**ملخص لسلسلة CS50**

علوم الحاسوب: هي عمليات لحل المشكلات على النحو التالي:

المدخلات 🡸 الحاسب 🡸 المخرجات

(المشكلات) (آلة المعالجة) (الحل)

**كيف نتواصل مع الحاسب؟**

الحاسب لا يفهم سوى الأصفار والآحاد (0,1) (وتدعى نظام العد الثنائي **Binary System**)، فهو لا يملك عقلا، وإنما يملك **الترانزستر** الذي يعطي إما صفر (أي أنه لا يوصل كهرباء)، أو واحد (أي أنه يوصل كهرباء)، وهذه الوحدة تسمى **بيت (Bit)،** وهنالك وحدة **البايت (Byte)** التي تساوي (8) بيت، وبهذه الطريقة يمكننا إعطاء أوامر للحاسب عن طريق إخباره بأن الرسالة المعينة تعني شيء معين، وهكذا مثلت الأحرف كما في الشكل:

في نظام العد الثنائي في الـ **Byte** كل خانة لها قيمة مختلفة عن الأخرى، مثل نظام العد العشري **(Decimal System)** الذي يحتوي على خانات، فنظام العد العشري سمي بهذا الاسم لأنه لا يحوي سوى على عشر أشكال للأرقام وهم (9,8,7,6,5,4,3,2,1,0)، أما الأعداد الأكبر من ذلك فهي مكونة من الأعداد العشرة السابقة، لهذا تم إنشاء فكرة الخانات والتي تمثل بالشكل التالي:

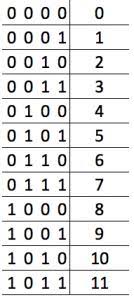
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **اسم الخانة** | **الآحاد** | **العشرات** | **المئات** | **الألوف** |
| **الأساس الأس** | **100** | **101** | **102** | **103** |
| **قيمة المبزلة** | **1** | **10** | **100** | **1000** |

من الشكل السابق نجد أن الخانة الأولى تضرب بعشرة قوة صفر والتي تساوي واحد، أما في الخانة الثانية فتضرب بعشرة قوة واحد والتي تساوي عشرة، أما في الخانة الثالثة فتضرب بعشرة قوة اثنان والتي تساوي مئة، وهلم جر.

وفي النظام الثنائي بنفس الطريقة، ولكن يضرب بدل العشرة باثنين، كما في الشكل:

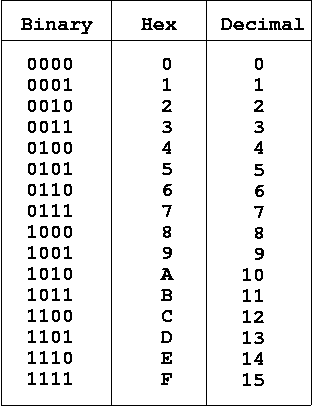
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **الأساس الأس** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** |
| **قيمة المنزلة** | **1** | **2** | **4** | **8** | **16** | **32** | **64** | **128** |

أمثلة على ذلك:



**نظام العد السادس عشر الـ (Hexadecimal):**

لقد تعرفنا على نظامي العد الثنائي والعشري، والآن حان الوقت لنتعرف على نظام العد السادس عشر الـ (**Hexadecimal**)، ولكن لماذا نحن نحتاجه مع أن لدينا نظامين أخرين للعد؟ هذا بالطبع لأننا نريد أن نسهل على أنفسنا، ونحن البشر نفضل النظام العشري على النظام الثنائي؛ لأننا نكتب أعداد كبيرة بعدد خانات أقل، بواسطة النظام العشري، لهذا فكرنا في أن نطور نظام عد آخر، يمكننا من كتابة أرقام أكبر بواسطة خانات أقل، لهذا قررنا أن نبتكر نظام العد السادس عشر، وهو مكون من ست عشر رمزا هم: (**F**,**E**,**D**,**C**,**B**,**A**,**9**,**8**,**7**,**6**,**5**,**4**,**3**,**2**,**1**,**0**)، والشكل التالي يبين قيم كل رمز بالنسبة للنظامين الثنائي والعشري:

