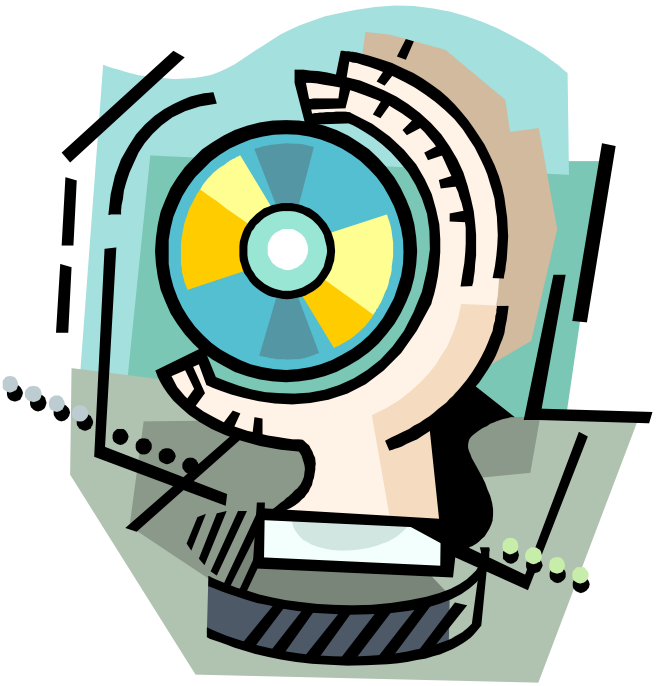
سلسلة الكتب الإلكترونية
الطبعة الأولى نوفمبر ٢٠٠٧



مقدمة في برمجة لغة الليـنجو

(لغة برنامج الدايركتور)

Introduction in Lingo Programming

Macromedia/Adobe Director

تأليف

م. محمد إبراهيم حسانين

اهداء

إلى أمي وأبي الحبيب, اللذان وقفا ورائي حتى أصبحت ما عليه الأن..

إلى زوجتي الغالية لمياء, العظيمة التي تقف ورائي والملهمة لكل ما أفعل وأعمل..

إلى كل مجتهد في هذه الحياة يحب العمل وتطوير هذه الأمة العربية والإسلامية ودفعها للتقدم..

إلى كل هؤلاء أهدي هذا الكتاب المتواضع..

بسم الله الرحمن الرحيم

وَقُل رَّبِّ زِدْنِي عِلْماً

عن المؤلف:

اسمي محمد ابراهيم حسانين من خريجي كلية الفنون التطبيقية للعام ٢٠٠٠, تخصص قسم الإعلان وقد درست معظم برامج التصميم (فوتوشوب - كوريل درو - أدوب انديزاين - أدوب بريمير - أدوات الرسم وأدوات التحويل وغيرها) ثم اتجهت لدراسة الدايركتور بجدية بعد التخرج مباشرة وبعد سنتين جائتني فرصة للعمل في وزارة التربية والتعليم بالمملكة العربية السعودية على رأس فريق لإنتاج الوسائط المتعددة التعليمية, وقد كانت فرصة رائدة للإبحار عبر الدايركتور وتقنيات الوسائط المتعددة لمدة أربع سنوات متصلة أنتجت فيها مع فريق العمل عشرات العناوين التعليمية المصممة والمبرمجة بالدايركتور والفلاش والفيديو, كانت تجربة رائدة ومتميزة كنت مسؤلا في ذلك الوقت على تصميم الشاشات بالفوتوشوب وكذلك برمجة البرنامج ذاته وتطوير وبرمجة الاختبارات الالكترونية والتمارين الخاصة بكل برنامج, وفي نفس هذه الفترة استطعت تطوير حوالي ٣٠ تطبيق في كل مجالات الوسائط المتعددة من البومات الكترونية وألعاب تعليمية وكذلك برامج المؤتمرات والعروض, حاليا أعمل مدير تطوير الوسائط المتعددة في احدى شركات القطاع الخاص المتخصصة في الانتاج الفني بالمملكة العربية السعودية وأعمل حاليا في عدة مشروعات معا وما توفيقي إلا بالله.

يمكن الاتصال بي على البريد الإلكتروني medos20@yahoo.com

مقدمة

لمن هذا الكتاب؟

هذا الكتاب الإلكتروني موجه إلى هؤلاء الأشخاص اللذين يستخدمون الدايركتور ولكن يودون التوسع إلى فهم لغة برمجة الدايركتور وهي لغة اللينجو **Lingo** كما ذكرت سابقا.

ماذا يقدم هذا الكتاب؟

هذا الكتاب يقدم مقدمة شاملة عن برمجة لغة اللينجو, المفاهيم الأساسية للغة, هيكلية برمجة اللغة أو ما يسمى بـ **Lingo Terminology** أي علم اللغة كمفهوم, بالطبع ابتعدت تماما عن البرمجة الخطية, هذا الكتاب مفيد لكل من المبرمجين والمصممين الجرافيكيين اللذين يرغبون في تعلم لغة اللينجو على حد سواء.

ماذا يفترض هذا الكتاب؟

هذا الكتاب لا يفترض شئ بخصوص خبرتك أو مستواك في لغة اللينجو, فحتى لو كنت مبرمجا محترفا فسوف تستفيد من هذا الكتاب, وأيضا سيستفيد المدربون والمعلمون المتخصصون في لغة اللينجو من هذا الكتاب لأنه يحمل رؤية خاصة وفريدة من نوعها, هذا الكتاب المتواضع هو نتيجة شهور طويلة من البحث والتطوير والبحث عن الأفضل والله الموفق لكل خير.

ماذا بعد هذا الكتاب؟

بإذن الله تعالى سأجتهد لإستكمال الكتاب ككل وجعله مرجعا عربيا لبرنامج الدايركتور, سأحاول مجتهدا تقديم مرجعا عربيا خالصا قيما في برنامج الدايركتور بحيث أتناول جميع امكانيات البرامج وأتوسع في شرح البرنامج واللغة وكذلك سأضمن بعون الله تطبيقات تجارية حقيقية ومشاريع قيمة تم تنفيذها في أرض الواقع, سأقدم كل ذلك مجانا في قرص الكتاب المرفق وسأضمن شرح المشاريع داخل الكتاب داخل ملحق التطبيقات, أتوقع أنني سأستغرق وقتا طويلا قبل الانتهاء من كتابة الكتاب ومراجعته بشكل دقيق وربما أصدرت أدوب الإصدار الجديد من دايركتور أثناء كتابتي للكتاب وفي هذه الحالة سأعيد تعديل الكتاب ليتوافق مع الإصدار الجديد, وبالطبع ستكون فصول هذا الكتاب الإلكتروني جزءا من الكتاب النهائي بعد تعديلها وتطويرها وأدعو لي بالتوفيق والهمة لإنجاز هذه المهمة الصعبة.

اتصل بي!

يمكنك الاتصال بي على البريد الإليكتروني medos20@yahoo.com وكذلك في منتدى الدايركتور في منتديات (الميديا للجميع) www.media4all.net بالإسم DirectorMan وكذلك في منتدى صندوق الكتاب www.boxbok.com من خلال المنتدى الفرعي (صندوق الوسائط المتعددة) وبنفس الإسم السابق.

الفصل الأول

لغة اللينجو

في هذا الفصل سنقوم بشرح برمجة الدايركتور بواسطة لغة اللينجو, لأن الدايركتور وحده رغم كونه برنامج قوي جدا ينافس الباوربوينت بمراحل متعددة فما زال غير قادر على تنفيذ كل خيالات مصممي الوسائط الطموحين الى الجموح بخيالهم الواسع لتحقيق كل أحلامهم في مجال الوسائط وتقديم عروض جبارة لا يستهان بها ولذلك كانت شركة ماكروميديا **Macromedia** حريصة على إضافة خصائص غير محدودة للبرنامج وهذه الخصائص الغير محدودة للبرنامج أتت عن طريق تزويد البرنامج بلغة برمجة خاصة به تسمى لغة اللينجو **Lingo Scripting Programming language** وهي اللغة التي اخترعها كبير علماء الشركة وهو جون – جي تي – سومبسون **John (JT) Thompson** والمشهور بلقب دكـــتور اللينجو **Dr .Lingo** والدايركتور كبرنامج ولغة قد كتب أصلا بلغة **Microsoft Visual C++** ولجون رأي آخر في لغة اللينجو حيث يقول "أن الناس تقول: لا تستطيع أن تفعل ذلك باللينجو, فهؤلاء لم يجربوا قوة اللينجو الحقيقية" ومختصر معنى كلمات العالم الكبير جون هو أن لغة اللينجو التي تأتي مع الدايركتور هي لغة كاملة الوظائف أتت لتخدم مصمم الوسائط ومصمم البرامج في نفس الوقت, فلغة اللينجو هي لغة برمجة كاملة الوظائف تتعدى أحيانا جوانب ضعف نراها كثيرا في لغة مثل لغة الفيجوال بيسك بمراحل!, لأن اللغة تحتوي على دوال وتعليمات روتينية شاملة تخدم الوسائط نفسها وتخدم كل الخدمات البرمجية الأخرى كما أن لغة اللينجو من اللغات المفتوحة التي يمكن الاضافة اليها عن طريق الاضافات الخاصة **Xtra Plugins** مما يزيد من قوة وفعالية اللغة المستخدمة, والظريف في لغة اللينجو هي أنها لغة تعليمات منتظمة **Scripting Language** وتعتمد في نفس الوقت على تقنية **OLE** وتستطيع تشغيل واعتماد أوامر **API** كما يمكنك تشغيل تقنية **ActiveX** أيضا بالاضافة الى اعتمادها تقنية **MUI** الخاصة بها, ربما ما ينقص اللينجو هو امكانية الربط بقواعد البيانات **DataBase** فحتى الإصدارة الحالية لا يمكن فعل ذلك إلا باستخدام إضافات خاصة تباع منفصلة فقد قامت العديد من الشركات الكبرى بحل هذا الموضوع بشكل كامل, عموما مصمم الوسائط المتعددة ليس مضطرا في أي حال من الأحوال إلى تعلم اللغة بالكامل لكنه بامكانه تعلم أجزاء من اللغة التي ستفيده في انهاء عمله بطريقة مرضية وذلك لشمولية أوامر اللغة المستخدمة وهنا في هذا الكتاب ساقوم بشرح مقدمة عن منهج لغة اللينجو **Lingo Language Structure** حتى يستطيع المصمم أو مبرمج الوسائط التعامل معها بسهولة ويسر وما يجب أن تعرفه أو تكون على علم به هو مدى حاجتك لتعلم لغة اللينجو فاذا كنت ممن يتعاملون مع العروض التقديمية أو الأقراص التقديمية للمنتجات والشركات فسيكفيك أن تتعلم جزءا يسيرا من لغة اللينجو لكي تقوم بتنفيذ مهامك في تصنيع هذه المشاريع وإخراجها بشكل جيد أما إذا كنت ممن يهتمون بإنتاج البرامج التعليمية والألعاب التعليمية فيجب أن تدرس اللغة حتى مستوى متقدم, أما إذا كنت تريد أن تكون لديك القدرة على إنتاج جميع أنواع الوسائط المتعددة بواسطة الدايركتور فيجب عليك الإلمام بكل أوامر اللغة بحيث تكون مدركا لحدودها وإمكانياتها المتاحة.

أول ساعة في البرمجة!

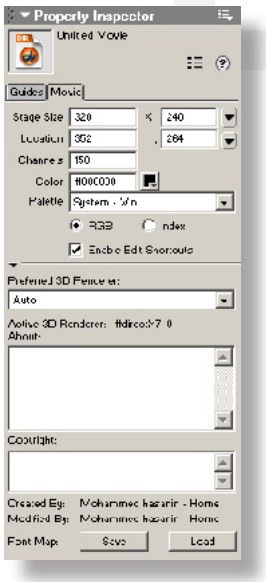
لا يهم ماذا تنوي أن تفعل بالأوامر التي تنوي كتابتها ولكن المهم هو أنك ستتبع نفس الأسلوب في كل مرة تكتب فيها الأكواد فهناك خطوات ستأخذها بعين الاعتبار في كل مرة وهي تستلزم منك ورقة بيضاء فارغة وقلم ثم تقوم بكتابة الخطوات التالية:

١- ما هي العملية التي أرغب في تنفيذها باللينجو؟ أجب السؤال!
٢- ما هي الأوامر التي يجب استخدامها؟
٣- محاولة التوصل للأوامر التي يجب كتابتها وتجربتها
٤- تصحيح الأخطاء والوصول للنتائج بسرعة

فكلمة **Script** وكلمة **Behavior** هما مرادفان لكلمة واحدة وهي الأكواد ولكن الفرق بينهما وظيفي!, فالمسمى العام لمعنى الكود هو كلمة **Script** وهي تعني أي منظومة خوارزمية* تحتوي على أوامر مرتبة تؤدي الى تنفيذ أمر معين أو تجعل الكمبيوتر يستجيب لأمر معين, بينما كلمة **Behavior** وهي تعني السلوك وهو أيضا عبارة عن مجموعة من الأوامر المرتبة خوارزميا* ولكن السلوك يختلف في كنه عمله اذ انه احيانا يجعل الكمبيوتر يستجيب لأمر معين وأحيانا أخرى يجبر الكمبيوتر على التقيد بأمر معين فمثلا اذا أردت ارسال رسالة من المملكة السعودية الى دولة الكويت فهذه الرسالة ربما تسلك طريق الجو الى صاحبها وهو طريق اجباري تتعهده شركة البرق والهاتف بينما يمكن التحايل على ارسال الرسالة عن طريق البر وفي كلتا الحالتين سوف تصل الرسالة وسوف نفهم من الأمثلة القادمة والتجربة العملية الفروق الجوهرية بين طريقة عمل الكود **Script** و السلوك **Behavior** من خلال طريقة استجابة الدايركتور.

كتابة كود بسيط!

١- قم باختيار ملف جديد بالضغط على **Ctrl + N** أو باختيار ملف جديد من **File > New > Movie**
٢- اختر من قائمة **Window** الاختيار **Inspectors Property** أو استخدم المختصر **Ctrl+Alt+S** كما بالشكل التالي:



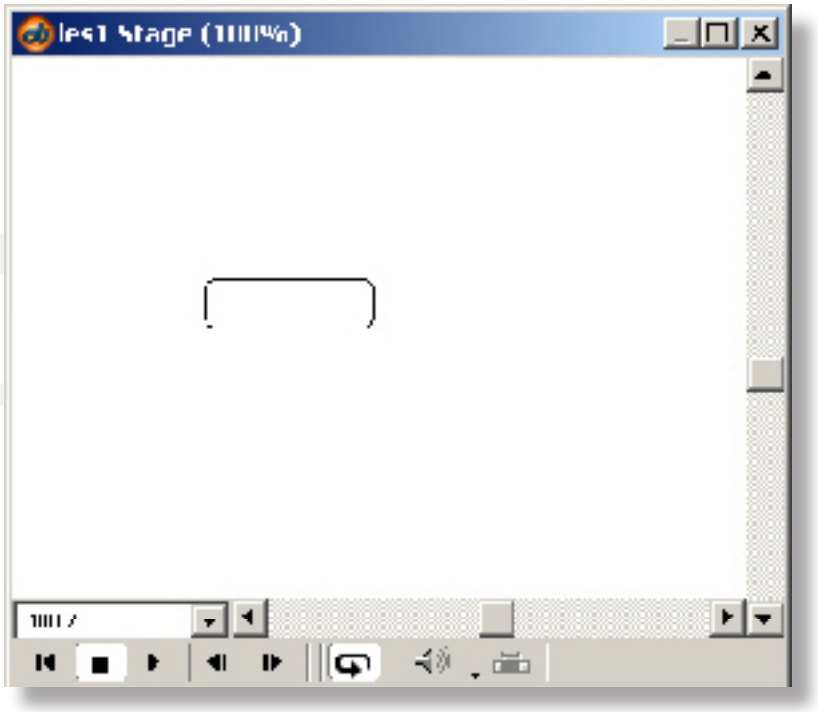
* نسبة للعالم الاسلامي الخوارزمي والخوارزمية هي مجموعة تعليمات مرتبة تؤدي عملية منطقية *Logical Algorithm*

٣- كما ترى من الشكل السابق فقم باختيار التبويب **Movie** ثم الاختيار **Centered** الذي يعادل **352X264** وتأكد من أن المقاس هو **320X240** ليكون هذا هو مقاس الصفحة المسرح **Stage**, ثم غير لون المسرح باستخدام الزر  للون الأبيض.

٤- قم بفتح بالتة الأدوات على اليسار واذا لم تكن البالتة مفتوحة فقم بفتحها من القائمة **Window > Tool Palette** أو الضغط على **Ctrl + 7** من لوحة المفاتيح حيث يظهر لك الصندوق كما في الشكل **ToolPalette**

٥- ومن خلال هذه الأداة على اليمين **Tool Palette** نلاحظ الأدوات العديدة فقم باختيار **Push Button**  وهي الأداة التي سوف نحتاجها في هذا الدرس وبعد اختيارها اضغط على أي مكان فارغ في مساحة المسرح **Stage** ثم اضغط بالزر الأيسر للماوس لكي ترسم الزر المطلوب ويجب ان تكون شاشتك قريبة الشبه بالشكل التالي:

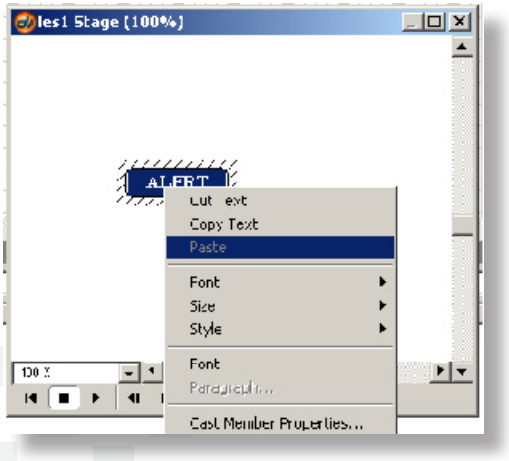
Tool Paleete



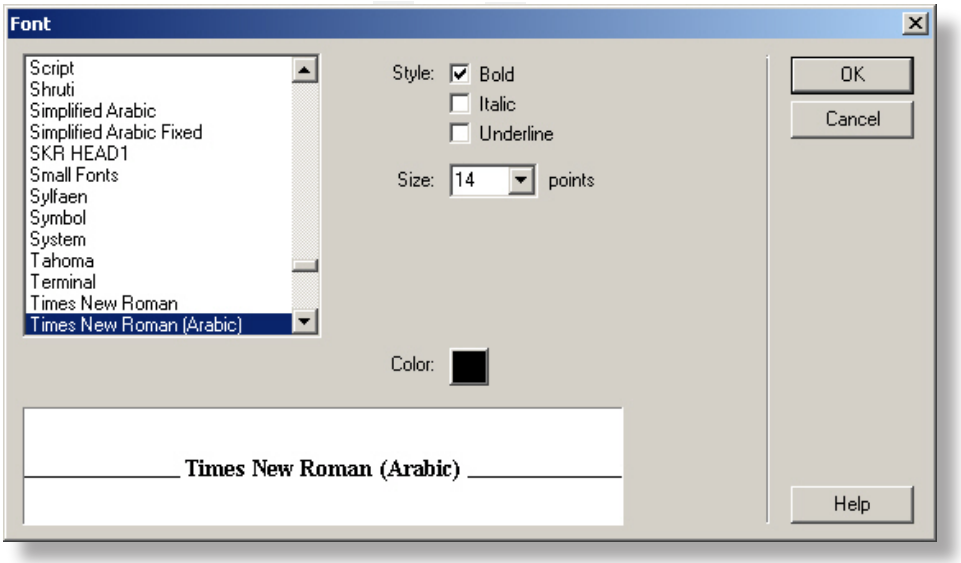
Stage

٦- ومن ثم قم بالنقر المزدوج على الزر الذي رسمته توا لكي تقوم بكتابة عنوان لهذا الزر من لوحة المفاتيح وأكتب **Alert** ثم اضغط الماوس خارج الزر على المساحة البيضاءالفارغة.

٧-اذا لم تتمكن من كتابة العنوان من المرة الأولى فاضغط ضغطا مزدوجا على الزر **Alert** لكي تتمكن من الكتابة داخله حيث ستلاحظ أن مؤشر الكتابة يومض كما يحدث تماما في برنامج الوورد وستلاحظ أن حجم الخط صغير جدا ولذلك فقم باختيار الخط (الفونت) بالزر الأيسر للماوس ثم اضغط بالزر اليمن للماوس ومن القائمة التي ستظهر لك قم باختيار **Font** كما ترى في الشكل التالي:



٨- قم الأن باختيار **Font** من القائمة فيظهر لك المربع التالي:

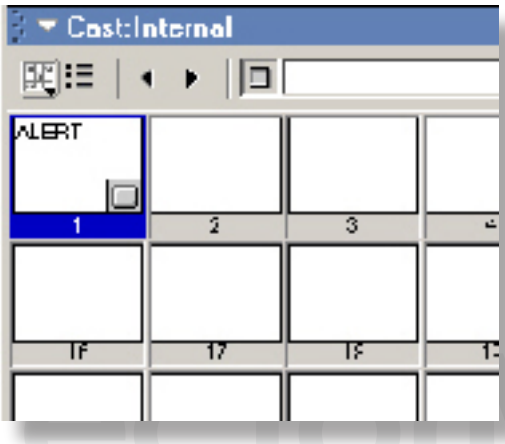


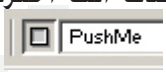
والأن تلاحظ وجود أنواع الخطوط على اليسار بينما هناك خصائص الخط المعتادة على اليمين فقم باختيار حجم أكبر لحجم الخط إذا أردت ذلك من القائمة كما يمكنك إختيار لون للخط وتلاحظ تغير حجم الخط والزر تلقائيا في نافذة المعاينة

٩- قم باختيار ايقونة الأعضاء **Cast**  من قائمة الأزرار في أعلى الشاشة.

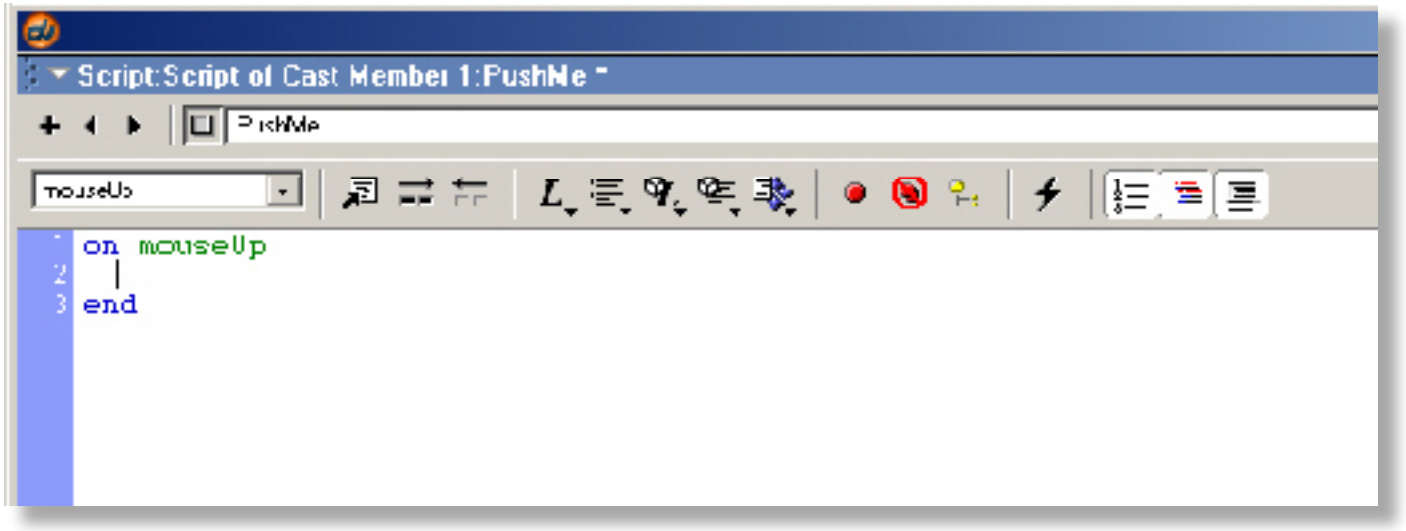
١٠-والأن قم باختيار العنصر **Alert** من الكاست باختياره بالماوس فقط بالزر الأيسر ولا تضغط ضغطا مزدوجا.

١١- الأن من الاختيار **Cast View Style**  من نافذة الكاست اضغط عليه لكي تبدل هيئة الكاست للهيئة الاعتيادية كما ترى في الشكل التالي:



١٢- الأن بعدما تأكدت انك اخترت الزر **Alert** من الكاست قم بكتابة اسم له في خانة الاسم التي تراها ذات خلفية بيضاء في نافذة الكاست 

فمثلا اكتب **PushMe** كإسم لهذا الزر والان إختر نافذة كتابة الكود لهذا الزر من خلال الزر **Cast member Script**  أو اضغط مفتاح **Ctrl + '** من على لوحة المفاتيح الا انني افضل الاختيار بالماوس مبدئيا وستلاحظ ظهور نافذة الكود **Script** الخاصة بهذا العنصر وهو الزر الذي قمنا بعمله منذ قليل كما الشكل التالي في الصفحة التالية الذي يمثل مقطعا من نافذة كتابة الكود:



وستلاحظ معلومات هامة تظهر في عنوان النافذة وهي جملة **Script of Cast member 1 : PushMe** وهي تعني ان هذه النافذة هي نافذة الكود الخاصة بالعضو رقم واحد في الكاست والذي اسمه **PushMe** وتلاحظ أسفل النافذة جملتان وهي

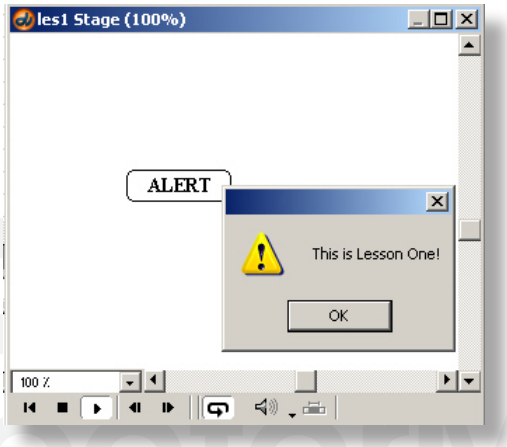
```
on mouseUp

end
```

وهنا سنتوقف لكي نشرح هذه الجمل:
فجملة **on mouseUp** تعني تعريف حدث معين وهذا الحدث هو افتراض أن مؤشر الماوس يضغط على الزر فهذا هو معنى **on mouseUp** فهي تعني اخبار دايركتور بالتحفز لعمل رد فعل معين اذا ضغط المستخدم الماوس فوق الزر كما نلاحظ وجود جملة **end** وهي هنا تعني نهاية هذا الحدث ولكن حتى الأن لا توجد اية احداث لكي يستجيب لها الدايركتور فتعريف الحدث **on mouseUp** يليه سطر فارغ ثم يليه جملة **end** وهنا مربط الفرس فالسطر الفارغ الذي تراه نصب عيناك هذا هو السطر الذي يجب أن تبدأ بكتابة الأحداث التي ترغب ان ينفذها دايركتور عند الضغط على هذا الزر وهنا قم بكتابة الأمر **Alert**"This is Lesson One!" وهو أمر يقوم بمجرد عرض رسالة تنبيه تحمل جملة **This is Lesson One!** والأن يجب أن يكون بالشكل التالي:

```
on mouseUp
  Alert"This is Lesson One!"
end
```

الان بعد انتهائك من كتابة الأمر اضغط الزر  وهذا الزر **Recompile Script** كل مهمته هو التأكد من صحة كتابة "نحو الأكواد" أي التأكد من صحة ادخال الأوامر بالنحو الصحيح **Right Syntax** وبالشكل الصحيح **Right Tide** وبعد أن تأكدت من ان كل شئ صحيح يمكنك اغلاق النافذة فدايركتور لن يظهر لك اية رسالة خطأ اذا ضغطت عليالزر السابق اذا لك يكن هناك خطأ فعلا وهذا معناه أن الكود مكتوب بطريقة صحيحة وهنا يمكن أن تغلق النافذة تمهيدا لتجربة الكود الجديد الذي قمت بكتابته والأن أغلق كل النوافذ المفتوحة أمامك ليظهر لك دايركتور فارغا من كل النوافذ ثم من شريط الأزرار قم باختيار زر المسرح **Stage**  فتظهر لك نافذة المسرح درجة مناسبة ومن شريط الأزرار قم بالضغط على الزر **play**  لكي تلعب التطبيق الذي انشأته الأن.. ومن ثم فقد عمل التطبيق وحان الوقت أخيرا لكي ترى نتاج عملك والأن اضغط الزر فورا لكي ترى ماذا سيحدث, والأن هل ترى رسالة التنبيه؟ وتلاحظ تغير حالة الزر!, ممتاز لقد دخلت عالم برمجة الدايركتور بلغة اللينجو لتوك!, يجب أن تبدو شاشتك مماثلة تقريبا للشكل التالي:

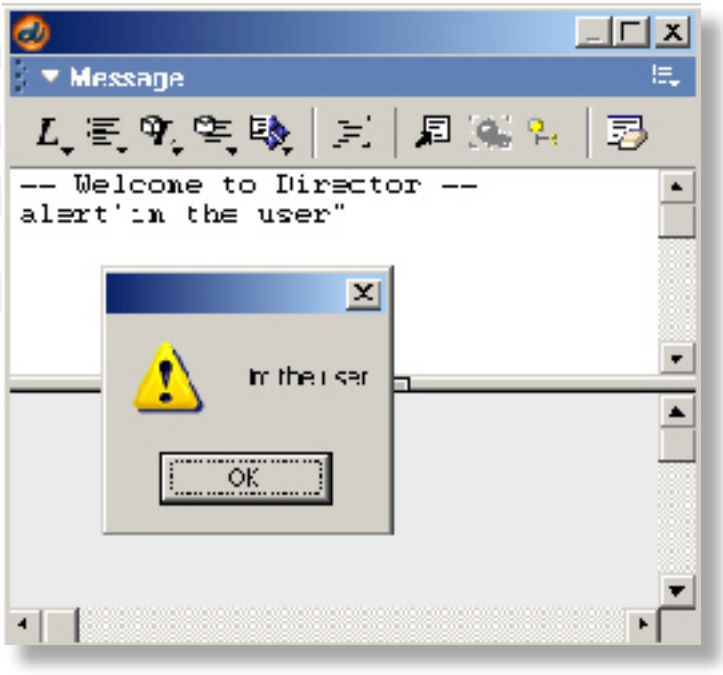


والأن بعد التأكد من صحة عمل الزر قم بايقاف التطبيق بالضغط على الزر  من خلال شريط الأوامر أمامك, اذا كان هناك أي خلل في طريقة عمل التطبيق فيجب أن تراجع صحة خطوات عملك بالعودة الى الصفحات السابقة والتأكد من أن كل شئ يعمل على ما يرام, وهذا الحدث الذي استجاب دايركتور له يسمى في لغة اللينجو المحدد **Handler** فالمحددات **Handlers** في دايركتور لها سمات معينة ولها مكتبة خاصة بها ويمكن للمستخدم أيضا أن يقوم بكتابة محدداته الخاصة به, والأن وقد رأينا كيفية استجابة دايركتور للأحداث, وهذا مثال بسيط وهو النوع الأول من الاستجابة للأحداث فلغة اللينجو في دايركتور هي لغة من نوعية لغات **OOP** أي لغة مقادة بالأحداث **Object Oriented Programming** ويجب أن تفهم ذلك جيدا وليس الضغط بالزر الأيسر للماوس هو الحدث الوحيد الذي يمكن للغة اللينجو الاستجابة له بل هناك أحداثا اخرى كثيرة كالضغط على لوحة المفاتيح أو الضغط بالزر الأيمن للماوس أو أحداث أخرى غير مرئية كانتظار دايركتور لمؤقت زمني معين أو انتظار مدخل معين من المستخدم أو انتظار ناتج عملية حسابية معينة ولذلك فالمحددات **Handlers** يمكنك أن تفهمها بأنها محددات الأحداث **Events handlers** فالمحدد يهيئ الدايركتور للاستجابة لحدث معين وعند فهمك المحيط للغة فستستخدمها كثيرا وربما تنشئ محدداتك الخاصة والتي طبعا ستكون مجموعة من الأكواد التي تريدها أن تقوم بتنفيذ أمر معين كما سترى وفي الصفحات القادمة سنتعرف أكثر على أنواع من المحددات الخاصة باللينجو وطريقة الاستجابة التي تفعلها اللغة مع دايركتور.

إستخدام نافذة التراسل Message window

الأن ستتعلم طريقة جديدة لادخال وتجربة الأكواد مع دايركتور وهي طريقة "نافذة التراسل" **Message Window** وهي نافذة متحفزة لادخال الأوامر تذكرنا بنظام التشغيل دوس القديم حيث تنتظر هذه النافذة أي كود تكتبه وبمجرد الضغط على مفتاح الادخال **Enter** فهي تستجيب فورا بدءا من تنفيذ أوامر مباشرة أو تخزين قيم في ذاكرة الدايركتور تمهيدا لاختبارها من خلالها ايضا ولذلك فمبرمجين دايركتور باللينجو يعتبرون هذه النافذة في غاية الأهمية لأنها طريقة سهلة لاختبار وتجربة الأكواد بدلا من القيام بعدة خطوات كما فعلنا سابقا ويمكنك تجربة ذلك بنفسك حالا, والأن افتح نافذة التراسل **Message Window** بالضغط على مفتاحي **Ctrl+m** أو من نافذة **Window** ثم اختيار **Message Window** وكما ترى فتعلم المختصرات مفيد ويوفر الوقت والمجهود خصوصا والدايركتور برنامج ملئ بالنوافذ.
والأن تظهر لنا نافذة **Message Window** كالتالي:

وكما تلاحظ ظهور عبارة **Welcome to Director** وهي عبارة ترحيبية للنافذة, ويمكن البدء فورا في إدخال الأوامر الى النافذة وملاحظة طريقة الاستجابة الى الأوامر والأن أدخل الأمر
alert "im the user"
من خلال النافذة واضغط مفتاح الادخال ولاحظ ماذا حدث؟ فعلى الفور فقد استجابت نافذة التراسل وظهر أمامك نافذة التنبيه وبداخلها الرسالة **im the user** كما بالشكل التالي:



وأمر **alert** قد سبق أن استخدمناه منذ قليل في المثال السابق بتنفيذ الضغط على الزر الأيسر للماوس ونعلم أن الفائدة الوحيدة لهذا الأمر هو اخراج نافذة اخبارية تنبه المستخدم برسالة تعرضها عليه كما رأينا في المثال السابق والحالي ولكنك تلاحظ الاستجابة الفورية التي استجابت لها نافذة التراسل للمستخدم فور ضغطه على زر الادخال في لوحة المفاتيح فهذه النافذة نموذجية جدا لادخال الكود على الطاير كما يقولون ولتجربة رد فعل دايركتور فورا في الاستجابة للاحداث فالحدث التي تستجيب له هذه النافذة مباشرة بعد ادخال الكود هو حدث الضغط على مفتاح الادخال **enter** كما لاحظت عزيزي القارئ, والأن قم بتجربة الأمر **beep** ثم قم بالضغط على مفتاح الادخال **Enter** وتذكر دائما أنه لي تستجيب لك نافذة التراسل فعليك دائما ضغط مفتاح الادخال بعد كل أمر تكتبه .

كل لغة برمجة من اللغات المعروفة تحتوي على عناصر أساسية عندما تتحدهذه العناصر معا فانها تكون لنا ما يسمى بالكود البرمجي أو خوارزمية البرنامج كما شرحنا من قبل والعناصر التي تتكون منها لغة اللينجو هي:

- الأوامر **Commands** :

هي التي تخبر دايركتور بأنه يجب عليه القيام بعملية ما أو تنفيذ أمر ما كالأمر "اذهب" **go to** الذي يأمر دايركتور بالذهاب لموضع معين في البرنامج.

- الوظائف **Functions** :

هي محددات القيم فمثلا محدد التاريخ **Date** يحدد التاريخ ويحمل قيمة معينة وهي تمثل التاريخ.

- الخصائص **Properties**:

هي عبارة عن معاملات مخصصة لعنصر ما فمثلا الخاصية **the color depth** وهي تعني عمق الدرجة اللونية هي خاصية محددة لشاشة الكمبيوتر فقط وقد تكتب الخصائص للعناصر مفصولة بنقطة كخاصية لون الخبر للعنصر التي تكتب بالينجو **sprite(1).ink** وهكذا.

- المعاملات **Operators**:

وهي معاملات منطقية حسابية غالبا كمعاملات الضرب والطرح أو المعاملات العلاقية مثل أكبر واصغر أو يساوي

- الثوابت **Constants**:

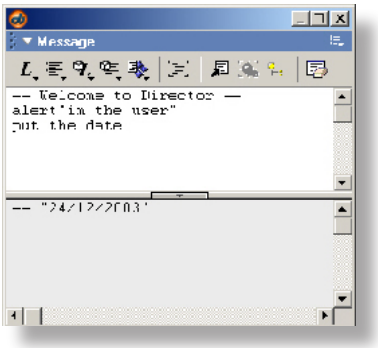
وهي الثوابت التي لا يمكن تغييرها ابدا لانها من سمات العناصر أو المنطق مثل كلمة **TRUE** التي تعبر دائما عن قيمة الرقم **1** بينما كلمة **FALSE** تعبر دائما عن قيمة الرقم **0** في الدايركتور كما سنرى لاحقا.

