FUNCIONES DE GESTIÓN DE FUNCIONES	OBSERVACIONES
(apply función lista) *Transmite una lista de argumentos a la función especificada*	• La función apply puede utilizarse con funciones integradas del programa (subrs) y funciones definidas por el usuario (las creadas con defun o lambda).
(defun sím list-argumentos expr ...) *Define una función*	• La función defun define una función con el nombre sím (el nombre de la función aparece entrecomillado automáticamente). Detrás del nombre de la función aparece una lista con los argumentos (puede no haberlos), que puede estar seguida por una barra oblicua y los nombres de uno o más símbolos locales para la función. Esta barra oblicua debe ir separada del primer símbolo local y del último argumento, si existe, por un espacio como mínimo. Si no declara ningún argumento ni símbolo local, debe escribir un conjunto vacío de paréntesis tras el nombre de la función. • La función defun devuelve el nombre de la función que se va a definir. Al invocar esta función, sus argumentos se evalúan y se asocian a los símbolos correspondientes. Puede utilizar los símbolos locales en la función sin necesidad de cambiar sus asociaciones con otros niveles exteriores. La función devuelve el resultado de la última expresión evaluada; Las expresiones anteriores sólo tienen efectos secundarios. • No utilice nunca el nombre de una función interna o un símbolo como sím, ya que quedaría inaccesible. Para obtener una lista de funciones internas y definidas anteriormente, véase "atoms-family".
(eval expr) *Devuelve el resultado de evaluar una expresión AutoLISP*	
(lambda argumentos expr ...) *Define una función anónima*	• Se utiliza lambda cuando los recursos usados para definir una función nueva no estén justificados. Esta función también hace que la intención del programador sea más aparente al ejecutar la función en el punto donde debe usarse. lambda devuelve el valor de su última expr, y suele usarse conjuntamente con apply y/o mapcar para ejecutar una función de una lista.
(trace función ...) *Facilita la depuración de AutoLISP*	
(untrace función ...) *Borra el indicador de rastreo de las funciones especificadas*	

METODOLOGIA DE USO DE LAS FUNCIONES DE GESTIÓN DE FUNCIONES

Un módulo ejecutable queda definido en AutoLISP mediante la función DEFUN; dado que ésta es, a su vez, una función más de la biblioteca, resulta que se pone de manifiesto una característica relevante: *todos los recursos fundamentales del entorno de trabajo se construyen a partir de la misma estructura: la lista*; esta circunstancia se mantendrá con los nuevos elementos que se estudiarán en los próximos capítulos.

El primer argumento de la función DEFUN, su nombre de función, determina si se trata de una principal o una subfunción. Una función LISP de tipo principal se ejecuta desde el entorno de AutoCAD como si fuera un nuevo comando; una subfunción se utiliza para ser invocada por otra. El nombre de una función principal tiene siempre el prefijo **C:** mientras que el de una subfunción no lo puede tener. Así, por ejemplo, el siguiente código constituye un nuevo comando de AutoCAD que solicita tres puntos y una distancia y dibuja la recta que pasa por los dos primeros, la circunferencia de centro el tercero y de radio la distancia introducida y los puntos de intersección de ambas entidades:

```
( defun C:interc ( )
        ( setq     pt1 ( getpoint "\nPUNTO PT1 DE LA RECTA" )
                   pt2 ( getpoint "\nPUNTO PT2 DE LA RECTA" )
                   C   ( getpoint "\nCENTRO DE LA CIRCUNFERENCIA" )
                   R   ( getdist  "\nRADIO DE LA CIRCUNFERENCIA" )
        )
        ( setq  ang  ( angle pt1 pt2 )
                   aux1 ( polar c ( + ang ( / pi 2 ) ) 10.0 )
                   aux2 ( inters pt1 pt2 c aux1 nil )
                   d    ( distance aux2 c )
                   x    ( sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
        )
        ( setq     s2 ( polar aux2 ang x )
                   s1 ( polar aux2 ( + ang pi ) x )
        )
        ( command "linea" pt1 pt2 "" )
        ( command "circulo" c r )
        ( command "punto" s1 )
        ( command "punto" s2 )
)
```

