الوحدة الأولى: الأسس والمفاهيم الجوهرية

المحاضرة الأولى: مدخل عام للبرمجة

تتناول هذه المحاضرة المفاهيم الأساسية التي تشكل حجر الزاوية في مجال البرمجة، والتي تُعد الأساس اللازم للانتقال إلى دراسة الذكاء الاصطناعي.

1. تعريف البرمجة(Programming Definition)

تُعد البرمجة (Programming) حجر الزاوية في العصر الرقمي، وهي في جوهرها عملية إبداعية وهندسية لإنشاء حلول للمشكلات باستخدام الحاسوب، ولا يقتصر التعريف على مجرد "كتابة تعليمات"، بل يشمل مراحل التفكير النقدي، وتصميم الخوارزميات، واختيار هياكل البيانات المناسبة، وتبدأ العملية بتحليل المشكلة أو المهمة المراد إنجازها بدقة متناهية، ثم يتم تصميم خطة عمل (Plan of Action) أو خوارزمية (Algorithm) تمثل سلسلة منطقية من الخطوات اللازمة للحل، وهذه الخوارزمية تُترجم لاحقًا إلى شفرة مصدرية (Source Code) باستخدام إحدى لغات البرمجة (Programming Languages) مثل Python أو Java أو C++ .

والهدف الأساسي هو تحويل الأفكار والمنطق البشري إلى شكل يمكن للآلة تنفيذه بكفاءة وفعالية، مما يمكن الحاسوب من أداء مهام تتراوح من العمليات الحسابية البسيطة إلى تشغيل أنظمة معقدة وإدارة كميات هائلة من البيانات.

يُمكن النظر إلى البرمجة على أنها فن التواصل مع الحاسوب، لكن هذا التواصل يتميز بالدقة المتناهية وعدم وجود أي مجال للغموض، فالتعليمات المكتوبة يجب أن تكون دقيقة ومُتسلسلة(Precise and Sequential) ، تمامًا كوصفة طعام صارمة؛ لا يُمكن للحاسوب تخمين الخطوة التالية أو تصحيح الأوامر الناقصة، ويتم هذا التواصل عبر لغات متخصصة تُعرف بلغات البرمجة، التي تعمل كجسر بين المنطق البشري وهندسة الآلة، وتختلف هذه اللغات في تركيبها ومجالات استخدامها، فبعضها مخصص لتطوير تطبيقات الويب (مثل(JavaScript ، وبعضها لتطبيقات الهاتف المحمول، والبعض الآخر للتحليل العلمي والذكاء الاصطناعي (مثلPython)، وبغض النظر عن اللغة المُستخدمة، فإن المبرمج يمارس عملية تجريد(Abstraction) ، حيث يأخذ مشكلة واقعية معقدة ويُبسطها إلى نموذج يمكن التعبير عنه في شكل متغيرات، ودوال، وهياكل تحكم (مثل الحلقات والشروط) ضمن الشفرة.

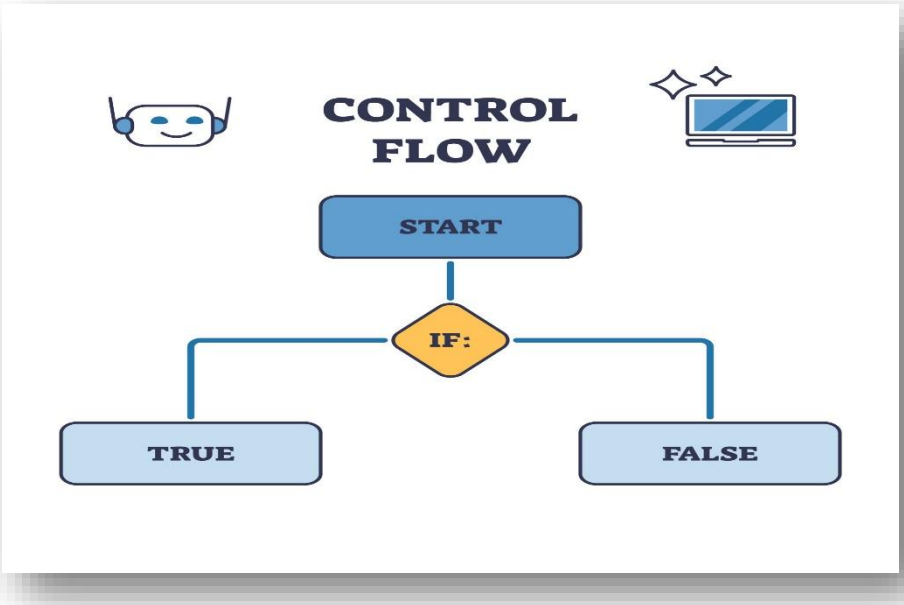
إن الهدف الجوهري من عملية البرمجة هو إنجاز مهمة معينة أو حل مشكلة محددة، وهذا يعني أن كل سطر من الشفرة المكتوبة يخدم غرضًا وظيفيًا واضحًا، سواء كان ذلك بجمع رقمين، أو ترتيب قائمة من الأسماء أبجديًا، أو إدارة حركة مرور البيانات على شبكة عملاقة، وعملية كتابة التعليمات لا تتوقف عند مرحلة الترجمة والتنفيذ؛ بل تشمل أيضًا الاختبار(Testing) ، والتصحيح(Debugging) ، والصيانة .(Maintenance) فالبرنامج يجب أن يُثبت قدرته على العمل بشكل صحيح في جميع الظروف المتوقعة، والتعامل مع الأخطاء غير المتوقعة (مثل إدخال المستخدم لبيانات خاطئة) بأمان، ولذا، يتطلب دور المبرمج مزيجًا من المهارات التحليلية لحل المشكلات، والمهارات المنطقية لتصميم التسلسل الصحيح للخطوات، والمهارات التقنية لإتقان بناء الجملة (Syntax) للغة المختارة.