- المتغيرات **Variables**:

وهي القيم المتغيرة التي يمكن اعطاؤها لعناصر محددة وأحيانا تمثل المتغيرات قيم افتراضية لمحددات افتراضية كأن نقول رياضيا بأن س = ٥ فنحن هنا نفترض ذلك وربما قلنا أن ص = ٥ * س وهكذا والمتغيرات لا تشمل فقط الأرقام والقيم بل تمتد لتشمل الأسماء فهناك متغيرات رقمية وأخرى أسميه وسوف نتعرض للمتغيرات بالتفصيل لاحقا باعتبارها أهم ما يجب أن يستوعبه مبرمج اللينجو تماما قبل الشروع في إنتاج برامجه الخاصة.

والأن سنقوم بالتدرب سويا على التعرف الى عناصر لغة اللينجو عمليا عن طريق استخدام نافذة التراسل التي قمنا بشرحها منذ قليل, والأن في نافذة التراسل **Message Window** اكتب الأمر التالي:

put the date



وستلاحظ أن النافذة استجابت فورا بعد ضغط **Enter** وقدمت لك التاريخ في السطر التالي, والأن جرب الأمر:

put the time

وكما رأيت فان النافذة كتبت أيضا الوقت الصحيح!, والسؤال الأن كيف عرف الدايركتور التاريخ والوقت الصحيحين؟ فكما قلنا سابقا فهذه من وظائف الدايركتور **Functions** فهذه أوامر وظيفتها عرض الوقت والتاريخ من معلومات الويندوز نفسه.

والأن اكتب الأمر التالي من نافذة التراسل:

put the colordepth

والأن من المفترض أن يكتب لك دايركتور قيمة عمق اللون كما ضبطها تماما في كارت الشاشة لديك وهذا الأمر من خصائص لغة اللينجو **property**

والأن اكتب الأمر التالي:

5+put 5

والنتيجة طبعا ستكون **10** فالأمر **put** هنا صار كانه اجمع **5+5** ويمكنك تجربة معاملات حسابية أخرى كالطرح والقسمة وهذا يعتبر أمر مباشر ويحوي معاملات منطقية كما شرحت سابقا, والأن اكتب الأمر التالي في نافذة التراسل:

put 5<9

والنتيجة التي ستشاهدها هي الرقم **1** فما معنى هذا؟ لقد أخبرت الكمبيوتر بأن يضع ناتج أن الرقم **5** أصغر من الرقم **9** وهذه ليست عملية حسابية انما معامل منطقي ولذلك كان الرد من الكمبيوتر بأن المعامل المنطقي حقيقي والرقم 5 أصغر فعلا من الرقم **9** لذلك فالناتج **1** يمثل القيمة **TRUE** أي أن المعامل المنطقي حقيقي بينما لو كتبت أنت الأمر التالي :

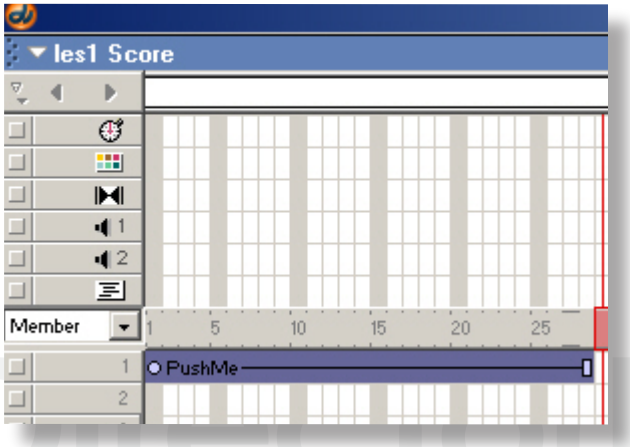
put 5>9

فهذا معامل منطقي غير حقيقي لأنك تخبر الكمبيوتر بأن الرقم **5** أكبر من الرقم **9** وهذا غير حقيقي وغير منطقي طبعا ولذلك فالرد هو الرقم 0 أي أن نتيجة المعاملة المنطقية هي **FALSE**وهذا هو المعنى الرمزي للثوابت **Constants** التي قمنا بشرحها منذ قليل.

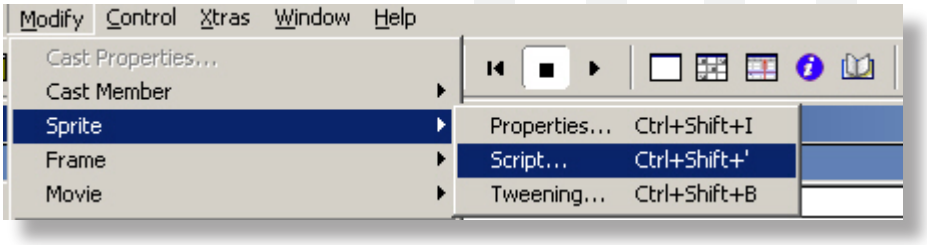
الاستجابة في لغة اللينجو وطريقة استجابتها للأحداث من أهم المواضيع التي يجب على أي مبرمج مبتدئ أو متقدم أن يكون ملما بها تماما فيجب أن يعلم المبرمج متى يقوم دايركتور بتنفيذ الأوامر المعطاة له ؟, فهل يبدأ دايركتور بتنفيذ أوامر **Movie Script** أم يبدأ في تنفيذ أوامر **Cast member script** أم ينفذ **Score Script** وهذه المعلومات الهامة يجب أن تكون ملما بها حتى تستطيع أن تكتب الكود المناسب في المكان المناسب مع الأخذ بعين الاعتبار أولوية التنفيذ لأن ذلك سيؤثر في سرعة تطبيقك من ناحية ومن ناحية أخرى سيؤثر في طريقة ظهور عملك, وفي الأمثلة التالية سنقوم بتبسيط هذا المفهوم لأقصى درجة وفي النهاية ساقدم جدول سهل الاطلاع ليكون بحوزتك وقت الحاجة ينبهك الى أوقات تنفيذ الأوامر.

مثال:

الأن قم بتصميم مسرح مشروع بمساحة **320X240 pixel** كما تعلمت مما سبق ثم أضف زر **button** كما فعلت سابقا في أي مكان وقم بتسميته **alert** كما تعلمت سابقا, واذا كنت قمت بتخزين المشروع سابقا فعليك بتحميل الملف الذي قمت بتخزينه وعموما قم بفتح نافذة خط الزمن**Score** لهذا العنصر بالضغط على الأيقونة  أو الضغط على **Ctrl+4** ثم اختر الزر من على خط الزمن كما الشكل التالي:



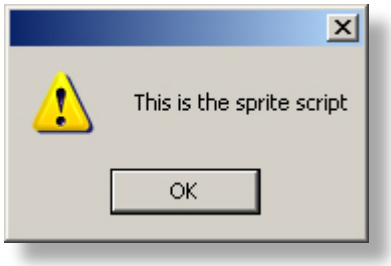
ثم اضغط **Ctrl+shift+'** من على لوحة المفاتيح لفتح نافذة الكود او اختار الكود من قائمة **modify** ثم **sprite** ثم **Script** كما الشكل التالي:



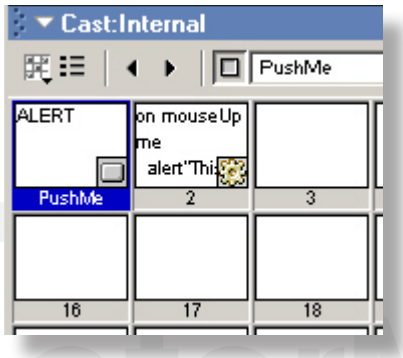
ثم اكتب الكود التالي داخل النافذة **Behavior Script** التي أمامك..

```
on mouseUp
  alert "this is the sprite script"
end
```

وطبعا ستلاحظ أن الكود **on mouseup** و**end** مكتوبين فعلا لذلك فقم بكتابة السطر الأوسط فقط ولا بأس من اعادة كتابتهم كلهم ولكن لا حاجة لك لذلك!, والأن أغلق النافذة واضغط على الزر ثم لاحظ ما يحدث, فسوف تلاحظ ظهور رسالة تنبيه داخل نافذة صغيرة وتعرض الرسالة **this is the sprite script** كما الشكل التالي:



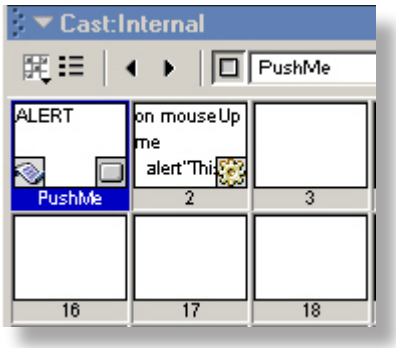
والأن توجه الى نافذة الكاست **Cast window** وهي بالاسم **Internal Cast** ثم لاحظ شكل الأيقونات التي تراها أمامك فيجب أن تبدو مقاربة للشكل التالي:



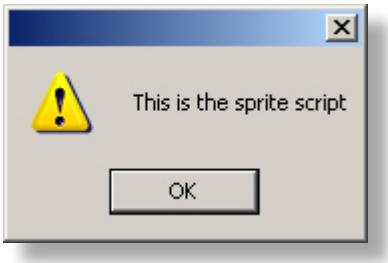
وهنا تلاحظ وجود أيقونتين الأولي من اليسار للزر الذي أنشأته والثانية للكود الذي قمت بكتابته لتوك والأن لا تغلق هذه النافذة بل قم باختيار الزر الموجود على اليسار ثم اضغط مفتاحي **Ctrl+'** من على لوحة المفاتيح أو اضغط الأيقونة  من النافذة نفسها واكتب داخل النافذة الأوامر التالية:

```
on mouseUp
  alert "this is the cast script"
end
```

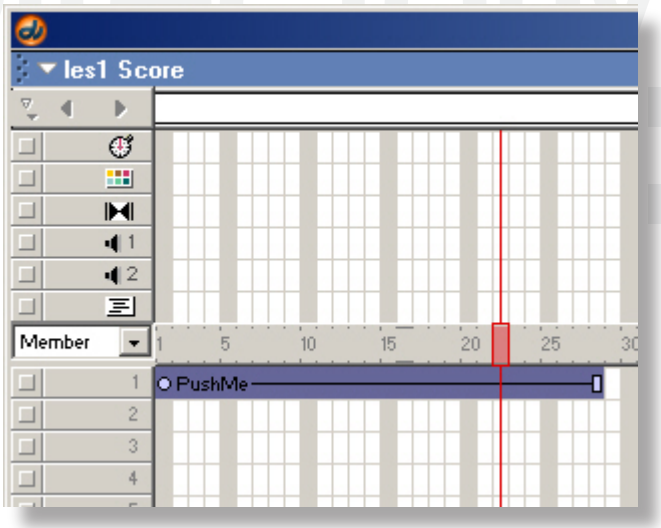
وكما تلاحظ فانك لن تضطر الا لكتابة السطر الأوسط فقط لأن باقي الأسطر موجودة فعلا, والأن نافذة الكاست لديك ستبدو كالتالي:



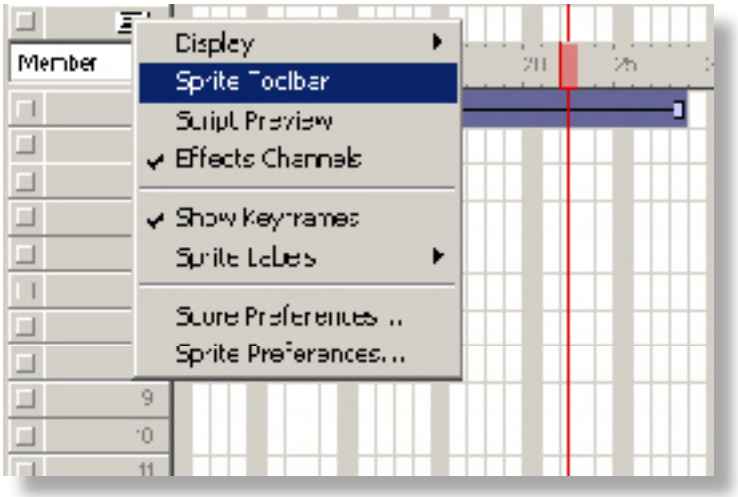
والأن أصبح لديك أمر واحد مكرر مرتين والاثنان مرتبطان بالزر فالأمر الأول يجعل الزر يظهر رسالة **this is sprite script** وهو مكتوب من خلال **behavior script** والثاني مكتوب في نافذة **Script of cast member 1** ويظهر رسالة **this is cast script** فكل رسالة من الاثنين تصف علاقة الكود بالعنصر "الزر" وطبعا لاحظت أنه في الحالة الأولى فقد قمنا باختيار كتابة الكود من خلال خط الزمن بينما في الحالة الثانية قمنا بكتابة الكود من خلال نافذة الكاست نفسها ولا تقلق فاختلاف طريقة كتابة هذه الأكواد لا تؤثر على العنصر ولكنها ميزة في الدايركتور وتابع معنا لكي تفهم المقصد, الأن قم بلعب المشروع بالضغط على الزر **Enter** من أقصى يمين لوحة المفاتيح أو الضغط على زر  وأنا أذكرك فحسب واحسبك تعلم طبعا!, والأن ماذا حدث؟ لقد تكررت نفس الرسالة الأولى واستجاب دايركتور للكود الأول الذي قمنا بكتابته وظهر نفس الشكل التالي:



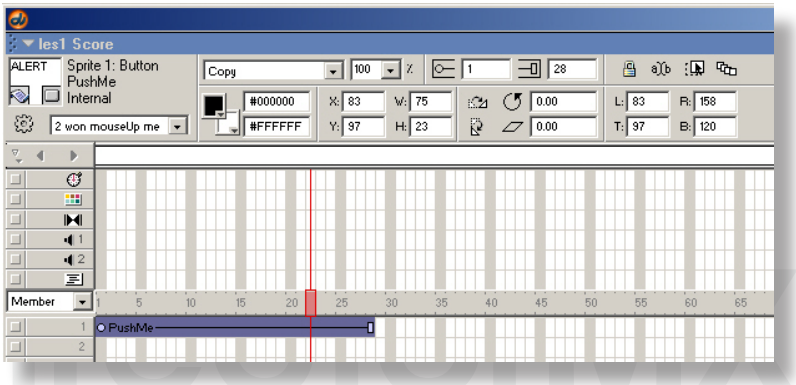
وسبب حدوث ذلك أن دايركتور يقوم بتنفيذ أوامر أعضاء الكاست على خط الزمن **Score Sprites** أولا قبل تنفيذ كود أعضاء الكاست المرابطين **Cast Members** بمعنى أن طالما أعضاء المشروع موجودن في الكاست وغير مستخدمين في خط الزمن **Score** فان دايركتور ينفذ الأكواد المرتبطة بهم أولا والتي تسمى **Cast member scripts** بينما اذا كان أعضاء المشروع أو بعضهم متواجدا في خط الزمن **score** فان دايركتور يلتفت الى أكواد الأعضاء "ان وجدت" المرتبطة بهم والتي تسمى **Behavior Script** وبعض المبرمجين يطلقون عليها **Score Sprite Script** ولكن المسمى الأول هو الأصح والأن قم باظهار شريط خصائص الأعضاء على خط الزمن الذي يعرف بالاسم **Sprite ToolBar** اذا لم يكن ظاهرا لديك فالشكل التالي يوضح **Score** بدون ظهور شريط الأعضاء كما الشكل التالي :



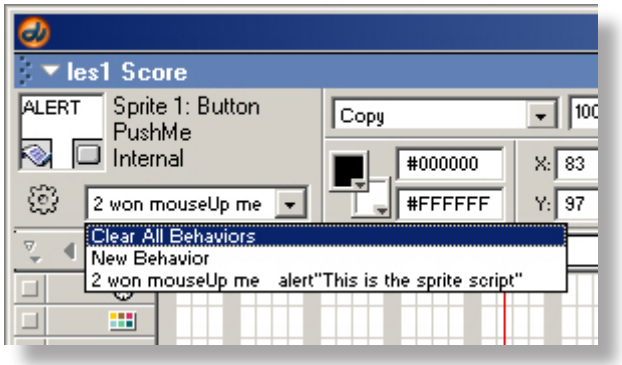
ولاظهار شريط خصائص الأعضاء **Sprite ToolBar** يجب الضغط بالزر الأيمن على الأيقونة  فتظهر القائمة التي تراها في الشكل التالي والأن قم بوضع علامة "صح" على **Sprite ToolBar** كما هو موضح في الشكل في الصفحة التالية.



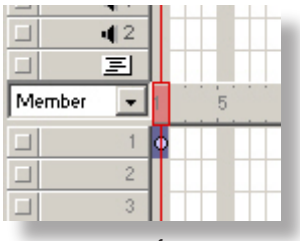
حيث يظهر شريط خصائص أعضاء خط الزمن **Sprite Toolbar** كما ترى في الشكل في الصفحة التالية:



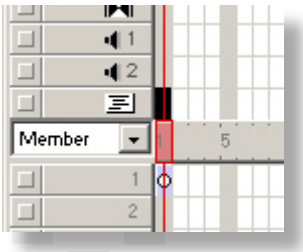
والأن قم باختيار العضو رقم "1" في خط الزمن , وهو العضو في القناة واحد ويسمى **Sprite 1** كما ترى الوصف في **Sprite ToolBar** وكما تلاحظ فقد أصبح شريط خصائص الأعضاء "فعالا" **Active** ويمكن الاختيار منه فقم باختيار الاختيار **clear all behaviors** من القائمة المنسدلة على أقصى الشمال كما ترى في الشكل التالي:



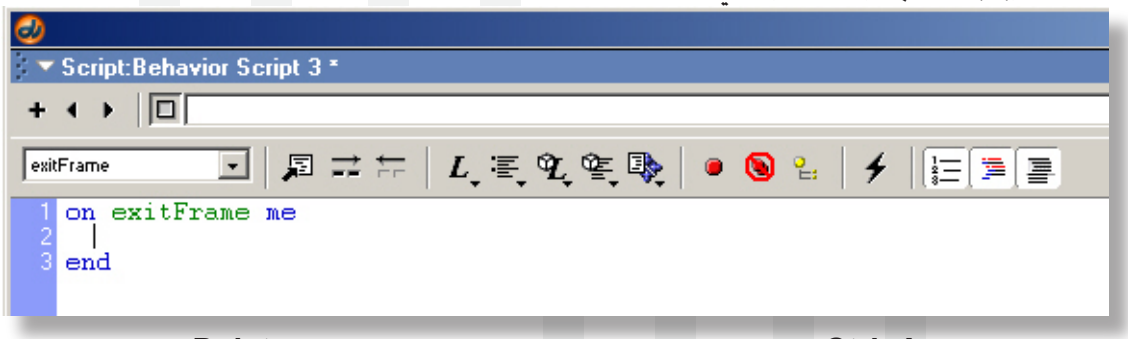
وذلك سيؤدي الى مسح كل الأكواد المرتبطة بالعنصر المختار على خط الزمن فقط أي أن كود الكاست سيظل كما هو وسيختفي فقط ال **behavior script** وتذكر ان ارتباط الكود بالعنصر سيختفي لكن سيظل الكود مرابطا في الكاست, والأن اضغط بالماوس بالزر الأيسر واسحب العضو على خط الزمن من طرفه باتجاه اليسار حتى يحتل مكان كادر واحد فقط كما الشكل التالي:



والأن بنفس الزر الأيسر للماوس اضغط ضغطا مزدوجا على أول كادر **frame 1** من خط الزمن وهو المربع الأسود الذي تراه في الصوره التالية:



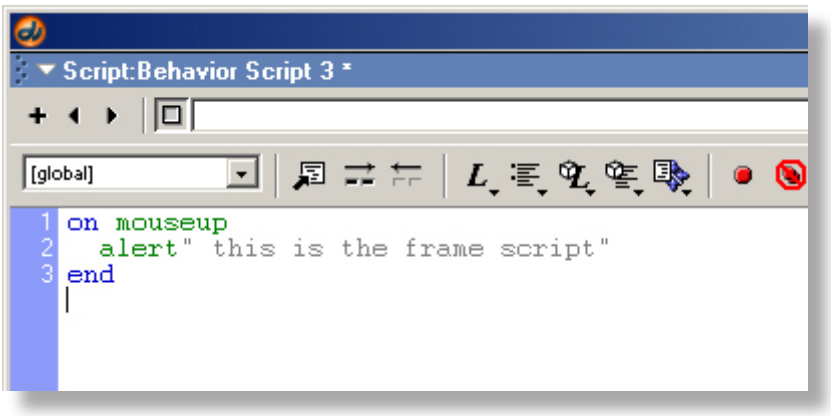
وبعدها ستشاهد نافذة أمامك كما النافذة التالية:



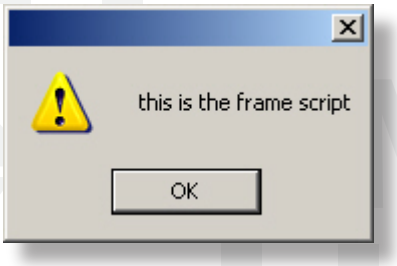
والأن قم بالضغط على مفتاحي **Ctrl+A** لكي تقوم باختيار النصوص واضغط مفتاح **Delete** من على لوحة المفاتيح ثم اكتب النص التالي في المساحة التي أصبحت خالية أمامك:

```
on mouseup
  alert" this is the frame script"
end
```

والأن يجب أن تبدو شاشتك كما الشكل التالي تقريبا:

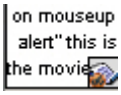


والأن أضغط عليالأيقونة الذي أمامك لكي تقوم بالتأكد من الادخال الصحيح للكود الذي كتبته وتذكر دائما أن تستخدم هذه الأداة عند كتابة أي كود في أي مكان والأن أظهر نافذة المسرح **stage** بالضغط على **Ctrl+1** اذا لم تكن ظاهرة أمامك أو على أيقونة ثم اضغط الأيقونة لكي تلعب المشروع أو اضغط المفتاح **enter** من على أقصى يمين لوحة المفاتيح اذا أردت فكلاهما يقوم بنفس العمل واضغط على أي مكان فارغ أمامك من نافذة المسرح عدا الزر ولاحظ ماذا يحدث؟ لقد ظهرت لك رسالة تنبيه كالشكل التالي:



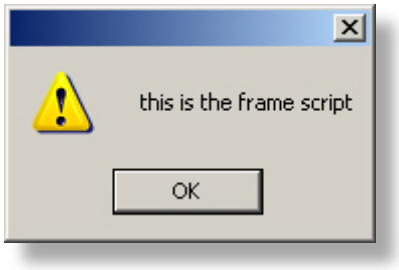
هل لاحظت ماذا حدث؟ لقد ظهرت نافذة الأمر **alert** وهي تخبر رسالة **this is the frame script** كما ترى , فانتبه جيدا.. والأن اضغط **ok** لاغلاق نافذة **alert** أمامك وقم بايقاف المشروع بالضغط على مفاتيح **Ctrl+Alt+.** أو الضغط على أيقونة و الأن قم باختيار نافذة الكاست **Internal Cast** ثم اضغط على مفاتيح **Ctrl+Shift+U** والأن ظهرت أمامك نافذة **Movie Script** وهي التي عرفناها من قبل بأنها أحد أنواع الكود **Script Type** وهي الكود المستجيب للمشروع والذي تؤثر الأوامر التي تكتب داخله على المشروع ككل وهنا سنكتب داخلها التالي:

```
on mouseup
  alert" this is the movie script"
end
```

وتلاحظ في نافذة الكاست ظهور أيقونة الكود بعد كتابته 

وهي تشير الى أن هذا الكود من نوعية **Movie Script** ودايركتور يتميز بهذه الميزة حيث يمكنك بالنظر التعرف الى أنواع الكود الموجودة في الكاست بمجرد النظر وقد سبق أن تعرفت على النوعين السابقين من قبل والأن أغلق نافذة ال **Movie Script** وقم بلعب المشروع وكفانا تذكيرك بالطريقة التي تقوم بتشغيل المشروع بها فقد احترفتها على ما أظن.

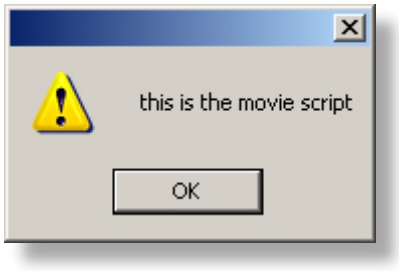
والأن اضغط على الزر , وتلاحظ أن رسالة **alert** نفسها التي تخبر **this is the cast script** والأن اضغط على أي مكان فارغ غير الزر الذي أمامك في الملعب الأبيض الذي تراه, والأن مفاجأة!, تلاحظ ظهور رسالة **this is the frame script** كما ترى في النافذة التالية:



وكما تلاحظ فان دايركتور قد أهمل كود المشروع **Movie Script** وقام بتنفيذ كود خط الزمن **Score Script** أولا والمغزى من هذه التجربة هي أن تعلم أن دايركتور يهتم أولا بالأكواد على خط الزمن بدءا من كود العضو **Cast script** أو يعرف بالاسم **Sprite Script** نهاية بكود الكادر **frame script** قبل أن ينتقل الى تنفيذ **Movie script** وهي كود المشروع وكثيرين من المبرمجين يعتقدون بأن كود المشروع **Movie Script** ينفذ أولا وهذا خطأ شائع يؤدي لمشاكل في مشاريعهم لا يلاحظونها الا فيما بعد, والأن لنعد لكي نكمل تجربتنا المثيرة للاهتمام.

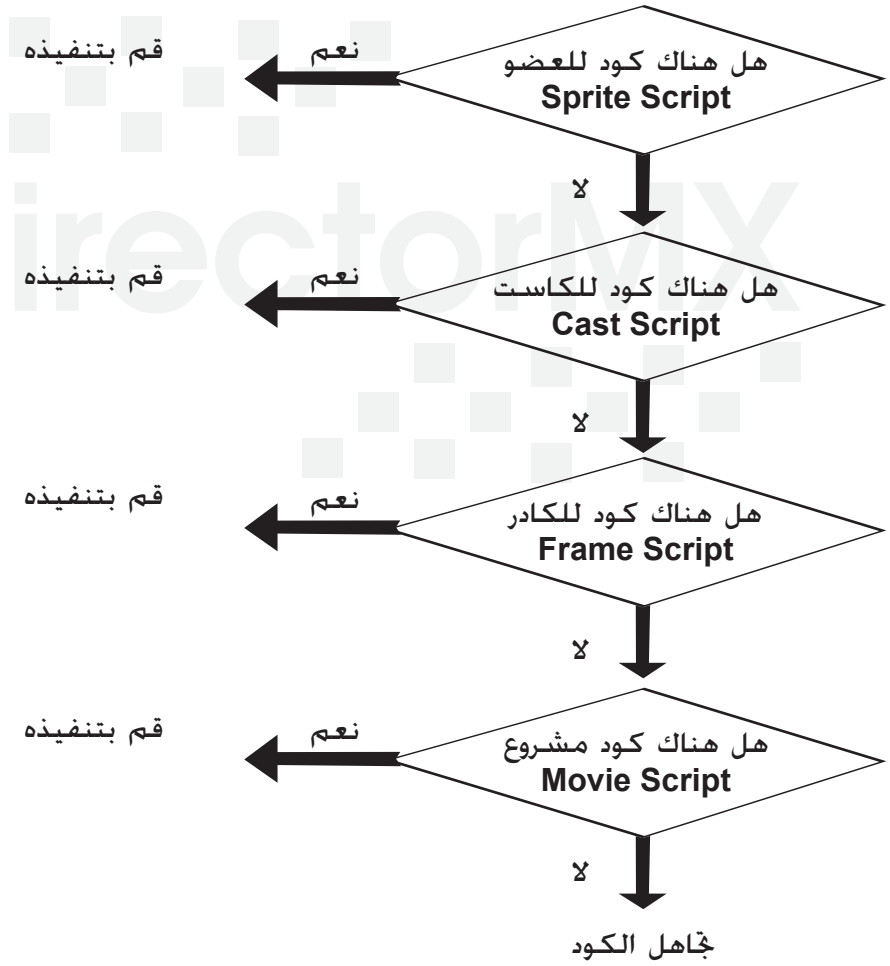
الأن وكما فعلنا سابقا قم بازالة الكود من الكادر الأول **frame 1** وذلك بالاختيار من صندوق شريط خصائص الأعضاء **Tool Bar Sprite**, فقم أولا بضغط الزر الأيسر للماوس ضغطة واحدة على الكادر الأول على الأيقونة الزرقاء للكود الذي قمت لتوك بكتابته على خط الزمن ثم اختار **clear all behaviors** .

وهنا تلاحظ اختفاء الكود ذو الأيقونة الزرقاء الذي كان موجودا في الكادر الأول **frame 1** من خط الزمن وهنا يتبقى لدينا كود الكاست المرتبط بالزر والذي لم يمسح وكود المشروع وقم بلعب المشروع  والأن اضغط على الزر ثم اضغط **ok** لكي تغلق نافذة التنبيه ثم اضغط أي مكان فارغ على المسرح **stage** ثم أخبرني ماذا حدث؟ .. أه. لقد ظهرت الرسالة التي في الشكل التالي:



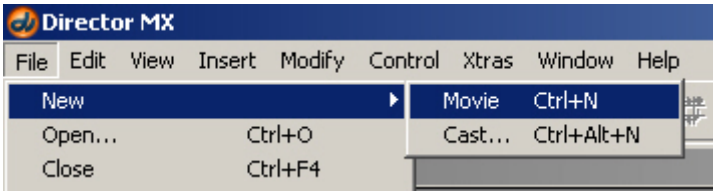
وهكذا فقد فهمت المغزى؟, فكود المشروع **Movie Script** لم ينفذ الا عندما أزلنا كود الكادر **frame Script** فهو يأتي في المرتبة الثانية في التنفيذ, والأن ما هي الفائدة العظمى من هذا الاختبار الذي أجريناه الأن على الدايركتور؟, ان الفائدة تكمن في معرفة تسلسل تنفيذ الأحداث والأوامر في الدايركتور فنحن أو أنا وأنت نلعب دور المبرمجين للدايركتور فنحن ان كنا قد قبلنا هذا الدور فنحن نتوقع استجابة معينة ومنطقية من البرنامج والأن يمكنك من دراسة التسلسل المنطقي للبرمجة أن تقوم باصطياد الأخطاء بسهولة **Syntax errors** وتقوم بعملية تحسين وتتبع وتطوير ومعالجة للكود الذي قمت بكتابته في أي مكان وهي مرحلة تسمى الـ**Debugging** وهي مرحلة نهائية تتم لتحسين أداء الكود بتتبع مسيرته وتتبع طريقة عمله وهذه ما هي الا مقدمة بسيطة لتتبع الأحداث ولكنها اساسية جدا ويجب أخذها بعين الاعتبار, والأن سأقدم هنا رسما توضيحيا وملخصا لعملية تسلسل تنفيذ الأحداث في الدايركتور والتي عرفناها بأنها طريقة الاستجابة في لغة اللينجو, وهكذا انتهت الدروس الأساسية في لغة اللينجو.

ملخص طريقة الإستجابة في لغة اللينجو



لكي نبسط الأمر أكثر فلنذكر أمثلة عمليه, فقد ذكرت من قبل أن الضغط على زر الماوس هو حدث والضغط على زر في لوحة المفاتيح هو حدث أيضا وكل ما يجب أن يكون نتيجة فعل معين يطلق عليه حدث **Event** وله محدد **Handler** , ولذلك فعندما نريد أن نخبر الدايركتور بأننا نتوقع رد فعل معين من جراء ضغط المستخدم على الزر الأيسر للماوس فإننا نكتب **On mouseUp** فالاجراء **On** هو تعريف للمحدد **mouseUp** بينما الحدث نفسه فهو سيكون مجموعة الأوامر والإجراءات التي سينفذها دايركتور عندما يقوم المستخدم بالنقر على الزر الأيسر للماوس وهو ما نطلق عليه **Event** وهذا على سبيل المثال... ويمكنك عمل تجربة بسيطة قبل الإنطلاق في تفاصيل أكبر.

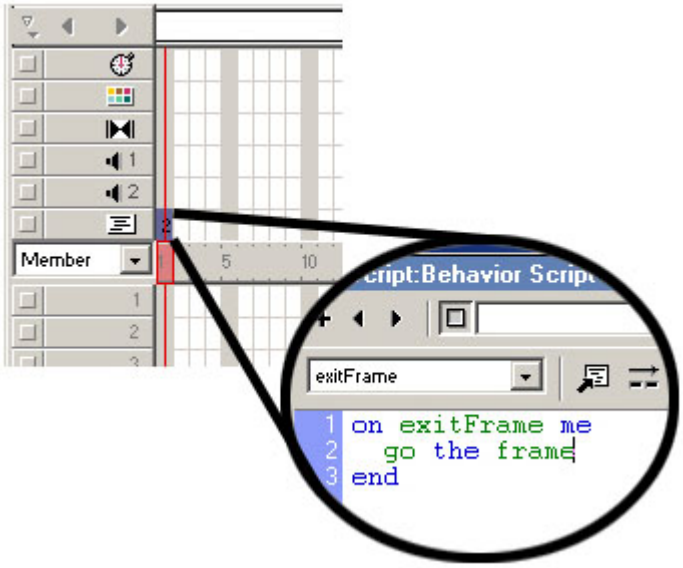
١. أنشئ مشروعا جديدا في الدايركتور بالضغط على **CTRL+N** أو كما الشكل التالي من قائمة **File**



٢.من خلال نافذة الأعضاء **Cast Window** أنشئ كود مشروع جديد **Movie Script** وذلك بالضغط على مفاتيح **CTRL+SHIFT+U** أو **CTRL+0** ثم اكتب بداخله السطور التالية:

```
on mouseUp me
  Beep
end
```

٣.إذهب للكادر رقم واحد في نافذة خط الزمن **Score Window** ثم انقر نقرا مزدوجا لكي تفتح نافذة الكادر الخاصة به والتي ستلاحظ بداخلها المحدد **On exitFrame me** وأمر **end** ومن ثم فاكتب في منتصفهم الأمر **go the frame** وهو الذي يأمر الدايركتور بالتوقف عند هذا الكادر,وهذه الخطوة ضرورية لتجنب توقف المشروع لدي بدء تشغيله من المستخدم كما الشكل التالي:



٤. الأن إفتح سماعة الكمبيوتر المتصلة بها وإضبطها على مستوى صوت معقول ثم قم بتنفيذ البرنامج بالضغط على **CTRL+ALT+1** أو بإختيار ▶ من نافذة الأدوات الرئيسة للبرنامج.

٥.تأكد من أن المسرح ظاهرا أمامك **Stage** أو إضغط أيقونته ▭ لكي يظهر.

٦.الأن إضغط في أي مكان فارغ على المسرح ؟ ماذا تسمع؟ انك تسمع صوت ما وهو نتاج تنفيذ الأمر **Beep** فسماع الصوت هو نتاج تنفيذ الأمر الذي قمت من أجله بالضغط بالزر الأيسر للماوس على المسرح, لأنك بدون هذا الضغط فلن تسمع الصوت أظن أنك فهمت ما أقصده الأن.

هناك ١٥ محدد قياسي في الدايركتور للأستجابة للأحداث, وكل إصدار جديد من دايركتور فإن الشركة المنتجة تضيف له الكثير من المحددات الجديدة وفق آخر المستجدات والتطوير ولكن هناك خمسة عشر محدد قياسي ويستخدمون بكثرة من قبل المبرمجين وأضف الى ذلك فالمستخدم قادر على تصميم عدد لا نهائي من المحددات الخاصة به كما سنر ى لاحقا, وقبل إستعراض الخمسة عشر محددا القياسيا في الدايركتور فيجب أن نراجع معلوماتنا حول المحددات, فالمحدد **Handler** هو وظيفة إنتظار **Wait** تنتظر من المستخدم رد فعل معين؟ أو تنتظر من المبرمج إشارة اتصال **Call Sign** كما سنرى لاحقا, وإشارة الإتصال قد تكون على شكل إستجابة تفاعلية بالماوس مثلا أو عن طريق الضغط على لوحة المفاتيح؟ أو قد تكون إتصال فعلي من داخل البرنامج ذاته, فعند تعمقنا أكثر في البرمجة باللينجو ستكتشف بأنك تستطيع الربط بين كودين منفصلين عن طريق إشارة اتصال بينهم, فيمكنك على سبيل المثال أن تصمم الكود الخاص بك في كود المشروع **Movie Script** ولكن في الوقت ذاته تجعل نقرة ماوس على زر ما على المسرح يقوم باستدعاء هذا الكود وتنفيذه كما سنرى, وخلاصة القول بأن الأحداث في دايركتور **Events** وهي غالبا ما تكون عبارة عن مجموعة الأكواد أو الشفرة التي تقوم بتصميمها للقيام بشئ معين هي أحداث مبنية على إنتظار رد فعل من المستخدم أو من إشارة اتصال.