Tras cargar la función (ver función LOAD más adelante en este capítulo), la ejecución de esta función se llevará a cabo de la siguiente forma:

```
Comando: interc
PUNTO PT1 DE LA RECTA ( se designa un pto )
PUNTO PT2 DE LA RECTA ( se designa un pto )
CENTRO DE LA CIRCUNFERENCIA ( se designa un pto )
RADIO DE LA CIRCUNFERENCIA ( se designan dos ptos )
```

Toda función, sea principal o no, debe presentar una lista, a continuación de su nombre, que contiene sus argumentos de la función y sus variables locales. Para distinguir unos de otros dentro de la lista se incorpora una barra inclinada como separador. Las funciones principales no pueden contener argumentos de definición pero sí variables locales, de modo que la función INTERC recientemente construida puede ser escrita de dos modos diferentes

```
MODO1. Sin variables locales:
( defun C:interc ( )
        cuerpo de la función definida en la página anterior
)
MODO 2. Con variables locales:
( defun C:interc ( / pt1 pt2 c r ang aux1 aux2 d x s1 s2 )
        cuerpo de la función definida en la página anterior
)
```

La determinación de las variables locales y globales depende del programador; por defecto, una variable no declarada como local es global. En cuanto a su vigencia, se sigue la norma propia de todo lenguaje de programación, es decir, las variables locales sólo conservan valor asociado durante el tiempo que se ejecuta la función y desaparecen de la pila de variables cuando aquélla finaliza. En cualquier caso, la invocación a la función no cambia

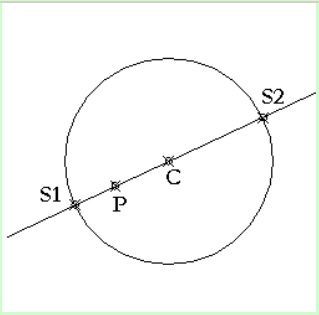


Figura 1.11.1.1

Supóngase ahora que se desea escribir una función que, a partir de dos datos de punto, C y P, y un dato de radio R, calcule la potencia de P respecto de la circunferencia de centro C y radio R. Por definición, ese valor puede ser calculado como el producto de distancias P-S1 * P-S2, donde S1 y S2 son los puntos de corte de la recta P-C con la circunferencia de centro C y radio R. El croquis se muestra en la figura 1.11.1.1.

En este caso, la obtención de los puntos de corte de recta y circunferencia puede plantearse como una operación auxiliar del cometido principal de la función, que es obtener la potencia del punto P respecto de la circunferencia dada y el programa puede construirse según la siguiente secuencia:

1. *Obtener los datos C, R y P.*
2. *Obtener los puntos S1 y S2 de corte de la recta C-P con la circunferencia C-R como variables globales.*
3. *Calcular el producto P-S1 * P-S2.*

El código correspondiente se ofrece a continuación:

```
( defun interc ( pt1 pt2 c r )

        ( setq  ang  ( angle pt1 pt2 )
                aux1 ( polar c ( + ang ( / pi 2 ) ) 10.0 )
                aux2 ( inters pt1 pt2 c aux1 nil )
                d    ( distance aux2 c )
                x    ( sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
        )
        ( setq     s2 ( polar aux2 ang x )
                   s1 ( polar aux2 ( + ang pi ) x )
        )
)
; *********************************************************************
( defun c:potencia ()
; Paso 1 : obtener los datos.
        ( setq c ( getpoint "\nINTRODUCIR CENTRO DE LA CIRCUNFERENCIA")
                 r ( getdist  "\nINTRODUCIR RADIO DE LA CIRCUNFERENCIA")
                 p ( getpoint "\nINTRODUCIR PUNTO P " )
        )
; Paso 2: obtener S1 y S2.
        ( interc p c c r )
; Paso 3:  calcular el producto de distancias.
        ( setq sol ( *  ( distance p s1 )
                        ( distance p s2 )
                   )
        )
        ( print (strcat "potencia de P = " ( rtos sol ) ) )
)
```

