1. Just in time compilation – it combines the benefit of both interpretation and ahead of time compilation , offering the flexibility of cross platform code distribution with the performance of native machine code execution.
2. AOT Compilation: Fast startup, fixed optimizations, less flexible, platform-dependent.

JIT Compilation: Slower startup, runtime optimizations, more flexible, platform-independent up to runtime.

Interpreted Languages: Slowest, highest portability, no machine code generated unless combined with JIT.

1. **JIT Compilation:**

Compilation happens at runtime, during program execution.

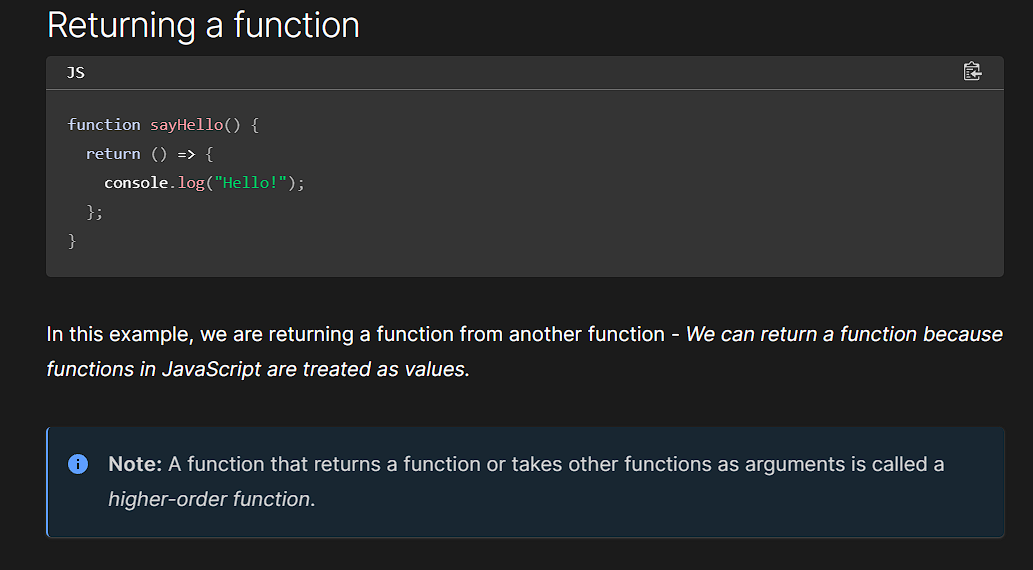
The program starts as platform-independent intermediate code, which is then compiled into platform-specific machine code by the runtime environment.

Runtime optimizations are possible based on actual execution conditions, but there may be a slight delay at startup due to the on-the-fly compilation.

1. **Passing a function as an argument**
2. function sayHello() {
3. return 'hello,';
4. }
5. function greeting(*hellomessage*, *name*) {
6. console.log(hellomessage() + name);
7. }
8. greeting(sayHello, 'gandu');
9. *// we are passing sayHello function as an argument to the greeting function*

 The function that we pass as an argument to another function is called a *callback function.* *sayHello() is a callback function.*

5. **returning a function**

****

When you sayHello function, it returns the arrow function without executing it. To print hello , you would need to call the returned function .

function sayHello() {

  return () => {

    console.log('Hello!');

  };

}

let greet = sayHello();

greet();  *// This will log "Hello!" to the console*

The key concept here is that **returning a function** does not execute the function; it merely provides the function to be used later. This is fundamental to how functions work in JavaScript, especially with first-class functions

6. **Dynamically-typed languages** are those (like [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript)) where the interpreter assigns [variables](https://developer.mozilla.org/en-US/docs/Glossary/Variable) a [type](https://developer.mozilla.org/en-US/docs/Glossary/Type) at runtime based on the variable's [value](https://developer.mozilla.org/en-US/docs/Glossary/Value) at the time.

7 . **API -**  the api can be seen as a simple contract(interface) between the application offering it and other items such as 3rd party software/hardware.

It is a set of code features that a developer can use in their apps for interacting with components of a user’s web browser , other software on user’s computer or 3rd party website and services

8. The **DOM** (Document Object Model) is an [API](https://developer.mozilla.org/en-US/docs/Glossary/API) that represents and interacts with any [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) or [XML](https://developer.mozilla.org/en-US/docs/Glossary/XML)-based markup language document. The DOM is a document model loaded in the [browser](https://developer.mozilla.org/en-US/docs/Glossary/Browser) and representing the document as a [node](https://developer.mozilla.org/en-US/docs/Glossary/Node/DOM) tree, or **DOM tree**, where each node represents part of the document (e.g. an [element](https://developer.mozilla.org/en-US/docs/Glossary/Element), text string, or comment).

The DOM is one of the most-used [API](https://developer.mozilla.org/en-US/docs/Glossary/API)s on the [Web](https://developer.mozilla.org/en-US/docs/Glossary/World_Wide_Web) because it allows code running in a browser to access and interact with every node in the document. Nodes can be created, moved, and changed. Event listeners can be added to nodes and triggered on the occurrence of a given event.