



**الوحدة التدريبية الرابعة
مقدمة في علم البرمجة**

**رمز الوحدة
ADS-U4-CAVT**

المتطلبات السابقة:

لا يوجد

نتائج التعلم:

عند الإنتهاء من دراسة هذه الوحدة واكتساب مهارتها الأدائية والإتجاهات السلوكية الصحيحة والعلوم المهنية المرافقة من خلال التفاعل مع انشطتها وخبراتها المختلفة يصبح المتدرب قادرًا على أداء نتائج التعلم الآتية :

1. توضيح مفهوم البرمجة والتعرف على أنواع لغات البرمجة المختلفة وتقسيماتها.
2. كتابة خوارزمية لحل أية مشكلة ورسم المخطط الإنساني للخوارزمية .
3. التعرف على لغات البرمجة المختلفة وأشهرها Java , C++ , Python .
4. كتابة برامج صحيحة قواعديةً ومنطقياً بمختلف لغات البرمجة والتي تحتوي على أي من أدوات اللغة على سبيل المثال لا الحصر (الجمل الشرطية وجمل الدوران والدوال ... الخ).

مصادر التعلم	الأنشطة التعليمية والتدريبية
الوحدة التدريبية	قراءة المعلومات النظرية
الشبكة العنكبوتية	البحث في الواقع الإلكتروني التعليمية
منصة الكلية التعليمية	روابط التعليم الإلكتروني
المشغل / المختبر	تنفيذ التمارين العملية
سوق العمل	التدريب الميداني

روابط التعلم الإلكتروني

سيتم تزويـد المـتدربـين بـروابـط التـعلم الـإلكـتروـني من خـلـال المـدـرـبـ

مقدمة

البرمجة هي عملية كتابة التعليمات والأوامر الموجهة إلى الحاسوب لتنفيذ مهام محددة. تُعتبر البرمجة حجر الأساس في عالم التكنولوجيا الحديثة، فهي الوسيلة التي تُمكننا من تطوير التطبيقات والواقع الإلكتروني والأنظمة الذكية. تعتمد البرمجة على استخدام لغات برمجية متنوعة مثل Python, Java, C#, حيث تختلف كل لغة في استخدامها وميزاتها.

تساعد البرمجة في أتمتة المهام وتوفير الوقت والجهد، كما تساهم في تطوير حلول مبتكرة لمشاكل معقدة في مجالات عدّة، مثل الذكاء الاصطناعي وتحليل البيانات والأمن السيبراني. يُعد التفكير المنطقي وحل المشكلات من المهارات الأساسية التي يجب أن يمتلكها أي مبرمج، إذ يتطلب بناء الأكواد القدرة على تحليل المتطلبات وتصميم الحلول بفعالية.

مع تطور التكنولوجيا، أصبحت البرمجة أكثر انتشاراً في حياتنا اليومية، حيث تُستخدم في تطوير تطبيقات الهواتف الذكية، الألعاب الإلكترونية، وحتى التحكم بالأجهزة المنزلية الذكية. لذلك، تعلم البرمجة يُعتبر مهارة ضرورية في العصر الرقمي، فهي تفتح آفاقاً واسعة للإبداع والابتكار وتمكن الأفراد من بناء مستقبل أكثر ذكاءً وتطوراً.

١- مقدمة عن لغات البرمجة (**Introduction to Programming Language**) : لغات البرمجة هي الوسيلة الأساسية التي يتم من خلالها التفاعل مع أجهزة الكمبيوتر لتوجيهها للقيام بوظائف معينة. يمكن تعريف لغات البرمجة بأنها مجموعة من القواعد والمفردات التي تتيح للمبرمج كتابة التعليمات والأوامر التي يفهمها الكمبيوتر. تعتبر لغات البرمجة حلقة وصل بين العقول البشرية والنظمومات الرقمية، وتلعب دوراً محورياً في تطور التكنولوجيا وحياتنا اليومية.

• **أقسام لغات البرمجة:** تقسم لغات البرمجة بشكل عام إلى قسمين أساسيين وهما :

أ. **لغات البرمجة منخفضة المستوى (Low-Level Programming Languages)** : هي لغات برمجة قريبة جداً من اللغة التي يفهمها الكمبيوتر مباشرة، وهي تعتمد بشكل رئيسي على معالجة البيانات على مستوى المكونات المادية (Hardware). هذه اللغات توفر للمبرمجين القدرة على التحكم المباشر في مكونات الكمبيوتر المادية مثل وحدة المعالجة المركزية (CPU) والذاكرة. وهذه اللغات تتميز بصعوبة قراءتها وكتابتها من قبل المبرمجين مقارنة باللغات عالية المستوى.

◦ أمثلة عليها : لغة الآلة (Machine Language) ولغة التجميع (Assembly Language).

ب. **لغات البرمجة عالية المستوى (High-Level Programming Languages)** : هي لغات برمجة مصممة لتكون سهلة الاستخدام وقريبة من اللغة الطبيعية، مما يجعلها ملائمة للمبرمجين لإنشاء التطبيقات والبرامج بدون الحاجة للتعامل مع تفاصيل المكونات المادية (Hardware) أو لغة الآلة. وهذه اللغات تتميز بسهولة قراءتها وكتابتها من قبل المبرمجين مقارنة باللغات متدنية المستوى.

◦ أمثلة عليها : C++, Java ، Python .

• **أنواع لغات البرمجة:** يوجد أنواع كثيرة لغات البرمجة تبعاً للاستخدام أو منهجية العمل ويتم تقسيمها إلى قسمين :

أ. **لغات البرمجة الهيكلية (Structured Programming Languages)** : وهي لغات البرمجة التي تعتمد على مفهوم تقسيم البرامج إلى وحدات صغيرة ومنظمة تُعرف بالإجراءات أو الدوال. المبدأ الأساسي من البرمجة الهيكلية هو تحسين قابلية القراءة والصيانة وتقليل التعقيد، وذلك من خلال استخدام تراكيب برمجية واضحة ومنطقية. ومن الأمثلة عليها لغات البرمجة (C , Fortran , Pascal , B)

ب. **لغات البرمجة الكيونية (Object-Oriented)** : تعتمد هذه اللغات على مفهوم برمجة الكائنات (OOP)، وهو نهج تصميمي يركز على الكائنات (Objects) التي تمثل البيانات والوظائف معاً. يهدف هذا النوع من البرمجة إلى

تحسين التنظيم، تسهيل الصيانة، وإعادة الاستخدام من خلال تغليف البيانات والوظائف ذات الصلة في كائن واحد. ومن الأمثلة عليها لغات البرمجة (C++, Java).

- اختيار لغة البرمجة المناسبة: يعتمد اختيار لغة البرمجة المناسبة على عدة عوامل ترتبط بطبيعة المشروع وأهدافه. ولتحديد لغة البرمجة الأمثل لبرمجة أي تطبيق يفضل اخذ المعايير التالية بعين الاعتبار:

أ- تحديد نوع المشروع:

- إذا كنت تعمل على تطوير تطبيقات ويب، قد تكون لغات مثل JavaScript مع إطار عمل مثل React أو Angular أو Python مع Flask مناسبة.
- لتطبيقات الهاتف الذكية، يمكنك استخدام Swift لتطبيقات iOS أو Kotlin/Java لتطبيقات Android.
- إذا كنت بحاجة إلى تطوير أنظمة متكاملة أو برمجيات تعتمد على الأداء العالي، قد تكون C++ خياراً مناسباً.