La obtención de los datos se realiza en el paso 1 del programa principal **c:potencia**; sobre el paso 2 surgen los siguientes comentarios:

- INTERC, que en este caso no lleva prefijo, constituye una subfunción o lo que en otros lenguajes se denomina una función creada por el usuario.
- Una función creada por el usuario es definida mediante la función DEFUN e invocada como una función LISP convencional, a partir del nombre con el que se ha definido.
- La función creada por el usuario puede tener argumentos o no; en el primero de los casos, éstos se incorporan en la lista que va a continuación del nombre de la función. Esta lista lleva, en consecuencia, lo que se denominan argumentos de definición.
- Las variables que se utilizan en la función creada son, por defecto, globales. En nuestro ejemplo, todas las variables que se utilizan en INTERC, a excepción de los argumentos, son globales.
- Si en una subfunción se desea declarar variables locales, se procede del mismo modo que se ha comentado para la función principal; por ejemplo, si se hubiese deseado declarar como variables locales todas las de INTERC, a excepción de S1 y S2, la función habría quedado definida así:

```
( defun interc ( pt1 pt2 c r /  ang aux1 aux2 d x )
        . . .
)
```

- La invocación **(interc p c c r)** tiene el funcionamiento clásico de las invocaciones: la corriente del programa se transfiere a la función invocada, los argumentos de la invocación (o argumentos de uso) pasan a valorar a los argumentos de definición y, una vez finalizada la función llamada, la corriente del programa es transferida a la línea siguiente a aquélla que provocó la llamada. Los argumentos se pasan por valor y ha de mantenerse la conocida regla: los argumentos de definición y los de uso han de corresponder en orden número y tipo.
- Puesto que los argumentos se pasan por valor, las variables y simbolos de nombre coincidente están preservadas en ambas funciones, la que llama y la llamada. Por ejemplo, posibles cambios de las variables C y R en el programa INTERC no afectan a las variables C y R del programa C:POTENCIA.
- Una función AutoLISP creada por el usuario devuelve, al ser invocada, el resultado de la última expresión evaluada. Si la función utiliza variables globales para los valores que obtiene, esto no sipone ningún inconveniente; así por ejemplo, en nuestra función C:PRINCIPAL la invocación a INTERC ha servido para valorar las variables globales S1 y S2, mientras que el valor devuelto por la invocación **(interc p c c r)** se ha perdido. No obstante S1 y S2 pueden ser usadas desde el programa principal.
- Sin embargo, lo usual es proponer que todas las variables de un subprograma sean locales y combinar la invocación con una orden **setq** para recoger el valor devuelto por la llamada; en nuestro ejemplo:

```
( defun interc ( pt1 pt2 c r /  ang aux1 aux2 d x s1 s2)
        . . .
)
( defun c:potencia ()
        . . .
        ( setq sol ( interc p c c r ) )
        . . .
)
```

Sin embargo, de los principios propuestos se deduce que tal llamada es errónea, porque las variables S1 y S2 se han declarado como locales y, por lo tanto, no se pueden usar directamente desde el programa principal; por otra parte, la variable SOL del programa C:POTENCIA recibe el valor de S1, que es el último evaluado en INTERC.

En resumen, la gestión de los valores devueltos por una función llamada sigue estos principios:

- Si se utilizan variables globales para los valores a devolver, la función puede ser invocada directamente sin combinarla con una función SETQ. No obstante, esta técnica no es recomendable porque las variables de una función auxiliar deben estar preferentemente declaradas como locales.
- Si la función devuelve un único valor, es recomendable que la variable que lo recibe en el subprograma sea local y que la llamada se efectúe combinando la invocación con una orden setq.
- Si la función devuelve varios valores, es recomendable que las variables que lo reciben en el subprograma sean locales y que todos ellos se integren en una lista; de ese modo la función devolverá el contenido de la lista y la llamada se podrá de nuevo efectuar combinando la invocación con una orden setq.

