Capítulo 4

Funciones

¿Qué son?¿Para qué sirven?

- Son un grupo de sentencias bajo el mismo nombre que realizan una tarea específica.
- Sirven para facilitar la resolución de problemas mediante la aplicación del paradigma **“Dividir y Conquistar”.**

Diferencia entre El Programa y las Funciones

- Las funciones y los programas se parecen mucho, pero difieren:
  - Los programas son usados por un usuario externo.
  - Las funciones son utilizadas por un programador.
  - El usuario de un programa que imprima “Hola Mundo” no conoce que es la función disp.
  - El programador que usa disp o mprintf no siempre conocerá explícitamente como ésta hace para mostrar información en pantalla.
  - El programador que escribió disp o mprintf conoce exactamente su funcionamiento interno.

Conceptos Básicos

- Función
  - Grupo de sentencias bajo el mismo nombre que realizan una tarea específica.
- Llamada a una función
  - Ejecuta el grupo de sentencias de una función.
- Retorno
  - Una vez “llamada” la función, esta hace su trabajo, y regresa al mismo punto donde fue llamada.

Funciones

- Vamos a conocer tres cosas muy importantes sobre las funciones:
  - ¿Cómo se declaran?
  - ¿Cómo se implementan?, y
  - ¿Cómo se usan?

Creación de Funciones

- De forma similar a las variables, las funciones deben ser declaradas:
- La forma de declarar una función es siguiendo la forma predefinida:

```
function [res1, res2, ...] = nombrefuncion(par1, par2, ...)
...
...
endfunction
```

- El significado de res1 es resultado 1 o también parámetro de salida 1. El significado de par2 es parámetro de entrada 2 o simplemente parámetro 2.
- Cuando la función tiene un único resultado, el esquema puede ser simplemente:

```
function result = nombrefuncion(par1, par2, ...)
...
...
endfunction
```

Implementación de Funciones

```
function [resultado] = potencia(base, exponente)
...
sentencias
...
endfunction
```

```
function [conversion] = farenheitACelsius(farenheit)
...
sentencias
...
endfunction
```

Entre function.. endfunction se escriben las sentencias que ejecutan lo que debe realizar la función

¿Cómo Devuelven valores?

- Si la función debe generar un valor, devolverá el ultimo valor de la variable dentro del cuerpo de la función.
- Endfunction**,** especifica que la función debe terminar, devolviendo el valor calculado.
- Hay funciones que no devuelven datos, solo ejecutan un grupo de sentencias

Uso de Funciones

- En Scilab hay dos tipos de programas: los scripts y las funciones.
- Un script es simplemente una secuencia de órdenes de Scilab. No tiene parámetros (“argumentos”) de entrada ni de salida. En cambio una función sí los tiene.

Uso de Funciones

- Generalmente el nombre de los archivos de funciones tienen la extensión .sci.
- El archivo que contiene las funciones se debe cargar mediante la orden

  exec('C:\.....\scilab\ejemplos\func.sci');

  en el archivo del script principal .sce para que la función pueda ser usada en el programa principal.

Archivo .sce

```
//programa con funciones
exec('C:\scilab\ejemplos\func.sci');
base=input("Ingrese base: ")
exponente=input("Ingrese exponente: ")
resultado=potencia(base,exponente)
disp(resultado)
farenheit=input("Ingrese farenheit: ")
resultado=farenheitACelsius(farenheit)
disp(resultado)
```

Uso deFunciones

El siguiente archivo llamado c:\funciones\misfunc.sci tiene varias funciones

```
//-------------------------------------------------------
function [x, y] = polarCart(r, t)
// Conversion de coordenadas polares a cartesianas.
x = r*cos(t)
y = r*sin(t)
endfunction
//-------------------------------------------------------
function [x, y] = polarCartGr(r, t)
// Conversion de coordenadas polares a cartesianas,
// el angulo esta dado en grados.
[x, y] = polarCart(r, t*%pi/180)
endfunction
//-------------------------------------
```

Otra característica de las funciones es que una función puede llamar una o varias funciones.
La función polarCartGr utiliza la función polarCart.

