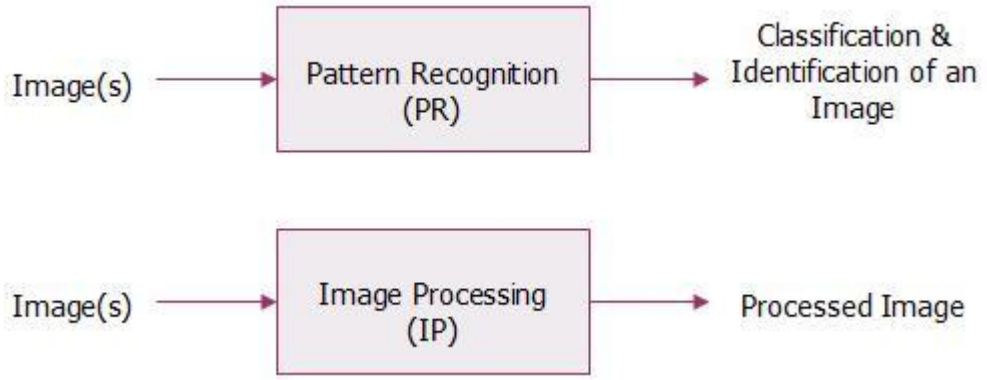
مقدمة في تمييز الأنماط ومعالجة الصور
Introduction to Pattern Recognition and Image Processing

وهو فرع من فروع الذكاء الاصطناعي، ومعنى تمييز الأنماط ومعالجة الصور يوضحهما الرسم التالي Block diagrams:



بمعنى أن أي برنامج لتمييز الأنماط يدخل له صورة فيعطي تصنيف أو تعرف للصورة، وأي برنامج للعمليات على الصور يدخل له صورة فيعطي صورة تمت بعض العمليات عليها.

نحصر حديثنا عن تمييز الأنماط والتعرف عليها ونشرح أشهر الطرق المستخدمة فيها.

لو سألنا سؤال: كيف يتعرف الإنسان أو الطفل على صورة السيارة كمثال بسيط؟! لكان الجواب البديهي هو لأنه يعلم أنها سيارة. من هنا نتفق أن أي طريقة للتعرف على الأنماط أو أي شئ في العالم لابد من أن يسبقها مرحلة تعلم لهذه الأنماط وهذه الأشياء، إذن مراحل التعرف على أي نمط مرحلتين:

- **مرحلة التعلم.learning**

- مرحلة التصنيف.classification or recognition

يوجد أربعة طرق أساسية مستخدمة في عالم تمييز الأنماط Pattern Recognition Approaches or Methods:، ألا وهي:

1. Template-Matching and Correlation Method.
2. Statical Approach.
3. Syntactic and Structural Approach.
4. Neural Networks Approach.

سنشرح كل طريقة مروراً بالمرحلتين:

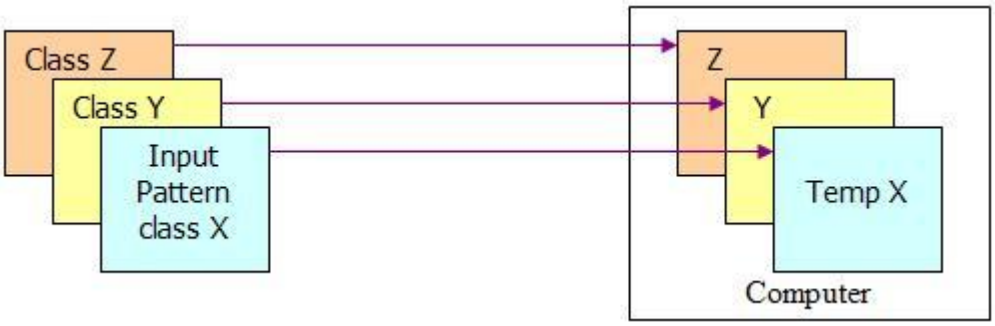
1. كيف تتم مرحلة التعليم learning فيها.
2. وأخيراً كيف تتم مرحلة تمييز أو تصنيف الأنماط فيها.

بسم الله نبدأ...

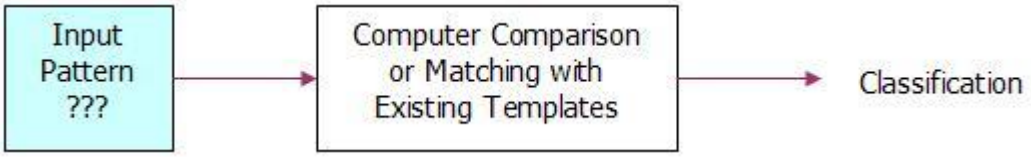
-=-=-=-=-=-=-

الطريقة الأولى Template-Matching and Correlation Method

مرحلة التعليم في هذه الطريقة تقوم على تخزين مجموعة من القوالب Templates أوالنماذج Prototypes، قالب من كل صنف في الحاسوب كما يوضح الرسم:



وفي مرحلة التصنيف تقارن الصورة الداخلة Input pattern مع templates كل صنف فإن كانت نتيجة مقارنتها مع الصنف س أكبر من نتيجة مقارنتها مع الصنف ص فإنها تصنف ضمن الصنف س وهكذا. كما يوضح الرسم:

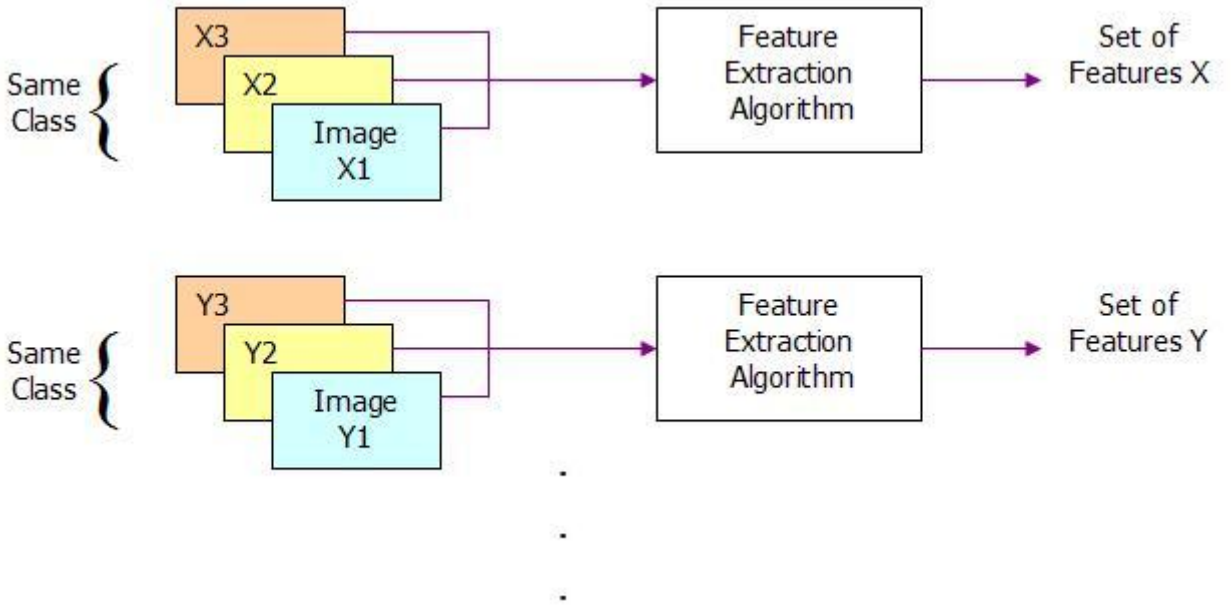


ربما تتساءلون كيف تتم عملية المقارنة، ببساطة تخزن الصورة الداخلة على شكل مصفوفة وتقارن مع القوالب الموجودة في الجهاز pixel by pixel وتعطي قيمة للمقارنة.
كما ترون تعتبر هذه الطريقة طريقة سهلة وبدائية جداً، الصعوبة الوحيدة في هذه الطريقة هي الاختيار الجيد للقوالب من كل صنف بالإضافة إلى تحديد معايير المقارنة وخصوصاً لو كانت الصورة الداخلة تحمل تشوهات!
فمثلاً لو استخدمنا هذه الطريقة للتعرف على المجرميين، لابد أن نأخذ لكل مجرم عدة لقطات كي تخزن على جهاز الحاسوب: لقطتان جانبيتان واحدة من كل جهة، لقطة أمامية، ولقطتان بزاوية نظر 45 درجة عن الكاميرا. ولكم أن تتخيلوا المساحات التخزينية اللازمة لكل هذه القوالب!

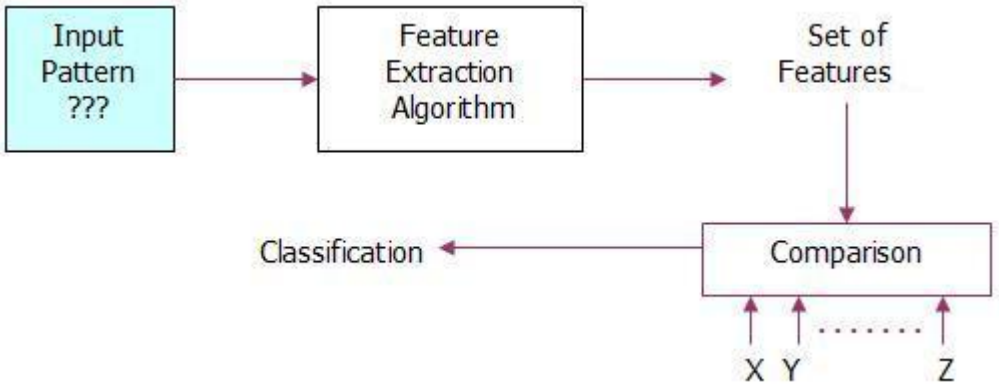
-=-=-=-=-=-=-

الطريقة الثانيةStatical Approach

في هذه الطريقة، يوصّف كل pattern بواسطة مجموعة من الخصائص set of features والتي من الممكن أن نعبر عنها بقيم حقيقية. في مرحلة التعلم: يقدّم كل نمط patternكمتجه من الخصائص feature vector كما توضح الصورة:



أما في مرحلة التعرف او التمييز أو التصنيف، فهذه عادة تتم عن طريق تقسيم مساحة الصورة إلى مناطق مجزأة، كل منطقة تقارن مع صنف كما توضح الصورة:



فمثلاً لو كنّا نريد التعرف على صورة تفاحة، ماهي خصائص التفاحة التي نخزنها في مرحلة التعلم؟! هي على سبيل المثال: اللون، الشكل، الدوران، المنطقة السفى، المنطقة العليا.... ألخ. وكذلك يتم التعرف على التفاحة، تقسم الصورة إلى أجزاء وكل جزأ نقارن الخصائص الموجودة فيه مع خصائص الصنف المخزنة وهكذا.
الصعوبة هنا هي في اختيار مجموعة الخصائص لكل فئة وقواعد القرار في التعرف على النمط.
بهذا نكون انتهينا من طريقتين في الدرس الأول، تابعنا في الدرس الثاني لتتعرف على بقية الطرق :)

تابع مقدمة في تمييز الأنماط ومعالجة الصور

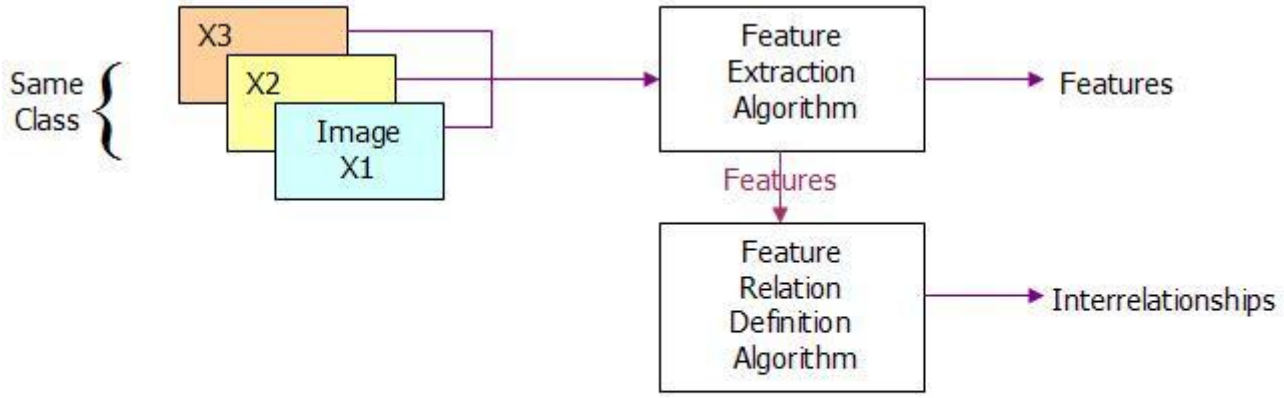
نستكمل حديثنا اليوم عن الطريقتين المتبقيتين:

الطريقة الثالثةSyntactic and Structural Approach

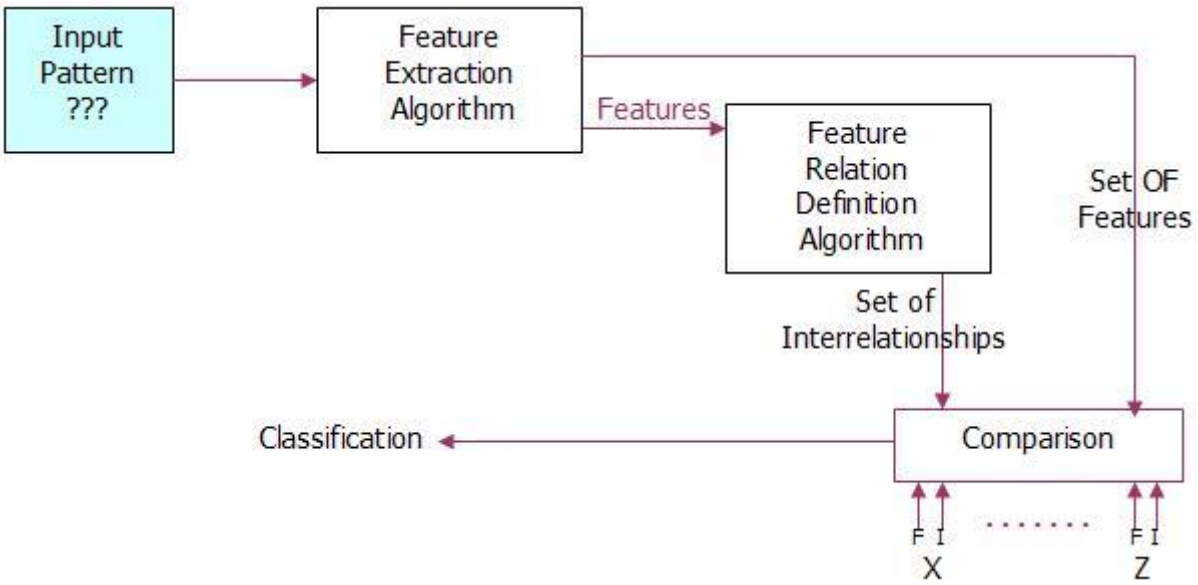
في هذه الطريقة لا نكتفي فقط بالقيم الرقمية لخصائص كل صنف، ولكن نضيف عليها العلاقات البينية Interrelationships or Interconnection of Features بين هذه الخصائص في كل صنف والتي تتيح لنا معلومات هيكلية ضرورية في التعرف على الأنماط!

آخر الدراسات في هذا المجال توصلت إلى أن أقوى طريقة للتعرف على الأنماط هي الطريقة التي تجمع بين Statistic pattern recognition approach مع Syntacticكطريقة واحدة تسمى Syntactic-pattern recognition Semantic approach.

في مرحلة التعلم في هذه الطريقة يمثل النمط عادة كشجرة tree أو رسم بياني graph او سلسلة حرفية string من العناصر الأولية primitives والعلاقات بينها relations كما توضح الصورة:



وعملية اتخاذ القرار في مرحلة التعرف أو التصنيف هي عبارة عن عملية تحليل Syntax analysis أو بمعنى آخر برنامج تعريب .parsing procedure وأعلى نسبة مقارنة ناتجة من مقارنة الصورة المدخلة مع كل شجرة) tree أو graph أو string على حسب التمثيل المعتمد في التطبيق) مخزنة تحدد الصنف الذي تنتمي إليه الصورة المدخلة! الشكل التالي يوضح عملية التصنيف في هذه الطريقة:



لنأخذ مثال:
صورة مدخلة يوجد بها دائرتين (خصائص) لو كانت المسافة بينهما ما بين 1 إلى 2 سم فمن الممكن أن تصنف الصورة على أنها صورة نظارة مع الأخذ بالاعتبار الخصائص الأخرى وعلاقتها فيما بينها، اما لو كانت المسافة بينهم متر تقريباً فمن الممكن أن تصنف على أنها إنوار سيارة مع الأخذ بعين الاعتبار الخصائص الأخرى طبعاً... وهكذا

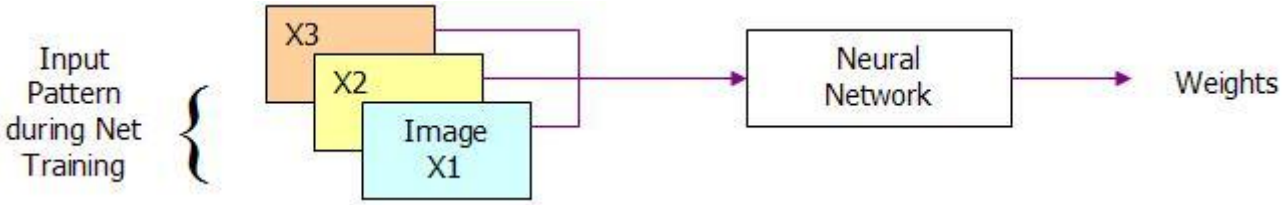
تستخدم هذه الطريقة في التعرف على الأهداف أو الصواريخ target recognition وكذلك في التعرف على الأحرف character recognition وغيرها.

-=-=-=-=-=-=-

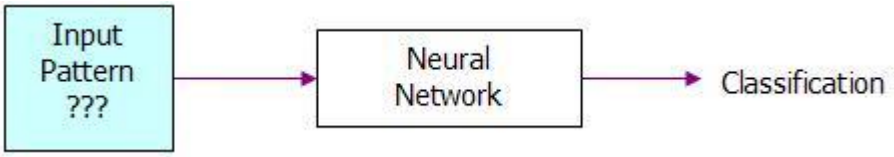
الطريقة الرابعةNeural Networks Approach

الشبكات العصبية علم قائم بحد ذاته اهتم به العلماء لسنوات عديدة بهدف الوصول إلى طريقة أشبه ما تكون بطريقة الإنسان في التعرف على الأنماط. ولا يسعنا شرح مفاهيم العلم في هذه الوهلة ولا بسلسلة دروس متكاملة*، ولكن باختصار يقوم على استخدام المعالجة المتوازية للبيانات في وقت واحد، هذه المعالجة تتم في معالجات او وحدات units أو طرفيات- nodesوكلاهم بمعنى واحد- تتصل ببعضها البعض عن طريق وصلات

ذات أوزان weights والتي ضبطت أثناء عملية تدريب الشبكة. وفي مجال تمييز الأنماط مجموعة من الصور patternsتدخل إلى الشبكة العصبية فتقوم الشبكة العصبية بضبط أوزانها طبقاً لميكانيزم معين وعمليات طويلة:



بعد ذلك وفي مرحلة التصنيف يقدم للشبكة pattern وبناء على الأوزان فيها تقوم بتصنيف هذا النمط:



-=-=-=-=-=-=-

بهذا نكون انهينا الحديث عن الموضوع الذي خصصناه لحديثنا، أصل هذين الدرسين كان سؤال للنقاش حول كيفية التعرف على بصمة الإصبع في ساحات الموسوعة تجده هنـــا.

مقدمة عن الذكاء الصناعيArtificial Intelligence

في محاولة لتوضيح مدى أهمية دراسة علوم الذكاء الاصطناعي، وفي محاولة للتفكير حول ماهية هذا العلم، ستفيدك عزيزي القارئ هذه المقدمة إن شاء الله.

البشر وحدهم من تطلق عليهم صفة العقل، لأن قدراتنا العقلية هامة في كل صغيرة وكبيرة في حياتنا تمام أهميتها لنا أنفسنا. مجال الذكاء الاصطناعي يعني بميكنة الذكاء الإنساني ودراسة قدراته العقلية، فمن أهم الأسباب لدراسة الذكاء الاصطناعي هو محاولة فهمنا لعمليات العقل البشري، عقلي وعقلك وعقل كل قارئ كريم بطريقة تبتعد عن علم الفلسفة وعلم النفس وعلم التشريح والتي تعنى بدورها أيضاً بالعقل البشري. فعلم الذكاء الاصطناعي يكافح لبناء الذكاء بالقدر الذي يعنى

فيه بفهم هذا الذكاء. السبب الثاني لدراسة هذا العلم هو أن برنامجنا الذكي مفيد بحد ذاته وفعّال في عدة مجالات في حياتنا التي أصبحت رقمية! فمع أن لا أحد يستطيع أن يتنبأ بتفاصيل المستقبل، إلا أنه من الواضح أن الحاسوب مع الذكاء الإنساني سيكون له تأثير ضخم وواضح في حياتنا اليومية وفي صناعة الحضارة .

الذكاء الاصطناعي يعتبر لغز مهم: كيف من الممكن لهذا الدماغ الصغير، سواء كان بيولوجيّاً أو إلكترونياً، أن يفهم ويدرك ويتنبأ ويتفاعل مع عالم أكبر وأعقد من الدماغ نفسه؟ كيف لنا أن نسلك طريق يعنى بصناعة مثل هذا الدماغ الصغير بكل صفاته المعقدة؟ هذا سؤال صعب، ولكن بخلاف البحث عن وسيلة مواصلات أسرع من سرعة الضوء فإن الباحث في علم الذكاء الاصطناعي والدارس له يجد أن هذا العلم قائم على أسس متينة وممكنة، كل ما عليه هو النظر إلى المرآة ليجد مثالاً حيّا عن النظام الذكي.

الذكاء الاصطناعي علم معرفي حديث، بدأ رسمياً في الخمسينات من القرن الماضي. أما قبل هذه الفترة، فنجد أن عدد من العلوم الآخرى عنيت بشكل أو بآخر بالذكاء الاصطناعي وبطريقة غير مباشرة. باستعراض علم الوراثة؛ نجد مايرتبط بالذكاء في حقل دراسة جينات العلماء في محاولة لإعزاء ذكاءهم للوراثة. في الفيزياء نجد أن جميع الطلاب بلا شك يشعروا بأن جميع الأفكار الجيدة أخذت من غاليليو وآينشتاين ونيوتن وبقية العلماء، ولابد من الدراسة لأعوام عديدة حتى يتسنى لأحدهم تقديم اكتشاف جديد!. في المقابل فإن الذكاء الاصطناعي لا يزال مفتوحاً ليشغِل بدراسته آينشتاين جديد جميع أوقاته.
البحث عن ماهية الذكاء كذلك شغلت الفلاسفة قبل أكثر من ألفي عام، فقد حاولوا فهم كيف تتم رؤية الأشياء، وكيف يتم التعلم، والتذكر والتعليل. ومع حلول استخدام الكمبيوتر في الخمسينات تحولت هذه البحوث إلى أنظمة تجريبية واقعية.

حالياً، فإن للذكاء الاصطناعي تطبيقات عديدة، سواء كانت تطبيقات ذات أغراض عامة مثل الإدراك والتعليل المنطقي، أو كانت مهمات ذات غرض خاص مثل لعب الشطرنج أو التشخيص الطبي! غالباً فإن الخبراء والعلماء يتوجهون إلى الذكاء الاصطناعي لحفظ خبراتهم وتجاربهم التي قضوا بها حياتهم. فالذكاء الاصطناعي مجال عالمي يصلح لجميع التوجهات.

ما هو الذكاء الاصطناعي؟

الذكاء الاصطناعي علم مثير ولا نملك تفسير لهذه الإثارة، كما لا نملك تعريف دقيق له لأننا في الأصل لا نملك تعريف دقيق للذكاء!
لكن يمكننا أن نقول أنه الطريق الوحيد لإنتاج برامج ذكية. أما التعريفات التي وردت في الكتب الورقية المختصة بعلم الذكاء الاصطناعي فهي كثيرة، نستطيع أن نجملها في هذا التعريف الشامل والذي أخذناه من قاموس الموسوعة العربية للكمبيوتر والإنترنت:

الذكاء الاصطناعي: اختصاره .AI مصطلح يطلق على علم من أحدث علوم الحاسب الآلي، وينتمي هذا العلم الى الجيل الحديث من أجيال الحاسب الالي ويهدف إلى أن يقوم الحاسب بمحاكاة عمليات الذكاء التي تتم داخل العقل البشري، بحيث تصبح لدى الحاسوب المقدرة على حل المشكلات واتخاذ القرارات باسلوب منطقي ومرتب وبنفس طريقة تفكير العقل البشري.

هذه العمليات تتضمن:

- **التعليم: اكتساب المعلومات والقواعد التي تستخدم هذه المعلومات.**
- **التعليل: استخدام القواعد السابقة للوصول إلى استنتاجات تقريبيه أو ثابتة.**
- **التصحيح التلقائي أو الذاتي.**

باختصار: هو فرع من فروع علوم الحاسوب يُعنى بميكنة السلوك الذكي عند الإنسان. وفيه نحتاج إلى:

- نظام بيانات :يستخدم لتمثيل المعلومات والمعرفة.
- خوارزميات :نحتاج إليها لرسم طريقة استخدام هذه المعلومات.
- لغة برمجة :تستخدم لتمثيل كلاً من المعلومات والخوارميات.

تطبيقات علم الذكاء الاصطناعي:

تطبيقات الذكاء الاصطناعي كثيرة جداً من أكثرها شيوعاً:

1. تطبيقات الألعاب Game Playing.
2. تطبيقات ميكنة التعليل وإثبات النظريات Automated Reasoning & Theorem Proving.
3. تطبيقات الأنظمة الخبيرة Expert Systems .
4. تطبيقات التعرف على الصوت Natural Language Understanding & Semantic Modeling ومنها Natural Language Processing.
5. تطبيقات الرؤية عن طريق الآلة Machine Vision.
6. صياغة أداء الانسان Modeling Human Performance.
7. التخطيط و الاتمتة (كالإنسان الآلي) Planning & Robotics.
8. لغات و بيئات للذكاء الاصطناعي Languages & Environments for AI.
9. تعليم الالآت Machine Learning.
10. الحوسبة الظاهرة و المعالجة الموزعة المتوازية Parallel Distributed Processing (PDP) & Emergent Computation.
11. التصنيف الارشادي Heuristic Classification.
12. الفلسفة و الذكاء الاصطناعي AI & Philosophy.

فمثلاً: عند استخدام هذا العلم لتطوير الانظمة الحديثة يتم تخزين الملايين من المعلومات داخل الحاسب لتكوين قاعدة بيانات رئيسية له مثل ما تخزن المعلومات داخل العقل البشري من خلال التعلم والخبرات اليومية التي

يكتسبها.

ثم يتم بعد ذلك تطوير برامج خاصة، ليستطيع الحاسب استخدامها في التعامل مع هذه البيانات واستخدامها بطريقة منطقية في حل المشكلات اللازمة لصنع القرار. وقد نجح العلماء حتى الآن في تطوير بعض النماذج الصغيرة من نظم الذكاء الاصطناعي، ومنها اجهزة الروبوت والحاسبات الشخصية التي تستطيع اجراء الحوار مع الانسان وتنفيذ أوامره الصوتية. ولكن مازالت هذه النماذج تحت التطوير والتجربة ويتم تحديثها يوما بعد يوم.

فروع علم الذكاء الاصطناعي:

- منطق الذكاء الاصطناعي logical AI.
- البحث search.
- التمييز النمطي و النموذجي pattern recognition.
- التمثيل representation.
- الاستدلال والاستنتاج inference.
- التعليل common sense knowledge and reasoning.
- التعلم بالخبرة learning from experience.
- التخطيط planning.
- نظرية المعرفة epistemology.
- علم الوجود ontology.
- الارشاد heuristics.
- البرمجة الوراثية genetic programming.

لغات البرمجة المستخدمة لإنتاج برامج الذكاء الاصطناعي:

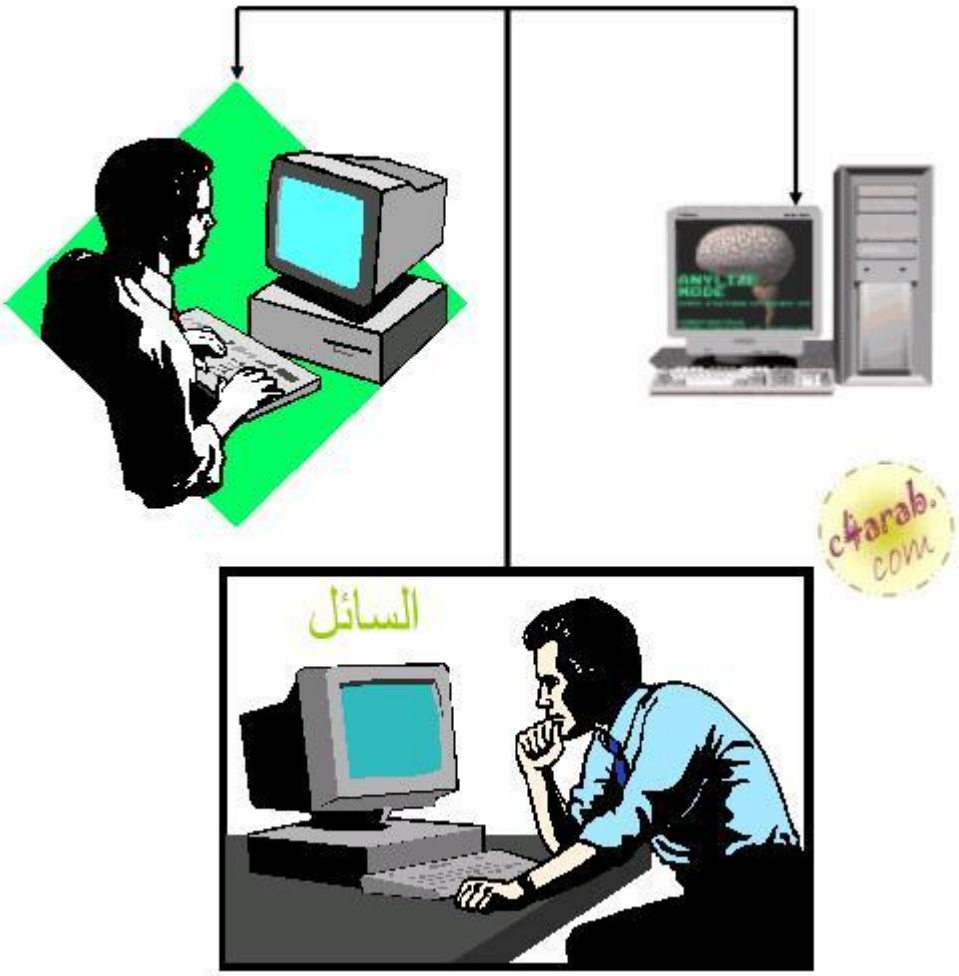
- Lisp .
- Python.
- Prolog.
- Java.
- C++ .

وبما ان 90% من برامج الذكاء الاصطناعي تمت برمجتها باستخدام لغة الـ Lisp, فإن شاء الله ستخصص الموسوعة العربية للكمبيوتر والإنترنت العدد الأكبر من الدروس القادمة لتكون عبارة عن مدخل لكيفية برمجة برامج الذكاء الاصطناعي بواسطة لغة الـ Lispلما تحتويه هذه اللغة من مميزات وما توفره من دوال لإنتاج برامج ذكية. كما ستجد قسماً آخر للبرولوج.

مقياس البرامج الذكية[(Turing Test)]

ماهو اختبار تيورينجTuring Test ؟

في علوم الذكاء الاصطناعي ،اختبار تيورينج هو طريقة لتحديد ما إذا كان الحاسوب (البرنامج الذكي) قادر على التفكير مثل الإنسان أم لا؟! أي أنه طريقة لتحديد ما إذا كان البرنامج ذكياً ام لا؟!
اسم هذا الاختبار يعود إلى مخترعه عالم الرياضيات الإنجليزي "آلن تيورينج" والذي اخترع هذا الاختبار في الخمسينات من القرن الماضي. خلال هذا النوع من الاختبار، نعتبر أن لدى الحاسوب ذكاء اصطناعي إن استطاع تقليد رد فعل الإنسان في ظروف محددة .



توضح الصورة السابقة مكونات اختبار تيورينج، حيث يتكون هذا الاختبار في الأساس من ثلاث أجهزة حاسوب كل جهاز معزول عزل تام عن الجهازين الآخرين، إثنان

منهم تشغل بواسطة بشر لديهم خبرة في مجال البرنامج الذكي الذي نقوم باختباره، والجهاز الثالث يعمل بواسطة البرنامج الذكي، الإنسان الذي يعمل على الجهاز الأول يعتبر السائل، في حين يعتبر كل من الإنسان الذي يعمل على الحاسوب الثاني، والحاسوب الثالث (الذي يشغل بواسطة البرنامج الذكي).. يعتبران المستجوبين، يقوم السائل باستجواب كلاً من الإنسان الآخر والحاسوب بصيغة معينة وخلال فترة زمنية محددة عدّة مرات، ثم بعد أن يحصل على الإجابات منهما يحاول أن يحدد مصدر كل إجابة هل هي من الإنسان أم من الحاسوب؟ وإذا لم يستطع السائل تحديد ما إذا كانت الإجابة الذي حصل عليها مقدمه من الحاسوب أو إنسان آخر؛ يعتبر الحاسوب (البرنامج الذكي) قد اجتاز هذا الامتحان، وحينها يصح ان ننعت هذا البرنامج بالذكاء لأنه حاكى وماثل طريقة تفكير الإنسان وأعطى نفس إجاباته.

سلبيات الاختبار:

1. **أبرز سلبية لهذا الإختبار تكمن في أن طبيعة الأسئلة التي يقدمها السائل قد تساعد في تحديد مصدر الإجابة، فنحن نعلم أن هناك نوعين من الأسئلة:**
   - ***الأسئلة الرمزية Symbolic:*** **والتي يتم فيها توصيف المشكلة رمزياً كأن نقول: مع أحمد مائتان وخمسون ريال أنفق سدسها في المكتبة فكم بقي معه؟!**
   - ***الأسئلة الحسابية Numerical:*** **والتي يتم فيها ذكر قيم محددة وتحل عن طريق معادلات رياضية واضحة.**

هنا، نجد أن الكمبيوتر أسرع وأدق من الإنسان في النوع الثاني من الأسئلة (العمليات الرياضية) في حين ان الأنسان يتفاعل مع الأسئلة الرمزية بطريقة أكثر فاعلية!!
كما أن الأنسان يستطيع تفسير الاستعارات اللفظية والفن والأدب أفضل من الكمبيوتر، فكلمة " قمر " تعني للكمبيوتر ذلك الكوكب الصغير التابع لأي

كوكب آخر في المجموعة الشمسية، بينما قد يكون لهذه الكلمة أكثر من معنى عند الإنسان. كذلك لا يتوقع من الحاسوب أن يحاكي مشاعر وأحاسيس الإنسان أبداً.

2. **كذلك، يفترض أن هذا البرنامج يحاكي سلوك الإنسان، والإنسان بطبعه يمّل في أوقات كما يصل إلى مرحلة شلل تفكيري في بعض الأوقات الأخرى - عند المرض مثلاً - ويفترض أن البرنامج يحاكي هذه الحالات وهذا يعتبر عيب طفيف!**

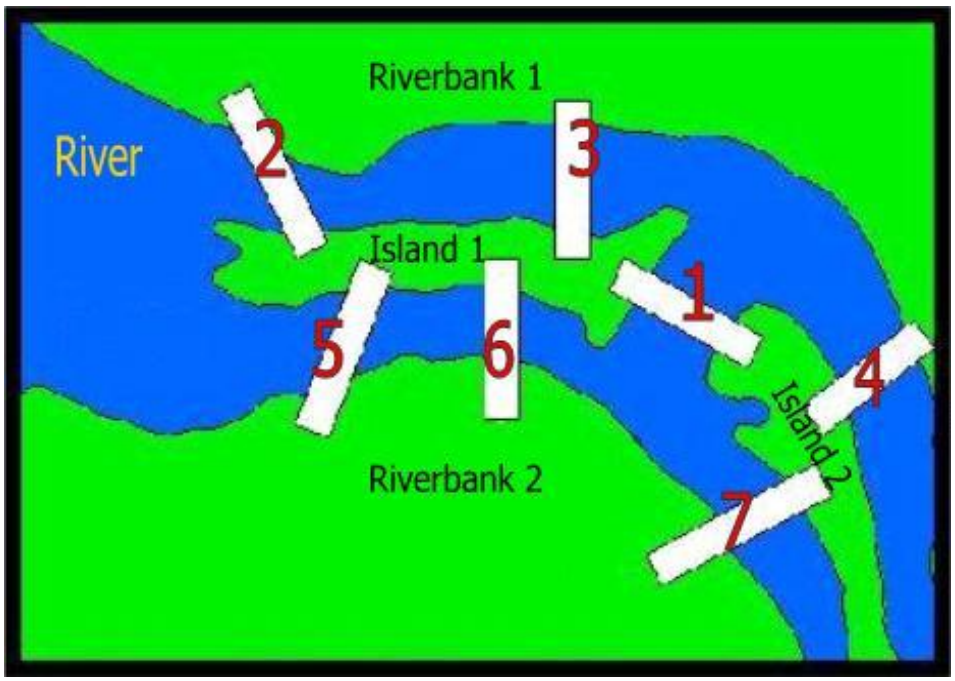
ايجابيات الاختبار:

1. **إيجابيات هذا الاختبار كثيرة جداً أهمها أنه يتفادى الجدل القائم حول ماهيّة الذكاء نفسه؛ فهذا الاختبار ينتج لنا برنامجاً ذكياً في حين أننا لم نتمكن من التوصل إلى تعريف شامل ومتكامل للذكاء عالمياً !!**
2. **كذلك، في هذا الاختبار لايشترط أن تكون العمليات الذهنية التي تمت في عقل الإنسان هي نفسها العمليات التي تمت داخل الحاسوب!! وهذه نقطة إيجابية كبيرة في صناعة البرامج الذكية.**
3. **لا يوجد فيه تحيّز للحاسوب أو الإنسان حيث أنه يتم بعزل تام كما وضحنا.**

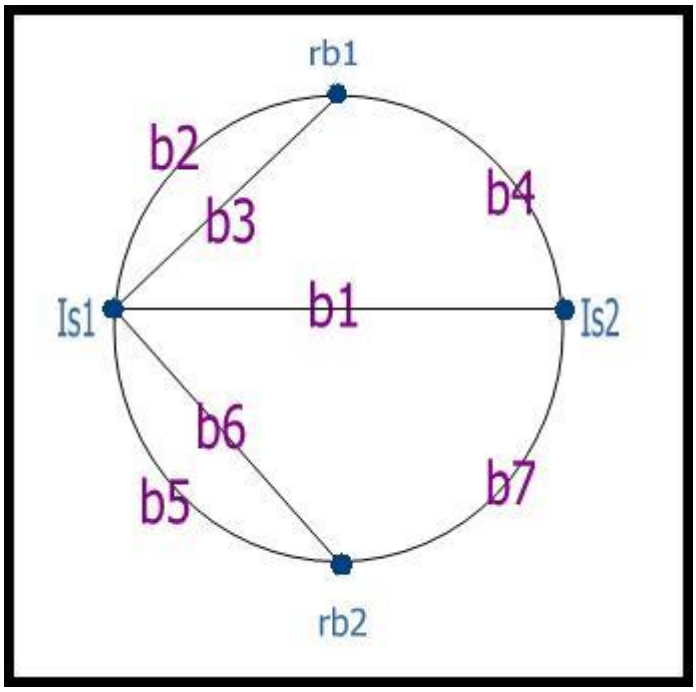
في النهاية يعتبر هذا الاختبار أفضل وأقوى طريقة لاختبار البرامج الذكية وتحديد درجة ذكاءها من الخمسينات وحتى اليوم ..

نظرية التخطيط البيانيGraph Theory

سنحكي اليوم قصة عالم رياضيات نمساوي اسمه Leonhard Eular كان السبب بعد إرادة الله سبحانه، في اختراع نظرية اسمها Graph Theory في أوائل القرن الثامن عشر، اخترع هذه النظرية لإيجاد حل للمشكلة التي واجهته حينما زار مدينة Königsberg في ألمانيا، هذه المدينة يخترقها نهر river عليه جزيرتين صغيرتين Island1 & Island2. ترتبط هاتان الجزيرتان بضفتي النهر Riverbank1 & Riverbank2 وببعضهما البعض عن طريق سبعة من الجسور Bridges، كما توضح الصورة التالية:



أراد هذا العالم التجول في إرجاء هذه المدينة بكاملها بدون المرور على جسر أكثر من مرة!
لدراسة إمكانية ذلك؛ قام العالم برسم خريطة توضيحية بسيطة للمدينة، مثّل فيها الجهات التي يريد التنقل فيما بينها (كلاً من Is1, Is2, rb1 and rb2) كنقاط أو أطراف *nodes*، ثم مثّل فيها كل جسر كوصلة *link* تربط بين هذه الأطراف كما هي موجودة في أرض الواقع، هذا التمثيل سمِّيَ *Graph* أو تمثيل بياني:
-حيث أن Is= Island, rb= Riverbank and b=bridge



بعد ذلك أوجد مفهوم جديد يسمى *Degree* of the Node of the Graphأو درجة الطرف في التخطيط، حيث أن لكل طرف درجة هذه الدرجة هي عدد الوصلات التي تصل هذا الطرف مع الأطراف الأخرى المجاورة له، أي عدد الـ links الداخلة أو الخارجة من هذا الطرف، من الممكن أن يكون هذا العدد فردي أو زوجي بطبيعة الحال.
لنوجد معاً درجة كل طرف في التخطيط:

Node	Degree
Is1	5
Is2	3
rb1	3

توصل بعد ذلك إلى النظرية التي تقول:
يمكنك حل المشكلة والمشي في أرجاء المدينة مع العبور على كل جسر مرة واحدة فقط في حالتين:

1. إذا كان لديك طرفين فقط يحملون درجة فردية two odd degree nodes.
2. إذا لم يكن لديك ولا طرف من درجة فردية zero odd degree nodeبمعنى أن جميع الأطراف لديك من درجة زوجية.

عدا ذلك فإن المشكلة مستحيلة الحل ولا يمكنك المشي في أرجاء المدينة دون العبور على أحد الجسور أكثر من مرة!

ثم حدد مسار السير في الحالات التي يمكن حل المشكلة فيها، كالتالي:

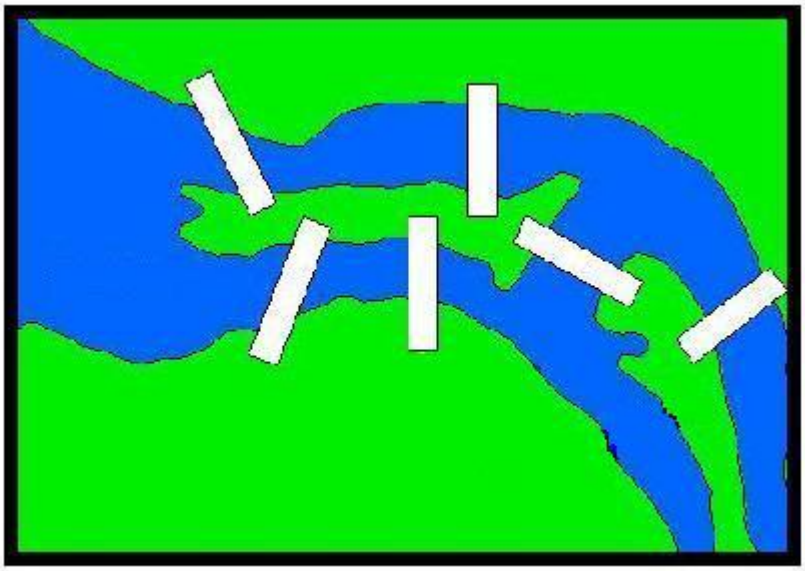
- إذا كان لديك طرفين من درجة فردية، فإن السير سيبدأ من الطرف ذو الدرجة الفردية الأول، وينتهي عند الطرف ذو الدرجة الفردية الثاني.
- إذا لم يكن لديك ولا طرف من درجة فردية، بمعنى أن كل الأطراف من درجة زوجية، فإن المشي سيبدأ من أحد هذه الأطراف وينتهي عند نفس الطرف!

ثم أقرّ بأنه لا يمكنه التجوال في أرجاء مدينة Königsberg بدون العبور على جسر أكثر من مرة، لأنه يوجد أربعة أطراف من درجة فردية في graph هذه المدينة!!

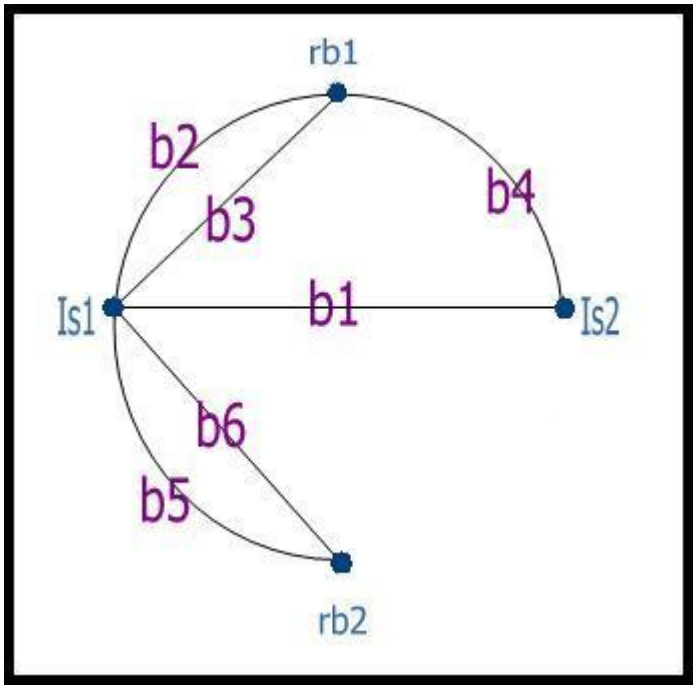
-=-=-=-=-=-=-

عرفت هذه المشكلة باسم "Bridge of Königsberg problem"ومؤخراً أصبحت معروفة باسم العالم:
"Finding an *Eular* path through a graph"، "وهي تعتبر حجر الأساس وأول نظرية في عالم الـGraph.
وهذا يقودنا للسؤال: ما معنة كلمةGraph ؟!
الـ Graphكما رأينا هو مجموعة من الأطراف nodes تربط ما بينهما مجموعة من الوصلات، links من الممكن أن

نعتبر كل طرف يمثّل حالة، وللإنتقال من حالة إلى أخرى نستخدم الوصلة التي تصل بينهما.
يفيد تمثيل المشاكل بهذه الطريقة في اختزال واحتواء المشكلة، وزيادة فهمها مما يسهل الطريق إلى حلها، كما تعتبر Graph Theoryأفضل أداة للتعليل reasoning في أي تركيب structure يحوي مجموعة من الكائنات objects بينهما مجموعة من العلاقات.relations
في علوم الذكاء الاصطناعي، تستخدم هذه النظرية في تقنيات البحث وخصوصاً في State-space search بنوعيها Depth-first and Breadth-first.
والآن بعد أن انتهينا من سرد القصة وتعرفنا على النظرية كاملة، ما رأيكم بمثال آخر نطبّق النظرية عليه ونرى هل يمكننا حل مشكلته أم لا؟!
إليكم هذه الخريطة:



أول خطوة، نمثلها كـ:Graph



ثم نوجد درجة كل node فيها:

Node	Degree
Is1	5
Is2	2
rb1	3
rb2	2

ثم نحدد عدد الأطراف من الدرجة الفردية، يوجد لدينا طرفين من درجة فردية.
إذن المشكلة ممكنة الحل والمشي بدون العبور أكثر من مرة على كل جسر ممكن.
لنحدد معاً مسار المشي الذي لابد من أن يبدأ من أحد الأطراف ذات الدرجة الفردية (Is1 or rb1) وينتهي عند الطرف الآخر، من الممكن أن يكون أحد المسارات التالية:

1. **Is1(through b5) --> rb2(b6) --> Is1(b2) --> rb1(b3) --> Is1(b1) --> Is2(b4) --> rb1**
2. **Is1(b1) --> Is2(b4) --> rb1(b3) --> Is1(b5) --> rb2(b6)--> Is1(b2) --> rb1**
3. **Is1(b2) --> rb1(b3) --> Is1(b5) --> rb2(b6) --> Is1(b1) --> Is2(b4) --> rb1**

4. -- rb1(b2) --> Is1(b5) --> rb2(b6) --> Is1(b3) --> rb1(b4) --> Is2(b1) --> Is1

5. -- rb1(b4) --> Is2(b1) --> Is1(b5) --> rb2(b6) --> Is1(b2) --> rb1(b3) --> Is1

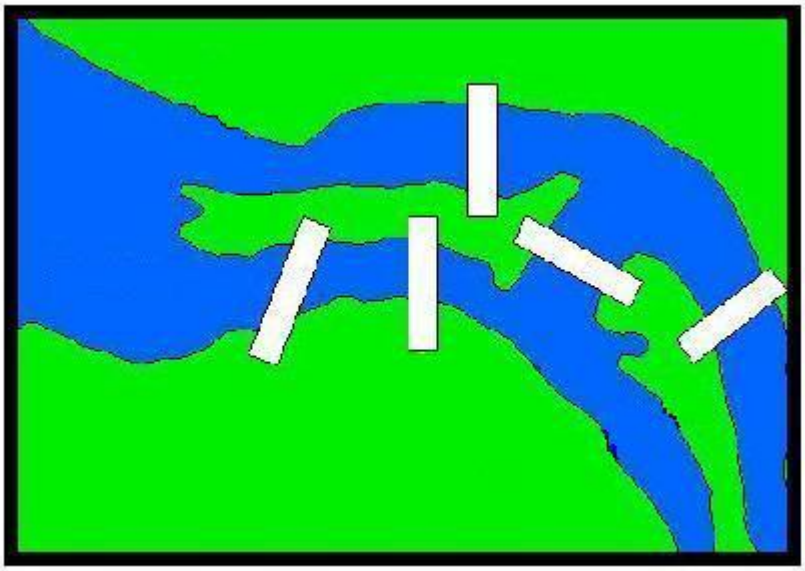
.

.

