الوحدة التدريبية الخامسة
المساهمة في تطوير التطبيقات

رمز الوحدة
ADS-U5-CAVT

المتطلبات السابقة:

لا يوجد

نتاجات التعلم:

عند الانتهاء من دراسة هذه الوحدة واكتساب مهاراتها الأدائية والاتجاهات السلوكية الصحيحة والعلوم المهنية المرافقة من خلال التفاعل مع أنشطتها وخبراتها المختلفة يصبح المتدرب قادراً على أداء نتاجات التعلم الآتية:

1. يستخدم لغة برمجة كائنية (object-oriented programming) أو غيرها.
2. يوثق البرنامج بالتعليمات المناسبة حسب إجراءات الشركة و أفضل تعليمات البرمجة الهيكلية.
3. يشارك في تقديم اقتراحات لتوظيف مبادئ وإرشادات تصميم واجهة المستخدم/ تجربة المستخدم UI/UX .
4. يشارك في تنفيذ تقنيات وأطر الواجهة الامامية لتنفيذ تصميمات واجهة المستخدم/تجربة المستخدم UI/UX بشكل فعال.

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

روابط التعلم الإلكتروني

سيتم تزويد المتدربين بروابط التعلم الإلكتروني من خلال المدرب

- انظر إلى تطبيق معقد حيث تعمل الميزات المختلفة معًا بسلاسة. هل تساءلت يومًا كيف يدير المطورون مثل هذا التعقيد؟

1. استخدام لغات البرمجة الكينونية (Object-oriented programming)

- **مقدمة في البرمجة الكينونية (OOP)**

- **تعريف OOP**

- البرمجة الكينونية (OOP) هي نموذج برمجة يعتمد على مفهوم "الكائنات"، والتي يمكن أن تحتوي على بيانات، على شكل حقول، تُعرف غالبًا باسم السمات، أكواد برمجية، على شكل إجراءات، تُعرف غالبًا باسم طرق.
- الكائن هو مثيل لفئة، حيث يمكن اعتبار الفئة بمثابة مخطط للكائنات.

- **أهمية OOP :**

- النمطية: تسمح OOP بالتصميم المعياري للبرامج عن طريق تقسيم المشكلات المعقدة إلى أجزاء أصغر وأكثر قابلية للإدارة.
- قابلية إعادة الاستخدام: يمكن إعادة استخدام الكود عبر برامج متعددة من خلال الوراثة واستخدام مكتبات الكائنات.
- قابلية التوسع: يسهل OOP إدارة مشاريع البرامج الكبيرة ويتيح إنشاء أنظمة أكثر تعقيدًا وقابلة للتطوير.
- قابلية الصيانة: التحسينات والصيانة أسهل بسبب تغليف البيانات والأساليب داخل الكائنات، مما يقلل من الترابط.
- الإنتاجية: يتم تحسين كفاءة التطوير باستخدام الكائنات والمكتبات الموجودة مسبقًا، مما يؤدي إلى إكمال المشروع بشكل أسرع وأكثر فعالية من حيث التكلفة.

- **المفاهيم الأساسية لـ OOP :**

- **الفئات والكائنات**

- الفئات: الفصل عبارة عن مخطط أو قالب لإنشاء الكائنات. فهو يحدد مجموعة من السمات والأساليب التي ستمتلكها الكائنات التي تم إنشاؤها.
- مثال: فئة Car ذات سمات مثل color وmodel وطرق مثل startEngine() stopEngine() .
- الكائنات: الكائن هو مثيل لفئة. إنه يمثل تطبيقًا محددًا للفئة ويمكنه أداء الوظائف المحددة بواسطة الفئة.
- مثال: myCar هو كائن من فئة Car ذو سمات محددة (على سبيل المثال، اللون: أحمر، الطراز: Tesla).

- **ميراث**

- يسمح الوراثة لفئة جديدة بوراثة خصائص وأساليب فئة موجودة، مما يعزز إمكانية إعادة استخدام التعليمات البرمجية وإنشاء علاقات هرمية بين الفئات.
- مثال: يمكن أن ترث فئة ElectricCar من فئة Car، مع إضافة سمات وطرق محددة تتعلق بالسيارات الكهربائية.

- **تعدد الأشكال**

- يسمح تعدد الأشكال بمعاملة الكائنات كمثيلات لفئتها الأصلية بدلاً من فئتها الفعلية. فهو يوفر طريقة لاستخدام واجهة واحدة لتمثيل أنواع البيانات الأساسية المختلفة.
- مثال: يمكن استخدام الطريقة ()drive مع أنواع مختلفة من كائنات Car (على سبيل المثال، ElectricCar، GasolineCar)، حيث ينفذ كل منها نسخته الخاصة من الطريقة.

- **التغليف**

- التغليف هو آلية لتقييد الوصول المباشر إلى بعض مكونات الكائن وأساليبه ويمكن أن يمنع التعديل العرضي للبيانات.
- مثال: لا يمكن تعديل السمات الخاصة في الفصل الدراسي إلا من خلال الطرق العامة، مما يضمن التحكم في الوصول والتعديل.

- **التجريد**

- يعمل التجريد على تبسيط الأنظمة المعقدة من خلال نمذجة الفئات المناسبة للمشكلة، مع التركيز على الصفات الأساسية بدلاً من التفاصيل المحددة.
- مثال: يمكن أن تكون فئة "مركبة" تجريدًا يمثل أنواعًا مختلفة من المركبات، مثل السيارات والشاحنات والدراجات، بدون تفاصيل تفاصيل كل مركبة.

- **تمارين وأمثلة عملية**

1. مثال على إنشاء كائن بإستخدام لغة Python

```
Class and Object Creation
  python
  class Car:
      def __init__(self, color, model):
          self.color = color
          self.model = model

      def startEngine(self):
          print("Engine started")

      def stopEngine(self):
          print("Engine stopped")

  myCar = Car("red", "Tesla")
  print(myCar.color)   Output: red
  myCar.startEngine()   Output: Engine started
```

2. مثال على التوريث بإستخدام لغة Python

```
Inheritance Example
  python
  class ElectricCar(Car):
      def __init__(self, color, model, battery_capacity):
          super().__init__(color, model)
          self.battery_capacity = battery_capacity

      def chargeBattery(self):
          print("Battery charging")

  myElectricCar = ElectricCar("blue", "Tesla", "100 kWh")
  myElectricCar.startEngine()   Output: Engine started
  myElectricCar.chargeBattery()   Output: Battery charging
```

3. مثال على تعدد الأشكال بإستخدام لغة Python

```
Polymorphism Example
  python
  class GasolineCar(Car):
      def refuel(self):
          print("Refueling gasoline")

  def start_vehicle(vehicle):
      vehicle.startEngine()

  myGasolineCar = GasolineCar("green", "Toyota")
  start_vehicle(myCar)   Output: Engine started (from Car class)
  start_vehicle(myElectricCar)   Output: Engine started (from ElectricCar class)
  start_vehicle(myGasolineCar)   Output: Engine started (from GasolineCar class)
```

4. مثال على التغليف بإستخدام لغة Python

```
Encapsulation Example
  python
  class Account:
      def __init__(self, owner, balance):
          self.owner = owner
          self.__balance = balance   Private attribute

      def deposit(self, amount):
          self.__balance += amount

      def withdraw(self, amount):
          if amount <= self.__balance:
              self.__balance -= amount
          else:
              print("Insufficient funds")

      def get_balance(self):
          return self.__balance

  myAccount = Account("John Doe", 1000)
  myAccount.deposit(500)
  print(myAccount.get_balance())   Output: 1500
  myAccount.withdraw(2000)   Output: Insufficient funds
```

5. مثال على التجريد بإستخدام لغة Python

```
Abstraction Example
  python
  from abc import ABC, abstractmethod

  class Vehicle(ABC):
      @abstractmethod
      def move(self):
          pass

  class Car(Vehicle):
      def move(self):
          print("Car is moving")

  class Bike(Vehicle):
      def move(self):
          print("Bike is moving")

  myCar = Car()
  myBike = Bike()
  myCar.move()   Output: Car is moving
  myBike.move()   Output: Bike is moving
```

- **نظرة عامة على اللغات OOP :**

توفر لغات البرمجة الكينونية (OOP) إطارًا لبناء البرامج بطريقة تمثل كيانات العالم الحقيقي باستخدام الفئات والكائنات. يقدم هذا القسم بعض لغات OOP الأكثر استخدامًا، ويستكشف ميزاتها وتركيب جملتها وتطبيقاتها النموذجية.

- **لغة البرمجة جافا (Java) :**

هي لغة OOP واسعة الاستخدام ومستقلة عن النظام الأساسي ومتعددة الاستخدامات. ، مما يجعلها مناسبة لمجموعة واسعة من التطبيقات، بدءًا من تطوير الويب وحتى حلول المؤسسات.

- **دلائل الميزات:**

أ. إستقلال النظام الأساسي: الكتابة مرة واحدة، والتشغيل في أي مكان (WORA) بفضل Java Virtual Machine (JVM).

ب. مكتبة قياسية قوية: تدعم المكتبات والأطر الشاملة المهام المختلفة.

ج. إدارة الذاكرة: يساعد التجميع التلقائي للبيانات المهملة على إدارة الذاكرة بكفاءة.

مثال

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

- **التطبيقات التي تستخدم بها اللغة :**
  - أ. تطبيقات المؤسسات: تستخدم في أنظمة المؤسسات واسعة النطاق نظرًا لقوتها وأمانها.
  - ب. تطوير Android: اللغة الأساسية لتطوير تطبيقات Android.
  - ج. تطبيقات الويب: غالبًا ما تستخدم مع أطر عمل مثل Spring لبناء تطبيقات الويب.

- **لغة البرمجة (C++) :**

تعد لغة C++ امتدادًا للغة البرمجة C، حيث تضيف ميزات OOP إلى قدرات البرمجة الإجرائية السابقة لها. فهي توفر تحكمًا دقيقًا في موارد النظام والذاكرة.

