المقدمة

ان مستخدم البرامج في معظم الاحيان يجد نفسه موجهاً بأحد اتجاهين: اما أن يستخدم البرامج او الدوال الجاهزة للحصول على النتائج التي يريدها دون ان يتدخل أو يعرف الكثير عنها وعن كيفية عملها او حتى يعرف بعض مفاهيم البرمجة، او العكس أي انه درس لغات البرمجة وبالتالي يميل الى انجاز جميع اعماله البرمجية من الصفر، فتراه يكتب البرامج الطويلة والمفصلة لأداء عمل معين دون ان يدري ان هناك برنامج او دالة جاهزة تنجز له هذا العمل بسرعة وبساطة، وفي معظم الأحيان يكون البرنامج الذي يكتبه اقل كفاءة من الناحية البرمجية من تلك الدوال الجاهزة. الحل الامثل هو الدمج بين الاثنين أي ان يتمكن المبرمج من مفاهيم البرمجة فيستطيع ان يتدخل بمرونة عالية في توجيه البرنامج بالاتجاه الذي يريده وفي نفس الوقت يكون عارفاً ومستخدماً جيداً للدوال والمهام الجاهزة التي تتيحها له البرامج الحديثة.

ولما كان المستخدم لا يعرف ما الذي سيحتاجه فعلاً في المستقبل هل ستكون البرامج الجاهزة التي تعمل مثل الصندوق المغلق هي التي تلبي احتياجاته ام انه سوف يحتاج الى بعض التدخل والبرمجة صار من الضروري ان يتسلح الطالب والمبرمج بالاثنين معاً وان برنامج الـ MATLAB هو الافضل من بين البرامج التي تعتني بالأمرين معاً، بالإضافة طبعاً الى قوته وسهولته في الاستخدام.

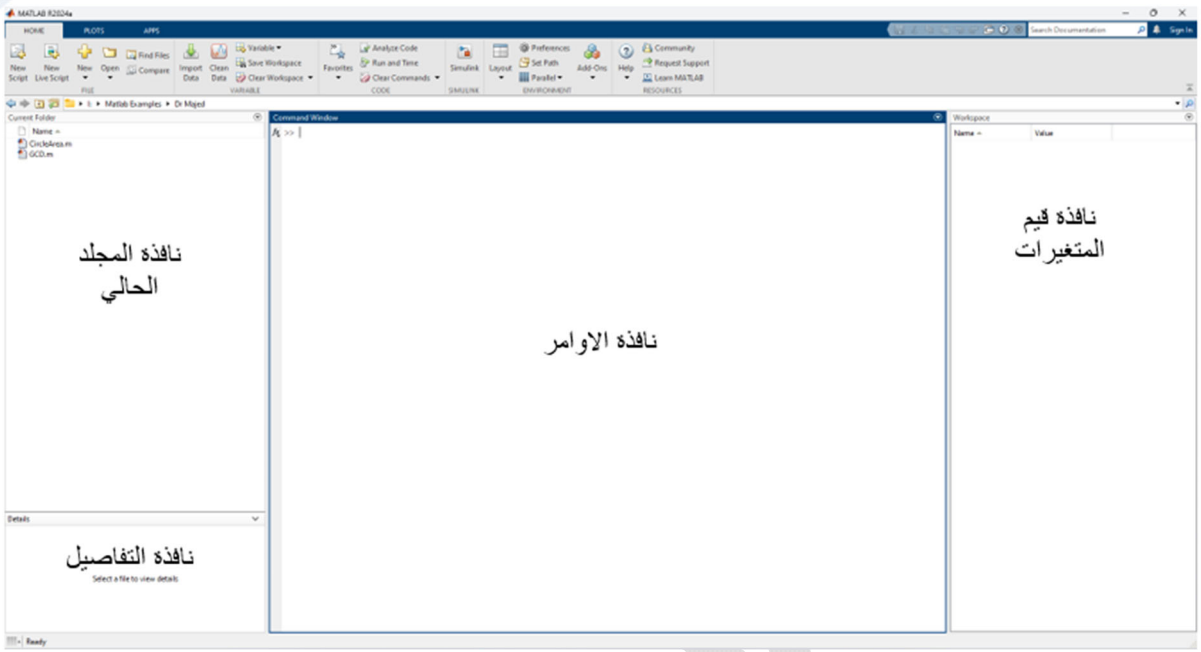
برنامج الماتلاب يمكن استخدامه بطريقتين الاولى هي الطريقة التفاعلية الـ Interactive وهي تشبه العمل مع الحاسبة الصغيرة Calculator حيث ان المبرمج يدخل العبارات الواحدة بعد الاخرى والبرنامج ينفذ كل عبارة في مكانها ويخرج نتيجتها مباشرة على الشاشة وهذه عادة تنفع في الحسابات السريعة والمباشرة، اما الاستخدام الثاني فهو ان يكتب المبرمج برنامجاً كاملاً من الايعازات وعبارات البرمجة قد يكون عدة صفحات ثم بعد ذلك يشرع في تنفيذه دفعة واحدة وهنا سوف يقوم البرنامج بقراءة العبارات بالتسلسل ويقوم بتنفيذها وتخرج النتائج دفعة واحدة بعد الانتهاء من قراءة كل البرنامج.

بيئة وواجه البرنامج

الطريقة الاسهل في الدخول وبداية التعامل مع برنامج الماتلاب هي الطريقة التفاعلية اي ان المستخدم فقط يكتب العبارة المطلوبة وبرنامج الماتلاب سوف يتفاعل معه مباشرة ويظهر له النتائج في نفس صفحة العمل ومن المناسب ان نبدأ بهذه الطريقة بغية التعرف على بيئة عمل برنامج الماتلاب.

عندما نفتح البرنامج سوف تظهر واجهته المتكونة اساساً من **أربع او خمس نوافذ مفتوحة** (حسب نسخة واعدادات البرنامج) بالإضافة طبعاً الى الاشرطة الاساسية التي تأتي مع مختلف التطبيقات التي تعمل في بيئة نظام التشغيل الويندوز من شريط عنوان وشريط قوائم واشرطة ادوات.

النوافذ المفتوحة وأهمها هي نافذة الاوامر المباشرة Command Window والتي تكون هي الاكبر وفي وسط الشاشة، في هذه النافذة سوف يظهر محث الاوامر Command Prompt وهو يأخذ الشكل (>>) ووجوده يدل على ان البرنامج او نافذة الايعازات جاهزة لاستلام الاوامر.



ان التحكم بظهور هذه النوافذ بشكلها الحالي او حتى غلق بعضها يكون من خلال احد الازرار في شريط الادوات وهو layout ويفضل في البداية ان نجعل هذه النوافذ تظهر في حالتها الافتراضية اي الـ Default.

النافذة الكبيرة التي في الوسط، نافذة الاوامر، هي النافذة التفاعلية التي تستقبل الاوامر وتنفذها مباشرة بعد الضغط على مفتاح الادخال Enter.

وهنا يمكننا ان نكتب اي امر يفهمه الـ MATLAB كأن يكون الامر حسابي مثلاً:

```
>> a=5
```

أو نكتب:

```
>> b=7+20
```

او نستفيد من بعض الدول الرياضية المبنية داخل البرنامج:

```
>> a=pi
a =
   3.1416
>> a=exp(1)
a =
   2.7183
>> a=sin(pi/4)
a =
   0.7071
```

ولاحظ هنا ان الزوايا التي يفهمها البرنامج هي بالوحدات نصف القطرية Radian اما اذا اردنا العمل مع الزوايا بوحدة الدرجاتDegree ، فعلينا ان نحولها الى الزوايا نصف القطرية قبل العمل عليها:

```
>> a=sin(45*pi/180)
a =
   0.7071
>> a=log(100)
a =
   4.6052
```

لاحظ هنا ان دالة ()log لوحدها تعني اللوغاريثم الطبيعي (اي ان الاساس هو الرقم الطبيعي e) الذي عادة يسمى في الرياضيات Ln اما اذا اردنا اللوغاريثم للأساس عشرة فسيكون الايعاز هو ()log10:

```
>> a=log10(100)
a =
    2
>> a=log(2.7183)
a =
   1.0000
```

او يمكن ندخل متغيرات رمزية وليس رقمية او حسابية (وهذه المتغيرات توضع بين قابسات اما 'منفردة' او "مزدوجة" وسوف يفهمها البرنامج على انها Character or String اي مجموعة من الرموز غير الحسابية او غير الرقمية):

```
>> a="Hello MATLAB R2024"
a =
   "Hello MATLAB R2024"
>> b= 'Hi, We are ready to start'
b =
   'Hi, We are ready to start'
```

تمرين: اكتب العمليات الحسابية التالية في نافذة الأوامر Command Window وتفحص النواتج:

1. `5 + 2`
2. `5*2`
3. `5/2`
4. `3 + 2*(4 + 3)`
5. `2.54*8/2.6`
6. `6.3 – 2.1045`
7. `3.6^2`
8. `1+2^2`
9. `sqrt(5)`
10. `cos(pi)`

او نكتب امر تنفيذي غير حسابي من (الكلمات المفتاحية keywords او الايعازات او الاوامر) التي يفهمها البرنامج ليقوم بعمل معين، مثلاً نكتب doc ليفتح لنا نافذة فيها قائمة من المصادر او النشريات التي تشرح البرنامج documentation او نكتب help ليعطينا نافذة فيها العديد من المواضيع التي يمكن ان نبحث فيها عن مساعدة أو مثلاً نكتب clc الذي يقوم بتنظيف نافذة الاوامر دون ان يلغي قيم المتغيرات، او نكتب الايعاز clear الذي يقوم بتنظيف الشاشة وكذلك يلغي قيم كل المتغيرات، او نكتب quit للخروج نهائياً من البرنامج علماً ان هذا الخروج يمكن ان يكون بطريقة اخرى وهي ان نذهب الى قائمة File ثم Exit او نغلق النافذة كلها باستخدام الرمز × الموجود في اعلى يمين الشاشة.

اما النافذة الى يسار نافذة الاوامر فهي نافذة المجلد الحالي Current Folder ويظهر فيها مسار المجلد الفعال الحالي اي المجلد الذي سوف تخزن فيه ملفات العمل، ومن خلال هذه النافذة نستطيع ان نغير ذلك المجلد حسب الرغبة. عند التقدم في العمل مع برنامج الماتلاب سوف نجد ان بعض البرامج التي ننشئها ستتكون من أكثر من ملف واحد مثلاً تكون هناك ملفات للإدخال واخرى للإخراج وملفات الرسم وملف البرنامج الاساسي عندها سوف نرى فائدة هذه النافذة والتي سوف تظهر لنا كل الملفات التي نعمل عليها في مجلد واحد ودفعة واحدة، كما إنك لو وضعت مؤشر الماوس على أحد هذه الملفات سوف تلاحظ ان تفاصيل هذا الملف سوف تظهر في النافذة السفلى الى اليسار وهي نافذة الـ Details.

قد تظهر ايضاً نافذة الـ Command History وتظهر فيها كل الايعازات او الاوامر التي قمت بكتابتها منذ بداية الجلسة.

اما النافذة العليا الى اليمين فهي نافذة الـ Workspace وتظهر فيها قيم المتغيرات التي تدخلها تباعاً.

بالعودة الى العبارة التي كتبناها سابقاً وهي:

```
>>a=5
```

ان هذه العبارة في البرمجة تختلف عن مثيلتها في الرياضيات؟ حيث هنا لا تسمى بالمساواة بل تسمى بعبارة الاسناد، اي اننا نسند كل ما موجود في الطرف الايمن (سواء كان رقم واحد او حتى تعبير رياضي معقد) للمتغير a اي نحجز للمتغير a مكان في ذاكرة الحاسوب ونضع فيه القيمة التي ستنتج من التعبير الحسابي الموجود على الطرف الايمن، ففي الرياضيات نستطيع ان نكتب a+b=9 بينما في البرمجة لا نستطيع ان نجعل الطرف الايسر الا متغير واحد أي لا نستطيع ان نجعله رقم ولا نستطيع ان نجعله تعبير رياضي Expression.

كذلك في البرمجة نستطيع ان نكتب:

```
>>x=x+3
```

بينما في الرياضيات لا يجوز ذلك لأنها عبارة غير صحيحة رياضياً. هذه العبارة في البرمجة تعني أن المتغير x الذي لديه مكان خاص في ذاكرة الحاسوب، سوف يغير قيمته بعد هذا السطر مباشرة وذلك بأخذ قيمته (القديمة) مضافا لها 3. والعملية هذه تشبه عندما نعدل ونضيف على ملف معين في الحاسوب ثم نحفظه بنفس الاسم القديم. فالملفان القديم والجديد غير متساويان ولكن الملف الجديد هو الذي تم حفظه في الذاكرة اخيراً.

تسلسل العمليات الحسابية في الـ MATLAB

ان برنامج الماتلاب يخضع الى نفس قواعد تسلسل العمليات الحسابية المعروفة في الرياضيات وهي:

1) تجرى اولاً العمليات داخل الاقواس وتبدأ من الاقواس الداخلية الى الخارجية.
2) ثانياً تجرى العمليات التي فيها اسس.
3) ثالثاً تجرى عمليات الضرب والقسمة ويكون تسلسل تنفيذها من اليسار الى اليمين.
4) واخيراً تجرى عمليات الجمع والطرح بالتسلسل من اليسار الى اليمين ايضاً.

ملاحظة: هناك اقواس ضرورية توضع في المقادير الجبرية لتعطي الاولوية لاجراء الحسابات، بينما يمكنك كذلك اضافة اي عدد من الاقواس حسب الرغبة لجعل العبارات واضحة للقراءة. وكذلك فان برنامج الماتلاب لا يقرأ او يهتم بالفراغات داخل التعبير الرياضي فيمكنك وضع الفراغات اينما اردت لكي تكون العبارات واضحة امام القارئ.

TRY IT! Compute $\frac{3*4}{2^2+4/2}$.

```
>> (3*4)/(2^2 + 4/2)
ans = 2
```

TRY IT! Compute 3 divided by 4, then multiply the result by 2, and then raise the result to the 3rd power.

```
>> 3/4
ans = 0.75
>> ans*2
ans = 1.5
>> ans^3
ans = 3.3750
```

TRY IT! Compute $1/0$, $1/\infty$, and $\infty \cdot 2$ to verify that MATLAB handles infinity as you would expect.

```
>> 1/0
ans =
   Inf
>> 1/Inf
ans =
     0
>> Inf * 2
ans =
   Inf
```

TRY IT! Compute ∞/∞.

```
>> Inf/Inf
ans =
   NaN
```

TRY IT! Compute the imaginary sum $2 + 5i$.

```
>> 2 + 5*i
ans =
   2 + 5i
```

TRY IT! Compute the number of seconds in 3 years using scientific notation.

```
>> 3e0*3.65e2*2.4e1*3.6e3
ans =
    94608000
```

تمرين: اكتب العبارات التالية وتأكد من نتائجها حسب التسلسل الرياضي لأداء العمليات الحسابية :

1. 6/6 + 5
2. 2*6^2
3. (3 + 5)*2
4. 3 + 5*2
5. 4 * 3/2 * 8
6. 3 – 2/4 + 6^2
7. 2^3^4
8. 2^(3^4)
9. 3^5+4
10. 3^(5+4)

تمرين: جد النواتج الصحيحة للتعابير الرياضية التالية داخل الماتلاب:

11. $\frac{5+3}{9-1}$
12. $2^3 - \frac{4}{5+3}$
13. $\frac{5^{2+1}}{4-1}$
14. $4\frac{1}{2} * 5\frac{2}{3}$
15. $\frac{5 + 6*\frac{7}{3} - 2^2}{\frac{2}{3} * \frac{3}{3*6}}$

العمليات المنطقية Logical Expressions

العمليات المنطقية حسابياً وبرمجياً هي دائماً العمليات التي يكون جوابها اما "نعم" او "لا" وفي اللغة الانكليزية يعبر عنها بـ "True" or "False" اما في البرمجة فتأخذ "0" or "1" اي ان الصفر يعني False والواحد يعني True.

وابرز المشغلات المنطقية Operators (المقارنة او المتراجحة) في لغة الماتلاب هي:

Operator	Meaning
>	greater than
<	less than
>=	greater than or equals
<=	less than or equals
==	equality
~=	inequality

Operator	Meaning
|	or (for scalars)
&&	and (for scalars)
~	not

وهذه العمليات المنطقية لها دور كبير في البرمجة لانها تسمح للمبرمج ان يضع شروط معينة ان تحققت اي كان جوابها "نعم" فان البرنامج يذهب في مسار معين، وان لم تتحقق اي كان الجواب "لا" فيمكن للبرنامج ان يغير مساره ويذهب في اتجاه اخر.

TRY IT! Compute the logical expression for "Is 5 equal to 4?" and "Is 2 smaller than 3?"

```
>> 5 == 4
ans =
     0
>> 2 < 3
ans =
     1
```

TRY IT! Assuming *P* is true, use MATLAB to determine if the expression (P AND NOT(Q)) OR (P AND Q) is always true regardless of whether or not *Q* is true. Logically, can you see why this is the case?
First, assume Q is true:

```
>> (1&&~1) || (1&&1)
ans = 1
```

Now assume Q is false:

```
>> (1&&~0) || (1&&0)
ans = 1
```

TRY IT! Compute $(1 + 3) > (2 + 5)$.

```
>> 1 + 3 > 2 + 5
ans = 0
```

المثال الاخير يوضح ان تسلسل تنفيذ العمليات المنطقية يقع في اخر القائمة، وان كان يفضل ان نضع الاقواس في هذا المثال حتى لا يحصل ارباك للقارئ.

تطبيق: ماذا سينتج عن الكودات التالية:

1) >> 6<10 && 3<2
2) >> 6<10 || 3<2
3) >> 6<10 && ~3<2
4) >> 6 ~=12/2
5) >> 7 >= 7.5
6) >> 22 ==3*7

كما مر بنا سابقاً، فان البرنامج افتراضياً سوف يظهر نتائج كل عبارة نكتبها بعد ان نضغط على المفتاح Enter فمثلاً اذا كتبنا:

```
>> b=10-3
```

وضغطنا بعدها على مفتاح الادخال Enter فسوف يظهر ما يلي:

```
b=
     7
```

بالإضافة الى ذلك فسوف تظهر قيمة المتغير b في نافذة الـ workspace اي سيظهر ان المتغير b قد اسند اليه الرقم 7 وفي معظم الاحيان عند تقدمنا في العمل سوف لن نرغب في ظهور نتيجة كل ما نقوم به في السطر اللاحق لأن ذلك ربما يسبب لنا بعض الارباك وعدم التركيز على الصورة او المهمة الكاملة التي نحن بصدد حسابها لذلك نستطيع إنهاء الجملة التي نكتبها بالفارزة المنقوطة ((;)) Semicolon ووظيفتها هي انهاء الجملة وتنفيذها ولكن عدم اظهار ناتج الجملة مباشرة. لذلك نلخص القول: ان الجملة التي نهايتها مفتوحة سوف تظهر نتائجها مباشرة في السطر اللاحق بعد ضغطنا على مفتاح الادخال بينما الجملة المنتهية بالفارزة المنقوطة فأن البرنامج سوف يقوم بحسابها ولكنه لن يظهر نتيجتها مباشرة.