. وهكذا :)

-=-=-=-=-=-=-

إليك مثال آخر:



يبدو أن مشكلته ممكنة الحل، هل يمكنك حلها؟! نعم أنا واثقة من ذلك، طبّق الدرس عليها مباشرة وأوجد المسارات، ستجد أن كل مسار يبدأ وينتهي في نفس الطرف node.

-=-=-=-=-=-=-

أرجو عزيزي القارئ أن يكون هذا الدرس قد أعطاك طريقة ذكية لحل مشكلات التنقل أثناء النظر إلى أي خريطة في حياتك اليومية، كما أرجو أن يكون قد وضع أقدامك على نقطة البداية في دراسة أساليب البحث في برامج الذكاء الاصطناعي.. والسلام عليكم :)

الشبكات العصبية ((Neural Network

كيف يتعلم الإنسان؟!

تنتشر في جسم الإنسان ملايين الخلايا العصبية والتي تتفرع بدورها إلى الملايين من الزوائد العصبية، حيث تنقل هذه الخلايا العصبية الإحساس و ردّات الفعل من و إلى العقل البشري بواسطة الحبل الشوكي.

ومن خلال هذه الخلايا العصبية يتم تخزين المعرفة عن العالم الخارجي في العقل البشري، وذلك عن طريق ضبط الأوزان داخل هذه الخلايا.

لو أخذنا مثال بسيط و يحدث دائماً دون أن نشعر بذلك في حياتنا، وهو تعلّم الطفل للتعرف على صور الحيوانات في السنوات الأولى من عمره.

فمثلاً لو عرضنا على طفل في الثالثة من عمره صورة لبقرة ثم عرضنا صورة لقط ثم صورة لدجاجة مع ذكر اسم كل حيوان أمامه. و كررنا هذه الصور لعدة مرات. بعد ذلك تأتي مرحلة الاختبار ويتم فيها عرض الصور السابقة مع صور حيوانات أخرى لنقل صورة عصفور بحيث يطلب منه معرفة اسم الحيوان الظاهر في الصورة، نلاحظ أن الطفل سيتعرف على صور الحيوانات التي تعلمها أثناء مرحلة التعليم ولكن عند عرض صورة العصفور فإن الطفل سيتعرف على الصورة على أساس أنها صورة الدجاجة!

وذلك لأن صورة العصفور مشابهة في كثير من الخصائص الخارجية لصورة الدجاجة والتي تم تخزينها في عقله. ولكن مع تنويع الصور وتكرارها سيتعلم الطفل أكثر في كل مرة.

فكّر العلماء في طريقة يستطيعون من خلالها محاكاة هذه العملية التي تحدث في العقل البشري، وتوصلوا إلى علم الشبكات العصبية Neural Network والذي يندرج تحت علوم الذكاء الصناعي، بحيث يجعلون من أجهزة الحاسوب أجهزة ذكية، بإمكانها أن تكتسب المعرفة بنفس الطريقة التي يكتسب بها الإنسان المعرفة، وهي طريقة ضبط الأوزان أثناء التعلم.

الشبكات العصبية الاصطناعية:

هو جهاز مصمم لمحاكاة الطريقة التي يؤدي بها العقل البشري مهمة معينة، وهو عبارة عن معالج ضخم موزع على التوازي، ومكون من وحدات معالجة بسيطة، بحيث يقوم بتخزين المعرفة العملية ليجعلها متاحة للمستخدم وذلك عن طريق ضبط الأوزان.

طرق المحاكاة:

هناك عدة طرق لمحاكاة الطريقة التي يتعلم بها الإنسان وهي:

- عن طريق الورقة والقلم، بحيث يتم إدخال الخصائص كمدخلات ثم القيام بعمليات حسابية معينة تضبط فيها الأوزان لتعطي النتيجة المرغوبة وهذه الطريقة غير عملية وتستخدم عادة لتوضيح المفهوم الذي تعمل به الشبكة العصبية فقط.
- عن طريق عدة أشخاص مع كل شخص منهم آلة حسابية بسيطة، بحيث يمثل كل شخص منهم خلية عصبية تقوم بعملية ضبط الأوزان. وهذه الطريقة غير فعالة لنفس الأسباب السابقة.
- عن طريق عدد كبير جداً من أجهز الحاسوب المتصلة ببعضها البعض، بحيث يمثل كل جهاز منها خلية عصبية تقوم بعمليات حسابية بسيطة لضبط الأوزان، وهذه الطريقة غير فعالة نظراً للعدد الكبير جداً من الأجهزة والتي تكون في الغالب مكلفة جداً.
- عن طريق برنامج يحاكي هذه العملية، وهذه هي الطريقة الأمثل و الأسهل والأقل تكلفة علاوة على

كونها الأكثر انتشارا، وهي التي سنعتمدها إن شاء الله في هذا الدرس لبناء شبكة عصبية.

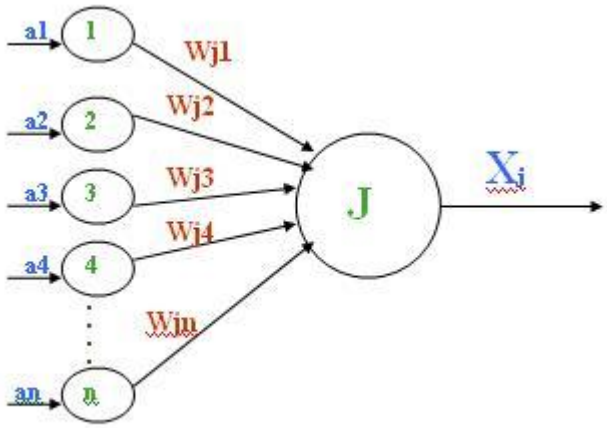
التطبيقات التي تستخدم الشبكات العصبية:

الشبكات العصبية أعطت حلولاً ذات كفاءة عالية للكثير من التطبيقات في العديد من المجالات نذكر منها:

- تمييز الأنماط والتعرف على الصور.
- القدرة على التعرف على الصور المشوهة.
- إكمال الصور التي فقدت جزء منها، مثل الصور المرسلة بواسطة الأقمار الصناعية.
- عمليات التصنيف إلى عدد من الفئات. مثل تصنيف الحيوانات إلى أليفة و مفترسة.

لو أخذنا مثلاً على عملية التعرف على الكائنات الحية، و قمنا ببرمجة برنامج بالطريقة التقليدية للتعرف على هذه الحيوانات فإن ذلك سيكون صعب للغاية فضلاً عن كونه محدود القدرات. فالتطبيقات التقليدية تمر بعدة مراحل تتطلب في معضمها وجود الإنسان، و تتطلب برنامج ضخم للتعرف على كل حيوان على حدة!
بينما في الشبكات العصبية فإنه الشبكة تتبع نفس الطريقة التي يتعلم بها الإنسان وذلك عن طريق عرض صور الحيوانات وضبط الأوزان حتى يتم تخزين المعرفة بصورة صحيحة في ذاكرة الحاسوب ومع تكرار الصور و تنوعها تتعلم الشبكة وتصبح قادرة على إعطاء إجابات صحيحة، وكل ذلك لا يتطلب كتابة برنامج ضخم كما في التطبيقات التقليدية.

مكونات الشبكة العصبية:



كما أن للإنسان وحدات إدخال توصله بالعالم الخارجي وهي حواسه الخمس، فكذلك الشبكات العصبية تحتاج لوحدات إدخال، ووحدات معالجة يتم فيها عمليات حسابية تضبط بها الأوزان و تحصل من خلالها على ردة الفعل المناسبة لكل مدخلٍ من المدخلات للشبكة.
فوحدات الإدخال تكوّن طبقة تسمى طبقة المدخلات، و وحدات المعالجة تكوّن طبقة المعالجة وهي التي تخرج نواتج الشبكة. وبين كل طبقة من هذه الطبقات هناك طبقة من الوصلات البينية التي تربط كل طبقة بالطبقة التي تليها والتي يتم ضبط الأوزان الخاصة بكل وصلة بينية، وتحتوي الشبكة على طبقة واحدة فقط من وحدات الإدخال ولكنها قد تحتوي على أكثر من طبقة من طبقات المعالجة.

طرق تعليم شبكة عصبية:

تتعلم الشبكة عن طريق إعطائها مجموعة من الأمثلة، التي يجب أن تكون مختارة بعناية، لأن ذلك سيساهم في سرعة تعلم الشبكة. ومجموعة الأمثلة هذه تسمى فئة التدريب.
وتنقسم طرق تعليم شبكة عصبية إلى قسمين حسب فئة التدريب التي تعرض على الشبكة. وهما:

- التعليم بواسطة معلم:Supervised Learning

في هذه الطريقة تكون فئة التدريب التي تعرض على الشبكة عبارة عن زوجين من المتجهات، متجه المدخلات

وهو عبارة عن القيم المدخلة للشبكة، ومتجه المخرجات وهو عبارة عن القيم التي يجب أن تخرجها الشبكة.
مثال:

Input:(0 1 0 1 0 0 0 1) Output:(0 1 1)

- التعليم بدون معلم:Unsupervised learning

في هذه الطريقة تكون فئة التدريب عبارة عن متجه المدخلات فقط دون عرض المخرجات على الشبكة.
في الدرس التالي سنعرف على أول شبكة تتعلم بواسطة معلم وهيPerceptron Network

الشبكة العصبيةPerceptron

(Perceptron Network)

Perceptron Neural Network:

بعد أن تعرفنا على ماهية الشبكات العصبية وما هو الهدف منها سنتعرف على أحد الطرق التي يتم فيها بناء شبكة عصبية، وهي.Perceptron

الـ Perceptron Network تستخدم في التطبيقات التي تحتاج إلى تصنيف الأنماط Pattern Classification إلى فئتين فقط. أي التي يمكن فصل الأنماط بها بيانياً بواسطة خط مستقيم وتسمى هذه النوعية من الأنماط: الأنماط القابلة للفصل خطياً Linear separability patterns.

وتتكون هذه الشبكة من طبقة المدخلات وطبقة واحدة أو طبقتين من طبقة المعالجة بحيث لا تزيد طبقة المعالجة

عن طبقتين، بالإضافة إلى أنه يتم ضبط الأوزان لطبقة واحدة فقط من طبقات الوصلات البينية التي تربط بين الطبقات السابقة، لتبقى الطبقة الأخرى (إن وجدت) ثابتة الأوزان. و هذه الشبكة تتعلم عن طريق معلم.

ولهذه الشبكة مميزات وعيوب، من مميزاتها أن بناء برنامج لها سهل جداً.

ومن عيوبها:

- أنها لا تستطيع أن تصنف الأنماط لأكثر من فئتين.
- عدد طبقات المعالجة فيها محدد بطبقة واحدة أو طبقتين فقط.
- ضبط أوزان الوصلات البينية فيها يتم على طبقة واحدة فقط من طبقات الوصلات البينية.

يمر تعليم هذه الشبكات العصبية بمرحلتين وهما:

- مرحلة التعليم
- مرحلة الاختبار

مرحلة التعليم :

هي المرحلة التي يتم فيها ضبط أوزان الوصلات البينية حتى تصل إلى أوزان قادرة على إعطاء إجابات صحيحة. ويتم ذلك عن طريق قيام وحدات المعالجة بثلاث عمليات رئيسية:

عملية جمع الأوزان :Weighted Sum

تقوم كل وحدة معالجة بعملية الجمع لكل وزن مدخل لها و الملحق بالوصلة البينية التي تربط بينها وبين الوحدة الموجودة في الطبقة التي تسبقها، مضروباً في القيمة الخارجة من تلك الوحدة، وهو على الصيغة:

$$Sj=\sum ai \ \ wji$$

حيث wji هو الوزن الملحق بالوصلة البينية التي تربط وحدة المعالجة j بالوحدة i الموجودة في الطبقة التي تسبقها.

و ai هي القيمة الخارجة من الوحدة i

و Sj هي ناتج عملية الجمع لكل وحدة معالجة .j

عملية التحويل :Transformation
تتم هذه العملية في الطبقة الأخيرة من طبقات المعالجة حيث يتم تحويل ناتج عملية الجمع المذكور في العملية السابقة إلى أحد القيم التي يفترض أن تكون ضمن نواتج الشبكة المرغوب بها. فمثلاً لو كانت الشبكة ستتعلم كيف تصنف الأعداد إلى فردي وزوجي، على أن تعطي كل عدد فردي القيمة 0 و كل عدد زوجي القيمة 1.
فإن قيمة Sj وهو ناتج عملية الجمع لن يعطي القيمة 0 أو 1 غالباً، لذا لا بد من تحويل هذا الناتج إلى أحد هذه القيمتين، وذلك عن طريق قاعدة التحويل والتي يحددها المبرمج. فمثلاً تكون القاعدة كالتالي:

if Sj>0 then Xj=1

if Sj<=0 then Xj=0

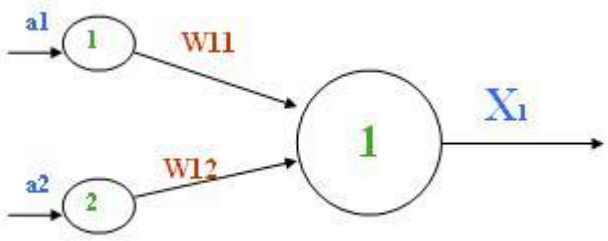
حيث Xj هي القيمة الخارجة من وحدة المعالجة j
عملية ضبط أوزان الشبكة :weights adjustment
بعد إتمام عملية التحويل يتم مقارنة الناتج الذي تعطيه الشبكة مع الناتج الصحيح الذي يفترض أن تعطيه الشبكة وذلك عن طريقة طرح الناتج الهدف(الصحيح) من ناتج الشبكة، فإذا كان ناتج الطرح مساوياً للصفر فهذا يعني أن الشبكة أخرجت ناتجاً صحيح، أما إن كان غير ذلك فالشبكة تحتاج لضبط أوزانها، وذلك من خلال قاعدة التعليم :learning rule

wjinew= wjiold+C(tj-Xj)ai

حيث wjinew هي قيمة الوزن الجديد الملحق بالوصلة البينية بين الوحدة j والوحدة i
و wjiold هي قيمة الوزن القديم الملحق بالوصلة البينية بين الوحدة j والوحدة i
و C هو معدل التعلم learning rate وهي قيمة ثابتة عادة ما تكون قيمة أقل من 1
و tj هي القيمة الهدف للشبكة
و Xj هي القيمة التي أنتجتها الشبكة
و ai هي المخرج من الوحدة i
إذا لم تفهم كل هذه العمليات من خلال القراءة، فهذا أمر طبيعي، سنحتاج أولاً لتطبيق عملية المحاكاة بواسطة الورقة والقلم أولاً حتى نفهم عملية التعليم ثم نصمم برنامج مناسب لها كما سنرى في الدرس القادم.

تطبيق مثال عملي على شبكة Perceptron

المثال:



لنفرض أن لدينا شبكة من النوع Perceptron ونريد تعليمها كيف تطبق الدالة المنطقية ،AND و هذه الدالة تعطي القيمة 1 إذا كانت كلاً من قيمة المدخلين مساوية للواحد، وصفر في الحالات الأخرى.

وإليك مجموعة الأمثلة التالية لتدريب الشبكة:

t	I2	I1
0	0.0	0.0
0	1.0	0.0
0	0.0	1.0
1	1.0	1.0

حيث I1 و I2 ترمز للمدخلات، بينما t ترمز للمخرج أو النتيجة الهدف) المرغوب بها).

وقاعدة التحويل هي:

if Sj>0 then Xj=1

if Sj<=0 then Xj=0

وسنفرض معدل التعلم بالقيمة .C=0.5

لتمثيل هذه الشبكة فإننا نحتاج لوحدتي إدخال لأن هناك مدخلين، ووحدة معالجة واحدة لأن هناك مخرج واحد كالتالي:

من الرسم نجد أن هناك طبقة واحدة من الوصلات البينية وسنفرض لها قيم أوزان مختلفة على أن تكون في المدى من 0.5- إلى 0.5

w11=0.1

w12=0.3

سنبدأ الآن بالمدخل الأول ونطبق أول عميلة وهي عملية الجمع:

$$S= 0.1 * 0.0 + 0.3 * 0.0$$

$$S=0.0$$

وحيث أن وحدة المعالجة هذه هي آخر وحدة معالجة فيجب تطبيق قاعدة التحويل:

$$S=0.0<=0.0$$

$$X=0 \text{ إذاً}$$

وبعد ذلك سنطبق عملية ضبط الأوزان، ولكن أولاً نتحقق مما إذا كان هناك حاجة لضبط الأوزان أم لا:

الناتج الهدف للمدخل الأول هو 0 والناتج الذي أخرجته الشبكة هو 0 وحاصل طرح هذين الناتجين هو ،0 وهذا يدل على أن أوزان الشبكة ليست بحاجة إلى تعديل.

لننتقل إلى المدخل الثاني مستخدمين نفس الأوزان السابقة لأنه لم يتم تعديلها لعدم الحاجة لذلك:

نبدأ من جديد بتطبيق عملية الجمع:

$$Sj=\sum ai \ wji$$

$$S= 0.1 * 0.0 + 0.3 * 1.0$$

$$S=0.3$$

ثم قاعدة التحويل:

$$if\ Sj>0\ then\ Xj=1$$

$$if\ Sj<=0\ then\ Xj=0$$

S=0.3>0.0

X=1

وبعد ذلك عملية ضبط الأوزان:

الناتج الهدف للمدخل الثاني هو 0 أيضاً، والناتج الذي توصلت له الشبكة هو 1 بطرح ناتج الشبكة من الناتج الهدف يعطي القيمة 1-

وهي غير مساوية للصفر لذا لا بد من ضبط جميع أوزان الشبكة.

wjinew= wjiold+C(tj-Xj)ai

w11new=0.1+ 0.5 * (-1)* 0.0

w11new=0.1

w12new=0.3+ 0.5 * (-1)* 1.0

w12new= -0.2

وبذلك نكون قد انتهينا من المدخل الثاني لنبدأ في الثالث مستخدمين الأوزان بعد تعديلها، متبعين نفس الخطوات السابقة:

قاعدة الجمع:

S= 0.1 * 1.0 +(- 0.2) * 0.0

S=0.1

قاعدة التحويل:

S=0.1>0.0

X=1

بعد ذلك عملية ضبط الأوزان:

الناتج الهدف للمدخل الثاني هو 0 أيضاً، والناتج الذي توصلت له الشبكة هو 1 بطرح ناتج الشبكة من الناتج الهدف يعطي القيمة 1-

وهي غير مساوية للصفر لذا لا بد من ضبط جميع أوزان الشبكة.

عملية ضبط الأوزان:

$$w11new=0.1+ 0.5 * (-1)* 1.0$$

$$w11new= -0.4$$

$$w12new= -0.2+ 0.5 * (-1)* 0.0$$

$$w12new= -0.2$$

ننتقل للمدخل الرابع والأخير مستخدمين الأوزان بعد التعديل:

عملية الجمع:

$$S= -0.4 * 1.0 +(- 0.2) * 1.0$$

$$S= -0.6$$

قاعدة التحويل:

$$S= -0.6<0.0$$

$$X=0$$

بعد ذلك عملية ضبط الأوزان:

الناتج الهدف للمدخل الثاني هو 1 أيضاً، والناتج الذي توصلت له الشبكة هو 0 بطرح ناتج الشبكة من الناتج الهدف يعطي القيمة 1

وهي غير مساوية للصفر لذا لا بد من ضبط جميع أوزان الشبكة.

عملية ضبط الأوزان:

$$w11new= -0.4+ 0.5 * (1)* 1.0$$

$$w11new= 0.1$$

$$w12new= -0.2+ 0.5 * (1)* 1.0$$

$$w12new= 0.3$$

وبهذا نكون قد انتهينا من عرض جميع الأمثلة) متجهات من الأنماط) على الشبكة وعرض جميع الأمثلة على الشبكة تسمى محاولة، ففي هذه المحاولة أعطت الشبكة نتيجة واحدة فقط صحيحة من أصل 4 نتائج، أي أن الشبكة لم تتعلم بعد وتحتاج لمحاولات أخرى بنفس الطريقة، حتى تستطيع إعطاء إجابات صحيحة لكل الأمثلة، حينها نقول أن الشبكة تعلمت، ويتبقى فقط اختبارها.

مرحلة الاختبار:

اختبار الشبكة مشابه تماماً لعملية التعليم إلا أن الشبكة في هذه المرحلة لا تضبط أوزانها، وإنما فقط تقوم بعمليتي الجمع والتحويل ومقارنة الناتج الذي تنتجه الشبكة بالناتج الهدف. حيث يتم عرض فئة اختبار على الشبكة وتحتوي هذه الفئة على مجموعة من المدخلات والمخرجات المصاحبة لكل مدخل. ويفضل أن تكون فئة الاختبار مختلفة عن فئة التدريب.

فإذا استطاعت الشبكة اجتياز الاختبار وإعطاء إجابات صحيحة، يكون تعليم الشبكة ناتج، وتصبح الشبكة جاهزة للاستخدام.

أسباب عدم تعلم الشبكة:

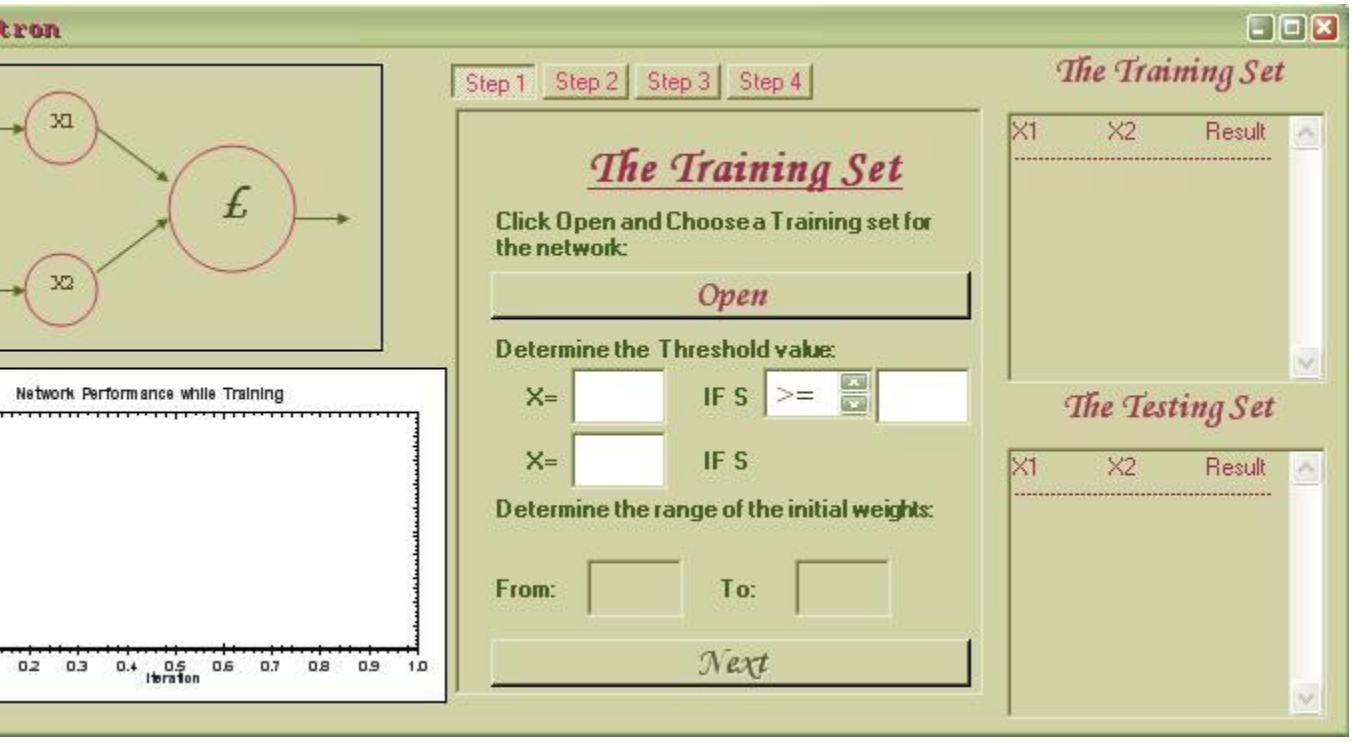
قد يتم تصميم الشبكة وتدريبها، ولكنها بالنهاية تفشل! و أسباب فشلها في الواقع متعددة منها:

- نوعية الشبكة لا تناسب التطبيق الذي تم تدريبها عليه، وهذا يستلزم اختيار شبكة أخرى.
- عدد وحدات المعالجة غير مناسب.
- الأوزان التي تبدأ بها الشبكة غير مناسبة.
- معدل التعلم غير مناسب.
- فئة التدريب لم يتم اختيارها بعناية.
- قاعدة التحويل غير مناسبة.

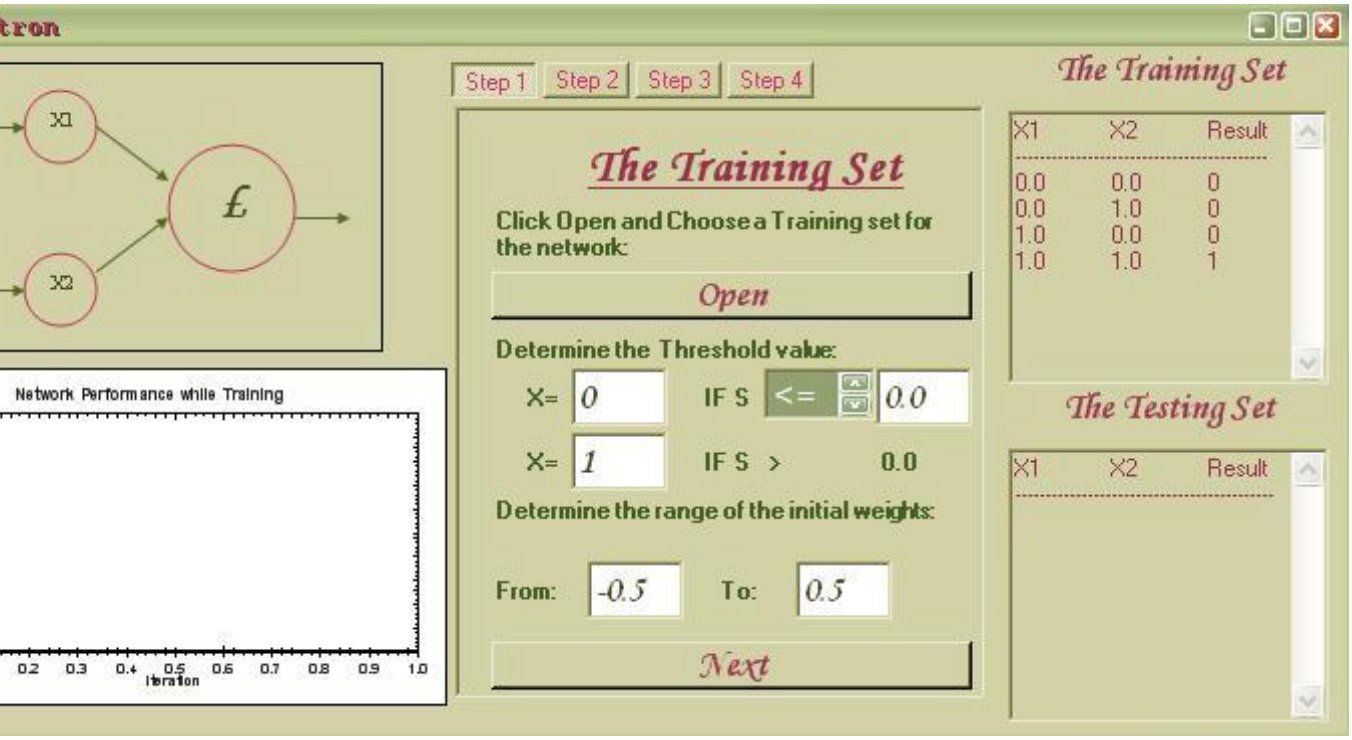
برنامج يحاكي الـ Perceptron Network:

الصور التالية تعرض لقطات لبرنامج تم تصميمه بلغة C# ليحاكي عمل هذه الشبكة، وتم تطبيق المثال السابق عليه:

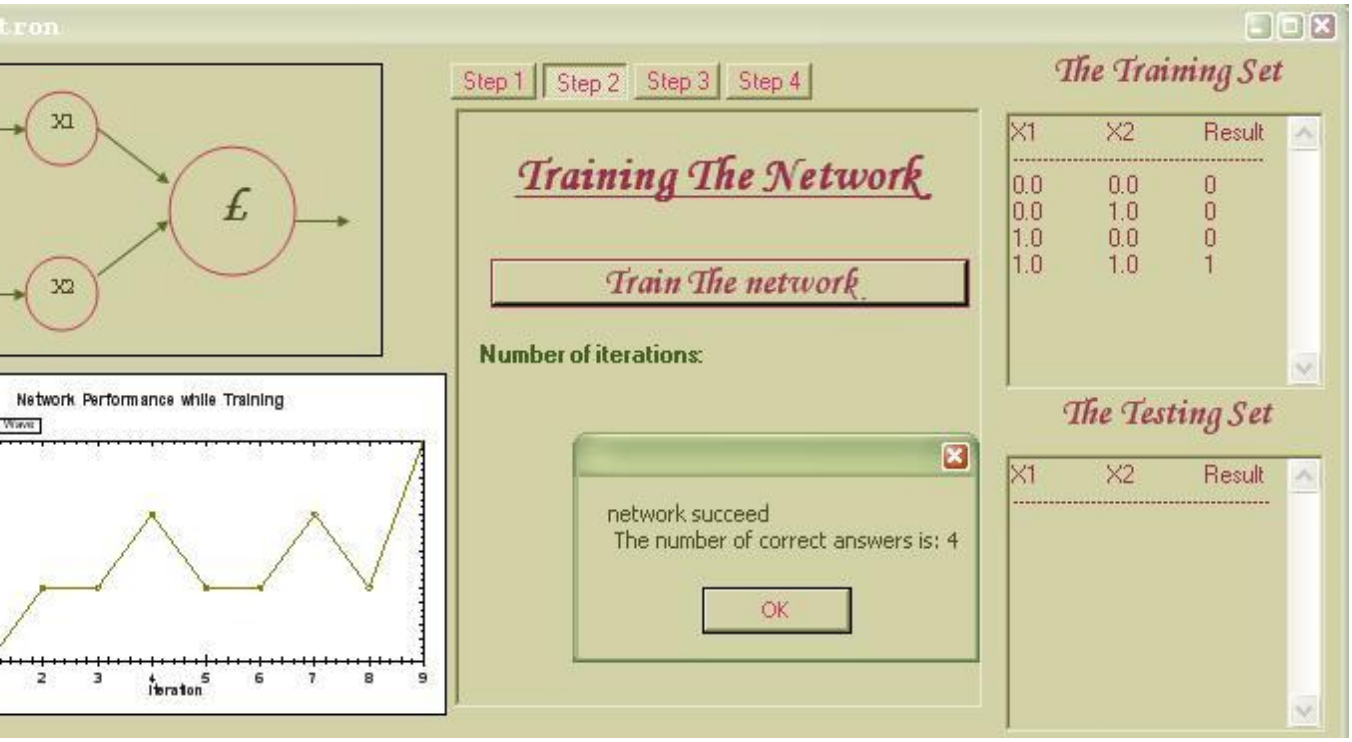
- بداية البرنامج:



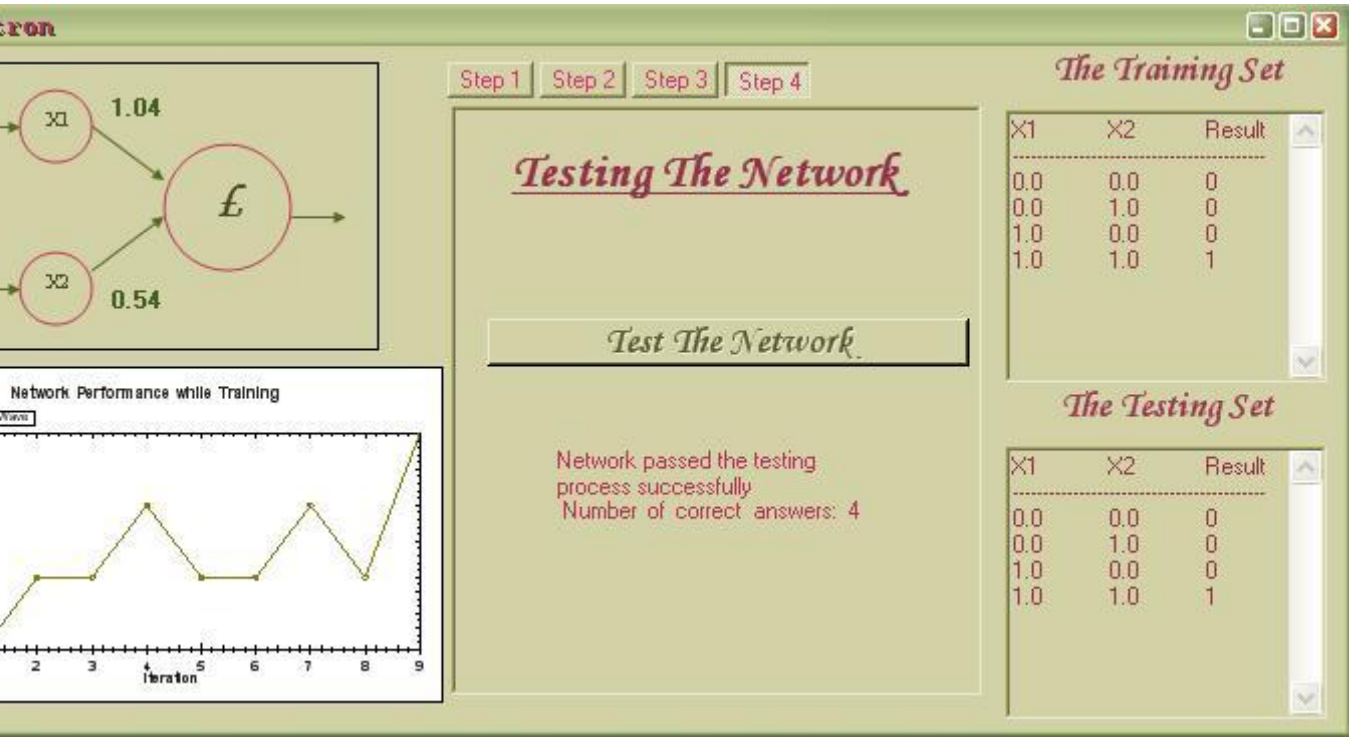
- يتم اختيار ملف يحتوي على فئة التدريب، ويتم تحديد قاعدة التحويل و المدى الذي سيتم تحديد قيم عشوائية منه لتكون القيم الابتدائية للأوزان:



- **بعد ذلك ننتقل للخطوة الثانية وهي تدريب الشبكة، ويتم ذلك عن طريق تطبيق العمليات السابقة برمجياً على فئة التدريب ولعدة محاولات حتى تتعلم الشبكة:**



- ثم يأتي دور اختبار الشبكة، حيث تحدد فئة الاختبار، وبالنهاية سيتم عرض نتيجة الاختبار وهنا الشبكة قد تعلمت واستطاعت أن تعطي إجابات صحيحة لكل فئة الاختبار:



بإمكانك تحميل البرنامج من هنا:

لتحميل البرنامج

لتحميل الشيفرة المصدرية للبرنامج

```
public void PerceptronAlg()
{
int testPerformance;
string text;
testPerformance=NetPerformance();
good[tries]=testPerformance;
while(tries < SESSION)
```

```
{
if(testPerformance != total)
{
TeachNet();
tries++;
testPerformance=NetPerformance();
good[tries]=testPerformance;
}
else
break;
}
x = new double[tries+1];
y = new double[tries+1];
int i;
for ( i=0; i< FONT>
{
x[i] = Convert.ToDouble(i);
y[i] = Convert.ToDouble( good[i] );
}
zedGraphControl1.GraphPane.AddCurve( "Sine
Wave", x, y, Color.Olive, SymbolType.Circle);
```

```
zedGraphControl1.GraphPane.AxisChange();

zedGraphControl1.Refresh();

if(NetPerformance()!= total)

{

text="network failedn The number of correct
answers is: "+
Convert.ToString(NetPerformance());

MessageBox.Show(text);

}

else

{

text="network succeedn The number of correct
answers is: "+
Convert.ToString(NetPerformance());

MessageBox.Show(text);

}

}
```

```
public void TeachNet()

{

Vector pattern;
```

```
int correctAnswer;
int netOutput;
for(int i=0;i<TOTAL;I++)< FONT>
{
pattern = patterns[i];
correctAnswer = target[i];
netOutput = ThresholdRule(pattern);
if( netOutput!= correctAnswer)
{
UpdateWeight(pattern,netOutput,correctAnswer);
}
}
}
```

```
public int ThresholdRule(Vector inputPat)
{
switch(s)
{
case "<":
if(WeightedSum(inputPat) < S )
return cat1;
```

```
else
return cat2;
break;
case ">":
if(WeightedSum(inputPat) > S )
return cat1;
else
return cat2;
break;
case "<=":
if(WeightedSum(inputPat) <= S )
return cat1;
else
return cat2;
break;
case ">=":
if(WeightedSum(inputPat) >= S )
return cat1;
else
return cat2;
```

```
break;

default:

return 0;

break;

}

}
```

```
public void UpdateWeight(Vector inputPat, int output, int desiredOutput)

{

w.x1 = w.x1 + C * (desiredOutput - output) * inputPat.x1;

w.x2 = w.x2 + C * (desiredOutput - output) * inputPat.x2;

bais = bais + C * (desiredOutput - output);

}
```

```
public double WeightedSum(Vector inputs)

{

return (inputs.x1 * w.x1) + (inputs.x2 * w.x2) + bais;

}
```

```
public int NetPerformance()
```

```
{
Vector pattern;
int counter=0;
for(int i=0;i<TOTAL;I++)< FONT>
{
pattern=patterns[i];
if(ThresholdRule(pattern)==target[i])
counter++;
}
return counter;
}
```

```
public void TestNet()
{
int [] dis=new int [total1];
int netPass=0;
Vector pattern;
int correctAnswer;
int netOutput;
for(int i = 0;i<TOTAL1;I++)< FONT>
{
```

```
pattern = testingPattern[i];

correctAnswer = testingTarget[i];

netOutput = ThresholdRule(pattern);

dis[i]=netOutput;

if( netOutput== correctAnswer)

{

netPass++;

}

}

if(netPass == total1)

label6.Text+="Network passed the testing
process successfullyn Number of correct answers:
"+Convert.ToString(netPass)+"t";

else

label6.Text+="Network didn't pass the testing
process successfullyn Number of correct answers:
"+Convert.ToString(netPass)+"t";

}
```

مقدمة في العمليات على القوائم

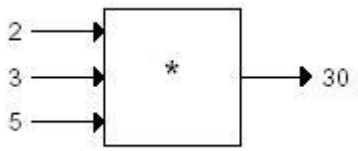
بما أننا في الموسوعة العربية للكمبيوتر والإنترنت خصصنا دروس قسم الذكاء الاصطناعي للغة الـ Lisp وهي اختصار لـ List Processing، ستفيدنا هذه المقدمة في استيعاب أسلوب البرمجة في الذكاء الاصطناعي وفن التعامل مع القوائم إن شاء الله.

محتوى الدرس:

1. **الدوال والبيانات Function & Data.**
2. **القوائم Lists.**
3. **العمليات على القوائم ومفهوم تنفيذها List Processing & EVAL Notation.**

أولاً: الدوال والبيانات:Function & Data

لو نظرنا إلى الشكل التالي والذي يمثل دالة للضرب أدخلنا لها ثلاث مدخلات، وطلب منّا وصف هذا الشكل.. فكيف سنصفه؟

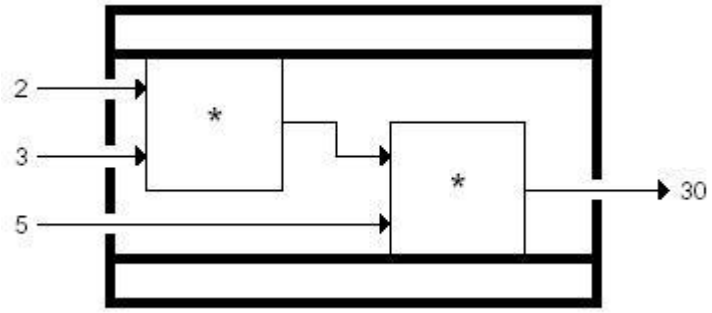


هناك عدة طرق للوصف:

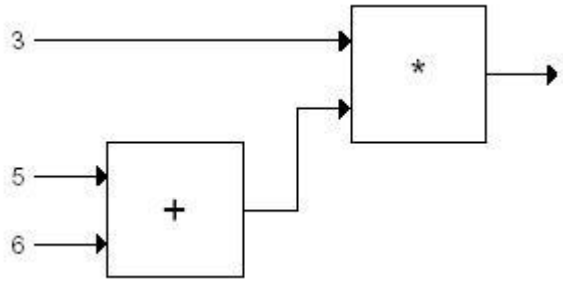
- **الوصف من جهة المدخلات: فنقول أن هناك ثلاث بيانات مدخلة لدالة الضرب التي ستقوم بعملية ضربهم وإعادة الناتج.**
- **الوصف من جهة المخرجات: فنقول أن الدالة ستعيد إلينا 30 نتيجة ضرب ثلاثة أرقام هي 2، 3 و5.**

- **الوصف من وجهة نظر المبرمج: وهنا سنقول أن لدينا دالة وظيفتها هي الضرب، تأخذ عدد من المدخلات وتعيد إلينا الناتج.**

ما يهمنا هنا كمبرمجين هو الوصف من وجهة نظر المبرمج، هنا تكمن فكرة العمليات على القوائم.
ولو نظرنا إلى الشكل التالي:



نلاحظ أننا حصلنا على نفس النتيجة ولكن باستخدام الدالة مرتين، في المرة الأولى أخذت بيانين كمدخلين، وفي المرة الثانية أخذت البيان الثالث مع نتيجة المرة الأولى كمدخلين.
كذلك، يمكننا استخدام أكثر من دالة بنفس الطريقة كما يوضح الشكل التالي:



والسؤال هنا: كيف تتم كتابة قائمة لهذا الوصف؟! لابد من أن نتعرف على القوائم وتركيبها للإجابة على هذا السؤال.

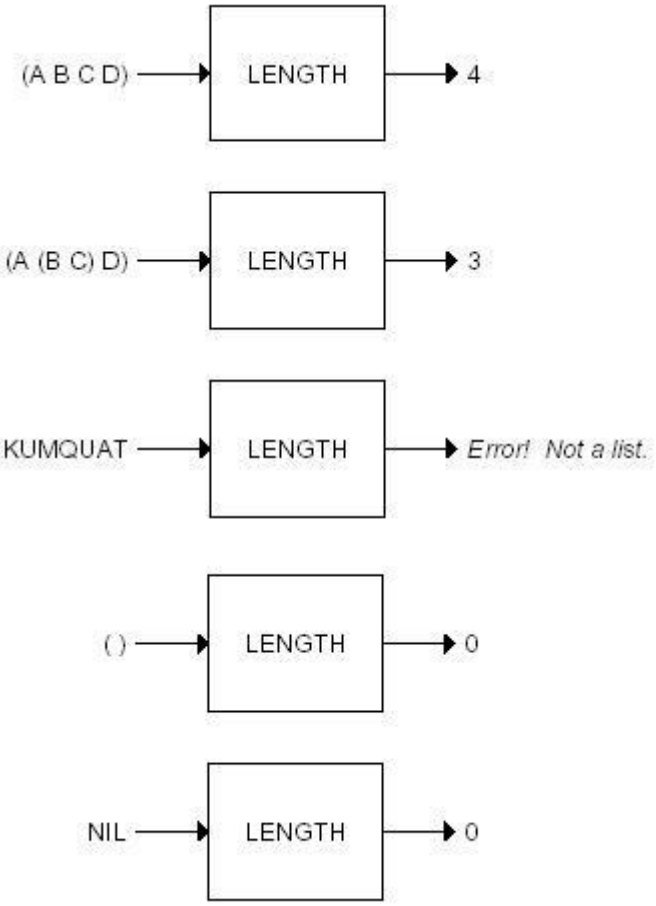
ثانياً: القوائم:Lists

س/ ماهي القائمة؟

القائمة هي مجموعة من العناصر التي وضعت داخل قوسين، نفصل بين كل عنصر والآخر بواسطة مسافة فقط، من الممكن أن تكون العناصر عبارة عن قوائم أخرى.
أمثلة:

- (RED GREEN BLUE) : قائمة بها ثلاث عناصر.
- (2 3 5 7 11 13 17) : قائمة بها سبعة عناصر.
- (MOHD 3 MAKKAH 459) : قائمة بها أربعة عناصر وتمثل سجل لشخص.
- ((BLUE SKY) (GREEN GRASS) (BROWN EARTH)) : قائمة تتكون من ثلاث عناصر كل عنصر عبارة عن قائمة من عنصرين.
- ( ) أو : NIL قائمة فارغة!

وتوضح الأشكال التالية دالة جاهزة في لغة الليسب اسمها Length تأخذ قائمة كمدخل وتخرج لنا عدد العناصر في هذه القائمة:



ثالثاً: العمليات على القوائم ومفهوم تنفيذها List Processing & EVAL Notation:

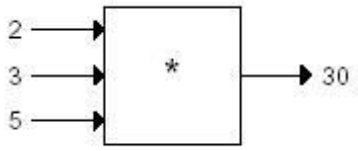
قبل بدء رحلتنا مع لغة الليسب، لابد من أن نتعرف على مفهوم مرن يدعى "EVAL notation" وفيه نستبدل المربعات بالقوائم عند تمثيل الدوال، حيث نلاحظ أن:

1. أي مفهوم برمجي يمثل بواسطة المربعات نستطيع تمثيله بسهولة بواسطة Eval notation.
2. Eval notationسهل الطباعة باستخدام لوحة المفاتيح، على عكس المربعات.
3. في لغة الليسب: هذا المفهوم يسمح لنا بكتابة دوال تقبل دوال آخرى كمدخلات، كما سنتعرف في الدروس القادمة.

هذا المفهوم يعتبر القلب النابض في لغة الليسب، وظيفته هو التعبيرعن العمليات بواسطة القوائم في هذه اللغة، يتم هذا التعبير بذكر الدالة function أولاً ثم تتبعها مجموعة المدخلات الخاصة بها في القائمة. مثال: لو كتبنا التعبير التالي:

(* 2 3 5)

والذي يقابل أول مثال ذكرناه في درسنا:



وهنا نقول:

the expression (* 2 3 5) evaluates to 30.

وبهذا الاسلوب تكتب الأكواد والبرامج بلغة الليسب، سنتعرف على هذه اللغة بالتفصيل في الدروس القادمة إن شاء الله، وماهذا الدرس سوى مقدمة لمفهوم العمليات على القوائم.

مقدمة:

لغة Lisp من أشهر اللغات المستخدمة لكتابة تطبيقات وبرامج الذكاء الاصطناعي، حيث أن 90% من برامج وتطبيقات الذكاء الاصطناعي الموجودة حاليا تم صناعتها باستخدام لغة Lisp .

Lisp Common ماهي إلا نسخة Version قياسية وشائعة جدا من لغة Lisp.

مميزات لغة Lisp: Common

1- يتم ترجمة الكود سطرا سطرا باستخدام المفسر Interpreter في الجزء Listener من البرنامج، بينما يتم عمل تقييم وتقدير Evaluation للكود ككل وإظهار الأخطاء مرة واحدة بعد قراءة جميع أسطر الكود في المحرر Text Editor الموجود في البرنامج.

2- لغة Lisp هي لغة إجرائية Procedural أي أن الكود يعكس الخوارزم Algorithm مباشرة وبالتالي فإنه يكون متسلسل ومتتالي تماما كالخوارزم أو خريطة التدفق Flowchart التي يمثلها.
وبالرغم من أن هذه الطريقة تعد قديمة في مقابل اللغات

الحديثة كالجافا والسي شارب والتي تستخدم تقنية البرمجة الشيئية Object Oriented Programming ، إلا أن جميع لغات الذكاء الاصطناعي لا تزال تستخدم الطريقة القديمة لأنها تعكس بوضوح الخوارزم مما يجعل الكود أكثر وضوحا.

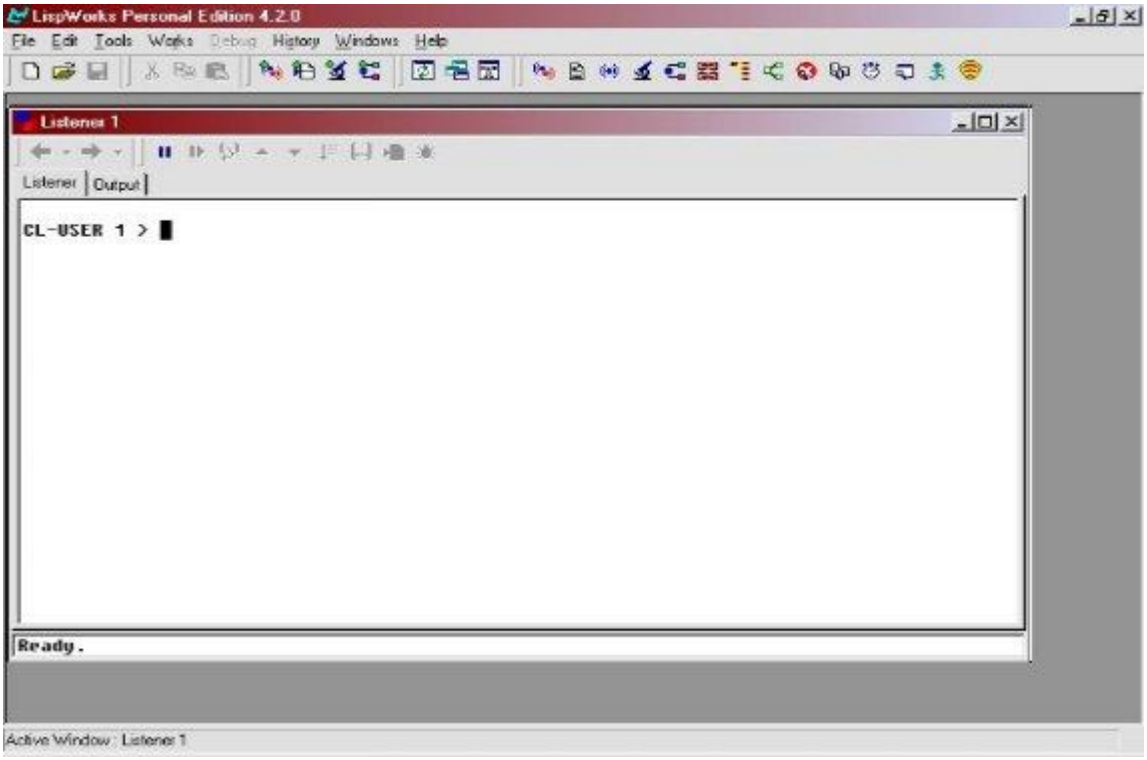
3- تعتمد هذه اللغة بشكل رئيسي على استخدام الدوال functions وهي على قسمين:

- دوال معرفة ومبنية مسبقا داخل اللغة Built-in Functions ، وهي كثيرة جدا، وتعتبر من أفضل الدوال المبنية داخل لغات البرمجة الأخرى، وسنتعرف عليها جميعا وبالتفصيل في الدروس القادمة بإذن الله.
- دوال يقوم بتعريفها المبرمج User-defined Functions ، وسنتعرف على كيفية تعريف أي دالة في لغة Lisp وكيفية الاحتفاظ بها لاستخدامها في التطبيقات التي تصنعها في الدروس القادمة بإذن الله..