Los siguientes cuadros resumen los posibles casos de invocación y gestión de variables devueltas:

CASO 1. UNO O VARIOS VALORES, DEVUELTOS COMO VARIABLES GLOBALES

```
(defun subpr ( a b c )
        . . .
        ( setq sol1 ( . . . )
                sol2 ( . . . )
                sol3 ( . . . )
        )
)
; *******************************************************
( defun c:pral ()
        . . .
        ( subpr x y z )
; a partir de esta invocación las variables sol1, sol2 y sol3 ya se pueden utilizar.
        . . .
)
```

CASO 2. UN ÚNICO VALOR, DECLARADO COMO VARIABLE LOCAL

```
(defun subpr ( a b c / sol1)
        . . .
        ( setq sol1 ( . . . ) )
; CONDICIÓN: la última expresión evaluada en el subprograma es la que ;retorna la programa que lo llama.
)
; *******************************************************
( defun c:pral ()
        . . .
        ( setq a ( subpr x y z ) )
; la variable a recibe el valor de la variable local sol1.
        . . .
)
```

CASO 3. VARIOS VALORES, DECLARADOS COMO VARIABLES LOCALES

```
(defun subpr ( a b c / sol1 sol2 sol3 lsol)
        . . .
        ( setq    sol1 ( . . . )
                  sol2 ( . . . )
                  sol3 ( . . . )
        )
        ( setq lsol ( list sol1 sol2 sol3 ) )
; CONDICIÓN: todos los valores a devolver se empaquetan en una lista, ; La estructura que devuelve la
función es ( sol1 sol2 sol3 ).
)
; *******************************************************
( defun c:pral ()
        . . .
        ( setq l ( subpr x y z ) )
; la variable l recibe la lista ( sol1 sol2 sol3 ) y puede ser ya manipulada desde el programa principal
        . . .
)
```

Las intrucciones de generación y manejo de listas serán tratadas con más detalle en el capítulo siguiente. Para finalizar, se ofrece en la tabla siguiente los tipos posibles de funciones y subfunciones en relación con sus argumentos y variables locales:

TIPO GENÉRICO DE FUNCIÓN:	***INVOCACIÓN***
(DEFUN SUBPR ( ) . . .)	(SUBPR)
(DEFUN SUBPR (A B) . . .)	(SUBPR X Y)
(DEFUN SUBPR (A B / M N P) . . .)	(SUBPR X Y)
(DEFUN SUBPR (/ M N P) . . .)	(SUBPR)
(DEFUN C:PRAL ( ) . . .)	ORDEN: PRAL
(DEFUN C:PRAL (/ L F T) . . .)	Id. las dos anteriores

EJEMPLOS SOBRE FUNCIONES MONOLINEA

La función **apply** suele ser ejecutada en dos contextos diferentes: el primero se produce cuando se desea aplicar una función de AutoLISP a todos los elementos de una lista; el segundo cuando se combina con una función **Lambda**, en cuyo caso se está efectuando lo que en otros lenguajes de programación se entiende como una función monolínea creada por el usuario.

```
( apply '+ '( 2 3 4 5 6 ) )
20
```

Un ejemplo ilustrativo del primero de los contextos es el siguiente: se desea obtener la suma de todos los elementos de una lista:

Obsérvese que, en este contexto, la función AutoLISP de suma (el operador +) va precedido del símbolo **quote** (el apóstrofo). Ello es debido a que, según el principio general de construcción de funciones LISP:

(función arg-1 arg-2 . . .),

sólo la primera palabra o símbolo después del paréntesis izquierdo puede ser un nombre de función, mientras que el resto deben ser argumentos de esa función. En nuestro caso, el nombre de función es APPLY y el operador + no tiene carácter de función, sino de argumento; por eso debe ir precedido del apóstrofo (eq uivalente a la función QUOTE) que le elimina su carácter ejecutable y lo transforma en un símbolo convencional. En cuanto al segundo argumento, la lista (2 3 4 5 6), también es precisop precederla del apóstrofo para indicarle al intérprete LISP que no se trata de una función ejecutable. En el capítulo siguiente, al estudiar las funciones de manejo de listas, se ampliarán conceptos a este respecto.

