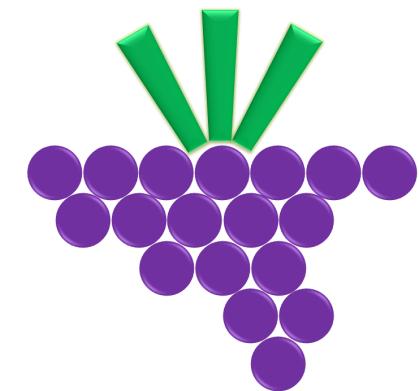




para ciencias aplicadas e ingeniería



## Unidad 4



San Rafael, Argentina, Mayo-Junio de 2021



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



FACULTAD DE  
**CIENCIAS APLICADAS**  
A LA INDUSTRIA

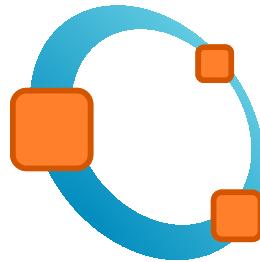


**MoCCAi**  
MODELADO COMPUTACIONAL EN CIENCIAS APLICADAS E INGENIERÍA



# Introducción a Octave

## para ciencias aplicadas e ingeniería



**Daniel Millán**



**Nicolas Muzi**



Eduardo Rodríguez



Nora Moyano



Rodrigo Bautista



Brian Villegas



Brando Martinelli



Heber Duran

San Rafael, Argentina, Mayo-Junio de 2021



# Programación en Octave

- 1. ***if/else***
  - 2. ***switch/case***
  - 3. ***for***
  - 4. ***while***
  - 5. ***Function***
  - 6. Definición de funciones de usuario ***function()***.
  - 7. ***Help*** para las funciones de usuario.
- } **Programación Estructurada**





# Programación Estructurada

- Comúnmente es necesario realizar *guiones* que requieren utilizar ciertas órdenes estándares de **Programación Estructurada**.
- La **programación estructurada** es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando únicamente tres estructuras [*secuencia, selección e iteración*] y subrutinas.
  - **Secuencia** de instrucciones, el fin de una da inicio a la siguiente.
  - **Selección** mediante sentencias condicionales o bifurcaciones.
  - **Iteración** o repetición de sentencias mediante bucles.
  - Ejecución independiente de una **subrutina** o subprograma.