- **دلائل الميزات:**
  - أ. الأداء: أداء وكفاءة عالية، مناسب للبرمجة على مستوى النظام.
  - ب. نماذج متعددة: يدعم البرمجة الإجرائية والموجهة للكائنات.
  - ج. مكتبة قياسية غنية: توفر مكتبة النماذج القياسية (STL) هياكل بيانات وخوارزميات مفيدة.

```
Syntax Example
cpp
include <iostream>
using namespace std;

class HelloWorld {
public:
    void display() {
        cout << "Hello, World!" << endl;
    }
};

int main() {
    HelloWorld obj;
    obj.display();
    return 0;
}
```

- **التطبيقات التي تستخدم بها اللغة :**
  - أ. برامج النظام: أنظمة التشغيل وبرامج التشغيل والأنظمة المدمجة.
  - ب. تطوير الألعاب: تستخدم على نطاق واسع في صناعة الألعاب للتطبيقات ذات الأداء الحيوي.
  - ج. أنظمة الوقت الحقيقي: التطبيقات التي تتطلب الأداء في الوقت الحقيقي والتفاعل مع الأجهزة.

- **لغة البرمجة بايثون (Python) :**

بايثون هي لغة مفسرة وعالية المستوى ومكتوبة ديناميكيًا ومعروفة بسهولة القراءة والبساطة. غالبًا ما يتم استخدامه للتطوير السريع للتطبيقات والبرمجة النصية.

- **دلائل الميزات :**

أ. سهولة التعلم: بناء جملة بسيط وسهل القراءة، مما يجعله خيارًا ممتازًا للمبتدئين.

ب. الكتابة الديناميكية: لا حاجة للإعلان عن الأنواع المتغيرة، مما يجعلها عملية البرمجة أسرع.

ج. المكتبات الشاملة: مجموعة كبيرة من المكتبات في مجالات مختلفة مثل تطوير الويب وعلوم البيانات والتعلم الآلي.

```
Syntax Example
python
class HelloWorld:
    def display(self):
        print("Hello, World!")

obj = HelloWorld()
obj.display()
```

- **التطبيقات التي تستخدم بها اللغة :**

أ. تطوير الويب: تحظى أطر العمل مثل Django وFlask بشعبية كبيرة في إنشاء تطبيقات الويب.

ب. علوم البيانات والتعلم الآلي: مكتبات مثل NumPy وPandas وTensorFlow.

ج. الأتمتة والبرمجة النصية: تستخدم على نطاق واسع لكتابة البرامج النصية للأتمتة وأتمتة المهام.

ملخص

يعد فهم لغات البرمجة الكينونية OOP المختلفة وتطبيقاتها أمرًا ضروريًا للمتدربين لاختيار الأداة المناسبة لمتطلبات مشروعهم المحددة. تتمتع كل من Java وC++ وPython بنقاط قوة فريدة ومناسبة لأنواع مختلفة من مهام تطوير البرامج. ومن خلال استكشاف هذه اللغات، يمكن للمتدربين تطوير مجموعة مهارات متعددة الاستخدامات تنطبق على مجموعة واسعة من سيناريوهات البرمجة.

- **تمارين عملية**

أ. **تمرين Java** : قم بإنشاء برنامج Java بسيط يحدد فئة Car بسمات مثل make وmodel وطرق مثل startEngine() إنشاء مثيل للكائنات واستدعاء أساليبها.

ب. **تمرين C++** : اكتب برنامج C++ يعرّف فئة "Person" بالسمات "name" و"age" والطرق " displayDetails()". استخدم المُنشئين لتهيئة الكائنات.

ج. **تمرين بايثون** : قم بتطوير برنامج بايثون النصي الذي ينشئ فئة كتاب بالسمات العنوان والمؤلف، والطريقة getInfo(). إنشاء مثيل للفصل وطباعة تفاصيل الكتاب.

- **تمارين عملية(كتابة برامج بسيطة باستخدام لغة OOP)**

التمرين 1: برنامج Hello World

التعليمات: اكتب برنامجًا بسيطًا بلغة OOP التي اخترتها (Java أو C++ أو Python أو C) يطبع "Hello, World!" إلى وحدة التحكم.

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

```
C++
    include <iostream>
    using namespace std;

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

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

التمرين 2: آلة حاسبة بسيطة

التعليمات: اكتب برنامجًا يأخذ رقمين كمدخلات من المستخدم ويقوم بإجراء العمليات الحسابية الأساسية (الجمع والطرح والضرب والقسمة).

```
Java
    import java.util.Scanner;

    public class Calculator {
        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter first number: ");
            double num1 = scanner.nextDouble();
            System.out.print("Enter second number: ");
            double num2 = scanner.nextDouble();

            System.out.println("Addition: " + (num1 + num2));
            System.out.println("Subtraction: " + (num1 num2));
            System.out.println("Multiplication: " + (num1  num2));
            System.out.println("Division: " + (num1 / num2));
        }
    }
```

```
C++:
    include <iostream>
    using namespace std;

    int main() {
        double num1, num2;
        cout << "Enter first number: ";
        cin >> num1;
        cout << "Enter second number: ";
        cin >> num2;

        cout << "Addition: " << num1 + num2 << endl;
        cout << "Subtraction: " << num1 num2 << endl;
        cout << "Multiplication: " << num1  num2 << endl;
        cout << "Division: " << num1 / num2 << endl;

        return 0;
    }
```

```
Python
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    print("Addition:", num1 + num2)
    print("Subtraction:", num1 num2)
    print("Multiplication:", num1  num2)
    print("Division:", num1 / num2)
```

- **تمارين عملية (إنشاء الكائنات ومعالجتها) :**

التمرين 1: إنشاء الفئة والكائن

التعليمات: حدد فئة شخص بالسمات الاسم والعمر، والطريقة displayDetails(). قم بإنشاء كائن للفئة واستدعاء الطريقة لعرض التفاصيل.

```
Java
    public class Person {
        String name;
        int age;

        public Person(String name, int age) {
            this.name = name;
            this.age = age;
        }

        public void displayDetails() {
            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
        }

        public static void main(String[] args) {
            Person person = new Person("John Doe", 30);
            person.displayDetails();
        }
    }
```

C++

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

class Person {
public:
    string name;
    int age;

    Person(string name, int age) {
        this->name = name;
        this->age = age;
    }

    void displayDetails() {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

int main() {
    Person person("John Doe", 30);
    person.displayDetails();
    return 0;
}
```

Python

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def displayDetails(self):
        print("Name:", self.name)
        print("Age:", self.age)

person = Person("John Doe", 30)
person.displayDetails()
```

- **تمارين عملية (تطبيق التوريث (Inheritance) وتعدد الأشكال) .**

التمرين 1: مثال Inheritance

التعليمات: قم بإنشاء فئة أساسية Animal باستخدام طريقة ()makeSuond . أنشئ فئات مشتقة Dog وCat تتجاوز طريقة ()makeSuond إنشاء مثيل لكائنات Dog وCat واستدعاء أساليب ()makeSuond الخاصة بهم.

```
Java
    class Animal {
        public void makeSound() {
            System.out.println("Animal sound");
        }
    }

    class Dog extends Animal {
        @Override
        public void makeSound() {
            System.out.println("Woof");
        }
    }

    class Cat extends Animal {
        @Override
        public void makeSound() {
            System.out.println("Meow");
        }
    }

    public class Main {
        public static void main(String[] args) {
            Animal myDog = new Dog();
            Animal myCat = new Cat();
            myDog.makeSound(); // Output: Woof
            myCat.makeSound(); // Output: Meow
        }
    }
```

C++

```
include <iostream>
using namespace std;

class Animal {
public:
    virtual void makeSound() {
        cout << "Animal sound" << endl;
    }
};

class Dog : public Animal {
public:
    void makeSound() override {
        cout << "Woof" << endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        cout << "Meow" << endl;
    }
};

int main() {
    Animal myDog = new Dog();
    Animal myCat = new Cat();
    myDog->makeSound(); // Output: Woof
    myCat->makeSound(); // Output: Meow

    delete myDog;
    delete myCat;
    return 0;
}
```

Python

```
class Animal:
    def makeSound(self):
        print("Animal sound")

class Dog(Animal):
    def makeSound(self):
        print("Woof")

class Cat(Animal):
    def makeSound(self):
        print("Meow")

myDog = Dog()
myCat = Cat()
myDog.makeSound()   Output: Woof
myCat.makeSound()   Output: Meow
```

التمرين 2: مثال على تعدد الأشكال (Polymorphism)

التعليمات: أنشئ فئة أساسية "الشكل" باستخدام طريقة "الرسم ()". قم بإنشاء فئات مشتقة Circle وRectangle تتجاوز طريقة draw() استخدم تعدد الأشكال لاستدعاء الأسلوب draw() على كائنات Circle وRectangle من خلال مرجع من النوع Shape.

Java

```
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a rectangle");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape myCircle = new Circle();
        Shape myRectangle = new Rectangle();
        myCircle.draw(); // Output: Drawing a circle
        myRectangle.draw(); // Output: Drawing a rectangle
    }
}
```

C++