****

والشكل التالي يبين كل منزلة بالقيمة التي تضرب بها لتعطي قيم الرموز في كل خانة:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **الأساس الأس** | **160** | **161** | **162** | **163** |
| **قيمة المنزلة** | **1** | **16** | **256** | **4096** |

وما يلي أمثلة على ذلك:

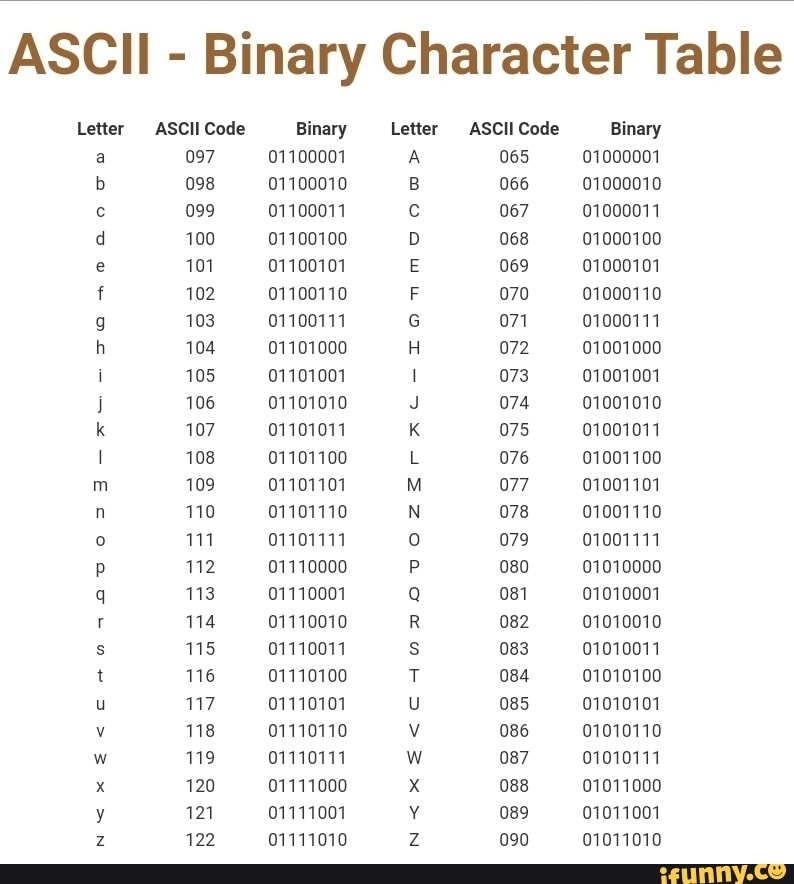
|  |  |
| --- | --- |
| **العدد بنظام العد السادس عشر** | **العدد الذي يساويه بنظام العد العشري** |
| **1** | **1** |
| **10** | **16** |
| **100** | **256** |
| **F5** | **230** |
| **AC** | **172** |

**استخدامات نظام العد السادس عشر:**

1. الذاكرة (**Memory**): الذاكرة مكونة من **Bytes**، ولكل **Bytes** في الذاكرة له عنوان، وهذه العناوين مكتوبة بنظام العد السادس عشر، ولكن الأرقام الأولى في نظام العد السادس عشر، هي موجودة أيضا في نظام العد العشري، لذلك لتفريق بينهما تم وضع (**0x**) قبل أعداد نظام العد السادس عشر، فأصبحت بهذا الشكل: (…**0x4**,**0x3**,**0x2**,**0x1**,**0x0**).
2. الألوان (**Color**): يتم تمثيل الألوان كما سوف نذكر بالتفصيل لاحقا بنظام العد الثنائي بهذه الطريقة (**11111111**)، بالطبع هذا الأمر للألوان الرئيسية فقط، وسنذكر هذا لاحقا أيضا، وتمثل هذه القيمة في نظام العد العشري بالقيمة (**255**)، وهذه هي أكبر قيمة للألوان، ولكن يتم تمثيل هذه القيمة بثلاث خانات، ولكن بنظام العد السادس عشر يتم تمثيل هذه القيمة بخانتين فقط، وهما (**FF**)، وهذا أفضل بالطبع.

