

Herramientas y Elementos de Programación

Ruben Weht^{1,2}

Mariano Forti^{1,3}

¹ Instituto de Tecnología Prof. Jorge Sabato

²Física del Sólido, Edificio TANDAR, weht@cnea.gov.ar, interno 7104

³División Aleaciones Especiales, Edificio 47 (microscopía), mforti@cnea.gov.ar,
interno 7832

1. Problema Ingenieril o Físico

Cuando un equipo profesional se dispone a resolver un problema Ingenieril o físico, es posible dividir la tarea en tres etapas, como se muestra en la Figura 1. El *preproceso* abarca toda la etapa de planteo del problema desde la identificación del modelo físico a usar, las condiciones de contorno, toma de medidas, modelo matemático, etc.

Como *proceso* del problema, podemos ubicar la etapa de implementación del modelo matemático. Cuando se usen computadoras para resolver esta parte del problema, la discretización y matricialización de las funciones de estado y de las variables independientes pueden ubicarse en esta etapa. La resolución, i.e. la ejecución del comando final, y el registro de los resultados pueden ubicarse aquí.

Por último, en la etapa de *postproceso* podemos ubicar todas las tareas de medición y análisis sobre los resultados obtenidos en la etapa anterior.

En todas las etapas de la resolución del problema, el ingeniero o profesional hará uso de ciertas herramientas que le permitirán resolver el problema. La participación (i.e. tiempo de trabajo) estará repartida entre el usuario y las herramientas en sí (típicamente la PC o software a utilizar).

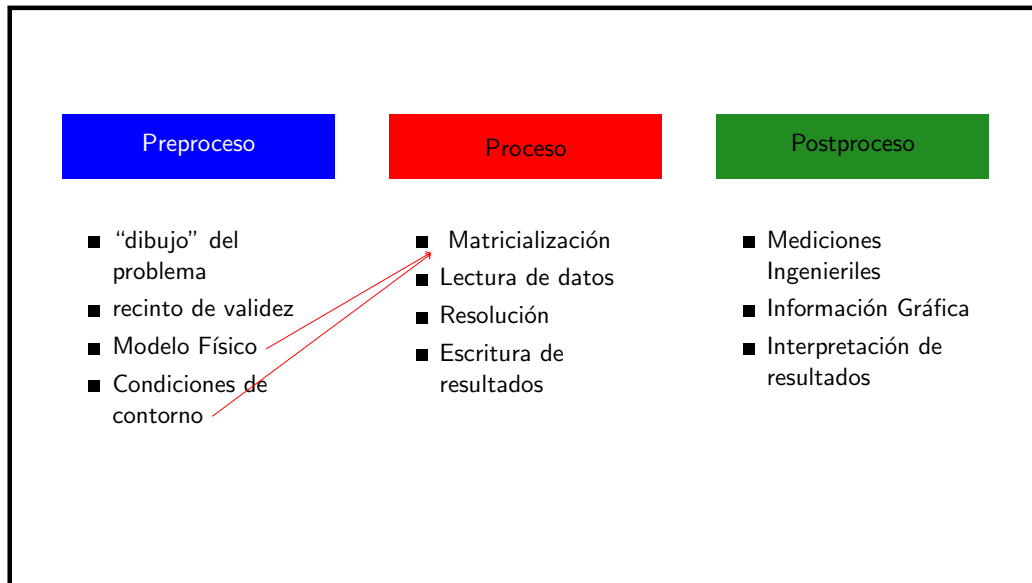


Figura 1: División del problema a resolver en etapas de análisis

2. Herramientas

En numerosas ocasiones, el profesional se encontrará con el hecho de que no es suficiente el uso de una única herramienta la solución integral de un problema. Esto se debe a que algunas herramientas se desarrollan con cierto grado de especificidad para alguna tarea. Por ejemplo, puede ser necesaria una cinta métrica para medir un recinto y un software de elementos finitos para resolver el modelo computacional en el recinto medido.

Dentro de las herramientas computacionales, es posible usar distintos paquetes o programas en distintas etapas. Típicamente esto se aplica a los programas para cálculo numérico necesarios durante el proceso, y los programas de visualización de campos escalares o vectoriales en la etapa de postproceso. Resulta evidente entonces que estos programas deberán asegurar la compatibilidad en el registro de resultados para poder integrar el análisis. Por ejemplo, durante la materia necesitaremos escribir los resultados de nuestros modelos de elementos finitos en formato de texto plano con alguna sintaxis especial para poder visualizar desplazamientos y fuerzas en un programa de visualización de mallas.

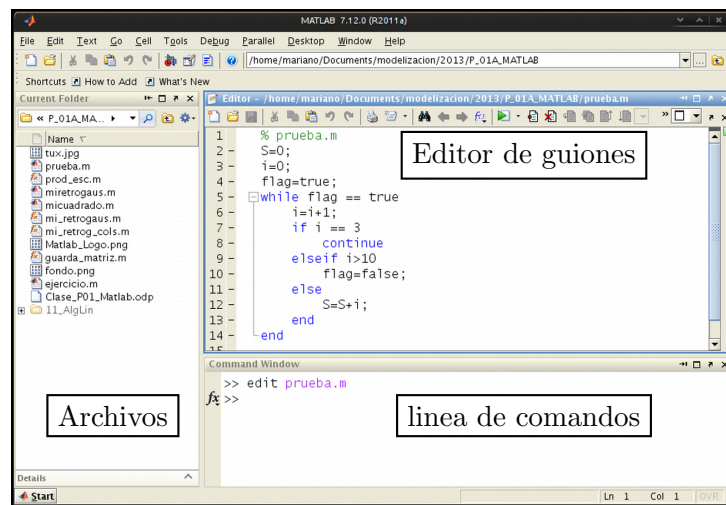


Figura 2: Ventana de Matlab. se observan el arbol de archivos del directorio de trabajo, el editor de guiones y la línea de comandos.