```
include <iostream>
using namespace std;

class Shape {
public:
    virtual void draw() {
        cout << "Drawing a shape" << endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a circle" << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() override {
        cout << "Drawing a rectangle" << endl;
    }
};

int main() {
    Shape myCircle = new Circle();
    Shape myRectangle = new Rectangle();
    myCircle->draw(); // Output: Drawing a circle
    myRectangle->draw(); // Output: Drawing a rectangle

    delete myCircle;
    delete myRectangle;
    return 0;
}
```

Python

```
class Shape:
    def draw(self):
        print("Drawing a shape")

class Circle(Shape):
    def draw(self):
        print("Drawing a circle")

class Rectangle(Shape):
    def draw(self):
        print("Drawing a rectangle")

myCircle = Circle()
myRectangle = Rectangle()
myCircle.draw()   Output: Drawing a circle
myRectangle.draw()   Output: Drawing a rectangle
```

اسم التمرين: **إنشاء واستخدام Java class بسيطة** رقم التمرين: 1/6

الزمن المخصص لتنفيذ التمرين:

اهداف التمرين العملي:

- التعرف على مفهوم الفئات (Classes) في Java
  - فهم دور الفئات في البرمجة الكائنية التوجه (OOP).
  - معرفة كيفية تعريف الفئة باستخدام الكلمة المفتاحية class.
  - إنشاء فئة (Class) بسيطة
- تحديد الخصائص (Attributes) باستخدام المتغيرات.
  - إنشاء الدوال (Methods) لتنفيذ العمليات داخل الفئة.
  - استخدام المُنشئ (Constructor)
- تعلم كيفية إنشاء مُنشئ لتهيئة كائنات الفئة.
  - فهم الفرق بين المُنشئ الافتراضي والمُنشئ المُخصّص.
  - إنشاء كائنات من الفئة (Objects)
- استخدام الكلمة المفتاحية new لإنشاء كائنات.
  - استدعاء الدوال والوصول إلى الخصائص من خلال الكائن.
  - تنفيذ العمليات الأساسية على الكائنات
- تعيين القيم واسترجاعها باستخدام الـ Getters و Setters.
  - طباعة معلومات الكائن باستخدام دالة toString
  - التعامل مع تعدد الكائنات (Multiple Objects)
- إنشاء عدة كائنات من نفس الفئة والتعامل معها.

تعليمات المدرب:

يقدم المدرب توضيح للخطوات حسب تعليمات السلامة المتبعة في المملكة الاردنية الهاشمية ويراعي ان يوضح كل من التعليمات التالية والتي تتفق مع اجراءات السلامة حسب القو انين المرعية.

- **الخروج من موقع العمل عند سماع جرس الانذار.**
- **اسلك الطرف الايسر من الممرات الامنة و اترك الطرف الايمن خاليا**
- **عند الخروج من الغرف تفقد اتجاه الخطر قبل الخروج مثلا تفقد يد الباب عند الحريق للتأكد من عدم وجود حرارة مرتفعة.**

- **اغلاق الباب عند الخروج من الاماكن المغلقة لمنع انتشار الحريق**
- **الصراخ عند الخروج او استخدام مكبر الصوت منبها الاخرين بكلمة اخلاء مع التكرير عند كل نقطة**
- **الاتصال بالدفاع المدني إذا تطلبت الحاجة يوضع رقم الدفاع المدني امام المتدربين بطريقة ظاهرة , رقم الطوارئ الموحد 911.**
- **التوجه الى مو اقع التجمع المحددة.**
- **تفقد فريق عملك وحدد الغائبين.**
- **ابلاغ فريق الانقاذ بأعداد الحضورواسماء الغائبين.**

التسهيلات التدريبية اللازمة لتنفيذ التمرين العملي :

الاجهزة والآلات	العدد والادوات	المواد الاولية
✓ أدوات الاتصال والمراسلة • أدوات ادارة المشروع • أدوات التحرير والتوثيق مثلMicrosoft Office • منصات التواصل الاجتماعي أدوات لتحليل أنماط التعاون ومقاييس الاتصال وتقدم المشروع مثل , Google Analytics Microsoft Power point	أجهزة الحاسوب، الشاشات، انترنت عالي السرعة، سماعات الرأس، كاميرات الويب، الواح تفاعلية، طابعات، ماسحات ضوئية	• قرطاسية (ورق للكتابة ، اقلام حبر، اقلام تخطيط • أدوات العصف الذهني مثل السبورة الورقية Flip Chart

خطوات العمل لتنفيذ التمرين	
خطوات العمل ومعايير ادائها (الرقمية & الوصفية)	الشكل / الصورة
إنشاء ال Class -اختر اسمًا مناسبًا: بما أننا نمثل كتابًا، سيكون اسم ال class هو Book. -أنشئ ملف Java جديدًا: قم بإنشاء ملف جديد بامتداد.java ، وسمِّه.Book.java -اكتب تعريف ال class: داخل الملف Book.java، ابدأ بكتابة تعريف ال class	
إضافة الخصائص(Attributes) -حدد أنواع البيانات -أضف الخصائص داخل ال class	
إضافة الطرق (Methods) - أنشئ طريقة لطباعة معلومات الكتاب	
انشاء كائن (Object) من ال Class	
تقييم جودة المنتج: وهي مواصفات جودة ودقة ناتج الأداء المطلوب سواء كان منتج أو خدمة	
وضوح الأهداف والتخطيط	
تفاعل المشاركين والمشاركة الفعالة	
جودة المحتوى والأدوات المستخدمة	
تأثير التمرين على المشاركين	
التقييم والتغذية الراجعة	
تقييم الاتجاهات والسلامة المهنية: وهي الاتجاهات المتعلقة بالمحافظة على الأجهزة والأدوات وترتيبها وتنظيفها وتخزينها والسلامة المهنية المتعلقة بالأشخاص والآلات والمعدات والمواد أثناء تأدية التمرين العملي	

الوعي والالتزام بقواعد السلامة	
السلوكيات والممارسات اليومية	
الثقافة العامة للسلامة المهنية	
تقييم بيئة العمل	
الالتزام باللوائح والقوانين	
التدريب والتطوير	

تقييم التمرين العملي رقم (1/6) **ترفق بعد كل بطاقة تمرين عملي**

قيم ادائك بعد انتهائك من تنفيذ التمرين من خلال:

- ضع اشارة " ✓ " بجانب الخطوات التي اتقنتها.
- ضع إشارة " × " بجانب الخطوات التي لم تتقنها مع ذكر الاسباب وكيف يمكنك تحسينها بالمناقشة مع مدربك
- ضع إشارة " × " بجانب الخطوات غير القابلة للتطبيق مع ذكر الاسباب المؤدية لعدم قابلية تطبيق وتنفيذ الخطوة وكيف يمكنك اعادة تطبيقها بإتقان وتحسينها بالمناقشة مع مدربك.

الرقم	اسم الخطوة **(عليك اتقان جميع الخطوات أدناه حتى تظهر درجة الإتقان للأداء المطلوب في التمرين)**	اتقنها	لا اتقنها	غير قابلة للتطبيق	ذكر اسباب عدم الاتقان ولماذا هي غير قابلة للتطبيق
1	هل تم تحديد هدف التمرين				
2	هل استطاع شرح أهمية إنشاء واستخدام Java class بسيطة				
3	هل قام بتحليل استخدام الوقت الحالي				
4	هل قام بتقديم أدوات وتقنيات إنشاء واستخدام Java class بسيطة				
5	هل تم توضيح الأهداف والتخطيط				
6	هل كان جودة المحتوى والأدوات المستخدمة جيدة				
7	هل هنالك التزام بقواعد السلامة				
8	السلوكيات والممارسات اليومية				
9	الثقافة العامة للسلامة المهنية				
10	هل بيئة العمل جيدة				
11	هل تم الالتزام باللوائح والقوانين				
أنجزت اداء التمرين العملي المطلوب		ضمن الوقت المحدد	زيادة عن الوقت المحدد	غير قابل للإنجاز	ذكر اسباب: ✓ زيادة عن الوقت المحدد ✓ ولماذا التمرين غير قابل للإنجاز

- انظر إلى مدى كفاءة برنامجك المفضل وقابليته للصيانة. هل تساءلت يوما لماذا؟
- اكتشف أنماط البرمجة والتصميم المنظمة، وأفضل ممارسات الصناعة لكتابة تعليمات برمجية واضحة وقوية وقابلة لإعادة الاستخدام.

2. إجراءات البرمجة الهيكلية بما يتوافق مع أفضل ممارسات صناعة البرمجة الهيكلية (Design Pattern) (أنماط التصميم)

- **مقدمة في البرمجة الهيكلية**

- **تعريف البرمجة الهيكلية**

هي نموذج برمجة يهدف إلى تحسين وضوح وجودة ووقت تطوير برنامج كمبيوتر باستخدام نهج منضبط في الترميز. ويؤكد على استخدام هياكل التحكم الواضحة والبسيطة مثل التسلسلات والحلقات والشروط، مع تقليل أو القضاء على استخدام التفرعات التعسفية، مثل عبارة "goto". تشجع البرمجة المنظمة على تقسيم البرنامج إلى وظائف أو وحدات أصغر يمكن إدارتها، كل منها مصممة لأداء مهمة محددة. تعمل هذه الوحدات على تعزيز قابلية القراءة والصيانة وتصحيح الأخطاء، مما يجعل الكود أسهل للاختبار وإعادة الاستخدام.

- **مبادئ البرمجة الهيكلية**

1. النمطية: ينقسم البرنامج إلى أقسام أو وحدات متميزة، كل منها مسؤول عن جزء معين من الوظيفة. هذا التصميم المعياري يجعل الكود أسهل في الفهم والاختبار والصيانة.
2. التدفق المتسلسل: تتبع البرمجة الهيكلية تسلسلًا واضحًا للتنفيذ، مع بداية ووسط ونهاية محددة. يضمن هذا المبدأ أن البرنامج ينفذ التعليمات بترتيب معين.
3. هياكل التحكم: تستخدم ثلاثة أنواع رئيسية من هياكل التحكم لتحديد تدفق التنفيذ:
   - التسلسل: تنفيذ العبارات واحدة تلو الأخرى.
   - الاختيار: اتخاذ القرارات بناءً على الشروط (على سبيل المثال، عبارات if-else).
   - التكرار: تكرار كتلة من التعليمات البرمجية (على سبيل المثال، حلقات for وحلقات while).
4. النهج من أعلى إلى أسفل: تبدأ عملية التصميم من أعلى مستوى من التجريد وتعمل وصولاً إلى التفاصيل. يساعد هذا الأسلوب في تنظيم البرنامج إلى أجزاء أصغر يمكن التحكم فيها.
5. تجنب عبارات GOTO: لا تشجع البرمجة الهيكلية على استخدام عبارات GOTO، مما قد يؤدي إلى كود السباغيتي ويجعل من الصعب متابعة البرنامج وصيانته.

مثال على " التسلسل"

```
def greet_user():
    print("Hello, user!")
    print("Welcome to the program.")
    print("Enjoy your experience!")

greet_user()
```

مثال على " الاختيار"