لاحظ اننا في الماتلاب لا نحتاج ان نعلن عن أنواع المتغيرات منذ بداية البرنامج (كما في بعض لغات البرمجة الاخرى)، بل ان نوع المتغير سوف يفهمه البرنامج تلقائياً من الشكل الذي نعطيه إياه واليك الأمثلة التالية:

>>n=7	% Integer Number	متغير عدد صحيح
>>R=2.5	% Real (float) Number	متغير عدد حقيقي
>>A=2*pi*R	% Real (float) number	متغير عدد حقيقي
>>myname='Ahmed'	% character	متغير رمزي قابسة منفردة
>>myname="Ahmed"	% String	متغير رمزي قابسة مزدوجة
>>d=2+3i	% Complex Scalar	متغير عدد معقد
>>b=[1 2 3]	% 1 by 3 row vector	متغير متجه صفي (مصفوفة ببعد واحد)
>>c=[1;2;3]	% 3 by 1 column vector	متغير متجه عمودي (مصفوفة ببعد واحد)
>>M=[1 2 ; 3 4]	% 2 by 2 square matrix	متغير مصفوفة ببعدين
>> a=5>2	%Logical (True or False)	متغير منطقي نتيجته (صفر أو واحد) او

برنامج الماتلاب يتعامل مع المتغيرات الرمزية characters او strings والاولى عادة تعني رمز واحد بينما الثانية تعني سلسلة من الرموز المتصلة. ويخزنها في مصفوفة (او الاصح متجه) بحيث تستطيع ان تنادي رمز واحد او مجموعة رموز منها.

TRY IT! Assign the `char` 'E7 is Awesome' to the variable `s`. Retrieve only the letter `E` and only the word `Awesome` from `s` using array indexing.

```
>> s = 'E7 is Awesome';
>> E = s(1)
E =
     E
>> Awesome = s(7:end)
Awesome =
     Awesome
```

الايعاز التالي sprintf وظيفته هو الطباعة على الشاشة، وسوف نأخذ اوامر القراءة والطباعة (الادخال والاخراج) لاحقاً.

TRY IT! Use the `sprintf` function to make the strings `s1 = 'My name is Timmy'` and `s2 = 'My name is Alex'`.

```
>> s1 = sprintf('My name is %s', 'Timmy')
s1 =
    My name is Timmy

>> s2 = sprintf('My name is %s', 'Alex')
s2 =
    My name is Alex
```

Shortcut يمكننا ضغط مفتاح السهم العلوي ↑ في لوحة المفاتيح وعندها سيقوم البرنامج بكتابة اخر ايعاز قمنا بإدخاله، ويمكننا الاستفادة من هذا الاختصار عندما نكتب ايعازاً طويلاً ويكون خاطئاً فبدلاً من كتابته مرة اخرى من جديد نستخدم هذا الاختصار ثم نقوم بتصحيح الخطأ.

تطبيق

اكتب في الـ Command Window الايعازات التالية واستكشف ما الذي سيفعله البرنامج

```
>> x=1:10;
>>y=log(x);
>> plot(x,y)
```

هذه النتيجة المذهلة التي ستحصل عليها ليست بهذه السهولة والاحترافية في كثير من لغات البرمجة الاخرى.

المتغيرات Variables

كل لغات البرمجة ومن بينها MATLAB تسمح باعطاء اسماء للمتغيرات التي تدخل في مسألة معينة، بحيث يمكن للبرنامج ان يغير قيم هذه المتغيرات اثناء مسيرته في حل تلك المسألة وحسب العمليات التي سينفذها على هذه المتغيرات.

شروط تسمية المتغيرات:

1- المتغير يجب ان يبدأ بحرف ابجدي وبعد ذلك يمكن ان تلحقه حروف اخرى او ارقام او الـ Under_score ويجب ان لا يتخلله فراغات.
2- الماتلاب على خلاف الفورتران يهتم ويميز بين الحروف الكبيرة والصغيرة؟ اي ان المتغير الذي اسمه Myroot ليس نفسه المتغير myroot فهما متغيرات مختلفان مستقلان عن بعضهما، ان هذا ربما يسبب بعض الارباك لذلك لا ينصح باستخدام هكذا متغيرات في البرنامج الواحد. وربما يفضل ان يحدد المبرمج منذ البداية ستراتيجيته في تسمية المتغيرات مثلاً انه يستخدم الحروف الصغيرة فقط، او غير ذلك.
3- احياناً هذا التلاعب بالحروف الصغيرة والكبيرة يجعلنا نستغني عن استخدام الـ _Underscore التي احياناً ينصح بالابتعاد عنها والسبب ان بعض البرامج التي ممكن ان تتفاعل مع الماتلاب قد لا تستخدم هذه الـ _Underscore ويكون تجنبها مثلاً باستخدام الحروف الصغيرة والكبيرة معاً فمثلاً نستطيع ان نكتب my_root بالشكل myRoot ليكون سهل القراءة.

4- طبعاً هناك بعض الكلمات المفتاحية المحجوزة للبرنامج نفسه لا يمكن ان نستخدمها كأسماء للمتغيرات مثلاً كلمة Print او كلمة File وغيرها الكثير، فعلينا تجنب مثل هذه الكلمات في أسماء المتغيرات.

5- ولا ننسى قاعدة ان تكون الاسماء معبرة عن طبيعة المتغير فمثلاً إذا كان المتغير المقصود هو نصف قطر دائرة فيفضل ان نسميه radius أو R بدلاً من ان نسميه x وان كان الأخير صحيحاً ولن يعترض عليه البرنامج ولكن الأول أفضل في التعبير وأسهل لمن يتتبع البرنامج.

6- ملاحظة مهمة: ان (كل أوامر و دوال) الـ MATLAB عندما نستخدمها يجب أن تبدأ بالحروف الصغيرة وليس الكبيرة، واليك قائمة بالاوامر أو الدوال الشائعة (لاحظ انها تبدأ بالحروف الصغيرة):

Function	Meaning
y=cos(x);	% cosine of x
y=sin(x);	% sine of x
y=tan(x);	% tangent of x
y=acos(x);	% arc cosine of x
y=asin(x);	% arc sine of x
y=atan(x)	% arc tangent of x
y=exp(x);	% exponential of x
y=log(x);	% natural log of x
y=log10(x);	% common log of x
y=sqrt(x);	% square root of x
y=abs(x);	% absolute value
z=mod(y,x);	%reminder of y/x

Function	Meaning
y=round(x);	%round nearest to x
y=floor(x);	%round x down
y=ceil(x);	%round x up
y=num2str(x);	%convert num to string
y=str2num(x);	%convert string to num
y=real(x);	% real part of x
y=imag(x);	% imaginary part of x
y=angle(x);	% angle of complex x
y=max(x);	% find maximum of x
y=min(x);	%find minimum of x
[n,m]=size(A);	%dimension of array A
y=rand	%generate random num

هناك بعض الايعازات المختصة في اعطاء معلومات عن المتغيرات التي ادخلناها لحد الان مثلاً:

who هذا الايعاز يعطي ملخص بالمتغيرات التي استخدمناها لحد الان.

whos يشبه الاول ولكنه يعطي معلومات اكثر تفصيلاً عن هذه المتغيرات مثل نوعها وحجمها.

clear وهو سوف يمحو كل المتغيرات التي استخدمناها وينهي وجودها.

clear *variablename* وهذا يمحو متغيرات نختارها بعينها.

احياناً يكون التعبير الرياضي الذي نكتبه طويلاً جداً ولا يسعه سطر واحد، لذلك سنحتاج الى رمز يشير الى الـ Continuation اي الاستمرارية أي ذلك الرمز الذي نضعه في منتصف الجملة لكي نستطيع ان نكمل بقيتها في السطر التالي بحيث لا يعتبرها البرنامج جملة جديدة وانما تكملة للجملة التي قبلها، ان هذا الرمز في برنامج الماتلاب هو (... ثلاث نقاط او اكثر) اي نستطيع ان نكتب ما يلي:

```
>>a=3+55-62+4-5...
+22-1

a=
        16
```

أبرز الـ Operators المستخدمة في برنامج الماتلاب (للعمليات الحسابية):

+	addition
−	negation, subtraction
*	multiplication
/	division (divided by, e.g., 10/5 is 2)
\	division (divided into, e.g., 5\10 is 2)
^	exponentiation (e.g., 5^2 is 25)

Table 2.1 Arithmetic Operations Between Two Scalars (Binary Operations)

Operation	Algebraic Syntax	MATLAB Syntax
Addition	$a + b$	**a + b**
Subtraction	$a - b$	**a - b**
Multiplication	$a \times b$	**a*b**
Division	$\frac{a}{b}$ or $a \div b$	**a/b**
Exponentiation	a^b	**a^b**

الثوابت Constant

يمكننا ان نعطي رقم ثابت لأي رمز بحيث نعتبره ثابت على طول البرنامج، ولكن هناك الثوابت الاكثر شيوعاً (العالمية) مثلاً:

pi	3.14159....
i	$\sqrt{-1}$
j	$\sqrt{-1}$
inf	infinity ∞
NaN	stands for "not a number," such as the result of 0/0

لاحظ ان استخدام حرفين مختلفين أعلاه للإشارة الى الرقم المعقد له أسباب تفضيلية، فمثلاً في اختصاص الكهرباء فهم يستخدمون حرف الـ i للتعبير عن الـ Current اي التيار، لذلك يمكنهم ان يستخدموا الحرف j للتعبير عن الرقم المعقد وبذلك لا يحصل الخلط عندهم.

اما العدد الطبيعي e=2.7183 فهو غير موجود كثابت في برنامج الماتلاب ولكننا يمكن ان نحصل عليه كلما احتجنا الى ذلك عن طريق كتابة الدالة exp(1) حيث انها ستعطي نفس النتيجة المطلوبة.

وفي هذا السياق (وربما بسبب ذلك) يجدر بنا تذكر ان حرف الـ e يستخدمه برنامج الماتلاب للتعبير عن الارقام بالصيغة الاسية (العلمية) وهو يعبر عن كل المقدار (×10^) فمثلاً إذا أردنا ان نكتب العدد p=6000 فيمكننا كتابته بالصيغة العلمية أي بالشكل p=6e3. وهذه الصيغة يفضل استخدامها للأرقام اما الكبيرة جداً او الصغيرة جداً مثلاً:

```
Avogadro's_constant = 6.022e23;
Iron_diameter = 140e-12; or
Iron_diameter = 1.4e-10
```

وهنا ضروري ان لا نترك فراغ بين العدد الحقيقي الاول وحرف الـ e لكي لا يعتبرها البرنامج رقمين وانما رقم واحد.

توليد رقم عشوائي

احياناً نحتاج الى Random Number اي رقم عشوائي وفي الماتلاب يمكننا انشاء أو توليد Generate رقم عشوائي بكتابة كلمة واحدة فقط وهي rand بدون أقواس (لانها ليست دالة) وهذه الكلمة سوف تعطينا دائماً رقم عشوائي يتراوح بين (1-0) اي انه دائماً رقم عشري اقل او يساوي واحد واكبر او يساوي صفر، وفي كل مرة نستخدمه سيعطينا رقم مختلف (وهذه هي الفكرة من الرقم العشوائي) مثلاً الايعاز التالي ممكن ان يعطي:

```
>>rand
Ans=
        0.9501
>> rand
ans=
        0.2311
```

(يمكن ان نستخدم هذه التقنية في تطبيقات القرعة غير المتحيزة، حيث لا احد يستطيع ان يتنبأ بالرقم الذي سوف تختاره الحاسبة)

ان ناتج هذا الايعاز سيكون دائماً رقم حقيقي (فيه فاصلة عشرية) وهو دائماً اقل من الواحد، ولكننا اذا اردنا هذا الرقم ان يكون اكبر من الواحد فعلينا أن نضربه بالرقم المناسب، فأذا اردناه بين الواحد والعشرة فنضربه في 10 أو اذا اردناه بين الـ 10 والـ 100 فنضربه في 100 وهكذا نتحكم في حدود قيمته حسب رغبتنا فيكون:

```
>>rand*10
ans=
        4.216
```

أما اذا اردنا نحن ان نتحكم بالضبط بالرقم العشوائي بحيث يقع بين حدين معينين احدهما الـ (low) والثاني الـ (high) فيمكننا القيام بذلك باستخدام التعبير الحسابي التالي:

```
>>low=3;
>>high=15;
>>n=rand*(high-low)+low
```

هذا الايعاز سوف يعطينا رقم عشوائي حقيقي يقع بين الـ 3 والـ 15

```
>> n=rand*(15-3)+3
n =
  13.8695
```

ايعازات التقريب (أو جبر الكسور) Round, Ceil, Floor & Fix

الايعازات او الدوال الأربعة أعلاه كلها تعمل على تقريب الاعداد الحقيقية الى اعداد صحيحة، أي جبر الكسور، ولكن كل واحد منها يسلك مسلكاً مختلفاً في التقريب الى العدد الصحيح، فربما يكون الناتج اكبر من العدد الأصلي او أصغر منه حسب الدالة التي نختارها.

الدالة round تقوم بتقريب العدد الحقيقي الى العدد الصحيح الاقرب فعلاً (جبرياً) مثلاً:

```
>> round (5.4)
Ans= 5
>> round (5.8)
Ans = 6
```

وهذا ينطبق على الاعداد الموجبة والسالبة. اما الحالة الخاصة عندما يكون الكسر هو بالضبط نصف مثلاً (4.5) او (12.5-) فهنا فقط سيكون سلوك هذا الايعاز هو الابتعاد عن الصفر على خط الاعداد، اي يذهب للعدد الاكبر في حالة الموجب والى الاصغر في حالة السالب.

```
>> round (5.5)
Ans= 6
>> round (-7.5)
Ans = -8
```

الدالة او الايعاز Ceil وتعني السقف ((الجبري)) اي انه سيعطي العدد الاكبر جبرياً سواء للموجب او السالب، اي انه دائماً يتجه في التقريب الى يمين خط الاعداد.

```
>>ceil(4.1)
Ans = 5
```

```
>>ceil(-8.7)
Ans = -8
```

الدالة او الايعاز floor وهو عكس الايعاز السابق لأنه يعني الأرضية لذلك فهو يعني الاصغر ((جبرياً) اي انه سيعطي العدد الأصغر جبرياً سواء للموجب او السالب، اي انه دائماً يتجه في التقريب الى يسار خط الاعداد.

```
>>floor(16.9)
Ans = 16
>>floor(-25.5)
Ans = -26
```

الدالة او الايعاز fix وهو يتجه الى الصفر دائماً على خط الاعداد لأنه فقط يقوم بالتخلص من الكسور اياً كانت قيمتها.

```
>>fix(67.889)
Ans = 67
>>fix(-24.628)
Ans = -24
```

ولتلخيص ايعازات جبر الكسور السابقة نقول:

round(x): هو التقريب المنطقي الشائع الى الاقرب جبرياً، وفي حالة النصف يبتعد عن الصفر في خط الاعداد

fix(x): يتخلص من الكسر نهائياً وبالتالي فهو يقترب الى الصفر في خط الاعداد

ceil(x): يعطي سقف العدد جبرياً أي يتجه الى يمين خط الاعداد

floor(x): يعطي أرضية العدد جبرياً أي يتجه الى يسار خط الاعداد

إذا أردنا ان نحصل على رقم عشوائي وفي نفس الوقت يكون صحيح Integer فيمكننا ان نستفيد من أحد الدوال الأربعة أعلاه مع الكلمة rand ويكون الايعاز العام والشامل هو:

```
>>round(rand*(high-low)+low)
```

مثل:

```
>> round(rand*(15-3)+3)
ans =
   5
```

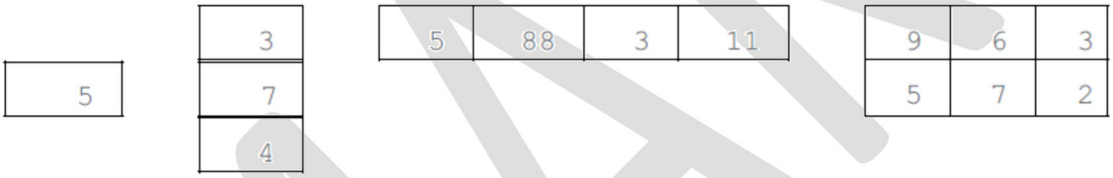
هذا الايعاز سوف يعطينا رقم عشوائي صحيح يتراوح بين الصفر والعشرة

```
>> round(rand*10)
ans =
   9
```

المتجهات والمصفوفات Vectors and Matrices

المتجه والمصفوفة هي مفاهيم رياضية وفيزيائية. اما في البرمجة فاما ان تعبر عن نفس المفاهيم الرياضية والفيزيائية او يمكن ان تستخدم لأغراض اخرى. ففي الحاسوب هي عبارة عن كمية Entity تخزن عدد معين من الارقام او حتى الرموز والكلمات، ويمكن ان تكون شبيهة بالجداول فالجدول بغض النظر عن محتوياته لديه صفوف ولديه اعمدة. بالنسبة للمتجه فيكون من بعد واحد وقد يكون متجه (صفي) أي Row Vector او متجه (عمودي) Column Vector اما المصفوفة فتكون ذات بعدين ويعبر عنها r×c أي عدد من الصفوف rows وعدد من الاعمدة columns وبالتالي فأن العدد الكلي للعناصر التي تختزن داخل هذه المصفوفة هو n= r×c.

الشكل التالي يمثل من اليسار الى اليمين قيمة عددية مجردة Scalar ثم متجه عمود Column Vector ثم متجه صفي Row Vector ثم مصفوفة ذات بعدين (2×3) Matrix of. القيم الموجودة داخل هذه المصفوفات تعرف بعناصر المصفوفة Elements. (يمكننا اعتبار الرقم المجرد حالة خاصة لمصفوفة 1×1 كما ان المتجهات هي ايضاً حالة خاصة من المصفوفات ببعد واحد).



ان برنامج الماتلاب Matlab قد كتب اصلاً للتعامل مع المصفوفات. حيث ان تسميته هي مختصر لـ Matrix Laboratory اي مختبر المصفوفات لذلك فان عملية انشاء او تكوين المصفوفات بأنواعها والتعامل معها يكون أسهل بكثير واقوى في هذا البرنامج عما موجود في معظم لغات البرمجة الاخرى.