3. Conceptos de Programación

En lo que sigue daremos una revisión de los elementos de programación mínimos necesarios para esta materia. Como lenguaje oficial elegimos **MATLAB**¹, no solo por razones históricas, sino por su accesibilidad para profesionales sin experiencia previa en programación. En los casos donde sea posible, se darán las indicaciones equivalentes en otros lenguajes de uso corriente en la material

Busque en su pc el ícono de inicio de MATLAB, o bien inicie el programa desde la línea de comandos.

Al iniciar MATLAB en cualquier plataforma, se observan las ventanas de la Figura 2. Revise las configuraciones para obtener el escritorio que le sea cómodo. Sin embargo, puede encontrar de utilidad un escritorio como el que se muestra.

Si bien especificamos las indicaciones para esta herramienta en particular, la estructura general vale para cualquier Entorno Integrado de Desarrollo (IDE) que pueda encontrar. Busque por ejemplo el entorno *PyCharm* para **Python**, sus funcionalidades son prácticamente idénticas.

¹ Esto puede estar en cambio. Recientemente hemos ampliado nuestro interes en Python debido a la amplia disponibilidad de entornos libres y de código abierto, sino también por el alto grado de incorporación en el ambiente profesional durante los últimos tiempos.

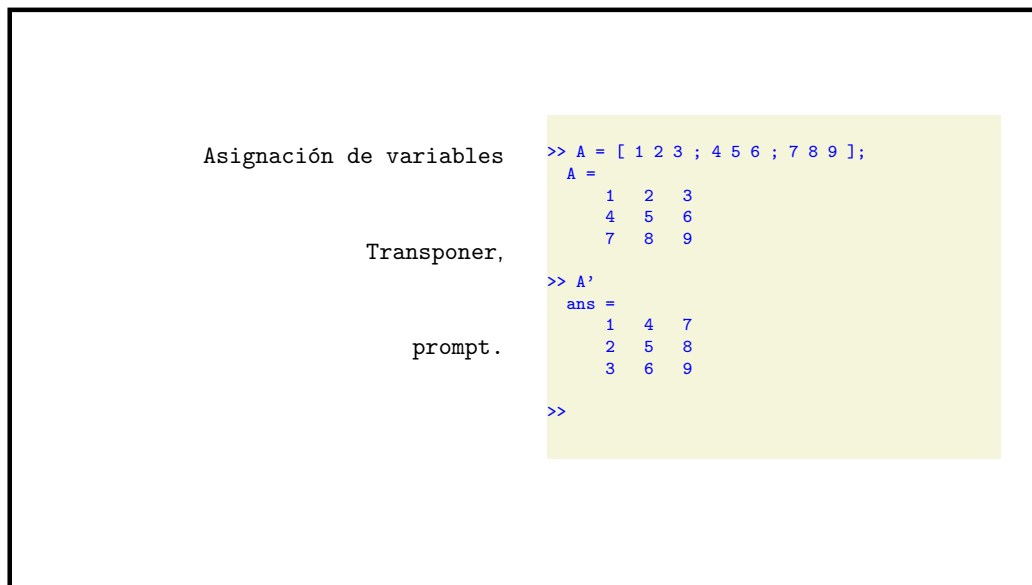


Figura 3: Asignación y operaciones básicas en MATLAB

3.1. Asignación de Variables

En cualquier lenguaje de programación, la asignación de variables es la operación básica que debe aprender. A partir de las variables asignadas usted podrá generalizar sus programas para maximizar el número de casos de uso.

La asignación de variables se lleva a cabo mediante el operador `=` de la siguiente como se muestra en la Figura 3. notar el uso del punto y coma al final de la orden en la línea de comandos. cuando no se usa, matlab muestra en pantalla el valor asignado a la variable.

3.2. Indexación de Variables

Una vez asignada la matriz, es posible indexar sus componentes. Pueden referirse individualmente el elemento de la fila i y la columna j pidiendo el elemento $A(i, j)$. Sin embargo, es posible realizar operaciones más complejas. Por ejemplo, puede referirse a un *slice* de la matriz indicando un rango de índices en un vector, como se muestra en la Figura 5. La idea de los slices de arrays de una o más dimensiones persiste en otros lenguajes, y será particularmente útil más adelante en esta materia, por lo que se sugiere que verifique su implementación en el lenguaje de programación que elija.

Observe en el ejemplo de la Figura 5 para *Python* la necesidad de usar la

Asignación de variables	<pre>(ins)>>> import numpy as np</pre>
Transponer,	<pre>(ins)>>> A = np.array([[1,2,3] , [4, 5, 6] ,[7, 8, 9]])</pre>
prompt.	<pre>(ins)>>> A array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) (ins)>>> np.transpose(A) array([[1, 4, 7], [2, 5, 8], [3, 6, 9]]) (ins)>>></pre>

Figura 4: Asignación y operaciones básicas en Python. Observe la importación del módulo *Numpy* para el tratamiento de Matrices.

función `Numpy.ix_` para generar todas las combinaciones de índices dadas por los índices de las columnas (i_1 , i_2) y de las columnas (j_1 , j_2) , mientras que en matlab no es necesario una función extra. Verifique en su lenguaje de programación cómo debe indexar las filas y las columnas para obtener el resultado del ejemplo.

3.3. Sentencias de Control de Flujo: Bucles

Frecuentemente se encontrará con la necesidad de repetir una serie de comandos un número finito de veces o bien hasta que se cumpla alguna condición lógica. Las Sentencias de Control de flujo que se usan en esas ocasiones son el **for** y el **while**, respectivamente, cuyas sintaxis se muestran en la Figura 6. A la pieza de código que forman estas sentencias se las conoce como lazos o bucles (loop).

En el caso de usar la sentencia **for**, los valores que debe tomar el iterador se dan en un rango de la forma $[i_{min} : \Delta i : i_{max}]$, estos valores enteros positivos. Nuevamente se ha dado la sintaxis en matlab pero pueden encontrarse formas equivalentes en otros lenguajes.