ب- التوافق مع النظام الأساسي:

- إذا كنت تعمل في بيئة معينة، مثل Windows أو Linux، قد تفضل لغات تتكامل جيداً مع هذا النظام.

ت- مدى سهولة اللغة:

- إذا كنت مبتدئاً، يمكنك اختيار لغة مثل Python بسبب بساطتها وسهولة تعلمها.

ث- الدعم والمجتمع:

- اختر لغة برمجة ذات مجتمع دعم واسع لتتمكن من الوصول إلى الموارد والإجابات عند الحاجة.

ج- الأداء وسرعة التنفيذ:

- إذا كان المشروع يتطلب أداءً عالياً، مثل تطبيقات الألعاب أو أنظمة البيانات الكبيرة، قد تحتاج إلى لغة مثل C++ أو Java.

ح- التوجه المستقبلي:

- اختر لغة برمجة توافق الاتجاهات الحالية ولها مستقبل جيد في الصناعة.

-2- الخوارزميات وخططات الإنساب (Algorithms And Flow chart)

الخوارزميات (Algorithms) : هي مجموعة خطوات مرتبة لحل مشكلة أو تنفيذ مهمة معينة. بمعنى آخر، هي وصف دقيق لسلسلة من التعليمات التي تؤدي إلى تحقيق هدف أو نتيجة. على سبيل المثال، يمكن اعتبار وصفة الطبخ نوعاً من الخوارزميات في حياتنا اليومية؛ فهي تتكون من خطوات متتابعة يجب اتباعها للحصول على الطبق النهائي. في علم الحاسوب، تأخذ الخوارزمية شكل تعليمات واضحة للكمبيوتر حول كيفية حل مسألة أو إنجاز مهمة ما . نحتاج إلى الخوارزميات في كل وقت نريد فيه حل مشكلة بشكل منجي أو تنفيذ مهمة معقدة خطوة بخطوة. فبدون وجود خوارزمية واضحة، قد تكون عملية الحل عشوائية أو غير فعالة. تعتمد البرمجة بشكل أساسي على تصميم الخوارزميات، لأن الكمبيوتر يحتاج إلى تعليمات تفصيلية واضحة ليتمكن من أداء المهام المطلوبة. الخوارزميات تساعدنا أيضاً في تفصيل المشكلة وتحليلها إلى خطوات أبسط، مما يسهل فهمها وحلها. باختصار، نستخدم الخوارزميات عند مواجهة أية مشكلة تتطلب حلاً منظماً، سواء في حياتنا اليومية (طريقة حل مسألة رياضية أو اتباع خطوات إجراء تجاري) أو في كتابة برنامج للحاسوب لحل مهمة معينة.

مثال توضيحي (من الحياة اليومية): تخيل أنك تريد إعداد فنجان من الشاي. يمكنك كتابة خوارزمية بسيطة لهذا الأمر:

خطوات الحل :

1. اغلِّي الماء.
2. ضع كيس الشاي في الكوب.
3. اسكب الماء الساخن في الكوب.
4. انتظر 5 دقائق.
5. اسحب كيس الشاي وأضف السكر أو الحليب حسب الرغبة.
6. حرك الشاي وقدمه.

هذه الخطوات تمثل خوارزمية يومية واضحة لترتيب عملية إعداد الشاي. بنفس الطريقة، عند البرمجة نحدد خطوات واضحة ليقوم بها الحاسوب من البداية حتى تحقيق الهدف المطلوب.

تعريف الخوارزميات من المنظور البرمجي : هي مجموعة من الخطوات أو التعليمات المحددة التي تُستخدم لحل مشكلة معينة أو تنفيذ مهمة محددة. يمكن اعتبارها وصفاً تفصيلياً للطريقة التي يجب اتباعها للوصول إلى النتيجة المطلوبة.

• **خصائص الخوارزمية الصحيحة :**

1. **الوضوح (Clarity):** يجب أن تكون كل خطوة في الخوارزمية محددة بوضوح ولا تحتوي على أي غموض.
2. **الإنتاجية (Finiteness):** يجب أن تكون الخوارزمية من عدد محدود من الخطوات للوصول إلى النتيجة المطلوبة.
3. **المدخلات والمخرجات (Input and Output):** يجب أن يتم تحديد مدخلات الخوارزمية ويجب أن تنتج مخرجات مفهومة بناءً على المدخلات.
4. **القابلية للتكرار (Repeatability):** عند تنفيذ نفس الخوارزمية باستخدام نفس المدخلات، يجب أن تقدم نفس النتيجة.
5. **القابلية للتطبيق (Applicability):** يجب أن تكون الخوارزمية قابلة للكتابة بإحدى لغات البرمجة المعروفة.

• **مكونات الخوارزمية :** تتكون الخوارزميات من مجموعة أقسام رئيسية وهي :

1. **الإسم (Name) :** يجب أن يتم إعطاء الخوارزمية اسم واضح .
2. **المدخلات (Givens) :** المعطيات .
3. **القيم المؤقتة أو الوسطية (Intermediates) :** وهي قيم مؤقتة يتم احتساب قيمها من أجل إيجاد القيم النهائية للمخرجات .
4. **تعريف الخوارزمية (Definition) :** التعريف العام للخوارزمية ويحتوي على اسمها والمدخلات والمخرجات .
5. **خطوات الحل (Method) :** تسلسل خطوات الخوارزمية خطوة بخطوة .
6. **النتائج (Results) :** مخرجات الخوارزمية المطلوبة .

• الكود الكاذب (Pseudo-code)

هو أسلوب لكتابه خطوات الخوارزمية بلغة مبسطة أشبه باللغة الطبيعية أو خليط من اللغة الطبيعية ولغة البرمجة أي أنه لا يتقييد بقواعد نحوية صارمة للغات البرمجة، بل يستخدم عبارات سهلة الفهم لوصف الخطوات.

الهدف من الكود الكاذب هو توضيح منطق الحل دون الانشغال بتفاصيل تركيبية في لغة برمجة معينة. عادةً ما يحمل الكود الكاذب تفاصيل مثل إعلان المتغيرات أو علامات الترميز الخاصة، لأنه موجه للبشر وليس للآلية. من مزايا كتابة الخوارزمية بالكود الكاذب أنها مستقلة عن اللغة البرمجية؛ أي يمكن لأي مبرمج قراءتها وفهم الخوارزمية المقصودة ثم ترجمتها إلى آية لغة برمجة يريدها. كما أن كتابة الكود الكاذب ومراجعته أبسط من محاولة قراءة كود برمجي مليء بالتفاصيل في المراحل الأولى من الحل.