يمكننا انشاء متجه صفي Row Vector بكل بساطة وذلك مباشرة بكتابة عناصره اما مفصولة بفراغات Spaces او فوارز اعتيادية Commas وحصر كل هذه العناصر داخل زوج واحد من الاقواس المربعة [العبارتان التاليتان تؤديان نفس الغرض]:

```
>>V = [ 1      5      3      4]
V=
1      5      3      4
>>V=[1 ,5,3,4]
V=
1      5      3      4
```

بينما اذا اردنا متجه عمود فان الفاصلة هنا بين العناصر يجب ان تكون الفارزة المنقوطة Semicolon

```
>> v=[4;12;3]
v =
```

```
   4
  12
   3
```

وبالتالي فان ادخال مصفوفة 2×2 مثلاً سيكون بالشكل التالي:

```
>> M=[3 5 ; 6 7]
M =
   3   5
   6   7
```

المشغل (: Colon) والايعاز (linspace)

في احيان كثيرة (وليس كل الأحيان) فأن عناصر المصفوفة او المتجه تكون ذات سياق او توجه معين pattern اي مثلاً تكون على شكل ارقام متسلسلة تفصلها خطوة ثابتة مثلاً نريد ان ننشأ متجه صفي عناصره هي الارقام من واحد الى ستة، او مثلاً الاعداد الزوجية من 2 الى 10، ففي هذه الحالات هناك مشغل Operator معد خصوصاً لهذا الغرض. وهو النقطتين الشارحة (:) The Colon Operator وهذه النقطتين الرأسية تقوم بأنشاء اي سلسلة من الاعداد المفصولة **بخطوة ثابتة** مثلاً:

```
>>V=1:6

V=

1     2     3     4     5     6
```

والشكل في الايعاز أعلاه يفترض ان الخطوة التي سوف يسلكها المتجه في الوصول من الرقم الأول الى الرقم الاخير هي الخطوة الافتراضية وهي هنا في برنامج الماتلاب وحدة واحدة.

أما عندما تكون الخطوة ليس بالضرورة مساوية للواحد، فنستخدم الشكل العام وهو **first:step:last**

```
>>V=2:2:10

V=

2     4     6     8     10
```

والسؤال الذي يتبادر هنا ماذا لو ان الاضافة الاخيرة لم تصل بالضبط الى الحد الاخير وجواب الماتلاب هنا هو ان الحد الاول سيكون دائماً موجود في المتجه، ثم الحدود الوسطية سوف تكون عبارة عن الاضافات المتكررة على الحد الاول بمقدار الخطوة اي الـ Step اما الحد الاخير فليس شرطاً ان يكون دائماً موجود في المتجه كما لا نتوقع من الايعاز ان يدخل ارقام هي اصلاً خارج النطاق الأصلي المعطى لذلك فأن الحد الاخير في المتجه هو دائما أما يساوي الحد الأخير او اصغر منه مثال على ذلك:

```
>>V=1:2:8
```

V=

1 3 5 7

والسبب ببساطة ان الإضافة على الرقم الأخير أي الـ 7 سوف تخرجنا من النطاق الأصلي للايعاز لذلك يتوقف المتجه عند الرقم 7.

ملاحظة يمكننا ان نستخدم قيمة سالبة للخطوة مثلاً:

>>VV=8:-2:1

VV=

8 6 4 2

في احيان اخرى نحتاج ان نقوم بعمل يشبه الى حد ما عمل المشغل colon اعلاه، ولكن فيه اختلاف نوعي في الحاجة من الايعاز. أي اذا كان لدينا الحد الأدنى والحد الأعلى ونريد ان نقسم المسافة بينهما الى قطع متساوية فماذا ستكون النقاط الوسطية؟ (اي أن عدد النقاط هنا معلوم ولكن الخطوة هي المجهولة) والايعاز هو linspace مثال:

>>vv=linspace(3,10,5)

vv=

3.0000 4.7500 6.5000 8.2500 10.0000

ملاحظات:

1) هذا الايعاز يشبه الدالة والسبب هو حاجته للأقواس حيث هذا الامر يشبه ان نكتب sin(3.14).
2) لاحظ ان الايعاز يحتاج الى ثلاثة operands وهي حسب التسلسل من اليسار الى اليمين، الحد الأدنى ثم الحد الأعلى ثم (**عدد النقاط الكلية**) المطلوبة (**بما في ذلك النقطتين الطرفيتين**).
3) الملاحظة الثالثة التي تختلف عن المشغل (:) هو اننا نرجح ان تكون الخطوة هذه المرة في معظم الأحيان عدداً حقيقياً وليس عدداً صحيحاً ربما يكون مثلاً بأربعة مراتب بعد الفارزة، لأن الايعاز هو الذي سيحسبها وحسب المعادلة التالية:

$$\Delta x = \frac{X_n - X_1}{n - 1}$$

بينما في الايعاز السابق لأننا نحن الذين نختار الخطوة هناك ففي معظم الأحيان ستكون اما عدد صحيح او بفاصلة واحدة او اثنتين.

التعامل مع عناصر المصفوفات (او مناداتها)

بالنسبة الى المتجهات احادية البعد فأن عنوان اي عنصر داخل المتجه هو عبارة عن تسلسل (أي موقع) ذلك العنصر داخل المتجه علماً ان برنامج الـ MATLAB يبدأ التسلسل دائماً من الواحد (لاحظ ان هناك بعض لغات البرمجة مثل Python وغيرها ولأغراض رياضية متقدمة تبدأ التسلسل من الصفر) ولكن هنا في الماتلاب فأن التسلسل يبدأ اعتيادياً من الواحد، فالعنصر الذي يقع في العمود رقم 100 له التسلسل 100.

إذا اردت مناداة العنصر رقم 6 مثلاً في متجه من 20 عنصر فكل الذي تقوم به إنك تكتب نفس اسم المتجه وليكن V ولكن بدلاً من استخدام الاقواس المربعة الكبيرة التي تستخدم في التعريف، تستخدم الاقواس الاعتيادية الصغيرة وتذكر تسلسل ذلك العنصر بالذات مثال:

```
>>x=V(6);
```

يمكن ان ننادي ايضاً أكثر من عنصر واحد (**نطاق من العناصر**) في نفس الوقت مثلاً:

```
>>b=V(3:7);
```

هذا سوف يجلب العناصر التي تسلسلها من 3 الى 7 من المتجه الاصلي V وسوف يعطيها الى المتغير b ويجب ان نلاحظ ان المتغير b مادام قد اخذ صف من الارقام فهو إذا سوف لن يكون متغيراً رقمياً عادياً scalar بل هو متجه صفي ذو خمسة عناصر. (وهذه الميزة من مواطن القوة الكبيرة في برنامج الماتلاب حيث اننا صنعنا الان متجه بشكل عرضي وبطريقة بسيطة جداً، بينما كان في اللغات الأخرى انشاء متجه او مصفوفة يحتاج الكثير من التحضيرات وحجز Dimension له منذ البداية).

طبعاً يمكننا تغيير قيمة اي عنصر محدد داخل المتجه مثلاً:

```
>>v(5)=1000;
```

هذا سوف يجعل العنصر رقم 5 في المتجه الأصلي يبدل قيمته من القيمة القديمة الى القيمة الجديدة وهي 1000.

ومن المرونة العالية التي يتعامل بها برنامج الماتلاب مع المصفوفات (على خلاف الكثير من لغات البرمجة التي تحتاج ان تحدد ابعاد كل المصفوفات في البرنامج منذ البداية ولا تستطيع ان تغير هذه الابعاد اطلاقاً داخل البرنامج) بينما برنامج الماتلاب يمكنه ان يغير الابعاد والعناصر في اي لحظة مثلاً:

```
>>d=[3      4      7]
d=
3      4      7
```

الان لو كتبنا:

```
>>d(5) = 9
```

فان ابعاد المتجه ستتغير من حالتها الاصلية وهي ثلاثة عناصر الى الحالة الجديدة وهو متجه بخمسة عناصر، والعنصر الخامس سوف يأخذ القيمة 9 كما موجود في الايعاز ولكن السؤال ما الذي سوف يحصل للعنصر الرابع الذي لم يذكر لا في المتجه الأصلي ولا حتى في الايعاز؟ في الحقيقة ان الماتلاب عنده توجه عام في اعطاء قيمة الصفر كقيمة افتراضية للكثير من الحالات، وهذه احدى هذه الحالات، اما الأسباب فهي رياضية وعملية حيث تتكرر هذه الحاجة كثيراً في التطبيقات الرياضية والعددية لذلك فأن المتجه الجديد سوف يكون:

D=

3 4 7 0 9

متجه العمود Column Vector

ان طريقة انشاء او كتابة متجه العمود هي مشابهة كثيراً للمتجه الصفي فنحن ايضاً سوف نستخدم الاقواس المربعة لتحيط بكل عناصر المتجه ولكننا بدلاً من ان نفصل بين هذه العناصر بالفراغ او بالفوارز الاعتيادية فهنا يجب ان نفصلها بالفوارز المنقوطة (;) semi colon.

```
>>c=[1;2;3;4]
```

C=

1

2

3

4

لاحظ اننا هنا مع الـ Column Vectors لا نمتلك تلك الادوات السهلة والسريعة التي كنا نمتلكها لإنشاء Row Vector والمقصود هنا كل من مشغل النقطتين الرأسية (:) وكذلك ايعاز الـ linspace حيث انهما يختصان في انشاء المتجهات الصفية فقط، ولكن البرنامج يتغلب على هذه الحاجة بوجود ايعاز اخر بسيط جداً وظيفته فقط قلب أي متجه صفي الى متجه عمودي أو العكس، وهذه العملية هي اصلاً مشهورة وكثيرة الاستخدام في عالم المصفوفات وهي تسمى بالتدوير او الـ Transpose ولذلك فأن الطريقة المتبعة اذا احتجنا الى متجه عمود يمتلك المواصفات التي يمكن انشاءها عن طريق المشغل (:) او عن طريق الايعاز linspace فأننا ننشأه اولاً كمتجه صفي بإحدى هذه الطرق السريعة ثم بعمل الـ Transpose حيث ان هذه العملية عندما نجريها على متجه صفي فأنها ستحوله الى متجه عمود واذا اجريناها على متجه عمود تحوله الى متجه صفي واذا اجريناها على مصفوفة مستطيلة m×n فأنها سوف تحولها الى مصفوفة مستطيلة اخرى n×m.

ان هذا المشغل Operator في الماتلاب هو بكل بساطة الفاصلة الصغيرة العليا (‘) apostrophe وللتذكير فأن علامة الـ Transpose في موضوع المصفوفات في الرياضيات هي ان يوضع الحرف الكبير (T) كأس للمصفوفة مثل b^T ، اما هنا في الماتلاب فأننا نستخدم الفاصلة الصغيرة العليا b′ . الان لننظر الى التطبيق التالي:

```
>> b=1:3
b =
   1   2   3
>> c=b'
c =
   1
   2
   3
```

انشاء المصفوفات المستطيلة Matrix Creation

من خلال تعرفنا على طريقة انشاء وكتابة كل من المتجه الصفي ومتجه العمود يمكننا بكل بساطة الان ان ننشئ المصفوفة المستطيلة وذلك باستخدام نفس الاقواس المربعة ثم اننا لعناصر الصفوف نفصل بينها اما بالفراغ او بالفارزة (كالسابق) وعناصر الاعمدة نفصلها بالفارزة المنقوطة (او مفتاح الادخال).

```
>>mat=[4 3 1 ; 2 5 6]
Mat=
4       3       1
2       5       6
```

يجب ان ننتبه هنا عندما نكتب عناصر الصفوف فأنها يجب ان تكون متجانسة بالعدد ولا نتوقع ان البرنامج سوف يعوض النقص في صف معين بإضافة الاصفار لأن البرنامج سيفترض حصول خطأ في الادخال وسوف يعترض ويعطي رسالة خطأ ويعتبر المصفوفة غير متجانسة ويرفض انشاءها.

```
>> mat1=[1 2 3 ; 5 7]
Error using vertcat
Dimensions of arrays being concatenated are not consistent.
```

يمكننا في انشاء المصفوفات ان نستفيد من مشغل الـ Colon (:) في الصفوف كما كنا نستفيد منه في المتجهات الصفية:

```
>>mat=[2:4;3:5]
mat=
2       3       4
3       4       5
```

يمكننا البرنامج ايضاً من الاستعاضة عن الفارزة المنقوطة التي تفصل الصفوف عن بعضها بضغط مفتاح الادخال Enter في كل مرة مثل:

```
>>newmat=[6 2 7

4 5 3

7 7 1]

newmat=

6     2     7

4     5     3

7     7     1
```

تكوين مصفوفة عشوائية العناصر

كنا قد تناولنا سابقاً ان مجرد كتابة الكلمة او الامر rand فانه سوف يعطينا رقم عشوائي موجب أصغر من الواحد. الان التعامل مع المصفوفات العشوائية له ايضا بعض الخصوصية، فأذا كتبنا rand(3) فأن ذلك سوف يعطينا مصفوفة مربعة ابعادها هي 3×3 وعناصرها تكون عشوائية وهي موجبة أصغر من الواحد كلها.

وهنا سيخطر سؤالين؟ كيف سيعني هذا الايعاز مصفوفة؟ ولماذا هي مربعة؟

سنجيب عن هذين السؤالين في الفقرة التالية، اما اذا اردنا ان نعمل متجه صفي عشوائي، فهنا يجب ان نكتب الايعاز ونعطي بعدين للمصفوفة احدهما 1 والاخر نذكر فيه عدد العناصر المطلوبة:

```
>> rand(1,4)
ans =
   0.8147   0.9058   0.1270   0.9134
>> rand(3,1)
ans =
   0.6324
   0.0975
   0.2785
```

المصفوفات المربعة الخاصة (مصفوفة الصفر ومصفوفة الوحدة)

الملاحظة المهمة هنا ان برنامج الـ MATLAB يهتم كثيراً **بالمصفوفات المربعة** لأنها المصفوفات الاكثر ظهوراً في التطبيقات الرياضية في الكثير من الاختصاصات.

ونظراً للحاجة المتكررة والملحة لنوعين من المصفوفات الخاصة وهي المصفوفة الصفرية أي التي كلها اصفار والمصفوفة الأخرى التي كل عناصرها هي قيمة الواحد، فان الماتلاب له دالتان تشبهان عمل الـ rand (من حيث فرض ان المصفوفة مربعة) تقومان خصوصاً بهاتين المهمتين:

>>zeros(3) هذا الايعاز سوف يعطي مصفوفة مربعة فيها تسعة اصفار

>>ones(5) هذا الايعاز سوف يعطي مصفوفة مربعة من خمس وعشرون عنصر كلها واحد

>>rand(4) هذا الايعاز سيعطي مصفوفة مربعة من 16 عنصر كلها ارقام عشوائية اصغر من واحد

>>eye(6) وهذا الايعاز سيعطي مصفوفة مربعة من 36 عناصرها اصفار ولكن عناصر القطر الرئيس كلها واحد

لاحظ ان الايعازات الأربعة السابقة استخدمنا فيها الشكل السريع وهو الخاص بأنشاء مصفوفة مربعة سريعة، ولكن نفس هذه الايعازات يمكن ان تعمل بشكلها العام الذي يكون المصفوفات المستطيلة وحسب الابعاد التي نريدها مثلاً:

```
>>C=zeros(2,5);
>>P=ones(6,1);
>>Q=rand(3,4)
```

طبعاً مناداة او استدعاء أحد عناصر المصفوفة بعينه يخضع لنفس القواعد السابقة في المتجهات ويجدر الانتباه هنا في ان المصفوفات التي فيها صفوف واعمدة فأننا دائماً نذكر الصف اولاً ثم العمود اي إذا أردنا استدعاء عنصر معين يقع في الصف السادس والعمود الثامن للمصفوفة mat فيكون استدعائه بالشكل: mat(6,8)

كان ذلك استدعاء عنصر واحد من مصفوفة مستطيلة ماذا لو أردنا ان نستدعي جزء أو نطاق من المصفوفة والذي سيكون بدوره عبارة عن مصفوفة اصغر sub matrix يمكننا هنا طبعاً الاستفادة من مشغل الـ colon (:) والملفت للنظر اننا هنا نستطيع الاستفادة منه في كل من الصفوف والاعمدة في نفس الوقت لاحظ:

```
>>mat=[2:4 ; 3:5]
mat=
2       3       4
3       4       5
>>mat(2,2)
Ans=
4
>>mat(1:2;2:3)
Ans=
3       4
4       5
```

لاحظ هنا وجود مسألة دقيقة جداً ومربكة في نفس الوقت؟ وهي ان استخدام الـ (:) Colon في الايعاز الأول **[عندما استخدمنا الاقواس المربعة الخاصة بانشاء المصفوفات]** يختلف تماماً عن استخدامه نفسه في الايعاز الثاني **(مع الاقواس الصغيرة والخاصة بالمناداة)** ففي الحالة الأولى كان لتوليد العناصر نفسها أي ان 2:4 كانت تعني انشاء العناصر 2، 3، 4 بينما في الحالة الثانية كان لا يعني العناصر نفسها **(بل كان يعبر عن العناوين فقط)** وليس له أي علاقة بقيم العناصر الفعلية؟ أي ان الـ 1:2 في الايعاز الأخير لم تكن تعني وجود عناصر هي 1 و 2 بل تعني العنصر رقم واحد والعنصر رقم 2 التي يمكن ان تكون قيمها أي شيء آخر.

Shortcut عندما نكتب فقط الـ (:) colon داخل الاستدعاء بدون حد ادنى وحد اعلى فانه سيعني اما كل الصفوف او كل الاعمدة حسب موقعه مثلاً: mat(:,10) هذا سوف يعني كل الصفوف الموجودة في العمود العاشر، بينما mat(1:3,:) سوف يعني الصفوف من واحد الى ثلاثة اي الاول والثاني والثالث مع كل الاعمدة، اي المطلوب هو كل الاعمدة اما الصفوف فهي فقط الاول والثاني والثالث.

