)
        (slot  b) )
(defclass  y (is-a  USER)
         (slot  d (create-accessor read-write) )
(defclass  z (is-a  USER)
         (slot  a (create-accessor read-write)) )
(defrule  start
 =>
(make-instance  [x1]  OF  x  (a 20) (b 3))
(make-instance  [x2]  OF  x  (a 40) (b 5))
(make-instance  [x3]  OF  x  (a 10) (b 4))
(make-instance  [y1]  OF  y  (d 50))
(make-instance  [y2]  OF  y  (d 20))
(make-instance  [z1]  OF  z  (a 70)) )
(defrule  R1
  (object  (is-a  x) (a ?n1))
  (object  (is-a  y) (d ?n1))
   =>
(printout t ?n1 crlf) )
(defrule  R2
  ?inst <- (object  (is-a  x|y))
   =>
(printout t ?inst  crlf) )
(defrule  R3
  ?inst <- (object  (is-a  ~x))
   =>
(printout t ?inst  crlf)
 )
```

بيطبع قيم الـ instance المتساوية فى الـ x,y

بيطبع عنوان الـ instance بتاع الـ x,y

بيطبع عنوان الـ instance اللى مش من الكلاس x

③ Properties of variables

(create-accessor read OR read-write)

- يعنى ممكن اخلى المتغير للقراءة فقط او للقراءة وادخال قيم ايضاً.

EX: (slot x1 (create-accessor read))

EX: (slot Pi (create-accessor read) (default 3.14))

- الـ default هنا عشان لو مدخلتش قيمة يديني قيمة افتراضية.

④ unmake-instance → to delete instance

```
EX: (defclass  human (is-a  USER)
       (slot nam (create-accessor  read-write))
       (slot age (create-accessor  read-write))
       )
(defrule  R1
  (initial-fact)
=>
(make-instance  [h1]  of human (nam ahmed) (age 20))
(make-instance  [h2]  of human (nam ali) (age 29))
(make-instance  [h3]  of human (nam hoda) (age 36))
(make-instance  [h4]  of human (nam noha) (age 19))
)
(defrule  R2
  ?x <- (object (is-a human) (age ?a))
(test (> ?a 25))
=>
(unmake-instance ?x)
)
```

هنا بحذف الناس اللى اعمارهم اكبر من 25 سنة

⑤ Slot

→ Single (slot name)

→ multi-value (multislot degrees)

```
EX: (defclass color (is-a USER)
      (multislot prm (create-accessor read)
          (default red green blue))
      )
```

- ال prm بتاخد قيم متعددة.

⑥ modify-instance

- Syntax: (modify-instance اسم الinstance اسم المتغير ((القيمة الجديدة

```
EX: (defclass A (is-a USER)
        (slot foo)
        (slot bar))
(make-instance a of A)
(modify-instance a (foo 0) )
```

⑦ Send & Put → تخصيص قيمة لمتغير فى ال instance

- Syntax: (send [عنوان ال instance] put-قيمة اسم المتغير اللى هخصصله القيمة)

```
EX_complete (page 142):
(send [stu2] put-HighScore 150)
```

6] Files

① Open

•(open "مسار الملف" logicalName "mode") → (mode w,r,r^+,a)

- الـ logicalName دا اسم الملف ان اللى بحدده عشان هستخدمه فالكود بس ملوش علاقة باسم الملف نفسه.

- For example "D:\\ filename.txt"

- الـ w : بمعنى ان الملف موجود وعاوز اكتب فيه.
- الـ r : بمعنى ان الملف للقراءة فقط.
- الـ r^+ : بمعنى ان الملف للقراءة والكتابة.
- الـ a : وتعنى اضافة او انشاء نلف جديد.

- لو مضفتش الـ mode هتبقى القيمة الإفتراضية r.

EX: (open "D:\\ test.txt" F1 "a")

- كدا عمل ملف جديد اسمه test ونوع البيانات اللى بيخدها text.

① Close → to close file

•(close logicalName)

EX: (close F1)

EX: (close) → close all files

③ Rename

•(rename "الاسم القديم" "الاسم الجديد")

EX: (rename "D:\\ test1.txt" "D:\\ output.txt")

④ Remove

•(remove "filePath" OR "fileName")

EX: (remove "D:\\ output.txt")

⑤ Printout → To write in file

- (printout logicalName "text")

EX: (printout F1 "Hello World")

⑥ Read

- (read logicalName)

EX: (read F1)

- عشان اقرأ من ملف لازم اكون عامل open للملف ومحدد الـ mode بتاعه بـ r, r^+.
- لو وصل لنهاية الملف او لو الملف فاضى يطلعلى رسالة End Of File(EOF).

⑦ Redline → لقراءة سطر سطر

- (readline logicalName)

EX:- Read all data lines in the file

```
(defrule readData
  =>
(close)
(open "D:\\myText.txt" myfile "r")
(while TRUE do
   (bind ?d (readline myfile))
(if (eq ?d EOF)
     then (break))
(printout t ?d crlf)
)
(close myfile)
)
```