Luego, en el caso de utilizar la sentencia **while**, el criterio de ejecución se da con una sentencia lógica, por ejemplo la comparación entre dos números

	Matlab	Python
Rango de filas, todas las columnas	<pre>>> A(1:2,:) ans = 1 2 3 4 5 6</pre>	<pre>(ins)>>> A[0:2 ,...] array([[1, 2, 3], [4, 5, 6]])</pre>
Vector de Índices	<pre>>> A([1 3],:) ans = 1 2 3 7 8 9 >> A([1 3],[2 3]) ans = 2 3 8 9</pre>	<pre>(ins)>>> A[[0,2] , :] array([[1, 2, 3], [7, 8, 9]]) (ins)>>> A[[0,2] , ...] array([[1, 2, 3], [7, 8, 9]]) (ins)>>> A[np.ix_([0,2],[0,2])] array([[1, 3], [7, 9]])</pre>

Figura 5: Indexación de Matrices con utilización de listas de índices para obtener un slice de la matriz

	Matlab	Python
for	<pre>%suma en un rango S=0; for i=1:10 S=S+i; end disp(S)</pre>	<pre># notar limite superior +1 ! # indentacion delimita el for! S = 0 for i in range(1, 11): S = S+i print(S)</pre>
while	<pre>% suma hasta condicion S=0; i=0; while i<10 i=i+1; S=S+i; end disp(S);</pre>	<pre># suma hasta condicion S = 0 i = 0 while i < 10: i = i+1 S = S+i print(S)</pre>

Figura 6: Sintaxis para las sentencias for y while para el control de flujo de ejecución

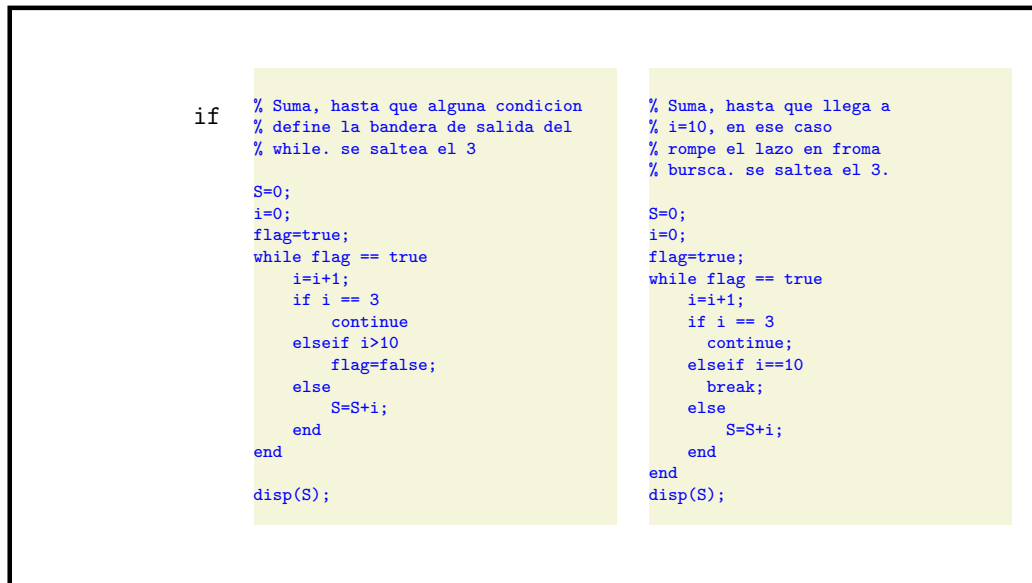


Figura 7: Esquema de la aplicación de los condicionales para evaluar piezas de código sujetas a condiciones particulares o generales. Notar el uso de los comandos **continue** y **break** para forzar la omisión en un lazo o la salida del mismo

reales, o simplemente la evaluación de una variable tipo lógico. Si se necesita un contador, en estos casos debe actualizarse el valor del mismo en forma manual. Es buena práctica poner además una condición que fuerce la terminación del lazo en caso de que la condición lógica no se cumpla en una cantidad de iteraciones razonable.

3.4. Control de Flujo: Condicionales

Otro caso de trabajo frecuente es el de tener la necesidad de condicionar la ejecución de una pieza de código a la ocurrencia de una condición lógica. Para esto se usa la sentencia **if**. La estructura más general **if**, **else if**, **else**, **end** evalúa una serie de condiciones en forma posicional y excluyente, como se esquematiza en la Figura 7. Las sentencias **else if** y **else** son opcionales. Si la condición de argumento en **if** se cumple, se ejecuta la serie de comandos hasta **else** o **end**, lo que se encuentre primero. Al terminarse la serie de comandos, no se evalúan las condiciones que son argumento de los siguientes **else if**, sino que se prosigue a partir de **end**.

Si la condición lógica que es argumento de **if** no se cumple, se evalúa la

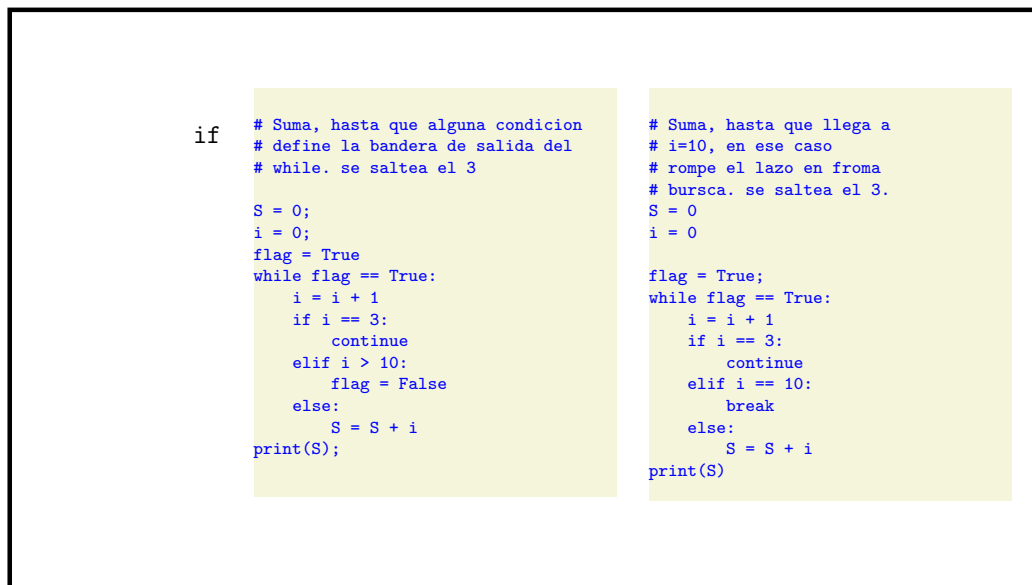


Figura 8: Ejemplos de ejecucion de uso de **continue** y **break** en **Python**. omisión en un lazo o la salida del mismo. En **Python** existen otras maneras mas elegantes y eficientes de escribir bucles y condicionales, pero dejamos estos refinamientos para más adelante.