لتوضيح كيف تُترجم المفاهيم المجردة للبرمجة إلى خطوات عمل ملموسة، يمكننا النظر إلى أمثلة من الحياة اليومية التي تعتمد على هذه العملية، ويوضح الجدول أدناه كيف يتم تكسير مهمة بسيطة ومعقدة إلى خطوات برمجية أساسية، وكيف ترتبط لغة البرمجة المختارة بالناتج النهائي، والإلمام بهذه الأمثلة يُبين أن البرمجة هي أساس التشغيل الآلي الذي نعتمد عليه في حياتنا اليومية، من الآلة الحاسبة التي نستخدمها وصولًا إلى خوارزميات البحث المعقدة:

المفهوم البرمجي	الوصف التوضيحي للتعليمات	مثال تطبيقي بسيط (الآلة الحاسبة)	مثال تطبيقي مُعقد (تطبيق الخرائط)
تعريف المتغيرات والإدخال	تحديد مكان تخزين البيانات التي سيعمل عليها البرنامج (مثل الأرقام المدخلة).	إدخال رقمين (الرقم الأولA ، الرقم الثانيB).	تحديد موقع المستخدم الحالي (المتغير $_1Lat1, _1Lon_1$) وموقع الوجهة (المتغير $_2Lat_2, _2Lon_2$).
المعالجة/الخوارزمية	تطبيق العمليات المنطقية والحسابية على البيانات المُدخلة.	تنفيذ عملية الجمع: Sum = A + B	تشغيل خوارزمية البحث عن أقصر مسار (مثل خوارزميةDijkstra) بناءً على بيانات الطرق وتوقعات الازدحام.
بنية التحكم (الشرط)	استخدام عبارات مثل "إذا كان... فافعل...".	**إذا** كان الإدخال غير رقمي، **فأظهر** رسالة خطأ بدلاً من إجراء العملية الحسابية.	**إذا** كان الطريق مغلقًا أو مزدحمًا جدًا، **فاختر** طريقًا بديلاً وقم بتحديث الوقت المُقدر للوصول(ETA) .
الإخراج	تقديم النتيجة النهائية للمستخدم بصيغة مفهومة.	عرض قيمة المتغير Sumعلى الشاشة.	عرض المسار المُقترح كنقاط على الخريطة، مع عرض الوقت المقدر للوصول (ETA) والمسافة.

2. خصوصيات البرمجة(Programming Characteristics)

تميز البرمجة بكونها نشاطًا فكريًا ومنهجيًا يرتكز على مجموعة من الخصائص الفريدة التي تضمن أن تكون النتائج التي تُنتجها الحواسيب قابلة للتنبؤ وموثوقة. أبرز هذه الخصائص هي الدقة والمنطق(Precision and Logic) ، ويجب أن تكون كل تعليمة برمجية واضحة ومحددة بشكل قاطع، فالحاسوب جهاز منطقي بحت يقوم بالتنفيذ الحرفي (Literal Execution)للتعليمات المُعطاة له؛ أي أنه يفتقر إلى القدرة على التخمين، أو الاستنتاج، أو فهم النوايا البشرية الغامضة، وهذا يتطلب من المبرمجين التفكير بمنهجية رسمية، حيث يتم تفكيك المشكلة إلى أصغر وحداتها المنطقيةk على سبيل المثال، لتحديد ما إذا كان الرقم موجبًا، يجب أن تكون التعليمة المنطقية هي A > 0؛ لا يوجد مجال لتعريفات مرنة أو تقديرية، مما يضمن أن تتطابق المخرجات تمامًا مع المدخلات والمنطق المُطبق عليها.