طبعاً باستخدام هذا الاختصار صار من السهولة بمكان استدعاء اي صف او اي عمود داخل المصفوفة لوحده مثلاً:

>>mat(:,14) هذا الايعاز يعني استدعاء العمود 14 كله (اي بجميع صفوفه)

>>mat(3,:) هذا الايعاز يعني استدعاء الصف الثالث كله (اي بجميع اعمدته)

هناك مسألة ربما تكون مربكة بعض الشيء في استدعاء عناصر المصفوفة ثنائية الابعاد باستخدام index واحد، اي رغم ان المصفوفة قد تكون مستطيلة ولكننا نستطيع ان نستدعي احد عناصرها بطريقة اخرى فبدلاً من ان نذكر رقم الصف ورقم العمود الذي يقع فيه ذلك العنصر نستطيع ان نذكر رقم واحد، ولكن ماذا يمثل ذلك الرقم الواحد، في الحقيقة فأن برنامج الماتلاب يعطي في داخله ترقيم اخر مختلف لعناصر المصفوفة وهو يمشي في ذلك الترقيم بشكل عمود عمود اي انه ينزل اولاً في العمود الاول عنصراً عنصراً من الاعلى الى الاسفل حتى اذا انتهى من العمود الاول انتقل الى العمود الثاني من الاعلى الى الاسفل وهكذا اي ان الترقيم يكون من الاعلى الى الاسفل ثم من اليسار الى اليمين؟

```
>>matt=[1 7 9 2 ;3 5 6 4]

matt=

1     7     9     2

3     5     6     4

>>matt(4)

Ans=

5
```

ابعاد المصفوفة Dimensions

هناك ايعازان هما **length** و **size** كلاهما يعطي ابعاد المصفوفة ولكن الاول يختص بالمتجهات الصفية أو العمودية فقط لأنه يعطي قيمة واحدة فقط وهي طول المتجه او عدد عناصره. بينما الثاني size فيستخدم مع كل من المتجهات وكذلك المصفوفات ذات البعدين، وهو في الحالتين سيعطي رقمين وهما عدد الصفوف وعدد الاعمدة. وتجدر الإشارة هنا ان الـ length له استخدام اخر (له علاقة بطول المتجهات) وهو انه يجد طول المتغير الرمزي، حيث لو كان لدينا متغير رمزي هو عبارة عن كلمة 'Book' مثلاً فأن استخدام هذا الايعاز معه سوف يعطينا طول هذا المتغير الرمزي وهو هنا 4 أي انه متكون من أربعة حروف او رموز.

```
>>vec= -2:1
vec=
-2     -1     0     1
>>length(vec)
Ans=
4
>>size(vec)
Ans=
1     4
>> a='hello';
>> b=length(a)
b =
   5
```

هناك ايعاز مقارب لهذين الايعازين ولكنه يعطي العدد الكلي لعناصر المصفوفة او المتجه بغض النظر عن عدد الصفوف وعدد الاعمدة والايعاز هو **numel** وهو مختصر لـ number of elements.

```
>>h=numel(vec)

h=

4
```

العمليات الحسابية وغير الحسابية على المصفوفات

يجب ان ينتبه الطالب الى ان موضوع المصفوفات هو بالاساس موضوع او فرع قائم بذاته في علم الرياضيات وله قواعده الخاصة والدقيقة وتستخدمه مختلف العلوم التطبيقية (رياضياً) في حل العديد من المسائل.

ومن جانب البرمجة فيمكن اما التعامل مع نفس موضوع المصفوفات الرياضي الأصيل واجراء مختلف العمليات الرياضية عليها. او هناك جانب اخر وهو اعتبار المصفوفة برمجياً هي مجرد جدول فيه مجموعة من الارقام او الرموز وهناك حاجة للتعامل معها بطريقة ما.

العمليات الحسابية

في الرياضيات يمكن (جمع) او (طرح) مصفوفتين بشكل مشابه كثيراً لما نفعله مع الارقام المجردة Scalar على ان تكون المصفوفتان متطابقتان تماماً بالحجم، حيث ان الجمع او الطرح سيعطي مصفوفة ثالثة بنفس الحجم وعناصرها هي حاصل (جمع) او (طرح) العناصر المتقابلة في المصفوفتين.

```
>> a=[2 3 6 ; 1 4 5; 7 5 -2]
a =
   2   3   6
   1   4   5
   7   5  -2
>> b=[-1 3 0 ; 2 8 4 ; 0 6 2]
b =
  -1   3   0
   2   8   4
   0   6   2
>> c=a+b
c =
   1   6   6
   3  12   9
   7  11   0
```

اما عملية ضرب وقسمة المصفوفات رياضياً فهو مختلف تماما عن عالم الارقام Scalar وله قواعده الخاصة الدقيقة جداً. فمثلاً لا يمكن ضرب مصفوفتين مع بعضهما الا اذا كان (عدد **اعمدة** الاولى الى **اليسار**) مساو (لعدد **صفوف** الثانية الى اليمين)، وبالتالي اذا استطعنا ان نضرب مصفوفتين AB فربما لا نستطيع ان نضربهما بالشكل المعكوس BA.

أما القسمة، فيمكن القول ان "**لا توجد عملية قسمة اصلاً في المصفوفات**"، وانما نأخذ المصفوفة الموجودة في المقام، ونعمل لها مقلوب **Inverse** (وهذه عملية طويلة ومفصلة) ثم نضرب هذا المقلوب في المصفوفة الموجودة في البسط، اذا كانت عملية الضرب ممكنة من حيث عدد الاعمدة للاولى والصفوف للثانية.

(طبعاً استناداً للكلام اعلاه فان المصفوفات المربعة كحالة خاصة والمتساوية بالحجم يمكن ضربها وقسمتها على بعضها) لان عدد اعمدة الاولى ستساوي عدد صفوف الثانية.

```
>> d=a*b
d =
   4  66  24
   7  65  26
   3  49  16
```

```
>> e=a/b
e =
 -11.0000  -4.5000  12.0000
  -9.0000  -4.0000  10.5000
  25.0000  16.0000 -33.0000
```

كذلك اذا اردنا ضرب رقم Scalar في مصفوفة في الرياضيات فسوف يضرب هذا الرقم بكل عناصر المصفوفة، واذا اردنا ان نقسمه على المصفوفة (فلا يمكن ذلك) اما اذا اردنا ان نقسم المصفوفة نفسها على رقم معين، فاننا سوف نقلب ذلك العدد ثم نضربه في المصفوفة:

```
>> f=10*a
f =
   20   30   60
   10   40   50
   70   50  -20
>> g=20/a
Error using  /
Matrix dimensions must agree.
>> g=1/20*a
g =
   0.1000   0.1500   0.3000
   0.0500   0.2000   0.2500
   0.3500   0.2500  -0.1000
```

عمليات خارج رياضيات المصفوفات

لاجراء بعض العمليات على كل عناصر المصفوفة عنصراً عنصراً دفعة واحدة، مثلاً نريد ان نقسم عدد معين على جميع عناصر المصفوفة (وهذا يختلف عن قسمة عدد على مصفوفة) او اردنا ان نقلب كل عناصر المصفوفة عنصراً عنصر (وهذا يختلف عن قلب المصفوفة)، او اي عملية اخرى فان الـ MATLAB يعطي هذه الامكانية بمجرد وضع رمز النقطة (.) Dot قبل تلك العملية الحسابية، ليفهم البرنامج ان هذه العملية الحسابية سوف تجرى على عناصر المصفوفة واحداً واحد وليس على المصفوفة نفسها كقيمة رياضية.

مثلاً اذا اردنا ان نضرب مصفوفتين ببعضهما ولكن بطريقة ان كل عنصر يضرب بالعنصر الذي يقابله (وهذا ليس التعريف الرياضي لضرب المصفوفات) فلنأخذ المثال التالي:

```
>>a=ones(3)
>>b=rand(3)
>>c=10*a;
>>d=b * c
>>f=b .* c
```

لاحظ الفرق بين الايعازين الأخيرين، اذ ان الاول هو الضرب الرياضي لمصفوفتين وله شروط واجراءات بينما الثاني هو ببساطة ان يحصل الضرب بين كل عنصرين متقابلين.

لقد ذكرنا من بين المصفوفات المربعة الخاصة مصفوفة **ones(n)** ومصفوفة الـ **zeros(n)** ومصفوفة الـ **rand(n)** حيث ان الأولى تعطي مصفوفة مربعة كلها واحدات، والثانية تعطي مصفوفة مربعة كلها اصفار، والثالثة تعطي مصفوفة مربعة كلها ارقام عشوائية موجبة تقع بين الصفر والواحد.

والمصفوفة الخاصة الرابعة هي مصفوفة **eye(n)** وهذه المصفوفة تسمى بمصفوفة الوحدة أي Unit Matrix ويرمز لها عادة بالحرف I وسبب التسمية هذا ان هذه المصفوفة في حالة ضرب المصفوفات تقابل الرقم واحد في ضرب الاعداد؟ أي انها النظير الضربي في حالة المصفوفات أي عندما نضربها في أي مصفوفة او نعمل العكس أي نضرب أي مصفوفة بها فأن الناتج سوف يعطي نفس المصفوفة الاصلية أي:

A I = A (عبارة رياضية وليست برمجية، لماذا؟)

I A = A (عبارة رياضية وليست برمجية، لماذا؟)

```
>>eye(4)

[ 1   0   0   0
  0   1   0   0
  0   0   1   0
  0   0   0   1]
```

برمجة الماتلاب MATLAB Programming

في الجزء الأول كنا قد استخدمنا برنامج الماتلاب بالطريقة التفاعلية اي ان المستخدم يكتب الايعاز المقصود والبرنامج مباشرة ينفذ ذلك الايعاز ويخرج النتائج في نفس النافذة، هذه الطريقة مفيدة مع المهام البسيطة والقصيرة اما اذا كانت المهام من النوع الاكثر تعقيداً او غير المباشرة فربما نحتاج الى اجراء العديد من التوجيهات والايعازات وربما اتخاذ بعض القرارات في سلسلة مستمرة من الخطوات للوصول الى النتائج المرجوة، ان هذه العملية هي ما تسمى بالبرمجة اي كتابة برنامج متكون من مجموعة ارشادات وتوجيهات وايعازات وحسابات مترابطة فيما بينها وبتسلسل منطقي بحيث تؤدي في مجموعها الى حل مسألة معينة واخراج النتائج المطلوبة.

في برنامج الماتلاب تكون كتابة البرنامج في ملف خارجي يكون نوع هذا الملف بامتداد m. ويمكن انشاء هذا الملف النصي اما باستخدام برنامج الماتلاب نفسه، او باستخدام اي برنامج محرر نصوص مثل برنامج الـ Notepad الذي يأتي مع الـ Windows او غيره لكن يجب تغيير امتداد الملف من txt. او dat. الى الامتداد m. لكي يفهمه الـ MATLAB، (يذكر ان الامتدادات **.dat** & **.txt** هي الامتدادات الاشهر للملفات النصية البسيطة والتي ليس فيها تنسيقات عالية، اما الملفات النصية ذات التنسيق العالي فاشهرها هو doc.) ثم يجب ان نتأكد من المجلد الذي سوف نضع فيه هذا الملف بحيث يكون هو المجلد الحالي اثناء التنفيذ حيث ان كل الملفات الوسطية وملفات الادخال والاخراج ان وجدت يجب أن تكون او تنشأ داخل نفس هذا المجلد وهو المجلد الحالي او المجلد الفعال.

اما الطريقة المباشرة والاسهل هي ان ننشئ ملف البرنامج من داخل برنامج الـ MATLAB نفسه، وذلك بالذهاب الى شريط الادوات والضغط على مفتاح New Script فتفتح لنا صفحة بيضاء نكتب فيها البرنامج المطلوب وهنا على سبيل المثال سوف نكتب برنامج قصير جداً لحساب مساحة الدائرة فيكون مثلاً بالشكل التالي:

```
R=5;

A=R^2*pi;

disp(A)
```

نقوم بحفظ هذا البرنامج او الملف عن طريق المفتاح Save في شريط الادوات والذي سيفتح نافذة تبين المكان الذي سيحفظ فيه الملف وكذلك تعطي خيار لتسمية هذا الملف ونحن سوف نسميه Circle، ولأننا استخدمنا الـ MATLAB نفسه في انشاءه فسوف يعطيه الامتداد الافتراضي وهو Circle.m.

الان لكي ننفذ هذا البرنامج ونحصل على النتائج ننقر على المفتاح Run ويكون مصاحب للشكل ▶ في شريط الادوات، فنحصل على النتيجة بشكل مباشر على الشاشة وفي نافذة الـ Command Window.

```
>> Circle

  78.5398
```

كيف نكتب البرنامج

بعض المبرمجين يقومون بكتابة البرنامج مباشرة في سلسلة من المحاولة والخطأ ثم معالجة الخطأ وهكذا الى ان يصل الى البرنامج الصحيح المطلوب، ولكن هذه الطريقة تنفع مع البرامج البسيطة التي ربما تتكون من صفحة واحدة او صفحتين، اما اذا كان البرنامج المطلوب احترافي ربما يتكون من عشرة صفحات او اكثر فأنه لا يمكن البدء به مباشرة في المحاولة والخطأ بل يجب على المبرمج ان يضع لنفسه مخططاً للخطوات اللازمة وتجزئة المهام التي سوف يقوم بها البرنامج وهذا المخطط او الخطة التي غالباً ما تكون على شكل رسم مخطط سهمي للتسلسل المنطقي للمهام هو ما يسمى بالخوارزمية Algorithm.

الخوارزمية Algorithm

ولتوضيح هذا المفهوم يمكننا اقتراح الخوارزمية اللازمة لحساب مساحة الدائرة:

1- تحديد البيانات المطلوبة كمدخلات للبرنامج. وهنا مثلاً تعني الحصول على قيمة نصف قطر الدائرة. ويصاحب هذه الخطوة سؤال: من اين سيتم ادخال نصف القطر هل هو موجود في ملف خارجي ام سيكون ادخاله من لوحة المفاتيح مباشرة.
2- اخذ قيمة نصف القطر، واسنادها الى متغير معين داخل البرنامج، وهذا غالباً سيتضمن:
   a. ابلاغ المستخدم بضرورة ادخال قيمة نصف القطر بالوحدات المعينة. (ويكون ذلك بطباعة عبارة نصية تبلغ المستخدم بالحاجة الى ادخال نصف القطر)
   b. اسناد قيمة نصف القطر المدخلة الى متغير داخل البرنامج (وتكون هذه بعبارة اسناد او مساواة مثلاً R=5).
3- استخدام قيمة نصف القطر المدخلة من الخطوة السابقة في معادلة لحساب المساحة. وهذا سيحتاج الى قيمة النسبة الثابتة التي اما ان يكون البرنامج مجهز بها او انها تدخل على شكل رقم مباشر مع الدخلات.
4- تحديد المكان الذي سوف نطبع فيه النتائج وهو هنا اما ان يكون الى ملف خارجي او يكون مباشرة الى الشاشة.
5- طباعة النتيجة لمساحة الدائرة، ولكن يفضل ان لا تكون النتيجة فقط رقم مجرد، بل يفضل ان تكتب عبارة واضحة تبين ان الناتج هو مساحة الدائرة وبالابعاد المعينة مثلاً:

   The area of a circle with r=5cm is A= 78.54 cm2

ان الجانب السيء في هذه الخوارزمية انها تبدو معقدة ومفصلة رغم ان المهمة الاصلية سهلة جداً. اما الجانب الجيد فهو ان البرامج الكبيرة مهما تعقدت فهي بشكل عام سوف تتبع نفس هذه الخطوات الخمسة او هذه الخوارزمية، ولهذا السبب فأن كتابة الخوارزميات يكون مناسباً جداً للبرامج الكبيرة بينما هي ليست بأهمية في البرامج الصغيرة لأن البرامج الصغيرة يمكن كتابتها مباشرة بطريقة المحاولة والخطأ.

لاحظ ان كل من البرامج الصغيرة والكبيرة تشترك في خطوات عامة رئيسية ثلاثة، ولكنها طبعاً سوف تختلف في التفاصيل وهذه الخطوات الثلاث هي:

1- ادخال المدخلات.
2- اجراء الحسابات على المدخلات.
3- اخراج النتائج.

كتابة البرنامج في الماتلاب:

وتكون بالذهاب الى File→new→script وهنا سوف يفتح محرر Editor يمكنك من كتابة البرنامج ولاحظ ان السطور سوف ترقم بالتسلسل اثناء الكتابة.

بعد اكمال كتابة البرنامج او حتى في منتصفه يفضل عمل حفظ للملف عن طريق ايعاز Save الموجود في شريط الادوات، ولاحظ ان الملف يجب ان يحفظ بامتداد m. ولاحظ ايضاً ان الاسم الذي تعطيه للملف يجب ان يبدأ بحرف وليس رقم.

يفضل عندما تحفظ الملف لأول مرة ان تنشأ له مجلد معروف في مسار معروف ويفضل في البداية ان تنشأ ذلك المجلد على أحد السواقات مباشرة لكي يكون الوصول اليه سهلا جداً.

بعد ان صار لديك المجلد الذي حفظت به ملف البرنامج عليك الان ان تجعله المجلد الحالي او المجلد الفعال، وهذه العملية سهلة جداً بمجرد الذهاب الى النافذة في اليسار التي هي Current Folder ثم باستعراض المجلدات Browsing حتى تصل الى مجلدك المطلوب.

الان بعد ان أصبح مجلد البرنامج هو نفسه المجلد الفعال Current Folder يمكنك الان ان تتعامل مع البرنامج مباشرة من خلال نفس نافذة الايعازات Command Window مثلاً يمكنك ان تفتح (تستعرض) البرنامج باستخدام الايعاز type متبوعاً باسم البرنامج بدون الامتداد.

اما تنفيذ البرنامج فيكون بعدة اشكال اسهلها هو النقر على الأداة Run (▶) في شريط الأدوات وطريقة أخرى هي كتابة اسم البرنامج بدون الامتداد في الـ Command Window.

احد الأمور الجيدة ايضاً انك بعد ان تصل الى الملف وتفتحه ثم تنقر على زر التنفيذ، فحتى لو لم يكن مجلد البرنامج هو المجلد الحالي، فأن البرنامج سوف يسألك (هل تريد ان اجعل هذا المجلد هو المجلد الفعال؟) وبالاجابة عن هذا السؤال بنعم، فأن البرنامج سوف ينفذ وفي نفس الوقت سوف يكون مجلد البرنامج هو المجلد الحالي لكي تتمكن من مشاهدة ملفات الإخراج او أي شيء اخر امامك مباشرة.

لاحظ عند التنفيذ ان البرنامج كله سوف ينفذ دفعة واحدة، ولاحظ ان كل سطر في البرنامج لا ينتهي بالفارزة المنقوطة سوف تظهر نتيجته عند التنفيذ اما السطور التي تنتهي بالفارزة المنقوطة فأنها سوف تنفذ ولكن لن تخرج نتائجها الوسطية مع المخرجات، ونستفيد من هذه الخاصية في تحديد الأشياء التي نريد رؤيتها فعلاً والاشياء التي لا نحتاج الى رؤيتها من المخرجات.

لاحظ ان الايعاز Type في الـ Command Window وظيفته فقط استعراض البرنامج لمشاهدته ولن يسمح لك بالتحرير او التعديل عليه وانت غالباً سوف تحتاج الى عملية التحرير هذه خصوصاً إذا كان البرنامج يحتوي على بعض الأخطاء. لذلك فهناك طريقتين لفتح البرنامج ولكن هذه المرة في بيئة التحرير وليس بيئة المعاينة، ويكون ذلك:

اما بالذهاب الى File→open→the file او باستخدام نافذة الـ Current Folder الى اليسار حيث سوف تشاهد الملف امامك وما عليك الا النقر عليه مرتين ليفتح امامك للتحرير.

التعليق داخل البرنامج Commenting:

يفضل عندما تكتب أي برنامج ان تضيف بعض التعليقات التي توضح ما تقوم به بعض الايعازات (على كل حال ليس من المتوقع ان تكتب تعليق لكل سطر في البرنامج!)، ويفضل كتابة التعليقات أما لجعله سهل الاستخدام من قبل الغير او حتى يسهل عليك انت استخدامه بعد فترة من الزمن. وفي الماتلاب تكون التعليقات هي كل ما يأتي بعد رمز النسبة المئوية % الى نهاية السطر، علماً إنك يمكن ان تضع هذا الرمز % في أي مكان ولا يشترط ان تضعه فقط في بداية السطر.