Ejemplo

- Escriba un programa que utilice las funciones disponibles en el archivo misfunc.sci

  Una vez cargado el archivo, las funciones se pueden utilizar.
  Por ejemplo, son válidas las siguientes órdenes:

```
[x1, y1] = polarCart(2, 0.7854)
[u, v] = polarCartGr(3, 30)
```

```
//programa con funciones
exec('C:\funciones\misfunc.sci');
[x1, y1] = polarCart(2, 0.7854)
[u, v] = polarCartGr(3, 30)
disp(x1,y1)
disp (u,v)
```

Funciones

- Cuando una función produce más de un resultado, también se puede utilizar asignando menos de los resultados previstos.
- Por ejemplo, la utilización completa de polarCartGr puede ser

  [a, b] = polarCartGr(3,30)

  lo cual produce el resultado

  b = 1.5

  a = 2.5980762
- En cambio, la orden

  c = polarCartGr(3,30)

  produce el resultado

  c = 2.5980762

Funciones Matematicas

- Scilab tiene predefinidas muchas funciones matemáticas. A continuación está la lista de las funciones elementales más comunes.

abs : valor absoluto
acos : arcocoseno
acosh : arcocoseno hiperbolico
asin : arcoseno
asinh : arcoseno hiperbolico
atan : arcotangente
atanh : arcotangente hiperbolica
ceil : parte entera superior
cos : coseno
cosh : coseno hiperbolico
cotg : cotangente
coth : cotangente hiperbolica
exp : funcion exponencial: *ex*

fix : redondeo hacia cero (igual a int)
floor : parte entera inferior
int : redondeo hacia cero (igual a fix)
log : logaritmo natural
log10 : logaritmo decimal
log2 : logaritmo en base dos
max : maximo
min : minimo
modulo : residuo entero
rand : n´umero aleatorio
round : redondeo
sin : seno
sinh : seno hiperbolico
sqrt : raiz cuadrada
tan : tangente
tanh : tangente hiperbolica

…Funciones

- Otra funcion matematica, dos parametros de entrada, es modulo.
- Sus dos parametros deben ser enteros. El resultado es el residuo de la division entera.

  modulo(17,5)

  da como resultado 2.
- Para tener informacion mas detallada sobre alguna funcion basta con digitar help y a continuacion el nombre de la funcion o de la orden. Por ejemplo

  help floor
- Obviamente se requiere que la funcion floor exista. Si no se conoce el nombre de la funcion, pero se desea buscar sobre un tema, se debe utilizar apropos. Por ejemplo:

  apropos polynomial

  da informaci´on sobre las funciones que tienen que ver con polinomios. En cambio,

  help polynomial

  informa que no hay manual para polynomial.

Funciones Predicado

- Las funciones que retornan valores lógicos se conocen como **funciones predicado.**
- Llamar a una función de predicado es equivalente a hacer una pregunta donde la respuesta puede ser Verdadera (TRUE) o Falsa (FALSE).

La Verdad detrás de las Funciones

- Hay una mayor complejidad de la mostrada, en el uso de funciones. Tomemos como ejemplo el siguiente programa:

Si vemos el programa principal y la función como un todo, puede resultar un tanto confuso:
- **Hay dos variables con nombre i**
- En el programa se usa la variable i como el número al que se le calcula el factorial.
- **En la función la variable a la que se le calcula el factorial se llama n**

```
//Llamada a funciones
exec('C:\scilab\funciones\misfunc.sci')
;

for i=0:10
        mprintf("%i \n",factorial(i))
end
function [fact]=factorial(n)
        fact = 1;
        for i = 1:n
                fact = fact * i
        end
endfunction
```

Si vemos sólo el programa principal, tiene sentido, pues se imprime el factorial de todos los números hasta el 10

Si vemos sólo la función, tiene sentido, pues se encarga de calcular el factorial de la variable n