الشكل 01: مخطط تدفق التحكم (Control Flow)

المخطط يجسد مبدأ **الاختيار(Selection)** ، وهو إحدى البنى الأساسية في البرمجة، يخبر هذا التدفق الحاسوب بما يلي: "ابدأ، ثم قيِّم هذا الشرط، **إذا** كان الشرط صحيحًا، اتبع المسار الأيمن؛ **وإلا** (ELSE)، اتبع المسار الأيسر." هذا يسمح للبرامج باتخاذ قرارات وتنفيذ إجراءات مختلفة بناءً على البيانات المُدخلة أو حالة النظام.

الخاصية الثانية والأكثر أهمية لضمان عمل البرنامج بشكل صحيح هي التسلسل(Sequence) ، وتُنفذ الأوامر البرمجية عادةً خطوة بخطوة بترتيب زمني أو منطقي دقيق ومحدد سلفًا، وهذا ما يُعرف باسم تدفق التحكم(Control Flow)، ففي معظم لغات البرمجة، يُنفذ السطر الذي يليه، ما لم يتم إدخال بنى تحكم صريحة مثل الحلقات التكرارية (Loops)أو العبارات الشرطية(Conditionals) ، إن أي خلل في ترتيب هذه التعليمات قد يؤدي إلى نتائج غير صحيحة أو، في أسوأ الأحوال، إلى انهيار (Crash) البرنامج. على سبيل المثال، إذا كان البرنامج يهدف إلى حساب ضريبة المبيعات، فيجب أن تتم عملية حساب الضريبة بعد إدخال قيمة السلعة وقبل إظهار السعر النهائي، ويمثل المبرمج تدفق التحكم هذا غالبًا باستخدام مخطط انسياب (Flowchart) لضمان المنطقية قبل كتابة الشفرة.

تتميز البرمجة الناجحة بخاصية العمومية(Generality) ، والتي تعكس جودة تصميم البرنامج، ولا يهدف البرنامج الجيد إلى حل مشكلة واحدة فقط باستخدام بيانات مُحددة مسبقًا (مثل جمع رقمي 5 و 3)، بل يجب أن يكون قابلاً لإعادة الاستخدام (Reusable) للتعامل مع أي مجموعة من البيانات ضمن نفس نطاق المشكلة (مثل جمع أي رقمين يختارهما المستخدم)، وهذا يتحقق من خلال كتابة تعليمات تعتمد على المتغيرات (Variables) بدلاً من القيم الثابتة، واستخدام الدوال (Functions) أو الإجراءات (Procedures) التي يمكن استدعاؤها مرارًا وتكرارًا، وتزيد العمومية من قيمة البرنامج وتقيه من التقادم، حيث يُمكنه التكيف مع مجموعات بيانات مختلفة أو سيناريوهات جديدة دون الحاجة لإعادة كتابة الشفرة بالكامل، مما يختصر وقت التطوير والصيانة بشكل كبير.

الخاصية الرابعة والحاسمة، خاصة في الأنظمة الكبيرة والمعقدة، هي الفاعلية(Efficiency) ، وتركز الفاعلية على الأداء الأمثل للبرنامج من حيث استهلاك موارد الحاسوب، وهذا يشمل التعقيد الزمني(Time Complexity) ، أي سرعة إنجاز المهمة، والتعقيد المكاني(Space Complexity) ، أي كمية الذاكرة المطلوبة لتشغيل البرنامج، ويتمثل التحدي هنا في كتابة خوارزميات (Algorithms) تستخدم أقل قدر ممكن من وقت المعالجة والذاكرة. فمثلًا، قد يكون هناك عدة طرق لفرز قائمة من الأرقام، لكن خوارزمية الفرز الفعالة ستنجز المهمة بشكل أسرع بكثير عند التعامل مع ملايين الإدخالات مقارنةً بالخوارزمية الأقل كفاءة، ويعمل المبرمج على موازنة الوضوح والبساطة في الشفرة (للتسهيل على الصيانة) مقابل السرعة القصوى واستهلاك الذاكرة الأقل (لتحقيق الفاعلية).

