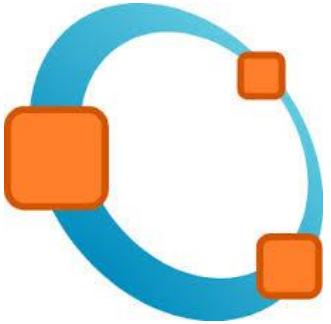
Introducción a Octave

Unidad 5-II



Daniel Millán
Nora Moyano & Iván Ferrari

San Rafael, Argentina 2018

MoΩCCAi
MODELADO COMPUTACIONAL EN CIENCIAS APLICADAS E INGENIERÍA

Departamento de
Ingeniería Mecánica

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD DE
CIENCIAS APLICADAS
A LA INDUSTRIA

Objetivos

- Adquirir un conocimiento básico de órdenes avanzadas y su modo de empleo en lenguaje *m* de Octave.

- Desarrollar un pensamiento sistemático y analítico de programación estructurada en Octave vía *sripting.*

Bucles, condiciones y funciones

- Las computadoras son herramientas particularmente útiles cuando se desean realizar tareas repetitivas de forma eficiente y precisa sobre un conjunto de datos.
- Los **bucles** (*loops*) permiten realizar partes repetitivas de un proceso, usando una condición para verificar las reglas de inicio y finalización. Octave proporciona una manera simple de definir y ejecutar bucles como se verá luego.
- Una **sentencia condicional** es un conjunto de instrucciones que se puede ejecutar o no en función del valor de una condición.
- Un programa puede requerir llamar un conjunto de instrucciones en diferentes momentos. Este conjunto de instrucciones se puede definir como una **función**, el cual se puede solicitar para realizar el cálculo en un momento deseado.
- De esta manera, una tarea complicada se puede dividir en muchas partes pequeñas mediante funciones.

Programación en Octave II

1. ***if/else***
2. ***switch/case***
3. ***for***
4. ***while***
5. ***function***

Programación Estructurada

6. Definición de funciones de usuario ***function*()**.
7. ***Help*** para las funciones de usuario.
8. Funciones ***inline*** y ***anónimas "@"***.

Programación Estructurada

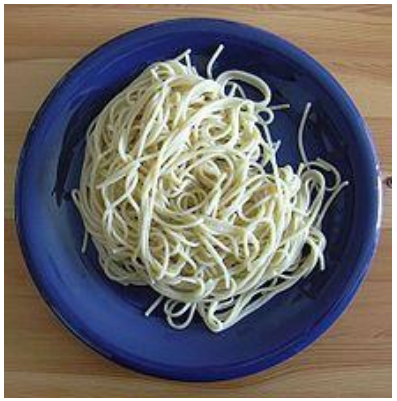
- Comúnmente es necesario realizar *guiones* que requieren utilizar ciertas órdenes estándares de **Programación Estructurada**.
- La **programación estructurada** es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando únicamente subrutinas y tres estructuras:

  secuencia, selección, e iteración.

  - Selección mediante sentencias condicionales o bifurcaciones:
    - **if/else**: de acuerdo a una condición
    - **case/switch**: de acuerdo al valor de una variable
  - Iteración o repetición mediante bucles:
    - **for**: un número determinado de veces
    - **while**: mientras se cumpla una condición
  - Ejecución independiente de una subrutina o subprograma:
    - **function**: realiza una tarea específica

Programación Estructurada

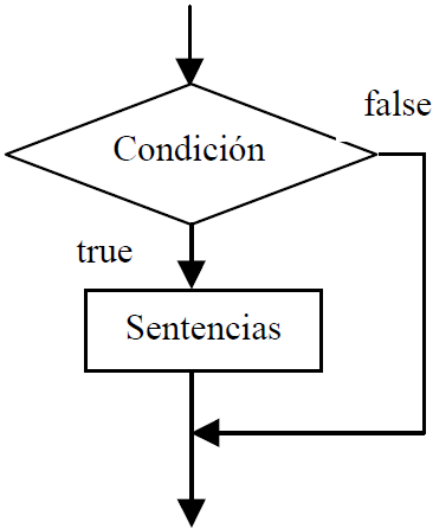
- Programación estructurada comparada con el código *spaghetti*.

