CS50: Introduction to Computer Science

ملخص المحاضرة الأولى CS50

◈عنوان المحاضرة Introduction to Computer Science :
في هذه المحاضرة، يتم تقديم المفاهيم الأساسية لعلوم الحاسب، مع التركيز على **التفكير الخوارزمي** وفهم كيفية عمل أجهزة الكمبيوتر والبرمجة.

1 ◈ما هو علوم الحاسب؟

- علوم الحاسب ليست فقط **البرمجة**، بل تشمل:
  - ✅ **الخوارزميات** (Algorithms)
  - ✅ **هياكل البيانات** (Data Structures)
  - ✅ **الذكاء الاصطناعي** (AI)
  - ✅ **الأمن السيبراني** (Cybersecurity)
  - ✅ **قواعد البيانات** (Databases)

2 ◈ما هو الكمبيوتر؟ وكيف يعمل؟

- الكمبيوتر هو **آلة تأخذ مدخلات(Input) ، تعالجها(Processing) ، ثم تعطي مخرجات.(Output)**
- يستخدم الكمبيوتر **النظام الثنائي(Binary System)** ، حيث يعتمد على القيم **0و 1** فقط.
- أي بيانات في الكمبيوتر (نصوص، صور، فيديوهات) يتم تمثيلها بسلسلة من **البتات**.**(Bits)**

3 ◈مفهوم الخوارزميات(Algorithms)

◈تعريف:

الخوارزمية هي **مجموعة من الخطوات المنطقية** لحل مشكلة معينة.

◈مثال على خوارزمية بسيطة: البحث في دفتر الهاتف📖

1. افتح الدفتر من المنتصف.📖
2. إذا كان الاسم قبل الصفحة الحالية → اذهب إلى النصف الأول.
3. إذا كان الاسم بعد الصفحة الحالية → اذهب إلى النصف الثاني.
4. كرر حتى تجد الاسم المطلوب.

هذا يشبه خوارزمية "البحث الثنائي (Binary Search)

4. ◈مقدمة في البرمجة

💡**لغات البرمجة** هي الوسيلة التي نستخدمها للتواصل مع الكمبيوتر.
🛠️أمثلة على لغات البرمجة:

- **C (** اللغة الأساسية المستخدمة فيCS50)
- **Python (**لغة سهلة وقوية)
- **JavaScript (**لغة تطوير الويب)

👨‍💻 أول كود نكتبه عادة هو "Hello, World!" في لغة C:

```c
#include <stdio.h>

int main(void) {
    printf("Hello, World!\n");
}
```

⚡ **تحليل الكود:**

- `#include <stdio.h>` → استدعاء مكتبة الإدخال والإخراج.
- `int main(void)` → الدالة الرئيسية للبرنامج.
- `printf("Hello, World!\n");` → طباعة النص على الشاشة.

5. 🔷أهمية التفكير الحسابي(Computational Thinking)

- القدرة على **تحليل المشاكل** وتقسيمها إلى أجزاء صغيرة لحلها بكفاءة.
- استخدام **الخوارزميات** لتنفيذ العمليات الحسابية أو أي عمليات أخرى.

6. 🔷نظرة على Scratch

- **Scratch**هو برنامج تعليمي يعتمد على **البرمجة البصرية** .(Drag & Drop)
- يساعد في فهم أساسيات البرمجة **بدون الحاجة لكتابة أكواد معقدة**.
- يتم تنفيذ البرامج باستخدام **كائنات (Objects) وأوامر برمجية** متصلة ببعضها مثل الـ .**Puzzle**

🔹 7. الفرق بين لغات البرمجة

اللغة	الاستخدام
C	برمجة منخفضة المستوى، أداء عالي
Python	لغة سهلة، مناسبة للذكاء الاصطناعي والتطوير السريع
JavaScript	تطوير الويب والتطبيقات التفاعلية
SQL	إدارة قواعد البيانات

8. 🔷 مفاهيم أساسية من المحاضرة

- **المتغيرات** :**(Variables)** تخزن البيانات.
- **الحلقات** :**(Loops)** تكرار العمليات البرمجية.
- **الشروط** :**(Conditions)** اتخاذ قرارات بناءً على القيم.

💡خلاصة المحاضرة:

✅ علوم الحاسب ليست مجرد برمجة، بل تتعلق بحل المشكلات.
✅ الكمبيوتر يعمل بالنظام الثنائي (0 و 1).
✅ الخوارزميات هي أساس أي برنامج أو نظام حاسوبي.
✅ تعلمنا أول برنامج في C باستخدام .printf()
✅ البرمجة تبدأ بمفاهيم مثل المتغيرات، الحلقات، والشروط.

ملخص المحاضرة الثانية - "Arrays, Loops, and Conditions"

🔷 في المحاضرة الثانية، تم التركيز على **الهياكل الأساسية في البرمجة مثل المصفوفات(Arrays) ، الحلقات(Loops) ، والجمل الشرطية.(Conditions)**

1. 🔷 مراجعة على C وذكر أهمية الدوال(Functions)

💡**في لغةC ، أي كود يتم تنفيذه داخل دالة.(Function)**
✅ رأينا دالة ,()main وهي نقطة البداية لأي برنامج بلغة.C
✅ تعلمنا كيف نستخدم **مكتبة stdio.h** لطباعة القيم أو أخذ إدخال من المستخدم.

🔹 مثال على إدخال وإخراج بيانات:

```c
#include <stdio.h>

int main(void) {
    int x;
    printf("Enter a number: ");
    scanf("%d", &x);
    printf("You entered: %d\n", x);
}
```

🔹 `scanf("%d", &x);` ← يأخذ عدد صحيح من المستخدم ويخزنه في `x`.

🔹 `printf("%d", x);` ← يطبع قيمة `x` على الشاشة.

2. 🔷 الجمل الشرطية (Conditions)

🔷 مفهوم الجمل الشرطية:

- الجمل الشرطية تساعدنا في **اتخاذ قرارات** بناءً على القيم المدخلة.
- يتم استخدامها بكلمة `if` وأحيانًا مع `else` أو `else if`.