Paso de Argumentos a Funciones

En el programa principal, se calcula el factorial de i.
En la función se calcula el factorial de n
¿Cómo es que hay diferentes identificadores para el mismo valor?
La respuesta esta muy relacionada con el concepto de argumento:

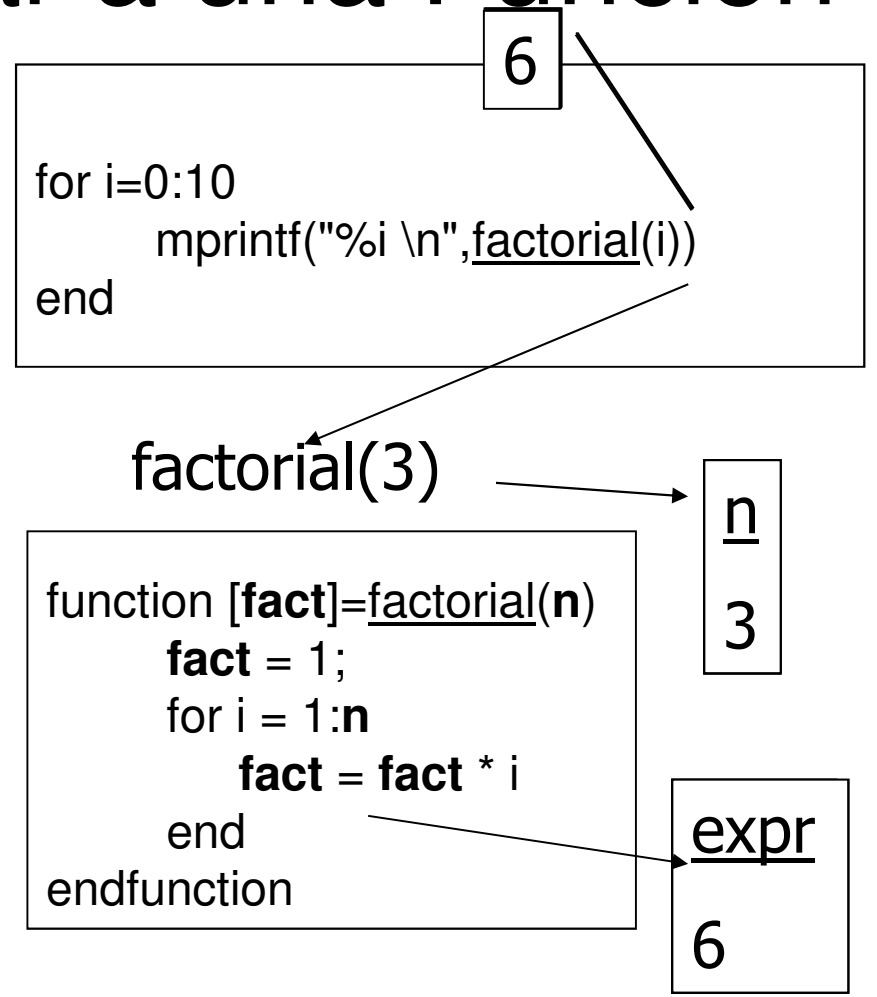
- En el programa principal, i representa el argumento enviado a la función Factorial.
- En la función Factorial, alguna variable debe recibir el enviado, para representar dicho valor. Esta variable puede tener cualquier nombre, en este caso se le dio el nombre n.
- Una variable definida en la cabecera de una función, que sirve para recibir el valor de un argumento, se conoce como **parámetro.**

```
for i=0:10
        mprintf("%i \n",factorial(i))
end


function [fact]=factorial(n)
        fact = 1;
        for i = 1:n
            fact = fact * i
        end
endfunction
```

