البيانات : Data

غالبا ما تشير كلمة بيانات (البيانات المدخلة) إلى مجموعة من الحقائق الضرورية التي تعبر عن مواقف و أفعال معينة سواء أكان ذلك التعبير بأرقام أو رموز أو كلمات أو إشارات أو ما شابه ذلك ولا تفيد هذه البيانات في شئ و هي على صورتها الحالية , و من الأمثلة على ذلك :-

- درجة الطالب في مقرر ما .
- عدد ساعات العمل لموظف في الأسبوع.
- أجرة الساعة الواحدة لعمل الموظف.
- عدد الرحلات الجوية بين مدينتين.

و البيانات على أنواع منها:-

❖ **الثوابت** Constants : هو مقدار غير متغير و ذا قيمة ثابتة خلال فترة تنفيذ الخوارزمية او البرنامج. و يمكن التمييز بين نوعين من الثوابت :

- الثوابت العددية Numerical Constants و هي سلسلة من الأرقام (٠ إلى ٩) تستخدم في تمثيل المعطيات العددية. و يمكن أن يكون الثابت العددي على عدة أنواع منها الصحيح و الحقيقي و الأسي .
- **ثوابت السلسلة الرمزية** String Constants و هي عبارة عن سلسلة من الرموز تكتب بين علامتي اقتباس . يستخدم هذا النوع في تمثيل المعطيات غير العددية .

❖ **المتغيرات** Variables : هو اسم رمزي يمثل موقعا تخزينيا" في وحدة التخزين و تتغير قيمة المتغير عدة مرات أثناء تنفيذ البرنامج و هناك نوعين من المتغيرات:

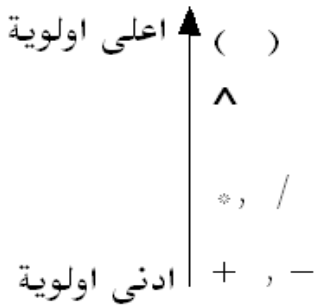
✓ *المتغيرات العددية* Numerical Variables و هي المتغيرات التي تستخدم لتخزين القيم العددية فقط .

✓ *المتغيرات الحرفية (الرمزية)* String Variables و تستخدم لتخزين القيم الحرفية .

التعابير Expressions :- تقسم إلى نوعين :

التعابير الحسابية : Arithmetic Expressions

- هو مجموعة من الحدود تتكون من ثوابت عددية و متغيرات عددية بالإضافة إلى رموز العمليات الحسابية (+ , - , * , / , ^) . وإذا اجتمع في التعبير الحسابي أكثر من عملية واحدة فان تسلسل تنفيذ العمليات يتم من اليسار إلى اليمين وفقا لسلم الأولويات :



التعابير المنطقية : Logic Expressions

تستعمل التعابير المنطقية مع عبارات الشرط (IF) للمقارنة وتستخدم للمقارنة العوامل المنطقية مثل (… ,=, # ,=> ,<= ,> ,<)

المعلومات : Information

هي المعرفة التي تكونت نتيجة لتحليل البيانات المدخلة بإجراء العمليات المطلوبة عليها و المعلومات الناتجة تكون أكثر معنى من البيانات و تساعد متخذي القرارات في تحقيق أغراض معينة, و من أمثلة ذلك :-

- معدل الطالب.
- دخل الموظف الأسبوعي.

معالجة البيانات : Data Processing

و تعني إخضاع البيانات المدخلة للتحليل و إجراء مجموعة من العمليات عليها باستخدام وسائل معينة بغرض الحصول على معلومات مفيدة في اتخاذ القرارات, و الشكل التالي يوضح العلاقة التي تربط بين البيانات , و المعلومات , ومعالجة البيانات:-

مثال : أفترض أن المطلوب هو حساب معدل الطالب في مجموعة من الدروس فان :-

- البيانات المدخلة تتمثل في درجات الطالب الواحد في كل الدروس.
- عمليات المعالجة تتمثل في مجموعة من الأوامر المتتابعة:-
  - ☒ إيجاد مجموع درجات الطالب.
  - ☒ إيجاد معدل الطالب.
  - •
- المعلومات الناتجة تتمثل في إظهار (طباعة) معدل الطالب.

الخوارزميات : Algorithms

تتمثل الخطوات المهمة في تحليل المسائل بما يلي:-

١. فهم المسألة :-
يتطلب ذلك قراءتها و دراستها جيدا" حيث أن محاولة حل مسالة دون استيعابها يؤدي بالتأكيد إلى نتائج خاطئة أو ربما نتائج ناقصة أو عدم الحصول على أي نتائج مطلقا".

٢. تحديد معطيات المسألة :-
يجب توفير البيانات التي يتطلبها حل المسالة حيث أن هذه البيانات على درجة كبيرة من الأهمية بالنسبة لمعالجة الخوارزمية و تسمى المعطيات (Inputs).

٣. تحديد النتائج المطلوبة :-
تتمثل هذه الخطوة بتحديد الهدف من حل الخوارزمية , أي تحديد النتائج التي يفترض أن نحصل عليها و تسمى هذه النتائج بالمخرجات (Outputs).

٤. طريقة الحل :-
يجب وضع خطوات مناسبة للحل بالاعتماد على المعطيات وصولا إلى النتائج.

تاريخ الخوارزمية

يُستخدم مُصطلح Algorithmهذه الأيّام ليعني "مجموعة الأوامر أو القواعد التي تَتبعها الآلة (عادةً الحاسوب) لتحقيقِ هدفٍ مُحدَّد. والمُصطلح بديلٌ مُتَّفَقٌ عليه لسيرورة حلّ المسألة problem " "solving procedureوهو يرتبط عادةً بالوصف التسلسليّ الذي يستهدف تحقيقه. كالبحث مثلاً في مخزونٍ ضخمٍ من البيانات لاستخراج مجموعةٍ من البيانات ضمن مُواصفاتٍ دليليّة keywords محدَّدة مُسبقاً؛ أو السيرورة لتعمية (أو تجفير encryptions) المعلومات أو الرسائل كي لا يستطيع فَهمها إلّا المسموح لهم بذلك.

بدأ استخدامُ المُصطلح بهذا المعنى المُستحدَث أوائل القرن العشرين في حقلَي الرياضيّات والحَوسبة الآليّة، لكنّ جذوره في التاريخ أعمق من ذلك بكثير. فالمَعاجِم الإنكليزيّة القديمة توضح أنّ المُصطلح يُشير إلى نِظام العدّ العشريّ أو العربيّ الذي أسماه الأوروبيّون algoritmi de numero، وقد كان الاعتقاد سائداً بأنّه جاء من دمْج الكلمتَيْن (algiros) أي (مؤلِم) و (arithmos) أي رقم وكان دونالد كونوث Knuth Donald (جامعة ستانفورد) من أوائل مَن أَوضحَوا أنّ المُصطلَح مَنسوبٌ إلى عالِم الرياضيّات في القرن التاسع الميلادي أبي جعفر محمّد بن موسى الخوارزمي. وهو من خوارزم، وهي واحة كبيرة تقع على نهر جيحون في غرب آسيا الوسطى، يحدّها من الشمال بحر الآرال، ومن الشرق صحراء قيزيل، ومن الجنوب صحراء قرّه قوم، ومن الشرق هضبة استجورت وتقع خوارزم اليوم في أوزبكستان وكازاخستان وتركمانستان.

يُعتبر الخوارزمي من أوائل عُلماء الرياضيّات، وقد أَسهمت أعماله بشكلٍ كبير في تقدُّم الرياضيّات في عصره والعصور اللّاحقة. اتّصلَ بالخليفة المأمون وعملَ في بَيت الحكمة في بغداد وكَسب ثقة الخليفة فأولاه بَيت الحِكمة، كما عَهَدَ إليه برسم خارطة للأرض عمل فيها أكثر من سبعين جغرافيّاً. وقبل وفاته، في ٨٥٠م، كان الخوارزمي قد تركَ العديد من المؤلّفات في علوم الرياضيّات والفلك والجغرافيا، من بينها كِتاب "المُختصر في حساب الجبر والمقابلة "الذي يُعَدّ من أهمّ كُتبه.

يعود الاهتمام المُعاصِر بمُصطلح Algorithmإلى الفترة 1980 - 1960عندما كان المُهتمّون بعِلم الحاسوب والبَرْمَجة يُحاولون قصارى جهودهم منْح هذا الحقل العِلمي الجديد (هذا إذا كان عِلما) صِفةً اعتباريّة وأكاديميّة مُستقلّة عن باقي التخصّصات والحقول القريبة منه. هنا بَرَزَ دَور دونالد كونوث في إسهامتين رئيسيّتَيْن: الأولى في مقدّمة موسوعته "فنّ البَرْمَجة الحاسوبيّة The Art of - Computer Programming"التي عمل عليها منذ العام ١٩٦٢ وخطَّط لأن تكون بسبعة أجزاء، لكنّه توقَّف عن إتمامها بعد صدور الجزء الرّابع، وتوجَّهت اهتماماته البحثيّة نحو موضوعاتٍ أخرى. وقد صَدرت الأجزاءُ الأربعة في الفترة ١٩٦٨ - ١٩٧٣. ويَعترف كونوث في مقدّمة الجزء الأوّل أنّ كلمة "فنّ "التي جاءت في العنوان مُقتبَسة من التسمية العربيّة للإشارة إلى أعلى مستويات الأداء العِلمي أو المهني، فيُقال فنّ العمارة وفنّ الحيل (الميكانيك) للإشارة إلى المستوى المُتقدِّم في هذه المِهن. جاء في المقدّمة أيضاً توضيحٌ لمَصدر مُصطلح Algorithmيُشير إلى أنّه منسوب لعالِم الرياضيّات محمّد بن موسى الخوارزمي.

تعريف الخوارزميات Algorithm Definition

الخوارزميات و مفردها خوارزمية (بالإنجليزية Algorithm) يمكن تعريف **الخوارزمية** على انها مجموعة من الأوامر التي يجب أن يتبعها جهاز الكمبيوتر لإجراء العمليات الحسابية أو غيرها من عمليات حل المشكلات.

وتعريفها الرسمي، **الخوارزمية** هي مجموعة محدودة من التعليمات التي يتم تنفيذها بترتيب معين لأداء مهمة معينة. الخوارزمية ليست برنامج أو كود يمكن كتابته لحل مشكلة بصورة تفصيلية، إلا إنها منطق بسيط لمشكلة يتم تمثيلها كوصف غير رسمي في شكل مخطط انسيابي أو كود زائف.

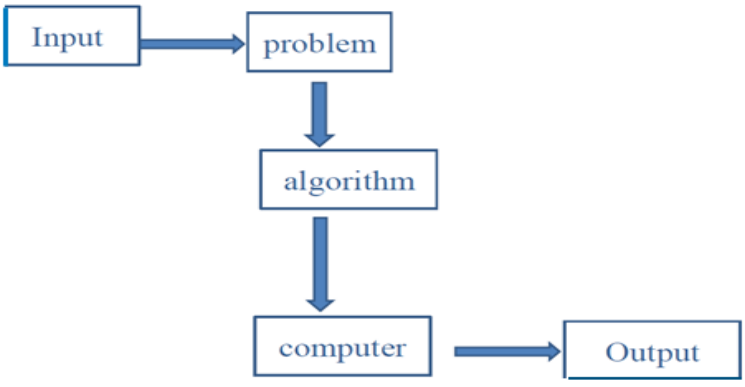
يجب أن يكون لكل خوارزمية مجموعة من الخصائص (الصفات)التالية:-

١. الخوارزمية تكون دقيقة :-
كل خطوة في الخوارزمية يجب أن تكون واضحة تماما دون غموض أو شك عن العملية المقصودة بتلك الخطوة. وان تقسم إلى خطوات معينة و متتالية تؤدي إلى النتيجة المطلوبة , وتصلح لحل جميع المسائل من نفس النوع أو الطراز.

٢. الخوارزمية يجب أن تكون منتهية:-
عندما ينفذ شخص ما أو الآلة خوارزمية معينة يجب الوصول إلى في النهاية إلى نقطة تكون عندها المهمة قد اكتملت أي لا يستمر تنفيذ الخوارزمية إلى ما لانهاية. وان يكون الزمن اللازم لتنفيذها معقولا لأنه من المنطق أن نرفض الخوارزمية التي تأخذ وقتا طويلا في التنفيذ مهما كانت درجة صحتها.

٣. الخوارزمية يجب أن تكون كاملة و فعالة:-
الخوارزمية تأخذ بنظر الاعتبار جميع الظروف والاحتمالات التي يمكن أن تجابه طريق التنفيذ , كما لا يجب أن تحتوي على تعليمات يصعب على الشخص أو الآلة تتبعها لتنفيذ الخوارزمية.

بالتالي يمكن توضيح استراتيجية عمل الخوارزمية من خلال الشكل (١-١)



الخطوات العامة لعمل الخوارزمية

يمكن شرح بعض المصطلحات المهمة في تعريف الخوارزمية.

- **المشكلة Problem** : يمكن تعريف المشكلة على أنها مشكلة في العالم الحقيقي أو مشكلة مثيل في العالم الحقيقي والتي تحتاج إلى تطوير برنامج أو مجموعة من التعليمات لها. الخوارزمية هي مجموعة من التعليمات.
- **الخوارزميةAlgorithm** : يتم تعريف الخوارزمية على أنها عملية خطوة بخطوة سيتم تصميمها لحل مشكلة ما.
- **الإدخالInput** : بعد تصميم الخوارزمية، يتم إعطاء الخوارزمية المدخلات الضرورية والمرغوبة.
- **وحدة المعالجةProcessing unit** : سيتم تمرير المدخلات إلى وحدة المعالجة لإنتاج المخرجات المطلوبة.
- **المخرجاتOutput** : ويشار إلى نتيجة أو نتيجة البرنامج بالمخرجات.

* **ملاحظة**: اعطاء مثال حياتي عن امكانية استخدام الخوارزمية لتنظيم مسار الحياة اليومية

التفكير الخوارزمي وتحلل المشكلة .Problem decomposition

تحلل المشكلة في علوم الكمبيوتر هو عملية يتم فيها تقسيم مشكلة أو نظام معقد إلى أجزاء أصغر وأكثر قابلية للإدارة. يمكن بعد ذلك تحليل هذه الأجزاء الصغيرة أو حلها أو معالجتها بشكل منفصل لحل المشكلة الأكبر أو لفهم النظام ككل بشكل أفضل. يخلق هذا المبدأ الأساس لمجموعة متنوعة من المفاهيم المهمة في علوم الكمبيوتر، ودورها في حل المشكلات.

لا يقتصر التحلل في علوم الكمبيوتر على تحليل المشكلات فحسب. يتعلق الأمر بفهم وتنظيم أسلوبك في المهام والأنظمة المعقدة. قد تبدو المشكلة في كثير من الأحيان شاقة، خاصة عندما تظهر كمهمة واحدة كبيرة. ومع ذلك، باستخدام التحليل، يمكنك تقسيم المشكلة إلى مجموعات فرعية، والتي غالبًا ما تكون أسهل في التعامل معها وتصحيح أخطائها وصيانتها. في الواقع، يمكن تصور التحلل كأداة مفيدة في صندوق أدوات حل المشكلات لديك. بدلاً من معالجة المشكلة بأكملها مرة واحدة، يمكنك تقسيمها إلى مشاكل فرعية، والتعامل مع كل واحدة منها على حدة. على سبيل المثال، تحتاج إلى إنشاء موقع ويب، وهي مهمة معقدة في حد ذاتها. ومع ذلك، باستخدام التحليل، يمكنك تقسيم المهمة إلى مهام أصغر مثل إنشاء شريط التنقل، أو إعداد الصفحة الرئيسية، أو إضافة نموذج الاتصال. تمثل

كل مهمة من هذه المهام مشكلة فرعية يمكنك معالجتها بشكل فردي، مما يجعل عملك أكثر قابلية للإدارة.

يلعب التحلل في علوم الكمبيوتر دورًا حاسمًا في حل المشكلات. إنه أحد المفاهيم الأساسية في مجالات مثل تصميم الخوارزمية وتطوير البرمجيات. يساعدك التحلل في:

- تصميم وتنفيذ الخوارزميات
- إنشاء أنظمة برمجية معقدة
- تطوير الألعاب
- بناء المواقع

من الأمثلة الجيدة على التحلل في تصميم الخوارزميات طريقة فرق تسد، حيث يتم تقسيم المشكلة إلى مشاكل فرعية أصغر. يتم بعد ذلك حل كل مشكلة فرعية بشكل مستقل، ويتم دمج الحلول لحل المشكلة الأصلية.

المبادئ الأساسية للتحلل في علوم الكمبيوتر

يعتمد التحلل في علوم الكمبيوتر على مبادئ أساسية معينة:

- التقسيم: يتم تقسيم المشكلة أو النظام إلى أجزاء أصغر.
- التجريد: يتم تبسيط كل جزء والتعامل معه بمعزل عن بقية المشكلة أو النظام.
- الحل: يتم بعد ذلك حل أو معالجة كل جزء على حدة.
- التكامل: يتم بعد ذلك دمج حلول الأجزاء الفردية لحل المشكلة الأصلية أو لتشكيل النظام بأكمله.

توجه هذه المبادئ استخدام التحليل في علوم الكمبيوتر وهي ضرورية لتطبيقه الفعال في مهام حل المشكلات. تشكل هذه المبادئ أيضًا الأساس للعديد من الاستراتيجيات الخوارزمية وممارسات البرمجة وتقنيات تصميم البرمجيات. تذكر دائمًا أن التحلل لا يقتصر فقط على حل المشكلات. يتعلق الأمر بتبسيط المشكلات وجعلها قابلة للإدارة. وفي عالم علوم الكمبيوتر، ليس دائمًا حجم المشكلة هو المهم، ولكن كيفية حلها.

كيفية عمل الخوارزمية

الخوارزميات هي إجراءات خطوة بخطوة مصممة لحل مشاكل محددة وتنفيذ المهام بكفاءة في مجالات العلوم كافة. تشكل هذه المجموعات القوية من التعليمات العمود الفقري للتكنولوجيا الحديثة وتحكم كل شيء بدءًا من عمليات البحث على الويب وحتى الذكاء الاصطناعي. وإليك كيفية عمل الخوارزميات:

- الإدخال Input: تأخذ الخوارزميات بيانات الإدخال، والتي يمكن أن تكون بتنسيقات مختلفة، مثل الأرقام أو النصوص أو الصور.
- المعالجةProcessing : تقوم الخوارزمية بمعالجة البيانات المدخلة من خلال سلسلة من العمليات المنطقية والرياضية، ومعالجتها وتحويلها حسب الحاجة.
- الإخراجOutput : بعد اكتمال المعالجة، تنتج الخوارزمية مخرجات، والتي يمكن أن تكون نتيجة أو قراراً أو بعض المعلومات الأخرى ذات المعنى.
- الكفاءة :Efficiency أحد الجوانب الرئيسية للخوارزميات هو كفاءتها، والتي تهدف إلى إنجاز المهام بسرعة وبأقل قدر من الموارد.
- التحسين :Optimization يبحث مصممو الخوارزميات باستمرار عن طرق لتحسين خوارزمياتهم، مما يجعلها أسرع وأكثر موثوقية.
- التنفيذ :Implementation يتم تنفيذ الخوارزميات بلغات برمجة مختلفة، مما يمكن أجهزة الكمبيوتر من تنفيذها وتحقيق النتائج المرجوة.