جدول المحددات القياسيةStandard Events Handlers

تعريف المحدد	الاسم الكودي للمحدد
محدد ضغط الزر الأيسر للماوس	mouseDown
محدد ضغط الزر الأيسر للماوس ثم رفع الماوس عن الزر	mouseUp
محدد دخول مؤشر الماوس لمنطقة معينة	mouseEnter
محدد خروج مؤشر الماوس من منطقة معينة	mouseLeave
محدد بقاء مؤشر الماوس في منطقة معينة	mouseWithin
محدد ضغط الزر الأيسر للماوس ثم إبعاد المؤشر مع تحرير الزر	mouseUpoutside
محدد بداية ظهور العنصر على المسرح Stage	beginSprite
محدد التحضير لظهور كادر معين حيث من الممكن أن تسبقه أحداث أخرى	prepareFrame
محدد الدخول الى كادر معين فمن المفترض أن تنفذ أحداث معينة أثناء دخول الكادر	enterFrame
محدد التحضير لحدث معين قبل أن يقوم دايركتور بعرض أول كادر على المسرح	prepareMovie
محدد بدء المشروع لحدث معين يحدث بعد ظهور أول كادر من المشروع على المسرح	startMovie
محدد ينفذ عند نهاية المشروع أو توقفه	stopMovie
محدد لحدث ينفذ تعليماته بتتابع في الوقت الذي تنفذ فيه تعليمات خط الزمن بدءا من الكادر الأول للأخير	Idle
محدد ينفذ أوامر تحدث نتيجة الضغط على أحد أزرار لوحة المفاتيح	keyDown
محدد ينفذ أوامر تحدث نتيجة الضغط على أحد أزرار لوحة المفاتيح ثم رفع الاصبع عن الزر المضغوط	keyUp

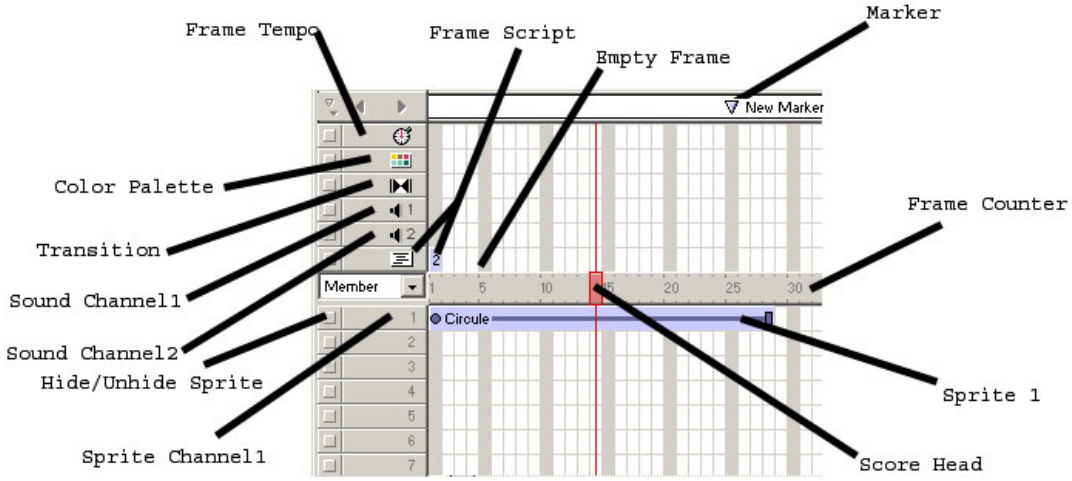
المحددات القياسية
Standard Handlers

ملاحظات هامة على المحددات:

•تلاحظ من الجدول السابق بأن معظم المحددات هي عبارة عن جملة من كلمتين مثل **mouseDown** وأريد ان أنوه الى أن هذه هي طريقة كتابة كود أتفق عليها من قبل المطورين ولكنها غير ملزمة فالمحدد المكتوب بالطريقة **mouseDown** يقوم بنفس عمل المحدد المكتوب بالطريقة **MOUSEDOWN** فعملية إستخدام الحروف الكبيرة **Capital** أو الصغيرة **Small** عملية غير ملزمة في الدايركتور ولا تؤثر في طريقة إستجابته للأوامر ولكن جرت العادة على الكتابة بهذه الطريقة لسهولة قراءة الكود المكتوب فتكون الكلمة الأولي **Small letters** بينما يكون الحرف ألأول من الثانية **Capital letter**.

•المقصود بالكادر هو الفرام **Frame** وهو مرقم على نافذة المشروع **Score Window** وقد يشغل العنصر الواحد عددا من الكادرات والعنصر هو الشبح **Sprite** وقد يتواجد على قناة أو أكثر وتسمى **Sprite Channel** وعادة تلحق رقم القناة باسم العنصر كأن ترى على الشاشة **Sprite1** أو **Sprite2** وهكذا دواليك.

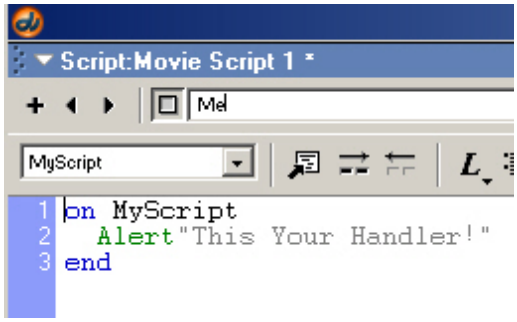
لاحظ الشكل التالي لكي تتعرف عن قرب عن المسميات الدارجة التي سبق وأن عرضانها أثناء الحديث عن واجهة البرنامج



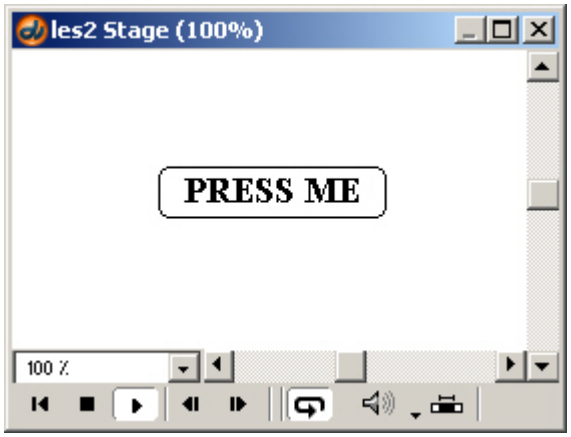
من المهم جدا دراسة الشكل أعلاه وهو مقطع من نافذة المشروع **Score Window** دراسة متأنية وحفظ مسميات الأماكن التي تشير اليها الأسهم لأننا أثناء برمجتنا للدايركتور في الخطوات القادمة سنشير الى الكثير منها بالأسماء فقط فعلى المستخدم دراستها جيدا لكي نتجنب تكرار المعلومة أثناء عرضنا لتقنيات البرمجة.

نلاحظ دائما بأن دايركتور يميز الأكواد عادة باللون الأخضر والأزرق وهذا شئ جميل من شركة ماكروميديا وذلك يسهل قراءة الكود كثيرا ويجعلك تقرأ الأكواد بسهولة والأن سنجرب تكوين محدداتنا الخاصة بنا والتي يلونها دايركتور باللون الأسود تميزا لها عن غيرها من المحددات القياسية وأكواد البرمجة الأخرى.

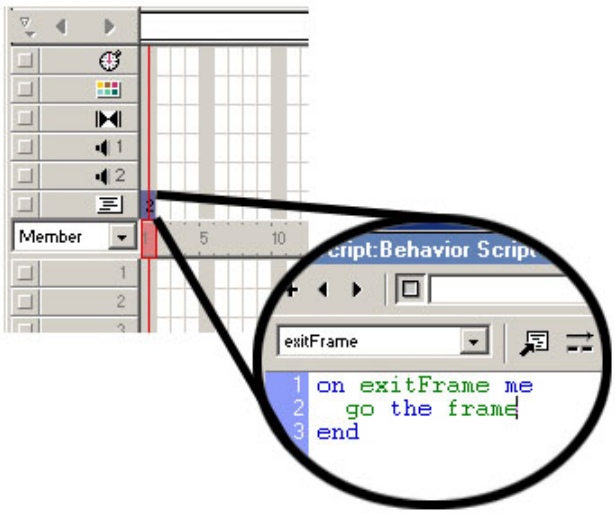
١. قم بفتح نافذة كود مشروع **CTRL+0** ثم اكتب التالي كما الشكل..



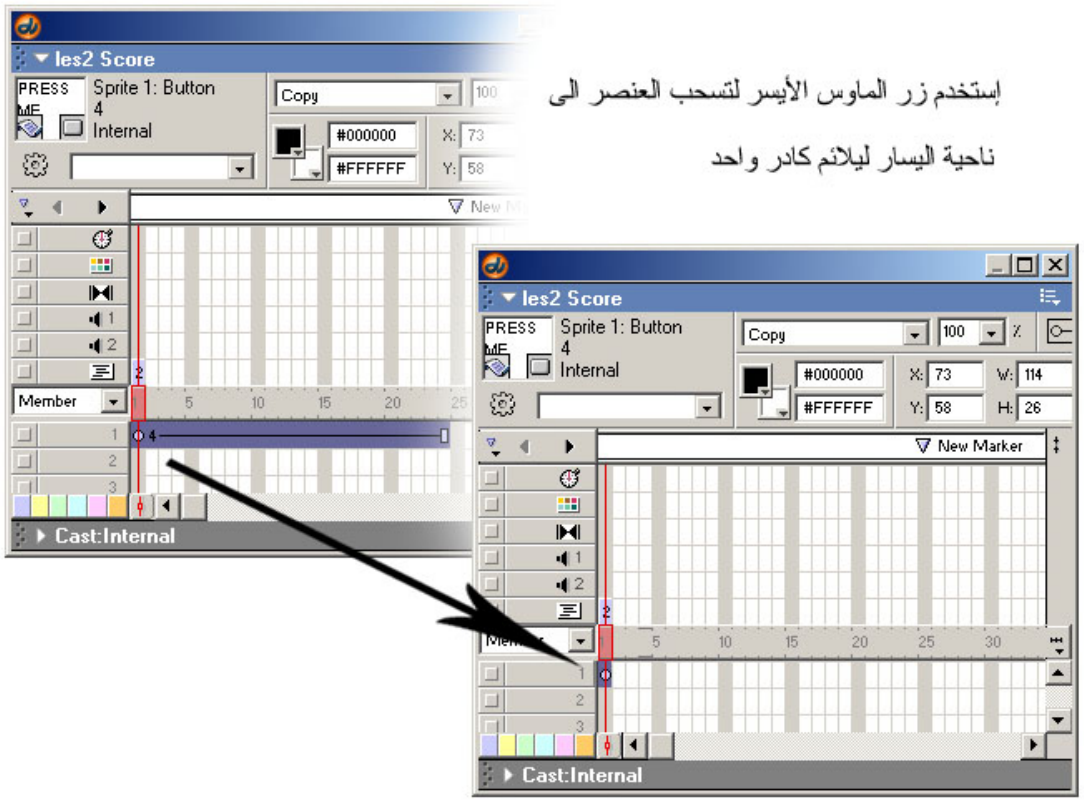
٢. ثم قم بتصميم زر كما تعلمت من الدرس السابق في نافذة المسرح **Stage** كما الشكل



٣. الأن ضع الكود **Go the frame** داخل أول كادر على نافذة كود المشروع كما الشكل التالي.

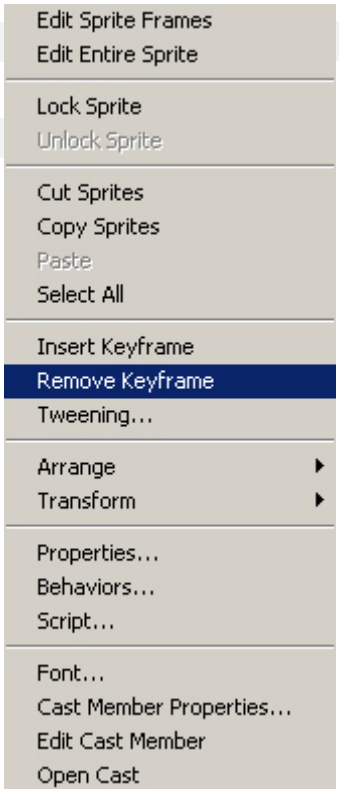


١. قم بتقصير العنصر **Sprite 1** لكي يلائم كادر واحد في نافذة المشروع كما الشكل التوضيحي المقابل..إسحب بالماوس مع الضغط على **ALT**



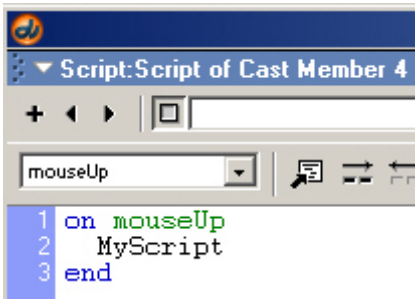
إستخدم زر الماوس الأيسر لتسحب العنصر الى ناحية اليسار ليلائم كادر واحد

٢.والأن إضغط بزر الماوس الأيمن على الزر الذي أمامك فتظهر قائمة منبثقة وإختر منها الاختيار إزالة مفتاح الحركة **Remove Keyframe** كما الشكل المقابل.



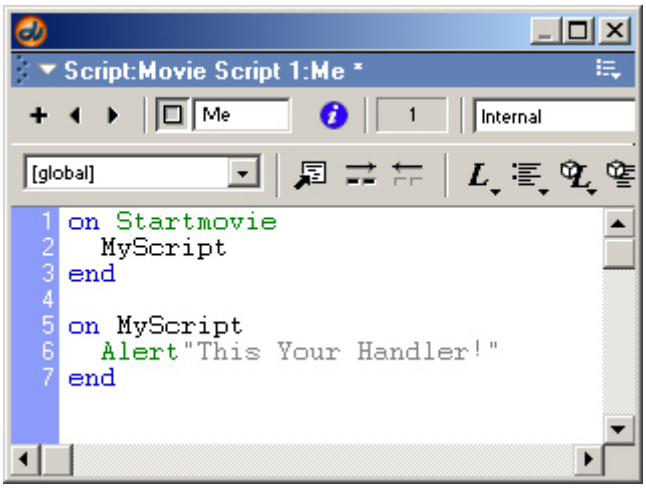
١. والأن افتح كود الزر من الكاست كما تعلمتCast Member Script من داخل الكاست أو بالضغط على Ctrl+' كما الشكل

٢. فتلاحظ المحدد On mouseUp مكتوب وجاهز كما تلاحظ جملة End أيضا مكتوبة بينما يوجد سطر أوسط فارغ, أكتب في هذا السطر الجملة MyScript كما الشكل



٣. الأن قم بتنفيذ المشروع بالضغط على Play أو Ctrl+alt+1 من لوحة المفاتيح, واضغط على الزر Press ME ماذا تلاحظ؟ يجب أن تلاحظ ما حدث في الشكل التالي..

الأن أظنك قد فهمت كيفية إنشاء محدداتك الخاصة.. فكما لاحظت من المثال السابق فإن المحدد قمنا بإنشائه داخل كود مشروع ثم قمنا بإستدعاء المحدد من الزر, ويمكنك إستدعاء المحدد من أي مكان أو من أي زر آخر أو حدث آخر..بل يمكنك إستدعاء المحدد من كود المشروع ذاته, لاحظ معي الشكل التالي..

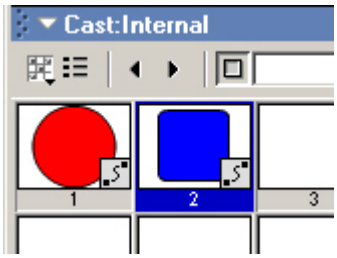


هنا قمت بإستخدام المحدد القياسي**on Startmovie** لكي أقوم بإستدعاء المحدد الذي أنشأناه منذ قليل.. والأن قم بتشغيل المشروع فستلاحظ ظهور الرسالة **this Your Handler** كما تتوقع فالمحدد بدء المشروع قام بتنفيذ الأوامر أسفل المحدد **MyScript** ويمكك ملاحظة أنني إتبعت أسلوب مشابه في كتابة الكود حيث قمت باستخدام الحرف الكبير **Capital** في الجملة الأولى والثانية, وبالطبع يمكنك إستخدام ما تشاء من أسماء لمحدداتك كما يمكن تأليف ما يحلو لك من أكواد, وكل يوم تتعلم فيه شيئا جديدا من البرمجة ستجد أن الحياة أسهل وأفضل مع دايركتور.

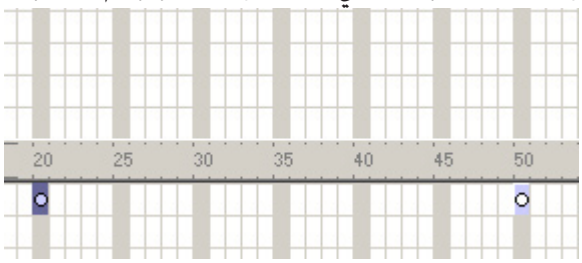
التجول داخل دايركتور باستخدام اللينجو

الأن بعد أن تعلمنا كيفية كتابة محدداتنا الخاصة فقد حان الوقت كي نتعلم شيئا جديدا وهو كيفية التجول داخل الدايركتور وهو مايطلق عليه ال **Navigation** وهو تصفح مادة الوسائط المتعددة أي أن نجعل اللينجو تنقلنا من كادر الى كادر آخر أو من ملف **DIR** الى ملف **DIR** آخر وسنتعلم أيضا كيفية إضافة المؤقت بلغة اللينجو وكيفية الإستفادة منه وربطه بالتصفح داخل اللغة.. لكي نبدأ بفهم مبسط لنحو اللغة ونتعلم كيفية إشراك الأوامر في مكان واحد.

١. الأن قم بفتح مشروع جديد **Ctrl+N**
٢. قم بعمل شكل لدائرة وآخر لمربع وضع الدائرة في الكادر **50** بينما ضع المربع في الكادر **20**, للتسهيل قم باستخدام محرر المنحنيات الحرة **Vector Shape Window** بالضغط على الزر
٣. يجب أن تبدو نافذة الأعضاء لديك **Cast** شبيهة بالشكل التالي..



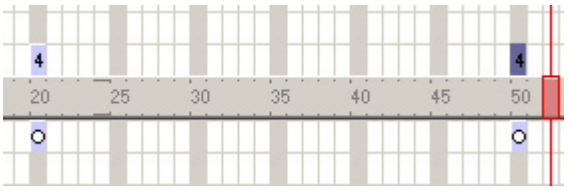
٤. الأن قم بسحب كلا من الشكلين كما الخطوة **2** في كلا من الكادر رقم **50** والكادر رقم **20** ..كما الشكل التالي..



٥. قم بتصميم كود خط الزمن **Score Script** بالنقر مرتين بالزر الأيسر للماوس فوق العنصر الموجود في الكادر رقم **20** ثم أكتب التالي..

```
on exitFrame me
  go the frame
end
```

٦. والأن قم بنسخ هذا الكود بالضغط على **Ctrl+C** ثم لصقه في كود خط الزمن فوق العنصر الموجود في الكادر رقم **50** بالضغط على **Ctrl+X** ويمكنك أيضا عمل النسخ واللصق من قائمة **edit** أو بالضغط بالزر الأيمن للماوس فوق العنصر ذاته, شاشتك الأن تشبه الشكل التالي..



٧. والأن إفتح نافذة المشروع **Movie Script** لكي نبدأ بكتابة الأكواد فيها.

```
on StartMovie
  go to frame 20
end
```

٨. والأن أظهر نافذة المسرح بالضغط على الزر  في نافذة الأدوات العليا

٩. والأن قم بتشغيل المشروع. ماذا تلاحظ؟..لقد قفز مؤشر خط الزمن فعلا الى الكادر رقم **20** وتوقف عنده, فقد طلبنا من دايركتور عن طريق كود المشروع بأن يقفز الى الكادر رقم **20** بالأمر **go to frame 20** بينما نحن نضع كود التوقف عند هذا الكادر باستخدام كود خط الزمن الذي أنشأناه **go the frame** والذي يؤدي بدوره الى تثبيت مؤشر خط الزمن **Score** على **Head** الكادر رقم **20**.

١٠. والأن قم بتغيير كود المشروع ليلائم الذهاب الى العنصر الموجود في الكادر رقم **50**, لن تحتاج الا الى تغيير الرقم **20** الى **50**.

١١. قم بتشغيل المشروع ولاحظ النتيجة؟, أرأيت؟ ان التجول هو أمر في غاية البساطة؟, دعنا الأن نقوم ببعض العبث مع كود المشروع..

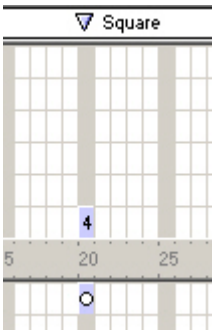
١٢. الأن غير كود المشروع الى الأمر التالي...

```
on StartMovie
  go to frame 49+1
end
```

١٣. الأن قم بتشغيل المشروع, وأخبرني بالنتيجة؟ ان النتيجة هي أن دايركتور سيأخذنا مرة أخرى الى الكادر رقم **50** في نافذة خط الزمن فالخدعة التي فعلناها هي خدعة رياضية فقد طلبنا من دايركتور أن يأخذنا للكادر رقم **49** مضاف اليه رقم **1** وهذا يعادل **50** ولذلك فقد انطلق بنا دايركتور الى الكادر رقم **50**.

١٤. الأن سنتعلم حيلا جديدة أكثر إثارة في التجول داخل دايركتور, فسنتعلم كيفية إضافة أسماء الكادرات وهي ما يطلق عليه **Markers** في الدايركتور- راجع الشكل في صفحة **5**, فبدلا من أن نخبر دايركتور الى الذهاب الكادر رقم **20** مثلا وهو يحوي المربع سنخبره بأن يذهب الى كادر المربع, وطبعا تتساءل كيف يحدث ذلك؟ تابع معي..

١٥. قم بفتح نافذة خط الزمن **Score Window** وانقر بالماوس نقرة واحدة على الشريط الأبيض الذي يعلو شبكة الكادرات وأكتب كلمة **Sqaure** فوق الكادر رقم **20**.. انظر الشكل التالي وقد قمنا باضافة اسم للكادر رقم **20** بالاسم **Sqaure**



١٦. الأن قم بفتح كود المشروع مرة ثانية واكتب فيه الأمر التالي..

```
on StartMovie
  go to frame "Square"
end
```

١٧. والأن تلاحظ بعد تنفيذ المشروع بأن دايركتور قد ذهب للكادر رقم **20** والذي يحوي شكل المربع, أو بالأصح فقد انطلق للكادر ذو الاسم **Sqaure** ولاحظ مرونة لغة اللينجو في الأمر **go to frame "Sqaure"** والأن غير كود المشروع للمرة الثانية وأكتب الأمر التالي..

```
on StartMovie
  go to Marker ("Square")
end
```

١٨. ماذا تلاحظ بعد تنفيذ المشروع؟ لقد انطلق دايركتور للمرة الثانية الى الكادر ذو الإسم **Sqaure** فالأمر السابق ما هو الا مرادف للأمر الذي سبقه منذ قليل, وهذه هي مرونة في لغة اللينجو , لكن إذا أردت رأيي الشخصي فهذه المرونة قد تربك المستخدم المبتدئ في البرمجة ولا أعلم لماذا كررت ماكروميديا من أوامرها المرادفة في لغة اللينجو, ولكن لا بأس, سنحاول تجنب ذلك مستقبلا بقدر الإمكان, وللعلم فليست الأوامر فقط هي المكررة في لغة اللينجو إنما فان شركة ماكروميديا قد صممت أكثر من اسلوب لكتابة الأوامر, لكني أتبع هنا أسهل أسلوب ممكن!.

١٩. الأن جرب كود المشروع التالي..

```
on StartMovie
  go to Marker ("Square")+1
end
```

٢٠.ماذا تلاحظ؟ لقد إنطلق دايركتور الى العنصر رقم **50** بعد برهة من الوقت؟. أن الكود السابق يذهب للدايركتور الى الكادر الذي يلي اسم الكادر **Sqaure** بفرق كادر **1** فقط وهذا ما جعلك تنتظر برهة الى أن استقر الى الكادر رقم 50 والذي من البديهي أن ينتظر عنده لأنه ينفذ كود خط الزمن **go the frame** الذي يأمر مؤشر خط الزمن بالتوقف والانتظار.

٢١.والأن جرب بنفسك وضع اسم للكادر الذي يحوي الدائرة وقم ببعض العبث بلغة الينجو ولاحظ النتائج وإكتسب خبرات ودرب نفسك جيدا على التجول داخل دايركتور.

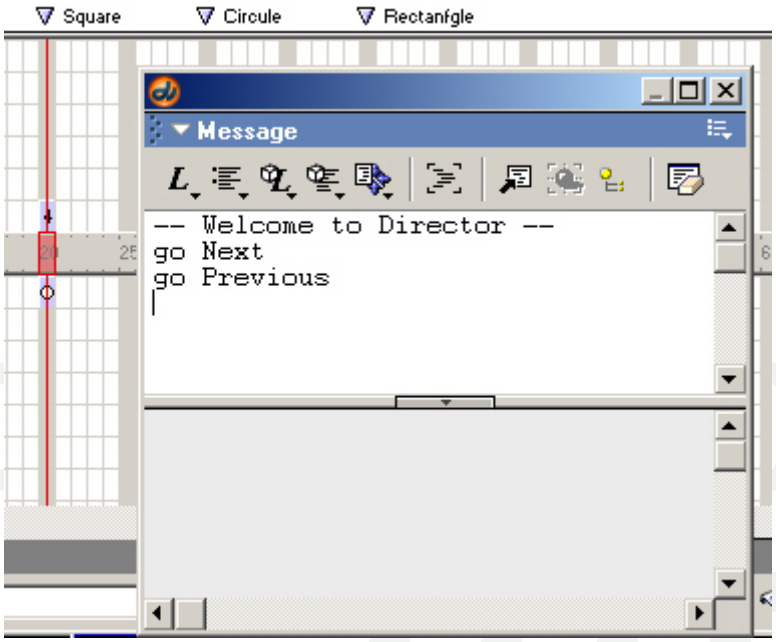
لمحة سريعة:
لإزالة أسماء الكادرات **Markers** فانك عادة تلاحظ تغير شكل مؤشر الماوس لدي الإقتراب منهم لكي تستطيع تحريكهم يمينا ويسارا ونقلهم ويمكنك أيضا بالضغط على الزر الأيسر للماوس ثم رفع المؤشر للأعلى بأن تزيل الماركر من موضعه نهائيا وتمسحه, سارع وقم بتجربة ذلك عمليا وبنفسك!.

١. ادرس الأمر التالي جيدا..

```
go  to frame("Memory") of movie("main")
```

في الأمر السابق فان دايركتور سيأخذنا للكادر ذو الاسم Memory الموجود في المشروع main أي أن دايركتور سيقوم بمناداة مشروع DIR آخر ذو الاسم main, ويمكنك ايضا إستخدام أرقام الكادرات في هذا المثال فكما تعلمت فالليننجو تتميز بمرونة كبيرة جدا.

٢. تستطيع التجول بسهولة بين أسماء الكادرات باستخدام الأوامر go Next و go Previous وهي تعني إذهب للتالي وإذهب للسابق, فاذا كنت تصمم مشروع ضخم يحوي كتالوجا للصور على سبيل المثال فيمكنك تصميم أزرار للتجول على شكل أسهم ويكون زر التالي يحوي الأمر go Next بينما زر السابق يحوي الأمر go Previous, يمكنك تجربة ذلك من نافذة التراسل Message Window كما الشكل التالي.



لمحة سريعة:

يمكنك أيضا التجول مباشرة الى ملف دايركتور آخر بالامتداد **DIR** عبر إستخدام الأمر المباشر

```
go to movie"main"
```

تصميم المؤقت بلغة اللينجو

في كثير من الأحيان أثناء تصميمك لتطبيقات الوسائط المتعددة تقوم بالاحتياج الى مؤقت زمني, تذكر المؤقت الزمني اللازم لتصميم الإختبارات وتوقيتها, أو يمكنك عمل لوحة للأسئلة بمؤقت زمني معين, وعموما سنبدأ درسنا الجديد الأن بإختبار عدة طرق لتصميم المؤقتات في الدايركتور..

١. افتح مشروعا جديدا في الدايركتور وصمم شكل مربع أو دائرة كما يحلو لك.
٢. اسحب المربع أو الدائرة على الكادر رقم **20** على خط الزمن ثم قم باعادة تحجيمه بحيث يشغل كادرا واحدا فقط.
٣. الأن اكتب كود خط الزمن التالي فوق الشكل الذي صممته مباشرة

```
on exitFrame me
  go the frame
end
```

٤. الأن قم بالذهاب الى الكادر رقم على خط الزمن واكتب الكود التالي..

```
on EnterFrame
   delay 4 * 60
end
```

٥.الأن قم باخفاء كل النوافذ عدا نافذة خط الزمن فقط **Score Window** ثم قم بتشغيل المشروع الأن؟ ماذا تلاحظ؟.
٦ .لقد قام دايركتور بلعب المشروع وعندما وصل الى الكادر رقم **15** فقد توقف لمدة أربعة ثوان قبل أن يكمل العمل ويذهب للكادر رقم **20** فقد استخدمت المحدد **on EnterFrame** هذه المرة لأخبر دايركتور بضرورة تنفيذ الأمر عند بدء الدخول إلى الكادر بينما قمت بتصميم الحدث Event وهو يتمثل بالأمر **delay 4*60** بمعنى قم بالتأخير المتعمد لمدة اربعة ثوان وأربعة ثوان في الدايركتور تمثل رياضيا بـ **4*60** , وهذا هو أسرع مؤقت يمكك تصميمه على الإطلاق! بلغة اللينجو وتعال الأن لنلقي نظرة على مؤقتات أكثر تعقيدا.

سـأستخدم طريقة جديدة معقدة قليلا عن الطريقة السابقة بإستخدام الأمر **StartTimer** والأن سنقوم بتجربتها.

١. قم بفتح نافذة مشروع جديد بمقاس **320X240** ثم قم بتصميم شكل مربع وضعه على الكادر رقم **20**
٢. الأن في الكادر رقم **15** قم بنقر كود خط الزمن مرتين لفتح نافذة الكود ثم أكتب الكود التالي:

```
on exitFrame
  startTimer
end
```

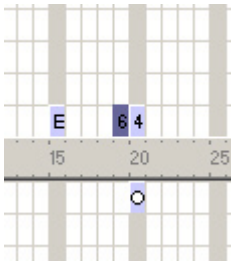
٣. الأن في الكادر رقم **19** قم بنقر كود خط الزمن مرتين لفتح نافذة الكود ثم اكتب الكود التالي:

```
on exitFrame
  IF THE TIMER <2*60 THEN GO THE FRAME
  ELSE GO THE FRAME + 1
end
```

٤. في الكادر رقم **20** قم بنقر نافذة الكود مرة أخرى وأكتب الكود التالي:

```
on exitFrame me
  go the frame
end
```

٥. الأن يجب أن تبدو شاشتك على خط الزمن تشبه التنسيق في الشكل التالي:



٦. الأن قم بتشغيل المشروع وستلاحظ بأن مؤشر خط الزمن يأتي الى الكادر رقم **19** ويتوقف لمدة ثانيتين قبل أن يستقر في الكادر رقم **20**

سبب ذلك هو أننا كتبنا في كود الكادر رقم **15** أمر بدء التوقيت **StartTimer** وهو أمر يسترعي إنتباه برنامج الدايركتور ويأخذ بالحسبان عملية بدء توقيت معين, ثم اتبعنا ذلك بوضع كود قاعدة (إذا) **IF** في الكادر رقم **19** والكود هو

IF The Timer <2*60 Then go The frame
Else go the frame +1

والسطر الأول من الكود يشترط على دايركتور مرور ثانيتين من الوقت حيث يقول له انه إذا تأخر المؤقت عن تسجيل ثانيتين من الوقت فظل في مكانك, بينما السطر الثاني يخبر بأنه عدا ذلك قم بالذهاب للكادر التالي**Else go the frame +1** , وهذه هي المرة الأولى التي نتعرض لها للأمر **IF** الذي سيكون لنا معه شأن آخر في الدروس المستقبلية, فهذه هي قاعدة شرطية في لغة اللينجو فالشرط يتبعه فعل والقاعدة هي **IF>THEN>ELSE** يعني (إذا-إذن-أو) ففي الكود السابق كان الشرط مرور ثانيتين فاذا كان الوقت المار أقل من ثانيتين فاذن فهو يظل مكانه أو اذا تعدى الوقت ثانيتين فإنه يذهب للكادر التالي.

وقياس الوقت في دايركتور لا يعتمد أساسا فقط على الثانية كأقل قيمة بل هناك ما يعرف بالتكة **tick** وهي تساوي واحد على ستون جزء من الثانية ويمكننا أيضا برمجة الوقت بإستخدام هذه القيمة الصغيرة جدا, ولتجربة ذلك يمكنك إستبدال الأكواد الموجودة في كلا من الكادرين **15** و**19** على التوالي بالأكواد التي توجد أسفل الفقرة.

```
--script for first frame
on exitFrame
  global gTimer
  set gTimer = the ticks
end

--script for Second frame
on exitFrame
  global gTimer
  if the ticks < gTimer + (10 * 60) then
    go to the frame
  end if
end
```

ملاحظات هامة

لمحة سريعة:

لنستخدم عادة الرمز بين القوسين(–)مكررا (--) لإضافة التعليقات على برامجنا وأكوادنا التي نكتبها فعادة تريد إضافة شرح مبسط لكود معين أو توضيح أو وضع عنوان للفقرة البرمجية التي تكتبها وتلاحظ وجود الأزرار  في كل نافذة كود تفتحها فالمفعلة منها تشير لتفعيلها ووضعها قبل الحرف الأول من كود السطر بينما الثانية غير المفعلة فهي تشير الى إزالتها من سطر مضافة إليه, جربها بنفسك.

في الدرس القادم سندرس بتعمق المتغيرات **Variables** ثم يليها قاعدة الشرط **IF** وسترى بنفسك أهميتها الكبيرة جدا كما سندرس عدة أوامر جديدة وسنتعلم سوية كيفية تتبع الأخطاء في لغة اللينجو وإستخدام بقية الأدوات في نافذة الكود وسنقوم بعمل الكثير من الأمثلة والتجارب كما سندرس كيفية تعامل لغة اللينجو مع الأعضاء الموجودين في نافذة الأعضاء لكي نقوم بتصميم زر مبسط.

مقدمة:

هناك نوعان من مبرمجي دايركتور, النوع الأول هو المبرمج العادي الذي هو في الأصل يكتب برامجه بلغات مثل السي أو الفيجوال بيسك وهذا النوع يعلم ماذا سيفعل عند تعلم لغة برمجة جديدة فهو يتوقع أشياء محددة أثناء قرائته ويبحث بين السطور على قواعد معينه تعينه على أداء مهامه والمبرمج العادي يتعلم اللينجو لكي يقوم بحل المشاكل التي قد تواجهه في إنتاج البرامج, والنوع الثاني هو مصمم الجرافيك الذي يود إنتاج الوسائط المتعددة من خلال برنامج الدايركتور وهو النوع الذي سوف أركز عليه في الشرح لأن مصمم الجرافيك يعتبر اللينجو مجموعة من أسطر الأكواد التي تقف بينه وتقف بين إنتاج برنامجه وهذان النوعان من المبرمجين لن يختلفوا في مبدأ المهارات الأساسية التي يجب أن يكتسبوها في اللينجو قبل البدء في إنتاج مشروعاتهم الخاصة بالوسائط المتعددة ولكن ربما إختلفت أهدافهم فالمبرمج العادي قد درس الجبر الخطي ودرس قواعد المعلومات ولذلك فالمبرمج العادي سيكتب برامجه بحيث تحقق أهدافها مع إهمال القيمة الفنية بينما مصمم الجرافيك الذي سوف يصبح مبرمج للغة اللينجو فهو يريد إضافة القيمة والفعالية على برامجه ومشاريعه الخاصة, والقاعدة أو المهارة الأساسية لفهم عملية البرمجة بلغة اللينجو هو أن تستوعب تماما من أنك تعمل داخل بيئة برمجة الكائنات **OOP** وهي البرمجة الموجهة بالكائنات **Obeject Oriented Programming** فقد سبق أن شرحنا المحددات وهي كائنات وهي توجه البرنامج من خلال تنفيذ التعليمات والأسطر في داخلها وفي هذه البيئة من البرمجة لا يمكن التعامل مع البرامج الضخمة مباشرة بل يجب تقسيمها, فكلمة برمجة بحد ذاتها تعني وضع الحلول, فعندما تريد أن تقوم بحل مسألة ما برمجيا فهذا يعني أنك تود أن تضع حلا لها ولذلك إذا اردت كتابة برنامج لعبة مثلا في الدايركتور فمن الخطأ الكبير كتابة كل أسطر البرنامج في مكان واحد لأنك ستواجه عقبات كبرى أمامك والحل هو تقسيم مشكلة برمجة اللعبة مثلا الى عدة أكواد ترتبط فيما بعضها البعض ويكون لها القدرة على مناداة بعضها وهذا هو المقصود من البرمجة الموجهة بالكائنات فهذه الطريقة هي الطريقة الصحيحة والمثالية لبناء البرامج بدلا من إستخدام الطرق الخطية القديمة والتي تضع أمامنا المئات من الأسطر مثل لغة البيسك الخطية القديمة أو الباسكال أو اللوجو والتي كنا نعاني من عملية تصحيحها وتنقيحها لكي تستجيب لنا كما يجب أن نتوقع منها وفي هذا الدرس سأقوم بإذن الله بعرض المهارات الأساسية التي يحتاجها مبرمج لغة اللينجو, وستتعرف بنفسك على جميع مهارات كتابة اللينجو وبطريقة سهلة ومبسطة, الأن جهز برادا من الشاي و آخر من القهوة, ثم إبدأ رحلتك معي.

.

نظرة عميقة ومبسطة على أنواع المتغيرات Variables

سبق أن أشرت الى المتغيرات من قبل وقمنا بعمل شرح سريع لها أثناء عرض عناصر لغة اللينجو والأن حان الوقت لتقـــديم الصورة الكاملة للمتغيرات لكي يتعرف المبرمج عن قرب عليها ويفهمها فهما جيدا.