```
include <iostream>
using namespace std;

int main() {
    int number;
    cout << "Enter a number: ";
    cin >> number;

    if (number % 2 == 0) {
        cout << "The number is even." << endl;
    } else {
        cout << "The number is odd." << endl;
    }
    return 0;
}
```

مثال على " الدوران"

```
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Iteration " + i);
        }
    }
}
```

- **الاختلافات بين البرمجة الهيكلية و البرمجة الكينونية (OOP) من منظور:**

1. التركيز النموذجي :

- البرمجة الهيكلية : تركز على العملية أو الإجراءات التي يقوم بها البرنامج. وتؤكد على وظائف أو إجراءات العمل على البيانات.
- البرمجة الكينونية (OOP): تركز على الكائنات التي تمثل كيانات العالم الحقيقي. وتؤكد على البيانات (الكائنات) والأساليب (الوظائف) التي تعمل على البيانات الموجودة داخل الكائنات.

2. نهج التصميم :

- البرمجة الهيكلية : تستخدم نهج التصميم من أعلى إلى أسفل، وتقسيم البرنامج إلى مجموعة من الوظائف أو الإجراءات.
- OOP: يستخدم أسلوب التصميم من الأسفل إلى الأعلى، مع التركيز على إنشاء الكائنات التي تتفاعل مع بعضها بعضا .

3. البيانات والوظائف :

- البرمجة الهيكلية : الوظائف والبيانات منفصلة. تعمل الوظائف على البيانات التي يتم تمريرها كوسائط.
- OOP: يتم تغليف البيانات والوظائف داخل الكائنات. تتفاعل الكائنات مع بعضها بعضا من خلال الأساليب.

4. النمطية :

- البرمجة الهيكلية : تحقق النمطية من خلال الوظائف والإجراءات.
- OOP: يحقق النمطية من خلال الفئات والكائنات.

5. قابلية إعادة الاستخدام :

- البرمجة الهيكلية : إمكانية إعادة الاستخدام المحدودة. يمكن إعادة استخدام الوظائف ولكنها تفتقر إلى مرونة وراثة OOP وتعدد الأشكال.
- OOP: قابلية إعادة الاستخدام العالية. تسمح الوراثة وتعدد الأشكال بإعادة استخدام الأشياء وتوسيعها بسهولة.

6. التغليف :

- البرمجة الهيكلية : التغليف ليس متأصلاً. غالبًا ما تكون البيانات متاحة عالميًا.
- OOP: التغليف هو مبدأ أساسي. يتم إخفاء البيانات داخل الكائنات ويمكن الوصول إليها من خلال الأساليب، مما يعزز الأمان والنزاهة.

ملخص

يعد فهم مبادئ البرمجة الهيكلية وكيفية اختلافها عن البرمجة الموجهة للكائنات أمرًا ضروريًا للمتدربين لفهم تطور نماذج البرمجة. حيث تركز البرمجة الهيكلية على تحسين وضوح البرنامج وقابلية صيانته من خلال هياكل الوحدات والتحكم، بينما توفر OOP طريقة أكثر طبيعية لنمذجة مشاكل العالم الحقيقي من خلال الكائنات والتغليف والميراث. تزود هذه المعرفة المتدربين بأساس متين في ممارسات تطوير البرمجيات وتعدهم لمفاهيم البرمجة المتقدمة.

- **تمارين عملية**

أ. **تمرين البرمجة الهيكلية** : اكتب برنامجًا يقوم بحساب مضروب رقم باستخدام كل من الطرق التكرارية (بأحد لغات البرمجة الهيكلية).

ب. **تمرين OOP** : اكتب برنامجًا يمثل حسابًا مصرفيًا بسيطًا يحتوي على فئات "Account" و"SavingsAccount" و"CheckingAccount"، موضحًا الوراثة وتعدد الأشكال في لغة OOP (بأحد لغات البرمجة).

- **نظرة عامة على أنماط التصميم (Design Pattern) :**

- **تعريف أنماط التصميم :**

أنماط التصميم هي حلول قياسية للمشاكل الشائعة في تصميم البرمجيات. إنها تمثل أفضل الممارسات التي يمكن لمطوري البرامج استخدامها لحل مشكلات التصميم المتكررة ضمن سياق معين. توفر هذه الأنماط نموذجًا أو مخططًا مثبتًا لمواجهة تحديات محددة، مما يضمن أن الحل يتسم بالكفاءة والفعالية.

- **أهمية أنماط التصميم في تطوير البرمجيات :**

أ. قابلية إعادة الاستخدام: تعمل أنماط التصميم على تعزيز إعادة استخدام التعليمات البرمجية من خلال توفير قالب قياسي لحل المشكلات الشائعة. وهذا يقلل من الوقت والجهد اللازمين لتطوير حلول جديدة ويضمن الاتساق عبر المشاريع.

ب. قابلية الصيانة: من خلال اتباع أنماط التصميم، غالبًا ما يكون الكود الناتج أكثر قابلية للقراءة وأسهل في الصيانة. توفر الأنماط بنية واضحة وتحدد العلاقات بين الفئات والكائنات، مما يسهل على المطورين فهم التعليمات البرمجية وتعديلها.

ج. الكفاءة: يمكن أن يؤدي استخدام أنماط التصميم إلى تطوير برمجيات أكثر كفاءة. توفر الأنماط حلولاً محسنة يمكنها تحسين أداء البرنامج وموثوقيته.

د. قابلية التوسع: تسهل أنماط التصميم تطوير أنظمة قابلة للتطوير. أنها توفر نهجا منظما لتوسيع وتعزيز البرامج دون إدخال أي تعقيد.

هـ. التواصل: تخلق أنماط التصميم مفردات مشتركة للمطورين، مما يسهل مناقشة الحلول ومشاركتها. يؤدي ذلك إلى تحسين التعاون ويضمن أن يكون أعضاء الفريق على نفس الصفحة عند مناقشة استراتيجيات التصميم.

- **مثال عملي**

النمط الفردي (الإبداعي): يضمن أن الفصل يحتوي على مثال واحد فقط ويوفر نقطة وصول عالمية إليه.

Example in Java:

```
public class Singleton {
    private static Singleton instance;

    private Singleton() {
        // private constructor to prevent instantiation
    }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

انظر إلى مدى سهولة فهم البرامج الموثقة جيدًا وصيانتها. هل تساءلت يوما لماذا؟
اكتشف أهمية التوثيق الواضح الذي يتبع إجراءات الشركة وأفضل ممارسات البرمجة الهيكلية .

3. توثيق البرنامج بالتعليمات وفقا لإجراءات الشركة و أفضل تعليمات البرمجة الهيكلية

- **دور التوثيق وأهمية في تطوير البرمجيات :**

- **التوثيق** (Documentation): يشير التوثيق في تطوير البرمجيات إلى النص المكتوب أو الرسوم التوضيحية المصاحبة لبرامج الكمبيوتر لشرح كيفية عملها أو كيفية استخدامها.
- **الأهمية** : يعد التوثيق جزءًا مهمًا من عملية تطوير البرمجيات. فهو يساعد على ضمان أن جميع أصحاب المصلحة لديهم فهم واضح للبرنامج ووظائفه وكيفية استخدامه أو صيانته.

- **دور التوثيق :**

1. التواصل:

- يسهل التواصل بين أعضاء الفريق وأصحاب المصلحة والمستخدمين من خلال توفير معلومات واضحة ومنظمة حول البرنامج.
- يساعد على سد الفجوة بين أصحاب المصلحة التقنيين وغير التقنيين من خلال شرح المفاهيم المعقدة بطريقة مفهومة.

2. قابلية الصيانة:

- يوفر مرجعًا للمطورين لفهم قاعدة التعليمات البرمجية، مما يسهل صيانة البرنامج وتصحيح أخطائه وتحسينه.
- يساعد أعضاء الفريق الجدد على العمل بسرعة من خلال تزويدهم بمعلومات مفصلة حول بنية البرنامج ومكوناته ووظائفه.

3. الاتساق:

- يضمن الاتساق في عملية التطوير من خلال توثيق معايير الترميز وقرارات التصميم وأفضل الممارسات.
- يساعد في الحفاظ على التوحيد في طريقة كتابة التعليمات البرمجية والحفاظ عليها عبر الفريق.

4. الامتثال وضمان الجودة:

- يدعم الامتثال لمعايير الصناعة والمتطلبات التنظيمية من خلال توثيق جميع المعلومات الضرورية.
- يساعد في عمليات ضمان الجودة من خلال توفير حالات اختبار مفصلة وإجراءات الاختبار والنتائج.

5. دعم المستخدم والتدريب:

- يزود المستخدمين النهائيين بتعليمات حول كيفية استخدام البرنامج، مما يعزز تجربة المستخدم ورضاه.
- يساعد في تدريب المستخدمين وموظفي الدعم من خلال تقديم أدلة وأدلة مستخدم مفصلة.

- **أنواع التوثيق :**

1. تعليقات الكود (التعليمات البرمجية) :

- **التعريف** : تفسيرات مضمنة داخل الكود المصدري تصف ما يفعله الكود ولماذا تم اتخاذ قرارات معينة.
- **الغرض** : مساعدة المطورين على فهم منطق التعليمات البرمجية وتدفقها، مما يسهل تصحيح الأخطاء وصيانتها.
- **أفضل الممارسات:**
  - أ. اكتب تعليقات واضحة وموجزة.
  - ب. تجنب التعليقات المتكررة التي تعيد صياغة الكود فقط.
  - ج. استخدم التعليقات لشرح المنطق المعقد والخوارزميات والغرض من الكود.