مفهوم الخوارزميات في الحاسوب

إذا أردت للحاسب أن يقوم بأي شيءٍ، فيجب عليك أولاً كتابة برنامج حاسوبي، ولكتابة برنامجٍ حاسوبيٍّ، عليك أن تخبر حاسوبك بدقةٍ وبخطواتٍ مفصلةٍ ماذا تريد منه أن يفعل، ليقوم بعدها الحاسب بتنفيذ هذه الخطوات بحرفيةٍ تامّةٍ، وبالتالي تنفيذ البرنامج، وصولاً إلى تطبيق الهدف المعني.
عندما تطلب من الحاسب أن يفعل شيئًا ما، تستطيع أيضاً أن تختار كيف يفعل هذا الشيء، وهنا يأتي دور الخوارزميات في الحاسوب فهي تعتبر تقنيةً بسيطةً لإنجاز الهدف المعني.
في عالم البرمجة الحاسوبية، هنالك أكثر من طريقةٍ، أي خوارزمية، لإنجاز المهمات والعمليات المطلوبة، وتختلف كل خوارزميةٍ بسلبياتٍ وإيجابياتٍ نظراً لاختلافها في تنفيذ العمليات.

انواع الخوارزميات

- ✓ ***الخوارزمية التكرارية Recursive Algorithm***: طريقة تقسم المشكلة إلى مشاكل فرعية أصغر متشابهة وتطبق نفسها بشكل متكرر لحلها حتى الوصول إلى الحالة الأساسية، مما يجعلها فعالة للمهام ذات الهياكل العودية. على سبيل المثال حساب عدد الارقام الزوجية لـ ١٠٠ عدد صحيح، حيث يمكن كتابة الخوارزمية بالطريقة التكرارية كمايلي:

١. البداية

٢. تعريف متغير من النوع العددي (i) و إسناد القيمة 1كقيمة مبدئية له.

٣. تعريف متغير من النوع العددي (stop) يدل على متى تتوقف الحلقة و إسناد القيمة ١٠٠ كقيمة مبدئية له.

٤. عمل حلقة whileعلى المتغير i بشرط i يكون أقل من أو يُساوي stop.

٥. في كل لفة للحلقة نفحص i إذا كان i زوجياً نقوم بطباعته ثم نضيف 1للعداد i.

٦. لفحص عدد ما إذا كان زوجياً نقسم هذا العدد على ٢ و نُقارن باقي قسمته إذا كان يُساوي صفر فهو عدد زوجي.

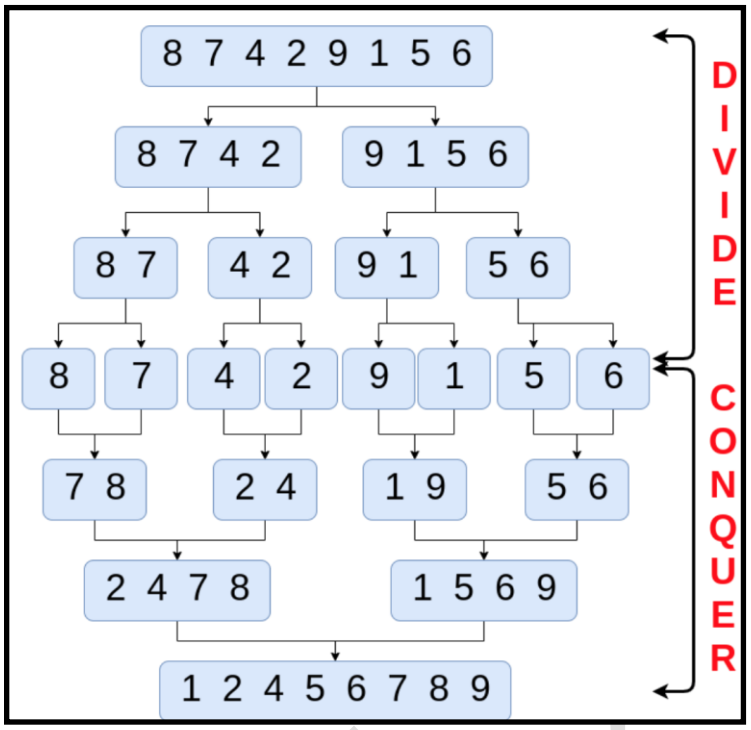
٧. هل وصل stopالى العدد ١٠٠: نعم : اذهب الى 8، لا: ارجع الى الخطوة 5.

٨. طباعة i.

٩. النهاية.

- ✓ ***خوارزمية فرق تسدDivide and Conquer***

هذه طريقة فعالة أخرى لحل العديد من المشكلات. حيث يتم تقسيم الخوارزمية إلى قسمين؛ تقسم الأجزاء الأولى المشكلة المطروحة إلى مشكلات فرعية أصغر من نفس النوع. ثم، في الجزء الثاني، يتم حل هذه المشكلات الصغيرة ثم يتم جمعها معًا لإنتاج الحل النهائي للمشكلة. على سبيل المثال، اعادة ترتيب الارقام بالشكل (١-٢).



ترتيب الأرقام تصاعدياً باستخدام خوارزمية فرق تسد

✓ ***خوارزمية البرمجة الديناميكية Dynamic Programming Algorithm***

تعمل هذه الخوارزميات من خلال تذكر نتائج التشغيل الماضي واستخدامها للعثور على نتائج جديدة. بمعنى آخر، تعمل خوارزمية البرمجة الديناميكية على حل المشكلات المعقدة عن طريق تقسيمها إلى عدة مشكلات فرعية بسيطة ومن ثم تقوم بحل كل واحدة منها مرة واحدة ثم تقوم بتخزينها للإستخدام المستقبلي. على سبيل المثال، عملية حساب أعداد فيبوناتشي بطريقة تكرارية العديد من المسائل الفرعية التي يجري حلّها مرة تلو الأخرى.

Fibonacci(N) = 0 (for n=0)

Fibonacci(N) = 0 (for n=1)

= Fibonacci(N-1)+Finacchi(N-2) (for n>1)

مثال: اكتب خوارزمية لحساب مساحة دائرة (A) لها نصف القطر (r)

١-البداية

٢- اقرأ نصف القطر r.

٣-اقرأ قيمة النسبة الثابتة Pi.

٤-احسب مساحة الدائرة من قانون المساحة ($A=Pi\ r^2$).

٥- اطبع قيمة المساحة A.

٦- النهاية.

مثال: اكتب خوارزمية برنامج يقوم بطباعة معدل ٣ درجات

١. البداية.

٢. أقرا الدرجة الأولى. X

٣. أقرا الدرجة الثانية. Y

٤. أقرا الدرجة الثالثة. Z

٥. حساب قيمة المجموع (Sum) وحسب العلاقة (Sum=X+Y+Z).

٦. حساب المعدل(Average) وحسب العلاقة (Average=Sum/3)

٧. طباعة المعدل.

٨. النهاية.

مثال :اكتب خوارزمية لايجاد اكبر عدد بين ٣ اعداد

١) اقرا a,b,c
٢) اذا كان a>bو a>c اطبع A اذهب ٥
٣) اذا كان b>a و b>c اطبع b و اذهب ٥
٤) أطبع c
٥) انتهى

مثال :اكتب خوارزمية لحساب تقدير الطالب

١) اقرا apprec
٢) اذا كان apprec >=90 و apprec <=100 اطبع "ممتاز"
٣) اذا كان apprec >=80 و apprec <90 اطبع "جيدجدا"
٤) اذا كان apprec >=70 و apprec <80 اطبع "جيد"
٥) اذا كان apprec <=80 و apprec <70 اطبع "متوسط"
٦) اذا كان apprec <=70 و apprec <60 اطبع "مقبول"
٧) انتهى

المخطط الانسابي Flowchart

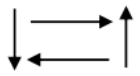
هو تمثيل رسومي لتسلسل العمليات في نظام المعلومات أو البرنامج. توضح المخططات الانسيابية لنظام المعلومات كيفية تدفق البيانات من المستندات المصدر عبر الكمبيوتر إلى التوزيع النهائي للمستخدمين. تُظهر المخططات الانسيابية للبرنامج تسلسل التعليمات في برنامج واحد أو روتين فرعي. يتم استخدام رموز مختلفة لرسم كل نوع من المخططات الانسيابية.

فوائد المخططات الانسيابية:-

يمكن تلخيص الفوائد التي يمكن الحصول عليها من جراء استعمالنا للمخططات الانسيابية كما يلي:

١. توضيح الخطوات: تقدم المخططات صورة واضحة وبسيطة للخطوات المتبعة لحل المشكلة أو تنفيذ البرنامج، مما يسهل فهمها وتنفيذها.
٢. معالجة القرارات: تساعد المخططات في حل المشكلات التي تتضمن قرارات متعددة وبدائل مختلفة، حيث يمكن تمثيل هذه البدائل والتفاصيل بوضوح في المخطط.
٣. تسهيل التعلم: تجعل المخططات من السهل على المستفيدين والدارسين دراسة البرامج أو النظم وفهم كيفية عملها والعلاقات بين العناصر المختلفة.
٤. توثيق البرنامج: تعتبر المخططات جزءًا هامًا من توثيق البرامج، حيث يمكن استخدامها لشرح كيفية عمل البرنامج بوضوح.
٥. مراجعة وتعديل البرنامج: يمكن استخدام المخططات لسهولة مراجعة البرنامج واكتشاف الأخطاء والتعديلات اللازمة، مما يسهل عملية تحسين البرنامج وتصحيح الأخطاء."

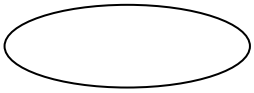
رموز المخطط الانسيابي Flowchart symbol

الشكل أو الرمز	الغاية	شرح لطريقة استخدامه
	بداية / النهاية	إذا وضع في أعلى المخطط فأنة يشير إلى بداية البرنامج أما إذا وضع في أسفل المخطط فانه يمثل نهاية البرنامج
	إدخال / إخراج	يشير إلى عملية إدخال البيانات أو إخراجها (طباعتها على الشاشة مثلا)
	المعالجة	يوضح عملية معالجة أو مجموعة من العمليات من خلال تنفيذ عملية حسابية
	اتخاذ القرار	يستخدم للتعبير عن نقطة اتخاذ القرار مثل عملية مقارنه بين قيمتين على أساسها يتم اتخاذ قرار باتجاه معين
	خط الانسياب	وتمثل اتجاه التدفق المنطقي لحل المسألة
	نقطة الربط	تستخدم عند تجزئة مخطط كبير إلى أجزاء ترتبط عند هذه النقاط التي تحمل الرموز نفسها

رموز المخطط الانسيابي

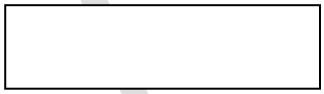
ويمكن تلخيص مجموعة الأشكال كما يلي:-

١. البداية والنهاية: عبارة عن شكل بيضوي تكتب فيه كلمة بداية (Start) أو كلمة نهاية (End) و يشير إلى نقاط البداية و النهاية في المخطط الانسيابي ويكون للمخطط نقطة بداية واحدة فقط و لكن ربما يكون له أكثر من نقطة نهاية ممكنة والشكل هو :

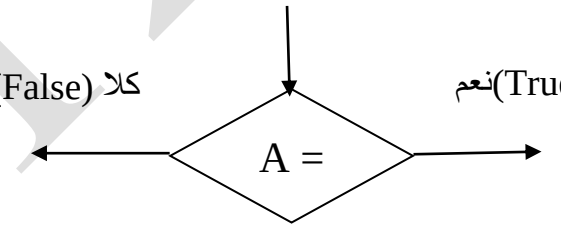


٢. الإدخال / الإخراج: (بدون تحديد وسيلة الإدخال و الإخراج) يستخدم شكل متوازي الأضلاع في تمثيل عملية الإدخال / الإخراج للبيانات أو المعلومات كما في الشكل التالي:-

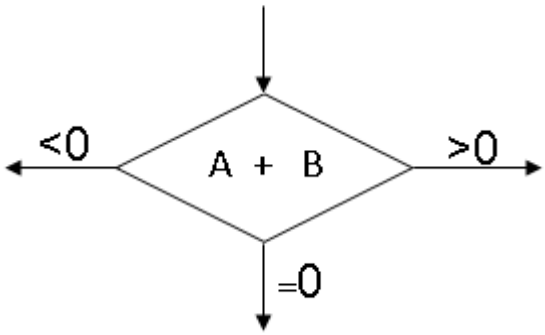
٣. العمليات الحسابية: يستخدم شكل المستطيل لبيان وجود عملية حسابية يراد انجازها و الشكل هو:-



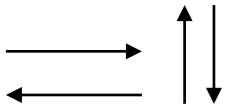
٤. القرار المنطقي: يستخدم الشكل المعيني لبيان اتجاه التسلسل المنطقي لتنفيذ العمليات حسب جواب الشرط المتضمن في عملية منطقية داخل الشكل المعيني وقد يكون للشكل اتجاهين الأول إذا كان جواب الشرط نعم (True) و الثاني إذا كان جواب الشرط كلا (False) كما في الشكل التالي:-



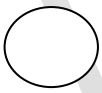
أو يكون له ثلاث اتجاهات تعتمد على قيمة التعبير المستخدم في الشرط مثلا:-



أسهم الاتجاهات: تستخدم الأسهم بالاتجاهات الأربعة لبيان انسياب السيطرة في الخوارزمية و كما في الشكل التالي:



٥. الاتصال: تستخدم الدائرة في وصل نقاط تتحد عندها مسارات مختلفة معا و الشكل المستخدم هو الدائرة:

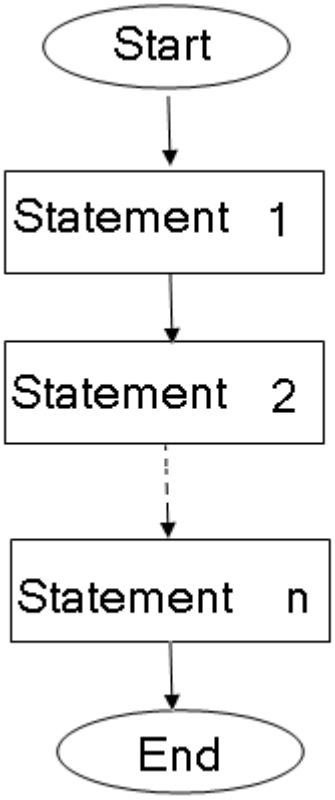


أنواع الخوارزميات لتمثيل المخطط الانسيابي :-

تقسم الخوارزميات الى عدة أنواع هي :-

١. *الخوارزميات المتسلسلة* : Sequential Algorithms

في هذا النوع من الخوارزميات لا نجد أي تفرعات أو تكرارات و إنما تتم كتابة و تنفيذ التعليمات الواحدة تلو الأخرى . و يمكن تمثيلها بمخطط انسيابي بشكل عام بالشكل التالي:-



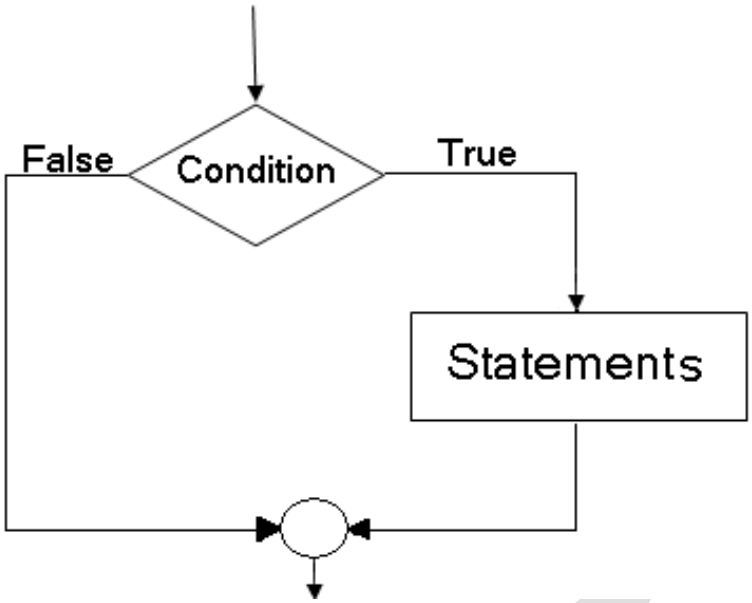
٢. *الخوارزميات الشرطية :* Conditional Algorithms

هي الخوارزميات التي تحتوي واحدة أو أكثر من خطواتها على أوامر شرط (اتخاذ قرار) و تكون على نوعين:-

أ- خوارزميات شرط ذات مسار واحد One way selection وتكتب بالشكل التالي:-

إذا شرط إذن خطوة (أو مجموعة خطوات)

و يمكن تمثيلها بمخطط انسيابي (كجزء من خوارزمية متكاملة) بالشكل التالي:-

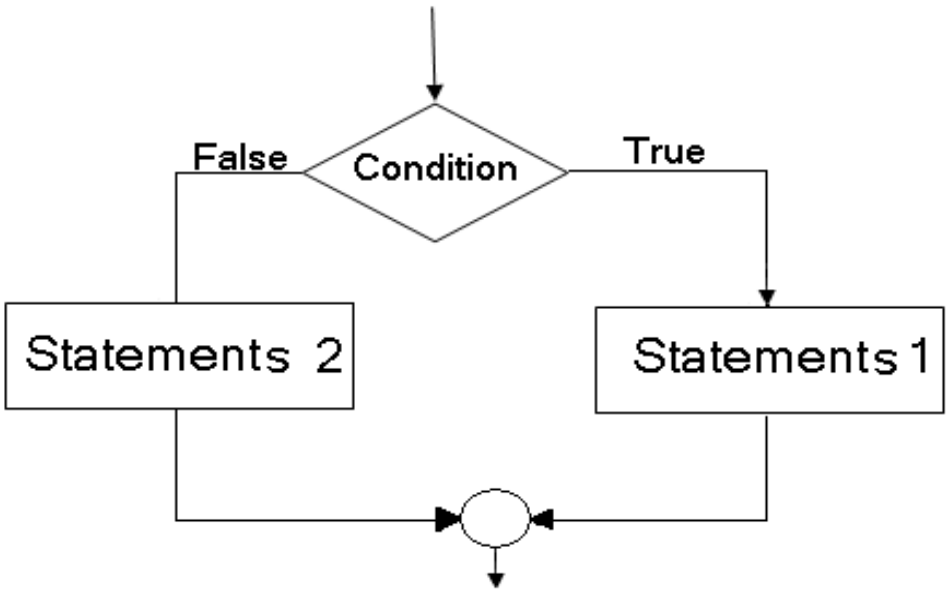


ب- خوارزميات شرط ذات مسارين Two way selection

إذا شرط إذن خطوة ١ (أو مجموعة خطوات)

و إلا خطوة ٢ (أو مجموعة خطوات)

و يمكن تمثيلها بمخطط انسيابي (كجزء من خوارزمية متكاملة) بالشكل التالي:-



٣. *خوارزميات التكرار*: Iteration Algorithms

في هذا النوع من الخوارزميات، يتم تنفيذ مجموعة من التعليمات عدة مرات، وهناك نوعان رئيسيان من هذا النوع:

١. التكرار الثابت (Fixed Iteration): في هذا النوع من الخوارزميات، يتم تكرار مجموعة من التعليمات لعدد محدد مسبقًا من المرات. على سبيل المثال، إذا كنت ترغب في تنفيذ نفس العملية ١٠ مرات دون تغيير، يمكنك استخدام التكرار الثابت.
٢. التكرار المشروط (Conditional Iteration): في هذا النوع، يتم تكرار مجموعة من التعليمات حسب شرط معين. يمكن للتكرار المشروط أن يكون معلقًا على شرط معين، وعندما يتم تحقيق الشرط، يتوقف التكرار. على سبيل المثال، يمكنك استخدام التكرار المشروط لتكرار عملية ما حتى يتم الانتهاء من قائمة معينة أو حتى يتم تحقيق شرط معين، مثل البحث عن عنصر معين في قائمة.