🔹 مثال على `if-else`:

```c
#include <stdio.h>

int main(void) {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);

    if (age >= 18) {
        printf("You are an adult.\n");
    }
    else {
        printf("You are a minor.\n");
    }
}
```

✅ إذا كان العمر **18 أو أكثر** → يطبع "You are an adult."

✅ إذا كان العمر **أقل من 18** → يطبع "You are a minor."

3. ◈ الحلقات التكرارية(Loops)

تستخدم الحلقات لتكرار مجموعة من الأوامر عدة مرات بدلًا من كتابتها يدويًا.

◈ أنواع الحلقات:

1. **for loop** (تكرار محدد)
2. **while loop**(تكرار بناءً على شرط)
3. **do-while loop**(تنفذ الكود مرة على الأقل)

◈ أ. حلقة for

يتم استخدامها عند معرفة عدد التكرارات مسبقًا .

```
Edit Copy

for (initialization; condition; update) {
    // code to be executed
}
```

- مثال عملي:

```
Edit Copy

#include <stdio.h>

int main(void) {
    for (int i = 0; i < 5; i++) {
        printf("Hello, CS50!\n");
    }
}
```

✅ يتم طباعة `Hello, CS50!"` `5"` مرات لأن `i` يبدأ من 0 حتى 4.

◈ ب. حلقة while

📌 يتم استخدامها عندما لا نعرف عدد التكرارات مسبقًا ولكن لدينا شرط معين.

🔹 الصيغة العامة:

```
while (condition) {
    // code to execute
}
```

🔹 مثال عملي:

```
#include <stdio.h>

int main(void) {
    int i = 0;
    while (i < 5) {
        printf("Iteration %d\n", i);
        i++;
    }
}
```

✅ يتم طباعة `"Iteration"` والقيمة الحالية لـ `i` حتى تصل إلى 5.

🔷 ج. حلقة do-while

📌 تنفذ الكود مرة على الأقل، حتى لو كان الشرط غير صحيح من البداية.

🔹 الصيغة العامة:

```
do {
    // code to execute
} while (condition);
```

🔹 مثال عملي:

```
#include <stdio.h>

int main(void) {
    int x;
    do {
        printf("Enter a positive number: ");
        scanf("%d", &x);
    } while (x <= 0);

    printf("You entered: %d\n", x);
}
```

✅ يستمر في طلب إدخال عدد موجب حتى يدخله المستخدم.

4. 🔷 المصفوفات(Arrays)

💡**المصفوفة هي مجموعة من القيم المخزنة تحت اسم واحد في الذاكرة.**
🔷كل عنصر داخل المصفوفة يمكن الوصول إليه باستخدام **index** يبدأ من 0.

- **تعريف مصفوفة في C:**

```
int numbers[5] = {10, 20, 30, 40, 50};
```

✅ المصفوفة `numbers` تحتوي على **5 عناصر** (10, 20, 30, 40, 50).
✅ للوصول إلى عنصر معين، نستخدم **index**:

```
printf("%d\n", numbers[2]); // ستطبع 30
```

- **استخدام المصفوفات مع الحلقات:**

```
#include <stdio.h>

int main(void) {
    int numbers[5] = {10, 20, 30, 40, 50};

    for (int i = 0; i < 5; i++) {
        printf("Element %d: %d\n", i, numbers[i]);
    }
}
```

💡خلاصة المحاضرة:

✅تعلمنا الجمل الشرطية if-else لاتخاذ القرارات.
✅فهمنا الحلقات التكرارية.(for, while, do-while)
✅تعرفنا على **المصفوفات** وكيفية استخدامها في البرامج.
✅استخدمنا الحلقات للوصول إلى عناصر المصفوفات بسهولة.

💡ملخص المحاضرة الثالثة– "Algorithms and Complexity"

في هذه المحاضرة، ركزنا على الخوارزميات (Algorithms) وتحليل التعقيد الزمني (Time Complexity) لفهم كيفية كتابة كود أكثر كفاءة .

1. ◈ما هي الخوارزميات؟

◈ الخوارزمية هي مجموعة من الخطوات المحددة لحل مشكلة معينة.
◈ يمكن تنفيذ الخوارزميات بعدة طرق، بعضها أسرع وأكثر كفاءة من غيرها.

📌مثال على خوارزمية بسيطة: البحث عن رقم داخل قائمة أرقام.
إذا كان لديك قائمة مثل:
[2, 4, 7, 10, 15]وتريد البحث عن 10، هناك أكثر من طريقة للقيام بذلك.

2. ◈البحث (Searching Algorithms)

📌توجد خوارزميات مختلفة للبحث، أشهرها:

أ. البحث الخطي(Linear Search)

◈ يبحث عن العنصر عن طريق فحص كل عنصر في القائمة واحدًا تلو الآخر.
◈ الوقت المستغرق في أسوأ الحالات هو .O(n)

🔹 كود البحث الخطي:

Edit Copy

```c
#include <stdio.h>

int linear_search(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // العنصر موجود، نعيد موقعه
        }
    }
    return -1; // العنصر غير موجود
}

int main(void) {
    int numbers[] = {2, 4, 7, 10, 15};
    int target = 10;
    int index = linear_search(numbers, 5, target);

    if (index != -1) {
        printf("Found at index: %d\n", index);
    } else {
        printf("Not found\n");
    }
}
```

✅ **عيب البحث الخطي:** بطيء جدًا إذا كانت القائمة كبيرة.

ب. البحث الثنائي(Binary Search)

📌 يعمل فقط على القوائم المرتبة!
🔹 يقسم القائمة إلى نصفين في كل خطوة، مما يجعله أسرع من البحث الخطي.
🔹 تعقيده الزمني في أسوأ الحالات هو .O(log n)

🔹 كود البحث الثنائي:

Edit Copy

```c
#include <stdio.h>

int binary_search(int arr[], int size, int target) {
    int left = 0, right = size - 1;

    while (left <= right) {
        int mid = (left + right) / 2;

        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

int main(void) {
    int numbers[] = {2, 4, 7, 10, 15}; // القائمة مرتبة
    int target = 10;
    int index = binary_search(numbers, 5, target);

    if (index != -1) {
        printf("Found at index: %d\n", index);
    } else {
        printf("Not found\n");
    }
}
```

✅ أسرع بكثير من البحث الخطي لأن عدد المقارنات أقل.
✅ **عيبه:** يجب أن تكون القائمة مرتبة مسبقًا.