مثال :

Start

Read A, B, C

Sum = A + B + C

Average = Sum / 3

Print "The average is:", Average

End

1. ابدأ
2. اقرأ الأرقام الثلاثة وضعها في المتغيرات A,B,C
3. أوجد مجموع الأرقام الثلاثة وضعها في المتغير SUM
4. أوجد الوسط الحسابي وذلك من خلال المعادلة $SUM/3$ وضع الناتج في المتغير AVG
5. اطبع الناتج AVG
6. النهاية

كما نرى بأن المثال السابق يوضح بأنه من الممكن كتابة الكود الكاذب باللغة الإنجليزية أو باللغة العربية وهو مفهوم بشكل جيد دون الالتزام بقواعد آية لغة برمجة معروفة .

مثال 1 : اكتب خوارزمية لإيجاد ناتج قسمة عددين :

NAME: Division

GIVENS: X, Y

RESULTS: Quotient

INTERMEDIATES: None.

DEFINITION: Quotient := Division(X, Y)

METHOD:

Get X

Get Y

Let Quotient = X/Y

Give Quotient

مثال 2 : اكتب خوارزمية لإيجاد مجموع وحاصل ضرب عددين :

NAME: SumTimes

GIVENS: Num1, Num2

RESULTS: Total, Product

INTERMEDIATES: None

DEFINITION: Total & Product := SumTimes(Num1, Num2)

METHOD:

Get Num1

Get Num2

Let Total = Num1 + Num2

Let Product = Num1 * Num2

Give Total

Give Product

مثال 3 : اكتب خوارزمية لإيجاد معدل الطالب من 100 في 3 مساقات علمًا بأن المساق الأول من 50 علامة و المساق الثاني من 20 علامة المساق الثالث من 70 علامة .

NAME: avgMark

GIVENS: A1, A2, A3

RESULTS: Mark

INTERMEDIATES: Total, MaxMark (Constant)

DEFINITION: Mark := avgMark(A1, A2, A3)

METHOD:

Set MaxMark= 140 (Constant)

Get A1

Get A2

Get A3

Let Total = A1 + A2 + A3

Let Mark = Total/MaxMark * 100

Give Mark

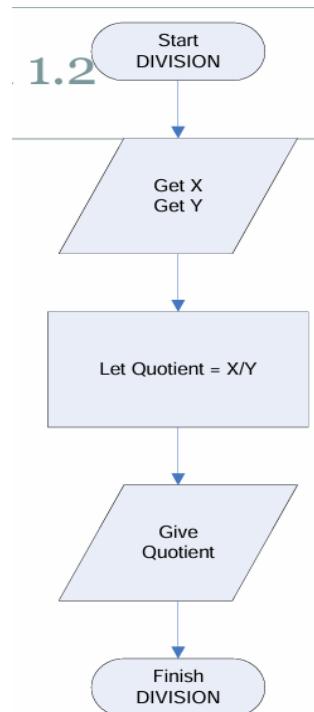
تمرين : اكتب الخوارزمية لقراءة عددين ثم طباعة العدد الأكبر بينهما .

- **المخطط الانسيابي (Flowchart)** : هو رسم توضيحي يمثل خطوات الخوارزمية بشكل مرئي باستخدام رموز هندسية متفق عليها. يساعد المخطط الانسيابي على رؤية تدفق العملية المنطقية بطريقة سهلة، حيث ترسم كل خطوة بشكل صورة (رمز) متصل بالخطوة التي تلها بسهم يشير إلى ترتيب التنفيذ. يتكون أي مخطط انسياطي من أشكال قياسية كل منها يعبر عن نوع معين من الأوامر البرمجية وهي :

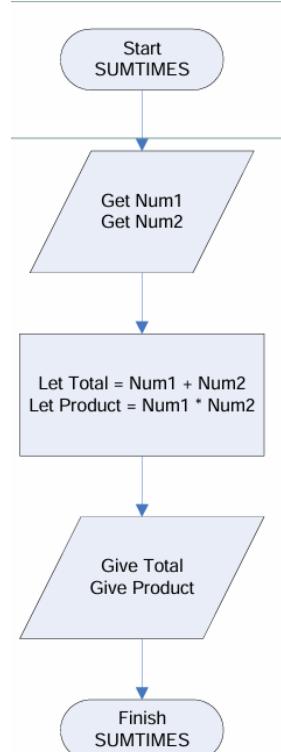
	شكل بيضاوي (بيضاوي): يُمثل بداية الخوارزمية أو نهايتها (يكتب داخل الشكل "Start" للبداية و "Stop" للنهاية).
	مستطيل: يمثل عملية معالجة أو تنفيذ أمر ما (مثل عملية حسابية أو إسناد قيمة).
	متوازي الأضلاع: يمثل عملية إدخال أو إخراج بيانات (على سبيل المثال قراءة قيمة من المستخدم، أو طباعة نتيجة).
	معين (شكل ماسي): يمثل حالة شرطية أو اتخاذ قرار؛ يتفرع منه خطان أو أكثر حسب نتيجة الشرط (مثلاً "نعم" / "لا").
	سهم : يمثل انتقال التدفق من خطوة إلى أخرى باتجاه معين.

استخدام هذه الرموز الموحدة يجعل قراءة المخطط الانسيابي وفهمه أسهل بكثير، حتى لمن لا يجيد لغة برمجة. فعندما ننظر إلى المخطط ندرك البداية والنهاية والعمليات والقرارات المتخذة أثناء الحل . المخططات الانسيابية مفيدة أيضًا في توثيق البرامج وتصميمها، فهي تعطي نظرة شاملة على منطق البرنامج قبل كتابته فعلياً بالكود. إذ يمكن للمبرمج رسم المخطط الانسيابي ثم استخدامه كمرجع عند كتابة البرنامج، أو مشاركته مع الآخرين لفهم طريقة الحل المقترنة.

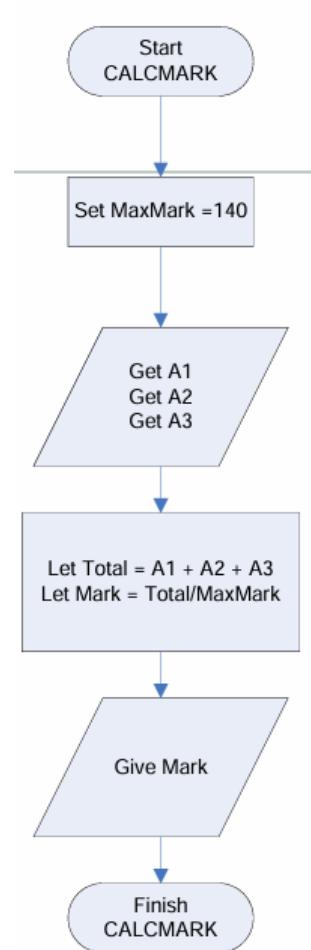
مثال 1 : ارسم المخطط الإنسيابي لخوارزمية إيجاد ناتج قسمة عددين :



مثال 2 : ارسم المخطط الإنسيابي لخوارزمية إيجاد مجموع وحاصل ضرب عددين :



مثال 3 : ارسم المخطط الإنسيابي لخوارزمية إيجاد معدل الطالب من 3 مساقات علماء بأن المساق الأول من 50 علامة و المساق الثاني من 20 علامة المساق الثالث من 70 علامة .