هذه الخوارزميات تُستخدم بشكل واسع في البرمجة لتنظيم التكرار وتنفيذ الأوامر بناءً على الحاجة والشروط المحددة.

مراحل كتابة الخوارزمية

١. **تصميم الخوارزمية** : Algorithm Design

قبل البدء بكتابة البرنامج يجب التفكير في التصميم العام للخوارزمية و ما هو المطلوب للوصول الى الحل الأمثل للمشكلة المطلوب حلها.

٢. **وضع المخطط الانسيابي للخوارزمية** :- Flow Chart

اي رسم الخوارزمية باستخدام الاشكال المناسبة في مخطط انسيابي للتمكن من تتبع سير العمليات في الخوارزمية و سهولة اكتشاف الاخطاء.

٣. **اختيار لغة البرمجة** : Programming Language

و هي خطوة مهمة جدا , حيث ان اختيار لغة البرمجة المناسبة يساعد في الحصول على النتائج المرغوبة بدقة, اذ تختلف امكانيات و اغراض لغات البرمجة , وان قرار اختيار لغة البرمجة يتخذ من قبل محلل الانظمة.

٤. **كتابة البرنامج** : Program Writing

تلي عملية اختيار لغة البرمجة كتابة البرنامج على وسط ادخال مناسب و غالبا ما تكون مهمة كتابة البرنامج من مهام المبرمج . ويعتمد المبرمج في كتابة البرنامج على المخطط الانسيابي الذي يعده محلل الانظمة او المبرمج نفسه.

أنواع الأخطاءالبرمجية:

هناك ثلاثة أنواع من الأخطاء البرمجية وهي:

- **أخطاء هجائية:** Syntax Errors

وهو خطأ في سوء كتابة جملة معينة في البرنامج، على سبيل المثال نجد هناك كلمات محجوزة التي يجب كتابتها كما هي دون تغيير, يمكن أن يخطأ المبرمج في كتابتها، فيكتشفها البرنامج على الفور ، وهذه الأخطاء تظهر أثناء كتابة البرنامج ، أو بمعني أدق كتابة الكلمة أو الجملة في صورة غير صحيحة.

- **أخطاء أثناء التنفيذ:** Runtime Errors

هذا الخطأ لا يظهر إلا بعد ترجمة البرنامج وتنفيذه، لأن لغة البرمجة لا تلاحظها إلا بعد عملية التنفيذ ، علي سبيل المثال إذا قمت بقسمة أي عدد على صفر :

Num = 50 / 0

هذه العملية تقوم بقسمة ٥٠ علي ٠ ، ونحن نعلم جميعاً أن قسمة أي رقم علي صفر تعطي رقم غير نهائي ، وهذا لا يجوز .

- **أخطاء منطقية:** Semantic Errors

هذا النوع يعتبر أخطر نوع من الأخطاء، حيث أن لغة البرمجة لا ولن تكتشفها بعد، لأن هذا الخطأ خطأ في المعالجة الحسابية

مثل كتابة القانون الخاصة بحساب مساحة المثلث لحساب مساحة المربع

٥. **التنفيذ والاختبار :** Running & Testing

***مزايا البرنامج الجيد*:-**

ان اهم الخصائص النوعية الجيدة التي يجب توافرها في البرنامج هي كالاتي:-

١. **البرنامج يؤدي العمل**: يجب ان يؤدي البرنامج العمل المطلوب منه وفق المواصفات المعرفة لوظيفته و مدخلاته و مخرجاته و العمليات التي يؤديها.

٢. **البرنامج قابل للقراءة والفهم** : يجب ان يكون البرنامج المصمم واضح القراءة و بصياغة تسهل فهم خطواته و وظائفه من قبل الاخرين. ويجب اعتماد خصائص البرمجة المهيكلة التي تؤدي الى استخدام افضل لوقت المبرمج و كتابة برنامج يحقق الاستخدام الامثل لموارد منظومة الحاسب خصوصا ما يتعلق بالمساحة الخزنية و وقت التنفيذ.

٣. **البرنامج قابل للتطوير**: يجب تصميم البرنامج بطريقة يمكن تطويره لاحقا او تعديله بسهولة اذا تطلب الامر ذلك بجهود مقبولة و وقت مناسب, اذ ان الضرورة قد تقتضي تعديل البرنامج في مرحلة التصميم و الاختبار.

٤. **انجاز البرنامج**: يجب ان يعد البرنامج بصورته المتكاملة و بمراحله المختلفة ضمن الفترة الزمنية و التخصيص المالي المحددين له واي تجاوز على ذلك سيفقد جدواه الاقتصادية.

مثال :اكتب خوارزمية ومخطط انسيابي لفحص رقمين وتحددين زوجي او فردي

١- ادخل الرقم

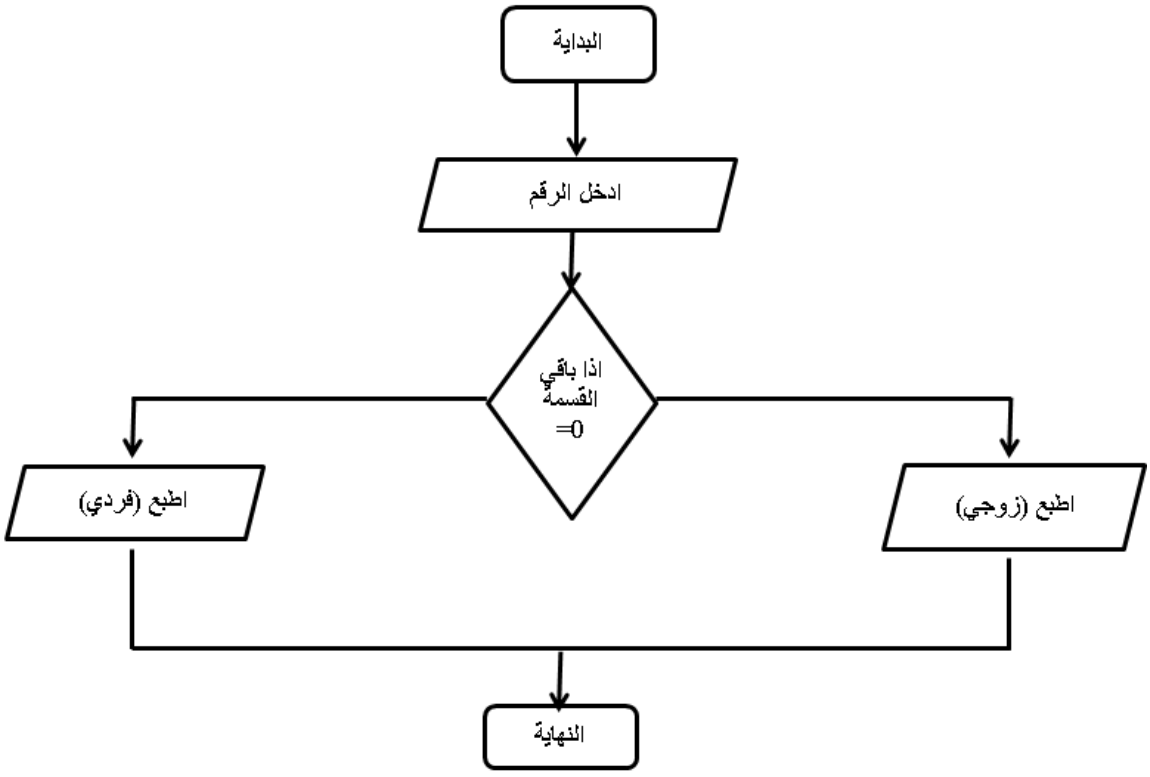
٢- اذا كان باقي قسمة الرقم = ٠ اذهب الى الخطوة ٣

٣- اطبع (الرقم زوجي)

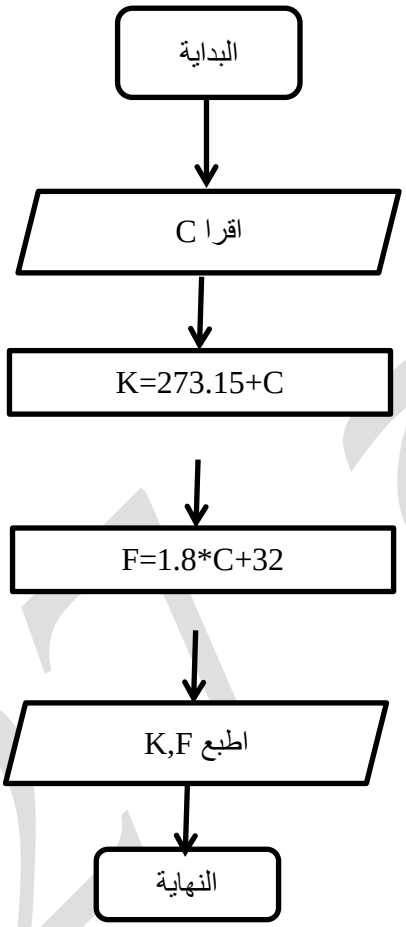
٤- اذا كان باقي القسمة لا يساوي صفر اذهب الى الخطوة ٥

٥- اطبع (الرقم فردي)

٦- النهاية



مثال : ارسم مخطط انسيابي لبرنامج يقوم بقراءة درجة الحرارة (بالمقياس المئوي C) وحساب قيمتها وطباعتها بمقياس الكلفن K و مقياس فهرنهايت F.



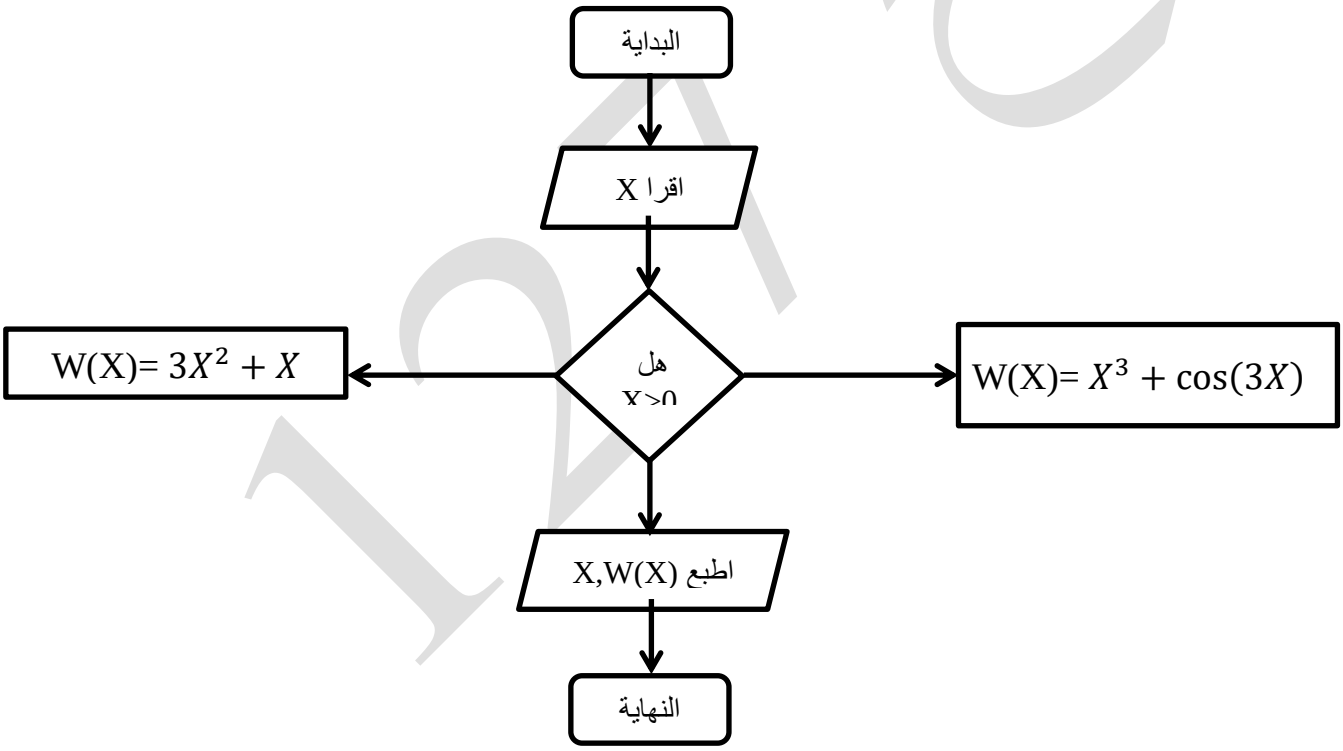
مثال : خوارزمية لحساب تباين قيمتين عن معدلهما علما ان التباين هو ناتج طرح القيمة من المعدل؟
٤ و ٢ فالمعدل ٣ والتباين ١ والثانية -١

١) اقرأ x,y
٢) a=(x+y)/2
٣) b1=x-a
٤) b2=y-a
٥) اطبع b1,b2
٦) انتهى

مثال :اكتب خوارزمية وارسم المخطط الانسيابي برنامج مخصص لقراءة المتغير (X) ثم حساب قيمة الدالة (W(X)) تبعا لقيمة (X) من خلال ما يأتي :

$$W(X) = \begin{cases} X^3 + \cos(3X) & X > 0 \\ 3X^2 + X & X \leq 0 \end{cases}$$

١. البداية
٢. اقرأ(X)
٣. قارن قيمة (X) بالصفر
٤. اذا كانت موجبة اذهب الى الخطوة (٥) والا اذهب الى الخطوة (٦)
٥. احسب قيمة W(X) من المعادلة $W(X)= X^3 + \cos(3X)$ ثم اذهب الى الخطوة (٧)
٦. احسب قيمة W(X) من المعادلة $W(X)= 3X^2 + X$
٧. اطبع قيمة X,W(X)
٨. النهاية



مفهوم البرمجة:

يقصد بها تخصيص نظام عملي معين بقوانين وضوابط معينه في إطار مخصص به، فمثلاً الهاتف وهو جهاز تمت برمجته ليعمل بنظام معين، وفي إطار ضوابط مخصصة له.

البرمجة او البرمجيات التقنية : ويقصد بها إنشاء نظام عمل ومحدد بإحدى لغات البرمجة التي تفهمها الالة لتقوم بوظيفة معينة.

وتتبع عملية البرمجة قواعد خاصة باللغة التي اختارها المبرمج وكل لغة برمجة لها خصائصها التي تميزها عن الأخرى وتجعلها مناسبة بدرجات متفاوتة لكل نوع من أنواع البرامج وحسب المهمة المطلوبة من هذا البرنامج .كما أن اللغات البرمجية أيضاً لها خصائص مشتركة وحدود مشتركة بحكم أن كل هذه اللغات صممت للتعامل مع الحاسوب .

تدخل البرمجة في التكنولوجيا الخاصة بصناعة الروبوتات الذكية وتطوير الذكاء الاصطناعي وتطوير تطبيقات الهواتف الذكية, ففي هاتفك الأندرويد أو الـ ios ستجد التطبيقات التي تستخدمها بشكل يومي تم الاعتماد علي البرمجة في تطويرها وتم بناء نظام التشغيل " ويندوز " علي سبيل المثال باستخدام إحدى اللغات البرمجية وهي لغة " c++ " وبالتالي لولا وجود البرمجة لما استطعنا الوصول إلي هذه التقنية.

كذلك البرامج المختلفة التي تعمل على نظام ويندوز أو نظام ماك هذه البرامج كلها تم تكويدها بإحدى اللغات البرمجية لذلك أصبح بإمكاننا الحصول على برامج تسهل علينا أداء الوظائف والعمل باستخدام البرمجة.

ويمكن مقارنة اللغات البرمجية باللغة الانجليزية مثلاً او اي لغة غير لغتك الأم ، فلا يستطيع الشخص التعامل مع أجنبي بدون تعلم لغته، وبالمثل لا يمكن لألة ولا للحاسوب ان يفهمنا دون لغة مفهومة لكلينا لذلك فلابد من وجود البرمجة حتى نقوم ببرمجة الآلات.

يتم إنشاء البرامج من خلال لغات البرمجة للتحكم في سلوك ومخرجات الآلة من خلال خوارزميات دقيقة، تشبه عملية الاتصال البشري

- لغة الآلة

تسمى أيضًا رمز الآلة، هي لغة كمبيوتر منخفضة المستوى تم تصميمها بحيث يمكن للكمبيوتر فهمها مباشرة وهي اللغة التي يجب تحويل جميع البرامج إليها قبل أن يتم تشغيلها. وهي تتألف بالكامل من ثنائي، ٠ و ١. في لغة الآلة، يتم تمثيل جميع التعليمات ومواقع الذاكرة والأرقام والحروف بالصفر والواحد. على سبيل المثال، قد تبدو قطعة نموذجية من لغة الآلة مثل، ١٠٠٠٠٠٠٠ ٠٠٠٠٠١٠٠.

- لغة التجميع

من الصعب للغاية على البشر قراءة لغة الآلة لأنها تتكون فقط من أنماط من البتات (أي ٠ و١). وبالتالي، فإن المبرمجين الذين يرغبون في العمل على مستوى لغة الآلة بدلاً من ذلك عادة ما يستخدمون لغة التجميع، وهي عبارة عن تدوين يمكن قراءته بواسطة الإنسان للغة الآلة. تستبدل لغة التجميع التعليمات

التي تمثلها أنماط ٠ و ١ برموز أبجدية رقمية تسمى أيضًا أساليب تقوية الذاكرة لتسهيل تذكرها والعمل معها بما في ذلك تقليل فرص ارتكاب الأخطاء. على سبيل المثال، رمز إجراء عمليات الجمع والطرح هو،

ADD 3, 5, result

SUB 1, 2, result

- لغة عالية المستوى

اللغة عالية المستوى تشبه لغة الإنسان وأقل شبهاً بلغة الآلة. تتم كتابة اللغات عالية المستوى بشكل قريب من لغتنا البشرية، مما يتيح للمبرمجين التركيز فقط على المشكلة التي يتم حلها. اللغات عالية المستوى مستقلة عن النظام الأساسي مما يعني أنه يمكن تنفيذ البرامج المكتوبة بلغة عالية المستوى على أنواع مختلفة من الأجهزة. يُطلق على البرنامج المكتوب بلغة عالية المستوى اسم البرنامج المصدر أو الكود المصدري، وهو عبارة عن مجموعة من تعليمات الكمبيوتر التي يمكن للإنسان قراءتها. ومع ذلك، لكي يتمكن الكمبيوتر من فهم وتنفيذ برنامج مصدر مكتوب بلغة عالية المستوى، يجب ترجمته إلى لغة الآلة. تتم هذه الترجمة باستخدام إما مترجم أو مفسر.

تطوير البرمجيات

تطوير البرمجيات هو عملية يتم من خلالها إنشاء برامج مستقلة أو فردية باستخدام لغة برمجة معينة. يتضمن كتابة سلسلة من أكواد البرمجة المترابطة، والتي توفر وظائف البرنامج المطور. قد يُطلق على تطوير البرمجيات أيضًا اسم تطوير التطبيقات.

تمر عملية تطوير البرمجيات بسلسلة من المراحل بطريقة تدريجية تعرف باسم دورة حياة تطوير البرمجيات (SDLC). إنه أسلوب منهجي لتطوير البرمجيات (الشكل ١.١). إنه ينشئ هيكلًا للمطور لتصميم وإنشاء وتقديم برامج عالية الجودة وفقًا لمتطلبات العميل. كما يوفر منهجية لتحسين جودة المنتج المطلوب.