أمثلة توضيحية لخصائص البرمجة

الخاصية	مثال على تطبيق الخوارزمية	سبب أهمية الخاصية	مثال على خطأ ناتج عن إهمال الخاصية
الدقة والمنطق	استخدام جملة شرطية **IF** :(الرصيد> 0) **THEN** السماح بالتحويل.	تمنع سوء تفسير الآلة للتعليمات الغامضة وتضمن تنفيذاً صحيحاً.	محاولة قسمة عدد على صفر (10/0) دون شرط منطقي لتجنب الانهيار.
التسلسل	في عملية تسجيل الدخول: 1. إدخال كلمة المرور، 2. تشفيرها، 3. مقارنتها بالبيانات المخزنة.	يضمن تنفيذ الخطوات بترتيبها الصحيح والمنطقي للوصول إلى النتيجة المرجوة.	محاولة المقارنة (الخطوة 3) قبل إكمال عملية التشفير (الخطوة 2)، مما يؤدي إلى فشل التحقق.
العمومية	كتابة دالة تقوم بجمع أي رقمين يتم تمريرهما إليها كمدخلات، بدلاً من أرقام ثابتة.	تجعل الشفرة قابلة لإعادة الاستخدام للعديد من السيناريوهات والمدخلات المختلفة (Scalability).	إنشاء برنامج جديد في كل مرة تتغير فيها مجموعة البيانات المراد جمعها أو معالجتها.

الفاعلية	استخدام خوارزمية بحث متقدمة (مثل البحث الثنائي) للعثور على عنصر في قائمة مرتبة.	تقلل من استهلاك موارد الحاسوب (الوقت والذاكرة)، خاصة عند التعامل مع بيانات ضخمة.	استخدام خوارزمية بحث خطي بسيطة (تفحص عنصراً عنصراً) في قائمة تحتوي على مليون عنصر، مما يسبب بطء غير مقبول.

3. لغة البرمجة(Programming Language)

لغة البرمجة هي **وسيط التواصل** بين المبرمج والحاسوب. وهي مجموعة من القواعد (Syntax) والكلمات المفتاحية (Keywords)التي تترجم الأفكار البشرية إلى تعليمات يمكن للمعالج (CPU) فهمها وتنفيذها.

أنواع لغات البرمجة:

مثل لغات البرمجة الوسيلة التي يتفاعل بها المبرمج مع الآلة، وتطورت هذه اللغات عبر الزمن في محاولة لسد الفجوة بين المنطق البشري وقدرة الحاسوب على التنفيذ، وتنقسم بشكل أساسي إلى فئتين رئيسيتين: اللغات منخفضة المستوى واللغات عالية المستوى، وتُعد اللغات **منخفضة المستوى (Low-Level Languages)** هي الأقرب إلى البنية المادية للحاسوب، حيث تتعامل مباشرة مع المعالج والذاكرة، وتنتمي لهذه الفئة **لغة الآلة (Machine Language)** التي تُكتب باستخدام الشيفرة الثنائية (الصفر والواحد) وهي اللغة الوحيدة التي يفهمها الحاسوب مباشرة دون الحاجة لأي ترجمة، ويترتب على هذا القرب أنها تتمتع بكونها أسرع اللغات تنفيذاً وأكثرها كفاءة في استخدام الموارد، إلا أنها في المقابل تُعتبر صعبة الكتابة والقراءة والفهم بالنسبة للبشر، مما يجعلها قليلة الاستخدام اليوم إلا في المهام الحرجة التي تتطلب تحكماً دقيقاً في العتاد، مثل تطوير أنظمة التشغيل أو برامج تشغيل الأجهزة.(Drivers)

الفئة الأخرى هي **لغات التجميع (Assembly Language)** والتي تُعتبر أيضاً ضمن اللغات منخفضة المستوى، على الرغم من أنها تمثل خطوة أولى نحو التسهيل على المبرمجين، فهي لا تزال قريبة جداً من فهم العتاد، لكن بدلاً من استخدام الأصفار والآحاد، تستخدم لغة التجميع رموزاً مختصرة (Mnemonics) مفهومة نسبياً مثل ADD للجمع و MOV للنقل، مما يجعلها أسهل قليلاً في الكتابة مقارنة بلغة الآلة، لكنها لا تزال تتطلب فهماً عميقاً لهندسة المعالج وهياكل الذاكرة الخاصة به، وتتطلب هذه اللغات عملية ترجمة بسيطة (باستخدام المجمّع Assembler) - لتحويلها إلى لغة الآلة القابلة للتنفيذ، وتُستخدم لغة التجميع عادةً في تحسين أداء أجزاء محددة وحرجة من الشفرة أو في البرمجة المدمجة (Embedded Programming) حيث تكون موارد الذاكرة والمعالجة محدودة للغاية.