تمرين : ارسم المخطط الإنسيابي لخوارزمية لقراءة عددين ثم طباعة العدد الأكبر بينهما .

3- المفاهيم الأساسية في لغات البرمجة :

- **المتغيرات وأنواع البيانات (Variables & Data Types)** :
- **المتغير** : هو عبارة عن مكان في الذاكرة يتم تخزين قيمة فيه ويعطى اسمًا. حيث يمكن تغيير هذه القيمة خلال تنفيذ البرنامج علماً بأن لكل متغير نوع بيانات ويتم تحديد النوع بناءً على القيمة التي سيتم تخزينها بداخله.

تخيل أن المتغير هو مثل صندوق له اسم، يمكننا وضع شيء بداخله (مثل رقم أو كلمة)، ويمكننا فتحه وتغييره عند الحاجة.

مثال توضيحي : نفترض أن لديك صندوق مكتوب عليه "العمر" :

في البداية، تضع فيه الرقم 20 ← الآن المتغير العمر = 20

بعد سنة، تفتح الصندوق وتغير الرقم إلى 21 ← الآن العمر = 21

ملاحظات هامة :

- أ. يجب أن يكون اسم المتغير مطابق لقواعد تسمية المتغيرات في لغة البرمجة المستخدمة . " سيتم توضيحها لاحقا"
- ب. يجب تحديد نوع المتغير بناءً على الأنواع المتوفرة في لغة البرمجة المستخدمة وحسب الحاجة .

- **أنواع البيانات (Data Types)** : يوجد الكثير من أنواع البيانات المتوفرة في لغات البرمجة والتي من الممكن استخدامها حسب الحاجة ، والكثير منها مشترك في لغات البرمجة ونستعرض منها ما يلي :

أمثلة	الوصف Description	نوع البيانات Data Type
10 , -15 , 678	عدد صحيح	Integer
10.0 , 68.326 , -526.12	عدد عشري	Float or Double
'A' , '@' , '+'	حرف	Character
"Hello" , "how Are You"	نص	String
True, False	قيمة منطقية	Boolean

• قواعد تسمية المتغيرات (Variable Naming Rules) : كما تحدثنا سابقاً بأنه هنالك قواعد لتسمية المتغيرات

وهذه القواعد تختلف من لغة إلى أخرى ولكن غالباً ما تشتراك لغات البرمجة في الكثير من القواعد ومنها :

- . أ. يجب أن يكون الاسم باللغة الانجليزية وخلط من الأرقام والحروف و " _ " .
- . ب. لا يجوز أن يبدأ برقم .
- . ج. لا يجوز أن تحتوي على رموز خاصة مثل \$,%,& .
- . د. لا يسمح باستخدام المسافة .
- . هـ. يجب أن لا يكون كلمة محظوظة مثل Int, If , while .
- . و. في بعض اللغات الأسماء حساسة للأحرف أي أن المتغير Avg مختلف عن المتغير avg .
- . زـ. يفضل أن يعطى المتغير اسمًا وضحاً يدل على الغرض من استخدامه وعلى المحتوى .

أمثلة	
أسماء متغيرات غير صحيحة	أسماء متغيرات صحيحة
2age	age
Sum	sum
Student#name	Student_name
Cin	_car_type
while	Color2

مثال 1 : تعريف متغيرات بلغة البرمجة C++

```
int myNum = 5;                                // Integer (whole number)
float myFloatNum = 5.99;                         // Floating point number
double myDoubleNum = 9.98;                        // Floating point number
char myLetter = 'D';                            // Character
bool myBoolean = true;                           // Boolean
string myText = "Hello";                         // String
```

مثال 2 : تعريف متغيرات بلغة البرمجة Java

```
int myNum = 5;           // Integer (whole number)
float myFloatNum = 5.99f; // Floating point number
char myLetter = 'D';     // Character
boolean myBool = true;   // Boolean
String myText = "Hello"; // String
```

مثال 3 : ملاحظة بعض لغات البرمجة لا يتم تحديد نوع للمتغير وإنما يتم تحديده بشكل تلقائي مثل لغة البرمجة

. Python

```
x = 5
```

- جمل الإدخال والإخراج (Input / Output) : كما درسنا سابقاً في الخوارزميات فإن كل خوارزمية تحتوي على جمل لإدخال المعطيات و جمل لإظهار النتائج أو طباعتها وسنعرف الآن على جمل الإدخال والإخراج لمجموعة من لغات البرمجة.

أمثلة :

لغة C++

تحتوي لغة C++ على جملة الإدخال وهي `CIN` وجملة الإخراج وهي `COUT` ومثال عملي على الجملتين

```
int x, y;
int sum;
cout << "Type a number: ";
cin >> x;
cout << "Type another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

لغة Java

```
Scanner input = new Scanner(System.in);
int age = input.nextInt();
System.out.println(age);
System.out.println(3);
System.out.println(358);
System.out.println("Asmaa");
```

لغة Python

```
age = int(input(" : أدخل عمرك"))
print("عمرك هو:", age)
```

تمرين : اكتب البرامج التالية على البيئة الحقيقة وتتبع النتائج

لغة C++

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

int main()
{
    cout << "Hello World!";
    return 0;
}
```

لغة Java

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

لغة Python

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

- العمليات الحسابية (Arithmetic Operations) : تُستخدم العمليات الحسابية لإجراء العمليات الرياضية على الأرقام، وهي من أساسيات البرمجة .

مثال	الرمز	العملية
$5+5=10$	+	الجمع
$10-5=5$	-	الطرح
$2*3=6$	*	الضرب
$10/2=5$	/	القسمة
$10\%3=1$	%	باقي القسمة

: أمثلة :

لغة C++

```
int a = 10, b = 3;
cout << a + b << endl; // 13
cout << a - b << endl; // 7
cout << a * b << endl; // 30
cout << a / b << endl; // 3 (قسمة عددين صحيحين)
cout << a % b << endl; // 1
```

لغة Java

```
int a = 10, b = 3;
System.out.println(a + b); // 13
System.out.println(a - b); // 7
System.out.println(a * b); // 30
System.out.println(a / b); // 3
System.out.println(a % b); // 1
```

```

a = 10
b = 3
print(a + b)      # الناتج: 13
print(a - b)      # الناتج: 7
print(a * b)      # الناتج: 30
print(a / b)       # الناتج: 3.333...
print(a % b)       # الناتج: 1

```

- العمليات المنطقية: تُستخدم العمليات المنطقية لاتخاذ قرارات في البرمجة، و نتيجتها إما صحيح (True) أو خطأ (False). وهي نوعان:

- المقارنات المنطقية (Comparison Operators)

النتيجة	مثال على تعبير	الرمز	العملية
False	6==5	==	يساوي
True	6!=5	!=	لا يساوي
False	3<2	<	أكبر من
True	4>1	>	أصغر من
True	12<=20	=<	أكبر أو يساوي
False	12>=20	=>	أصغر أو يساوي

C++ لغة

```
int a = 5, b = 3;  
cout << (a == b) << endl; // 0 (False)  
cout << (a != b) << endl; // 1 (True)  
cout << (a > b) << endl; // 1  
cout << (a < b) << endl; // 0
```

Java لغة

```
int a = 5, b = 3;  
System.out.println(a == b); // false  
System.out.println(a != b); // true  
System.out.println(a > b); // true  
System.out.println(a < b); // false
```

Python لغة

```
a = 5  
b = 3  
print(a == b) # False  
print(a != b) # True  
print(a > b) # True  
print(a < b) # False
```

- المعاملات المنطقية (Logical Operations) : هناك عدة عمليات منطقية رئيسية، منها:

- عملية (AND): تُنتج نتيجة صحيحة (True) فقط إذا كانت كلا المدخلات صحيحة.
- عملية (OR): تُنتج نتيجة صحيحة (True) فقط إذا كان أي من المدخلات صحيحة.
- عملية (NOT): تُعكس قيمة المدخل، فإذا كان المدخل صحيحاً (True) يصبح خاطئًا (False)، والعكس صحيح.