الشكل (١-١) المراحل المختلفة لدورة حياة تطوير البرمجيات

لغة بايثون PYTHON

انطلقت لغة البايثون في بداية التسعينيات على يد Guido van Rossum وأطلق عليها هذا الأسم بسبب اعجابه بفرقة مسرحية شهيرة في بريطانيا كانت تُسمى مونتي بايثون. يُمكن استخدام لغة البرمجة بايثون في بناء وتطوير البرامج والتطبيقات الصغيرة والمتوسطة والضخمة، ويُنصح بتعلمها في البداية لكل شخص يرغب بتعلم البرمجة بشكل عام، وذلك لأنها سهلة التعلم.

استخدامات لغة بايثون

- إنشاء تطبيقات الويب ومهام سير العمل
- التعامل مع البيانات الضخمة وأداء الرياضيات المعقدة
- استخدامها للنماذج الاولية السريعة او لتطوير البرامج الجاهزة للانتاج
- يمكن لـ Python الاتصال بأنظمة قواعد البيانات وقراءة الملفات وتعديلها
- إملاء النماذج بشكل تلقائي عبر الانترنت
- تحديث الملفات النصية الى جداول بيانات

لماذا بايثون؟

- تعمل لغة Python على منصات مختلفة (Windows ، وMac، وLinux، وRaspberry P) وما إلى ذلك.
- لدى Python بناء جملة بسيط مشابه للغة الإنجليزية.
- لدى بايثون بناء جملة يسمح للمطورين بكتابة برامج ذات أسطر أقل من بعض لغات البرمجة الأخرى.
- تعمل بايثون على نظام مترجم فوري، مما يعني أنه يمكن تنفيذ التعليمات البرمجية بمجرد كتابتها. وهذا يعني أن النماذج الأولية يمكن أن تكون سريعة جدًا.
- يمكن التعامل مع بايثون بطريقة إجرائية، أو بطريقة موجهة للكائنات، أو بطريقة وظيفية.

تهيئة بيئة تطوير برامج بلغة بايثون

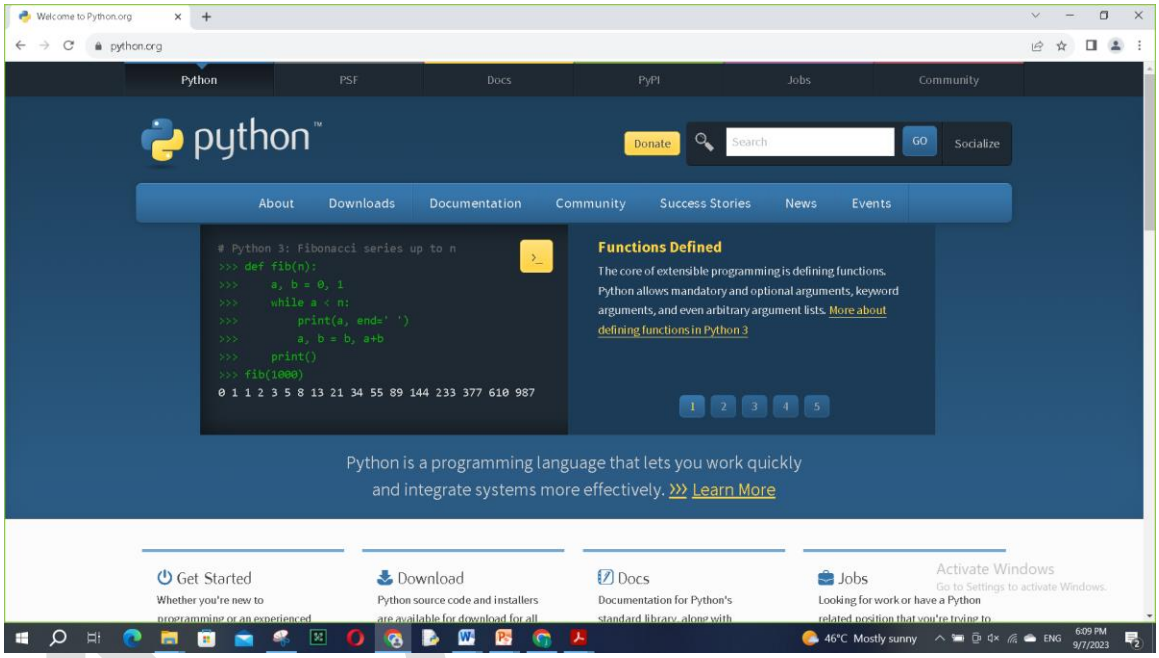
هناك العديد من البيئات التي تعمل عليها لغة بايثون منها ما يتم تنصيبه على الحاسبة ومنها ما يتم العمل عليها من خلال شبكة الانترنيت. ان انشاء أي ملف في بايثون سيكون بامتداد (py.).

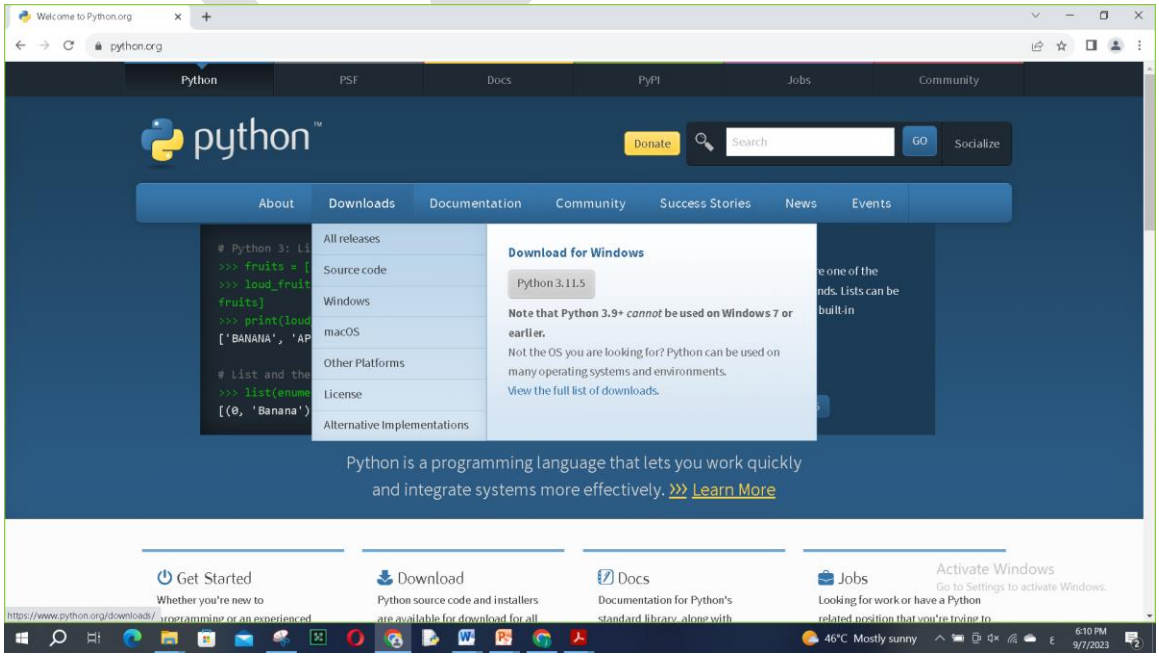
- Anaconda Python
- PyCharm IDE
- IDLE
- Jupyter Notebook
- Kaggle

تثبيت لغة بايثون

بالإمكان استخدام أي برنامج للعمل على برامج البايثون, في هذا المنهج سوف نستخدم برنامج IDLE ولتنصيب البرنامج نتبع الخطوات التالية :

- اذهب الى موقع python.org
- اضغط على downloads وستظهر لنا قائمة نضغط على زر التحميل



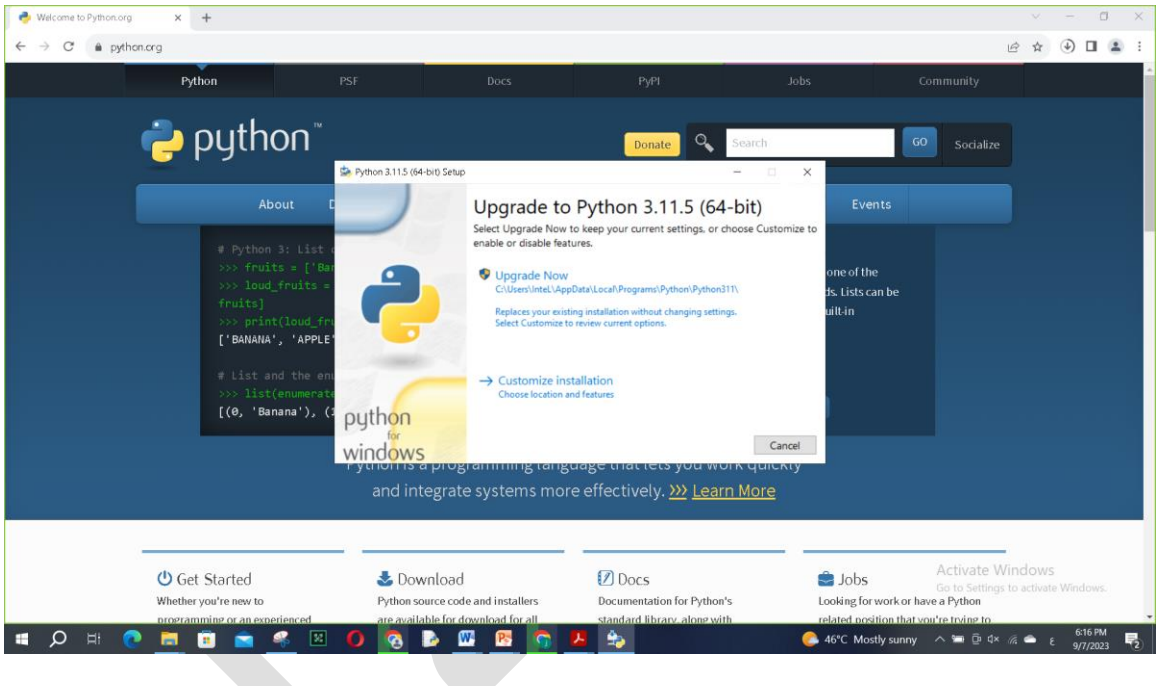


❖ بعد ذلك سيبدأ تحميل ملف التثبت بامتداد exe. احفظ الملف في المكان الذي تريده ثم شغله بالضغط عليه مرتين.

❖ ستظهر شاشة التحميل, تأكد من اختيار add python 3.11 to path وذلك لتتمكن من تشغيل البايثون من شاشة الأوامر, ثم اضغط على خيار التثبيت الان Install Now.

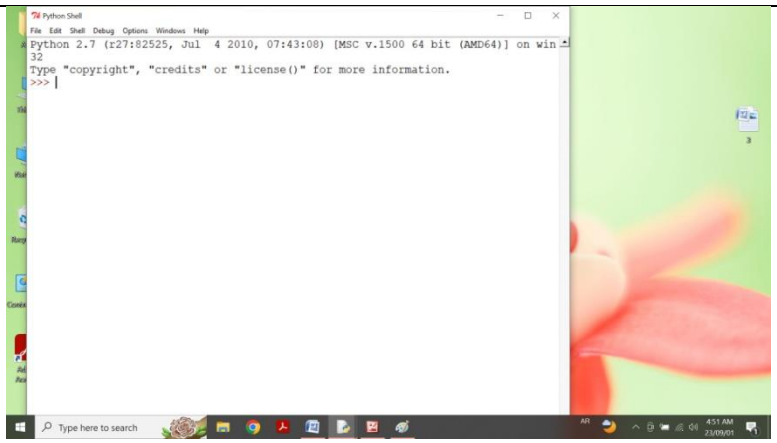
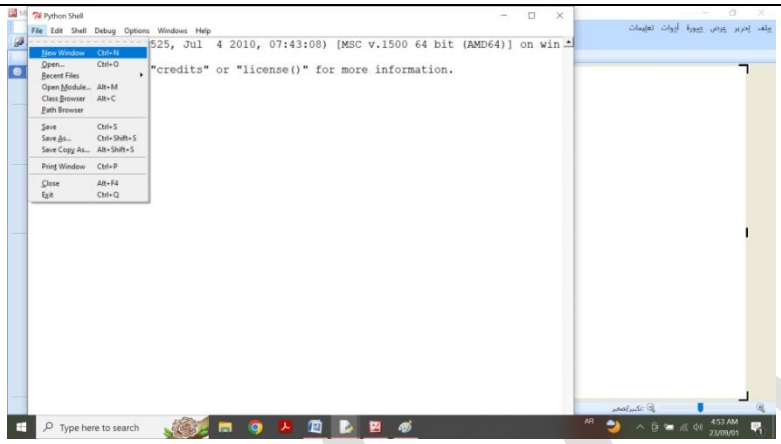
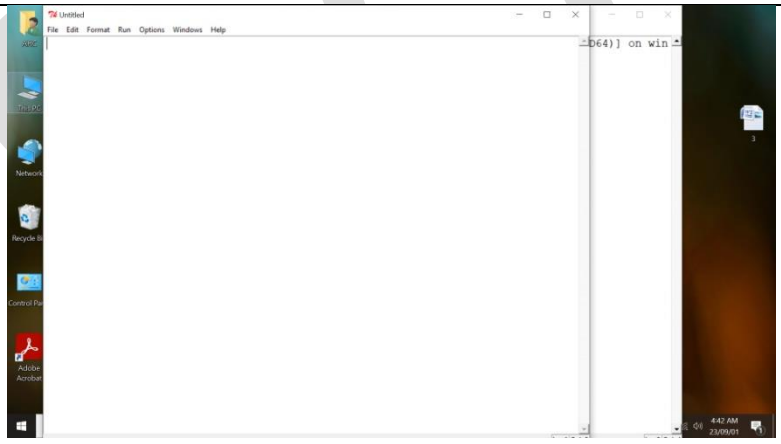
ملاحظة : النافذة التالية ظهر فيها خيار upgrade to python بسبب وجود نسخة سابقة تم تنصيبها على الجهاز

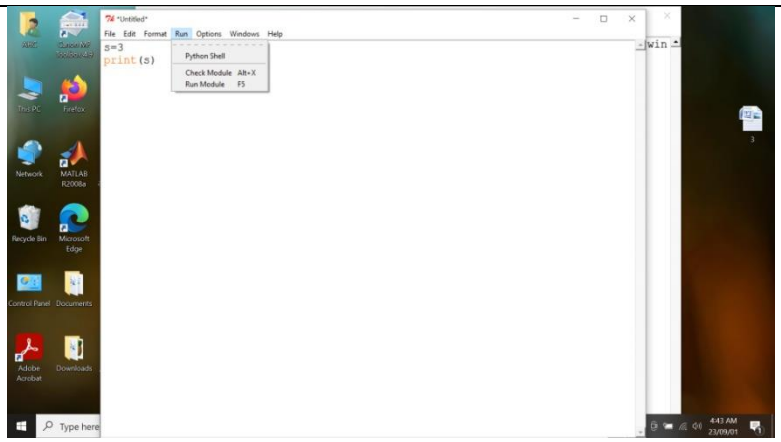
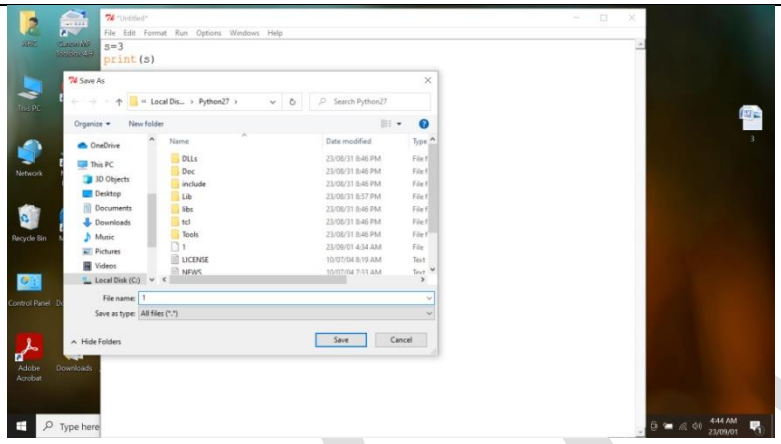
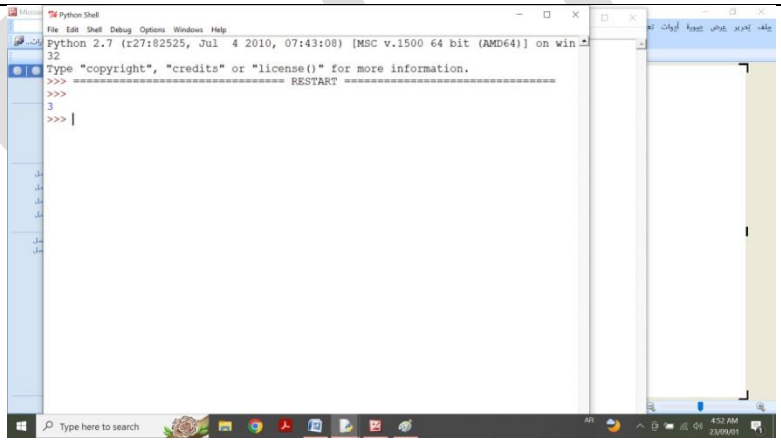
❖ بعد الانتهاء من التثبيت سيكون البرنامج جاهز لكتابة الاوامر



تشغيل برنامج بايثون

بعد اكتمال تنصيب البرنامج نذهب الى قائمة start نكتب في خانة البحث idle لتظهر لنا ضمن النتائج الـ IDLE Python GUI فنقوم بفتحها لتظهر الواجهة الرسومية لتطوير وتطبيق البرامج وكما في الصور الموضحة لاحقاً والتي تبين اننا ممكن البدء بكتابة البرامج بعد ظهور علامات الحث >>> او من خلال فتح نافذة جديدة لكتابة البرامج ومن ثم تنفيذها من خلال الايعاز Run والذي يتطلب خزن البرنامج او المودل وتسميته قبل التنفيذ.

 Python Shell File Edit Shell Debug Options Windows Help Python 2.7 (r27:82525, Jul 4 2010, 07:43:08) [MSC v.1500 64 bit (AMD64)] on win 32 Type "copyright", "credits" or "license()" for more information. >>> Type here to search 4:51 AM 23/09/01	١
 Python Shell File Edit Shell Debug Options Windows Help New Window Ctrl+N Open... Ctrl+O Recent Files Open Module... Alt+M Class Browser Alt+C Path Browser Save Ctrl+S Save As... Ctrl+Shift+S Save Copy As... Alt+Shift+S Print Window Ctrl+P Close Alt+F4 Exit Ctrl+Q 525, Jul 4 2010, 07:43:08) [MSC v.1500 64 bit (AMD64)] on win "credits" or "license()" for more information. Type here to search 4:53 AM 23/09/01	٢
 Untitled File Edit Format Run Options Windows Help D64)] on win 4:42 AM 23/09/01	٣

	 "Untitled" File Edit Format Run Options Windows Help s=3 print(s) Python Shell Check Module Alt+X Run Module F5	٤
	 "Untitled" File Edit Format Run Options Windows Help s=3 print(s) Save As Local Dis... > Python27 > Search Python27 Organize New folder OneDrive This PC 3D Objects Desktop Documents Downloads Music Pictures Videos Local Disk (C:) Name Date modified DLLs Doc include Lib libs tcl Tools 1 LICENSE File name: 1 Save as type: All files (*.*) Hide Folders Save Cancel	٥
	 Python Shell File Edit Shell Debug Options Windows Help Python 2.7 (r27:82525, Jul 4 2010, 07:43:08) [MSC v.1500 64 bit (AMD64)] on win 32 Type "copyright", "credits" or "license()" for more information. >>> ========== RESTART ========== >>> 3 >>> Type here to search	٦

إنشاء برنامج بايثون

Python هي لغة برمجة مترجمة (interpreted), وهذا يعني انه عند كتابة ملفات مثل(filename.py) في محرر النصوص ثم وضعها في مترجم بايثون ليتم تنفيذها.