condición lógica que es argunmento del primer **else if**, si ésta no se cumple se siguen evaluando los argumentos de las sentencias **else if** hasta que se encuentra una condición verdadera. En ese caso ocurre lo mismo que se explicó antes: cuando se termina de ejecutar la serie de comandos habilitados por esta instancia de **else if**, se prosigue a partir de **end**.

La sentencia **else** da lugar a una serie de comandos que se ejecutarán solo si ninguna de las condiciones lógicas evaluadas es verdadera. Podría decirse que marca lo que debe ejecutarse por descarte de las condiciones evaluadas.

En forma general, puede tenerse en cuenta que las condiciones sobre las sentencias **if**, **else if** *detectan* casos especiales mientras que **else** da lugar a la ejecución sobre el caso más general pero en forma excluyente de los casos particulares. Este concepto nos será de utilidad en la separación de las condiciones de contorno sobre un recinto de integración para la solución de ecuaciones diferenciales.

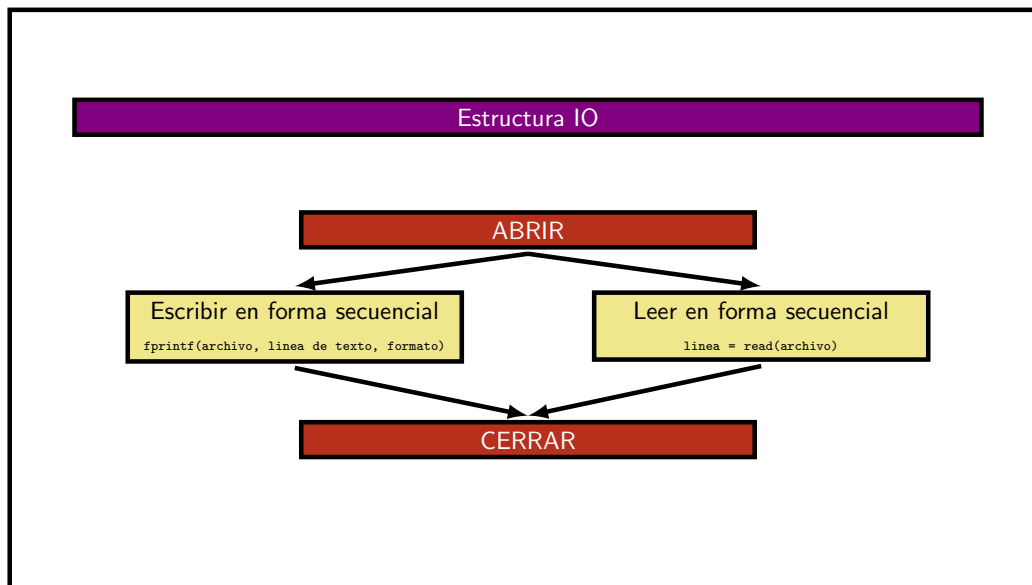


Figura 9: Flujo de ejecución para la secuencia de lectura/escritura de los archivos de registro de resultados y de datos de entrada .

3.5. Entrada y Salida de datos

El registro de los resultados de un problema es tan importante como la trazabilidad de los datos que se tuvieron en cuenta para resolverlo. Por lo tanto, en general se guardan ambos tipos de datos en archivos, que serán escritos y leídos por nuestra herramienta informática oportunamente. notar entonces que la legibilidad y la compatibilidad con otras herramientas serán características obligadas de los archivos de entrada y salida generados.

La secuencia de comandos que deben usarse para estos fines responden al diagrama de flujo de la Figura 9. Siempre habrá un comando para abrir el archivo, habrá comandos para leer (o escribir) en forma secuencial el archivo de entrada (o de salida), y por último se ejecutará un comando para cerrar el archivo.

3.6. Apertura y cierre de archivos

En todos los lenguajes de programación se utiliza algún comando para abrir los archivos, por ejemplo **fopen** en matlab. Este comando suele aceptar un argumento para indicar el nombre de archivo a abrir, que puede ser una variable de tipo string o cadena de caracteres. Otro argumento que se le da

Abrir archivo existente para lectura:	<pre>>> fid = fopen('data.dat','r') fid = 3</pre>
Abrir archivo <i>inexistente</i> para lectura:	<pre>>> fid = fopen('data2.dat','r') fid = -1</pre>
Abrir otro archivo para <i>escritura</i> :	<pre>>> fid = fopen('data2.dat','w') fid = 4</pre>

Figura 10: Comandos básicos para la apertura de un archivo

a **fopen** es el permiso con el que el archivo debe ser abierto, por ejemplo de escritura (generalmente **w**, solo lectura **r** , o para agregar contenido al final o *append* **a**. En la Figura 10 se esquematiza la utilización de estos comandos.

Es necesario notar que el comando **open** devuelve una variable de salida que en el ejemplo se asigna en la variable **fid**. La importancia de la misma radica en que a partir de la asignación **fid** se convierte en un *indicador del archivo*, una especie de puntero a la primer línea disponible del archivo. En caso de haberlo abierto con permisos de escritura, el puntero indica la primer línea del archivo que debe escribirse. Por el contrario al abrir el archivo con permisos de lectura el puntero apunta a la primer línea que puede leerse. El concepto quedará claro enseguida.

