الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للأستاذة ورقلة
قسم الرياضيات

مطبوعة بيداغوجية:

إعلام آلي: دروس وتمارين

موجهة لطلبة السنة الثانية رياضيات متوسط/ثانوي

إعداد: د. ياسين خالدي
أستاذ محاضر –ب-

السنة الجامعية: 2026/2025

الديباجة

في ظل التطور السريع الذي يشهده العالم في مجال التكنولوجيا الرقمية والمعلوماتية، أصبح من الضروري إعداد كفاءات قادرة على فهم أساسيات الإعلام الآلي واستخدامها بفعالية في مختلف الميادين العلمية والتربوية. وانطلاقا من أهمية هذا المجال في تكوين الأستاذ المستقبلي، تأتي هذه المطبوعة البيداغوجية لتكون أداة دعم وتوجيه لطلبة السنة الثانية رياضيات متوسط/ثانوي، قصد تمكينهم من اكتساب المعارف والكفاءات الأساسية في مقياس الإعلام الآلي.

تهدف هذه المطبوعة إلى تسهيل عملية التعلم الذاتي وتدعيم الجانب التطبيقي للمحتوى النظري، من خلال عرض مبسط ومنهجي للمفاهيم الأساسية مدعوم بأمثلة وتمارين تطبيقية تساعد الطالب على فهم وتحليل المشكلات بطريقة منطقية ومنهجية.

كما تسعى هذه المطبوعة إلى مواكبة متطلبات المقاربة بالكفاءات المعتمدة في التعليم العالي، والتي تركز على جعل المتعلم محور العملية التعليمية، من خلال توجيهه نحو اكتساب المهارات العملية والفكرية التي تمكنه من ربط النظرية بالتطبيق، وتوظيف ما تعلمه في مواقف جديدة خلال مساره الأكاديمي والمهني.

ملخص

تُقدّم هذه المطبوعة البيداغوجية استكشافًا معمّقًا للمفاهيم الأساسية في علوم الحاسوب والبرمجة، وهي موجهة لطلبة السنة الثانية في تخصص الرياضيات. تشمل المطبوعة مجموعة واسعة من المواضيع، مثل مكونات الأجهزة والبرمجيات، وأنظمة العد المختلفة، ومبادئ الخوارزميات، بالإضافة إلى أساسيات البرمجة باستخدام لغة بايثون. تعتمد المطبوعة على نهج منظم لبناء المعرفة، يجمع بين الأسس النظرية والتمارين العملية لتعزيز مهارات حل المشكلات لدى الطلاب. تُساعد مقدمة البرمجة بلغة بايثون الطلاب على اكتساب الأدوات اللازمة لكتابة الكود وتحليله بفعالية. بشكل عام، تُشكّل هذه المطبوعة قاعدة قوية للطلبة لمواصلة دراستهم في علوم الحاسوب وتطبيق التفكير الحاسوبي لحل المسائل الرياضية والمشكلات العملية.

الكلمات المفتاحية: علوم الحاسوب، الخوارزميات، أنظمة العد، البرمجة بلغة بايثون، التفكير الحاسوبي، الرياضيات، مطبوعة بيداغوجية، حل المشكلات.

Abstract

This pedagogical publication provides an in-depth exploration of fundamental concepts in computer science and programming, designed specifically for second-year students in mathematics. It covers a wide range of topics, including the essential components of computer hardware and software, various numbering systems, the principles of algorithms, and the fundamentals of programming using Python. The publication offers a structured approach to building knowledge, combining both theoretical foundations and practical exercises to enhance students' problem-solving skills. The introduction to Python programming equips students with the necessary tools to write and analyze code efficiently. Overall, this pedagogical material serves as a solid foundation for students to pursue further studies in computer science and apply computational thinking to solve mathematical and real-world problems.

Keywords: Computer Science, Algorithms, Number Systems, Python Programming, Computational Thinking, Mathematics, Pedagogical Material, Problem Solving

الفهرس

قائمة الجداول

قائمة الأشكال

المقـدمة

تتناول هاته المطبوعة البيداغوجية مجموعة من المفاهيم والأساسيات في مجال الإعلام الآلي والبرمجة، موجّهًة لطلبة السنة الثانية تخصص رياضيات في التعليم المتوسط والثانوي. تهدف إلى تقديم المعرفة الأساسية التي تساعد على فهم كيفية عمل الحواسيب وبرمجتها بشكل صحيح، مع ربط هذه المفاهيم بأساليب تطبيقية من خلال التمارين العملية.

تنظيم المطبوعة:

1- الفصل الأول: علوم الحاسوب

يستعرض هذا الفصل المبادئ الأساسية للحاسوب، مثل مكونات الأجهزة (Hardware) والبرمجيات (Software)، وأنظمة التشغيل المختلفة مثل DOS و Windows و Linux. بالإضافة إلى ذلك، يتم تقديم لغات البرمجة وأدواتها.

2- الفصل الثاني: أنظمة العد

يركز على شرح الأنظمة العددية المستخدمة في الحوسبة، مثل النظام الثنائي والثماني والسادس عشري، مع تطبيقات على كيفية التحويل بين هذه الأنظمة.

3- الفصل الثالث: الخوارزميات

يقدم هذا الفصل مفاهيم أساسية في تصميم الخوارزميات مع أمثلة على المتغيرات والجمل الشرطية والحلقات التكرارية، مما يوفر للطلبة الأدوات اللازمة لتطوير التفكير المنطقي.

4- الفصل الرابع: لغة البرمجة بايثون

يُخصص الفصل لتقديم أساسيات البرمجة بلغة بايثون، التي تُعتبر واحدة من أكثر لغات البرمجة شهرة وسهولة في التعلم، مع تطبيقات عملية على مختلف المواضيع التي تمت دراستها في الفصول السابقة.

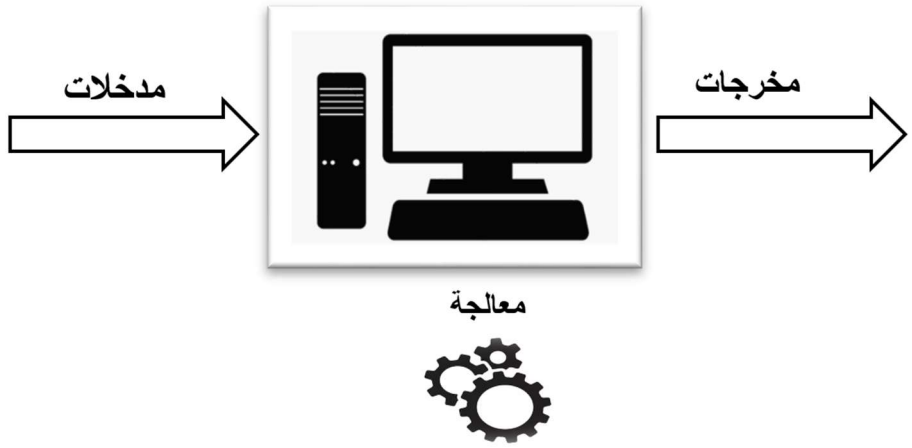
الفصل 1: علوم الحاسوب (Computer Sciences)

1. مقدمة

يعدّ علم الحاسوب من الركائز الأساسية التي تساهم في تطوير العديد من المجالات التكنولوجية والحياتية. يُعرَّف الحاسوب بأنه جهاز إلكتروني قادر على معالجة البيانات بسرعات فائقة ودقة عالية، مما يمكّنه من أداء وظائف متعددة تشمل التخزين والمعالجة والاتصال. من خلال هذا الفصل، نستعرض أبرز مفاهيم الحاسوب، مثل مكوناته الأساسية التي تنقسم إلى معدات وبرمجيات، بالإضافة إلى الأنظمة التشغيلية المختلفة مثل DOS، Windows، و Linux. كذلك، يتم تسليط الضوء على لغات البرمجة التي تُمكننا من برمجة الأجهزة لأداء مهام محددة. تهدف هذه الرحلة العلمية إلى فهم أعمق للبنية الأساسية التي يقوم عليها الحاسوب ودوره المتزايد في حياة الأفراد والمؤسسات.

2. ما هو الحاسوب (Computer)

الحاسوب أو الكمبيوتر هو جهاز إلكتروني قادر على تنفيذ مجموعة متنوعة من العمليات الحسابية والمنطقية بسرعة ودقة كبيرة. يعتمد الحاسوب في عمله على مبدأ استقبال البيانات، معالجتها، وتخزينها أو إخراجها في شكل معلومات مفيدة. يعتبر الحاسوب اليوم أحد الأدوات الرئيسية في العديد من المجالات مثل العلوم، التعليم، الأعمال، والاتصالات. الشكل 1 يوضح مبدأ عمل الكمبيوتر.



الشكل 1. مبدأ عمل الكبيوتر.

أهم الخصائص الرئيسية التي تميز الحاسوب عن غيره من الأدوات الإلكترونية هي:

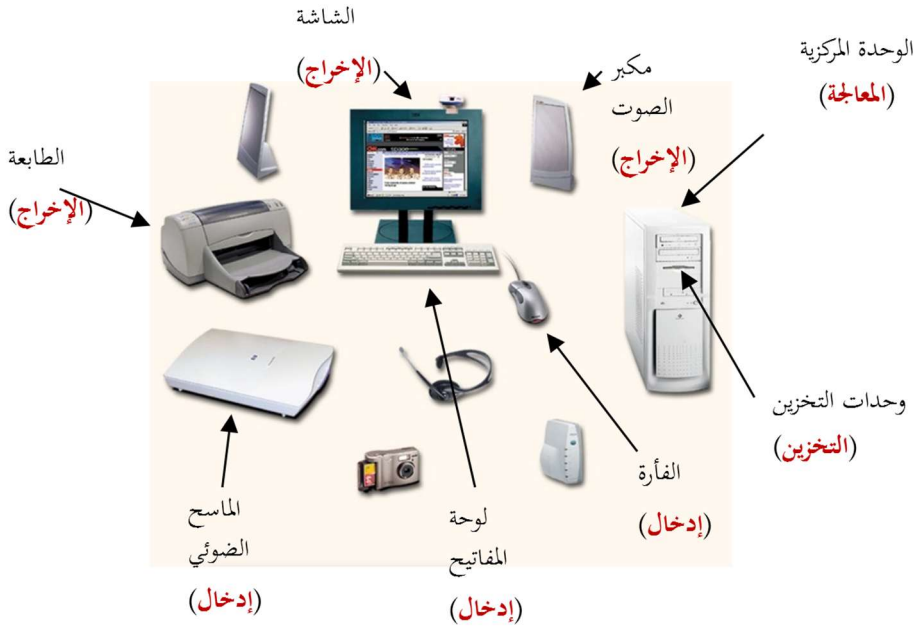
1) **قابلية البرمجة (Programmable):** الحاسوب يمكن برمجته لتنفيذ العديد من المهام وفقاً لتعليمات محددة.
2) **السرعة (Speed):** يتميز الحاسوب بسرعة فائقة في معالجة البيانات مقارنةً بالطرق التقليدية.

3) **الموثوقية (Reliability):** يعتبر الحاسوب جهازًا موثوقًا به، حيث يمكنه تنفيذ المهام بدون أخطاء كبيرة طالما أنه مبرمج بشكل صحيح.
4) **الدقة (Accuracy):** يقوم الحاسوب بتنفيذ العمليات الحسابية والمنطقية بدقة عالية، مما يقلل من نسبة الخطأ في النتائج.
5) **تخزين كميات هائلة من البيانات والمعلومات (Storage):** يمتلك الحاسوب قدرة تخزينية هائلة، تُمكّنه من حفظ كميات كبيرة من البيانات لفترات طويلة.
6) **التواصل مع أجهزة الكمبيوتر الأخرى (Communication):** يتمتع الحاسوب بإمكانية الاتصال مع أجهزة حواسيب أخرى أو شبكات عبر تقنيات الاتصال المتنوعة.

هذه الخصائص تجعل الحاسوب أداة قوية ومتعددة الاستخدامات في العديد من المجالات.

1.2. مكونات الكمبيوتر

الشكل 2 يبين مكونات الحاسوب الأساسية التي تنقسم إلى ثلاث فئات رئيسية: الإدخال، الإخراج، والمعالجة.



الشكل 2. مكونات الحاسوب الأساسية.

1- أجهزة الإدخال:

هي الأدوات التي تستخدم لإدخال البيانات إلى الحاسوب. تشمل هذه الأجهزة:

- الفأرة: تستخدم للتفاعل مع الحاسوب عبر النقر والتحريك.

- لوحة المفاتيح: تستخدم لإدخال النصوص والأوامر.
- الماسح الضوئي: يحول المستندات والصور الورقية إلى صيغة رقمية.
- الكاميرا الرقمية: تلتقط الصور الرقمية لإدخالها إلى الحاسوب.

2- أجهزة الإخراج:

هي الأجهزة التي تعرض البيانات أو المعلومات بعد معالجتها. تشمل هذه الأجهزة:

- الشاشة: تعرض الصور والنصوص بشكل مرئي.
- الطابعة: تطبع البيانات والمستندات الورقية.
- مكبر الصوت: يصدر الأصوات بعد معالجتها من الحاسوب.

3- وحدة المعالجة المركزية:

هي تمثل عقل الحاسوب وتقوم بمعالجة البيانات المدخلة، وتتحكم في جميع العمليات.

4- وحدات التخزين:

تُستخدم لتخزين البيانات على المدى الطويل مثل الملفات والمستندات.

هذه المكونات تعمل معًا لتنفيذ مختلف المهام التي يقوم بها الحاسوب، حيث يتم إدخال البيانات عبر أجهزة الإدخال، معالجتها داخل الوحدة المركزية، ثم إخراج النتائج عبر أجهزة الإخراج أو تخزينها.

يمكن أن نصنف مكونات الكمبيوتر إلى صنفين: المعدات (Hardware) والبرمجيات (البرمجيات):

1- المعدات: أجهزة إلكترونية مختلفة تؤدي العمليات الأساسية.

2- البرمجيات: برامج تقوم بتنسيق العمليات الأساسية لإنجاز المهام.

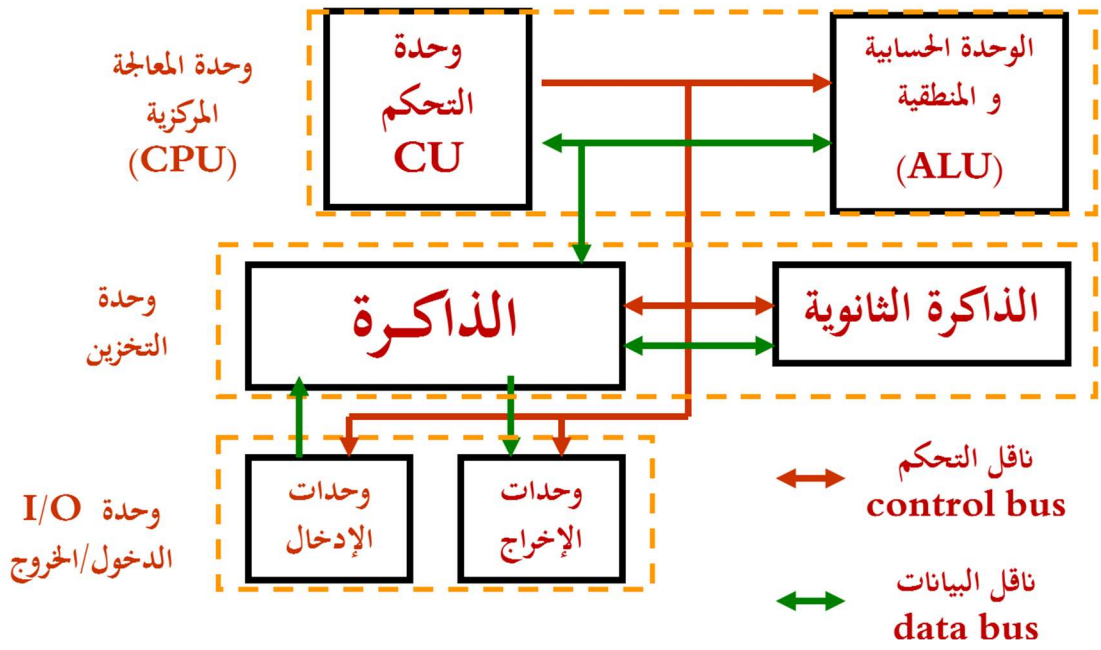
2.2. المعدات (Hardware)

مكونات الحاسوب المادية أو المعدات (Hardware) هي الأجزاء الملموسة والمادية التي يتكون منها الحاسوب وتعمل معًا لتوفير القدرة على تشغيل البرمجيات وتنفيذ العمليات المختلفة. تشمل هذه المكونات:

- **وحدة المعالجة المركزية (CPU):** تضم وحدة التحكم والوحدة الحسابية والمنطقية.
- **وحدة التحكم (CU):** تضم وحدة التحكم (Control Unit) تقوم وحدة التحكم بتنظيم وتكامل عمليات الكمبيوتر والتحكم بشكل متكرر بجلب التعليمات من وحدة الذاكرة إلى وحدة التحكم عبر ناقل البيانات. ثم تقوم وحدة التحكم بعد ذلك بتفسير التعليمات (Interpretation) وتنسيق عمليات الوحدات الأخرى عبر ناقل التحكم.