على النقيض تماماً، ظهرت **اللغات عالية المستوى (High-Level Languages)** بهدف جعل عملية البرمجة أسهل وأكثر كفاءة للمبرمجين، وتُعد هذه اللغات أقرب بكثير إلى اللغة البشرية الطبيعية (غالباً الإنجليزية)، حيث تستخدم قواعد بناء (Syntax) وكلمات مفتاحية مألوفة، مثل if وwhile وprint، مما يقلل بشكل كبير من الأخطاء ويسرّع عملية تطوير البرامج، وتشمل هذه الفئة معظم اللغات الحديثة والشائعة مثل Python، Java، JavaScript، C#، وRuby، وتتطلب هذه اللغات عملية ترجمة معقدة إما عن طريق المُترجِم (Compiler) الذي يحوّل الشفرة دفعة واحدة إلى لغة الآلة، أو عن طريق المُفسِّر (Interpreter) الذي يقوم بتنفيذ الشفرة سطراً سطراً.

إن الميزة الرئيسية للغات عالية المستوى لا تكمن فقط في سهولة كتابتها وقراءتها، بل أيضاً في خاصية الاستقلالية عن المنصة(Platform Independence) ، فغالباً ما تسمح هذه اللغات بتشغيل البرنامج المكتوب على أنواع مختلفة من أنظمة التشغيل والعتاد (مثل Windows أو macOS أوLinux) دون الحاجة لإعادة كتابة الشفرة، وهذا يُقارن باللغات منخفضة المستوى التي تكون مقيدة جداً بنوع المعالج الذي كُتبت من أجله، لذلك تُعد اللغات عالية المستوى هي الخيار السائد والأساسي لتطوير جميع أنواع التطبيقات الحديثة، بما في ذلك تطبيقات الويب، وتطبيقات الهاتف

المحمول، والذكاء الاصطناعي، وتحليل البيانات، نظراً لما توفره من بيئات عمل متكاملة ومكتبات ضخمة جاهزة للاستخدام، مما يسمح للمبرمج بالتركيز على حل المشكلة (Problem Solving) بدلاً من إدارة تفاصيل العتاد المعقدة.

نوع اللغة	أمثلة شائعة	القرب من...	سهولة الاستخدام للمبرمج	سرعة التنفيذ	مجال الاستخدام النموذجي
منخفضة المستوى	لغة الآلة، لغة التجميع	العتاد والحاسوب	صعبة جداً وتتطلب معرفة معمقة	عالية جداً (الأسرع)	تطوير أنظمة التشغيل، برامج تشغيل الأجهزة، الأنظمة المدمجة
عالية المستوى	Python, Java, C++, JavaScript	اللغة البشرية (الإنجليزية)	سهلة وواضحة جداً	معتدلة إلى سريعة (أبطأ من الآلة)	تطوير الويب، تطبيقات الهاتف، الذكاء الاصطناعي، تطوير الألعاب

4. نظام الخوارزميات(Algorithms System) :

الخوارزمية هي قلب أي برنامج .قبل كتابة أي سطر من الشفرة البرمجية، يجب أولاً تصميم الخوارزمية.

تعريف الخوارزمية:

تُمثّل الخوارزمية (Algorithm) المخطط الأولي والمنطقي الذي يسبق عملية كتابة الشفرة، فهي ليست مجرد قائمة من التعليمات، بل هي مجموعة منظمة ومنهجية من الخطوات يتم تصميمها بعناية فائقة لضمان الوصول إلى حل صحيح للمشكلة المراد معالجتها، وتتميز الخوارزمية بكونها محددة (Defined) بشكل دقيق، مما يعني أن كل خطوة فيها يجب أن تكون واضحة وغير قابلة للتأويل أو الغموض، فالحاسوب لا يستطيع فهم التعليمات المبهمة أو العامة، ويجب أن تكون هذه الخطوات أيضاً مرتبة ومتسلسلة(Sequential) ، حيث يؤدي تنفيذ كل خطوة إلى الانتقال إلى الخطوة التي تليها في سياق منطقي محدد، تماماً كخطوات وصفة طعام لابد من الالتزام بترتيبها لضمان نجاح الطبق، وتُعد الخوارزمية بمثابة "الخطة التنفيذية "التي تُترجم لاحقاً إلى إحدى لغات البرمجة.

من أهم الخصائص الجوهرية التي يجب أن تتوافر في أي خوارزمية صالحة هي الانتهاء(Finiteness) ، وهذا يعني أن الخوارزمية يجب أن تتضمن نقطة توقف واضحة ومضمونة بعد عدد محدد من الخطوات، فلا يُمكن اعتبار أي سلسلة تعليمات تستمر إلى ما لا نهاية (حلقة غير منتهية) كخوارزمية صحيحة، فإذا لم تضمن الخوارزمية التوقف، فلن يكون لها قيمة عملية في حل المشكلات، بالإضافة إلى ذلك، يجب أن تكون الخوارزمية فاعلة(Effective) ، بمعنى أن تكون كل خطوة قابلة للتنفيذ عملياً وخلال فترة زمنية معقولة باستخدام الموارد المتاحة للحاسوب، وتُستخدم الخوارزميات في كل مكان في عالم الحوسبة، بدءاً من عمليات البحث البسيطة على الإنترنت، ومروراً بفرز البيانات، ووصولاً إلى خوارزميات الذكاء الاصطناعي المعقدة، مما يؤكد أنها العمود الفقري لأي نظام حوسبي ناجح.