وبالتالي فإننا نقول عن لغة Lisp بأنها Functional Language لأنها لغة تعتمد على الدوال ووظائفها ويكون الكود فيها مكون من دوال تستدعي بعضها لتؤدي معا وظيفة محددة.

* لمزيد من المعلومات عن لغة Lisp في قاموس الموسوعة العربية على الوصلة:
http://www.c4arab.com/qamoos/mean.php?word=Lisp

عندما تفتح البرنامج LispWorks لكتابة أكواد باستخدام لغة Common Lisp فإن واجهة البرنامج ستظهر لك بالشكل التالي:



والآن هل أنت متشوق للتعرف على دوال لغة Lisp وكتابة برامج الذكاء الاصطناعي؟
إذن لنبدأ بتحميل البرنامج الذي سنعمل عليه..

البرنامج: LispWorks

هناك نسخ متعددة للغة Lisp تم إنتاجها من قبل شركات متعددة، وأفضلها وأحدثها على الإطلاق LispWorks من إنتاج شركة xanalyz ، والموقع الرسمي للشركة المنتجة لـ LispWorks موجود على الوصلة التالية:
http://www.lispworks.com/company.html

تستطيع تحميل برنامج LispWorks على أي من أنظمة التشغيل التالية:

- **نظام التشغيل ويندوز بجميع إصداراته**
- **نظام التشغيل لينكس**
- **نظام التشغيل يونكس.**

وقد تم تخصيص ثلاث إصدارات أو ثلاث نسخ من LispWorks بإصدارات ونسخ مختلفة ومتطورة لكل نظام من هذه الأنظمة ، وهي كالتالي:

أولا: النسخة الشخصية

Personal Edition

نسخة مصممة من أجل البرمجة بلغة Lisp للأغراض الشخصية أو التعليمية.
هذه النسخة مجانية لجميع إصدارات أنظمة التشغيل، وتحتوي على نفس المترجم Compiler وبيئة التطوير الموجودة في لغة Common Lisp القياسية.

ثانيا: النسخة الإحترافية

Edition Professional

وتحتوي على أي شيء يحتاجه المبرمج من أجل برمجة وتطوير تطبيقات وبرامج Common Lisp التجارية.
والتطبيقات المطورة على هذه النسخة يمكن أن يتم توزيعها مجانيا. وتتضمن هذه النسخة نظام إدارة واجهات Common Lisp من أجل زيادة قابلية نقل البرامج...

* ستجد تعريف ومميزات نظام إدارة واجهات Common Lisp أو اختصارا CLIM في قاموس الموسوعة العربية على الوصلة التالية:

http://www.c4arab.com/qamoos/mean.php?word=Common%20Lisp%20Interface%20Manager

ثالثا: النسخة المتقدمة للمشاريع

Edition Enterprise

وتحتوي على جميع مميزات وخصائص النسخة الإحترافية بالإضافة لكونها تزودك بدعم إضافي للبرمجيات التي تحتاجها في المشاريع المتقدمة جدا ويتضمن ذلك:

- object-oriented Database access through SQL/ODBC libraries.
الوصول لقاعدة بيانات من خلال مكتبات SQL/ODBC.
- computing Industry standard distributed through LispWorks ORB.
حسابات موزعة قياسية صناعية من خلال LispWorks ORB.
- through Expert system programming KnowledgeWorksTM which has an embedded Prolog compiler.
برمجة النظم الخبيرة من خلال قاعدة معرفية KnowledgeWorks والتي تتضمن على مترجم لغة برولوج compiler Prolog داخلها.

تنصيب LispWorks على جهاز يعمل بنظام التشغيل ويندوز.

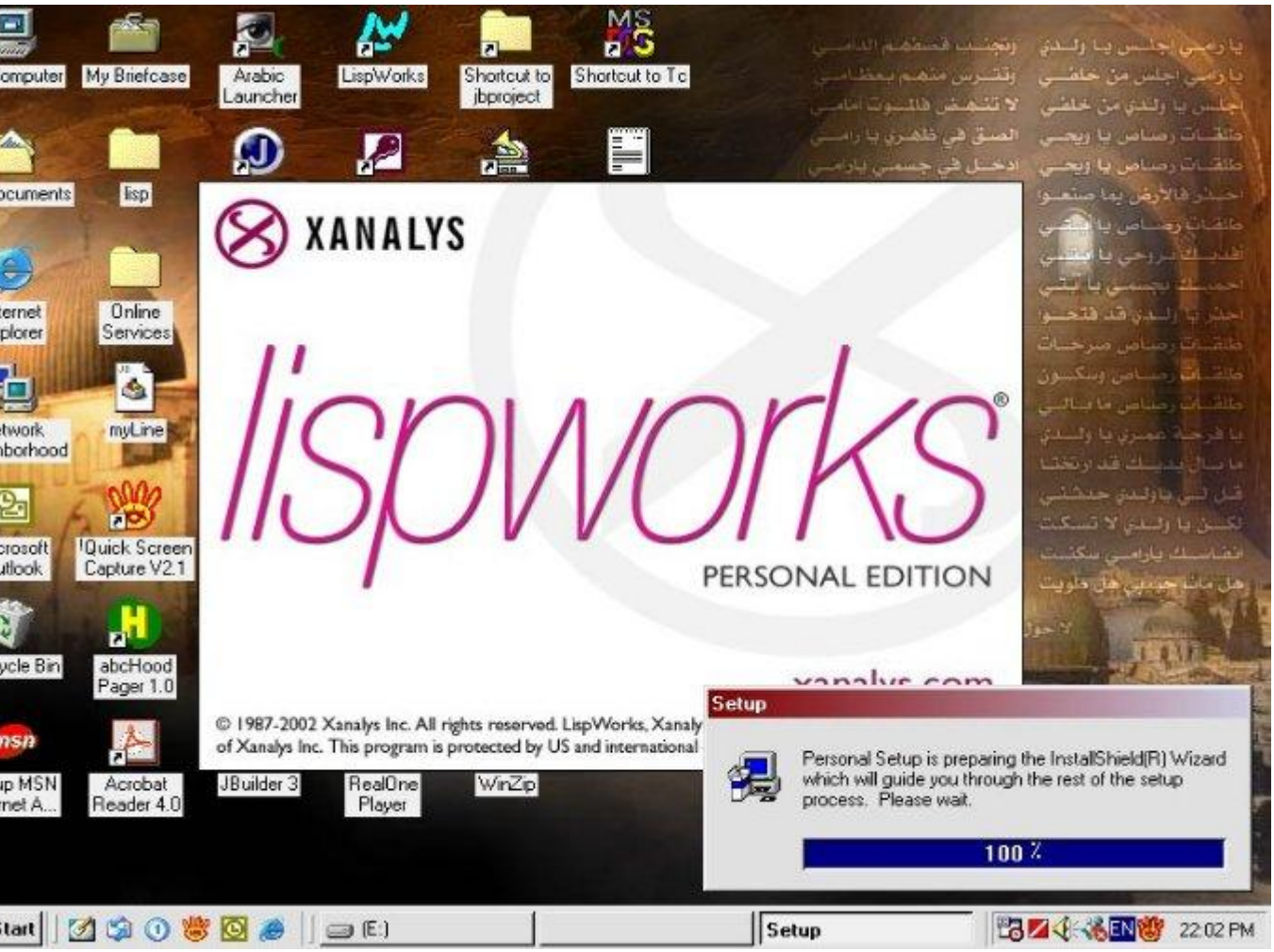
والآن بعد أن قمت بتحميل برنامج LispWorks والـDocuments الخاصة به من موقع الشركة في الدرس السابق، سأشرح لك كيف ستقوم بتنصيبها على جهازك..

خطوات سهلة جدا يمكن أن تقوم بها بنفسك إذا كنت من محترفي تنصيب البرامج على جهازك، وإذا كانت لغتك الإنجليزية جيدة.

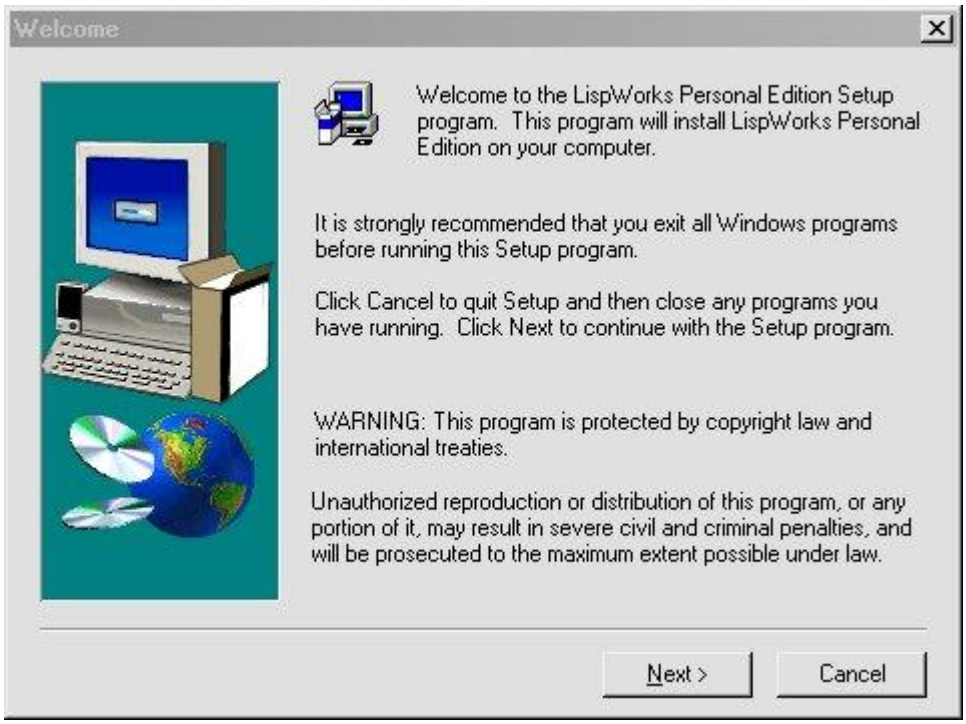
أولا: تحميل Software: LispWorks

1- بعد اكتمال تحميل LispWorks من على الشبكة، اتجه إلى المجلد الذي وضعت في الملف وستجد الأيقونة التالية:

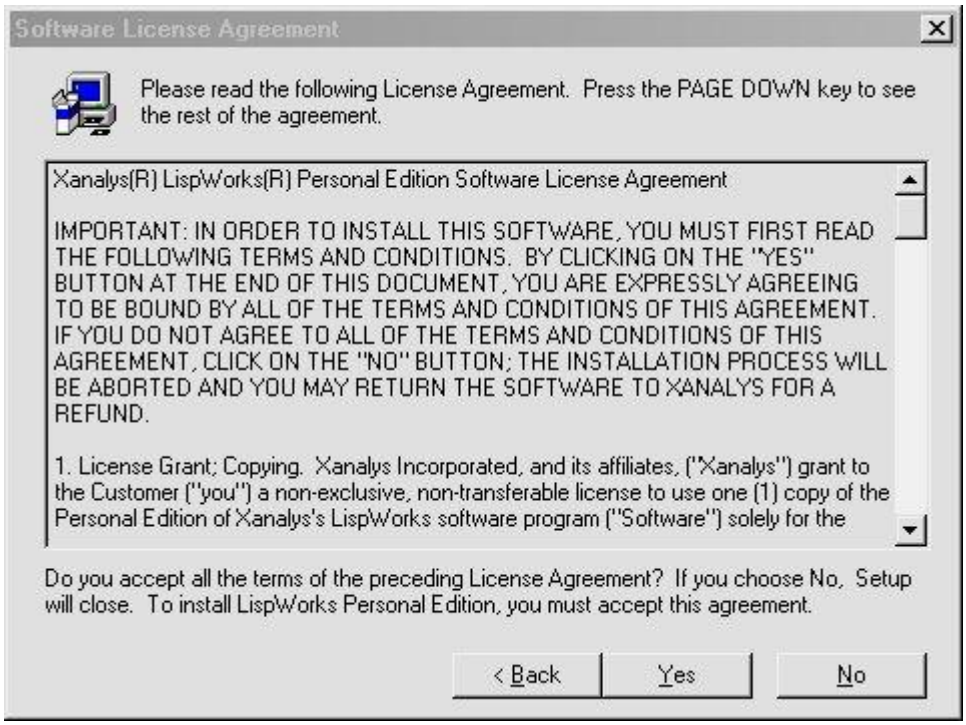
انقر عليها مرتين وستظهر لك الشاشة:



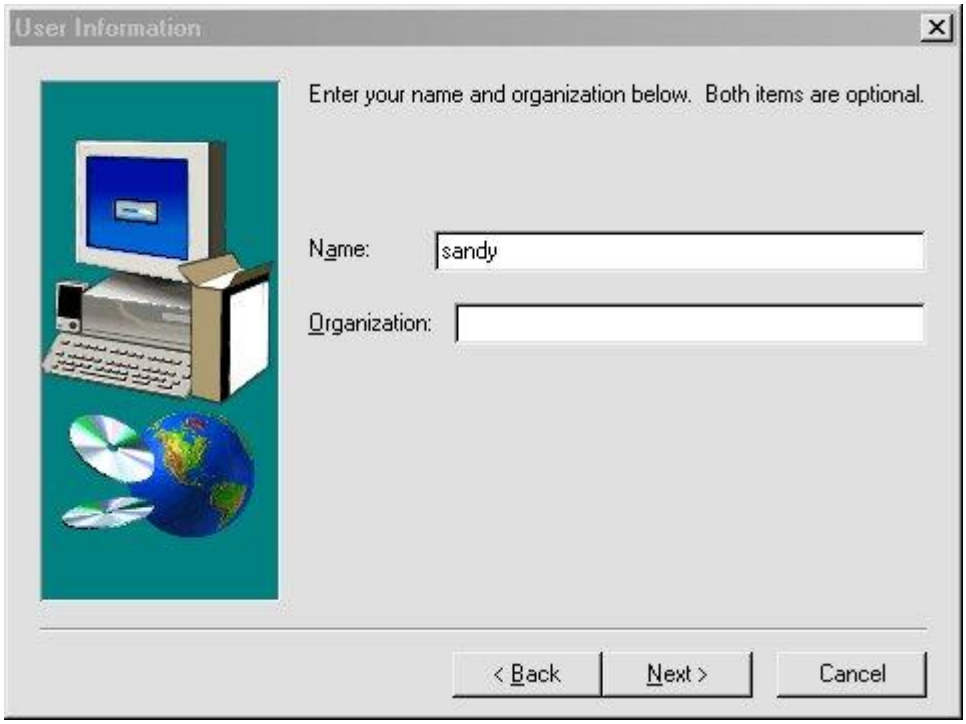
2- النافذة الأولى هي نافذة ترحيبية ، انقر Next



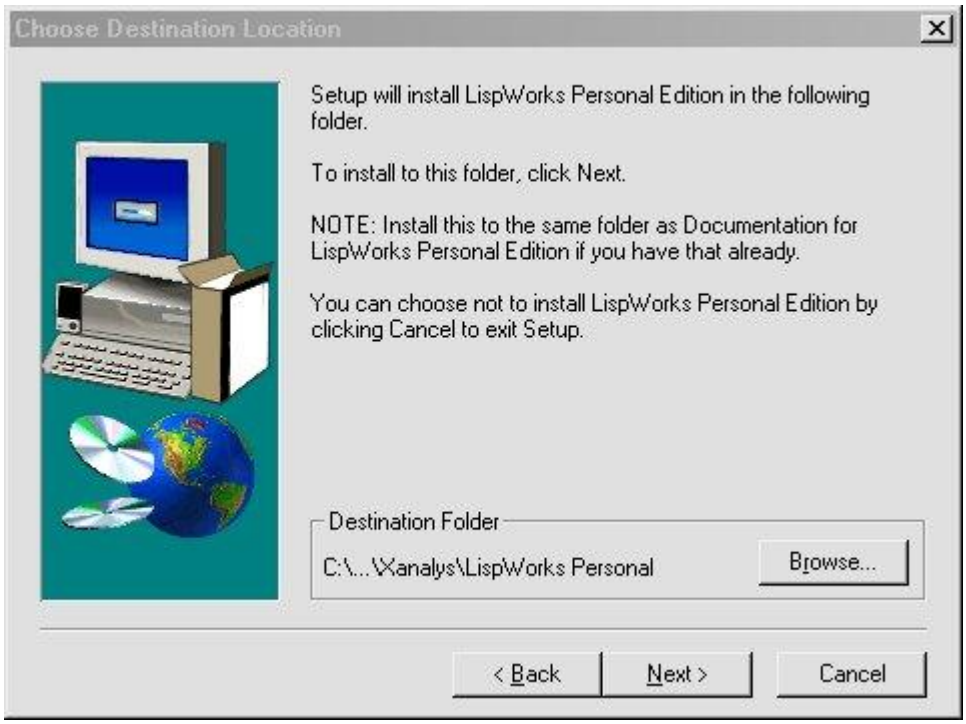
3- النافذة التي تليها هي اتفاقية الترخيص، انقر Yes



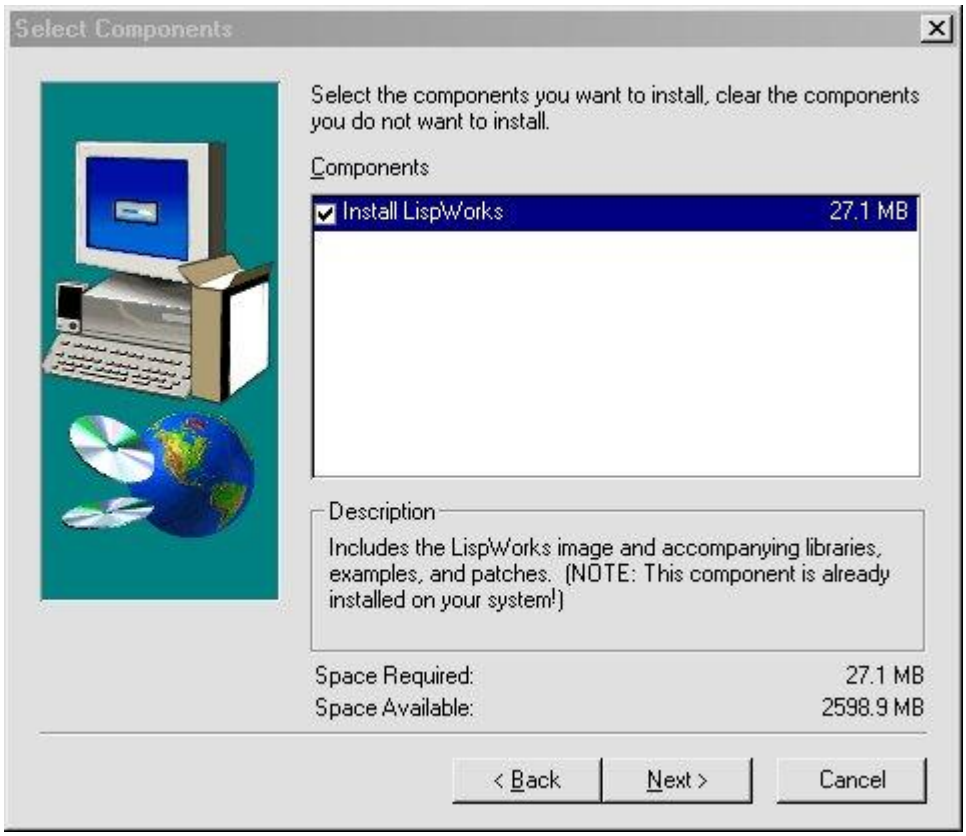
4- النافذة التي تليها تطلب منك إدخال اسمك واسم المنظمة التي تتبعها، وهي اختيارية وليست إجبارية.



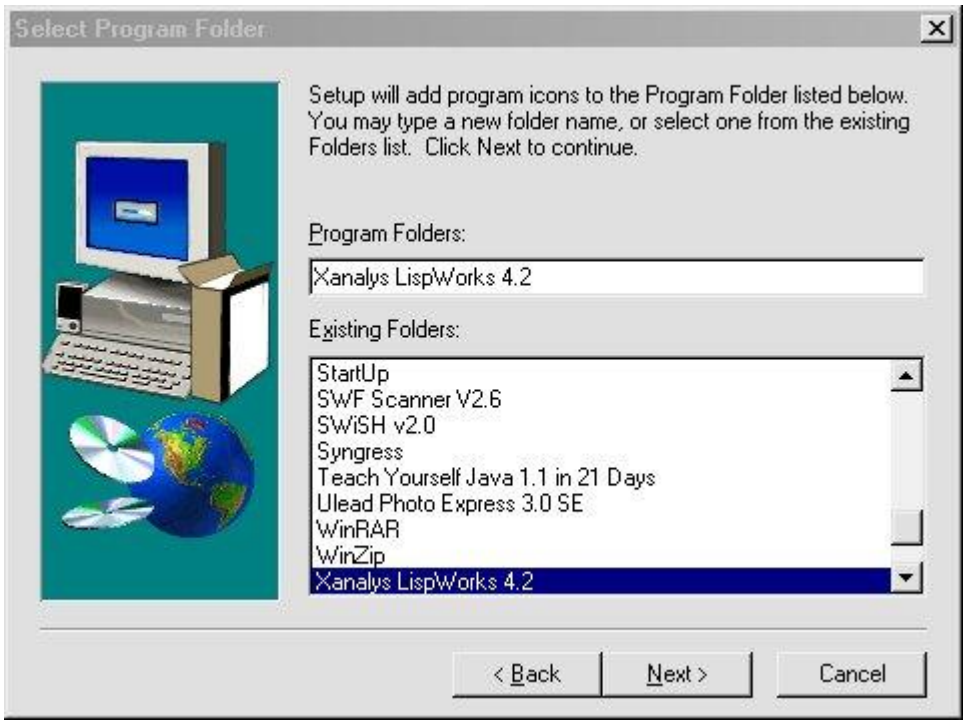
5- النافذة التي تليها تبين لك المسار الذي سيتم تنصيب البرنامج فيها، اضغط على Browse إذا كنت تريد تغييره، ثم انقر Next للمواصلة.



6- النافذة التالية تبين المكونات التي سيتم تحميلها، سيتم تحميل LispWorks فقط ولا يوجد أي مكونات إضافية، قم بالتأشير إذن على هذا الخيار ، انقر Next للمتابعة.



7- بعد ذلك تعرض لك النافذة مكان ظهور برنامج LispWorks مع البرامج الموجودة في قائمة ابدأ



8- هذه النافذة تعرض لك خيارين (وكلاهما غير مهم لتحميل البرنامج) :

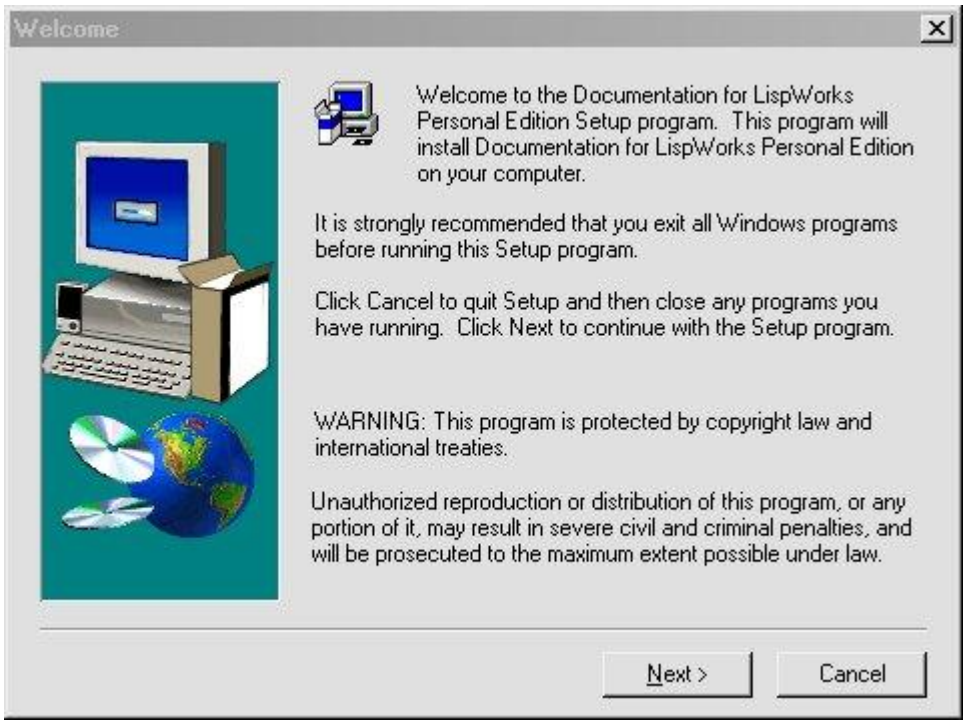
الأول: يعطيك خيار عرض ملف ReadMe الذي يعطي معلومات حول النسخة الشخصية من برنامج LispWorks.
الثاني: يعطي خيار وضع أيقونة هذا البرنامج مع البرامج التي لها خاصية الوصول السريع من خلال الشريط Quick Launch الموجود على شريط المهام.

اختر منها ما يلائمك أو اتركها جميعا ثم انقر على Finish.

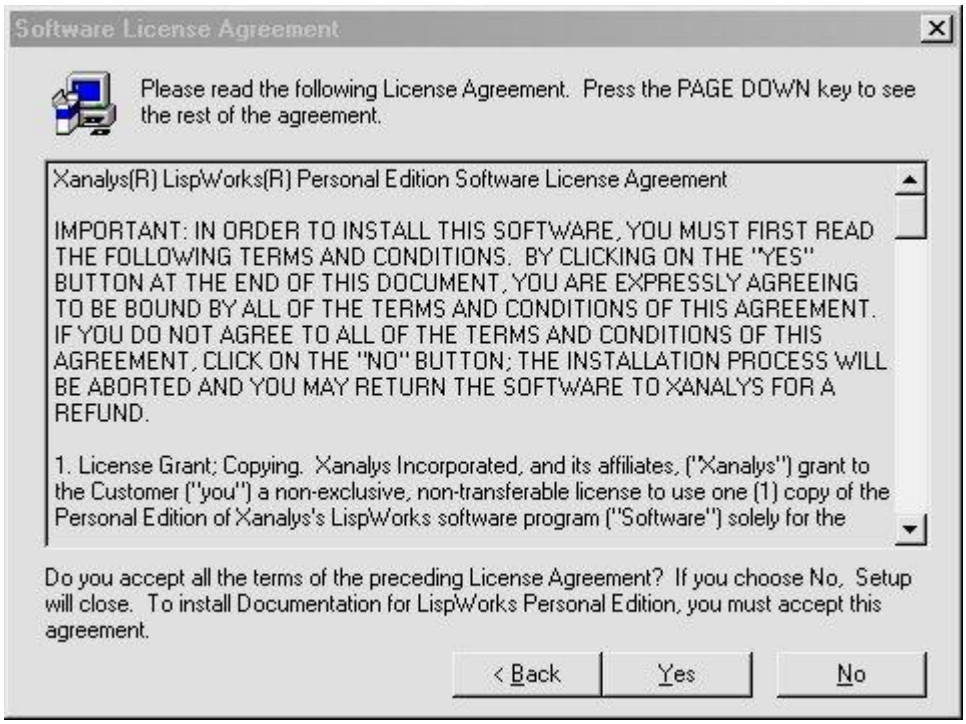
ثانيا: تنصيب وثائق LispWorks:

انقر على الأيقونة الخاصة به بعد اكتمال التحميل، واتبع الـWizard التالي:

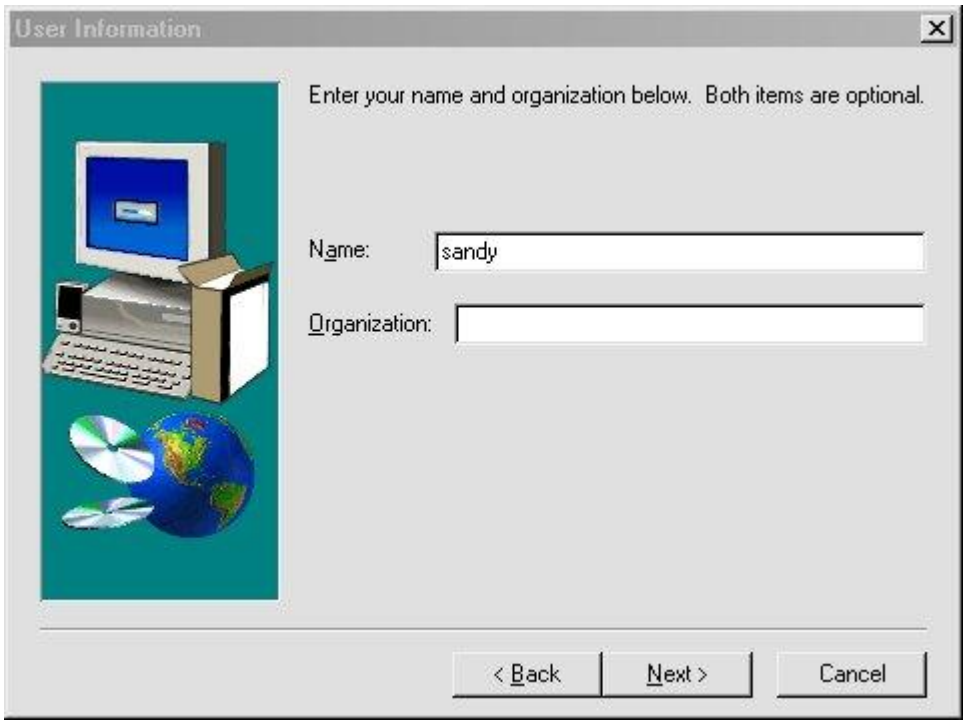
2- النافذة الأولى هي نافذة ترحيبية ، انقر Next



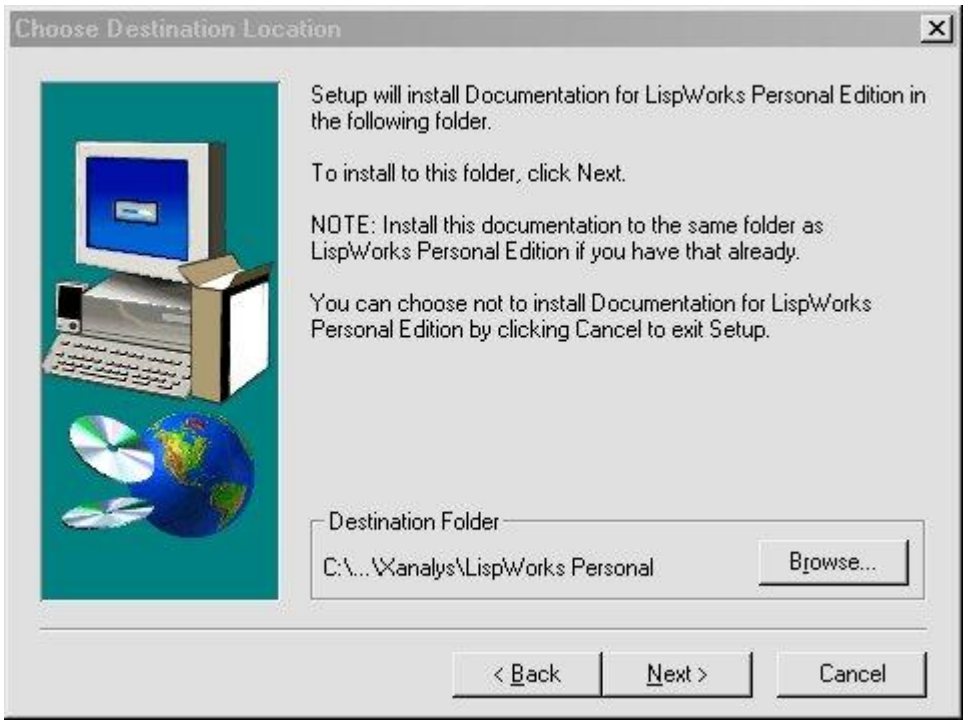
3- النافذة التي تليها هي اتفاقية الترخيص، انقر Yes



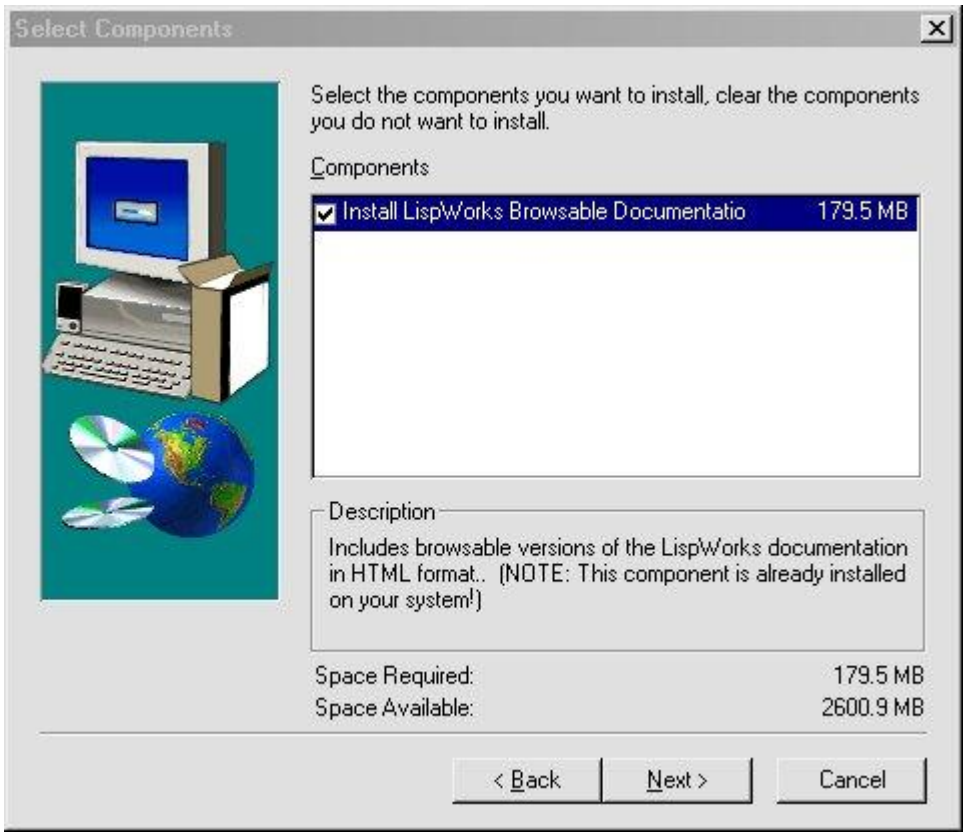
4- النافذة التي تليها تطلب منك إدخال اسمك واسم المنظمة التي تتبعها، وهي اختيارية وليست إجبارية.



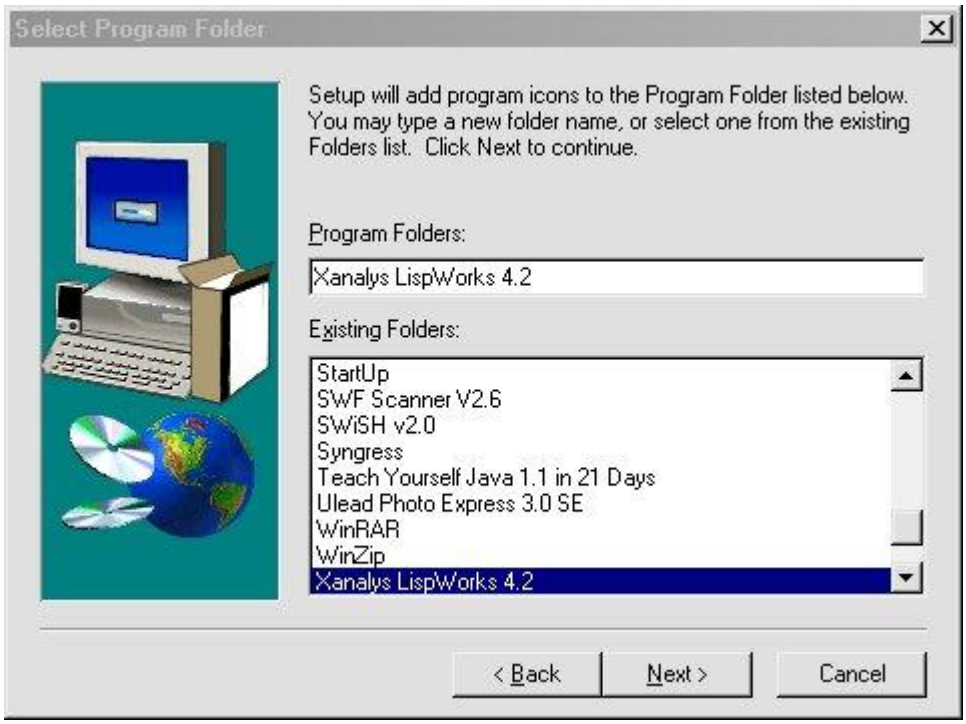
5- النافذة التي تليها تبين لك المسار الذي سيتم تنصيب البرنامج فيها، اضغط على Browse إذا كنت تريد تغييره، ثم انقر Next للمواصلة.



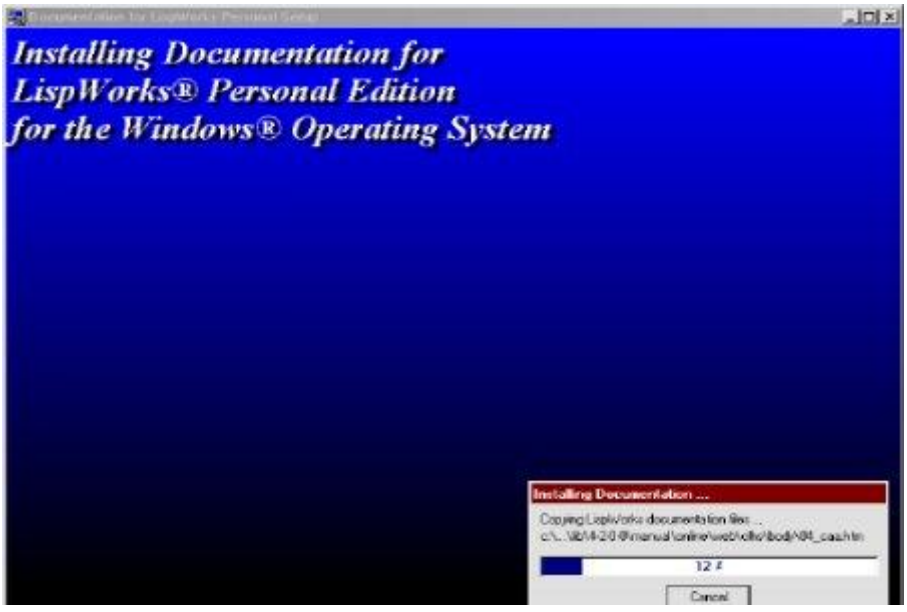
6- النافذة التالية تبين المكونات التي سيتم تحميلها، سيتم تحميل LispWorks فقط ولا يوجد أي مكونات إضافية، قم بالتأشير إذن على هذا الخيار ، انقر Next للمتابعة.



7- بعد ذلك تعرض لك النافذة مكان ظهور برنامج LispWorks مع البرامج الموجودة في قائمة ابدأ



وسيبدأ البرنامج بالتحميل:

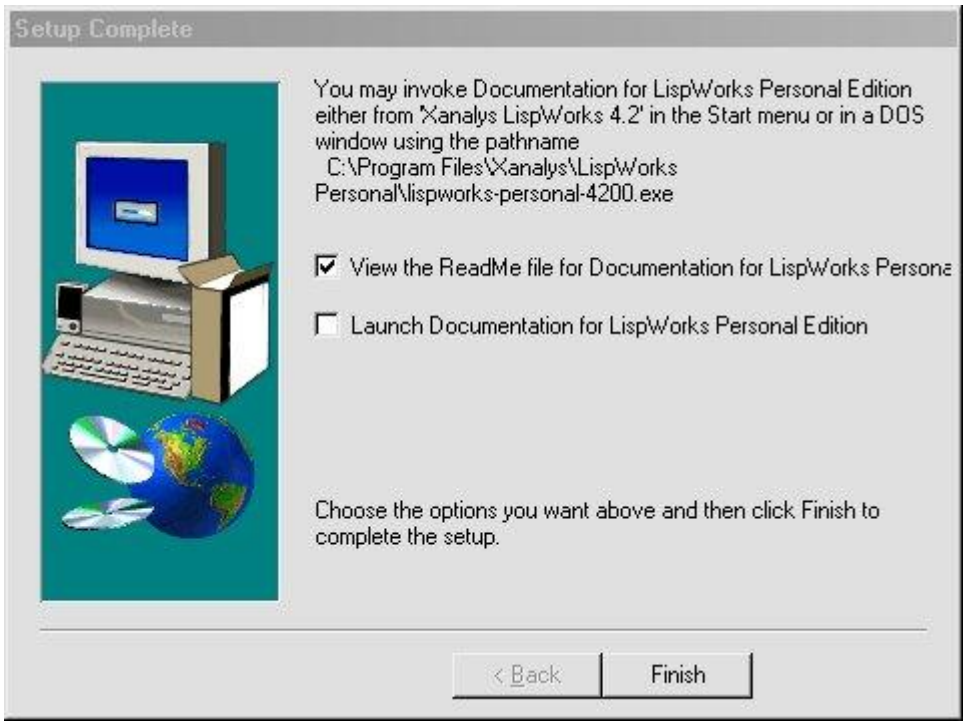


وكما تلاحظ فأنه سيتم وضع الوثائق والبرنامج في نفس المجلد في قائمة ابدأ.

8- هذه النافذة تعرض لك خيارين (وكلاهما غير مهم لتحميل البرنامج) :

الأول: يعطيك خيار عرض ملف ReadMe الذي يعطي معلومات حول النسخة الشخصية من برنامج LispWorks.

والثاني: يعطي خيار وضع أيقونة هذا البرنامج مع البرامج التي لها خاصية الوصول السريع من خلال الشريط Quick Launch الموجود على شريط المهام.



اختر منها ما يلائمك أو اتركها جميعا ثم انقر على Finish.

من المهم جدا أن نتعرف على البيئة التي سنعمل عليها قبل البدء في الدروس الفعلية للبرمجة.. هذا سيساعدني كثيرا على شرح لغة Lisp وأنا مطمئنة بأنكم ستستطيعون تطبيق الدروس والتعامل مع الأكواد بأفضل وجه ممكن.

قد تصادفون بعض الأكواد التي استخدمتها للتطبيق على بعض الأدوات، فلا تشغل بالك كثيرا بمعنى وطريقة كتابة الكود، لأنني واثقة بأنها ستكون مفهومة لك بإذن الله في الدروس القادمة بإذن الله والتي ستشرح لك بالتفصيل هذه اللغة الجميلة والذكية بحق!
كل ما أريده منك الآن أن تكتسب من هذه الدروس بعض الخبرات والمهارات وأن تتعرف وتعتاد على شكل بيئة LispWorks التي ستعمل عليها، وسيكون للبرمجة نصيب الأسد في السلسلة القادمة من الدروس بإذن الله :).

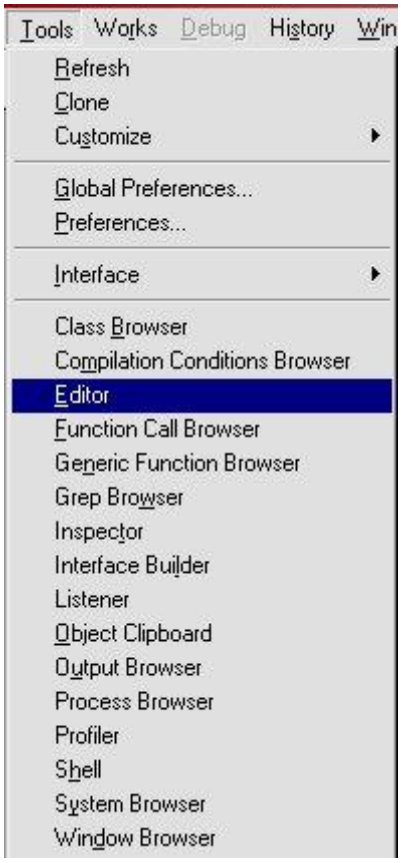
سنتعرف في هذا الدرس على الأدوات المتخصصة داخل برنامج LispWorks..

لماذا يهمنا معرفة هذه الأدوات؟

لأن هذه الأدوات تعمل على توفير بيئة مناسبة لإنشاء وتطوير الأكواد باستخدام لغة Lisp Common.

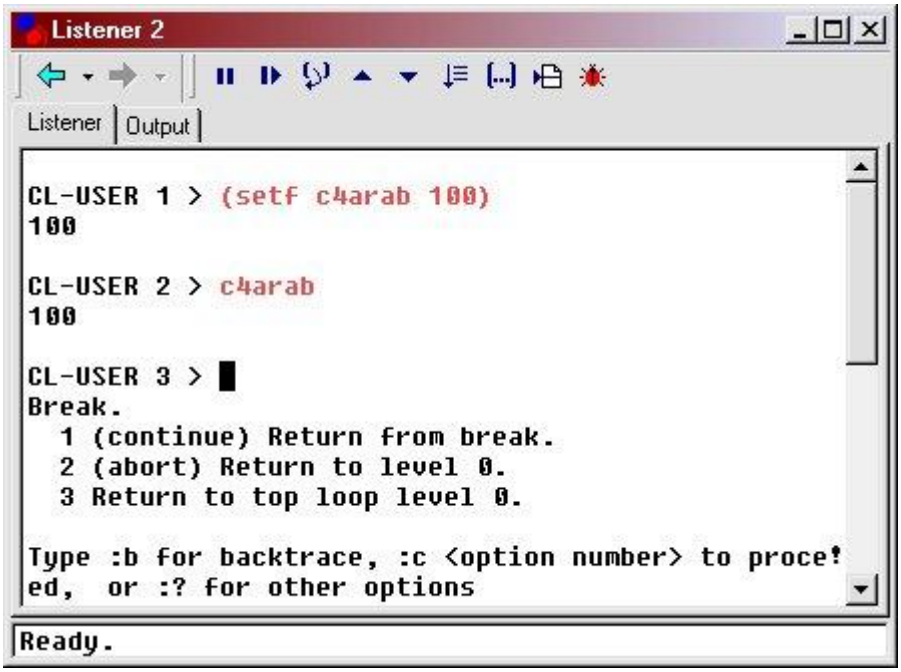
كيف سنصل لهذه الأدوات؟

يمكنك اختيار أي أداة من هذه الأدوات للعمل عليها من القائمة Tools.



والآن سنتناول أهم الأدوات الموجودة في القائمة أعلاه :

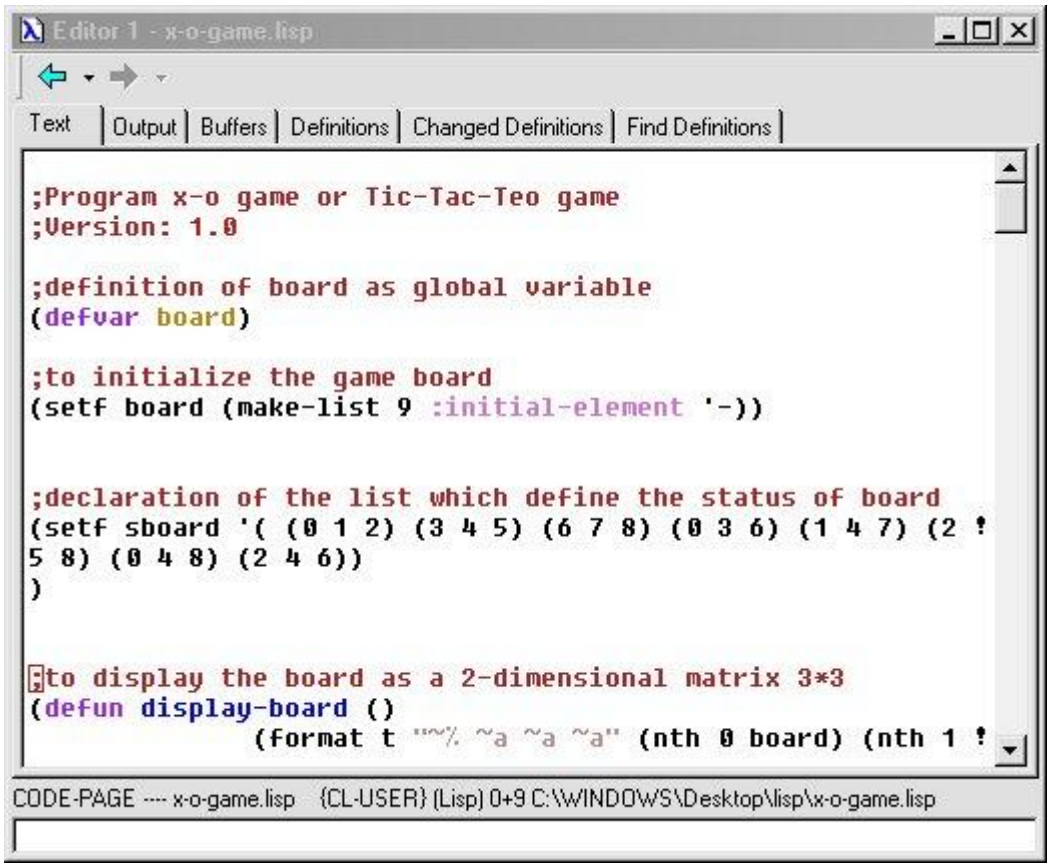
(1) - The listener



من أهم الأدوات التي يوفرها LispWorks وهذه الأداة مصممة خصيصا لاختبار الكود الذي تكتبه أولا بأول دون الحاجة لعمل ترجمة Compilation أو تقدير وتقييم Evaluation للكود ككل.

وتظهر هذه الأداة بمجرد تشغيل برنامج LispWorks.