لنكتب اول تعليمة في Python والتي ستقوم بطباعة جملة (HELLO WORLD) كالتالي:

```
>> print('HELLO WORLD')
```

لمخرجات يجب ان تكون كالتالي: HELLO WORLD

المتغيرات في بايثون

المتغير بشكل عام هو موقع ذاكرة حيث يمكن للمبرمج تخزين قيمة ونلاحظ انه لايوجد إعلان مطلوب ويمكن ان تكون قيمة المتغير ارقام او نصوص او قيم منطقية ويمكن الوصول الى القيمة المخزنة في متغير او تحديثها لاحقاً ويتم تحديد نوع المتغير من قبل Python .

تسمية واستخدام المتغيرات

عندما تستخدم المتغيرات في بايثون، فإنك تحتاج إلى الالتزام ببعض القواعد

١. يمكن أن تحتوي أسماء المتغيرات على أحرف وأرقام وشرطات سفلية فقط ويمكن أن تبدأ بحرف أو شارحة سفلية، ولكن ليس برقم. على سبيل المثال، يمكنك استدعاء متغير (message_1) وليس (1-message).
٢. غير مسموح بالمسافات في أسماء المتغيرات، ولكن يمكن استخدام الشارحة السفلية لفصل الكلمات في أسماء متغيرة على سبيل المثال، تعمل Greeting_message
٣. تجنب استخدام الكلمات الأساسية وأسماء الوظائف كأسماء المتغيرات على سبيل المثال، لا تستخدم الكلمة print كأسم متغير.
٤. يجب أن تكون أسماء المتغيرات قصيرة ولكنها وصفية على سبيل المثال، name هو أفضل من n، student_name أفضل من s_n .
٥. كن حذرًا عند استخدام الحرف الصغير l والحرف الكبير O لأنه من الممكن أن يتم الخلط بينهم وبين الرقمين ١ و٠.
٦. اسماء المتغيرات حساسة لحالة الاحرف (مثلاً name , Name متغيرين مختلفين).

ملاحظة: يجب أن تكون متغيرات Python التي تستخدمها في هذه المرحلة بأحرف صغيرة، لن تحصل أخطاء إذا كنت تستخدم أحرفاً كبيرة، ولكن الأحرف الكبيرة في أسماء المتغيرات لها خصائص خاصة.

الكلمات المحجوزة في بايثون

جميع الكلمات التالية محجوزة للغة بايثون, أي لا يمكن إستخدامها كـ.Identifiers

and	else	In	raise
assert	exceptexec	Islambda	return
break	False	None	Truetry
class	finally	Nonlocal	while
continuedef	fromglobal	Not	with
del	if	orpass	yield
elif	import	print	

مثال....مقطع من ايعازات لغة بايثون

```
>>> name="SARA"تعيين قيمة لمتغير
>>> print(name) استخدام دالة للطباعة
SARA الناتج
>>> type(name) استخدام دالة لمعرفة نوع المتغير
<class 'str'> الناتج
>>> age=20
>>> print(age)
20
>>> type(age)
<class 'int'>
```

تعيين قيم لمتغيرات متعددة:

تسمح لغة بايثون بتعيين قيم لمتغيرات متعددة في سطر واحد كالمثال التالي:

```
x, y, z=1 , 2 ,3
```

ويمكن ايضاً تعيين قيمة لمتغيرات عدة في نفس الوقت, مثلاً:

```
x=y=z=7
```

الثوابت في بايثون:-

أ)الثوابت العددية

تدعم بايثون ثلاثة انواع رقمية مميزة:

- ✓ Integers الأرقام الصحيحة
- ✓ Floating point numbers الأرقام العائمة
- ✓ Complex numbers الأقام المركبة

✓ **الأعداد الصحيحة (Int , Integers):**

هي عدد صحيح موجب او سالب بدون كسور عشرية بطول غير محدود.

شرح لإيعازات اللغة	مثال....مقطع من ايعازات لغة بايثون
دالة: هو أيعاز يمكن للغة تنفيذه ()print: هو أيعاز يقوم بالطباعة ()type:هو أيعاز لإيجاد نوع	x=-20 z=25648697896 print(type(x)) <class 'int'> print(type(z)) <class 'int'>

✓ **الاعداد العائمة (float):**

هي عدد موجب اوسالب يحتوي على رقم عشري واحد او اكثر ويمكن ان يكون ايضاً رقم علمي بحرف e للاشارة الى الاس.

مثال...مقطع من ايعازات لغة بايثون
a=2.20 b=-5.5e4 print(type(a)) <class 'float'> print(type(b)) <class 'float'>

✓ **الاعداد المركبة (complex):**

هي اعداد بها جزء تخيلي ويتم كتابتها بحرف j.

y=4+6j نوع العدد هو مركب

complex(y)

تحويل البيانات

يمكن تحويل البيانات من نوع الى آخر باستخدام الدوال int(), float(), complex() كالمثال التالي:

```
>>> x=1
>>> type(x)
<class 'int'>    الناتج
>>> b=float(x)
>>> type(b)
<class 'float'>  الناتج
```

ملاحظة..لايمكن تحويل العدد المركب الى نوع آخر.

ب)الثوابت الرمزية

string عبارة عن سلسلة من الأحرف وفي بايثون اي شيء بداخل علامات الاقتباس عبارة عن string وتكون أما علامات اقتباس مفردة او علامات أقتباس مزدوجة, كالمثال التالي:

شرح لإيعازات اللغة	مثال....مقطع من ايعازات لغة بايثون

١.استخدام محدد تنسيق %s هنا لاخبار بايثون بمكان استبدال قيمة الاسم الممثلة كـstring
٢.استخدام علامات الحث <<< تكون موجودة قبل كل ايعاز في البرنامج نفسه دون الحاجة الى كتابتها
	>>> name="Hamed"
	>>> type(name)
	<class 'str'> الناتج
	>>> print('Hello,%s' %name)
	Hello,Hamed الناتج

جـ)الثوابت المنطقية

تمثل القيم المنطقية Boolean إحدى القيمتين (True او False) وفي البرمجة غالباً ما نحتاج إلى معرفة اذا كان التعبير البرمجي صحيح او خطأ من الناحية المنطقية، وكالمثال التالي:

شرح لإيعازات اللغة	مثال....مقطع من ايعازات لغة بايثون

علامات المقارنة =<,=>,>,< يمكن إيجاد القيم المنطقية بها علماً إن في حالة المحاولة لمعرفة تساوي قيمتين يتم استخدام الـ == لغرض المقارنة	>>> print(20<40)
	الناتج True
	>>> print(40==20)
	الناتج False
	>>> print(20<=40)
	الناتج True

العمليات العددية في بايثون

تدعم بايثون العمليات الحسابية المعروفة وتمتلك جميع العمليات الحسابية اولوية أعلى من عمليات المقارنة وتكون الاولوية هي نفس قواعد الرياضيات وهي مرتبة من الأعلى إلى الاسفل وإذا كان هناك أكثر من عاملين لهما نفس الأسبقية يتم التقييم من اليسار إلى اليمين:

١) ما بين القوسين
٢) الأسس
٣) الضرب والقسمة لهما نفس الاسبقية
٤) الجمع والطرح

العملية	النتيجة	الملاحظات

x+y	إضافة x الى y	>>> x,y=2,5
		>>> x+y
		7

x-y	طرح y من x	>>> x,y=2,5
		>>> x-y
		-3

x*y	ضرب x في y	>>> x,y=2,5
		>>> x*y
		10

x/y	قسمة x على y	>>> x,y=15,3
		>>> x/y
		5.0

x//y	الحاصل التقريبي لقسمة x على y	تكون نتيجة القسمة عدداً صحيحاً كاملاً ولكن ليس بالضرورة ان يكون الناتج من نوع عدد صحيح فمثلاً
		1//2=0
		(-1)//2=-1
		1//(-2)=-1
		(-1)//(-2)=0

x%y	الباقي من قسمة xعلى y	>>> x,y=15,2
		>>> x%y
		1
		ايمكن استخدامها مع الاعداد المركبة

-x	القيمة السالبة لـx	>>> x=10
		>>> -x
		-10

+x	لاتتغير قيمة x	>>> x=10
		>>> +x
		10

abs(x)	القيمة المطلقة	>>> x=-10
		>>> abs(x)
		10

complex(re,im)	انشاء عدد تخيلي يكون الجزء الحقيقي فيه re والجزء التخيلي فيه هو im يأخذ الجزء التخيلي القيمة الافتراضية 0	complex(5,6) (5+6j)

divmod(x, y)	العمليتان(x//y, x%y) معاً	>>> x,y=10,3
		>>> divmod(x,y)
		(3,1)
		لايمكن استخدامها مع الاعداد المركبة

pow(x,y)	رفع x الى الاس y	>>> x,y=10,4
		>>> pow(x,y)
		10000

x**y	رفع x الى الاس y	نتيجة التعبيرين 0**0 و pow(0, 0) هي 1 في بايثون كما هو الحال في لغات البرمجة الاخرى.
		>>> x,y=5,4
		>>> x**y
		625

العوامل في بايثون

➢ عوامل الموازنة Comparison Operators

تُستخدم تلك العوامل للموازنة بين القيم والمتغيرات، إذ يكون الأمر أشبه بطرح سؤال ما على لغة بايثون مثل هل قيمة المتغير كذا تساوي قيمة المتغير كذا؟ أو هل قيمة المتغير كذا لا تساوي قيمة المتغير كذا؟ فتلك الأسئلة من الضروري سؤالها للغة في كثير من البرامج لأنه بناءً على الإجابات سيُتَّخذ قرار بتنفيذ كتلة شيفرة ما، كما أنّ عوامل الموازنة تتشابه في كثير من لغات البرمجة وتكون في بايثون كما يلي:

العامل	الاستخدام	مثال
==	هل القيمة التي على يسار العامل تساوي القيمة التي على يمينه؟	<<<x,y=6,7 <<<x==y False
=!	هل القيمة التي على يسار العامل لا تساوي القيمة التي على يمينه	<<<x,y=6,7 <<<x!=y True
>	هل القيمة التي على يسار العامل أكبر من القيمة التي على يمينه	<<<x,y=6,7 <<<x>y False
<	هل القيمة التي على يسار العامل أصغر من القيمة التي على يمينه	<<<x,y=6,7 <<<x<y True
>=	هل القيمة التي على يسار العامل أكبر أو تساوي القيمة التي على يمينه؟	<<<x,y=6,7 <<<x>=y False
<=	هل القيمة التي على يسار العامل أصغر أو تساوي القيمة التي على يمينه	<<<x,y=6,6 <<<x<y True

➢ العوامل المنطقيةLogical Operators

تُستخدَم العوامل المنطقية لتركيب أسئلة منطقية أكثر تعقيدًا من خلال عوامل الموازنة السابق ذكرها مثلًا، نفترض وجود أربعة متغيرات وهي x و y و z وv ، وإذا سألنا السؤال التالي: هل قيمة المتغير x تساوي قيمة المتغير y أو قيمة المتغير z تساوي قيمة المتغيرv ؟

يتكون السؤال المذكور من سؤالين فرعيين يفصل بينهما كلمة **أو** وهي عامل منطقي في البرمجة، وهنا تُعالَج الأسئلة الفرعية، ثم يُتخَذ القرار بناءً على ذلك العامل المنطقي.

توجد ثلاثة عوامل منطقية هي and و or و not بصورة أساسية، وكل منها يتعامل مع الجمل بطرق مختلفة، فالعامل الأول والثاني يعالجان أسئلةً معقدةً تتكون من أكثر من سؤال فرعي؛ أما الثالث فيتعامل مع جملة محدَّدة بحيث ينفي نتيجتها المنطقية أي يُحوّل الصح True إلى خطأ.False

وفيما يلي استخدامات العوامل المنطقية:

- and : إذا كانت جميع الجمل أو الأسئلة الفرعية صحيحةً، فستنتُج إجابة صحيحةTrue ، وإذا كانت أيّ من الأسئلة الفرعية خاطئةً، فسينتُج خطأ.False

```
x = 1;  y = 1
x == 1  and y == 1 # >> True
```

- or: إذا كانت أي من الجمل صحيحةً، فستنتُج إجابة صحيحة حتى إذا كانت باقي الجمل غير صحيحة

```
x = 1;  y =1
x == 2  or y == 1 # >> True
```

- not: يعكس القيمة المنطقية الناتجة.

```
x = 1
not x == 1  # >> False
```

➢ عوامل العضويةMembership Operators

وهي إحدى العوامل المستخدَمة في الموازنة، أو بمعنى أبسط المستخدَمة في طرح أسئلة يُبنَى عليها اتخاذ قرار ما في البرنامج، ويوجد في بايثون عاملان فقط تحت ذلك التصنيف وهما عامل الوجود **in** وعامل عدم الوجود **not in** ، إذ يستخدَم العامل الأول للتحقق من وجود قيمة ما بداخل سلسلة من القيم؛ أما الثاني فيستخدَم للتحقق من عدم وجود القيمة في سلسلة ما. تعني سلسلة القيم هنا القوائم والصفوف والمفاتيح الموجودة في القواميس، بمعنى أنه يمكننا التحقق من وجود قيمة معينة بداخل قائمة أو صف، أو من وجود قيمة معينة مثل المفاتيح الموجودة في قاموس، وفيما يلي ثلاثة أمثلة على التحقق من وجود قيمة باستخدام العامل الأول:

```
x = [5, 6, 7]
print (5 in x)
>> True
y = (5, 6, 7)
print (5 in y)
>> True
z = {5: 'x', 6: 'y', 7: 'z'}
print (5 in z)
>> True
```

أما العامل الثاني فيتحقق من عدم وجود القيمة في السلسلة، وفيما يلي الأمثلة السابقة نفسها ولكن باستخدام العامل الثاني:

```
x = [5, 6, 7]
print (5 not in x)
>> False
y = (5, 6, 7)
print (5 not in y)
>> False
z = {5: 'x', 6: 'y', 7: 'z'}
print (5 not in z)
>> False
```

➢ عوامل الهويةIdentity Operators

وهي إحدى الأنواع المستخدَمة في الموازنة، إذ توازن بين شيئين للتحقق من تطابقهما تمامًا، ويوجد عاملان فقط تحت هذا التصنيف وهما عامل is وعامل نفيهis not ، إذ يستخدَم العامل الأول للتحقق من تطابق كائنين؛ أما الثاني فيستخدَم للتحقق من عدم تطابق كائنين، وفيما يلي مثال على ذلك:

```
x = 5
print (x is 5)
>> True
print (x is not 5)
>> False
```

➢ عوامل الإسنادAssignment Operators

يوجد عامل إسناد واحد بصورة أساسية في بايثون وفي معظم لغات البرمجة وهو(=) ، إذ يُستخدَم لإسناد قيمة إلى متغير، فإذا كان لدينا متغير ما يدعى x وقيمته الرقم ١، فنستطيع كما يلي إجراء عوامل المنطقيةLogical Operators

في الجدول التالي العوامل مع استخداماتها:

العامل	الاستخدام
=	يسند القيمة الموجودة على يمينه كما هي إلى المتغير الموجود على يساره إذا كان المتغير موجودًا بالفعل، أو ينشئ المتغير في الذاكرة إذا لم يكن موجودًا ثم يسند إليه القيمة الموجودة على يمينه كما هي.
=+	يضيف القيمة الموجودة على يمينه إلى قيمة المتغير الموجود على يساره.
=-	يطرح القيمة الموجودة على يمينه من قيمة المتغير الموجود على يساره ثم يسند الناتج إلى المتغير.
=*	يضرب قيمة المتغير الموجود على يساره بالقيمة الموجودة على يمينه ثم يسند الناتج إلى المتغير.
=/	يقسِّم قيمة المتغير الموجود على يساره على القيمة الموجودة على يمينه ثم يسند الناتج إلى المتغير.
=%	يُوجِد باقي قسمة قيمة المتغير الموجود على يساره على القيمة الموجودة على يمينه ثم يسند الناتج إلى المتغير.
=**	يُجري حسابًا أسيًا على قيمة المتغير، إذ يكون الأس هو القيمة الموجودة على يمينه، ثم يسند الناتج إلى المتغير.
=//	يُجري عملية قسمة صحيحة على قيمة المتغير من اليسار على القيمة الموجودة على يمينه، ثم يسند الناتج إلى المتغير.

مثال: اكتب برنامج لايجاد مساحة المستطيل

```
length=float(input('enter length '))
width=float(input('enter width '))
area=length*width
print('area= ',area)
```

الناتج

enter length 8

enter width 6

area= 48.0

مثال:

بناء جملة بايثون python syntax

ال syntax: هو مجموعة من القواعد تحدد كيفية كتابة برنامج بايثون .

لتنفيذ بناء جملة بايثون يمكن عن طريق :

- الكتابة مباشرة في سطر الاوامر.
- محرر الاكواد.

الطريقة الاولى : عن طريق سطر الاوامر مباشرة.

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

الطريقة الثانية: عن طريق انشاء ملف بايثون بامتداد (py) في محرر أكواد (مثل VS Code أو PyCharm) وتشغيله في سطر الاوامر .

```
print("Hello world”)
```

عبارات بايثون متعددة الأسطر

عادةً ما تنتهي العبارات في بايثون بسطر جديد، ومع ذلك، تسمح بايثون باستخدام حرف متابعة السطر (\) للإشارة إلى أن السطر يجب أن يستمر،

مثال:

```
total = item_one + \
item_two + \
item_three
total = 5 + \
10 + \
15
total=30
```

التعليقات في بايثون

التعليق هو شرح أو تعليق مقروء من قبل المبرمج في كود مصدر بايثون، تمت إضافته بغرض تسهيل فهم الكود المصدري، ويتم تجاهله بواسطة مترجم بايثون، ويبتدأ التعليق في لغة بايثون بالرمز:"#"

تذكر: أنت لست مجبراً على وضع تعليقات في برامجك و لكننا ننصحك بوضع تعليقات دائماً حتى تساعدك في فهم الكود الذي كتبته.

مثال:

```
# First comment

#    هذا تعليق يتألف من سطر واحد و هو لا يؤثر أبداً على الكود الموضوع

#  هذا تعليق آخر, يمكنك وضع العدد الذي تريده من التعليقات
```

كتابة أكثر من أمر واحد على نفس السطر

إفتراضياً، بايثون تعتبر أن كل أمر يكتب على سطر واحد. إذا أردت كتابة أكثر من أمر على نفس السطر قم بوضع فاصلة منقوطة ; بين كل أمرين و هكذا سيفهم مترجم لغة بايثون أن السطر عليه أكثر من أمر.

مثال:

هنا قمنا بوضع ثلاث أوامر على سطر واحد.

فعلياً, كل أمر هنا عبارة عن تعريف متغير و إعطائه قيمة.

```
x = 1; y = 2; z = 3
```

كتابة أمر واحد على أكثر من سطر

إذا أردت كتابة أمر واحد على أكثر من سطر قم بوضع الرمز \ في نهاية كل سطر و هكذا سيفهم مترجم لغة بايثون أن الأمر يتألف من أكثر من سطر.

مثال:

Test.py

```
# هنا قمنا بتعريف ثلاث متغيرات

item_1 = 10

item_2 = 20

item_3 = 30

# الثلاث أسطر التالية عبارة عن أمر واحد

# total إذا هنا سيتم جمع قيم المتغيرات item_1 و item_2 و item_3 و وضع الناتج في المتغير

total = item_1 + \

    item_2 + \

    item_3

# total هنا قمنا بعرض قيمة المتغير

print("total contains:", total)
```

سنحصل على النتيجة التالية عند التشغيل.

```
total contains: 60
```

ملاحظة: الجمل التي تحتوي على الرموز [] أو () أو {} يمكن كتابتها مباشرةً على عدة أسطر. أي لا تحتاج \ في نهاية كل سطر.