Una aplicación muy frecuente de apply según este primer contexto es la siguiente:

```
( command "pol" )
 (apply 'command '( ( 10 10 ) ( 20 20 ) ( 10 40 ) ) )
( command "" )
```

fragmento que dibuja una polilínea que sale del punto (10 10) y pasa por el (20 20) y finaliza en el (10 40). Técnicamente descritas, la primera lista del fragmento lanza una polilínea que deja el sistema esperando la contestación a la pregunta *del punto:*; la función **apply** envía a la pregunta de la orden polilínea el punto (10 10) y luego los otros; finalizada la orden **apply**, todavía la orden polilínea queda abierta; con el último **command** se cierra la instrucción. La secuencia descrita es, a efectos de diálogo, equivalente a esta:

```
( command "pol" )
( command  '( 10 10 ) )
( command  '( 20 20 ) )
( command  '( 10 40 ) )
( command "" )
```

donde la razón para usar el apóstrofo en las listas de punto es la misma expuesta en la página anterior. Puede comprobarse que la función apply permite notaciones muy compactas para ciertas operaciones que se han de aplicar sobre todos los elementos de una lista

En el segundo contexto, el primer argumento de la función **apply** es la función **lambda** y el segundo es una lista de los argumentos sobre los que se aplica dicha función. Por ejemplo:

```
(apply '(lambda (x y) ( sqrt (+ (expt x 2) (expt y 2)))) '(3 4))
5.0
```

donde lambda (x y) es una forma de denotar la expresión *f(x,y) = sqrt (x^2 + y^2)*. Téngase presente que la primera lista de la función **lambda** es la lista *de argumentos de definición* de la función y la segunda lista es la definición en sí de la función en términos LISP; en efecto, es la definición de una función monolínea

Por lo demás, se entiende fácilmente que cuando lambda aparece dentro de una función apply, el último argumento es una lista sobre la que se aplica la función lambda, es decir, en realidad es la lista con los *argumentos de uso*. En resumen, la función apply está proponiendo la expresión f(3,4) = *sqrt (3^2 + 4^2) = 5.0*.

En cuanto a loa apóstrofos, caben las mismas consideraciones que las expuestas en la página anterior.

La función mapcar, que se estudiará en el capítulo siguiente, suele ser utilizada también en combinación con la función lambda para aplicar una función a los elementos de varias listas simultáneamente.