(2) - The editor

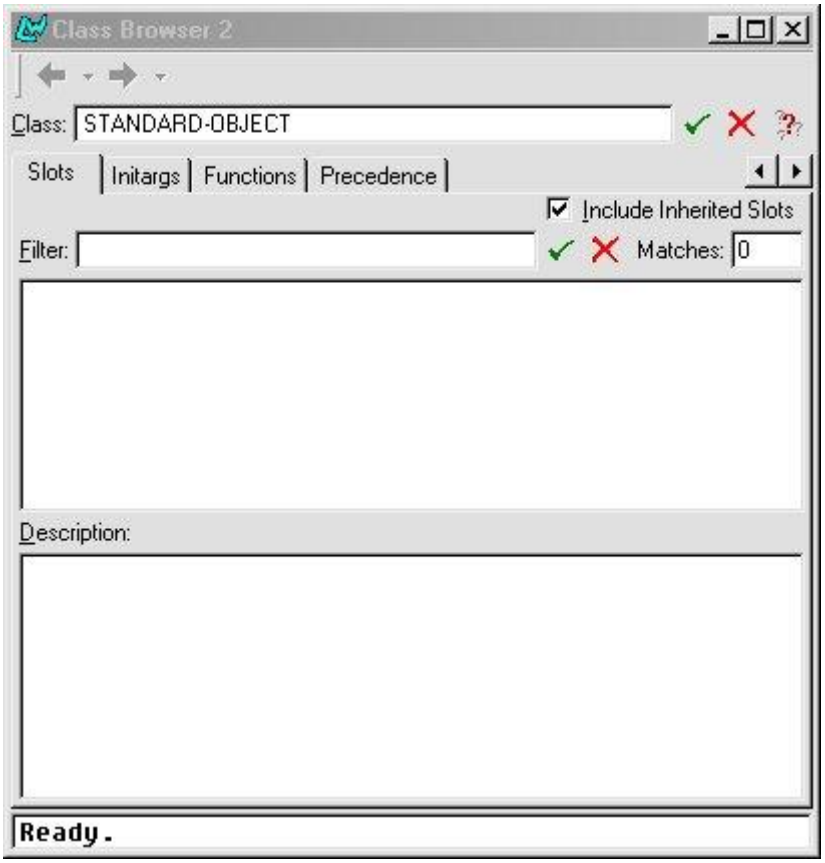


محرر نصوص مبني داخل برنامج LispWorks ليساعدك على بناء أكوادك وبرامجك باستخدام لغة Common Lisp.

هذا المحرر مزود بعدد كبير جدا من الدوال المصممة خصيصا لكي تساعدك على:

- **تحرير وكتابة أكواد Lisp Common .**
- **جعل الكود المكتوب داخل هذا المحرر متفاعلا مع بقية الأدوات وأهمها الـListener مما يساعد على سرعة وسهولة اختبار الكود أو استدعاء الدوال المكتوبة داخل المحرر.**

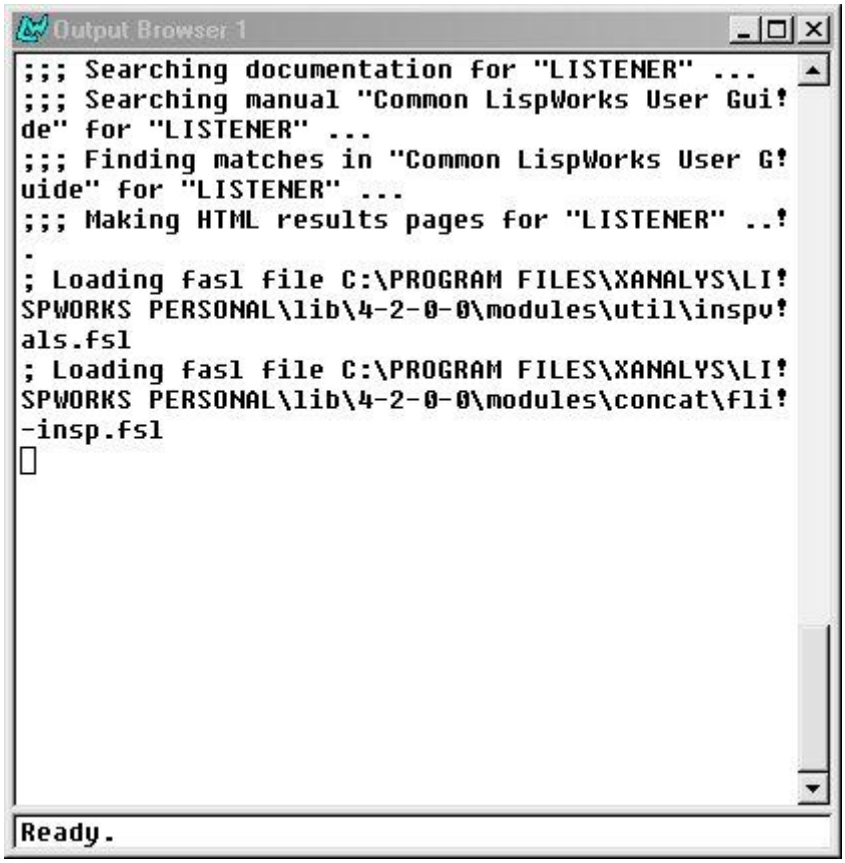
(3) - The class browser



هذه الأداة تتيح لك اختبار الفئات Classes المكتوبة باستخدام لغة Common Lisp والتي تم تعريفها داخل برنامجك.
تستطيع عن طريق هذه الأداة مشاهدة الفئات العلوية superclasses والفئات الفرعية subclasses لكلاس معين كما يمكنك مشاهدة العلاقات Relationships الموجودة فيما بينهم.

بالإضافة إلى أنه يمكنك عن طريق هذه الأداة اختبار الدوال functions والطرق methods الموجودة داخل كلاس معين.

(4) - The output browser

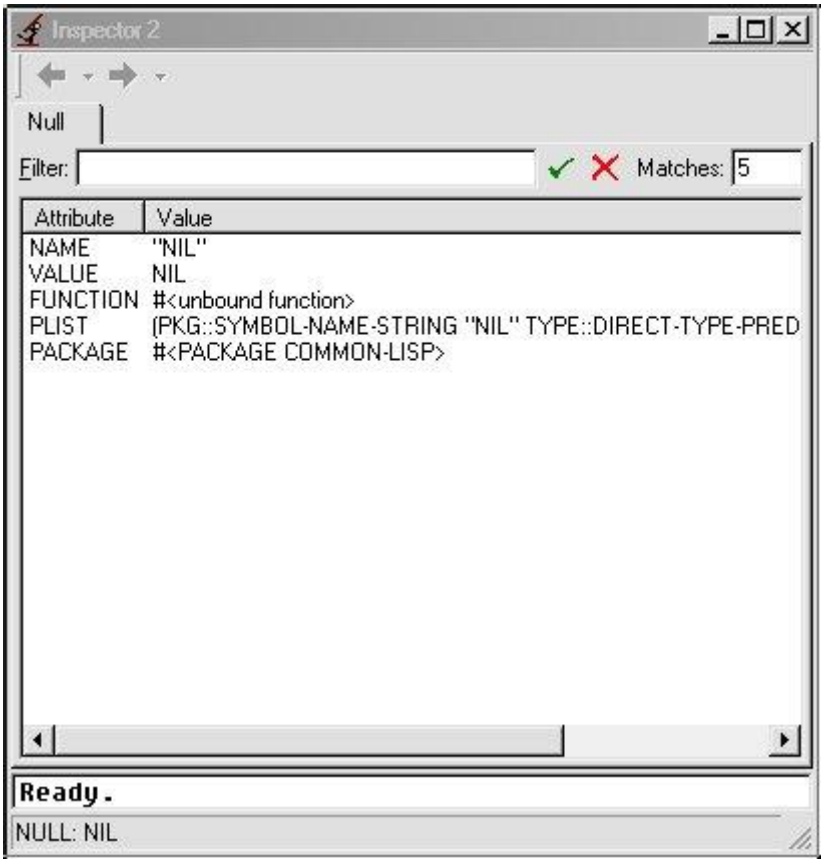


هذا المستعرض يجمع ويعرض جميع المخرجات التي تنتج من البيئة التي تعمل عليها، فإذا كنت تعمل مثلا على الـListener والـEditor Text في نفس الوقت، فان الـ Output Browser سيعرض مخرجات ونتائج كليهما.

ويتضمن ذلك رسائل التحذير والخطأ التي تظهر أثناء عمل ترجمة Compilation للكود، وكذلك كما تلاحظ في الصورة تتضمن النافذة المخرجات التي تنتج من تتبع مسار الدوال التي استخدمت خلال بيئة العمل على LispWorks.

مع ملاحظة أن معظم الأدوات الأخرى وأهمها الـ Text Editor و الـ Listener تحتوي على مستعرض للمخرجات التي تنتج من استخدام الأداة نفسها.

(5) - The inspector



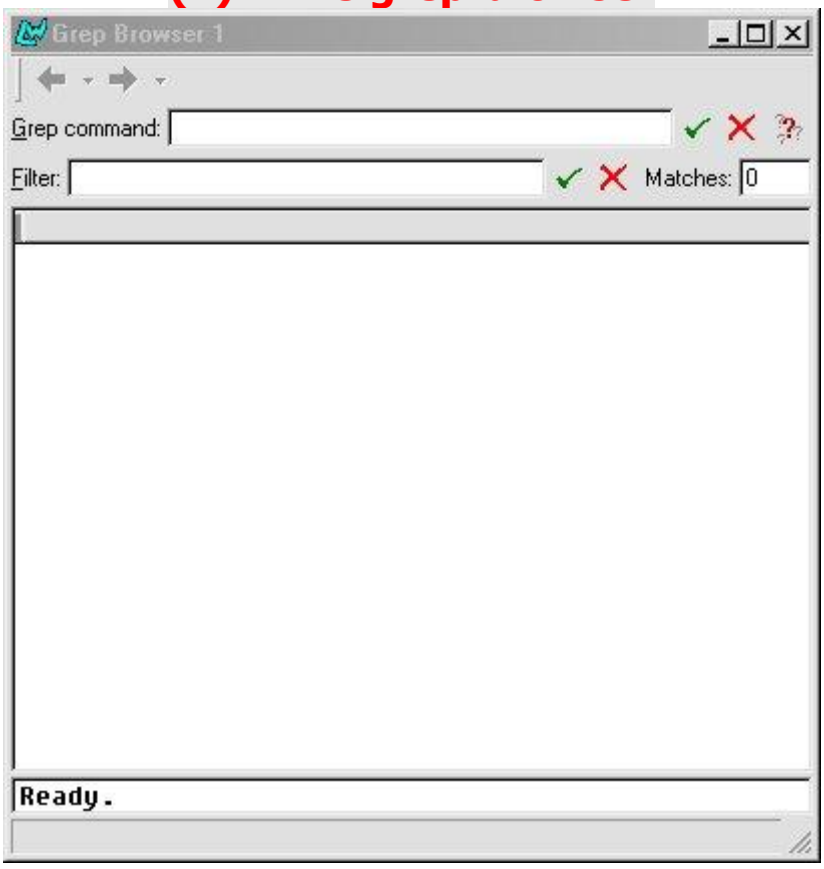
تتيح لك هذه الأداة اختبار وتعديل محتويات كائنات Lisp Common.
هذه الأداة لا تقدر بثمن :) ولها فوائد كبيرة جدا خلال تطوير وبناء برامجك، حيث تتيح للمبرمج فحص واختبار أي جزء من البيانات أثناء مرحلة التنفيذ.

ويمكن أيضا الإطلاع على أو تعديل قيم أي جزء من هذه البيانات إذا دعت الضرورة إلى ذلك أو تعديل قيم البيانات من أجل اختبار التأثيرات التي تنتج من هذه القيم، وذلك تفاديا لإحداث أي تغيرات في الكود الأصلي Code Source قبل تجربتها.

(6) - The shell tool

أداة شل تسمح لك بإعادة تسمية أو نسخ الملفات وكذلك بعض العلميات التي يقوم بها نظام التشغيل على هذه الملفات من خلال بيئة LispWorks.

(7) - The grep browser

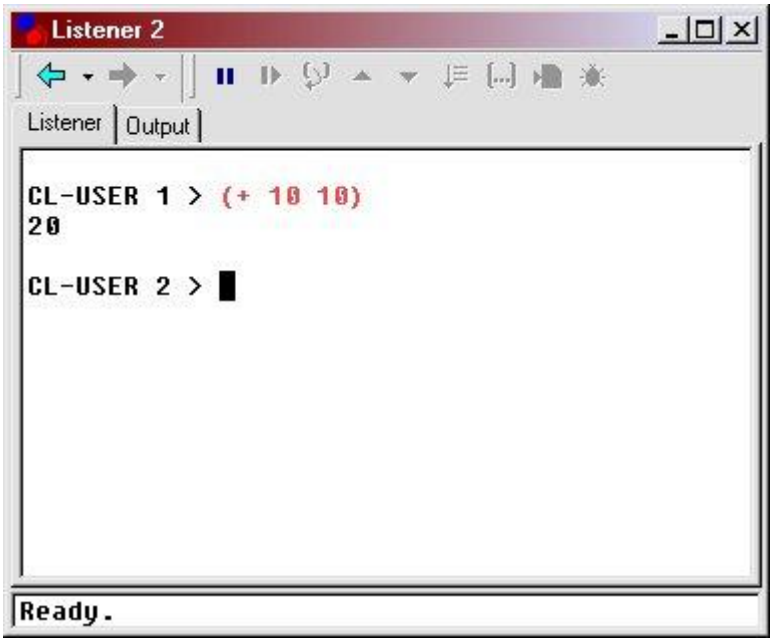


يستخدم مستعرض قرب من أجل البحث عن الملفات التي تحتوي على أشياء محددة في الفهرس الحالي الذي نعمل عليه.

(8) - The object clipboard

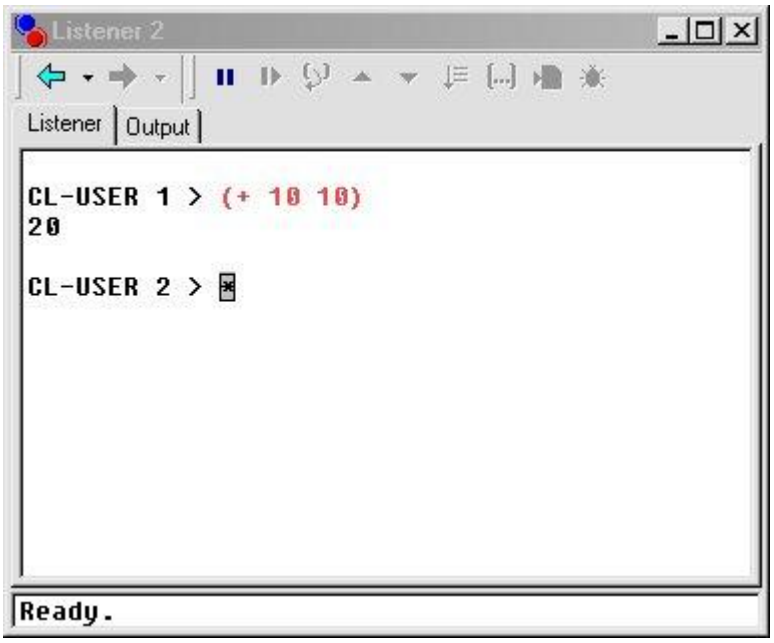
تستخدم هذه الأداة لإدارة العديد من الكائنات المنسوخة.
وكمثال لنسخ أحد كائنات Lisp إلى هذه الأداة، يتم إتباع الخطوات التالية:

- قم بعمل تقييم وتقدير Evaluation لأحد التعبيرات الرياضية المكتوبة بلغة Lisp.
  خذ المثال البسيط التالي لإضافة 10 إلى 10، الموجود بالصورة:

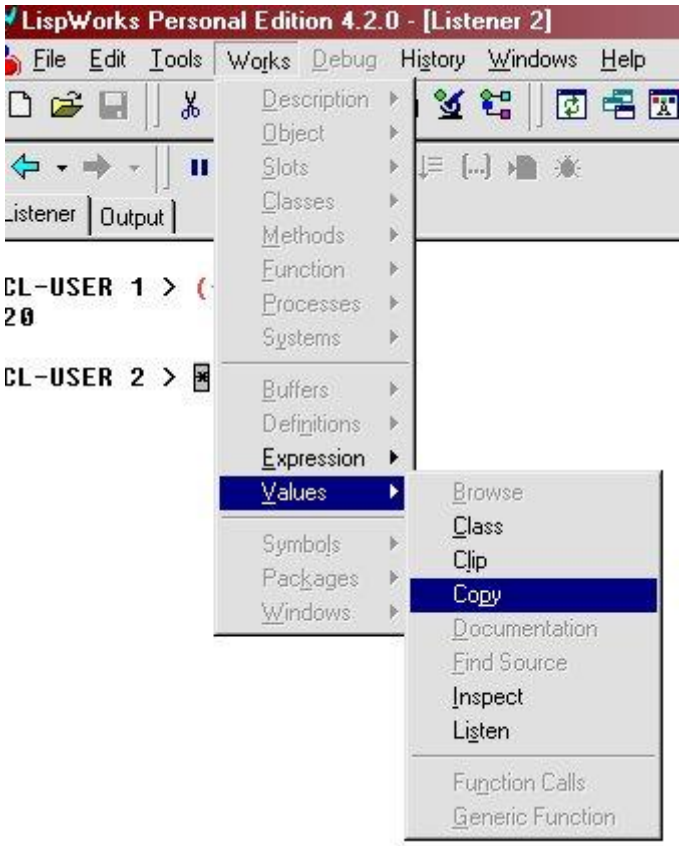


- أمام علامة الاستعداد Prompt الخاص بـ LispWorks اكتب علامة النجمة * ولا تضغط على مفتاح Enter.

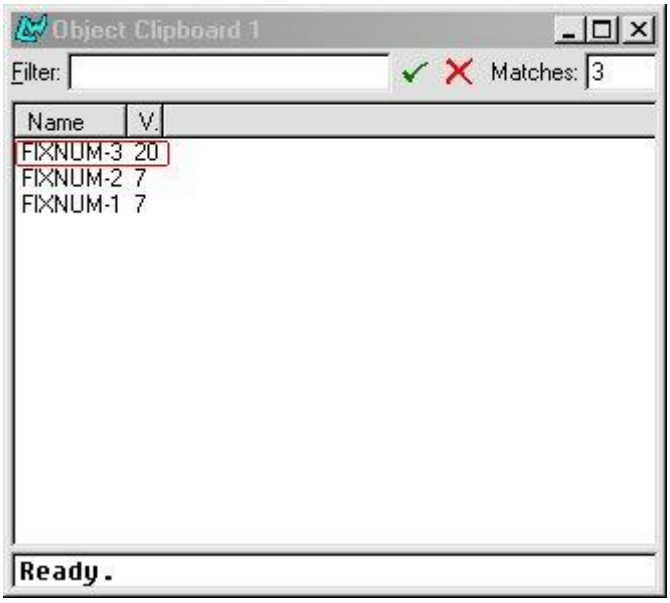
- استخدم الفأرة لتظليل علامة النجمة الموجودة في نافذة الـ Listener، كما بالصورة:



- من القائمة Works --> اختر Values --> ثم Clip

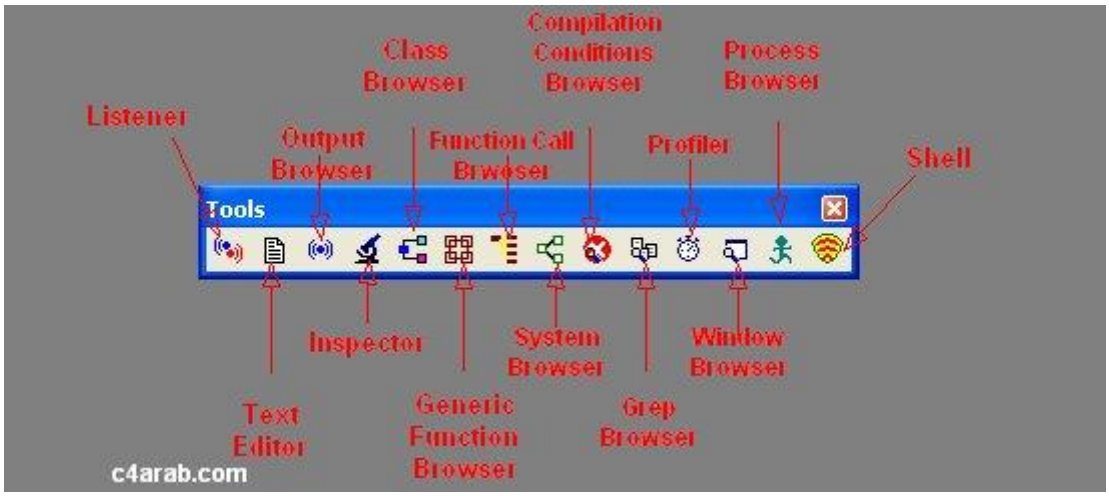


- **والآن قم بفتح أداة Clipboard Object من قائمة أدوات Tools وستجد أن ناتج التعبير الحسابي أي حاصل الجمع 20 تم الاحتفاظ به فيها.**



تلميحة :

يمكن الوصول لأي أداة من هذه الأدوات من شريط الأدوات Tools كما هو واضح من الصورة:



ما هو الـ Text Editor؟

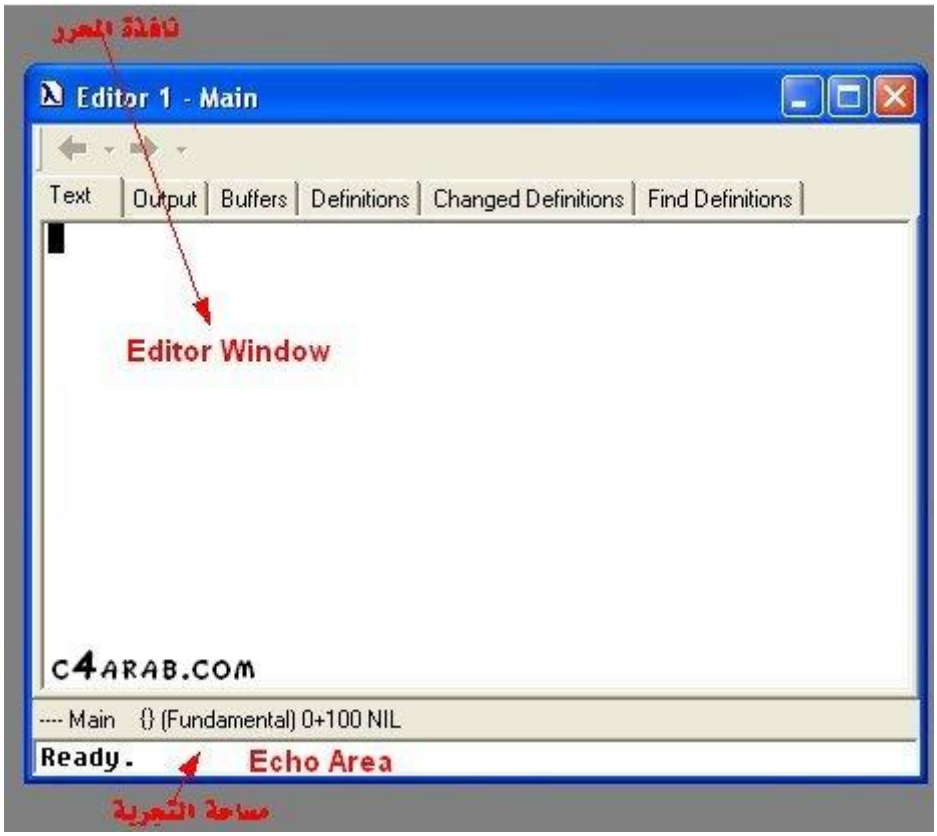
عبارة عن محرر نصوص مضمن داخل برنامج LispWorks من أجل تسهيل عملية كتابة وتحرير أكواد الـ Lisp.

المميزات التي يتسم بها الـ Text Editor هي كالتالي:

- احتوائه على عدد من القوائم التي تظهر عن الضغط بالزر الأيمن على المساحة البيضاء Area Text الموجودة داخله، هذه القوائم تحتوي على عدد كبير من الخيارات التي يمكن استخدامها والاستفادة منها عند التعامل مع الـText Editor.

- احتوائه على واجهة مكونة من عدد من الأدوات الأخرى من مثل الـOutput Browser لاستعراض نتائج ومخرجات الكود المكتوب على هذا المحرر.

نافذة الـ Text Editor:



كما تلاحظ من الصورة، تحتوي نافذة الـ Editor Text على الآتي:

- **Text --> ويسمح بقراءة وتحرير الملفات التي يتم الاحتفاظ بها على أحد الفهارس على الجهاز.**

- **Output --> ويسمح باختبار و الإطلاع على رسائل المخرجات.**

- **تستطيع تحرير العديد من الملفات في نفس الوقت وعلى نفس المحرر، وبالتالي فان الـBuffer --> يساعد على على استعراض الملفات المفتوحة.**

- **Definitions --> يساعد على مشاهدة الفئات Classes والدوال Functions والماكرو Macros والمتغيرات Variables التي تم تعريفها في الملف الحالي.**

- **قد تحتوي الملفات على الكثير من التعريفات Definitions وبالتالي فان Find Definitions --> يساعد على البحث عن تعريف معين موجود خلال هذه الملفات.**

بالنسبة للـ Area Echo الموجودة أسفل النافذة، فهي مساحة صغيرة معدة لأجل سرعة تجربة أحد الدوال أو المتغيرات دون الحاجة لعمل Evaluation للكود ككل أو استخدام الـListener من أجل التجربة.

ملاحظة:
سنجد أيضا أن الـEcho Area متوفر أيضا في أداة الـ Listener التي ستعرف عليها بالتفصيل في الدرس القادم.

كيف نستفيد من الـ:Echo Area

افترض أنك تريد أثناء العمل على الكود معرفة قيمة أحد المتغيرات الموجودة في الكود وليكن المتغير Var على سبيل المثال:

- **ببساطة قم بالضغط على ESC مرتين وستلاحظ كتابة Eval: في الـEcho Area مما يدل على الاستعداد للحساب أو التقدير Evaluation**

- **ثم اكتب Var في الـEcho Area واضغط على Enter وستظهر لك قيمة المتغير.**

افترض مثلا أنك قمت بتعريف دالة بالاسم square ، وتستقبل عدد واحد، وتعود بمربع هذا العدد ، لتجربة هذه الدالة بسرعة:

- **قم بالضغط على ESC مرتين وستلاحظ كتابة Eval: في الـArea Echo مما يدل على الاستعداد للحساب أو التقدير Evaluation**

- **ثم اكتب ((4 square في الـEcho Area واضغط على Enter وسيظهر لك الناتج 16 (مربع العدد 4) طبعا في حال كانت الدالة تعمل بشكل صحيح .**

لا تقلق فأنا أريدك فقط أن تكتسب خبرة عن بعض المهارات وعن الأدوات المستخدمة في البيئة التي ستعمل عليها..
صدقني هذا سيساعدني كثيرا بإذن الله على شرح الأكواد البرمجية وأنا مطمئنة أنك ستتمكن من تطبيقها بأفضل وجه ممكن..

أما بالنسبة لطريقة كتابة أكواد بلغة Lisp Common فستكون لها دروس كثيرة وعديدة وسنشرح فيها كل شيء يساعدك على أن تصبح محترفا ومتمكنا في Lisp بإذن الله.

كيف نفتح المحرر Text Editor؟

لفتح محرر جديد اختر -- Editor <Tools

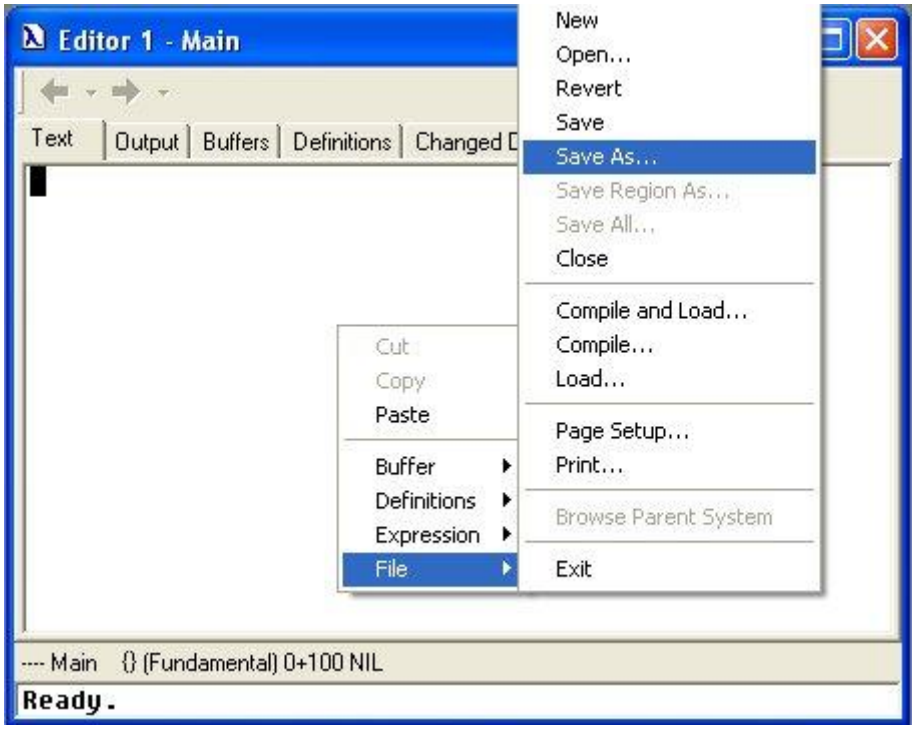
ولفتح ملف موجود اختر -- Open <File أو قم بالضغط مباشرة على الأيقونة 

وسيتم عرض المحرر على الواجهة الافتراضية والتي ستخدمها بشكل رئيسي ألا وهي واجهة عرض النصوص Text.

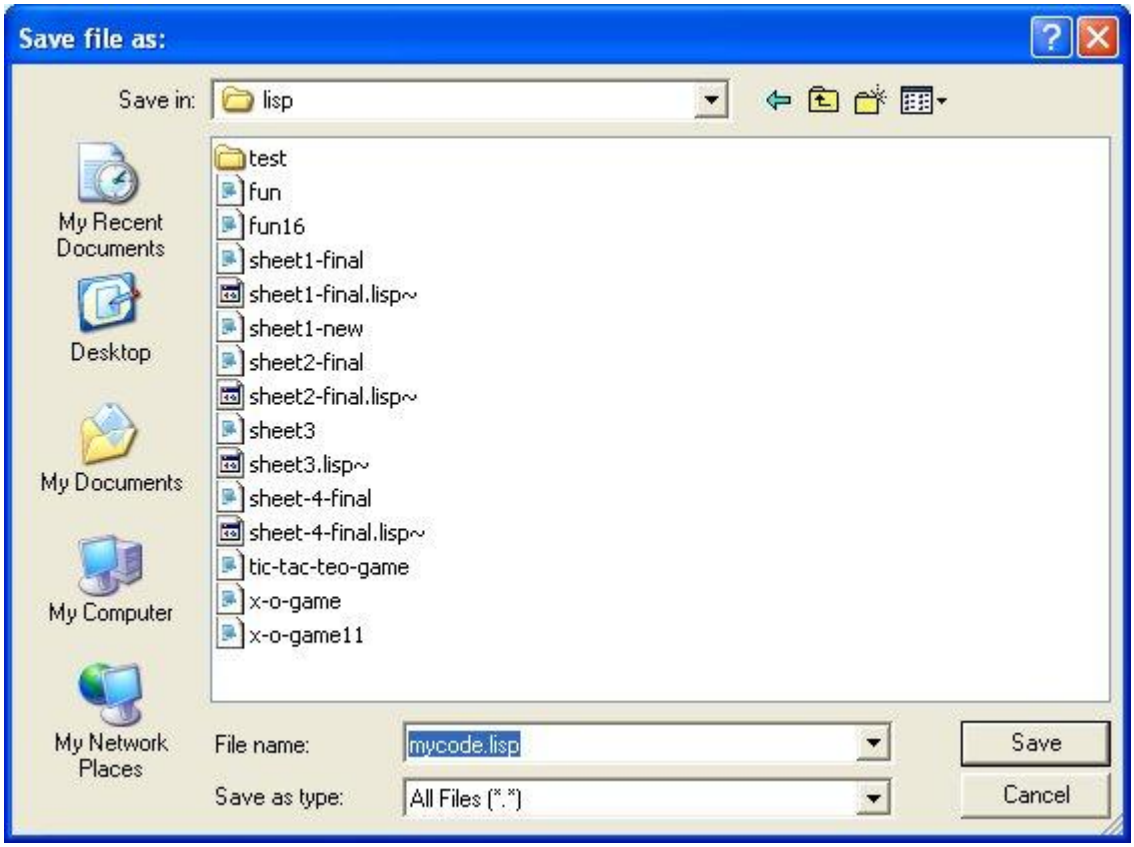
عرض وتحرير الملفات:

1- أفضل طريقة لحفظ وتلوين الكود على المحرر هي حفظه بامتداد .lisp ، ولعمل ذلك اتبع الخطوات التالية:

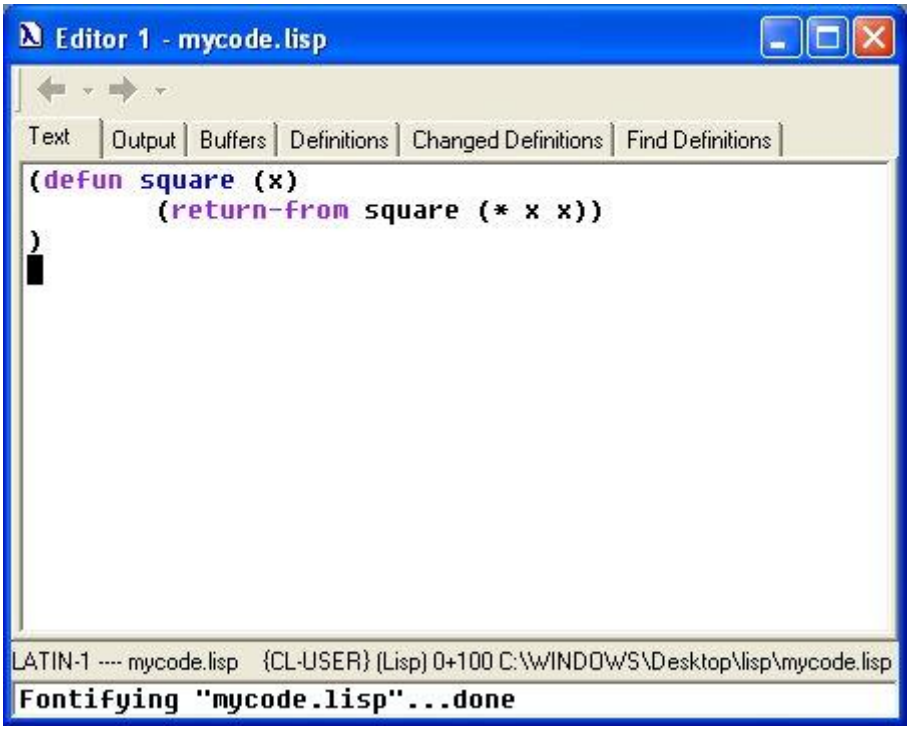
1.1 قم بفتح المحرر ثم اضغط بالزر الأيمن ثم -- File> As Save



1.2 في خانة اسم الملف File Name قم بكتابة اسم الملف بامتداد .lisp وليكن هذا الاسم على سبيل المثال: mycode.lisp ، واجعل خانة حفظ كنوع Save as type على الخيار جميع الملفات All Files (*.*).



1.3قم بكتابة الكود داخل المحرر، ثم اختر file --> Save لحفظ التغيرات، وستجد أن الكود سيكون ملونا بهذا الشكل:



محتويات الـText Editor الأخرى:

output

كيف نشاهد المخرجات من كود تم تحريره على هذه الأداة ؟!

عند القيام بتشغيل الكود المكتوب في المحرر فانه يتم عرض المخرجات والنتائج في هذه النافذة.

خطوات عمل Evaluation للكود عن طريق المحرر ومشاهدة المخرجات:

1- قم بكتابة الكود داخل المحرر أو قم بفتح ملف موجود مسبقا.
2- اضغط بالزر الأيمن للفأرة على أي مساحة خالية على المحرر ثم اختر Buffer ثم Evaluate.
3- اضغط على الـ Output Tap لمشاهدة النتائج أو المخرجات.

والصورة التالية توضح نتائج تشغيل أحد الأكواد:

وكما تلاحظ في الصورة السابقة، تتضمن المخرجات رسائل التحذير Warning Massages ، وتتضمن أسماء المتغيرات وأسماء الدوال التي تم تعريفها بنجاح وبدون أخطاء داخل المحرر.

لو فرضنا أنه يوجد خطأ في الكود فإن رسالة خطأ Error Massage ستظهر لك مبينة سبب الخطأ.

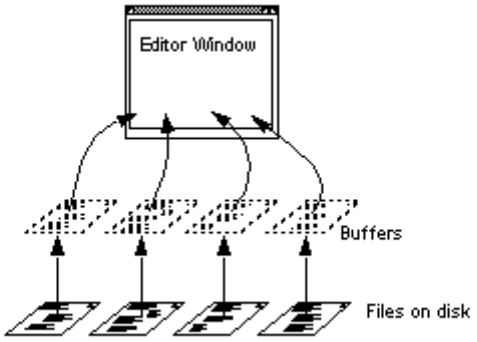
Buffers

فتح أكثر من ملف على الـEditor وطريقة التبديل بين الملفات:

يسمى الملف الذي قمت بفتحه للتو على الـEditor بالـBuffer ، وعند الكتابة أو التعديل في محتويات ملف ما

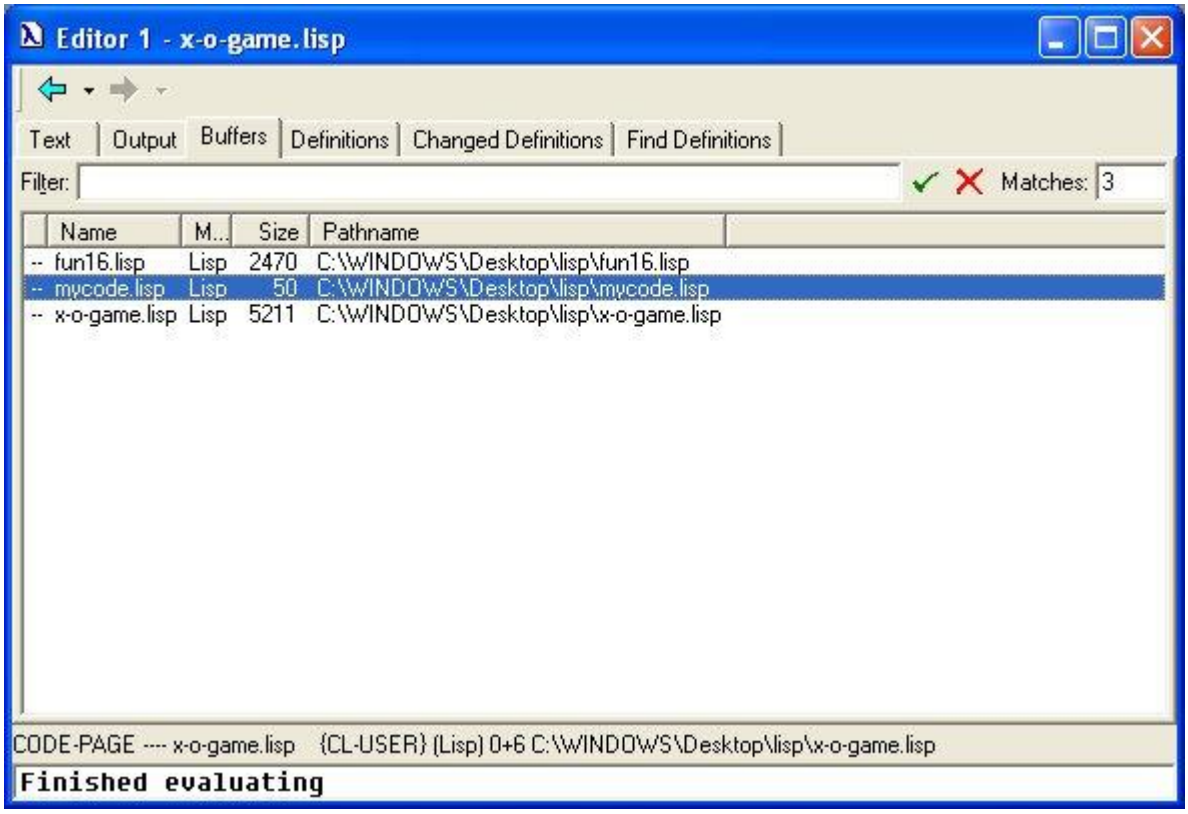
فسيتم التغيير فقط في الـBuffer الذي أمامك، ولحفظ التغيرات علك الضغط بالزر الأيمن على الـEditor ثم اختيار File --> Save As إذا كنت تحفظ الملف لأول مرة أو اختيار File --> Save إذا كان الملف موجود مسبقا.

لاحظ الصورة لتفهم أكثر ما قصدته بالعبارات السابقة:



وتستطيع من خلال الـEditor فتح أكثر من ملف للعمل عليه، ولكن لا يسمح لك الـEditor بفتح أكثر من نافذة فيه، وبالتالي فانك ستتمكن من التحرير والكتابة على أحد هذه الملفات، وإذا أردت التبديل بين هذا الملف وملف مفتوح أيضا (ولكن غير ظاهر لك) قم بالضغط على Buffers ثم انقر مرتين على اسم الملف.

كما تلاحظ إذن يوفر لك الـBuffers View قائمة بجميع الملفات التي فتحها من خلال الـEditor ، فمثلا قمت بفتح ثلاث ملفات fun16.lisp و mycode.lisp و x-o-game.lisp كما بالصورة:



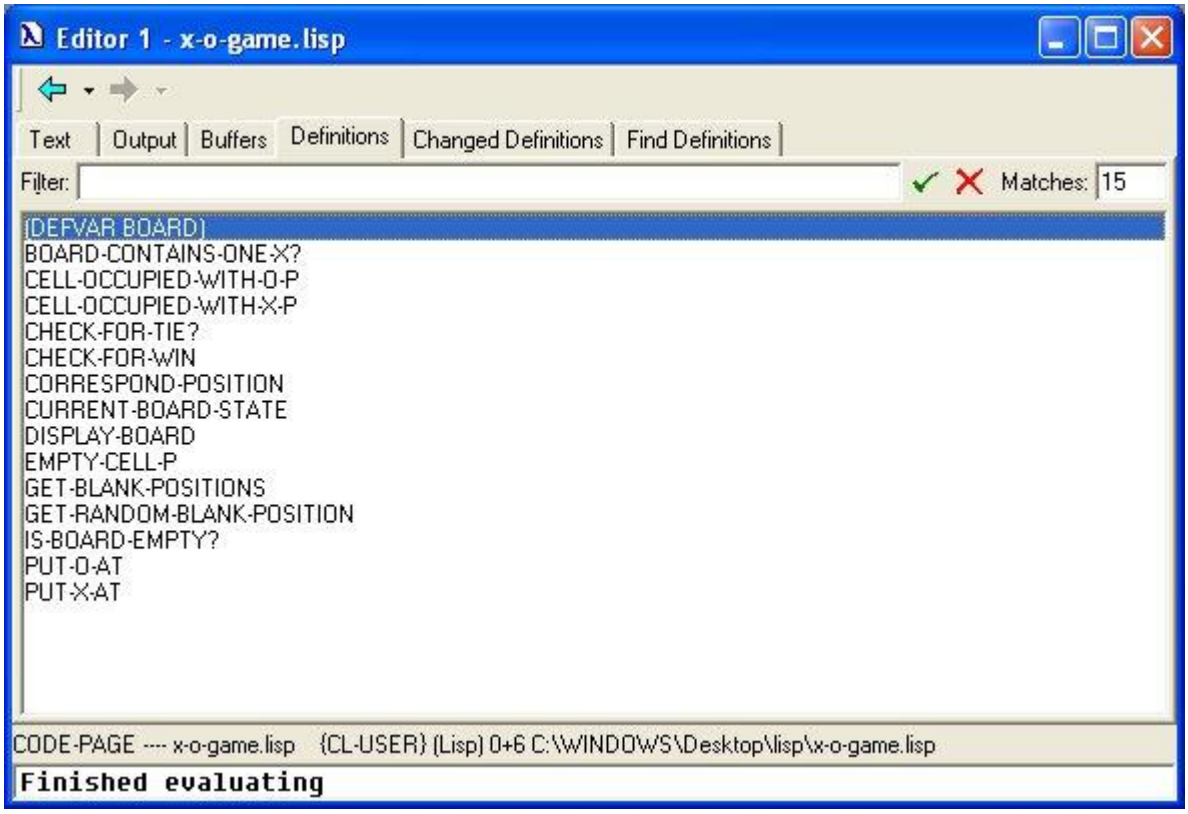
أستطيع الآن بعد أن كنت أعمل على ملف x-o-game.lisp (ظاهر في شريط العنوان)، العمل على الملف mycode.lisp بالنقر المزدوج عليه من نافذة Buffers.

Definitions

كيف تعرف أسماء المتغيرات والدوال الموجودة في الكود؟

ببساطة قم بالضغط على الـDefinitions Tap وستجد قائمة بكل الدوال والمتغيرات التي تم تعريفها في الكود المكتوب في الـEditor.

ملاحظة: لن تظهر هذه القائمة إلا بعد أن تقوم أولا بعمل Evaluation للكود كما تعلمنا سابقا:).



إذا قمت بالنقر المزدوج على اسم أحد الدوال فانه سيتم العودة بك للكود مع تظليل الجزء الخاص بالتعريف الدالة نفسها.

Changed definitions

يقوم هذا الجزء بعرض قائمة بأسماء المتغيرات والدوال التي تم تعريفها في الملف المفتوح حاليا والتي لم تكن معرفة ومكتوبة عند فتح هذا الملف.

definitions Finding

تستخدم هذه الأداة للبحث عن متغير أو دالة تم تعريفها خلال الكود باسم معين. فمثلا لو أردت البحث عن متغير Var قمت بتعريفه في أحد أسطر الكود اكتب فقط Var أمام الحقل Name ثم انقر Enter.

خواص أخرى للمحرر:

1- طباعة ملف:

- يمكن ببساطة طباعة أحد الملفات المفتوحة على الـEditor بالضغط على أي مكان في المحرر بالزر الأيمن ثم اختيار File --> Print ..
- يمكن ببساطة اختيار print من القائمة File.

2- للقيام لعمليات القص والنسخ واللصق:

قص:
قم ببساطة باختيار Cut من القائمة Edit.
أو باستخدام الاختصار: Ctrl+W

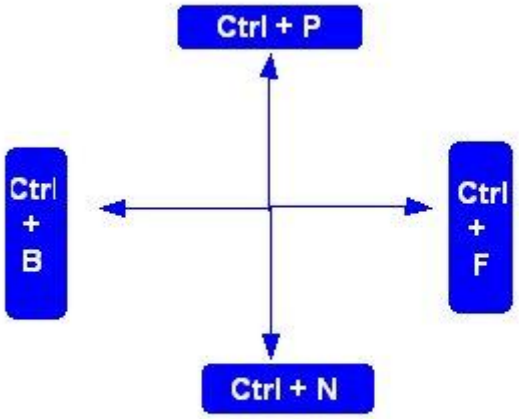
نسخ:
قم ببساطة باختيار Copy من القائمة Edit.
أو باستخدام الاختصار: Alt+W

لصق:
قم ببساطة باختيار Paste من القائمة Edit.
أو باستخدام الاختصار: Ctrl+Y

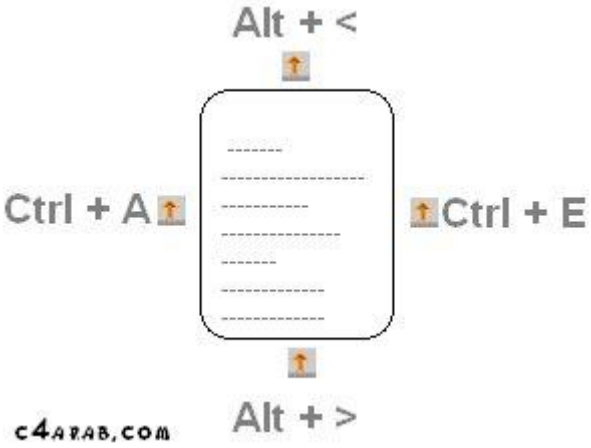
أو ستظهر لك هذه الخيارات مباشرة عند النقر بالزر الأيمن على المحرر نفسه.