أمثلة :

لـغـة C++

النـتـيـجـة	مـثـال عـلـى تـعـبـير	الـرـمـز	الـعـمـلـيـة
	$x < 5 \&\& x < 10$	<code>&&</code>	عملية (AND)
	$x < 5 x < 4$	<code> </code>	عملية (OR)
	$!(x < 5 \&\& x < 10)$	<code>!</code>	عملية (NOT)

لـغـة Java

النـتـيـجـة	مـثـال عـلـى تـعـبـير	الـرـمـز	الـعـمـلـيـة
	$x < 5 \&\& x < 10$	<code>&&</code>	عملية (AND)
	$x < 5 x < 4$	<code> </code>	عملية (OR)
	$!(x < 5 \&\& x < 10)$	<code>!</code>	عملية (NOT)

لـغـة Python

النـتـيـجـة	مـثـال عـلـى تـعـبـير	الـرـمـز	الـعـمـلـيـة
	$x < 5 \text{ AND } x < 10$	<code>AND</code>	عملية (AND)
	$x < 5 \text{ OR } x < 4$	<code>OR</code>	عملية (OR)
	<code>not(x < 5 and x < 10)</code>	<code>NOT</code>	عملية (NOT)

تمرين : أوجد النتيجة في الجداول السابقة إذا كانت قيمة $X = 5$ ثم أوجد النتيجة مرة أخرى إذا كانت قيمة $X = 7$.

- **الأخطاء البرمجية (Programming Errors)**: الأخطاء البرمجية هي أية مشاكل أو عيوب تظهر أثناء تطوير البرامج، مما يؤدي إلى عدم عمل البرنامج كما هو متوقع. يمكن تصنيف الأخطاء البرمجية إلى نوعين رئисيين بناءً على طبيعتها:
 - **الأخطاء اللغوية (Syntax Errors)** : وهي أخطاء تحدث عندما يكون هناك خطأ في كتابة الكود بما يخالف قواعد اللغة البرمجية.

أمثلة:

- نسيان إضافة فاصلة منقوطة (;) في نهاية الجملة.
- فتح قوس بدون إغلاقه.
- استخدام كلمات محظوظة بطريقة خاطئة.
- يتم اكتشافها أثناء الترجمة (Compilation) من قبل المترجم (Compiler).

لغة C++

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

int main()
{
    cout << "Hello" << endl
    return 0;
}
```

- الأخطاء المنطقية (Logical Errors) : وهي أخطاء تحدث عندما يكون الكود مكتوبًا بشكل صحيح من ناحية القواعد البرمجية، لكنه لا يعمل بالشكل المتوقع.

أمثلة:

- استخدام معادلة خاطئة لحساب نتيجة.
- برمجيات تتجاهل حالة معينة أو إدخال غير متوقع.
- لا يتم اكتشافها أثناء الترجمة، وإنما تظهر أثناء تشغيل البرنامج.

لغة Python

```
grade1 = 80
grade2 = 90
grade3 = 100
average = grade1 + grade2 + grade3 / 3
print("المتوسط", average)
```

الخطأ: القسمة تمت فقط على `grade3` وليس على المجموع الكامل.

 النتائج الخطأ: المتوسط = $203.33 = 33.33 + 90 + 80 = (3 / 100) + 90 + 80$ 

الحل الصحيح:

 الأقواس ضرورية # `average = (grade1 + grade2 + grade3) / 3`

 النتائج الصحيحة: المتوسط = $90.0 = 3 / (100 + 90 + 80)$ 

4 - الجمل الشرطية (Conditional Statements) : هي جمل تُستخدم لاتخاذ قرارات بناءً على شروط معينة. إذا تحقق الشرط، يتم تنفيذ مجموعة معينة من الأوامر. وفي حالة عدم تتحقق الشرط، يمكن تنفيذ أوامر بديلة وهنالك مجموعة جمل شرطية تختلف قواعدها من لغة برمجة إلى أخرى ولكتها تؤدي نفس الغرض. يمكن استخدام الجمل الشرطية بطرق متعددة، وهي :

- جملة الشرط (IF) : تُستخدم للتحقق من شرط وتنفيذ الكود إذا كان الشرط صحيحًا وإذا كان الشرط خطأ لا يتم تنفيذ أي شيء .

أمثلة :

لغة C++

```
int x = 20;
int y = 18;
if (x > y) {
    cout << "x is greater than y";
}
```

لغة Java

```
int x = 20;
int y = 18;
if (x > y) {
    System.out.println("x is greater than y");
}
```

لغة Python

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

- جملة الشرط (IF Else) : تُستخدم للتحقق من شرط وتنفيذ الكود إذا كان الشرط صحيحًا وإذا كان الشرط خطأ يتم تنفيذ جمل Else .

أمثلة :

لغة C++

```
int time = 20;
if (time < 18) {
    cout << "Good day.";
} else {
    cout << "Good evening.";
}
```

لغة Java

```
int time = 20;
if (time < 18) {
    System.out.println("Good day.");
} else {
    System.out.println("Good evening.");
}
```

لغة Python

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
else :
    print("a and b are equal")
```

- جملة الشرط (Nested If) : تُستخدم للتحقق من شرط وتنفيذ الكود إذا كان الشرط صحيحاً وإذا كان الشرط خطأ يتم السؤال عن شرط آخر وهكذا

أمثلة :

لغة C++

```
int time = 22;
if (time < 10) {
    cout << "Good morning.";
} else if (time < 20) {
    cout << "Good day.";
} else {
    cout << "Good evening.";
}
```

Java لغة

```
int time = 22;
if (time < 10) {
    System.out.println("Good morning.");
} else if (time < 18) {
    System.out.println("Good day.");
} else {
    System.out.println("Good evening.");
}
```

Python لغة

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

- جملة التبديل (Match / Switch) : تُستخدم جمل التبديل لاختيار أحد الفروع بناءً على قيمة معينة.

وهي بديل عن استخدام if-else المتكررة إذا كانت المقارنة على نفس المتغير.