EGI-CAD

Universidad de Cantabria

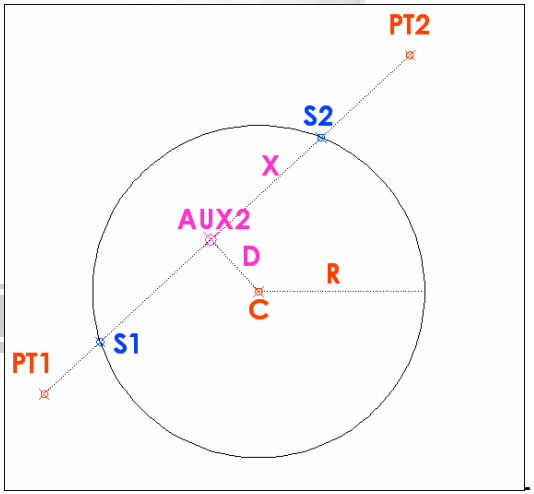
Dpto. de Ing. Geográfica

Grupo EGICAD

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Índice


```
(setq pt1 (getpoint "\nPUNTO PT1 DE LA RECTA: ")
      pt2 (getpoint "\nPUNTO PT2 DE LA RECTA: ")
      C (getpoint "\nCENTRO DE LA CIRCUNFERENCIA: ")
      R (getdist "\nRADIO DE LA CIRCUNFERENCIA: ")
  )
(setq ang (angle pt1 pt2)
      aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
      aux2 (inters pt1 pt2 c aux1 nil)
      d (distance aux2 c)
      x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
  )
(setq s2 ( polar aux2 ang x )
      s1 (polar aux2 ( + ang pi ) x)
  )
( command "linea" pt1 pt2 "" )
( command "circulo" c r )
( command "punto" s1)
( command "punto" s2)
```



EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Índice


```
(defun c:interc()
(setq pt1 (getpoint "\nPUNTO PT1 DE LA RECTA: ")
      pt2 (getpoint "\nPUNTO PT2 DE LA RECTA: ")
      C (getpoint "\nCENTRO DE LA CIRCUNFERENCIA: ")
      R (getdist "\nRADIO DE LA CIRCUNFERENCIA: ")
   )
(setq ang (angle pt1 pt2)
      aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
      aux2 (inters pt1 pt2 c aux1 nil)
      d (distance aux2 c)
      x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
   )
(setq s2 ( polar aux2 ang x )
      s1 (polar aux2 ( + ang pi ) x)
   )
( command "linea" pt1 pt2 "" )
( command "circulo" c r )
( command "punto" s1)
( command "punto" s2)
)
```

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Índice

→ Función LISP de tipo principal, se ejecuta desde AutoCAD como si fuera un nuevo comando.

(defun c:interc())

→ Lista a continuación del nombre de función. Contienen dos tipos de datos, separados por una barra inclinada / :

- → Argumentos de la función
- → Variables Locales

→ Las funciones principales no pueden contener argumentos, solo variables locales.

→ El tiempo de vida de las variables locales se limita al tiempo de ejecución de la función. Desaparecen de la pila de variables cuando ésta finaliza.

→ Las variables no declaradas como locales son globales. Mantendrán su valor hasta que cerremos AutoCAD

(defun c:interc (/ pt1 pt2 c r ang aux1 aux2 d x s1 s2))

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Supongamos ahora que eliminamos la parte de toma de datos y la parte de dibujo de la función, creando una subfunción únicamente de análisis:

```
(defun interc()
(setq ang (angle pt1 pt2)
        aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
        aux2 (inters pt1 pt2 c aux1 nil)
        d (distance aux2 c)
        x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
    )
(setq s2 ( polar aux2 ang x )
        s1 (polar aux2 ( + ang pi ) x)
    )
)
```


→ ¿Qué ocurre con pt1, pt2, c y r? Deberán ser argumentos de la función:

```
(defun interc ( pt1 pt2 c r / ang aux1 aux2 d x s1 s2 )
```

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Ejecución de la función en consola:

```
_$ (intercc (list 0 0) (list 5 5) (list 3 3) 5)
    (-0.535534 -0.535534)
```

```
(defun interc( pt1 pt2 c r )
(setq ang (angle pt1 pt2)
        aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
        aux2 (inters pt1 pt2 c aux1 nil)
        d (distance aux2 c)
        x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) )
    )
(setq s2 ( polar aux2 ang x )
        s1 (polar aux2 ( + ang pi ) x)
    )
)
```


→ ¿Qué devuelve la función? El último valor calculado: s1

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Índice

→ Intercc, que en este caso no lleva prefijo, es una subfunción, o función creada por el usuario.

→ Una función creada por el usuario es definida mediante DEFUN e invocada como una función LISP convencional a partir de su nombre.

→ Puede tener argumentos o no.

→ Sus variables son por defecto globales, salvo que las definamos como locales en la lista de inicio.

→ Devuelve la última expresión evaluada, en este caso S1. De todos modos, al ser variables globales, podemos acceder a todas ellas.