لاحظ ان البرامج الاحترافية دائماً ما تبدأ بتعليق يكون في السطر الأول وغالباً ما يشرح هذا التعليق وظيفة هذا البرنامج، ويكتسب هذا التعليق بالذات أهمية خاصة في الماتلاب حيث انه هو الذي سوف يظهر عندما يكتب المستخدم ايعاز help متبوعا باسم البرنامج.

لبرنامج الدائرة سوف نضيف التعليق التالي في بداية البرنامج:

```
% This program calculates a Circle Area & Perimeter
R=5
A=pi*R^2
P=2*R*pi
```

لاحظ ان التعليق لن يظهر مع النتائج وانما هو فقط لانعاش ذاكرة المستخدم عن ماهية البرنامج.

تطبيق

اكتب البرنامج التالي واحفظه باسم معين ثم نفذه

```
x=(1:100)'; % Column Vector
for k=1:5
    y(:,k)=k*log(x);
end
plot(x,y)
```

البرنامج اعلاه والذي سنناقشه ونفهمه في المحاضرات اللاحقة، هو فقط ليبين كيف يمكن في الـ MATLAB بناء برنامج بسيط وقصير ليرسم خمس منحنيات لوغارثمية في آن واحد.

عبارات الادخال والإخراج Input & Output (I/O)

بالنسبة الى ادخال البيانات Data Input ففي معظم البرامج وكذلك الحال مع الماتلاب فهناك ثلاث طرق لادخال المعلومات وهي اما تضمين البيانات داخل البرنامج نفسه، وهذه عادة تستخدم للمتغيرات التي لن نحتاج الى تغيير قيمها كثيراً مثل الـ Parameters مثلاً نكتب R=5 داخل البرنامج نفسه، وعندما نريد ان ننفذ البرنامج لحالة ثانية ونغير هذه المدخلات فأننا يجب ان ندخل الى البرنامج نفسه مرة اخرى ونغير قيمة R مثلاً. الطريقة الثانية وهي الطريقة التفاعلية، وهنا المبرمج يوفر للمستخدم امكانية ان يدخل البيانات المطلوبة أثناء تنفيذ البرنامج عن طريق لوحة المفاتيح، وعادة يزرع المبرمج عبارة داخل البرنامج تخبر المستخدم بالبيانات التي يجب ان يدخلها الان مثلاً تظهر للمستخدم Input Radius of

Circle، وهذه الطريقة اكثر مرونة من الاولى لأنها تسمح للمستخدم ان يغير قيم المعطيات في كل مرة ينفذ فيها البرنامج وهي فعالة ايضاً عندما تكون المدخلات قليلة مثلاً خمس مدخلات، أما الطريقة الثالثة وهي ان يتم ادخال البيانات من خلال قراءة ملف خارجي موجود في مكان ما على أحد سواقات الحاسوب Disk drives وهذه هي الطريقة المتبعة في البرامج الاحترافية التي فيها عدد كبير من البيانات او المعطيات، مثلاً قراءة درجات الحرارة المسجلة خلال شهر او سنة كاملة، والشيء المهم هنا بالاضافة الى امكانية ان تكون المدخلات كثيرة جداً، هو اننا عندما نريد ان نغير المدخلات فان هذا لن يكون له علاقة بالبرنامج الاصلي وانما فقط نذهب الى ملف الادخال الخارجي ونقوم بالتغيير او نحضر او نستورد ملف ادخال جديد كلياً، لسنة سابقة مثلاً.

العبارة التفاعلية الأولى للإدخال لمثال الدائرة تأخذ الشكل:

```
% This program calculates a Circle Area & Perimeter
R=input("please inter value of R:")
A=pi*R^2
P=2*R*pi
```

هذا البرنامج فيه اضافة مهمة عن النسخة الاولى، لانه الان يعطي الحرية للمستخدم لادخال اي نصف قطر وسوف يحسب مساحة ومحيط الدائرة له، بينما البرنامج الاول كان فيه نصف القطر =5 مكتوبة داخل البرنامج نفسه.

طبعاً اذا لم نحدد نوع المتغير الذي سوف ندخله داخل هذه العبارة فأن الافتراض هو رقم حقيقي، فأذا ادخلنا كلمة مثلاً وليس رقم فسوف ينتج خطأ لذلك وفي هذه الحالة لبرنامج اخر غير برنامج الدائرة يجب ان يكون الايعاز بصورته الاعم وهي:

```
>> b=input('please inter value for b:','s')

        please inter value for b:        good morning
```

ان وجود الرمز 's' في الايعاز السابق يعني متغير نصي، أي ان الايعاز كله سوف يقرأ بالشكل التالي: يرجى ادخال قيمة معينة للمتغير b وهذه القيمة يجب ان تكون متغير نصي، أي ان البرنامج يتوقع من المستخدم ان يدخل قيمة نصية عبارة عن حرف واحد او مجموعة حروف ولا ينتظر منه ان يدخل رقم حقيقي او رقم صحيح واذا فعل ذلك ففي هذه الحالة بالذات فأن البرنامج سوف يعتبر تلك الأرقام رموز وليست ارقام أي لن يستطيع جمعها او طرحها.

ان عبارة input هذه أكثر استخدامها عندما نحتاج الى ادخال متغير واحد، اما ما نضعه داخل الاقواس فهو فقط رسالة لابلاغ المستخدم بالإدخال، ولكنها مطلوبة في هذا الايعاز. وكل ما سيتم إدخاله من قبل المستخدم سوف يسند الى المتغير الموجود في الطرف الايسر من علامة الاسناد (المساواة).

ملاحظة:

عندما تبحث في الـ Help داخل البرنامج وتسأله عن عبارة Input سوف يظهر لك الشرح التالي:

input

Request user input

R2024a

collapse all in page

Syntax

```
x = input(prompt)
txt = input(prompt,"s")
```

Description

example

`x = input(prompt)` displays the text in `prompt` and waits for the user to input a value and press the **Return** key. The user can enter expressions, like `pi/4` or `rand(3)`, and can use variables in the workspace.

- If the user presses the **Return** key without entering anything, then `input` returns an empty matrix.
- If the user enters an invalid expression at the prompt, then MATLAB® displays the relevant error message, and then redisplays the prompt.

example

`txt = input(prompt,"s")` returns the entered text, without evaluating the input as an expression.

Examples

collapse all

Request Numeric Input or Expression

Request a numeric input, and then multiply the input by 10.

```
prompt = "What is the original value? ";
x = input(prompt)
y = x*10
```

لاحظ ان البرنامج سوف يعطيك شرح كامل مع الامثلة عن كل ايعاز تسأله عنه.

وكلمة Prompt هنا تعني: "العبارة الارشادية التي يضعها المبرمج للمستخدم لترشده الى الشيء الذي يتوجب عليه عمله الان" مثلاً تظهر له عبارة Please, input the radius of the Circle?

اما **عبارات الإخراج** فأبرزها هي عبارة disp وعبارة fprintf وكل واحدة تكون مناسبة لحالات معينة.

لاحظ إنك اذا تركت أي عبارة تنفيذية في البرنامج دون ان تنهي السطر بالفارزة المنقوطة، فأن البرنامج سوف يظهر نتيجة ذلك السطر مع المخرجات، لذلك يمكننا ان نعتبر هذه أول طريقة بدائية من طرق الإخراج.

اما استخدام الـ disp فاليك المثال التالي:

```
>>disp('Hello')
        Hello
>>disp(4^3)
        64
```

عبارة الإخراج **disp** محدودة لانها قادرة على اخراج متغير واحد وبدون اي تنسيق، وهذا يجعلها تشبه عبارة **input** التي تقدر على ادخال قيمة واحدة فقط.

اما إذا أردنا مع الإخراج ان نعمل بعض التنسيق Formatting أو اردنا اخراج مجموعة من المخرجات دفعة واحدة، فعلينا ان نستخدم عبارة **fprintf** التي تعمل بشكل مختلف يعطي تحكماً أكبر في كيفية كتابة المخرجات، ومع هذه العبارة هناك ما يعرف بالـ place holder وهو كالأتي:

%d	تستخدم لحجز مكان للأعداد الصحيحة integer
%f	تستخدم لحجز مكان للأعداد الحقيقية float
%c	تستخدم لحجز مكان للرمز المنفرد single character
%s	تستخدم لحجز مكان لكتابة مجموعة من الرموز في متغير نصي string

واليك المثال التالي الذي يوضح هذه العبارة:

```
>>fprintf('the value is %d meters.\n',4^3)

     the value is 64 meters.
```

اي انك تكتب الايعاز fprintf ثم تفتح اقواس، وداخل الاقواس تكتب عبارة نصية واحدة، يتخلل هذا النص حجز للمتغيرات التي تريد قيمها ان تظهر كنتائج، ثم تأتي الفوارز، والتي بينها تضع اسماء المتغيرات الفعلية التي تريدها ان تظهر مع النتائج.

وبتطبيق هذا على مثال الدائرة:

```
% This program calculates a Circle Area & Perimeter
R=input("please give value of R:")
A=pi*R^2;
P=2*R*pi;
% First Print
fprintf("for circle with radius %f\n",R)
% Second Print
fprintf("the area is %f, and the perimeter is %f",A,P)
```

ما سيظهر عند ضغط المفتاح ▶ Run لهذا البرنامج هو:

```
please give value of R:5
R=
5
for circle with radius 5.000000
the area is 78.539816, and the perimeter is 31.41592
```

لاحظ ان الرمز %f يقوم بأمرين هنا، الأول هو انه يحجز المكان الذي ستطبع فيه النتيجة، وهي هنا في وسط الكلام، والثاني انه يبلغ عن نوع او طبيعة تنسيق النتيجة، وهو هنا اذا كان d سيظهر الرقم كعدد صحيح واذا كان f سيظهره كعدد حقيقي وهكذا.

أما الرمز الأخير (\n) في عبارة الطباعة الاولى هو ايضاً رمز خاص وهو يعني هنا بعد ان تطبع هذا السطر افتح سطر جديد، new line وضرورة وجوده هنا لكي لايجعل ما نطبعه في السطور المختلفة يكون ملتصقاً مع بعضه في سطر واحد، أي اننا عندما ننهي بعبارة افتح سطر جديد، فأن أي طباعة لاحقة في البرنامج سوف تظهر في سطر جديد. لاحظ ان الايعاز السابق disp لم يكن يحتاج الى ذلك وانما هو يفتح سطر جديد تلقائياً.

ويمكننا هذا الرمز ايضاً من ان نفتح سطر فارغ في المخرجات وما علينا سوى ان نكرر هذا الرمز في عبارة الـ fprintf.

طبعاً علينا هنا ان لا نخلط بين علامة الـ % الموجودة داخل ايعاز fprintf وبين نفس هذه العلامة التي تستخدم للتعليقات.

ان عبارة fprintf يمكنها ان تطبع اكثر من متغير واحد، ولكي تقوم بهذا الامر فعلينا ان نضع الـ Place Holder أي الـ (%d, %f , %c , %s) اكثر من مرة داخل العبارة، وكذلك في النهاية يجب ان نضع عدد مساو لها من المتغيرات على الترتيب، واليك المثال:

```
>>fprintf('the integer is %d, and the string is %s\n',7-3,'GOOD')

    the integer is 4, and the string is GOOD
```

عبارات التحكم والعبارة الشرطية If statement

من اقوى العبارات التي تجعل الحاسوب يبدو وكأنه يفكر مثل الانسان ويأخذ القرارات هي العبارة الشرطية، لان هذه العبارة عندما تكتب داخل برنامج معين فانها تقوم باختبار او اجراء عملية منطقية يكون جوابها اما "نعم" او "لا" وبالاعتماد على هذه الاجابة فيمكن للبرنامج ان يغير مساره ويقوم بمهام محددة.

عندما يجد البرنامج عبارة if فانه سينظر في الشرط المرافق لها، فاذا تحقق الشرط فان البرنامج سينفذ كل العبارات الواقعة بعدها حتى يصل الى نهاية الـ block وسوف يعرف النهاية لأن الـ block الواحد يجب ان ينتهي بعبارة end اما اذا لم يتحقق الشرط فان البرنامج سوف يقفز ويهمل كل ذلك الـ block ويبدأ بتنفيذ العبارات التي تأتي بعد عبارة end.

```
a = 2;
b = 3;
c = 1;

if a < b
    c = 0;
end

disp(c);
```

Output:

```
0
```

هناك ايضاً **if …else** وهناك **if ….elseif….else** وهي فقط تفرعات من العبارة الاصلية if وتستخدم عندما يكون هناك اكثر من شرط واحد وتحقق اي شرط من تلك الشروط سوف يؤدي الى تنفيذ فقط الـ block المرفق به ثم يقفز الى النهاية لما بعد عبارة end.

```
a = 4;
b = 7;
if a < b
    disp(a);
else
    disp(b);
end
```

Output:

```
4
```

تمرين: اكتب برنامج يقرأ معدل الطالب ثم بالاعتماد على المعدل سيخرج بنتيجة واحدة فقط وهي اما ان تكون (Excellent-Very Good, Good, Intermediate, Fair, Fail)

CONSTRUCTION: Extended If-Else Statement Syntax

```
if logical expression P
    code block 1
elseif logical expression Q
    code block 2
elseif logical expression R
    code block 3
else
    code block 4
end
```

EXAMPLE: What will be the value of y after the following script is executed?

```
x = 3;
if x > 1
    y = 2;
elseif x > 2
    y = 4;
else
    y = 0;
end
```

```
>> y
y =
     2
```

EXAMPLE: What will be the value of y after the following code is executed?

```
x = 3;
if x > 1 && x < 2
    y = 2;
elseif x > 2 && x < 4
    y = 4;
else
    y = 0;
end
```

```
>> y
y =
     4
```

الدوارات او الحلقات Loops

من اكثر ما يميز الحاسوب عن الانسان هو تكرار العمل او الحسابات لعدد هائل من المرات دون ملل او كلل، لذلك تلعب الدوارات Loops دوراً مهماً في كل لغات البرمجة. وهذه القدرة مع القدرة على اتخاذ القرار هي احد اهم اسرار الأتمتة Automation التي يمتاز بها الكومبيوتر.

هناك عبارتان مهمتان للدوارات او الحلقات في الماتلاب وهي for و while وتستخدم الاولى for غالباً عندما نكون عارفين مسبقاً عدد التكرار الذي نريده في الحلقة الواحدة، بينما تستخدم الـ while غالباً عندما لا نعرف عدد التكرار المطلوب.

```
for i = 0:4
    disp(i);
end
```

```
Output:

0 1 2 3 4
```

البرنامج السابق يقرأ كالاتي: مؤشر الحلقة هو i وهذه الـ i ستتغير قيمتها من 0 الى 4 بزيادة قيمتها وحدة واحدة، وفي كل مرة تتغير هذه القيمة على البرنامج تنفيذ كل عبارات الـ block الذي ينتهي بالعبارة end لذلك فان عملية الطباعة disp سيمر عليها البرنامج وينفذها خمس مرات.

```
i = 0;

while i < 5
    fprintf('i = %d\n', i);  % Print the value of i.
    i = i + 1;               % Increment i by 1.
end
```

```
Output:

i = 0
i = 1
i = 2
i = 3
i = 4
```

بينما في المثال اعلاه جعلنا حلقة الـ while تقوم بنفس العمل، ولكن علينا الانتباه الى الاختلاف بين الحالتين، ففي حالة الـ while نحن نضع معها فقط شرط condition وهو هنا i<5 لذلك فان الـ i يجب ان تكون قيمتها معروفة قبل الوصول الى هذه العبارة، والامر الاخر ان هذه الـ i يجب ان يعاد حسابها "داخل الحلقة نفسها" لكي لا تبقى هذه الحلقة تعيد نفسها الى المالانهاية، وانما يجب في احد التكرارات ان تتغير قيمة i بحيث لا تعود تحقق الشرط الموجود مع while وهنا فقط سينتهي التكرار وسيخرج البرنامج من الحلقة ويذهب الى ما وراء العبارة end.

```
for i = 10:-1:6
    disp(i)
end
```

```
Output:

10 9 8 7 6
```

في المثال التالي هناك ملاحظة يختلف فيها الماتلاب عن معظم لغات البرمجة

```
for i = 10:-2:6
    i = i - 1;    % Decrement i by one more.
    disp(i);      % Display the value of i.
end
```

```
Output:

9 7 5
```

نلاحظ هنا ان قيمة i وهو مؤشر الحلقة الذي استخدم مع for قد حصلت عليه عمليات حسابية وتغيرت قيمته داخل الحلقة نفسها، وهذا كان غير ممكن في معظم لغات البرمجة، ولكنه هنا مع الماتلاب مقبول ولن يؤثر على عدد حلقات التكرار التي قررناها منذ البداية وهي في هذه الحالة ثلاث دورات. وهذه المسألة مربكة وهي تعني ان مؤشر الحلقة بعد ان يقوم بدوره الرئيسي هو تحديد عدد الدورات، سوف لن يتأثر فيما بعد بالعمليات الحسابية التي تجرى عليه داخل الحلقة!

```
a = 5;

for i = 0:(a-1)
    disp(a - i);
end
```

```
Output:

5 4 3 2 1
```

المثال اعلاه هو لتتبع ماذا سيحصل لقيمة i وقيمة a عند تكرار الحلقة ومعرفة ما الذي سيطبعه البرنامج.

```
a = 0;

for i = 1:5
    a = a + (i + 4 * 3);
end

disp(a);
```

```
Output:

75
```

المثال اعلاه يحتاج الى تتبع دقيق لما سيحصل لقيمة المتغير a خصوصاً وان عبارة الطباعة موجودة خارج الحلقة وليس داخلها، لذلك فان قيمة a سيحصل لها تجميع او تراكم وستحصل على قيمة جديدة لكل تكرار ثم في الاخير يتم طباعة النتيجة الأخيرة لها مرة واحدة خارج الحلقة اي بعد ان تكتمل كل التكرارات.