3- التنقل خلال أحد الملفات:

يمكن ببساطة التنقل خلال أحد الملفات وتغيير موقع المؤشر Cursor Position عن طريق استخدام اختصارات لوحة المفاتيح كما هو موضح بالصورة التالية:

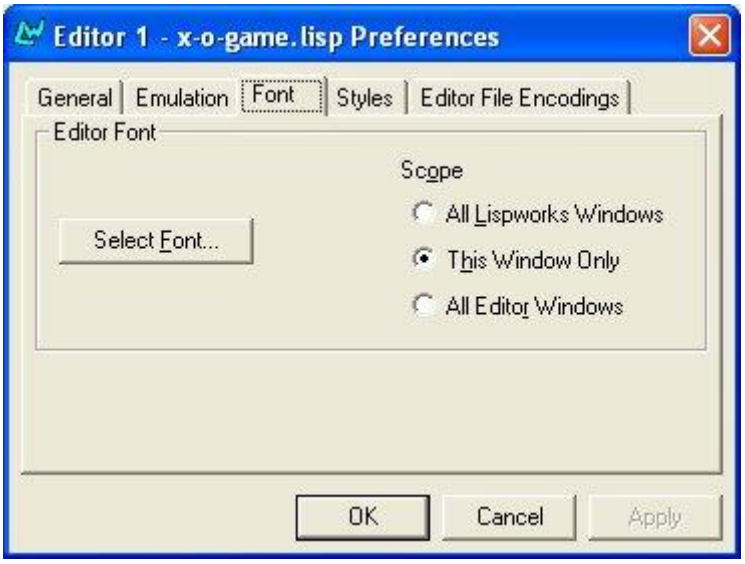


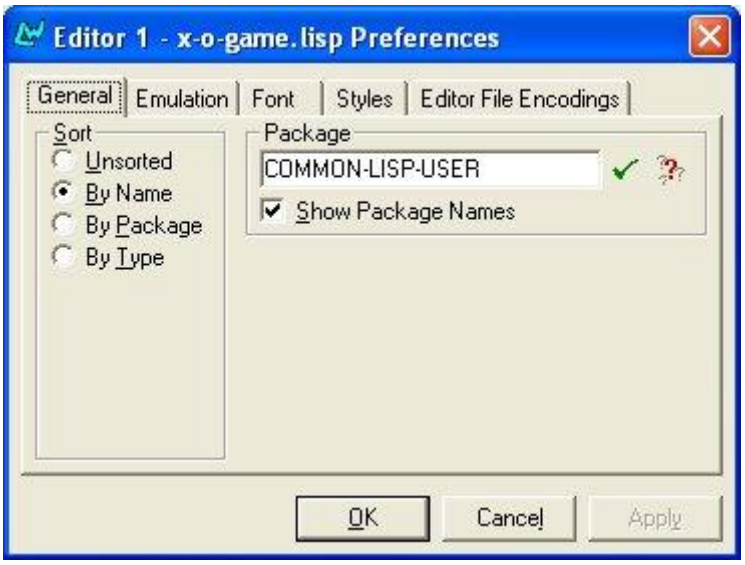
وللتنقل إلى بداية ونهاية أي سطر أو بداية ونهاية الـBuffer استخدم الاختصارات الموضحة بالصورة:

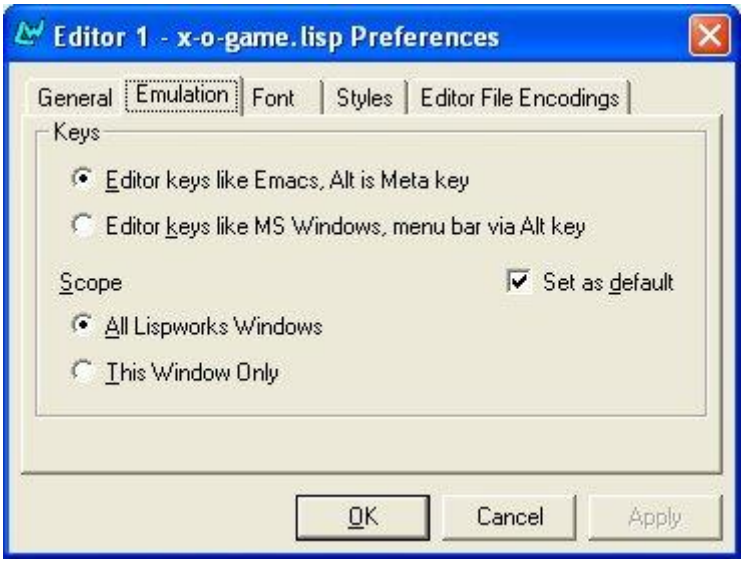


4- كيف يتم تغيير إعدادت الـEditor ؟

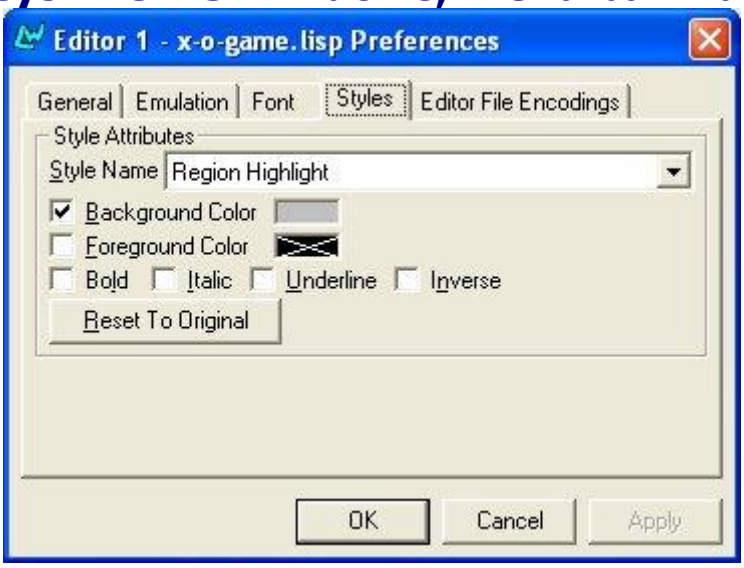
يمكنك تغيير الاعدادات الافتراضية للـEditor من خطوط وألوان وخلفية وخصائص أخرى عن طريق الخيار Preferences من القائمة Tools.

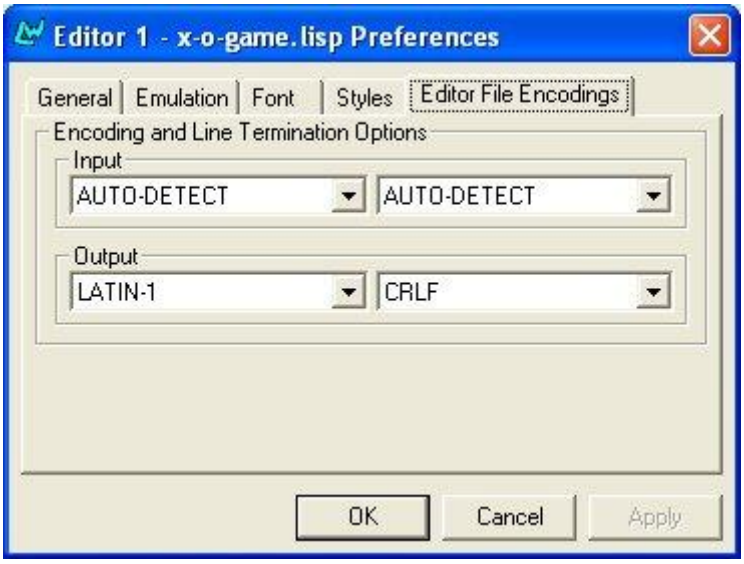

Editor 1 - x-o-game.lisp Preferences
General
Emulation
Font
Styles
Editor File Encodings
Editor Font
Select Font...
Scope
All Lispworks Windows
This Window Only
All Editor Windows
OK
Cancel
Apply


Editor 1 - x-o-game.lisp Preferences
General
Emulation
Font
Styles
Editor File Encodings
Sort
Unsorted
By Name
By Package
By Type
Package
COMMON-LISP-USER
Show Package Names
OK
Cancel
Apply


Editor 1 - x-o-game.lisp Preferences
General
Emulation
Font
Styles
Editor File Encodings
Keys
Editor keys like Emacs, Alt is Meta key
Editor keys like MS Windows, menu bar via Alt key
Scope
Set as default
All Lispworks Windows
This Window Only
OK
Cancel
Apply

ملاحظة: إذا كنت معتادا على اختصارات ميكروسوفت ويندوز خصوصا في النسخ واللصق والقص وغيرها وتريد التعامل بها يمكنك تغيير خيار Keys من النافذة السابقة إلى :
Editor Keys like MS Windows, menu bar via Alt key





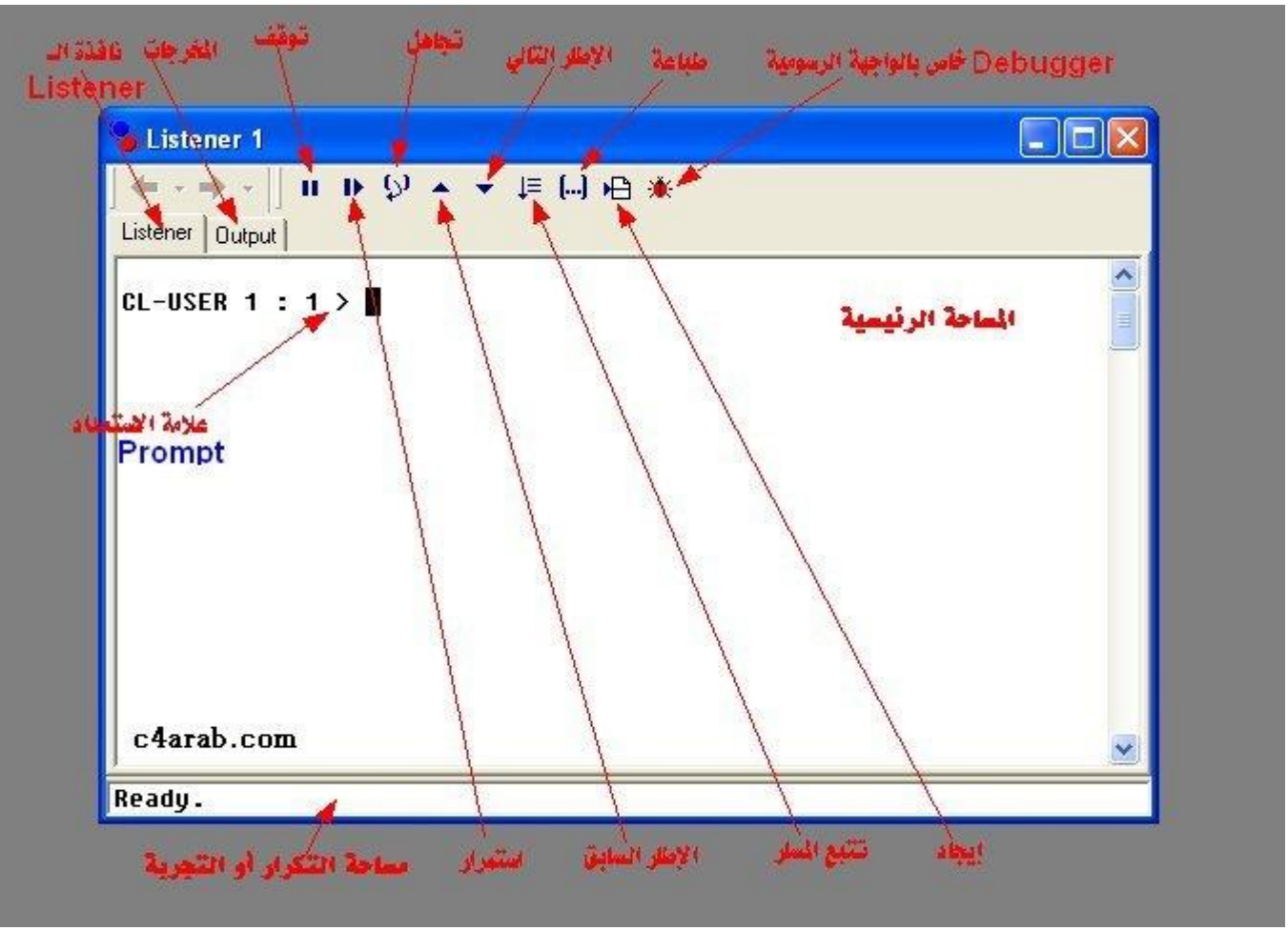
ما هي أداة Listener؟

عبارة عن أداة تسمح بكتابة الكود وتنفيذه مباشرة لمشاهدة النتائج.
تفتح لك هذه الأداة بمجرد الدخول إلى بيئة برنامج LispWorks.

تعتبر هذه الأداة أهم الأدوات على الإطلاق، فلماذا نحتاج لهذه الأداة بشكل كبير؟
من المفيد بالطبع لأي مبرمج تجربة الكود الذي يكتبه جزءا جزاء بدلا من تنفيذه دفعة واحدة، حيث أن تنفيذ الكود دفعة واحدة قد يؤدي لظهور أخطاء كثيرة وقد لا يسهل على المبرمج أن يكتشفها، ولكن باستخدام هذه الأداة التي تتفاعل مع المبرمج ومع الكود حيث تظهر الأخطاء في كل جزء ((Form من الكود، سيتمكن المبرمج بإذن الله من بناء كود كبير جدا وبدون أخطاء.

أهم خصائص الـ Listener :

يتم فتح الـListener بمجرد الدخول إلى بيئة LispWorks ومكونات النافذة تظهر بوضوح في الصورة التالية:



كما نلاحظ أن الـ Listener يحتوي على علامة استعداد Prompt تساعد المبرمج على كتابة كود باستخدام لغة Lisp Common جزءا جزءا، كل جزء أو ما يسمى بالفورم Form يتم كتابته أمام الـ Prompt وبعد الانتهاء والنقر على زر Enter فإنه سيتم عمل تقييم أو تقدير Evaluation لهذا الفورم.

ونتيجة عمل Evaluation في الـListener ستكون أحد أمرين:

1. سيتم ظهور رسالة خطأ على نفس الـListener في حال وجود خطأ في الـForm .
2. ظهور نتيجة العملية الحسابية أو الرياضية أو المنطقية في حال كتابة تعبير لأي منها، أو ظهور اسم الدالة أو المتغير في حال تم تعريف دالة أو متغير بشكل صحيح.

تتميز علامة الاستعداد في الـ Lsitener عن غيرها من علامات الاستعداد التي توجد في الموجهات النصية من مثل الموجودة في موجة أوامر الدوس Dos Prompt بأنها تحمل معلومات مفيدة للمبرمج، فهي تشير إلى عدد الأسطر الموجودة منذ بدء استخدام الـ Listener. بمعنى آخر، تشير إلى نقطة ومكان عمل Evaluation لأي Form في الـListener.

لمشاهدة المخرجات: قم بالضغط على Tap Output.

حساب القيم لنماذج بسيطة باستخدام هذه الأداة:

Evaluating simple forms

سنتناول الآن بعض الأمثلة البسيطة لنتعرف على آلية عمل Evaluating للنماذج Forms التي يتم كتابتها باستخدام هذه الأداة.
لن أفسر المعنى والـStructure البرمجي لها كثيرا فالدروس القادمة بإذن الله ستتناول ذلك بالتفصيل..

مثال1:

1- اكتب الرقم 12 أمام علامة الاستعداد ثم اضغط على Enter

```
CL-USER 1 > 12
12
```

2- اكتب * أمام علامة الاستعداد ثم اضغط على Enter

```
CL-USER 2 > *
12
```

كتابة * تعني إظهار نتيجة التعبير السابق، سيظهر لنا هنا الرقم 12 حيث يعتبر ناتج العملية الحسابية التي تم إجراؤها في الفورم السابق.

3- استخدم جملة الإسناد في Lisp وهي setq لإسناد القيمة 12 إلى متغير ما وليكن x كالتالي:

```
CL-USER 3 > (setq x 12)
12
```

4- لإظهار قيمة المتغير x، اكتب اسم المتغير x أمام علامة الاستعداد ثم اضغط على Enter

```
CL-USER 4 > x
12
```

5- يمكنك أيضا جمع x مع نفسها ثلاث مرات بكتابة الكود كالتالي:

```
CL-USER 5 > (+ x x x)
36
```

مثال2:

الدالة List تعتبر من أهم الدوال في لغة Lisp، حيث تعرف قائمة مكونة من عدد معين من العناصر..

1- لتعريف قائمة مكونة من العناصر من 1 إلى 5 اكتب الكود كالتالي:

```
CL-USER 6 > (list 1 2 3 4 5)
(1 2 3 4 5)
```

وبالتالي فانه تم إنشاء قائمة مكونة من الأرقام من 1 إلى 5.

طريقة أخرى:

يمكن تعريف القائمة بالشكل التالي:

```
CL-USER 7 > '(1 2 3 4 5)
(1 2 3 4 5)
```

أي بوضع العلامة ' قبل القوس بدلا من كتابة الدالة list داخل القوس.

2- يمكن إسناد عناصر القائمة لأي متغير وليكن xlist باستخدام جملة الإسناد setq كالتالي:

```
CL-USER 8 > (setq xlist (list 1 2 3 4 5))
(1 2 3 4 5)
```

أو بطريقة أخرى:

```
CL-USER 9 > (setq xlist '(1 2 3 4 5))
(1 2 3 4 5)
```

3- اكتب xlist أمام علامة الاستعداد لإظهار محتواها:

```
CL-USER 10 > xlist
(1 2 3 4 5)
```

تلاحظ أنه ظهرت لك قائمة مكونة من الأعداد من 1 إلى 5 كقيمة للمتغير xlist.

إعادة حساب قيم نماذج موجودة:

Re-evaluating forms

بفرض أنك قمت بتعريف متغير اسمه var ويأخذ ، يمكنك أن تقوم بتغيير قيمة هذا المتغير والتعامل مع القيمة الجديدة مرة أخرى أثناء العمل على الكود ..

لتتعلم ذلك مارس المهارات البسيطة التالية:

1- لإسناد القيمة الجديدة ولتكن واحد إلى المتغير var قم بكتابة السطر التالي:

```
CL-USER 12 > (setq var 1)
1
```

2- يمكنك الآن التعامل مع القيمة الجديدة، تستطيع على سبيل المثال جمع المتغير مع نفسه ثلاث مرات:

```
CL-USER 13 > (+ var var var)
3
```

وستلاحظ أن نتيجة الجمع 3 ظهرت بمجرد النقر على Enter.

3- لمعرفة قيمة المتغير مرة أخرى، اكتب مباشرة اسم المتغير ثم انقر Enter.

```
CL-USER 14 > var
1
```

تلاحظ أنه تم جمع قيمة var على نفسها ثلاث مرات دون إلغاء القيمة القديمة.

4- إذا أردت إسناد ناتج عملية الجمع إلى المتغير var فقم بكتابة السطر التالي:

```
CL-USER 15 > (setq var (+ var var var))
3
```

5- وإذا أردت التأكد من أنه تم بالفعل إسناد القيمة الجديدة (ناتج عملية جمع المتغير مع نفسه ثلاث مرات) إلى المتغير var نفسه، قم بكتابة var أمام علامة الاستعداد ثم انقر Enter.

```
CL-USER 16 > var
3
```

ستلاحظ بالفعل أنه تم إسناد نتيجة المع 3 إلى المتغير var.

.*.*.*.*.*.*.*.*.*.*.*.*.*.*.

والآن:

هل تمكنت من ممارسة هذه المهارات على الـListener ببساطة؟

هل لديك الآن فكرة مبسطة وواضحة عن بيئة وأدوات LispWorks تؤهلك للبدء بتعلم لغة Common Lisp ؟

تتضمن دوال المقارنة الأنواع الآتية:

- **يساوي، وتكتب =**
- **لا يساوي، وتكتب /=**
- **أكبر من، وتكتب >**
- **أقل من، وتكتب <**
- **أقل من أو يساوي، وتكتب <=**
- **أكبر من أو يساوي، وتكتب >=**

أولا: دالة المساواة =

الشكل العام:

(= numbers)

الوظيفة:

اختبار تحقق شرط المساواة بين رقمين أو أكثر.

- **تعود هذه الدالة بـT (True) إذا كانت جميع الأرقام لها نفس القيمة.**
- **تعود بـnil (False) خلاف ذلك.**

أمثلة:

```
CL-USER 1 > (= 3 3)
T

CL-USER 2 > (= 3 5)
NIL
CL-USER 3 > (= 3 3 3 3)
T

CL-USER 4 > (= 3 3 5 3)
NIL

CL-USER 5 > (= 3 6 5 2)
NIL


CL-USER 6 > (= 3 2 3)
NIL
```

ثانيا: دالة عدم المساواة =/

الشكل العام:

(/= numbers)

الوظيفة:

اختبار تحقق شرط عدم المساواة بين رقمين أو أكثر.

- تعود هذه الدالة بـT (True) إذا كانت جميع الأرقام ليس لها نفس القيمة.
- تعود بـnil (False) خلاف ذلك.

أمثلة:

```
CL-USER 1 > (/= 3 3)
NIL

CL-USER 2 > (/= 3 5)
T

CL-USER 3 > (/= 3 3 3 3)
NIL

CL-USER 4 > (/= 3 3 5 3)
NIL

CL-USER 5 > (/= 3 6 5 2)
T

CL-USER 6 > (/= 3 2 3)
NIL
```

ثالثا: دالة أقل من <

الشكل العام:

(< numbers)

الوظيفة:

اختبار تحقق أن كل رقم أقل من الرقم الذي يليه.

- تعود هذه الدالة بـT (True) إذا كانت الأرقام مرتبه تصاعديا.
- تعود بـnil (False) خلاف ذلك.

أمثلة:

```
CL-USER 1 > (< 3 5)
T

CL-USER 2 > (< 3 -5)
NIL

CL-USER 3 > (< 3 3)
NIL

CL-USER 4 > (< 0 3 4 6 7)
T

CL-USER 5 > (< 0 3 4 4 6)
NIL
```

رابعا: دالة أقل من أو يساوي <=

الشكل العام:

(<= numbers)

الوظيفة:

اختبار تحقق أن كل رقم أقل من الرقم الذي يليه أو يساويه.

- **تعود هذه الدالة بـT (True) إذا كانت الأرقام مرتبه تصاعديا (أو يتساوى اثنين منهما).**
- **تعود بـnil (False) خلاف ذلك.**

أمثلة:

```
CL-USER 1 > (<= 3 5)
T

CL-USER 2 > (<= 3 -5)
NIL
```

```
CL-USER 3 > (<= 3 3)
T


CL-USER 4 > (<= 0 3 4 6 7)
T


CL-USER 5 > (<= 0 3 4 4 6)
T
```

خامسا: دالة أكبر من >

الشكل العام:

(> numbers)

الوظيفة:

اختبار تحقق أن كل رقم أكبر من الرقم الذي يليه.

- **تعود هذه الدالة بـT (True) إذا كانت الأرقام مرتبه تنازليا.**
- **تعود بـnil (False) خلاف ذلك.**

أمثلة:

```
CL-USER 1 > (> 4 3)
T


CL-USER 3 > (> 4 3 2 1 0)
T


CL-USER 4 > (> 4 3 3 2 0)
NIL


CL-USER 5 > (> 4 3 1 2 0)
NIL
```

```
CL-USER 6 > (> 0.0 -0.0)
NIL
```

سادسا: دالة أكبر من أو يساوي >=

الشكل العام:

(>= numbers)

الوظيفة:

اختبار تحقق أن كل رقم أكبر من الرقم الذي يليه أو يساويه.

- **تعود هذه الدالة بـT (True) إذا كانت الأرقام مرتبه تنازليا (أو يتساوى اثنين منهما).**
- **تعود بـnil (False) خلاف ذلك.**

أمثلة:

```
CL-USER 1 > (>= 4 3)
T
```

```
CL-USER 2 > (>= 4 3 2 1 0)
T
```

```
CL-USER 3 > (>= 4 3 3 2 0)
T
```

```
CL-USER 4 > (>= 4 3 1 2 0)
NIL
```

```
CL-USER 5 > (>= 0.0 -0.0)
T
```

قبل البدء:

*** من هذا الدرس فصاعدا سأكتب LISP بدلا من Common Lisp للتبسيط والاختصار. مع العلم أنني لم أتناول إلا الصيغ والدوال الموجودة في لغة Common Lisp القياسية.**

*** بصراحة لم أحب ترجمة المصطلحات الأجنبية الشائعة في هذه اللغة، بل تركتها مثلما هي مع كتابة المرادف العربي إن وجد بجانبها.. والسبب الوحيد لذلك هو أنني أريد تسهيل قراءة المراجع الأجنبية عندما تريد تعلم المزيد عن هذه اللغة فلا يوجد أبدا للأسف مراجع عربية في هذا المجال.. فإذا ما قلت كل شيء بالعربي دون الإشارة إلى المصطلح الأجنبي فان هذا يعني أنه من الصعب وقد يكون من المستحيل الربط بين دروسي هذه وبين ما تقرؤه في المراجع..**

والله من وراء القصد.

المراجع التي اعتمدت عليها في هذه الدروس:

المرجع الأول: الـDocuments الموجودة مع البرنامج.. فهي كتاب متكامل يشرح هذه اللغة وتفاصيلها الدقيقة.. وصانع الشيء خبير به أكثر من غيره :)

المرجع الثاني: كتاب Common Lisp
Computation A Gentle Introduction to Symbolic

المرجع الثالث: Artificial Intelligence
Structures and Strategies for Complex Problem Solving

وكذلك بالإطلاع على عدد كبير من المواقع الأجنبية المتخصصة في هذا المجال.

سندخل اليوم في مقدمة عن الصيغ الرمزية والقواعد اللغوية وبعض الأمثلة البسيطة ولن أتكلم فيه عن دوال بعينها فذلك سيكون في الدروس القادمة إن شاء الله.

الصيغ الرمزية، والقواعد اللغوية

في لغة Common Lisp

تعتمد لغة LISP في صياغتها على ما يسمى بالصيغ الرمزية أو الصيغ التي تستخدم الرموز Symbolic Expressions ويطلق عليها اختصارا s-expressions.

والصيغ الرمزية s-expressions توجد في صورتين:

- إما أن تكون atom
- أو في صورة list

ما هو الـ Atom ؟

يعتبر الـAtom وحدة بسيطة جدا في اللغة بحيث لا يمكن أن يتم تبسيطها لأكثر من ذلك.. يمكن أن يتضمن الـAtom حروف أو أرقام أو رموز.

أمثلة على بعض الـ Atoms :

3.1416

100

x

c4arab

some-name**

nil

ما هو الـ List ؟

عبارة عن سلسلة من الـ Atoms أو من الـ Lists يفصل بينها بالمسافات Blank spaces وتحاط بأقواس دائرية ( ).
فالقائمة list هي عنصر غير بسيط Non-atomic في LISP.

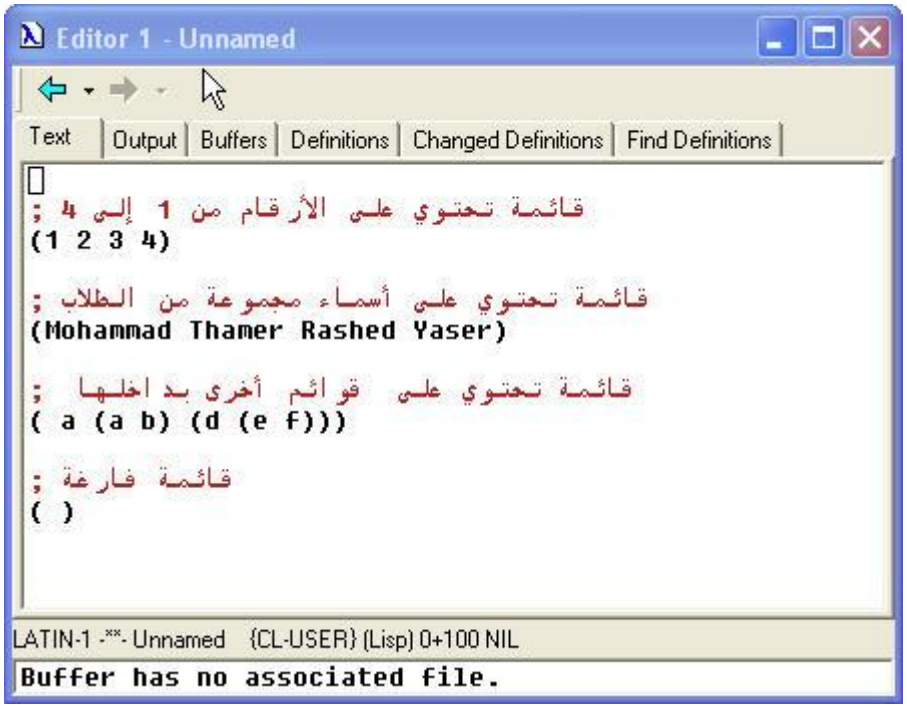
أمثلة على القوائم :Lists

4 قائمة تحتوي على الأرقام من 1 إلى ;
(4 3 2 1)

قائمة تحتوي على أسماء مجموعة من الطلاب ;
Mohammad Thamer Rashed Yaser))

; قائمة تحتوي على قوائم أخرى بداخلها
a (a b) (d (e f))))

; قائمة فارغة
( )



نلاحظ أن الـlist يمكن أن تكون عنصرا في list أخرى..
وهذا الشكل قد يكون مفيد جدا في إنشاء هياكل Structures متداخلة وبأي درجة من التعقيد..

وسنرى أن لذلك فوائد عديدة جدا أثناء التطبيقات والأمثلة وفي المشاريع المتقدمة بإذن الله.

بالنسبة للقائمة الفارغة Empty List فهي تلعب دور خاص عند إنشاء ومعالجة هياكل البيانات Data Structures وتسمى غالبا بالاسم nil في لغة LISP.

نتيجة:

يعتبر nil العنصر الرمزي s-expression الوحيد في لغة LISP الذي يعتبر من كلا النوعين Atom و List في الوقت ذاته!

والآن، كنت قد تحدثت كثيرا في الدروس السابقة عن النماذج Forms وطريقة عمل Evaluation لها، فما هو الـ Form ؟

النموذج أو الـForm ما هو إلا تعبير رمزي s-expression يتم عمل تقييم evaluation له على أداة الـ Listener أو الـ Editor Text للحصول على نتيجة معينة.

إذن فالـ Form قد يكون Atom أو List. وعندا يكون عبارة عن list فان العنصر الأول يعامل كاسم للدالة function name وبقية العناصر تعامل كأنها متغيرات الدالة function arguments.
مثال على ذلك: عندما نكتب قائمة مكونة من الأحرف x, y, z بالشكل التالي:

```
x y z))
```

فانه سيتم اعتبار x كاسم للدالة function name وسيتم اعتبار y و z المتغيرات التي يتم تمريرها إلى الدالة function arguments.

ماذا يحدث عند عمل تقييم Evaluation لأي تعبير رمزي s-expression ؟!

تعلمنا في القسم المبتدئ من هذه الدروس كيفية عمل تقييم أو تقدير Evaluation عن طريق أداة الـ Listener أو أداة الـ Editor. Text
فماهي ياترى نتيجة التقييم لأي s-expression ؟
ستكون واحدة من ثلاث حالات:

- **عندما يكون الـ s-expression عبارة عن رقم number ، فانه سيتم إعادة هذا الرقم.**

- **عندما يكون الـ s-expression عبارة عن atom فانه سيتم إعادة قيمة ذلك الـatom فيما إذا تم تخصيص قيمة مسبقا له، وغير ذلك سينتج لنا error.**
- **عندما يكون الـ s-expressionعبارة عن ، List فانه سيتم عمل تقييم لعناصر القائمة بدءا من العنصر الثاني وحتى العنصر الأخير، ومن ثم إرسالها إلى الدالة إلى لها الاسم الموجود في أول عنصر في القائمة.**
بفرض انه تم كتابة أي من الـarguments أو اسم الدالة بشكل خاطئ، أو بفرض انه لم يتم تعريف دالة تحمل الاسم المخصص في أول عنصر في القائمة مسبقا فانه سينتج لنا رسالة Error.

مكونات الدرس:

أولا: أنواع الأعداد أو البيانات الرقمية في لغة LISP.

ثانيا: كيفية التعامل مع هذه الأنواع، ونوع الرقم الناتج من إجراء عملية حسابية بين نوعين مختلفين.

ثالثا: سلسلة الدوال الخاصة بإجراء العمليات الحسابية.

1. **دالة الجمع**
2. **دالة الطرح**
3. **دالة الضرب**
4. **دالة القسمة**

رابعا: دوال استرجاع الرقم السابق successor أو اللاحق predecessor
والآن سنتناول هذه النقاط واحدة تلو الأخرى على بركة الله.

أولا: أهم أنواع الأعداد في لغة LISP..

1- الأعداد الصحيحة Numbers Integer

وهي الأعداد التي لا تحتوي على أي أجزاء عشرية، وتكون موجبة أو سالبة أو صفرا.

بعض الأمثلة على الأعداد الصحيحة:

```
CL-USER 1 > 0
0

CL-USER 2 > 1
1

CL-USER 3 > -1
-1
```

2- الأعداد الكسرية Numbers Rational

وهي الأعداد التي تتكون من بسط Numerator ومقام Denominator. لغة LISP لغة ذكية وهي تتعامل مع المبرمج بذكاء؛ وخاصية كتابة الأعداد بصورة كسرية فيها تمثل جزء من ذكاء هذه اللغة..

بعض الأمثلة على الأعداد الكسرية:

```
CL-USER 4 > 1/2
1/2

CL-USER 5 > 2/2
1

CL-USER 6 > -1/4
-1/4

CL-USER 7 > 0/2
0

CL-USER 8 > 2/0
Error: Division-by-zero caused by / of (2 0).
```

3- الأعداد الحقيقة Numbers Real وFloating- Numbers Point

وهي الأعداد التي تتضمن العلامة العشرية.. يختلف النوع Real Numbers عن النوع Floating-Point Numbers في الدقة الموضوعة لكل منهما.

بعض الأمثلة على الأعداد الحقيقية:

```
CL-USER 9 > 1.2
1.2

CL-USER 10 > -1.4
-1.4

CL-USER 11 > 1.00
1.0

CL-USER 12 > 1.0001
1.0001

CL-USER 13 > -.999
-0.999
```

4- الأعداد المركبة Complex Numbers

وهي الأعداد التي تكون من جزأين: جزء حقيقي Real Part، وجزء تخيلي Imagine Part، وتكتب في لغة LISP مسبوقة بالعلامة #C كما في الأمثلة التالية:

```
CL-USER 17 > #C(1 3)
#C(1 3)

CL-USER 18 > #C(2.5 4)
#C(2.5 4.0)
```

ثانيا: كيف يتم التعامل مع هذه الأنواع عند إجراء العمليات الحسابية عليها؟

- إذا تم إجراء عملية حسابية بين أعداد صحيحة فستظهر النتيجة كعدد صحيح أيضا. وبالمثل لو قمنا بإجراء عملية حسابية بين أعداد كسرية فقط فستظهر النتيجة كعدد كسري ولو أجرينا العملية الحسابية على أعداد حقيقية (تحتوي فاصلة عشرية) فستظهر النتيجة كعدد حقيقي أيضا.

- إذا تم إجراء عملية حسابية بين أعداد صحيحة وكسرية فستظهر النتيجة على هيئة كسر.

- إذا تم إجراء عملية حسابية بين مجموعة من الأعداد أحدها على الأقل حقيقي فستظهر النتيجة كعدد حقيقي أيضا..

هكذا نستنتج أن:
شكل ظهور النتيجة يعتمد على نوع الأعداد، والأولوية فيها تكون للأعداد الحقيقية ثم الكسرية ثم العشرية.

ثالثا: الآن سنتناول الـــدوال الخاصة بإجراء العمليات الحسابية..

1- دالة الجمع +

الوظيفة:
تقوم هذه الدالة بإجراء عملية الجمع على عددين أو أكثر، مع إجراء عمليات التحويل للنوع المناسب من الأعداد.
الشكل:

(+ [number[s]])

أمثلة:
لاحظ نوع الرقم الناتج في كل حالة.

```
CL-USER 15 > (+)
0

CL-USER 16 > (+ 1)
1

CL-USER 17 > (+ 2 3)
5

CL-USER 18 > (+ 2.1 3)
5.1

CL-USER 19 > (+ 2.0 1/3)
2.3333333333333335

CL-USER 20 > (+ 31/100 69/100)
1

CL-USER 21 > (+ 1/5 0.8)
1.0

CL-USER 22 > (+ 1/5 2/3)
13/15
```

عند كتابة علامة الجمع فقط فسيظهر لنا المحايد الجمعي

النتيجة ستكون عدد صحيح

النتيجة ستكون عدد حقيقي

النتيجة ستكون عدد كسري

2- دالة الطرح –

الوظيفة:

تقوم هذه الدالة بإجراء عملية الطرح بين عددين أو أكثر، مع إجراء عمليات التحويل للنوع المناسب من الأعداد.

الشكل:

(- number[s])

أمثلة:

لاحظ نوع الرقم الناتج في كل حالة.

```
CL-USER 25 > (- 55.55)
-55.55

CL-USER 26 > (- 3 4)
-1

CL-USER 27 > (- 7 2)
5

CL-USER 28 > (- 10 1 2 3 4)
0

CL-USER 29 > (- 1/2 1/4)
1/4

CL-USER 30 > (- 2.5 2)
0.5
```

3- دالة الضرب *

الوظيفة:

تقوم هذه الدالة بضرب عددين أو أكثر، مع إجراء عمليات التحويل للنوع المناسب من الأعداد.

الشكل:

(* [number[s]])

أمثلة:

لاحظ نوع الرقم الناتج في كل حالة.

```
CL-USER 31 > (*)
1

CL-USER 32 > (* 3 5)
15

CL-USER 33 > (* 2 3 4 5)
120

CL-USER 34 > (* 1/2 1/2)
1/4

CL-USER 35 > (* .2 .3 .4)
0.024
```

4- دالة القسمة /

الوظيفة:

تقوم هذه الدالة بإجراء عملية القسمة بين عددين أو أكثر، مع إجراء عمليات التحويل للنوع المناسب من الأعداد.

الشكل:

(/ number[s])

أمثلة:

لاحظ نوع الرقم الناتج في كل حالة.

```
CL-USER 36 > (/ 2)
1/2

CL-USER 37 > (/ -8)
-1/8

CL-USER 38 > (/ 12 4)
3

CL-USER 39 > (/ 13 4)
13/4

CL-USER 40 > (/ 3 4 5)
3/20

CL-USER 41 > (/ 0.5)
2.0

CL-USER 42 > (/ 20 5)
4

CL-USER 43 > (/ 5 20)
1/4
```

```
CL-USER 44 > (/ 60 -2 3 5.0)
-2.0
```

رابعا: دوال استرجاع الرقم السابق successor أو اللاحق predecessor

1- الدالة 1+

الوظيفة:

تقوم هذه الدالة بإعادة الرقم الذي يزيد عن الرقم المعطى لها بمقدار واحد.

الشكل:

(1+ number)

أمثلة:

```
CL-USER 47 > (1+ 99)
100
```

```
CL-USER 50 > (1+ 2/3)
5/3
```

2- الدالة 1-

الوظيفة:

تقوم هذه الدالة بإعادة الرقم الذي يقل عن الرقم المعطى لها بمقدار واحد.

الشكل:

(1- number)

أمثلة:

```
CL-USER 48 > (1- 100)
99
```

```
CL-USER 49 > (1- 5/3)
2/3
```

سنتناول في الدروس الأربعة القادمة الدوال المبنية مسبقا Functions Predefined في لغة LISP والتي تختص بإجراء عمليات على الأعداد أو ما يسمى بالبيانات الرقمية..

إليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

1- الدالة max
2- الدالة min
3- الدالة PLUSP
4- الدالة MINUSP
5- الدالة ZEROP
6- الدوال TAN SIN, COS,
7- الدوال ASIN, ACOS, ATAN

8- الدوال SINH, COSH, TANH
9- الدوال ASINH, ACOSH, ATANH

ثم سنختم الدرس بالحديث عن المتغير PI أحد المتغيرات المعرفة داخل اللغة Predefined Variable.

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

1- الدالة max

الوظيفة:

تقوم هذه الدالة بإعادة أكبر عدد من بين مجموعة الأعداد المعطاة لها.

الشكل:

(max number[s])

أمثلة:

```
CL-USER 54 > (max 3)
3

CL-USER 55 > (max 2 3 4)
4

CL-USER 56 > (max -2 -3 -4)
-2

CL-USER 57 > (max 8 6 -10)
8

CL-USER 58 > (max 2.5 1/2)
2.5
```

2- الدالة min

الوظيفة:

تقوم هذه الدالة بالوظيفة العكسية للدالة السابقة، حيث تعيد أصغر عدد من بين مجموعة الأعداد المعطاة لها.

الشكل:

(min number[s])

أمثلة:

```
CL-USER 64 > (min 3)
3

CL-USER 65 > (min 2 3 4)
2

CL-USER 66 > (min -2 -3 -4)
-4

CL-USER 67 > (min 8 6 -10)
-10

CL-USER 68 > (min 2.5 1/2)
1/2
```

3- الدالة PLUSP

الوظيفة:

تقوم هذه الدالة باختبار العدد المعطى لها، وتعود بالقيمة true (T) إذا كان أكبر من الصفر، وتعود بالقيمة false (NIL) فيما عدا ذلك.

الشكل:

(plusp number)

أمثلة:

```
CL-USER 69 > (plusp 2)
T

CL-USER 70 > (plusp -2)
NIL

CL-USER 71 > (plusp 0)
NIL
```

MINUSP الدالة -4

الوظيفة:

تقوم هذه الدالة بالوظيفة العكسية للدالة السابقة،
فتختبر العدد المعطى لها وتعود بالقيمة T (true) إذا كان
أقل من الصفر وتعود بالقيمة NIL (false) فيما عدا ذلك.

الشكل:

(minusp number)

أمثلة:

```
CL-USER 72 > (minusp -2)
T

CL-USER 73 > (minusp 2)
NIL

CL-USER 74 > (minusp 0)
NIL
```

ZEROP الدالة -5

الوظيفة:

تقوم هذه الدالة باختبار العدد المعطى لها وتعود بالقيمة
T (true) إذا كان العدد المعطى لها صفرا وتعود بالقيمة
NIL (false) فيما عدا ذلك.

الشكل:

(zerop number)

أمثلة:

```
CL-USER 75 > (zerop 0)
T

CL-USER 76 > (zerop 1)
NIL

CL-USER 77 > (zerop -0.0)
T

CL-USER 78 > (zerop 0/100)
T
```

6- الدوال SIN, COS, TAN

الوظيفة:

تسمى هذه الدوال بالدوال المثلثية وتعيد قيمة الـ Sine ، Cosine و Tangent للزاوية المعطاة، ويجب أن تكون الزاوية مقدرة بالراديان أو ما يسمى بالتقدير الدائري.

الشكل:

(sin radian_angle)

(cos radian_angle)

(tan radian_angle)

أمثلة:

أمثلة على تطبيق الدالة SIN على أرقام متعددة:

```
CL-USER 79 > (sin 0)
0.0

CL-USER 80 > (sin 22/7)
-0.00126448893037729

CL-USER 81 > (sin 0.7853982)
0.7071068070684596
```

أمثلة على تطبيق الدالة COS على أرقام متعددة:

```
CL-USER 82 > (cos 0)
1.0

CL-USER 83 > (cos 22/7)
-0.999999200533553

CL-USER 84 > (cos 0.7853982)
0.7071067553046345
```

أمثلة على تطبيق الدالة TAN على أرقام متعددة:

```
CL-USER 85 > (tan 0)
0.0

CL-USER 86 > (tan 22/7)
0.0012644899412945707

CL-USER 87 > (tan 0.7853982)
1.0000000732051062
```

7- الدوال ATAN ASIN, ACOS

الوظيفة:

تسمى هذه الدوال بالدوال المثلثية العكسية وتعيد قيمة الـarc cosine ، arc sine و arc tangent للزاوية المعطاة، ويجب أن تكون الزاوية مقدرة بالراديان.

الشكل:

(asin radian_angle)

(acos radian_angle)

(atan radian_angle)

أمثلة:

أمثلة على تطبيق الدالة ASIN على أرقام متعددة:

```
CL-USER 88 > (asin 0)
0.0

CL-USER 89 > (asin .5)
0.5235987755982989

CL-USER 90 > (asin 0.7853982)
0.9033391698991327
```

أمثلة على تطبيق الدالة ACOS على أرقام متعددة:

```
CL-USER 91 > (acos 0)
1.5707963267948966

CL-USER 92 > (acos 0.5)
1.0471975511965976

CL-USER 93 > (acos 0.7853982)
0.6674571568957639
```

أمثلة على تطبيق الدالة ATAN على أرقام متعددة:

```
CL-USER 94 > (atan 0)
0.0

CL-USER 95 > (atan 0.5)
0.4636476090008061

CL-USER 96 > (atan 0.7853982)
0.665773772666536
```

8- الدوال TANH SINH, COSH

الوظيفة:

تسمى هذه الدوال بالدوال الزائدية وتعيد قيمة الـHyperbolic sine

Hyperbolic cosine و Hyperbolic tangent للزاوية المعطاة، ويجب أن تكون الزاوية مقدرة بالراديان.

الشكل:

(sinh radian_angle)

(cosh radian_angle)

(tanh radian_angle)

أمثلة:

أمثلة على تطبيق الدالة SINH على أرقام متعددة:

```
CL-USER 97 > (sinh 0)
0.0

CL-USER 98 > (sinh -0.5)
-0.5210953054937474

CL-USER 99 > (sinh 0.7853982)
0.868671009970083
```

أمثلة على تطبيق الدالة COSH على أرقام متعددة:

```
CL-USER 100 > (cosh 0)
1.0

CL-USER 101 > (cosh .5)
1.1276259652063807

CL-USER 102 > (cosh 0.7853982)
1.3246091210475806
```

أمثلة على تطبيق الدالة TANH على أرقام متعددة:

```
CL-USER 103 > (tanh 0)
0.0

CL-USER 104 > (tanh .5)
0.4621171572600098

CL-USER 105 > (tanh 0.7853982)
0.6557942234937093
```

9- الدوال ATANH ASINH, ACOSH

الوظيفة:

تسمى هذه الدوال بالدوال الزائدية العكسية وتعيد قيمة الـHyperbolic arc sine و Hyperbolic arc cosine و tangent Hyperbolic arc للزاوية المعطاة، ويجب أن تكون الزاوية مقدرة بالراديان.

الشكل:

(asinh radian_angle)

(acosh radian_angle)

(atanh radian_angle)

أمثلة:

أمثلة على تطبيق الدالة ASINH على أرقام متعددة:

```
CL-USER 106 > (asinh 0)
0.0

CL-USER 107 > (asinh .5)
0.48121182505960347

CL-USER 108 > (asinh 0.7853982)
0.7212255175124574
```

أمثلة على تطبيق الدالة ACOSH على أرقام متعددة:

```
CL-USER 109 > (acosh 0)
#C(0.0 1.5707963267948966)

CL-USER 110 > (acosh .5)
#C(-2.2204460492503131E-16 1.0471975511965979)

CL-USER 111 > (acosh -5)
#C(2.2924316695611777 3.141592653589793)

CL-USER 112 > (acosh 0.7853982)
#C(0.0 0.6674571568957639)

CL-USER 113 > (acosh 22/7)
1.8119507608214132

CL-USER 114 > (acosh 200)
5.991458297049387 5
```

أمثلة على تطبيق الدالة ATANH على أرقام متعددة:

```
CL-USER 115 > (atanh 0)
0.0

CL-USER 116 > (atanh .5)
0.5493061443340548

CL-USER 117 > (atanh -.5)
-0.5493061443340548

CL-USER 118 > (atanh 0.7853982)
1.059306266353925

CL-USER 119 > (atanh 22/7)
#C(0.3296228144421321 -1.5707963267948966)
```

المتغير PI

يعتبر PI أحد المتغيرات المحجوزة داخل اللغة ويعطيك قيمة دقيقة جدا للقيمة PI، جرب واكتب اسم هذا المتغير على الـListener ثم انقر على Enter لترى النتيجة التالية:

```
CL-USER 1 > pi
3.141592653589793
```

سنتابع في هذا الدرس الدوال الخاصة بالتعامل مع الأرقام.

وإليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

10- الدالة ABS
11- الدالة EVENP
12- الدالة ODDP
13- الدالة EXP
14- الدالة EXPT
15- الدالة LOG
16- الدالة GCD
17- الدالة GCD

18- الدالة INCF
19- الدالة DECF

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

10- الدالة ABS

الوظيفة:

تقوم هذه الدالة بإعادة القيمة المطلقة Absolute Value للرقم المعطى لها أيا كان نوعه.

الشكل:

(abs number)

أمثلة:

```
CL-USER 2 > (abs 0)
0

CL-USER 3 > (abs -1)
1

CL-USER 4 > (abs 12/13)
12/13

CL-USER 5 > (abs -12/13)
12/13

CL-USER 7 > (abs -1.09)
1.09
```

11- الدالة EVENP

الوظيفة:

تقوم هذه الدالة باختبار الرقم الصحيح المعطى لها وتعود بالقيمة T (True) إذا كان الرقم زوجي، وتعود بالقيمة False خلاف ذلك.

الشكل:

(evenp integer)

أمثلة:

```
CL-USER 11 > (evenp 2)
T

CL-USER 12 > (evenp 0)
T

CL-USER 13 > (evenp 10000000)
T

CL-USER 14 > (evenp 5)
NIL

CL-USER 15 > (evenp 1.2)
Error: In EVENP of (1.2) arguments should be of type INTEGER.
```

12- الدالة ODDP

الوظيفة:

تقوم هذه الدالة باختبار الرقم الصحيح المعطى لها وتعود بالقيمة T (True) إذا كان الرقم فردي، وتعود بالقيمة False خلاف ذلك.

الشكل:

(oddp integer)

أمثلة:

```
CL-USER 18 > (oddp -1)
T

CL-USER 19 > (oddp 0)
NIL

CL-USER 20 > (oddp 3)
T

CL-USER 21 > (oddp 2)
NIL

CL-USER 22 > (oddp 1000005)
T

CL-USER 23 > (oddp -3)
T
```

13- الدالة EXP

الوظيفة:

تقوم هذه الدالة بإعادة الدالة الأسية e وقيمتها تساوي 2.718281828459045 مرفوعةً إلى الرقم الذي أرسلناه للدالة (أيا كان نوعه).

الشكل:

(exp number)

أمثلة:

```
CL-USER 26 > (exp 1)
2.718281828459045

CL-USER 27 > (exp 2)
7.38905609893065

CL-USER 28 > (exp -1)
0.36787944117144233

CL-USER 29 > (exp -22/3)
6.533919798673805E-4

CL-USER 30 > (exp 1/2)
1.6487212707001282

CL-USER 31 > (exp 0.5)
1.6487212707001282
```

14- الدالة EXPT

الوظيفة:

تختلف هذه الدالة عن سابقتها في أنها تستقبل رقمين: الأول يعبر عن الأساس والثاني يعبر عن الأس أو القوة التي سيرفع لها الرقم، وستقوم بإعادة نتيجة رفع الرقم للقوة المعطاة أيا كان نوع الرقم (الأساس) وأيا كان نوع القوة (الأس).

الشكل:

(expt base-number power-number)

أمثلة:

```
CL-USER 41 > (expt 2 4)
16

CL-USER 42 > (expt 4 .5)
2.0

CL-USER 43 > (expt 2 -1)
1/2
```

15- الدالة LOG

الوظيفة:

تعود هذه الدالة بلوغاريتم العدد المعطى لها مع افتراض أن الأساس هو العدد الطبيعي e إذا لم نحدد لها الأساس، أما إذا قمنا بتحديد الأساس فإنها ستعود بلوغاريتم العدد لنفس الأساس المعطى. شاهد الشكل والأمثلة ليتضح ذلك أكثر.

الشكل:

(expt number [optional base])

أمثلة:

```
CL-USER 48 > (log 2)
0.6931471805599453

CL-USER 49 > (log 2 10)
0.30102999566398114

CL-USER 50 > (exp (log 5))
4.999999999999999

CL-USER 51 > (log 100 10)
2.0
```

الدالة الأسية واللوغاريتمية تلغي إحداهما لأخرى وفي الرياضيات نكتب الناتج 5 مباشرة

شاهد الكود التالي لتتأكد مما ذكرته في الصورة السابقة بنفسك:

```
CL-USER 52 > (log (exp 5))
5.0
```

16- الدالة GCD

الوظيفة:

اختصار لـgreatest-common-denominator .
تعود هذه الدالة بالقاسم المشترك الأكبر لمجموعة الأعداد الصحيحة المعطاة لها.