المتغير **Variable** هو وعاء لحمل وحفظ البيانات والمقصود أي نوع من البيانات سواء كانت عددية أو حلقية بمعنى بيانات تحوي نصوص أو أرقام تعامل كنص أو الإثنان معا والمتغير قد يكون عدد صحيح مثلا أو قيمة رياضية كجذر تربيعي أو قيمة لوغاريتمية, والهدف من تصميم وتخزين المتغيرات هو التفاعل مع البرنامج وجعله يفكر ويتخذ القرارات فالمتغير يمثل لنا الـمدخل **Input** بينما يمثل رد فعل الكمبيوتر **OutPut** وقد يكون المتغير هو حلقة وصل بين البرنامج ومستخدمه فبناءا على البيانات المتغيرة التي يدخلها المستخدم يحدث رد فعل معين, وهذا يسمى التفاعلية **Interactivity** .

هناك ثلاث أنواع رئيسية للمتغيرات في لغة اللينجو وهم,**Local,Global,Property** ولا يوجد تعريب موحد لهذه المسميات لكني سأحاول بقدر الإمكان وضح تعريب مبسط للمعاني المرادفة بالعربية لتبسيط الفهم.

أولا : المتغير المحلي Local

من الممكن وصف المتغير **Local** بأنه متغير محلي (مكاني) أو مخصص لعملية بذاتها ولا يمكنه العمل خارجها.

مثال(١):

جرب كتابة المحدد التالي في نافذة كود المشروع **Movie Script (Control+0)** لفهم المتغير المحلي **Local.**

```
on setLocalVar
localVar = "I am a local"
end
```

ومن ثم قم بتشغيل البرنامج بواسطة الزر ▶ ثم باستخدام نافذة التراسل**Message Window** لديك قم بإستدعاء المحدد وذلك بكتابة الأمر **setLocalVar** وبهذا فقد قمت بتفعيل عمل المحدد ثم اكتب مباشرة بعدها الأمر **Put LocalVar** وشاهد النتيجة أمامك وهنا سيعطيك دايركتور **<Void>--** أو ستلاحظ عدم إستجابة دايركتور تماما لما أدخلته وسبب ذلك هو أن دايركتور يرفض أن يعطيك نتائج معطيات متغير محلي **Local** إذ أن أية عمليات تتم على المتغير المحلي تتم فقط من داخل المحدد الذي أنشأ داخله والأن قم بتعديل البرنامج السابق بكتابة **Put LocalVar** بعد السطر الثاني ليصبح برنامجك كالتالي:

```
on setLocalVar
localVar = "I am a local"
Put LocalVar
end
```

ومن ثم قم بتشغيل البرنامج مرة أخرى ومن نافذة التراسل قم بكتابة الأمر **SetLocalVar** مرة أخرى وهنا سيستجيب دايركتور للأمر وسيعطيك النتيجة التي تتوقعها وهي قيمة **LocalVar** التي تتمثل في **"Iam a Local"** وكما قلت فسبب ذلك هو أن المتغير المحلي لا ينفذ الا من خلال المحدد الخاص به, وفي المثال السابق قمنا بوضع قيمته من نفس مكانه.

مثال(٢):

في هذا المثال ستكون **myName** و **myAge** متغيرات محلية.

```
on StartMovie
 myName = "Mohammed Ibrahim"
 myAge = "28"
 Alert "You Are"&&myName&&"Your Age is"&&myAge
 Showlocals
end
```

في المثال السابق يقوم البرنامج بتخصيص المتغير **myName** الى الاسم محمد إبراهيم ثم يخصص المتغير **myAge** الى العمر وهو هنا **28** ثم يقوم السطر ذو الأمر **Alert** بكتابة العبارة شاملة المتغيرات ويقوم السطر الأخير بعرض المتغيرات المحلية داخل نافذة التراسل **Message Window** حيث تراه يعرض القيمتين **myName** و **myAge** داخل النافذة مسبوقين بعبارة **local Variables**, وكما ترى فالمتغير المحلي يخصص مباشرة من داخل محدد الحدث **Event handler** دون أية إجراءات أخرى, ويمكنك إستخدام الأمر **ShowLocals** دائما لإختبار برامجك في حالة إستخدامك لمتغير محلي, والرمز **&** هو رمز الربط (و) **AND** وتكراره مرتين متتاليتين يعني ترك مسافة أثناء الربط مع الجملة أو الكلمة التي تليه.

يمكنك مشاهدة وتطبيق المثال السابق من الملف **myname.dir** الموجود داخل الدليل **Essential** على القرص المرفق مع الكتاب.

ثانيا : المتغير العام Global

بعكس المتغير المحلي فالمتغير العام له تعريف ويسمح باستدعائه والتعامل مع قيمته من خلال عدة محددات عبر برنامجك ومن عبر عدة أكواد ولعمل متغير عام فيجب تعريف دايركتور به والأن قم بفتح نافذة كود مشروع جديدة **movie Script** عن طريق ضغط **Ctrl+0** ثم اكتب التالي

```
on startMovie
 global NameOne,NameTwo
 NameOne = "Ekrami"
 NameTwo = "Mohammed"
end
```

ملحوظة هامة جدا: تذكر دائما قبل تجربة الأوامر من خلال نافذة التراسل أن تضغط على الزر ⚡ في نافذة الكود لكي يقوم دايركتور بمعالجة الكود في الذاكرة وتنجح عملية التراسل بين الكود ونافذة التراسل.

الأن من ألأيقونة ▶ قم بتشغيل البرنامج ثم من نافذة التراسل **Message Window** قم بكتابة الأمر التالي:

```
put NameOne
--"Ekrami"
put NameTwo
--"Mohammed"
```

الأن مباشرة من نافذة التراسل **Message Window** قم بكتابة الأمر **ShowGlobals** لكي يعرض لك دايركتور كافة المتغيرات العامة الموجودة.

```
ShowGlobals
--Global Variables--
--Version = "10.1"
--"Ekrami"
--"Mohammed"
```

تلاحظ من نافذة التراسل **Message Window** أن دايركتور قام بعرض رقم النسخة الخاصة به أولا ثم قام بعرض بقية المتغيرات وهذا شئ عادي, فدايركتور مبرمج على ذلك عند إستدعاء التنفيذ للأمر **ShowGlobals** والأن جرب كتابة الأمر **ClearGlobals** مباشرة وأنت في نفس النافذة ولاحظ ما يحدث.

```
--ClearGlobals
```

تلاحظ عدم حدوث شئ , حسنا لنتأكد انه لم يحدث شئ, أكتب مرة ثانية في نافذة التراسل **Message Window** الأمر **Put NameOne** ولاحظ ما يحدث؟

```
--Put NameOne
--<void>
--Put NameTwo
--<void>
```

حسنا الأن لقد فهمت أن الأمر **ClearGlobals** يقوم بمسح جميع المتغيرات من الذاكرة, والأن شغل البرنامج مرة ثانية وعاود تجربة وضع المتغيرات في نافذة التراسل ولاحظ ما يحدث؟ لقد عادت المتغيرات للظهور بكل تأكيد؟ أليس كذلك؟.

يمكن تعريف المتغيرات العامة بصورة مفردة أو مزدوجة فمثلا يمكنك كتابة **global myName** ثم كتابة **global myAge** كمتغيرين منفصلين أو يمكن كتابة **global myName,myAge** وهذه الطرق لا تؤثر في سير برنامجك فيمكنك إختيار ما ترتاح إليه أثناء تصميمك لبرامجك الخاصة.

حسنا هذا ليس كل شئ بالنسبة للمتغيرات العامة, هناك شيئا مهما يجب أن تأخذه بعين الإعتبار لأنه سيسهل عليك برمجة المتغيرات العامة وهو مكان تعريف المتغير, فما هو أنسب مكان لتعريف المتغير؟ تابع معي لكي تتعلم.

لو إفترضنا أن هناك اكثر من (محدد حدث) **Event Handler** يستخدم نفس المتغير العام فكيف سيقوم بالتعرف الى قيمته؟ يجب وضع تعريف المتغيرات العامة دائما قبل البدء في إنشاء المحددات الخاصة أو العامة ولذلك لكي تستطيع وظائف المحددات إستدعاء قيمة المتغير العام ولكي تستوعب ذلك عمليا سنقوم بعمل بعض التجارب المثيرة على المتغيرات, قم بفتح الملف **globalPlace_1.dir** من داخل الدليل **Essential** على القرص المرفق مع الكتاب ثم قم بكتابة الأمر **AddTwoNumbers** في نافذة التراسل **Message Window** ولاحظ ما يحدث؟ لقد ظهرت القيمة صفر؟ لماذا, الأن توجه الى الكاست **Cast** ثم قم بتحرير كود المشروع **Movie Script** وفتحه لكي تقرأ ما داخله من أسطر الأكواد.

المفترض الأن أن شاشة تحرير كود المشروع **Movie Script** يوجد بداخلها الأسطر التالية:

```
global NumberOne,NumberTwo
on IntroduceNumbers
  NumberOne = 1
  NumberTwo = 2
end

on AddTwoNumbers
  put NumberOne+NumberTwo
end
```

حسنا, لنقوم الأن بمراجعة الأسطر لكي نعلم لماذا وضعت لنا نافذة التراسل **Message Window** القيمة صفر كرد عندما قمنا بتشغيل المحدد **AddTwoNumbers** من داخلها.

في السطر الأول يوجد تعريف للمتغيرات العامة ولدينا هنا متغيرين إثنين وهما **NumberOne** و **NumberTwo** كناية عن متغيران لعددان, ثم بعد ذلك قمت بعمل محدد خاص **Event Handler** بالإسم **IntroduceNumbers** ثم قمت بتعريف قيم المتغيرات العامة داخل هذا المحدد حيث أخذ المتغير **NumberOne** القيمة **1** بينما أخذ المتغير الثاني **NumberTwo** القيمة 2 ثم انتهى المحدد على هذه الصورة بالأمر **end**, وهناك أيضا المحدد الخاص **AddTwoNumbers** الذي يقوم بجمع كلا من المتغيرين **NumberOne,NumberTwo** بواسطة الأمر **put** ثم ينتهي المحدد بالأمر **end**, حسنا لو لاحظت معي الأسطر جيدا ستجد أولا خطأ منطقي وهي أن القيمة التعريفية للمتغيرات العامة موجودة داخل المحدد الخاص **IntroduceNumbers** وذلك المكان أدى الى فشل المحدد الثاني **AddTwoNumbers** في التعرف الى قيمتهم وكأن المبرمج قام بتعريف المتغيرين بواسطة الأمر **global** ثم نسى أن يضيف لهم قيم خاصة ولذلك فقد تصرف دايركتور على هذا الأساس وقام بوضع القيمة صفر في نافذة التراسل **Message Window** عند إستدعاء تنفيذ المحدد, إذا هنا يجب أن ننقل عنوان المحدد **IntroduceNumbers** داخل المحدد **AddTwoNumbers** لكي يستطيع دايركتور قراءة قيم المتغيرين قبل إجراء عملية الجمع عليهم ليصبح كالتالي:

```
on AddTwoNumbers
  IntroduceNumbers
  put NumberOne+NumberTwo
end
```

سبب قيامنا بذلك هو تمكين البرنامج من القراءة الصحيحة لقيم المتغيرين وهنا قمنا بإستدعاء المحدد **IntroduceNumbers** داخل المحدد **AddTwoNumbers** وهذه الطريقة تسمى النداء **Calling** فيما بين المحددات **Handlers** وعندما تقوم الأن بتجربة تنفيذ المحدد **AddTwoNumbers** ستجد أن نافذة التراسل **Message Window** تخبرك بأن القيمة هي **3** وذلك هو تأكيد أن برنامجك يسير كما يرام أو يمكنك مشاهدة البرنامج كاملا عن طريق فتح الملف **globalplace_2.dir** من نفس الدليل **Essential** من على القرص المصاحب للكتاب.

إذا تعلمنا معا في المثال السابق كيفية تعريف وتحديد قيم المتغيرات العامة وكيفية إتصال المحددات ببعضها البعض والأن قم بعمل مشروع جديد **Movie** ثم إكتب التالي في كود المشروع **Movie Script** :

```
on StartMovie
  global NumberOne,NumberTwo
  NumberOne =1
  NumberTwo =2
end
```

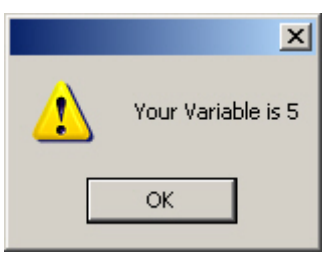
ثم عليك تشغيل البرنامج بالضغط على زر ◂ كنك إستدعاء قيم المتغيرات العامة من خلال نافذة التراسل **Message Window** مباشرة بكتابة **put NumberOne+NumberTwo** أو إستدعاء قيمة كل متغير على حدة بكتابة **put NumberOne** على سبيل المثال لوضع قيمة المتغير **NumberOne**.

حسنا هذا ليس كل شئ عن المتغيرات العامة, يجب أن تعلم أن قيمة المتغير العام **Global Variable** تظل في الذاكرة فدايركتور يظل محتفظا بهذه القيمة ولا يقوم بمسحها من الذاكرة إلا بناءا على تعليمات المبرمج والأن سنقوم بعمل تطبيق جديد لكي نختبر به إحتفاظ دايركتور بقيمة المتغير العام **Global** داخل ذاكرة الكمبيوتر بحيث يمكن *تمرير* قيمة المتغير العام عبر أكثر من مشروع دايركتور **Movie DIR** وذلك مفيد طبعا في تصميم البرامج, والأن تابع معي المثال التالي.

قم بفتح الملف **flyvar_a.dir** من الدليل **Essential** الموجود على القرص المصاحب للكتاب, والأن قم بفتح كود المشروع **Movie Script** الموجود في العضو الأول من اليسار في الكاست **Cast** ثم لاحظ معي أسطر البرنامج

```
global FlyVariable
on startMovie
  FlyVariable = 5
  go to movie"flyvar_b"
end
```

تلاحظ أني قمت بوضع متغير عام بالإسم **FlyVariable** ثم قمت بتخصيص قيمة له داخل المحدد **StartMovie** وهي القيمة **5** ثم طلبت من البرنامج الإنتقال الى ملف مشروع آخر **DIR** وهو الملف **flyvar_b.dir** والأن قم بتشغيل المشروع من الزر ◂ ثم لاحظ ماذا حدث؟ لقد ظهرت الرسالة التالية:



تلاحظ أن الرسالة تخبرك **Your Variable is 5** والأن إضغط **OK** لغلق الرسالة ولاحظ الأن أنك تعمل من خلال الملف **flyvar_b.dir** والأن قم بفتح كود المشروع **Movie Script** الخاص بهذا الملف.

والأن إقرأ معي الكود المكتوب:

```
global FlyVariable
on startMovie
  Alert"Your Variable is"&& FlyVariable
end
```

تلاحظ من الكود السابق أن البرنامج قام أولا بتعريف المتغير العام **FlyVariable** ثم من المحدد **StartMovie** وهو محدد بدء المشروع قام بعرض رسالة تنبيه بإستخدام الأمر **Alert** التي تعرض قيمة المتغير وهي الرقم **5** بإستخدام رمز الربط **&** كما شرحت سابقا, وتكرار الرمز يقوم بعمل مسافة.

ومن المثال السابق نستطيع إستيعاب كيفية إستخدام دايركتور للمتغيرات العامة وتمريرها عبر المشاريع المختلفة, وذلك شئ في غاية السهولة والبساطة كما رأيت وهناك بالطبع طرق أكثر فائدة وأكثر تقدما لعرض وإستخدام المتغير العام **Global** كما سترى بعد قليل في المثال التالي كيفية تصميم متغير عام **global** ووضع قيمته داخل حقل نصي في مشروع آخر **DIR**.

تطبيق آخر على تمرير قيم المتغيرات العامة لمشروع آخر بتحويلها لمتغير حلقيSTRING:

المتغير الحلقي**String Variable** ببساطة هو متغير يحمل قيمة نصية أو عددية (حروف وأرقام) معا أو منفصلين ويعامل كقيم نصية ودائما يكون المتغير الحلقي محاطا بعلامتي تنصيص في بدايته ونهايته, كالمثال التالي:

myString = "Number 9"

أنظر درس المتغيرات الحلقية لمزيد من التفاصيل (في نفس هذا الفصل).

قم بفتح الملف **global_pass.dir** من الدليل **Essential** من الدليل **Book Code Source** المرفق مع الكتاب, الأن شغل البرنامج ثم قم بالضغط على الزر **Set gVariable = 1** ثم إضغط مباشرة على الزر **NEXT MOVIE**, وتلاحظ أن المتغير الذي قمت بتسجيله وهو الرقم **1** قد انتقل اتوماتيكيا الى الحقل النصي في المشـــــــروع ذو الإسم **global_pass2.dir** وكل ما قام دايركتور عمله هو أنه سجل قيمة المتغير **gVariable** على أساس أنها تساوي **1** وإذا ألقيت نظرة على كود حقل النص**blank field** فستلاحظ أن الكود المكتوب هو:

```
global gVariable
on exitframe
member("blank field").text = string(gVaribale)
end
```

والكود السابق يعمل فور ضغط المستخدم على الزر **NEXT MOVIE** لأنه فور الخروج من الكادر الذي يؤدي لإظهار الحقل النصي فإن دايركتور يقوم بتسجيل قيمة المتغير الذي إحتفظ به في ذاكرة الكومبيوتر داخل الحقل النصي **blank field** ويحوله الى متغير حلقي **String**, هذا التطبيق الجميل جدا يوضح لك مباشرة كيف يتعامل دايركتور بسهولة شديدة مع المتغيرات العامة **globals** وهي لا تسبب له أدنى مشكله ومن السهل جدا نقلها عبر ملفات المشاريع المختلفة الأخرى.

تذكر انك تستطيع بسهولة مسح كل المتغيرات العامة من الذاكرة بكتابة الأمر **clearglobals** , يمكنك تجربة ذلك من نافذة التراسل **Message Window**, وإعادة إستدعاء المتغيرات للتجربة ولرؤية رد فعل دايركتور معها.

ثالثا: المتغير الخاص Property

يتميز المتغير الخاص بانه يعمل من خلال كود السلوك **Behaviour** أو من الكود الأب **Parent Script** ولكنه لا يعمل من خلال كود المشروع **Movie Script** وهذا المتغير الخاص يتميز بانه لا يحمل الا القيمة المتغيرة ***الخاصة*** التي تعطى له من ناحية مبرمج المشروع من جهة ومن داخل السلوك من جهة أخرى وسأقوم بالقاء الضوء على طريقة إستخدام المتغير الخاص.

مثال(١):

الأن قم بفتح الملف **Property_EX1.dir** من دليل **Essential** على القرص المرفق مع الكتاب, وقم بتشغيل البرنامج أولا, ماذا تلاحظ؟, تلاحظ بأن العنصر **Sprite 1** قد تحرك الى أعلى نقطة في يسار الشاشة, الأن إضغط الزر **Rewi** للخروج من البرنامج وإعادة العنصر **Sprite 1** إلى مكانه ثم قم بفتح نص الكود **Behaviour** المرتبط مع العنصر **Sprite 1** من الكاست **Cast** أو من خط الزمن **Score**, الأن أقرأ نص الكود ولاحظه جيدا:

```
property SpriteNum

on BeginSprite me
  Sprite(SpriteNum).loc = point(12,12)
end
```

في هذا المثال قمت أولا بعمل تعريف للمتغير الخاص **Declaration** بالإسم **SpriteNum** و **SpriteNum** هي في الأساس قيمة متغيرة جاهزة تعني رقم العنصر الحالي, وهنا قمت بربطها كمتغير خاص, ثم في السطر التالي هناك المحدد **On BeginSprite me** وهو محدد بدء ظهور العنصر على شاشة المسرح **Stage** وفي السطر الذي يليه قمت بوضع المتغير الخاص في مكان رقم العنصر **Sprite** مع خاصية موقع العنصر **Loc** حيث **Loc** تعبر عن موقع العنصر **Location** ***(أنظر درس خصائص العناصر Sprite Properties في نفس هذا الفصل)***, وفي النهاية فإن موقع العنصر **Loc** أخذ قيمة النقطة **Point** على المحور السيني والصادي وهي تساوي **12**, إذن تفهم من المثال هنا أن المتغير هو متغير لرقم العنصر, وهذا الكود لو قمت بربطه مع أي عنصر على خط الزمن فإنه سينفذ نفس التعليمات والسبب في ذلك هو المتغير الجاهز داخل لغة اللينجو **SpriteNum** الذي قام المتغير الخاص بتحويله لمتغير يخص العنصر **Sprite** المرتبط معه.

لا يجب أن تخلط الفهم بين مصطلح **Property** كمتغير وبين مصطلح **Property** كخاصية للعنصر **Sprite** فالعناصر **Sprites** لها خواص مثل اللون **Color** أو الموقع **Loc** كالمثال السابق وغيرها الكثير مما ستعرفه لاحقا في نفس هذا الفصل, وهذه الخواص لها طرق معينه لتحديدها وهي أساسا جاهزة في قلب البرنامج واللغة بينما المتغير الخاص **Property** هو متغير يخص العناصر بسلوك معين كما رأيت في المثال السابق.

مثال(٢):

الأن قم بفتح الملف **Property_EX2.dir** من دليل **Essential** على القرص المرفق مع الكتاب, وقم بتشغيل البرنامج أولا, ماذا تلاحظ؟ إن البرنامج الحالي يقوم بنفس عمل البرنامج في المثال السابق, حسنا هيا بنا لكي نقوم بتحرير نص الكود المرتبط بالعنصر **1** ونقرأ ما في داخله:

```
property ThisSprite

on BeginSprite me
  ThisSprite = 1
  Sprite(ThisSprite).loc = point(12,12)
end
```

هذه المرة لم أستخدم المتغير الجاهز **SpriteNum** وأقوم بتخصيصه, بل قمت بإبتكار متغيري المخصص بالإسم **ThisSprite** ومن ثم قمت بإعطاء القيمة **1** لهذا المتغير وهي تعبر طبعا عن رقم العنصر **Sprite**, وبإستخدام نفس خاصية موقع العنصر **Sprite Location** قمت بتحديد نفس النقاط على المحاور السينية والصادية **X,Y** , ولكن الفرق بين هذا الكود وسابقه هو أن هذا الكود سيكون مخصصا فقط للعنصر رقم **1** ما لم يتدخل المبرمج ويعدل قيمة المتغير, الأن لاحظت الفرق بين متغير خاص قمت أنت بإنشائه وبين متغير خاص قام بتوظيف المتغير الجاهز **SpriteNum** الذي يعبر عن رقم العنصر.

مثال(٣):

حسنا, في هذا المثال سأقوم بعمل متغير لقيمة الموقع نفسه **Point** وهي تحمل القيمة **Point(12,12)** كما تعلم, الأن قم بفتح الملف **Property_EX3.dir** ثم قم بتشغيل البرنامج, تلاحظ انه يقوم بنفس عمل المثال السابق, الأن قم بفتح نافذة تحرير الكود لكي نرى ماذا فعلت من تغييرات:

```
Property SpriteNum
Property SpriteLocation

on BeginSprite me
  SpriteLocation = Point(12,12)
  Sprite(SpriteNum).Loc = SpriteLocation
end
```

كما ترى لقد قمت بتخصيص متغير خاص جديد بالإسم **SpriteLocation** ثم أسفل المحدد **BeginSprite** قمت بوضع القيمة ذاتها لموقع العنصر السيني والصادي بينما إستخدمتها في السطر الأخير لتحديد مكان العنصر الجديد.

نستنتج من تجاربنا السابقة مدى مرونة توظيف المتغير الخاص **Property** وفي نفس الوقت تعلمنا كيف وأن برنامجا واحدا يمكن أن يكتب بأكثر من طريقة, ويمكنك أنت أيضا إبتكار طرقا أخرى لبرمجة نفس هذا المثال.

مثال(٤):

حسنا, في هذا المثال سأقوم بتوظيف المتغير الجاهز **SpriteNum** توظيفا أفضل بحيث يفيدنا في تصميم زر بلغة اللينجو بالتعاون مع المتغير الخاص **Property**, الأن قم بفتح الملف **Property_EX4.dir** من دليل **Essential** على القرص المرفق مع الكتاب, وقم بتشغيل البرنامج ثم قم بالنقر بالزر الأيسر للماوس فوق المربع الأزرق الذي يظهر أمامك, انه يتحول للون الأحمر كما تلاحظ, عاود الضغط لكي تشاهد ذلك مرة أخرى, ثم قم بالخروج من البرنامج وقم بفتح كود العنصر لكي ندرسه معا.

```
property spriteNum
on mouseDown me
  sprite(spriteNum).member = member("DownPict")
end
on mouseUp me
  sprite(spriteNum).member = member("UpPict")
end
```

هنا قمت بتخصيص المتغير **SpriteNum** بالمتغير الخاص **Property** ثم قمت بتصميم محددين **Handlers** بالإسم **MouseDown** ثم **MouseUp** لكي أعين حدثا عند ضغط الزر الأيسر الماوس في المحدد الأول وحدثا آخر عند رفع الإصبع من الزر الأيسر للماوس في المحدد الثاني, فأسفل المحدد **MouseDown** قمت بتغيير شكل العنصر **Sprite** من العضو الحالي المرتبط به الى العضو **DownPict** من الكاست **Cast** بإستخدام الخاصية **Member** وهي خاصية تعيين عضو العنصر المرتبط بالعنصر على خط الزمن **Sprite**, فنحن نعلم أن العناصر **Sprites** مرتبطة تماما بما يمثلها في الكاست, وهذا التكنيك أدى الى شعور المستخدم بأنه نقر زرا ما بالماوس لأن الشكل تغير, وعند رفع الإصبع من زر الماوس الأيسر فإن العنصر يعود كما كان, يمكنك فتح نافذة أعضاء المشروع **Cast** لكي تشاهد بنفسك أسماء العناصر.

مثال(٥):

قم بتصميم سلوك جديد **Behaviour** وقم بربطه مع عنصر **Sprite 1** كما تعلمت في الدروس السابقة, يمكنك إستخدام زر **PushButton** من قائمة **Insert** على سبيل المثال ثم أكتب في داخله الأسطر التالية:

```
property pPropertyVar
on beginSprite
pPropertyVar = "I am a property"
end
```

الأن قم بتشغيل البرنامج وأثناء تشغيل البرنامج قم باستدعاء قيمة المتغير الخاص من نافذة التراسل**Message Window** بكتابة الأمر التالي **put sprite(1).pPropertyVar** والأن تلاحظ بعد إدخال الأمر بأن القيمة الخاصة للمتغير قد ظهرت داخل نافذة التراسل **"I am a property"" --** وفي حالة إستخدام المتغيرات الخاصة **Properties** فإن قيمها تخزن داخل ما يسمى **Script Instance list** وهي تحمل كل القيم المتغيرة الخاصة لكل متغير **Property** ويمكننا وصفها بأنها قائمة **List** كما سنتعلم فيما بعد أو مصفوفة تخزن بداخلها معلومات القيم المتغيرة.

مثال(٦)

في هذا المثال سأقوم بشرح كيفية التعامل مع متغير خاص **Property** بإضافة معامل خاص به **Parameter** وتعريب هذه الكلمة هو ***معامل*** أو ***بارميتر*** وهو تعبير يدل على تخصيص الشئ بميزة أو بمعامل معين يؤثر في سلوكه الأن قم بفتح مشروع جديد وأضف عنصرين جديدين **Sprite 1,2** في كل من القناة واحد واثنان على خط الزمن **Score** ثم أضف لكلاهما الكود التالي:

```
property pPropertyVar
on setProperty me, varData
pPropertyVar = varData
end
on getProperty me
put pPropertyVar
end
```

فقد تم إضافة المعامل المتغير **me** الى الكود لكي نوضح أن هذا الكود محدد ومخصص للعنصر الذي يرتبط معه الكود وبهذه الطريقة يمكن تخصيص قيمة المتغير **pPropertyVar** بحيث تحمل قيم مختلفة للعناصر **Sprites** التي من الممكن أن تكون على إرتباط بهم ويمكن أن نقوم بتطبيق ذلك الأن فمن خلال نافذة التراسل **Message Window** قم بكتابة الأكواد التالية:

```
setProperty sprite 1, "property of sprite 1"
setProperty sprite 2, "property of sprite 2"
```

لا تنسى ضغط مفتاح الإدخال **Enter** بعد كل سطر فالأسطر السابقة قد أضافت قيمة متغيرة لكل من المعامل **me** والقيمة المتغيرة **varData** والأن من نفس نافذة التراسل أكتب السطر التالي:

```
getProperty sprite 1
```

فتكون النتيجة هي **"property of sprite 1" --** وإذا قمت بكتابة الأمر **getProperty sprite 2** فستحصل على النتيجة **"property of sprite 2" --** وبذلك نستنتج من تجربتنا بأن القيمة المخصصة **pPropertyVar** يمكن تخصيصها بإستخدام المعاملات الخاصة المتغيرة **me**

ملحوظة هامة جدا: تذكر دائما قبل تجربة الأوامر من خلال نافذة التراسل أن تضغط على الزر ⚡ في نافذة الكود لكي يقوم دايركتور بمعالجة الكود في الذاكرة وتنجح عملية التراسل بين الكود ونافذة التراسل.

كما هو الحال مع المتغيرات العامة يمكن تعريف المتغيرات الخاصة بصورة مفردة أو مزدوجة فمثلا يمكنك كتابة **Property myName** أو يمكن كتابة **Property myName,myAge** وهذه الطرق لا تؤثر في سير برنامجك فيمكنك إختيار ما ترتاح إليه أثناء تصميمك لبرامجك الخاصة, ولكن يفضل تصميم المتغير الخاص بحيث يكون مفردا لكي يسهل لك توثيق البرنامج فيما بعد كما ستتعلم في نهاية هذا الفصل.

كلمة **me** في الدايركتور تعني معامل أو بارمتر كما ذكرت من قبل والمعامل أو البارمتر هو في الأصل متغير يضاف الى المحددات **handlers** لكي يضيف لها المرونة ولست في كل الأحوال أنت مضطر الى إضافة البارمتر **me** الى أكوادك التي تكتبها, فقط أضفها الى الأكواد التي تتوقع منها تحكما خاصا في عناصرك على المسرح **Stage** أو التي تنتظر منها رد فعل معين, ومن المفضل دائما إستخدام المتغيرات الخاصة **Properties** عن تلك العامة **Globals** خاصة اذا كنت تقوم بتصميم برامج كبيرة لأن المتغيرات العامة **Globals** غالبا ما تؤثر في سير برامجك وتحتفظ في الذاكرة بقيم قد لا تحتاج اليها بينما المتغيرات الخاصة فهي تحتفظ بالقيم المتغيرة في داخلها في قائمة خاصة دون أن تؤثر في برنامجك بأي شكل من الأشكال, وفي التطبيقات العملية بإذن الله تعالى سنقرأ الكثير عن طريقة إستخدام المتغيرات وستتعرف عمليا على فوائدها وقوتها.

تذكر دائما بأن أوامر ووظائف لغة اللينجو لا تستعمل كمتغيرات عامة أو مخصصة داخل برامجك فمثلا لا يمكنك مثلا أن تخصص **startmovie =1** أو أي معطيات أخرى خاصة بلغة اللينجو, ذلك خطأ فادح وسيؤدي لفشل برامجك في عملها أو الكثير من الأخطاء المنطقية, بعض أوامر اللغة نفسها تكون ثوابت **Constants** أو تسمح ببعض التخصيص وعدا ذلك فمن الأفضل تجنب إستخدام تعبيرات وأوامر اللغة كمتغيرات على أي مستوى كان.

مصطلحات مهمة تتعلق بالمتغيرات:

المتغيرات دائما ما تكون لها تعريف **Declaration** وبدء التخصيص **Intialization**, وهاتان العمليتان تعبران عن عملية تصميم المتغير كأن تكتب **global myName** وهنا نعرف المتغير العام **Declare The global variable** بينما عندما تكتب **myName = "Mohammed"** فهنا أنت قمت بعمل بدء التخصيص للمتغير بقيمة معينة وهي عملية **Intialize The global Variable**, وهذا هو مفهوم كلا من **Declaration** و**Intialization**, تستخدم هذه المصطلحات مع لغات البرمجة الأخرى ذات الطبيعة المعتمدة على الكائنات **Object Oriented Programming**.

إختبار المتغيرات بإستخدام VOID:

السؤال الأن هل يوجد طريقة لتتبع المتغيرات وإختبارها؟, لنفترض مثلا بأن المبرمج قام بعملية تعريف للمتغير مثل المتغير العام وكتب مثلا **global myVariable** ونسي تخصيص المتغير بقيمة معينة, فكيف يمكنه الكشف عن ذلك؟, قم بفتح مشروع جديد وقم بعمل كود مشروع **Movie Script** جديد ثم اكتب في داخله الأسطر التالية:


```
global myVariable
on StartMovie
  if myVariable = VOID then
    Alert"Your Varibale Not Intialized Yet!"
  end if
end
```

الأن قم بتنفيذ البرنامج حيث تلاحظ ظهور رسالة **ALERT** تنبيه, لأن المتغير لا يحمل قيمة, هنا في المثال تلاحظ أنه تم إستخدام قاعدة **IF** وهي قاعدة الشرط (***ستجد معلومات مفصلة عنها لاحقا في هذا الفصل***), هذه القاعدة **IF** تعبر عن حالة الشرط وهي (إذا) حيث نخبر البرنامج أنه في حالة تساوي المتغير **myVariable** مع **VOID** وهذه العبارة تعني أنه إذا لم يكن للمتغير قيمة حيث أن **VOID** تعبر عن عدم وجود قيمة منطقية قابلة للتعامل معها, ونكمل بالكود **THEN** وهو (إذن) قم بعرض رسالة التنبيه **ALERT**, بالطبع يمكن مستقبلا تطوير الكود السابق بحيث يقوم بعمل تخصيص للمتغير,هذه هي أبسط طريقة لإختبار متغير تسمى الطريقة المباشرة, يمكنك فتح ومشاهدة هذا المثال من الملف **Void_EX1.dir** في دليل **Essential** الموجود في الدليل المصدري للكود المرفق مع الكتاب.

إختبار المتغيرات بإستخدام VoidP:

سنفترض في هذا المثال بأن المبرمج قام بعملية تعريف للمتغير مثل المتغير العام وكتب مثلا **global myVariable** ونسي أيضا تخصيص المتغير بقيمة معينة, فلنفترض أنه كان ينوي تخصيص القيمة **5** للمتغير **myVariable** الذي قام بتعريفه, فكيف يستطيع المبرمج تجنبا للخطأ بإختبار حالة المتغير والتأكد إذا كانت هناك قيمة مخصصه له أم لا؟, خاصة وأن هناك برامج يمكن تصميمها قد تحتوي على مخصصات متغيرة للمتغير الواحد, حسنا دعونا نشاهد ذلك عمليا, الأن قم بفتح مشروع جديد وقم بعمل كود مشروع جديد **Movie Script** بالضغط على **Ctrl+0** كما تعلمت سابقا وأكتب الأسطر التالية

.


```
on startMovie
  global myVariable
  if VoidP(myVariable) then
    myVariable = 5
    Alert"Your Variable has no value and Will be Intialized by 5 Value"
  else
    Alert"Your Variable already Intialized"
  end if
end
```

الأن من نافذة التراسل **Message Window** قم بمسح جميع المتغيرات العامة للتأكد من خلو الذاكرة من المتغيرات العامه بكتابة الأمر **Clearglobals** ثم قم بتشغيل البرنامج, تلاحظ ظهور الرسالة **ALERT** التي تقول:
Your Variable has no value and Will be Intialized by 5 Value

الأن إضغط **OK** لغلق نافذة **ALERT** ثم قم بإعادة تشغيل البرنامج مرة أخرى, ماذا تلاحظ؟ لقد ظهرت لك رسالة **ALERT** جديدة تقول: **Your Variable already Intialized** ,الأن إضغط **OK** مرة ثانية لغلق نافذة **ALERT** , الأن قم بتشغيل البرنامج مرة أخرى, ماذا تلاحظ؟ الرسالة الأخيرة تعاود الظهور مجددا؟, والأن ماذا حدث؟ هيا بنا نعود الى أسطر الكود لكي ندرسه بعناية ونستوعب ماذا حدث تماما.