3. 🔹 الترتيب(Sorting Algorithms)

📌 يستخدم الترتيب لجعل البحث أكثر كفاءة وتحسين أداء البرامج.

أ. خوارزمية الفقاعات(Bubble Sort)

◈تقوم بمقارنة كل عنصر مع العنصر التالي، وإذا كان أكبر، يتم التبديل بينهما.
◈تتكرر العملية حتى تصبح القائمة مرتبة.
◈تعقيدها الزمني .$O(n^2)$

♦ كود Bubble Sort:

Edit Copy

```c
#include <stdio.h>

void bubble_sort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main(void) {
    int numbers[] = {10, 2, 8, 6, 7};
    int size = 5;

    bubble_sort(numbers, size);

    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");
}
```

✅ **عيبه:** بطيء جدًا مع القوائم الكبيرة.

ب. خوارزمية الاختيار(Selection Sort)

- تبحث عن **أصغر عنصر** في القائمة وتضعه في مكانه الصحيح.
- تستمر العملية حتى تصبح القائمة مرتبة.
- تعقيدها $O(n^2)$.

🔹 كود Selection Sort:

Edit Copy

```c
#include <stdio.h>

void selection_sort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int min_index = i;

        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[min_index]) {
                min_index = j;
            }
        }

        int temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
    }
}

int main(void) {
    int numbers[] = {10, 2, 8, 6, 7};
    int size = 5;

    selection_sort(numbers, size);

    for (int i = 0; i < size; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");
}
```

✅ أسرع قليلًا من **Bubble Sort** لكنه ليس الأفضل.

4. 🔷 تحليل التعقيد الزمني(Big O Notation)

Big O 📌 يحدد سرعة الخوارزمية بالنسبة لحجم البيانات.(n)

✅ بعض التعقيدات الزمنية الشائعة:

- O(1): ثابت، أسرع خوارزمية.
- O(log n): مثل **البحث الثنائي**، سريع جدًا.
- O(n): مثل **البحث الخطي**، متوسط السرعة.

- $O(n^2)$: مثل **Bubble Sort** و**Selection Sort**، بطيء جدًا مع البيانات الكبيرة.

🔹 **مثال بياني للمقارنة بين التعقيدات المختلفة:**

التعقيد الزمني	الأداء
`O(1)`	ممتاز جدًا
`O(log n)`	ممتاز
`O(n)`	جيد
`O(n²)`	سيئ مع القوائم الكبيرة

💡**خلاصة المحاضرة:**

✅تعلمنا مفهوم **الخوارزميات** وأهميتها في حل المشكلات.
✅فهمنا **البحث الخطي** و**البحث الثنائي**، ومتى نستخدم كل منهما.
✅استعرضنا خوارزميات **الترتيب** مثل **Bubble Sort** و**Selection Sort**.
✅فهمنا **تحليل التعقيد الزمني (Big O Notation)** وتأثيره على أداء الكود.

💡ملخص المحاضرة الرابعة -"Memory and Pointers"

في هذه المحاضرة، تعلمنا كيف تتعامل الذاكرة (Memory) مع البيانات في لغة C، وكيفية استخدام المؤشرات (Pointers)للوصول إليها بكفاءة💻 .

1. 🔷 الذاكرة في الكمبيوتر(Memory in Computers)

📌**الكمبيوتر يستخدم الذاكرة العشوائية (RAM) لتخزين البيانات أثناء تشغيل البرامج.**
📌**كل قطعة من البيانات في الذاكرة لديها عنوان (Address) يمكن الوصول إليه.**
📌**البيانات في الذاكرة تكون على شكل مصفوفة من البايتات.(Bytes)**

2. 🔷 المؤشرات (Pointers) في لغةC

📌**المؤشر هو متغير يخزن عنوان (Address) متغير آخر في الذاكرة.**
📌**نستخدم *للإشارة إلى قيمة المتغير المخزن في هذا العنوان .**

🔹 مثال على المؤشرات:

```c
#include <stdio.h>

int main(void) {
    int x = 5;
    int *ptr = &x; // تخزين عنوان x في المؤشر ptr

    printf("قيمة x: %d\n", x);
    printf("عنوان x في الذاكرة: %p\n", &x);
    printf("عنوان المخزن في ptr: %p\n", ptr);
    printf("قيمة x عبر المؤشر ptr: %d\n", *ptr); // dereferencing
}
```

📌 `&x` تُستخدم للحصول على عنوان المتغير `x`.
📌 `ptr` يحمل عنوان `x`، وعند استخدام `ptr*` نحصل على القيمة المخزنة في العنوان.

3. 🔷 المؤشرات والمصفوفات(Pointers and Arrays)

📌 **المصفوفات في C يتم التعامل معها كعناوين في الذاكرة!**
📌 عند تعريف مصفوفة، اسمها يمثل **عنوان أول عنصر فيها**

🔹 مثال:

```c
#include <stdio.h>

int main(void) {
    int numbers[] = {10, 20, 30};

    printf("عنوان أول عنصر: %p\n", numbers);
    printf("قيمة أول عنصر: %d\n", *numbers); // يساوي numbers[0]
}
```

📌 `numbers` هو عنوان أول عنصر في المصفوفة.
📌 `numbers*` يطبع أول عنصر، ويمكننا استخدام `(numbers + 1)*` للوصول إلى العنصر الثاني وهكذا.

4. 🔷 تخصيص الذاكرة ديناميكيًا (Dynamic Memory Allocation)

📌 أحيانًا نحتاج إلى تخصيص الذاكرة أثناء تشغيل البرنامج وليس عند كتابته.
📌 نستخدم **malloc** و **free** لإدارة الذاكرة يدويًا .