تمرين: جد ناتج جمع الاعداد الزوجية المحصورة بين 1 والـ 100

```
r = 0;
for i = 1:10
    for j = 1:10
        r = r + 1;
    end
end

disp(r);
```

```
Output:

100
```

المثال اعلاه يستخدم الحلقات المتداخلة، اي توجد حلقة خارجية تتغير فيها قيمة i من 1 الى 10 اي تتكرر هذه الحلقة 10 مرات، ثم توجد حلقة الـ j داخل الحلقة الرئسية وهذه الحلقة الداخلية سوف تتكرر 10 مرات لكل حلقة واحدة خارجية ولذلك فان عدد التكرار في هذا المثال والذي ستعدل فيه قيمة r سيكون 100 مرة. وهذه العملية تشبه ما نراه في الساعة الالكترونية التي تظهر فيها الدقائق والثواني، فلو كنا في الدقيقة 4 مثلاً، فان الثواني سوف تتغير من 1 الى 60 ثانية وعندها ستتغير الدقيقة من 4 الى 5 ثم في الدقيقة 5 سوف تتغير الثواني مجددا من 1 الى 60 لكي تتغير هي مرة واحدة وهكذا.

```
for i = 0:1
    for j = 0:2
        fprintf('i = %d, j = %d\n', i, j);
    end
end
```

Output:

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
i = 1, j = 1
i = 1, j = 2
```

في المثال اعلاه لاحظ ان عبارة الطباعة موجودة داخل الحلقة وليس خارجها اي ان البرنامج سوف يمر عليها وينفذها عدة مرات لذلك عليك تتبع ماذا سيحصل لقيمة كل من i و j في كل مرة تنفذ فيها عبارة الطباعة.

برمجة المصفوفات والجداول Arrays Programming

تناولنا العديد من العمليات على المصفوفات في الجزء الاول والان جاء دور البرمجة مع المصفوفات

```
% a(i) vector
% a(i,j) = matrix
% a(i,j,x) = tensor
% a(i,j,x,y,.......

% Define vectors.
a = [2, 5, 7];
b = [6, 8, 9];

% Concatenation.
c = [a, b];
disp(c);

% Addition.
disp(a+b);
```

Output:

```
2,5,7,6,8,9
8,13,16
```

في المثال اعلاه، العبارات الاولى المسبوقة بعلامة % هي فقط تعليقات توضيحية لا ينفذها البرنامج، ثم يأتي بعدها تعريف متجهين صفيين واعطاءهما القيم المحددة. العبارة التي بعدها هي ضم المتجين الاصليين الى بعضهما في متجه واحد اكبر. والعبارة الاخيرة هي جمع المتجهين كما موضح ذلك في النتائج Output. لاحظ عند الجمع او الطرح يجب ان تكون ابعاد المصفوفتين متساوية.

عند مناداة عنصر واحد او مجموعة عناصر من المصفوفة فيجب استخدام الاقواس الصغيرة بينما التعبير عن المصفوفة كلها او تعريفها يكون بالاقواس الكبيرة المربعة.

```
a = [2, 5, 7]; % Creating array a.
b = [6, 8];    % Creating array b.

% In MATLAB, array indices start from 1.
c = a(2) + b(1);

disp(c);
```

```
Output:

11
```

في المثال اعلاه بعد تعريف المصفوفات فان عبارة الجمع تنادي عنصر واحد فقط من المصفوفة الاولى وهو "العنصر الثاني" وتجمعه مع "العنصر الاول" من المصفوفة الثانية، فيكون ناتج الجمع عبارة عن رقم Scalar وهو 11.

```
A = ["a", "b", "c"];

x = A(2);
y = A(3);

disp([x, y]);
```

```
Output:

bc
```

المثال اعلاه يوضح امكانية ان تكون عناصر المصفوفة رموز strings وليس ارقام مع امكانية التعامل معها.

```
a = [2, 5, 7];
b = [6, 8];

% Decrementing the
% second element of a.
a(2) = a(2) - 1;

% Decrementing the
% first element of b.
b(1) = b(1) - 1;

% Sum of the
% updated elements.
c = a(2) + b(1);

disp(c);
```

```
Output:

9
```

يوضح المثال امكانية تغيير عناصر محددة داخل المصفوفة ثم التعامل معها.

```
a = [5, 6, 8];
b = length(a); % Length of the array.

disp(b); % Displaying the length.
```

```
Output:

3
```

يبين المثال اعلاه عمل الدالة length() وهي تعطي طول المتجه او عدد عناصره.

```
A = [1, 2, 3];

if A(1) < A(2)
    A(3) = A(3) + 1;
end

disp(['A[3]=', num2str(A(3))]);
```

```
Output:

A[3]=4
```

في المثال اعلاه بالاضافة الى العبارة الشرطية فانه يوضح امكانية الامر disp في ان يظهر اكثر من جزء واحد بشرط ان تكون المدخلات رموز وليس ارقام. وكذلك يبين الايعاز الجديد num2str() ووظيفته هو تحويل العدد الى رمز، اي بعد ان يحول العدد 235 مثلاً الى الرمز 235 فان هذا الاخير لم يعد عدداً اي لا يمكننا جمعه او طرحه لأنه اصبح رمزاً فقط وليس عدد.

```
a = {'x', 'y', 2};  % mixed cell array.

for i = 1:length(a)
    disp(a{i});
end
```

```
Output:

x
y
2
```

في المثال اعلاه شكل جديد من اشكال المصفوفات يعرف بالخلية Cell وهو يستخدم الاقواس المزخرفة { } وهي تختلف عن المصفوفة فقط في انها يمكن ان تحوي على عناصر غير متجانسة من حيث النوع، اي مثلاً تكون بعض عناصرها ارقام وعناصر اخرى رموز او كلمات وهكذا.

```
a = [5, 6, 8];

for j = 1:length(a)
    disp(a(j));
end
```

```
Output:

5
6
8
```

```
a = [5, 6, 8];
b = 0;

for j = 1:length(a)
    b = b + a(j);
end

disp(b);
```

```
Output:

19
```

المثال اعلاه فيه معالجة مشهورة نقوم بها عندما نريد ان نجمع عدد كبير من الارقام، وهي اننا نضع متغير معين مثل b خارج الحلقة ونضع له قيمة ابتدائية صفر، ثم نعمل على تراكم هذه القيمة داخل الحلقة مع كل عناصر المصفوفة او المتجه واحداً واحد، ثم نطلب طباعة هذا المتغير بعد ان ننتهي من الحلقة كلها. فيكون الناتج هو مجموع عناصر ذلك المتجه او المصفوفة.

```
a = [5, 6, 8];

for j = 1:length(a)
    a(j) = 2 * a(j); % Doubling each element of a.
end

disp(a);
```

```
Output:

10,12,16
```

في المثال اعلاه نقوم بتغيير كل عناصر المتجه لينتج لنا متجه جديد والتغيير هنا هو اننا ضاعفنا كل العناصر.

```
% Pre-allocating an array with 11 elements.
a = zeros(1, 11);

for j = 1:11
    a(j) = j - 1;
end

disp(a);
```

```
Output:

0,1,2,3,4,5,6,7,8,9,10
```

المثال اعلاه بدأ بالمصفوفة الصفرية وهي من صف واحد مع 11 عمود وكل عناصرها اصفار، ثم دخلنا الى حلقة وفي هذه الحلقة قمنا بتعديل تلك العناصر بعملية الطرح.

```
a = [2, 3, 4, 5, 9, 8, 3];
l = length(a)

maxV = 0;

for k = 1:l
    if a(k) > maxV
        maxV = a(k);
    end
end

disp(maxV);
```

```
Output:

9
```

المثال اعلاه يحسب القيمة الاكبر من بين مجموعة ارقام وذلك بحفظها اولاً داخل متجه ثم وضع متغير الحفظ بقيمة افتراضية صفر، ثم بعد ذلك عمل حلقة لقراءة كل عناصر المتجه وفي كل مرة فحص ان كان هذا العنصر اكبر من القيمة لمتغير الحفظ واستبداله ان تحقق الشرط وهكذا سيعمل متغير الحفظ على حفظ اكبر قيمة داخل المتجه عندما تكتمل الحلقة.

```
a = [3, 3, 4, 2, 9, 8, 3];
l = length(a);

min = a(1);

for k = 1:l
    if a(k) < min
        min = a(k);
    end
end

disp(min);
```

```
Output:

2
```

هذا المثال يشبه المثال السابق وهو يحسب اصغر قيمة في متجه وهو هذه المرة يقوم بشيء افضل من المثال السابق وهو انه يجعل القيمة الابتدائية لمتغير الحفظ هي اول قيمة في المتجه، وهذا افضل لانه لو

اختار الصفر مثلاً فربما لن يجد عنصر في المتجه اصغر من الصفر لذلك سيعطي ناتج خاطئ، وهذا الامر ينطبق نفسه تماماً على المثال السابق فوضع القيمة الاولى في المتجه هي القيمة الابتدائية لمتغير الحفظ يكون افضل.

```
a = [5, 6, 2, 9, 44, 200];
n = length(a);

b = 0;
e = 0;

for j = 1:n
    b = b + a(j);
end

x = b / n;

for j = 1:n
    e = e + (a(j) - x)^2;
end

s = sqrt(e / (n - 1));
c = s / x;

fprintf('AV = %.2f\n', x);
fprintf('SD = %.2f\n', s);
fprintf('CV = %.2f\n', c);
```

```
Output:

AV = 44.33
SD = 77.83
CV = 1.76
```

```
% Or, a second version:

a = [5, 6, 2, 9, 44, 200];

x = mean(a);
s = std(a);
c = s / x;

fprintf('AV = %.2f\n', x);
fprintf('SD = %.2f\n', s);
fprintf('CV = %.2f\n', c);
```

المثال اعلاه هو لأشهر ثلاث دوال في الاحصاء وهي المعدل Average والانحراف المعياري Standard Deviation والتغير Variation وهو محلول بطريقتين، الطريقة الاولى وهي التفصيلية بالدخول الى كل عناصر المتجه وعمل حلقات ومعادلات، والطريقة الثانية هي استخدام مباشر للدوال الموجودة اصلاً في برنامج الماتلاب.

في الحقيقة ان هذا المثال هو احد الامثلة التي تبين الطفرة النوعية التي حصلت في لغات البرمجة ومن بينها برنامج الـ MATLAB حيث سابقاً كان العمل ان المستخدم هو من يقوم بكتابة البرنامج المطول والشفرة التفصيلية للوصول الى نتيجة معينة، بينما في النسخ الحديثة فان معظم لغات البرمجة تم تسليحها بدوال جاهزة هي التي تقوم بذلك العمل بالخفاء، وتعطي النتيجة مباشرة.

ايعازات التنقل بين المصفوفات والمتجهات او تغير شكلها

تحويل مصفوفة الى مجموعة متجهات:

Time=timetemp(1,:);

Temp=timetemp(2,:);

واضح انه كانت لدينا مصفوفة من صفين، وفي الايعازين السابقين فصلناها الى متجهين صفيين.

اما تحويل المصفوفة بكل عناصرها الى شكل اخر بأبعاد أخرى أي اما ان تبقى مستطيلة ولكن بأبعاد مختلفة او حتى الحالة الخاصة بان تصبح متجه صفي او عمودي فإيعاز **reshape** هو الذي يقوم بهذا الامر، ولكن الأصل في الايعاز انه يعمل Column wise سواء على المصفوفة الأولى او حتى على المصفوفة الجديدة، أي انه يقوم بالقراءة من المصفوفة الأولى عمود عمود، ثم يرتب هذه العناصر نفسها في المصفوفة الجديدة أيضاً عمود عمود:

B=reshape(A,m,n);

ويحصل خطأ اذا كانت المصفوفة الاصلية A لا تحتوي حقاً على عناصر مقدارها mxn.

يمكنك في هذا الايعاز ان تضع الـ Place holder [] في احد الأماكن فقط بحيث ان البرنامج هو الذي يحسب قيمته، شرط ان تكون المصفوفة A فعلاً قادرة على ان تقسم نفسها بالمقدار الذي طلبته: مثلاً اذا كانت لدينا المصفوفة A(3x4) فيمكننا كتابة:

B=reshape(A,2,[])

والبرنامج تلقائياً سوف يحولها الى صفين وستة أعمدة، أي انه هو سيحسب الستة.

اما الايعاز **vec2mat**(V,4) فهو بالعكس سوف يحول المتجه ذو البعد الواحد الى مصفوفة حسب عدد الاعمدة المطلوبة وهنا المثال هو 4 اما اذا لم تكفي العناصر لملأ كل المصفوفة فأنه سوف يضيف اصفار في نهاية المصفوفة لكي تكتمل أربعة أعمدة كاملة بالأرقام. وهنا هو يقوم بالعكس تماماً أي انه يقرأ المتجه، ثم عندما يوزعه الى مصفوفة جديدة فأنه يملأ الخانات صفاً صفاً وليس عموداً عموداً.

C=vec2mat(V,5);

ويمكننا التغلب على مشكلة الـ Column wise الموجودة في الماتلاب بسهولة وذلك بان نعمل للمصفوفة الـ Transpose.

العمليات الرياضية على المصفوفات

```
% how many 1's in matrix.

a = [
    1, 1, 0, 0, 0;
    0, 1, 0, 0, 1;
    1, 0, 0, 1, 1;
    0, 0, 0, 0, 0;
    1, 0, 1, 0, 1
    ];

n = size(a, 1);
m = size(a, 2);
k = 0;

for i = 1:n
    for j = 1:m
        if a(i, j) == 1
            k = k + 1;
        end
    end
end

fprintf('k = %d\n', k);
```

Output:

```
k = 10
```

البرنامج اعلاه يعمل حلقتين متداخلتين، الحلقة الخارجية تقوم بالمرور على الصفوف صفاً صفاً، ثم الحلقة الداخلية تقوم بالمرور على العناصر عمود عمود، وبالتالي تتم القراءة داخل الحلقة الداخلية، ثم تعمل الدالة الشرطية if على فحص هل ان الرقم مساو الى 1 فاذا كان الجواب نعم تقوم بمراكمة الجمع، واذا كان الجواب لا فانها تقفز وتعيد الحلقة مرة اخرى لتبحث عن عنصر اخر وهكذا حتى يتم جمع جميع عناصر المصفوفة التي قيمتها 1.

Sum all values from matrix elements

```
m = [
    [2, 4, 6];
    [3, 5, 6];
    [3, 5, 4]
    ];

r = 0;

for i = 1:size(m, 1)
    for j = 1:size(m, 2)
        r = r + m(i, j);
    end
end

disp(r);
```

Output:

```
38
```

هذا المثال يقوم بجمع كل عناصر المصفوفة.

Multiplication involving a scalar and a matrix.

```
m = [
    [2, 4, 6];
    [3, 5, 6];
    [3, 5, 4]
    ];

s = 3; % scalar.

for i = 1:size(m, 1)
    for j = 1:size(m, 2)
        m(i, j) = s * m(i, j);
    end
end

disp(m);
```

Output:

```
6 12 18
9 15 18
9 15 12
```

المثال اعلاه يقوم بضرب كل عناصر المصفوفة في 3 ويعدل عناصر المصفوفة الاصلية على هذا الاساس.

Sum all values from the rows of the matrix

```
m = [
    [2, 4, 4],
    [3, 5, 6],
    [3, 5, 4]
    ];

r = zeros(1, size(m, 1));

for i = 1:size(m, 1)
    r(i) = 0;

    for j = 1:size(m, 2)
        r(i) = r(i) + m(i, j);
        % r(i) = r(i) * m(i, j);
    end
end

disp(r);
```

Output:

```
10 14 12
```

المثال اعلاه يقوم بجمع صفوف المصفوفة كل على حده.

Sum all values from the colums of the matrix

```
m = [
    [2, 4, 4],
    [3, 5, 6],
    [3, 5, 4]
    ];

% Initialize c as a
% row vector of zeros.

c = zeros(1, size(m, 2));

for i = 1:size(m, 1)
    for j = 1:size(m, 2)
        c(j) = c(j) + m(i, j);
        % c(j) = c(j) * m(i, j);
    end
end

disp(c);
```

Output:

```
8 14 14
```

المثال اعلاه يقوم بجمع اعمدة المصفوفة كل على حده.

Sum all values from the left diagonal of the square matrix

```
a = [
    1, 2, 3;
    4, 5, 6;
    7, 8, 9
    ];

% Get the size of the matrix.
[n, ~] = size(a);

d = 0;

for i = 1:n
    d = d + a(i, i);
    disp(a(i, i));
end

disp('---');
disp(d);
```

Output:

```
1
5
9
---
15
```

المثال اعلاه يقوم بطباعة عناصر القطر الرئيسي ثم بعد ذلك يطبع مجموعها وذلك باستخدام القراءة المفصلة والحلقات.

```
% or a short version with MATLAB built-in
functions (i.e. diag and flipud):


a = [
    1, 2, 3;
    4, 5, 6;
    7, 8, 9
    ];

% Extract the main diagonal
% elements using diag().

d = diag(a);

% Sum the diagonal
% elements using sum().

s = sum(d);

disp(d)
disp('---')
disp(s);
```

المثال اعلاه يقوم بالضبط بنفس عمل المثال الذي قبله، ولكن هذه المرة بالطريقة السريعة وذلك باستخدام الدوال الجاهزة التي يوفرها برنامج MATLAB.

Sum all values from the right diagonal of the square matrix

```
a = [
    [1, 2, 3];
    [4, 5, 6];
    [7, 8, 9]
    ];

[n, m] = size(a);
d = 0;

for i = 1:n
    d = d + a(i, n - i + 1);
    disp(a(i, n - i + 1));
end

disp('---');
disp(d);
```

Output:

```
3
5
7
---
15
```

المثال اعلاه يقوم بطباعة عناصر القطر الثانوي ومن ثم جمعها بالطريقة التفصيلية والحلقات.