# Programación Estructurada

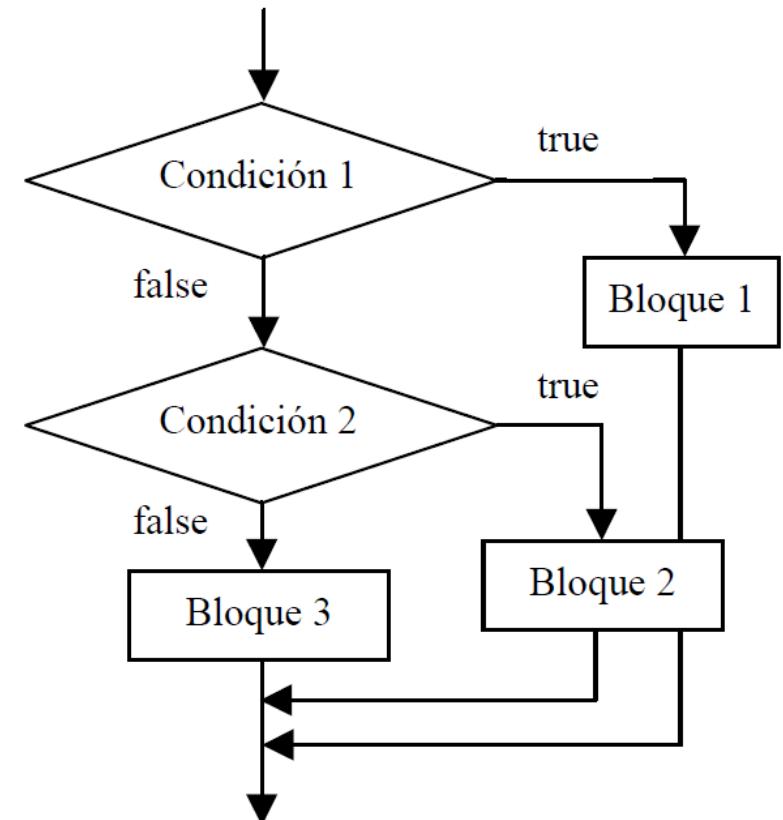
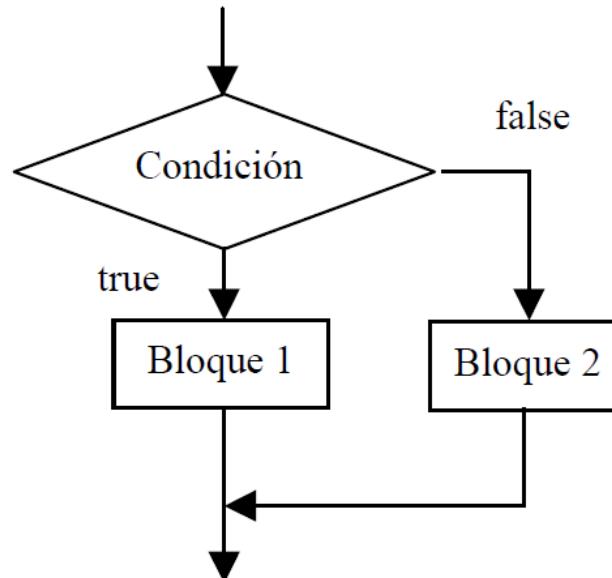
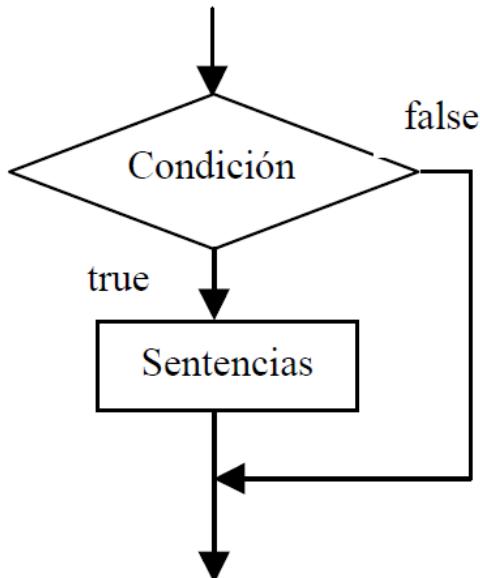
- Programación secuencial comparada con el código *spaghetti*.
  - Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.
  - Los programas son más fáciles de entender, dado que es posible su lectura secuencial y no hay necesidad de hacer engorrosos **GOTO <=> spaghetti**.
  - La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
  - Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los fallos o errores del programa (*debugging*) es más simple.
  - Reducción de los costos de mantenimiento.  
Modificar o extender los programas resulta más fácil.
  - Los programas son más sencillos y por ende más rápidos de confeccionar.

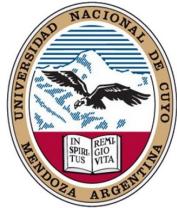




# Sentencia condicional *bifurcaciones*

- Las **bifurcaciones** permiten realizar una u otra operación según se cumpla o no una determinada condición.





# 1. Sentencia condicional **if**

## if-elseif-else-end

- En **m-scripting** es posible realizar saltos dependiendo del resultado de cumplir o no alguna condición *test*:

**if** (*condición1*)

    órdenes-si-*condición1*-es-verdadero

**elseif** (*condición2*)

    órdenes-si-*condición2*-es-verdadera

...

**elseif** (*condiciónN*)

    órdenes-si-*condiciónN*-es-verdadera

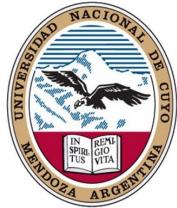
**else**

    órdenes-si-*condiciones1,2,...,N*-son-falsas

**end**

- La condiciones 1, 2,...,N pueden implicar características de archivos o de cadenas de caracteres sencillas o comparaciones numéricas.





# 1. Sentencia condicional **if**

**Ejemplo:** Generamos un número aleatorio entero entre 1 y 10, imprimimos mensajes en la terminal de acuerdo a su valor.