خصائص الخوارزميات:

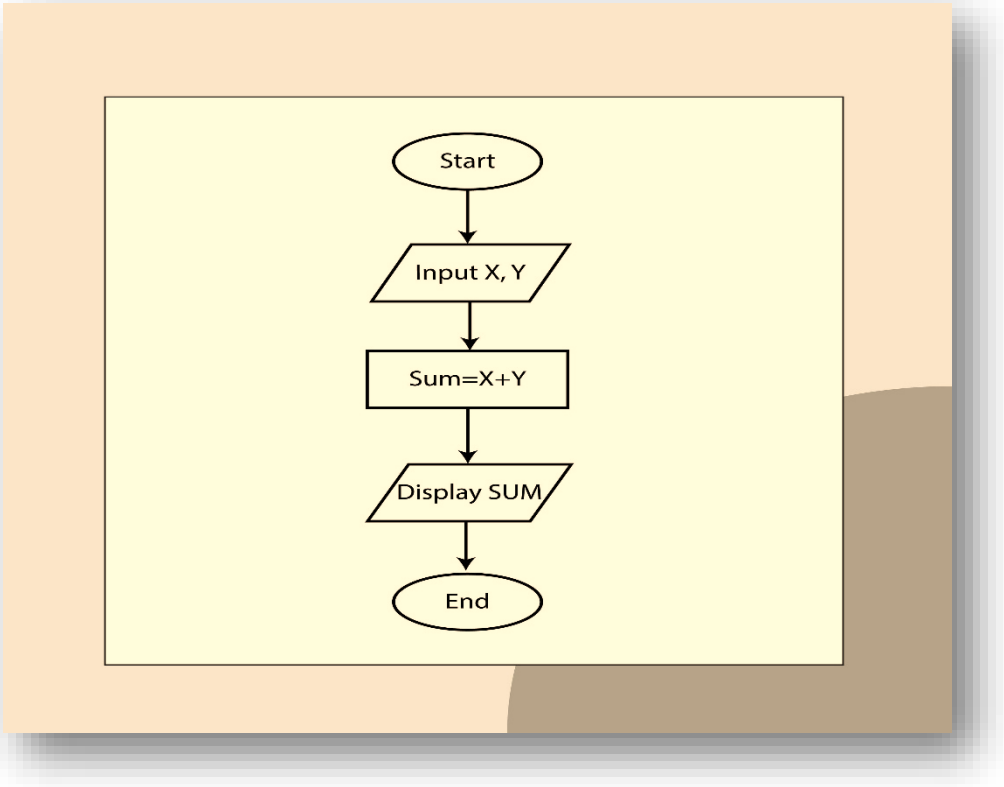
- **المدخلات(Input) :** يجب أن تقبل الخوارزمية صفر أو أكثر من المدخلات المحددة جيداً.
- **المخرجات(Output) :** يجب أن تنتج الخوارزمية نتيجة واحدة على الأقل (حل المشكلة).
- **الفعالية(Finiteness) :** يجب أن تتوقف الخوارزمية بعد عدد محدد من الخطوات.
- **الوضوح(Definiteness) :** يجب أن تكون كل خطوة من خطوات الخوارزمية واضحة ولا تحتمل التأويل.
- **الصحة(Correctness) :** يجب أن تنتج الخوارزمية الحل الصحيح للمشكلة المراد حلها.

تمثيل الخوارزميات:

يتم تمثيل الخوارزميات عادةً بإحدى طريقتين رئيسيتين قبل تحويلها إلى كود برمجي:

- **الشفرة الوهمية(Pseudocode) :** استخدام لغة طبيعية تشبه الإنجليزية ولكن بهيكل يشبه لغات البرمجة.

- **المخطط الانسيابي(Flowchart) :** تمثيل رسومي يستخدم أشكالاً هندسية وخطوط لبيان تسلسل العمليات واتجاهها.