أمثلة :

C++ لغة

```
int day = 4;
switch (day) {
    case 1:
        cout << "Monday";
        break;
    case 2:
        cout << "Tuesday";
        break;
    case 3:
        cout << "Wednesday";
        break;
    case 4:
        cout << "Thursday";
        break;
    case 5:
        cout << "Friday";
        break;
    case 6:
        cout << "Saturday";
        break;
    case 7:
        cout << "Sunday";
        break;
}
```

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
}
```

```
day = 4
match day:
    case 1:
        print("Monday")
    case 2:
        print("Tuesday")
    case 3:
        print("Wednesday")
    case 4:
        print("Thursday")
    case 5:
        print("Friday")
    case 6:
        print("Saturday")
    case 7:
        print("Sunday")
```

5 – جمل الدوران (Loops) : جمل الدوران هي عبارة عن هيكل برمجي يُستخدم لتكرار تنفيذ جزء معين من الكود البرمجي، بناءً على شروط محددة مسبقاً. يمكن لجمل الدوران تنفيذ تعليمات واحدة أو مجموعة من التعليمات عدة مرات، مما يوفر مرونة وأداءً أعلى عند التعامل معمجموعات بيانات أو عمليات متكررة .

• كيفية عمل جمل الدوران :

- أ. تبدأ الحلقة (الدوران) عادة بتهيئة قيمة أولية أو حالة البداية.
- ب. يتم التتحقق من شرط الحلقة قبل تنفيذ التعليمات (أو بعدها حسب نوع الحلقة).
- ج. إذا كان الشرط صحيحًا، يتم تنفيذ التعليمات داخل الحلقة.
- د. بعد ذلك، يتم تحديث الحالة (مثل زيادة أو نقصان متغير التحكم).
- هـ. تستمرة العملية حتى يصبح الشرط غير صحيح، وعندما تخرج الحلقة من التكرار.

• أهمية جمل الدوران:

- أ. زيادة الكفاءة: حيث تسهل كتابة برامج تعالج كميات كبيرة من البيانات.
- ب. الديناميكية: تتمكن من تشغيل كود بناءً على مدخلات أو حالات مختلفة.
- ج. تنظيم الكود: تقلل من التكرار اليدوي وتجعل الكود أكثر نظافة وسهولة لفهم.

• أنواع جمل الدوران في لغات البرمجة:

• جملة الدوران (For) : تستخدم عندما يكون لديك عدد محدد مسبقاً للتكرار، غالباً ما تحتوي على ثلاثة مكونات:

- أ. عداد (Counter) ويتم إعطاؤه قيمة ابتدائية .
- ب. التتحقق من الشرط الخاص بالعداد .
- ج. تحديث قيمة العدد .

أمثله :

C++ لغة

```
for (int i = 0; i < 5; i++) {  
    cout << i << "\n";  
}
```

Java لغة

```
for (int i = 0; i <= 10; i = i + 2) {  
    System.out.println(i);  
}
```

Python لغة

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)  
    if x == "banana":  
        break
```

- جملة الدوران (While) : تُستخدم لتكرار الكود بناءً على شرط معين. تستمر الحلقة (الدوران) في العمل طالما أن الشرط صحيح وأحياناً من الممكن أن تحتوي جملة الدوران While على عدد مثل جملة الدوران For .

ملاحظة : الدوران من الممكن أن لا يتم تنفيذه أبداً وذلك في حالة عدم انطباق الشرط من أول مرة .

أمثلة :

لغة C++

```
int i = 0;
while (i < 5) {
    cout << i << "\n";
    i++;
}
```

لغة Java

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

لغة Python

```
i = 1
while i < 6:
    print(i)
    i += 1
```

- جملة الدوران (Do While) : تُستخدم لـتكرار الكود بناءً على شرط معين. تستمر الحلقة (الدوران) في العمل طالما أن الشرط صحيح وأحياناً من الممكن أن تحتوي جملة الدوران Do While على عدد مثل جملة الدوران For.

ملاحظة : الدوران يجب أن يتم تنفيذه مره واحدة على الأقل لأن الشرط يتم التأكد من صحته بعد تنفيذ جملة الدوران .

أمثلة :

لغة C++

```
int i = 0;
do {
    cout << i << "\n";
    i++;
}
while (i < 5);
```

لغة Java

```
int i = 0;
do {
    System.out.println(i);
    i++;
}
while (i < 5);
```

لغة Python

```
# Example of a do-while equivalent in Python
number = 0
while True:  # Mimicking the "do" part
    print("Current number is:", number)
    number += 1
    if number >= 5:  # Mimicking the "while" condition
        break
```

- نقطه هامة: عند كتابة جمل الدوران من الممكن أن يحصل خطأ منطقى وهو حدوث دوران غير متهى (Infinite Loops) ويحدث عندما يتم نسيان تحديث قيمة العداد داخل الحلقة، مما يؤدي إلى استمرارها.

أمثلة:

لغة C++

```
int i = 0;
do {
    cout << i << "\n";
    i--;
}
while (i < 5);
```

لغة Java

```
int i = 0;
while (i < 5) {
    System.out.println(i);
}
```

لغة Python

```
while True:
    for i in range(5):
        print("i,", "تكرار داخلي")
# حلقة لا نهائية → لا يوجد شرط للخروج من while
```

• جمل (Break and Continue) :

- جملة التوقف (Break) : تُستخدم جملة break لإنهاء الحلقة (الدوران) بشكل فوري. بمجرد تنفيذها، يتم الخروج من الحلقة بغض النظر عن باقي الشروط أو التكرارات المتبقية.

لـغـة C++

```
for (int i = 0; i < 10; i++) {  
    if (i == 4) {  
        break;  
    }  
    cout << i << "\n";  
}
```

لـغـة Java

```
int i = 0;  
while (i < 10) {  
    System.out.println(i);  
    i++;  
    if (i == 4) {  
        break;  
    }  
}
```

لـغـة Python

```
for i in range(1, 10):  
    if i == 5:  
        break  
    print(i)
```

- جملة الإستمراية (Continue) : تُستخدم جملة continue لتخطي التكرار الحالي للحلقة (الدوران) والانتقال إلى التكرار التالي. حيث يتم تجاهل باقي الكود بعد جملة continue داخل نفس التكرار.

لغة C++

```
int i = 0;
while (i < 10) {
    if (i == 4) {
        i++;
        continue;
    }
    cout << i << "\n";
    i++;
}
```

لغة Java

```
for (int i = 0; i < 10; i++) {
    if (i == 4) {
        continue;
    }
    System.out.println(i);
}
```

لغة Python

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

٦- هياكل البيانات (Data Structure): هيكلة البيانات هي أحد الأساسيات في علوم الحاسوب وتعتبر الطريقة التي يتم بها تنظيم البيانات وترتيبها في الذاكرة لتسهيل الوصول إليها وإدارتها بكفاءة. يمكن تصنيفها إلى عدة أنواع، ومن أبرزها المصفوفات (Arrays)، والمؤشرات (Pointers).

- **المصفوفات (Arrays / List)** : المصفوفة هي نوع بسيط من هيكلة البيانات تُستخدم لتخزين مجموعة من العناصر التي لها نفس النوع في ذاكرة متغيرة. تتيح الوصول السريع إلى العناصر عبر الفهرسة (Index)، ولكنها تكون ثابتة الحجم، مما قد يكون قيدها في بعض التطبيقات. تعتبر المصفوفات مفيدة جدًا عندما تحتاج إلى تنظيم بيانات متسلسلة ومحددة الحجم. ومن الممكن أن تكون المصفوفة ذات بعد واحد (One Dimension) أو مصفوفة ذات بعدين (Two Dimension Array).