لقد بدأنا الكود بكتابة محدد بدء المشروع **StartMovie** ثم قمنا بتعريف متغير عام بالإسم **myVariable** وفي السطر الثالث قامت قاعدة الشرط **IF** وهي تعني حالة (إذا) وهنا يقول السطر إذا المتغير **myVariable** لا توجد له قيمة مخصصة إذن وهي هنا **THEN** قم بتخصيص قيمة له وهي **myVariable = 5** ثم تبع ذلك رسالة **ALERT** الأولى **Your Variable has no value and Will be Intialized by 5 Value** التي ظهرت لنا في أول مرة حيث لم تكن هناك قيمة مخصصة للمتغير العام **myVariable** ثم يكمل البرنامج بجملة وإلا **ELSE** وهي تعني وإلا فقم بعرض رسالة **ALERT** الثانية **Your Variable already Intialized** وهنا نفهم من ذلك أن السطر **if VoidP(myVariable)** هو السطر الأساسي في عملية إختبار وجود قيمة مخصصة للمتغير, فالوظيفة **VoidP** هي التي تقوم بعملية الإختبار, هذا المثال كان على المتغير العام فقط, لكن يمكنك إستخدام نفس هذه الوظيفة مع أي نوع من المتغيرات, كثيرا ما نرى هذه الوظيفة مستخدمة داخل كود السلوك **Behaviour**, قم بفتح الملف **VoidTesting.dir** من الدليل **Essential** لمشاهدة المشروع, الفرق الأساسي بين **VOID** و **VOIDP** هو أنك يمكنك إختبار المنطق لـ **VOIDP**, لعمل ذلك أكتب في نافذة التراسل **Message Window**, الأمر **Put VoidP(myVariable)** ثم تلاحظ ظهور الرقم **0** لإن المتغير له قيمة بينما يظهر الرقم **1** إذا لم يكن للمتغير قيمة,

إختبار المتغيرات بإستخدام ILK:

الأمر **ILK** له صلة مباشرة بإختبار معظم خصائص البرمجة, ويستطيع الكشف عن نوعية أجزاء البرنامج سواء كان مدخلات أو مخرجات أو أوامر, وله إستخدامات كثيرة جدا, ولكني سأعرض هنا فقط طريقة إستخدامه مع المتغيرات ولاحقا سنتعرض له تفصيلا بإذن الله تعالى في الفصول القادمة, أكتب معي الكود التالي في نافذة مشروع جديدة وقم بتشغيل البرنامج وإجعل نافذة التراسل**Message Window** أمامك بحيث تكون واضحة أثناء تنفيذك للبرنامج.

```
on StartMovie
  myVariable = the systemDate
  if ilk(myVariable, #date) then put myVariable
end
```

تلاحظ ظهور التاريخ **Date** في نافذة التراسل**Message Window** وهو أمر وظيفي **Function**, حسنا دعنا نراجع الكود السابق لكي نختبره ونرى ماذا حدث بالضبط, في الكود السابق وضعنا المحدد **StartMovie** ثم قمت بعمل متغير محلي **Local** وقيمة هذا المتغير هي نفس قيمة تاريخ النظام **SystemDate** , وفي السطر الثالث تلاحظ وجود قاعدة الشرط **IF** حيث تشترط هنا إذا كان نوع المتغير **ILK** الذي يحمل القيمة **myVariable** مساويا لقيمة التاريخ إذن **THEN** قم بوضعه **PUT** فالبرنامج السابق كله يمكن أن يكتب من نافذة التراسل بالشكل **Put the SystemDate** وسيعطيك نفس النتيجة, من هنا نفهم بأن **ILK** قامت بإختبار نوع المتغير بحالة شرطية, وهذه هي الفائدة الأساسية من **ILK** حيث حددت نوعا معينا للمتغير, يمكن فتح البرنامج السابق وهو **ilk_locVar.dir** من الدليل **Essential** من على القرص المرفق مع الكتاب.

في المثال السابق قمت بإستخدام الوظيفة **SystemDate** لمعرفة التاريخ, هناك أيضا **Date** وهناك الوقت **Time**, الغرض من وضع بعض الأمثلة التي ستجد تفاصيلها لاحقا هو التعرف على جميع الطرق الممكنة لكتابة الكود **Script** وطريقة تصميم المنطق الممكنة **LOGIC** حتى تتعود على طريقة التفكير الشاملة التي تستطيع توظيف كل إمكانيات اللغة معا لتحقيق

تلاحظ دائما إستخدام قاعدة الشرط (إذا, إذن, والإ) **IF THEN ELSE** في معظم البرامج أو المحددات التي تقوم بعملية إتخاذ القرار, إتخاذ القرار دائما يعتمد على قاعدة الشرط كما يعتمد على المنطق أيضا, ستجد في هذا الفصل معلومات مفصلة عن إستخدام الشرط وجميع ملحقاته وستجد العديد من التطبيقات دائما في الكتاب تعتمد على الشرط, في الواقع يكاد لا يخلو برنامجا مكتوبا من إستخدام قاعدة الشرط بشكل أو بآخر, في هذا الكتاب قد تضطر الى إعادة قراءة بعض الفصول أو الفقرات, أو قد تضطر للقفز قليلا للأمام ثم العودة مرة ثانية للوراء, لا بأس من ذلك ما دام الهدف هو الإستيعاب التام.

العمليات الرياضية في لغة اللينجو Math in Lingo:

لقد تعلمت من قبل كيف تتعامل اللينجو مع العمليات الحسابية البسيطة في الدرس السابق, ولكن ما زال هناك الكثير لكي تتعلمه بخصوص هذا الموضوع, ولغة اللينجو تحتوي على مجموعة تعليمات كاملة للتعامل مع الأرقام كما هو الحال مع أي لغة برمجة أخرى.

العمليات الحسابية على الأرقام:

يمكنك أداء أية نوع من العمليات على الأرقام من جمع وطرح وضرب وقسمة والرموز المستخدمة في مثل هذه العمليات هي +,-,*,/ والرمز * يستخدم في عملية الضرب وهو يرادف علامة الاكس المعروفة لدينا ولكن لغة اللينجو تستخدم هذا الرمز للتعبير عن عملية الضرب الحسابية والرمز / يستخدم في عملية القسمة الحسابية وبقية الرموز بالطبع معروفة لدينا.

واليك بعض العمليات الحسابية التي يمكنك تجربتها عبر نافذة التراسل **Message Window** .

```
put 4+5
--9
put 7-3
--4
put 6*6
--36
put 8/2
--4
put 9/2
--4
```

الأعداد الصحيحة والأخرى ذات النقطة العائمة:

لاحظ أن آخر عملية حسابية في السطر السابق وهي تسعة تقسيم اثنان قد أعطتنا النتيجة **4** وذلك سببه أن دايركتور يتعامل هنا فقط مع الأعداد الصحيحة ولذلك فالرقم **4.5** لا يعتبر صحيحا رغم أنه الإجابة الصحيحة فهذا الرقم هو رقم ذو نقطة عائمة **floating point number** أو يطلق عليه رقم عائم ويجب أن تعلم بأن الرقم **4** على سبيل المثال يعتبر رقما صحيحا وفي نفس الوقت يعتبر رقما ذو نقطة عائمة بينما الرقم **4.5** لا يعتبر رقما صحيحا إنما هو رقم ذو نقطة عائمة, وفي لغة اللينجو ستلاحظ أن جميع الأرقام ذان النقطة العائمة تظهر بفاصلة عشرية وهكذا يستطيع دايركتور التمييز بين كلا من نوعي الأرقام.

إذا لم تحدد لدايركتور نوعية العملية الحسابية التي تقوم بها فهو يهمل قيمة النقطة العائمة ولذلك عندما تسأل دايركتور عن قيمة **7/2** فهو يخبرك بأن النتيجة هي **3** وليس **3.5** .

يمكنك إجبار دايركتور على إستخدام العمليات الحسابية للنقطة العائمة **floating point** , قم بتجربة الأمثلة التالية في نافذة التراسل**Message Window**.

```
put 7/4
--1
put 7.0/4
--1.7500
put 7/4.0
--1.7500
put 7.0/4.0
--1.7500
```

يمكنك أيضا إستخدام وظائف اللينجو الحسابية **functions** مثل **integer** و **float**

```
put 7
--7
put float(7)
--7.0000
put 7.75
--7.7500
put integer(7.25)
--7
put integer(7)
--7
put integer(7.75)
--8
```

فهذه الوظائف **integer** و **float** تقومان بتحويل قيم الأرقام سواء كان رقما صحيحا أو رقما ذو نقطة عائمة الى الشكل الذي تريده فمعني **put float(7)** تعني ضع الرقم بصيغة النقطة العائمة بينما الأمر **put integer(7.75)** يعني أعطني الرقم الصحيح للقيمة **7.75** وهي هنا الرقم **8**, الأن يمكن تجربة إستخدام الوظائف داخل الأوامر لخلق عملية حسابية أكثر تعقيدا..قم بتجربة التالي.

```
put 8/5
--1
put float(8)/5
--1.6000
put float(8/5)
--1.0000
put integer(float(8)/5)
--2
```

لاحظ ان العملية الحسابية **put float(8)/5** يعادل الأمر **put 8.0/5** بينما الأمر **put float(8/5)** أعطانا النتيجة **1.0000** وذلك سببه هو اننا طلبنا من دايركتور أن يقوم بحل المسألة **8/5** أولا قبل أن يعطينا قيمة النقطة العائمة لها.

الأسبقية:

في العمليتين الحسابيتين الأخيرتين قمنا بعرض موضوع مهم, وهو موضوع أولوية التنفيذ أو أولوية تنفيذ العمليات والتي تسمى ***الأسبقية*** , فعندما تكون هناك أكثر من عملية سيتم حدوثها أو خليط من العمليات والوظائف يجب أن تعرف ما هي العملية التي سيقوم دايركتور بتنفيذها أولا؟, قم بتجربة ذلك الأن في نافذة التراسل **Message Window**.

```
put 5+3
--8
put 8*6
--48
put 5+3*6
--23
```

إضافة الخمسة الى الثلاثة سيعطينا رقم ثمانية وهذا أكيد, ضرب ثمانية في الرقم ستة سيعطينا **48**, ومع ذلك فلو حاولت عمل هذه العمليات دفعة واحدة وفي سطر واحد سوف تحصل على **23**, لماذا؟ حسنا, هناك بعض عمليات الضرب والجمع يقوم دايركتور بالتعامل معها أولا ثم في النهاية يقوم بالجمع أو الطرح ولذلك في الجملة **6*3+5** بدأت عملية الضرب أولا فأعطت النتيجة **18** ثم في النهاية جمع الناتج مع الرقم **5** فأعطانا النتيجة **23**, ومعظم لغات البرمجة تأخذ نفس المنوال عند تنفيذ العمليات الحسابية عليها بهذا الشكل.

وبالرغم من ذلك فيمكنك إجبار البرنامج أو العملية الحسابية بلغة اللينجو على أن تتمشى وفق وجهة نظرك أو وفق ما تريده فيمكنك تخطي نظام الأسبقية, الأن قارن بين العمليتان الحسابيتين في نافذة التراسل**Message Window**.

```
put 5+3*6
--23
put (5+3)*6
--48
```

ان استخدام الأقواس هنا قام باجبار الدايركتور على السير وفق طريقة معينة فاستخدام الأقواس يجبر دايركتور على إجراء العمليات الحسابية داخل الأقواس أولا ثم من بعدها يقوم بالعملية خارج الأقواس وهي الضرب في حالتنا هنا, ويمكنك تجربة التالي في نافذة التراسل **Message Window**.

```
put ((5+3)*6+(7-3)*4)*8
--512
```

العمليات الحسابية والمتغيرات:

الأن بعد أن فهمت العمليات الحسابية الأساسية ستكون الخطوة التالية هي عمل تطبيق هذا الفهم مع المتغيرات, فسنقوم هنا بإستخدام المتغيرات مع العمليات الحسابية لكي نقوم بتوسعة مداركنا بشكل أكبر, قم الأن بتطبيق هذه الأمثلة في نافذة التراسل **Message Window**.

```
myNumber = 5
put myNumber
--5
put myNumber+1
--6
put myNumber*7
--35
put (myNumber+2)*3
--21
```

ويمكن أيضا عمل متغير لجعله يساوي قيمة عملية حسابية ما.

```
myNumber = 7+4
put myNumber
--11
myNumber = (7+4)*2
put myNumber
--22
myNumber = 5
myOtherNumber = 6
mySum = myNumber+myOtherNumber
put mySum
--11
```

في المثال الأخير رأيت كيفية توظيف المتغير في عملية حسابية ما وكيف يستطيع متغير ما من أن يحمل قيمة رقمية معينة بل وتستطيع إجراء العمليات الحسابية على المتغيرات ما دامت تحمل قيم رقمية مفهومة مثل المثال السابق.

جدول الوظائف الرياضية في لغة اللينجو:

الوظيفة الرياضية	الوصف	مثال شكلي
abs	مطلق: وهي وظيفة إعطاء القيمة المطلقة للرقم فهي تحول أي رقم ذو قيمة سالبة لرقم مطلق	abs(–7) = 7
Log	يعطي قيمة اللوغاريتم الطبيعي لرقم عشري أكبر من الصفر	Answer = Log(10.5)
exp	تعطي قيمة اللوغاريتم الطبيعي المرتكز على القيمة (2.7183)	exp(3) = 20.0855
float	يحول الرقم الى قيمة النقطة العائمة	float(4) = 4.0000
integer	رقم صحيح:يقرب أرقام النقطة العائمة الى أقرب رقم صحيح	integer(7.8) = 8
mod	يستخدم لعمل عمليات modulus التي تقوم بقسمة عدد صحيح على عدد صحيح آخر مع إظهار قيمة العملية الصحيحة المتبقية	4 mod 3 = 1
sqrt	مربع: القيمة التربيعية لرقم ما	sqrt(4) = 2
Power	القوة أو الأس لعدد ما مثل القوة 2 للعدد 10 وهي تساوي مائة في المثال الشكلي	Power(10,2)
tan	ظا: تعطي ظل الزاوية	tan(–1) =1.5574
atan	ظتا: تعطي ظل تمام الزاوية فهي تستخدم نظام حساب المثلثات بدلا من النظام القياسي	atan(1.0) = 0.7854
sin	جا: قيمة جيب الزاوية	sin(3.14/2) = 1.000
cos	جتا: تعطي قيمة جيب تمام الزاوية	cos(3.14) = –1.000
PI	القيمة الرياضية لباي في حساب الدائرة	PI =3.1416

جدول الخصائص المتعلقة بالعمليات الرياضية في لغة اللينجو:

الخاصية الرياضية	الوصف	مثال عملي
MAX	مهمة الخاصية وضع أعلى قيمة من مجموعة	put max(6,9,12) --12
MIN	مهمة الخاصية وضع أقل قيمة من مجموعة	put min(5,4,3) --3
floatPrecision	تحدد دقة معامل النقطة العائمة قبل تطبيق العمليات الحسابية على أرقام ذات نقطة عائمة المعامل يجب أن يكون رقم صحيح ما بين 1 و 15	the floatPrecision = 3 put sqrt(3.0) --1.732
NAN	رد فعل يحدث إذا كانت قيمة عملية حسابية لا تعطينا قيمة رقمية NAN = Not A Number	put sqrt(-1) --NAN
INF	رد فعل يحدث إذا كانت قيمة عملية حسابية ما تعطينا قيمة تبدو كأنها لا نهائية INF =Infinite	put power(10,9999)*power(10,9999) --INF

في الأمثلة العملية في الجدول السابق قمت بالإعتماد على نافذة التراسلMessage Window لتنفيذ الأوامر ويمكنك أنت أيضا تجربتها عمليا.

تطبيقات على الوظائف الرياضية:

الأن سأقوم بتقديم طرق إستخدام وتوظيف الأوامر التي قمت بعرضها سابقا في شكل جداول لكي تتعرف بنفسك على الطريقة المثلى لإستخدامها, وسأقوم بالإختبار من خلال نافذة التراسل **Message Window**, ولذلك قم الأن بفتح نافذة التراسل , يمكنك فتحها مباشرة عبر الإختصار **Ctrl+M**.

الأن أكتب أمر المطلق **abs** وهو وظيفة إعطاء القيمة المطلقة للرقم فهي تحول القيمة السالبة **Negative** لقيمة مطلقة.

```
put abs(-7)
--7
put (-7).abs
--7
```

في المثال السابق قمت بكتابة الأمر مرتان متتاليتان ولكن بشكل مختلف فالطريقة الأولي هي الطريقة المطولة **Verbose Syntax** بينما الطريقة الثانية هي الطريقة المنقوطة **Dot Syntax** كما ذكرت سابقا, والأفضل أن تعود نفسك على طريقة واحدة, عموما لا يوجد مشكلة في معرفة الإثنان معا خاصة في الوظائف الرياضية لأنها تعبيرات برمجية سهلة الحفظ والكتابة.

يمكن طبعا عمل متغيرات خاصة ودمجها مع الوظائف الرياضية مثل عمل متغير يحمل قيمة جذر تربيعي, أنظر معي

```
Number4Square = Sqrt(4)
put Number4Square
--2
```

يمكنك أيضا تعريف هذا المتغير بالطريقة التي تفضلها حسب سير برنامجك فيمكنك تعريفه على أنه محلي أو عام أو مخصص كما تعلمت سابقا.

يمكنك دمج عدة وظائف رياضية معا كما المثال التالي:

```
put (Power(10,2))+(Power(10,2))
--200.0000
```

في المثال السابق أخبرت الدايركتور أن يضع قيمة الرقم **10** مرفوعا بالقوة (الأس) **2** مجموعا لنفسه وهي تساوي **200**, لاحظ أني وضعت الوظيفة (الأس) **Power** داخل قوسين وذلك ضروري لسير المنطق في اللينجو, لأنك عندما تريد معرفة العملية الرياضية للرقم **10** مرفوعا بالقوة **2** فإنك تكتب **Put Power(10,2)** وذلك كعملية بمفردها لكن في حالة وضع عمليتان معا فذلك يستلزم وضع الأقواس, لذلك فإنتبه جيدا لذلك أثناء تصميمك لمعادلاتك الرياضية المدمجة الخاصة.

الأن جرب معي عمل تعشيش **Nesting** للأمر **power** عبر وضع عناصره الرقم والأس في شكل متغيرات:

```
powerOF10 = Power(10,2)
ThePowerItSelf = 2
Put Power((powerOF10),(ThePowerItSelf))
--10000.0000
```

الأن هل لاحظت كم هي مرنة لغة اللينجو؟ يمكنك أن تبتكر العديد من الأشكال الرياضية التي توفر عليك الكثير من الوقت أثناء عملك بالبرمجة.

من الأشياء المهمة جدا والتي يجب عليك أن تتذكرها بإستمرار أن هناك ثوابت عددية في العمليات الرياضية, فمثلا لا يمكنك أن تقوم بوضع **PI** كمتغير لإنها ذات قيمة ثابته وهي **3.1416** , جرب بنفسك

```
PI = "Mohammed"
put PI
--3.1416
```

من الضروري الإنتباه لذلك ومعرفة أن هناك كلمات محجوزة لا يمكن تخصيصها, في الواقع كل ما هو مخصص للغة اللينجو من أوامر وتعليمات ووظائف لا يجب تخصيصه كمتغير الإ إذا كان الأمر نفسه متغير وإلا فإن ذلك يؤثر على المنطق ويؤدي لأخطاء ويعطي نتائج غير مضمونة.

تطبيقات على الخصائص الرياضية:

تمكنك الخصائص الرياضية المضافة الى لغة اللينجو من التدخل في طريقة سير برنامجك وتوجيهها كما تريد فلو عدت معي إلى جدول الخصائص المتعلقة بالعمليات الرياضية في لغة اللينجو ستجد خواص مهمة جدا أهمها الخاصية **floatPrecision** وهي خاصية تحدد دقة معامل النقطة العائمة قبل تنفيذ الأوامر المتعلقة بالنقطة العائمة, فالهدف من ذلك هو تخصيص مدى دقة المخرجات التي يتعامل معها دايركتور مع المستخدم أو مبرمج البرنامج في إجراء العمليات الحسابية المتعلقة بحسابات رياضية ينتج عنها مخرجات ذات فواصل عشرية أو بما يعرف بالنقطة العائمة كمصطلح رياضي **float** وأقصى قيمة يمكن تخصيصها لمعامل دقة النقطة العائمة هي **15** بينما الرقم الإفتراضي الذي يخصص من قبل لغة اللينجو مسبقا هو الرقم **4** وجميع القيم التي يمكن تخصيصها لمعامل الدقة العائمة يجب أن تكون صحيحة فيما بين **(1-15)** كما ذكرت سابقا, والأن أكتب في نافذة التراسل **Message Window** الأمر التالي لتجربة تأثير التغير لمعامل دقة النقطة العائمة:

```
the floatPrecision =3
put Sqrt(3.0)
--1.732
the floatPrecision =8
put Sqrt(3.0)
--1.73205081
put the floatPrecision
--8
```

الأن لاحظت الفرق وعرفت كيف وأن دقة القيمة للنقطة العائمة تتغير حسب تغير معامل دقتها, وكما ترى في السطر قبل الأخير أنه بإمكانك سؤال دايركتور عن قيمة معامل النقطة العائمة, جرب تغيير معامل الدقة وقم بوضع قيمة **PI** ولاحظ الفرق.

توسعة قدرات اللينجو الرياضية!:

هناك طريقتان يمكن إستخدامهم لتوسعة قدرات لغة اللينجو الرياضية, الأولي هي تصميم سلوكيات **Behaviour** برمجية تكون موجهة ومرنة وقادرة على تنفيذ عمليات حسابية معقدة بسهولة, والثانية والأسهل هي إستخدام الإضافات **Xtras** وهناك العديد من الإضافات التي تقوم بمعالجة عمليات حسابية معقدة وتختصر وقت برمجتها منفصلة بإستخدام لغة اللينجو, وسأشرح في الأمثلة القادمة طريقة إستخدام **FreeConvert Xtra** وهي **Xtra** مجانية مضمنة مع قرص الكتاب في دليل **FreeXtras**, ولذلك فإذا لم تكن قد قمت بتنصيب هذه الإضافة داخل الدايركتور فيجب عليك فعل ذلك كما هو مدون في صفحة إستخدام القرص المرفق مع الكتاب.

إستخدام وبرمجة ConvertXtra للتحويل بين الأنظمة الرقمية:

لا تستطيع لغة اللينجو التحويل بين أنظمة الأرقام الثنائية والعشرية والسدس عشرية ويستلزم أداء هذه العمليات برمجتها من خلال تصميم كود خاص بها, وتوفر هذه الإضافة المجانية التي قام ببرمجتها المهندس أندراس كينز **Andras Kenz** نظاما سهلا للبرمجة للتحويل بين الأرقام وأنظمتها المختلفة, لمزيد من المعلومات حول هذه الإضافة وللحصول على الدعم الفني رجاء زيارة الموقع **http://freextras.freeweb.hu/**.

تشمل الإضافة ستة وظائف مختلفة للتحويل وسيلي الأن شرحهم, ولذلك قم بتجهيز وفتح نافذة التراسل **Message Window** حيث ستتم تجربة الوظائف من خلالها الأن بإستخدام الأمر **Put**.

الوظيفة الأولى ()FreeDecToBin:

هذا الأمر يقوم بتحويل رقم النظام العشري الى رقم بالنظام الثنائي.

```
put FreeDectoBin(168)
-- "10101000"
```

الوظيفة الثانية()FreeDecToHex:

هذا الأمر يقوم بتحويل رقم النظام العشري الى رقم بالنظام السدس عشري.

```
put FreeDectoHex(168)
-- "a8"
```

الوظيفة الثالثة()FreeBinToDec:

هذا الأمر يقوم بتحويل رقم بالنظام الثنائي الى رقم بالنظام العشري.

```
put freeBinToDec("10101000" )
-- 168
```

الوظيفة الرابعة ()FreeBinToHex:

هذا الأمر يقوم بتحويل رقم بالنظام الثنائي الى رقم بالنظام السدس العشري.

```
put freeBinToHex("10101000" )
--"a8"
```

الوظيفة الخامسة FreeHexToDec():

وهذا الأمر يقوم بتحويل رقم النظام السدس عشري الى رقم بالنظام العشري.

```
put freeHexToDec("a8" )
--168
```

الوظيفة السادسة FreeHexToBin():

وهذا الأمر يقوم بتحويل رقم النظام السدس عشري الى رقم بالنظام الثنائي.

```
put freeHexToBin("a8" )
--"10101000"
```

تستطيع هذه الإضافة **Xtra** أن تعمل على الدايركتور بدءا من الإصدار **8.5** حتى الإصدار **MX 2004**, وكما رأيت مدى السهولة الشديدة في إستخدام الإضافة بدلا من برمجة سلوكيات معقدة للقيام بعمليات التحويل, تهدف الإضافات دائما الى توفير الوقت اللازم لتطوير التطبيقات كما أنها تقوم بالعمليات التي يصعب تنفيذها أو تأخذ وقتا وجهدا كبيرين من المستخدم أو المبرمج, ويمكنك زيارة الموقع **www.mediamacros.com** لكي تقرأ العديد من الأكواد **Scripts** المكتوبة لحل العديد من المشاكل الرياضية التي قد تواجهك, وسنتعرض لاحقا في الفصول القادمة لتوظيف العمليات والوظائف الرياضية في حل المشاكل البرمجية المفترضة.

إستخدام المتغيرات الحلقيةSTRINGS VARIBALES:

يقصد بكلمة ***حلقات*** هي المتغيرات التي تحمل بداخلها معلومات نصية من حروف أو رموز, فالمتغير الحلقي قد يحمل حرف واحد أو رمز أو حرف ورقم معا أو من الممكن أن يكون عبارة عن جملة كاملة أو عدة كلمات أو حتى عدة أسطر أو من الممكن أن يكون عبارة عن صفحات ولكن ما يميز المتغير الحلقي أنه دائما ما يكون داخل علامات تنصيص مما يدل على طبيعته بأنه يحمل معلومات نصية والأمثلة القادمة يمكنك تجربتها داخل نافذة التراسل **Message Window**.

```
myString = "A"
put myString
--"A"
myString = "Hello"
Put myString
--"Hello"
myString = "Hello World"
put myString
--"Hello World"
myString = " "
put myString
--" "
```

قمنا هنا بعمل متغير حلقي له القيمة **A** ثم طلبنا من دايركتور أن يضع القيمة على الشاشة وهي **"A"**, وهكذا في بقية الأمثلة, حتى أنه من الإمكان وضع قيمة نصية فارغة كما هو المثال الأخير وسيقبل دايركتور به كما ترى.

التعبيرات المستقطعة من الحلقاتCHUNK EXPRESSIONS:

يمكنك إستخدام اللينجو لكي تستخلص من المتغيرات الحلقية حرف واحد أو مجموعة من الحروف, وهذا الحرف أو المجموعة الفرعية من الحروف تسمى ***تعبيرمستقطع Chunk***, ولكي نقرب المفهوم أكثر يمكنك تجربة وظائف الكلمات مثل **Char** أو **Word** كما سنفعل الأن في تجربتنا من خلال نافذة التراسل **Message Window**.

```
myString = "Hello World"
put Char 1 of myString
--"H"
put Word 1 of myString
--"Hello"
put Char 2 to 4 of myString
--"ell"
```

في الأمثلة السابقة قمنا بعمل متغير حلقي وأعطيناه القيمة **Hello World** ومن ثم طلبنا من الدايركتور أن يضع لنا الحرف الأول من المتغير ففعل ووضع لنا الحرف **H** ومن ثم طلبنا منه وضع الكلمة الأولى من المتغير فوضع **Hello** ثم بعد ذلك طلبنا منه وضع الحرف الثاني الى الحرف الرابع من المتغير فأعطانا **ell**.

يلاحظ المستخدم أننا في الأمثلة الأخيرة قمنا بإستخدام التعبير **of** مثل جملة **put Char 2 to 4 of myString** وهذه الطريقة القديمة لم تعد تستخدم إلا قليلا فبدئا من دايركتور ٨ الى الأن فقد قامت ماكروميديا بوضع أوامر منقوطة **dot Syntax** بدلا من إستخدام أوامر الجمل الطويلة **verbose Syntax** لكي تمكنك من تطبيق نفس الأمر ولكن مع إستخدام أكثر الطرق إختصارا لكتابة الأمر فيمكننا إعادة كتابة الأوامر السابقة بالشكل التالي:

```
myString = "Hello World"
put myString
--"Hello World"
put myString.char[1]
--"H"
put myString.char[2..4]
--"ell"
```

في الواقع نحن المبرمجين والمطورين عبر الدايركتور نعيب على ماكروميديا أنها قامت بوضع طريقتين لكتابة الأوامر داخل الدايركتور لكنا المميز في طريقة الجمل الطويلة **verbose Syntax** هو أنها طريقة قريبة جدا من اللغة الانجليزية ويمكن على سبيل المثال للمبرمج المبتدئ بأن يقوم بإستخدامها ولو على الأقل في بعض التعبيرات البرمجية البسيطة حتى يعتاد أسلوب البرمجة ومن ثم ينتقل إلى الطريقة الثانية **dot Syntax** ولو تريد أن تأخذ رأيي فأنا أقوم بدمج الطريقتان معا ولكني أقوم بتخصيص مجموعة أوامر معينة قمت بتعلمها وأكتبها بالطريقة المطولة بينما أخصص أوامر أخرى وأكتبها بالطريقة المنقوطة وذلك لكي لا أخلط بين طرق كتابة الأمر الواحد, وماكروميديا لم تعد تقدم الدعم الفني لطريقة **verbose synatax** داخل دايركتور ام اكس ٢٠٠٤ الجديد والطريف في الأمر إن ما زال موقعها الرسمي يقدم دروسا وأمثلة في لغة اللينجو بالطريقة المطولة, في دايركتور ام اكس ٢٠٠٤ تستطيع كتابة الأوامر بالطريقة المطولة لكنك لن تجد في مساعدة البرنامج أي دليل يرشدك لذلك, كذلك قامت ماكروميديا بتعديل طرق كتابة معظم أوامر لغة اللينجو لكي تكون قريبة جدا من طريقة كتابة أوامر وتعبيرات لغة الجافا سكريبت المزودة داخل دايركتور ام اكس ٢٠٠٤, سبب ذلك هو رغبة ماكروميديا في إستقطاب أكبر عدد ممكن من مطوري الجافا ليقوموا بالتطوير تحت مظلة دايركتور, لا تقلق فدايركتور يتوافق مع الطرق القديمة للينجو.

هناك العديد من الأوامر الأخرى التي تستخدم مع المتغيرات الحلقية والتعبيرات المستقطعة منها فهناك الأمران **line** والأمر **paragraph** اللذان يقومان بالتحكم في الأسطر والفقرات, وأيضا يوجد الأمر **item** وهو الذي يعطينا مقطع من المتغير الحلقي في حالة إذا كانت مقاطع المتغير مفصولة بفاصلة مثل **"Apple, Orange"** ويمكنك الأن تجربة ذلك في نافذة التراسل **Message Window**.

```
myString = "apples, oranges, pears, peaches, bananas"
put item 2 of mySring
--"Oranges"
put myString.item[3]
--"Pears"
```

وكما ترى في المثال السابق كيف قمنا باستخدام الأمر **item** بكلا الطريقتين لكتابة الكود.

في المثال السابق قلنا بأن الأمر **item** يستخدم لإستقطاع العناصر الموجودة في المتغيرات الحلقية والمفصولة بفاصلة, وهذا صحيح في الأحوال العادية لكن من الممكن للمبرمج أن يعين للأمر **item** المعامل الخاص به الذي يستبدل الفاصلة العادية فيمكنك إفتراض تعيين متغير حلقي على شكل جملة تحوي عدة كلمات مفصولة عن بعضها بفاصلة منقوطة ; وهذا المعامل يسمى **itemDelimiter** ويستخدم كما رأيت بنفس التهجئة عند إستخدامه, ويمكنك التعرف عليه عن قرب من خلال المثال التالي من خلال نافذة التراسل **Message Window**.

```
myString = "walnuts;peanuts;pears;peach;sunflower seeds"
the itemDelimiter = ";"
put item 2 of myString
--"peanuts"
put myString.item[3]
--"sunflower seeds"
```

المثير في المتغيرات الحلقية هو أنه بإمكانك إستخلاص المعلومات منها دفعة واحدة من خلال مرونة اللينجو الكبيرة أثناء التعامل معها فيمكنك الحصول على حرف من كلمة من سطر, يمكنك تجربة التالي في نافذة التراسل **Messagw Window**.

```
the itemDelimiter = ","
myString = "red,yellow,blue,green,light brown"
put myString.item[5]
--"light brown"
put myString.item[5].word[2]
--"brown"
put myString.item[5].word[2].char[4]
--"w"
put Char 4 of Word 2 of Item 5 of myString
--"w"
```

جدول الأوامر المستخدمة مع المتغيرات الحلقية والتعبيرات المستقطعة:

الوظيفة	الوصف	مثال
Char	حرف مفرد أو مجموعة من الحروف.	myString.char[1] myString.char[2..4]
Word	مجموعة مفردة من الحروف أو كلمات مفصولة بمسافة.	myString.word[1] myString.word[2..4]
Item	عنصر وهو المتغير الحلقي المفصول بفاصلة أو بمعامل متحدد في الـItemDelimiter.	myString.item[2..4]
ItemDelimiter	معامل تحديد نوع الفاصلة بين أجزاء المتغير الحلقي فبدلا من الفاصلة يمكن تحديد فاصلة منقوطة مثلا.	the itemDelimiter = ";"
line	مجموعة مفردة من الحروف مفصولة عن بعضها بمفتاح Enter.	myString.line[1] myString.line[2..4]
paragraph	مجموعة مفردة من الأسطر مفصولة عن بعضها بمفتاح Enter.	myString.paragraph[1] myString.paragraph[2..4]

عمليات أخرى على المتغيرات الحلقية:

يمكنك إجراء العديد من العمليات الأخرى على المتغيرات الحلقية, مثل الإضافة لها أو وضع مسافة بين جملة في متغير حلقي لتتحول الى كلمات مثلا, من نافذة التراسل **Message Window**, أكتب التالي:

```
myString = "Hello"
put myString&"World."
--"HelloWorld."
myOtherString = "World."
put myString&myOtherString
--"HelloWorld."
put myString&&myOtherString
--"Hello World."
```

في السطر الأول قمت بإستخدام متغير حلقي له قيمة وهي الكلمة **Hello** وفي السطر الثاني قمت بإضافة كلمة **World.** بإستخدام العلامة **&** فهي تستخدم لجمع حلقتين معا ولذلك كان الناتج هو **HelloWorld.** وفي المثال الذي يليه قمت بعمل متغير حلقي وشاهدنا كيف وأننا أعدنا إستخدام العلامة **&** لإضافة المتغير الحلقي الجديد الى القديم الذي قمنا بعمله من قبل, وفي المثال الأخير قمنا بوضع زوجين من العلامة **&** لتصبح **&&** وهذا يعني في لغة اللينجو وضع مسافة وكأن المبرمج قد قام بضغط مفتاح المسافة **Space Bar** ولكننا قمنا بإستخدامها مع المتغيرين الحلقيين الذين قمنا بإنشائهما من قبل, يمكن الأن إجراء بعض التجارب المشابهة بنفسك, هذه العلامة **&** تسمى علامة التسلسل أو الربط.

يمكنك إستبدال محتويات متغير حلقي بطريقة أخرى بإستخدام الأمر **put** بإستخدام الوظائف **after,before,into** ويمكنك تجربة التالي بنفسك في نافذة التراسل **Message Window**.

```
myString = "Moham"
put "med" after myString
put myString
--"Mohammed"
put "My Name is" before myString
put myString
--"My Name is Mohammed"
```

الأن لاحظت انه بإمكانك إستخدام الأمر **put** لإضافة النصوص الى المتغيرات الحلقية بينما كان بإمكانك فعل الشئ ذاته بإستخدام علامة التسلسل **&** التي قمت بشرحها من قبل, ولكن كان من الضروري تعلم الطرق السابقة لأنها هامة ومفيدة, يوجد طريقة أخرى جيدة لوضع مسافة فارغة بين كلمتين,جرب التالي في نافذة التراسل **Message Window**.

```
myString = "HelloWorld"
put " " after myString.char[5]
put myString
--"Hello World"
```

الأن تبقى لنا معرفة طريقة إستخدام الوظيفة **into** لخدمة المتغيرات الحلقية فهذه الوظيفة تستخدم مع الأمر **put** لإدخال حرف أو كلمة أو جملة داخل متغير حلقي أو قد تستخدم لإستبدال جزء من المتغير الحلقي, ويمكنك الأن فتح نافذة التراسل **Message Window**.

```
myString = "Hello World"
put "Earth" into myString.char[7..11]
put myString
--"Hello Earth"
```

والأن كما تعلمنا الأمر **Put** وهو يعني ضع هناك الأمر **delete** وهو يعني إمسح ويمكنك تجربته من خلال نافذة التراسل **Message Window**.

```
myString = "Hello World"
delete myString.char[2]
put myString
--"Hllo World"
```

الأمر **delete** يعمل مع كلا من الحروف والكلمات والأسطر والعناصر.

مقارنة المتغيرات الحلقية منطقيا:

أحد أهم المهام التي سوف ترغب في أن تؤديها لغة اللينجو لك هي إتخاذ القرارات, ولكي يقوم الكمبيوتر بإتخاذ القرار فيجب تغذيته بالمعلومات, والكمبيوتر لا يفهم المعلومات إلا على الصورة صفر وواحد, والصفر يعني قيمة مهملة **FALSE** بينما الواحد فيعني له قيمة حقيقية **TRUE**.

ولغة اللينجو تستخدم هذه الطريقة عند إتخاذ القرار, فعندما تقوم بمقارنة المتغيرات على سبيل المثال فإنك غالبا تريد معرفة إذا ما كان متغير ما يساوي قيمة ما أو هل سيتساوى متغير مع متغير آخر والإجابة قد تكون بنعم أو بلا, وكمثال عملي على ما نقول فرجاء فتح نافذة التراسل**Message Window**, وأكتب التالي:

```
put 1=1
--1
put 1=2
--0
put"abc" = "def"
--0
put"abc" = "abc"
--1
```

يمكنك ملاحظة أنه إذا كانت الجمل حقيقية في الأمثلة السابقة فإن دايركتور يعطي القيمة واحد بينما إذا كانت الجمل مخالفة للمنطق فدايركتور يعطي القيمة صفر, وهي تعادل **TRUE** و**FALSE**, جرب التالي للتأكد..

```
put TRUE
--1
Put FALSE
--0
```

لا تستخدم فقط المعامل يساوي = في مقارنات المنطق مع المتغيرات فهناك طرق كثيرة أخرى بجانب هذا المعامل, والجدول التالي يوضح لك المعاملات الأخرى المنطقية التي يمكن إستخدامها.