- **مثال على `malloc`:**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int *ptr = malloc(3 * sizeof(int)); // تخصيص ذاكرة لثلاثة أعداد صحيحة

    if (ptr == NULL) {
        printf("فشل تخصيص الذاكرة!\n");
        return 1;
    }

    ptr[0] = 10;
    ptr[1] = 20;
    ptr[2] = 30;

    printf("العناصر المخزنة: %d %d %d\n", ptr[0], ptr[1], ptr[2]);

    free(ptr); // تحرير الذاكرة
}
```

📌 `malloc` يحجز ذاكرة بحجم معين.
📌 `free` يجب استخدامها لتحرير الذاكرة بعد الانتهاء منها.

5. 🔷 تسرب الذاكرة (Memory Leaks)

📌 إذا لم نستخدم free() بعد malloc()، ستبقى الذاكرة محجوزة حتى بعد انتهاء البرنامج، مما يؤدي إلى **تسرب الذاكرة**.
📌 **يجب دائمًا تحرير الذاكرة عند الانتهاء من استخدامها!**

💡 خلاصة المحاضرة:

✅ تعلمنا كيف تعمل **الذاكرة في الكمبيوتر**.
✅ فهمنا **المؤشرات** وكيفية استخدامها للوصول إلى البيانات.
✅ عرفنا كيف يمكن للمؤشرات التعامل مع **المصفوفات**.
✅ استخدمنا **malloc** و **free** لإدارة الذاكرة يدويًا.
✅ تعلمنا أهمية تجنب **تسرب الذاكرة** للحفاظ على كفاءة البرامج.

💡ملخص المحاضرة الخامسة -"Data Structures"

في هذه المحاضرة، تعلمنا عن هياكل البيانات (Data Structures) التي تساعدنا في تخزين البيانات وإدارتها بطريقة أكثر كفاءة💾 .

1. ◆ما هي هياكل البيانات؟

📌هي طرق لتنظيم البيانات داخل الكمبيوتر بحيث يمكننا استخدامها بكفاءة.
📌تساعدنا على تسريع العمليات مثل البحث، الإضافة، والحذف.

2. ◆المصفوفات(Arrays)

📌المصفوفة هي مجموعة من العناصر المخزنة في الذاكرة بتسلسل متجاور.
📌كل عنصر يمكن الوصول إليه باستخدام الفهرس.(Index)

🔹 **مثال:**

Edit Copy

```c
#include <stdio.h>

int main(void) {
    int numbers[3] = {10, 20, 30};

    printf("العنصر الأول: %d\n", numbers[0]);
    printf("العنصر الثاني: %d\n", numbers[1]);
    printf("العنصر الثالث: %d\n", numbers[2]);
}
```

📌 المشكلة في المصفوفات؟

- حجمها ثابت ولا يمكن تغييره بعد التعريف.
- عمليات الإضافة والحذف قد تكون مكلفة من حيث الأداء.

3. ◆القوائم المتصلة(Linked Lists)

📌هيكل بيانات أكثر مرونة من المصفوفة، حيث يمكن تغيير حجمه بسهولة.
📌تتكون من **عقد(Nodes)** ، وكل عقدة تحتوي على:

- **القيمة.(Value)**
- **مؤشر إلى العقدة التالية.(Pointer to next node)**

🔹 مثال على قائمة متصلة بسيطة:

Edit Copy

```c
#include <stdio.h>
#include <stdlib.h>

// تعريف العقدة
typedef struct node {
    int value;
    struct node *next;
} node;

int main(void) {
    node *head = malloc(sizeof(node)); // تخصيص ذاكرة للعقدة الأولى
    head->value = 10;
    head->next = malloc(sizeof(node)); // عقدة ثانية
    head->next->value = 20;
    head->next->next = NULL; // نهاية القائمة

    printf("العنصر الأول: %d\n", head->value);
    printf("العنصر الثاني: %d\n", head->next->value);

    free(head->next);
    free(head); // تحرير الذاكرة
}
```

📌 **لماذا نستخدم القوائم المتصلة؟**

✅ يمكن تغيير حجمها بسهولة.

✅ عمليات الإضافة والحذف أسرع مقارنة بالمصفوفات.

❌ تحتاج إلى مساحة إضافية لتخزين المؤشرات.

4. المكدسات(Stacks) 🔷

📌 **هي بنية بيانات تعمل بطريقة "LIFO" (Last In, First Out)** أي أن العنصر الأخير الذي يدخل هو الأول الذي يخرج.

📌 العمليات الأساسية:

- **(push():)** إضافة عنصر.
- **(pop():)** إزالة العنصر العلوي.

🔹 **مثال:**

Edit Copy

```c
#include <stdio.h>
#define SIZE 5

int stack[SIZE], top = -1;

void push(int value) {
    if (top == SIZE - 1) {
        printf("المكدس ممتلئ!\n");
    } else {
        stack[++top] = value;
    }
}

int pop() {
    if (top == -1) {
        printf("المكدس فارغ!\n");
        return -1;
    } else {
        return stack[top--];
    }
}

int main(void) {
    push(10);
    push(20);
    printf("تمت إزالة: %d\n", pop()); // 20 يزيل
}
```

📌 أين تُستخدم المكدسات؟

- عند تنفيذ الدوال العودية (Recursion).
- في التراجع في المتصفحات (Browser Back Button).

5. الطوابير (Queues) 🔶

📌 **هيكل بيانات يعمل بطريقة "FIFO" (First In, First Out)** أي أن العنصر الأول الذي يدخل هو الأول الذي يخرج.
📌 العمليات الأساسية:

- **enqueue()**: إدخال عنصر.
- **dequeue()**: إزالة العنصر الأول.

🔹 **مثال:**

Edit Copy

