المحاضرة الأولى

تطور لغات البرمجة و أساليبها

1-1 مقدمة عن لغات البرمجة: Programming Languages Introduction

نعلم أن دراسة علوم الحاسب تنقسم إلى قسمين هما:

- العتاد Hardware.
- البرمجيات Software.

ولتسهيل الدراسة يتم تجزئة كل قسم على حده. فتم تقسيم العتاد إلى وحدات الإدخال، وحدات الإخراج، و وحدة النظام (المكونة من وحدة المعالجة المركزية و وحدة الذاكرة). وتم تقسيم البرمجيات إلى نظم التشغيل، **لغات البرمجة**، والبرامج التطبيقية.

لغات البرمجة:

تنقسم لغات البرمجة بصفة عامة إلى مستويين أساسين هما:

- لغات المستوى المنخفض Low-Level Languages.
- لغات المستوى العالي High-Level Languages.

و بالطبع هناك فارق كبير بين هذين المستويين في الإمكانيات، وسهولة التعامل مع الحاسب، بالإضافة إلى سهولة تعلم اللغة وفهمها. وبما أن لغات المستوى العالي تستخدم كلمات إنجليزية معينة ورموز رياضية مألوفة، فهي أسهل في تعلمها وفهمها.

1-1-1 لغات المستوى المنخفض Low-Level Languages

تنقسم لغات هذا المستوى إلى قسمين هما:

- لغة الآلة Machine Language.
- لغة التجميع Assembly Language.

- **لغة الآلة Machine Language**

هي أول اللغات ظهور، وهي اللغة الوحيدة التي يفهمها الحاسب مباشرة دون وسيط. وتتكون من رمزين هما: الصفر و الواحد.هذان الرمزان يعبران عن الأوامر المختلفة والبيانات التي يتكون منها البرنامج. إلا أن هذه اللغة صعبة التعلم وخاصة أن لكل حاسب لغة آلة خاصة به، وتتطلب معرفة واسعة في تصميم الحاسب، بالإضافة إلى صعوبة في اكتشاف الأخطاء. مما أدى إلى تطوير هذه اللغة إلى لغة التجميع.

- **لغة التجميع Assembly Language**

تعتمد هذه اللغة على الرموز المختزلة mnemonic code، أي اختصارات لكلمات ذات مدلول لغوي محدد مثل: ADD تدل على الجمع، و MOV تدل على النقل، وهكذا...، مما جعل تعلم هذه اللغة أسهل نسبياً من لغة الآلة، بالإضافة إلى سهولة الكشف عن الأخطاء وتصحيحها. ولكن البرنامج في هذه اللغة يتم تجميعه وتحويله إلى لغة الآلة عن طريق ما يسمى بالمجمع Assembler. مما يؤكد أن الحاسب لا يتعامل مباشرة إلا مع لغة الآلة. ومع ذلك تبقى هذه اللغة صعبة التعلم، ولها عيوب من أبرزها ارتباطها بالآلة ، فكل آلة لها لغة تجميع خاصة بها، ويقصد بالآلة هنا تحديداً المعالج Processor أو المعالج الصغير Microprocessor.

بناءً على ما سبق نقول إن كتابة البرامج بلغات المستوى المنخفض تعتمد على معرفة واسعة بالتصميم الداخلي للحاسب (المعالجات، المقاطعات، مسارات البيانات، عناوين الذاكرة،...). مما جعل العلماء يفكرون بلغات تعزل المبرمج نسبياً عن التصميم الداخلي للحاسب.

1-1-2 لغات المستوى العالي High-Level Languages

تعتمد هذه اللغات على كلمات إنجليزية واضحة المدلول مثل:write, read, input print، وتم عزل المبرمج عن مشقة الخوض في متاهات التصميم الداخلي للحاسب، مما سهل تعلم هذه اللغات والإقبال عليها لحل المشاكل والتطبيقات العلمية والتجارية وغيرها. إلا أن تنفيذ البرنامج بهذه اللغات يحتاج إلى كشف الأخطاء وتتبع التعليمات خطوة خطوة وذلك عن طريق ما يسمى بالمفسر Interpreter ثم ترجمته وتحويله إلى لغة الآلة عن طريق ما يسمى بالمترجم Compiler.