3.7. Escritura Secuencial

Una estrategia para registrar los datos en un archivo es escribir en el mismo línea por línea, o en forma *secuencial*. Una vez abierto el archivo con permisos para escritura, se escriben los datos con el comando **fprintf**. El mismo acepta como argumentos el indicador de archivo, un *string* de formato y la lista de variables que se guardan. El mecanismo se ilustra en la Figura 12.

El string de formato suele usar caracteres especiales para indicar la posición en la que la lista de variables debe imprimirse. No daremos detalle aquí, pero

Abrir archivo existente
para lectura:

```
(ins)>>> F0=open('data.dat','r')
(ins)>>> F0
<open file 'data.dat', mode 'r' at 0x7f443abfc5d0>
(ins)>>> F0.close()
(ins)>>> F0
<closed file 'data.dat', mode 'r' at 0x7f443abfc5d0>
(ins)>>> FP=open('data2.dat','r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'data2.dat'
(ins)>>> FP
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'FP' is not defined
(ins)>>> FP=open('data2.dat','w')
(ins)>>> FP
<open file 'data2.dat', mode 'w' at 0x7f443abfc6f0>
```

Figura 11: Comandos básicos para la apertura de un archivo

puede buscar en los manuales el uso de los *format identifiers*, *indicadores de formato*, ya que serán de suma utilidad. la correspondencia entre el indicador de formato y la variable que se escribe es posicional, y en general para escribir un número real se usa el indicador `%f` con algun indicador extra para la precisión. Observe la necesidad de imprimir los cambios de línea o *retornos de carro* para indicar los fines de línea.

Los archivos siempre deben ser cerrados al terminar la operación de escritura o lectura.

3.8. Funciones

Normalmente se desarrolla una pieza de código con el objetivo de poder repetir la implementación en la mayor cantidad de casos posibles. Una **función** permite justamente reutilizar una pieza de código generalizada en relación a datos (variables) de entrada. La **función** entrega variables de salida que permiten *comunicar* el resultado de la operación al flujo principal del programa. Estos conceptos se muestran en la Figura 14

Generalmente las funciones pueden escribirse en un guión con alguna sintaxis especial.

Por ejemplo en la Figura 15 puede verse la definición de una función de

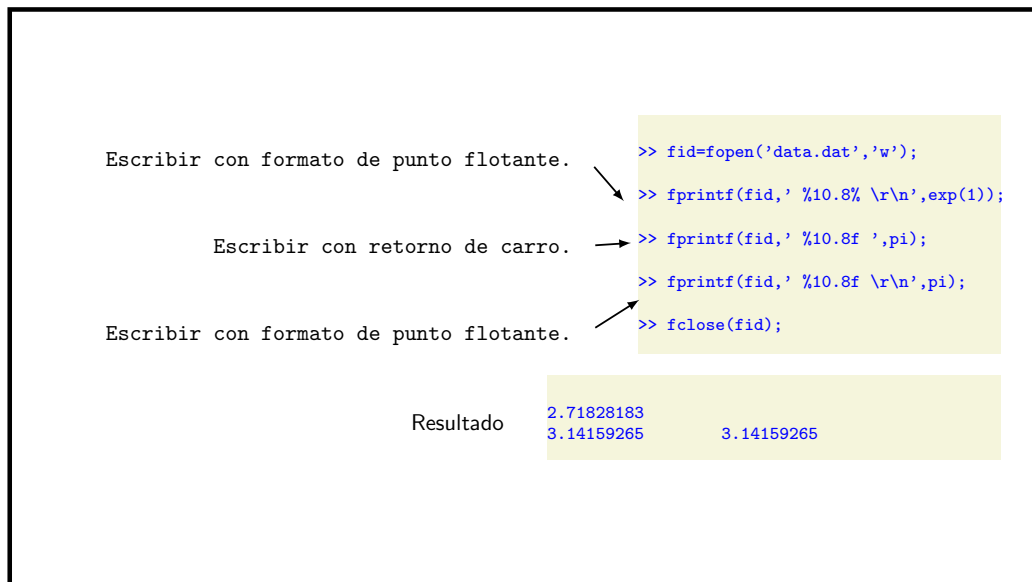


Figura 12: Algunos ejemplos de escritura secuencial con formato



Figura 13: Algunos ejemplos de escritura secuencial con formato con Python

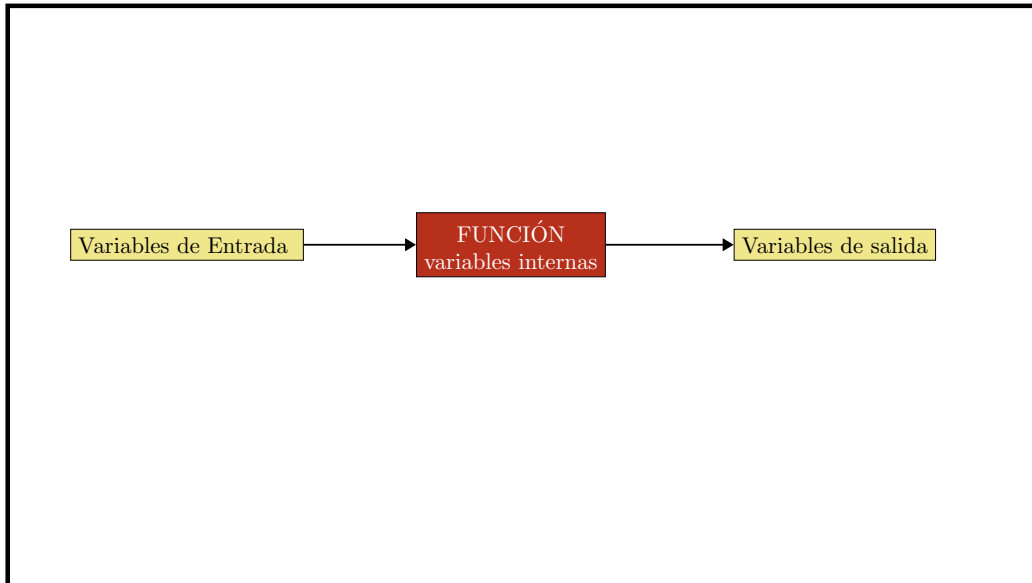


Figura 14: Las funciones deben recibir variables de entrada y entregan variables de salida

python en su propio módulo (archivo separado), y la forma de acceder y ejecutar la función desde otro guión. Notar la facilidad para declarar casos en los que debe entenderse que se ha cometido un error por parte del usuario. Lo ilustrado con la sentencia **raise** es muy rudimentario y existen mejores prácticas para esos fines.