C++ لغة

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
cout << cars[0];

//this program to calculates the average of different ages:
int ages[8] = {20, 22, 18, 35, 48, 26, 87, 70};

float avg, sum = 0;
int i;

// Get the length of the array
int length = sizeof(ages) / sizeof(ages[0]);

// Loop through the elements of the array
for (int age : ages) {
    sum += age;
}

// Calculate the average by dividing the sum by the length
avg = sum / length;

// Print the average
cout << "The average age is: " << avg << "\n";
```

مثال على مصفوفة ثنائية البعد:

```
string letters[2][4] = {  
    { "A", "B", "C", "D" },  
    { "E", "F", "G", "H" }  
};  
  
for (int i = 0; i < 2; i++) {  
    for (int j = 0; j < 4; j++) {  
        cout << letters[i][j] << "\n";  
    }  
}
```

لغة Java

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  
cars[0] = "Opel";  
System.out.println(cars[0]);
```

```
// This program finds the lowest age among different ages:  
int ages[] = {20, 22, 18, 35, 48, 26, 87, 70};  
// Get the length of the array  
int length = ages.length;  
// Create a 'lowest age' variable and assign the first array element of  
ages to it  
int lowestAge = ages[0];  
// Loop through the elements of the ages array to find the lowest age  
for (int age : ages) {  
    // Check if the current age is smaller than the current 'lowest age'  
    if (lowestAge > age) {  
        // If the smaller age is found, update 'lowest age' with that element  
        lowestAge = age;  
    }  
}  
// Output the value of the lowest age  
System.out.println("The lowest age in the array is: " + lowestAge);
```

مثال على مصفوفة ثنائية البعد :

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };  
for (int i = 0; i < myNumbers.length; ++i) {  
    for (int j = 0; j < myNumbers[i].length; ++j) {  
        System.out.println(myNumbers[i][j]);  
    }  
}
```

لغة Python

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)
```

```
// Add item  
thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)
```

- المؤشرات (Pointers) : المؤشر هو متغير يستخدم لتخزين عنوان ذاكرة عنصر آخر. تُستخدم المؤشرات بشكل واسع في لغات البرمجة للربط بين هيكلية البيانات ولتطبيق العمليات مثل الديناميكية في تخصيص الذاكرة (Dynamic Memory Allocation). المؤشرات قوية ولكنها قد تسبب أخطاء مثل الوصول إلى عناوين غير صالحة إذا لم يتم استخدامها بحذر.

لغة C++

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

int main() {
    string food = "Pizza"; // A string variable
    string* ptr = &food; /* A pointer variable that stores the address of
food */

    // Output the value of food
    cout << food << "\n";
    // Output the memory address of food
    cout << &food << "\n";
    // Output the memory address of food with the pointer
    cout << ptr << "\n";
    // Output the values of food with the pointer
    cout << *ptr << "\n";
    return 0;
}
```

7 - الوظائف والدوال (Function & Methods): الدوال هي جزء أساسي في البرمجة، وتساعد في تنظيم الكود، تحسين قراءته، وتقليل التكرار. يمكن تقسيمها إلى نوعين رئيسيين: الدوال الجاهزة (Built-in Functions) والدوال التي يكتبها المبرمج (User-defined Functions).

- إن استخدام الدوال في البرمجة لها فوائد عديدة تجعل الكود أكثر تنظيماً وفعالية. إليك أبرز الفوائد:
 - أ. إعادة الاستخدام (Reusability): عندما تكتب دالة، يمكنك استخدامها عدة مرات في أماكن مختلفة من البرنامج دون الحاجة إلى إعادة كتابة الكود نفسه. هذا يوفر الوقت والجهد.
 - ب. تقليل التكرار (Avoid Redundancy): بدلاً من كتابة نفس الكود مراضاً وتكراراً، يمكنك وضعه في دالة واستدعاؤها عند الحاجة. هذا يجعل الكود أكثر نظافة وسهولة في القراءة.
 - ج. تنظيم الكود (Code Organization): الدوال تساعد في تقسيم البرنامج إلى أجزاء صغيرة واضحة. كل دالة يمكن أن تكون مسؤولة عن مهمة معينة، مما يجعل البرنامج أكثر قابلية للفهم والصيانة.
 - د. الصيانة والتعديل (Maintainability): إذا كنت بحاجة إلى تعديل وظيفة معينة، يمكنك تعديل الدالة فقط بدلاً من تعديل الكود في أماكن متعددة.
 - هـ. التعاون (Collaboration): في المشاريع التي يعمل عليها فريق من المطوريين، يمكن لكل فرد كتابة دوال خاصة به، مما يسهل التعاون وتقسيم العمل.
 - وـ. تسهيل الاختبار (Testing): يمكنك اختبار دالة معينة بشكل منفصل للتأكد من أنها تعمل بشكل صحيح، بدلاً من اختبار البرنامج بأكمله في كل مرة.
 - زـ. الوضوح (Readability): عندما تقرأ كود يحتوي على دوال، يكون من السهل فهم الوظائف المختلفة للبرنامج، لأن أسماء الدوال غالباً ما تعبّر عن الغرض منها.
 - حـ. المرونة (Flexibility): الدوال تسهل التعامل مع البيانات المختلفة عن طريق تمرير المعاملات (Parameters)، مما يجعل الكود أكثر ديناميكية.
- الدوال الجاهزة (Built-in Functions): هذه الدوال تأتي مع اللغة نفسها أو مع مكتباتها المدمجة و يتم تصميمها لأداء وظائف شائعة مثل العمليات الحسابية، معالجة النصوص، أو التعامل مع الملفات.

أمثلة :

C++ لغة

```
include <cmath> // مكتبة الرياضيات

int main() {
    double number = 25;
    double result = sqrt(number); // دالة لحساب الجذر التربيعي
    std::cout << "الجذر التربيعي لـ " << number << " هو " << result <<
std::endl;
    return 0;
```



للاطلاع على دوال أكثر في لغة C++ اطلع على الرابط التالي :

https://www.w3schools.com/cpp/cpp_ref_math.asp

Java لغة

```
public class Main {
    public static void main(String[] args) {
        double number = 16;
        double result = Math.sqrt(number); // دالة لحساب الجذر التربيعي
        System.out.println("الجذر التربيعي لـ " + number + " هو " + result);
    }
}
```



للاطلاع على دوال أكثر في لغة JAVA اطلع على الرابط التالي :

https://www.w3schools.com/java/java_ref_math.asp