حالات خاصة:

- **عندما نعطي لهذه الدالة عدد واحد فإنها ستعود بالقيمة المطلقة لهذا العدد.**
- **أما عندما لا نرسل لها أي عدد فإنها ستعود بالرقم صفر حيث يعتبر العنصر المحايد في هذه العملية.**

الشكل:

(gcd [integer[s]])

أمثلة:

```
CL-USER 1 > (gcd)
0

CL-USER 2 > (gcd 5)
5

CL-USER 3 > (gcd -4)
4

CL-USER 4 > (gcd 60 42)
6

CL-USER 5 > (gcd 3333 -33 101)
1

CL-USER 6 > (gcd 91 49)
7

CL-USER 7 > (gcd 63 42 35)
7

CL-USER 8 > (gcd 3333 33 1002001)
11
```

17- الدالة GCD

الوظيفة:

اختصار لـleast-common-multiple.
تعود هذه الدالة بالمضاعف المشترك الأصغر لمجموعة الأعداد المعطاة لها من أي نوع.

حالات خاصة:

- **عندما نعطي لهذه الدالة عدد واحد فإنها ستعود بالقيمة المطلقة لهذا العدد.**
- **عندما لا نرسل لها أي عدد فإنها ستعود بالرقم واحد حيث يعتبر العنصر المحايد في هذه العملية.**
- **عندما نرسل لها الدالة عددين كلاهما لا يساوي الصفر، فإن:**

(lcm a b) == (/ (abs (* a b)) (gcd a b))

- **عندما يكون كلا المتغيرين أو أحدهما يساوي الصفر، فإن:**

lcm a 0) == (lcm 0 a) == 0)

- أما إذا أرسلنا أكثر من عددين فإنه سيتم حساب المضاعف المشترك الأصغر لهذه الأعداد كالآتي:

(lcm a b c ... z) == (lcm (lcm a b) c ... z)

الشكل:

(lcm [integer[s]])

أمثلة:

```
CL-USER 11 > (lcm)
1

CL-USER 12 > (lcm 10)
10

CL-USER 13 > (lcm -10)
10

CL-USER 14 > (lcm 25 30)
150

CL-USER 15 > (lcm -24 18 10)
360

CL-USER 16 > (lcm 13 35)
455

CL-USER 17 > (lcm 0 5)
0

CL-USER 18 > (lcm 1 2 3 4 5 6)
60
```

18- الدالة INCF

الوظيفة:

اختصار لـ Increment Function.
ونرسل لهذه الدالة متغير يخزن فيه الرقم الأصلي، ورقم يعبر عن مقدار الزيادة.. وتعود هذه الدالة بالرقم المخزن في المتغير مضافا إليه مقدار الزيادة المطلوبة.

الشكل:

(incf place [delta-form])

أمثلة:

في البداية سنستخدم الدالة setq لإسناد قيمة عددية ولتكن 5 إلى متغير اسمه x كالتالي:

```
CL-USER 27 > (setq x 5)
5
```

ولتتأكد أنه تم تخزين القيمة العددية 5 في المتغير الذي أسميناه x اكتب x فقط ثم اضغط على enter:

```
CL-USER 28 > x
5
```

والآن ستستطيع تطبيق الدالة incf بأشكال مختلفة على المتغير x ، مع ملاحظة أنه يجب أن يكون المتغير x معرف في نفس الـ Listener أو أن يتم عمل Evaluation له إذا كان مكتوبا في الـ Text Editor .
تابع معي هذه الأمثلة:

```
CL-USER 29 > (incf x)
6

CL-USER 30 > x
6
```

في المثال السابق نلاحظ أن قيمة x زادت بمقدار واحد.

```
CL-USER 31 > (incf x 3)
9

CL-USER 32 > x
9
```

في المثال السابق نلاحظ أن قيمة x زادت بمقدار 3.

```
CL-USER 33 > (incf x 0.5)
9.5

CL-USER 34 > x
9.5
```

في المثال السابق نلاحظ أن قيمة x زادت بمقدار 0.5 .

```
CL-USER 35 > (incf x -0.5)
9.0

CL-USER 36 > x
9.0
```

في المثال السابق نلاحظ أن قيمة x زادت بمقدار -0.5 .

ملاحظة هامة:

لاحظنا أنه عندما نرسل للدالة incf المتغير فقط، فان مقدار الزيادة الافتراضي هو واحد، أي أن الرقم سيزيد في هذه الحالة بمقدار واحد.
وبالتالي ستقوم الدالة incf بنفس وظيفة الدالة 1+ ، فما الفرق بينهما إذن؟
الفرق هو أننا عندما نرسل متغير للدالة 1+ فان الدالة ستعود بالمتغير مضافا إليه واحد دون أن تخزن القيمة الجديدة في المتغير. أي أن المتغير سيبقى بنفس قيمته

القديمة دون تغيير.

أما عندما نرسل متغير للدالة incf فان الدالة ستعود بالرقم مضافا إليه مقدار الزيادة المطلوبة وكذا فإن القيمة الجديدة سيتم تخزينها مكان القيمة القديمة لذلك المتغير.
ولتوضيح ذلك سنضرب المثال التالي:
قم بتخزين العدد 3 في المتغير K:

```
CL-USER 39 > (setq k 3)
3

CL-USER 40 > k
3
```

استخدم الدالة 1+ وسترى بأنها ستعود بالرقم 4:

```
CL-USER 41 > (1+ k)
4
```

ومع هذا لم تخزن القيمة الجديدة 4 في المتغير k:

```
CL-USER 41 > (1+ k)
4

CL-USER 42 > k
3
```

أما لو استخدما الدالة incf مع المتغير k كالتالي:

```
CL-USER 43 > (incf k)
4
```

فان القيمة الجديدة أربعة سيتم تخزينها في المتغير k:

```
CL-USER 43 > (incf k)
4

CL-USER 44 > k
4
```

19- الدالة DECF

الوظيفة:
اختصار لـ Decrement Function.
تقوم هذه الدالة بالوظيفة العكسية لسابقتها، ونرسل لها متغير يخزن فيه الرقم الأصلي، ورقم يعبر عن مقدار النقصان.. وتعود هذه الدالة بالرقم المخزن في المتغير مطروحا منه المقدار المطلوب.

الشكل:

(decf place [delta-form])

أمثلة:

في البداية سنستخدم الدالة setq لإسناد قيمة عددية ولتكن 5 إلى متغير اسمه x كالتالي:

```
CL-USER 27 > (setq x 5)
5
```

بالمثل تستطيع تطبيق الدالة decf بأشكال مختلفة على المتغير x ، مع ملاحظة أنه يجب أن يكون المتغير x معرف في نفس الـ Listener أو أن يتم عمل Evaluation له إذا كان مكتوبا في الـ Text Editor .
تابع معي هذه الأمثلة:

```
CL-USER 53 > (decf x)
4

CL-USER 54 > x
4
```

نلاحظ أن قيمة x نقصت بمقدار واحد.

```
CL-USER 55 > (decf x 2)
2

CL-USER 56 > x
2
```

نلاحظ أن قيمة x نقصت بمقدار 2.

```
CL-USER 57 > (decf x -2)
4

CL-USER 58 > x
4
```

نلاحظ أن قيمة x نقصت بمقدار -2 أي أنها زادت بمقدار 2.

```
CL-USER 61 > (decf x .5)
3.5

CL-USER 62 > x
3.5
```

نلاحظ أن قيمة x نقصت بمقدار 0.5 .

```
CL-USER 63 > (decf x -.5)
4.0

CL-USER 64 > x
4.0
```

نلاحظ أن قيمة x نقصت بمقدار -0.5 أي أنها زادت بمقدار 0.5 .

وبالمثل سنلاحظ القيمة الجديدة ستُخزن في المتغير عند استخدام الدالة decf:

```
CL-USER 76 > (setq k 5)
5

CL-USER 77 > k
5
CL-USER 78 > (decf k)
4

CL-USER 79 > k
4
```

بينما عندما نرسل متغير للدالة 1+ فان الدالة ستعود بالمتغير منقوصا منه واحد دون أن تخزن القيمة الجديدة في المتغير. أي أن المتغير سيبقى بنفس قيمته القديمة دون تغيير:

```
CL-USER 80 > (1- k)
3

CL-USER 81 > k
4
```

سنتابع في هذا الدرس الدوال الخاصة بالتعامل مع الأرقام.. وإليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

20- الدالة SIGNUM
21- الدالة NUMBERP
22- الدالة RANDOM
23- الدالة SQRT
24- الدالة ISQRT
25- الدالة CIS
26- الدالة MOD
27- الدالة COMPLEX
28- الدالة COMPLEXP
29- الدالتان REALPART, IMAGPART

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

20- الدالة SIGNUM

الوظيفة:

هذه الدالة تستقبل عدد واحد، وتقوم بإعادة رقم ذي دلالة على نوعية الرقم المعطى لها، هل هو موجب أم سالب أم صفرا.

- **إذا كان الرقم المرسل موجب فإن الدالة تعود بالقيمة 1**
- **إذا كان الرقم المرسل سالب فإن الدالة تعود بالقيمة 1-**
- **إذا كان الرقم المرسل صفرا فإن الدالة تعود بالقيمة 0**

الشكل:

(signum number)

أمثلة:

```
CL-USER 5 > (signum 0)
0

CL-USER 6 > (signum 99)
1

CL-USER 7 > (signum 4/5)
1

CL-USER 8 > (signum -99)
-1

CL-USER 9 > (signum -99/100)
-1

CL-USER 10 > (signum 0.0)
0.0

CL-USER 11 > (signum 1.5)
1.0

CL-USER 12 > (signum -1.5)
-1.0
```

21- الدالة NUMBERP

الوظيفة:

هذه الدالة تعود بالقيمة T (True) إذا كان الكائن المعطى لها يمثل رقما، وتعود بالقيمة Nil (False) إذا كانت قيمة الكائن المعطى لها ليست رقمية.

الشكل:

(numberp object)

أمثلة:

```
CL-USER 13 > (numberp 12)
T

CL-USER 14 > (numberp (exp 3))
T

CL-USER 15 > (numberp #c(7/3 7.2))
T
CL-USER 16 > (numberp nil)
NIL
```

وبفرض أننا احتفظنا في متغير ما وليكن var بأي قيمة غير عددية ولتكن الجملة "c4arab" فبالتأكيد ستعود الدالة numperp بالقيمة NIL عند إرسال المتغير var لها.

```
CL-USER 21 > (setq var "c4arab")
"c4arab"

CL-USER 22 > var
"c4arab"

CL-USER 23 > (numberp var)
NIL
```

22- الدالة RANDOM

الوظيفة:

دالة هامة جدا!.. تعود هذه الدالة بأي رقم عشوائي غير سالب أقل من الرقم (limit) المرسل لها ومن نفس نوعه.
مع ملاحظة أن الرقم المعطى لها يجب أن يكون غير سالب أو صفري.

الشكل:

(random limit_number)

أمثلة:

```
CL-USER 1 > (random 1000)
705

CL-USER 2 > (random 3)
2

CL-USER 7 > (random 5.5)
5.0420854045874615

CL-USER 8 > (random .5)
0.4647966315928823
```

اذا قمنا بإرسال قيمة سالبة لهذه الدالة فإن رسالة خطأ ستظهر لك مبينة أنه لا يمكن تطبيق هذه الدالة على الأعداد السالبة:

```
CL-USER 3 > (random -1)

Error: Arithmetic error in RANDOM of (-1 #S(RANDOM-STATE SYSTEM::J 22 SYSTEM::K
::SEED #(2192705 2501849 2409239 4093862 1225727 581186 2200121 642917 1834949
20155 2551685 377966 255479 434798 2336357 3985652 3338144 826394 1709237 18891
 3647963 3504914 2076740 2940677 1867784 2624564 3068981 1627352 1706828 389591
1447904 1814006 1076177 891245 3775088 3285521 3142196 ...))): First argument i
sitive real.
  1 (continue) Return a value to use.
  2 (abort) Return to level 0.
  3 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,  or :? for other options

CL-USER 4 : 1 > :c

Supply a form to be evaluated and used: 3
```

كذلك إذا قمنا بإرسال صفر لهذه الدالة فإن رسالة خطأ ستظهر أيضا، فلا يمكن تطبيق دالة توليد الأرقام العشوائية على قيمة صفرية:

```
CL-USER 5 > (random 0)

Error: Arithmetic error in RANDOM of (0 #S(RANDOM-STATE SYSTEM::J 22 SYSTEM::K !
53 SYSTEM::SEED #(2192705 2501849 2409239 4093862 1225727 581186 2200121 642917!
 1834949 1823069 4020155 2551685 377966 255479 434798 2336357 3985652 3338144 8!
26394 1709237 188917[illegible] 1826582 3647963 3504914 2076740 2940677 1867784 2624564 3!
068981 1627352 1706828 3895910 2111351 1447904 1814006 1076177 891245 3775088 3!
285521 3142196 ...))): First argument is not a positive real.
  1 (continue) Return a value to use.
  2 (abort) Return to level 0.
  3 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,  or :? for other options

CL-USER 6 : 1 > :c 3
```

23- الدالة SQRT

الوظيفة:

تعود هذه الدالة بالجذر التربيعي للعدد المرسل لها من أي نوع.. يمكن إرسال عدد سالب لها ليتم حساب وإعادة الجذر التخيلي root Imaginary أو ما يسمى بالـ Complex Number في لغة LISP.

ملاحظة:

يتم كتابة الأعداد أو الجذور التخيلية في لغة LISP باستخدام العلامة #c قبلها مما يدل على أن العدد التابع لها تخيلي أو غير حقيقي.

الشكل:

(sqrt number)

أمثلة:

```
CL-USER 1 > (sqrt 9)
3.0

CL-USER 2 > (sqrt -9)
#C(0.0 3.0)

CL-USER 3 > (sqrt -1)
#C(0.0 1.0)

CL-USER 4 > (sqrt 12)
3.4641016151377544

CL-USER 5 > (sqrt 12.5)
3.5355339059327378

CL-USER 6 > (sqrt #c(0 2))
#C(1.0000000000000002 1.0)
```

24- الدالة ISQRT

الوظيفة:

تعود هذه الدالة بالجذر التربيعي لأي عدد طبيعي أي أنه يشترط أن يكون العدد موجب.. إذن هذه الدالة لا تقبل الأعداد السالبة لأنها لا تستطيع حساب الجذور التخيلية بعكس الدالة السابقة.

عندما نرسل لهذه الدالة عدد حقيقي موجب محتوي على علامة عشرية فإن هذه الدالة تعود بأكبر عدد صحيح موجب أقل من أو يساوي الجذر الطبيعي، شاهد الأمثلة لترى بنفسك.

الشكل:

(isqrt natural_number)

أمثلة:

```
CL-USER 9 > (isqrt 9)
3

CL-USER 12 > (isqrt 300)
17

CL-USER 13 > (isqrt 325)
18

CL-USER 14 > (isqrt 25)
5
```

```
CL-USER 7 > (isqrt .5)
1

CL-USER 8 > (isqrt 12.4)
3
```

أما إذا أدخلنا عدد سالب فسيظهر الخطأ:

```
CL-USER 15 > (isqrt -1)

Error: In ISQRT of (-1) arguments should be of type INTEGER.
```

25- الدالة CIS

الوظيفة:

تستقبل هذه الدالة الزاوية مقدرة بالراديان (يسمى أيضا التقدير الدائري) وتعود بالقيمة e^i* radians وهو عدد مركب الجزء الحقيقي فيه يمثل قيمة الجتا Cosine للزاوية والجزء التخيلي يمثل قيمة الجا Sine للزاوية.

الشكل:

(cis radians)

أمثلة:

```
CL-USER 19 > (cis 0)
#C(1.0 0.0)
```

26- الدالة MOD

الوظيفة:

تعود هذه الدالة بقيمة الـModulus للرقم المرسل لها.

الشكل:

(mod number divisor)

أمثلة:

```
CL-USER 21 > (mod -1 5)
4

CL-USER 22 > (mod 13 4)
1

CL-USER 23 > (mod -13 4)
3

CL-USER 24 > (mod 13 -4)
-3

CL-USER 25 > (mod -13 -4)
-1

CL-USER 26 > (mod 13.4 1)
0.40000000000000036

CL-USER 27 > (mod -13.4 1)
0.5999999999999996
```

COMPLEX الدالة -27

الوظيفة:

تقوم هذه الدالة بتوليد عدد مركب الجزء الحقيقي له هو الـargument الأول والجزء التخيلي هو الـargument الثاني (فيما لو تم إرساله)

الشكل:

(complex realpart [imagpart])

أمثلة:

```
CL-USER 28 > (complex 0)
0

CL-USER 29 > (complex 0.0)
#C(0.0 0.0)

CL-USER 30 > (complex 1 1/12)
#C(1 1/12)

CL-USER 31 > (complex 1 .99)
#C(1.0 0.99)

CL-USER 32 > (complex 3/2 0.0)
#C(1.5 0.0)
```

COMPLEXP الدالة -28

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عدد مركب وتعود بالقيمة Nil (False) خلاف ذلك.

الشكل:

(complexp object)

أمثلة:

```
CL-USER 33 > (complexp 1.2d2)
NIL

CL-USER 34 > (complexp #c(5/3 7.2))
T
```

29- الدالتان, IMAGPART REALPART

الوظيفة:

تعود الدالة REALPART بالجزء الحقيقي في العدد المركب المرسل لها.

بينما تعود الدالة IMAGPART بالجزء التخيلي للعدد المركب المرسل لها، وإذا تم إرسال عدد حقيقي غير مركب لها فان هذه الدالة ستعتبر الجزء التخيلي صفرا.

الشكل:

(realpart number)

(imagpart number)

أمثلة:

```
CL-USER 35 > (realpart 0)
0

CL-USER 36 > (realpart #c(23 41))
23

CL-USER 37 > (realpart #c(23 -41))
23

CL-USER 38 > (imagpart 23.0)
0.0

CL-USER 39 > (imagpart #c(23 41))
41

CL-USER 40 > (imagpart #c(23 -41))
-41
```

هذا هو الدرس الرابع والأخير في الدوال الخاصة بالتعامل مع الأرقام..

وإليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

30- الدالة REALP
31- الدالتان RATIONAL, RATIONALIZE
32- الدالة RATIONALP
33- الدالتان DENOMINATOR NUMERATOR,
34- الدالة INTEGER-LENGTH
35- الدالة INTEGERP
36- الدالة FLOAT
37- الدالة FLOATP

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

30- الدالة REALP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة عن عدد حقيقي..
بينما تعود بالقيمة Nil (False) خلاف ذلك.

الشكل:

(realp object)

أمثلة:

```
CL-USER 41 > (realp 12)
T

CL-USER 42 > (realp #c(5/3 7.2))
NIL

CL-USER 43 > (realp nil)
NIL

CL-USER 44 > (realp -12.00003)
T
```

31- الدالتان RATIONAL, RATIONALIZE

الوظيفة:

تقوم هاتان الدالتان بتحويل العدد المعطى لها إلى صورة عدد كسري مكون من بسط ومقام.

وتختلف الدالتان عن بعضهما في الدقة الموضوعة لكل منهما:

فالدالة RATIONAL تضع أعدادا تنتج عند قسمتها من جديد العدد العشري نفسه (بنفس الدقة) بينما الدالة RATIONALIZE تقرب الأعداد أكثر لتعطي كسرا أبسط ولكن عند القسمة سنحصل على رقم عشري قريب جدا من الرقم العشري المعطى لها.

حالات خاصة:

- هاتان الدالتان تعودان بالقيمة صفر عندما يكون العدد صفر هو المرسل إليهما.
- كذلك عند إرسال عدد صحيح Integer أو كسري Rational فان هاتان الدالتان تعودان بنفس العدد.

الشكل:

(rational number)

(rationalize number)

أمثلة:

عند إرسال صفر:

```
CL-USER 49 > (rational 0)
0

CL-USER 50 > (rationalize 0)
0
```

عند إرسال عدد صحيح:

```
CL-USER 55 > (rational 1)
1

CL-USER 56 > (rationalize 1)
1
```

عند إرسال عدد كسري:

```
CL-USER 51 > (rational -11/100)
-11/100

CL-USER 52 > (rationalize -11/100)
-11/100
```

عند إرسال عدد حقيقي (عشري):

```
CL-USER 53 > (rational .1)
3602879701896397/36028797018963968

CL-USER 54 > (rationalize .1)
1/10
```

32- الدالة RATIONALP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة عن عدد كسري.
بينما تعود بالقيمة Nil (False) خلاف ذلك.

الشكل:

(rationalp object)

أمثلة:

```
CL-USER 45 > (rationalp 12)
T

CL-USER 46 > (rationalp 6/5)
T

CL-USER 47 > (rationalp -6/5)
T

CL-USER 48 > (rationalp 1.212)
NIL
```

33- الدالتان DENOMINATOR NUMERATOR,

الوظيفة:

تعود الدالة NUMERATOR ببسط العدد الكسري المرسل لها.
بينما تعود الدالة DENOMINATOR بمقام العدد الكسري المرسل لها.

الشكل:

(realpart number)

(imagpart number)

أمثلة:

استخدام الدالة NUMERATOR:

```
CL-USER 57 > (numerator 1/2)
1

CL-USER 58 > (numerator -1)
-1

CL-USER 59 > (numerator (/ 8 -6))
-4
```

استخدام الدالة DENOMERATOR:

```
CL-USER 60 > (denominator 12/36)
3

CL-USER 61 > (denominator (/ -33))
33

CL-USER 62 > (denominator (/ 8 -6))
3
```

34- الدالة INTEGER-LENGTH

الوظيفة:

تستقبل هذه الدالة أي رقم صحيح وتعود بعدد البت اللازمة لتمثيل هذا العدد في صيغة المكمل الثنائي format two's-complement binary.

الشكل:

(integer-length integer)

أمثلة:

عند تطبيق الدالة على الرقم صفر نحصل على:

```
CL-USER 63 > (integer-length 0)
0
```

عند تطبيق الدالة على أرقام صحيحة موجبة:

```
CL-USER 64 > (integer-length 1)
1

CL-USER 65 > (integer-length 3)
2

CL-USER 66 > (integer-length 4)
3

CL-USER 67 > (integer-length 7)
3

CL-USER 68 > (integer-length 6)
3
```

عند تطبيق الدالة على أرقام صحيحة سالبة:

```
CL-USER 69 > (integer-length -1)
0

CL-USER 70 > (integer-length -4)
2

CL-USER 71 > (integer-length -7)
3

CL-USER 72 > (integer-length -8)
3
```

أمثلة متقدمة على استخدامات هذه الدالة:

```
CL-USER 73 > (integer-length (expt 2 9))
10

CL-USER 74 > (integer-length (1- (expt 2 9)))
9

CL-USER 75 > (integer-length (- (expt 2 9)))
9

CL-USER 76 > (integer-length (- (1+ (expt 2 9))))
10
```

35- الدالة INTEGERP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة عن عدد صحيح.
بينما تعود بالقيمة Nil (False) خلاف ذلك.

الشكل:

(integerp object)

أمثلة:

```
CL-USER 77 > (integerp 0)
T

CL-USER 78 > (integerp 1)
T

CL-USER 79 > (integerp -1)
T

CL-USER 80 > (integerp (expt 2 130))
T

CL-USER 81 > (integerp 6/5)
NIL

CL-USER 82 > (integerp (random 4))
T

CL-USER 83 > (integerp nil)
NIL
```

36- الدالة FLOAT

الوظيفة:

تقوم هذه الدالة بتحويل العدد الحقيقي إلى عدد عشري.
الفرق بين الـReal Numbers والـfloat Numbers هو في الدقة أي في عدد المنازل العشرية فقط.

الشكل:

(float number [prototype])

إذا تم إرسال Prototype لهذه الدالة فإن الرقم العائد يكون له نفس الدقة أي نفس الشكل الموجود في هذا الـPrototype.
لاحظ الأمثلة التالية لترى نتيجة استخدام الـPrototype بنفسك.

أمثلة:

```
CL-USER 86 > (float 0)
0.0

CL-USER 87 > (float 1 .5)
1.0

CL-USER 88 > (float 1.0)
1.0

CL-USER 89 > (float 1/2)
0.5
```

```
CL-USER 90 > (= (float 1.0 1.0d0) 1.0d0)
T
```

37- الدالة FLOATP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة عن عدد عشري من النوع . Float
بينما تعود بالقيمة False (Nil) خلاف ذلك.

الشكل:

(floatp object)

أمثلة:

هذه الدالة تعود بالقيمة T عندما نرسل لها أرقاما عشرية:

```
CL-USER 91 > (floatp 1.2d2)
T

CL-USER 92 > (floatp 1.212)
T

CL-USER 94 > (floatp -1.0000000034)
T
```

بينما لو أرسلنا أرقاما صحيحة سنجد أنها تعود بالقيمة NIL:

```
CL-USER 95 > (floatp 44)
NIL

CL-USER 96 > (floatp (expt 2 130))
NIL
```

~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

وهكذا تناولنا معظم وأهم الدوال المعرفة في LISP للتعامل مع الأرقام.. من السهولة جدا تعلم ومعرفة ماهية عمل هذه الدوال، أليس كذلك؟! :)
من الممتع أيضا أننا نتدرج في المعلومة لذا ستجدون بإذن الله أن كل كلمة مستساغة ومفهومة.. عندما نصل إلى القسم المتوسط من الدروس سيكون كل شيء سهل وواضح..

في هذا الدرس والدرس القادم سأتحدث بالتفصيل عن الرموز Characters في LISP ماهيتها وكيفية التعامل معها في LISP والدوال المعرفة مسبقا في اللغة للتعامل مع الرموز أو إجراء عمليات عليها.

مكونات هذا الدرس:

أولا: تعريف الرمز CHARACTER ؟

ثانيا: سلسلة الدوال المختصة بالتعامل مع الرموز وهي 14 دالة سيتم شرحها في هذا الدرس.

ثالثا: ثم سنختم الدرس بالتعرف على المتغير CHAR-CODE-LIMIT المعرف مسبقا Predefined Variable داخل اللغة، ماهيته وقيمته التي يأخذها.

وسندخل في درسنا مباشرة على بركة الله:)

أولا: ما هو الرمز Character ؟!

التعريف موجود في قاموس الموسوعة العربية على الوصلة التالية:

http://www.c4arab.com/qamoos/mean.php?word=Character

ولا يختلف هذا المفهوم للحرف أو تعميما الرمز Character بين جميع لغات البرمجة بما فيها لغتنا الجميلة .LISP

ثانيا: دوال التعامل مع الرموز :Characters

سنتناول في هذا الدرس معظم الدوال المعرفة داخل لغة LISP والمخصصة لغرض التعامل مع الرموز أو الأحرف..
وإليكم تعدادا ثم تفصيلا لجميع الدوال التي سنتناولها في هذا الدرس:

1- الدالة CHARACTER
2- الدالة CHARACTERP
3- الدالة ALPHA-CHAR-P
4- الدالة ALPHANUMERICP
5- الدالة GRAPHIC-CHAR-P
6- الدالة CHAR-UPCASE
7- الدالة CHAR-DOWNCASE
8- الدالة UPPER-CASE-P
9- الدالة LOWER-CASE-P

10- الدالة BOTH-CASE-P
11- الدالة CHAR-CODE
12- الدالة CHAR-INT
13- الدالة CODE-CHAR
14- الدالة NAME-CHAR

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

1- الدالة CHARACTER

الوظيفة:

تستقبل هذه الدالة رمز أو حرف وحيد، وتعود به مسبوقا بالعلامة #\ إذا لم يكن مسبوقا بها أو إذا كان مسبوقا فقط بـ\ .. والعلامة #\ تدل على أن ما وراءها هو Character.
غير ذلك فإن هذه الدالة تعود بخطأ ERROR وسنشاهد ذلك في الأمثلة.

الشكل:

(character character)

أمثلة:

عند إرسال رمزا حرفيا:

```
CL-USER 1 > (character #\a)
#\a

CL-USER 2 > (character "a")
#\a

CL-USER 3 > (character 'a)
#\A

CL-USER 4 > (character '\a)
#\a
```

أما عندما نرسل رقما فإن رسالة الخطأ التالية ستظهر:

```
CL-USER 6 > (character 65.)

Error: 65 is not a designator for an object of type CHARACTER.
```

وكذلك عندما نرسل أكثر من حرف فإن رسالة الخطأ التالية ستظهر:

```
CL-USER 8 > (character 'apple)

Error: APPLE is not a designator for an object of type CHARACTER.
```

2- الدالة CHARACTERP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة عن رمز أو حرف وحيد ومسبوق بالعلامة #\.
تعود بالقيمة Nil (False) خلاف ذلك.

الشكل:

(characterp character)

أمثلة:

```
CL-USER 1 > (characterp #\a)
T
```

```
CL-USER 6 > (characterp #\=)
T

CL-USER 7 > (characterp #\$)
T

CL-USER 8 > (characterp #\!)
T


CL-USER 2 > (characterp 'a)
NIL

CL-USER 3 > (characterp "a")
NIL

CL-USER 4 > (characterp 65.)
NIL
```

فنلاحظ في الأمثلة السابقة أن LISP يتعرف على الرمز Character عندما يكون مسبوقا بالعلامة #\ وغير ذلك سيعود بالقيمة NIL كما رأينا.

في الأمثلة التالية ستظهر لنا رسالة خطأ لكوننا أرسلنا أكثر من رمزا في الوقت ذاته لهذه الدالة. هذه الدوال تُعنى فقط بالرمز أو الحرف فقط، أما السلسلة التي تتكون من أكثر من رمز أو حرف وقد يتخللها أرقام أيضا وهي ما تسمى بالـStrings فسيتم التعرف على دوالها في الدروس القادمة.

```
CL-USER 9 > (characterp #\!!)

Error: Wrong character name: !!.
  1 (abort) Return to level 0.
  2 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,  or :? for other options

CL-USER 9 : 1 > :c 2


CL-USER 10 > (characterp #\aaa)

Error: Wrong character name: aAA.
  1 (abort) Return to level 0.
  2 Return to top loop level 0.

Type :b for backtrace, :c <option number> to proceed,  or :? for other options

CL-USER 10 : 1 > :c 2

CL-USER 11 >
```

3- الدالة ALPHA-CHAR-P

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة حرف Alphabetic وحيد ومسبوق بالعلامة #\.
وتعود بالقيمة Nil (False) خلاف ذلك (فيما لو أرسلنا أحد الرموز أو الأرقام على سبيل المثال).

الشكل:

(alpha-char-p character)

أمثلة:

```
CL-USER 11 > (alpha-char-p #\a)
T

CL-USER 12 > (alpha-char-p #\5)
NIL

CL-USER 13 > (alpha-char-p #\@)
NIL
```

4- الدالة ALPHANUMERICP

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا كان الكائن المرسل لها عبارة حرف Alphabetic أو عدد Numeric وحيد ومسبوق بالعلامة #\.
وتعود بالقيمة Nil (False) خلاف ذلك (فيما لو أرسلنا أحد الرموز على سبيل المثال).

الشكل:

(alphanumericp character)

أمثلة:

```
CL-USER 14 > (alphanumericp #\Z)
T

CL-USER 15 > (alphanumericp #\9)
T

CL-USER 16 > (alphanumericp #\@)
NIL
```

5- الدالة GRAPHIC-CHAR-P

الوظيفة:

تعود هذه الدالة بالقيمة (True (T إذا كان الكائن المرسل لها عبارة رمز رسومي Graphic Character (أي رمز من الرموز الموجودة على لوحة المفاتيح من مثل ! @ $ % ^ & ~ وغيرها أو حرفا Alphabetic أو عددا .(numeric وتعود بالقيمة (False (Nil خلاف ذلك.

الشكل:

(graphic-char-p character)

أمثلة:

```
CL-USER 17 > (graphic-char-p #\G)
T

CL-USER 18 > (graphic-char-p #\g)
T

CL-USER 19 > (graphic-char-p #\#)
T

CL-USER 20 > (graphic-char-p #\2)
T

CL-USER 21 > (graphic-char-p #\Space)
T

CL-USER 22 > (graphic-char-p #\Newline)
NIL
```

6- الدالة CHAR-UPCASE

الوظيفة:

تقوم هذه الدالة بتحويل الحرف إلى صيغة الحروف الكبيرة Capital Letters في اللغة الإنجليزية من مثل A, B, C وهكذا....

حالات خاصة:

- عندما تستقبل هذه الدالة حرفا كبيرا Capital فإنها تعود به كما هو.
- كذلك عندما تستقبل هذه الدالة رمزا ليس حرفيا، أو عندما تستقبل رقما فإنها تعود به كما هو ولاحظ ذلك خلال الأمثلة.

الشكل:

(char-upcase character)

أمثلة:

```
CL-USER 23 > (char-upcase #\a)
#\A

CL-USER 24 > (char-upcase #\A)
#\A

CL-USER 25 > (char-upcase #\9)
#\9

CL-USER 26 >  (char-upcase #\@)
#\@
```

7- الدالة CHAR-DOWNCASE

الوظيفة:

هذه الدالة لها عكس وظيفة الدالة السابقة، حيث تقوم بتحويل الحرف إلى صيغة الحروف الصغيرة Small Letters في اللغة الإنجليزية من مثل a, b, c وهكذا....

حالات خاصة:

- عندما تستقبل هذه الدالة حرفا صغيرا Small فإنها تعود به كما هو.

- كذلك عندما تستقبل هذه الدالة رمزا ليس حرفيا، أو عندما تستقبل رقما فإنها تعود به كما هو، ولاحظ ذلك خلال الأمثلة.

الشكل:

(char-downcase character)

أمثلة:

```
CL-USER 27 >  (char-downcase #\a)
#\a

CL-USER 28 > (char-downcase #\A)
#\a

CL-USER 29 > (char-downcase #\9)
#\9

CL-USER 30 > (char-downcase #\@)
#\@
```

8- الدالة UPPER-CASE-P

الوظيفة:

تعود هذه الدالة بالقيمة (True (T إذا كان الحرف المرسل لها حرف كبير .Capital Letter
وتعود بالقيمة (False (Nil خلاف ذلك، أي إذا أرسلنا حرفا صغيرا Small Letter أو رقما أو رمزا آخر.

الشكل:

(upper-case-p character)

أمثلة:

```
CL-USER 31 > (upper-case-p #\A)
T

CL-USER 32 > (upper-case-p #\a)
NIL

CL-USER 33 > (upper-case-p #\5)
NIL

CL-USER 34 > (upper-case-p #\@)
NIL
```

9- الدالة LOWER-CASE-P

الوظيفة:

تعود هذه الدالة بالقيمة (True (T إذا كان الحرف المرسل لها حرف صغير .Small Letter

وتعود بالقيمة False (Nil خلاف ذلك، أي إذا أرسلنا حرفا كبيرا Capital Letter أو رقما أو رمزا آخر.

الشكل:

(lower-case-p character)

أمثلة:

```
CL-USER 37 > (lower-case-p #\A)
NIL

CL-USER 38 > (lower-case-p #\a)
T

CL-USER 39 > (lower-case-p #\3)
NIL

CL-USER 40 > (lower-case-p #\$)
NIL
```

10- الدالة BOTH-CASE-P

الوظيفة:

تعود هذه الدالة بالقيمة T (True) إذا أرسلنا لها حرفا Alphabetic حيث أن الحروف هي الرموز الوحيدة التي تأخذ حالتين: إما حرف صغير Small Letter أو حرف كبير Capital Letter.
وتعود بالقيمة False (Nil) خلاف ذلك، أي إذا أرسلنا أي رقم أو رمز آخر.

الشكل:

(both-case-p character)

أمثلة:

```
CL-USER 42 > (both-case-p #\a)
T

CL-USER 43 > (both-case-p #\A)
T

CL-USER 44 > (both-case-p #\^)
NIL

CL-USER 45 > (both-case-p #\5)
NIL
```

11- الدالة CHAR-CODE

الوظيفة:

تعود هذه الدالة بقيمة الكود الرقمي للرمز المرسل لها أيا كان نوعه (حرف – رقم –رمز).

الشكل:

(char-code character)

أمثلة:

```
CL-USER 46 > (char-code #\a)
97

CL-USER 47 > (char-code #\A)
65

CL-USER 48 > (char-code #\3)
51

CL-USER 49 > (char-code #\#)
35

CL-USER 50 > (char-code #\$)
36

CL-USER 51 > (char-code #\~)
126
```

12- الدالة CHAR-INT

الوظيفة:

تشبه هذه الدالة سابقتها وتعود بعدد صحيح غير سالب يمثل الشيفرة الخاصة بالرمز المرسل لها أيا كان نوعه (حرف – رقم –رمز).

تختلف هذه الدالة عن سابقتها في أن طريقة حساب الرقم الصحيح العائد من هذه الدالة يعتمد من نسخة LISP إلى أخرى.

الشكل:

(char-int character)

أمثلة:

في نسخة LISP التي لدي نجد أن:

```
CL-USER 52 > (char-int #\A)
65
```

بينما في نسخ أخرى سنجد أن الرقم العائد قد يكون:

```
(char-int #\A) =>  577
(char-int #\A) =>  262145
```

13- الدالة CODE-CHAR

الوظيفة:

تقوم هذه الدالة بالوظيفة المعاكسة للدالة CHAR-CODE ، حيث تأخذ هذه الدالة قيمة الكود الرقمي وتعود بالرمز

المقابل لهذا الرقم أيا كان نوعه (حرف – رقم –رمز).

وكما رأينا في الدالة السابقة الكود الرقمي للرموز يختلف من نسخة إلى أخرى أي أنه Implementation Dependent.

الشكل:

(code-char character)

أمثلة:

```
CL-USER 53 > (code-char 65.)
#\A

CL-USER 54 > (code-char 51)
#\3

CL-USER 55 > (code-char 35.)
#\#

CL-USER 56 > (code-char 126)
#\~

CL-USER 57 > (code-char 0)
#\Null

CL-USER 58 > (code-char (char-code #\Space))
#\Space
```

14- الدالة NAME-CHAR

الوظيفة:

هناك أسماء محجوز داخل لغة LISP تكون خاصة ببعض الرموز ، على سبيل المثال:
الاسم NewLine يكتب للدلالة على سطر جديد
والاسم Space يكتب للدلالة على المسافة البيضاء

تقوم الدالة NAME-CHAR بتحديد ما إذا كان الاسم المرسل لها محجوزا في اللغة لأحد الرموز أم لا.. فإذا كان الاسم المرسل لها محجوزا بالفعل فإنها تعود به أما إذا كان غير محجوز داخل اللغة فإنها تعود بالقيمة Nil.

الشكل:

(name-char name)

أمثلة:

```
CL-USER 60 > (name-char 'space)
#\Space

CL-USER 61 > (name-char "space")
#\Space

CL-USER 62 > (name-char "Space")
#\Space

CL-USER 63 > (name-char "NewLine")
#\Newline

CL-USER 64 > (name-char "newline")
#\Newline

CL-USER 65 > (name-char "copy")
NIL
```

وكما نلاحظ في الأمثلة السابقة فإن لغة LISP غير حساسة لحالة الأحرف فلا فرق مثلا بين أن نكتب الاسم Space أو space..

وهذا مثال متقدم نوعا ما للاستفادة من الدالة السابقة:

```
CL-USER 67 > (let ((x (char-name #\a)))
               (or (not x) (eql (name-char x) #\a)))
T
```

انتهينا من الدوال المتعلقة بالرموز.. وسنتعرف الآن على المتغير CHAR-CODE-LIMIT..

ثالثا: المتغير CHAR-CODE-LIMIT

يعتبر هذا المتغير من المتغيرات المحجوزة داخل لغة LISP (شأنه شأن المتغير PI الذي تعرفنا عليه مسبقا) وتختلف قيمته من نسخة لأخرى..

يعطي هذا المتغير أقصى قيمة للكود الرقمي الممثل للرموز وهي مختلفة من نسخة إلى أخرى، ولكن من الضروري ألاّ تقل هذه القيمة 96.

```
CL-USER 59 > CHAR-CODE-LIMIT
65536
```

وبعد أن تعرفنا في الدرس الماضي على الدوال الخاصة بالرموز CHARACTERS والتي تفيد في إجراء بعض العمليات على الرموز، يهمنا أن نتعرف الآن على مفهوم المقارنة بين رمزين أو أكثر وكذلك الدوال الخاصة بالمقارنة بين الرموز.

أولا: مفهوم المقارنة:

تفيد المقارنة بين الرموز في عمليات وتطبيقات كثيرة جدا.. وسنرى بإذن الله العديد من الأمثلة في القسم المتقدم من هذه الدروس..
يمكننا مثلا ترتيب عنا صره قائمة تحتوي على مجموعة من الحروف تصاعديا أو تنازليا.. أو يمكننا اختبار تحقق شرط تساوي حرفين أو رمزين لإجراء عملية معينة مثلا..

ثانيا: الدوال الخاصة بالمقارنة بين الرموز

هناك 12 دالة تستخدم للمقارنة بين رمزين أو أكثر، وإليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

1- الدالة CHAR=

2- الدالة CHAR\=

3- الدالة CHAR>

4- الدالة CHAR<

5- الدالة CHAR<=

6- الدالة CHAR>=

7- الدالة CHAR-EQUAL

8- الدالة CHAR-NOT-EQUAL

9- الدالة CHAR-LESSP

10- الدالة CHAR-GREATERP

11- الدالة CHAR-NOT-GREATERP

12- الدالة CHAR-NOT-LESSP

والآن سنفصل في الحديث عن وظيفة وماهية كل دالة وطريقة كتابتها مع التزويد بأمثلة وأكواد متعددة للتوضيح والترسيخ..

1- الدالة CHAR=

الوظيفة:

تختبر هذه الدالة تطابق رمزين أو أكثر، ويجب أن يسبق كل رمز بالعلامة الخاصة بالرموز في LISP وهي #\.

- تعود هذه الدالة بالقيمة True T عندما تكون جميع الرموز المرسلة لها متطابقة.
- بينما تعود بالقيمة False Nil عندما لا يتطابق اثنان أو أكثر من بين الرموز المعطاة لها.

الشكل:

(char= character[s])

أمثلة:

```
CL-USER 72 > (char= #\#)
T

CL-USER 73 > (char= #\d #\d)
T

CL-USER 74 > (char= #\A #\a)
NIL

CL-USER 75 > (char= #\d #\x)
NIL

CL-USER 76 > (char= #\d #\D)
NIL

CL-USER 79 > (char= #\d #\d #\d #\d)
T

CL-USER 80 > (char= #\d #\d #\x #\d)
NIL

CL-USER 83 > (char= #\d #\y #\x #\c)
NIL

CL-USER 84 > (char= #\d #\c #\d)
NIL
```

2- الدالة CHAR\=

الوظيفة:

تختبر هذه الدالة عدم تطابق رمزين أو أكثر، ويجب أن يسبق كل رمز بالعلامة الخاصة بالرموز في LISP وهي #\.

- **تعود هذه الدالة بالقيمة True T عندما تكون جميع الرموز المرسلة لها غير متطابقة.**
- **بينما تعود بالقيمة False Nil عندما يتطابق اثنان أو أكثر من بين الرموز المعطاة لها.**

الشكل:

(char/= character[s])

أمثلة:

```
CL-USER 85 > (char/= #\d)
T

CL-USER 86 > (char/= #\d #\d)
NIL

CL-USER 87 > (char/= #\d #\x)
T

CL-USER 88 > (char/= #\d #\D)
T

CL-USER 89 > (char/= #\d #\d #\d #\d)
NIL

CL-USER 90 > (char/= #\d #\d #\x #\d)
NIL

CL-USER 91 > (char/= #\d #\y #\x #\c)
T

CL-USER 92 > (char/= #\d #\c #\d)
NIL
```

3- الدالة CHAR<

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون الرموز المرسلة لها مرتبة تصاعديا.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(char< character[s])

أمثلة:

```
CL-USER 93 > (char< #\d)
T

CL-USER 94 > (char< #\d #\x)
T

CL-USER 95 > (char< #\d #\d)
NIL

CL-USER 96 > (char< #\a #\e #\y #\z)
T

CL-USER 97 > (char< #\a #\e #\e #\y)
NIL
```

4- الدالة CHAR>

الوظيفة:

- تقوم هذه الدالة بعكس الوظيفة التي تقوم بها سابقتها، حيث تعود بالقيمة True T عندما تكون الرموز المرسلة لها مرتبة تنازليا.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(char> character[s])

أمثلة:

```
CL-USER 98 > (char> #\e)
T

CL-USER 99 > (char> #\e #\d)
T

CL-USER 100 > (char> #\e #\e)
NIL

CL-USER 101 > (char> #\d #\c #\b #\a)
T

CL-USER 102 > (char> #\d #\d #\c #\a)
NIL

CL-USER 103 > (char> #\b #\d #\c #\a)
NIL

CL-USER 104 > (char> #\e #\d #\b #\c #\a)
NIL
```

5- الدالة CHAR<=

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون الرموز المرسلة لها مرتبة تصاعديا أو يتساوى إثنان أو اكثر منها.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(char<= character[s])

أمثلة:

```
CL-USER 105 > (char<= #\d)
T
```

```
CL-USER 106 > (char<= #\d #\x)
T

CL-USER 107 > (char<= #\d #\d)
T

CL-USER 108 > (char<= #\a #\e #\y #\z)
T

CL-USER 109 > (char<= #\a #\e #\y #\z)
T

CL-USER 110 > (char<= #\a #\e #\e #\y #\y #\z)
T

CL-USER 112 > (char<= #\x #\e #\e #\y #\y #\z)
NIL
```

6- الدالة CHAR>=

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون الرموز المرسلة لها مرتبة تنازليا أو يتساوى إثنان أو اكثر منها.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(char>= character[s])

أمثلة:

```
CL-USER 113 > (char>= #\e)
T

CL-USER 114 > (char>= #\e #\d)
T

CL-USER 115 > (char>= #\e #\e)
T

CL-USER 116 > (char>= #\d #\c #\b #\a)
T

CL-USER 117 > (char>= #\d #\d #\c #\a)
T

CL-USER 118 > (char>= #\e #\d #\b #\c #\a)
NIL
```

والآن سنأتي لست دوال أخرى تقوم بنفس وظائف الدوال السابقة مع فارق بسيط ولكنه هام! .. سيتم توضيح الفارق بمثال بسيط بعد سردها..

7- الدالة CHAR-EQUAL

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR=

8- الدالة CHAR-NOT-EQUAL

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR\=

9- الدالة CHAR-LESSP

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR>

10- الدالة CHAR-GREATERP

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR<

11- الدالة CHAR-NOT-GREATERP

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR=>

12- الدالة CHAR-NOT-LESSP

تقوم هذه الدالة بنفس وظيفة الدالة .CHAR=<

تختلف الدوال CHAR-EQUAL, CHAR-NOT-EQUAL, CHAR-LESSP, CHAR-GREATERP, CHAR-NOT-GREATERP, CHAR-NOT-LESSP عن الدوال CHAR=, CHAR/=, CHAR>, CHAR<, CHAR=>, CHAR=< على الترتيب في الآتي:

الدوال CHAR=, CHAR/=, CHAR>, CHAR<, CHAR=>, CHAR=< :

تكون حساسة لحالة الأحرف هل هي في صيغة lower-case أو في صيغة upper-case فنجد أن:

```
CL-USER 119 > (char= #\a #\A)
NIL
```

فكما نلاحظ تم اعتبار الحرف a مختلفا عن الحرف A لكونهما في حالتين مختلفة وبالتالي عادت الدالة بالقيمة False.

بينما الدوال CHAR-EQUAL, CHAR-NOT-EQUAL, CHAR-LESSP, CHAR-GREATERP, CHAR-NOT-GREATERP, CHAR-NOT-LESSP :

غير حساسة لحالة الأحرف فلا فرق بين أن يكون الحرف

في صيغة lower-case أو في صيغة upper-case فنجد أن:

```
CL-USER 120 > (char-equal #\a #\A)
T
```

أي أنه تم اعتبار حالة التساوي بغض النظر عن حالة الحرف، وبالتالي عادت الدالة بالقيمة True

في هذا الدرس والدرس القادم سأتحدث بالتفصيل عن سلاسل الرموز Strings في LISP ماهيتها وكيفية التعامل معها في LISP وكذلك الدوال المعرفة مسبقا في اللغة للتعامل معها أو إجراء عمليات عليها.

مكونات هذا الدرس:

أولا: تعريف السلاسل الرمزية STRINGS ؟

ثانيا: سلسلة الدوال المختصة بالتعامل مع سلاسل الرموز وهي 12 دالة سيتم شرحها في هذا الدرس.

وسندخل في درسنا مباشرة على بركة الله:)

أولا: ما هي سلسلة الرموز String؟

تستطيع معرفة هذا المصطلح عن طريق زيارة الوصلة التالية على قاموس الموسوعة العربية:

http://www.c4arab.com/qamoos/mean.php?word=Character%20String

هذا المفهوم لسلاسل الرموز لا يختلف بين جميع لغات البرمجة.

ثانيا: دوال التعامل مع سلاســـــل الرموز

Strings Functions..

وإليكم تعدادا ثم تفصيلا للدوال التي سيتم شرحها في هذا الدرس:

1- الدالة MAKE-STRING
2- الدالة STRING
3- الدالة STRINGP
4- الدالة STRING-UPCASE
5- الدالة STRING-DOWNCASE
6- الدالة STRING-CAPITALIZE
7- الدالة NSTRING-UPCASE
8- الدالة NSTRING-DOWNCASE
9- الدالة NSTRING-CAPITALIZE
10- الدالة STRING-TRIM
11- الدالة STRING-LEFT-TRIM
12- الدالة STRING-RIGHT-TRIM

1- الدالة MAKE-STRING

الوظيفة:

تعود هذه الدالة بسلسلة من الرموز بالطول الذي تم تحديده، وإذا تم تحديد العنصر الابتدائي فانه يتم ملئ السلسلة بهذا العنصر .

ملاحظة: هذا الدالة شبيهة جدا بدالة هامة تستخدم لصناعة القوائم lists واسمها MAKE-LIST وسيتم شرحها

في الجزء الخاص بالقوائم لكونها عماد لغتنا الرائعة LISP في القسم المتقدم من هذه الدروس بإذن الله.

الشكل:

(make-string size [:initial-element element-type])

أمثلة:

```
CL-USER 4 > (make-string 10)
"^_^P^A^@^@^@^@^@^P^@"

CL-USER 5 > (setq svar (make-string 10))
"□…□…□□□□^P^@"

CL-USER 6 > svar
"□…□…□□□□^P^@"
```

نلاحظ أنه نتيجة عدم تحديد العنصر الابتدائي الذي يكون من المفترض ملء السلسلة به، فإن LISP سيضع رموزا من خياله لملئ السلسلة بالطول الذي تم تحديده..

```
CL-USER 8 > (make-string 10 :initial-element #\a)
"aaaaaaaaaa"
```

رأينا أنه تم إنشاء سلسلة مكونة من 10 رموز، وسيتم ملؤها مبدئيا بالحرف a. تابع المثال التالي أيضا:

```
CL-USER 9 > (setq xstring (make-string 10 :initial-element #\5))
"5555555555"

CL-USER 10 > xstring
"5555555555"
```

2- الدالة STRING

الوظيفة:

هذه الدالة تعود برمز أو سلسلة من الرموز حسب حالة x الذي تم إرساله لها.