مثال (حساب مساحة الدائرة**):**

python

```
  def calculate_area(radius):
       Check if the radius is a positive number
      if radius <= 0:
          return "Invalid radius"
       Calculate the area of the circle
      area = 3.14  radius  2
      return area
```

2. وثائق واجهة برمجة التطبيقات (**API**) :

- التعريف: توثيق تفصيلي لواجهة برمجة التطبيقات (API)، بما في ذلك نقاط النهاية والأساليب والمعلمات والاستجابات.
- الغرض: تزويد المطورين بالمعلومات اللازمة للتكامل مع واجهة برمجة التطبيقات (API) واستخدامها بشكل فعال.
- عناصر:
  - أ. أوصاف نقطة النهاية: تفاصيل كل نقطة نهاية لواجهة برمجة التطبيقات والغرض منها.
  - ب. طرق الطلب: معلومات عن طرق HTTP (GET، POST، PUT، DELETE) التي تستخدمها كل نقطة نهاية.
  - ج. المعلمات: وصف المعلمات المطلوبة والاختيارية لكل طلب.
  - د. تنسيقات الاستجابة: تفاصيل الاستجابات المتوقعة، بما في ذلك رموز الحالة وهياكل البيانات.
  - هـ. أمثلة: نماذج الطلبات والاستجابات لتوضيح كيفية استخدام واجهة برمجة التطبيقات.

مثال (استخدام طريقة GET):

```
Get User Details
  Endpoint: /api/users/{id}
  Method: GET
  Description: Retrieve the details of a user by their ID.

  Parameters:
id (required): The ID of the user.

  Response:
200 OK:
  json
    {
      "id": 1,
      "name": "John Doe",
      "email": "john.doe@example.com"
    }
```

مثال على الخطأ (404 غير موجود)

```
  json
    {
      "error": "User not found"
    }
```

Example Request:

```
shell
  curl -X GET "https://api.example.com/api/users/1"
```

3. **أدلة المستخدم :**

- التعريف: أدلة شاملة تزود المستخدمين النهائيين بتعليمات حول كيفية استخدام البرنامج.
- الغرض: مساعدة المستخدمين على فهم ميزات البرنامج، والتنقل في الواجهة، وتنفيذ المهام المختلفة.
- عناصر:
  أ. المقدمة: نظرة عامة على البرنامج والغرض منه.
  ب. تعليمات التثبيت: خطوات تثبيت البرنامج وإعداده.
  ج. دليل واجهة المستخدم: وصف مكونات واجهة المستخدم ووظائفها.
  د. أوصاف الميزات: معلومات تفصيلية عن كل ميزة وكيفية استخدامها.
  هـ. استكشاف الأخطاء وإصلاحها: المشكلات الشائعة وحلولها.
  و. الأسئلة الشائعة: الأسئلة والأجوبة المتداولة.

مثال:

```
   User Manual for ABC Software

   Introduction
Welcome to ABC Software! This user manual will guide you through the features and
functionalities of the software, helping you to make the most of it.

   Installation Instructions
  1. Download the installer from our website.
  2. Double-click the installer and follow the on-screen instructions.
  3. Once installed, launch the software from the Start menu or desktop shortcut.

   User Interface Guide
Main Menu: Provides access to the software's main features.
Toolbar: Contains shortcuts for frequently used actions.
Status Bar: Displays information about the current status of the software.

   Feature Descriptions
 Creating a New Project
  1. Click on File > New Project.
  2. Enter the project name and select the desired settings.
  3. Click Create to initialize the new project.

   Troubleshooting
Issue: The software crashes on startup.
Solution: Ensure your system meets the minimum requirements and try reinstalling
the software.

   FAQs
Question: How do I reset my password?
Answer: Click on Forgot Password on the login screen and follow the instructions.
```

- **ملخص:**

يلعب التوثيق دورًا حيويًا في تطوير البرمجيات من خلال تعزيز الاتصال وقابلية الصيانة والاتساق والامتثال ودعم المستخدم. يعد فهم الأنواع المختلفة من الوثائق، بما في ذلك تعليقات التعليمات البرمجية ووثائق واجهة برمجة التطبيقات (API) وأدلة المستخدم، أمرًا ضروريًا لإنشاء وثائق شاملة وفعالة تلبي احتياجات جميع أصحاب المصلحة. ومن خلال اتباع أفضل الممارسات وتوفير معلومات واضحة ومفصلة، يمكن للمطورين التأكد من أن برامجهم موثقة جيدًا وسهلة الاستخدام والصيانة.

تمارين عملية

- تمرين رقم 1 : تعليقات التعليمات البرمجية: خذ جزءًا من التعليمات البرمجية وأضف تعليقات ذات معنى لشرح المنطق والتدفق.
- تمرين رقم 2 : توثيق واجهة برمجة التطبيقات: قم بتوثيق نقطة نهاية واجهة برمجة التطبيقات البسيطة، بما في ذلك الطريقة والمعلمات وتنسيق الاستجابة وطلب المثال.
- تمرين رقم 3 : على دليل المستخدم: اكتب مقتطفًا من دليل المستخدم لتطبيق برمجي افتراضي، يغطي المقدمة وتعليمات التثبيت ووصف الميزة.

- **أفضل الممارسات لكتابة تعليقات التعليمات البرمجية (الكود):**

- **كن واضحًا وموجزًا:** اكتب تعليقات يسهل فهمها وفي صلب الموضوع. و تجنب التفاصيل غير الضرورية التي يمكن أن تؤدي إلى فوضى التعليمات البرمجية.
- **اشرح السبب، وليس ماذا:** ركز على شرح سبب وجود جزء معين من الكود أو سبب اتخاذ قرارات معينة، بدلاً من ما تفعله الكود (الذي يجب أن يكون واضحًا من الكود نفسه).
- **قم بتحديث التعليقات بانتظام:** تأكد من تحديث التعليقات لتعكس أي تغييرات في الكود. التعليقات القديمة يمكن أن تكون مضللة.
- **استخدم التعليقات باعتدال:** قم بالتعليق فقط على الأجزاء المعقدة أو غير الواضحة من الكود. الإفراط في التعليق يمكن أن يجعل قراءة الكود أكثر صعوبة.
- **استخدم القواعد النحوية والإملائية الصحيحة:** اكتب التعليقات في جمل كاملة مع القواعد النحوية والإملائية الصحيحة للحفاظ على الاحترافية والوضوح.

- **الفرق بين تعليقات التعليمات البرمجية الجيدة و السيئة :**
  - **تعليقات الترميز الجيدة**

1. توضيح الغرض والمنطق: شرح سبب القيام بشيء ما، وليس ما تم القيام به. ويجب أن توفر التعليقات الجيدة سياقًا واضح.

مثال:

```
//Using binary search because the list is sorted and large, for efficiency
```

2. شرح المنطق أو الخوارزميات المعقدة: تلخيص الخوارزميات ، وخاصة إذا كانت معقدة أو غير تقليدية.

مثال:

```
//This function implements the quicksort algorithm to sort the list in ascending order
```

3. توثيق الافتراضات أو القيود أو التحذيرات: لاحظ أي افتراضات أو قيود أو تحذيرات للمطورين في المستقبل.

مثال:

```
//Assumes input is always a non-negative integer
```

4. الحفاظ على الصلة والدقة: تظل التعليقات الجيدة محدثة بتغييرات الكود، مما يعكس بدقة ما يفعله الكود.

- **تعليقات الترميز السيئة**

1. بيان التوضيح: التعليقات التي تكرر فقط ما يشير إليه الكود

مثال:

```
int x = 10; // set x to 10
```

2. شرح الكود البسيط أو الذي لا يحتاج إلى شرح: يمكن أن تجعل التعليقات غير الضرورية على العمليات الأساسية .

مثال:

```
sum += item; // add item to sum
```

3. تقديم معلومات قديمة أو مضللة: عندما لا يتم تحديث التعليقات لتعكس التغييرات في الكود، فقد تؤدي إلى التضليل والارتباك.

مثال:

```
//Process customer orders (if the code now processes both orders and returns)
```

4. إضافة ملاحظات شخصية: التعليقات التي تتضمن ملاحظات شخصية أو ملاحظات غير ذات صلة وغير احترافية وتشتت الانتباه عن الغرض الكود.

مثال:

```
//I hate this part of the code, but it works
```

انظر إلى مدى سهولة ومتعة تطبيقاتك المفضلة. هل تساءلت يومًا كيف حققوا ذلك؟
اكتشف كيف يؤدي استخدام مبادئ تصميم واجهة المستخدم/تجربة المستخدم إلى إنشاء تجارب جذابة وسهلة الاستخدام.

4. مبادئ وإرشادات تصميم واجهة المستخدم/تجربة المستخدم (UI/UX) :

- **تعريفات :**

- **واجهة المستخدم (UI):** تشير إلى العناصر المرئية والمكونات التفاعلية للتطبيقات البرمجية التي يتفاعل معها المستخدمون ، بما في ذلك الأزرار والقوائم والنماذج والعناصر المرئية الأخرى.

- **تجربة المستخدم (UX):** تشمل جميع جوانب تفاعل المستخدم مع أحد التطبيقات البرمجية، مع التركيز على التجربة الشاملة والرضا.

- **أهمية واجهة المستخدم/تجربة المستخدم في تطوير البرمجيات**

أ. **يعزز رضا المستخدم:**

- يخلق التصميم الجيد لواجهة المستخدم/تجربة المستخدم تجربة بديهية وممتعة للمستخدمين، مما يزيد من رضاهم وتفاعلهم مع البرنامج.

- من المرجح أن يعود المستخدمون إلى البرامج سهلة الاستخدام وتلبي احتياجاتهم والتوصية بها.

ب. **يحسن سهولة الاستخدام:**

- تصميم UI/UX الفعال يجعل البرامج سهلة التنقل والاستخدام، مما يقلل من منحنى التعلم للمستخدمين الجدد.

- يساعد تصميم واجهة المستخدم الواضح والمنطقي المستخدمين على إكمال المهام بكفاءة وفعالية.

ج. **يعزز إمكانية الوصول:**

- تضمن مبادئ تصميم واجهة المستخدم/تجربة المستخدم إمكانية الوصول إلى البرامج لمجموعة واسعة من المستخدمين، بما في ذلك الأشخاص ذوي الإعاقة.

- يؤدي دمج ميزات إمكانية الوصول مثل قارئات الشاشة والتنقل باستخدام لوحة المفاتيح وأوضاع التباين العالي إلى تحسين شمولية البرنامج.