**كيف يفهم الحاسب اللغات؟**

اجتمع العلماء على إنشاء نظام يحول أي رمز أو حرف بالغة الإنجليزية إلى وحدات وأصفار، والذي يسمى بنظام الأسكي كود **(ASCII CODE)**، تجد في الجدول التالي الأحرف فقط:



ولكن مجموع الثمانية **Byte** هو (255)، وهذا الرقم غير كافي لتمثيل جميع لغات العالم، لهذا اضطررنا إلى إنشاء نظام جديد سمية أنكي كود **(Unicode)**، والذي يعطينا بدل الـ (8) **Bit** يعطينا (16) و (32) **Bit**.

**كيف يفهم الحاسب الألوان؟**

تم اعتماد نظام الـ **(RGB)**، والذي يتكون من الألوان الثلاثة الأساسية (الأحمر، والأخضر، والازرق)، والذي يعطينا ألوان أخرى بواسطة وضع درجات مختلفة من الألوان، وتم تمثيل درجات هذه الألوان بالـ **Bytes**، فالألوان الأساسية مكونة من واحد **Bit**، واللون الناتج من خلط الألوان الثلاثة الأساسية مكون من ثلاثة **Bytes**، وكل **Byte** قيمته العظمى (255)، وأقل قيمة (0)، وتم تمثيل درجات الألوان الأساسية بقيم الـ **Bytes**.

**كيف يفهم الحاسوب الصور؟**

كل صورة مكونة من بكسل **(Pixel)،** والبيكسل مكون من لون، واللون هو **(RGB)،** والـ **(RGB)** هو **Byte**:

صورة 🡸 بيكسل 🡸 لون 🡸 بايت.

**كيف يفهم الحاسب الفيديوهات؟**

الفيديوهات هي عبارة عن مجموعة من الصور.

**كيف يفهم الحاسب الأصوات؟**

الأصوات هي عبارة عن نغمات، ويمكن تمثيل النغمات بقيمة الـ **Byte**.

**الملخص من (كيف يفهم الحاسب)؟**

نحاول جعل كل شيء بالأصفار والآحاد.

**الخوارزميات:**

هي مجموعة من الخطوات تهدف إلى الوصول إلى أفضل، وأقصر، وأسرع، حل ممكن للمشكلة.

**علاقة الخوارزميات بالبرمجة:**

عند إنشاء مشروع معين أنت بحاجة لإعلام الآلة بكل ما يجب أن تفعله، فهي لا تستطيع استنتاج أي أمر من ذاتها، لهذا وعند البدء بإنشاء أي مشروع يجب عليك كتابة الخطوات (وهذه ما يسمى بالسودو كود **Pseudocode**)، وهذا يساعد في حل المشاكل.

**أنواع خوارزميات البحث:**

سنأخذ مثالا، وهو أن لدينا مصفوفة تحتوي على ألف عنصر مختلف، ونريد البحث عن عنصر واحد ضمن هذه

العناصر، هنا نحن لدينا طرق أو خوارزميات عدة يمكننا استخدامها، والآن سوف ندرس احتمالات كل خوارزمية بخوارزميتها:

1. (**Linear Search**): وتعني البحث الخطي، وسميت كذلك لأنها تبحث عنصرا عنصرا بشكل خطي من أول المصفوفة وحتى نهايتها، ويكون أسوء احتمال أو ما يسمى بالـ (السيناريو) لهذه الخوارزمية والذي يسمى بالـ (بيك أوو) (**Big O**) هو عدد عناصر هذه المصفوفة، والذي هو ألف في مثالنا، وأفضل سيناريو للخوارزمية والذي يسمى بالـ (بيك أوميجا) (**Big Ω**) هو واحد، أي أن نجد العنصر من المرة الأولى، ومقياس السرعة لأي خوارزمية هو أسوء سيناريو، ونرمز لعدد عناصر المصفوفة بـ (**n**)، ونرمز لسرعة هذه الخوارزمية بـ **O**(**n**).