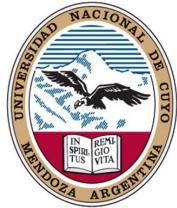


**Ayuda:** utilizar el script **U4\_ej\_if\_else.m** subido a la web del curso.

```
x = ceil(rand(1,1)*10); %nro aleatorio entre 1 y 10  
printf("\tVariable x=%d\n",x); %imprime valor de x
```

```
%según alguna condición se ejecuta una sentencia  
if (x==1)  
    disp("Variable is 1")  
elseif (x==6 || x==7)  
    disp("Variable is either 6 or 7")  
else  
    disp("Variable is neither 1, 6 nor 7")  
end
```





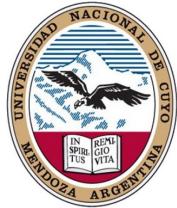
## 2. Sentencia condicional switch

### switch-case-otherwise-end

- Se utiliza como una forma conveniente para llevar a cabo tareas multipunto, donde un valor de entrada **variable** se debe comparar con varias alternativas específicas:

```
switch (variable)
    case var_expresión
        órdenes-bloque1
    case {var_expr2, var_expr3,...}
        órdenes-bloque2
    ...
    otherwise % opción por defecto
        órdenes-bloqueN
end
```





## 2. Sentencia condicional switch

**Ejemplo:** Generamos un número aleatorio entero entre 1 y 10, imprimimos mensajes en la terminal de acuerdo a su valor.



**Ayuda:** utilizar el script **U4\_ej\_switch.m** subido a la web del curso.

```
x = ceil(rand(1,1)*10); %nro aleatorio entre 1 y 10  
printf("\tVariable x=%d\n",x); %imprime valor de x
```

%según alguna condición se ejecuta una sentencia  
**switch** (x)

**case** 1

**disp**("Variable is 1")

**case** {6, 7}

**disp**("Variable is either 6 or 7")

**otherwise**

**disp**("Variable is neither 1, 6 nor 7")

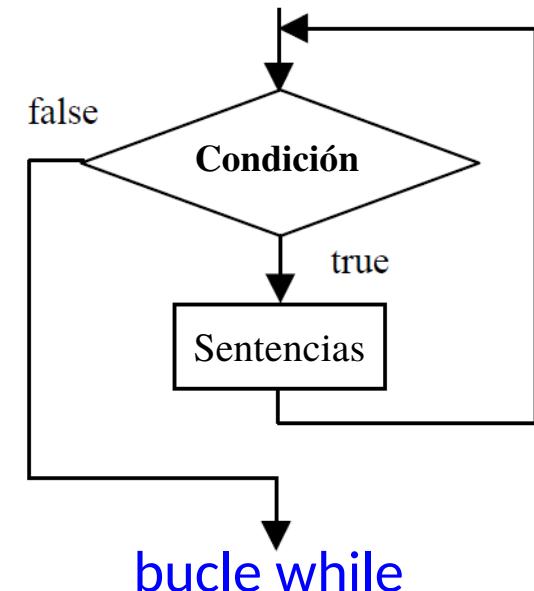
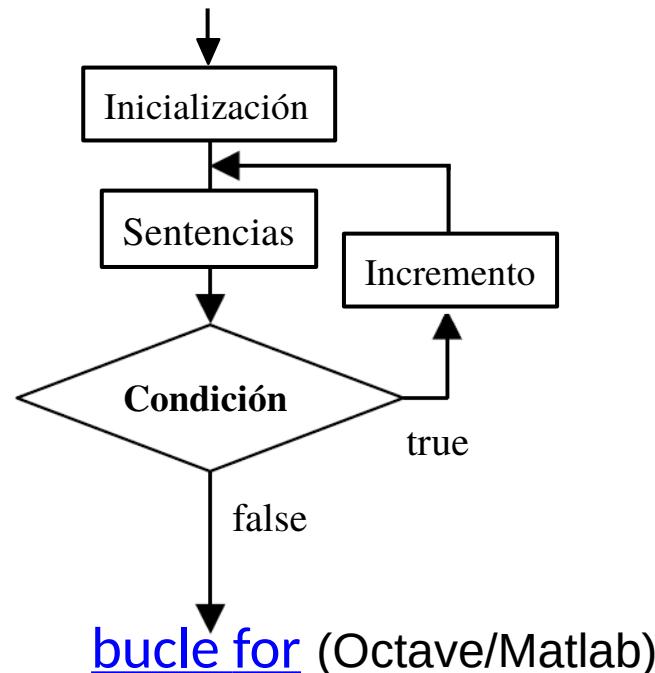
**end**





# Bucle o ciclo (*loop*)

- Un **bucle** o **ciclo** (*loop*), es una sentencia que se realiza de forma repetida en una porción aislada de código, hasta que la condición asignada a dicho bucle deja de cumplirse.
- Generalmente se emplea un bucle para evitar tener que escribir varias veces el mismo código, lo cual ahorra tiempo, procesos y deja el código más claro y facilita su modificación.
- Los bucles más utilizados son el bucle for y el bucle while.