- إذا كان x عبارة عن سلسلة من الرموز String فان الدالة تعود بها.
- إذا كان x عبارة عن رمز Symbol فان هذه الدالة تعود باسمه.
- إذا كان x عبارة عن حرف Character فان هذه الدالة تعود بـString تحتوي على هذا الرمز فقط.

الشكل:

(string x)

أمثلة:

```
CL-USER 1 > (string "already a string")
"already a string"

CL-USER 2 > (string 'elm)
"ELM"

CL-USER 3 > (string #\c)
"c"
```

3- الدالة STRINGP

الوظيفة:

- تعود هذه الدالة بالقيمة True T إذا كان الكائن المرسل لها عبارة عن سلسلة من الرموز String.
- وتعود بالقيمة False Nil خلاف ذلك.

الشكل:

(stringp object)

أمثلة:

```
CL-USER 4 > (stringp "aaaaaa")
T

CL-USER 5 > (stringp "c4arab")
T

CL-USER 6 > (stringp "a")
T
```

لاحظ في المثال السابق أنه من الممكن أن تحتوي السلسلة على رمز أو حرف وحيد، ولكن العكس صحيح! فلا يمكن أن يكون الـCharacter محتويا على أكثر من رمز أو حرف.

```
CL-USER 7 > (stringp #\a)
NIL
```

وهاهي الدالة STRINGP تكشف عن الرمز Character الذي قمنا بإرساله لها لأن علامة الرمز #\ موجودة قبله.

4- الدالة STRING-UPCASE

الوظيفة:

تعود هذه الدالة بنفس السلسلة String المرسلة لها مع استبدال جميع حروفها إلى صيغة الحروف العلوية Upper-case Letters أي الحروف الكبيرة Capital Letters.
وللتوضيح فإن هذه الدالة تؤدي عملها عن طريق تطبيق الدالة char-upcase التي تعرفنا عليها في درس " دوال التعامل مع الرموز Characters " على كل حرف من حروف السلسلة.

الشكل:

(string-upcase string [:start index] [:end index])

أمثلة:

```
CL-USER 1 > (string-upcase "c4arab")
"C4ARAB"
```

```
CL-USER 2 > (string-upcase "Dr. Livingston, I presume?")
"DR. LIVINGSTON, I PRESUME?"

CL-USER 3 > (string-upcase "Dr. Livingston, I presume?" :start 6 :end 10)
"Dr. LiVINGston, I presume?"
```

5- الدالة STRING-DOWNCASE

الوظيفة:

تقوم هذه الدالة بالوظيفة العكسية للدالة السابقة، حيث تعود بنفس السلسلة String المرسلة لها مع استبدال جميع حروفها إلى صيغة الحروف السفلية Down-case Letters أي الحروف الصغيرة Small Letters.
وهذه الدالة تؤدي عملها عن طريق تطبيق الدالة char-downcase التي تعرفنا عليها في درس " دوال التعامل مع الرموز Characters " على كل حرف من حروف السلسلة.

الشكل:

(string-downcase string [:start index] [:end index])

أمثلة:

```
CL-USER 4 > (string-downcase "C4ARAB")
"c4arab"

CL-USER 5 > (string-downcase "Dr. Livingston, I presume?")
"dr. livingston, i presume?"

CL-USER 6 > (string-downcase "DR. LIVINGSYON, I presume?" :start 6 :end 10)
"DR. LIvingSYON, I presume?"
```

6- الدالة STRING-CAPITALIZE

الوظيفة:

تعود هذه الدالة بنسخة من السلسلة String المرسلة لها مع استبدال حالة الحرف الموجود في بداية كل كلمة إلى حرف علوي Upper-Case Letter.

الشكل:

(string-capitalize string [:start index] [:end index])

أمثلة:

```
CL-USER 7 > (string-capitalize "elm 13c arthur;fig don't")
"Elm 13c Arthur;Fig Don'T"

CL-USER 8 > (string-capitalize " hello ")
" Hello "

CL-USER 9 > (string-capitalize " c4arab ")
" C4arab "

CL-USER 11 > (string-capitalize 'kludgy-hash-search)
"Kludgy-Hash-Search"

CL-USER 12 > (string-capitalize "DON'T!")
"Don'T!"

CL-USER 13 > (string-capitalize "pipe 13a, foo16c")
"Pipe 13a, Foo16c"

CL-USER 14 > (string-capitalize
               "occlUDeD cASEmenTs FOreSTAll iNADVertent DEFenestraTION")
"Occluded Casements Forestall Inadvertent Defenestration"
```

الدوال STRING-UPCASE, STRING-DOWNCASE, STRING-CAPITALIZE السابقة تقوم بإعادة السلسلة دون تحديث المتغير الذي يحويها.. أي أنه على سبيل

المثال، لو وضعنا السلسلة "C4arab" كقيمة لمتغير ما وليكن x كالتالي:

```
CL-USER 15 > (setq x "C4arab")
"C4arab"

CL-USER 16 > x
"C4arab"
```

ثم قمنا بتطبيق الدوال السالفة الذكر على هذا المتغير كالآتي:

```
CL-USER 17 > (string-upcase x)
"C4ARAB"

CL-USER 18 > x
"C4arab"

CL-USER 19 > (string-downcase x)
"c4arab"

CL-USER 20 > x
"C4arab"

CL-USER 21 > (string-capitalize x)
"C4arab"

CL-USER 22 > x
"C4arab"
```

فكما نلاحظ أن المتغير x لا يزال يحتفظ بنفس شكل السلسلة "C4arab" داخله بالرغم من تطبيق الدوال السابقة عليه.. ماذا لو أردنا الاحتفاظ بالشكل الجديد للسلسلة؟!!

هناك طريقتين لذلك:

الطريقة الأولى:

استخدام setq لنجعل الشكل الجديد للسلسلة والناتج عن تطبيق إحدى الدوال السابقة يتخزن داخل المتغير x .. فمثلا لو أردنا تطبيق الدالة string-upcase على السلسلة المخزنة في x ومن ثم الاحتفاظ بالعائد من الدالة داخل هذا المتغير نكتب:

```
CL-USER 23 > (setq x (string-upcase x))
"C4ARAB"

CL-USER 24 > x
"C4ARAB"
```

الطريقة الثانية:

استخدام الدوال NSTRING- NSTRING-UPCASE, والتي تقوم DOWNCASE, NSTRING-CAPITALIZE بنفس وظيفة الدوال السابقة ولكن مع تخزين السلسلة العائدة في المتغير المرسل لها..

7- الدالة NSTRING-UPCASE

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING-UPCASE مع تخزين السلسلة الجديدة في المتغير المرسل لها.

الشكل:

(nstring-upcase string [:start index] [:end index])

أمثلة:

بفرض أننا قمنا بتخزين السلسلة C4arab.Com في المتغير str على النحو الآتي:

```
CL-USER 1 > (setq str (copy-seq "C4arab.Com"))
"C4arab.Com"
```

وبتطبيق الدالة nstring-upcase:

```
CL-USER 2 > (nstring-upcase str)
"C4ARAB.COM"
```

سنجد أن المتغير str احتفظ بالسلسلة الجديدة العائدة من الدالة السابقة:

```
CL-USER 3 > str
"C4ARAB.COM"
```

8- الدالة NSTRING-DOWNCASE

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING-DOWNCASE مع تخزين السلسلة الجديدة في المتغير المرسل لها.

الشكل:

(nstring-downcase string [:start index] [:end index])

أمثلة:

```
CL-USER 4 > (setq str (copy-seq "0123ABCD890a"))
"0123ABCD890a"

CL-USER 5 > str
"0123ABCD890a"

CL-USER 6 > (nstring-downcase str :start 5 :end 7)
"0123AbcD890a"

CL-USER 7 > str
"0123AbcD890a"
```

9- الدالة NSTRING-CAPITALIZE

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING-CAPITALIZE مع تخزين السلسلة الجديدة في المتغير المرسل لها.

الشكل:

(nstring-capitalize string [:start index] [:end index])

أمثلة:

```
CL-USER 8 > (setq str (copy-seq "c4arab.com is best"))
"c4arab.com is best"

CL-USER 9 > str
"c4arab.com is best"
CL-USER 10 > (nstring-capitalize str)
"C4arab.Com Is Best"

CL-USER 11 > str
"C4arab.Com Is Best"
```

المزيد من الدوال:

10- الدالة STRING-TRIM

الوظيفة:

تعود هذه الدالة بجزء من السلسلة المرسلة لها يحتوي على جميع الحروف الموجودة في السلسلة الأصلية باستثناء المحتوي على مع إزالة الأجزاء المحتوية على الـ character-bag من بداية أو نهاية السلسلة.

الـ character-bag هو الجزء الذي نرغب بمسحة وإزالته من السلسلة ونقوم بإرساله إلى هذه الدالة مع السلسلة كما سنرى في الأمثلة..

الشكل:

(string-trim character-bag string)

أمثلة:

```
CL-USER 12 > (string-trim "abc" "abcaakaaakabcaaa")
"kaaak"

CL-USER 13 > (string-trim '(#\Space #\Tab #\Newline) " garbanzo beans
        ")
"garbanzo beans"

CL-USER 14 > (string-trim " (*)" " ( *three (silly) words* ) ")
"three (silly) words"
```

11- الدالة STRING-LEFT-TRIM

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING-TRIM ولكنها تقوم بعملية الإزالة للـ character-bag من بداية الـString فقط..

الشكل:

(string-left-trim character-bag string)

أمثلة:

```
CL-USER 15 > (string-left-trim "abc" "labcabcabc")
"labcabcabc"

CL-USER 16 > (string-left-trim " (*)" " ( *three (silly) words* ) ")
"three (silly) words* ) "
```

12- الدالة STRING-RIGHT-TRIM

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING-TRIM ولكنها تقوم بعملية الإزالة للـ character-bag من نهاية الـString فقط..

الشكل:

(string-left-trim character-bag string)

أمثلة:

```
CL-USER 17 > (string-right-trim "abc" "labcabcabc")
"l"

CL-USER 18 > (string-right-trim " (*)" " ( *three (silly) words* ) ")
" ( *three (silly) words"
```

وبعد أن تعرفنا في الدرس الماضي على الدوال الخاصة بسلاسل الرموز Strings والتي تفيد في إجراء بعض العمليات على السلاسل الرمزية، يهمنا الآن أن نتعرف على الدوال الخاصة بالمقارنة بين سلسلتي رموز أو أكثر.

وإليكم تعدادا ثم تفصيلا للدوال التي سنتناولها في هذا الدرس:

1- الدالة =STRING

2- الدالة =\STRING

3- الدالة <STRING

4- الدالة >STRING

5- الدالة =<STRING

6- الدالة STRING>=

7- الدالة STRING-EQUAL

8- الدالة STRING-NOT-EQUAL

9- الدالة STRING-LESSP

10- الدالة STRING-GREATERP

11- الدالة STRING-NOT-GREATERP

12- الدالة STRING-NOT-LESSP

1- الدالة STRING=

الوظيفة:

تختبر هذه الدالة تطابق سلسلتين أو أكثر..

- تعود هذه الدالة بالقيمة True T عندما يكون لهما نفس الطول وتحتويان على نفس الرموز Characters بنفس المواقع Positions داخل كلا السلسلتين.
- بينما تعود بالقيمة False Nil عندما يختل أحد الشرط السابقة.

الشكل:

(string= string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 19 > (string= "foo" "foo")
3

CL-USER 20 > (string= "foo" "Foo")
NIL

CL-USER 21 > (string= "c4arab" "c4arab")
6

CL-USER 22 > (string= "foo" "bar")
NIL
```

```
CL-USER 23 > (string= "together" "frog" :start1 1 :end1 3 :start2 2)
3

CL-USER 24 > (string= "abcd" "01234abcd9012" :start2 5 :end2 9)
4
```

```
CL-USER 25 > (string= "abcd" "01234abcd9012" :start1 1 :end1 3 :start2 5 :end2 9)
NIL
```

2- الدالة STRING\=

الوظيفة:

تختبر هذه الدالة عدم تطابق سلسلتين أو أكثر..

- **تعود هذه الدالة بالقيمة True T في حال عدم تطابق السلسلتان.**
- **بينما تعود بالقيمة False Nil عندما تتطابقان (أي عندما يكون لهما نفس الطول وتحتويان على نفس الرموز Characters بنفس المواقع Positions داخل كلتيهما).**

الشكل:

(string/= string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 28 > (string/= "foo" "foo")
NIL

CL-USER 29 > (string/= "C4arab" "c4arab")
0

CL-USER 30 > (string/= "SaRa" "SaRaaaaaa")
4
```

```
CL-USER 31 > (string= "abcd" "01234abcd9012" :start2 5 :end2 9)
4

CL-USER 32 > (string/= "abcd" "01234abcd9012" :start2 5 :end2 9)
NIL

CL-USER 33 > (string/= "abcd" "01234abcd9012" :start1 2 :start2 5 :end2 9)
2
```

3- الدالة STRING<

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون السلسلة الأولى أقل من السلسلة الثانية.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(string< string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 34 > (string< "aaaa" "aaab")
3

CL-USER 35 > (string< "c4arab" "c4arab.com")
6


CL-USER 36 > (string< "c4arab" "c4arab")
NIL

CL-USER 37 > (string< "bbbba" "aaab")
NIL
```

4- الدالة STRING>

الوظيفة:

- تقوم هذه الدالة بعكس الوظيفة التي تقوم بها سابقتها، حيث تعود بالقيمة True T عندما تكون السلسلة الأولى أكبر من السلسلة الثانية.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(string> string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 38 > (string> "bbbba" "aaab")
0


CL-USER 39 > (string> "c4arab" "c4arab.com")
NIL

CL-USER 40 > (string> "c4arab" "c4arab")
NIL

CL-USER 41 > (string> "aaab" "baba")
NIL
```

5- الدالة STRING<=

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون السلسلة الأولى أقل من السلسلة الثانية أو متطابقة معها.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(string<= string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 42 > (string<= "aaaa" "aaab")
3

CL-USER 43 > (string<= "c4arab" "c4arab")
6

CL-USER 44 > (string<= "bbba" "aaab")
NIL
```

6- الدالة STRING>=

الوظيفة:

- تعود هذه الدالة بالقيمة True T عندما تكون الرموز السلسلة الأولى أكبر من السلسلة الثانية أو متطابقة معها.
- بينما تعود بالقيمة False Nil خلاف ذلك.

الشكل:

(string>= string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 45 > (string>= "bbba" "aaab")
0

CL-USER 46 > (string>= "c4arab" "c4arab")
6

CL-USER 47 > (string>= "aaab" "bbba")
NIL
```

يلاحظ أن الدوال السابقة تأخذ حالة الأحرف Lower-Case و Upper-Case في الاعتبار فلا تتطابق السلسلتان "C4arab" و "c4arab" لتغير حالة الحرف C من كبير Capital إلى صغير .. Small
وبالتالي لا بد أن نتعرف على الدوال التالية:

7- الدالة STRING-EQUAL

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING=.. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-equal string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 51 > (string= "C4arab" "C4ARAB")
NIL

CL-USER 52 > (string-equal "C4arab" "C4ARAB")
6
```

8- الدالة STRING-NOT-EQUAL

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING \=. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-not-equal string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 56 > (string\= "C4arab" "C4arab")
6

CL-USER 57 > (string-not-equal "C4arab" "C4arab")
NIL
```

9- الدالة STRING-LESSP

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING <. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-lessp string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 69 > (string< "C" "c")
0

CL-USER 70 > (string-lessp "C" "c")
NIL


CL-USER 73 > (string-lessp "012AAAA789" "01aaab6" :start1 3 :end1 7
                                    :start2 2 :end2 6)
6
```

10- الدالة STRING-GREATERP

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING >. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-greaterp string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 74 > (string-greaterp "012AAAA789" "01aaab6" :start1 3 :end1 7
                                    :start2 2 :end2 6)
NIL
```

11- الدالة STRING-NOT-GREATERP

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING =>. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-not-greaterp string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 75 > (string-not-greaterp "Abcde" "abcdE")
5
```

12- الدالة STRING-NOT-LESSP

الوظيفة:

تقوم هذه الدالة بنفس وظيفة الدالة STRING >=. مع تجاهل حالة الأحرف فلا فرق بين أن يكون الحرف كبيرا Capital أم صغيرا Small..

الشكل:

(string-not-lessp string1 string2 [:start1 index] [:end1 index] [:start2 index] [:end2 index])

أمثلة:

```
CL-USER 76 > (string-not-lessp "Abcde" "abcdE")
5
```

سنتعرف في هذا الدرس على الدوال المنطقية وكيفية إجراء العمليات المنطقية بين الكائنات من أي نوع في لغة LISP.

1- الدالة AND

الوظيفة:

تقوم هذه الدالة بإجراء العملية المنطقية Logical AND بين الكائنات Objects أو تعميما بين النماذج Forms المرسلة لهذه الدالة.

يتم عمل تقييم Evaluation لهذه النماذج بالترتيب من اليسار إلى اليمين.

عندما تكون نتيجة تقييم أحد هذه الكائنات أو النماذج هي القيمة False NIL فستعود الدالة AND بالقيمة False NIL أيضا دون أن يتم عمل تقييم للنماذج أو الكائنات التي تليه.

عندما ينتهي تقييم جميع النماذج أو الكائنات، وتكون نتيجة تقييمها جميعا القيمة True T (أو مايدل عليها كما سأوضح بعد قليل) فإن AND تعود بالقيمة التي عاد بها آخر نموذج Form تم تقييمه..

إذن يجب أن نلاحظ جيدا بأنه ليس شرطا أن يعود النموذج بالقيمة True T بل إن أي نتيجة عائدة من النموذج تشير إلى أنه تم عمل تقييم لهذا النموذج (رقم، رمز، سلسلة رموز، قائمة أو غيرها...) وبالتالي تعتبر الدالة AND أن هذا النموذج عاد بالقيمة True T.

الشكل:

(and [(form[s])])

أمثلة:

القيمة الافتراضية للدالة AND هي T:

```
CL-USER 10 > (and)
T
```

وهنا تستخدم الدالة AND للتحقق من أن العائد من جميع الدوال الموجودة هو T:

```
CL-USER 2 > (and (<= 2 3) (> 8 7 6))
T

CL-USER 3 > (and (<= 2 3) (> 6 7 8))
NIL
```

لو أسندنا القيمة 1 إلى المتغيرات الثلاث temp1, temp2, temp3 كالآتي:

```
CL-USER 4 > (setq temp1 1 temp2 1 temp3 1)
1
```

ثم قمنا بعمل AND بين عدة نماذج تستخدم الدالة incf كالآتي:

```
CL-USER 5 > (and (incf temp1) (incf temp2) (incf temp3))
2
```

فإننا نلاحظ أن العائد من الدالة هو الرقم 2 وهو ناتج التقييم لآخر Form كما أسلفنا.

وهنا المزيد من الأمثلة:

```
CL-USER 6 > (and (eql 2 temp1) (eql 2 temp2) (eql 2 temp3))
T

CL-USER 7 > (decf temp3)
1

CL-USER 8 > (and (decf temp1) (decf temp2) (eq temp3 'nil) (decf temp3))
NIL

CL-USER 9 > (and (eql temp1 temp2) (eql temp2 temp3))
T
```

2- الدالة OR

الوظيفة:

تقوم هذه الدالة بإجراء العملية المنطقية Logical OR بين الكائنات Objects أو تعميما بين النماذج Forms المرسلة لهذه الدالة.

يتم عمل تقييم Evaluation لهذه النماذج بالترتيب من اليسار إلى اليمين.

عندما تكون نتيجة تقييم أحد هذه الكائنات أو النماذج القيمة هي True T فستعود الدالة OR مباشرة بالقيمة True T أيضا بدون إكمال عملية تقييم النماذج أو الكائنات التي تليه.

عندما ينتهي تقييم جميع النماذج أو الكائنات، وتكون نتيجة تقييمها جميعا القيمة True T (أو مايدل عليها كما

سأوضح بعد قليل) فإن OR تعود بالقيمة التي عاد بها آخر نموذج Form تم تقييمه..

إذن يجب أن نلاحظ جيدا بأنه ليس شرطا أن يعود النموذج بالقيمة True T بل إن أي نتيجة عائدة من النموذج (رقم، رمز، سلسلة رموز، قائمة أو غيرها...) تشير إلى أنه تم عمل تقييم لهذا النموذج وبالتالي تعتبر الدالة OR أن هذا النموذج عاد بالقيمة True T.

الشكل:

(or [(form[s])])

أمثلة:

القيمة الافتراضية للدالة OR هي NIL.

```
CL-USER 11 > (or)
NIL
```

إذا قمنا بإسناد القيم nil و 10 و 20 و30 على التوالي للمتغيرات temp0, temp1, temp2, temp3 ثم قمنا بعمل OR بين المتغيرين temp0 و temp1 مثلا مع الدالة setq التي ستقوم بإسناد قيمة جديدة للمتغير temp2 فإننا سنلاحظ أن الدالة OR وجدت قيمة على الأقل تساوي T وبالتالي ستعود بالقيمة العائدة من آخر form فيها وهذا يؤكد ما ذكرناه سابقا.

```
CL-USER 12 > (setq temp0 nil temp1 10 temp2 20 temp3 30)
30

CL-USER 13 > (or temp0 temp1 (setq temp2 37))
10

CL-USER 14 > temp2
20
```

لاحظ أيضا المثال التالي والتغيرات التي حدثت في كل متغير والقيمة التي عادت من الدالة OR:

```
CL-USER 15 > (or (incf temp1) (incf temp2) (incf temp3))
11

CL-USER 16 > temp1
11

CL-USER 17 > temp2
20

CL-USER 18 > temp3
30
```

في الأمثلة التالية سنستخدم دالة جديدة مبنية مسبقا داخل اللغة واسمها Values ووظيفتها العودة بكائن Object يحوي قيما متعددة (وهي القيم التي نقوم بإرسالها لهذه الدالة).

```
CL-USER 19 > (or (values) temp1)
11

CL-USER 20 > (or (values temp1 temp2) temp3)
11

CL-USER 21 > (or temp0 (values temp1 temp2))
11
20

CL-USER 22 > (or (values temp0 temp1) (values temp2 temp3))
20
30
```

في المثال السابق ونتيجة أنه أمكن حساب القيم Values في كلا النموذجين:

(values temp0 temp1)
(values temp2 temp3)

فان الدالة OR ستعود بالقيمة العائدة من آخر نموذج فيها 20 و 30 ..

3- الدالة NOT

الوظيفة:

هذه الدالة تقوم بإجراء العملية المنطقية Logical Not.

- تعود هذه الدالة بالقيمة True T عندما تكون نتيجة التقييم للنموذج Form المرسل لها هي False NIL.
- خلاف ذلك تعود بالقيمة NIL. False

الشكل:

(not (form))

أمثلة:

```
CL-USER 23 > (not nil)
T

CL-USER 24 > (not '())
T

CL-USER 25 > (not (integerp 'sss))
T

CL-USER 26 > (not (integerp 1))
NIL

CL-USER 27 > (not 3.7)
NIL

CL-USER 28 > (not 'apple)
NIL
```

نتيجة:

والآن، هل عرفتَ ما هي القيم NIL و T ؟!

تعتبر NIL و T من المتغيرات المحجوزة أو المعرفة مسبقا داخل لغة LISP

predefined variables at lisp i.e NIL and T are

وتقابل NIL القيمة False أو 0 في لغات البرمجة الأخرى.
بينما تقابل T القيمة True أو 1 في لغات البرمجة الأخرى.

سنتعرف في هذا الدرس على الدوال التي تمكن من عمل إجراءات معينة في حال تحقق شروط معينة.. ولا يخفى علينا كمبرمجين أهمية هذه الدوال في لغة LISP شأنها شأن جميع لغات البرمجة الأخرى..
فهي تمكن من عمل تطبيقات صغيرة ولكنها ذكية وذات

فائدة كبيرة وتخدم أمثلة وتطبيقات متعددة في مجال الذكاء الاصطناعي..

سنتعرف في هذا الدرس على دالتان شرطيتان هما:

1- الدالة COND
2- الدالة IF

وهي كفيلة بتحقيق جميع مطالبنا في الدروس المتقدمة من سلسلة LISP ولا تقلق فستكون هناك مشاريع برمجية وأمثلة أكثر متعة، ولكن عندما يكون لدينا أساس في كل شيء :)

على بركة الله :)

1- الدالة COND

الوظيفة:

تقوم هذه الدالة بتنفيذ نموذج Form أو أكثر اعتمادا على نتيجة اختبار نموذج Form معين، سنسميه عادة بـ test-form أو للتبسيط Condition. قد نحتاج إلى اختبار نموذج واحد أو أكثر في الوقت نفسه (أي قد يوجد أكثر من test-form واحد) وعندئذ سنحتاج لنربط بينهم بإحدى العلامات المنطقية AND أو OR.

إذا لم يتم كتابة أي نماذج Forms لتنفيذها عند تحقق شرط الدالة COND فان الدالة COND ستعود بالقيمة العائدة من الـtest-form.

وعندما لا يتحقق الـtest-forms فإن الدالة COND سوف تعود بالقيمة NIL False.

والآن بعد هذا التوضيح البسيط للدالة COND سوف نتعرف على شكل هذه الدالة وأمثلة تطبيقية عليها.

الشكل:

(cond ((condition) (form[s])...)

أمثلة:

مثال1:

المثال التالي يحتوي على Condition واحد وForms واحد يتم تنفيذه عند تحقق هذا الشرط:

```
CL-USER 7 > (cond
              ((> x 0) (print "X is greater than 0"))
             )

"X is greater than 0"
"X is greater than 0"
```

ماذا لو كان الشرط غير متحقق؟!

سنعيد إسناد قيمة أقل من الصفر للمتغير X ثم سنجرب تطبيق الدالة COND مرة أخرى

```
CL-USER 8 > (setq x -3)
-3

CL-USER 9 > (cond
              ((> x 0) (print "X is greater than 0"))
             )
NIL
```

كما رأينا فالدالة COND عادت بالقيمة NIL لعدم تحقق الشرط Condition وهو أن تكون قيمة المتغير X أكبر من الصفر.

مثال2:

سنقوم في هذا المثال بوضع أكثر من Condition ونربط بينهم مرة بالعلامة المنطقية AND ومرة بالعلامة المنطقية OR وسيكون هناك Form واحد للطباعة.

آه تذكرت! بخصوص دالة الطباعة لا تقلق فسوف أشرحها أيضا في الدرس القادم إن شاء الله.

```
CL-USER 10 > (setf x 3)
3

CL-USER 11 > (cond
              ((and (> x 0) (< x 5)) (print "X is greater then 0 and less than 5"))
             )

"X is greater then 0 and less than 5"
"X is greater then 0 and less than 5"
```

مثال3:

```
CL-USER 12 > (setf x -5)
-5

CL-USER 13 > (cond
               ((or (> x 0) (> x -10))
                (print "X is positive or greater than the negative integer 10")
               )
              )

"X is positive or greater than the negative integer 10"
"X is positive or greater than the negative integer 10"
```

مثال4:

```
CL-USER 14 > (setf x -11)
-11

CL-USER 15 > (cond
              ((or (> x 0) (> x -10))
               (print "X is positive or greater than the negative integer 10")
              )
             )
NIL
```

مثال5:

سنقوم في هذا المثال بتنفيذ أكثر من form عند تحقق الـtest-form أو الـCondition.

```
CL-USER 31 > (cond
              ((stringp str)
               (progn
                 (print "This is a string")
                 (setq str "C4arab is the best")
                 (print str)
               )
              )
             )

"This is a string"
"C4arab is the best"


CL-USER 32 > str
"C4arab is the best"
```

إذن ماذا تلاحظ في المثال السابق؟

1. **الدالة COND تقبل بوجود أكثر من Form بمساعدة دالة أخرى هي الدالة PROGN**
2. **لكتابة أكثر من Form واحد بحيث يتم تنفيذها جميعا عند تحقق شرط الدالة COND نكتب PROGN ثم نفتح قوسين دائريين ( ) ونكتب بداخلها كل الـForms**

التي نرغب بتنفيذها مع وضع أقواس دائرية ( ) في بداية ونهاية كل form كما لاحظنا..

3. لا تنس ترتيب برنامجك وجعل الـStructure-Style له جيد أي أن تكتب الدالة على عدة سطور وفي عدة مستويات حتى يكون الكود أكثر وضوحا More Readable and clearable.

مثال6:

سنتعرف في أول درس من دروس القسم المتقدم من سلسلة LISP على كيفية تعريف الدوال من قبل المبرمج.. وسأعرف لكم العديد جدا من الدوال وسنبني العديد من التطبيقات الذكية والصغيرة، لكني أحببت وضع المثال هذا هنا حتى يكون في مكانه:)

في البداية، سنقوم بتعريف دالة بسيطة جدا اسمها select-options وتقوم باستخدام الدالة COND في عملها..

```
CL-USER 18 > (defun select-options ()
               (cond ((= a 1) (setq a 2))
                     ((= a 2) (setq a 3))
                     ((and (= a 3) (floor a 2)))
                     (t (floor a 3))))
Warning: Syntactic warning for form A:
   A assumed special.
SELECT-OPTIONS
```

وبعد تعريف الدالة بفرض أننا أسندنا القيمة واحد للمتغير a:

```
CL-USER 19 > (setq a 1)
1
```

سنقوم باستدعاء الدالة عدة مرات وبناء على طبيعة تحقق شرط الـدالة COND راقب النواتج العائدة من الدالة select-options في كل مرة:

```
CL-USER 20 > (select-options)
2

CL-USER 21 > (select-options)
3

CL-USER 22 > (select-options)
1
```

والآن بفرض أننا وضعنا القيمة 5 للمتغير a ثم استدعينا الدالة مرة أخرى:

```
CL-USER 23 > (setq a 5)
5


CL-USER 24 > (select-options)
1
2
```

وبعد أن تعرفنا على الدالة COND بقي لنا أن نتعرف على دالة أخرى هي الدالة If

2- الدالة IF

الوظيفة:

تسمح هذه الدالة بتنفيذ إجراء معين اعتمادا على تحقق test-form أو Condition وحيد..
ولذا عند وجود أكثر من Condition فإنه لابد أن نربطهم بالعلامة المنطقية AND أو OR حسب حالة وشكل الشرط الذي نريده.

وكما سنلاحظ في شكل الدالة فإن كتابة جزء خاص بالـ else يعد اختياريا يمكن كتابته أو تركه ولذا وضعنا هذا الجزء داخل أقواس مربعة [ ] للدلالة على أن ما بينهما اختياري.

وآلية عمل هذه الدالة والنتائج المحتملة هي كالتالي:

- عند تحقق الـشرط Condition أو الـtest-form فان الجزء then-part يتم تنفيذه..
- أما عند عدم تحقق الشرط فان الجزء else-part يتم تنفيذه اذا كان موجودا.. أما في حال عدم وجوده فان الدالة if ستعود بالقيمة NIL.

الشكل:

(if (condition) (then-part) [(else-part)])

أمثلة:

مثال1:

افترض أننا أسندنا قيمة معينة للمتغير x ولتكن القيمة العددية 3 ثم قمنا باستخدام الدالة if للتحقق من أن قيمة x أكبر من الصفر فإذا ماكان الأمر كذلك فإنه سيتم طباعة العبارة

"X is greater than 0"

```
CL-USER 36 > (setf x 3)
3
```

```
CL-USER 43 > (if (> x 0) (print "X is greater than 0"))

"X is greater than 0"
"X is greater than 0"
```

مثال2:

في هذا المثال نشترط أن يكون X عدد صحيح وأكبر من أو يساوي 5
فاذا ما وضعنا لـX قيمة أقل من 5، أو قيمة أكبر من 5 ولكنها عشرية مثلا 5.5 فإن الدال IF ستعود بالقيمة NIL.

```
CL-USER 44 > (setf x 5)
5

CL-USER 45 > (if
                (and (integerp x) (>= x 5))
                (print "X is equal or greater then 5")
             )

"X is equal or greater then 5"
"X is equal or greater then 5"
```

مثال3:

```
CL-USER 46 > (setf x 10)
10

CL-USER 47 > (setf y -10)
-10

CL-USER 48 > (if
                 (or (> x 0) (> y 0))
                 (progn
                   (print "Either of the numbers is greater than 0")
                   (incf x)
                   (incf y)
                 )
               )

"Either of the numbers is greater than 0"
-9



CL-USER 49 > x
11

CL-USER 50 > y
-9
```

مثال4:

حالات خاصة:

هذا المثال يبين أن المعرفة مسبقا في اللغة أو التي نقوم بتعريفها تأخذ القيمة T.. ماعدا المتغير NIL المعرف في اللغة فإنه يأخذ القيمة NIL..

لذا لاحظ المثال التالي:

```
CL-USER 51 > (if t 1)
1

CL-USER 52 > (if nil 1 2)
2
```

مثال4:

الدالة test المعرفة في هذا المثال تقوم بعمل حلقة تكرارية داخل عناصر القائمة ((t nil 1 (a b c –لاحظ أن القائمة يمكن أن تحتوي على قوائم أخرى– فيتم تناول عناصر هذه القائمة واحدا تلو الآخر عن طريق الدالة dolist وفي كل مرة نتناول في أحد هذه العناصر سنسميه بالاسم truth-value..
وخلال الحلقة dolist سيتم تنفيذ two forms:

الأول: IF

سيتم فيه اختبار truth-value:
فإذا كانت قيمته T فسيتم طباعة True
أما إذا كانت قيمته NIL فسيتم طباعة False

الثاني: PRINT

وسيتم فيه طباعة قيمة truth-value أيا كانت.

```
CL-USER 54 > (defun test ()
               (dolist (truth-value '(t nil 1 (a b c)))
                 (if truth-value (print 'true) (print 'false))
                 (prin1 truth-value)))
TEST
```

وعند استدعاء الدالة test فإنها ستعود بطباعة عبارة TRUE أو FALSE حسب حالة العنصر ثم طباعة العنصر نفسه لجميع عناصر القائمة..

```
CL-USER 55 > (test)

TRUE T
FALSE NIL
TRUE 1
TRUE (A B C)
NIL
```

نتيـــجة:

هل اكتشفت وجه الشبه والاختلاف بين الدالتين COND و IF ؟! :)

(if test-form then-form else-form) == (cond (test-form then-form) (t else-form))

سنتعرف في هذا الدرس على الدوال التي تمكن من عمل إجراءات معينة في حال تحقق شروط معينة.. ولا يخفى علينا كمبرمجين أهمية هذه الدوال في لغة LISP شأنها شأن جميع لغات البرمجة الأخرى..
فهي تمكن من عمل تطبيقات صغيرة ولكنها ذكية وذات فائدة كبيرة وتخدم أمثلة وتطبيقات متعددة في مجال الذكاء الاصطناعي..

سنتعرف في هذا الدرس على دالتان شرطيتان هما:

1- الدالة COND
2- الدالة IF

وهي كفيلة بتحقيق جميع مطالبنا في الدروس المتقدمة من سلسلة LISP ولا تقلق فستكون هناك مشاريع برمجية وأمثلة أكثر متعة، ولكن عندما يكون لدينا أساس في كل شيء :)

على بركة الله :)

1- الدالة COND

الوظيفة:

تقوم هذه الدالة بتنفيذ نموذج Form أو أكثر اعتمادا على نتيجة اختبار نموذج Form معين، سنسميه عادة بـ test-form أو للتبسيط Condition. قد نحتاج إلى اختبار نموذج واحد أو أكثر في الوقت نفسه (أي قد يوجد أكثر من test-form واحد) وعندئذ سنحتاج لنربط بينهم بإحدى العلامات المنطقية AND أو OR.

إذا لم يتم كتابة أي نماذج Forms لتنفيذها عند تحقق شرط الدالة COND فان الدالة COND ستعود بالقيمة العائدة من الـtest-form.

وعندما لا يتحقق الـtest-forms فإن الدالة COND سوف تعود بالقيمة False NIL.

والآن بعد هذا التوضيح البسيط للدالة COND سوف نتعرف على شكل هذه الدالة وأمثلة تطبيقية عليها.

الشكل:

(cond ((condition) (form[s])...)

أمثلة:

مثال1:

المثال التالي يحتوي على Condition واحد وForms واحد يتم تنفيذه عند تحقق هذا الشرط:

```
CL-USER 7 > (cond
              ((> x 0) (print "X is greater than 0"))
             )

"X is greater than 0"
"X is greater than 0"
```

ماذا لو كان الشرط غير متحقق؟!

سنعيد إسناد قيمة أقل من الصفر للمتغير X ثم سنجرب تطبيق الدالة COND مرة أخرى

```
CL-USER 8 > (setq x -3)
-3

CL-USER 9 > (cond
             ((> x 0) (print "X is greater than 0"))
            )
NIL
```

كما رأينا فالدالة COND عادت بالقيمة NIL لعدم تحقق الشرط Condition وهو أن تكون قيمة المتغير X أكبر من الصفر.

مثال2:

سنقوم في هذا المثال بوضع أكثر من Condition ونربط بينهم مرة بالعلامة المنطقية AND ومرة بالعلامة المنطقية OR وسيكون هناك Form واحد للطباعة.

آه تذكرت! بخصوص دالة الطباعة لا تقلق فسوف أشرحها أيضا في الدرس القادم إن شاء الله.

```
CL-USER 10 > (setf x 3)
3

CL-USER 11 > (cond
              ((and (> x 0) (< x 5)) (print "X is greater then 0 and less than 5"))
             )

"X is greater then 0 and less than 5"
"X is greater then 0 and less than 5"
```

مثال3:

```
CL-USER 12 > (setf x -5)
-5

CL-USER 13 > (cond
              ((or (> x 0) (> x -10))
               (print "X is positive or greater than the negative integer 10")
              )
             )

"X is positive or greater than the negative integer 10"
"X is positive or greater than the negative integer 10"
```

مثال4:

```
CL-USER 14 > (setf x -11)
-11

CL-USER 15 > (cond
              ((or (> x 0) (> x -10))
               (print "X is positive or greater than the negative integer 10")
              )
             )
NIL
```

مثال5:

سنقوم في هذا المثال بتنفيذ أكثر من form عند تحقق الـtest-form أو الـCondition.

```
CL-USER 31 > (cond
              ((stringp str)
               (progn
                 (print "This is a string")
                 (setq str "C4arab is the best")
                 (print str)
               )
              )
             )

"This is a string"
"C4arab is the best"


CL-USER 32 > str
"C4arab is the best"
```

إذن ماذا تلاحظ في المثال السابق؟

1. الدالة COND تقبل بوجود أكثر من Form بمساعدة دالة أخرى هي الدالة PROGN
2. لكتابة أكثر من Form واحد بحيث يتم تنفيذها جميعا عند تحقق شرط الدالة COND نكتب PROGN ثم نفتح قوسين دائريين ( ) ونكتب بداخلها كل الـForms التي نرغب بتنفيذها مع وضع أقواس دائرية ( ) في بداية ونهاية كل form كما لاحظنا..
3. لا تنس ترتيب برنامجك وجعل الـStructure-Style له جيد أي أن تكتب الدالة على عدة سطور وفي عدة مستويات حتى يكون الكود أكثر وضوحا More Readable and clearable.

مثال6:

سنتعرف في أول درس من دروس القسم المتقدم من سلسلة LISP على كيفية تعريف الدوال من قبل المبرمج.. وسأعرف لكم العديد جدا من الدوال وسنبني العديد من التطبيقات الذكية والصغيرة، لكني أحببت وضع المثال هذا هنا حتى يكون في مكانه:)

في البداية، سنقوم بتعريف دالة بسيطة جدا اسمها select-options وتقوم باستخدام الدالة COND في عملها..

```
CL-USER 18 > (defun select-options ()
               (cond ((= a 1) (setq a 2))
                     ((= a 2) (setq a 3))
                     ((and (= a 3) (floor a 2)))
                     (t (floor a 3))))
Warning: Syntactic warning for form A:
   A assumed special.
SELECT-OPTIONS
```

وبعد تعريف الدالة بفرض أننا أسندنا القيمة واحد للمتغير a:

```
CL-USER 19 > (setq a 1)
1
```

سنقوم باستدعاء الدالة عدة مرات وبناء على طبيعة تحقق شرط الـدالة COND راقب النواتج العائدة من الدالة select-options في كل مرة:

```
CL-USER 20 > (select-options)
2

CL-USER 21 > (select-options)
3

CL-USER 22 > (select-options)
1
```

والآن بفرض أننا وضعنا القيمة 5 للمتغير a ثم استدعينا الدالة مرة أخرى:

```
CL-USER 23 > (setq a 5)
5


CL-USER 24 > (select-options)
1
2
```

وبعد أن تعرفنا على الدالة COND بقي لنا أن نتعرف على دالة أخرى هي الدالة If

2- الدالة IF

الوظيفة:

تسمح هذه الدالة بتنفيذ إجراء معين اعتمادا على تحقق test-form أو Condition وحيد..
ولذا عند وجود أكثر من Condition فإنه لابد أن نربطهم بالعلامة المنطقية AND أو OR حسب حالة وشكل الشرط الذي نريده.

وكما سنلاحظ في شكل الدالة فإن كتابة جزء خاص بالـ else يعد اختياريا يمكن كتابته أو تركه ولذا وضعنا هذا الجزء داخل أقواس مربعة [ ] للدلالة على أن ما بينهما اختياري.

وآلية عمل هذه الدالة والنتائج المحتملة هي كالتالي:

- عند تحقق الـشرط Condition أو الـtest-form فان الجزء then-part يتم تنفيذه..
- أما عند عدم تحقق الشرط فان الجزء else-part يتم تنفيذه اذا كان موجودا.. أما في حال عدم وجوده فان الدالة if ستعود بالقيمة NIL.

الشكل:

(if (condition) (then-part) [(else-part)])

أمثلة:

مثال1:

افترض أننا أسندنا قيمة معينة للمتغير x ولتكن القيمة العددية 3 ثم قمنا باستخدام الدالة if للتحقق من أن قيمة x أكبر من الصفر فإذا ماكان الأمر كذلك فإنه سيتم طباعة العبارة

“X is greater than 0”

```
CL-USER 36 > (setf x 3)
3
```

```
CL-USER 43 > (if (> x 0) (print "X is greater than 0"))

"X is greater than 0"
"X is greater than 0"
```

مثال2:

في هذا المثال نشترط أن يكون X عدد صحيح وأكبر من أو يساوي 5
فاذا ما وضعنا لـX قيمة أقل من 5، أو قيمة أكبر من 5 ولكنها عشرية مثلا 5.5 فإن الدال IF ستعود بالقيمة NIL.

```
CL-USER 44 > (setf x 5)
5

CL-USER 45 > (if
                 (and (integerp x) (>= x 5))
                 (print "X is equal or greater then 5")
             )

"X is equal or greater then 5"
"X is equal or greater then 5"
```

مثال3:

```
CL-USER 46 > (setf x 10)
10

CL-USER 47 > (setf y -10)
-10
```

```
CL-USER 48 > (if
                 (or (> x 0) (> y 0))
                 (progn
                   (print "Either of the numbers is greater than 0")
                   (incf x)
                   (incf y)
                 )
              )

"Either of the numbers is greater than 0"
-9
```

```
CL-USER 49 > x
11

CL-USER 50 > y
-9
```

مثال4:

حالات خاصة:

هذا المثال يبين أن المعرفة مسبقا في اللغة أو التي نقوم بتعريفها تأخذ القيمة T.. ماعدا المتغير NIL المعرف في اللغة فإنه يأخذ القيمة NIL..

لذا لاحظ المثال التالي:

```
CL-USER 51 > (if t 1)
1

CL-USER 52 > (if nil 1 2)
2
```

مثال4:

الدالة test المعرفة في هذا المثال تقوم بعمل حلقة تكرارية داخل عناصر القائمة ((t nil 1 (a b c –لاحظ أن القائمة يمكن أن تحتوي على قوائم أخرى– **فيتم تناول عناصر هذه القائمة واحدا تلو الآخر عن طريق الدالة dolist وفي كل مرة نتناول في أحد هذه العناصر سنسميه بالاسم truth-value..**
وخلال الحلقة dolist سيتم تنفيذ two forms:

الأول: IF

سيتم فيه اختبار truth-value:
فإذا كانت قيمته T فسيتم طباعة True

أما إذا كانت قيمته NIL فسيتم طباعة False

الثاني: PRINT
وسيتم فيه طباعة قيمة truth-value أيا كانت.

```
CL-USER 54 > (defun test ()
               (dolist (truth-value '(t nil 1 (a b c)))
                 (if truth-value (print 'true) (print 'false))
                 (prin1 truth-value)))
TEST
```

وعند استدعاء الدالة test فإنها ستعود بطباعة عبارة TRUE أو FALSE حسب حالة العنصر ثم طباعة العنصر نفسه لجميع عناصر القائمة..

```
CL-USER 55 > (test)

TRUE T
FALSE NIL
TRUE 1
TRUE (A B C)
NIL
```

نتيـــجة:

هل اكتشفت وجه الشبه والاختلاف بين الدالتين COND و IF ؟! :)

(if test-form then-form else-form) == (cond (test-form then-form) (t else-form))

سنتعرف في هذا الدرس على الدوال التي تمكن من عمل إجراءات معينة في حال تحقق شروط معينة.. ولا يخفى علينا كمبرمجين أهمية هذه الدوال في لغة LISP شأنها شأن جميع لغات البرمجة الأخرى..

فهي تمكن من عمل تطبيقات صغيرة ولكنها ذكية وذات فائدة كبيرة وتخدم أمثلة وتطبيقات متعددة في مجال الذكاء الاصطناعي..

سنتعرف في هذا الدرس على دالتان شرطيتان هما:

1- الدالة COND
2- الدالة IF

وهي كفيلة بتحقيق جميع مطالبنا في الدروس المتقدمة من سلسلة LISP ولا تقلق فستكون هناك مشاريع برمجية وأمثلة أكثر متعة، ولكن عندما يكون لدينا أساس في كل شيء :)

على بركة الله :)

1- الدالة COND

الوظيفة:

تقوم هذه الدالة بتنفيذ نموذج Form أو أكثر اعتمادا على نتيجة اختبار نموذج Form معين، سنسميه عادة بـ test-form أو للتبسيط Condition. قد نحتاج إلى اختبار نموذج واحد أو أكثر في الوقت نفسه (أي قد يوجد أكثر من test-form واحد) وعندئذ سنحتاج لنربط بينهم بإحدى العلامات المنطقية AND أو OR.

إذا لم يتم كتابة أي نماذج Forms لتنفيذها عند تحقق شرط الدالة COND فان الدالة COND ستعود بالقيمة العائدة من الـtest-form.

وعندما لا يتحقق الـtest-forms فإن الدالة COND سوف تعود بالقيمة False NIL.

والآن بعد هذا التوضيح البسيط للدالة COND سوف نتعرف على شكل هذه الدالة وأمثلة تطبيقية عليها.

الشكل:

(cond ((condition) (form[s])...)

أمثلة:

مثال1:

المثال التالي يحتوي على Condition واحد وForms واحد يتم تنفيذه عند تحقق هذا الشرط:

```
CL-USER 7 > (cond
             ((> x 0) (print "X is greater than 0"))
            )

"X is greater than 0"
"X is greater than 0"
```

ماذا لو كان الشرط غير متحقق؟!

سنعيد إسناد قيمة أقل من الصفر للمتغير X ثم سنجرب تطبيق الدالة COND مرة أخرى

```
CL-USER 8 > (setq x -3)
-3

CL-USER 9 > (cond
             ((> x 0) (print "X is greater than 0"))
            )
NIL
```

كما رأينا فالدالة COND عادت بالقيمة NIL لعدم تحقق الشرط Condition وهو أن تكون قيمة المتغير X أكبر من الصفر.

مثال2:

سنقوم في هذا المثال بوضع أكثر من Condition ونربط بينهم مرة بالعلامة المنطقية AND ومرة بالعلامة المنطقية OR وسيكون هناك Form واحد للطباعة.

آه تذكرت! بخصوص دالة الطباعة لا تقلق فسوف أشرحها أيضا في الدرس القادم إن شاء الله.

```
CL-USER 10 > (setf x 3)
3

CL-USER 11 > (cond
              ((and (> x 0) (< x 5)) (print "X is greater then 0 and less than 5"))
             )

"X is greater then 0 and less than 5"
"X is greater then 0 and less than 5"
```

مثال3:

```
CL-USER 12 > (setf x -5)
-5

CL-USER 13 > (cond
               ((or (> x 0) (> x -10))
                (print "X is positive or greater than the negative integer 10")
               )
              )

"X is positive or greater than the negative integer 10"
"X is positive or greater than the negative integer 10"
```

مثال4:

```
CL-USER 14 > (setf x -11)
-11

CL-USER 15 > (cond
              ((or (> x 0) (> x -10))
               (print "X is positive or greater than the negative integer 10")
              )
             )
NIL
```

مثال5:

سنقوم في هذا المثال بتنفيذ أكثر من form عند تحقق الـtest-form أو الـCondition.

```
CL-USER 31 > (cond
              ((stringp str)
               (progn
                 (print "This is a string")
                 (setq str "C4arab is the best")
                 (print str)
               )
              )
             )

"This is a string"
"C4arab is the best"


CL-USER 32 > str
"C4arab is the best"
```

إذن ماذا تلاحظ في المثال السابق؟

1. **الدالة COND تقبل بوجود أكثر من Form بمساعدة دالة أخرى هي الدالة PROGN**
2. **لكتابة أكثر من Form واحد بحيث يتم تنفيذها جميعا عند تحقق شرط الدالة COND نكتب PROGN ثم نفتح قوسين دائريين ( ) ونكتب بداخلها كل الـForms**

التي نرغب بتنفيذها مع وضع أقواس دائرية ( ) في بداية ونهاية كل form كما لاحظنا..

3. لا تنس ترتيب برنامجك وجعل الـStructure-Style له جيد أي أن تكتب الدالة على عدة سطور وفي عدة مستويات حتى يكون الكود أكثر وضوحا More Readable and clearable.

مثال6:

سنتعرف في أول درس من دروس القسم المتقدم من سلسلة LISP على كيفية تعريف الدوال من قبل المبرمج.. وسأعرف لكم العديد جدا من الدوال وسنبني العديد من التطبيقات الذكية والصغيرة، لكني أحببت وضع المثال هذا هنا حتى يكون في مكانه:)

في البداية، سنقوم بتعريف دالة بسيطة جدا اسمها select-options وتقوم باستخدام الدالة COND في عملها..