Por otro lado, la Figura 16 ilustra el mismo ejemplo implementado en matlab. Notar la simpleza de esta implementación, que permite hacer el mismo trabajo sin necesidad de llamar a librerías extras.

4. Conclusiones

Se ha hecho una rapidísima recorrida por los elementos de programación más básicos. De ninguna manera ha sido una introducción exhaustiva. Como principales puntos a destacar, tome la necesidad de declarar y manipular matrices, incluyendo indexación y **slicing** avanzado, el uso de los graficadores, las sentencias de control de flujo y el uso de librerías propias o funciones.

De todas formas surgirán casos en los que serán necesarias nuevas herramientas y conceptos. Google y duckduckgo son sus amigos !

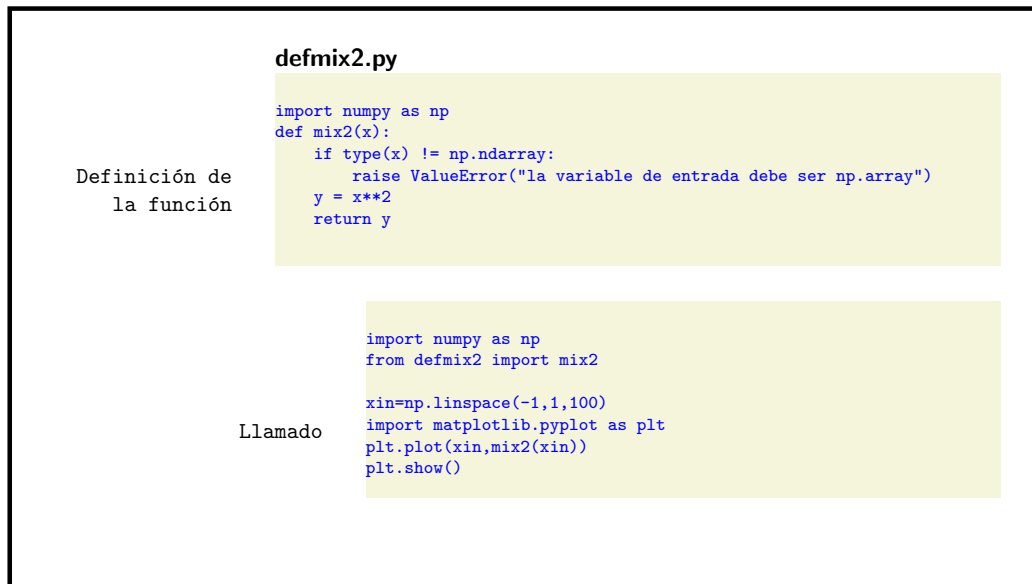


Figura 15: Definicion y llamado de una funcion en python. La funcion puede ser creada en el mismo guion donde es llamada, pero deben respetarse los indentados

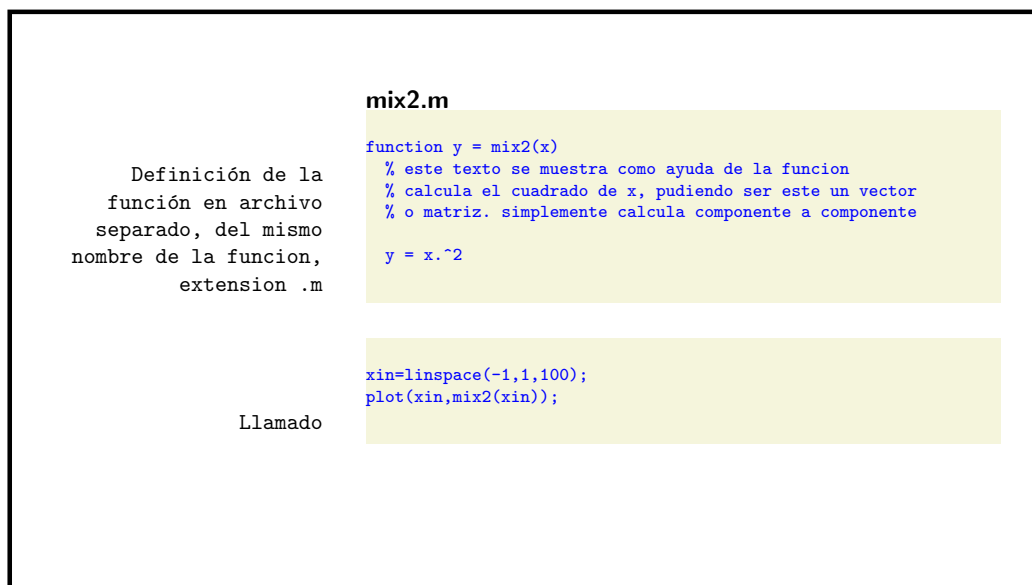


Figura 16: Definicion y llamado de una funcion en Matlab. En general la funcion se crea en un guion aparte, que debe llamarse igual que la funcion.