النصوص(strings)

لتعريف نص في بايثون نستخدم الرمز ' أو الرمز " أو الرمز """ .

هل يوجد فرق بين هذه الرموز؟

- بالنسبة للرمز ' و الرمز " فإنه لا يوجد أي فرق بينهما. و يمكن إستخدام أي واحد منهما لتعريف نص يتألف من سطر واحد.
- بالنسبة للرمز ''' و الرمز """ فإنه لا يوجد أي فرق بينهما. و يمكن إستخدام أي واحد منهما لتعريف نص كبير يتألف من عدة أسطر.

في المثال التالي قمنا بتعريف ثلاث متغيرات تحتوي على قيم نصية. لاحظ أننا قمنا بتعريف كل متغير بواسطة رمز مختلف.

مثال :

Test1.py

```
# هنا قمنا بتعريف ثلاث متغيرات تحتوي على قيم نصية

name = 'Mhamad'

job = "Programmer"

message = '''This string that will span across multiple lines. No need to use newline characters for the next lines.

The end of lines within this string is counted as a newline when printed.'''

# هنا قمنا بعرض قيم المتغيرات النصية بأسلوب مرتب

print('Name: ', name)

print('Job: ', job)

print('Message: ', message)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Name: Mhamad

Job: Programmer
```

Message: This string that will span across multiple lines. No need to use newline characters for the next lines.

The end of lines within this string is counted as a newline when printed.

مثال

في المثال التالي قمنا بتعريف نص يحتوي على نفس الرموز التي تستخدم لتعريف النصوص.

Test2.py

```
# هنا قمنا بتعريف متغير إسمه text يحتوي على قيمة نصية
text = """In this line we print 'single quotations'
In this line we print "double quotations" """
# هنا قمنا بعرض قيمة المتغير text
print(text)
```

سنحصل على النتيجة التالية عند التشغيل.

```
In this line we print 'single quotations'
In this line we print "double quotations"
```

العمليات الأساسية على النصوص (string)

1. التسلسل

التسلسل هو عملية ربط سلسلتين أو أكثر معًا.

مثال

```
string1="hello"
string2="world"
result=string1+" "+stirng2
```

الناتج

Hello world

٢. طول النصوص

يمكن العثور على طول السلسلة (عدد الأحرف) باستخدام دالة ()len.

مثال

```
Text="Python Programing"
Length=len(text)
```

الناتج

Length=18

٣. تقطيع النصوص(السلسلة)

يتيح لك تقطيع السلسلة استخراج جزء من السلسلة عن طريق تحديد مؤشرات البداية والنهاية (سيتم شرحها لاحقاً بالتفصيل).

٤. تسلسل النصوص مع المتغيرات

يمكنك ربط السلاسل مع المتغيرات لإنشاء نص ديناميكي.

مثال

```
name="alica"
age=30
greeting="hello, "+name+"!you are "+str(age)+"years old"
```

الناتج

```
hello alica you are 30 years old
```

٥. جمع(دمج) النصوص

في بايثون، يمكن ربط السلاسل باستخدام علامة + ويتم استخدام عامل التشغيل * لإنشاء ملف تسلسل متكرر من السلاسل.

مثال :

```
>>> string_1 = "face"
>>> string_2 = "book"
 >>> concatenated_string = string_1 + string_2
 >>> concatenated_string
'facebook'
```

مثال

```
 >>> singer = str(50) + "cent"
 >>> singer
'50cent'
>>> repetition_of_string = "wow" * 5
>>> repetition_of_string
'wowwowwowwowwow'
```

تحويل السلسلة

```
num_str="42"

num_int=int(num_str)
```

مقارنة النصوص

يمكنك استخدام (>، <، <=، >=، ==، !=) لمقارنة سلسلتين مما يؤدي إلى قيمة منطقية صحيحة أو خاطئة. تقارن Python السلاسل باستخدام قيمة ASCII للأحرف.

مثال :

```
>>> "january" == "jane"

False

 >>> "january" != "jane"

True

>>> "january" < "jane"

False

>>> "january" > "jane"

True

>>> "january" <= "jane"

False

>>> "january" >= "jane"

True

>>> "filled" > ""

True
```

الدوال المضمنة في النصوص

len()	تعيد طول (أي عدد عناصر) كائن معين, في النصوص تحسب عدد الأحرف في السلسلة. يتم أيضًا حساب أحرف المسافة البيضاء.
max()	تعيد اكبر عنصر من عناصر كائن قابل للتكرار او اكبر معامل من معاملين او اكثر. في النصوص تقوم الدالة بإرجاع حرف له أعلى قيمة ASCII.
min()	تعيد أصغر عنصر من عناصر كائن قابل للتّكرار أو أصغر مُعامل من مُعاملين أو أكثر تقوم الدالة بإرجاع حرف له أدنى قيمة ASCII

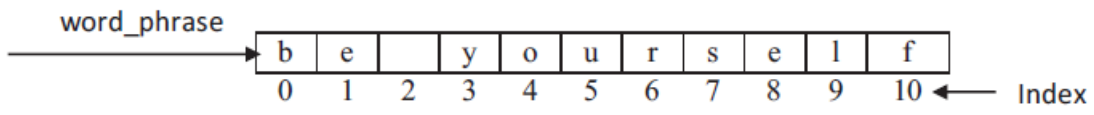
مثال

```
>>> count_characters = len("eskimos")
>>> count_characters
7
>>> max("axel")
'x'
>>> min("brad")
'a'
```

الوصول إلى الأحرف في النصوص حسب رقم الفهرس

يحتل كل حرف في السلسلة موضعًا في السلسلة. يتوافق كل حرف من أحرف السلسلة مع رقم فهرس. الحرف الأول موجود في الفهرس ٠؛ الحرف التالي موجود في الفهرس ١، وهكذا. طول السلسلة هو عدد الأحرف الموجودة فيها. يمكنك الوصول إلى كل حرف في سلسلة باستخدام عامل تشغيل منخفض، أي قوس مربع. تُستخدم الأقواس المربعة لإجراء فهرسة في سلسلة للحصول على القيمة عند فهرس أو موضع محدد. ويسمى هذا أيضًا عامل التشغيل المنخفض.

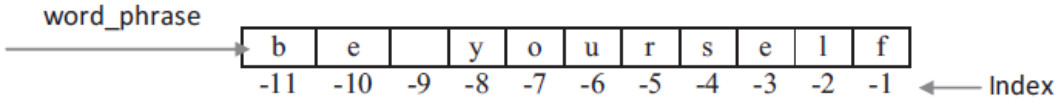
يظهر أدناه تفاصيل الفهرس للسلسلة " be yourself" المخصصة لمتغير السلسلة word_phrase.



حيث يكون الفهرس عادة في حدود ٠ إلى واحد أقل من طول السلسلة. يجب أن تكون قيمة الفهرس دائمًا عددًا صحيحًا وتشير إلى الحرف الذي سيتم الوصول إليه.

```
>>> word_phrase = "be yourself"
 >>> word_phrase[0]
'b'
>>> word_phrase[1]
'e'
 >>> word_phrase[2]
' '
 >>> word_phrase[3]
'y'
```

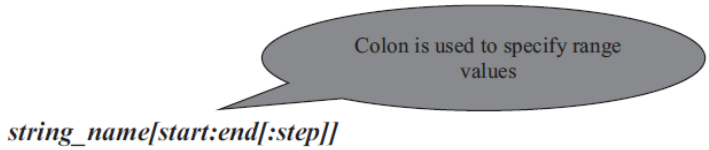
يمكنك أيضًا الوصول إلى الأحرف الفردية في سلسلة باستخدام الفهرسة السلبية. إذا كان لديك سلسلة طويلة وتريد الوصول إلى أحرف النهاية في السلسلة، فيمكنك العد التنازلي من نهاية السلسلة بدءًا من رقم الفهرس −١. يظهر أدناه تحليل الفهرس السلبي للسلسلة "be yourself " المخصصة لمتغير سلسلة word_phrase.




```
1. >>> word_phrase[-1]
'f'
2. >>> word_phrase[-2]
'l'
```

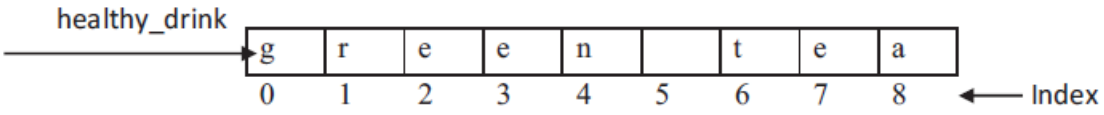
تقطيع النصوص والدمج بينها باستخدام المؤشر (index)

يعد بناء الجملة طريقة سهلة للإشارة إلى الأجزاء الفرعية لتسلسل الأحرف داخل ملف السلسلة الأصلية.
بناء الجملة لتقطيع السلسلة هو،



باستخدام تقطيع السلسلة:

- يمكنك الوصول إلى سلسلة من الأحرف عن طريق تحديد نطاق من أرقام الفهرس مفصولة بنقطتين.
- يُرجع تقطيع السلسلة سلسلة من الأحرف تبدأ من البداية وتمتد حتى النهاية ولكن لا تشملها.
- يجب أن تكون قيم فهرسة البداية والنهاية أعدادًا صحيحة.
- يمكن إجراء تقطيع السلسلة باستخدام الفهرسة الإيجابية أو السلبية.

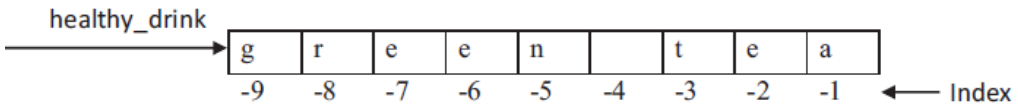


مثال

```
>>> healthy_drink = "green tea"
 >>> healthy_drink[0:3]
'gre'
 >>> healthy_drink[:5]
'green'
 >>> healthy_drink[6:]
'tea'
 >>> healthy_drink[:]
'green tea'
```

```
>>> healthy_drink[4:4]
' '
>>> healthy_drink[6:20]
'tea'
```

يمكن استخدام الفهرس السالب للوصول إلى الأحرف الفردية في السلسلة. الفهرسة السلبية يبدأ بفهرس ١– المطابق للحرف الأخير في السلسلة ثم الفهرس يتناقص بمقدار واحد عندما نتحرك إلى اليسار.



مثال

```
>>> healthy_drink[-3:-1]
'te'
>>> healthy_drink[6:-1]
'te'
```

تحديد خطوة لعملية تقطيع النصوص

في عملية التقطيع، يمكن تحديد وسيطة ثالثة تسمى الخطوة وهي اختيارية مع أرقام فهرس البداية والنهاية. تشير هذه الخطوة إلى عدد الأحرف التي يمكن تخطيها بعد بدء فهرسة الحرف في السلسلة. القيمة الافتراضية للخطوة هي واحدة. في أمثلة التقطيع السابقة، لم يتم تحديد الخطوة وفي حالة عدم وجودها، يتم استخدام قيمة افتراضية واحدة. على سبيل المثال،

```
1. >>> newspaper = "new york times"
2. >>> newspaper[0:12:4]
'ny'
3. >>> newspaper[::4]
'ny e
```

إدخال بيانات من المستخدم

الدالة()input

- لجعل المستخدم قادر على إدخال بيانات في البرنامج أثناء اشتغاله نستخدم دالة جاهزة إسمها . input()
- في كل مرة تقوم فيها باستدعاء هذه الدالة يقوم مفسّر لغة بايثون بانتظارك لإدخال ما تريد من لوحة المفاتيح (Keyboard) .
- بعد الإنتهاء من الإدخال و النقر على الزر Enter سيتم إرجاع الشيء الذي قمت بإدخاله كنص في المكان الذي تم منه إستدعاء الدالة input() .

ملاحظة : عند استدعاء الدالة input() فإنك حتى لو قمت بإدخال رقم فإنها سترجعه كنص.

لذلك في حال كنت تريد من المستخدم أن يدخل رقم, سيكون عليك تحويل ما ترجعه الدالة لرقم.

.

مثال : إنشاء برنامج يطلب من المستخدم إدخال أسمه ثم يعرضه له.

Test1.py

```
هنا قمنا بإظهار رسالة تطلب من المستخدم أن يدخل إسمه. ويتم تخزينه في المتغير name #
name = input("What's your name? ")
هنا قمنا بعرض جملة ترحيب مبنية على إسم المستخدم الذي قمنا بتخزينه قبل قليل في المتغير name #
print("Nice to meet you", name)
```

سنحصل على النتيجة التالية عند تشغيل الملف Test1 مع الإشارة إلى أننا قمنا بتعليم البيانات التي إنتظرنا البرنامج لإدخالها من لوحة المفاتيح باللون الأصفر.

```
What's your name? mhamad
Nice to meet you mhamad
```

مثال : قم بإنشاء برنامج يطلب من المستخدم إدخال عددين صحيحين، ثم يعرض له ناتج جمعهما.

يجب وضع الدالة input() بداخل الدالة int() حتى يتم تحويل الرقم الذي سيدخله المستخدم إلى عدد صحيح قبل تخزينه في المتغير.

لو لم نفعل ذلك لتم إعتبار الأرقام التي أدخلها المستخدم عبارة عن نصوص و بالتالي كان ذلك سيسبب خطأ منطقي إظهار ناتج الجمع.

Test2.py

```
a = int(input("Enter a: "))
b = int(input("Enter b: "))
print('a + b =', a + b)
```

سنحصل على النتيجة التالية عند تشغيل الملف Test1 مع الإشارة إلى أننا قمنا بتعليم البيانات التي انتظرنا البرنامج لإدخالها من لوحة المفاتيح باللون الأصفر.

```
Enter a: 5
Enter b: 7
a + b = 12
```

تعليمة الطباعة print

لعرض مخرجات الأكواد في لغة بايثون يتم استخدام تعليمة الطباعة(print) وتستخدم لعرض قيم وقيم متغيرات ونصوص وهناك العديد من الاشكال لهذه التعليمة منها:

١. طباعة نص ثابت

```
print(“Hello,world”)
```

٢. طباعة قيمة متغير

```
var=5
print(var)
```

٣. طباعة عدة متغيرات

```
name="nada"
age=20
print ("Name:",name,"Age:", age)
```

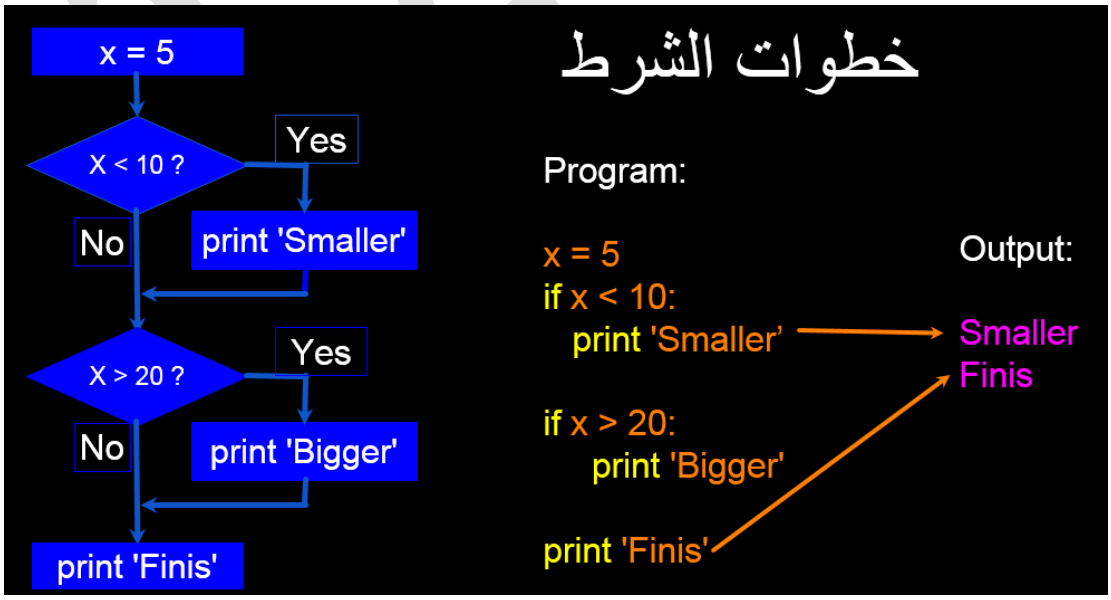
الجمل الشرطية Condition statements

تعتبر الجمل الشرطية من أهم وأساسيات لغة البرمجة Python وأي لغة برمجة أخرى، تسمح هذه الجمل بتحقق شرط معين وتنفيذ أكواد مختلفة استناداً إلى ما إذا كان الشرط صحيحاً (True) أم غير صحيح (False). تسهم الجمل الشرطية في جعل البرامج قادرة على اتخاذ قرارات مستندة إلى متغيرات وبيانات معينة، مما يزيد من قوة ومرونة التطبيقات.

تدعم لغة البايثون الشروط المنطقية الموجودة في الرياضيات . كما يمكن استخدام هذه الشروط بعدة طرق و اكثرها شيوعا مع if و loop , و هذه الشروط المنطقية موضحة بالجدول التالي :

الرمز	المعنى	
a == b	Equals	يساوي
a != b	Not Equals	لا يساوي
a < b	Less than	اقل
a <= b	Less than or equal to	اقل أو يساوي
a > b	Greater than	اكبر
a >= b	Greater than or equal to	اكبر أو يساوي

وكما يعرف الجميع فأن العبارات(statements) لا تنفذ الا إذا كان الشرط (condition) صحيحاً واما إذا كان الشرط خطأ فيتم القفز مباشرة الى العبارات بعد بلوك ال(if) وكما في المخطط التالي:



امثلة على الشروط المنطقية :

```
a = 2
b = 20
if b > a:
   print("b is greater than a")
else :
   print("b is Less than a")
```

حسب المثال أعلاه يتحقق الشرط لذا سيكون الاخراج :

```
b is greater than a
```

بينما في المثال التالي لا يتحقق الشرط :

```
a = 200
b = 20
if b > a:
   print("b is greater than a")
else :
   print("b is Less than a")
```

إخراج البرنامج يكون كالتالي :

```
b is Less than a
```

كتابة الجملة الشرطية في لغة البرمجة بايثون

خطوات كتابة الجملة الشرطيةif

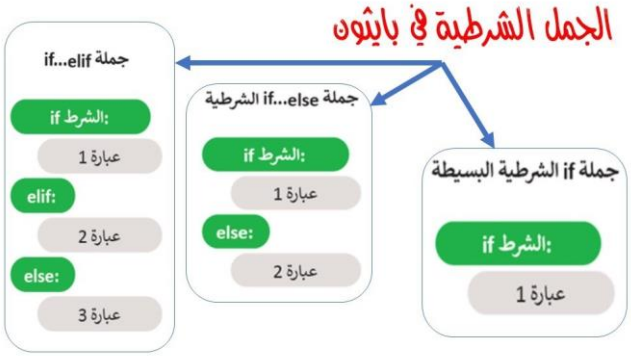
- في حالة تحقق الشرط نكتب اداة الشرط **if** وهي كلمة محجوزة.
- نكتب التعبير او الجملة التي نريد ان نتحقق منها بجانب. if
- بعد ذلك نكتب نقطتين فوق بعض. (:)
- ننزل سطر ونمشي اربع مسافات لكي تصبح الجملة تتبع اداة الشرط **if** ونكتب الحالة التي نريد تنفيذها عند تحقق الشرط.