الشكل 02: مخطط الانسياب (Flowchart)

- **Start**: Like any good algorithm, the program begins at a clear starting point. A programmer would use an oval shape to indicate the start and end of a program. This is important because it helps to create a clear and logical flow of instructions.
- **Input:** The next step is to input the data that the program will use. This flowchart is very simple, so it only requires two numbers. The parallelogram shape is used to indicate any input or output of data.
- **Process**: Next, the program needs to process the data that was just inputted. The flowchart uses a rectangular shape to indicate any type of process, such as addition, subtraction, multiplication, or division. In this case, it performs a very simple addition of the two inputs.
- **Output**: After the program has processed the data, it needs to output the results. In this case, the program will display the sum of the two numbers that were inputted. The same parallelogram shape is used to indicate the output of the data.
- **End**: Finally, the program needs to end. The same oval shape is used to indicate the end of the program. This is important because it helps to create a clear and logical flow of instructions.

مكونات مخطط الانسياب(Flowchart)

- **البداية(Start) :** مثل أي خوارزمية جيدة، يبدأ البرنامج من نقطة انطلاق واضحة. يستخدم المبرمجون **الشكل البيضاوي (Oval)** للإشارة إلى بداية ونهاية البرنامج، وهذا التحديد مهم لأنه يساعد في إنشاء تسلسل واضح ومنطقي للتعليمات.

- **الإدخال(Input)** : تتمثل الخطوة التالية في إدخال البيانات التي سيستخدمها البرنامج، وهذا المخطط الانسيابي بسيط جدًا، لذلك لا يتطلب سوى رقمين. يُستخدم **شكل متوازي الأضلاع (Parallelogram)** للإشارة إلى أي عملية إدخال أو إخراج للبيانات.
- **المعالجة(Process)** : بعد ذلك، يحتاج البرنامج إلى معالجة البيانات التي تم إدخالها للتو، ويستخدم المخطط الانسيابي **الشكل المستطيل (Rectangular)** للإشارة إلى أي نوع من عمليات المعالجة، مثل الجمع أو الطرح أو الضرب أو القسمة، وفي هذه الحالة، يقوم بإجراء عملية جمع بسيطة جدًا للمُدخلين.
- **الإخراج(Output)** : بعد أن يعالج البرنامج البيانات، يحتاج إلى إظهار النتائج. في هذه الحالة، سيعرض البرنامج مجموع الرقمين اللذين تم إدخالهما، ويُستخدم **شكل متوازي الأضلاع (Parallelogram)** نفسه للإشارة إلى إخراج البيانات.
- **النهاية(End)** : أخيرًا، يجب أن ينتهي البرنامج، ويُستخدم **الشكل البيضاوي (Oval)** نفسه للإشارة إلى نهاية البرنامج، وهذا ضروري للمساعدة في الحفاظ على تسلسل واضح ومنطقي للتعليمات.

تمارين وتطبيقات عملية في مفاهيم البرمجة والخوارزميات

تطبيق مفهوم الخوارزمية وتدفق التحكم(Control Flow)

الهدف من هذه التمارين هو تحويل المشكلات اليومية إلى خطوات منطقية **محددة ومنتهية (Defined and Finite Steps)** كما تتطلب الخوارزمية.

التمرين 1: تصميم خوارزمية بسيطة (حساب متوسط ثلاثة أرقام)

المشكلة: صمم خوارزمية لحساب المتوسط الحسابي لثلاثة أرقام صحيحة يتم إدخالها من المستخدم.

مرحلة الخوارزمية	الشفرة الوهمية (Pseudocode) المقابلة	الشرح والتطبيق على الخصائص
البداية	START	**الوضوح(Definiteness)** : تحديد نقطة انطلاق واضحة للعملية.
المدخلات	READ Num1, Num2, Num3	**المدخلات(Input)** : استقبال ثلاثة مدخلات محددة من المستخدم.
المعالجة (التسلسل)	Sum = Num1 + Num2 + Num3 Average = Sum / 3	**التسلسل(Sequence)** : لا يمكن حساب المتوسط قبل حساب المجموع. **المعالجة(Process)** : عملية حسابية واضحة.
المخرجات	PRINT Average	**المخرجات(Output)** : يجب أن تنتج الخوارزمية نتيجة واحدة على الأقل، وهي المتوسط.
النهاية	END	**الفعالية/الانتهاء(Finiteness)** : ضمان توقف الخوارزمية.

التمرين 2: تطبيق بنية التحكم(Decision Making)

المشكلة: اكتب خوارزمية لتحديد ما إذا كان الطالب ناجحاً أم راسباً في مادة ما، على افتراض أن درجة النجاح هي **50**.

الخطوة	الشفرة الوهمية(Pseudocode)	الخاصية المطبقة
1. البداية	START	التسلسل.
2. الإدخال	READ Student_Score	المدخلات.
3. نقطة القرار	IF Student_Score >= 50 THEN	**الدقة والمنطق:** تحديد الشرط بدقة متناهية> =50.
4. نتيجة صحيحة	PRINT "ناجح"	المعالجة.
5. نتيجة خاطئة	ELSE / PRINT "راسب"	**تدفق التحكم (Control Flow):** تحديد مسار بديل.
6. النهاية	END	الانتهاء.

