Introducción

En este artículo vamos a ver con mayor formalismo los tipos de datos que podemos encontrar en Octave. Veremos como funcionan los comandos *do-until* y *switch* y la notación de una función *function*. Finalmente daremos la solución del reto que os dejamos en el anterior número: el lomo del gato.

1. Tipos de datos, variables y expresiones

1.1. Tipos de datos

En Octave podemos distinguir diferentes tipos de datos como son los número reales, los números complejos, las matrices, las cadenas de carácteres, los tipos estructurados y celdas. Más concretamente Octave trabaja con 3 tipos tipos elementales de datos:

- Numéricos: enteros (con o sin signo, con 8, 16, 32 o 64 bits) o reales representados en coma flotante simple (32 bits) o doble (64 bits).
- Lógicos: que lo representan con 0 ó 1 (falso o verdadero, respectivamente) mediante 8 bits.
- Caracter: que se representa con 16 bits.

Octave tiene un comando que nos desvela de qué tipo de dato es una variable específica. El comando es *typeinfo(x)* y nos da la información de que tipo de dato es x. Pasemos el siguiente código por Octave,

```
a=1;
b=2+i;
c=[1 2 3;4 5 6];
d='obrigada';
e={a,b,c,d};
typeinfo(a)
typeinfo(b)
typeinfo(c)
typeinfo(d)
typeinfo(e)
```

Podemnos ver que la salida que se obtiene es la siguiente:

```
ans = scalar
ans = complex scalar
ans = matrix
ans = sq_string
ans = cell
```

Si usamos la instrucción *typeinfo()* nos devolverá la información de todos los tipos de datos con los que cuenta Octave. Para mayor información de tipos de datos y de construcción de datos estructurados puedes remitirte al tutorial oficial de Octave que puedes encontrar aquí `http://www.gnu.org/software/octave/octave.pdf`.

Para datos numéricos tenemos un comando que puede ser de utilidad y es el comando *format* que sirve para cambiar el tipo de visualización en pantalla de los datos. Por defecto está activado el formato *short*, que muestra 5 dígitos significativos. Puedes ver algunas de sus opciones en el código siguiente:

```
octave:12> pi
ans =     3.1416
octave:13> format long
octave:14> pi
ans =  3.14159265358979
octave:15> format long e
octave:16> pi
ans =    3.14159265358979e+00
octave:17> format long E
octave:18> pi
ans =    3.14159265358979E+00
octave:19> format rat
octave:20> pi
ans = 355/113
```

1.2. Variables

En programación, una variable está formada por un espacio en el sistema de almacenaje (memoria principal de un ordenador) y un nombre simbólico (un identificador) que está asociado a dicho espacio. Ese espacio contiene una cantidad o información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. El identificador, en el codigo fuente de la computadora puede estar ligado a un valor durante el tiempo de ejecución y el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa. El concepto de variables en computación puede no corresponder directamente al concepto de variables en matemática. En Octave la notación es la siguiente:

```
variable=expresión
```

Algunos comandos que pueden ser de interés relacionados a información sobre variables son los siguientes:

- *whos*: nos da la lista de las variables del espacio de trabajo con sus características: tamaños y tipos.
- *clear*: este comando borra el espacio de trabajo.
- *save* y *load*: si queremos conservar ciertas variables para usar en otro momento, podemos grabarlas con el comando save y recuperarlas en cualquier monmento con el comando *load*. Estas, se grabarán en el espacio de trabajo que estés trabajando (el cual puedes ver con el comando *cd*).

En general las variables se mantendrán es el espacio de trabajo de la sesión. Cuando se inicia una sesión de trabajo en Octave el espacio de variables estará vacío e irá almacenando las que se vayan generando. Una vez cerrada la sesión de trabajo las variables serán eliminadas. Si estás interesado en guardar alguna una variable podrás hacerlo con el comando *save* y recuperarla con el comando *load*.

1.3. Expresiones

No vamos a entrar en detalle pero vamos a dar varios ejemplos de diferentes expresiones en Octave para que puedas hacerte a la idea del potencial y de sus distintos usos a la hora de programar.