د. **يزيد من الاحتفاظ بالمستخدمين:**

- تجربة المستخدم الإيجابية تشجع المستخدمين على الاستمرار في استخدام البرنامج واستكشاف ميزاته.
- غالبًا ما يكون الاحتفاظ بالمستخدمين أكثر فعالية من حيث التكلفة من الحصول على مستخدمين جدد، مما يجعل التصميم الجيد لواجهة المستخدم/تجربة المستخدم استثمارًا قيمًا.

هـ. **يدعم صورة العلامة التجارية:**

- تنعكس الواجهات وتجارب المستخدم المصممة جيدًا بشكل إيجابي على العلامة التجارية، مما يعزز سمعتها ومصداقيتها.
- يعمل تصميم UI/UX المتسق عبر المنتجات على تعزيز هوية العلامة التجارية والاعتراف بها.

و. **يقلل من تكاليف التطوير:**

- يمكن أن يؤدي الاستثمار في تصميم واجهة المستخدم/تجربة المستخدم في وقت مبكر من عملية التطوير إلى تحديد مشكلات قابلية الاستخدام ومعالجتها قبل أن يصبح إصلاحها مكلفًا.
- يمكن لنهج التصميم الذي يركز على المستخدم أن يبسط التطوير من خلال التركيز على الميزات التي تلبي احتياجات المستخدم حقًا.

مثال عملي :

دراسة حالة: تطبيق جوال لتقديم خدمة مصرفية باستخدام أي لغة برمجه متاحه لديك.

- تصميم جيد لواجهة المستخدم/تجربة المستخدم: يمكن للمستخدمين التنقل بسهولة للتحقق من رصيد حساباتهم، وتحويل الأموال، ودفع الفواتير. يوفر التطبيق تعليقات واضحة حول إجراءات المستخدم، ويتم تحميله بسرعة، ويستخدم خطوطًا وألوانًا يمكن الوصول إليها.
- تصميم ضعيف لواجهة المستخدم/تجربة المستخدم: يواجه المستخدمون صعوبة في العثور على الميزات الأساسية، ويواجهون رسائل خطأ غير واضحة، ويواجهون أوقات تحميل بطيئة، مما يؤدي إلى الإحباط والتخلي المحتمل عن التطبيق.

- **المبادئ الأساسية لتصميم واجهة المستخدم/تجربة المستخدم**

1. **الاتساق:**

- تعمل عناصر التصميم المتسقة، مثل الألوان والخطوط وأنماط الأزرار، على إنشاء تجربة مستخدم متماسكة ويمكن التنبؤ بها.
- يساعد الاتساق في التنقل والتخطيط المستخدمين على بناء الألفة والثقة في استخدام البرنامج.

2. **البساطة:**

- تصميمات بسيطة ومرتبة تسهل على المستخدمين التركيز على المهام والمعلومات الأساسية.
- تجنب الميزات والعناصر غير الضرورية يقلل من الحمل المعرفي ويعزز سهولة الاستخدام.

3. **ردود الفعل:**

- تقديم تعليقات فورية وواضحة حول تصرفات المستخدم يساعد المستخدمين على فهم نتائج تفاعلاتهم.
- تشمل الأمثلة الإشارات المرئية ورسائل الحالة والإشعارات الصوتية.

4. **إمكانية الوصول:**

- يضمن التصميم الملائم لإمكانية الوصول أن جميع المستخدمين، بما في ذلك الأشخاص ذوي الإعاقة، يمكنهم استخدام البرنامج بفعالية.
- تشمل الاعتبارات الرئيسية التنقل باستخدام لوحة المفاتيح، ودعم قارئ الشاشة، وتباين الألوان.

5. **الرؤية:**

- يجب أن تكون المعلومات والضوابط الهامة مرئية ويمكن الوصول إليها بسهولة دون إرباك المستخدم.
- استخدم التسلسل الهرمي المرئي لتوجيه انتباه المستخدمين إلى العناصر الأكثر أهمية أولاً.

6. **التحكم في المستخدم:**

- تمكين المستخدمين من خلال توفير الخيارات والسماح لهم بالتراجع عن الإجراءات أو التعافي من الأخطاء.
- تتضمن الأمثلة أزرار "الإلغاء"، ووظائف التراجع، والتعليمات الواضحة لاستعادة الأخطاء.

7. **المرونة والكفاءة:**

- التصميم لكل من المستخدمين المبتدئين وذوي الخبرة من خلال توفير الاختصارات والميزات المتقدمة للمستخدمين المتميزين.
- السماح للمستخدمين بتخصيص تجربتهم وتكييف الواجهة حسب تفضيلاتهم.

8. **الجماليات:**

- تصميم جميل يعزز تجربة المستخدم الشاملة ويمكن أن يجعل استخدام البرنامج أكثر متعة.
- تحقيق التوازن بين المظهر المرئي والوظيفة، مما يضمن أن خيارات التصميم تدعم سهولة الاستخدام.

- **أمثلة عملية لتطبيق مبادئ واجهة المستخدم/تجربة المستخدم:**

الاتساق:

- استخدم دليل الأسلوب للتأكد من أن جميع الشاشات والمكونات تتبع نفس لغة التصميم.
- مثال: جميع الأزرار لها نفس الشكل واللون والسلوك عبر التطبيق.

البساطة:

- إزالة العناصر غير الضرورية والتركيز على الوظائف الأساسية.
- مثال: شاشة تسجيل دخول تحتوي على الحقول الأساسية فقط (اسم المستخدم وكلمة المرور) وزر واضح للحث على اتخاذ إجراء.

ردود الفعل:

- تقديم تعليقات في الوقت الحقيقي حول إجراءات المستخدم.
- مثال: نموذج يظهر رسالة نجاح عند تقديمه بشكل صحيح أو يبرز أخطاء الحقول غير الصحيحة.

إمكانية الوصول:

- تنفيذ الميزات التي تدعم المستخدمين ذوي الإعاقة.
- مثال: تأكد من أن جميع الصور تحتوي على نص بديل وصفي ويمكن التنقل بين العناصر التفاعلية باستخدام لوحة المفاتيح.

الرؤية:

- تصميم تسلسل هرمي مرئي واضح للتأكيد على المعلومات المهمة.
- مثال: استخدم نصًا أكبر حجمًا وأكثر جرأة للعناوين والإجراءات الأساسية، بينما تكون المعلومات الثانوية أصغر حجمًا وأقل بروزًا.

التحكم في المستخدم:

- تزويد المستخدمين بخيارات لتصحيح الأخطاء وتحديد الاختيارات.
- مثال: السماح للمستخدمين بالتراجع عن حذف عنصر من القائمة.

المرونة والكفاءة:

- اختصارات التصميم والميزات المتقدمة للمستخدمين ذوي الخبرة.
- مثال: اختصارات لوحة المفاتيح للإجراءات الشائعة، مثل Ctrl+C للنسخ و Ctrl+V للصق.

الجماليات:

- إنشاء تصميم جذاب بصريًا يتماشى مع هوية العلامة التجارية.
- مثال: استخدم لوحة ألوان متسقة وطباعة واضحة وتخطيطات متوازنة لتحسين الشكل والمظهر العام.

ملخص

يعد فهم أهمية واجهة المستخدم/تجربة المستخدم في تطوير البرمجيات وإتقان المبادئ الأساسية لتصميم واجهة المستخدم/تجربة المستخدم أمرًا ضروريًا لإنشاء تطبيقات برمجية ناجحة. يعمل التصميم الجيد لواجهة المستخدم/تجربة المستخدم على تعزيز رضا المستخدم وتحسين سهولة الاستخدام وتعزيز إمكانية الوصول ودعم صورة العلامة التجارية. من خلال اتباع المبادئ الأساسية مثل الاتساق والبساطة والتغذية الراجعة وإمكانية الوصول والرؤية والتحكم في المستخدم والمرونة والجماليات، يمكن للمطورين إنشاء تجارب مستخدم بديهية وممتعة.

- **تمارين عملية**

أ. تمرين تحليل واجهة المستخدم/تجربة المستخدم: تقييم تصميم واجهة المستخدم/تجربة المستخدم لتطبيق برمجي شائع. تحديد نقاط القوة ومجالات التحسين بناءً على المبادئ الأساسية لتصميم واجهة المستخدم/تجربة المستخدم.

ب. تمرين إعادة التصميم: اختر واجهة سيئة التصميم وأنشئ إعادة تصميم تعالج مشكلات قابلية الاستخدام وتطبق أفضل ممارسات واجهة المستخدم/تجربة المستخدم.

ج. تمرين النماذج الأولية: استخدم أداة النماذج الأولية (مثل Figma وSketch وAdobe XD) لإنشاء نموذج أولي عالي الدقة لتطبيق برمجي جديد، يتضمن مبادئ واجهة المستخدم/UX الأساسية.

د. تمرين اختبار المستخدم: قم بإجراء اختبار المستخدم على تطبيق برمجي لجمع التعليقات حول تصميم واجهة المستخدم/تجربة المستخدم الخاصة به. تحليل النتائج واقتراح التحسينات بناءً على تعليقات المستخدمين ومبادئ التصميم.

ستساعد هذه التمارين المتدربين على تطبيق المعرفة النظرية لتصميم واجهة المستخدم/تجربة المستخدم في سيناريوهات عملية، مما يعزز قدرتهم على إنشاء تطبيقات برمجية سهلة الاستخدام وفعالة.

- **إرشادات تصميم واجهة المستخدم/تجربة المستخدم :**

أ. **التناسق :**

التعريف : الاتساق في تصميم واجهة المستخدم/تجربة المستخدم يعني ضمان أن العناصر المتشابهة تتصرف بطرق مماثلة عبر التطبيق. فهو يساعد المستخدمين على تطوير الشعور بالألفة والقدرة على التنبؤ، مما يعزز تجربتهم الشاملة.

