**Q1. Difference between Statically typed and Dynamically typed language?**

Statically typed language:

A language is statically typed if the type of a variable is known at compile time. For some languages this means that you as the programmer must specify what type each variable is; other languages (e.g.: Java, C, C++) offer some form of type inference, the capability of the type system to deduce the type of a variable (e.g.: OCaml, Haskell, Scala, Kotlin).

The main advantage here is that all kinds of checking can be done by the compiler, and therefore a lot of trivial bugs are caught at a very early stage.

Examples: C, C++, Java, Rust, Go, Scala.

Dynamically typed language:

A language is dynamically typed if the type is associated with run-time values, and not named variables/fields/etc. This means that you as a programmer can write a little quicker because you do not have to specify types every time (unless using a statically-typed language with type inference).

Examples: Perl, Ruby, Python, PHP, JavaScript, Erlang

Most scripting languages have this feature as there is no compiler to do static type-checking anyway, but you may find yourself searching for a bug that is due to the interpreter misinterpreting the type of a variable.

**Q2.Scrippting language and Programming language?**

**Scrippting language:**

Scripting languages are programming languages that don't require an explicit compilation step.For example, in the normal case, you have to compile a C program before you can run it. But in the normal case, you don't have to compile a JavaScript program before you run it. So JavaScript is sometimes called a "scripting" language

Scripting languages are interpreted within another program. JavaScript is embedded within a browser and interpreted by that browser.

**Examples of scripting languages**

1. JavaScript
2. Perl
3. Python

**Advantages of Scripting languages:**

1. Simple – Scripting languages are easier to write than programming language.
2. Fewer Lines of Code (LOC)

**Programming language:**

Programming Language : Is compiled to machine code and run on the hardware of the underlying Operating System.

Scripting Language : Is unstructure subset of programming language. It is generally interpreted. it basically "scripts" other things to do stuff. The primary focus isn't primarily building your own apps but getting an existing app to act the way you want, e.g. JavaScript for browsers, TCL etc

Programming languages like Java are compiled and not interpreted by another application in the same way.

**Examples programming languages**

1. C
2. C++ and
3. Java

Q3.Programming paradigm:

A programming paradigm is a way or style of programming.

Some languages make it easy to write in paradigm but others not.

**Some Common Paradigms**

You should know these:

* **Imperative**: Programming with an explicit sequence of commands that update state.
* **Declarative**: Programming by specifying the result you want, not how to get it.
* **Structured**: Programming with clean, goto-free, nested control structures.
* **Procedural**: Imperative programming with procedure calls.
* **Functional** (Applicative): Programming with function calls that avoid any global state.
* **Function-Level** (Combinator): Programming with no variables at all.
* **Object-Oriented**: Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces. Object orientation can be:
  + **Class-based**: Objects get state and behavior based on membership in a class.
  + **Prototype-based**: Objects get behavior from a prototype object.
* **Event-Driven**: Programming with emitters and listeners of asynchronous actions.
* **Flow-Driven**: Programming processes communicating with each other over predefined channels.
* **Logic** (Rule-based): Programming by specifying a set of facts and rules. An engine infers the answers to questions.
* **Constraint**: Programming by specifying a set of constraints. An engine finds the values that meet the constraints.
* **Aspect-Oriented**: Programming cross-cutting concerns applied transparently.
* **Reflective**: Programming by manipulating the program elements themselves.
* **Array**: Programming with powerful array operators that usually make loops unnecessary.

Q4.Write a blog on differenc between HTTP1 and HTTP2?

HTTP2 Vs. HTTP1 is not a debate at all. HTTP2 is much faster and more reliable than HTTP1. HTTP1 loads a single request for every TCP connection, while HTTP2 avoids network delay by using multiplexing.

HTTP is a network delay sensitive protocol in the sense that if there is less network delay, then the page loads faster. However, an impressive increase in network bandwidth only slightly improves page load time. This is key to understanding the differences in performance efficiencies between the different versions of HTTP. Back in the day when people used dial up modems web pages were simple and it was the actual data transfer between the server and the client that contributed towards the largest chunk of the page load time. Today the actual downloading of resources from server takes a negligible portion of the total page load time due to the tremendous increase in bandwidth availability. It is the time taken to establish the TCP connection and making requests that impacts performance. It was initially recommended to use only two connections per hostname but today most browsers use six connections per hostname. When we talk about http vs http2 in terms of performance it is important to note that a lot of performance optimizations adopted by HTTP/1.1 introduced complexities in terms of developmental efforts as well as network congestion that HTTP/2 attempts to address.

The table below points out the differentiating factors between http2 vs http1:

Header CompressionHeaders are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire.Header compression is included by default in HTTP/2 using HPACK.Performance OptimizationProvides support for caching to deliver pages faster.Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the ‘six connections per host’ rule.Removes the need for unnecessary optimization hacks.Protocol TypeText based protocol that is in the readable form.It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it.SecuritySSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication.Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

Header CompressionHeaders are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire.Header compression is included by default in HTTP/2 using HPACK.Performance OptimizationProvides support for caching to deliver pages faster.Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the ‘six connections per host’ rule.Removes the need for unnecessary optimization hacks.Protocol TypeText based protocol that is in the readable form.It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it.SecuritySSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication.Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

**Q5. Write a blog about objects and its internal representation in Javascript.**

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;