1) What is a prefab?
Unity's **Prefab** system allows you to create, configure, and store a **GameObject** complete with all its components, property values, and child GameObjects as a reusable Asset. The Prefab Asset acts as a template from which you can create new Prefab instances in the **Scene.**

2) What is Monobehaviour
MonoBehaviour is the base class from which every Unity script derives.

3) If needed, explain OOP:

What are the main principles of OOP?

Object-oriented programming is based on the following principles:

- **Encapsulation.** This principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.

- **Abstraction.** Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.

- **Inheritance.** Classes can reuse code from other classes. Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.

- **Polymorphism.** Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

4) Unity order of execution (just the basic ones):

```
private void Awake()
{

}

private void OnEnable()
{

}
```

```
    private void Start()
    {

    }

    private void Update()
    {

    }

    private void FixedUpdate()
    {

    }

    private void LateUpdate()
    {

    }

    private void OnDrawGizmos()
    {

    }

    private void OnApplicationQuit()
    {

    }

    private void OnDisable()
    {

    }
```

5) Change order of execution of scripts:
https://docs.unity3d.com/Manual/class-MonoManager.html

6) Character Camera Control

•Character: Character design is one of the areas where some of the big studios like Nintendo and Ubisoft spend a significant amount of time. You want to make sure your character is not only unique, but also that the mechanics they use feel natural and cohesive with your narrative, the characters themselves, and the world you have created. This C also stands for understanding what the metrics/abilities of your character are: how far they can jump, how much damage they can deal, how quick they run/walk, etc. Nailing this C will ensure you have a solid character that is recognizable, unique and that feels good to control.

•Camera: there are different types of cameras you can use depending on the type of game you are making. It can be first person, third person, isometric, top-down, etc. The important takeaway here is that you need to choose the right one for your game. For example, a third person camera allows you better visibility of the world, vs an isometric camera that gives you a strategic view of the world you're in. The main thing here is that it needs to show the player the important things in the game to be able to move forward and most importantly not get in the way of the player's actions.
•Control: The last and possibly most crucial of the C's is control. I think we can all agree that among the best games ever, those that are most salient are the ones with the best 'feel.' It feels good to control the character, it feels natural to move or to perform certain actions. This is one of the most difficult things to get right, but when you do, it's noticeable. A good tip for all C's, but this one in particular, is to playtest and keep playtesting and making sure to spend a lot of time on them before moving on to level design and other things

7) Debugging in Visual Studio + Debug logs and displays

8) Delta time + Frame rate independent

9) User inputs using input manager (touch input next session): https://gamedevbeginner.com/input-in-unity-made-easy-complete-guide-to-the-new-system/

10) SerializeField + private member rather than public

11) KeyDown, KeyUp and GetKey

12) Change vector3 or translate : https://docs.unity3d.com/ScriptReference/Vector3.html