# 3. Bucle **for**

## for-end

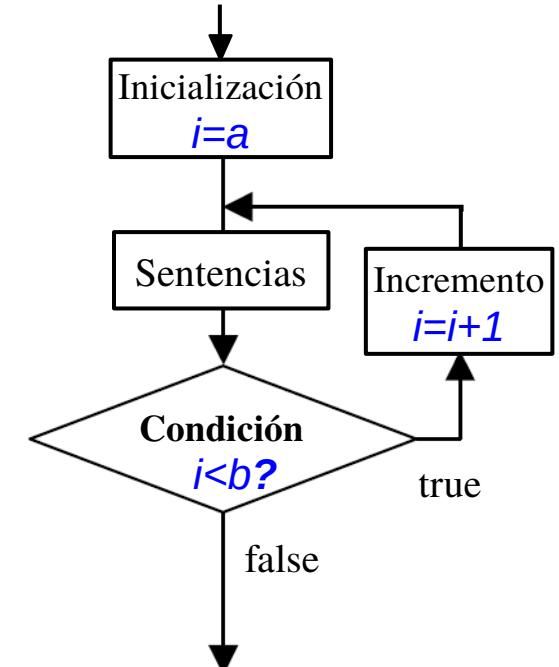
- En Octave **scripting** es posible realizar bucles **for**, lo que nos permite realizar ciertas operaciones un número determinado de veces.
- Este tipo de bucle es muy útil por ejemplo cuando queremos movernos a través de una lista de archivos e ir ejecutando algunas órdenes en cada archivo de la lista.

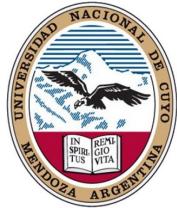
*%a, b son números enteros tal que  $a \leq b$*

```
for i=a:b  
    declaraciones  
end
```

**Ejemplo:** analice el funcionamiento del bucle **for**  
si se tiene: **x=1:10**, **x=10:-1:1**.

```
for i=1:length(x)  
    disp( x(i) )  
end
```





# 4. Bucle **while**

## while-end

- En Octave **scripting** también es posible realizar bucles **while**, que permite realizar ciertas operaciones de forma cíclica mientras se cumpla alguna condición **test**:

**while ( test )**

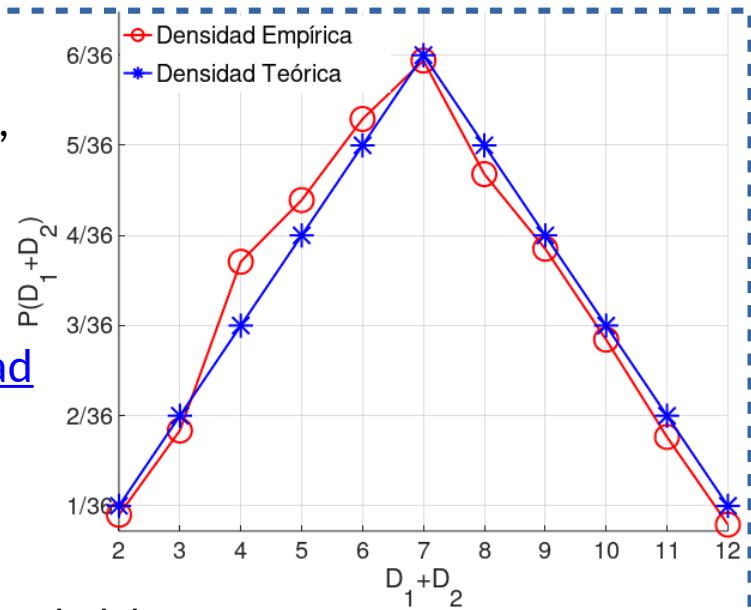
*ejecuta órdenes mientras test es verdadero*

**end**

**Ejemplo:** simule tirar dos dados.

- Repita hasta que la suma de ambas caras sea 7, cuente el número de tiradas. Repita esto 1000 veces y estime la probabilidad de sacar un 7 y compare esta con el valor teórico.
- Muestre la [Función de Densidad de Probabilidad](#) teórica y del conjunto muestral para  $N=1000$  y  $N=10000$  tiradas.