**Pseudocode:**

**For I from 0 to n-1**

**If number behind** (**name of array**)[**i**]

**Return true**

**Return false**

1. (**Binary Search**): وتعني البحث الثنائي، ويتم استخدام هذه الخوارزمية إذا علمت أن المصفوفة مرتبة تصاعديا أو تنازليا، ولكل من هذه الحالتين طريقة مختلفة قليلا عن الأخرى، سنفترض أن لدينا مصفوفة مرتبة تصاعديا، سنقوم ببدء البحث من المنتصف ونرى إذا كان العدد الذي نبحث عنه أصغر من العدد الذي ظهر؛ ننتقل إلى منتصف الأعداد التي على يسار العدد (لأن العد يبدأ من اليسار) الذي ظهر، ونعيد الأمر، فإذا كان العدد الذي نبحث عنه أكبر من العدد الذي ظهر ننتقل إلى منتصف الأعداد التي على يمين العدد الذي ظهر، ونكرر ذلك حتى نصل إلى الحل، وهكذا يكون أسوء سيناريو هو **O**(**log n**)، وهو عشرة في مثالنا، وهي تعني أن هذه الخوارزمية يتم انقاص نصف المشكلة كل خطوة.

**Pseudocode:**

**If no** (**values**)

**Return false**

**If number behind** (**name of array**)[**middle**]

**Return true**

**Else if number <** (**name of array**)[**middle**]

**Search** (**name of array**)[**0**] **through** (**name of array**)[**middle-1**]

**Else if number >** (**name of array**)[**middle**]

**Search** (**name of array**)[**middle+1**] **through** (**name of array**)[**n-1**]

**أنواع خوارزميات الترتيب:**

1. (**Selection Sort**): وتعني الترتيب الاختياري، تقوم هذه الخوارزمية بأخذ أول عنصر من المصفوفة، ومقارنته مع كل عناصر المصفوفة، وتبدل بينه وبين أصغر عنصر في المصفوفة، ثم تأخذ العنصر الثاني، وتقوم بمقارنته مع عناصر المصفوفة التي بعده... وهكذا حتى آخر عنصر من المصفوفة، عدد الخطوات لهذه الخوارزمية يكون على الشكل التالي، بفرض أن عدد عناصر المصفوفة هو (**n**): **n + (n-1) + (n-2) +** (**n-3**)… **+** (**n-n)** ، أسوء سيناريو هو **O**(**n2**)، ولكن أفضل سيناريو هو أيضا (**n2**)**Ω**.

**Pseudocode:**

**For I from 0 to n-1**

**Find smallest number between** (**name of array**)[**i**] **and** (**name of array**)[**n-1**]

**Swap smallest number with** (**name of array**)[**i**]

1. (**Bubble Sort**): وتعني الترتيب الفقاعي، وهي تقوم بالمقارنة بين أول عنصرين، فإذا كان العنصر الأول أكبر من العنصر الثاني بدلت بين مكانيهما، وإلا أبقتهما على حالهما، وفي كلا الحالتين تذهب بعدها وتقارن بين العنصر الثاني، والعنصر الثالث،وبنفس الطريقة حتى تصل إلى آخر عنصر، وبهذا يكون العنصر الأكبر أخذ مكانه في آخر المصفوفة، ثم تعود من جديد وتبدأ بالمقارنة، ولكن لا تمر على آخر عنصر تم ترتيبه وهكذا حتى تنتهي، وأسوء سيناريو هو**O**(**n2**) أيضا، ولكن أفضل سيناريو هو (**n**)**Ω**.

**Pseudocode:**

**Repeat n-1 times**

**For I form 0 to n-2**

**If** (**name of array**)[**i**] **and** (**name of array**)[**i**] **out of order**

**Swap them**

**If no swaps**

**Quit**

1. (**Merge Sort**): وتعني الترتيب الدمجي، وهي قائمة على مبدأ التقسيم والتجميع، فأولا هي تقوم بتقسيم المصفوفة، وهي تقسمها من النصف، ثم تقسم المصفوفات التي نتجت من التقسيم الأول من المنتصف أيضا وهكذا، حتى لا تستطيع التقسيم، ثم تقوم بدمج العنصرين الأوليين بشكل مرتب مع بعضهما، والعنصرين اللذان هما بعدهما بنفس الطريقة، حتى نهاية العناصر، فتتكون مصفوفات مكونة من عنصرين، ولكنها مرتبة، ثم تدمج المصفوفتين الأوليتين بشكل مرتب، فاللتان بعدهما، فاللتان بعدهما وهكذا، حتى يتم إعادة تكوين المصفوفة الرئيسية، ولكن بشكل مرتب.