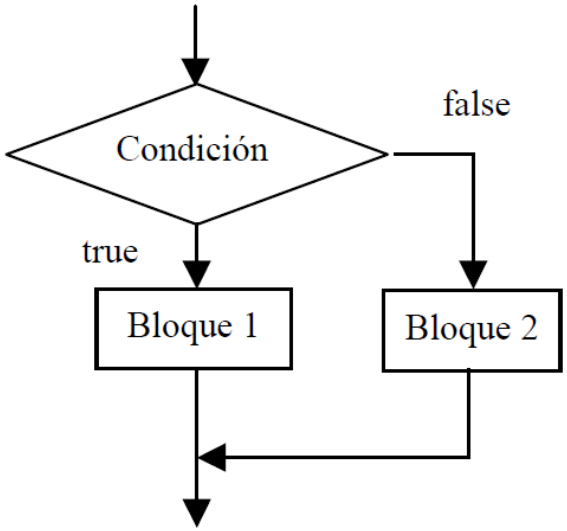


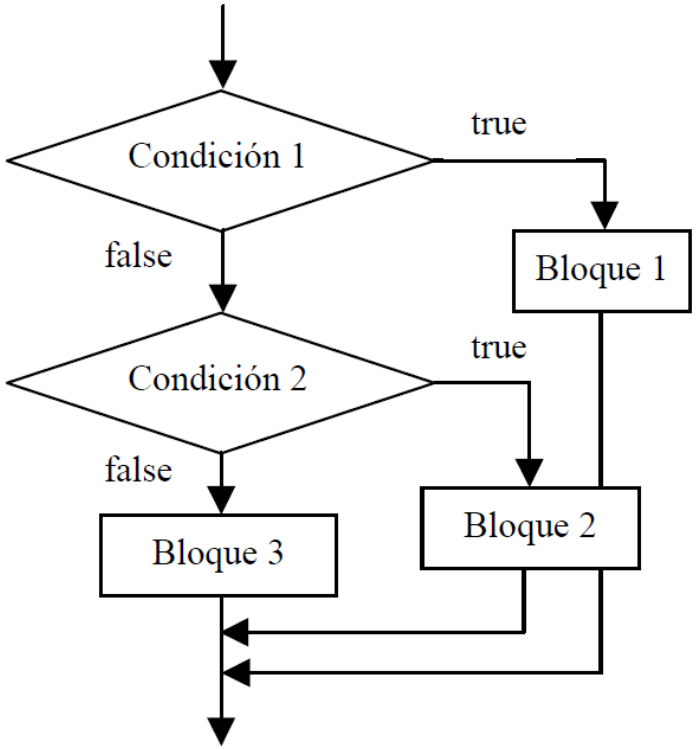
- Los programas son más fáciles de entender, dado que es posible su lectura secuencial y no hay necesidad de hacer engorrosos *GOTO*.
- La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
- Reducción del esfuerzo en las pruebas y depuración. 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 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*

Ejercicio:

```
%ejemplo if-elseif-else-end

x = ceil(rand(1,1)*10);
printf("\tVariable x=%d\n",x);

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 ***swicth***

- **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:

```
switch (variable)
        case var_expresión
                órdenes-bloque1
        case {var_expr2, var_expr3,...}
                órdenes-bloque2
        ...
        otherwise % opción por defecto
                órdenes-bloque3
end
```

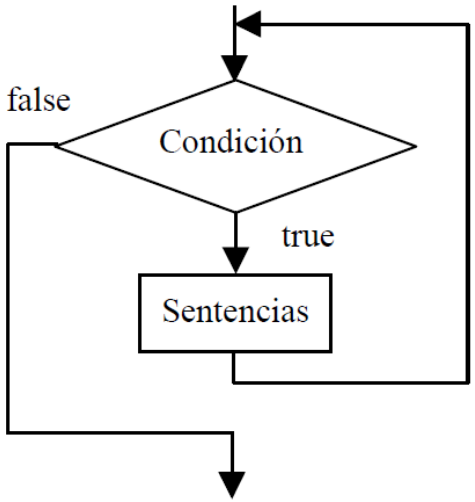
2. Sentencia condicional ***switch***

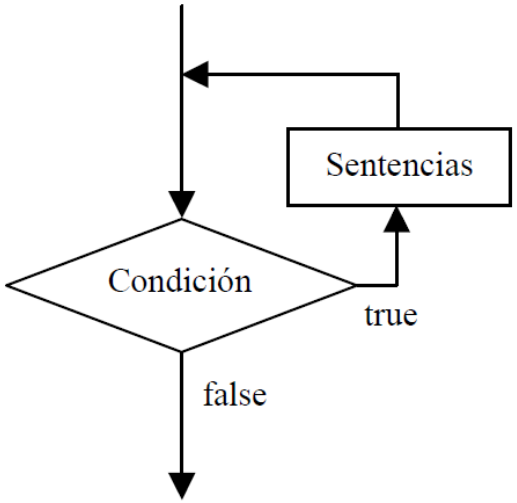
Ejercicio:

```
%ejemplo switch-case-otherwise-end

x = ceil(rand(1,1)*10);
printf("\tVariable x=%d\n",x);

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*), en programación, es una sentencia que se realiza repetidas veces en un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.
- Generalmente, un bucle es utilizado para hacer una acción repetida sin tener que escribir varias veces el mismo código, lo que ahorra tiempo, procesos y deja el código más claro y facilita su modificación en el futuro.
- Los dos bucles más utilizados en programación son el bucle while y el bucle for.