Expresiones con índices	Llamadas de funciones	Expresiones booleanas
a=[1 2;3 4;5 6;7 8]	sqrt($x^2 + y^2$)	a & & b
a([1 3],2)	ldivide(x,y)	and(a,b)
a(:,2)	eye(3)	not(a)
a(1:2:end,:)	rand()	or(a,b)
a=[a;13 13 13 13]	sqrt($x^2 + y^2$)	a&&b++

2. Funciones

Programas complicados en Octave pueden ser simplificados mediante funciones. Estas pueden ser definidas o bien en la línea de comandos de la sesión interactiva de Octave, o bien en un archivo con extensión .m. Recordad que los archivos de extensión .m pueden ser o bien scripts: que recogen diferentes ordenes que se ejecutan cuando hacemos el llamado en la línea de comandos con el nombre del scripts, o funciones: que pueden tener variables de entradas y variables de salida y cuyo nombre de la funcion debe coincidir con el nombre del archivo que la contiene. La tipografía de una función puede simplificarse como sigue en la Tabla 3.

```
function name
body
endfunction
```

```
function name (arg-list )
body
endfunction
```

```
function [ret-list ] = name (arg-list )
body
endfunction
```

Cuadro 1: De izquierda a derecha: sin input ni output, con input, con output e input

Veamos tres ejemplos de funciones.

```
function simple1()
printf("Imaginação leva-nos
 a qualquer lado.");
endfunction
```

```
function simple2(nombre,fecha)
  printf("Firmado \n %s \n %s\n",nombre,fecha);
endfunction;
```

Cuadro 2: De izquierda a derecha: sin input ni output, con input

Os recomendamos leais detenidamente los códigos de las anteriores funciones y saquéis vuestras propias conclusiones. Después os recomendamos ejecutarlas y ver si habéis entendido el código que allí aparece. Un ejemplo de ejecución es el que ofrecemos a continuación:

```
function [palabra longitud]=simple3(nombre)
  if (ischar(nombre)==1)
    palabra='True';
    longitud=length(nombre);
    else
      palabra="False";
      longitud=0;
   endif
endfunction
```

Cuadro 3: Función con argumentos de entrada y de salida

```
>>simple1()
Imaginação leva-nos a qualquer lado.

>>simple2("maria","2 de septiembre de 1996")
Firmado
 maria
 2 de septiembre de 1996

>>[sal1 sal2]=simple3("Antonio Yus")
sal1 = True
sal2 =  11
```

3. Comandos: *do-until* y *switch*

En el anterior númro estuvimos estudiando como crear sentencias de condicionales y crear bucles con los comandos *for* y *while*. en esta ocación os explicaremos otros comandos que son menos utilizados pero que en ocasiones pueden ayudarnos a economizar código.

3.1. switch

Es muy común realizar diferentes acciones dependiendo de los valores de una variable. Esto puede hacerse con el comando *if*. Veamos el siguiente ejemplo:

```
if (X == 1)
   haz_algo ();
elseif (X == 2)
   haz_algo_distinto ();
else
   haz_algo_completamente_distinto ();
endif
```

Otra opción para realizar este tipo de órdenes mediante el comando *switch* es la que sigue:

```
switch (X)
 case 1
    haz_algo ();
 case 2
    haz_algo_distinto ();
 otherwise
    haz_algo_completamente_distinto ();
endswitch
```

3.2. do-until

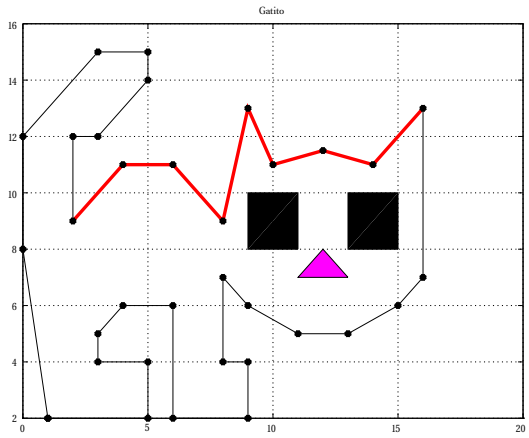
El comando *do-until* es muy parecido al comando *while* que vimos en el número anterior pero con las siguientes diferencias: las repeticiones se realizan hasta que la condición sea cierta y el test de la condición se realiza al final, al contrario que el comando *while* que evalúa la condición al principio.

```
do
body
until (condition )
```

4. Solución al reto de la anterior semana

La semana anterior planteamos el reto de interpolar mediante splines el lomo (parte roja) del gato. ¿Cómo os fue? Nosotros os damos la siguiente solución. Puedes encontrar el programa solución en la Sección de Códigos del formato y la salida que se tiene es la siguiente:

El Reto



Una solución al Reto