```c
#include <stdio.h>
#define SIZE 5

int queue[SIZE], front = 0, rear = 0;

void enqueue(int value) {
    if (rear == SIZE) {
        printf("الطابور ممتلئ!\n");
    } else {
        queue[rear++] = value;
    }
}

int dequeue() {
    if (front == rear) {
        printf("الطابور فارغ!\n");
        return -1;
    } else {
        return queue[front++];
    }
}

int main(void) {
    enqueue(10);
    enqueue(20);
    printf("تمت إزالة: %d\n", dequeue()); // يزيل 10
}
```

📌 **أين تُستخدم الطوابير؟**

- في إدارة المهام (Task Scheduling).
- في الطابور في الطابعات (Printer Queue).

💡خلاصة المحاضرة:

✅ **المصفوفات (Arrays):** تخزين متسلسل لكن بحجم ثابت.
✅ **القوائم المتصلة (Linked Lists):** مرنة لكن تحتاج إلى مؤشرات.
✅ **المكدسات (Stacks):** تستخدم في **LIFO** آخر داخل، أول خارج.
✅ **الطوابير (Queues):** تستخدم في **FIFO** أول داخل، أول خارج.

ملخص المحاضرة السادسة -"Algorithms"

في هذه المحاضرة، تعلمنا عن الخوارزميات (Algorithms) وكيفية تحسين أداء البرامج من خلال تحليل الكفاءة باستخدام الزمن المستغرق (Time Complexity).

1. ما هي الخوارزميات؟

الخوارزمية هي مجموعة من الخطوات المنطقية التي تحل مشكلة معينة بطريقة فعالة.
يمكن تنفيذها بطرق مختلفة باستخدام هياكل البيانات المختلفة.

2. تحليل كفاءة الخوارزميات(Big O Notation)

Big O Notation تُستخدم لقياس أداء الخوارزميات وفقًا لحجم البيانات المدخلة.
بعض القيم الشائعة:

- **O(1) – ثابتة (Constant Time) أسرع**
- **O(log n) – لوغاريتمية (Logarithmic Time)**
- **O(n) – خطية (Linear Time)**
- **O(n log n) – خطية مضروبة في لوغاريتم**
- **O(n²) – تربيعية (Quadratic Time) بطيئة**
- **O(2ⁿ) – أسية (Exponential Time) بطيئة جدًا**

3. خوارزميات البحث(Searching Algorithms)

البحث الخطي(Linear Search) - O(n)

يبحث عن العنصر عن طريق المرور على كل عنصر في القائمة.

البحث الثنائي(Binary Search) - O(log n)

يعتمد على تقسيم القائمة إلى نصفين والبحث في النصف الصحيح.
يجب أن تكون القائمة مرتبة مسبقًا!

🔹 مثال: البحث الثنائي

Edit Copy

```c
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 10;
    int result = binarySearch(arr, 0, size - 1, target);

    if (result != -1) printf("تم العثور على العنصر في الفهرس %d\n", result);
    else printf("العنصر غير موجود\n");
}
```

4. 🔷 خوارزميات الفرز (Sorting Algorithms)

📌 **تُستخدم لترتيب البيانات لتسريع البحث وتحليلها بشكل أسرع.**

🔷 فرز الفقاعات (Bubble Sort) - $O(n^2)$

📌 يستبدل العناصر المجاورة إذا كانت بترتيب خاطئ.

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
```

🔹فرز الاختيار(Selection Sort) - O(n²)

📌يبحث عن **أصغر عنصر** في كل مرة وينقله إلى موضعه الصحيح.

🔹فرز الدمج(Merge Sort) - O(n log n)

📌يعتمد على **تقسيم القائمة إلى أجزاء صغيرة** ثم دمجها بعد الفرز.

```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}
```

🔹فرز كويك(Quick Sort) - O(n log n)

📌يعتمد على اختيار **عنصر محوري (Pivot)** ثم تقسيم القائمة حوله.

Edit Copy

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high], i = (low - 1);
        for (int j = low; j < high; j++)
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
            }
        int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;

        quickSort(arr, low, i);
        quickSort(arr, i + 2, high);
    }
}
```

خلاصة المحاضرة:

- ✅ **الخوارزميات تساعدنا في حل المشكلات بطريقة فعالة.**
- ✅ **تُستخدم Big O لقياس كفاءة الخوارزميات.**
- ✅ **البحث الثنائي أسرع من البحث الخطي.**
- ✅ **فرز الدمج والكويك أكثر كفاءة من فرز الفقاعات والاختيار.**

ملخص المحاضرة السابعة -"Memory"

في هذه المحاضرة، تعمقنا في إدارة الذاكرة وكيفية التعامل مع المؤشرات(Pointers) ، المصفوفات(Arrays) ، والتخصيص الديناميكي للذاكرة (Dynamic Memory Allocation).

1. ما هي الذاكرة وكيف يتم تخزين البيانات؟

- **الذاكرة في الكمبيوتر عبارة عن سلسلة من العناوين(Addresses) ، كل عنوان يحتوي على قيمة.(Value)**
- **عند تخزين متغير، يتم حفظه في عنوان معين داخل ذاكرة الوصول العشوائي.(RAM)**

2. المؤشرات (Pointers) 🔷

📌 **المؤشر هو متغير يخزن عنوان ذاكرة متغير آخر.**
📌 **يُستخدم في تمرير القيم بالمرجع (Pass by Reference) وفي إدارة الذاكرة الديناميكية.**

🔹 تعريف المؤشر واستخدامه

Edit Copy

```c
#include <stdio.h>

int main() {
    int x = 10;
    int *ptr = &x; // المؤشر يحمل عنوان المتغير x

    printf("قيمة x: %d\n", x);
    printf("عنوان x في الذاكرة: %p\n", &x);
    printf("المؤشر ptr يحمل العنوان: %p\n", ptr);
    printf("القيمة المخزنة في العنوان الذي يشير إليه ptr: %d\n", *ptr);
}
```

📌 `&x` تعطي عنوان `x`.
📌 `ptr` يخزن العنوان.
📌 `ptr*` يعطي القيمة المخزنة في العنوان.

3. الفرق بين Pass by Value و Pass by Reference 🔷

📌 في **تمرير بالقيمة (Pass by Value)** يتم تمرير **نسخة** من المتغير.
📌 في **تمرير بالمرجع (Pass by Reference)** يتم تمرير **العنوان الفعلي للمتغير**، مما يسمح بتغيير قيمته داخل الدالة.

🔹 مثال: تمرير بالمرجع باستخدام المؤشرات

Edit Copy