Bucle o ciclo (*loop*)

- Sentencia **break:** hace que se termine la ejecución del bucle ***for*** y/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.

3. Bucle ***for***

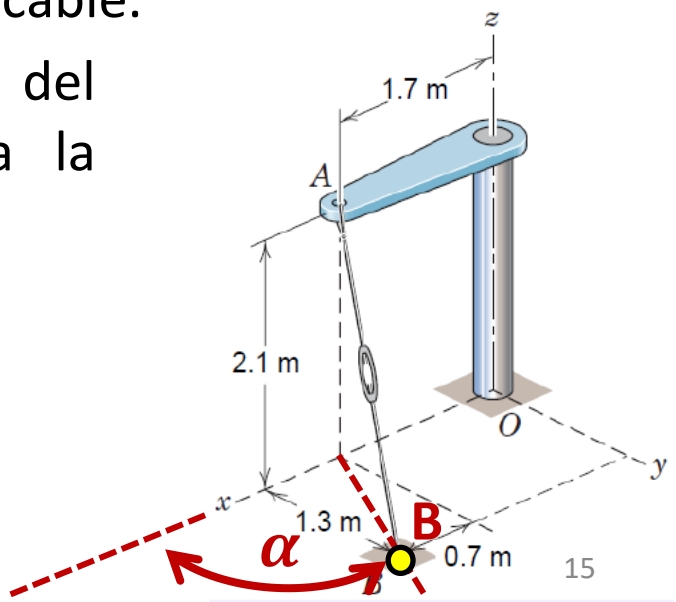
- **for-end**
- En Octave ***scripting*** es posible realizar bucles **for**, lo que nos permite realizar ciertas operaciones un número determinado de veces.
- Este tipo de bucle es muy útil por ejemplo cuando queremos movernos a través de una lista de archivos e ir ejecutando algunas órdenes en cada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

```
for i=a:b          %a, b son números enteros tal que a<b
        declaraciones
end
```

3. Bucle *for*

Ejercicio: Una estructura rígida se carga en el extremo libre como se muestra en la figura (ver figura). La tensión en el cable se puede ajustar hasta un valor máximo de 2.5kN.
La tensión del cable, que actúa en el punto **A**, produce el momento **M=r** x **T** respecto al punto **O**. Se desea conocer como afecta la posición **B** sobre el plano ***xy***, en la cual es fijado el cable si se mantiene constante la longitud del cable.

- Graficar la norma de **M** en función del ángulo $\alpha = [0, 2\pi]$ que parametriza la posición de **B** en el plano *xy*.



Ayuda: utilizar el *script* subido a la web del curso TorqueBrazoTensionado.m.

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.

- 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 FDP (o *PDF*) del conjunto muestral para N=10000 tiradas.

Ayuda: utilizar el *script* 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.
- Concepto
  - 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 de 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 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.

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: cree una función que sume dos números y devuelva la raíz cuadrada o el valor al cuadrado si la suma es par o impar, genere la documentación de esta función.

8. Funciones *inline* y anónimas @

- Funciones **inline** se dejará de emplear en el futuro - no usar.
- Las funciones anónimas **@** constituyen una forma muy flexible de crear funciones sobre la marcha, bien en la línea de órdenes, bien en una línea cualquiera de una función o de un fichero ****.m***.
- La forma general de las funciones anónimas es la siguiente:

```
fhandle = @(argumentos) expresión;
```

- Después de ser creada, la función anónima puede ser llamada a través del ***fhandle*** seguido de la lista de argumentos actuales entre paréntesis, o también puede ser pasada a otra función como argumento, también por medio del ***fhandle***.

Ejemplo: calcular el valor del seno del ángulo doble:

```
senoAngDoble = @(ang) 2*sin(ang).*cos(ang);
```

>> senoAngDoble(pi/4)

>> ans: 1

There is really no secret about our approach. We keep moving forward opening new doors and doing new things because we are curious. And curiosity keeps leading us down new paths. We are always exploring and experimenting. At WED*, we call it **Imagineering**. *The blending of creative imagination with technical know-how.*

Walt E. Disney 1965 Presentation ``Total Image''

*Disney called WED to ``My back yard laboratory, my workshop away from work.''