```
% or a short version with MATLAB built-in
functions (i.e. diag and flipud):


a = [
    [1, 2, 3];
    [4, 5, 6];
    [7, 8, 9]
    ];

% Get the second
% diagonal using diag.

s = diag(flipud(a));

% Sum the values
% in the second diagonal.

d = sum(s);

disp(flipud(s));
disp('---');
disp(d);
```

المثال اعلاه يقوم بالضبط بنفس عمل المثال الذي قبله، ولكن هذه المرة بالطريقة السريعة وذلك باستخدام الدوال الجاهزة التي يوفرها برنامج MATLAB. لاحظ ان الدالة **flipud** وهي مختصر flip up down اي انها تقلب المصفوفة رأساً على عقب، اي تجعل الصف الاخير هو الاول والصف الاول سيكون هو الصف الاخير.

Show bottom – left part of the matrix

```
a = [
    1, 2, 3, 4;
    5, 6, 7, 8;
    9, 0, 1, 2;
    3, 4, 5, 6
    ];

n = size(a, 1);  % rows.
m = size(a, 2);  % columns.

d = zeros(n, m);

r = '';

for i = 1:n
    for j = 1:i
        d(i, j) = a(i, j);
        r = [r, num2str(d(i, j)), ' '];
    end
    r = [r, '\n'];
end

fprintf(r);
```

Output:

```
1
5 6
9 0 1
3 4 5 6
```

```
%{
1 - - -
5 6 - -
9 0 1 -
3 4 5 6
%}
```

المثال اعلاه يقوم بطباعة عناصر المصفوفة فقط الواقعة على القطر الرئيسي او اسفل منه.

Show top – right part of the matrix

```
a = [1, 2, 3, 4;
     5, 6, 7, 8;
     9, 0, 1, 2;
     3, 4, 5, 6];

n = size(a, 1);
m = size(a, 2);

r = '';

for i = 1:n
    for j = i:m
        if j == i
            d(i, j) = a(i, j);
        end
        r = [r, num2str(a(i, j)), ' '];
    end
    r = [r, '\n'];
end

fprintf(r);
```

Output:

```
1 2 3 4
6 7 8
1 2
6
```

```
%{
1 2 3 4
- 6 7 8
- - 1 2
- - - 6
}%
```

المثال اعلاه يقوم بطباعة عناصر المصوفة الواقعة فقط على القطر الثانوي او اعلى منه.

Matrix flip vertical

```
% flip vertical

a = [
    1, 2, 3, 4;
    5, 6, 7, 8;
    9, 0, 1, 2;
    3, 4, 5, 6
    ];

r = '';
[n, m] = size(a);
d = zeros(n, m);

for i = 1:n
    for j = 1:m
        d(i, j) = a(n-i+1, j);
        r = [r, num2str(d(i, j)), ' '];
    end
    r = [r, '\n'];
end

fprintf(r);
```

Output:

```
3 4 5 6
9 0 1 2
5 6 7 8
1 2 3 4
```

المثال اعلاه يقوم بقلب المصفوفة رأساً على عقب بالطريقة التفصيلية واستخدام الحلقات.

تمرين: كيف تنفذ نفس المثال السابق ولكن بالطريقة السريعة باستخدام الدوال الجاهزة وبدون حلقات.

Matrix flip horizontal

```
% flip horisontal.

a = [
    1, 2, 3, 4;
    5, 6, 7, 8;
    9, 0, 1, 2;
    3, 4, 5, 6
    ];

[n, m] = size(a);
d = zeros(n, m);

for i = 1:n
    for j = 1:m
        d(i, j) = a(i, m-j+1);
    end
end

disp(d);
```

Output:

```
4 3 2 1
8 7 6 5
2 1 0 9
6 5 4 3
```

هذا المثال يقوم بقلب اعمدة المصفوفة يمين يسار.

تمرين: ابحث في مساعد البرنامج Help هل هناك ايعاز جاهز وسريع يقوم بحل المثال السابق؟

Sum values from elements of a matrix based on a map

```
a = [2, 4, 6;
     3, 5, 6;
     3, 5, 4];

b = [0, 0, 1;
     1, 1, 0;
     0, 0, 1];

[n, m] = size(a);
r = 0;

for i = 1:n
    for j = 1:m
        if b(i,j) == 1
            r = r + a(i,j);
        end
    end
end

disp(r);
```

Output:

```
18
```

المثال اعلاه يقوم بجمع عناصر محددة من داخل المصفوفة وتحديد هذه العناصر المطلوبة يكون عن طريق مصفوفة اخرى نحن نضع فيها قيم الصفر والواحد حسب الرغبة.

Add two matrices into a third

```
a = [2, 4, 6; 3, 5, 6; 3, 5, 4];
b = [2, 4, 6; 3, 5, 6; 3, 5, 4];

c = zeros(size(a));

for i = 1:size(a,1)
    for j = 1:size(a,2)
        c(i, j) = a(i, j) + b(i, j);

        % Alternatively for subtraction:
        % c(i, j) = a(i, j) - b(i, j);

    end
end

disp(c);
```

Output:

```
4  8  12
6  10  12
6  10  8


%{ for substraction:
0 0 0
0 0 0
0 0 0
%}
```

المثال اعلاه هو عن كيفية جمع (او طرح) مصفوفتين متساويتين بالحجم وانتاج مصفوفة ثالثة، وذلك بالطريقة التفصيلية واستخدام الحلقات.

تمرين: هل هناك ايعاز سريع يقوم بحل المثال السابق؟

Matrix multiplication with three for loops

```
a = [2, 4, 6; 3, 5, 6; 3, 5, 4];
b = [2, 4, 6; 3, 5, 6; 3, 5, 4];

c = zeros(size(a));
r = "";

for i = 1:size(a, 1)
    r = r + newline;
    for j = 1:size(a, 2)
        for k = 1:size(a, 2)
            c(i, j) = c(i, j) + ...
                      a(k, j) * b(i, k);
        end
        r = r + c(i, j) + " ";
    end
end

disp(r);
```

Output:

```
34 58 60
39 67 72
33 57 64
```

المثال اعلاه هو لعملية الضرب الرياضي لمصفوفتين وذلك باستخدام الطريقة التفصيلية وفيها ثلاث حلقات متداخلة.

Matrix multiplication with two for loops

```
a = [2, 4, 6;
     3, 5, 6;
     3, 5, 4];

b = [2, 4, 6;
     3, 5, 6;
     3, 5, 4];

i = 1; j = 1; r = "";
c = cell(1, 1);

n1 = size(a, 1);
n2 = size(a, 2);
q = n1 * n2;

c{1} = [];

for v = 1:q

    j = mod(v - 1, n2) + 1;
    if (j == 1 && v ~= 1 && i <= n1 && v ~= q)
        i = i + 1;
        c{i} = [];
        r = r + newline;
    end

    c{i}(j) = 0;

    for k = 1:size(a, 2)
        c{i}(j) = c{i}(j) + a(k, j) * b(i, k);
    end

    r = r + c{i}(j) + " ";

end

disp(r);
```

Output:

```
34 58 60
39 67 72
33 57 64
```

المثال اعلاه نفس عملية ضرب المصفوفات في حلقتين متداخلتين

```
% or a short version with
% MATLAB built-in functions:


a = [
    2, 4, 6;
    3, 5, 6;
    3, 5, 4
    ];

b = [
    2, 4, 6;
    3, 5, 6;
    3, 5, 4
    ];

c = a * b';

disp(c);
```

بينما في الـ Code البديل في المثال اعلاه، فانه يحل مسألة ضرب مصفوفتين بايعاز واحد في سطر واحد، وذلك باستخدام الدوال المتقدمة المبنية داخل البرنامج Built-in Functions.

الدوال الداخلية والخارجية Built in and External Functions

في كل لغات البرمجة هناك دوال داخلية تكون مبنية داخل اللغة او البرنامج نفسه وهي بشكل من الاشكال تعطي القوة التي يمتلكها ذلك البرنامج او اللغة فكلما كانت الدوال اكثر واوسع كان البرنامج اقوى. فمثلاً بعض اللغات تركز على الدوال الرياضية ولغات اخرى على الاحصاء وهكذا، ويعتبر الماتلاب من اقوى البرامج التي تحتوي مكتبة هائلة من الدوال الرياضية والاحصائية والهندسية. واستخدام الدوال الجاهزة والمبنية داخل البرنامج يجنب المبرمج من الدخول في كل التفاصيل وتعطيه فرصة لأن يركز على المهمة التي يريد برمجتها. وهي كذلك تسهل عملية قراءة البرنامج وفهمه. بالاضافة طبعاً الى انها تقلل احتمال الوقوع في اخطاء برمجية.

والدوال ببساطة هي تشبه الصناديق التي تأخذ من المستخدم فقط متغير واحد او مجموعة متغيرات، ثم تعمل على هذه المتغيرات عمليات محددة تكون غير ظاهرة للمستخدم ثم تعيد له ايضاً ناتج واحد او اكثر حسب وظيفة الدالة وطبيعة عملها.

Built-in *Sin, Exp*

```
a = 3.1415;
b = sin(a);
c = exp(sin(a));
disp(b);
disp(c);
```

Output:

```
0.00009265358966049026
1.000092657882137
```

Max between two integer variables

```
maxA = 6;
maxB = 10;
maxValue = max(maxA, maxB);

disp(maxValue);
```

Output:

```
10
```

Max over the values from an array

```
a = [6, 7, 1, 9];
b = [2, 5, 1, 1];
maxA = max(a);
maxB = max(b);
disp(maxA);
disp(maxB);
```

Output:

```
9
5
```

Random integers from 0 to 99 in an array

```
n = 10;
a = zeros(1, n);

for k = 1:n
    a(k) = randi(100) - 1;
end

disp(a);
```

Output:

```
41,72,71,20,2,
8,40,0,99,38
```

الدالة **randi(n)** هي دالة تعطي رقم عشوائي (صحيح) بين الواحد والـ n لذلك كان علينا ان نطرح منه قيمة واحد لكي نحصل على رقم عشوائي ما بين الصفر والـ 99، حسب مطلب السؤال في الاعلى.

Split string to integers by using a delimiter symbol

```
a = '2#5#7#1#1#2';
b = strsplit(a, '#');

disp(['b = ', strjoin(b, ' ')]);
```

Output:

```
b = 2 5 7 1 1 2
```

Built-in *sort* function

```
b = [3, 6, 2, 78, 99, 1, 4];
b = sort(b);

disp(b);
```

Output:

```
1,2,3,4,6,78,99
```

الدوال الخارجية User Defined Functions

كانت الامثلة الاخيرة اعلاه لدوال داخلية مبنية داخل البرنامج نفسه، وهي طبعاً كثيرة جداً وتحتاج من المستخدم ان يراجع مستندات البرنامج ليتعرف على اكبر عدد منها، او على الاقل يتعرف على الدوال التي يمكن ان يحتاجها في مجال اختصاصه.

اما الدوال الخارجية، فهي عندما يجد المستخدم نفسه محتاجاً الى حساب قيمة معينة او دالة بشكل متكرر ولا يجد في البرنامج ما يحقق هذه الغاية. لذلك يقوم هو ببرمجة هذه الدالة ويعطيها اسماً معيناً، ثم يستخدم هذه الدالة التي هو صنعها كلما احتاج الى ذلك.

ان مفهوم الدوال الخارجية التي يقوم بتعريفها المستخدم هو مفهوم مهم وموجود في كل لغات البرمجة وله تسميات مختلفة منها User defined functions او External او Subroutine حيث يقوم المستخدم بعمل برنامج فرعي صغير يقوم بوظيفة محددة وعادة تكون كثيرة الحاجة ثم عندما يحتاج هذه الوظيفة داخل اي برنامج فانه لا يعيد برمجتها من جديد وانما فقط يستدعي تلك الدالة الخارجية التي عملها مسبقاً. وهذه الفكرة تجعل البرامج اكثر تنظيماً واسهل في القراءة والفهم. وكذلك الاستفادة من جهود الاخرين وتتبع الاخطاء وعدم تكرارها.

المسألة المهمة جداً في الدوال الخارجية ان ينتبه المستخدم الى عدد المدخلات التي سوف تدخل الى الدالة وعدد المخرجات التي سترجع منها. وخطوات عمل الدالة في الماتلاب هي كالاتي:

1) يبدأ برنامج الدالة الخارجية بالكلمة المفتاحية Function ثم يتلوه اسم الدالة الذي يسند الى متغير معين.
2) نختار اسم للدالة ويكون متبوع بقوسين صغيرين ثم تذكر داخل القوسين كل المتغيرات التي تحتاجها تلك الدالة لتقوم بعملها.
3) بناء هيكل او جسم الدالة واجراء كل العمليات الحسابية او غير الحسابية بحيث بالنتيجة تقوم هذه الدالة باجراء المطلوب منها.
4) انهاء عمل الدالة كلها بالكلمة المفتاحية end.

الان لكي نستدعي هذه الدالة من داخل البرنامج الرئيسي ما علينا سوى ان نذكر اسمها ونتبعه بالاقواس الصغيرة ونضع داخل الاقواس الارقام التي نحتاج من الدالة ان تعمل عليها، وهنا ستعود لنا الدالة بنتيجة واحدة فقط نستطيع ان نستخدمها داخل برنامجنا. والمسألة هنا تشبه عمل الدوال الداخلية مثلاً عندما نكتب cos(0) سترجع لنا هذه الدالة الداخلية بناتج هو 1 في هذه الحالة.

Making of a function

```
a = 10;
b = compute(a);

disp(b);

function y = compute(x)
    y = x + x / x - x * x;
end
```

Output:

```
-89
```

في المثال اعلاه، البرنامج الرئيسي هو فقط الثلاث اسطر الاولى، اما دالتنا فهي الاسطر الثلاثة الاخيرة، ونلاحظ اننا في البرنامج الرئيسي قمنا باستدعاء الدالة compute(a) وهنا يجب ان تكون قيمة a معلومة لكي تذهب وتنتقل الى جسم الدالة ثم تقوم الدالة بالعمل عليها وترجع لنا النتيجة المطلوبة.

لاحظ ان جسم الدالة فيه المتغيرات x و y بينما في برنامجنا الرئيسي لم يكن لدينا لا x ولا y وانما كان البرنامج الرئيسي يتحدث عن متغيرين a و b لذلك فان الدالة compute في هذا المثال يدخل لها بأي قيمة عددية معلومة سوف تنوب عن المتغير x ثم تخرج الدالة بنتيجة واحدة فقط وهذه النتيجة سوف تخزن في اسم الدالة نفسها اي compute.

Making of a function with more than one parameter

```
disp(mul(2,5));

function result = mul(a, b)
    result = a * b;
end
```

Output:

```
10
```

المثال اعلاه يتحدث عن دالة يدخل لها اكثر من متغير واحد (والعدد هنا مفتوح) ثم في جسم الدالة تحصل عمليات حسابية على هذه المتغيرات ثم ترجع الدالة بنتيجة واحدة فقط، وهي هنا تقوم بوظيفة الضرب الحسابي لاي عددين.

Gauss summation - sum all from 0 to *n*

```
disp(gs(100));

function result = gs(n)
    result = n*(n+1)/2;
end
```

Output:

```
5050
```

المثال اعلاه لو كنا نحتاج بشكل متكرر ان نجمع الاعداد من صفر الى n فاننا بدلاً من ان نعيد هذه العملية في كل مرة نحتاجها، نقوم بعمل دالة خارجية وظيفتها فقط ان تقوم بهذه المهمة، ثم نستدعيها كلما احتجنا الى ذلك.

الان لو فرضنا ان المستخدم يحتاج الى دالة تقوم بحساب "العامل المشترك الاكبر" لأي عددين، ولم يجد دالة جاهزة داخل البرنامج تقوم بهذه المهمة. فيمكن ان يكتب الكود الخاص به ويعطيه اسم مثلاً GCD كما موضح في المثال التالي:

Greatest common divisor (GCD)

```
% greatest common divisor (GCD).

disp(gcd(45, 12));

function result = gcd(a, b)
    if a == 0
        result = b;
        return;
    end

    while b ~= 0
        if a > b
            a = a - b;
        else
            b = b - a;
        end
    end
    result = a;
end
```

Output:

```
3
```

تشريح المثال اعلاه يكون بالشكل التالي:

السطر الاول ;(disp (gcd(45,12)) يجد البرنامج نفسه مطالب بطباعة قيمة دالة اسمها gcd تعمل على الارقام 45 و 12 فيبحث البرنامج في الدوال الداخلية للـ MATLAB فلا يجد مثل هذه الدالة معرفة عنده. لذلك الان يبحث في الدوال الخارجية في الـ Code نفسه، فيجد ان هناك تعريف يبدأ بالكلمة المفتاحية function وهذه الكلمة تعني ان المستخدم يقوم الان بتعريف دالة خارجية جديدة.

الدالة الخارجية اسمها gcd(a,b) وهي تسند قيمتها بعد ان تحسب الى متغير اسمه result يكون هذا المتغير حتماً موجود داخل تعريف جسم الدالة.

اما الـ (a,b) فهذه هي المتغيرات او الارقام او الـ Arguments التي تتوقع الدالة ان تستلمها من البرنامج الرئيسي لكي تعمل عليها.

ملاحظة: في حال عدم وجود الرمز في لوحة المفاتيح المطلوب في السطر السابع والذي يعني عدم المساواة فيمكن الاستعاضة عن ذلك الشرط بالعبارة المنطقية OR كما في النسخة الثانية من نفس البرنامج ادناه:

```
disp(gcd(45,12));
function result=gcd(a,b);
if a==0
    result=b;
    return
end
while b<0 || b>0
    if a>b
        a=a-b;
    else
        b=b-a;
    end
end
result=a;
end
```

المثال التالي يبين كيف يمكن تعريف دالة خارجية داخل دالة خارجية اخرى

Function calls to other functions

```
function example
    d = main_app(66, 100);
    disp(d);
end

function cc = main_app(x, y)
    cc = sebastian(x, y);
end

function p = sebastian(a, b)
    p = daniela(a, b);
end

function result = daniela(a, b)
    result = a + b;
end
```

Output:

```
166
```

رغم ان المثال اعلاه لا يعمل شيء مهم فهو فقط يجمع عددين ولكنه يوضح كيف يمكن للبرنامج الرئيسي ان يستدعي دالة ثم هذه الدالة تستدعي دالة اخرى وهكذا الى ان نحصل على النتيجة المطلوبة.

دوال الادخال والإخراج (I/O) من والى الملفات

ان ادخال البيانات التي نحتاجها عملياً يمكن ان تقسم الى عدة اقسام:

1) قراءة مجموعة من القيم تعرف عادة اما بالـ Parameters او الـ Control values وهي القيم التي توجه أو تتحكم بمسار البرنامج، وهذه القيم اذا كانت قليلة فربما نفضل ان نقرأها من لوحة المفاتيح والشاشة مباشرة عن طريق الايعاز Input الذي مر علينا سابقاً. اما اذا كانت كثيرة فيفضل ان نضعها في ملف مستقل يمكن ان يطلق عليه Control File.
2) وعند الرغبة في انشاء الـ Control File فهناك خيارين، اما ان ننثر القيم لوحدها على اسطر مختلفة بدون أي إيضاح، وهذا سيفترض ان المستخدم يعرف بالضبط ماذا تعني كل قيمة حتى يستطيع تغييرها حسب حاجته، او الطريقة الأفضل ان نكتب مع كل قيمة شرح او ايضاح يبين ماذا تعني هذه القيمة وربما حتى الحدود المسموحة لها، ولكن هذا الخيار الأخير يجبرنا على ان نحسب حساب هذه التعليقات وان نتمكن من جعل البرنامج يتجاوزها او يقفز عليها ويقرأ فقط القيم التي نريدها دون ان يقرأ التعليقات الايضاحية.
3) النوع الثاني من البيانات التي نقرأها من ملف خارجي هو قراءة مجموعة صغيرة او كبيرة من القيم على شكل مصفوفة أو "جدول"، مثلاً قراءة درجات الحرارة العظمى والصغرى لعدد ايام سنة كاملة وهنا ستكون لدينا مصفوفة من 365 صف وعمودين، او قراءة درجات سبع مواد لألف طالب مثلاً فيكون لدينا مصفوفة من 1000 صف مع 7 اعمدة، وهذا النوع من البيانات له ايعاز خاص في الـ MATLAB سوف نبدأ به.

القراءة من ملف الى مصفوفة

الايعاز **load** يقوم بقراءة ملف كامل فيه قيم فقط، وهذه القيم مرتبة على شكل مصفوفة أي أعمدة ثابتة العدد وصفوف، وبدون اي تعليقات او اشياء اخرى، وبالتالي سوف يحفظ كل القيم الموجودة في الملف داخل مصفوفة يكون لها نفس اسم الملف، وبعد ان نحصل على المصفوفة فيمكننا بما تعلمناه سابقاً ان نصل الى اي قيمة داخل هذه المصفوفة والتعامل معها.