- **الوحدة الحسابية-المنطقية (ALU):** الدور الأساسي للوحدة الحسابية-المنطقية (Arithmitic-Logic Unit) هو إجراء العمليات الرياضية بناءً على العمليات الحسابية الأساسية مثل الجمع والطرح والضرب والقسمة. ومسؤولة أيضًا عن اتخاذ القرارات بناءً على العمليات المنطقية مثل المساواة (=، ≠) والعلاقات (<، ≧، >، ≦).
- **الذاكرة (Memory):** تعمل كوسيط لتخزين البيانات التي يتم معالجتها أو استخدامها بشكل مؤقت. يمكن أن تكون هذه الذاكرة هي ذاكرة الوصول العشوائي (RAM) التي تُستخدم لتخزين البيانات أثناء العمليات وتتميز بكونها قصيرة المدى وسريعة الوصول ومنخفضة السعة مقارنة بالذاكرة الثانوية.
- **الذاكرة الثانوية (Secondary Memory):** تمثل وحدات التخزين الدائمة مثل الأقراص الصلبة أو وحدات التخزين ذات الحالة الصلبة، وهي تُستخدم لحفظ البيانات لفترات طويلة وتتميز بأنها طويلة المدى وبطيئة الوصول وعالية السعة.
- **وحدات الإدخال والإخراج (I/O):** تشمل جميع الأجهزة التي تسمح بإدخال البيانات إلى الحاسوب أو إخراجها منه، مثل لوحة المفاتيح والشاشة والطابعة.
- **ناقل البيانات (Data Bus):** يُستخدم لنقل البيانات بين مكونات الحاسوب المختلفة مثل الذاكرة، وحدة المعالجة المركزية، ووحدات الإدخال والإخراج.
- **ناقل التحكم (Control Bus):** يُستخدم لنقل إشارات التحكم بين وحدة التحكم والمكونات الأخرى في الحاسوب، لتنظيم العمليات وضمان تنفيذ التعليمات بشكل صحيح.

الشكل 3 يبين معدات الكمبيوتر.



الشكل 3. معدات الكمبيوتر (Hardware).

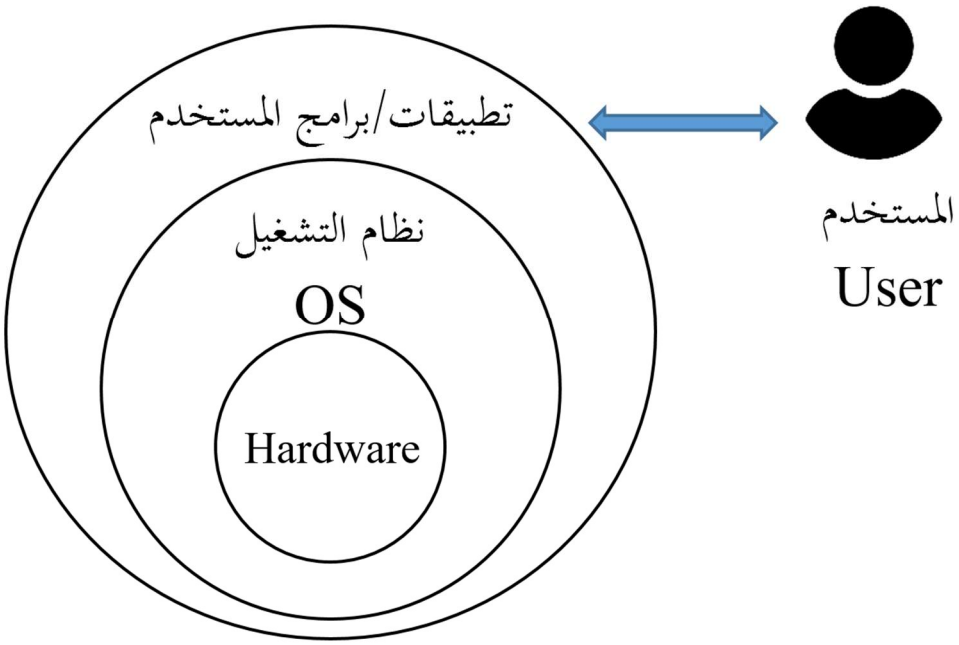
3.2. البرمجيات (Software)

البرمجيات (Software) تمثل الجزء الثاني الأساسي في الحاسوب، وهي المجموعة غير الملموسة من البرامج والتعليمات التي تتحكم في مكونات الحاسوب المادية (Hardware) وتوجهها لتنفيذ المهام. البرمجيات هي التي تمكن المستخدم من التفاعل مع الحاسوب وأداء وظائف متنوعة مثل معالجة النصوص، تصفح الإنترنت، أو تشغيل الألعاب.

تنقسم البرمجيات إلى فئتين رئيسيتين:

1) **برمجيات النظام (System Software):** وهي البرمجيات التي تدير مكونات الحاسوب وتتحكم في عملها. من أمثلتها نظام التشغيل (Operating System) مثل ويندوز أو لينكس، الذي يعمل كوسيط بين المستخدم والمكونات المادية، حيث يدير الموارد مثل الذاكرة والمعالج ويوفر بيئة لتشغيل التطبيقات الأخرى.
2) **البرمجيات التطبيقية (Application Software):** هي البرامج التي يستخدمها المستخدم لتنفيذ مهام محددة. تتضمن برامج تحرير النصوص مثل Microsoft Word، برامج التصميم مثل Adobe Photoshop، أو متصفحات الإنترنت مثل Google Chrome. هذه البرامج تعتمد على برمجيات النظام لتعمل بشكل صحيح.

البرمجيات تعمل بالتكامل مع الأجهزة المادية، حيث تقدم التعليمات اللازمة لتشغيل المهام المعقدة، وتحويل الأوامر إلى إشارات تستطيع المكونات المادية فهمها وتنفيذها. بدون البرمجيات، تبقى الأجهزة المادية غير قادرة على العمل أو تقديم الفائدة للمستخدم. الشكل 4 يوضح البنية الطبقية للكمبيوتر.



الشكل 4. تمثيل البنية الطبقية للكمبيوتر.

2.4. تمثيل البيانات وتخزينها (Data representation and storage)

الشكل في الحاسوب، يتم تخزين البيانات باستخدام النظام الثنائي (Binary System) الذي يعتمد على وحدات البت (Bits) والبايت (Bytes). البت هو أصغر وحدة تخزين، ويمكن أن يكون إما 0 أو 1. عندما نستخدم أكثر من بت واحد، يمكننا تمثيل المزيد من المعلومات. على سبيل المثال، باستخدام 2 بت، يمكننا تشفير 4 خيارات مختلفة (00، 01، 10، 11). إذا استخدمنا 3 بت، يمكننا تمثيل 8 خيارات، وإذا استخدمنا 4 بت، يمكننا تمثيل 16 خيارًا.

لتمثيل جميع الحروف الإنجليزية (البالغ عددها 26 حرفًا)، نحتاج إلى 5 بتات فقط، حيث يمكن لـ 5 بتات تمثيل 32 خيارًا، وهو كافٍ لتشفير جميع الأحرف.

تُستخدم طرق تشفير قياسية مثل ASCII لتخزين الأحرف على شكل أرقام ثنائية، حيث يتم تحويل كل حرف إلى قيمة رقمية محددة. كمثال على التشفير القياسي هو تحويل الأحرف إلى أرقام باستخدام كود ASCII، حيث يتم تحويل كل حرف إلى سلسلة من الأرقام الثنائية التي يمكن تخزينها في الذاكرة.

أكثر وحدات التخزين استعمالا هي:

- البايت (Byte): يتكون من 8 بتات.

- الكيلوبايت (KB): يساوي 1024 بايت.
- الميجابايت (MB): يساوي 1024 كيلوبايت.
- الجيجابايت (GB): يساوي 1024 ميجابايت، أو حوالي 8.6 مليار بت.

الذاكرة في الحاسوب تعمل على أساس هذه الوحدات لتخزين كميات هائلة من البيانات، مثل النصوص والصور والبرامج، وتستخدم هذه البتات والبايتات في جميع مستويات التخزين، سواءً في الذاكرة المؤقتة (RAM) أو في وحدات التخزين الدائمة مثل الأقراص الصلبة (HDD) أو وحدات التخزين ذات الحالة الصلبة (SSD).

3. نظام التشغيل DOS

نظام التشغيل DOS (Disk Operating System) هو نظام تشغيل بسيط تم تطويره في أوائل الثمانينيات من قبل شركة مايكروسوفت لأجهزة الكمبيوتر الشخصية. يُعتبر DOS واحدًا من أول أنظمة التشغيل الشائعة لأجهزة الكمبيوتر الشخصية، وكان له دور كبير في بدايات صناعة الحواسيب. استخدم DOS واجهة نصية، حيث يتفاعل المستخدم مع النظام عبر كتابة أوامر نصية بدلاً من استخدام واجهة رسومية كما هو الحال في أنظمة التشغيل الحديثة.

3.1. ميزات نظام التشغيل DOS

- **واجهة سطر الأوامر (Command Line Interface):** يتم تشغيل DOS باستخدام واجهة نصية تُعرف بـ CLI، حيث يكتب المستخدم الأوامر مباشرة للتفاعل مع النظام.
- **التحكم في الملفات:** يسمح DOS بإدارة الملفات والمجلدات على الأقراص الصلبة وأقراص الفلوبي.Floppy Disks)
- **إدارة القرص:** يتعامل DOS مع الأقراص بشكل مباشر، ويسمح بتهيئتها (Format) وتجهيزها للاستخدام، وكان يدعم أجهزة التخزين مثل الأقراص الصلبة وأقراص الفلوبي.
- **البساطة والكفاءة:** كان DOS نظامًا بسيطًا وفعالًا جدًا نظرًا لمتطلباته المتواضعة من حيث العتاد (الذاكرة والمعالجة). وقد كان مناسبًا للأجهزة في ذلك الوقت.

في المقابل، نظام التشغيل DOS به عدة عيوب أهمها:

- عدم وجود واجهة رسومية: على عكس الأنظمة الحديثة مثل Windows، لم يكن DOS يحتوي على واجهة رسومية، مما جعله أكثر صعوبة في الاستخدام بالنسبة للمستخدمين العاديين.
- قدرات محدودة متعددة المهام DOS :لم يكن يدعم تشغيل العديد من البرامج في وقت واحد.

- قيود الذاكرة DOS :كان محدودًا في استغلال الذاكرة العشوائية، مما جعله غير مناسب للتطبيقات المتطورة أو ذات الحجم الكبير.

أشهر إصدار لنظام DOS هو MS-DOS (Microsoft Disk Operating System) الذي تم تطويره من قبل مايكروسوفت. كان DOS النظام الأساسي الذي بنيت عليه الإصدارات الأولى من Windows، حيث كان Windows 3.1 وما قبله يعتمد على DOS كنظام تشغيل أساسي. لكن, مع تطور أنظمة التشغيل، خصوصًا مع ظهور Windows بإصداراته المختلفة التي تقدم واجهة مستخدم رسومية وتحكمًا أفضل في الموارد، قل استخدام DOS وبحلول أواخر التسعينيات، أصبح DOS قديمًا وغير مستخدم على نطاق واسع.

نظام التشغيل DOS كان له دور كبير في بداية انتشار الحواسيب الشخصية، وقد ساعد في تمهيد الطريق لتطوير أنظمة تشغيل أكثر تعقيدًا وسهولة في الاستخدام.

3.2. أهم الأوامر في نظام التشغيل DOS

في نظام التشغيل DOS، يتم التعامل مع النظام من خلال كتابة أوامر نصية لتنفيذ العمليات المختلفة. يُظهر لنا الموجّه (Prompt) الدليل الحالي الذي نتواجد فيه. على سبيل المثال، عندما يظهر الموجّه بالشكل:

C:\dos\commands>

فهذا يعني أننا حالياً في المجلد commands الموجود داخل المجلد dos على القرص C. عند استخدام أي أمر، سيتم تنفيذه في المجلد commands إلا إذا قمنا بتحديد مجلد آخر. فيما يلي أهم الأوامر المستخدمة في نظام DOS:

- **الأمر dir:** يعرض قائمة الملفات والمجلدات في الدليل الحالي.

الصيغة : dir [path]

مثال : dir C:\

- **الأمر copy:** يستخدم لنسخ الملفات من موقع إلى آخر.

الصيغة : copy [source] [destination]

مثال : COPY C:\file.txt D:\backup\

- **الأمر del:** يستخدم لحذف ملف معين.

الصيغة : del [file_name]

مثال : DEL C:\oldfile.txt

- **الأمر ren:** يستخدم لإعادة تسمية الملفات.

الصيغة : ren [current_name] [new_name]

مثال : REN oldfile.txt newfile.txt

- **الأمر MKDIR أو MD:** يستخدم لإنشاء مجلد جديد.

الصيغة : md [folder_path]

مثال : MD C:\newfolder

- **الأمر RMDIR أو RD:** يستخدم لحذف مجلد فارغ.

الصيغة : rd [folder_path]

مثال : RD C:\oldfolder

- **الأمر MOVE:** يستخدم لنقل الملفات أو إعادة تسميتها.

الصيغة : move [source] [destination]

مثال : MOVE C:\file.txt D:\newfolder\

- **الأمر RENAME:** لإعادة تسمية الملفات أو المجلدات.

الصيغة : rename [source] [destination]

مثال : RENAME C:\data.txt newdata.txt

- **الأمر tree:** تُظهر شجرة المجلدات الفرعية لمجلد ما، كما يمكن إظهار قائمة الملفات كذلك باستخدام الخاصية F/

الصيغة : rename [path] [/F]

مثال : TREE C:\dos /F

4. نظام التشغيل ويناوز (Windows)

نظام التشغيل ويندوز (Windows) هو نظام تشغيل رسومي تم تطويره من قبل شركة مايكروسوفت. يُعتبر واحداً من أكثر أنظمة التشغيل انتشاراً واستخداماً في العالم، حيث يعتمد عليه ملايين المستخدمين سواء على أجهزة الكمبيوتر الشخصية أو في بيئات العمل المختلفة. نظام التشغيل ويندوز يُستخدم في مجموعة واسعة من التطبيقات بدءاً من الاستخدام الشخصي وحتى الاستخدام الاحترافي في مجالات مثل إدارة الأعمال، التصميم، البرمجة، الألعاب، وغير ذلك.

4.1. مميزات نظام التشغيل ويندوز

- **واجهة المستخدم الرسومية (GUI):** يعتمد ويندوز على واجهة رسومية سهلة الاستخدام تتضمن أيقونات، قوائم، وأزرار تجعل التفاعل مع النظام بسيطًا وسهلًا حتى للمستخدمين غير المحترفين.
- **دعم متعدد البرامج (Multitasking):** يمكن للمستخدم تشغيل عدة برامج وتطبيقات في وقت واحد، مما يتيح تنفيذ العديد من المهام في نفس الوقت.
- **التوافق مع الأجهزة:** نظام ويندوز يدعم مجموعة كبيرة من الأجهزة والبرمجيات، مما يجعله متوافقاً مع معظم أجهزة الحاسوب وملحقاتها.
- **دعم الألعاب والبرمجيات المتنوعة:** يُعتبر ويندوز المنصة الرئيسية للألعاب والبرامج الاحترافية مثل برامج التصميم، الإنتاج، وغيرها، نظراً لانتشاره الواسع ودعم الشركات المطورة له.
- **التحديثات الأمنية:** توفر مايكروسوفت تحديثات دورية لنظام ويندوز لتعزيز الأمان وحماية المستخدمين من التهديدات الرقمية مثل الفيروسات والبرمجيات الخبيثة.
- **إدارة الملفات:** يوفر ويندوز مستعرض ملفات يتيح للمستخدمين تنظيم وإدارة الملفات والمجلدات بسهولة من خلال واجهة مرئية، مما يجعل الوصول إلى البيانات سريعاً ومباشراً.

4.2. إصدارات ويندوز الرئيسية

- **ويندوز 95 و98:** من أوائل إصدارات ويندوز التي قدمت واجهة مستخدم رسومية متقدمة ودعماً للبرامج القديمة.
- **ويندوز XP:** يعتبر أحد أنجح إصدارات ويندوز، حيث جاء بتحسينات كبيرة في الأداء والثبات، وتم استخدامه على نطاق واسع لعدة سنوات.
- **ويندوز 7:** إصدار ناجح آخر، حصل على شهرة كبيرة بفضل استقراره وأدائه المحسن مقارنة بالإصدارات السابقة.
- ويندوز 11: أحدث إصدار مستقر من ويندوز، حيث تم تحسين الواجهة والوظائف وتم إضافة ميزات جديدة مثل المساعد الشخصي كورتانا (Cortana) ودعم التحديثات المستمرة.

5. نظام التشغيل لينكس (Linux)

نظام التشغيل لينكس (Linux) هو نظام تشغيل مفتوح المصدر يعتمد على نواة (kernel) لينكس التي تم تطويرها لأول مرة من قبل "لينوس تورفالدس" في عام 1991. يُستخدم نظام لينكس في العديد من

المجالات، بدءًا من أجهزة الكمبيوتر الشخصية والخوادم وصولاً إلى أنظمة الهواتف الذكية (مثل أندرويد) والأجهزة المدمجة.