القواعد الارشادية:

- الاتساق البصري: استخدم نظام ألوان وطباعة وأنماط أزرار وتخطيطًا متسقًا في جميع أنحاء التطبيق.
- الاتساق الوظيفي: تأكد من أن العناصر التفاعلية مثل الأزرار والروابط والنماذج تتصرف بنفس الطريقة عبر الشاشات المختلفة.
- الاتساق الداخلي: الحفاظ على التوحيد داخل التطبيق.
- الاتساق الخارجي: التوافق مع الأنماط والاصطلاحات المألوفة من التطبيقات والأنظمة الأساسية الأخرى.

مثال : جميع أزرار الإجراءات الأساسية في التطبيق باللون الأزرق وتقع في الركن الأيمن السفلي من كل شاشة.

ب. **التعليقات :**

التعريف : تشير التعليقات في تصميم واجهة المستخدم/تجربة المستخدم إلى تزويد المستخدمين باستجابات فورية وواضحة لأفعالهم. وهذا يساعد المستخدمين على فهم نتائج تفاعلاتهم وما إذا كانت الإجراءات ناجحة أم لا.

القواعد الارشادية:

- ردود الفعل المرئية: استخدم الرسوم المتحركة وتغييرات الألوان ومؤشرات الحالة لتظهر للمستخدمين أنه تم التعرف على أفعالهم.
- التعليقات السمعية: قم بتنفيذ المؤثرات الصوتية لإجراءات مثل الأخطاء والتأكيدات والإشعارات.
- التعليقات النصية: تقديم رسائل لإجراءات مثل عمليات إرسال النماذج والأخطاء وإشعارات النجاح.

مثال : عندما يقوم المستخدم بإرسال نموذج، يتم عرض رسالة نجاح، ويتغير لون زر الإرسال للإشارة إلى نجاح الإجراء.

ج. **البساطة :**

التعريف : البساطة في تصميم واجهة المستخدم/تجربة المستخدم تعني إنشاء واجهات واضحة وسهلة الاستخدام. يتضمن ذلك تقليل العناصر غير الضرورية والتركيز على الميزات والوظائف الأساسية.

القواعد الارشادية:

- التصميم البسيط: قم بإزالة أية عناصر غير ضرورية لا تساهم في تحقيق الأهداف الأساسية للمستخدم.
- مسح التسلسل الهرمي: استخدم التسلسل الهرمي المرئي لتوجيه انتباه المستخدمين إلى العناصر الأكثر أهمية.
- التنقل البديهي: صمم بنية تنقل بسيطة وبديهية يمكن للمستخدمين فهمها ومتابعتها بسهولة.

مثال : واجهة لوحة المعلومات التي تعرض فقط المعلومات الأكثر أهمية وتوفر تنقلًا واضحًا للتفاصيل الإضافية.

د. إمكانية الوصول

التعريف : تضمن إمكانية الوصول في تصميم واجهة المستخدم/تجربة المستخدم أن جميع المستخدمين، بما في ذلك الأشخاص ذوي الإعاقة، يمكنهم استخدام التطبيق بفعالية. يتضمن ذلك التصميم مع وضع مجموعة واسعة من القدرات والسياقات في الاعتبار.

القواعد الارشادية:

- التنقل عبر لوحة المفاتيح: تأكد من إمكانية الوصول إلى جميع العناصر التفاعلية والتحكم فيها باستخدام لوحة المفاتيح.
- قارئات الشاشة: توفير بدائل نصية للمحتوى غير النصي، مثل الصور والأيقونات، لدعم قارئات الشاشة.
- تباين الألوان: استخدم أنظمة ألوان عالية التباين للتأكد من أن النص والعناصر المهمة مرئية للمستخدمين ذوي الإعاقات البصرية.

مثال : موقع ويب يتضمن نصًا بديلًا لجميع الصور ونصًا عالي التباين واختصارات لوحة المفاتيح للتنقل.

هـ. سهولة الاستخدام

التعريف : تشير سهولة الاستخدام في تصميم واجهة المستخدم/تجربة المستخدم إلى السهولة التي يمكن للمستخدمين من خلالها تحقيق أهدافهم باستخدام التطبيق. مع التركيز على خلق تفاعلات بديهية وفعالة.

القواعد الارشادية :

- التصميم الذي يركز على المستخدم: إشراك المستخدمين في عملية التصميم من خلال اختبار قابلية الاستخدام وجلسات التعليقات.
- منع الأخطاء واستردادها: تصميم واجهات تمنع الأخطاء وتوفر تعليمات واضحة لاستعادة الأخطاء.
- الكفاءة: تحسين الواجهة لتمكين المستخدمين من إكمال المهام بسرعة وكفاءة.

مثال : موقع ويب للتجارة الإلكترونية يعمل على تبسيط عملية الدفع عن طريق تقليل عدد الخطوات وتوفير تعليمات واضحة في كل خطوة.

- **تمارين عملية لمبادئ وإرشادات تصميم الواجهة/تجربة المستخدم (UI/UX).**

- **التمرين 1: مراجعة ونقد تصميمات واجهة المستخدم/تجربة المستخدم الحالية**

تعليمات:

1. حدد تطبيقًا: اختر تطبيقًا برمجيًا أو موقعًا إلكترونيًا شائعًا. تشمل الأمثلة منصات الوسائط الاجتماعية أو مواقع التجارة الإلكترونية أو أدوات الإنتاجية.

2. حدد المجالات الرئيسية للمراجعة: ركز على المجالات الرئيسية مثل التنقل والتخطيط والتصميم المرئي والتفاعل وإمكانية الوصول.

3. التقييم بناءً على مبادئ واجهة المستخدم/تجربة المستخدم: استخدم المعايير التالية لتقييم تصميم واجهة المستخدم/تجربة المستخدم:

- الاتساق: التحقق من التوحيد في عناصر التصميم والتفاعلات.
- التعليقات: تقييم كيفية استجابة التطبيق لإجراءات المستخدم.
- البساطة: تقييم سهولة الاستخدام وما إذا كان التصميم خاليًا من الفوضى.
- إمكانية الوصول: ضع في اعتبارك مدى إمكانية الوصول إلى التطبيق للمستخدمين ذوي الإعاقة.
- سهولة الاستخدام: تحديد مدى فعالية المستخدمين في تحقيق أهدافهم.

4. قم بتوثيق النتائج التي توصلت إليها:

- اكتب نقدًا تفصيليًا يسلط الضوء على نقاط القوة والضعف في التصميم.
- استخدم لقطات الشاشة لتوضيح النقاط وتقديم أمثلة مرئية.

مثال للتقييم: التطبيق: Instagram (تطبيق الهاتف المحمول)

تناسق:

- القوة: الاستخدام المتسق للأيقونات والألوان في جميع أنحاء التطبيق.
- الضعف: بعض التناقضات في وضع الأزرار بين الشاشات المختلفة.

تعليق:

- القوة: ردود فعل فورية عند الإعجاب بمنشور (رسوم متحركة).
- الضعف: عدم وجود تعليقات عند تحميل الصورة حتى يتم تحميلها بالكامل.

بساطة:

- القوة: تصميم نظيف وبسيط مع التركيز على المحتوى.
- الضعف: بعض الميزات مخفية في القوائم، مما يجعلها أقل قابلية للاكتشاف.

إمكانية الوصول:

- القوة: نص عالي التباين وأهداف لمس كبيرة.
- الضعف: دعم محدود لقارئات الشاشة وعدم وجود بدائل نصية للصور.

سهولة الاستخدام:

- القوة: التنقل البديهي مع الميزات الرئيسية التي يمكن الوصول إليها بسهولة.
- الضعف: عملية تحرير معلومات الملف الشخصي ليست واضحة.

- **التمرين 2: اقتراح تحسينات في واجهة المستخدم/تجربة المستخدم**

تعليمات:

- **حدد التطبيق الذي تمت مراجعته:** استخدم نفس التطبيق أو موقع الويب من التمرين السابق.
- **تحديد المشكلات الرئيسية:** بناءً على انتقاداتك، حدد أهم المشكلات التي تحتاج إلى تحسين.

اقتراح الحلول:

- لكل مشكلة، اقترح حلولاً قابلة للتنفيذ بناءً على مبادئ واجهة المستخدم/تجربة المستخدم.
- قم بتضمين إطارات سلكية أو نماذج بالحجم الطبيعي لتصور التغييرات المقترحة.

برر توصياتك: اشرح كيف يعالج كل تحسين مقترح المشكلة المحددة ويعزز تجربة المستخدم.

مثال على الاقتراح: التطبيق: Instagram (تطبيق الهاتف المحمول)

المشكلة 1: التناقضات في وضع الزر

- التحسين المقترح: توحيد موضع أزرار الإجراءات (على سبيل المثال، ضع دائمًا الزر "متابعة" في الزاوية العلوية اليمنى من ملفات تعريف المستخدمين).
- التبرير: سيؤدي هذا التغيير إلى تحسين الاتساق، مما يجعل الواجهة أكثر قابلية للتنبؤ بها وأسهل في الاستخدام.

المشكلة 2: عدم وجود تعليقات عند تحميل الصور

- التحسين المقترح: تنفيذ مؤشر التقدم أو تحميل الرسوم المتحركة أثناء تحميل الصورة.
- المبررات: إن تقديم تعليقات مرئية سيُعلم المستخدمين بأن إجراءاتهم قيد المعالجة، مما يقلل من عدم اليقين ويحسن التجربة الإجمالية.

المشكلة 3: الميزات المخفية في القوائم

- التحسين المقترح: إعادة تنظيم بنية القائمة لإظهار الميزات شائعة الاستخدام وتقليل عدد القوائم المتداخلة.
- المبرر: سيؤدي ذلك إلى جعل الميزات أكثر قابلية للاكتشاف، مما يعزز سهولة الاستخدام والكفاءة للمستخدمين.

المشكلة 4: الدعم المحدود لقارئات الشاشة

- التحسين المقترح: أضف نصًا بديلًا وصفيًا إلى جميع الصور وتأكد من إمكانية الوصول إلى جميع العناصر التفاعلية عبر برامج قراءة الشاشة.
- المبررات: تحسين إمكانية الوصول سيجعل التطبيق قابلاً للاستخدام لمجموعة واسعة من المستخدمين، بما في ذلك الأشخاص الذين يعانون من إعاقات بصرية.