```c
#include <stdio.h>

void updateValue(int *ptr) {
    *ptr = 20; // تعديل القيمة داخل العنوان
}

int main() {
    int num = 10;
    printf("قبل التحديث: %d\n", num);

    updateValue(&num);

    printf("بعد التحديث: %d\n", num);
}
```

📌 هنا، يتم تمرير **عنوان** `num`، مما يسمح للدالة بتعديله مباشرة.

4. 🔷 التخصيص الديناميكي للذاكرة(Dynamic Memory Allocation)

📌يتم تخصيص الذاكرة **يدويًا** باستخدام **malloc** و**free**.
📌يسمح بإنشاء **هياكل بيانات بحجم غير ثابت** أثناء تشغيل البرنامج.

🔹 استخدام `malloc` و `free`

Edit Copy

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr = (int *)malloc(5 * sizeof(int)); // تخصيص مصفوفة ديناميكيًا

    if (ptr == NULL) {
        printf("فشل تخصيص الذاكرة!\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        ptr[i] = i * 10;
        printf("%d ", ptr[i]);
    }

    free(ptr); // تحرير الذاكرة بعد الاستخدام
    return 0;
}
```

📌 `malloc` تحجز مساحة في الذاكرة.
📌 `free` تُستخدم لتحرير الذاكرة بعد الانتهاء.

5. 🔷 التعامل مع السلاسل النصية في الذاكرة

📌في **C**، يتم تخزين السلاسل النصية في **مصفوفات من الأحرف**.**(char arrays)**
📌كل سلسلة نصية تنتهي بـ **محرف**.**(\0) NULL**

🔹 مثال: إنشاء نسخة من نص باستخدام `malloc`

Edit Copy

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char *original = "Hello";
    char *copy = malloc(strlen(original) + 1); // تخصيص ذاكرة للكلمة

    if (copy == NULL) {
        printf("فشل التخصيص!\n");
        return 1;
    }

    strcpy(copy, original); // نسخ النص إلى الذاكرة الجديدة
    printf("النسخة: %s\n", copy);

    free(copy); // تحرير الذاكرة
    return 0;
}
```

📌 `malloc(strlen(original) + 1)` تحجز مساحة كافية للسلسلة النصية مع محرف `\0`.

📌 `strcpy(copy, original)` تنسخ المحتوى من `original` إلى `copy`.

💡خلاصة المحاضرة:

✅ **الذاكرة في الكمبيوتر تتكون من عناوين وقيم.**

✅ **المؤشرات تُستخدم لتخزين عناوين المتغيرات.**

✅ **تمرير بالمرجع يمكننا من تعديل القيم داخل الدوال.**

✅ **التخصيص الديناميكي للذاكرة يتيح لنا إنشاء هياكل بيانات مرنة.**

✅ **يجب تحرير الذاكرة باستخدام free بعد استخدامها لتجنب تسريبات الذاكرة.**

ملخص المحاضرة الثامنة -"Structs" & "File I/O"💡

في هذه المحاضرة، تناولنا الهياكل(Structs) ، التي تسمح بتخزين بيانات متعددة بأنواع مختلفة في كيان واحد، والتعامل مع الملفات (File I/O) لقراءة وكتابة البيانات على القرص💾📝 .

1. 🔷الهياكل (Structs) في C

📌في C، لا توجد كائنات (Objects) مثل لغات البرمجة الكائنية(OOP) ، ولكن يمكن استخدام الهياكل (Structs) لجمع بيانات متعددة في كيان واحد.

📌مثال :تخزين بيانات طالب(Student)

🔹 تعريف هيكل Struct واستخدامه

Edit Copy

```c
#include <stdio.h>
#include <string.h>

// تعريف هيكل Student
typedef struct {
    char name[50];
    int age;
    float gpa;
} Student;

int main() {
    Student s1; // إنشاء كائن من الهيكل

    // إدخال البيانات
    strcpy(s1.name, "Mahmoud");
    s1.age = 20;
    s1.gpa = 3.8;

    // طباعة البيانات
    printf("Name: %s\n", s1.name);
    printf("Age: %d\n", s1.age);
    printf("GPA: %.2f\n", s1.gpa);

    return 0;
}
```

📌typedef struct يُستخدم لإنشاء اسم مختصر للهيكل.

📌يمكن تخزين بيانات متعددة في كائن واحد من النوع Student.

2. ◈ استخدام المؤشرات مع Structs

📌 يمكن استخدام المؤشرات (Pointers) للوصول إلى بيانات الهياكل Structs بطريقة أكثر كفاءة

🔹 مثال: تمرير Struct بالمرجع

Edit Copy

```c
#include <stdio.h>
#include <string.h>

typedef struct {
    char name[50];
    int age;
} Person;

void updateAge(Person *p) {
    p->age += 1; // استخدام السهم -> للوصول إلى عناصر الـ Struct
}

int main() {
    Person p1 = {"Ahmed", 21};

    printf("العمر قبل التحديث: %d\n", p1.age);

    updateAge(&p1);

    printf("العمر بعد التحديث: %d\n", p1.age);

    return 0;
}
```

📌 **p->age** تعني الوصول إلى **age** داخل الهيكل عند استخدام المؤشر p.
📌 تمرير الهيكل كمؤشر يقلل استهلاك الذاكرة بدلاً من نسخه بالكامل.

3. ◈ التعامل مع الملفات(File I/O)

📌 يمكن للبرامج قراءة وكتابة البيانات في **الملفات** باستخدام **fopen, fprintf, fscanf, fclose**.
📌 أنواع الملفات:

- **ملفات نصية (Text Files)**: تخزن البيانات بتنسيق قابل للقراءة.
- **ملفات ثنائية (Binary Files)**: تخزن البيانات بصيغة غير نصية، مثل الصور.

4. 🔷 فتح وقراءة الملفات النصية

📌 لفتح ملف نستخدم **fopen("filename", "r")** وضع القراءة.

📌 لقراءة البيانات نستخدم **fscanf** أو **fgets**.

🔹 مثال: قراءة ملف نصي

Edit Copy

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("data.txt", "r"); // فتح الملف بوضع القراءة

    if (file == NULL) {
        printf("فشل فتح الملف!\n");
        return 1;
    }

    char name[50];
    int age;

    while (fscanf(file, "%s %d", name, &age) != EOF) { // قراءة البيانات من الملف
        printf("Name: %s, Age: %d\n", name, age);
    }

    fclose(file); // إغلاق الملف بعد الانتهاء
    return 0;
}
```

📌 `fopen("data.txt", "r")` يفتح الملف بوضع **القراءة**.