5.1. مميزات نظام التشغيل لينكس

- **مفتوح المصدر:** يتميز لينكس بكونه نظام تشغيل مفتوح المصدر، مما يعني أن شفرته البرمجية متاحة للجميع، ويمكن لأي شخص التعديل عليها أو استخدامها وفقًا لاحتياجاته.
- **الثبات والأمان:** يُعرف لينكس بأنه نظام مستقر وآمن، حيث يوفر بيئة تشغيل أقل عرضة للفيروسات والبرمجيات الخبيثة مقارنة ببعض الأنظمة الأخرى. يتمتع لينكس بإدارة فعالة للذاكرة والموارد، مما يجعله مناسبًا للعمل في البيئات الحرجة مثل الخوادم.
- **التخصيص:** يُعتبر لينكس من أكثر أنظمة التشغيل قابلية للتخصيص، حيث يمكن للمستخدمين تعديل الواجهة والأدوات حسب احتياجاتهم. توجد العديد من التوزيعات (Distributions) التي تلبي احتياجات مختلفة.
- **تعدد التوزيعات (Distros):** يتوفر لينكس في العديد من التوزيعات، وكل توزيع يأتي بميزات مختلفة. من أشهر التوزيعات:

  - أوبونتو (Ubuntu): واحدة من أكثر التوزيعات شعبية وسهلة الاستخدام، موجهة للمستخدمين الجدد والمحترفين على حد سواء.
  - فيدورا (Fedora): توزيع يركز على الابتكارات الحديثة ويستخدم كثيرًا من قبل المطورين.
  - ديبيان (Debian): توزيعة تعتمد على الاستقرار الشديد، وغالبًا ما تُستخدم في الخوادم.
  - أرك لينكس (Arch Linux): توزيعة موجهة للمستخدمين المتقدمين الذين يرغبون في تخصيص نظامهم بالكامل.

- **إدارة الحزم:** يوفر لينكس نظامًا قويًا لإدارة الحزم، يتيح تثبيت البرامج وتحديثها بسهولة. أشهر أنظمة إدارة الحزم هي:
  - APT في توزيعات ديبيان وأوبونتو.
  - YUM أو DNF في توزيعات فيدورا.
  - Pacman في توزيعات أرك لينكس.

- **الأداء والكفاءة:** يتمتع لينكس بأداء عالٍ ويستخدم الموارد بكفاءة، مما يجعله الخيار الأمثل لتشغيل الخوادم العملاقة والتطبيقات التي تتطلب أداءً كبيرًا.
- **مجتمع داعم:** بفضل طبيعته المفتوحة المصدر، يتمتع لينكس بمجتمع نشط من المطورين والمستخدمين الذين يساهمون باستمرار في تطوير النظام، تقديم المساعدة والدعم الفني.
- **مجاني:** معظم توزيعات لينكس متاحة مجانًا، مما يجعلها خيارًا اقتصاديًا للكثيرين سواء على مستوى الأفراد أو المؤسسات.

5.2. استخدامات نظام التشغيل لينكس

- **الخوادم:** يُستخدم لينكس بشكل واسع في تشغيل الخوادم نظرًا لاستقراره وأمانه العالي، ويُعتبر خيارًا مفضلًا للشركات الكبرى مثل جوجل وأمازون وفيسبوك.
- **التطوير والبرمجة:** يُفضل العديد من المطورين استخدام لينكس كبيئة لتطوير البرمجيات نظرًا لتوافر أدوات البرمجة وتكاملها مع النظام.
- الأنظمة المدمجة (Embedded Systems): يُستخدم لينكس في تشغيل العديد من الأجهزة المدمجة مثل أجهزة الراوتر، التلفزيونات الذكية، والأنظمة الروبوتية.
- **سطح المكتب:** على الرغم من أن لينكس يُستخدم بشكل رئيسي في الخوادم، فإنه يتمتع أيضًا بشعبية بين المستخدمين على أجهزة الكمبيوتر الشخصية بفضل توزيعات سهلة الاستخدام مثل أوبونتو.
- **الهواتف الذكية:** يعتمد نظام التشغيل أندرويد (Android)، الذي يعد الأكثر انتشارًا في الهواتف الذكية، على نواة لينكس.

في الجدول 1 مقارنة بين نظام التشغيل ويندوز ولينكس.

جدول 1. مقارنة بين نظام التشغيل وانداوز و لينكس.

المعيار	ويندوز (Windows)	لينكس (Linux)
المصدر	مغلق المصدر	مفتوح المصدر
التكلفة	قد يكون مكلفًا	مجاني
الأمان	أقل أمانًا، عرضة للفيروسات	عالي الأمان، نادرًا ما يُصاب بالفيروسات
التخصيص	قليل التخصيص مقارنة بلينكس	مرن جدًا وقابل للتخصيص الكامل
الاستقرار	مستقر لكن أقل مقارنة بلينكس في الخوادم	مستقر للغاية، نادر الأعطال
إدارة الحزم	إدارة برامج عبر متجر ويندوز وتثبيت يدوي	إدارة حزم قوية مثل APT و YUM
التوافق مع الأجهزة	متوافق مع معظم الأجهزة والملحقات	دعم قوي للخوادم وبعض الأجهزة المكتبية
دعم الألعاب	دعم قوي للألعاب	أقل دعمًا للألعاب، لكن الوضع يتحسن
سهولة الاستخدام	سهل الاستخدام ويستهدف المستخدمين العاديين	يتطلب معرفة تقنية متقدمة
إدارة الموارد	يستهلك الموارد بشكل أكبر	إدارة موارد فعّالة جدًا

6. مدخل حول لغات البرمجة

لغة البرمجة هي نظام من القواعد والمفردات المستخدمة لكتابة أوامر وتعليمات يمكن للجهاز الحاسوبي فهمها وتنفيذها. تختلف هذه اللغات عن اللغات الطبيعية التي يستخدمها البشر في التواصل اليومي، حيث تعتمد على قواعد صارمة وخوارزميات منطقية. الهدف الرئيسي من لغات البرمجة هو تسهيل عملية كتابة البرامج التي تنفذ مهام معينة، بدءًا من العمليات الحسابية البسيطة وحتى تصميم الأنظمة المعقدة مثل الذكاء الاصطناعي أو تطبيقات الهواتف الذكية.

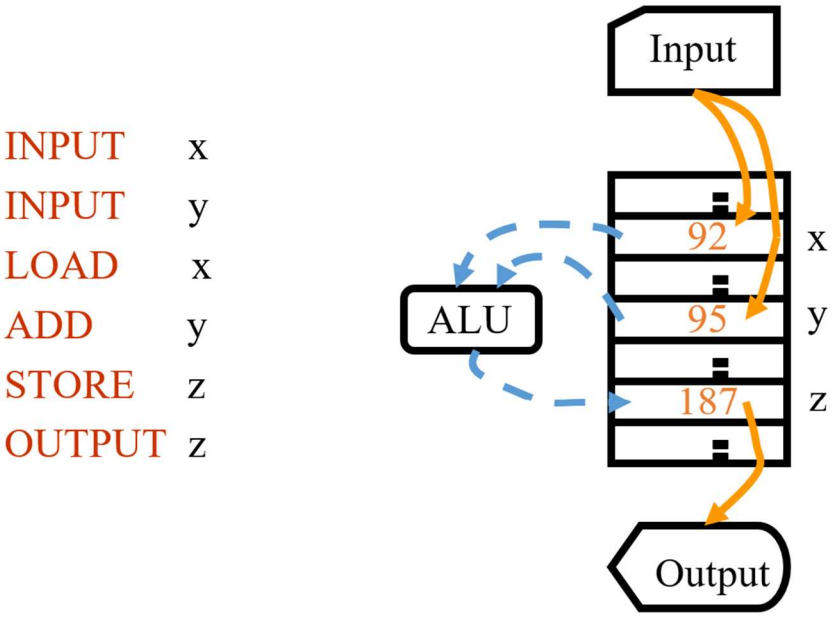
تطورت لغات البرمجة عبر عقود من الزمن، حيث شهدت تحولاً تدريجيًا من اللغات التي تعتمد على الأوامر المباشرة للجهاز (مثل لغة الآلة) إلى لغات عالية المستوى التي تعتمد على مفاهيم مجردة أكثر.

بدأت أولى لغات البرمجة الفعالة في الأربعينيات والخمسينيات من القرن الماضي مع ظهور لغة "Fortran"و"COBOL"، وتطورت اللغات بشكل ملحوظ لتصبح أكثر مرونة وسهولة في الاستخدام. مع مرور الوقت، ظهر العديد من اللغات الحديثة مثل"Python" ، "Java"، و"C++"، التي تتميز بقوتها وكفاءتها في معالجة البيانات وتصميم الأنظمة. هذه اللغات تسمح بكتابة برامج معقدة بطريقة أسهل وأكثر تنظيمًا مقارنة باللغات البدائية.

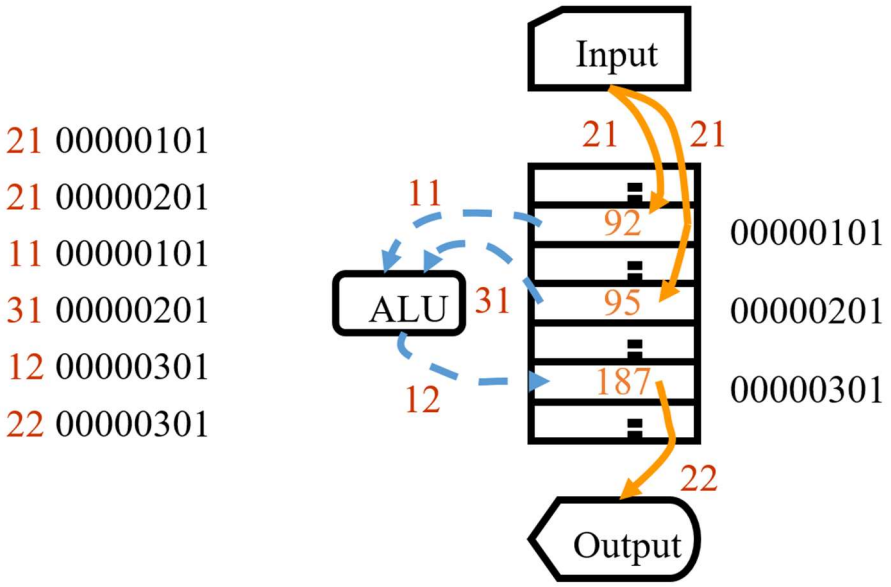
6.1. أنواع لغات البرمجة

توجد العديد من الأنواع المختلفة للغات البرمجة، ويمكن تصنيفها بناءً على عدة معايير. من أشهر التصنيفات:

1. **اللغات منخفضة المستوى (Low-Level Languages):** هذه اللغات تتعامل بشكل مباشر مع مكونات الحاسوب، مثل لغة الآلة (Machine Language) ولغة التجميع (Assembly Language). هذه اللغات صعبة التعلم والبرمجة بها معقد، لكنها تمنح تحكمًا مباشرًا ودقيقًا في العتاد المادي. الشكل 5 يقدم مثل لبرنامج بلغة التجميع, أما الشكل 4 يمثل نفس البرنامج مكتوب بلغة الآلة.



الشكل 5. مثال لبرنامج بلغة التجميع.



الشكل 6. مثال لبرنامج بلغة الآلة.

2. **اللغات عالية المستوى (High-Level Languages):** توفر هذه اللغات واجهات أكثر سهولة وقابلية للفهم للمبرمجين، حيث تعتمد على قواعد لغوية أقرب إلى اللغات البشرية. أمثلة على هذه اللغات تشمل بايثون (Python)، جافا (Java)، ولغة سي++ (C++). تعتبر هذه اللغات ملائمة لتطوير البرامج الكبيرة والمعقدة. الشكل 7 يقدم مثال لنفس البرنامج مكتوب بلغات مختلفة.

```cpp
#include<iostream>
using namespace std;

int main()
{

    cout<<"Hello World!"<<endl;
    return 0;

}
```

(أ)

```java
class Hello {

    public static void main(String[] args) {
        System.out.print("Hello World");
    }
}
```

(ب)

```
>>> print("Hello World!")
```

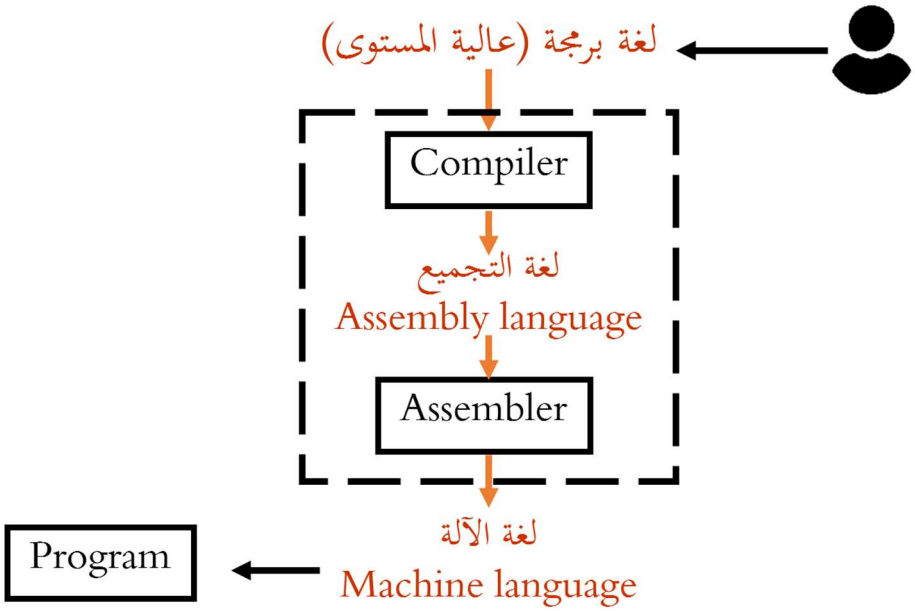
(ج)

الشكل 7. مثال لبرنامج يطبع على الشاشة عبارة "*Hello World*" مكتوب بـ: (أ) لغة C++، (ب) لغة جافا، (ج) لغة بايثون.

3. **لغات البرمجة النصية (Scripting Languages):** تستخدم هذه اللغات عادةً للمهام أو تطوير التطبيقات الصغيرة أو تصميم صفحات الويب. من أمثلتها"JavaScript" ، "Ruby"، و ."PHP"تتميز هذه اللغات بأنها سهلة التعلم وسريعة التنفيذ.

6.2. المترجم (Compiler)

المترجم (Compiler) هو برنامج أساسي في مجال البرمجة، حيث يُستخدم لتحويل الكود المكتوب بلغات البرمجة عالية المستوى (مثل C++، Java، وPython) إلى لغة الآلة أو الكود الثنائي (Binary Code) الذي يمكن لوحدة المعالجة المركزية (CPU) في الحاسوب فهمه وتنفيذه. تُعد هذه العملية ضرورية لأن الأجهزة لا تفهم إلا التعليمات بلغة الآلة، بينما تُكتب البرمجيات بلغات يسهل على البشر قراءتها وتعديلها. الشكل 8 يوضح دور المترجم في عملية البرمجة.



الشكل 8. دور المترجم في عملية البرمجة.

6.3. مراحل عملية الترجمة أو التجميع (Compiling)

1) **تحليل الكود (Lexical Analysis):** يقوم المترجم بتحليل الكود المصدر (Source Code) والتعرف على المكونات الأساسية مثل الكلمات المفتاحية، المتغيرات، العوامل (Operators)، والفواصل. الهدف من هذه الخطوة هو تحويل النص البرمجي إلى مجموعة من الرموز (Tokens).

2) **تحليل نحوي (Syntax Analysis):** يتحقق المترجم من أن الرموز المولدة تتبع القواعد النحوية الخاصة باللغة البرمجية. في حالة وجود خطأ نحوي، يتم إرسال تنبيه أو رسالة خطأ للمبرمج لتصحيحه.
3) **تحليل معنوي (Semantic Analysis):** بعد التحقق من قواعد اللغة، يقوم المترجم بفحص الصحة المنطقية للكود، مثل التحقق من نوع البيانات (Data Types) والتأكد من أن المتغيرات قد تم تعريفها قبل استخدامها.
4) **توليد الكود الوسيط (Intermediate Code):** بعد التحليلات السابقة، يقوم المترجم بإنشاء كود وسيط يُعتبر تمثيلاً بينيًا غير نهائي للبرنامج.
5) **تحسين الكود (Optimization):** يحسن المترجم الكود الوسيط لتحسين أداء البرنامج وتقليل حجم الكود النهائي.

6) **توليد الكود النهائي (Code Generation):** في هذه المرحلة، يتم تحويل الكود الوسيط إلى كود الآلة النهائي، وهو الكود القابل للتنفيذ من قبل الجهاز.
7) **الربط (Linking):** بعد توليد الكود، يتم ربط الوحدات البرمجية المختلفة مع المكتبات الخارجية اللازمة لإنشاء ملف تنفيذي نهائي.