جدول معاملات المقارنة المنطقية:

الوظيفة	الوصف(على المتغيرات الرقمية)	الوصف(على المتغيرات الحلقية)
=	هل الرقمان متساويان؟, هل القيمتان متساويتان؟	هل المتغيرين الحلقيين متساويين؟
<	هل الرقم الأول أصغر من الرقم الثاني؟	هل المتغير الحلقي الأول أصغر من الثاني؟
>	هل الرقم الأول أكبر من الرقم الثاني؟	هل المتغير الحلقي الأول أكبر من الثاني؟
<=	هل الرقم الأول أصغر من أو يساوي الثاني؟	هل المتغير الحلقي الأول أصغر من أو يساوي الثاني؟
>=	هل الرقم الأول أكبر من أو يساوي الثاني؟	هل المتغير الحلقي الأول أكبر من أو يساوي الثاني؟
<>	هل الرقم الأول لا يساوي الثاني؟	هل المتغير الحلقي الأول لا يساوي المتغير الحلقي الثاني؟

ويمكنك تجربة هذه المعاملات بنفسك من خلال نافذة التراسل **Message Window**.

```
put 1<2
--1
put 2<1
--0
put 2<=1
--0
put 2<=2
--1
put 2<=3
--1
put 2>=1
--1
put 2<>1
--1
put"this">"that"
--1
```

من المهم معرفة إن عند إستخدام المتغيرات الحلقية في عقد مقارنة بلغة اللينجو بإستخدام التساوي = فإن دايركتور لا يفرق بين **ABC** و **abc** بينما إذا قمت بإستخدام المعامل ‹ أو › فإن طريقة إستجابة اللينجو تختلف فإذا قمت بتجربة ذلك عمليا فستجد أن دايركتور يعطي الرقم واحد إذا وضعت **put "ABC" < "abc"** بينما يعطي صفر إذا عكست الإشارة, وذلك سببه أن دايركتور يتعامل بطريقة مختلفة في حالة وجود أحرف كبيرة **Capitals** وأخرى صغيرة **Smalls** ويمكنك تجربة أوضاع أخرى مثل **"Abc"** و **"aBc"** ولاحظ النتيجة وقارن بنفسك.

الوظائفFucntions:

سبق أن تحدثت بتعمق عن المحددات **Handlers** وعن محددات الأحداث **Events** والمحددات القياسية التي تأتي مع لغة اللينجو في الفصل الثاني, وهذه المرة سنقوم بعمل تطبيقات مهمة لما يسمى الوظائف **Functions** على هذه المحددات لأن بعض المحددات الخاصة التي ننشأها يمكن ان نطلق عليها المسمى **Fucntion** أي وظيفة وسبب هذه التسمية أن هذه المحددات تقوم بإعطائنا نتائج وقيم معينة **Values** لدي تنفيذها, فهي تعمل كأنها وظيفة حسابية **Math Function** على سبيل المثال لا الحصر, والمختلف بالطبع هو أنك تقوم بنفسك بتصميم وظيفة ما متخصصة وتقوم بدور معين.

هناك عنصرين أساسين يمكنك من التفرقة بين المحدد العادي وبين المحدد الوظيفي الذي يقوم بأداء عملية معينة وهما الدخل والخرج **Input and Output** فالمحدد الوظيفي عادة يقبل قيمة أو أكثر كدخل له ويقوم بالمقابل بإعطاء قيمة لنتيجة العملية الحسابية أو الوظيفية التي يقوم بها, ويسمى الدخل بالبارميتر **Parameter** بينما يسمى الخرج بالقيمة المعطاة **Returned Value**.

ولعمل محدد يقبل معامل **Parameter** فكل ما عليك فعله هو إضافة إسم المتغير لسطر تعريف المحدد وهو السطر الذي يبدأ بكلمة **On**, وإليك مثال على ذلك, قم بفتح نافذة **Movie Script** بإحدى الطرق التي تعلمتها من قبل واكتب التالي:

```
on myHandler myNumber
put myNumber
end
```

المثال السابق لا يوضح محدد وظيفي حقيقي لأنه لا يقوم بإعطاء قيمة حقيقية في المقابل لكنه في المقابل هو تصور مثالي لشكل المحدد الوظيفي, فبعد تعودك بإذن الله تعالى على كتابة برامجك الخاصة ستقوم بكتابة محددات كثيرة من هذا النوع, والأن كل ما ستفعله أنك ستقوم بإستبدال المتغير **myNumber** في سطر تعريف المحدد بقيمة حقيقية لكي تحضر المحدد الوظيفي للعمل والأن إفتح نافذة التراسل**Message Window** مع بقاء الكود في **Moive Script** أكتب:

```
myHandler(7)
--7
```

والأن لقد لاحظت أن الرقم **7** هو البارميتر وهو نفسه القيمة المعطاه لأن العملية الحسابية في محددنا الوظيفي الذي قمنا بإنشائه هي مجرد وضع المتغير المعطى **myNumber** ولنجرب تصميم محدد وظيفي أكثر فعالية بإضافة أكثر من متغير في سطر تعريف المحدد كما المثال التالي:

```
on myHandler myNumber, myOtherNumber
mySum = myNumber+myOtherNumber
put mySum
end
```

والأن من نافذة التراسل قم بإستدعاء وتشغيل المحدد الوظيفي بكتابة التالي:

```
myHandler(7,4)
--11
```

وهكذا فقد قمت بإستبدال قيم المتغيرات في محددنا الوظيفي بقيم حقيقية مما أدى الى تفعيله وإعطاء النتيجة.

والأن نحن على وشك تحويل محددنا الوظيفي السابق إلى وظيفة حقيقية تعطي نتائج عبر المتغيرات ولذلك قم بإستبدال السطر **put mySum** في المثال السابق بالسطر **Return mySum** ومن ثم قم بكتابة التالي في نافذة التراسل **Message Window**.

```
myNumber = myHandler(7,4)
put myNumber
--11
```

وهنا نلاحظ أننا قمنا بعمل متغير جديد وقمنا بمساوة قيمته بقيمة المحدد الوظيفي الذي قمنا بإنشائه لكي يضع البرنامج ذلك في الحسبان وبإستخدام الأمر **put** قام دايركتور بحساب النتيجة من خلال المحدد ومن الأمر **Return** قام بحل المسألة وقام بجمع الرقمين وأعطانا النتيجة(***سأتحدث تفصيلا عن فوائد الأمر Return لاحقا في نفس هذا الفصل***), وهكذا فقد إكتشفت بنفسك كيف وأن المحددات الخاصة يمكنها بسهولة التحول الى محددات وظيفية خاصة, والأن سنقوم بعمل تجربة أخرى مهمة وهي إنشاء محدد وظيفي جديد **Movie Script** ومن ثم كتابة التالي بداخله:

```
on addTwoNumbers num1,num2
sum = num1+num2
return sum
end
```

والأن قم بإختبار المحدد الوظيفي من خلال نافذة التراسل **Message Window**.

```
put addTwoNumbers(5,24)
--29
put addTwoNumbers(-7,2)
--5
put addTwoNumbers(4.5,2.1)
--6.6000
```

ويمكنك أيضا إستخدام المحددات الوظيفية لكي تقوم بإستخلاص مقارنة النتائج **TRUE** أو **FALSE** , أكتب المحدد الوظيفي التالي:

```
on isOneGreaterThen num1,num2
return(num1-1) = num2
end
```

والأن من نافذة التراسل أكتب

```
put isOneGreatorThan(5,4)
--1
put isOneGreatorThan(7,2)
--0
```

إستخدام قاعدة (إذا)..(إذن)..(وإلا)IF..THEN..ELSE:

الأن المفترض أنك فهمت من الدروس السابقة المتغيرات والمقارنات والمتغيرات الحلقية جيدا والمحددات الوظيفية والكثير من المهارات وإذا كنت ترغب في العودة الأن لقراءة أي جزء للتأكد من الفهم فلا تتردد لأني الأن على وشك الخوض في البرمجة الحقيقية التي تنتظرها وهي ستبدأ بشرح قاعدة إذا..إذن..وإلا فهذه القاعدة موضوعة في جميع لغات البرمجة وهي توجد في اللينجو بالطبع ووظيفتها الأساسية هي عقد المقارنات التي يترتب عليها نتائج ما, وإليك مثال

```
on testif num
 if num = 7 then
  put"you Entered the number 7"
 end if
end
```

والأن من نافذة التراسل قم بكتابة الأمر التالي:

```
testif(7)
--"You Entered the Number 7"
```

وشرح المثال السابق ببساطة هو إننا قمنا بعمل محدد وظيفي بسيط كما تعلمت سابقا ثم طلبنا من دايركتور أن يتوقع النتيجة بقاعدة **if** فإذا كان المتغير **num** يساوي **7** إذن وقد مثلت بالأمر **then** فدايركتور يجب أن يضع في نافذة التراسل الجملة **you Entered the number 7** ومن ثم يأتي السطر **end if** وهو سطر يلي دائما وجود **if** ويوضع بعدد **if** المستخدمة في المحدد الواحد, ومن ثم فقد إنتهى المحدد النهاية المعتادة بالأمر **end**, ويمكنك أيضا إعادة كتابة الكود السابق بالشكل التالي:

```
on testif num
 if num = 7 then  put"you Entered the number 7"
 end if
end
```

وهنا يتبين لك مرونة اللينجو في كيفية كتابة الأوامر حيث يمكنك وضع السطر **put** مباشرة بعد **then** .

وكما هو الحال فنحن نستخدم في حياتنا الواقعيه الأمر إذا ..إذن..عندما نطلب من شخص شيئا ما فأحيانا تطلب من أخيك أنه إذا وجد عند البقالة خبز إذن فعليه شراءه وإلا فعليه عمل شئ آخر وهذه الـ(إلا) تسمى في لغة اللينجو **ELSE**, ومن المثال قالقادم سنقوم بإستخدامها ورؤية كيفية عملها, قم بفتح نافذة **Movie Script** جديدة أو أضف المحدد التالي واكتبه أسفل السابق.

```
on testElse num
 if num = 7 then
  put"You Entered the number 7"
 else
  put"You Entered a number that is not 7"
 end if
end
```

والأن يمكنك إختبار الكود السابق من نافذة التراسل **Message Window**, من خلال كتابة التالي:

```
put testElse(7)
--"You Entered the number 7"
put testElse(9)
--"You Entered a number that is not 7"
```

وهكذا فقد رأيت بنفسك كيف إستجاب البرنامج مستخدما قاعدة **ELSE** لتحديد رد الفعل أمام المستخدم, ولكن ماذا لو أراد المبرمج وضع عدة إحتمالات للحدوث مقابل كل عدة قيم معينة؟, يمكننا الأن تجربة عمل تعشيش بسيط لقاعدة **IF** والتعشيش **Nesting** المقصود به وضع قاعدة **IF** داخل قاعدة **IF** أخرى وذلك يعدد الإحتمالات أمام المبرمج لتنفيذ ما يريده, ويمكنك تجربة الكود التالي لتجربة تعشيش مبسط لقاعدة **IF**.

```
on testElse2
 if num = 7
  put"You Entered the number 7"
 else if num = 9 then
  put"You Entered the number 9"
else
  put"You Entered number not 7 or 9"
end if
end
```

وفي المثال السابق فالبرنامج يحدد رد فعل معين أمام المستخدم في حالة إدخاله للرقم **7** والرقم **9** ويحدد رد فعل آخر في حالة إدخال أي رقم آخر دون **7** أو **9**.

في المثال السابق إستطعنا وضع وبرمجة ثلاثة إحتمالات ولكن ماذا سنفعل حيال برمجة أكثر من ثلاثة إحتمالات؟ هل سنظل نستخدم **IF**, يمكننا ذلك ولكن ذلك غير معقول فبرنامجنا سيكون كبيرا وسيكون من الصعب معالجة الأخطاء ولذلك فاللينجو توفر لنا أمرا جديدا يسمى الحالة **Case** وهي تختص بمعالجة الأعداد الكبيرة من الإحتمالات التي يضعها المبرمج.

إستخدام جملة CASE مع الأمر OTHERWISE:

كما ذكرت من قبل فالأمر **Case** يستخدم لعمل عدة إحتمالات وتوقع ردود أفعال معينة حيال كل إحتمال معين بذاته ولتجربة ذلك بنفسك فعليك تجربة الكود التالي:

```
on testCase num
case num of
7: put "You Entered the number 7"
9: put"You Entered the number 9"
otherwise: put"You Entered number not 7 or 9"
end case
end
```

وكما ترى فإستخدام أمر الحالة **Case** بسيط وسهل وسريع لوضع عدة إحتمالات, والأمر **otherwise** يستخدم لغلق قاعدة **Case** حيث يكتب بعده ألأسطر التي تقود البرنامج لأي رد فعل آخر مختلف عن الإحتمالات المفترضة.

تطوير البرامج الذكيه بإستخدام تعشيش قاعدة الشرط IF:

من المهم جدا فهم ديناميكية اللينجو عند برمجة أوامرها فقد سبق أن قمنا بعمل تعشيش مبسط لقاعدة **IF** والأن سنقوم بعمل تعشيش أكثر تعقيدا وذلك حتى تتعرف بنفسك على الإمكانيات والمرونة العالية التي توفرها اللينجو عند برمجة الأوامر, والأن ألق نظرة على المثال التالي:

```
on nestedif num
 if num<0 then
  if num = -1 then
   put"You Entered a -1"
else
   put"You Entered a naegative number other than -1"
 end if
else
 if num = 7 then
  put"You entered a 7"
 else
  put"You Entered a positive number other than 7"
  end if
 end if
end
```

يمكنك كتابة هذا الكود وتجربته أو فتح الملف **Nested_IF.dir** الموجود داخل القرص المصاحب للكتاب داخل الدليل الفرعي **Essential** داخل الدليل **Book Code Source** وبعد فتح الملف يمكنك من نافذة التراسل تجربة الكود بإختيار قيم عديدة لإختباره.

```
nestedif(7)
-- "You entered a 7"
nestedif(-2)
-- "You Entered a naegative number other than -1"
nestedif(5)
-- "You Entered a positive number other than 7"
```

في المثال السابق فالبرنامج يفترض أولا إذا كان الرقم المدخل هو رقم أقل من الصفر أو يساوي **1-** فالبرنامج يخبرنا بأن الرقم المدخل هو رقم واحد سالب وما عدا ذلك فإذا أدخلنا رقم سالب آخر فالبرنامج يخبرنا بأننا أدخلنا رقم سالب آخر غير رقم الواحد السالب, وفي نفس الوقت فالبرنامج يخبرنا أننا أدخلنا الرقم سبعة موجبة في حالة إدخالنا لهذا الرقم أما إذا قمنا بإدخال أي قيمة موجبة أخرى فإن البرنامج يدرك ذلك ويخبرنا بأننا قمنا بإدخال قيمة موجبة غير قيمة السبعة الموجبة, وهذه الطريقة المتبعة في البرمجة توضح لك كيفية تطوير برامج ذكيه لديها القدرة على التفكير والإستجابة للمستخدم.

المعاملات المنطقية (و) و (أو)و (ليس)AND-OR-NOT :

الأن سنتناول أمثلة أخرى مهمة لإستخدام تعشيش **IF** لنتعرف على إحتمالات أخرى يمكننا برمجتها والأن جرب المثال التالي:

```
on testTwoNumbers num1,num2
  if num1 = 7 then
   if num2 = 7 then
    put"Youe Entered two 7s"
   end if
 end if
end
```

والأن جرب من نافذة التراسل **Message Window** كتابة الأمر **testTwoNumbers(7,7)** وشاهد النتيجة بنفسك.

ملحوظة هامة جدا: تذكر دائما قبل تجربة الأوامر من خلال نافذة التراسل أن تضغط على الزر ⚡ في نافذة الكود لكي يقوم دايركتور بمعالجة الكود في الذاكرة وتنجح عملية التراسل بين الكود ونافذة التراسل.

هناك طريقة أخرى لبرمجة المثال السابق بإستخدام المعاملات المنطقية مثل المعامل (و) **and** فهي تمكن أيضا من تفحص حالة رد الفعل للبرنامج الأن غير الكود في المثال السابق الى الكود التالي:

```
on testTwoNumbers num1,num2
  if num1 = 7 and  num2 = 7 then
      put"Youe Entered two 7s"
 end if
end
```

يمكنك أيضا برمجة المثال السابق بواسطة المعامل المنطقي (أو) **or** داخل برنامجك, الأن غير الكود السابق ليضبح كالتالي:

```
on testTwoNumbers num1,num2
  if num1 = 7 or  num2 = 7 then
      put"Youe Entered two 7s"
 end if
end
```

وكما ترى فالمعاملات المنطقية قوية الأثر وتبسط وتقلل من عدد أسطر البرنامج, وهناك معامل منطقي هام آخر وهو المعامل (ليس) **not** وهو يختص بالعمليات العكسية فهو يراقب ما يحدث ويعطي النتيجة بناءا على ما إذا رأى ان المدخلات ليست هي الواقع المفترض أو عكس ما يجب عمله, كما يستخدم لعمل توقع ما في البرنامج, راقب المثال التالي:

```
on testNot num
 if not(num = 7) then
  put"You Did not Enter a 7"
 end if
end
```

في المثال السابق إذا قمت بتجربة أي رقم غير الرقم سبعة فسيخبرك البرنامج أنك تدخل رقما غير السبعة, وهذه الأمثلة السابقة توضح لك مدى أهمية إستخدام المنطق في برامجك المستقبلية, وفي المستقبل عندما نتطور في البرمجة سترى أمثلة قوية على تعشيش **IF**, معظم المبرمجين يستخدمون تقنيات التعشيش بشكل أو بآخر, فقاعدة **IF** وملحقاتها وأوامر المنطق وأوامر الحالة **Case** و **Otherwise** هي الأوامر التي تعطي الذكاء والقدرة على إتخاذ القرار لبرامجك التي تنتجها.

إستخدام الحلقات التكرارية Repeat loops:

من أهم المواضيع في البرمجة هي إستخدام الحلقات التكرارية التي تعني تكرار أمر ما في اللغة وتنفيذه عدة مرات, وكثيرا ستحتاج الى الحلقات التكرارية اثناء كتابة برامجك, ولكي تقوم بعملية التكرار في لغة اللينجو يجب أن تستخدم الأمر **Repeat** بمعنى (كرر), المعني في برمجة التكرار هو ***تكرار أمر معين لحين حدوث حدث معين***.

الحالة الأولى Repeat with:

تستخدم هذه الحالة في حالة رغبتنا في تصميم نظام للعد **Counting** وهذا مثال عملي على ذلك.

يمكنك فتح الملف **repeats** من الدليل **Essential** من داخل القرص المصاحب للكتاب بدلا من كتابة الأمثلة.

```
on countTo100
  repeat with i = 1 to 100
    put i
 end repeat
end
```

ملحوظة هامة جدا: تذكر دائما قبل تجربة الأوامر من خلال نافذة التراسل أن تضغط على الزر في نافذة الكود لكي يقوم دايركتور بمعالجة الكود في الذاكرة وتنجح عملية التراسل بين الكود ونافذة التراسل.

الأن قم بفتح نافذة التراسل **Message Window** واكتب **countTo100** ثم لاحظ النتيجة.

في المثال السابق قام البرنامج بإستخدام المتغير **i** للعد من واحد الى مائة وفي كل مرة كان البرنامج يعد رقما فقد كان يضعه في نافذة التراسل بواسطة الأمر **put i** وفي النهاية انتهت عملية التكرار بوضع **end repeat** , يمكنك أيضا عمل عد عكسي بإستخدام الوظيفة **Down** بتغيير السطر الى **repeat with i =100 down to 1**, جربها وإختبر بنفسك.

الحالة الثانية Repeat WHILE:

النوع الثاني من أوامر التكرار هو الأمر **Repeat while** ويستخدم لبرمجة تكرار في حين حدث ما أو متغير ما تتغير قيمته أو وضعه, لفهم ذلك راقب معي المثال التالي.

```
on repeatTo100
 i =1
 repeat while i<=100
  put i
  i=i+1
 end repeat
end
```

في المثال السابق المتغير **i** له القيمة واحد ويظل البرنامج يكرر ويكرر طالما أن المتغير **i** أقل من مائة أو يساويها, وفي كل مرة يحدث فيها تكرار يقوم البرنامج بفحص حالة المتغير i وفي كل مرة تكرار يقوم البرنامج بإضافة واحد الى المتغير, وبذلك فعندما تقوم بتجربة البرنامج من نافذة التراسل تكتشف أنك لديك برنامج قام بالعد من واحد الى مائة ولكن بشكل جديد.

بالطبع المثال السابق بسيط جدا بالنسبة لما هو ممكن فعله بالأمر **Repeat while** وإليك مثال أكثر قوة على إستخدام الأمر **Repeat while**.

```
on countWhileNoShift
   i=1
   repeat while not the shiftDown
    put i
    i=i+1
  end repeat
end
```

في المثال السابق نستخدم الوظيفة **ShiftDown** والتي تعني فحص الضغط على مفتاح **Shift** ***(في درس التحكم في لوحة المفاتيح ستتعلم كل شئ عن برمجة لوحة المفاتيح)***, وبنفس الطريقة فهناك متغير **i** يحمل القيمة واحد ويظل البرنامج يكرر ويكرر ويضع المتغير مضافا له واحد وفي كل مرة يقوم بفحص ضغط المفتاح **Shift** ويراقب إذا ما كان المستخدم قد ضغط عليه أم لا مستخدما أوامر المنطق **not** والتي قد سبق شرحها وكما تعلمت فالضغط على مفتاح **Shift** سيعطي **TRUE** وهذا ما يفعله أمر المنطق في سطر التكرار فطالما المستخدم لا يضغط المفتاح **shift** إذا فدايركتور هنا يرى الحالة على أنها غير حقيقية **FALSE** كما تعلمت من قبل أنظر درس ***(مقارنة المتغيرات والعاملات المنطقية في نفس هذا الفصل)***, والأن جرب هذا المثال عبر نافذة التراسل **Message Window**.

الحالة الثالثة: التكرار من مصفوفة

الحالة الثالثة هي حالة التكرار من مصفوفة, سنقوم في الدروس القادمة بدراسة وشرح المصفوفات بالتفصيل ومن خلال أمثلة قوية ولكن كتعريف سريع لمعنى المصفوفة يمكنك القول بأنها وعاء للبيانات يستطيع أن يحمل القيم الإسمية والرقمية أو الإثنان معا, وتستخدم المصفوفات رياضيا لتخزين المعلومات وكذلك هو الحال في لغة اللينجو, وقامت شركة ماكروميديا بإضافة هذه الخاصية التكرارية في دايركتور تسهيلا لبرمجة المصفوفات وتقليلا لحجم الخوارزميات المكتوبة لحل المسائل البرمجية, وفي المثال التالي نرى مصفوفة خطية تحمل قيم رقمية معينة.

```
on repeatbyNuminList
  repeat with i = [1,2,3,4]
  put i
 end repeat
end
```

والأن جرب المثال السابق من نافذة التراسل **Message Window** ولاحظ كيف وأن البرنامج يعد من **1** الى **4** ويتوقف وهي القيم المخزنة داخل المصفوفة الخطية المعطاة في المثال.

تطبيقات وإستخدامات أخرى للتكرار:

الأن سأقوم بعرض بعض التطبيقات الهامة للتكرار وأشرح كيفية الخروج من التكرار بطريقة جديدة, أكتب معي المثال التالي ومن ثم جربه في نافذة التراسل **Message Window**.

```
on countTo100orShift1
   i=1
   repeat while(i<=100) and (not the shiftDown)
     put i
     i = i+1
  end repeat
end
```

في المثال السابق تطبيق قوي لعملية المنطق المضافة لأمر التكرار فهنا البرنامج يشترط حالتين للإستمرار في التكرار وهي أن المتغير **i** يكون أقل من مائة أو يساويها والحالة الثانية هي أن المستخدم لا يضغط على المفتاح **Shift**, وقد إستخدمت المعامل المنطقي **and** للربط بين الشرطين, في المثال التالي سأقوم بإعادة برمجة التكرار ولكن بطريقة جديدة سأستخدم فيها أمر الخروج من التكرار.

```
on countTo100orShift2
  repeat with i =1 to 100
  put i
  if the shiftDown then exit repeat
 end repeat
end
```

في المثال السابق قمنا بنفس العملية لكن بشكل مختلف, سيظل البرنامج يعد من واحد الى مائة وإذا ضغط المستخدم على مفتاح **shift** فالبرنامج سيخرج من حلقة التكرار وقد إستخدمت **exit repeat** لعمل ذلك.

الأمر **exit repeat** يعمل مع كلا من **repeat with** و **repeat while** فهو في الواقع طريقة أخرى لإنهاء عمل التكرار, وفي الأمثلة السابقة قمنا بإستخدامات بسيطة له بينما يمكنك في المستقبل كتابة تكرارات معقدة في برامجك المستقبلية الطويلة.

هناك أيضا الأمر **next repeat** الذي يمنع تنفيذ التكرار في وقت معين وفق حدث معين, لفهم ذلك أكتب معي التطبيق التالي:

```
on countTo100WithShift
   repeat with i=1 to 100
   put"Counting..."
   if the shiftDown then next repeat
   put i
 end repeat
end
```

في المثال السابق يقوم البرنامج بالعد من **1** الى **100** وفي كل مرة يقوم التكرار بوضع العبارة **Counting** في نافذة التراسل ومن ثم يقوم أيضا بوضع قيمة المتغير في نافذة التراسل أيضا, ومع ذلك فإذا قمت بالضغط على مفتاح **shift** بينما يعمل البرنامج سيقوم الأمر **next repeat** بمنع دايركتور من تتابع وضع قيمة المتغير **i**, وهنا فقط سيظهر لك الكلمة **Counting** في نافذة التراسل**Message Window** وستظل تظهر وتتكرر حيث هنا وقف التكرار عند هذه العبارة, فعمل **next repeat** هنا توقف على مكانه بين أسطر البرنامج, ويمكن الإستفادة من هذه التقنية في برامجك لتصميم توقف للتكرار في أي وقت تشاء أو وفق حدوث حدث معين كالضغط على مفتاح كما في مثالنا السابق.

التكرار للأبد!:

في بعض الحالات قد يكون من الضروري برمجة حالة تكرار تظل تعمل حتى يتدخل الأمر **exit repeat** ويوقفها, وفي هذه الحالة فأنت غير مضطر لكتابة الأمر **repeat with** لأن ذلك الأمر يحدد للحلقة التكرارية عدد معين من مرات التكرار, حتى لو إستخدمت الأمر **Repeat while** أيضا ستضطر لإفتراض حدث آخر, ومع ذلك هناك خدعة برمجية لإستخدام الأمر **repeat while** بدون تحديد حالة معينة.أي انني سأبرمج تكرارا أبديا!, إليك المثال.

```
on repeatForEver
   repeat while TRUE
   put"repeating..."
   if the shiftDown then exit repeat
 end repeat
end
```

هذا المثال في الواقع لا يقوم ببرمجة الحلقة التكرارية للأبد, لكنه فقط يظل يكرر حتى يضغط المستخدم مفتاح الـ**SHIFT** ومع ذلك فالأمر **exit repeat** يستطيع إيقاف العملية كلها وذلك سببه وضع الحالة **TRUE** كحدث معين أمام **While** ولأن **TRUE** هي قيمة منطقية تدل على حقيقة حدوث الشئ فسيظل التكرار يكرر للأبد, أو على الأقل حتى يضغط المستخدم مفتاح الـ**Shift** حيث يقوم الأمر **exit repeat** بالخروج من التكرار.

تذكر إنك في بعض الحالات قد تقوم بكتابة بعض أنواع الكود التي قد تعمل بدون توقف أو قد تؤدي الى عدم إستقرار دايركتور, سيكون سبب ذلك غالبا من طريقة برمجتك أو إعتمادك لطرق غير صحيحة في معالجة المنطق في برنامجك أو ربما تقوم بتصميم تكرار أبدي كما في المثال السابق وتنسى وضع طريقة لإيقاف برنامجك مما قد يؤدي في النهاية الى توقف دايركتور عن العمل ولتجنب حدوث ذلك يمكنك دائما الضغط على مفتاح **Ctrl+.** ضغطا متكررا لإنقاذ برنامج دايركتور من الإنهيار.

تطبيق مفيد لإستخدام التكرار:

سنقوم الأن بكتابة تطبيق مفيد حول التكرار لتأكيد الفهم وتخيل المواقف التي يمكنك بها إستخدام التكرار, الأن أنظر المثال التالي:

```
on mouseDown
 repeat while the mouseDown
   beep
 end repeat
end
```

في المثال السابق سيظل التكرار يعمل على تشغيل الصوت **Beep** طالما أن المستخدم يضغط على مفتاح الفأرة الأيسر, وسيظل الصوت يعمل طالما المستخدم يضغط على الزر الأيسر للفأرة من كما ترى الأمر مباشر **repeat while the mouseDown** وسيظل الأمر يعمل حتى لو قام المستخدم بتحريك المؤشر خارج حدود العنصر **Sprite** طالما الزر الأيسر للفأرة مضغوط ولم يرفع المستخدم إصبعه عنه.

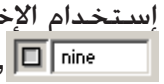
لحل المشكلة السابقة كان علينا إعادة كتابة المثال بالشكل التالي:

```
on mouseDown
  repeat while the stillDown
    beep
    if rollover (1) then exit repeat
  end repeat
end
```

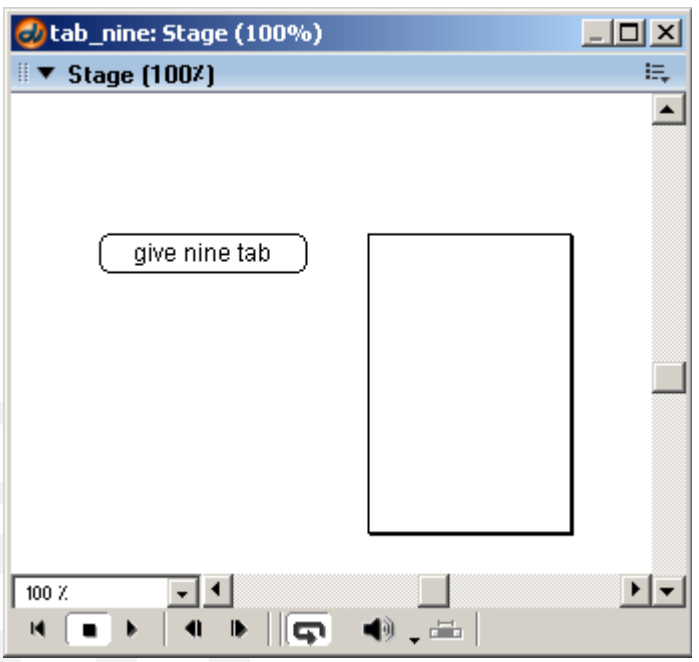
في المثال السابق قمت بإستخدام الأمر الجديد **stillDown** و **rollover** والأمر **stillDown** في جملة التكرار يعني كرر طالما زر الفأرة الأيسر مضغوطا, بينما يعني الأمر **rollover(1)** في قاعدة الشرط بأن إذا كان مؤشر الفأرة فوق العنصر **sprite** الأول فقم بالخروج من التكرار **exit repeat** وفي النهاية ينتهي التكرار **end repeat** وينتهي البرنامج.

لرؤية هذا المثال عمليا قم بفتح الملف **repeats clicks.dir** بدلا من كتابة الأكواد بنفسك, الملف في الدليل **Essential** من الدليل المصدري للكود **Book Code Source.**

تطبيق جدول الضرب بإستخدام التكرار:

قم بتصميم نافذة مشروع بحجم **320X240** ثم ضع على المسرح **Stage** زر **Push Button** و حقل نصي **Field** يمكنك إستخدام الإختصار **Ctrl+8** من داخل الكاست لإنشاء الحقل مباشرة ثم أضف له عنوانا واسحبه الى المسرح من خلال الأيقونة 

, يجب أن تبدو شاشتك كما الشكل التالي:



والأن قم بإنشاء كود عنصر **Sprite Script** أو **Cast member Script** واكتب داخله الأسطر التالية:

```
on mouseUp me
  Repeat with i = 1 to 9
put i&"X9="& i&9 into line i of field"nine"
end repeat
end
```

والأن قم بتشغيل المشروع وإضغط على الزر **Push Button** فستلاحظ أن جدول الضرب الخاص بالرقم **9** قد ظهر في الحقل النصي الذي يوجد أمامك على المسرح, في الكود السابق يقوم الأمر **Repeat** بعمل تكرار لقيمة المتغير **i** من القيمة **1** الى القيمة **9** والأمر **put** يقوم بتنفيذ عمليه الضرب الحسابية للمتغير ويضعها في السطر الذي يحمل نفس قيمة المتغير بحيث يظهر الجدول متتابعا داخل الحقل النصي.

والأمر **end repeat** كما نعلم يقوم بإنهاء التكرار تماما, وهذه الخدعة البرمجية **Technique** تستخدم كما رأينا توظيفا جيدا لإمكانيات الأمر **repeat** , ويمكنك الإستفادة من هذا الكود في تصميم برامجك التعليمية الخاصة أو فكر في كيفية تطوير الكود بحيث يقوم بعمليات حسابية أخرى كالجذر التربيعي, أو إعطاء المستخدم جدول ضرب للأعداد الأولية مثلا.

يمكنك تشغيل الكود ورؤيته عمليا عن طريق الملف **tab_nine.dir** الموجود داخل الدليل **Essential** داخل دليل **Book Code Source** من القرص المرفق مع الكتاب.

الأرقام العشوائية Random Numbers:

تستخدم تقنيات الأرقام العشوائية في برمجة الألعاب أو عندما تريد تصميم برامج بلغة اللينجو بحيث تفكر بنفسها أي كتصميم برامج الذكاء الصناعي التي لديها القدرة على إتخاذ القرارات وأحيانا تستخدم الأرقام العشوائية لكسر الملل في تصميم وبرمجة الوسائط المتعددة الخاصة بالعروض وسنرى أمثلة ممتازة لاحقا عن كيفية إستخدامها أما الأن فسأقوم بعرض كيفية توليد الأرقام العشوائية بواسطة أوامر اللينجو.

الأمر Random:

الأن جرب أن تفتح نافذة التراسل **Message Window** وكتابة الأمر التالي داخلها:

```
put random(6)
--3
```

بالنسبة لي فقد أعطاني دايركتور الرقم **3** أما انت فمن المحتمل أن يعطيك دايركتور قيمة تتراوح ما بين الواحد والستة فالأمر **put random(6)** يعني تحديدا أمرا مباشرا للدايركتور لكي يعطي لنا قيمة عشوائية رقمية مختارة من قبله ما بين رقم الواحد ورقم الستة.

الوظيفة RandomSeed:

تستخدم الوظيفة **RandomSeed** من أجل تغيير خوارزمية لغة اللينجو المعتمدة على رقم معين, فالدايركتور يعتمد على شفرة معينة لتوليد الأرقام العشوائية, وفي بعض الحالات التي يشعر فيها المبرمج بأن الأرقام العشوائية التي يختارها الكمبيوتر تتكرر كثيرا فذلك سببه يعود الى تكرار إستخدام نفس رقم المعادلة الخاصة بتوليد الأرقام العشوائية في لغة اللينجو, ولذلك فقد وضعت لغة اللينجو مزودة بأمر يمكن المبرمج من تعديل نتائج التحليلات العشوائية التي تقوم بها لغة اللينجو وذلك بتمكين المستخدم من إدخال رقم ما لكي تقوم معادلة توليد الأرقام العشوائية بتغيير نتائجها دائما, والأن جرب من نافذة التراسل **Message Window** الأمر **put the RandomSeed** وقم بمعاينة النتيجة بنفسك ويمكنك إستخدام الأمر **Set** لكي تقوم بتحديد قيمة جديدة لمتغير المعادلة العشوائية لكي تحصل على أقل حالات تكرار لقيمة ما تريدها, الأن اكتب:

```
set the RandomSeed = 600
put the RandomSeed
```

الأن يمكنك إلقاء نظرة على الملف **randomseed.dir** الموجود داخل دليل **Essential** داخل الدليل المصدري للكود في القرص المرفق مع الكتاب.

حاول إظهار نافذة التراسل جنبا الى جنب مع نافذة المسرح **Stage** ثم قم بتشغيل البرنامج وإنقر بالزر الأيسر للفأرة على نافذة المسرح ولاحظ النتيجة التي تظهر أمامك في نافذة التراسل **Message Window**, وكما ترى فالبرنامج البسيط يحاول أن يقلل من عدد التكرار الممكن حدوثه أثناء إختيار قيمة عشوائية عبر الكمبيوتر وذلك عن طريق إعادة تعيين متغير المعادلة العشوائية في لغة اللينجو **RandomSeed**, بالطبع هناك طرق أخرى بإستخدام لغة اللينجو نفسها لكي تقوم بصنع رقم عشوائي وتتجنب تكراره على الأقل في التخمين التالى للرقم, أدرس التطبيق التالي جيدا لفهم ذلك.