📌 `fscanf(file, "%s %d", name, &age)` يقرأ البيانات **حتى نهاية الملف (EOF)**.

📌 `fclose(file)` يُغلق الملف لتوفير الموارد.

5. ◈ الكتابة في الملفات النصية

📌 لفتح ملف في وضع الكتابة نستخدم **fopen("filename", "w")** يُنشئ ملف جديد أو يمسح المحتوى الحالي.
📌 يمكننا استخدام **fprintf** لكتابة البيانات .

🔹 مثال: إنشاء وكتابة ملف نصي

Edit Copy

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("output.txt", "w"); // فتح الملف بوضع الكتابة

    if (file == NULL) {
        printf("فشل فتح الملف!\n");
        return 1;
    }

    fprintf(file, "Name: %s\nAge: %d\n", "Mahmoud", 20); // كتابة بيانات في الملف

    fclose(file); // إغلاق الملف
    printf("تم حفظ البيانات في الملف!\n");

    return 0;
}
```

📌 `fopen("output.txt", "w")` يفتح الملف للكتابة (يمسح أي بيانات قديمة).
📌 `fprintf(file, "format", data)` يكتب البيانات إلى الملف.
📌 `fclose(file)` ضروري لإغلاق الملف بعد الاستخدام.

6. ◈ التعامل مع الملفات الثنائية (Binary Files)

📌 تُستخدم **fwrite** و **fread** لكتابة وقراءة الملفات الثنائية.
📌 تُستخدم هذه الطريقة لحفظ **الصور، الصوت، البيانات المعقدة**.

🔹 مثال: حفظ بيانات طالب في ملف ثنائي

Edit Copy

```c
#include <stdio.h>

typedef struct {
    char name[50];
    int age;
    float gpa;
} Student;

int main() {
    Student s1 = {"Omar", 22, 3.9};

    FILE *file = fopen("student.dat", "wb"); // فتح ملف ثنائي للكتابة

    if (file == NULL) {
        printf("فشل فتح الملف!\n");
        return 1;
    }

    fwrite(&s1, sizeof(Student), 1, file); // كتابة الهيكل في الملف

    fclose(file);
    printf("تم حفظ البيانات في ملف ثنائي!\n");

    return 0;
}
```

💡خلاصة المحاضرة:

- ✅ **الهياكل (Structs) توفر طريقة لتخزين بيانات متعددة في كيان واحد.**
- ✅ **المؤشرات تُستخدم للوصول إلى بيانات Structs بشكل أكثر كفاءة.**
- ✅ **الملفات (File I/O) تتيح تخزين البيانات بشكل دائم على القرص.**
- ✅ **fopen, fscanf, fprintf, fclose** تُستخدم للملفات النصية.
- ✅ **fwrite, fread** تُستخدم لحفظ البيانات في ملفات ثنائية.

ملخص المحاضرة التاسعة – Memory (الذاكرة)

في هذه المحاضرة، يتناول CS50 مفهوم **الذاكرة** في الحاسوب، وكيفية التعامل معها في **لغة C** ، بالإضافة إلى الأخطاء الشائعة وإدارتها بكفاءة.

أهم المفاهيم التي تم تناولها:

1 الذاكرة (Memory) وأنواعها

RAM (**ذاكرة الوصول العشوائي**) :المكان الذي تُخزن فيه البيانات مؤقتًا أثناء تشغيل البرنامج.

Stack vs Heap :

- Stack : **تُستخدم لتخزين المتغيرات المحلية وتُدار تلقائيًا.**
- Heap : **تُستخدم لتخصيص الذاكرة ديناميكيًا، لكن تحتاج إلى إدارة يدوية.**

2 المؤشرات (Pointers)

المؤشر هو متغير يخزن **عنوان ذاكرة** متغير آخر.

يتم تعريفه باستخدام *، مثل:

```
int x = 5;
int *ptr = &x; // ptr يخزن عنوان x
```

يتم استخدام &للحصول على عنوان المتغير، و *لفك المرجع.(dereferencing)

3 تخصيص الذاكرة ديناميكيًا (Dynamic Memory Allocation)

يتم استخدام دوال مثل:

- **malloc(): لحجز مساحة من الذاكرة.**
- **free(): لتحرير الذاكرة بعد الاستخدام لمنع تسريبات الذاكرة.(Memory Leaks)**

```
int *arr = malloc(5 * sizeof(int)); // حجز مصفوفة من 5 عناصر
free(arr); // تحرير الذاكرة
```

4 ◈ أخطاء الذاكرة الشائعة ⚠

◈ **Memory Leak تسريب الذاكرة :**يحدث عند نسيان تحرير الذاكرة باستخدام free().
◈ **Segmentation Fault :** يحدث عند محاولة الوصول إلى ذاكرة غير مخصصة.
◈ **Buffer Overflow :**يحدث عند محاولة تخزين بيانات تتجاوز حجم المصفوفة.

خلاصة المحاضرة:

✅فهم كيفية عمل الذاكرة في الحاسوب.
✅استخدام **المؤشرات** للوصول إلى البيانات المخزنة في الذاكرة.
✅التعامل مع **الذاكرة الديناميكية** باستخدام malloc() وfree().
✅تجنب **الأخطاء الشائعة** مثل **Memory Leaks** و**Segmentation Faults**.

ملخص المحاضرة العاشرة - Python

(1) لماذا Python ؟

◈لغة حديثة وسهلة القراءة والكتابة مقارنةً بـ **C**.
◈لا تحتاج إلى تصريف(Compilation) ، بل يتم تشغيلها مباشرةً.
◈تُدير الذاكرة تلقائيًا لا حاجة لـ malloc() وfree().
◈تحتوي على مكتبات قوية تساعد في مجالات مثل **الذكاء الاصطناعي، تحليل البيانات، تطوير الويب**.

(2)المتغيرات وأنواع البيانات

◈في **Python**، لا تحتاج إلى تحديد نوع المتغير، حيث يتم **تحديده تلقائيًا**:

Edit Copy