7. تمارين

1- ما هو دور الـ (CPU) في الحاسوب؟
أ) الإدخال البيانات ب) تخزين الإخراج ج) التواصل بين البيانات د) المعالجة المركزية

2- ما البرنامج الذي يتحكم في الأجزاء الأساسية للنظام ويبدأ عملية تشغيل الحاسوب؟
أ) برنامج المعالج ب) نظام الشبكة ج) نظام التشغيل د) برنامج الذاكرة

3- ما الفرق بين الذاكرة العشوائية (RAM) والذاكرة القابلة للبرمجة (ROM)؟
أ) RAM قابلة للكتابة والقراءة بينما ROM قابلة للقراءة فقط
ب) RAM قابلة للقراءة فقط بينما ROM قابلة للكتابة والقراءة
ج) كل منهما قابل للكتابة والقراءة
د) كل منهما قابل للقراءة فقط

4- ما هي أسرع ذاكرة في عملية الوصول بين الأنواع التالية؟
أ) القرص الصلب ب) ROM ج) RAM د) القرص المرن

5- بماذا تهتم علوم الحاسوب (Computer science)؟
أ) البرمجيات ب) المعدات ج) هندسة الحاسوب د) حل المشكلات

6- ماذا يمكن أن يفعل الكمبيوتر؟

أ) إدخال البيانات ب) إخراج البيانات ج) معالجة البيانات د) تخزين البيانات

7- أي من الخيارات التالية ليس نوعًا من أنواع أجهزة التخزين الثانوية؟

أ) قرص صلب ب) ذاكرة وصول عشوائي ج) قرص مدمج د) قرص مرن

8- ما هي الوحدة المسؤولة عن اجراء العمليات الرياضية؟

أ) CPA ب) CU ج) ALU د) Calculator

9- في صورة أي نظام تقوم أجهزة الحاسوب بحفظ البيانات؟

أ) نظام ثنائي ب) نظام ثماني ج) نظام عشري د) نظام سداسي عشر

10- باستخدام 10 بت (bits), كم من معلومة مختلفة يمكن تشفيرها في النظام الثنائي؟

أ) 20 ب) 100 ج) 200 د) 1024

11- اذا كان لديك أبجدية تحتوي على 50 حرف, كم هو عدد البتات <u>الأدنى</u> الكافي لتشفير كل الحروف في النظام الثنائي؟

أ) 25 ب) 10 ج) 6 د) 8

12- جهاز به ذاكرة ذات سعة 1.5 جيغابايت, كم هي سعة الذاكرة بوحدة الكيلوبايت؟

أ) 1572864 ب) 1536 ج) 12288 د) 12582912

8. خاتمة

تناول هذا الفصل الأساسيات الجوهرية في مجال الحاسوب، بما في ذلك مكونات الأجهزة والبرمجيات، وأهمية أنظمة التشغيل المختلفة في تنظيم عمل الحواسيب. كما تم تسليط الضوء على الفروقات بين أنظمة التشغيل المعروفة مثل Windows وLinux، مما يتيح للمستخدمين اختيار النظام المناسب حسب احتياجاتهم. وأخيرًا، تطرّقنا إلى مدخل حول لغات البرمجة ودورها في التحكم بمختلف الأجهزة الحاسوبية. هذه المعرفة تشكل قاعدة متينة لفهم التطبيقات المتقدمة في الحاسوب، وتمهد للفصول القادمة التي ستتناول أنظمة العد، الخوارزميات وبرمجة الحاسوب بشكل أعمق.

الفصل 2: أنظمة العد (Number Systems)

1. مقدمة

في هذا الفصل، نستعرض مفهوم أنظمة العد التي تُعدّ الأساس الذي تعتمد عليه جميع الأجهزة الرقمية والحواسيب في معالجة البيانات وتمثيلها. تُستخدم أنظمة العد المختلفة في تمثيل الأعداد بطرق متعددة بناءً على قواعد وأسُس خاصة بكل نظام. من أبرز هذه الأنظمة: النظام العشري (Decimal)، النظام الثنائي (Binary)، النظام الثماني (Octal)، والنظام السادس عشري (Hexadecimal). يُسلط هذا الفصل الضوء على كل من هذه الأنظمة، موضحًا كيفية استخدامها، مع تقديم طرق مختلفة للتحويل بينها. كما يهدف إلى تعزيز فهم الطلبة للعمليات الحسابية الأساسية داخل كل نظام عددي، مثل الجمع والطرح والضرب والقسمة، والتي تُعد ضرورية لفهم كيفية عمل الحواسيب وبرمجتها.

2. نظام العد base-r

تناول نظام العد هو طريقة تمثيل الأعداد باستخدام مجموعة من الرموز بناءً على قاعدة معينة r، حيث يشير r إلى عدد الرموز المستخدمة. كل نظام عد له أساس أو قاعدة r تحدد عدد الرموز الممكن استخدامها، ويعتمد تمثيل الأعداد على هذه القاعدة. كمثال العدد $(2024)_{10}$ هو العدد 2024 في النظام العشري.

3. الترميز الموضعي (Positional Notation)

الترميز الموضعي هو نظام يعتمد على قيمة الرقم بناءً على موضعه في العدد. في هذا النظام، تختلف قيمة الرقم بحسب المكان الذي يشغله، حيث يُضاعف كل موضع بقدرة معينة لقاعدة النظام العددي المستخدم. هذا الترميز يُعدّ أساس أنظمة العد الحديثة مثل العشري والثنائي والأنظمة الأخرى ذات القواعد المختلفة. يكون الترميز الموضعي على الشكل التالي:

$$N = (a_{n-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-m})_r$$

حيث:

N: العدد الذي يتم تمثيله

r: القاعدة أو الأساس

n: عدد الأرقام الصحيحة الموجودة على يسار نقطة الجذر

m: عدد الأرقام الكسرية على يمين نقطة الجذر

a_{n-1}: الرقم الأكثر أهمية (MSD)

a_{-m}: الرقم الأقل أهمية(LSD)

3. الترميز باستخدام كثير الحدود (Polynomial Representation)

يُستخدم الترميز بكثير الحدود لتمثيل الأعداد في أنظمة العد المختلفة، حيث يُعبَّر عن العدد على شكل كثير حدود، تعتمد حدوده على الأرقام المكوّنة للعدد وقواعد النظام العددي. هذا النوع من الترميز هو أداة رياضية قوية لفهم كيفية تمثيل الأعداد بناءً على القاعدة لكل نظام. يكون الترميز باستخدام كثير الحدود على الشكل التالي:

$$N = a_{n-1} \times r^{n-1} + a_{n-1} \times r^{n-1} + \ \ldots$$
$$+a_0 \times r^0 + a_{-1} \times r^{-1} + \ \ldots + a_{-m} \times r^{-m}$$
$$\Rightarrow N = \sum_{i=-m}^{n-1} a_i r^i$$

حيث:

N: العدد الذي يتم تمثيله

a_i: الأرقام التي تُكوّن العدد حيث $(r > a_i \geq 0)$

r: القاعدة أو الأساس

n: عدد الأرقام الصحيحة الموجودة على يسار نقطة الجذر

m: عدد الأرقام الكسرية على يمين نقطة الجذر

مثال:

$$N = (251.41)_{10} = 2 \times 10^2 + 5 \times 10^1 + 1 \times 10^0 + 4 \times 10^{-1} + 1 \times 10^{-2}$$

4. أنظمة العد

4.1. النظام الثنائي (Binary System)

النظام الثنائي هو نظام عدّ ذو أساس 2 (Base-2)، ويستخدم رقمين فقط، وهما: 0 و1. يُعد هذا النظام من أهم الأنظمة في مجال الحوسبة والإلكترونيات، حيث يعتمد كل ما يحدث داخل الأجهزة الرقمية على هذا التمثيل الثنائي للبيانات.

الرموز المستخدمة في النظام الثنائي هي = {0, 1}

مثال:

$$(11010.11)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$
$$= (26.75)_{10}$$

4.2. النظام الثماني (Octal System)

النظام الثماني هو نظام عدّ ذو أساس 8 (Base-8)، أي يستخدم ثمانية رموز فقط (0، 1، 2، 3، 4، 5، 6، 7). يُستخدم هذا النظام في بعض التطبيقات الحاسوبية والرقمية، حيث يُعتبر اختصارًا بسيطًا للنظام الثنائي، لأن كل 3 بتات ثنائية تعادل رقمًا واحدًا في النظام الثماني.

الرموز المستخدمة في النظام الثماني = {0, 1, 2, 3, 4, 5, 6, 7}

مثال:

$$(11010.11)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$
$$= (26.75)_{10}$$

4.3. النظام السادس عشري (Hexadecimal System)

النظام السادس عشري هو نظام عددي يستخدم الأساس 16 (base-16). يُعد أحد الأنظمة العددية الشائعة في علوم الحاسب، ويُستخدم بكثرة في تمثيل القيم الرقمية مثل عناوين الذاكرة، الألوان في تصميم الويب، وأحياناً في معالجة البيانات.

الرموز المستخدمة في النظام السادس عشري = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, F, E, D, C, B, A}

مثال:

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = (46687)_{10}$$

في الجدول 2 الأعداد من الصفر إلى غاية 16 بمختلف أنظمة العد.

جدول 2. الأعداد من الصفر إلى غاية 16 بمختلف أنظمة العد.

النظام السادس عشر	النظام الثماني	النظام الثنائي	النظام العشري
0	0	0	0
1	1	1	1
2	2	10	2
3	3	11	3
4	4	100	4
5	5	101	5
6	6	110	6
7	7	111	7
8	10	1000	8
9	11	1001	9
A	12	1010	10
B	13	1011	11
C	14	1100	12
D	15	1101	13

14	1110	16	E
15	1111	17	F
16	10000	20	10

5. العمليات الرياضية في أنظمة العد

في الأنظمة العددية المختلفة (الثنائي، العشري، السادس عشري، والثماني)، يمكن تطبيق العمليات الحسابية مثل الجمع والطرح والضرب والقسمة. ولكن هناك بعض الفروقات في كيفية إجراء هذه العمليات مقارنةً بالنظام العشري (الأساس 10) المعتاد.

5.1. الجمع (Addition)

1) الجمع في النظام الثنائي:

يتم الجمع بنفس طريقة الجمع العشري، ولكن كل خانة يمكن أن تحتوي فقط على القيم 0 أو 1. في حال جمع 1+1 تكون النتيجة 0 مع الاحتفاظ بـ 1 (كما هو الحال في العشري عند جمع 9 + 1).

مثال:

(في العشري 13) 1101
+ (في العشري 11) 1011

(في العشري 24) 11000

2) الجمع في النظام السادس عشر:

يتم الجمع بنفس طريقة الجمع الثنائي، لكن إذا تجاوزت النتيجة F (15)، يتم الانتقال إلى الخانة الأعلى مع الاحتفاظ بـ 1.

مثال:

(في العشري 169) A9
+ (في العشري 47) 2F

(في العشري 216) D8

5.2. الطرح (Subtraction)

1) الطرح في النظام الثنائي:

يتم الطرح بنفس الطريقة، ولكن إذا لم يكن الرقم في الخانة الأولى كافيًا، يتم الاستلاف (borrow) من الخانة المجاورة.

مثال:

(في العشري 10) 1010
- (في العشري 7) 0111

(في العشري 3) 0011

2) الطرح في النظام السادس عشر:

يتم الطرح مع الاستلاف إذا لزم الأمر، مع الأخذ بعين الاعتبار أن الاستلاف يعني إضافة 16 (في حالة النظام السادس عشري).

مثال:

```
  3B  (في العشري 59)
+ 1C  (في العشري 28)
------------------
  1F  (في العشري 31)
```

5.3. الضرب (Multiplication)

1) الضرب في النظام الثنائي:

مثال:

```
   101  (في العشري 5)
x   11  (في العشري 3)
------------------
   101
+1010
------------------
  1111  (في العشري 15)
```

2) الضرب في النظام السادس عشر:

مثال:

```
    2A  (في العشري 42)
x   3F  (في العشري 63)
------------------
  1E9A
+  6E0
------------------
  71A0  (في العشري 2646)
```

5.4. القسمة (Division)

1) القسمة في النظام الثنائي:

القسمة تشبه القسمة الطويلة في النظام العشري، ولكن باستخدام 0 و 1 فقط.

مثال:

```
1 1 1 1 0 1 | 1 0 0 1
1 0 0 1     | 1 1 0
0 1 1 0
  1 1 0 0
  1 0 0 1
```

0 0 1 1
1 1 1

2) القسمة في النظام السادس عشر:

مثال:

2F4	B
22	33
0D	
D4	
B4	
20	

6. التحويل بين أنظمة العد

6.1. التحويل باستخدام طريقة كثير الحدود

طريقة كثير الحدود (Polynomial Method) هي إحدى الطرق الأساسية لتحويل عدد من نظام عد معين (مثل الثنائي أو السادس عشري) إلى النظام العشري. تعتمد هذه الطريقة على اعتبار كل خانة في العدد تمثل معاملًا في كثير حدود، بحيث يكون لكل خانة وزن معين بناءً على الأساس الخاص بالنظام العددي.

لتحويل العدد N من الأساس A إلى الأساس B نقوم أولا بتمثيل العدد كالتالي في الأساس A ثم نحسب العبارة الناتجة باستخدام العمليات الرياضية للأساس B.

$$N = a_{n-1} \times r^{n-1} + a_{n-1} \times r^{n-1} + \ \ldots + a_0 \times r^0 + a_{-1} \times r^{-1} + \ \ldots + a_{-m} \times r^{-m}$$

مثال:

$$(627)_8 = (7 \times 8^0 + 2 \times 8^1 + 6 \times 8^2)_{10} = \ (407)_{10}$$

6.2. التحويل باستخدام طريقة الضرب

طريقة تُستخدم طريقة الضرب لتحويل الجزء العشري من العدد من الأساس العشري إلى أي أساس آخر B.

لتحويل العدد N من الأساس العشري إلى الأساس B نتبع الخطوات التالية:

1. نضرب الجزء العشري للعدد العشري N_{10} في الأساس B.
   a. ينتج عن العملية جزء صحيح I يمثل الرقم الأكثر أهمية للعدد الجديد N_b.
   b. والباقي يكون جزءًا عشريًا جديدًا (الذي سيتم استخدامه في التكرار التالي).
2. الجزء العشري الجديد يصبح هو N_{10} في الخطوة التالية.
3. نكرر العملية حتى نصل إلى النتيجة المطلوبة (حسب الدقة المرغوبة).

مثال:

$$(0.479)_{10} = (0.3651...)_8$$

MSD 3.832 ← 0.479 x 8
6.656 ← 0.832 x 8
5.248 ← 0.656 x 8
LSD 1.984 ← 0.248 x 8

6.3. التحويل باستخدام طريقة القسمة المتتالية

طريقة القسمة المتكررة تُستخدم لتحويل الأعداد من الأساس العشري (الأساس 10) إلى أي أساس آخر B.

للتحويل من الأساس A إلى الأساس B:

1) نقوم بقسمة العدد $(N)_A$ على $(B)_A$, ينتج حاصل Q و باقي R يمثل الرقم الأقل أهمية للعدد $(N)_B$
2) الحاصل Q يصبح هو $(N)_A$.
3) نكرر العملية إلى أن يصبح الحاصل $Q = 0$

مثال:

$$(315)_{10} = (473)_8$$

8 | 315	3	LSD
8 | 39	7	
8 | 4	4	MSD
0		

6.4. التحويل من النظام الثنائي إلى الثماني

1) نقّسم سلسلة الأرقام إلى مجموعات تتكون كل منها من **ثلاث** أرقام ابتداًء من اليمين.
2) نحول كل مجموعة أرقام من النظام الثنائي إلى الثماني.

مثال:

$$(1011110111001)_2 = (13671)_8$$

6.5. التحويل من النظام الثنائي إلى السادس عشر

1) نقّسم سلسلة الأرقام إلى مجموعات تتكون كل منها من **أربعة** أرقام ابتداًء من اليمين.
2) نحول كل مجموعة أرقام من النظام الثنائي إلى السادس عشر.

مثال:

$$(1011110111001)_2 = (17B9)_{16}$$

6.6. طريقة التحويل العامة من الأساس A إلى الأساس B

لتحويل عدد من الأساس A إلى الأساس B نتبع هذه الخطوات:

للتحويل من الأساس A إلى الأساس B:

1) نقوم تحويل العدد $(N)_A$ إلى $(N)_{10}$, باستخدام طريقة كثير الحدود.

2) نقوم تحويل العدد $(N)_{10}$ إلى $(N)_B$, باستخدام طريقة القسمة المتتالية.

هذه الطريقة أطول، ولكنها أسهل وأقل عرضة للخطأ.

مثال:

نريد تحويل $(18.6)_9$ إلى الأساس base-11:

1) التحويل إلى الأساس العشري:

$$\begin{aligned} N_{10} &= 1 \times 9^1 + 8 \times 9^0 + 6 \times 9^{-1} \\ &= 9 + 8 + 0.666\ldots \\ &= (17.666\ldots)_{10} \end{aligned}$$

2) التحويل إلى الأساس 11:

$$\begin{array}{r|l} 17 & 6 \\ 1 & 1 \\ 0 & \end{array}$$

$$\begin{aligned} 7.326 &\leftarrow 0.666 \times 11 \\ 3.586 &\leftarrow 0.326 \times 11 \\ 6.446 &\leftarrow 0.586 \times 11 \end{aligned}$$

$$N_{11} = (16.736\ \ldots)_{11}$$

7. تمارين

قم بإجراء التحويلات التالية مع توضيح الطريقة:

$(178)_{10} = (\quad)_2$	$(1087.55)_{10} = (\quad)_{16}$
$(98)_{10} = (\quad)_8$	$(31.8)_{10} = (\quad)_2$
$(14.625)_{10} = (\quad)_8$	$(14.625)_{10} = (\quad)_2$

8. خاتمة

تناول هذا الفصل الأساسيات المتعلقة بأنظمة العد المختلفة التي تُستخدم في الحوسبة الرقمية، مع التركيز على أهمية التحويل بين هذه الأنظمة في سياق عمل الأجهزة الحاسوبية. تم تقديم شرح مفصل للأنظمة الأساسية، مثل النظام الثنائي، الثماني، والسداسي عشري، إلى جانب العمليات الحسابية

الأساسية داخل كل نظام. إن الإلمام بهذه المفاهيم يُعدّ خطوة حيوية نحو استيعاب أعمق لتصميم الأنظمة الرقمية وفهم كيفية عمل الخوارزميات والبرمجيات على مستوى منخفض. بهذا، نكون قد وضعنا الأساس اللازم للتعمق في مواضيع البرمجة والخوارزميات التي ستُناقش في الفصول القادمة.