تطبيق على توليد رقم عشوائي لا يتكرر مرتين بصورة متتالية:

المشكلة التي نواجهها هنا هي رغبتي بتصميم برنامج يقوم بلعب ملف صوتي مختلف عن المرة التي سبقتها فلدي هنا ثمانية أصوات مختلفة في نافذة الأعضاء **Cast** بالأسماء **Sound1** حتى **Sound8** ولذلك قمت في السطر الأول بإختيار المحدد **on mouseUp** لتعيين وقت تنفيذ الحدث عند الضغط على الزر الأيسر للماوس ثم في السطر الثاني قمت بعمل متغير عام بالإسم **lastSound** ثم وضعنا القيمة العشوائية للرقم **8** في المتغير **r** ومن ثم فقد قمت بإستخدام منطق التكرار **repeat while True** حيث سيظل البرنامج يكرر الحدث طالما أن المنطق حقيقي والمنطق المقصود به هو مجموعة الأسطر التالية وهي أن البرنامج يضع المتغير **sound** مضافا إليه القيمة العشوائية **r** في متغير جديد **soundName** ومن ثم بإستخدام قاعدة **IF** يقوم البرنامج بمقارنة آخر صوت مسموع بآخر قيمة مسجلة في الذاكرة ولذلك فالبرنامج ينهي التكرار في حالة العثور على قيمة مختلفة ويقوم بلعب الصوت بإستخدام الأمر **puppetSound** وإلا فإنه يقوم بعملية التكرار مرة أخرى وفي السطر الأخير يقوم السطر **updateStage** بعمل تحديث للمعلومات الواردة إلى المسرح عند النقر بالماوس, والأن إليك التطبيق كاملا لكي تدرسه بعناية.

```
on mouseUp
  global lastSound
  put random(8) into r
  repeat while TRUE
    put "sound"&r into soundName
    if soundName <> lastSound then exit repeat
  end repeat
  put soundName into lastSound
  puppetSound soundName
  updateStage
end
```

يمكنك العثور على التطبيق كاملا بالإسم **randomsound.dir** داخل المجلد **Essential** داخل الدليل المصدري للكود في القرص المرفق مع الكتاب.

يمكنك تصميم العديد من الأفكار المتعلقة بتوليد الأرقام العشوائية, ستجد تطبيقا قويا لتوليد الأرقام العشوائية في تطبيقات المصفوفات التي سيلي شرحها لاحقا في نفس هذا الفصل وستجد معلومات تفصيلية عن إستخدام الأمر **puppetSound** عند التحدث عن برمجة الصوت لاحقا تذكر إنه من المهم جدا توظيف أوامر لغة اللينجو معا لكي تستطيع تصميم برامجك وجعلها تتصرف كما هو متوقع منها ولذلك يجب أن تقوم دائما بتجربة برامجك الخاصة دوريا وحاول أن تبدأ بكتابة خوارزمياتك على الورق كما شرحت سابقا ثم ترجم أفكارك الى اللينجو .

إستخدام الوظيفة Value لتحويل وتحديد القيم رياضيا:

الوظيفة **()Value** تقوم أساسا بإعطائنا قيمة رياضية يمكن إستخدامها أو يمكن إعادة توظيفها مرة أخرى, فبرامجنا التي نصممها ليست دائما تعطينا قيم يمكن إستخدامها رياضيا ولذلك نحتاج الى تحويلها بإستخدام هذه الوظيفة, وبإذن الله ستتمكن من الفهم العميق لهذه الوظيفة الهامة جدا عند المرور على تطبيقات عملية مباشرة لكنها مفيدة جدا.

إستخدام ()Value مع الأرقام:

إفتح نافذة التراسل **Message Window** ثم أكتب التعبيرات التالي, سأقوم بعمل متغير حلقي **String** يمثل عمري.

```
MyAgeis="28"
put MyAgeis
--"28"
```

الأن سأقوم بتحويل المتغير الخاص بعمري **MyAgeis** الى متغير ذو قيمة رياضية مفهومة يمكن التعامل معها رياضيا

```
put value(MyAgeis)
--28
```

الأن جرب إستخدام نفس التعبير مقسوما على **2**

```
put value(MyAgeis)/2
--14
```

كما تلاحظ في المثال فقد قام دايركتور بإحتساب القيمة الرياضية حيث أن **28/2 = 14**
الأن جرب كتابة **put value("5+1")** ماذا تلاحظ؟ لقد أعطانا الدايركتور الرقم **6**, إذن تستنتج من التجارب السابقة بأن الدايركتور يقوم بتحويل القيم ومعاملتها رياضيا بإستخدام الأمر **Value**.

إستخدام ()Value مع المصفوفات:

يمكن تحويل المتغير الحلقي الى مصفوفة (خطية مثلا) , قم بكتابة المثال التالي في نافذة التراسل **Message Window**.

```
MyString = "[1,2,3]"
put value(MyString)
--[1,2,3]
```

سأقوم لاحقا بعون الله بتفصيل الحديث عن المصفوفات الرياضية في لغة اللينجو بكلا نوعيها وستجد تطبيقات أخرى متقدمة حول إستخدام **Value** مع المصفوفات, سأتحدث أيضا عن التحويل وطرقه المتعددة فيما بعد.

إستخدام أوامر التجول:

سبق أن شرحنا في مقدمة البرمجة أوامر التجول **Navigation** المتمثلة في الأمر الرئيسي **Go** والوظائف التي يتبعها والجدول التالي يلخص أوامر التجول كتذكرة سريعة وهامة لما سبق.

جدول أنواع التجول داخل دايركتور:

مثال نوعي	وصف المثال النوعي	مثال تطبيقي
go to frame X	إذهب الى الكادر رقم س على خط الزمن	go to frame 8
go to frame "X"	إذهب الى الكادر ذو الأسم س على خط الزمن	go to frame "maged"
go to the frame	لا تذهب الى أي مكان وظل مكانك	go to the frame
go to frame +X	أضف رقم واحد للكادر الحالي وإذهب اليه	go to frame +5
go to frame –X	أنقص رقم واحد من الكادر الحالي وإذهب اليه	go to frame –5
go next	إذهب لإسم الكادر التالي	go next
go previous	إذهب لإسم الكادر السابق	go previous
go loop	إذهب لأقرب إسم كادر يسبق الكادر الحالي	go loop
go marker(x)	إذهب لإسم الكادر الذي يكون ذو رقم محدد	go marker(3)
go marker(–x)	إذهب لإسم الكادر الذي يكون منقوصا برقم محدد	go marker(–3)

التحكم في خصائص العناصر Sprites:

سبق أن أتفقنا بأن أي عنصر على خط الزمن **Score** يسمى **Sprite** وسواء كان هذا العنصر صورة نقطية أو رسوم متجهة أو فيديو رقمي أو صوت رقمي فإنه له خصائص معينة يمكن التحكم بها والتأثير عليها بواسطة لغة اللينجو, وفي هذا الدرس سندرس خصائص العناصر دراسة متأنية, لكي ترى بنفسك كيف التحكم بمرونة في العناصر بواسطة لغة اللينجو.

التحكم في موقع عنصر Sprite Location:

قم بعمل عنصر **Sprite** صورة متجهة أو نقطية كما تعلمت في الفصل الأول ثم ضعه على المسرح **Stage** في القناة واحد على خط الزمن ثم قم بفتح نافذة التراسل **Message Window** وجرب التالي:

```
Sprite(1).locH = 50
updateStage
```

الأن يجب أن تلاحظ بأن العنصر الذي أنشأته قد قام بالانتقال الى الإحداثي الأفقي **50** فالأمر الأول الذي قمت بكتابته في نافذة التراسل يعني تحريك العنصر في القناة واحد الى الموقع **50** في الإحداثي الأفقي **H** حيث **H** تشير الى كلمة **Horizental** أي (أفقي) باللغة العربية.

ولكنك مع كتابتك لهذا الأمر تلاحظ أن العنصر لم يتحرك الإ مع كتابتك للأمر التالي **updatestage** ثم ضغطك لمفتاح الإدخال **Enter** وذلك سببه أن هذا الأمر مهمته هي تنفيذ العمليات الخاصة بالتحكم في خصائص العناصر في ذاكرة الكمبيوتر ومن ثم عرضها على الشاشة من خلال المسرح **Stage** وذلك لأن البرنامج لا يعمل ولكن إذا قمت بتشغيل البرنامج وكان هذا الكود مكتوبا مثلا في كود المشروع **movieScript** فأن المسرح سيقوم تلقائيا بتحديث البيانات الموجودة في الذاكرة دون الحاجة إلى الأمر **updateStage** وعموما في بعض الحالات الخاصة نحتاج إلى كتابة هذا الأمر داخل أكواد برامجنا, وفي الأمثلة القادمة سنستخدم الأمر **updateStage** لأننا سننفذ الأمثلة عبر نافذة التراسل **Message window**.

تذكر أن الأمر **Sprite(1).LocH = 50** مكتوب بطريقة الـ**Dot Syntax** ويمكن كتابته بطريقة الـ **verbose Syntax** بكتابة **set the locH of Sprite 1 = 50** ولكن كما قلت من قبل ليس عمليا أن تكتب كودا بهذا الطول رغم انه قريب جدا من اللغة الإنجليزية ولكن هذه الملحوظة للتذكرة فقط بهذه الإمكانية وتذكر أنه ليس من الممكن كتابة كل أوامر لغة اللينجو بهذه الطريقة, إن إستيعاب ذلك على المبرمج سيكون حدسيا فيمكنك أحيانا توقع وتجربة الأوامر إذا أردت, ومن خلال الشرح ستدرك بنفسك أنه من المستحيل في بعض الأوامر كتابتها بطريقتين, أنا أفضل طريقة **Dot Syntax** كما ذكرت من قبل وإذا تعودت أنت أيضا عليها سيكون أفضل.

وكما قمت من قبل ببرمجة موقع العنصر مستخدما الإحداثي الأفقي **H** يمكنك أيضا برمجة الإحداثي الرأسي **Vertical** وذلك بواسطة **LocV** ويمكنك تجربة ذلك في نافذة التراسل بكتابة التالي:

```
Sprite(1).locV = 100
updateStage
```

هناك أيضا طريقة لوضع العنصر في موقع رأسي وأفقي في وقت واحد, وهذه الخاصية تسمى **Loc** وفي هذه المرحلة تأخذ هذه القيمة قيمة المحور السيني والصادي كما تعلمت في منهج الرياضيات من قبل, وهذه القيمة تسمى في لغة اللينجو نقطة **Point** أي مكان العنصر على مستوى المحورين الأفقي والرأسي والتي يستعاض بهما رياضيا بالموقع السيني **X** والموقع الصادي **Y** ولذلك تحدد الموقع بـ **Point(x,y)** ولتجربة ذلك عمليا أكتب في نافذة التراسل **Message Window**.

```
sprite(1).loc = point(200,225)
updatestage
```

يمكنك أيضا تغيير الكثير من خصائص العنصر غير خاصية الموقع التي شرحتها في الأمثلة السابقة, والجدول التالي يبين بالأمثلة خصائص العنصر التي يمكن برمجتها عبر اللينجو.

جدول خصائص العنصر:

الخاصية	وصف الخاصية	مثال تطبيقي
member	العضو المستخدم عبر العنصر على خط الزمن	member"Circule"
LocH	الموقع الأفقي للعنصر	Sprite(1).locH = 50
LocV	الموقع الرأسي للعنصر	Sprite(1).LocV = 100
Loc	موقع العنصر السيني والصادي معا	Sprite(1).loc= point(50,100)
rect	مقاس مستطيل العنصر	Sprite(1).rect=rect(25,25,75,75)
ink	لون العنصر	Sprite(1).ink = 8
blend	درجة شفافية العنصر مع الخلفية	Sprite(1).ink = 8
trails	تكرار رسم العنصر أثناء الحركة	Sprite(1).trails = TRUE
color	اللون الأمامي للعنصر	Sprite(1).color = rgb("#000000")
bgcolor	لون خلفية العنصر	Sprite(1).bgcolor= rgb(#FFFFFF")

عضو العنصر Sprite Member:

من الشائع إعادة برمجة إتصال العنصر على خط الزمن **Score** بعضو من الكاست فنحن نعلم بأن العناصر **Sprites** على خط الزمن هي أساسا أعضاءا في نافذة الأعضاء **Cast** أي لهم تمثيل حقيقي ولإعادة برمجة إتصال العنصر **sprite** بعضو **member** في نافذة الأعضاء **Cast** نستخدم الخاصية **member**, أدرس المثال التالي:

```
Sprite(1).member = member("bitmap2")
```

في السطر السابق قمت بإعادة إتصال العنصر **Sprite** في القناة واحد الى العضو **bitmap2** فإذا كان لديك عنصر **Sprite** هو في الأساس يمثل العضو **bitmap1** فيمكنك إعادة تمثيله مرة أخرى بواسطة الخاصية **member**, تستخدم هذه الطريقة في الغالب أثناء برمجة الأزرار وفي برمجة الزر أثناء وجود المؤشر فوقه أو الضغط عليه حيث تستخدم لتبديل العناصر الممثلة للأعضاء لإحداث التأثير المطلوب كما رأيت سابقا أثناء الحديث عن المتغير الخاص **Property** وكما سترى لاحقا أثناء التحدث عن تصميم السلوك **Behaviour**.

مستطيل العنصر Sprite Rectangle:

تلاحظ دائما على شاشة المسرح **Stage** بأن كل العناصر مهما كان شكلها يحيط بحدودها الخارجي مستطيل يحددها وهذا المستطيل من الممكن من خلاله مط العنصر وعمل تأثير الإستطالة والشد من خلاله كما تعلم ويمكن أيضا من خلال لغة اللينجو برمجة هذه الإستطالة والمط بإستخدام الخاصية **Rect** كما تلاحظ المثال في الجدول السابق.

حالة شكل العنصر Sprite Ink:

من المهم جدا فهم حالة شكل العنصر **Ink** وهي التي تحدد شكل العنصر وتفاعله مع الأشكال التي تقع خلفه وأمامه أيضا, فهل العنصر نسخة أصلية أم شفاف او هو عبارة عن قناع أو شكل شبحي أو ربما يكون ذو درجات فاتحة أو غامقة, ويمكنك القول أن حالة شكل العنصر هامة من أجل طريقة عرضه على الشاشة أمام مستخدم البرنامج, وفي الجدول في الصفحة التالية عرض لجميع الحالات التي يمكن أن يكون العنصر عليها.

جدول الحالة الحبرية للعنصر Sprite Ink Table:

الحالة الحبرية للعنصر Sprite	رقم كود الحالة (يستخدم مع INK)
Copy	0
Transparent	1
Reverse	2
Ghost	3
Not Copy	4
Not Transparent	5
Not Reverse	6
Not Ghost	7
Matte	8
Mask	9
Blend	32
Add Pin	33
Add	34
Subtract Pin	35
Background Transparent	36
Lightest	37
Subtract	38
Darkset	39
Lighten	40
Darken	41

لتجربة جميع حالات الحبر Ink للعنصر Sprite إفتح وشغل الملف ink_effects.dir من الدليل Essential من الدليل المصدري للكود من القرص المرفق مع الكتاب.

لون العنصر Sprite Color:

إن تغيير اللون **Color** ولون الخلفية **bgColor** يعتمد أساسا على نوع العنصر الذي نتعامل معه, فمثلا تستخدم صورة نقطية ذات لون بدقة واحد بيت لونين فقط وهما الأبيض والأسود لكي تحدد لونها, ولذلك فتستخدم خواص الحبر **Darken** و **Lighten** معها, فهناك العديد من الطرق لتحديد الألوان بلغة اللينجو والطريقة الأولي هي طريقة تحديد ألوان **RGB** وهي اللون الأحمر **Red** والأخضر **Green** والأزرق **Blue** ولعمل ذلك مع العناصر **Sprites** نستخدم الخاصية **rgb** لبرمجة لون العنصر, الأن جرب بنفسك وقم بفتح مشروع جديد وارسم مربع بواسطة أداة الصورة النقطية وليكن أزرق اللون ثم ضع العنصر على خط الزمن واجعله يحتل مسافة كادر واحد فقط على القناة الأولى ومن نافذة التراسل قم بتجربة الأوامر التالية بالتتابع:

```
Sprite(1).color = rgb(255,0,0)
updateStage
```

الأن تلاحظ تغير لون العنصر النقطي **Bitmap** على نافذة المسرح **Stage** اوتوماتيكيا للون قريب من اللون الأحمر, لرؤية المثال في صورته النهائية قم بفتح الملف **rgb color_1.dir** من الدليل **Essential** من دليل الكود **Book Code Source** على القرص المرفق مع الكتاب.

هناك أيضا طريقة تحديد الألوان بالطريقة السدس عشرية **hexadecimal** بوضع قيمة اللون في متغير حلقي, جرب الأمر

```
Sprite(1).color = rgb("#FF0000")
updateStage
```

هذه الطريقة تستخدم مع مصممي الويب والانترنت فهي نفسها تستخدم مع ملفات الـ **HTML** وكما تلاحظ فالقيمة اللونية السابقة هي نفسها القيمة اللونية الرقمية التي قمنا بعملها منذ قليل فالـ **FF** تعادل القيمة **255** وكلا من الـصفرين يعادلان صفرا واحدا في الطريقة الرقمية المباشرة, يمكنك الإختيار من كلا النظامين فالنتيجة واحدة, ألق نظرة على المثـال **rgb color_2.dir** من الدليل **Essential** مند دليل الكود **Book Code Source** على القرص المرفق مع الكتاب.

في بعض الحالات الخاصة قد نستخدم رسمة نقطية ولكننا نريدها أن تظهر بلون معين من ببالتة لون معينة, في هذه الحالة يمكنك إستخدام بالتة اللون التي قمت بإنشائها للمشروع خاصتك بإستخدام الأمر **Sprite(1).color = paletteIndex(35)** فهذا الأمر يقوم بإستخدام اللون ذو القيمة **35** من بالتة المشروع من أجل لون العنصر على القناة **1**, لاحظ أن التعامل مع أوامر اللينجو فيما يخص معالجة الصورة النقطية من النادر جدا, لذلك معظم المطورون يعمدون إلى تحضير ملفات الصور بشكل منفصل.

السيطرة على مسارات التنفيذ في لغة اللينجو Script Flow Control:

هذا الموضوع هام جدا, وهو أن تقوم أنت كمبرمج للغة اللينجو بالسيطرة على طريقة تنفيذ الأسطر التي تكتبها من كود لغة اللينجو, فمثلا, هل تريد من لغة اللينجو مثلا أن تقوم بالتوقف عن تنفيذ محدد ما **Event**, أو هل تريد من اللغة أن تقوم بمعالجة مسألة ما بطريقة منطقية؟, هل تريد أن تقوم بإستدعاء محدد ما آخر في مكان ما في وقت معين؟, كل ذلك يستلزم منك كمبرمج عدم الإلتزام بترتيب الأوامر التي تقوم عادة بكتابتها, بل يجب أن ينفذ البرنامج بالمنطق الصحيح بحيث يحقق الغرض المطلوب منه, البرامج الجيدة هي تلك البرامج التي تنفذ المطلوب منها في أقل وقت ممكن.

الأمر Return:

الأمر **return** في لغة اللينجو متشعب وله عدة طرق للإستخدام,وقد سبق أن تعرضنا له أثناء الحديث عن تصميم المحدد الوظيفي **Function** *(راجع الوظائف في نفس هذا الفصل)*, فيمكن إستخدام هذا الأمر لكي يعطينا نتيجة قيمة ما أو يمكن إستخدامه بحيث يتحكم في المنطق ونصمم به المنطق الخاص ببرامجنا وله إستخدام آخر حيث يمكنك الخروج من المحدد الذي قمت بتصميمه توا, وكذلك تعتبر اللينجو **return** كزر في لوحة المفاتيح وتنتظر منه رد فعل من الكود, ولذلك لفهم الطرق الخمسة لإستخدام هذا الأمر ووضعه في مكانه المناسب حسب الحاجة فذلك يستلزم منك إعداد فنجان من القهوة أو الكابوتشينو ثم التركيز جيدا في الأمثلة التالية.

الطريقة الأولى:إستخدام Return في برمجة المنطق:

أكتب معي المثال التالي في نافذة كود مشروع جديد, إضغط **Ctrl+0** ثم أكتب الأسطر التالية, وسندرسها معا فيما بعد.

```
on isDigit SomeChar
  if "0123456789" contains string(someChar) then
    Return TRUE
  else
    Return FALSE
  end if
end isDigit
```

هنا قمت بتصميم محدد وظيفي **Function** كما تعلمت سابقا, الأن قبل شرح هذا الكود أكتب التالي في نافذة التراسل **Message Window** ثم لاحظ النتائج.

```
isDigit(5)
--1
isDigit(-5)
--0
```

الأن ماذا حدث؟, يمكننا معرفة بسهولة من الكود السابق بأن هناك حالة شرط **IF** فهذه القاعدة تقوم بالتأكد من إذا كان المتغير الحلقي المكتوب هو أحد الأرقام **0123456789** أو جميعها, وبالتالي فدايركتور ملزم بتطبيق قاعدة المنطق المفروضة عليه وهو أن يعطينا **TRUE** أي أن القاعدة المنطقية حقيقية وهي تساوي الرقم **1** بينما لو كان المنطق المصمم لا يتوافق مع المعطيات ففي هذه الحالة يعطينا **FALSE** وهي تعادل الصفر, ولذلك عندما قمنا بكتابة **isDigit(5)** فقام دايركتور بفحص القيمة وقام بعقد مقارنة لها مع ما هو مكتوب داخل الكود ولذلك كانت النتيجة إيجابية والمنطق صحيح, بينما عندما قمت فقط بوضع علامة سالبة كان المنطق أيضا غير حقيقي لما هو مطلوب أن يكون والنتيجة هي **FALSE** وهي الصفر.

لا تنخدع الأن بالمنطق, فالمنطق المطلوب من البرنامج السابق ليس هو تجنب القيم السالبة, فإختبار قيمة سالبة هو أحد القيم التي من المتوقع أن يرفضها منطق البرنامج والأن جرب كتابة **put isDigit(500)** فستجد البرنامج أيضا يعطيك النتيجة صفر ولكن إذا كتبت **put isDigit(123)** فسيقوم البرنامج بوضع **1**, الأن جرب وضع وإختبار كل المتغير الحلقي بكتابة **put isDigit(0123456789)** وشاهد بنفسك, إذن فالعبرة من تصميم المنطق السابق هو فقط إختبار وجود كل أو بعض من المتغير الحلقي الذي يشمل الترقيم من الصفر الى الرقم تسعة.

حسنا في التجربة السابقة رأينا كيف يمكن توظيف الأمر **Return** في تصميم المنطق الخاص بنا, وأطلق لعنانك الخيال في تصميم المنطق الخاص بك, معظم ما يمكن أن تتخيله رياضيا يمكن أن يحدث وبشكل مذهل, عندما تقوم مستقبلا ببرمجة البرامج الكبيرة ستكتشف بنفسك أهمية تصميم المنطق الخاص بك.

الطريقة الثانية:إستخدام Return لعرض أو معالجة قيمة معينة:

في المثال التالي سأقوم بتصميم برنامج يوضح وظيفة أخرى للأمر **Return**, اكتب معي الأسطر التالية في كود مشروع **Movie Script**.

```
on findNumber99
  Repeat with n=1 to 100
    if n=99 then
      Return n
    end if
  end repeat
end
```

الأن من نافذة التراسل **Message Window** قم بتنفيذ المحدد بالطريقة التالية

```
put findNumber99()
--99
```

الأن ماذا تلاحظ؟, لقد ظهر الرقم **99** على الشاشة كما رأيت, الأن ما هو تفسير الكود السابق, وكيف تصرف الكود, سنشرح ذلك الأن,في الكود السابق قمت بعمل محدد جديد **Event** بالإسم **findNumber99** ثم قمت بعمل حلقة تكرار من الرقم **1** الى الرقم **100** وفي السطر الثالث وضعت قاعدة الشرط **IF** حيث إذا وصل البرنامج إلى الرقم **99** فعليه في هذه الحالة وضع قيمة المتغير العددي **n** ثم بالأمر **put findNumber()** قمت بإستدعاء وتنفيذ المحدد من نافذة التراسل, ولاحظ كيف قمت بوضع الأقواس في نهاية أمر الإستدعاء وهذه الطريقة تتبع في حالة رغبتنا في تنفيذ محدد يحوي قيمة سيتم إسترجاعها عبر الأمر **Return**,وكما رأيت في المثال السابق قمنا بعمل إسترجاع وعرض لقيمة معينة, وهذا إستخدام آخر لـ **Return.**

الأن جرب التالي في كود مشروع جديد**Movie Script**:

```
on returnSum
  return 5+5
end
```

الأن من نافذة التراسل **Message Window** أكتب **put returnSum()** ماذا تلاحظ؟ لقد ظهرت القيمة **10**, في هذا المثال قامت **Return** بنفس دور الأمر **Put** المعتاد, حسنا فلتتذكر ذلك الأن وتستوعبه جيدا.

الطريقة الثالثة:إستخدام Return بدون معامل:

يمكن أيضا إستخدام **Return** لوحدها بدون تعيين معامل أو قيمة وذلك سيؤدي الى نفس العمل المكافئ للأمر **exit** الذي سأقوم بعرضه لاحقا, وهو الخروج من المحدد والعودة الى نفس المكان الذي تم منه إستدعاء المحدد, ولفهم ذلك يجب أن نقوم بعمل التطبيق التالي.

```
global myAge

on DoSums
  if voidp(myAge) then
    myAge = 14*2
    Return
  else
    Alert"You Already Have a Value and its 28"
  end if
end
```

لكي يعمل هذا المحدد بشكل صحيح يجب أن تقوم أولا بفتح نافذة التراسل **Message Window** ثم تقوم بمسح جميع المتغيرات العامة بالأمر **Clearglobals** ثم قم بكتابة المحدد **DoSums** في نافذة التراسل **Message Window** فستجد أولا عدم حدوث شئ ثم أعد كتابة المحدد **DoSums** مرة ثانية وهنا ستجد رسالة التنبيه **Alert** تظهر لك الرسالة التي تجدها في الكود أعلاه **You Already Have a Value and its 28**, وفي المثال السابق إستخدام جديد للأمر **Return** وهي الخروج من المحدد, ففي السطر الأول يوجد فحص بإستخدام قاعدة الشرط **if VoidP(myAge) then** لمعرفة إذا كان هناك قيمة موجودة للمتغير **myAge** وفي حالة عدم وجودها فالبرنامج يقوم بوضع قيمة للمتغير وهي نتاج العملية الحسابية **14*2** ثم يخرج البرنامج من المحدد ويتوقف عن العمل وذلك يتم بالأمر **Return**, وفي المرة الثانية من تنفيذ المحدد يتعرف البرنامج على قيمة المتغير ويعرض رسالة **Alert**.

الطريقة الرابعة:إستخدام Return لعمل مسافات سطرية:

من إستخدامات الأمر **Return** المشهورة هي إستخدامها لعمل أسطر فارغة, ذلك كثيرا ما يحدث أثناء توثيق كود السلوك **Behaviour**, الأن جرب من نافذة التراسل **Message Window** كتابة الأمر التالي:

```
Alert"My Name is:"&RETURN&"Mohammed Ibrahim"
```

ستلاحظ من تنفيذ السطر السابق بأن جملة التنبيه **Alert** تعرض على سطرين بدلا من سطر واحد, **Return** هنا قامت بالتدخل ووضعت سطرا فارغا, ذلك مفيد في حالة رغبتك بعمل بالكتابة على سطر ثم ترك سطر.

الطريقة الخامسة:إستخدام Return كمعامل ثابت لبرمجته في لوحة المفاتيح:

رغم أن هذا الإستخدام سيعاد شرحه أثناء شرح برمجة لوحة المفاتيح, لكني رأيت أنه من الهام عرضه هنا تجنبا للخلط, فالأن سنقوم بآخر توظيف لـ **Return** وهي وضعها كمعامل ثابت **Constant Parameter** لبرمجة لوحة المفاتيح, ذلك شئ سهل جدا ,أنظر معي, قم بفتح بنقر كود الكادر **Frame Script** رقم واحد على خط الزمن وأكتب بداخله الأسطر التالية:

```
on exitFrame me
  if (the key = RETURN) then Alert"You Hit Return"
  else
    go the frame
  end if
end
```

الأن قم بتنفيذ البرنامج وتشغيله من الز وإضغط الزر **Return** لتجربة البرنامج, والسبيل الوحيد لك للخروج من البرنامج هو ضغط الزر **ESC**, ولاحقا سأقوم بشرح الأوامر التي تراها في فصل برمجة لوحة المفاتيح.

جميع الأمثلة السابقة ستجدها في الدليل **Return** داخل الدليل **Essential** الموجود في القرص المرفق مع الكتاب

إستخدام النتيجة Result:

تقوم هذه الوظيفة بعرض آخر قيمة قام فيها الأمر **Return** بتخزين قيمة عملية ما, يمكن تجربة ذلك على مثال المنطق السابق, الذي كنا نشرح فيه تصميم المنطق بالأمر **Return**

```
on isDigit SomeChar
  if "0123456789" contains string(someChar) then
    Return TRUE
  else
    Return FALSE
  end if
end isDigit
```

هنا لو قمت الأن بإختبار الكود السابق في نافذة التراسل**Message Window** بكتابة **isDigit(5)** مثلا فسوف يعطيك الكمبيوتر **1** في النافذة الأن جرب كتابة **put the Result** وستلاحظ أيضا أن النتيجة هي الرقم **1** فـ **Result** تعرض آخر قيمة كانت نتاج لعملية قام بها الأمر **Return**, الأن أنظر معي في المثال التالي جيدا.

```
on AddTwoNumbers
  return 5+5
end

on PutTheSum
  AddTwoNumbers
  roll = the result
  Alert"The End Value is"&&roll
```

الأن في المثال السابق, هو كود تم كتابته داخل كود مشروع **Movie Script**, يمكنك تجربة كتابته بنفسك أو الإكتفاء بمتابعة القراءة, الأن أكتب من نافذة التراسل **Message Window** الأمر **PutTheSum** وهو أمر تنفيذ المحدد **Event** الذي تراه في الكود السابق, هنا تلاحظ ظهور رسالة التنبيه **Alert** وهي تعرض رسالة التنبيه **The end value is 10** وهي قيمة آخر عملية قام بها الأمر **Return** في الكود السابق, يمكنك مشاهدة ملف المشروع من الملف **result.dir** من الدليل **Essential** الموجود داخل القرص المرفق مع الكتاب.

إستخدام الأمر Abort للتوقف عن تنفيذ أوامر متتالية في محدد Event:

الأمر **Abort** يقوم بالخروج من المحدد **Event handler** الذي ينفذ منه ولا يقوم بإكمال التنفيذ للأوامر التي تليه, بل يخرج تماما ويتوقف عن تنفيذ الأوامر التالية في المحدد, لفهم ذلك عمليا قم بفتح الملف **abort_command.dir** لكي تلاحظ تصميم المحدد المكتوب داخل كود مشروع **Movie Script** وذلك من الدليل **Essentail** من الدليل الموجود على القرص المصاحب للكتاب, المفترض أنك عندما تقوم بفتح نافذة الكود ستشاهد السطور التالية:

```
on TestRandom3
  Case (Random(3)) of
    1:
      Alert"The Random Value is 1"
      Alert"1"
    2:
      Alert"The Random Value is 2"
      Alert"2"

    3: Alert"The Random Value is 3"
      Abort
      Alert"3"
  end case
end
```

الأن قم بفتح نافذة التراسل **Message Window**, وقم بإستدعاء المحدد **TestRandom3** من النافذة ولاحظ ما يحدث, الكود السابق يقوم بوضع حالة **Case** عندما يقوم بالعثور على رقم عشوائي ما بين الرقم **1** والرقم **3**, وفي كل مرة يقوم فيها بالعثور على رقم يقوم بعرض رسائل تنبيه من نوع **Alert**, فيعرض رسالة **The Random Value is** متبوعة بالرقم ثم يقوم بعرض قيمة الرقم, ما عدا حالة العثور على الرقم **3**, ففي هذه الحالة يقوم فقط بعرض رسالة تنبيه **Alert** تفيد العثور على الرقم **3** ولكنه لا يكمل بسبب مقاطعة إكمال تنفيذ الأوامر بالأمر **Abort** حيث من المفترض أن يكمل بعرض الرسالة **Alert** بعد الأمر, لكن ذلك لا يحدث لأن الأمر **Abort** يقوم بعمل توقف لتنفيذ الأوامر, جرب بنفسك عدة مرات لكي يقوم البرنامج بعرض كل الإحتمالات الرقمية من **1** الى **3** ثم راقب تسلسل تنفيذ رسائل التنبيه.

إستخدام الأمر Exit للخروج من المحدد:

الأمر **Exit** مختلف عن **Abort** فهو يخرج فقط من المحدد الحالي ويعود للمحدد الذي تم إستدعاء الأمر منه, ولكن تنفيذ الأمر **exit** لا يعود بقيمة محددة يمكن إستخدامها, إذا أردت وضع قاعدة شرطية بشكل أفضل وتخزين قيمة يمكن التعامل معها فيما بعد فستجد مرادك في الأمر **Return**, قم بدراسة المثال التالي دراسة جيدة:

```
on CheckforQuickTime
  if not(the QuickTimePresent) then
    alert" You Cant View Quick Time Videos"
    exit
  end if
end CheckforQuickTime
```

في المحدد السابق **CheckforQuickTime** يقوم المحدد بإستخدام المنطق بفحص إذا ما كانت ويندوز مهيئة لتشغيل ولعب ملفات كويك تايم, وفي حالة عدم وجود كويك تايم فالمحدد يعرض رسالة تنبيه ثم يقوم بالخروج من تنفيذ المحدد ويعود مرة أخرى للمحدد الذي تم منه الإستدعاء, في البرامج الكبيرة يكون هناك الكثير من الأشياء التي يقوم بها البرنامج وهناك الكثير من تعشيش المحددات أو الوظائف **Nested Functions**, سبق أن تحدثنا وتعرضنا لهذا الموضوع بأكثر من شكل, المقصود أن ليست كل الأوامر التي تراها في لغة اللينجو يمكن الإستفادة منها مباشرة, بل بعضها يعمل بطريقة غير مباشرة, إذا لم يكن لديك كويك تايم مثبت في نظام التشغيل لديك فقم بفتح وإختبار الكود من الملف **checkquicktime.dir** الموجود في دليل **Essential** الموجود في الدليل المصدري للكود على القرص.

إستخدام الأمر Quit للخروج من البرنامج أو دايركتور إلى سطح المكتب:

يستخدم الأمر **Quit** للتوقف تماما والخروج من البرنامج, فلو كان البرنامج الذي تنفذه على هيئة مشروع نهائي **EXE** وهو ما يطلق عليه دايركتور **Projector** فإن الأمر يقوم بالخروج تماما إلى سطح المكتب في ويندوز, وكذلك لو قمت بتنفيذ هذا الأمر من داخل بيئة التطوير **Director** نفسه فإن دايركتور نفسه يغلق البرنامج الذي تم تنفيذ الأمر منه ثم يغلق نفسه بعدها ويخرج أيضا إلى سطح المكتب, هذا الأمر مشترك بين كل من بيئتي **Authoring** و **Executing** أو ما يطلق عليه **Authoring Time** و **Run Time**, حيث تمثل **Authorign Time** وقت التطوير والإنتاج أو بيئة التطوير وهي كل ما يتم تصميمه داخل الدايركتور بينما يسمى وقت تشغيل المشروع النهائي من الملف التنفيذي **EXE** المسمى **RunTime**, يمكنك الأن تجربة الأمر من نافذة التراسل **Message Windos** مباشرة أو من خلال محدد من تصميمك, كما تعلمت سابقا,الأن جرب من نافذة التراسل كتابة:

```
Quit
```

ستلاحظ بأن دايركتور أغلق كل النوافذ ثم أغلق نفسه تلقائيا بعد ذلك وخرج إلى سطح المكتب في نظام التشغيل.

إستخدام الأمر Halt للخروج من البرامج:

يستخدم الأمر **halt** للخروج من البرنامج والتوقف تماما, لكن ذلك لا يشمل برنامج الدايركتور نفسه كما يفعل **quit** ولكن يمكنك أيضا التعامل مع الأمر **halt** في وقت التصميم **Authoring Time** وفي وقت التنفيذ **Run Time**, عموما هذا الأمر هو الأكثر إستخداما من قبل مطوري ومبرمجي الدايركتور, يمكنك تجربة الأمر من خلال أي محدد, عادة يتم الخروج من البرامج عبر أزرار تحمل المحددات **mouseUp** أو **mouseDown** , جرب الأمر كيفا تشاء.