```
x = 5       # عدد صحيح
y = 3.14    # عدد عشري
name = "CS50"  # نص (String)
is_valid = True  # قيمة منطقية (Boolean)
```

(3) العمليات الأساسية في Python

- العمليات الحسابية (+, -, *, /, ** أس) % (باقي القسمة).
- عمليات المقارنة: ==, !=, <, >, <=, >=.
- العمليات المنطقية: and, or, not.

(4) الحلقات (Loops) والتكرار

- حلقة for:

```
for i in range(5):
    print(i)  # يطبع من 0 إلى 4
```

- حلقة while:

```
x = 0
while x < 5:
    print(x)
    x += 1
```

(5) القوائم (Lists) والمصفوفات

- القوائم في Python تُشبه المصفوفات لكنها أكثر مرونة:

```
numbers = [1, 2, 3, 4, 5]
print(numbers[0])  # طباعة العنصر الأول
numbers.append(6)  # إضافة عنصر للقائمة
```

(6)الدوال (Functions) في Python

🔷يتم تعريف الدوال باستخدام :def

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Mahmoud"))  # Hello, Mahmoud!
```

(7)التعامل مع الملفات 🗁

- لقراءة ملف:

```
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

- لكتابة ملف:

```
with open("file.txt", "w") as file:
    file.write("Hello, CS50!")
```

💡خلاصة المحاضرة:

✅ **Python** لغة سهلة الاستخدام مقارنة بـ **C**.
✅ لا تحتاج إلى تعريف أنواع المتغيرات يدويًا.
✅ توفر **حلقات** و **قوائم** و **دوال** مرنة.
✅ يمكنها التعامل مع **الملفات** بسهولة.

ملخص المحاضرة الحادية عشرة – الذكاء الاصطناعي🤖 (AI)

في هذه المحاضرة، يركز **CS50** على **الذكاء الاصطناعي(AI)** ، موضحًا كيف يمكن للكمبيوتر **محاكاة الذكاء البشري** باستخدام **الخوارزميات** ونماذج **التعلم الآلي**.

📌 أهم المفاهيم التي تم تناولها:

(1) ما هو الذكاء الاصطناعي؟

🔹 الذكاء الاصطناعي هو فرع من علوم الكمبيوتر يهدف إلى **محاكاة** قدرات التفكير والتعلم البشري.
🔹 يمكن تصنيفه إلى **ذكاء اصطناعي ضيق (Narrow AI)** مثل **Siri** أو **Google Assistant**، و**ذكاء اصطناعي عام (AGI)**، وهو ما يزال قيد التطوير.

(2)البحث (Search) في الذكاء الاصطناعي🔍

🔹 تعتمد **الخوارزميات** على البحث للوصول إلى الحلول المثلى.
🔹 أشهر طرق البحث:

- **بحث العمق أولًا (Depth-First Search - DFS)**: يستكشف كل فرع حتى نهايته قبل العودة.
- **بحث العرض أولًا (Breadth-First Search - BFS)**: يستكشف كل مستوى قبل الانتقال إلى المستوى التالي.

📌 مثال عملي على DFS في Python:

Edit Copy

```python
def dfs(graph, node, visited=set()):
    if node not in visited:
        print(node)
        visited.add(node)
        for neighbor in graph[node]:
            dfs(graph, neighbor, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}

dfs(graph, 'A')
```

(3)الذكاء الاصطناعي والاحتمالات📊

🔷في **التعلم الآلي**، نستخدم **الاحتمالات** لاتخاذ قرارات بناءً على البيانات.
🔷مثال على **مبرهنة بايز(Bayes' Theorem)** ، التي تساعد في **التنبؤ بالأحداث** بناءً على البيانات السابقة.

📌 مثال عملي في Python لحساب احتمالات بسيطة باستخدام مكتبة NumPy:

Edit Copy

```
import numpy as np

data = np.random.choice(["سليم" ,"مرض"], p=[0.2, 0.8], size=1000)
prob = np.sum(data == "مرض") / len(data)
print(f"احتمال الإصابة بالمرض: {prob:.2f}")
```

(4)التعلم الآلي🤖 (Machine Learning)

🔷يتم تدريب النماذج على البيانات لاستخراج **أنماط** والتنبؤ بالمستقبل.
🔷أشهر **أنواع التعلم الآلي**:

- **التعلم الخاضع للإشراف (Supervised Learning)**: النموذج يتعلم من بيانات مصنفة مسبقًا.
- **التعلم غير الخاضع للإشراف (Unsupervised Learning)**: النموذج يستخرج الأنماط دون تصنيفات مسبقة.
- **التعلم التعزيزي (Reinforcement Learning)**: النموذج يتعلم من خلال التجربة والخطأ) مثل.(AlphaGo

📌 مثال بسيط على تعلم آلي باستخدام Scikit-Learn:

Edit Copy

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])  # المدخلات
y = np.array([2, 4, 6, 8, 10])  # المخرجات

model = LinearRegression()
model.fit(X, y)  # تدريب النموذج

print(model.predict([[6]]))  # توقع ناتج جديد
```

خلاصة المحاضرة:

- ✅ **الذكاء الاصطناعي** يهدف إلى محاكاة التفكير البشري باستخدام **الخوارزميات**.
- ✅ **خوارزميات البحث (DFS, BFS)** تُستخدم لإيجاد حلول مثلى.
- ✅ **الاحتمالات** تلعب دورًا رئيسيًا في **التنبؤات**.
- ✅ **التعلم الآلي** هو أحد أقوى تطبيقات الذكاء الاصطناعي، ويستخدم في **التصنيف والتنبؤ**.

هذا الملخص مجرد نقطة انطلاق، لكن التعلم الحقيقي يأتي من التطبيق والممارسة.
استمر في الاستكشاف والتجربة، فالمعرفة لا حدود لها!

"في الختام، لا تنسوا الدعاء لإخواننا في غزة، اللهم كن لهم عونًا ونصيرًا، وارفع عنهم البلاء، واحفظهم بحفظك يا رحمن.♡PS "