الفصل 3: الخوارزميات (Algorithms)

1. مقدمة

تعد الخوارزميات العمود الفقري للعديد من التطبيقات الحاسوبية، إذ تساهم في تنظيم وحل المشكلات بطريقة منطقية وفعالة. تأتي أهمية دراسة الخوارزميات من كونها لا ترتبط فقط بلغة برمجة معينة، بل تشكل إطارًا عامًا يمكن استخدامه في مختلف بيئات البرمجة والأجهزة. يعزز هذا الفصل فهم الطلبة لكيفية تصميم وكتابة الخوارزميات لضمان تقديم حلول مبتكرة لمشكلات متنوعة.

يتناول هذا الفصل أيضًا كيفية تمثيل الخوارزميات، تعريف المتغيرات، وتنظيم التدفق عبر استخدام التعليمات الشرطية والحلقات التكرارية. من خلال أمثلة عملية مثل خوارزمية إيجاد القاسم المشترك الأكبر (GCD)، يوفر الفصل أدوات لفهم وتطبيق المبادئ الأساسية التي تحكم الخوارزميات، مما يمهد الطريق لتطوير مهارات برمجية متقدمة.

2. لماذا نحتاج للخوارزميات؟

تتسم بيئة البرمجة بوجود العديد من لغات البرمجة التي تختلف عن بعضها البعض من حيث النحو والقواعد اللغوية التي تحدد كيفية كتابة التعليمات. هذا التعدد يفتح المجال للمرونة والابتكار، لكنه أيضًا يخلق تحديات تتمثل في صعوبة التفكير والتطوير خارج الإطار التقليدي لكل لغة.

غالبًا ما تكون لغة البرمجة مرتبطة بنوع معين من الأجهزة أو أنظمة التشغيل، مما يُقيّد المبرمجين ويحد من قدرة التطبيقات على العمل بسلاسة عبر منصات مختلفة. هذا التقييد يؤدي إلى مشاكل تتعلق بتوافق الأجهزة، حيث يصبح من الضروري توفير حلول قادرة على تجاوز هذه القيود.

لذلك، تظهر الحاجة إلى لغة موحدة أو معايير قياسية (Standardization)، تهدف إلى تسهيل عملية حل المشاكل بعيدا عن قيود لغات البرمجة والأجهزة.

3. ما هي الخوارزمية؟

الخوارزمية هي مجموعة من التعليمات المرتبة والمتسلسلة التي تهدف إلى حل مشكلة معينة أو تنفيذ مهمة ما بطريقة واضحة ومنظمة. يُمكن اعتبار الخوارزمية مخططًا أو وصفًا تفصيليًا للخطوات التي يجب اتباعها لتحقيق هدف محدد. يجب أن تكون الخوارزمية:

1. منظمة وخالية من الغموض (Without Ambiguity).
2. دقيقة (Precise).
3. فعالة (Efficient).
4. تحل المشكل المطروح في وقت محدد.

مثال: خوارزمية أقليدس (Euclidean algorithm) لايجاد القاسم المشترك الأكبر (GCD).

1) نحفظ في m و n القيمة الأكبر والأصغر من القيم المدخلة
2) نقسم m على n و نحفظ الباقي في r
3) إذا لم تكن $r = 0$ ، نقوم بتخزين n في m ثم نقوم بتخزين r في n ، ثم نعود إلى الخطوة 2
4) إذا كانت $r = 0$ ، فإن القاسم المشترك الأكبر هو r

	m	n	r
1	369	27	
	369 = 27 x 13 + 18		
2~3	27 = 18 x 1 + 9		
	18 = 9 x 2 + 0		
4		9 ➔ GCD	

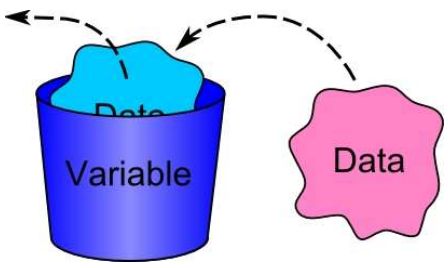
4. كيفية كتابة خوارزمية

تتبع كتابة الخوارزمية هيكلًا واضحًا ومنظمًا لضمان سهولة الفهم والتنفيذ. الخوارزميات يتم تمثيلها عادة باستخدام خطوات بسيطة ومرتبة توضح سير العمليات. فيما يلي النموذج العام لكتابة خوارزمية:

```
algorithm Name_Of_Algorithm;
// تعريف المتغيرات، الأنواع، والثوابت
// تعريف الإجراءات والدوال
begin
    // تعليمات الخوارزمية
end.
```

5. المتغيرات (Variables)

المتغير هو وحدة في الذاكرة تُستخدم لتخزين المعلومات والبيانات أثناء تنفيذ البرامج. يُعتبر المتغير وسيلة مرنة تُتيح للمبرمجين تخزين قيم مختلفة والاحتفاظ بها طوال فترة عمل البرنامج. الشكل 9 يحاكي دور المتغير في حفظ البيانات.



الشكل 9. تمثيل دور المتغير في الخوارزمية.

5.1. خصائص المتغيرات

1. المتغير كخانة ذاكرة:

يُشير المتغير إلى موقع محدد في الذاكرة يُخزن فيه قيمة أو أكثر. هذه القيمة قد تتغير خلال تنفيذ البرنامج.

2. اسم المتغير:

المتغير هو اسم يُطلق على خانة أو أكثر في الذاكرة، حيث يمكن استخدامه لاستدعاء القيم المخزنة أو تعديلها.

3. أنواع المتغيرات:

يتم تحديد نوع البيانات التي يمكن تخزينها في المتغير، مثل:

1) **Integer** : لتخزين الأعداد الصحيحة.
2) **Float** : لتخزين الأعداد العشرية.
3) **Character** : لتخزين حرف واحد.
4) **String** : لتخزين سلسلة من الحروف.
5) **Boolean**: لتخزين القيم المنطقية (True/False).

5.2. التصريح المتغيرات (Variable declaration)

التصريح بالمتغيرات هو عملية يتم من خلالها الإعلان عن متغير في الخوارزمية وإعطاؤه اسمًا ونوع بيانات لتحديد نوع القيم التي يمكن تخزينها فيه. هذه الخطوة أساسية لضمان تنظيم البيانات داخل البرنامج وتخصيص مساحة في الذاكرة للمتغيرات التي سيتم استخدامها لاحقًا.

1. التصريح بمتغير واحد (Single variable declaration):

يتم تعريف متغير واحد بتحديد اسمه ونوع البيانات التي يمكن تخزينها فيه كالتالي:

```
var name : variable_type;
```

بحيث:

var :كلمة مفتاحية للإعلان عن متغير.

name :اسم المتغير.

variable_type : نوع البيانات (مثل: integer, string, character, إلخ).

2. التصريح بعدة متغيرات من نفس النوع (Multiple variables declaration):

يمكن الإعلان عن عدة متغيرات بنفس النوع في سطر واحد، لتوفير الوقت وتقليل التكرار كالتالي:

```
var name1, name2, name3: variable_type;
```

3. التصريح بمتغيرات بأنواع مختلفة:

يمكن الإعلان عن عدة متغيرات بأنواع مختلفة كالتالي:

```
var x, y: integer;
    s: string;
    c: character;
    b: boolean;
```

6. الثوابت (Constants)

الثابت (constant) يبقى دون تغيير أثناء تنفيذ البرنامج, و يتم التصريح بالثوابت باستعمال الكلمة المحجوزة **const** كالتالي:

```
const identifier = constant_value;
```

مثال:

```
const pi = 3.14;
      MAX = 99999;
      MSG = "Hello";
```

7. القيم المنطقية (Booleans)

في البرمجة عموما والمنطق الرقمي، القيم المنطقية تمثل حالتين محتملتين هما صحيح (True) أو خاطئ (False). يتم استخدام ثلاثة عوامل منطقية (Logical Operators) شائعة لإجراء عمليات منطقية على القيم المنطقية وهي:

and : تكون النتيجة true إذا كانت جميع القيم المدخلة true.

or : تكون النتيجة true إذا كانت واحدة على الأقل من القيم المدخلة true.

not : يعكس القيمة، أي إذا كانت true تصبح false، والعكس صحيح.

الجدول 3 يمثل جدول الحقيقة (Truth Table) لأكثر العوامل المنطقية استعمالا.

جدول 3. جدول الحقيقة لأكثر العوامل المنطقية استعمالا.

B1	B2	B1 and B2	B1 or B2	not B1	not B2
false	false	false	false	true	true
false	true	false	true	true	false
true	false	false	true	false	true
true	true	true	true	false	false

8. العمليات الرياضية (Arithmetic Operators)

تُعد العمليات الرياضية جزءًا أساسيًا في تصميم وتنفيذ الخوارزميات، حيث تُستخدم لإجراء الحسابات والمعالجات على البيانات. تشمل هذه العمليات الجمع (+) لإضافة القيم معًا، والطرح (-) لحساب الفرق بين الأرقام، والضرب (*) لإيجاد حاصل ضرب عددين، والقسمة (/) لتقسيم القيم. كما تُستخدم عملية باقي القسمة (%) لاستخراج الباقي عند قسمة عددين صحيحين، وهي مفيدة في العديد من التطبيقات مثل التحقق من الأرقام الفردية والزوجية أو تحديد دورات زمنية متكررة. تلعب هذه العمليات دورًا حيويًا في الخوارزميات لأنها تتيح معالجة البيانات بفعالية وتساعد في تطوير حلول للمشكلات الرياضية والمنطقية.

9. التعليمات الأساسية (Basic Instructions)

التعليمات الأساسية في البرمجة عموما والخوارزميات تمثل اللبنات الأولى التي تُستخدم لتنفيذ الأوامر خطوة بخطوة لتحقيق الأهداف المطلوبة. تُعد هذه التعليمات أساسًا لكل برنامج أو خوارزمية، وتساعد في التحكم في تدفق التنفيذ ومعالجة البيانات بشكل صحيح. فيما يلي أهم التعليمات الأساسية المستخدمة:

9.1. الإسناد (Assignment)

هي تعليمة تسمح بإسناد قيمة (value) أو عبارة (expression) إلى متغير ما. يرمز لها بـ ← :

```
variable_name ← value_or_expr;
```

الجانب الأيسر لا بد أن يكون متغير واحد فقط.

مثال:

- بعض عمليات الإسناد الصحيحة والخاطئة:

```
✓ a ← 3.18;
✓ b ← a + 5.14;
✓ a ← 10;
✓ s ← "Hello world";
✗ a + b ← 4;
✗ 5 ← a ;
```

- خوارزمية توضح كيفية التصريح عن متغيرات مختلفة النوع وبعض عمليات الإسناد:

```
algorithm example;
var x, y: integer;
    s: string;
    result: real;
begin
 x ← 9;
 y ← 2;
 result ← (y * y) + (x / y);
 s ← “Result = “;
end.
```

9.2. الإدخال/الإخراج (Input/Output)

تعليمات خاصة تسمح بتواصل المستخدم مع الكمبيوتر، حيث تتيح تعليمات الإدخال للكومبيوتر الحصول على بيانات من المستخدم أو مصدر خارجي. أما تعليمات الإخراج فتُستخدم لعرض النتائج للمستخدم. التعليمتين الأساسيتين للإدخال/الإخراج هما:

1- **read**: تخزين البيانات المدخلة على لوحة المفاتيح في متغيرات.

2- **write**: عرض نص على الشاشة.

مثال:

1- خوارزمية تطلب من المستخدم إدخال قيمتين a و b ثم تقوم بإسناد نتيجة عملية الجمع a+b في المتغير c، ثم تقوم بعرض النتيجة على الشاشة.

```
algorithm sum;
var a, b, c: integer;
begin
 write(“a = ");
 read(a);
 write("b = ");
 read(b);
 c ← a + b;
 write(“The sum a + b = ", C);
end.
```

2- خوارزمية تطلب من المستخدم إدخال قطر الدائرة r ثم تقوم بحساب محيطها وعرض النتيجة على الشاشة.

```
algorithm calc_circle_perimeter;
var r, perimeter: real;
const PI = 3.1415;
begin
 write("The radius of the circle r = ");
 read(r);
 perimeter ← r * PI;
 write("Perimeter of the circle = ", perimeter);
end.
```

9.3. التعابير المنطقية (Logical Expressions)

في البرمجة عموما، التعبير المنطقي هو تعبير ينتج قيمة منطقية عند تقييمه أو حسابه أو تنفيذه (evaluated). بحيث تكون القيمة المنطقية الناتجة تكون إما true (صحيح) أو false (خطأ). تقييم التعبير المنطقي يشبه طرح سؤال إجابته "نعم" أو "لا". على سبيل المثال: التعبير $x > 2$ يعني "هل x أكبر من 2؟" إذا كانت قيمة x أكبر من 2، فإن النتيجة تكون true. وإذا لم تكن x أكبر من 2، فإن النتيجة تكون false.

تستخدم التعابير المنطقية على نطاق واسع في البرمجة لاتخاذ قرارات، مثل التحقق من الشروط داخل الجمل الشرطية (Conditional statements)، أو داخل الحلقات التكرارية (while أو for).

مثال:

الجدول 4 به أمثلة لبعض لتقييم بعض التعابير المنطقية حول المتغير x الذي قيمته الحالية 8.

جدول 4. أمثلة لبعض لتقييم بعض التعابير المنطقية حول المتغير x الذي قيمته الحالية 8.

Operator	الوصف	مثال مع: x ← 8	
$=$	يساوي	النتيجة خطأ (false)	$x = 7$
$\neq$	لا يساوي	النتيجة صحيح (true)	$x \neq 6$
$>$	أكبر من	النتيجة خطأ (false)	$x > 8$
$<$	أصغر من	النتيجة صحيح (true)	$x < 10$
>=	أكبر من أو يساوي	النتيجة صحيح (true)	$x \geq 8$
<=	أصغر من أو يساوي	النتيجة صحيح (true)	$x \leq 10$
$!\,(expr)$	نفي العبارة بين قوسين	النتيجة خطأ (false)	$!\,(x = 8)$

9.4. الجمل الشرطية (Conditional Statements)

التعليمة الشرطية «إذا كان» (if) تسمح بتنفيذ قسم من الخوارزمية إذا تحقق شرط ما. فيما يلي تركيب التعليمة الشرطية if بنوعيها:

```
if <condition> then
  تعليمات//
```

```
if <condition_1> then
  تعليمات//
else if <condition_2> then
  تعليمات//
else
  تعليمات//
endif
```

إذا كان الشرط (condition) صحيحًا (true)، يتم تنفيذ مجموعة التعليمات (statements) التي تلي then. في حالة عدم تحقق أي من الشروط السابقة، يتم تنفيذ مجموعة التعليمات داخل قسم else.

مثال:

```
if x > 10 then
    write("x 10 أكبر من")
else if x == 10 then
    write ("x 10 يساوي");
else
    write ("x 10 أصغر من");
```

إذا كنا نريد تنفيذ أكثر من تعليمة في جملة شرطية نستخدم begin و end; كالتالي:

```
if x > 10 then
begin
    write ("x 10 أكبر من");
    write ("x = ", x);
end;
```

9.5. الحلقة التكرارية for...do

تُستخدم في البرمجة للتحكم في تنفيذ مجموعة من التعليمات عددًا محددًا من المرات. فيما يلي البنية العامة لحلقة تكرارية for..do:

```
for <var> ← <inital_value> to [down to] <final_value> do
   //statements
```

ف البداية يتم إسناد قيمة ابتدائية للمتغير var، ثم يتم تكرار التعليمات مع زيادة قيمة المتغير var بواحد (أو إنقاص قيمته بواحد في حالة استخدام down to) إلى غاية أن تصل قيمته إلى final_value.

مثال:

هذا الكود يسمح بكتابة الأعداد من 1 إلى 10 مع العودة للسطر بعد كل عدد (الحرف الخاص '\n' يعيد مؤشر الكتابة إلى سطر جديد على الشاشة):

```
for i ← 1 to 10 do
begin
  write(i);
  write('\n');
end;
```

```
1
2
.
.
10
```

9.6. الحلقة التكرارية while..do

تسمح حلقة while..do بإجراء عمليات حسابية متكررة حتى يتم استيفاء بعض شروط. بمعنى آخر، تنفيذ -بشكل متكرر- التعليمات في جسم الحلقة طالما أن الشرط المحدد صحيح. الهيكل العام لهذه الحلقة التكرارية كالتالي:

```
While <condition> do
   //statements
```

مثال:

هذا الكود يسمح بكتابة الأعداد الزوجية بين 1 و10 مع العودة إلى السطر بعد كل عدد:

```
for i ← 1 to 10 do
begin
  write(i);
  write('\n');
end;
```

```
1
2
.
.
10
```

9.7. تعليمات التحكم في الحلقات

تعليمات التحكم في الحلقات (Loop control statements) تتيح التحكم في تكرار الحلقة وسيرها خارج التسلسل العادي.

break : تنهي تنفيذ الحلقة مباشرة و ينتقل التنفيذ إلى أول تعليمة بعد الحلقة.

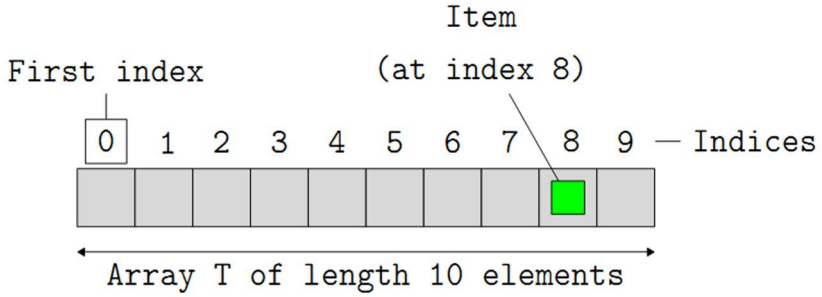
continue : تلغي تنفيذ بقية التعليمات في الدورة الحالية للحلقة و تنتقل إلى الدورة الموالية.