والصيغة التالية توضح كيفية كتابة الجملة الشرطيةif

```
If expression:
   statement
```

أنواع جملة if الشرطية

يوجد في بايثون ثلاث انواع من الجمل الشرطية هي كما يلي : "if"، "elif"، و"else" .



١ - جملة *if*

جملة "if" تستخدم لتنفيذ كود معين إذا كان شرط معين يعتبر صحيحًا. الصيغة العامة لاستخدام جملة "if"هي:

```
if condition
```

مثال : اكتب برنامج للتحقق من العدد المدخل زوجي او فردي:

```
num = int(input("Enter the number : "))
if num%2 == 0:
   print("Number is even")
else :
   print("Number is odd")
```

هنالك مسارين لتنفيذ البرنامج :

المسار الاول ان يكون العدد المدخل فردي مثلا ندخل العدد ١١ هنا سيكون التنفيذ بالشكل التالي:

```
Enter the number : 11
Number is odd
```

المسار الثاني ان يكون العدد المدخل زوجي مثلا ندخل العدد ١٢ هنا سيكون التنفيذ بالشكل التالي:

```
Enter the number : 1٢
Number is odd
```

مثال : اكتب برنامج باختيار أكبر عدد من بين ثلاثة أعداد مد خلة من قبل المستخدم.

```
a = int(input("Enter a? "));
b = int(input("Enter b? "));
c = int(input("Enter c? "));
if a>b and a>c:
   print("a is largest");
if b>a and b>c:
   print("b is largest");
if c>a and c>b:
   print("c is largest");
```

في المثال السابق سنقوم بإدخال ثلاثة أعداد لكل من a,b,c وسيقوم محرر بايثون بمقارنة الأرقام من خلال جملة الشرط if ويطبع الرقم الاكبر من بين الثلاثة الأعداد .كما إن المثال السابق يحتوي على ثلاث جمل شرطية منif

```
Enter a? 6
Enter b? 9
Enter c? 3
b is largest
```

في حالة كان لدينا شرط واحد فقط يمكننا كتابة هذا الشرط في سطر واحد .المثال التالي يوضح هذه الجملة

```
a=5
b=3
if a > b: print("a is greater than b")
```

output: a is greater than b

يحتوي المثال السابق على شرط واحد فقط ,فاذا تحقق الشرط سيتم طباعة جملة

a is greater than b

بينما في حال عدم تحقق الشرط لن يتم طباعة أي شي.

يمكن ايضا ان تحتوي الجملة الشرطية على جمل متداخلة من الجملة الشرطية if .المثال التالي يوضح كيف يمكن ان تحتوي الجملة الشرطية على جمل متداخلة من if

```
x = 30
if x > 10:
  print("x Greater than 10")
  if x > 20:
    print("and also x Greater than 20")
  else:
    print("but not less than 5")
```

الناتج

```
x Greater than 10
and also x Greater than 20
```

المثال السابق تم تعريف المتغير x بقيمة تساوي 30 وقمنا باستخدام العبارة الشرطية if ,وداخل هذا الشرط استخدمنا عبارة شرطية أخرى منif . هذا يثبت انه يمكن استخدام عبارات متداخلة من if

ملاحظة هامة

لا يمكن ان تكون عبارة if فارغة، ولتجنب حدوث الخطأ عند تكون جملة الشرطية او عبارة ifفارغة لسبب ما , نستخدم كلمة pass.

```
a = 2
b = 4
if b > a:
  pass
```

هنا لن يحدث اي خطأ عند تنفيذ الكود , وذلك بسبب وجود كلمة pass

لكن في حالة عدم كتابة كلمة pass سيحدث خطأ بالكود عند تنفيذه. لنجرب تنفيذ الكود السابق بدون إستخدام كلمةpass

```
a = 2
b = 4
if b > a:
```

Output : **Indentation Error:** expected an indented block

نلاحظ انه عند كتابة الجملة الشرطية بشكل فارغ اي انها لا تحتوي على اي شي وكذلك لم نستخدم كلمة passكما في المثال السابق نتج عن ذلك خطأ ,كما هو موضح في المثال السابق

٢ - جملة *elif*

هي كلمة محجوزة في لغة البايثون وتعني انه في حالة لم تكن الشروط السابقة صحيحة فجرب هذا الشرط .او يمكن ان تكون بمعنى التسلسل التالي if-else-if .ويمكن ان تحتوي الجملة الشرطية على عدد من عبارات elif

مثال :

```
a = 20
b =20
if b > a:
   print("b is greater than a")
if b < a:
   print("b is Less than a")
elif a == b:
   print("a and b are equal")
```

الناتج

a and b are equal

في المثال السابق قمنا بإعطاء كلا من a ,b قيم متساوية , واستخدمنا اداة الشرط ifانه في حال كانت قيمة b اكبر من قيمة a سيتحقق الشرط ويطبع جملة b is greater than a ,كما استخدمنا شرط اخر حيث اذا كانت قيمة b اقل من قيمة a يقوم بطباعة جملة b is Less than a , وقمنا باستخدام اداة الشرط elif اي انه في حال لم يتحقق كلا الشرطين جرب استخدام هذا الشرط، وبما ان a=b فان الشرط تحقق لذلك سيقوم بطباعة الجملة.

مثال

```
python is programming language
prog_language ="python"
if prog_language == "python":
   print ("python is programming language")
else :
    print ("other")
```

الناتج

python is programming language

هنا تحقق الشرط اي ان المتغير prog_language يساوي كلمة python تم طباعة جملة

python is programming language

ملاحظة :يجب علينا مراعاة المسافات لان لغة البايثون لغة حساسة للمسافات لكن في حالة لم يتحقق الشرط كما في المثال التالي:

```
prog_language ="python"
if prog_language == "Arabic":
   print ("python is programming language")
else:
   print ("other")
```

الناتج

other

هنا لم يتحقق الشرط اي ان المتغير prog_language لا يساوي كلمة python تم طباعة جملة other

٣- جملة *if-else*

else : كلمة محجوزة تستخدم في حالة عدم تحقق الشروط السابقة

مثال :اكتب برنامج لإدخال عمر المستخدم للسماح له بالدخول او عدم السماح

```
age = int (input("Enter your age : "))
if age>=18:
   print("You can log in ");
else:
   print("Sorry!You can not log in ");
```

الناتج

```
Enter your age : 17
Sorry!You can not log in
Enter your age : 22
You can log in
```

في حالة كان لدينا شرط if و else واحد فقط يمكننا كتابتهم في سطر واحد.

مثال :

```
a = 20
b = 50
print("A is greater than B") if a > b else print("A is Less than B")
```

output : A is Less than B

ويمكن ان يكون هناك عدة عبارات من else في سطر واحد.

مثال :

```
a = 20
b = 50
print("A is greater than B") if a > b else print("A is Less than B")
```

output : A is Less than B

احتوى المثال السابق على عدة جمل شرطية وقمنا بكتابة هذه الجمل في نفس السطر، كما قام محرر البايثون بالتحقق من جميع الشروط الموجودة في هذا السطر، وقام بطباعة الجملة الخاصة بالشرط المحقق.

إستخدام الجمل الشرطية مع المعامل المنطقي and

هي كلمة محجوزة في بايثون وهي عامل منطقي سبق ان تحدثنا عنه، تستخدم and للدمج بين العبارات الشرطية ويجب تحقق الشرطين معاً.

مثال :

```
a = 20
b = 10
c = 50
if a > b and c > a:
print("Both conditions are True")
```

الناتج

output : Both conditions are True

في المثال السابق استخدمنا and للدمج بين العبارتين الشرطيتين تحقق الشرط فتم طباعة الجملة الخاص بتحقق الشرط بينما في حال عدم تحقق الشرط السابق فلن يتم طباعة أي شي.

استخدام الجمل الشرطية مع المعامل المنطقي or

كلمة or هي كلمة محجوزة في بايثون تستخدم للدمج بن العبارات الشرطية كما تفعل and وتعني or انه في حالة تحقق احد الشرطين يتم تنفيذ او طباعة الجملة الخاصة بتحقق الشرط.

```
a = 200
b = 10
c = 50
if a > b or c > a:
  print("Both conditions are True")
```

في هذا المثال تحقق الجزء الاول من الشرط وهو ان a اكبر من b ,ولكن الجزء الثاني من الشرط لم يتحقق , حيث ان c ليست اكبر من a .وبما ان الرابط بين الجزئين كان العامل المنطقي or لذلك تحقق الشرط لأن في orلا يجب تحقق الجزئين من الشرط وانما يكتفي بتحقق شرط واحد على عكس and فيجب تحقق الجزئين من الشرط.

حلقات الدوران والتكرار Loops and Iteration

من المهام التي تبذل الآلات قصارى جهدها هي تكرار المهام المتماثلة من دون خطأ. هنالك العديد من الطرق لبرمجة المهام المتكررة. تكرار الكود او الامر عدة مرات حسب ما يريد المستخدم .وفي بايثون هناك نوعين من الحلقات التكرارية (while, for)

حلقات التكرار – العبارة while

في البرمجة , نسمي حلقات التكرار بنظام التعليمة الذي يكرر المهمة المحددة عدة مرات (او بشكل لا نهائي) .

الصيغة العامة لحلقة التكرار while

```
while conditional_expression:
   Code block of while
```

يتم اختبار الشرط المعطى في البداية في الحلقة التكرارية ,فاذا كان الشرط صحيحا سيتم ارجاع القيمة المنطقية , True وسوف يتم تنفيذ عبارات الحلقة , while وفي كل مرة سيتم التحقق من الشرط وفي حالة تم ارجاع القيمة المنطقية false فلن يتم تنفيذ العبارات التكرارية.
في بايثون يتم تفسير اي رقم غير الصفر على انه منطقي صحيح ويتم تفسير الخطأ بالصفر.

مثال :

مثال	الناتج

n = 8
i = 1
 	
while i <= n:
  print(i)
  i += 1
	1
	2
	3
	4
	5
	6
	7
	8

قمنا بتعريف متغير n =8 وهو المسؤول عن توقف الحلقة التكرارية , وكذلك قمنا بتعريف متغير i=1 وفي الحلقة التكرارية كتبنا شرط التوقف وهو انه في حال كان المتغير i اقل او يساوي من قيمة n اي من قيمة ٨ , يقوم بالدخول الى الدوارة او الى الحلقة التكرارية وطباعة الامر وهنا الامر هو طباعة المتغير i . في حال خالفت القيمة المدخلة الشرط الموجود بداية الدوارة عند ذلك سيتوقف الدوران اي سيخرج من الحلقة التكرارية.

مثال	الناتج

s = "Hello"
i = 0
while i < len(s):
  print(s[i])
  i += 1
	H
	e
	l
	l
	o

استخدام break فيwhile loop

تستخدم break للتوقف في الحلقة التكرارية, while فعند التحقق من الشرط سيتم تنفيذ الامر داخل الحلقة ولكن عند وجود كلمة break سيتوقف هنا تنفيذ الامر , اي ستتوقف الحلقة التكرارية.

مثال:

مثال	الناتج

n = 1
while n <= 10:
  print(n)
  if (n == 7):
  break
  n += 1
	1
	2
	3
	4
	5
	6
	7

في المثال السابق بعد ان تم التحقق من الشرط والتأكد ان الرقم اقل من ١٠ تم تنفيذ الامر وهو طباعة الاعداد ,ومن ثم زيادة العدد بواحد +١ . ونعود مجددا الى الشرط . وهكذا مع كل دورة يتم التحقق اولا من الشرط ولكن هناك شرط اخر موجود في الحلقة وهو انه في حال ان العدد يساوي ٧ اعمل . break اي سيتم الخروج من الدوارة.

استخدام continue في while loop

تستخدم كلمة continue للتخطي اي ان الدوارة تتخطى الرقم الموجود في الشرط وتستمر بالدوران ,على عكس كلمة break يتم الخروج من الدوارة . للتفريق اكثر قم بتطبيق المثال التالي وقارن بينه وبين المثال السابق من حيث استخدام كلا من كلمة , break وكلمة. continue

مثال	الناتج

n = 0
while n <= 10:
  n+= 1
  if (n == 7):
  continue
  print(n)
	1
	2
	3
	4
	5
	6
	8
	9
	10

حلقة التكرار for

تستخدم **For loop** لتكرار تنفيذ الامر او الكود عدة مرات بناء على عداد او متغير .اي يكون عدد مرات التكرار معلوم قبل الدخول في الحلقة على عكس **While loop** .

يمكن إستخدامها بطريقتين :

الأولى: للتكرار عبر كتله من التعليمات البرمجية باستخدام دالة النطاق

for val in range (begin ,end , step):

Val: متغير يأخذ قيمة العناصر داخل مدى الدالة لكل دورة .

Begin : اول قيمة في المدى اذا محذوف تأخذ ٠ كقيمة افتراضية .

End : القيمة الأخيرة في النطاق بمقدار ١ القيمة النهائية مطلوبة دائما ولا يجوز حذفها.

Step : مقدار الزيادة او النقصان اذا كانت القيمة محذوفة تكون قيمة الزيادة افتراضية بمقدار ١.

مثال	الناتج
for i in range(1 ,5): print(i)	1 2 3 4

قمنا بإعطاء المتغير i قيمة عدد التكرارات التي ستنفذ داخل الحلقة ,مع إعطاء بداية ونهاية للقيم التي ستسند للمتغير i او بمعنى اخر اعطاء مجال للقيم التي ستسند لهذا المتغير (مدى) , وهنا تم تحديد المدى من ١ الى ٥ . اي البداية ستكون من القيمة ١ والنهاية ستكون عند القيمة ٥ . ثم قمنا بطباعة المتغير i في كل دورة من حلقة التكرار , نلاحظ انه لم يتم طباعة قيمة النهاية القيمة التي تساوي ٥ ضمن التكرار.

مثال	الناتج
for i in range(0,25,5): print(i)	0 5 10 15 20

من خلال المثال السابق نلاحظ انه تم الزيادة لكل خطوة تكرار او لكل عدد واحد من التكرارات بمقدار ٥ . وكذلك تم تحديد بداية التكرار من القيمة ٠ , ونهاية التكرار الى القيمة ٥٠ وتم تخطي ٥ ارقام في كل دورة.

ثانياً: لتكرار عبر سلسلة (.list, tupe ,string, etc) تكرار السلسلة يسمى اجتياز

الصيغة العامة

for val in sequence:

body of val

val: متغير يأخذ العناصر داخل المتسلسلة لكل تكرار

الفرق بين while & for

إسم الحلقة	دواعي الإستخدام
For Loop	تستخدم الحلقة `for` للمرور على جميع عناصر السلسلة أو المصفوفة بسهولة بدون الحاجة لتعريف عداد و تحديد أين يبدأ و أين ينتهي. و تستخدم لتنفيذ الكود عدة مرات محددة.
While Loop	تستخدم الحلقة `while` لتنفيذ الكود عدة مرات غير محددة و يتوقف التنفيذ إذا تحقق شرط معين. لأن هذه الحلقة يتم توقيفها إذا تحقق الشرط الذي نضعه بين القوسين.

جمل التحكم في الحلقات في بايثون

جملة التحكم	تعريفها
Break Statement	الجملة `break` تستخدم بشكل عام لإيقاف الحلقة في حال تحقق شرط معين. ثم تنتقل للكود الذي يليها في البرنامج.
Continue Statement	الجملة `continue` تستخدم بشكل عام لإيقاف الدورة الحالية في الحلقة و الإنتقال إلى الدورة التالية فيها في حال تحقق شرط معين.

مثال: أكتب برنامج بلغة بايثون باستخدام حلقة التكرار while لطباعة جميع الأرقام من ١ الى ١٠ وأستخدام الجملة break لجعل الحلقة تتوقف عندما تصبح قيمة العداد تساوي ٥.

Counterهو قيمة العداد في المثال .

مثال	الناتج

counter = 1
while counter <= 10:
  print(counter)
  if counter == 5:
  break
  counter += 1
print('The loop was stopped when counter =', counter)
	1
	2
	3
	4
	5
	The loop was stopped when counter = 5

نفس المثال تم تطبيقه على for

مثال	الناتج

for n in range(1,11):
  print(n)
  if n == 5:
  break
print('The loop was stopped when n =', n)
	1
	2
	3
	4
	5
	The loop was stopped when n = 5

تمارين توضح الفرق بين for, while

مثال: اكتب برنامج لطباعة الاعداد من صفر الى ٩٩

while	for

i=0
while i<100:
  print(i)
  i=i+1
	For i in range(100):
	  Print(i)

مثال: اكتب برنامج لطباعة الاعداد الفردية من صفر الى ٩٩

while	for

i=1
while i<99:
  print(i)
  i=i+2
	For i in range(1,1002):
	  Print(i)

الحلقات الدورانية المتداخلة

هي عبارة عن دوارة داخل دوارة اخرى والدوارة الداخلية تنفذ مرة واحدة لكل تكرار في الدوارة الخارجية

مثال: اكتب برنامج لايجاد المتسلسلة التالية: $\sum_{i=1}^{5}\sum_{i=1}^{8} i+j$

while	for

i,s=1,0
while i<=5:
  j=1
  while j<=8:
  s=s+i+j
  j=j+1
  i=i+1
print('sum is ',s)
	s=0
	for i in range(1,6):
	  for j in range(1,9):
	  s=s+i+j
	print('sum is ',s)

مثال : اكتب برنامج لايجاد المتسلسلة: $\sum_{i=1}^{n}\sum_{i=1}^{m} i+j$

while	for

n=int(input("enter n"))
m=int(input("enter m"))
i,s=1,1
while i<=n:
  j=1
  while j<=m:
  s=s+(i/j)
  j=j+1
  i=i+1
print('sum is ',s)
	n=int(input("enter n"))
	m=int(input("enter m"))
	s=0
	for i in range(1,n+1):
	  for j in range(1,m+1):
	  s=s+(i/j)
	
	print('sum is ',s)

مثال : اكتب برنامج لطباعة النمط

1111

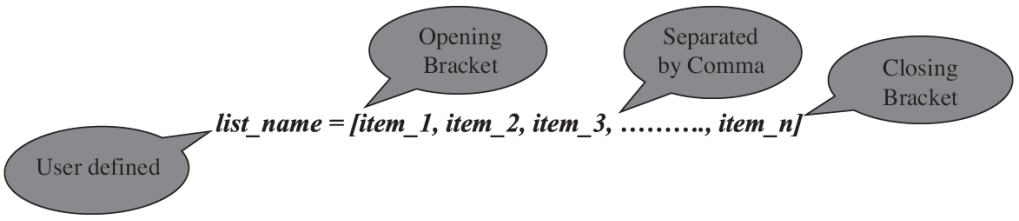
2222

while	for

i=1
while i<5:
  if i==3:
  break
  j=1
  while j<5:
  print(i,end="")
  j=j+1
  print()
  i=i+1
	for i in range(1,6):
	  if i==3:
	  break
	  for j in range(1,6):
	  print(i,end="")
	  print()

مفهوم list

لتخزين انواع مختلفة من البيانات في بايثون كالأرقام والمقاطع النصية(list) يتم استخدام القائمة لتعريف عناصر القائمة بشرط وضع فارزة بين العناصر, يتم استخدام الرمز [].
الصيغة العامة :



مثال:

```
>>> superstore = ["metro", "tesco", "walmart", "kmart", "carrefour"]
>>> superstore
    ['metro', 'tesco', 'walmart', 'kmart', 'carrefour']
```

مثال:

```
>>> number_list = [4, 4, 6, 7, 2, 9, 10, 15]
>>> mixed_list = ['dog', 87.23, 65, [9, 1, 8, 1]]
>>> type(mixed_list)
   <class 'list'>
>>> empty_list = []
>>> empty_list
    []
>>> type(empty_list)
    <class 'list'>
```

مثال:

```
numbers = [10, 20, 30, 40, 50]
print(numbers)
```

سنحصل على النتيجة التالية عند التشغيل.