**الأمور المشتركة بين أغلب لغات البرمجة:**

**أولاً** أنواع البيانات أو الـ (**Data Types**)، وهي متشابهة في أغلب اللغات وهي كالآتي:

وهذا النوع هو للأعداد الصحيحة (**Integer**)

وهذا النوع للأحرف والرموز (**Character**)

وهذا النوع للأعداد التي فيها فاصلة (**Float**)

وهذا النوع للأعداد التي فيها فاصلة وهو للأعداد الكبيرة وهو أعم من الـ (**Float**) (**Double**)

وهذا النوع يرجع قيمتين إما (صح (**True**)) أو (خطأ (**False**)) (**Boolean**)

**ملاحظة:**

* لتخزين البيانات ضمن هذه الأنواع، يجب إنشاء كائن يحمل صفة إحدى الأنواع، وهذا ما يسمى بالـمتغيرات أو الـ (**Variable**).
* هذه هي أنواع البيانات المتشابهة في أغلب اللغات، ولكن هنالك بعض اللغات تحتوي على عدد أكبر من أنواع البيانات هذه.

**ثانياً** التعليقات أو ما يسمى بالـ (**Comminate**) وطريقة استخدامها تختلف من لغة إلى أخرى مثال على ذلك:

في لغة (**c++**) ولغة (**c**) ولغة (**css**) وبعض اللغات الأخرى: (**//**) للسطر الواحد (**/\*\*/**) للأسطر المتعددة.

أما في لغة (**HTML**): (**<--!>**)...

**ثالثاً** كل أمر تأمر به الحاسب، وتريد أن ينفذه لك يسمى (**Function**)، وهذا المصطلح موجود في أغلب لغات

البرمجة، كل لغات البرمجة تقدم لك الدوال الجاهزة عبر استدعاء المكتبات، وأيضا في أغلب لغات البرمجة يمكننا أن ننشئ الدوال الخاصة بنا لاستعمالها متى شئنا، والآن لنتعلم كيف ننشئ بعض الدوال الخاصة بنا:

ملاحظة:

* الدوال تقسم إلى قسمين:

1. القسم الأول يسمى بالدوال الغير مرجعة؛ لأنها لا ترجع أي قيم.
2. القسم الثاني يسمى بالدوال المرجعة، وهي على عدة أنواع، وهي ترجع قيم مختلفة بحسب نوع الدالة.
3. **الدوال الغير مرجعة:**

عند إنشائها دائما ما يستخدم معها الكلمة الأساسية (**Void**)، أي أنها عاطلة لا ترجع أي قيمة.

**Void**(**name of function**)**()**{**body of function**}

1. **الدوال المرجعة:**

لها أنواع مختلفة، ويتم إنشاؤها باستخدام **أنواع البيانات**، وهي دائما ما ترجع قيمة.

**Data type**(**name of function**)**()**{**body of function**}

ملاحظات:

* إذا أردنا استدعاء دالتنا علينا كتابة اسم الدالة، وبعدها الأقواس المعرِفة (أي الأقواس التي تعرِّف المترجم أنها دالة)، داخل الدالة الرئيسية.
* الأقواس البنفسجية التي بعد اسم الدالة مباشرة؛ هي أقواس يتم تمرير القيم من خلالها عبر ما يسمى بالمعاملات أو بالمشغلات (**Operators, Arguments, Parameter**)، وهذه المعاملات هي متغيرات يتم تعريفها داخل الأقواس الرئيسية للدالة، وعند استدعاء هذه الدالة يجب تمرير قيمة داخل أقواس الدالة المعرفة، وهنالك طريقتين لتمرير القيم، سيتم شرحهم كما يلي:

1. طريقة النسخ (**Pass By Value**)**،** وهي تعني التمرير بالقيمة، وهنا نحن نأخذ قيمة المتغير الذي تم تمريره، وعند التعديل عليها، فنحن نعدل على قيمة المتغير الذي تم إنشاؤه داخل الدالة، ولا يتم التعديل على قيمة المتغير الذي تم تمريره، وهنا يتم تعريف المتغير بشكل عادي دون إضافات.
2. طريقة التأشير (**Pass By Reference**)**،** وهي تعني التمرير بالإشارة، وهنا نحن لا نأخذ قيمة المتغير الذي تم تمريره، وإنما نؤشر على مكان المتغير الذي تم تمريره عبر مؤشر (**Pointer**)**،** يتم تعريفه عن طريق وضع نجمة (**\***) بعد نوع البيانات مباشرة، أو قبل اسم المتغير مباشرة، وبهذه الطريقة يمكننا التعديل على قيمة المتغير الذي تم تمريره، وليس نسخ قيمته فقط.