→Lo usual es proponer que todas las variables sean locales y combinar la invocación con una orden setq desde la función principal.

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

```
(defun C:inters()
(setq pt1 (getpoint "\nPUNTO PT1 DE LA RECTA: ")
      pt2 (getpoint "\nPUNTO PT2 DE LA RECTA: ")
      C (getpoint "\nCENTRO DE LA CIRCUNFERENCIA: ")
      R (getdist "\nRADIO DE LA CIRCUNFERENCIA: ") )
(setq sol ( interc pt1 pt2 C R ) )
( command "linea" pt1 pt2 "" )
( command "circulo" c r )
( command "punto" s1)
( command "punto" s2) )

(defun interc( pt1 pt2 c r / ang aux1 aux2 d x s1 s2 )
(setq ang (angle pt1 pt2)
      aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
      aux2 (inters pt1 pt2 c aux1 nil)
      d (distance aux2 c)
      x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) ) )
(setq s2 ( polar aux2 ang x )
      s1 (polar aux2 ( + ang pi ) x) ) )
```

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

→ Devolución de los datos.

```
(defun interc( pt1 pt2 c r / ang aux1 aux2 d x s1 s2 sol)
(setq ang (angle pt1 pt2)
        aux1 (polar c (+ ang ( / pi 2.0 ) ) 10.0)
        aux2 (inters pt1 pt2 c aux1 nil)
        d (distance aux2 c)
        x (sqrt ( - ( expt r 2 ) ( expt d 2 ) ) ) )
(setq s2 ( polar aux2 ang x )
        s1 (polar aux2 ( + ang pi ) x) )
(setq sol ( list s1 s2) )
)
```

```
(defun C:inters()
.....
(setq sol ( interc pt1 pt2 C R ) )
.....
)
```

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

→ Función para captar los datos iniciales.

```
(defun datos( / pt1 pt2 C R)
(setq pt1 (getpoint "\nPUNTO PT1 DE LA RECTA: ")
      pt2 (getpoint "\nPUNTO PT2 DE LA RECTA: ")
      C (getpoint "\nCENTRO DE LA CIRCUNFERENCIA: ")
      R (getdist "\nRADIO DE LA CIRCUNFERENCIA: ")
      )
 )
```

→¿Qué devuelve esta función? R

→ Para poder devolver todos los datos, es necesario crear una lista con los mismos:

```
(setq resultado (list pt1 pt2 C R))
```



```
_$ (Datos)
    ((0.0 0.0 0.0) (5.0 5.0 0.0) (3.0 3.0 0.0) 5.0)
```

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

Índice

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

→ Si quisieramos llamar a la función de captación de datos desde la función principal:

```
(defun C:inters( / pt1 pt2 c r)
(setq entrada ( datos ) )  → ((0.0 0.0 0.0) (5.0 5.0 0.0) (3.0 3.0 0.0) 5.0)
(setq pt1 (car entrada)
(setq pt2 (cadr entrada )
(setq c ( caddr entrada )
(setq r ( cadddr entrada )
(setq sol ( interc pt1 pt2 C R ) ) → ((-0.535 -0.535 0.0) (6.535 6.535 0.0))

( command "linea" pt1 pt2 "" )
( command "circulo" c r )
( command "punto" s1)
( command "punto" s2)
)
```

→ Para acceder a los distintos datos, utilizaremos las funciones de manejo de listas.

EGI-CAD

Universidad de Cantabria

Dpto. de Ing. Geográfica

Grupo EGICAD

TEMA: FUNCIONES DE GESTIÓN DE FUNCIONES.

Punto 1.11 Teoría WEBCT

CONCLUSIONES:

→ Función Principal: siempre precedida de C:. Se ejecutará como comando de AutoCAD

(defun C:inters (/)

→ Lista a continuación del nombre:

→ Argumentos (nunca en función principal, sólo en subfunciones)

→ Variables Locales. Por defecto todas las variables son globales salvo que se especifique lo contrario.