والآن، يمكن إيجاز مميزات لغات المستوى العالي فيما يلي:

- عدم الارتباط بمعالجات معينة مثل لغات التجميع، وذلك لأن هذه اللغات مصممة أساساً لحل نوعية محددة من المشاكل وليست لنوعية محددة من المعالجات.
- سهولة تعلمها وسهولة كتابة البرامج فيها، وذلك لاستخدامها كلمات وتعبيرات مشابهة لما يستخدمه الإنسان.
- سهولة اكتشاف الأخطاء وتصحيحها.
- توفير الجهد والوقت الذي كان يقوم به المبرمجون عند كتابة البرامج بلغة الآلة أو لغة التجميع.

أما اللغات التي ظهرت في هذا المستوى فهي كثيرة جداً، من أبرزها وأشهرها: الفورتران *fortran* ، الكوبول *cobol* ، البيسك *basic* ، الباسكال *pascal* ، السي *c* ،...الخ.

ويمكن أن نطلق على هذه اللغات بأنها لغات خطية بسبب كونها تعتمد على التعليمات والأوامر المتسلسلة والمرتبة والتي تتوافق مع الخوارزمية. وقد تفاضلت هذه اللغات فيما بينها من حيث القوة والسهولة فكانت لغة البيسك Basic هي اللغة الأكثر شهرة وشعبية، وعامّة الاستخدام لجميع المبتدئين في البرمجة. وقد اشتقت حروفها من الحروف الأولى للعبارة الآتية:

Beginners All-Purpose Symbolic Instruction Code

وتعني: لغة التعليمات الرمزية المتعددة الأغراض للمبتدئين. ظهرت هذه اللغة في الستينات في كلية جامعية في الولايات المتحدة الأمريكية، ثم طورت من قبل معهد المقاييس الأمريكية ANSI عام 1968م، ومن أهم الإصدارات كانت **QBasic**.

بقيت هذه اللغات البرمجية بكافة أنواعها ضعيفة من حيث واجهات البرامج التي تنشئها، والواجهات المقبولة تتطلب كتابة آلاف الأسطر أثناء تصميم البرنامج، مما دفع الشركات لتطوير هذه اللغات إلى لغات مرئية، وخصوصاً بعد ظهور نظام النوافذ windows الذي يدعم البيئة الرسومية (Graphic User Interface)، في هذه الآونة ظهرت اللغات المرئية مثل: فيجول بيسك، الدلفي (فيجول باسكال)، فيجول سي++ ،...إلخ.

Visual Basic, Delphi, Visual C++,…etc.

وقد تبنت شركة مايكروسوفت لغة البرمجة **Qbasic** لتكون نواة للغة بيسك المرئية (فيجول بيسك) Visual Basic ، وقد ظهر أول إصدار لها عام 1991م وما يزال التحديث جارياً على هذه اللغة حتى الآن. وأصبحت لغة Visual Basic في إصداراتها الحديثة مصممة للكائنات (**Objects**). وفي هذا المقرر سيتم شرح كيفية التعامل مع هذه اللغة في الإصدار السادس **Visual Basic .Net 2005**.

تبين لنا مما سبق أن لغات البرمجة عالية المستوى تنقسم إلى قسمين هما:

- **لغات خطية:** تعتمد على كتابة الأوامر والتعليمات على شكل خطوات مرتبة ومنتهية، وهذه الخطوات تمثل ترجمة للخوارزمية، مثل: البيسك، الفورتران، الباسكال، وغيرها.
- **لغات مرئية:** تعتمد على الكائنات أو الأدوات والأحداث، حيث يتم الترابط بين الكائنات بعمليات برمجية، ثم يتم تنفيذ المشروع عن طريق الأحداث، ولذلك يسمون لغة بيسك المرئية باللغة الموجهة بالأحداث:

Event-driven Programming

أو اللغة المسيرة بالأحداث، أي اللغة التي تنفذ تعليماتها وإجراءاتها عند اختيار الأحداث، والحدث هو كل تأثير يتم على الفأرة أو لوحة المفاتيح، مثل: **Click** أو **Double click** أو غيرها من الأحداث.