10. المصفوفات (Arrays)

المصفوفة هي هيكل بيانات خطي يخزن مجموعة من القيم من نفس النوع في أماكن متجاورة في الذاكرة. يمكن الوصول إلى كل عنصر في المصفوفة باستخدام مؤشر (index). يمكن التفكير في المصفوفة على أنها مجموعة من المتغيرات من نفس النوع. بدلاً من التصريح بمتغيرات فردية مثل num0 و num1

و... num99، يمكنك الإعلان عن متغير مصفوفة واحد يسمى num ثم نستخدم num[0] ، num[1] ، ... للوصول لمختلف المتغيرات.

تقوم الجداول بتمثيل هياكل بيانات أخرى مثل القوائم (list) والأكوام (heap) وقوائم الانتظار (queue) والمكدسات (stack) والسلاسل (string). الشكل 10 يوضح بنية مصفوفة (أو جدول) ذات 10 عناصر.



الشكل 10. مثال لبنية مصفوفة (أو جدول) ذات 10 عناصر.

يكون الشكل العام للتصريح عن نوع مصفوفة ما كالتالي:

```
type array_type = array[size] of element_type;
```

بحيث: size هو حجم المصفوفة أو عدد عناصرها، أما element_type فهو نوع عناصر المصفوفة.

الوصول لعناصر المصفوفة (للقراءة أو الكتابة) يكون باستخدام الأقواس المربعة [index] بحيث يمثل index مؤشر أو رقم العنصر ابتداءا من 0 كمؤشر لأول عنصر.

مثال:

في هذا المثال سنقوم بتعريف النوع vector و هو عبارة عن مصفوفة بعنصرين حقيقين لتمثيل شعاع [x,y], و نوع ثاني اسمه grades و هو عبارة عن مصفوفة ذات 10 عناصر لحفظ 10 علامات امتحان لطالب. بعد ذلك نقوم بالتصريح عن متغيرين v1 و v2 من نوع vector و متغير من نوع grades اسمه student_grades.

```
type vector = array[2] of real;
     grades = array[10] of real;
var v1, v2: vector;
    student_grades: grades;
```

في هذا الخوارزمية، سنقوم بتعريف نوع جديد اسمه tab يمثل مصفوفة ذات 5 عناصر صحيحة, ثم نصرح عن المتغير t من نوع tab. بعدها نسند لمصفوفة t العناصر (2,3,6,1,0), ثم نغير قيم قيمة أول

عنصر إلى 10. أخيرا نسند مجموع العناصر الأربعة الأولى إلى آخر عنصر في المصفوفة. ليصبح محتوى المصفوفة t في الأخير (2,10,6,1,19).

```
algorithm array_example;
type tab = array[5] of integer;
var t : tab;
begin
 t ← (2, 3, 6, 1, 0);
 t[1] ← 10;
 t[4] ← t[0] + t[1] + t[2] + t[3];
end.
```

التصريح عن المصفوفات متعددة الأبعاد (Multi-dimensional Array) يكون مشابه لما سبق مع الاختلاف في أبعاد المصفوفة:

```
type array_type = array[size1][size2]… of element_type;
```

الوصول لعناصر المصفوفة يكون باستخدام الأقواس المربعة لكل بعد من أبعاد المصفوفة، مثلا للوصول للعنصر الموجود في السطر الثالث (index = 2) وفي العمود الخامس (index = 4) في مصفوفة ثنائية الأبعاد نستخدم:

```
t[2][4]
```

مثال:

في هذا المثال سنقوم بتعريف النوع Matrix وهو عبارة عن مصفوفة ثنائية الأبعاد (2D)، بعدها نقوم بالتصريح عن M متغير من نوع مصفوفة 2D و نثوم بإسناد قيم ابتدائية لعناصر المصفوفة M. ثم نقوم بإسناد القيمة 30 إلى العنصر الموجود في العمود السطر الثاني و في العمود الأول.

```
type Matrix= array[3][3] of real;
var M: Matrix;
begin
M ← ((1,2,3), (4,5,6), (7,8,9));
M[1][0] ← 30;
end.
```

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

مثال 2:

في الخوارزمية التالية نعرف النوع Tab (مصفوفة ذات بعد واحد من 5 عناصر صحيحة) ثم نصرح عن المتغير T من نوع Tab. و من خلال حلقة تكرارية for..do نسند مربعات الأعداد من 1 إلى 5 كعناصر للمصفوفة T. و في حلقة تكرارية ثانية for..do نعرض على الشاشة محتوى المصفوفة T.

```
algorithm exercice;
type Tab = array [5] of integer;
var T : Tab;
    i: integer;
begin
 for i ← 1 to 5 do
    T[i] ← i*i;
 for i ← 1 to 5 do
    write(T[i]);
end.
```

11. أنواع البيانات المخصصة (Custom Data Types)

هي أنواع بيانات يمكن تعريفها من قبل المستخدم لتناسب احتياجات خاصة في البرنامج. الغرض من هذه الأنواع هو تجميع مجموعة من المتغيرات المختلفة تحت اسم واحد بحيث يمكن التعامل معها كوحدة واحدة في الذاكرة. يُعد هذا النهج مفيدًا لتبسيط إدارة البيانات وتنظيمها، لا سيما عند الحاجة إلى تخزين أنواع متعددة من القيم ذات الصلة معًا. أحد الأمثلة على أنواع البيانات المخصصة هو السجل (Record)، الذي يُستخدم لتخزين مجموعة من الحقول (Fields)، حيث يمكن لكل حقل أن يكون له نوع بيانات مختلف. يُعرف السجل عادةً باسم معين، ويضم عدة متغيرات (حقول)، بحيث يتم تمثيل كل حقل داخل السجل بنوع بيانات خاص به. يكون الهيكل الأساسي لتعريف السجل كالتالي:

```
type <record_name> = record
    <field_1>: <field_type1>;
    <field_2>: <field_type2>;
    ...
    <field_n>: <field_typen>;
end;
```

حيث:

type <record_name> : يتم تحديد اسم السجل (نوع البيانات) باستخدام هذه الكلمة المفتاحية.

record : تشير إلى أن هذا النوع هو سجل يحتوي على مجموعة من الحقول.

<field_1>: <field_type1> : يمثل كل سطر حقلًا في السجل، حيث يتم تعريف اسم الحقل ونوع بياناته. على سبيل المثال، يمكن أن يكون الحقل الأول هو الاسم ونوعه string، والحقل الثاني هو العمر ونوعه integer.

end : تُستخدم لإنهاء تعريف السجل.

مثال:

نفترض أننا نرغب في إنشاء سجل يمثل طالب يحتوي على الحقول التالية: الاسم (نوع string)، العمر (نوع integer)، والمعدل (نوع real). يكون تعريف السجل على النحو التالي:

```
type Student = record
    Name: string;
    Age: integer;
    Grade: real;
end;
```

بعد تعريف السجل، يمكننا إنشاء متغير من نوع Student وتخزين بيانات طالب معين فيه كالتالي:

```
var student1: Student;
begin
    student1.Name := "أحمد";
    student1.Age := 21;
    student1.GPA := 3.5;
end;
```

12. البرامج الجزئية أو الفرعية (Subprograms)

البرامج الفرعية هي وحدات برمجية صغيرة تُستخدم لتنفيذ مهام محددة داخل البرامج الأكبر. يتم تجميع هذه البرامج الفرعية ضمن البرامج الرئيسية لتحقيق تنظيم أفضل وتقليل التكرار، بالإضافة إلى تسهيل الصيانة والتطوير المستمر. يمكن استدعاء البرنامج الفرعي داخل البرنامج الرئيسي أو بواسطة برنامج فرعي آخر. من أهم أنواع البرامج الفرعية:

12.1. الوظائف (Functions)

تقوم بإرجاع قيمة واحدة (كنتيجة) عند استدعائها (تقوم بإرجاع أكثر من قيمة في بعض لغات البرمجة مثل بايثون) وتُستخدم غالبًا عندما يكون الهدف هو حساب قيمة معينة، مثل حساب مجموع أو إرجاع طول سلسلة نصية. يكون الهيكل العام للوظيفة كالتالي:

```
function func1(<param1>: type1; <param2>: type2; ...): return_type;
local <declarations>;
begin
    //statements
    func1 ← expression;
end;
```

بحيث:

- الكلمة المفتاحية function لتعريف وظيفة جديدة اسمها func1 وهو الاسم الذي سيتم استدعاؤها من خلاله في باقي البرنامج.
- بين الأقواس يتم تمرير المدخلات أو المعاملات (Parameters) التي تحتاجها الوظيفة لتنفيذ مهامها، مع تحديد أنواع البيانات لكل معامل.
- return_type تحدد نوع القيمة المرجعة من الوظيفة. إذا كانت الوظيفة تقوم بإرجاع قيمة (مثل عدد أو سلسلة نصية)، يتم تحديد نوع القيمة هنا.
- يتم هنا تصريح عن المتغيرات المحلية باستخدام الكلمة المفتاحية local، هذه المتغيرات تكون مرئية فقط ضمن نطاق الوظيفة ولا يمكن الوصول إليها من خارج جسم الوظيفة الذي تحدده begin و end.
- آخر تعليمة في جسم الوظيفة هي: func1 ← expression، و من خلالها يتم إرجاع النتيجة عن طريق اسناد النتيجة إلى اسم الوظيفة، مما يعني أن القيمة التي سيتم إرجاعها عند استدعاء الوظيفة هي تلك النتيجة.

مثال:

الوظيفة Sum تقوم بإرجاع مجموع الرقمين الصحيحين المدخلين.

```
function Sum(a: integer; b: integer): integer;
begin
    Sum := a + b;
end;
```

أما استدعاء الوظيفة Sum فيكون بإحدى الطرق التالية على حسب الحاجة:

```
c ← Sum(a, b);
c ← Sum(10, 12);
write(Sum(63, 14));
```

12.2. الإجراءات (Procedures)

عل عكس الوظائف، لا تقوم الإجراءات بإرجاع قيمة مباشرة. فهي تُستخدم لتنفيذ مهمة معينة مثل طباعة نص أو تحديث قاعدة بيانات أو أي مهمة لا تتطلب إعادة نتائج. يكون الهيكل العام للإجراء كالتالي:

```
procedure proc1(<param1>: type1; <param2>: type2; ...);
local <declarations>;
begin
    //statements
end;
```

مثال:

الإجراء PrintMessage يقوم بطباعة النص المُمرر له ثم العودة لسطر جديد.

```
procedure PrintMessage(msg: string);
begin
    write(msg);
    write('\n');
end;
```

أما استدعاء الإجراء PrintMessage فيكون كالتالي:

```
PrintMessage("Hello World !");
PrintMessage(str);
```

12.3. البرامج الفرعية التراجعية (Recursive Subprograms)

البرامج الفرعية التراجعية أو التكرارية هي نوع من الوظائف أو الإجراءات التي يمكن أن تستدعي نفسها بشكل متكرر خلال التنفيذ. يتم استخدام هذا النوع من الوظائف عندما تكون المشكلة التي يتم حلها تحتوي على نفس النمط في أجزاءها الفرعية. يُطلق على هذا الاستدعاء المتكرر اسم recursive call، حيث تقوم الدالة باستدعاء نفسها مباشرة أو ضمن سلسلة من الاستدعاءات حتى يتم الوصول إلى حالة الإيقاف (base case)، وهي الحالة التي لا تتطلب استدعاءً آخر.

مثال: حساب المضروب (Factorial)

المضروب، الذي يرمز له بـ $n!$، يُعرّف رياضيًا على أنه حاصل ضرب جميع الأعداد الصحيحة الموجبة حتى العدد n. يمكن التعبير عنه بشكل تكراري كما يلي:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1 = n \times (n-1)!$$

$$(n-1)! = (n-1) \times (n-2)!$$

وهكذا إلى غاية:

$$1! = 1 \times 0!$$

مع الأخذ بعين الاعتبار حالة الإيقاف (base case) التالية:

$$0! = 1$$

بذلك تكون الوظيفة fact التي تحسب مضروب n كالتالي:

```
function fact(n: integer): integer;
begin
 if n = 0 then
  fact ← 1
 else
  fact ← n * fact(n-1);
end;
```

13. تمارين

1) حول الجمل التالية إلى تعبيرات منطقية:
   - الوزن w أقل من 80
   - x يختلف عن 24
   - x و y ليسا متماثلين
   - x بين 0 و 20
2) ما التعبير المنطقي الذي يحدد ما إذا كانت x و y أكبر من z؟

$(x > z)\ and\ (y > z)$	$(x\ and\ y > z)$
$(y > z)\ and\ (x > z)$	$(z < x)\ and\ (z < y)$

3) ما هي نتائج التعابير المنطقية التالية:
   - $(2 < 5)\ and\ (5 = 4 + 1)$
   - $(9 = 6 * 2)\ and\ (11 < 16)$
   - $(9 = 6 * 2)\ or\ (11 < 16)$
   - $(9 = 6 * 2)\ or\ (11 > 16)$
   - $!(3 < 8)$
   - $!(3 < 8 - 7)$

4) أكتب خوارزمية تقوم بـ:
   - إيجاد القاسم المشترك الأكبر GCD
   - بحساب !n
   - تحسب مجموع الأعداد من 1 إلى n
   - إيجاد قواسم عدد n
5) أكتب خوارزمية تقوم بطلب عدد صحيح من المستعمل ثم تعرض الـ 10 أعداد الموالية له.
   مثال: الرقم المدخل 12, الخوارزمية تعرض: 13 14 15 16 17 18 19 20
6) أكتب خوارزمية تطلب من المستخدم إدخال عدد n ثم تعرض جدول الضرب، مثلا n = 7 بهذه الطريقة:

$$7 \times 1 = 7$$
$$7 \times 2 = 14$$
$$7 \times 3 = 21$$
$$\dots$$

7) أكتب خوارزمية تقوم بحساب:
   - القاسم المشترك الأكبر (GCD) لعددين.
   - المضاعف المشترك الأصغر (LCM) لعددين.
8) أكتب خوارزمية تقوم بحساب مجموع أرقام عدد معين. مثلا من أجل 54632 تكون النتيجة 5+4+6+3+2 = 20
9) نسمي عدد طبيعي كامل كل عدد n بحيث يكون مساوي لمجموع قواسمه. مثلا 6 = 1 + 2 + 3
   أكتب خوارزمية تقوم بالتحقق إذا كان عدد مدخل n كاملا أو لا.
10) أكتب خوارزمية تقوم بما يلي:
   1. تطلب من المستخدم ادخال عدد n مكون من 3 أرقام. مثلا $n = 853$
   2. تحسب العدد n_r معكوس n (في هذا المثال $n_r = 358$)
   3. تحسب d القيمة المطلقة لـ $n_r n -$ ($d = 495$)
   4. تحسب d_r معكوس d ($d_r = 594$)
   5. تحسب المجموع d_r+d
   6. تتحقق أن المجموع d_r+d دائما مساوي لـ 1089
11) أكتب خوارزمية تسمح بالتحقق هل عدد n معطى أولي أو لا.

غير الخوارزمية السابقة لتعرض الأعداد الأولية الأقل من 100 (استخدم الوظائف functions)

14. خاتمة

في هذا الفصل، تعرفنا على المفاهيم الأساسية المتعلقة بالخوارزميات، بدءًا من تعريفها ووظائفها، مرورًا بكيفية كتابتها وتحليلها، وصولًا إلى تطبيقات عملية تتعلق بإنشاء حلول برمجية فعالة. تشكل الخوارزميات حجر الزاوية في تصميم الأنظمة البرمجية، حيث تلعب دورًا كبيرًا في تحسين أداء البرامج وتبسيط عملية التطوير. كما أن القدرة على كتابة خوارزميات واضحة وفعالة تسهم في بناء أنظمة أكثر استقرارًا وقدرة على التعامل مع تحديات متنوعة.

يعد هذا الفصل تمهيدًا ضروريًا لفهم أعمق لعالم البرمجة، حيث تنتقل الدراسة لاحقًا إلى لغات البرمجة وتطبيق الخوارزميات. إن تعلم كيفية تحليل واختيار الخوارزمية الأنسب لكل مشكلة يعزز من مهارات الطلبة ويجعلهم قادرين على مواجهة المشكلات التقنية بكفاءة.

الفصل 4: لغة البرمجة بايثون (Python)

1. مقدمة

تُعتبر لغة البرمجة بايثون (Python) من أكثر اللغات البرمجية شهرة وانتشارًا في العالم، بفضل سهولتها ومرونتها. تعد بايثون لغة ذات أغراض عامة وتُستخدم في العديد من المجالات مثل تطوير الويب، تحليل البيانات، الذكاء الاصطناعي، والتعلم الآلي. ما يميز بايثون هو دعمها لعدة أنماط برمجية مثل البرمجة الكائنية والوظيفية، مما يجعلها اختيارًا مثاليًا للمبتدئين والمحترفين على حد سواء. يهدف هذا الكتاب إلى تقديم مقدمة شاملة عن بايثون، بدءًا من كيفية إعداد بيئة العمل والتثبيت، وصولاً إلى كتابة البرامج والتعرف على أساسيات اللغة. خلال الفصول القادمة، سيتم تسليط الضوء على أهم الميزات والوظائف التي تقدمها بايثون مع أمثلة عملية تساعد في فهم أفضل.