Herramientas y Elementos de Programación

Modelización de Propiedades y Procesos 2019

Ruben Weht^{1,2} Mariano Forti^{1,3}

¹Instituto de Tecnología Prof. Jorge Sabato

²Física del Sólido, Edificio TANDAR, weht@cnea.gov.ar, interno 7104

³División Aleaciones Especiales, Edificio 47 (microscopía), mforti@cnea.gov.ar, interno 7832



Herramientas y Elementos de Programación

Problema Ingenieril o Físico

Problema Ingenieril o Físico

Preproceso

- “dibujo” del problema
- recinto de validez
- Modelo Físico
- Condiciones de contorno

Proceso

- Matricialización
- Lectura de datos
- Resolución
- Escritura de resultados

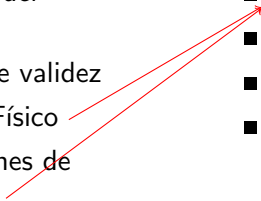
Problema Ingenieril o Físico

Preproceso

- “dibujo” del problema
- recinto de validez
- Modelo Físico
- Condiciones de contorno

Proceso

- Matricialización
- Lectura de datos
- Resolución
- Escritura de resultados



Problema Ingenieril o Físico

Preproceso

- “dibujo” del problema
- recinto de validez
- Modelo Físico
- Condiciones de contorno

Proceso

- Matricialización
- Lectura de datos
- Resolución
- Escritura de resultados

Postproceso

- Mediciones Ingenieriles
- Información Gráfica
- Interpretación de resultados



Resolucion de Problemas



PREROCESO

PROCESO

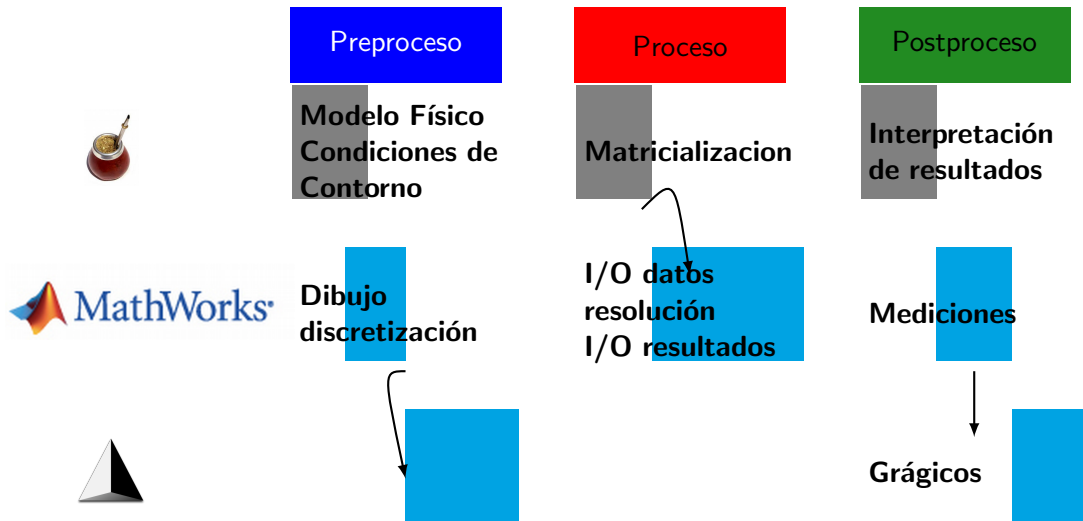
POSPROCESO

-  Usuario, Ingeniero o Profesional
-  Herramienta de Cálculo

Herramientas y Elementos de Programación

Herramientas

Uso de Herramientas



Herramientas Alternativas



Herramientas Alternativas



Herramientas y Elementos de Programación

Conceptos de Programación

Primeros Pasos en Matlab



MATrix LABoratory



Multiplataforma

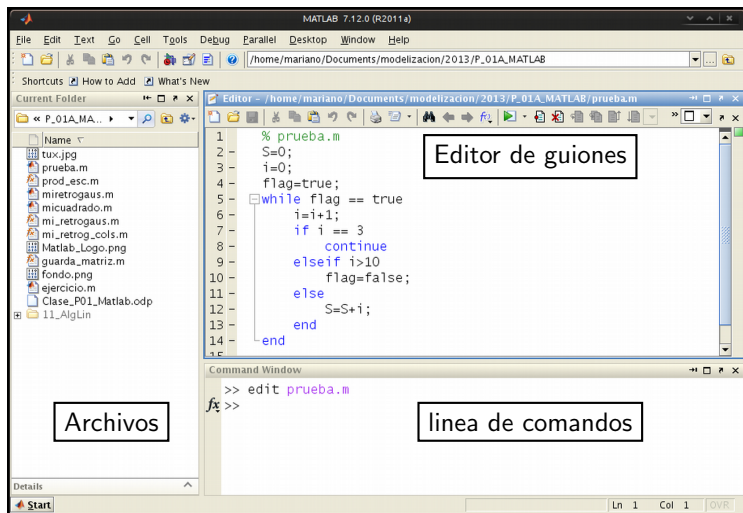


<http://www.mathworks.com/products/matlab>



Lenguaje de programación, consola programable, ejecución y redacción de scripts (guiones).

Aspecto del Escritorio de Matlab



Asignación de Variables .m

Asignación de variables

Transponer,

prompt.

```
>> A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ];  
A =  
    1    2    3  
    4    5    6  
    7    8    9  
  
>> A'  
ans =  
    1    4    7  
    2    5    8  
    3    6    9  
  
>>
```

Asignacion de Variables .py

Asignación de
variables

Transponer,

prompt.

```
(ins)>>> import numpy as np

(ins)>>> A = np.array([ [1,2,3] , [4, 5, 6] ,[7, 8, 9] ])
(ins)>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

(ins)>>> np.transpose(A)
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])

(ins)>>>
```

Indexación de Variables

Rango de filas,
todas las
columnas

Matlab

```
>> A(1:2,:)
ans =
     1     2     3
     4     5     6
```

Python

```
(ins)>>> A[ 0:2 ,...]
array([[1, 2, 3],
       [4, 5, 6]])
```

Vector de Índices

```
>> A([1 3],:)
ans =
     1     2     3
     7     8     9

>> A([1 3],[2 3])
ans =
     2     3
     8     9
```

```
(ins)>>> A[ [0,2] , : ]
array([[1, 2, 3],
       [7, 8, 9]])
(ins)>>> A[ [0,2] , ... ]
array([[1, 2, 3],
       [7, 8, 9]])
(ins)>>> A[np.ix_([0,2],[0,2])]
array([[1, 3],
       [7, 9]])
```