**رابعاً** الجمل الشرطية أو التي تسمى بـ (**Conditional Sentences**)، وهيكل الجملة الشرطية يكون كالآتي:

**if(condition){functions}**

**else if(condition){functions}**

**else{functions}**

الجملة الأولى، وهي للشرط الأول، بالطبع الشرط يسبق الأوامر، ويتم تنفيذ الأوامر إذا تحقق الشرط فقط، وإلا

فلا ينفذ شيء من الأوامر، ويتم تجاهلها، والجملة الثانية، هي جملة شرط أخرى، ينطبق عليها ما ينطبق على

الجملة الأولى، أما الجملة الثالثة، فهي الجملة التي تتحقق إذا لم تتحقق أي جملة شرط.

ملاحظات:

* يمكن بدلا من كتابة (**else if**) في كل مرة؛ الاكتفاء بكتابة (**if**) فقط.
* يمكن كتابة جمل شرطية بأعداد أكبر مما كتبناه، وهذا المثال للتوضيح فقط.

**خامساً** الجمل التكرار (الدورانية) أو ما يسمى بالـ (**Loop Sentences**)، ولها هيكليات عدة منها:

**while(condition){functions}**

**do{functions} while(condition)**

**for(many things[condition]){functions}**

كل الجمل التكرارية لها شرط، وإلا لكانت تكررت دون توقف.

الجملة الأولى، تقوم بفحص الشرط أولا، ثم إذا كان صحيحا نفذت الأوامر، وإلا تم تجاهلها، الجملة الثانية، تقوم بالعمل لمرة واحدة أولا، ثم تفحص الشرط، فإذا كان صحيحا أكملت العمل، وإلا توقفت، الجملة الشرطية الثالثة، هي جملة متطورة قليلا، وهي أكثر جملة يتم استخدامها، وهي أيضا تقوم بفحص الشرط أولا على غرار الجملة الأولى، فإذا كان صحيحا عملت وإلا توقفت.

**ملاحظة:**

* حاول تغير الشرط بعد كل دورة تقوم بها الجملة، لكيلا تبقى الجملة في مرحلة العمل للأبد.

**سادساً** المصفوفات أو الـ (**Arrays**)، والمصفوفات تعتبر متغير قابل لتخزين عدة قيم، يتم تعريف المصفوفة كما يتم تعريف متغير عادي؛ زائد إضافة واحدة، وهو كما يلي: (نوع البيانات + اسم المصفوفة + [حجم المصفوفة])، حجم المصفوفة هو مقدار القيم التي يمكن للمصفوفة تحمله، ويتم تعين هذا الحجم عن طريق المبرمج أو المستخدم، والقيمة الأولى في المصفوفة تساوي صفر، أي أن العدد (الذي هو حجم المصفوفة) يعبر عن عدد القيم التي يمكن تخزينها، ولكن لاستدعائها القيمة الأولى نكتب الرقم صفر، كالآتي: (اسم المصفوفة + [0]).

**سابعاً** الفئات أو الهياكل أو ما يسمى بالـ (**struct**, **class**)، ولكن لكل لغة برمجة طريقتها الخاصة في تفصيل هذا الأمر، ولا يتسع المقام لشرح كل الطرق؛ لهذا لن يتم شرح أي طريقة عل حساب الأخرى.

**كيف يفهم الحاسب لغات البرمجة:**

أغلب لغات البرمجة تكتب باللغة الإنجليزية، فهل يفهم الحاسب اللغة الإنجليزية؟

لا؛ ولكنه يحتاج لمترجم أو ما يسمى بالـ (**Compiler**)، ليترجم له.