المشكلة 5: عملية معقدة لتحرير معلومات الملف الشخصي

- التحسين المقترح: تبسيط عملية تحرير الملف الشخصي من خلال توفير الوصول المباشر إلى خيارات التحرير من صفحة الملف الشخصي.
- المبررات: سيسهل هذا على المستخدمين تحديث معلومات ملفاتهم الشخصية، مما يحسن سهولة الاستخدام بشكل عام.

ملخص

تعد مراجعة وانتقاد تصميمات واجهة المستخدم/تجربة المستخدم الحالية واقتراح التحسينات بناءً على مبادئ واجهة المستخدم/تجربة المستخدم مهارات حاسمة لتطوير تطبيقات برمجية سهلة الاستخدام. من خلال تقييم التصاميم باستخدام معايير مثل الاتساق، والتغذية الراجعة، والبساطة، وإمكانية الوصول، وسهولة الاستخدام، يمكن للمتدربين تحديد مجالات التحسين. يساعد اقتراح تحسينات قابلة للتنفيذ وتبريرها بناءً على مبادئ واجهة المستخدم/تجربة المستخدم على ضمان أن تكون الواجهات بديهية وسهلة الوصول وفعالة في تلبية احتياجات المستخدم.

- **تمارين عملية :**

أ. مراجعة التطبيق: حدد تطبيقًا مختلفًا عن التطبيق الذي تمت مراجعته مسبقًا، وقم بتقييم تصميم واجهة المستخدم/تجربة المستخدم الخاصة به، وقم بتوثيق النتائج التي توصلت إليها.

ب. اقتراح التحسينات: بناءً على مراجعتك، اقترح مجموعة من التحسينات لتحسين تجربة المستخدم، بما في ذلك الإطارات السلكية أو النماذج بالحجم الطبيعي لتصور اقتراحاتك.

ج. جلسة نقدية جماعية: قم بإجراء جلسة جماعية حيث يقدم كل متدرب مراجعته والتحسينات المقترحة. مناقشة وتقديم ردود الفعل على عمل بعضهم البعض.

د. تمرين التنفيذ: اختر أحد التحسينات المقترحة وقم بتنفيذه في نموذج أولي باستخدام أداة مثل Figma أو Sketch أو Adobe XD.

ستساعد هذه التمارين المتدربين على تطبيق مبادئ تصميم واجهة المستخدم/تجربة المستخدم في سيناريوهات عملية، مما يعزز قدرتهم على إنشاء تطبيقات برمجية فعالة وسهلة الاستخدام.

- انظر إلى كيفية عمل التطبيقات الحديثة بسلاسة عبر الأجهزة المختلفة. هل تساءلت يومًا كيف؟
- اكتشف التقنيات والأطر الأمامية التي توفر تصميمات UI/UX فعالة لمنصات متعددة.

5. استخدام تقنيات وأطر الواجهة الأمامية Front End لتنفيذ تصميمات واجهة المستخدم/تجربة المستخدم (UI/UX) بشكل فعال والاستجابة لبيئة عمل أكثر من منصة:

- **مقدمة**

- يتضمن تطوير الواجهة الأمامية إنشاء واجهة المستخدم وتجربة المستخدم لتطبيق الويب. التقنيات الأساسية المستخدمة لتطوير الواجهة الأمامية هي HTML وCSS وJavaScript.

- **HTML (لغة ترميز النص التشعبي)**

- التعريف: HTML هي لغة الترميز القياسية المستخدمة لإنشاء بنية ومحتوى صفحات الويب.
- الدور: يحدد HTML تخطيط صفحة الويب باستخدام عناصر مثل العناوين والفقرات والقوائم والروابط والصور والنماذج.

- المفاهيم الرئيسية:

- العناصر: الوحدات الأساسية لـ HTML، ممثلة بالعلامات (على سبيل المثال، <a>، <p>، <h1>).
- السمات: قم بتوفير معلومات إضافية حول العناصر (على سبيل المثال، href للروابط، src للصور).
- بنية الوثيقة: تحتوي مستندات HTML على بنية هرمية مع عنصر <html> جذر، يحتوي على قسمين <head> و<body>.

مثال :

html

```
<!DOCTYPE html>
<html>
<head>
    <title>My First Web Page</title>
</head>
<body>
    <h1>Welcome to My Website</h1>
    <p>This is a paragraph of text on my web page.</p>
    <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

- **CSS (أوراق الأنماط المتتالية)**

- التعريف: CSS هي لغة ورقة أنماط تستخدم لوصف العرض التقديمي وتخطيط صفحة الويب.
- الدور: يتحكم CSS في المظهر المرئي لعناصر HTML، بما في ذلك الألوان والخطوط والمسافات والموضع.

- المفاهيم الرئيسية:

- المحددات: استهدف عناصر HTML لتطبيق الأنماط (على سبيل المثال، محددات العناصر، محددات الفئة، محددات المعرف).
- الخصائص والقيم: تحديد الأنماط التي سيتم تطبيقها (على سبيل المثال، اللون، حجم الخط، الهامش).
- التتالي والوراثة: يحدد ترتيب وخصوصية قواعد CSS كيفية تطبيق الأنماط.

مثال:

css

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

h1 {
    color: blue;
    font-size: 24px;
}

p {
    color: gray;
    line-height: 1.5;
}
```

- **JavaScript**

-التعريف: JavaScript هي لغة برمجة تستخدم لإنشاء محتوى ويب ديناميكي وتفاعلي.

- الدور: تضيف JavaScript التفاعلية إلى صفحات الويب، مثل التحقق من صحة النماذج، والرسوم المتحركة، وتحميل البيانات غير المتزامنة.

- المفاهيم الرئيسية:

- المتغيرات وأنواع البيانات: تخزين البيانات ومعالجتها (مثل السلاسل والأرقام والمصفوفات والكائنات).
- الوظائف: تغليف كتل التعليمات البرمجية القابلة لإعادة الاستخدام.
- معالجة DOM: الوصول إلى عناصر وسمات HTML وتعديلها برمجيًا.
- الأحداث: الرد على إجراءات المستخدم (مثل النقرات وعمليات إرسال النماذج).

مثال:

html

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Example</title>
<script>
function showAlert() {
alert("Button clicked!");
}
</script>
</head>
<body>
<button onclick="showAlert()">Click Me</button>
</body>
</html>
```

- **تمارين عملية**

أ. تمرين HTML/CSS: قم بإنشاء صفحة ويب بسيطة تحتوي على رأس وفقرة وصورة ورابط. قم بتصميم الصفحة باستخدام CSS لتحسين مظهرها المرئي.

ب. تمرين JavaScript: أضف التفاعلية إلى صفحة الويب التي تم إنشاؤها في التمرين السابق عن طريق تنفيذ زر يغير نص الفقرة عند النقر عليه.

انظر إلى مدى سلاسة تدفق البيانات بين واجهة التطبيق والخادم. هل تساءلت يومًا كيف؟
اكتشف تطوير الواجهة الخلفية الذي يضمن الاتصال السلس وتوافق البيانات مع الواجهة الأمامية.

6. تطوير الواجهة الخلفية Back End للتواصل مع الخادم والبيانات المتوافقة مع الواجهة الأمامية Front End .

- **تعريف ونظرة عامة**

- تطوير الواجهة الخلفية (Back End): يشير تطوير الواجهة الخلفية إلى الجزء الخاص بالخادم من التطبيق. يتضمن إنشاء وإدارة الخادم وقاعدة البيانات ومنطق التطبيق الذي يعمل على تشغيل الواجهة الأمامية.
- الأهمية: تعد الواجهة الخلفية أمرًا بالغ الأهمية لمعالجة طلبات المستخدم، والتفاعل مع قواعد البيانات، وتنفيذ منطق الأعمال، وضمان الاتصال الآمن والفعال بين الواجهة الأمامية والخادم.

- **المسؤوليات الرئيسية للواجهة الخلفية**

1. **إدارة الخادم:** تدير الواجهة الخلفية الخادم حيث يتم تشغيل التطبيق. يتضمن ذلك تكوين بيئة الخادم وصيانتها للتأكد من أنها آمنة وموثوقة وفعالة.
2. **إدارة قواعد البيانات:** يتضمن تطوير الواجهة الخلفية التفاعل مع قواعد البيانات لتخزين البيانات واسترجاعها ومعالجتها. يتضمن ذلك تصميم مخططات قاعدة البيانات وكتابة الاستعلامات وتحسين أداء قاعدة البيانات.
3. **منطق التطبيق:** تنفذ الواجهة الخلفية المنطق الأساسي للتطبيق، بما في ذلك معالجة مدخلات المستخدم، وتنفيذ قواعد العمل، وإنشاء الاستجابات. يعد هذا المنطق ضروريًا لتلبية المتطلبات الوظيفية للتطبيق.
4. **تطوير واجهة برمجة التطبيقات (API):** يقوم مطورو الواجهة الخلفية بإنشاء واجهات برمجة التطبيقات (واجهات برمجة التطبيقات) التي تسمح للواجهة الأمامية بالتواصل مع الخادم. تحدد واجهات برمجة التطبيقات نقاط النهاية وطرق تبادل البيانات بين العميل والخادم.
5. **الأمان:** يعد ضمان أمان التطبيق مسؤولية خلفية بالغة الأهمية. يتضمن ذلك تنفيذ آليات المصادقة والترخيص، وتأمين البيانات أثناء النقل وأثناء الراحة، والحماية من التهديدات الأمنية الشائعة.
6. **تحسين الأداء:** يجب تحسين الواجهة الخلفية للأداء للتعامل مع طلبات المستخدم بكفاءة. يتضمن ذلك تحسين تكوينات الخادم واستعلامات قاعدة البيانات ومنطق التطبيق لتقليل زمن الوصول وضمان قابلية التوسع.

مثال : في أحد تطبيقات التجارة الإلكترونية، تعالج الواجهة الخلفية مصادقة المستخدم، وتعالج الطلبات، وتتفاعل مع قاعدة البيانات لجلب معلومات المنتج، وتدير معاملات الدفع.