```
CL-USER 18 > (defun select-options ()
               (cond ((= a 1) (setq a 2))
                     ((= a 2) (setq a 3))
                     ((and (= a 3) (floor a 2)))
                     (t (floor a 3))))
Warning: Syntactic warning for form A:
   A assumed special.
SELECT-OPTIONS
```

وبعد تعريف الدالة بفرض أننا أسندنا القيمة واحد للمتغير a:

```
CL-USER 19 > (setq a 1)
1
```

سنقوم باستدعاء الدالة عدة مرات وبناء على طبيعة تحقق شرط الـدالة COND راقب النواتج العائدة من الدالة select-options في كل مرة:

```
CL-USER 20 > (select-options)
2

CL-USER 21 > (select-options)
3

CL-USER 22 > (select-options)
1
```

والآن بفرض أننا وضعنا القيمة 5 للمتغير a ثم استدعينا الدالة مرة أخرى:

```
CL-USER 23 > (setq a 5)
5


CL-USER 24 > (select-options)
1
2
```

وبعد أن تعرفنا على الدالة COND بقي لنا أن نتعرف على دالة أخرى هي الدالة If

2- الدالة IF

الوظيفة:

تسمح هذه الدالة بتنفيذ إجراء معين اعتمادا على تحقق test-form أو Condition وحيد..
ولذا عند وجود أكثر من Condition فإنه لابد أن نربطهم بالعلامة المنطقية AND أو OR حسب حالة وشكل الشرط الذي نريده.

وكما سنلاحظ في شكل الدالة فإن كتابة جزء خاص بالـ else يعد اختياريا يمكن كتابته أو تركه ولذا وضعنا هذا الجزء داخل أقواس مربعة [ ] للدلالة على أن ما بينهما اختياري.

وآلية عمل هذه الدالة والنتائج المحتملة هي كالتالي:

- عند تحقق الـشرط Condition أو الـtest-form فان الجزء then-part يتم تنفيذه..
- أما عند عدم تحقق الشرط فان الجزء else-part يتم تنفيذه اذا كان موجودا.. أما في حال عدم وجوده فان الدالة if ستعود بالقيمة NIL.

الشكل:

(if (condition) (then-part) [(else-part)])

أمثلة:

مثال1:

افترض أننا أسندنا قيمة معينة للمتغير x ولتكن القيمة العددية 3 ثم قمنا باستخدام الدالة if للتحقق من أن قيمة x أكبر من الصفر فإذا ماكان الأمر كذلك فإنه سيتم طباعة العبارة

"X is greater than 0"

```
CL-USER 36 > (setf x 3)
3
```

```
CL-USER 43 > (if (> x 0) (print "X is greater than 0"))

"X is greater than 0"
"X is greater than 0"
```

مثال2:

في هذا المثال نشترط أن يكون X عدد صحيح وأكبر من أو يساوي 5
فاذا ما وضعنا لـX قيمة أقل من 5، أو قيمة أكبر من 5 ولكنها عشرية مثلا 5.5 فإن الدال IF ستعود بالقيمة NIL.

```
CL-USER 44 > (setf x 5)
5

CL-USER 45 > (if
                (and (integerp x) (>= x 5))
                (print "X is equal or greater then 5")
             )

"X is equal or greater then 5"
"X is equal or greater then 5"
```

مثال3:

```
CL-USER 46 > (setf x 10)
10

CL-USER 47 > (setf y -10)
-10

CL-USER 48 > (if
                 (or (> x 0) (> y 0))
                 (progn
                   (print "Either of the numbers is greater than 0")
                   (incf x)
                   (incf y)
                 )
               )

"Either of the numbers is greater than 0"
-9



CL-USER 49 > x
11

CL-USER 50 > y
-9
```

مثال4:

حالات خاصة:

هذا المثال يبين أن المعرفة مسبقا في اللغة أو التي نقوم بتعريفها تأخذ القيمة T.. ماعدا المتغير NIL المعرف في اللغة فإنه يأخذ القيمة NIL..

لذا لاحظ المثال التالي:

```
CL-USER 51 > (if t 1)
1

CL-USER 52 > (if nil 1 2)
2
```

مثال4:

الدالة test المعرفة في هذا المثال تقوم بعمل حلقة تكرارية داخل عناصر القائمة ((t nil 1 (a b c –لاحظ أن القائمة يمكن أن تحتوي على قوائم أخرى– فيتم تناول عناصر هذه القائمة واحدا تلو الآخر عن طريق الدالة dolist وفي كل مرة نتناول في أحد هذه العناصر سنسميه بالاسم truth-value..
وخلال الحلقة dolist سيتم تنفيذ two forms:

الأول: IF

سيتم فيه اختبار truth-value:
فإذا كانت قيمته T فسيتم طباعة True
أما إذا كانت قيمته NIL فسيتم طباعة False

الثاني: PRINT

وسيتم فيه طباعة قيمة truth-value أيا كانت.

```
CL-USER 54 > (defun test ()
               (dolist (truth-value '(t nil 1 (a b c)))
                 (if truth-value (print 'true) (print 'false))
                 (prin1 truth-value)))
TEST
```

وعند استدعاء الدالة test فإنها ستعود بطباعة عبارة TRUE أو FALSE حسب حالة العنصر ثم طباعة العنصر نفسه لجميع عناصر القائمة..

```
CL-USER 55 > (test)

TRUE T
FALSE NIL
TRUE 1
TRUE (A B C)
NIL
```

نتيـــجة:

هل اكتشفت وجه الشبه والاختلاف بين الدالتين COND و IF ؟! :)

(if test-form then-form else-form) == (cond (test-form then-form) (t else-form))

سنتعرف في هذا الدرس على الدوال المتخصصة في طباعة المخرجات النصية..
هناك دوال عديدة وكثيرة جدا لطباعة النصوص في LISP ولكننا سنكتفي بالتعرف على اثنتين منها فقط لكونها تفي بالغرض...

الدالة الأولى: FUNCTION PRINT

وهي دالة سهلة جدا ولكنها غير عملية ولا تفي بكل الاستخدامات.

الدالة الثانية: FUNCTION FORMAT

وهي أصعب نوعا من سابقتها ولكنها عملية جدا ولها استخدامات كثيرة ومتعددة.. حيث تمكننا من طباعة المخرجات في شكل جميل، ونستطيع عن طريقها طباعة المسافات والأسطر الفارغة.. وكذلك يمكننا عن طريقها طباعة النواتج الرقمية من أي نوع وبالدقة التي نريدها للأرقام العشرية..

يمكننا أيضا استخدام الإمكانيات الموجودة في هذه الدالة لطباعة الجداول وغيرها.

1- الدالة PRINT

الوظيفة:

تقوم هذه الدالة بطباعة ما نرسله لها من نصوص أو متغيرات أو القيم العائدة من دالة معينة..

الشكل:

(print form)

أمثلة:

مثال1:
لطباعة النص C4arab نكتب:

```
CL-USER 3 > (print "C4arab")

"C4arab"
"C4arab"
```

لطباعة النص Welcome To C4ARAB.COM المبدوء والمنتهي بمسافات بيضاء، نكتب:

```
CL-USER 4 > (print "       Welcome To C4ARAB.COM       ")

"       Welcome To C4ARAB.COM       "
"       Welcome To C4ARAB.COM       "
```

مثال2:
لطباعة الرقم 1000 نكتب:

```
CL-USER 5 > (print 1000)

1000
1000
```

مثال3:
يمكننا أيضا طباعة القيم العائدة من دوال مختلفة كدوال إجراء العمليات الحسابية أو دوال المقارنة:

```
CL-USER 6 > (print (* 2 3))

6
6

CL-USER 7 > (print (> 4 5))

NIL
NIL

CL-USER 8 > (print (+ (* 3 4) 10))

22
22
```

مثال4:
لطباعة متغير x يحوي قيما رقمية نكتب:

```
CL-USER 9 > (setf x 400.55555)
400.55555


CL-USER 10 > (print x)

400.55555
400.55555
```

ملاحظة: لاحظ أنه تم طباعة محتوى المتغير x لأننا لم نستخدم علامات التنصيص معه، لكن لو استخدمنا علامات التنصيص " " فإنه سيتم اعتباره نصا وليس متغيرا !

```
CL-USER 11 > (print "x")

"x"
"x"
```

مثال5:
لطباعة عناصر قائمة x-list نكتب:

```
CL-USER 12 > (setf x-list '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)


CL-USER 14 > (print x-list)

(1 2 3 4 5 6 7 8 9)
(1 2 3 4 5 6 7 8 9)
```

2- الدالة FORMAT

الوظيفة:

تقوم هذه الدالة بطباعة ما نرسله لها من نصوص أو متغيرات أو القيم العائدة من دالة معينة..

الشكل:

(format {t nil} control-string arguments)

لابد أن يتم كتابة هذه الدالة كما رأينا مع أحد المتغيرين t أو : nil

- **فإذا كتبنا t فإن الدالة Format تعود بـnil**
- **وإذا كتبنا nil فان الدالة Format تعود بـt**

بالنسبة للـ Control-string فهي عبارة التحكم التي تجعل دالة الطباعة Format ذات إمكانيات عالية..
يمكن كتابة أكثر من control-sting بحسب الشكل الذي نريد ظهوره ولأي عدد من الـarguments التي نريد طباعتها..

وهنا قائمة ببعض الـcontrol-strings ووظائفها (والأمثلة توضح لنا كيف نستخدم عبارات التحكم هذه):

~%

لطباعة سطر جديد

~&

لطباعة سطر جديد إذا لم نكن على سطر جديد.

~s

لطباعة s-expression مع علامة التنصيص " "

~a

لطباعة s-expression بدون علامة التنصيص " "

ملاحظة: تعرفنا على الـs-expression في أول درس من دروس القسم المتوسط

~d

لطباعة أعداد صحيحة Integers

~c

لطباعة رمز Character

~{str~}

للدوران داخل عناصر قائمة list وطباعتها عنصرا عنصرا.
~width,decimalsF
لطباعة الأرقام العشرية بطول معين للرقم وعدد محدد للمنازل العشرية.
حيث أن:
width --> عبارة عن الرقم الذي يحدد سعة الرقم الكلية، أي عدد الأرقام قبل العلامة العشرية وبعدها مع أخذ العلامة العشرية أيضا في الاعتبار، وإذا كان إجمالي

العدد أقل من العدد المحدد في عبارة التحكم هذه فإن LISP سيجعلها مساحات بيضاء قبل بداية الرقم.. انظر المثال
decimals --> ويستخدم لتحديد عدد المنازل العشرية.

أمثلة:

مثال1:

طباعة الرموز :Characters

```
CL-USER 15 > (format nil "~C" #\A)
"A"

CL-USER 16 > (format nil "~C" #\Space)
" "
```

```
CL-USER 17 >

;; This next example assumes an implementation-defined "Control" attribute.
 (format nil "~:C" #\Control-Space)
"Control-Space"
```

مثال2:

لطباعة متغير قيمته عبارة عن عدد صحيح نكتب:

```
CL-USER 7 > (setq k 540)
540

CL-USER 8 > (format t "~%     ~d" k)

     540
NIL

CL-USER 9 > (format t "~%   k =   ~d" k)

   k =   540
NIL
```

مثال3:

لطباعة الأرقام العشرية:

```
CL-USER 6 > (format t "~%~10,4f" pi)

    3.1416
NIL
```

مثال4:

لطباعة العبارة "C4arab" مرة بوجود علامات التنصيص ومرة بدونها

```
CL-USER 28 > (format t "~%~a" "c4arab")

c4arab
NIL

CL-USER 29 > (format t "~%~s" "c4arab")

"c4arab"
NIL
```

ماهي لغة الـprolog؟

هي لغة برمجة تعتمد على المنطق وليست على الحسابات وهي ليست لغة من الطراز التقليدي بل مختلفة عن اللغات الأخرى مثل الـc, pascal.
وهي مناسبة جدا لحلول المشاكل التي تشمل على الكائنات (object) والعلاقات(relations) بينهما.
والـ prolog اختصار لكلمةprogramming in logic

-انتشارها ونبذة عن توسعها:

انتشرت بسرعة كبيرة في أوروبا وأصبح لها شعبيتها ،وجاء اليابانيون واعتبروها اللغة الرئيسية لجيل الحاسبات الخامس قرابة العام 1981م.
أما بالنسبة لأمريكا فلم تجد الرواج الكبير لها هناك .
وذكر بعضهم سبب عدم انتشارها
الأول هو صعوبة اتصالها باللغات التقليدية مثل لغة فورترون.(FORTRAN)
والثاني هو بطء برامج البرولوج في طور الإنتاج . وعلى الرغم من تغلب البرولوج السريع (turbo PROLOG) على مثل هذه المشاكل إلا انه حقق ذلك على حساب خصائص أخرى للبرولوج مثل التوحيد.(Unification)

-كيف يمكنني التعامل مع prolog اذا كانت ليست على النمط التقليدي؟

تمكن المبرمج من تمثيل العلاقات بين الأشياء وتجميع وتنظيم هذه العلاقات حتى يمكن الوصول إلى استنتاج منطقي من الحقائق التي تمثلها تلك العلاقات .
وذلك على عكس اللغات التقليدية مثل الباسكال وسي التي تطلب من المبرمج كتابة الخطوات التفصيلية التي يجب إتباعها.
وتمكنك من فعل مالايستطيع العقل تصوره الا بالتطبيق . وذلك يعطيك طريقة جديدة في البرمجة وبشكل أرقى.

تطويرها:
كانت في أوائل 1970م مستخدمة المنطق كلغة وأول من طورها كل من:

- **kowalskiبجامعة أدنبرة من الناحية النظرية.**
- **maartenالبعد التجريبي من أدنبرة.**
- **Alain colmerauerمن جامعة مرسيليا بفرنسا واهتتم بالـ implementation وهو مخترع هذه اللغة.**
- **david d.h warrren.**

وظهرت مؤخرا CLP constrain logic programming عادة ماتعالج كجزء من لغة.system prolog

وفي عام 1996 خرجت نسخة قياسية ISO وتم نشرها وتعتبر النسخة الناشئة من جامعة أدنبرة هي النسخة القياسية.

ملاحظات:

-البرنامج في Prolog هي مايطلق عليه Knowldgebase .

-البرنامج فيه يتكون من حقائق وقواعد..(facts &ruls)

-وربما كأول ثلاث نقاط رئيسية يجب على المتعلم معرفتها:

1. **تعريف العلاقات بواسطة الحقائق.(facts)**
2. **تعريف العلاقات بواسطة القواعد.(rules).**
3. **كيف يجيب الprolog على الأسئلة.**

في هذا الدرس سوف نتعلم كيف ننفذ و نستخدم الـ Amzi Development Environment لتنفيذ برامجنا التي نكتبها

بواسطة لغة Prolog.

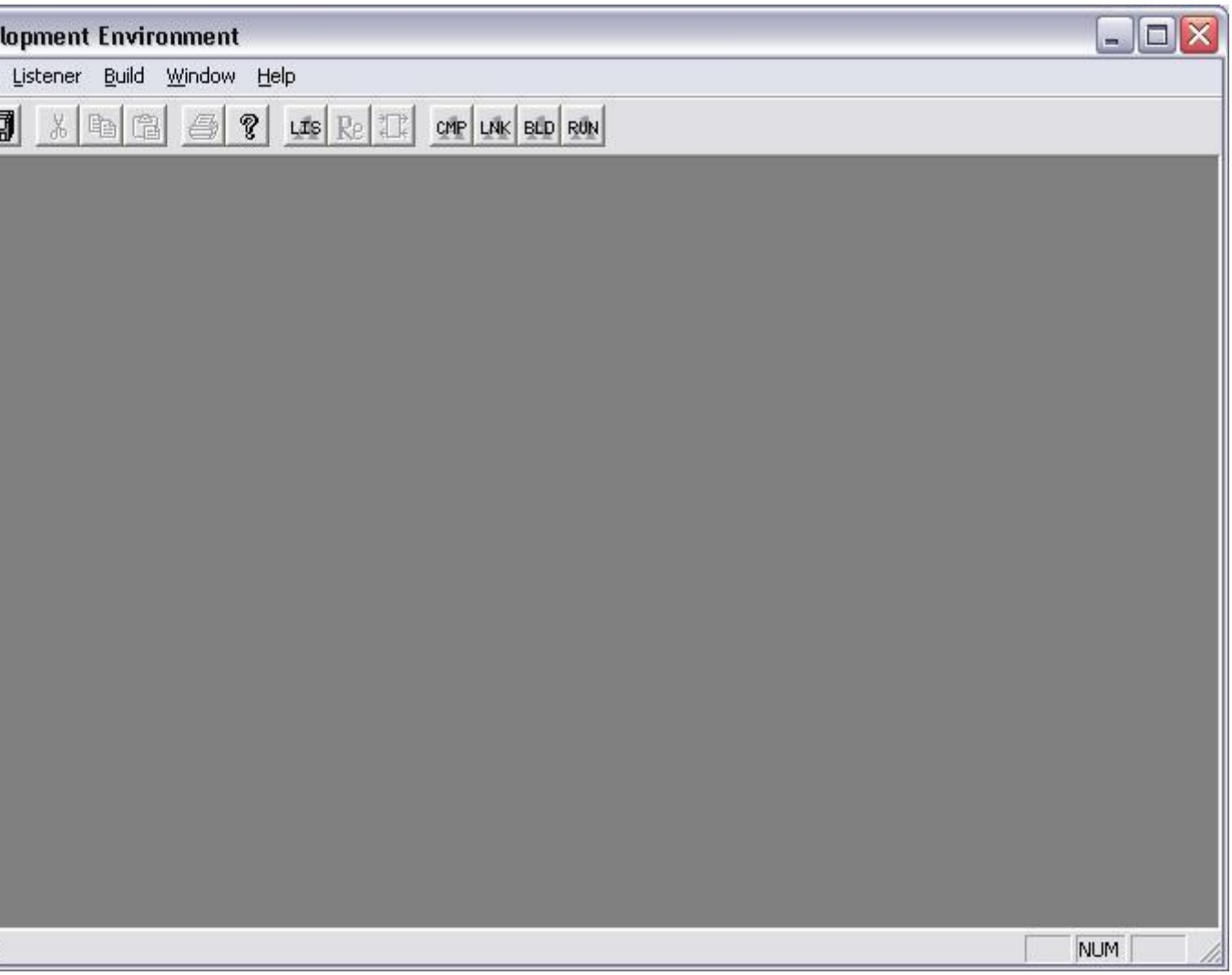
يمكنك تحميل الـ Amzi Development Environment من هنا:

http://www.amzi.com/download/freedist.htm

النسخة الأكاديمية مجانية.

و تحمليه سهل جداً لا يأخذ وقت.

في الأول سوف نشاهد الشكل المبدئي لواجهت الـ Amzi:



الآن سوف نفتح ملف جديد New file من خلال القائمة المنسدلة:

File -> New File

و سوف نكتب هذا الكود البسيط فيه:

```
animal(cat).
animal(dog).
animal(tiger).
```

لا يهم معرفتها الآن لمعنى الكود لأننا سوف نتطرق إليه في درسنا القادم إن شاء الله.

عموماً الكود عبارة عن ثلاثة Facts بسيطة جداً تعرف أن الـ cat, dog and tiger عبارة عن animal.

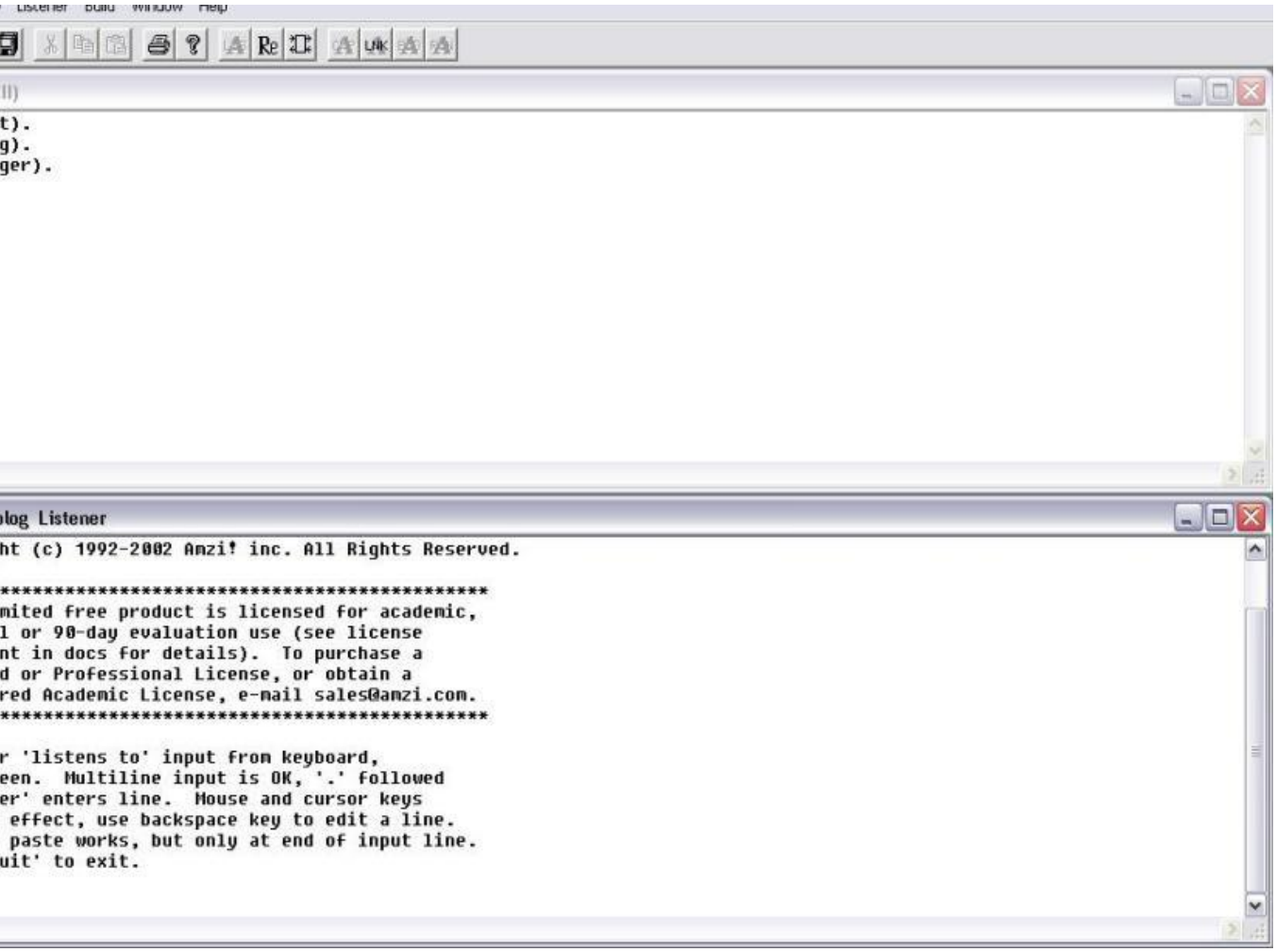
و كل جملة في لغة prolog تنتهي بنقطة.

الآن سوف ننفذ البرنامج من خلال الاختيار من القائمة المنسدلة :

Listener -> Start

كما هو واضح في الصورة:

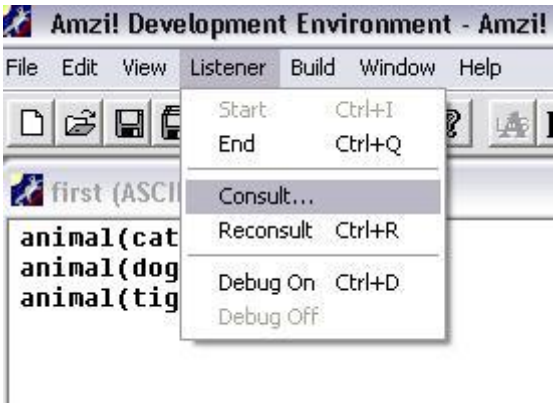
الآن سوف يفتح لنا الـ Amzi نافذة الـ Listener كما هو واضح في الصورة:



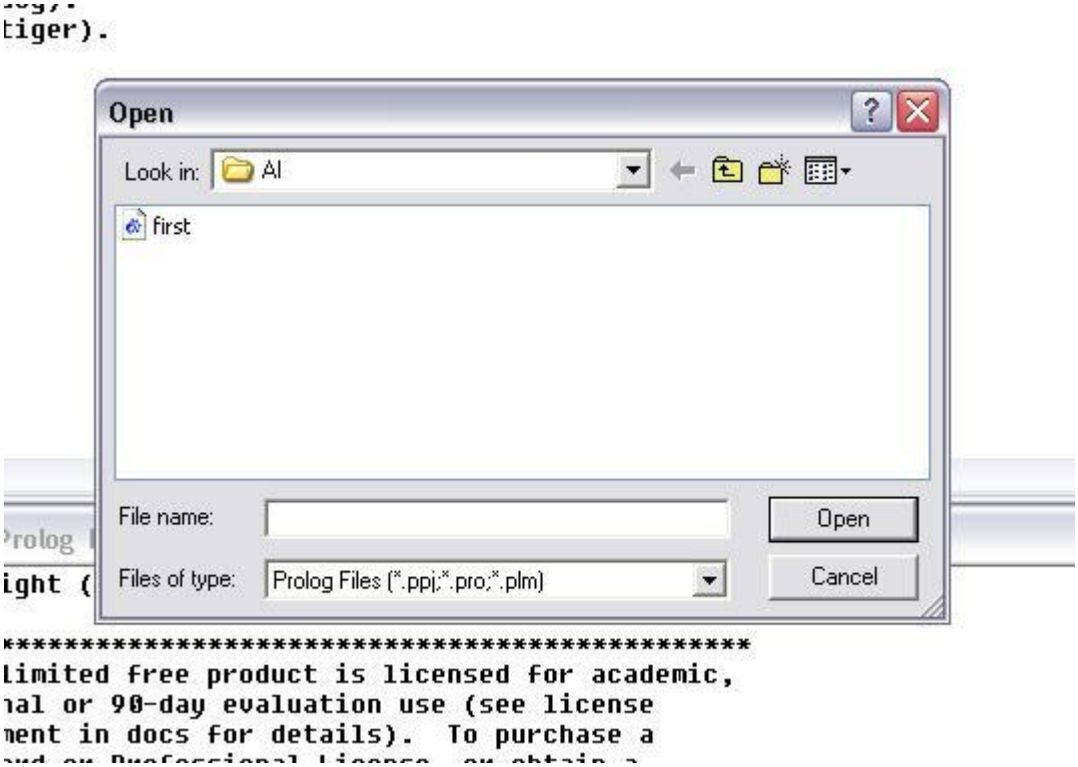
الآن سوف نسند برنامجنا إلى الـ Listener عن طريق الذهاب إلى القائمة المنسدلة :

Listener -> Consult

كما هو واضح في الصورة



الآن سوف نختار الملف المراد تنفيذه من خلال الـ Open Dialog .



الآن من المفترض أن يُكتب في نافذة الـ Listener ما يلي:

```
er' enters line.  Mouse and cursor keys
 effect, use backspace key to edit a line.
 paste works, but only at end of input line.
uit' to exit.

t('C:\\Documents and Settings\\user\\My Documents\\Source Code\\AI\\first.pro')
```

الآن قد هيئنا الـ Amzi لتنفيذ برنامجنا, و لم يبقى علينا الآن إلى الاستعلام و عمل Query للبرنامج.

الـ Query التي سوف نقوم بعملها هي بسيطة للغاية و هي:

animal(X).

ولكن تأكد أن حرف الـ X يكون كبير و ليس x و سوف نعرف الفرق بينهم في درسنا القادم بإذن الله.

الآن من المفترض أن تعطينا هذه الـ Query الـ animal التي كتبناها لها في الـ facts و هي cat, dog, and tiger.

لنكتب الآن

animal(X).

في الـ Listener .

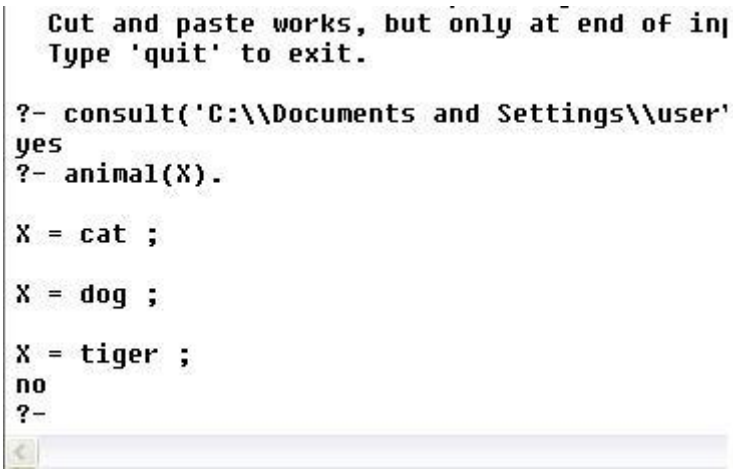
الآن عرض لنا أول fact وهي

X = cat

إذا أردنا باقي الـ facts سوف نضغط على زر الفاصلة المنقوطة ;

أما إذا أردنا الاكتفاء بهذه سوف نضغط المفتاح Enter.

و لنشاهد النتائج في الصورة:



تلاحظون أني استخدمت الفاصلة المنقوطة ; لكي أظهر جميع النتائج.

و كلمة no التي توجد في الأخير هي دلالة على انه لم يبقى هناك facts لكي تظهر.

و لكن الآن ماذا لو أردنا أن نعدل في البرنامج و ننفذه من جديد ؟

هل سنقوم بعمل السابق مرة أخرى ؟

الجواب هو طبعاً لا !!

الخطوات السابقة هي لتهيئة بيئة الـ Amzi فقط و لكن لو عدلنا في الكود و نريد التنفيذ مرة أخرى سوف نقوم بحفظ التعديل

طبعاً و من ثم من القائمة المنسدلة سوف نختار:

Listener -> Reconsult

فقط و الآن نفذ الـ Query التي تريد مرة أخرى.

سوف نبدأ في لغة Prolog كلغة برمجة و على مستوى الكود, و لكن يجب أن أذكر ببعض الأمور من أهمها:

- **اسم لغة Prolog مأخوذ من Programming in Logic , أي البرمجة المنطقية لذلك يجب أن نبتعد بفكرنا عن طريقة التفكير كلغات برمجة عادية مثل سي, سي ,++جافا ... الخ.**

- **لغة Prolog لغة علائقية أي تعتمد على العلاقات مثل علاقات الداتابيز و علاقات Is ... a يعني :**

cat is..a animal.
tiger is..a animal.

- **ومن أهم الفروق بين لغة Prolog و اللغات الأخرى هي أن لغة Prolog هي لغة ديناميكية Dynamic في مستوى كودها, حيث أن الكود ممكن أن يتغير مع تنفيذ البرنامج وذلك ببساطة لأن البرنامج يتعلم و**

يطور من نفسه على عكس اللغات التقليدية مثل السي, سي++, جافا ... هي لغات ثابتة Static على مستوى كودها.

بمعنى آخر لو وجد لدي كود للغة Prolog مكون من 20 سطر مع التنفيذ من الممكن أن يزيد إلى 25 أو أي عدد آخر على حسب
التعلم, أي أن البرنامج يضيف كود من عنده لمواجهة أمور و أحداث لم يبرمج لموجهتها.
بعكس لغات مثل السي عدد سطور كودها لن يتغير مهما كان السبب و لو صادفها شيء لم تبرمج عليه لتوقف البرنامج Crush.

- **البرنامج المكتوب بلغة Prolog هو بسيط للغاية و يتكون في الأساس من Facts و Rules أي حقائق و مسلمات و قواعد.**

- **المتغيرات في لغة Prolog تبدأ إما بحرف كبير مثل X, Talal, Var... أو بعلامة (_) مثل _talal_ ,_Talal , x**

و ما عداها لا يعتبر متغير.

---

1. الحقائق Facts :

في هذا الجزء سوف نشرح الـ Facts في أبسط صورها و هي شبيهه بالسجلات في الداتابيز.
و الشكل العام للـ Facts هو:

```
pred(arg1, arg2, ..., arg n).
```

حيث أن:
pred هو اسم الـ Facts

و , arg1 arg2, arg n هي الحقول التي تأخذها الـ Facts .
و النقطة التي في الأخير هي نهاية أي جملة في لغة Prolog.

و الحقول التي تأخذها الـ Facts هي أما أن تكون متغير من نوع integer أو atom وهو نص إما أن يبدأ بحرف صغير أو يكون بين علامتي ' '
أو يكون من نوع متغير يبدأ بحرف كبير أو علامة (_) أو يكون عبارة عن سجل.

عندما نكتب Fact معينه فإنها سوف تخزن في الـ knowledge base - قاعدة المعارف - و سيكون تعامل البرنامج مع قاعدة المعارف هذه.

الآن لنأخذ برنامج بسيط يصف أن القط عبارة عن حيوان في البداية نستطيع أن نقول :
cat is a animal
و الآن سنحول هذه الجملة إلى Facts في لغة Prolog بالشكل التالي:
```
animal(cat).
```
يعني قلنا لبرنامج Prolog خزن في قاعدة المعارف عندك أن الـ cat عبارة عن animal.
و نستطيع أن نضيف من الحيوانات ما نشاء مثل:
```
animal(dog).
animal(tiger).
animal(bird).
```
الآن كل هذه الـ Facts قد خُزنة في قاعدة المعارف لدينا, و الآن نريد استخراجها !! سيكون ذلك عن طريق عمل Query لقاعدة المعارف لاستخراجها.

الآن سوف ننفذ البرنامج سوية, كود البرنامج سيكون:
```
animal(cat).
animal(dog).
```

```
animal(tiger).
animal(bird).
```

سوف نقوم بعمل

```
Listener -> Start.
Listener -> Consult.
```

ثم نختار الملف المراد تنفيذه

الآن لا يوجد في برنامجنا سوى Fact من نوع واحد و هي animal و الـ Query التي سوف نعملها على قاعدة المعارف لدينا هي
أن نسرد أنواع الـ animal التي توجد في قاعدة معارفنا, و سيكون ذلك أن نكتب اسم الـ Fact التي نريد الاستعلام عنها و نرسل
لها متغير , لترجع لنا قاعدة المعارف أنواع الحيوانات في هذا المتغير.
و سنكتب الاستعلام في الـ Listener بالشكل التالي:

```
animal(X).
```

لاحظ أن حرف X هو حرف كبير أي متغير, الآن سوف يرسل الـ Listener هذا المتغير إلى قاعدة المعارف و سوف ترجع له قاعدة
المعارف أول نوع من الحيوانات - على حسب ترتيب الـ Fact -, في المتغير X و سيكون أول حيوان هو الـ cat و سيظهر بالشكل التالي:

```
X = cat
```

ثم سينتظر الـ Listener لكي أقرر أنا هل أريد المزيد أم لا ؟
إذا اكتفيت بأول ناتج سوف أضغط على مفتاح Enter للاكتفاء, أما إذا أردت المزيد سوف أضغط على (;) لكي تظهر لي
المزيد من النتائج و أستمر على هذا الحال إلى أن لا يبقى لدي نتائج و يكتب الـ Listener كلمة no أي أنه لم يعد هناك
نتائج لكي تظهر.
و ستكون النتائج على فرض أني أضغط على (;) كالتالي:

X = cat ;

X = dog ;

X = tiger ;

X = bird ;
no

هذا هو النوع الأول من سرد المعارف التي توجد لدي و لكن من الممكن أيضاً أن أسئل قاعدة المعارف أسئلة الـ yes .. no
حيث أسألها إن كان يوجد لديك كذا مثلاً, وهي ترد بنعم أو لا.
الآن سوف نسألها هي الـ cat من نوع animal لديها أم لا ؟
و سيكون ذلك بالشكل التالي:

animal(cat).

و من ثم سيرد علينا الـ Listener بنعم أو لا , على حسب وجود هذه الـ Fact عنده أم لا.
و لكن من الواضح أن هذه الـ Fact موجودة لديه لذلك سيرد بنعم كما سيظهر لكم في الـ Listener

?- animal(cat).

yes
?-

و علامة -? تدل على أن الـ Listener ينتظرك لتدخل Query .
الآن سوف نستعلم عن شيء غير موجود في قاعدة المعارف مثلاً سوف نسألها هل الكتاب عبارة عن حيوان أم لا ؟
طبعاً نحن لم ندخل الكتاب على أساس أنه حيوان في قاعدة معارفنا, وسيكون شكل الاستعلام كالتالي:

animal(book).

و كما هو واضح لديكم أن الرد هو بالشكل التالي:

```
?- animal(book).
no
?-
```

و سيرد الـ Listener بـ no إذا استعلمت عن Fact غير موجودة أصلاً مثلاً:

```
?- person(mammal).
no
?-
```

في هذا الجزء سوف نأخذ مثال عملي من واقع الحياة اليومية و نحوله إلى برنامج مكتوب بلغة Prolog.
لو كان لدينا عائلة فيها:
محمد أب لـ عبدالله و صالح , و عبدالله أب لـ خالد و صالح أب لـ أحمد و سلطان.
هذه عبارة عن حقائق نستطيع بالتأكيد تمثيلها بواسطة لغة Prolog, لنأخذ الجملة الأولى:
"محمد أب لـ عبدالله و صالح" .
نستطيع أن نحولها إلى :
"محمد أب لعبدالله" و "محمد أب لصالح".
عندئذ سوف نكتبها في Prolog كالتالي:

```
father(mohammad, abdullah).
father(mohammad, saleh).
```

أي أن محمد أب لعبدالله في الأولى و محمد أب لصالح في الثانية.
الآن لنأخذ الجملة الثانية:
"و عبدالله أب لـ خالد".
سوف نحولها في لغة Prolog بنفس الشكل السابق و لكن الأب هنا ليس محمد بل عبدالله.

```
father(abdullah, khalid).
```

أي أن عبدالله أب لخالد.

و لنأخذ الجملة الثالثة الآن:

"و صالح أب لـ أحمد و سلطان".
سنحول هذه الجملة إلى أبسط صورة لها وهي:
"صالح أب لأحمد" و "صالح أب لسلطان".
و سنحولها إلى لغة Prolog بنفس الطريقة السابقة:

```
father(saleh, ahmad).
father(saleh, sultan).
```

الآن هذه المعلومات الـ Facts أصبحت في قاعدة المعارف لدينا و لم يبقى لدينا إلا عمل بعض الـ Query عليها.

الآن لو أردنا أن أن نعرف والد خالد ؟
نحن جعلنا في الـ Father أو تعارفنا أن الحقل الأول هو الأب و الثاني هو ابنه, لذلك لو أردنا أن نعرف من هو أب خالد سوف
نستعلم عن father و نجعل الحقل الأول متغير ليعطينا الآباء و الحقل الثاني هو خالد لكي ينقي و ينقح لنا الآباء و يعطينا أب خالد بالذات.
و ستكون الـ Query كالتالي:

```
father(X, khalid).
```

و كما نلاحظ جميعاً أن أب خالد سوف يوضع في المتغير X
و يطبع في الـ Listener كالتالي:

```
?- father(X, khalid).

X = abdullah

yes
?-
```

و لكن ماذا لو أردنا أن نعرف أولاد محمد ؟
سيكون نفس المثال السابق و لكن بالعكس أي سنضع في الحقل الأول محمد و في الثاني متغير X أو أياً يكن ليسرد لنا أولاد محمد
و ستكون الـ Query كالتالي:

```
father(mohammad, X).
```

و كما نلاحظ أن النتائج ستكون كالشكل التالي:

```
?- father(mohammad, X).

X = abdullah ;

X = saleh ;
no
?-
```

طبعاً إذا أردنا سرد جميع الآباء و الأبناء سنكتب هذا الاستعلام:

```
father(Father, Child).
```

لاحظ أننا أرسلنا في الحقلين متغيرات Father و Child لأن أول حرف من كلاً منهم هو حرف كبير

الأنظمة الخبيرة (Expert Systems (ES

مقدمة:

الإنسان الخبير وحده من يستطيع تقديم أداء رفيع المستوى في نطاق تخصصه! الأنظمة الخبيرة تستخدم معلومات محصورة في نطاق محدد كي تقدم نفس أداء الإنسان الخبير في نفس النطاق!
تذكر أنه ليست جميع النطاقات تعتبر مجالات خصبة لبناء أنظمة خبيرة فيها، كما أنه لا يمكننا صياغة جميع الخبرات والمهارات التي لا تكون موجودة سوى لدى الإنسان الخبير؛ فبعضها لا يمكن صياغته!

من هنا نستنتج أن المعادلة الأولى في عالم الأنظمة الخبيرة هي:

النظام الخبير = الإنسان الخبير!

حسناً، وماهو النظام الخبير؟!

هو برنامج له سلوك الإنسان الخبير في نطاق محدد من التطبيقات، أو هو برنامج يحل المشاكل التي غالباً ما تحل عن طريق إنسان خبير، هذه المشاكل غالباً ما يطلق عليها اسم Expert-level Problems.

الفرق بين النظام الخبير والإنسان الخبير:

مع أن المعادلة الأولى في عالم الأنظمة الخبيرة تقتضي أن النظام الخبير = الإنسان الخبير، إلا أنه توجد فروق دقيقة عديدة بينهما، سنستنتج جميع هذه الفروق سوياً في الدروس القادمة إن شاء الله.
وبصورة مبدئية وبدائية جداً يكون الفرق بينهما:

1. **في المعرفة:**
   تتكون المعرفة التي توجد لدى الإنسان الخبير من معرفة نظرية Theoretical نابعة عن فهم المشكلة بالإضافة إلى معرفة تطبيقية Practical نتجت من تجاربه وأظهرت فاعلية كبيرة في حل المشاكل وأعطته خبرات و معلومات جديدة وعملية، على عكس النظام الخبير الذي لا يمكنه التعلم من تجاربه بإضافة خبرات جديدة إلى قاعدة المعرفة خاصته، إنما يحتاج إلى تدخل من قبل الإنسان كي يضيف هذه الخبرات إليه !
2. **عملياً:**
   يملك الإنسان الخبير مهارات حسية (في الجراحة أو أسلوب التوضيح أثناء الحديث) ويملك حدساً يمكنه من التعامل مع معلومات غير دقيقة، وغير مكتملة أو حتى ذات نسبة صحّة صغيرة نسبياً. بينما يفتقر النظام الخبير إلى المهارات الحسية، وإلى الحدس في حل المشكلة، كما أنه لا يمكن أن يتعامل سوى مع معلومات ذات نسبة صحّة كبيرة.

إذن نستطيع القول أن النظام الخبير هو نسخة مركبة مما يتركب منه عقل الإنسان الخبير الذي مرت عليه الكثير من التجارب ولديه العديد من الخبرات، فهي برامج عملية تستخدم استراتيجيات استكشاف طورت من قبل الإنسان لحل فئة معينة من المشاكل، وبسبب استراتيجيات الاستكشاف هذه؛ فإن طبيعة مركز المعرفة (قاعدة البيانات) في أي نظام خبير معني بحل مشكلة محددة لابد من أن تكون:

1. **تدعم عمليات التعليل، سواء كان تعليل وقتي لكل خطوة يقوم بها النظام، أو تعليل للقرار النهائي الذي يقدمه النظام .**
2. **تسمح بعمليات التعديل بسهولة، سواء كانت لإضافة بعض المهارات إلى قاعدة البيانات أو إلى حذف بعضها منها.**
3. **تعلّل عن طريق الاستكشاف، متشبهين هنا بطريقة التعليل في عقل الإنسان!**

ستتفهم هذه النقاط بشكل أعمق إن شاء الله مع الدروس القادمة.

الخصائص الواجب توفرها في النظام الخبير:

1. **لابد من أن يكون قادراً على شرح قراره وتعليله كما يفعل الإنسان الخبير، وذلك بهدف:**
   -تعزيز ثقة المستخدم بالنظام) .مثال على ذلك الطبيب، فإن كان قادراً على توضيح سبب تحديد الجرعة التي حددها لمريضه زادت ثقة المريض بالطبيب).
   -السماح للمستخدم بإيجاد نقاط الضعف التي من الممكن أن تكون موجودة في النظام عندما يقوم بشرح قراره حينها يمكننا إلقاء القبض على أي خطأ من الممكن أن يوجد في قاعدة المعرفة، مما يساعد بقوة في إصلاحه وتشذيب قاعدة معرفة النظام.
2. **لابد من أن يكون قادر على التعامل مع معلومات غير كاملة أو غامضة، كما يفعل الإنسان الخبير (فالطبيب الخبير أو الاستشاري يستطيع التشخيص على معطيات غير مكتملة) ولكن هذه النقطة بحاجة لوقفة تفصيلية سنتعرف عليها في الدروس القادمة إن شاء الله.**
3. **لابد من أن تحوي واجهة مستخدم ظريفة والتي بدورها تجعل تعليل النظام واضح للمستخدم وغير غامض.**

أسباب تدعونا لصنع أنظمة خبيرة:

1. **صيانة المعرفة من الاندثار أو الانقراض، نقصد بذلك المعرفة القيمة الفريدة التي لا تكون موجودة إلا عند إنسان خبير مميز في تخصصه.**
2. **لحل المشاكل، مما يحفظ الوقت والمال والجهد، وهذا ما يجعلنا نحتاج إلى قاعدة بيانات ضخمة بل هائلة.**
3. **زيادة الخبراء في المجال الذي يُصنع النظام الخبير فيه.**

محتويات الدرس:

- **المجالات الأكثر مناسبة لبناء أنظمة خبيرة.**
- **أنواع) تصنيف) المشاكل التي تحتاج إلى أنظمة خبيرة.**
- **المواصفات التي لابد من توافرها في المشاكل كي نحكم ونقول بأن هذه المشاكل تحتاج إلى أنظمة خبيرة لبناءها.**

- أهم أنظمة خبيرة معروفة في العالم اليوم-Well Known ES.

المجالات الأكثر مناسبة لبناء أنظمة خبيرة:

وهي المجالات التي طبِّقت فيها أنظمة خبيرة فعلياً:

- الطب Medicine
- الرياضيات Mathematics
- الهندسة Engineering
- الكيمياء Chemistry
- الجيولوجيا Geology
- علوم الكمبيوتر Computer Science
- التجارة Business
- القانون Low
- الدفاع Defense
- التعليمEducation

نرى أن هذه المجالات في مجملها مجالات بها تحدي كبير من نوع أو آخر!

أنواع) تصنيف) المشاكل التي تحتاج إلى أنظمة خبيرة:

1. التأويل والتفسير:Interpretation
   بمعنى المشاكل التي تحتاج لحلها: تشكيل نتائج أو توصيفات رفيعة المستوى من مجموعة من بيانات معطاة، مثل الجرائم.
2. التنبؤ:Prediction
   بمعنى المشاكل التي تحتاج لحلها: تصوّر عواقب محتملة نتيجة عوامل معطاة، مثال على ذلك: التنبؤ بأحوال الطقس.
3. التشخيص:Diagnosis
   بمعنى المشاكل التي تحتاج لحلها: تحدد سبب القصور ومواقع الضعف في الحالات المعقدة بناءاً على الأعراض الملاحظة.
4. التصميم:Design
   بمعنى المشاكل التي تحتاج لحلها: إيجاد تشكيل

مناسب لمكونات نظام يخدم أهداف متقدمة مع وجود العديد من القيود.

5. التخطيط:Planning
بمعنى المشاكل التي تحتاج لحلها: تدبير سلسلة من الأحداث المتعاقبة التي تحقق مجموعة من الأهداف بمعلومية شروط ابتدائية معينة وقيود تشغيل زمنية. مثال على ذلك: الذراع الآلية .
6. المراقبة:Monitoring
بمعنى المشاكل التي تحتاج لحلها: مقارنة السلوك المشاهد للنظام مع السلوك المتوقع له !
7. محاولة إكتشاف الأخطاء وإصلاحها Debugging and Repair:
بمعنى المشاكل التي تحتاج لحلها: توصيف وتطبيق علاج للقصور الموجود في نطاق معين.
8. التوجيه:Instruction
بمعنى المشاكل التي تحتاج لحلها: اكتشاف ومن ثم تصحيح نقاط الضعف لفهم موضوع معين .
9. التحكم:Control
بمعنى المشاكل التي تحتاج لحلها: السيطرة على سير العمل في بيئة معقدة.

المواصفات التي لابد من توافرها في المشاكل كي نحكم ونقول بأن هذه المشاكل تحتاج إلى أنظمة خبيرة لبناءها:

1. أن تكون الحاجة لحلول هذه المشاكل مبررة لما يستلزمه بناء نظام خبير من التكلفة والجهد.
2. عندما لا يتوفر الإنسان الخبير في كل الحالات التي نحتاج إليه فيها لحل المشكلة.
3. عندما تكون المشكلة ممكنة الحل بطرق الاستدلال الرمزي symbolic reasoning دون الحاجة إلى مهارات حسية perceptual skills.
4. عندما يكون نطاق المشكلة معرَّف well structured ولا يحتاج إلى حدس commonsense reasoning في حل بعض المشاكل التي تظهر فيه.
5. عندما لا يمكن حل المشكلة باستخدام طرق الحساب التقليدية.

6. عندما يتواجد خبراء في نطاق المشكلة مستعدين للتعاون باسلوب واضح وفصيح.
7. عندما يكون حجم ومجال المشكلة معقول ومناسب، يستحق الوقت والجهد.

أهم أنظمة خبيرة معروفة في العالم اليوم Well-Known ES:

توضح الصورة التالية أهم أنظمة خبيرة معروفة في العالم اليوم، تاريخ إنتاج كل منها، والمجال الذي تتعامل معه -مع ملاحظة أننا كنّا سنتطرق لمناقشة بعض هذه الأنظمة بالتفصيل في الدورة، ولكن يمكنك عمل بحث عن اسم أي نظام خبير في محرك بحث شهير ليقوم بإعطاءك صفحات لا نهائية تتحدث عنه بالتفصيل :-

Year	ES Name	Source	ES Job
1965	DENDRAL	Stanford	analyze mass spectrometry data
1965	MACSYMA	MIT	symbolic mathematics problems
1972	MYCIN	Stanford	diagnosis of blood diseases
1972	Prospector	SRI	Mineral Exploration
1975	Cadeceus	U. of PITT	Internal Medicine
1978	DIGITALIS	MIT	digitalis therapy advise
1979	PUFF	Stanford	obstructive airway diseases
1980	R1	CMU	computer configuration
1982	XCON	DEC	computer configuration
1983	KNOBS	MITRE	mission planning
1983	ACE	AT&T	diagnosis faults in telephone cables
1984	FAITH	JPL	spacecraft diagnosis
1986	ACES	Aerospace	satellite anomaly diagnosis
1986		Cambpell	diagnose cooker malfunctions
1986	DELTA/CATS	GE	diagnosis of diesel locomotives
1987		AMEX	credit authorization
1992	MAX	NYNEX	telephone network troubleshooting
1995		Caltech	PacBell network management
1997		UCI	planning drug treatment for HIV