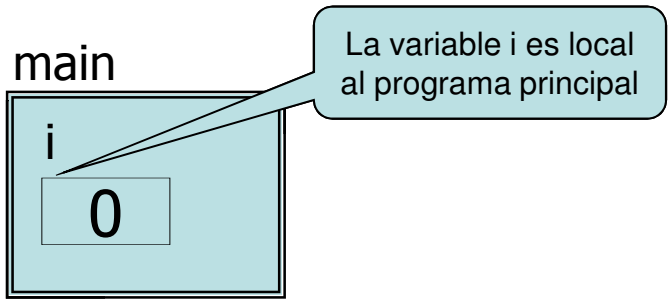
Pasos para llamar a una Función

- Se evalúan las expresiones enviadas como argumentos.
- El valor de cada argumento es copiado en orden en cada parámetro correspondiente de la función llamada.
- Se ejecutan una a una las sentencias del cuerpo de la función
- El programa que llamó a la función continúa, reemplazando en el lugar de la llamada, el valor retornado



Variables Locales

- En la función factorial se usa una variable i, y en el programa principal se usa otra variable i, pero no parece que se usaran para lo mismo, ¿son diferentes?.
- De hecho, si son diferentes.
  - Cada función puede usar sus propias variables, y estas sólo serán válidas dentro de la función, se conocen como variables locales.



Al llamar a la función Factorial, se crean 3 variables locales a Factorial, pueden tener cualquier nombre, en este caso: n, fact e i.
Las variables locales del programa aun existen, pero, no se pueden ver mientras Factorial este activa.
Cuando todo Factorial termina, retorna el valor, y las variables locales al programa permanecen iguales como antes de la llamada.

Más sobre …

Un tipo especial de funciones: Procedimientos

Procedimientos

- Existen funciones que no retornan ningún valor, como mprintf:

```
mprintf (“Hola Mundo\n”);
```

- Las funciones que no retornan nada, y que se llaman únicamente para que ejecuten su código, se les llama procedimientos.
- Muchos lenguajes de programación separan totalmente el concepto de funciones, con el de procedimientos, pero Scilab las trata de igual forma.
- Un procedimiento en Scilab, es una función sin valor de devolucion.

```
function []=menu()
```

- Los procedimientos pueden recibir tantos argumentos necesite.

Implementación de Procedimientos

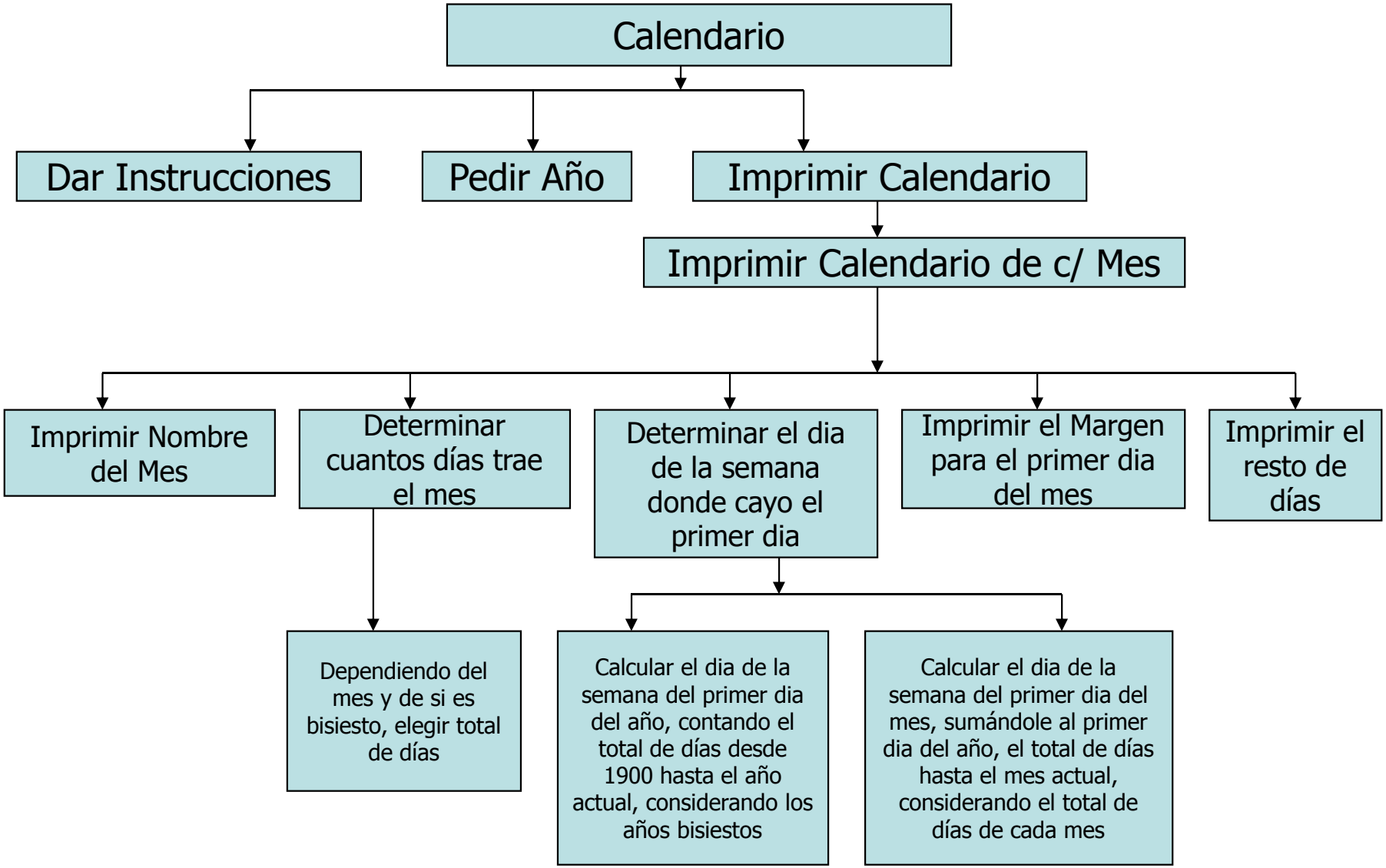
```
function []=menu()
        mprintf ("1. Tabla de Sumar\n")
        mprintf ("2. Tabla de Restar\n")
        mprintf ("3. Tabla de Multiplicar\n")
        mprintf ("4. Tabla de Dividir\n")
        mprintf ("5. Salir\n")
endfunction
```