**Ayuda:** utilizar el script **U4\_ej\_dosdados.m** subido a la web del curso.





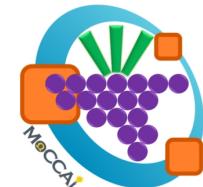
# Bucle o ciclo (*loop*)

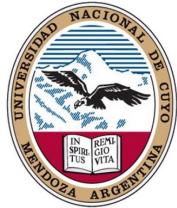
- **Sentencia break:** hace que se termine la ejecución del bucle **for** o **while** más interno de los que comprenden a dicha sentencia.
- **Sentencia continue:** hace que se pase inmediatamente a la siguiente iteración del bucle **for** o **while**, saltando todas las sentencias que hay entre el **continue** y el fin del bucle en la iteración actual.

**Ejemplo:** implemente en un *script* el siguiente algoritmo, imprima los resultados de pasos intermedios que considere oportunos.

- |   |
|---|
| ➤ Para $i=1$ a $10$ ejecutar  |
| ➔ Si el valor del contador $i$ no es múltiplo de $3$ pasar al siguiente |
| ➔ Acumular en $j$ la suma de los valores desde $1$ a $i$                |
| ➔ Mientras el valor de $j$ sea menor a $100$ ejecutar                   |
| • Si el valor de $j$ es par salir del bucle <b>while</b>                |
| • Si no es par sumar a $j$ un valor aleatorio entre $1$ y $10$          |
| ➔ Si el valor de $i+j$ es múltiplo de $5$ salir del bucle <b>for</b>    |

**Ayuda:** utilizar el script **U4\_ej\_bucles\_break\_continue.m** subido a la web del curso.





# 5. Funciones

- En **computación**, una **subrutina** o **subprograma** (también llamada **procedimiento**, **función**, **rutina** o **método**), como idea general, se presenta como un subalgoritmo que forma parte del **algoritmo** principal, el cual permite resolver una tarea específica.
- Algunos lenguajes de programación, como Fortran, utilizan el nombre función para referirse a subrutinas que devuelven un valor.
- Conceptos
  - Se le llama subrutina a un segmento de código separado del bloque principal y que puede ser invocado en cualquier momento desde este o desde otra subrutina.
  - Una subrutina, al ser llamada dentro de un programa, hace que el código principal se detenga y se dirija a ejecutar el código de la subrutina.





# 6. Definición de funciones de usuario *function()*.

- La **primera línea** de un fichero llamado **nombre.m** que define una función tiene la forma:

**function [valores retorno] = nombre(argumentos)**

donde **nombre** es el nombre de la función. Entre corchetes y separados por comas van los **valores de retorno** (siempre que haya más de uno), y entre paréntesis también separados por comas los **argumentos**.

- Puede haber funciones sin valor de retorno y también sin argumentos.
- Recuérdese que los **argumentos** son los **datos de entrada** de la función y los **valores de retorno** sus **resultados**.
- En Octave una función no modifica los argumentos que recibe, de cara al entorno que ha realizado la llamada.
- También es posible crear una **function** en un **script** sin necesidad de definir esta en un fichero.





# 7. Help para las funciones.

- También las funciones creadas por el usuario pueden tener su **help**, análogo al que tienen las propias funciones de Octave.
- Esto se consigue de la siguiente forma: las primeras líneas de comentarios de cada fichero de función son muy importantes, pues permiten construir un **help** sobre esa función a la cual se accede mediante:  
**>> help mi\_func**

**Ejemplo:** crear dos versiones de una función que sume dos números y devuelva la raíz cuadrada o el valor al cuadrado, si la suma es par o impar respectivamente.

- a) Una versión debe ser creada como una función “command-line” y cuyo nombre sea **U4\_raizpoteCL** y otra generada empleando un fichero **U4\_raizpote.m**.
- b) En la versión **U4\_raizpote.m** generar la documentación de esta función y comprobar su funcionamiento.

**Ayuda:** utilizar el script **U4\_ej\_raizpote.m** y la función **U4\_raizpote.m** subidos a la web del curso.