```
>>load testfile.dat
```

ان تنفيذ هذا الايعاز سوف ينشئ لدينا مصفوفة داخل البرنامج سيكون اسمها (testfile)، اما محتواها فسيكون جميع الارقام الموجودة في الملف النصي testfile.dat.

أما الايعاز **save** فيقوم بالضبط بعكس هذه العملية أي عكس الايعاز **load** لأنه يقوم بتحويل المصفوفة التي لدينا في البرنامج الى ملف خارجي (وبالتالي فهو ايعاز اخراج وليس ادخال) ويلاحظ هنا اننا نحتاج ان نعطيه اسم الملف المراد إخراجه و اسم المصفوفة التي لدينا، كما نلاحظ ان هناك لاحقة مع هذا الايعاز وهي **–ascii** تضمن ان يكون اخراج ارقام المصفوفة على شكل ملف نصي أي يمكن قراءته فيما بعد بواسطة احد برامج معالجة النصوص مثل الـ Notepad او برنامج الـ Word اما اذا لم نضع هذه اللاحقة فأن الحفظ سوف يكون بلغة الماكنة أي Binary وهذا النوع من الملفات لن نتمكن من قراءته او مشاهدة محتوياته. (وهنا تجدر الإشارة الى ان كل دوال القراءة والكتابة في الماتلاب لها الخيارين اما الـ (Text Mode or Binary Mode)

```
>>save testfile.dat mymat – ascii
```

تنفيذ هذا الايعاز يعني ان المصفوفة mymat التي لدينا أصلاً داخل البرنامج نريد إخراجها او تصديرها الى ملف خارجي سيكون اسمه testfile.dat.

فإذا كان الملف testfile.dat أصلاً موجود فأن إيعاز save سوف يعمل Overwrite أي انه يلغي ما موجود في الملف الأصلي القديم، هذا طبعاً ان كتبنا الايعاز بشكله الافتراضي لأنه بهذا الشكل دائماً يبدأ الطباعة من بداية الملف.

واذا لم يكن الملف testfile.dat موجود اصلاً في الحاسبة، فأن ايعاز save سوف ينشأه من الصفر. اما الحالة الثالثة وهي ان يكون الملف testfile.dat موجود في الحاسبة، ونحن لا نريد ان نلغي محتوياته القديمة وانما نريد ان نضيف عليها المصفوفة التي لدينا (وهذه المسألة نحتاجها كثيراً) فعلينا إضافة رمز ما يعرف بالـ qualifier وهو في حالتنا هذه كتابة –append في نهاية الايعاز فيأخذ الايعاز الشكل التالي:

```
>>save testfile.dat mymat – ascii - append
```

ويقرأ هذا الايعاز بالشكل التالي: احفظ المصفوفة التي لدينا وهي mymat في ملف خارجي اسمه testfile.dat بشكل نصي على ان لا تمسح محتويات الملف الاصلية بل تضيف المصفوفة في نهاية الملف.

ملاحظة: بما ان استخدام الايعاز save هو للمناقلة بين مصفوفة وملف، فلا يفضل ان يكون الملف حاوياً على جزء فيه عدد معين من الاعمدة مثلاً 5 أعمدة، ثم بعد ذلك نضيف عليه جزء اخر عن طريق الإيعاز save وباستخدام الإضافة append– ويكون ذلك الجزء المضاف بعدد أعمدة مختلفة كأن تكون 7 مثلاً، والسبب اننا إذا احتجنا في المستقبل ان نعيد ذلك الملف الى شكل مصفوفة فلن نستطيع لأن عدد الاعمدة غير ثابت على طول الملف.

ونتذكر هنا الملاحظة السابقة وهي اننا لا يمكن ان نستخدم الايعاز load اذا كان الملف المقصود غير متجانس من حيث عدد الاعمدة، كذلك اذا لم تكن البيانات متجانسة من حيث النوع، أي يجب ان تكون اما كلها ارقام او كلها حروف ...الخ.

تطبيق

اذا كان لدينا ملف نصي (وهو حسب قواعد الـ ASCII Format إما بامتداد dat او txt وكانت فيه البيانات التالية وهي عبارة عن سطرين السطر الأول فيه الوقت بالساعات من اليوم علماً ان الوقت 0 هو منتصف الليل، والسطر الثاني كانت فيه درجات الحرارة المقابلة لهذه الأوقات:

```
0     3     6     9     12     15     18     21
30.2  29.5  30.4  32.6  42.1   40.4   36.6   31.8
```

فأن كتابة البرنامج التالي سوف يقرأ هذا الملف ثم يحوله الى مصفوفة، ثم يقسم هذه المصفوفة الى عمودين الأول للوقت والثاني فيه درجة الحرارة ثم في الأخير سوف يرسم العلاقة بين الوقت ودرجة الحرارة:

```
%This reads time and temperature from a file then plot the data
load timetemp.dat
%the time is in the first row, and the temperature in the second
```

```
time=timetemp(1,:);
temp=timetemp(2,:);
%plot the data and label the plot
plot(time,temp,'k*')
xlabel('Time')
ylabel('Temperature')
title('The temperature of a day')
```

الايعازات الاخيرة من البرنامج اعلاه هي لاظهار المخطط بشكل منسق اكثر كأن تعطى علامة للنقاط وكذلك تعطى عناوين للمحاور وعنوان رئيسي للمخطط كله.

لاحظ انك عندما تريد ان تنشئ ملف البرنامج الأصلي أي بامتداد m. فمجرد ان تعمل حفظ فان الماتلاب سوف يقترح عليك هذا الامتداد تلقائياً وبالتالي ما عليك سوى ان تعطيه اسم الملف الذي تريده. اما اذا اردت ان تنشأ ملف الادخال وهو عادة يكون بامتداد اما txt. او dat. فيجب ان تنتبه، لأن الماتلاب سوف يقترح عليك مرة أخرى الامتداد m. وانت لا تريد ذلك هذه المرة، فالذي عليك فعله هنا هو ان تذهب الى اسفل مربع الحوار وتغير الـ Save as type وتجعله All Files ثم بعدها تعطي الاسم الذي تريده للملف وتعطيه كذلك الامتداد كأن يكون مثلاً me.txt او me.dat.

ملاحظة:

ايعاز flipud أي Flip up down يعمل مع المصفوفات وهو يقلبها رأساً على عقب من حيث الصفوف، أي يعيد تسلسلها بحيث يكون الصف الأخير هو الأول وهكذا حتى يصل الى الصف الأول فيجعله الصف الأخير، وهذا الايعاز يفيدنا اذا كانت لدينا ملفات غير مرتبة فنقرأها عن طريق الايعاز load لتتحول الى مصفوفة ثم نجري عملية القلب هذه، ثم نعيد طباعتها في ملف جديد عن طريق الايعاز save.

كان الإيعازان save & load هما اول ايعازات التعامل مع الملفات أي القراءة والكتابة في الملفات، ولكن يلاحظ ان لهما شروط حادة وهي تفيد في حالات خاصة محددة، وليس فيهما المرونة الكافية التي قد نحتاجها عملياً، مثل ان الملف يجب ان يحتوي على عدد ثابت من الاعمدة، كما ان بياناته يجب ان تكون كلها من نفس النوع، وكذلك الشرط المهم وهو ان يكون الملف نظيف تماماً اي فيه ارقام فقط بدون اي عناوين او ايضاحات ذلك لكي يستطيع البرنامج ان يقرأ الملف ويحوله الى مصفوفة. ويلاحظ ان هذين الايعازين لهما تخصص معين وهو التعامل مع المصفوفات فقط.

اما إذا اردنا مرونة وإمكانية اكبر من ذلك في التعامل مع الملفات فهناك في الماتلاب ما يعرف بالـ

Lower Level File I/O Functions

ويلاحظ في هذا النوع من الدوال التي تقرأ او تكتب في الملفات الخارجية بمرونة عالية هو اننا يجب ان **نفتح الملف** او ننشأه أولاً، ثم ان هذه الايعازات يكون لديها مؤشر يستطيع ان يميز اين هي بالضبط بداية الملف، ثم يستطيع هذا المؤشر ان يتحرك داخل الملف الى المكان المناسب ثم يقرأ ما نحتاج قراءته فقط، ثم بعد ذلك كله يجب علينا ان **نغلق الملف**.

والخطوات هي:

- Open or create the file
- Read or Write to the file
- Close the file

وهنا سنبدأ الحديث عن الخطوة الأولى والثالثة وهما فتح الملف وغلق الملف، اما الخطوة الوسطية فسنعالجها لاحقاً لأن فيها العديد من عبارات القراءة والكتابة.

فتح الملف: ويكون الشكل النموذجي لهذا الايعاز هو:

```
fid=fopen('filename','permission string')
```

والشرح من اليسار الى اليمين ان fid هو مجرد اسم متغير من النوع الصحيح (ويمكن اعطاءه اي اسم) وقد اخترنا هذا الاسم على أساس انه file ID المهم انها سوف تأخذ رقم معين من جراء كل هذه العملية (وهو رقم صحيح)، وهذا الرقم له معنى سوف نتعامل معه داخل البرنامج، فأذا كان هذا الرقم هو **-1** فهذا يدل على ان عملية فتح الملف لم تتم بنجاح، حيث ربما ان هذا الملف غير موجود في المكان المحدد أي في المجلد الفعال او أي أسباب أخرى، اما اذا نجحت عملية فتح الملف فأن هذا المتغير fid سيأخذ **اي رقم صحيح اخر** لنقل مثلاً 5 وهذا الرقم مهم لأننا في المستقبل يمكن ان نتعامل مع ذلك الملف عن طريق هذا الرقم الذي اصبح الان هوية الملف أو الـ File ID.

اما fopen فهي الدالة التي وظيفتها فتح الملف، ثم filename وهو اسم الملف المقصود ويجب ان نذكره كاملاً **مع الامتداد**، اما الـ permission string فله احتمالات كثيرة ثلاثة منها تهمنا الان وهي:

r: وهذه تعني للقراءة (أي افتح الملف للقراءة فقط وهذه هي الحالة الافتراضية)

w: وهذه تعني افتح الملف لغرض الكتابة.

A: وهذه تعني Append أي اننا سوف نواصل القراءة او الكتابة ولن نبدأ من السطر الأول.

بالنسبة الى غلق الملف او الملفات حيث اتفقنا ان جميع الملفات التي تم فتحها اثناء البرنامج يجب ان تغلق قبل نهاية البرنامج، فأن الايعاز هنا هو مشابه بالشكل لايعاز فتح الملف وهو:

```
Close1=fclose(fid);
```

لاحظ هنا اننا ايضاً وضعنا متغير الى الطرف الايسر وان هذه العملية كلها ستعطي رقماً لذلك المتغير، وهذا الرقم هنا له احتمالين فقط فأما ان يعطي **-1** وهذا يعني ان عملية الغلق **لم تتم بنجاح** او يعطي قيمة **صفر** وهي تعني ان عملية غلق الملف قد تمت بنجاح.

ولاحظ هنا كيف اننا استفدنا من الـ file identifier (fid) حيث ان الايعاز هنا عرف أي الملفات بالضبط التي نريد ان نغلقها عن طريق معرف الملف.

بقي ان نذكر اننا لو كنا قمنا بفتح العديد من الملفات اثناء البرنامج فيمكننا ان نغلقها كلها دفعة واحدة بايعاز واحد وشكل هذا الايعاز سيكون:

```
Closeresult=fclose('all');
```

لاحظ ان كلمة all هنا يجب ان توضع داخل قابسات.

وصلنا الان الى الخطوة الوسطية المهمة وهي القراءة او الكتابة في الملفات ولنبدأ بعبارات القراءة من الملفات ويوجد منها الكثير:

fscanf **textscan** **fgetl** **fgets**

فمثلاً الايعازان الاخيران **fgetl & fgets** يختصان بقراءة سطر واحد في المرة الواحدة، لذلك من جهة نتوقع ان يكون فيهما سيطرة اكثر على عملية القراءة ومن جهة اخرى يتوقع ان نضعهما داخل حلقة Loop عندما نريد قراءة اكثر من سطر واحد.

اما الايعازان **fscanf & textscan** فهما يقرءان الملف كله دفعة واحدة، لذلك فهما من حيث المرتبة او الـ Level يقعان بين ايعاز load وبين ايعاز fgetl حيث ان الأخير هو الأدنى مرتبه Low Level لأنه يستطيع ان يتحكم في أي سطر نريد قراءته بالضبط. والفرق بين الـ fgets والـ fgetl أن الأول يقرأ علامة نهاية السطر ويحتفظ بها ان وجدت، بينما الثاني فأنه يتجاهلها حتى لو كانت موجودة.

وتعتبر الـ fgetl هي الأكثر تحكماً في القراءة من بين كل ايعازات القراءة هذه. انها تقرأ السطر الواحد على شكل string ثم يمكننا التعامل مع هذا الـ string باستخدام الايعازات الخاصة بالتعامل معه.

عادة نستخدم ايعاز الـ **feof** مع هذا الايعاز، والسبب ان هذا الأخير هو عبارة عن مؤشر لوصولنا الى نهاية الملف، حيث انه يعطي قيمة منطقية صحيحة (**True or 1**) اذا انتهت بيانات الملف، ويبقى يعطي قيمة منطقية خاطئة (**False or 0**) اذا لم تنتهي بيانات الملف، لذلك يمكننا ان نضعه ضمن شرط أن استمر بالقراءة الى ان ينتهي الملف.

ان البرنامج التالي يبين مجموعة من الخصائص المترابطة التي نستخدمها عادة لقراءة الملفات سطراً سطراً:

تطبيق

```
fid=fopen('filename');
if fid==-1
disp('file open not successful')
else
while feof(fid)==0
%Read one line into a string variable
Aline=fgetl(fid);
%Use string functions to extract numbers, strings, etc from the line
%Do something with the data!
```

```
end
closeresult=fclose(fid);
if closeresult==0
disp('file close successful')
else
disp('file close is not successful')
end
end
```

ملاحظة ان كتابة عبارة الـ while يمكن ان تكون بالشكل:

while ~feof(fid)

وهي تقرأ بنفس الطريقة أي: "ما دمنا لم نصل الى نهاية الملف"

لحد الان لم نتحكم بأي الاسطر فعلاً التي نريد قراءتها، وهناك حالة تواجهنا كثيراً ففي معظم الملفات ذات البيانات الكثيرة، فأن هناك في بداية الملف ما يعرف بالـ Headers وهي اسطر عادة تكون كتابية أي ليست ارقام وفيها شرح لهذا الملف وبياناته، ونحن عادة لا نريد قراءتها.

اذن لكي يكون عملنا اكثر دقة وسيطرة نحتاج ان نبلغ البرنامج مثلاً بأن يبدأ القراءة من السطر رقم 9 الى السطر رقم 1023 مثلاً، فكيف نقوم بذلك؟

تجدر الاشارة هنا اننا لا نترك السطور التي لا نريدها بدون قراءة، وانما نقرأها بواسطة الايعاز fgetl مثلاً الذي يقرأ سطر سطر ولكننا لا نهتم بتلك القراءة، الى ان نصل الى السطر الذي فيه ارقام وهنا نأخذ ناتج القراءة ونفككه ونتعامل معه.

```
while feof(fid)==0
Aline=fgetl(fid);
Aline=Aline(26:end);
```

في الكود اعلاه نقرأ السطر الاول عن طريق الايعاز fgetl(fid) ثم بعدها نخزن المعلومات التي قرأناها ابتداءاً من العمود 26 فما بعده، ونهمل الاعمدة التي قبلها.

بعد أن صارت لدينا القدرة على قراءة كل الاسطر التي نريدها، يجب ان تأتي الخطوة التالية هنا وهي ان نفتح تلك الاسطر التي لحد الان خزنت على شكل Strings ولكننا في الحقيقة نريد ان نفككها الى عناصرها الحقيقية، وهي في الغالب مجموعة ارقام فكيف نقوم بذلك؟؟

Place holder format:

%5d five places width for integer

%10s ten places width for string

%6.2f six places width for the whole number including the two decimal places.

%.3f any width but the decimals are three.

قراءة الملفات الممزوجة نصية ورقمية وتفصيلها والتعامل معها كل على حده:

fgetl	Read line from a file, discard newline character
fgets	Read line from a file, keeping newline character
fopen	Open file
fprintf	Print formatted data to a file
fscanf	Read formatted data from a file

يأتي الان دور طباعة النتائج في ملف خارجي وسنأخذ الايعاز **fprintf**

ويكون بالشكل التالي

fprintf(fileID,formatSpec,A1,...,An)

من اليسار الى اليمين اطبع في الملف الخارجي ذو الرقم fileID حسب الفورمات formatSpec وهي هنا اما ارقام صحيحة او ارقام حقيقية او الاشكال الاخرى. كل من المتغيرات او النتائج A1,...,An.

تطبيق على القراءة والطباعة في ملفات خارجية لحساب مساحة مجموعة من الدوائر المتواجد انصاف اقطارها في ملف خارجي، مكتوب بالشكل التالي:

r1 r2 r3 r4 r5

5 7 10 12 20

```
% Program calculate area of many circles
inputfid=fopen('radius.dat')
if inputfid==-1
   disp("input file coludn't opened")
end
outputfid=fopen('output.dat','w')
if outputfid==-1
   disp("output file coludn't opened")
end
data0=fgetl(inputfid)
data1=fgetl(inputfid)
disp (data0)
disp (data1)
```

```
for i=1:5
area(i)=pi*data1(i)^2;
end
disp(area)
fprintf(outputfid,data0);
fprintf(outputfid,data1);
fprintf(outputfid,'area is %5.3f---',area);
closing = fclose('all')
```

وملف الاخراج سيكون بالشكل التالي:

r1 r2 r3 r4 r5 5 7 10 12 20area is 8824.734---area is 3216.991---area is 3216.991---area is 3216.991---area is 3216.991---

تمرين:

لاحظ ان ملف الاخراج للبرنامج السابق قد كتب كل النتائج في سطر واحد! كيف يمكنك ان تحسن عبارات الطباعة في البرنامج بحيث يطبع النتائج في ثلاث سطور لتظهر البيانات والنتائج على شكل جدول واضح؟

ما الذي سيختلف لو استبدلت الحرف l بالحرف s اي استخدمت عبارة القراءة fgets بدلاً من fgetl في البرنامج السابق؟

بقي ان نذكر ان هناك ايعازان يختصان بالقراءة والكتابة من ملفات الاكسل وهي بالشكل التالي:

قراءة مصفوفة من ملف اكسل:

```
M=xlsread('MK.xlsx',1,'A1:f6');
```

من اليسار الى اليمين: M المصفوفة التي سوف تتكون لدينا، ثم ايعاز القراءة، ثم اسم الملف الذي يحتوي على البيانات، ثم الرقم واحد يشير الى تسلسل الـ sheet او ورقة العمل داخل ملف الاكسل، ثم النطاق الذي نريد قراءته فقط.

```
xlswrite('out.xlsx',f);
```

بينما هذا الايعاز الأخير يقوم بإخراج المصفوفة f على شكل ملف اكسل.