```
name = "أسماء"  
length = len(name) # دالة لحساب عدد الأحرف في النص  
print("،", length): عدد الأحرف في الاسم"
```



للاطلاع على دوال أكثر في لغة Python اطلع على الرابط التالي :

https://www.w3schools.com/python/module_random.asp

- مميزات الدوال الجاهزة :

أ. جاهزة للاستخدام، لا تحتاج إلى كتابتها بنفسك.

ب. موثوقة وتتمتع بأداء عالي.

- الدوال التي يكتبها المبرمج (User-defined Functions) : هذه الدوال يتم إنشاؤها بواسطة المبرمج لتناسب احتياجات محددة. وهي توفر إمكانية تخصيص الكود لحل مشكلات معينة أو تنفيذ مهام محددة.

- المكونات الأساسية للدوال التي يكتبها المبرمج :

أ. اسم الدالة (Function Name): اسم يعبر عن وظيفتها.

ب. المعاملات (Parameters): قيم يمكن تمريرها للدالة لتعمل عليها.

ج. القيمة المعادة (Return Value): النتيجة التي تعدها الدالة.

د. الجسم: الكود الذي ينفذ داخل الدالة.

أمثلة :

C++ لغة

```
int main() {
    myFunction();
    return 0;
}

void myFunction() {
    cout << "I just got executed!";
}
```

```
int myFunction(int x, int y) {
    return x + y;
}

int main() {
    cout << myFunction(5, 3);
    return 0;
}
```

```

void swapNums(int &x, int &y) {
    int z = x;
    x = y;
    y = z;
}

int main() {
    int firstNum = 10;
    int secondNum = 20;
    cout << "Before swap: " << "\n";
    cout << firstNum << secondNum << "\n";
    // Call the function, which will change the values of firstNum and
    secondNum
    swapNums(firstNum, secondNum);
    cout << "After swap: " << "\n";
    cout << firstNum << secondNum << "\n";

    return 0;
}

```

لغة Java

```

public class Main {
    static void myMethod() {
        System.out.println("I just got executed!");
    }
    public static void main(String[] args) {
        myMethod();
    }
}

```

```
public class Main {  
    static void myMethod(String fname, int age) {  
        System.out.println(fname + " is " + age);  
    }  
    public static void main(String[] args) {  
        myMethod("Liam", 5);  
        myMethod("Jenny", 8);  
        myMethod("Anja", 31);  
    }  
}
```

```
public class Main {  
    static int myMethod(int x, int y) {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        int z = myMethod(5, 3);  
        System.out.println(z);  
    }  
}
```

```
def my_function():
    print("Hello from a function")
my_function()
```

```
def my_function(fname):
    print(fname + " are Application Development Trainer")
my_function("CAVT")
my_function("Muath")
my_function("Asmaa")
```

```
"""If the number of keyword arguments is unknown, add a double ** before the
parameter name:"""
def my_function(**kid):
    print("His last name is " + kid["lname"])
my_function(fname = "Tobias", lname = "Refsnes")
```

• تعدد أشكال الدالة (Function overloading) : هي ميزة في بعض لغات البرمجة تسمح بإنشاء عدة دوال بنفس الاسم ولكن تنفذ بطرق مختلفة بناءً على نوع أو عدد المعلمات (Arguments) التي يتم تمريرها إلى الدالة. هذا يعني أن نفس اسم الدالة يمكن أن يؤدي وظائف مختلفة حسب السياق الذي يتم استدعاؤها فيه.

• مزايا تعدد الأشكال (Function Overloading) :

- أ. تحسين قابلية القراءة: بدلاً من استخدام أسماء مختلفة للدوال التي تؤدي وظائف مشابهة، يمكن استخدام اسم واحد مع اختلاف في المعلمات.
- ب. تقليل التعقيد: يجعل الكود أكثر تنظيماً وسهولة في الفهم.
- ت. إعادة الاستخدام: يمكن استخدام نفس الدالة لأغراض متعددة بناءً على نوع أو عدد المعلمات.

• كيفية عمل (Function Overloading) : عندما يتم تعريف أكثر من دالة ولها نفس الاسم ولكن تختلف بعدد المعلمات أو نوعها فإنه سيتم تحديد الدالة التي ستستدعي بناءً على أحد العوامل التالية :

- أ. عدد المعلمات: إذا كانت الدالة تحتوي على عدد مختلف من المعلمات.
- ب. نوع المعلمات: إذا كانت المعلمات تختلف في النوع مثل `double`, `float`, `int`.
- ت. ترتيب المعلمات: إذا كان ترتيب المعلمات مختلفاً.

C++ لغة

```
int plusFunc(int x, int y) {  
    return x + y;  
}  
  
double plusFunc(double x, double y) {  
    return x + y;  
}  
  
int main() {  
    int myNum1 = plusFunc(8, 5);  
    double myNum2 = plusFunc(4.3, 6.26);  
    cout << "Int: " << myNum1 << "\n";  
    cout << "Double: " << myNum2;  
    return 0;  
}
```

Java لغة

```
static int plusMethod(int x, int y) {  
    return x + y;  
}  
  
static double plusMethod(double x, double y) {  
    return x + y;  
}  
  
public static void main(String[] args) {  
    int myNum1 = plusMethod(8, 5);  
    double myNum2 = plusMethod(4.3, 6.26);  
    System.out.println("int: " + myNum1);  
    System.out.println("double: " + myNum2);  
}
```

- الاستدعاء الذاتي (Recursive function) : في البرمجة، "الاستدعاء الذاتي" يُعرف أيضًا بـ "التعريف الذاتي" أو "الاستدعاء الذاتي التكراري" (Recursion). وهو مفهوم يشير إلى قدرة دالة (Function) أو عملية معينة على استدعاء نفسها أثناء التنفيذ.

لغة C++

```
int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}

int main() {
    int result = sum(10);
    cout << result;
    return 0;
}
```

لغة Java

```
public class Main {
    public static void main(String[] args) {
        int result = sum(5, 10);
        System.out.println(result);
    }

    public static int sum(int start, int end) {
        if (end > start) {
            return end + sum(start, end - 1);
        } else {
            return end;
        }
    }
}
```

توضيح : عند القيام باستدعاء دالة بطريقة الاستدعاء الذاتي في البرنامج المكتوب بلغة C++ فإن نتيجة الاستدعاء عند المعطى (k) تبقى معلقة لحين إيجاد نتيجة الإستدعاء عند المعطى (k-1) وهكذا حتى يتم ايجاد نتيجة الاستدعاء الأخير . ويتم تعويض النتائج رجوعاً للوصول إلى ايجاد نتيجة الاستدعاء الأول ولذلك سيعطي أيضاً الإستدعاء الذاتي التكراري .

وهذا توضيح كيفية إستدعاء المثالين السابقين في ذاكرة الحاسوب :

$10 + \text{sum}(9)$

$10 + (9 + \text{sum}(8))$

$10 + (9 + (8 + \text{sum}(7)))$

...

$10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + \text{sum}(0)$

$10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0$

```

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
print("5! =", factorial(5)) # الناتج : 120

return 0;
}

```

تمرين : اكتب دالة التكرار الذاتي المناسبة في الكود التالي وذلك لإيجاد مجموع الأرقام بين 5 و 10 .

```

public class Main {
    public static void main(String[] args) {
        int result = sum(5, 10);
        System.out.println(result);
    }
    public static int sum(int start, int end) {
        if (end > start) {
            return
        } else {
            return end;
        }
    }
}

```