شرح تدفق التحكم: يوضح هذا المثال أهمية **نقطة القرار (IF)** في تغيير مسار تنفيذ البرنامج بناءً على تقييم الشرط، فإذا تحقق الشرط، سيتم تنفيذ أمر الطباعة الأول وتجاهل أمر الطباعة الثاني، والعكس صحيح، وهذا هو جوهر **المنطق والدقة** في البرمجة.

التمرين 3: تحليل خصائص البرمجة في مثال عملي

افترض أن لديك دالة برمجية تقوم بفرز قائمة طويلة من الأسماء أبجدياً، حلل هذه الدالة بناءً على الخصائص الأربع الرئيسية للبرمجة:

الخاصية	كيف تنطبق على دالة فرز الأسماء؟	أهميتها في هذا السياق
الدقة والمنطق	يجب استخدام **خوارزمية فرز محددة** (مثل فرز الفقاعات أو الإدراج)، بحيث تكون خطوات المقارنة والتبديل واضحة.	يضمن أن يتم ترتيب الأسماء بشكل صحيح وأبجدي (A-Z)دون ترك أي اسم في غير موضعه الصحيح.
التسلسل	يجب أن يتم تنفيذ خطوات الخوارزمية بالترتيب الصحيح (مثل: قارن العنصر 1 بـ 2، ثم 2 بـ 3، ثم كرر)	إذا تم الخلط بين خطوات المقارنة والتبديل، ستكون القائمة الناتجة غير مرتبة أو تحتوي على أخطاء.
العمومية	يجب أن تقبل الدالة **أي قائمة أسماء** مهما كان عددها (5 أسماء أو 5000 اسم) ولا تكون مقتصرة على قائمة واحدة فقط.	تجعل الدالة قابلة للاستخدام في أي تطبيق يتطلب فرز أسماء (نظام بنكي، نظام جامعي).
الفاعلية	استخدام خوارزمية فرز ذات **تعقيد زمني منخفض** (مثل O(N \log N)) بدلاً من خوارزمية أبطأ مثل O(N^2) .	تضمن أن يتم فرز آلاف الأسماء بسرعة فائقة بدلاً من استغراق دقائق، مما يحسن تجربة المستخدم.

التمرين 4: تحديد نوع اللغة المناسب للمهمة

لغات البرمجة المختلفة تتناسب مع مهام مختلفة بسبب خصائصها (قربها من العتاد أو سهولة كتابتها). املأ الجدول التالي لتحديد الخيار الأنسب:

المهمة المطلوبة	نوع اللغة الأنسب (منخفضة أم عالية المستوى)	سبب الاختيار	اللغة المقترحة (مثال)
برمجة نظام التشغيل (Operating System)	**منخفضة المستوى**	تتطلب **تحكماً مباشراً بالذاكرة** والمعالج لتحقيق أقصى سرعة وأداء (الفاعلية).	Cأو لغة التجميع (Assembly)
تطوير تطبيق ويب تفاعلي(Frontend)	**عالية المستوى**	تتطلب سرعة في التطوير وسهولة في التعامل مع المنطق المعقد على مستوى المستخدم.	JavaScript
بناء نموذج للذكاء الاصطناعي(AI Model)	**عالية المستوى**	الاعتماد على **مكتبات ضخمة** (مثل NumPy و Pandas)وضرورة سهولة القراءة وتجريد تفاصيل العتاد.	Python
تطوير نظام تحكم لمعدات طبية دقيقة	**منخفضة المستوى**	تتطلب **أعلى درجات الدقة والتحكم بالعتاد المدمج** والذاكرة المحدودة.	Cأو لغة التجميع

ملخص الأشكال الهندسية في مخطط الانسياب:

لفهم كيفية ترجمة الخوارزميات إلى مخططات (وهو أساس التصميم البرمجي)، إليك تذكير بأهم الأشكال المستخدمة:

الشكل الهندسي	اسم الشكل	دلالته في الخوارزمية
الشكل البيضاوي	Start/End	**البداية والنهاية**)يضمن خاصية الانتهاء).
متوازي الأضلاع	Input/Output	**الإدخال والإخراج**)يمثل المدخلات والمخرجات).
المستطيل	Process	**المعالجة** أو العمليات الحسابية (يمثل خاصية المعالجة).
المُعين	Decision	**القرار أو الشرط**)يمثل بنية **IF/ELSE** ويطبق خاصية الدقة والمنطق).
الأسهم	Flow Lines	**خطوط التدفق**)توضح خاصية التسلسل واتجاه التنفيذ).