Refinamiento

- Cuando un problema es muy grande, se busca separarlo, para resolver todo por partes. Esto es ventajoso:
  - Las partes más pequeñas son mas fáciles de entender
  - Si algo falla, el error es mas fácil de encontrar.
- Al escribir un programa, usualmente se piensa en el programa principal,y se piensa en las tareas más importantes.
- Se piensa en dividir el programa en componentes individuales, los cuales pueden ser a su vez, divididos en piezas más pequeñas.
- Esto se conoce como **diseño top-down, o refinamiento paso a paso.**

Un Problema más Grande

- Se requiere escribir un programa que muestre el calendario completo de un año dado, que no puede ser menor a 1900.


Calendario
Dar Instrucciones
Pedir Año
Imprimir Calendario
Imprimir Calendario de c/ Mes
Imprimir Nombre del Mes
Determinar cuantos días trae el mes
Determinar el dia de la semana donde cayo el primer dia
Imprimir el Margen para el primer dia del mes
Imprimir el resto de días
Dependiendo del mes y de si es bisiesto, elegir total de días
Calcular el dia de la semana del primer dia del año, contando el total de días desde 1900 hasta el año actual, considerando los años bisiestos
Calcular el dia de la semana del primer dia del mes, sumándole al primer dia del año, el total de días hasta el mes actual, considerando el total de días de cada mes

Aplicación: Juegos de Azar

Generación de Números
Aleatorios

Generación de Números Aleatorios

- Función `rand`
  - Retornar números “aleatorios“

```
i = rand();
```

  - Números Pseudoaleatorios
    - Secuencia pre-establecida de números aleatorios
    - La misma secuencia para cada llamada a la función
- Para obtener un número entero aleatorio entre 1 y n:

```
int( rand() * n )+1;
```

  - `rand() * n` retorna un número entre `0` y `n - 1`
  - Agregando `1` genera números aleatorios entre `1` y `n`

```
int( rand() * 6)+1; //Número entre 1 y 6
```