تطور أساليب البرمجة Evolution of Programming Styles

لقد مرت أساليب البرمجة بثلاث مراحل:

- البرمجة العشوائية Spaghetti Programming
- البرمجية الهيكلية Structured Programming
- البرمجة الشيئية Object Oriented Programming

1. البرمجة العشوائية أو المكر ونية Spaghetti Programming

يركز هذا الأسلوب من البرمجة على حل المسألة برمجياً وتحقيق الهدف دون النظر إلى عملية تنظيم البرنامج، وفي هذه الحالة يعاني البرنامج من صعوبة في التطوير، ومن صعوبة في اكتشاف الأخطاء، وربما حصل تكرار في بعض المقاطع البرمجية، كما أن استخدام الأمر Goto بكثرة يعيق فهم البرنامج وصعوبة تتبع خطوات التنفيذ لذلك جاء تعبير البرمجة المكر ونية، وفي هذه الحالة ينظر للبرنامج كأنه كتلة واحدة. وهذا الأسلوب لا تظهر مشاكله إلا إذا كان البرنامج كبيراً وضخماً، أما في حالة البرامج التدريبية البسيطة ربما يكون مناسباً، لأنه لا يحتاج إلى تجزئة. سنقدم مثالاً يناسب هذا الأسلوب من البرمجة وآخر لا يناسب هذا الأسلوب.

2. البرمجة الهيكلية: Structured Programming

يعتمد هذا الأسلوب من البرمجة على تجزئة البرنامج إلى عدة برامج فرعية، بحيث يتم الربط بين هذه البرامج الفرعية لتشكل البرنامج العام. والمقصود بالبرامج الفرعية function أو procedure . يحتاج هذا الأسلوب إلى تخطيط جيد، وتظهر فاعليته في حالة المسائل المتوسطة الحجم والتي تطرح على الطلاب في المرحلة الجامعية، كما هذا الأسلوب يسهل من اكتشاف الأخطاء، وإجراء عمليات التطوير، وعدم تكرار المقاطع

3. البرمجة غرضية التوجه: Object Oriented Programming

تدعى أيضاً بالبرمجة الشيئية أو الكائنية المنحى وهي البرمجة التي تحاكي الواقع.

يعتمد هذا الأسلوب من البرمجة على بناء الكائنات التي تضم البيانات والإجراءات ، وجملة ترابط الكائنات تشكل المشروع.

ما هي البرمجة غرضية التوجه ؟

ظهرت البرمجة غرضية التوجه OOP في بداية السبعينات من هذا القرن، حيث بدت الحاجة ماسة لها، بعد أن واجهت البرمجة الهيكلية(الإجرائية) عدة مشاكل منها: البيانات غير المحمية، عدم القدرة على محاكاة الواقع، صعوبة تقسيم البرنامج إلى إجراءات، عدم القدرة على إعادة الاستعمال، وغيرها من الأسباب.

قد يتصور البعض أن البرمجة غرضية التوجه إنما هي لغة برمجية جديدة، ولكن الأمر ليس كذلك، إنما هي أسلوب تنظيمي في البرمجة بكافة توجهاتها وعلى اختلاف لغاتها، وأن استخدام أسلوب البرمجة غرضية التوجه لا يعني نسف كل ما تعلمناه كمبرمجين، لا بل هو الإلمام بالإمكانات المتاحة من خلال هذا الأسلوب.

إن مصممي البرمجة غرضية التوجه اعتبروا أن الأجسام من حولنا ما هي إلا كائنات تتفاعل مع بعضها وفق علاقات تتفق مع طبيعتها، بغض النظر عن كون هذه الكائنات حية أم جامدة. والهدف من ذلك هو مساعدة المبرمجين على كتابة برامج قابلة للتطوير وبمدة زمنية أقل.

لقد مر معنا ذكر الكائن (Object)، فما هو الكائن وفق ما تراه البرمجة غرضية التوجه.

الكائن (Object)

الكائن هو نمط جديد (متغير مركب يشبه نسبياً السجل في تعريفه البرمجي) ينتج من دمج البيانات Data والإجراءات Procedure التي تعمل على تلك البيانات في كينونة واحدة.

الآن، عندما تحاول حل مشكلة في البرمجة غرضية التوجه، لن تتساءل عن كيفية تقسيم المشكلة إلى إجراءات أو دوال، بل عن كيفية تقسيمها إلى كائنات. إن التفكير بالكائنات بدلاً عن الإجراءات له تأثير مفاجئ عن مدى سهولة تصميم البرنامج بهذا الأسلوب، وذلك بسبب المطابقة القوية بين الكائنات في المفهوم البرمجي والكائنات في الحياة الفعلية.