2. ما هي لغة البرمجة بايثون (Python)؟

تُعتبر بايثون من اللغات البرمجية ذات الأغراض العامة، حيث تمتاز بأنها تفاعلية وسهلة التعلم وتتميز بأسلوب كتابة بسيط وواضح. تُستخدم هذه اللغة في مجالات متعددة نظرًا لقدرتها على تقديم أداء عالٍ وتنفيذ مجموعة متنوعة من التطبيقات.

بالمقارنة مع لغات أخرى مثل جافا وسي++، فإن بايثون تُعد أسهل نسبيًا من حيث الكتابة والفهم. كما تدعم البرمجة متعددة الأنماط مثل البرمجة الكائنية والبرمجة الوظيفية، مما يجعلها مرنة وقابلة للتكيف مع احتياجات المستخدمين المختلفة.

بايثون هي لغة مفتوحة المصدر منذ إطلاقها، مما يتيح للمطورين استخدامها وتطويرها بحرية. وقد لاقت اللغة إقبالاً واسعًا حتى من شركات كبرى مثل Google التي اعتمدتها في العديد من مشاريعها منذ البداية، مما ساهم في زيادة انتشارها وشعبيتها عالميًا.

3. تجهيز بيئة العمل لاستخدام بايثون

لتثبيت بيئة العمل الخاصة بلغة Python على نظام التشغيل Windows، يمكنك اتباع الخطوات التالية:

افتح الموقع الرسمي لـ Python عبر الرابط:

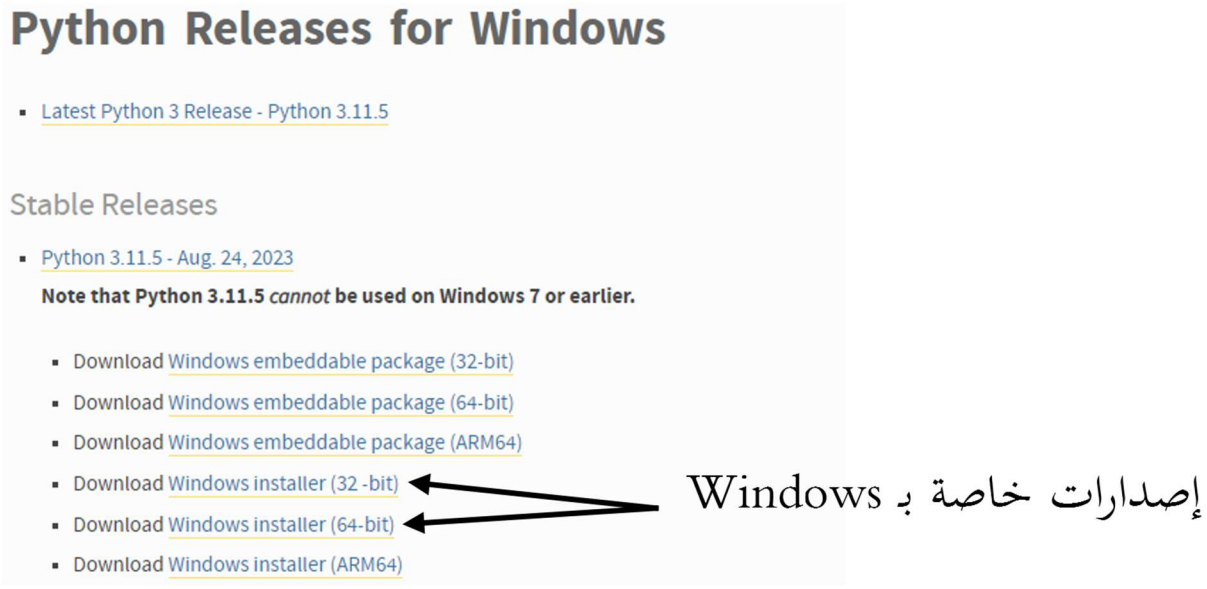
https://www.python.org/downloads/windows/

اختر الإصدار المناسب حسب بنية نظام التشغيل لديك كما هو موضح في الشكل 11:

- Windows Installer (32-bit) : لأنظمة التشغيل 32 بت.

- Windows Installer (64-bit) : لأنظمة التشغيل 64 بت.

تأكد من تنزيل النسخة المستقرة (Stable Release) الأحدث لضمان الحصول على آخر التحديثات والتحسينات.



الشكل 11. تحميل بايثون (Python).

<u>تنبيه حول التوافق:</u>

الإصدار 3.11.5 (على سبيل المثال) غير متوافق مع الأنظمة القديمة مثل Windows 7 أو أقدم. لذا يجب التأكد من أن جهازك يعمل بنظام تشغيل مدعوم قبل التثبيت.

بعد تحميل برنامج Python على جهازك، قم بتشغيل ملف التثبيت. ستظهر لك الشاشة كما هو موضح في الشكل 12، حيث يمكنك تحديد الميزات التي ترغب في تثبيتها:

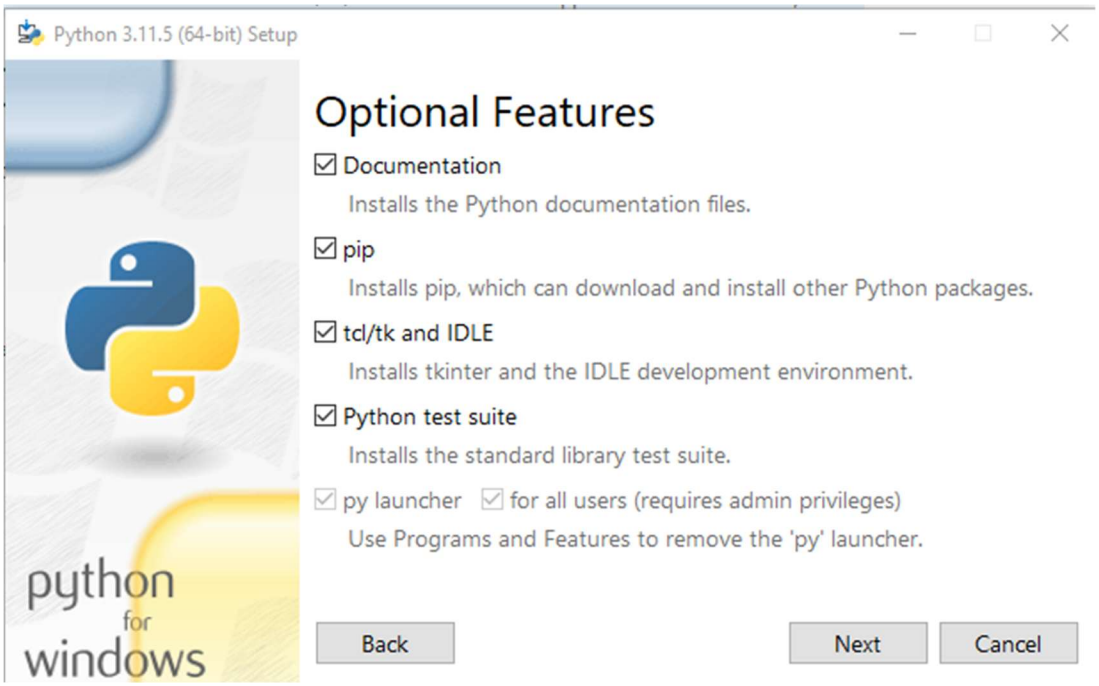
Documentation : تثبيت ملفات التوثيق الخاصة بـPython.

pip : أداة تثبيت الحزم، وهي ضرورية لتنزيل مكتبات Python الإضافية.

tcl/tk and IDLE : تثبيت مكتبة Tkinter لإنشاء واجهات رسومية مع بيئة التطوير المدمجة IDLE.

Python test suite : تثبيت حزمة لاختبار مكتبة Python القياسية.

py launcher : يتيح تشغيل Python من موجه الأوامر بسهولة.



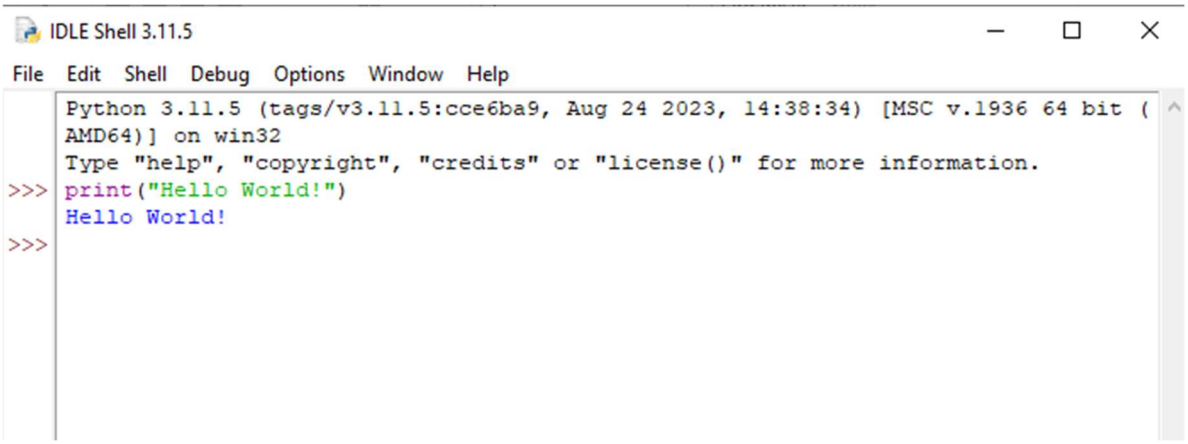
الشكل 12. خيارات تثبيت لغة البرمجة بايثون.

تأكد من تحديد خيار pip، حيث يعد ضروريًا لإدارة الحزم وتنزيل المكتبات الإضافية.

يمكنك اختيار IDLE إذا كنت تود استخدام بيئة التطوير الخاصة بـ Python لكتابة برامج بلغة بايثون أو تحميل بيئة تطوير خارجية مثل Visual Studio Code. يُفضل تحديد خيار "for all users" إذا كنت ترغب في إتاحة Python لجميع مستخدمي الجهاز. اضغط على زر Next لإكمال عملية التثبيت.

بعد تثبيت Python على جهازك، يمكنك العثور على IDLE من خلال قائمة Start في Windows، أو بالبحث عن "IDLE" مباشرة.

عند فتح IDLE، ستظهر واجهة شبيهة بالصورة الموضحة في الشكل 13.



الشكل 13. بيئة التطوير IDLE الخاصة بـ Python.

يمكنك كتابة أوامر Python مباشرة في سطر الأوامر. جرب كتابة الكود التالي:

```
print("Hello World!")
```

بعد الضغط على Enter، سيعرض البرنامج النص:

```
Hello World!
```

هذا يؤكد أن Python مثبتة وتعمل بشكل صحيح على جهازك.

مثال:

لكتابة برنامج يتكون من عدة أسطر انقر على القائمة File ثم New File في بيئة التطوير IDLE أو اضغط مباشرة الاختصار Ctrl+N على لوحة المفاتيح. تظهر بعدها نافذة جديدة نكتب عليها هذا البرنامج:

```
x = 34 - 23              # A comment(تعليقات ليست جزء من البرنامج).
y = "Hello"              # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"
print(x)
print(y)
```

لتنفيذ البرنامج، أنقر على القائمة Run ثم اختر العنصر Run Module في نافذة البرنامج أو يمكنك استخدام الاختصار F5 على لوحة المفاتيح. سيطلب منك حفظ الملف, اختر موقع و اسم مناسب لملف البرنامج لتتمكن من تنفيذه. تكون نتيجة التنفيذ كالتالي:

```
>>>
========== RESTART:
12
Hello World
>>>
```

ما يقوم به البرنامج

4. خصائص لغة البرمجة بايثون

لغة بايثون تتميز بالعديد من الخصائص الفريدة التي تسهم في سهولة استخدامها ومرونتها في تطوير البرامج، سواء كانت مشاريع صغيرة أو تطبيقات معقدة. فيما يلي استعراض لأهم خصائص بايثون:

1. المسافة البادئة (Spaces & Tabs):

- تعتبر المسافة البادئة في بايثون جزءًا أساسيًا من بنية الكود. على عكس اللغات الأخرى التي تعتمد على الأقواس {}، يتم تحديد كتل التعليمات البرمجية من خلال التباعد الصحيح.
- الخطأ في التنسيق (المسافة البادئة) قد يؤدي إلى أخطاء في تشغيل البرنامج، لأن بايثون تعتمد على ذلك لتحديد بداية ونهاية الكتل مثل الحلقات أو الجمل الشرطية.

2. الإسناد المتغيرات:
   - عند الإسناد الأول، يتم إنشاء المتغير تلقائيًا دون الحاجة إلى التصريح عنه مسبقًا.
   - بايثون تقوم باكتشاف نوع المتغير تلقائيًا، فلا حاجة لتحديد الأنواع كما هو الحال في لغات أخرى مثل Java أو C++.
   - الإسناد بـ = والمساواة ==
3. العوامل الرياضية والمنطقية:
   - العمليات الحسابية القياسية مثل: الجمع، الطرح، الضرب، القسمة، وباقي القسمة (%) متاحة.
   - كما توفر بايثون العوامل المنطقية and، or، و not التي تسهل تنفيذ العمليات المنطقية على القيم والمتغيرات.
4. عدم الحاجة إلى التصريح المسبق عن المتغيرات:
   - بايثون توفر طريقة ديناميكية في التعامل مع المتغيرات، حيث يمكن إسناد قيم متنوعة للمتغيرات دون الحاجة إلى تحديد النوع مسبقًا.
5. دعم الأنماط المتعددة:
   - تدعم بايثون أنماطًا برمجية متعددة، منها البرمجة الكائنية (OOP) والبرمجة الوظيفية، مما يتيح للمبرمجين اختيار الأسلوب الذي يناسب احتياجات مشاريعهم.

5. الأنواع الأساسية للبيانات

تدعم لغة بايثون عدة أنواع أساسية للبيانات التي يمكن استخدامها بسهولة في مختلف التطبيقات، وتتميز هذه الأنواع بمرونتها وقدرتها على التعامل مع بيانات متعددة. فيما يلي أبرز الأنواع الأساسية:

1- الأعداد الصحيحة (Integers)

تمثل القيم الرقمية الصحيحة، وهي الافتراضية عند التعامل مع الأرقام.

مثال:

```
x = 34
y = x * 3
x = 5 // 2          # النتيجة 2 (قسمة صحيحة دون باقي)
```

2- الأعداد الحقيقية (Floats)

تمثل الأرقام التي تحتوي على أجزاء عشرية. يمكن استخدامها لحسابات دقيقة.

مثال:

```
pi = 3.1415
```

3- السلاسل الحرفية (Strings)

يتم استخدام السلاسل الحرفية للتعامل مع النصوص. يمكن كتابتها باستخدام علامات الاقتباس المفردة ' أو المزدوجة ". يمكن استخدام الاقتباسات بشكل متداخل للتغلب على مشكلات التداخل.

مثال:

```
print("it's you")
text = """هذا نص
يحتوي على
عدة أسطر."""
print(text)
```

6. المسافات البادئة (spaces & tabs)

في بايثون، المسافات البيضاء ليست اختيارية أو مجرد أسلوب لتحسين عرض الكود، بل تُعد جزءًا من بناء الجملة (Syntax) وتؤثر مباشرة على كيفية تفسير البرنامج. المسافات في بداية الأسطر تُستخدم لتحديد مستوى التداخل (Indentation) وهو ما يحدد الكتل البرمجية بدلاً من استخدام الحاضنات {} أو كلمات مثل begin-end كما هو الحال في الخوارزميات وبعض اللغات الأخرى. كمثال, لاحظ البرنامج التالي حيث نستخدم مسافة لتحديد التعليمات التي ستنفذ داخل الجزء الشرطي if:

```
x = 10
y = 15
if x == 5:
    x = x + 5
    x = x * y
    print(x)
print(y)
```

Bloc if

عند الحاجة إلى كتابة تعليمات طويلة قد تتجاوز عرض السطر، يمكن استخدام الرمز \ في نهاية السطر لتوضيح أن التعليمات مستمرة في السطر التالي. مثلا يمكننا تقسيم عملية جمع كل من x و y و z و g على سطرين كالتالي:

```
x = x + y * \
z + g
```

7. الإسناد (Assignment)

يمكن في بايثون إسناد أكثر من قيمة إلى أكثر من متغير في نفس السطر باستخدام الفصل بينها بفاصلة. في المثال التالي, تم إسناد القيمة 2 للمتغير x وإسناد القيمة 4 للمتغير y.

```
x,y = 2,4
```

يوفر هذا النوع من الإسناد كتابة أكثر اختصارًا ووضوحًا، خاصة عند التعامل مع عدة متغيرات.

يمكن في بايثون إسناد نفس القيمة إلى عدة متغيرات في سطر واحد باستخدام إسناد متسلسل:

```
x = y = z = 10
```

8. التسلسلات (Sequence Types)

التسلسلات هي هياكل بيانات تُستخدم لتخزين عدة عناصر (قيم) في ترتيب معين. يمكن أن تحتوي هذه التسلسلات على أنواع بيانات مختلفة مثل الأعداد الصحيحة، النصوص، أو حتى كائنات أخرى. كل نوع من أنواع التسلسلات يقدم خصائص مميزة تساعد المبرمج على تنظيم البيانات والتعامل معها بكفاءة. أهم الأنواع هي: Tuple و List (قائمة) و Set (مجموعة).