إستخدام الأمر Nothing لعمل لا شئ:

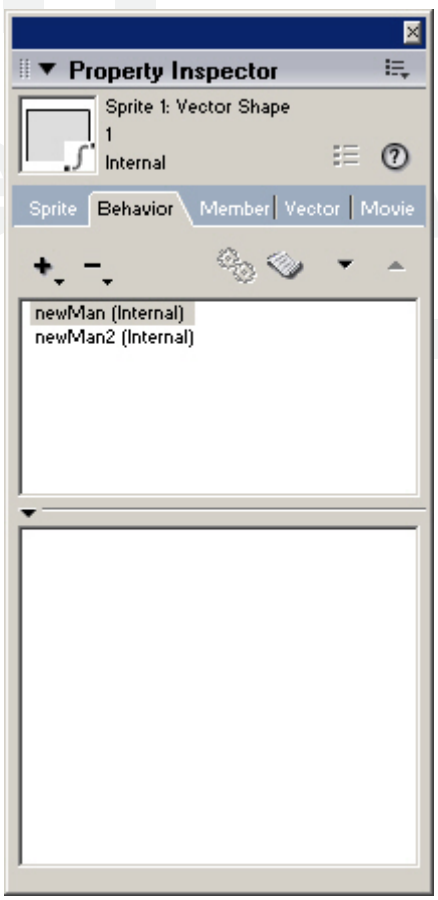
لا يقوم الأمر **nothing** بعمل أي شئ, فهو إسم على مسمى, ورغم ذلك فهذا الأمر هام جدا في تطبيق المنطق خاصة في وجود تعشيش لقاعدة الشرط **IF** , أدرس المثال التالي جيدا, ثم أقرأ التعليق أسفله.

```
on mouseDown
   if the clickOn = 1 then
      if sprite(1).moveableSprite = TRUE then member("Notice").text = "Drag the ball"
      else nothing
   else member("Notice").text = "Click again"
   end if
end
```

في المثال السابق يوجد المحدد **mouseDown** ويليه سطر قاعدة الشرط **if the clickOn = 1 then** وهو يعني لو حدث ضغط بزر الماوس على العنصر رقم **1** وهو **Sprite 1** إذا **Then** ويليها تعشيش آخر لقاعدة الشرط **IF** وهي إشتراط أن العنصر **Sprite 1** قابل للسحب والإسقاط(حر الحركة) وذلك بالخاصية **moveableSprite** إذا فالحقل النصي **Field** ذو الإسم **Notice** سيعرض الجملة **Drag the ball** والإ فقم بعمل لا شئ, أو المقصود لا تقوم بعمل أي شئ, ويكمل البرنامج بجملة وإلا **else** وهي أن يعرض الحقل النصي العبارة **Click again** ثم تنتهي قاعدة الشرط **end if**, إن ملخص الكود السابق هو إنه إذا تم الضغط بزر الماوس على **Sprite 1** وكان هذا العنصر قابلا للحركة إذا قم بعرض رسالة **Drag the ball** اي قم بسحب وإدراج الكرة وإلا فالبرنامج لا يقوم بعمل شئ والحقل سيعرض رسالة **click again.**

إستخدام الأمر ()StopEvent لإيقاف حركة التراسل بين الكود:

في النسخ القديمة من دايركتور كان هذا الأمر هو **dontPassEvent** وما زال هذا الأمر يعمل في دايركتور ام اكس **2004**, لأن سياسة ماكروميديا هي التوافق, وهذا الأمر يقوم بإيقاف تراسل الكود إلى العناصر أو إلى الكود ذاته, لفهم ذلك قم بفتح المثال **StopEvents_Command.dir** حيث ستجد مثالا عمليا يوضح لك أحد الطرق المثالية لإستخدام هذا الأمر, فهنا لدينا مربع في المسرح**Stage** وهو يمثل زرا لذلك قم بتشغيل البرنامج بالضغط على زر ▶ م شاهد ما يحدث, ستلاحظ ظهور رسالة تنبيه **ALERT** تقول**BLAH BLAH BLAH** وهذا أي كلام, ليس له معني محدد, الأن قم إضغط على العنصر أمامك على المسرح, حيث تلاحظ وجود كودين من نوع السلوك **Behaviour** مرتبطين معه, أحدهما عنوانة **newMan** والآخر عنوانه **newMan2**, من نافذة **Window** لديك أظهر النافذة الفرعية التي تسمى نافذة الخصائص **Property Inspector** أو إضغط على الإختصار الخاص بها **Ctrl+Alt+S**, ثم من التبويب **Behaviour** يمكنك بوضوح مشاهدة وجود السلوكين معا بحيث أن السلوك **newMan** يعلو السلوك **newMan2** في أسبقية التنفيذ, أنظر الشكل التالي:



الأن قم بتحرير الكود **newMan** بإختياره أولا ثم قم بالضغط على زر تحرير الكود  تحرير الكود.

الأن إقرأ معي أسطر الكود التالية:

```
on mouseDown
  global Flowers
  Flowers = 3
  if Flowers = 3 then
    Alert"BLAH BLAH BLAH"
      StopEvent()
  end if
end
```

الأن يمكنك ملاحظة أن هذا السلوك مرتبط بالشكل المربع على نافذة المسرح, وهو يبدأ بالمحدد **mouseDown** وعند الضغط عليه فالمحدد يقوم بعمل متغير عام ثم يعطيه القيمة **3** ثم يضع قاعدة الشرط **IF** ثم بناءا عليها يعرض رسالة التنبيه **Alert** التي لا تحمل معنى محدد, ثم تجد الأمر **()StopEvent** ولكنك لم تلاحظ شيئا قد حدث حقا عند تشغيلك للبرنامج سوى ظهور عبارة **Alert** بكلمات غير مفهومة, في الواقع لقد حدث أمر هام جدا, وللملاحظة ما حدث قم بفتح نافذة تحرير السلوك الثاني **newMan2** وأقرأ معي ما هو مكتوب داخلها.

```
on mouseDown
   Alert"The Events Stopped"
end
```

هناك محدد أخر وهو نفسه **mouseDown** وأسفله سطر يحمل الأمر **Alert** برسالة ما, ولكن الغريب كما ترى أن شيئا من هذا لم ينفذ على الإطلاق, لماذا؟, ذلك حدث بسبب الأمر **()StopEvent**, فهذا الأمر تم تنفيذه بعد رسالة **Alert** الأولى في السلوك **newMan** وذلك أدى الى توقف نظام التراسل في لغة اللينجو لهذا العنصر **Sprite 1**, إذن تستطيع أن تفهم مما سبق بأن حدث توقف في التسلسل الهرمي للتنفيذ للكود وهو ما يسمى بـ**Message hierarchy** , هذا إستخدام بسيط ومثال بسيط جدا لما يمكن ان تفعله في السيطرة على مسارات التنفيذ, يمكنك تصميم أكواد معقدة تستطيع أن تفكر بواسطة قواعد شرط محكمة وتقود البرنامج بالطريقة التي تودها, هذا الأمر من الأوامر القوية التي تقوم بالتدخل في نظام الكود **Script Flow**, لاحظ دائما أن الأمر **()StopEvent** يعتمد تماما على مكانه وموقعه بين الأكواد المختلفة وترتيبها وما هو مطلوب منها, لذلك قم بإستخدامه بحذر وتأكد من موضعه حتى لا يؤثر في المنطق الذي تقوم بتصميمه لبرامجك الخاصة,ولاحظ أيضا أن هذا الأمر يعمل فقط من خلال المحدد**Event** الذي إنطلق منه, ويوجد أمر آخر يقوم بعمل عكس هذا الأمر وهو الأمر **pass**.

إستخدام الأمر pass في تمرير التنفيذ للكود :

هذا الأمر يقوم بعكس عملية الأمر السابق, يمكنك تصميم برنامج يحويان الإثنان معا ويمكنك عمل ذلك بإستخدام قاعدة شرط قوية لكي تحكم الموضوع, هناك الكثير من الطرق المفيدة في إستخدام الأمر **pass** مثل إستخدامه للتأكد من أن المستخدم يضغط الأزرار الصحيحة في لوحة المفاتيح, أدرس معي المثال التالي:

```
on keyDown me
   legalCharacters = "1234567890"
   if legalCharacters contains the key then
      pass
   else
      beep
   end if
end
```

في المثال السابق, يوجد المحدد **KeyDown** المسؤول عن مراقبة الضغط على زر ما في لوحة المفاتيح, ولديك أيضا قاعدة الشرط **IF** حيث أن القاعدة هي مراقبة الطريقة التي يقوم بها المستخدم من إدخال الرموز من لوحة المفاتيح إلى حقل نصي إفتراضي **Field** حيث المسموح فقط هو الأرقام **1234567890** ,وعدا ذلك فلن يكمل الكود عمله بل سيسمع المستخدم صوت **Beep**, صوت التنبيه الذي قمنا بإستخدامه في الفصل الثاني.

الأن قم بفتح الملف **Pass_command.dir** لكي تتعرف على طريقة إستخدام **pass** و**()StopEvent** معا من الدليل **Essential** الموجود في القرص المرفق مع الكتاب من الدليل المصدري للكود.

الأن قم بقراءة الأسطر من السلوك **h1** في الكاست, وهو كالتالي:

```
on MouseUp
  CheckRandomValue
end

on CheckRandomValue
  MyNumber = Random(2)
  if MyNumber = 1 then
    put MyNumber
    StopEvent()
  else
    put MyNumber
    Pass
  end if
end
```

حسنا,لديك هنا قاعدة شرط أسفل المحدد **CheckRandomValue** بأنه لو كانت قيمة المتغير **MyNumber** تساوي الرقم **1** إذن فيجب إيقاف تمرير التنفيذ للكود بإستخدام **()StopEvent** بينما لو القيمة كانت عدا ذلك **Else** فقم بالتمرير**Pass**.

حسنا الأن قم بقراءة الكود **h2** الموجود أيضا في الكاست, ستجد التالي:

```
on MouseUp
  Alert"Your Number Sure is Two(2)"
end
```

أنه مجرد محدد لضغط الماوس يعرض رسالة تنبيه **Alert**.

الأن قم بفتح نافذة التراسل **Message Window** ,يمكنك ذلك بالإختصار **Ctrl+M** وضعها جنبا بحيث تكون واضحة, وإمسح أي شئ يوجد في داخلها لكي تكون واضحة, الأن قم بتشغيل البرنامج بالضغط على ال ▶ ثم قم بالضغط على المربع الذي تراه أمامك في نافذة **Stage** وهي المسرح وراقب في نفس الوقت ما يحدث داخل نافذة التراسل **Message Window**, الأن تلاحظ ظهور الرقمين واحد وإثنان, وعند ظهور الرقم **2** تظهر رسالة التنبيه, لو عدت معي لقراءة الكود السابق ستجد أن سبب ذلك هو أن المتغير يحمل قيمة عشوائية بين الرقمين **1** و **2**, وإذا كانت قيمة المتغير هي **1** فالبرنامج يوقف سير التنفيذ بالأمر **()StopEvent** أما لو كانت القيمة هي **2** فالبرنامج يستمر في التنفيذ **Pass** وينفذ الكود التالي وهو **h1** فتظهر رسالة التنبيه.

إستخدام Play و Play Done في توجيه مسارات البرنامج:

الأمر **play** يشبه أمر التجول **Go** لكنه يختلف عنه في أنه ينتظر رد فعل معين من الأمر **Play done** لذلك رأيت أنه من الأفضل شرح **play** و **play done** منفصلة عن أوامر التجول لأنها مرتبطة بموضوع توجيه كود البرنامج **Script Flow**, تابع معي لكي تستوعب طريقة عمل الأمران المرتبطان معا.

الأن قم بفتح الملف **play_command.dir** من الدليل **play** الموجود داخل الدليل **Essential** الموجود في القرص المرفق مع الكتاب, ثم قم بتشغيل المشروع بحيث تكون نافذة خط الزمن **Score** واضحة أمامك, أغلق نافذة المسرح **Stage** لكي يكون **Score** واضحا أمامك, الأن ماذا تلاحظ؟ ظهرت رسالة **ALERT** بداخلها عبارة **you are playing frame 50**, اضغط **ok** لغلق النافذة, لو تابعت عينك على ما يحدث في خط الزمن لعرفت أن رأس التشغيل المنزلقة **Playing Head** ذات اللون الأحمر تقفز من الكادر الأول **Frame 1** الى الكادر رقم **50** مباشرة, وفي ذلك الوقت تحديدا تظهر رسالة التنبيه **Alert**, حسنا دعنا الأن نغلق البرنامج لكي نقوم بمراجعة الكود الموجود على خط الزمن في كل من **frame 1** و **frame 50**, الأن إضغط **ESC** لغلق البرنامج.

الأن أنقر بزر الماوس الأيسر مرتان على الكود الموجود في الكادر الأول على خط الزمن, ستلاحظ فتح نافذة تحرير هذا الكود وستلاحظ وجود الأسطر التالية داخلها:

```
on exitFrame me
  play frame 50
end
```

هنا نجد الأمر **play frame 50** اسفل محدد **Exitframe** مباشرة, وهذا الأمر هو الذي يقوم بتوجيه التنفيذ لرأس التشغيل بحيث تذهب مباشرة لكي تلعب أو تشغل البرنامج بدءا من الكادر رقم **50**, الأن أغلق نافذة تحرير هذا الكود وقم بفتح النافذة الأخرى الموجودة في الكادر **50** من على خط الزمن بنفس الطريقة (بالنقر عليها مرتان بالزر الأيسر).

الأن بعد فتح الكود السابق تلاحظ ظهور الأسطر التالية:

```
on exitFrame me
  Alert"You are playing frame 10"
  play done
end
```

هذا الكود يستخدم نفس المحدد **exitframe** لكن هذه المرة تجد سطر يحمل رسالة **ALERT** والسطر **Play done** وهذا هو السطر الذي يحملك مرة ثانية إلى تنفيذ الكود في الكادر رقم **1**, إذن تستطيع أن تفهم من المثال السابق بأن **Play** تقوم بعملية الذهاب الى مكان ما بينما **play done** تنهي عملية الذهاب ثم تعود مرة أخرى الى نفس المكان الذي انطلق منه الأمر **play**, ذلك مفيد جدا في عملية التحكم في تنفيذ المشروع, يمكنك أيضا تجربة **play** مع **movie** , جرب ودون ملاحظاتك فهذا مفيد جدا.

طرق جديدة لربط برامجك مع الكود:

سأشرح الأن طرق جديدة عليك كلية في التعامل مع الكود الموجود في ملف مشروع **DIR** ما, هناك العديد من الأشياء الهامة التي أريدك أن تلم بها تماما, ورغم أن ما سأشرحه هنا نادرا ما تجده مستخدما, إلا إنه من الهام فهمه وإستيعابه.

الأن لفهم الطريقة الأولي الجديدة للتعامل مع الكود وقم بفتح الملف **Commander.dir** من الدليل **ScriptsEnabled** الذي تجده داخل الدليل **Essential** من الدليل المصدري للكود المرفق مع الكتاب.

الأن قم أظهر نافذة المسرح **Stage** بحيث تكون أمامك, إضغط **Ctrl+1** لكي تكون أمامك النافذة وظاهرة(الضغط مرة يخفيها ومرة أخرى يظهرها), ستجد عبارة **Linked Movie** باللون الأسود على خلفية بيضاء, إذا كان هذا ما ظهر أمامك فقم بتشغيل البرنامج بالضغط على زر **play** وهو الزر ◂ ما تعلم, والأن ماذا تلاحظ؟, لقد ظهرت لك رسالة **Alert** تخبرك بأن **The Random Number is** ثم تكتب أي رقم عشوائي, حسنا الأن أغلق البرنامج ثم قم بالتوجه الى الكاست **Cast** , يمكنك الضغط على **Ctrl+3** لإظهار أو إخفاء نافذة أعضاء المشروع **Cast**, الأن ماذا تلاحظ في نافذة الكاست **Cast**,أنظر الشكل التالي:

الشكل على اليمين مقارب لما يجب أن تبدو عليه شاشتك, لاحظ العنصر **Linked.dir** الموجود في الكاست ويحمل الرمز الشبيه بملف دايركتور الذي يعرف بالـ **DIR**.

الأن تسأل نفسك؟ شئ محير؟ أين ذهب الكود؟ أين ذهبت أسطر الكود؟ هل يوجد كود مشروع خفي **Movie Script**

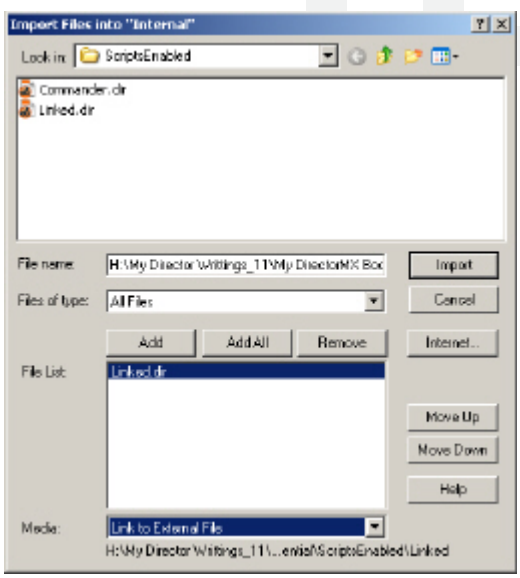
في الواقع, إن ما رأيته توا هو ملف دايركتور بصيغة **DIR** مربوط بالكامل ولكن بشكل خارجي **External** مع برنامجك الحالي, إنها ميزة ممتازة ومهمة في الدايركتور, وللتأكد من ذلك إذهب لفتح الملف **Linked.dir** الموجود في نفس المكان الذي وجدت فيه الملف **Commander.dir** وهناك سيمكنك من قراءة الكود التالي حيث ستجده بالإسم **RandomMan** في نافذة أعضاء المشروع **Cast** الموجودة في ملف **Linked.dir** الذي قمت لتوك بفتحه:

```
on StartMovie
  PutRandomNumber
end

on PutRandomNumber
  Alert"The Random Number is"&&Random(500)
end
```

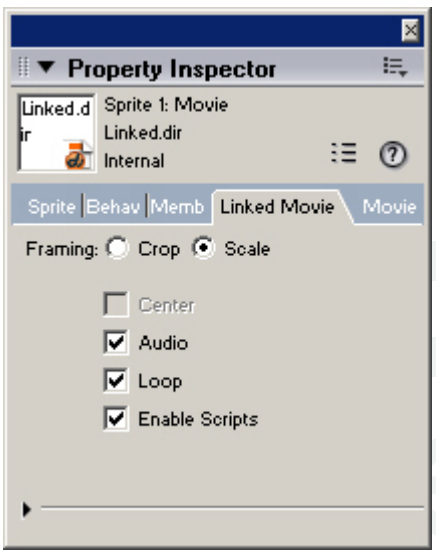
إن الكود السابق الذي تراه هو السبب لما تراه من تأثير عند تشغيل الملف **Commander.dir**, يمكنك الأن تشغيل البرنامج لتشاهد نفس ما تراه في ملف **Commander.dir** وهو ظهور رسالة التنبيه **Alert** برسالة تفيد بقيمة رقم عشوائي.

الأن تسأل نفسك عن الطريقة التي نفذ بها هذا التكنيك الجميل والمبهر؟, ذلك سهل جدا, فقط يمكنك فتح ملف مشروع جديد **New Movie** ثم من نافذة الكاست **Cast** يمكنك عمل إستيراد اي ملف **DIR** ترغبه وليكن هو نفسه ملفنا **Linked.dir** ثم تختار من نافذة **Import** الملف **Linked.dir** ثم تضغط الزر **Add** وقبل الضغط على الزر **Import** إختر من **Media** الإختيار **Linked to External File** ثم إضغط **Import**, أنظر الشكل التالي:



الأن قم بسحب العنصر **Linked.dir** من الكاست الى القناة الأولي **Channel 1** في خط الزمن ليتحول العضو إلى **Sprite 1** ثم قم بتشغيل المشروع بالضغط على ال

الأن إذا لم يعمل المشروع ولم تظهر رسالة التنبيه **Alert** فقم بفتح نافذة **Propert Inspector** وهي نافذة خصائص العضو من قائمة **Window** أو الضغط أولا على العنصر على خط الزمن **Score** ثم الضغط على مفاتيح **Ctrl+Alt+S** حتى تظهر لك النافذة, ستجد في التبويب **Linked Movie** إختيار بالإسم **Enable Script** قم بوضع علامة صح بجانبه ثم أعد تشغيل المشروع مرة أخرى, أنظر الشكل التالي:



تلاحظ في الشكل ظهور التبويب **Linked Movie** وهذا التبويب يحوي عدة خيارات منها **Audio** وهي لتمكين الصوت في الملف المربوط بالمشروع الحالي و **Loop** وذلك لكي يعمل الملف المرتبط بالمشروع الحالي بشكل مستمر, والإختيار الأخير **Enable Scripts** وذلك لكي يعمل الكود الموجود في الملف المرتبط الحالي, والمطلوب وضع علامة صح إذا لم تكن موجودة.

يمكنك أيضا تفعيل تشغيل الكود في الملف **Linked.dir** المرتبط بالمشروع عن طريق كتابة الأمر التالي من نافذة التراسل **Message Window**.

```
member("Linked.dir").ScriptsEnabled = TRUE
```

ستجد هذا الأمر أيضا موجود في كود المشروع **Movie Script** الخاص بالملف **Commander.dir**, وكما قمت سابقا بعمل إستيراد للملف **Linked.dir** كملف **External** قم بإعادة إستيراده من جديد كملف **Internal** بإختيار **Standard Import** عند ظهور نافذة الإستيراد **Import** ولاحظ الفرق, جرب بنفسك لكي تتعلم الفرق بين الإثنين.

هذه إحدى الطرق العجيبة للتعامل مع اللينجو من داخل الدايركتور, تابع معي..هناك العديد من المفاجآت الأخرى!.

إحفظ أكوادك بالدايركتور نفسه!:

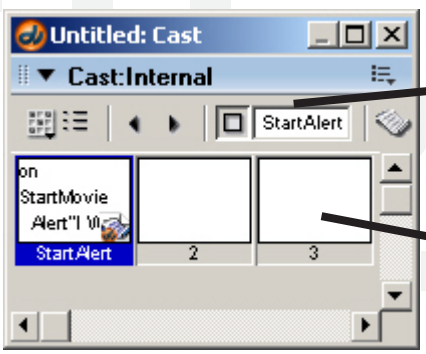
الأن سنجرب معا طريقة حفظ أكوادك التي تكتبها بواسطة الدايركتور نفسه, أكتب الأن أي كود تريده على سبيل المثال في كود مشروع جديد, لنكتب شئ بسيط مثل الأسطر التالية:

```
on StartMovie
  Alert"I Will Save my Codes Now!!"
end
```

الأن قم بتسمية الكود الذي أنشاته توا من نافذة الكاست بكتابة إسم مناسب للكود مثل **StartAlert** ثم قم بفتح نافذة التراسل **Message Window** وضع السطر التالي فيه:

```
member("StartAlert").linkAs()
```

الأن المفاجـــأة!, تلاحظ ظهور نافذة صندوق حواري تطلب منك حفظ الكود ووضع إسم له, حسنا ضع الإسم الذي تريده وليكن نفس الإسم وإحفظ الملف على سطح المكتب, ستلاحظ أن الملف قد تم إنشاؤه بالإسم **StartAlert.ls** حيث يأخذ الإمتداد **LS** , أنظر معي الأشكال التالية:



أعد إستيراد أكوادك المحفوظة بالدايركتور نفسه!:

الأن قم بعمل عملية عكسية!, قم بإعادة إستيراد الكود الذي قمت بحفظه على سطح المكتب مرة ثانية الى نافذة أعضاء المشروع **Cast**, الأن إضغط بالزر الأيمن على أي عضو كاست خالي (باللون البيض كالمربعات الخالية في الشكل أعلاه), الأن إختر **Import** كما ترى من الشكل التالي لإستيراد العنصر:

الأن قم بإختيار الملف **StartAlert.ls** الموجود إفتراضيا على سطح المكتب, أو من المكان الذي قمت بتخزينه فيه, ثم من الإختيار **Media** تأكد من أنه **Standard Import** ثم إضغط **Import** لإتمام الإستيراد, تلاحظ الأن ظهور الكود الذي قمت بإستيراده الى داخل نافذة الكاست مباشرة, لاحظ أن ملف الكود الذي قمت بحفظه منذ قليل بالإمتداد **LS** يمكن أيضا تحريره بإستخدام محرر الويندوز **NotePad**, الجميل أن ملف الكود يحتفظ بخصائصه ونوعه.

الطرق الديناميكة لتحرير الكود Dynamic Script Creation:

الأن قم بإعداد فنجان كبير من الشاي أو القهوة لأنك ستتعامل الأن مع شئ مهم جدا, ربما سيبهرك!, لكنه مهم جدا أن تستوعبه وتفهمه.

الأن سأشرح بعض الطرق التي يتبعها المحترفون فقط في تحرير وإنشاء الكود, الطريقة التي سأشرحها متقدمة للغاية, هي سهلة, لكن لا تستخدمها إذا لم تكن ذات فائدة بالنسبة لك, هي فقط تستخدم من قبل بعض المطورين, ملخص هذه الطريقة هي تصميم الكود المستخدم في برنامجك لكن بطريقة غير تقليدية, فبدلا من إنشاء كود مشروع **Movie Script** أو كود السلوك **Behaviour** فيتعمد المطورون كتابة أكوادهم على شكل نصوص **TEXT** ثم كتابة كود خاص يقوم بتحويل هذا النص **TEXT** الى كود, السبب الرئيسي في ذلك يكمن في جعل برامجهم مرنة التطوير, حيث يتبادل المطورون الملفات النصية بدلا من ملفات الدايركتور وبسهولة شديدة يتمكنون من تطويرها والإضافة عليها ثم إعادة مزجها بالمشروع الكامل للتطبيق الذي يقومون بتطويره, وعادة تستخدم الطرق الديناميكية لإنشاء الكود أثناء وقت التنفيذ ايضا **RunTime** للبرنامج, وتستخدم في هذه الطريقة عدة أوامر ووظائف جديدة مثل الأمر **Do** والخاصية **ScriptText** بالإضافة لتعاون كل ذلك مع حقول نصية **Fields**, عموما سأشرح الطرق بإذن الله الأن ولك مطلق الحرية فيما بعد في إستخدامها أو لا, لكن من المهم أن تكون على دراية بها.

إستخدام الأمر Do:

الأمر **do** يمكنه تحويل الأوامر المكتوبة على شكل نصوص **TEXT** داخل متغيرات حلقية **String** إلى نصوص وأوامر لغة اللينجو وتنفيذها والتعامل معها, يستخدم عادة في تنفيذ النصوص الموجودة داخل حقول نصية, جرب معي التالي من نافذة التراسل **Message Window**.

```
do "beep"
```

ستسمع صوت **Beep** إذا كانت سماعات الكمبيوتر تعمل, أو الهيدفون موصل بإذنك, الأن قم بعمل حقل نصي جديد في نافذة الكاست, قم بإختيار أي مكان خالي وإضغط **Ctrl+8** لإنشاء حقل نصي جديد, الأن ضع إسم لهذا الحقل النصي, أكتب مثلا الإسم **test** ثم أكتب داخل الحقل النصي العبارة **"Alert"This is Amazing**, تلاحظ أنك كتبت نصا هو نفسه الأمر **alert** وتليه عبارة بنفس الطريقة التي تتبعها تماما في كتابة الأمر في أي مكان كنافذة التراسل **Message Window**, الأن أغلق نافذة الحقل النصي **field** وعد مرة أخرى لفتح نافذة التراسل **Message Window**, الأن أكتب في النافذة الأمر التالي:

```
do field"test"
```

الأن تلاحظ أن رسالة **Alert** تم تنفيذها تماما كأنك قمت بكتابتها خلال نافذة التراسل **Message Window**, حسنا دعنا الأن نقوم ببعض الحيل الخاصة, لنستخدم **do** في تعريف متغير, ليكن متغير عام مثلا, الأن جرب التالي من نافذة التراسل:

```
do "global myglobal"
do "myglobal = 5"
put myglobal
```

الأن تلاحظ أن دايركتور فشل في تعيين المتغير العام, دايركتور يعرض **VOID** في نافذة التراسل **Message Window**, ذلك سببه أن الطريقة السابقة هي طريقة خاطئة, هناك طريقة أخرى لتعيين المتغير العام بإستخدام **do**.

الأن جرب كتابة الأمر التالي:

```
do "global myglobal" & RETURN & "myglobal = 5"
put myglobal
--5
```

الأن نجحت الطريقة السابقة في تعيين وتحديد القيمة **5** للمتغير العام **myglobal**, ممتاز, ماذا بعد يجب أن تعرفه عن الأمر **do**, الأن قم بعمل نافذة كود مشروع جديد **Movie Script** ثم قم بكتابة الأسطر التالية:

```
on SetWithDo
  global myAge
  do "MyAge = 28"
end
```

الأن قم بفتح نافذة التراسل **Message Window** ثم قم بكتابة إسم المحدد الذي أنشأته لتنفيذه **SetWithDo** وأكتب بعدها مباشرة **Put Myage**,ماذا تلاحظ؟ لقد ظهر الرقم **28**, الأن قم بالعديد من التجارب لو أردت حول الأمر **do**.

قم بفتح الملف **do_ex1.dir** لكي تشاهد الأمثلة السابقة عن قرب, وجرب بنفسك من دليل **Essential** الموجود داخل الدليل المصدري للكود في القرص المرفق مع الكتاب.

تحرير الكود في وقت التنفيذ Setting ScriptText at RunTime:

تستخدم الطرق التي سأشرحها الأن عادة في وقت التنفيذ **RunTime**, وليس في وقت التصميم **Authoring Time**, في الواقع عندما تتحول ملفاتك من **DIR** يمكن فتحها بواسطة الدايركتور إلى ملفات **DXR** محمية كما ستتعلم لاحقا, ستتحول كل الأكواد التي كتبتها بلغة اللينجو إلى لغة الآلة وتختفي تماما وتصبح عناصر الكود في نافذة الكاست موجودة ولكنها لا تحتوي على أي اسطر للكود, وبعض المبرمجين يلجأون لذلك لتجنب الفقد التام للكود أثناء عملية الحماية, وبعض المبرمجين يريدون بناء برامجهم بطرق معقدة بحيث لا تفهم من الآخرون.

هناك العديد من الأوامر التي تحقق ديناميكية إنشاء الكود وهي تعمل معا, ذلك يتم في كل من طوري التحرير **Authoring Time** والتنفيذ **RunTime** لكن طبعا تجهيز هذه الأكواد للعمل يتم في طور التحرير **Authoring** فقط.

الأن قم بفتح الملف **ScriptText.dir** الموجود في دليل **Essential** الذي يوجد داخل الدليل المصدري للكود في القرص المرفق مع الكتاب,الأن لاحظ أن نافذة الكاست لهذا الملف تحتوي على عضويين فقط احدهما حقل نصي **Field** والآخر هو كود مشروع **Movie Script** قم ألأن بفتح نافذة التراسل **Message Window** وإجعلها أمامك, ثم قم بكتابة الأمر التالي:

```
CreateCode
```

الأن إجعل نافذة الكاست أمامك؟ ماذا تلاحظ؟ لقد ظهر كود جديد ثالث, ظهر عضو ثالث في نافذة الكاست بالإسم **MovieScriptCode**, الأن ماذا حدث؟ الأن من قائمة **File** في الدايركتور إضغط الأمر **Revert** وعند ظهور رسالة تخيرك بين **Revert** و **Cancel** قم بإختيار **Revert**, المفترض إن ذلك يعيد ملفك الأصلي **DIR** بدون أي تغيير.

الأن قم بفتح وتحرير كود المشروع ذو الإسم **CreateCodes**, الإن إقرأ معي الأسطر المكتوبة:

```
on CreateCode
  set newMember = new(#script)
  set newMember.name = "MovieScriptCode"
  member("MovieScriptCode").scriptText = member("MovieScript").text
end
```

تلاحظ بداية وجود المحدد **CreateCode** وأسفله يوجد ثلاثة أسطر, السطر الأول يحدد قيمة للمتغير **newMember** بأنها كود جديد **new(#Script)** والسطر الثاني يقوم بإعطاء إسم جديد لهذا الكود الذي سوف ينشأ فور إستدعاء المتغير والإسم هو **MovieScriptCode** ثم السطر الأخير يقوم بتعبئة الكود الخالي الذي تم إنشاؤه بأسطر النصوص الموجودة في الحقل **MovieScript**, وفي النهايه يتم إنشاء الكود الجديد عندما تقوم بإستدعاء المحدد **CreatCode** كما فعلت من نافذة التراسل **Message Window** أو من خلال محدد آخر **Nested**.

يمكن أيضا تحديد نوع الكود إذا كان كود الأب **Parent** أو سلوك **Behaviour** أو **Movie** عن طريق إضافة سطر بعد آخر سطر في المحدد السابق وهو السطر :

```
newMember.scriptType = #score
```

حيث أن نوع الكود **ScriptType** تكون أما **Score** وهي كود السلوك **Behaviour** بينما تكتب **Movie** لكود المشروع **Movie Script** وتكتب **parent** وهي تمثل كود الأب **Parent Script** وذلك بعد الرمز **#** وهو الـ**Symbol**, سنفصل الحديث عنه لاحقا أثناء الحديث عن المصفوفات **Lists**.

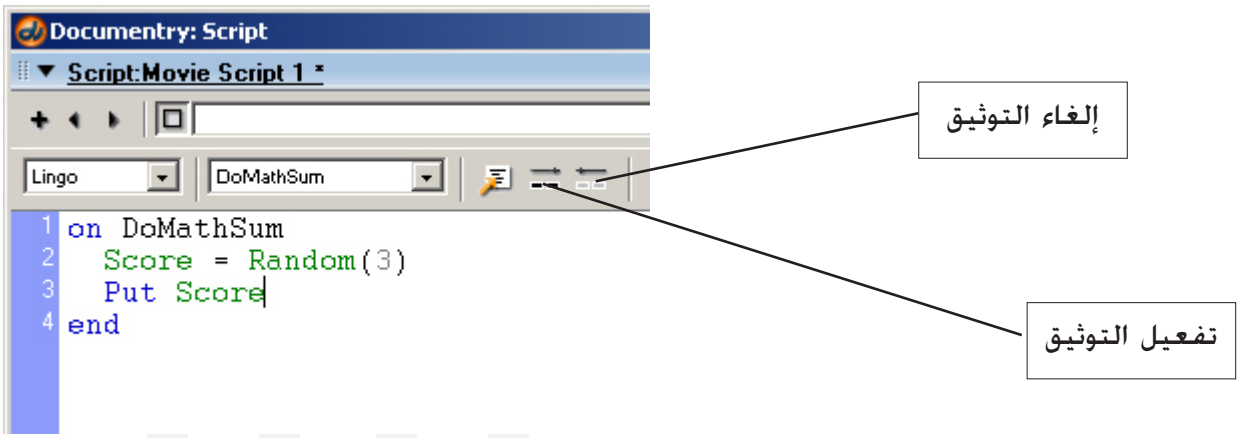
توثيق الكود Script Documentary:

الأن سأشرح شئ بسيط جدا, لكنه مهم جدا, وهي عملية توثيق الكود **Documentration** فهي عملية هامة جدا لكي تستطيع إضافة التعليقات على أكوادك المكتوبة, كأن تضيف عنوان للمحدد الذي تتعامل معه كعناوين حقيقية واضحة باللغة الانجليزية أو العربية لو أردت, لكن إستخدام الإنجليزية أفضل وأسرع, كما أن أدوات التوثيق تفيد أيضا في عملية إجراء التجارب على الأكواد حيث أنها تقوم بتفعيل أو عدم تفعيل الأكواد مما يتيح لك تجربة الكثير من الأوامر, لفهم ذلك عمليا سنبدأ بشرح التوثيق, الأن أنظر الى الكود التالي:

```
on DoMathSum
  Score = Random(3)
  Put Score
end
```

الكود واضح ومفهوم بالنسبة لك, ليس لأنه بسيط فقط بل لأنك كتبته توا, لكن مع مرور الزمن عندما تقوم بمراجعة الكود ربما تسأل نفسك ماذا يعني هذا الكود بالنسبة للبرنامج ككل, لذلك أضاف دايركتور أدوات فعالة لكي تستطيع كمبرمج ومطور أن تكتب تعليقاتك الخاصة على الكود, قبل او بعد أو أثناء كتابة الكود نفسه, يتم ذلك عن طريق إستخدام أدوات إضافة التعليقات التي توجد في جميع نوافذ الأكواد وهي ذات الشكل -- شرطتان متتابعتان لإضافة أو إزالة التعليقات.

الأن أنظر الشكل التالي لكي تتعرف عن قرب عن أدوات توثيق الكود وهو مقطع من نافذة كود مشروع **Movie Script:**



الأن قم بفتح الملف **Documentry.dir** وقم بتحرير المحدد بترك مسافة سطر واحد بين عنوان المحدد الظاهر أمامك وهو **DoMathSum** بالضغط على مفتاح الإدخال **Enter** قبل كلمة **on** وهنا يمكنك تفعيل زر التوثيق بالضغط على الزر ذو الشرطتان باللون الأسود, يمكنك فيما بعد كتابة ما تشاء, أنظر لتلاحظ معي في المثال التالي:

```
--This Code is For Score
on DoMathSum
  Score = Random(3)
  Put Score
end
```

هنا قمت بكتابة عبارة **This Code is For Score** , هذا أفضل من أن تترك الكود من غير عنوان, الأن إذا مر زمن طويل على هذا البرنامج فيمكنك ببساطة أن تعود إليه مرة أخرى, لأنه موثق **Documented** ويمكنك بسهولة معرفة طريقة عمل اجزاء برنامجك منها, ويمكنك أيضا وضع الشرطتان بعد نهاية اي سطر لكي تكتب عبارة تذكيرية, يمكن ذلك أن يناسب تذكيرك بمحددات معينة أو بمتغيرات قمت بإنشاؤها من قبل أو غير ذلك مما تطلبه أمور البرمجة وأنظر معي :

```
--This Code is For Answers
on DoAnswers
  global Answers
  Score = Answers
  Put Score--This is The Score of Answers
end
```

الأن لاحظت كيف وضعت الشرطتان بعد السطر الثالث **put Score** بحيث يكون التعليق تذكيري تماما, ويذكرك بتفاصيل ما تكتب فعليا, يمكنك ايضا أن تستخدم الشرطتان لإيقاف سطر ما أو أمر ما من التنفيذ وذلك بوضع المشيرة في نهاية السطر الذي تريده ثم الضغط على أيقونة تفعيل التوثيق, وذلك سيؤدي إلى تجاهل دايركتور للسطر أو الأسطر التي تسبقها شرطتان في بدايتها, ذلك مفيد في عملية تجربة أو تعديل البرنامج, بدلا من مسحك أو إستبدالك للأسطر أثناء عملية إبتكار برامجك الخاصة, يمكنك ملاحظة هذه الطريقة في نفس الملف **Documentry.dir** الذي قمت بالعمل عليه مؤخرا.

تم بحمد اللّٰه