Control de Flujo: Bucles

Matlab

for

```
%suma en un rango  
S=0;  
for i=1:10  
    S=S+i;  
end
```

```
disp(S)
```

while

```
% suma hasta condicion  
S=0;  
i=0;  
while i<10  
    i=i+1;  
    S=S+i;  
end
```

```
disp(S);
```

Python

```
# notar limite superior +1 !  
# indentacion delimita el for!  
S = 0  
for i in range(1, 11):  
    S = S+i
```

```
print(S)
```

```
# suma hasta condicion  
S = 0  
i = 0  
while i < 10:  
    i = i+1  
    S = S+i
```

```
print(S)
```

Control de flujo: Condicionales

```
if % Suma, hasta que alguna condicion
% define la bandera de salida del
% while. se saltea el 3

S=0;
i=0;
flag=true;
while flag == true
    i=i+1;
    if i == 3
        continue
    elseif i>10
        flag=false;
    else
        S=S+i;
    end
end

disp(S);
```

```
% Suma, hasta que llega a
% i=10, en ese caso
% rompe el lazo en froma
% bursca. se saltea el 3.

S=0;
i=0;
flag=true;
while flag == true
    i=i+1;
    if i == 3
        continue;
    elseif i==10
        break;
    else
        S=S+i;
    end
end

disp(S);
```


Control de flujo: Condicionales (Python)

if

```
# Suma, hasta que alguna condicion  
# define la bandera de salida del  
# while. se saltea el 3
```

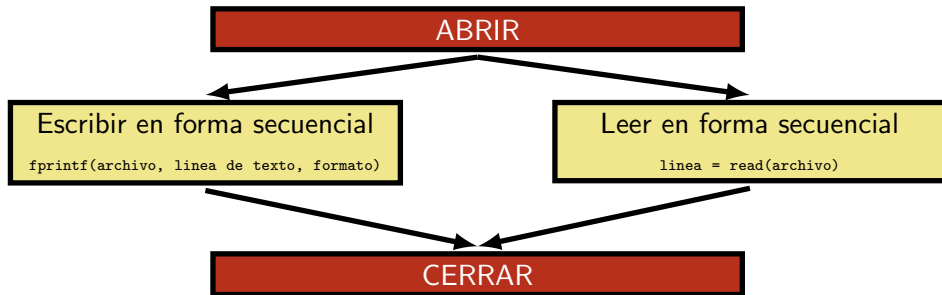
```
S = 0;  
i = 0;  
flag = True  
while flag == True:  
    i = i + 1  
    if i == 3:  
        continue  
    elif i > 10:  
        flag = False  
    else:  
        S = S + i  
print(S);
```

```
# Suma, hasta que llega a  
# i=10, en ese caso  
# rompe el lazo en forma  
# brusca. se saltea el 3.
```

```
S = 0  
i = 0  
  
flag = True;  
while flag == True:  
    i = i + 1  
    if i == 3:  
        continue  
    elif i == 10:  
        break  
    else:  
        S = S + i  
print(S)
```

Input-Output / Entrada-Salida de Datos

Estructura IO



Apertura y cierre de archivos

Abrir archivo existente para lectura:

```
>> fid = fopen('data.dat','r')  
  
fid =  
  
     3
```

Abrir archivo *inexistente* para lectura:

```
>> fid = fopen('data2.dat','r')  
  
fid =  
  
    -1
```

Abrir otro archivo para *escritura*:

```
>> fid = fopen('data2.dat','w')  
  
fid =  
  
     4
```

Apertura y cierre de archivos (Python)

Abrir archivo existente
para lectura:

```
(ins)>>> F0=open('data.dat','r')
(ins)>>> F0
<open file 'data.dat', mode 'r' at 0x7f443abfc5d0>
(ins)>>> F0.close()
(ins)>>> F0
<closed file 'data.dat', mode 'r' at 0x7f443abfc5d0>
(ins)>>> FP=open('data2.dat','r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    IOError: [Errno 2] No such file or directory: 'data2.dat'
(ins)>>> FP
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    NameError: name 'FP' is not defined
(ins)>>> FP=open('data2.dat','w')
(ins)>>> FP
<open file 'data2.dat', mode 'w' at 0x7f443abfc6f0>
```

Escritura línea por línea

Escribir con formato de punto flotante.

Escribir con retorno de carro.

Escribir con formato de punto flotante.

```
>> fid=fopen('data.dat','w');  
>> fprintf(fid,' %10.8% \r\n',exp(1));  
  
>> fprintf(fid,' %10.8f ',pi);  
>> fprintf(fid,' %10.8f \r\n',pi);  
  
>> fclose(fid);
```

Resultado

```
2.71828183  
3.14159265      3.14159265
```

Imprimir línea por línea en Python

Escribir con formato
de punto flotante

```
(ins)>>> import numpy as np
(ins)>>> FOUT=open('output.dat','wr')
(ins)>>> FOUT.write('e={:10.8f} \r\n'.format(np.exp(1)))
(ins)>>> FOUT.write('pi= {:10.8f} \r\n'.format(np.pi) )
(ins)>>> FOUT.close()
(ins)>>>
```

Resultado

```
e=2.71828183
pi= 3.14159265
```

Funciones



Definición y Llamado de funciones

defmix2.py

Definición de
la función

```
import numpy as np
def mix2(x):
    if type(x) != np.ndarray:
        raise ValueError("la variable de entrada debe ser np.array")
    y = x**2
    return y
```

Llamado

```
import numpy as np
from defmix2 import mix2

xin=np.linspace(-1,1,100)
import matplotlib.pyplot as plt
plt.plot(xin,mix2(xin))
plt.show()
```


Definición y Llamado de funciones (Matlab)

Definición de la
función en archivo
separado, del mismo
nombre de la funcion,
extension .m

Llamado

mix2.m

```
function y = mix2(x)
% este texto se muestra como ayuda de la funcion
% calcula el cuadrado de x, pudiendo ser este un vector
% o matriz. simplemente calcula componente a componente

y = x.^2
```

```
xin=linspace(-1,1,100);
plot(xin,mix2(xin));
```