**ملاحظة:**

* لغات البرمجة تقسم إلى مستويات عدة، أولى اللغات التي تم اختراعها لإعطاء الأوامر للحاسب، تدعى باللغات المتدنية المستوى (أي قريبة من لغة الحاسب)، ولكن هذه اللغات صعبة التعلم، والفهم، لهذا لم يعد أحد يستخدمها، وهذه اللغة لا تحتاج لمترجم ليفهمها الحاسب، ولكن اللغات التي يتم استخدامها الآن، تدعى باللغات عالية المستوى (أي بعيدة عن لغة الحاسب)، ولهذا تحتاج إلى مترجم لكي يفهمها الحاسب.

للمزيد من المعلومات: <https://ar.wikipedia.org/wiki/%D9%85%D8%AD%D9%88%D9%84_%D8%A8%D8%B1%D9%85%D8%AC%D9%8A>

**الـ (Recursion):**

هذا المصطلح؛ يعني تكرار الدالة باستدعائها داخل نفسها، ولكن هنالك عيوب لهذا الأمر، وهو أن الدالة تتكرر دون توقف، ولهذا علينا تحديد مرات التكرار عن طريق إعطاء الدالة معاملا، وعند استدعاء الدالة داخل نفسها ننقص من قيمة المعامل، ونضع شرطا يوقف الدالة عن العمل؛ عند وصول قيمة المعامل إلى قيمة معينة.

**الذاكرة رام (ذاكرة الوصول العشوائية) (RAM(Random Access Memory):**

الذاكرة تخزن البيانات بطريقة منظمة على النحو التالي:

|  |
| --- |
| **Machine Code** |
| **Global** |
| **Heap**  **Stack** |

**أولاً** الـ (**Machine Code**): وهي لغة الآلة؛ أي الأصفار، والآحاد، وهي أول شيء يعمل في أي برنامج.

**ثانياً** الـ (**Global**): وهي المتغيرات العامة التي يتم إنشاؤها في البرنامج، هنالك طريقتان لإنشاء المتغيرات وهما:

1. الطريقة العادية: وهي الطريقة التي يتم حجز مكان في الذاكرة للمتغير تلقائيا، وهي على نوعين أيضا:
2. المتغيرات المحلية (**Local**)، وهي المتغيرات التي يتم إنشاؤها داخل الدوال، والتي لا يمكن استخدامها إلا من داخل الدوال نفسها.
3. المتغيرات العامة (**Global**)، وهي المتغيرات التي يتم إنشاؤها خارج كل الدوال، والتي يمكن استخدامها من أي مكان من البرنامج، وهذه المتغيرات تخزن ثاني شيء في الذاكرة.
4. الطريقة اليدوية: وهي الطريقة التي يتم حجز مكان في الذاكرة يدويا عن طريق المطور.

**ملاحظة:**

* يجب حذف المتغيرات بعد إنشائها بطرقة يدوية؛ لكيلا تبقى مخزنة في الذاكرة العشوائية للأبد، وهذا الأمر يحدث يدويا عند إنشاء متغير بالطريقة العادية.

**ثالثاً** الـ (**Heap**): وهنا كل ما ينشأ ويُزال بشكل يدوي يخزن فيها، مثل المتغيرات التي يتم إنشائها بالطريقة اليدوية، وهي تمتلئ من الأعلى إلى الأسفل، وعندما تمتلئ يحدث مشكلة تسمى هذه المشكلة بـ (**Heap Over Flow**).

**رابعاً** الـ (**Stack**): وهنا كل ما ينشأ ويُزال بشكل تلقائي يخزن فيها، مثل المتغيرات المحلية التي يتم إنشاؤها بالطريقة العادية، والدوال أيضا، وهي تمتلئ من الأسفل إلى الأعلى، وعندما تمتلئ يحدث مشكلة تسمى هذه المشكلة بـ (**Stack Over Flow**).

**هياكل البيانات (Data Structure):**

وهي طريقة التعامل مع البيانات من حفظها وترتيبها وطريقة الوصول إليها والبحث عنها وحذفها والهدف منها هو تطبيق ما ذكر بأفضل طريقة لكي يكون البرنامج ذو أداء عالي.

**أنواع هياكل البيانات:**

**Primitive Non-Primitive**

**Int Array**

**Float Linked-List**

**Double Tree**

**Char Hash-Table**

**Bool Tries**