[10, 20, 30, 40, 50]

مثال :

نقوم بتعريف list لتعريف المقاطع النصية فقط كالتالي:

```
names = ['Rami', 'Sara', 'Nada']
print(names)
```

سنحصل على النتيجة التالية عند التشغيل.

```
['Rami', 'Sara', 'Nada']
```

مثال :

في المثال التالي قمنا بتعريف listوضعنا فيه أعداد صحيحة و نصوص.

```
data = [1, 'Mhamad', 'Harmush', 1500]
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
[1, 'Mhamad', 'Harmush', 1500]
```

العمليات الأساسية على list

في بايثون، يمكن أيضًا ربط القوائم باستخدام علامة +، ويتم استخدام عامل التشغيل * لإنشاء تسلسل متكرر لعناصر القائمة. **مثال**:

```
>>> list_1 = [1, 3, 5, 7]
>>> list_2 = [2, 4, 6, 8]
>>> list_1 + list_2
    [1, 3, 5, 7, 2, 4, 6, 8]
>>> list_1 * 3
    [1, 3, 5, 7, 1, 3, 5, 7, 1, 3, 5, 7]
>>> list_1 == list_2
    False
```

يمكنك التحقق من وجود عنصر في القائمة باستخدام عوامل تشغيل العضوية (**in**) وليس فيها. تقوم بإرجاع قيمة منطقية صحيحة أو خاطئة.

مثال :

```
>>> list_items = [1,3,5,7]
>>> 5 in list_items
      True
>>> 10 in list_items
       False
```

مثال:

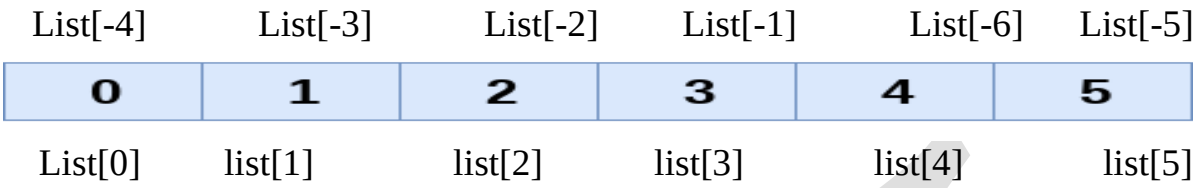
```
data = ['Mhamad', 'Rony', 'Rima', 'Sara']
x = 'Rima'
y = 'Ali'
print('Is Rima in the List?')
print(x in data)
print('Is Ali in the List?')
print(y in data)
```

سنحصل على النتيجة التالية عند التشغيل.

Is Rima in the List?
True
Is Ali in the List?
False

الوصول لعناصر القائمة(التسلسل)

للوصول لأي عنصر في list نستخدم الأرقام بطريقتين فمثلاً في حال أردت الوصول الى العنصر الاول من اليسار الى اليمين نستخدم index(0,1,…) وهكذا، اما في حال اردت الوصول للعنصر الاخير من اليمين الى ليسار فنستخدم index (-1,-2,…) للوصول الى العناصر



مثال :

وضعنا فيه نصوص بعدها قمنا بعرض قيمة أول و ثاني عنصر فيه.

```
names = ['Rami', 'Sara', 'Nada', 'Mhamad', 'Salem']
print(names[0])
print(names[1])
```

سنحصل على النتيجة التالية عند التشغيل.

Rami
Sara

مثال: وضعنا فيه نصوص. بعدها قمنا بعرض قيمة آخر و قبل آخر عنصر فيه.

```
names = ['Rami', 'Sara', 'Nada', 'Muhammad', 'Salem']
print(names[-1])
print(names[-2])
```

سنحصل على النتيجة التالية عند التشغيل.

Salem
Muhammad

تعديل العناصر

القوائم قابلة للتغيير بطبيعتها حيث يمكن تعديل عناصر القائمة بعد إنشاء القائمة.

❖ يمكنك تعديل القائمة عن طريق استبدال العنصر الأقدم بعنصر أحدث في مكانه ودون تخصيص القائمة لمتغير جديد تمامًا.

مثال:

```
>>> fauna = ["pronghorn", "alligator", "bison"]
>>> fauna[0] = "groundhog"
>>> fauna
      ['groundhog', 'alligator', 'bison']
>>> fauna[2] = "skunk"
>>> fauna
     ['groundhog', 'alligator', 'skunk']
>>> fauna[-1] = "beaver"
>>> fauna
     ['Groundhog', 'alligator', 'beaver']
```

مثال:

```
x=["apple" , 5 , 12.5 , True , -2 ]

x[0]=10

x[1]="orange"

print(x)
```

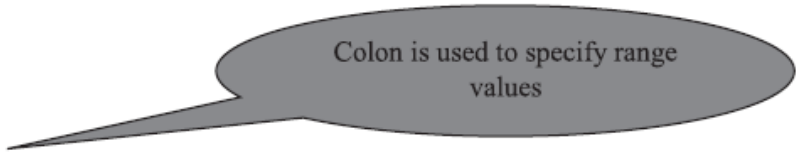
سوف تظهر لنا النتيجة التالية:

[10, "orange" , 12.5 , true , -2]

❖ عندما تقوم بتعيين متغير قائمة موجود إلى متغير جديد، فإن التعيين (=) في القوائم لا يؤدي إلى إنشاء نسخة جديدة. بدلاً من ذلك، يجعل التعيين كلا من أسماء المتغيرات تشير إلى نفس القائمة في الذاكرة.

```
>>> zoo = ["Lion", "Tiger", "Zebra"]
>>> forest = zoo
>>> type(zoo)
     <class 'list'>
>>> type(forest)
    <class 'list'>
>>> forest
     ['Lion', 'Tiger', 'Zebra']
>>> zoo[0] = "Fox"
>>> zoo
    ['Fox', 'Tiger', 'Zebra']
>>> forest
    ['Fox', 'Tiger', 'Zebra']
>>> forest[1] = "Deer"
>>> forest
    ['Fox', 'Deer', 'Zebra']
>>> zoo
    ['Fox', 'Deer', 'Zebra']
```

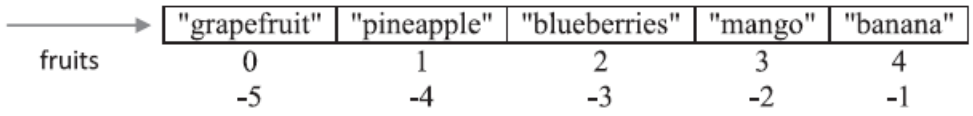
❖ يُسمح بتقسيم القوائم في لغة Python، حيث يمكن استخراج جزء من القائمة عن طريق تحديد نطاق الفهرس مع عامل النقطتين (:) الذي يمثل في حد ذاته قائمة. بناء جملة تقطيع القائمة هو،



list_name[start:stop[:step]]

حيث تكون كل من البداية والتوقف قيمتين صحيحتين (قيم موجبة أو سالبة). يقوم تقسيم القائمة بإرجاع جزء من القائمة من قيمة فهرس البداية إلى قيمة فهرس الإيقاف الذي يتضمن قيمة فهرس البداية ولكنه يستبعد قيمة فهرس الإيقاف. تحدد الخطوة قيمة الزيادة المطلوب تقسيمها وهي اختيارية.

مثال :




```
1.>>> fruits = ["grapefruit", "pineapple", "blueberries", "mango", "banana"]
2.>>> fruits[1:3]
    ['pineapple', 'blueberries']
3.>>> fruits[:3]
    ['grapefruit', 'pineapple', 'blueberries']
4.>>> fruits[2:]
    ['blueberries', 'mango', 'banana']
5.>>> fruits[1:4:2]
    ['pineapple', 'mango']
6.>>> fruits[:]
    ['grapefruit', 'pineapple', 'blueberries', 'mango', 'banana']
7.>>> fruits[::2]
    ['grapefruit', 'blueberries', 'banana']
8.>>> fruits[::-1]
    ['banana', 'mango', 'blueberries', 'pineapple', 'grapefruit']
9.>>> fruits[-3:-1]
    ['blueberries', 'mango']
```

جميع العناصر الموجودة في قائمة fruits بدءًا من قيمة الفهرس ١ وحتى قيمة الفهرس ٣ ولكن باستثناء قيمة الفهرس ٣ يتم تقطيعها إلى شرائح 2 .

إذا كنت ترغب في الوصول إلى عناصر البداية، فليست هناك حاجة لتحديد قيمة الفهرس صفر. يمكنك تخطي قيمة فهرس البداية وتحديد قيمة فهرس التوقف 3 فقط. وبالمثل، إذا كنت تريد الوصول إلى عناصر التوقف الأخيرة، فلا داعي لتحديد قيمة التوقف وعليك أن تذكر فقط قيمة فهرس البداية 4 .

عند تخطي قيمة فهرس الإيقاف، يمتد نطاق العناصر التي تم الوصول إليها حتى العنصر الأخير.

إذا تخطيت قيمتي فهرس البداية والإيقاف 6 وقمت بتحديد عامل النقطتين فقط بين قوسين، فسيتم عرض العناصر بأكملها في القائمة.

الرقم الموجود بعد النقطتين الثانيتين يخبر بايثون أنك ترغب في اختيار زيادة التقطيع الخاصة بك. بشكل افتراضي، تقوم بايثون بتعيين هذه الزيادة على ١، ولكن هذا الرقم بعد النقطتين الثانيتين يسمح لك بتحديد ما تريد أن تكون عليه. استخدام النقطتين المزدوجتين كما هو موضح في 7 يعني عدم وجود قيمة لفهرس البداية وعدم وجود قيمة لفهرس الإيقاف والانتقال بين العناصر بخطوتين. يتم استخراج كل عنصر ثانٍ من القائمة بدءًا من قيمة الفهرس البالغة صفر.

يمكن عرض كافة العناصر الموجودة في القائمة بترتيب عكسي عن طريق تحديد نقطتين مزدوجتين متبوعتين بقيمة فهرس تبلغ −١ 8 . يمكن أيضًا استخدام قيم الفهرس السالبة لقيم فهرس البداية والإيقاف

الدوال المضمنة المستعملة في list

هناك العديد من الوظائف المضمنة التي يمكن تمرير القائمة لها كوسيطة

الدالة	الوصف
len()	تُرجع عدد العناصر في القائمة list
sum()	تُرجع مجموع الأرقام في القائمة list
any()	تُرجع True إذا كانت أي من القيم المنطقية في القائمة list صحيحة.
all()	تُرجع True إذا كانت جميع القيم المنطقية في القائمة صحيحة، وإلا إرجاع False.
stored()	إرجاع نسخة معدلة من القائمة مع ترك القائمة الأصلية دون تغيير.

مثال:

```
>>> lakes = ['superior', 'erie', 'huron', 'ontario', 'powell']
>>> len(lakes)
        5
>>> numbers = [1, 2, 3, 4, 5]
>>> sum(numbers)
     15
>>> max(numbers)
    5
>>> min(numbers)
   1
>>> any([1, 1, 0, 0, 1, 0])
   True
>>> all([1, 1, 1, 1])
   True
>>> lakes_sorted_new = sorted(lakes)
>>> lakes_sorted_new
    ['erie', 'huron', 'ontario', 'powell', 'superior']
```

الحذف

تستخدم ايعاز (del) لحذف المصفوفة كما هي من الذاكرة أو لحذف عناصر محددة منه. عند حذف عنصر في list فإن مترجم لغة بايثون يقوم بإعادة ترتيب عناصره من جديد و تحديث رقم index الخاص بكل عنصر

مثال: في القائمة arr وضعنا أرقام بعدها قمنا بحذف عنصرين منه

```
arr = [10, 20, 30, 40, 50]
del arr[0]
del arr[1]
print(arr)
```

سنحصل على النتيجة التالية عند التشغيل.

[20, 40, 50]

الصورة التالية توضح كيف تم حذف العناصر.

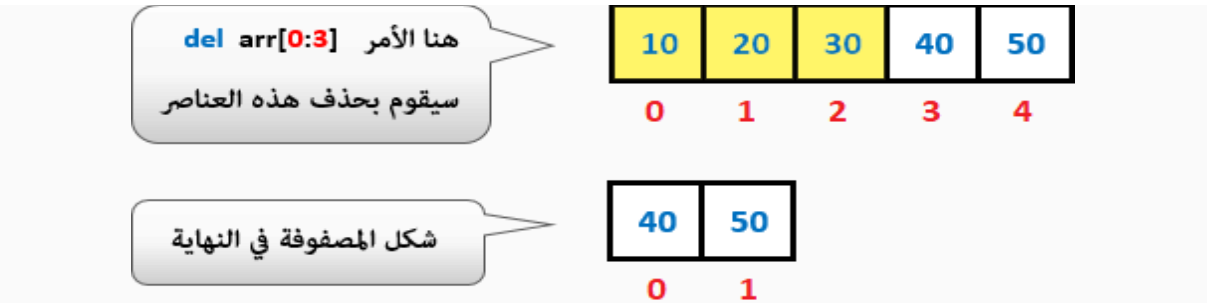
مثال: حذف عناصر محددة من القائمة **arr**

```
arr = [10, 20, 30, 40, 50]
del arr[0:3]
print(arr)
```

سنحصل على النتيجة التالية عند التشغيل.

[40, 50]

الصورة التالية توضح كيف تم حذف العناصر.



مثال: القائمة arr تحتوي على مجموعة من الارقام ثم أحذف جميع العناصر

```
arr = [10, 20, 30, 40, 50]
del arr
print(arr)
```

سنحصل على النتيجة التالية عند التشغيل.

```
NameError: name 'arr' is not defined
```

طرق القائمة method list

يتغير حجم القائمة ديناميكيًا عندما تقوم بإضافة العناصر أو إزالتها وليس هناك حاجة لإدارتها بنفسك.
توفر list مجموعة من الدوال لاستخدامها.

- ✓ **append()**

يضيف عنصرًا واحدًا إلى نهاية القائمة. لا تقوم هذه الطريقة بإرجاع قائمة جديدة وتقوم فقط بتعديل القائمة الأصلية.
الصيغة العامة :

```
list. Append(item)
```

مثال:

```
>>> cities = ["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> cities.append('brussels')
>>> cities
    ['washington', 'paris', 'seattle', 'london', 'washington', 'delhi', 'oslo', 'brussels']
```

- ✓ **count()**

يحسب عدد مرات ظهور العنصر في القائمة ويعيده.
الصيغة العامة

list. Append(item)

مثال:

```
>>> cities = ["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> cities.count('seattle')
    1
```

- ✓ **insert()**

يقوم بإدراج العنصر في الفهرس المحدد، مع نقل العناصر إلى اليمين
الصيغة العامة :

```
list.insert(index,item)
```

مثال:

```
x=["apple" , 5 , 12.5 , True , -2 ]
x.insert (1,"cat")
print(x)
```

سنحصل على النتيجة التالية عند التشغيل:

```
['apple', 'cat', 5, 12.5, True, -2]
```

- ✓ **extend()**

يضيف العناصر الموجودة في القائمة ٢ إلى نهاية القائمة.
الصيغة العامة:

```
list.extend(list2)
```

مثال:

```
>>>cities=["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> more_cities = ["brussels", "copenhagen"]
>>> cities.extend(more_cities)
>>> cities
    ['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'brussels',
'copenhagen']
```

- ✓ **index()**

يبحث عن العنصر المحدد من بداية القائمة ويعيد فهرسه. إذا ظهرت القيمة أكثر من مرة، فسوف تحصل على فهرس القيمة الأولى. إذا لم يكن العنصر موجودًا في القائمة، فسيتم طرح ValueError بهذه الطريقة.
الصيغة العامة

list.index(item)

مثال:

```
>>>cities=["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]

>>> cities.index('washington')
        2
```

✓ **remove()**

يبحث عن المثيل الأول للعنصر المحدد في القائمة ويزيله. إذا لم يكن العنصر موجودًا في القائمة، فسيتم طرح ValueError بهذه الطريقة.

الصيغة العامة :

```
list.remove(item)
```

مثال :

```
>>>cities=["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> cities.remove("brussels")
>>> cities
   ['delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'brussels', 'copenhagen']
```

✓ **sort()**

فرز العناصر الموجودة في القائمة. تقوم هذه الطريقة بتعديل القائمة الأصلية ولا تقوم بإرجاع قائمة جديدة.

الصيغة العامة:

```
list.sort()
```

مثال:

```
>>>cities=["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> cities.sort()
>>> cities
   ['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'washington']
```

✓ **reverse()**

عكس العناصر الموجودة في القائمة. تقوم هذه الطريقة بتعديل القائمة الأصلية ولا تقوم بإرجاع قائمة جديدة.

الصيغة العامة :

```
list.reverse()
```

مثال :

```
>>>cities=["oslo","delhi","washington","london","seattle","paris", "washington"]
>>> cities.reverse()
>>> cities
    ['washington', 'paris', 'seattle', 'london', 'washington', 'delhi', 'oslo']
```

- ✓ **()pop**

يزيل العنصر ويعيده إلى الفهرس المحدد. تقوم هذه الطريقة بإرجاع العنصر الموجود في أقصى اليمين إذا تم حذف الفهرس. الصيغة العامة :

```
list.pop([index])
```

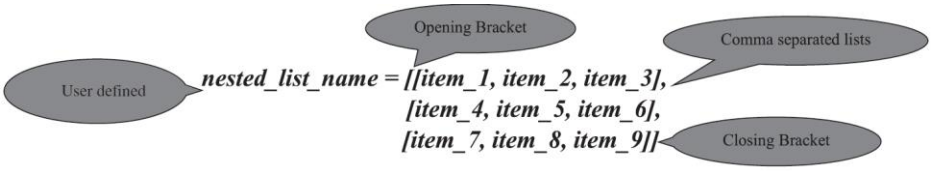
مثال:

```
>>>cities=['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'washington']
>>> cities.pop()
    'washington'
>>> cities
     ['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington']
```

القوائم المتداخلة Nested lists

تسمى القائمة الموجودة داخل قائمة أخرى بالقائمة المتداخلة ويمكنك الحصول على سلوك القوائم المتداخلة في بايثون عن طريق تخزين القوائم ضمن عناصر قائمة أخرى. يمكنك التنقل عبر عناصر القوائم المتداخلة باستخدام حلقة for.

الصيغة العامة :




```
>>> asia = [["India", "Japan", "Korea"],
            ["Srilanka", "Myanmar", "Thailand"],
            ["Cambodia", "Vietnam", "Israel"]]
>>> asia[0]
     ['India', 'Japan', 'Korea']
>>> asia[0][1]
     'Japan'
>>> asia[1][2] = "Philippines"
>>> asia
    [['India', 'Japan', 'Korea'], ['Srilanka', 'Myanmar', 'Philippines'], ['Cambodia',
'Vietnam', 'Israel']]
```

يمكنك الوصول إلى عنصر داخل قائمة هو نفسه داخل قائمة أخرى عن طريق ربط مجموعتين من الأقواس المربعة معًا. على سبيل المثال، في متغير القائمة أعلاه asia، لديك ثلاث قوائم تمثل مصفوفة ٣ × ٣. إذا كنت تريد عرض عناصر القائمة الأولى، فحدد متغير القائمة متبوعًا بفهرس القائمة التي تريد الوصول إليها بين قوسين، مثل asia[0] . إذا أردت الوصول إلى عنصر "اليابان" الموجود داخل القائمة، فأنت بحاجة إلى تحديد فهرس القائمة داخل القائمة ويتبعه فهرس العنصر الموجود في القائمة.