8.1. Tuple

هي تسلسل مرتب من العناصر وغير قابل للتغيير (Immutable) ويمكن أن يحتوي على عناصر متكررة. يُفضل استخدام هذا النوع في الحالات التي نحتاج فيها إلى حماية البيانات من التغيير، مثل الإحداثيات (Coordinates) أو البيانات الثابتة.

مثال:

```
t = (1, 2, 3, 1)
```

8.2. القائمة (List)

هي تسلسل مرتب من العناصر قابل للتغيير (Mutable)، مما يعني أنه يمكن إضافة، حذف، أو تعديل القيم داخله. ويمكن أن تحتوي على عناصر متكررة. تُستخدم القوائم في الحالات التي نحتاج فيها إلى تعديل البيانات، مثل تخزين مجموعة من القيم المتغيرة.

مثال:

```
lst = [1, 2, 4, 'a', 3, 3, 3, 6, 5]
print("Original list: ", lst)
lst[3] = 77
lst.append(0)
print("Newlist: ", lst)
```

تنفيذ هذا البرنامج يعرض على الشاشة:

```
Original list: [1, 2, 4, 'a', 3, 3, 3, 6, 5]
New list: [1, 2, 4, 77, 3, 3, 3, 6, 5, 0]
```

8.3. المجموعة (Set)

هي تسلسل غير مرتب من العناصر وغير قابل للتكرار، أي أن كل عنصر في المجموعة يجب أن يكون فريدًا. أيضا غير قابلة للتغيير بالنسبة لقيم العناصر (Immutable values)، لكن المجموعة نفسها قابلة للتعديل من حيث الإضافة أو الحذف. تُستخدم المجموعات عند الحاجة إلى تخزين عناصر فريدة، مثل حذف التكرارات من قائمة.

مثال:

```
set1 = set([1, 2, 4, 4, 3, 3, 3, 6, 5])    #تحويل قائمة إلى مجموعة
set1.add(8)                                 #إضافة عنصر إلى المجموعة set1
set2 = {1, 2, 3, 'c'}                       #إنشاء مجموعة جديدة
print("set1: ", set1)
print("set2: ", set2)
```

تنفيذ هذا البرنامج يعرض على الشاشة:

```
set1: {1, 2, 4, 3, 6, 5, 8}
set2: {1, 2, 3, 'c'}
```

9. الفهرسة في البيانات المتسلسلة (Index)

في لغة بايثون توجد الفهرسة الموجبة والسالبة. الفهرسة الموجبة (Positive Index) تبدأ من اليسار، حيث يكون أول عنصر في التسلسل عند الفهرس 0. يتم زيادة الفهرس بمقدار 1 لكل عنصر يليه في التسلسل.

مثال:

```
t = (23, 'abc', 4.56, [2, 3], 'def')
print(t[1])  # النتيجة: 'abc'
```

يشير t[1] إلى العنصر الثاني في Tuple وهو 'abc'. لاحظ أن العنصر الرابع t[3] هو عبارة عن قائمة بعنصرين، إذ يمكن في بايثون إنشاء أنواع متسلسلة مختلطة (Nested).

أما الفهرسة السالبة (Negative Index) فتبدأ الفهرسة من اليمين، حيث يكون آخر عنصر في التسلسل عند الفهرس -1. يتم تقليل الفهرس بمقدار 1 لكل عنصر سابق في التسلسل.

مثال:

```
print(t[3-])  # النتيجة: 4.56
```

تُستخدم الفهارس السالبة عندما نريد الوصول إلى العناصر بدءًا من نهاية التسلسل، دون الحاجة إلى معرفة الطول الكلي للتسلسل. وهذا مفيد عندما نحتاج إلى الوصول إلى العناصر الأخيرة من التسلسل بسهولة.

10. تقطيع التسلسلات (Slicing)

التقطيع (Slicing) هو عملية استخراج جزء من تسلسل (مثل قائمة أو Tuple أو سلسلة نصية) بناءً على فهارس معينة. يسمح التقطيع للمبرمج بالحصول على نسخة جزئية من التسلسل الأصلي دون تعديل المحتويات الأساسية. للحصول على جزء من التسلسل باستخدام التقطيع يجب تحديد بداية ونهاية التقطيع باستخدام الفهارس الموجبة أو السالبة. الشكل العام لعملية التقطيع كالتالي:

```
sequence[start:end:step]
```

حيث:

start : الفهرس الذي يبدأ عنده التقطيع (يُضمّن في النتيجة (Included)).

end : الفهرس الذي ينتهي عنده التقطيع (لا يُضمّن في النتيجة (Excluded)).

step : يحدد القفز بين العناصر (اختياري).

مثال:

للحصول على مجموعة جزئية ابتداءا من العنصر الثاني إلى غاية العنصر الثالث:

```
t = (23, 'abc', 4.56, [2, 3], 'def')
print(t[1:3])  # النتيجة: ('abc', 4.56)
```

للحصول على مجموعة متكونة من العنصر الثالث إلى غاية العنصر قبل الأخير:

```
print(t[2:-1])  # النتيجة: (4.56, [2, 3])
```

للحصول على نسخة إلى غاية العنصر الثالث:

```
print(t[:3])  # النتيجة: (23, 'abc', 4.56)
```

للحصول على نسخة إبتداءا من العنصر الثالث إلى آخر المجموعة:

```
print(t[2:])  # النتيجة: (4.56, [2, 3], 'def')
```

11. العمليات على التسلسلات

11.1. المعامل in

في بايثون، يتم استخدام المعاملين in و not in للتحقق مما إذا كانت قيمة معينة موجودة داخل تسلسل من العناصر (مثل قائمة، Tuple، مجموعة، أو سلسلة نصية). يعيد هذان المعاملان قيمة منطقية (Boolean): إما True إذا كانت القيمة موجودة، أو False إذا لم تكن كذلك.

مثال:

```
t = [1, 2, 3, 4]
print(3 in t)  # النتيجة: True
print(5 in t)  # النتيجة: False
print(4 not in t)  # النتيجة: False
print(5 not in t)  # النتيجة: True
```

11.2. المعامل +

يمكن استخدام المعامل (+) لدمج عدة أنواع من السلسلات مثل tuple، القوائم (list)، والسلاسل النصية (strings). يقوم هذا المعامل بإنشاء نسخة جديدة تحتوي على العناصر من التسلسلين أو النصوص المدمجة.

مثال:

```
tuple1 = (1, 2, 3) + (4, 5, 6)
print(tuple1)  # النتيجة: (1, 2, 3, 4, 5, 6)
list1 = [1, 2, 3] + [4, 5, 6]
print(list1)  # النتيجة: [1, 2, 3, 4, 5, 6]
str1 = 'Hello' + ' ' + 'World'
print(str1)  # النتيجة: 'Hello World'
```

يجب أن تكون المجموعات التي يتم دمجها من نفس النوع. على سبيل المثال، لا يمكن دمج قائمة مع توبل باستخدام (+).

11.3. المعامل *

يتيح المعامل * في بايثون تكرار التسلسلات بعدد معين من المرات. ينتج عن هذا التكرار مجموعة جديدة تحتوي على نفس العناصر مكررة بعدد المرات المحدد.

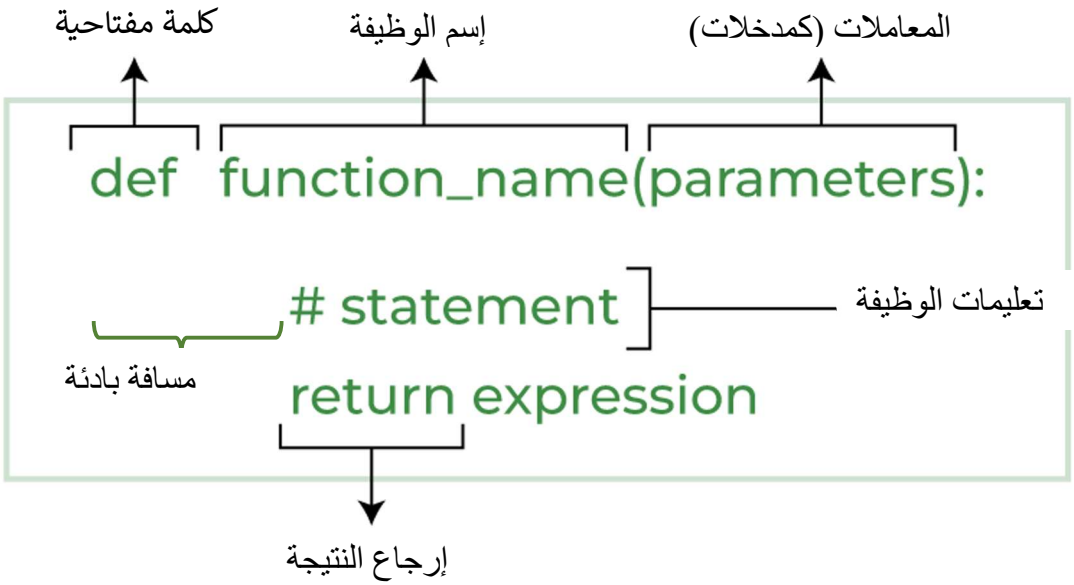
مثال:

```
tuple1 = (1, 2, 3) * 3
print(tuple1)  # النتيجة: (1, 2, 3, 1, 2, 3, 1, 2, 3)
list1 = [1, 2, 3] * 3
print(list1)  # النتيجة: [1, 2, 3, 1, 2, 3, 1, 2, 3]
str1 = 'Hello' * 3
print(str1)  # النتيجة: 'HelloHelloHello'
```

المعامل * لا يغير التسلسل الأصلي، بل ينشئ نسخة جديدة تحتوي على العناصر مكررة. ويمكن استخدام التكرار لتوليد أنماط أو سلاسل نصية بشكل سريع وسهل. أيضا عند التكرار، تظل العناصر مرتبة بنفس ترتيبها الأصلي في التكرار الجديد.

12. الوظائف (Functions)

الوظائف في بايثون هي كتل برمجية قابلة لإعادة الاستخدام تُعرف باستخدام الكلمة المفتاحية def. تُستخدم الوظائف لتقسيم البرنامج إلى أجزاء أصغر وأكثر تنظيمًا، مما يجعل الكود سهل القراءة والصيانة. يتم تعريف وظيفة جديدة في بايثون كما هو موضح في الشكل 14.



الشكل 14. الشكل العام لتعريف وظيفة جديدة في بايثون.

مثال:

```
def add_numbers(a, b):
    result = a + b  # جمع المعاملات
    return result  # إرجاع النتيجة

# استدعاء الوظيفة
sum_result = add_numbers(5, 3)
print(sum_result)  # النتيجة: 8
```

في المثال تم تعريف وظيفة باسم add_numbers تستقبل معاملين كمدخلات a و b. ثم داخل جسم الوظيفة، يتم جمع المعاملين وتخزين النتيجة في المتغير result. إرجاع النتيجة result يكون باستخدام الكلمة المفتاحية return.

13. تثبيت مكتبات غير قياسية في بايثون

تعتبر المكتبات في بايثون أدوات جاهزة تُسهّل تنفيذ المهام المعقدة مثل التعامل مع البيانات، الرياضيات، التعلم الآلي، وغيرها. يمكنك استخدام pip، وهو مدير الحزم الافتراضي في بايثون، لتثبيت المكتبات مثل Numpy. للقيام بتثبيت أي مكتبة نفتح أولا نافذة الأوامر (Command Prompt) للنظام عن طريق الضغط على Win + R ثم كتابة cmd. لتثبيت المكتبة نكتب الأمر التالي ثم نضغط Enter:

```
pip install <lib_name>
```

بحيث lib_name هو اسم المكتبة المراد تثبيتها. كمثال، لتثبيت مكتبة numpy ننفذ الأمر:

```
pip install numpy
```

Numpy هي اختصار لـ Numerical Python، وهي مكتبة برمجية مفتوحة المصدر تُستخدم في لغة بايثون لتوفير أدوات قوية للتعامل مع المصفوفات (Arrays) والعمليات الرياضية عالية الأداء. تُعد Numpy واحدة من أهم المكتبات في مجال تحليل البيانات والعلوم الحاسوبية، كمثال، البرنامج التالي يعرف مصفوفة بسيطة ثم يقوم بجمع عناصرها باستخدام تعليمة واحدة sum موجودة ضمن مكتبة numpy:

```
import numpy as np

# إنشاء مصفوفة Numpy
arr = np.array([1, 2, 3, 4, 5])

# طباعة المصفوفة
print("المصفوفة:", arr)

# طباعة مجموع العناصر
print("مجموع العناصر:", np.sum(arr))
```

بحيث:

- يتم استيراد مكتبة Numpy باستخدام import numpy as np.
- يتم إنشاء مصفوفة باستخدام np.array.
- يتم طباعة المصفوفة ومجموع العناصر باستخدام np.sum.

نتيجة تنفيذ البرنامج السابق هي:

```
المصفوفة: [1 2 3 4 5]
مجموع العناصر: 15
```

14. تمارين

1- أكتب وظيفة `isPrime(n)` تقوم بالتحقق ما إذا كان عدد معطى n أولي أو لا.

2- أكتب برنامج يقوم بعرض الأعداد الأولية الأقل من 100.

3- أكتب وظيفة `fact(n)` تحسب !n

4- أكتب برنامج بلغة بايثون يسمح بالتحويل بين مختلف أنظمة العد.

5- في سنة 2004 تم اكتشاف عدد أولي ضخم يتكون من 2357207 رقم، وهو:

$$28433 \times 2^{7830457} + 1$$

أكتب برنامج بايثون يقوم بإيجاد آخر 15 رقم من هذا العدد الأولي (الأرقام على اليمين).

6- أكتب وظيفة SolveFunction بلغة بايثون تحسب حلول المعادلات من الدرجة الأولى والثانية.

7- يمكن كتابة العدد $\frac{1}{2}$ على شكل مجموع مقلوب مربعات عدة أعداد طبيعية غير مكررة.

مثلا، باستخدام المجموعة: $\{2, 3, 4, 5, 7, 12, 15, 20, 28, 35\}$ نكتب:

$$\frac{1}{2} = \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{12^2} + \frac{1}{15^2} + \frac{1}{20^2} + \frac{1}{28^2} + \frac{1}{35^2}$$

باستخدام أعداد طبيعية محصورة بين $[2, 45]$ يمكن كتابة $\frac{1}{2}$على شكل مجموع مقلوب مربعات بثلاث طرق مختلفة, بالإضافة للمجموعة السابقة لدينا مجموعتين اضافيتين و هما $\{2,3,4,6,7,9,10,20,28,35,36,45\}$ و $\{2,3,4,6,7,9,12,15,28,30,35,36,45\}$

كم توجد من طريقة بحيث يمكن كتابة العدد $\frac{1}{2}$ على شكل مجموع مقلوب مربعات باستخدام أعداد محصورة بين $[2, 80]$.

15. خاتمة

في ختام هذا الفصل، نأمل أن يكون الطالب قد اكتسب فهمًا قويًا للأساسيات والمفاهيم التي تقدمها لغة البرمجة بايثون. لقد تناولنا خطوات تثبيت وإعداد بيئة العمل، بالإضافة إلى كتابة أولى البرامج واستخدام بيئة التطوير المدمجة (IDLE). كما استعرضنا بعض الميزات الفريدة في بايثون مثل سهولة التعامل مع المتغيرات، التعامل مع النصوص والمجموعات، وإنشاء الوظائف. تُعد بايثون بداية رائعة لكل من يرغب في تعلم البرمجة أو تطوير مهاراته، نظرًا لبساطتها وإمكانياتها الواسعة. الآن، ومع هذه المعرفة، يمكن للطالب التعمق في مجالات أكثر تخصصًا أو البدء في تطوير مشاريعه الخاصة باستخدام بايثون.

الخـاتمة

تختتم هذه المطبوعة رحلتها في الإعلام الآلي وأساسيات البرمجة، حيث تناولت موضوعات متنوعة بدءًا من المفاهيم الأساسية للحاسوب، مرورًا بأنظمة العد والخوارزميات، وصولًا إلى تطبيق البرمجة باستخدام لغة بايثون. من خلال هذا المحتوى، تم توفير مزيج متوازن بين النظرية والتطبيق العملي، مما يتيح للطلبة تطوير مهاراتهم التحليلية والبرمجية.

إن هذه المطبوعة تمثل قاعدة معرفية قوية تمكن الطلبة من مواصلة استكشاف مجالات أوسع في علوم الحاسوب والتكنولوجيا، سواء في تطوير البرمجيات أو في حل المشكلات الرياضية باستخدام الأدوات الحاسوبية. كما يفتح الباب نحو تعلم لغات برمجة أخرى وتطبيق ما تعلموه في سياقات حياتية وعملية مختلفة.

نأمل أن تكون هذه المطبوعة قد أضافت إلى الطلبة معرفة جديدة وساهمت في تعزيز شغفهم بالتعلم والاكتشاف في هذا المجال المتطور.

المـراجع

[1] Aho, A. V., & Ullman, J. D. (1992). Foundations of Computer Science. W. H. Freeman.

[2] Archer, J. E., Conway, R., & Schneider, F. B. (1986). Understanding Operating Systems. Holt, Rinehart, and Winston.

[3] Brookshear, J. G., & Brylow, D. (2014). Computer Science: An Overview. Pearson.

[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

[5] Downey, A. B. (2015). Think Python: How to Think Like a Computer Scientist (2nd ed.). O'Reilly Media.

[6] Gaddis, T. (2018). Starting Out with Python (4th ed.). Pearson.

[7] Guttag, J. V. (2016). Introduction to Computation and Programming Using Python (2nd ed.). MIT Press.

[8] Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley.

[9] Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.