1. Difference between Static typed and Dynamic typed programming language.

Statically typed language	**Dynamic types language**
Examples of Static typed language : C, C++, Java	Examples of Dynamic typed language : Python, JavaScript, perl, Ruby, Lua
They are compiled when executed.	They are interpreted when executed.
During execution, the data type is checked before runtime (while compiling) and throws error if the type is not suitable	During execution, the data type is checked at run time and does not throw error if the type is not suitable
Performance will be better comparing to dynamic typed language	Performance will be slow comparing to static typed languages
Type errors will be detected earlier at the time of compiling and makes it ideal for larger programs.	Type errors will be found during execution makes the program complicated and consumes more time on debugging
Variable types are not flexible as coercion cannot happen automatically.	Variable types are flexible as coercion will convert the type based on the operation.

2. Difference between programming languages and scripting languages.

Programming language	**Scripting language**
It is a compiled or compiler based language.	It is interpreted or interpreter based language.
It is used to build an application or software from scratch.	It is used to automate a specific task using existing elements.
It executes or runs independently and does not require a parent program.	It executes or runs inside a parent program.
It uses a compiler to convert source code to machine code.	It uses an interpreter to convert source code to machine code.
It requires compilation.	It doesn' require compilation.

It is comparatively difficult to write code and requires more lines of code for a specific operation.	It is comparatively easy to write code and requires less lines of code for a specific operation.
There is a high maintenance cost.	There is a low maintenance cost.
All programming languages are not scripting languages.	All scripting languages are programming languages.
It creates a .exe. file	It doesn't create a .exe file.
Examples : C, C++, Java, Scala, COBOL, etc.	Examples: Perl, Python, JavaScript, etc.

3. Comparison of HTTP 1.1 and HTTP 2.

   Release dates:
   HTTP1 was documented in1997 as 1.1.
   HTTP2 was published in 2015.

   Prioritization:
   In general, when a client requests data from the server. It will load n number of data to the client browser like html, css, javascript files. It should be prioritized which data needs to rendered on client's browser to make the process easier and keep client engaged. If the banner ads at the bottom of the page loads first, the client will think it is not loading and move on to the next website. If the heavy sized .js file loaded at first, it will block all the html and css files to render. To avoid this complications, there is an update brought out in HTTP2 as weighted prioritization. Using this feature, developers can decide which pages should be rendered on client's browser

   Multiplexing:
   HTTP2 can send multiple data by using multiplexing method. Usually HTTP1.1 sends data one by one based on the acknowledgement given by TCP. This makes the process slow. But HTTP2 can send a lot of data by converting them into multiple binary codes and numbering them for the browser understanding.

   Header Compression:
   Both HTTP 1.1 and HTTP2 compresses the header contents to speed up the transmission. In HTTP2, a more advanced header compression method called HPACK is used. This makes the web page even quicker to load.

   Server Push:

HTTP2 pulls additional files that may be required by the client to the browser before requesting it. It helps the client to access the data without wasting time until the files arrive. For example: If a client accesses a website, the html file will be loaded. Additionally, css, js files will be pushed automatically by the server to the client for immediate processing. This can also waste the bandwidth if the client doesn't request it.

4. Objects in JavaScript:

   Object is an entity having many states and behaviours which is also known as key and value pairs. States means properties and behaviours means methods.

   For example :

   Objects can be accessed using syntax : object.property.

```
var shankar = {
    age:"29",
    colour:"brown",
    nationality:"Indian",
}

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
```

Output:
29
Brown
Indian

Objects can be accessed using syntax : object["property"].

```
var shankar = {
    age:"29",
    colour:"brown",
    nationality:"Indian",
}

console.log(shankar["age"]);
console.log(shankar["colour"]);
```

```
console.log(shankar["nationality"]);
```

Output will be the same as above.

Objects can also have methods. But before creating a method, use the property name as same as function name within the object.

```
var shankar = {
    age:"29",
    colour:"brown",
    nationality:"Indian",
    details: function details(){ return `Shankar is ${this.age} years old
and ${this.colour}ish colour.
He is an ${this.nationality}`;}
}

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
console.log(shankar.details());
```

In this above example, we have added a method as a function named as details(). And also note that, property name is same as function name.

We can also use anonymous function here as mentioned below.

```
var shankar = {
    age:"29",
    colour:"brown",
    nationality:"Indian",
    details: function(){ return `Shankar is ${this.age} years old and
${this.colour}ish colour.
He is an ${this.nationality}`;}
}

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
console.log(shankar.details());
```

This will also work and give the same output.

Objects can be created by 3 types.

1. By object literal
2. By creating an instance of an object using the “new” keyword.
3. By using object constructor.

Object literal method :

```
var shankar = {
    age:"29",
    colour:"brown",
    nationality:"Indian",
    details: function(){ return `Shankar is ${this.age} years old and
${this.colour}ish colour.
He is an ${this.nationality}`;}
}

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
console.log(shankar.details());
```

By creating an instance of an object using the “new” keyword.

```
var shankar = new Object();
shankar.age=29;
shankar.colour="brown";
shankar.nationality="Indian";
shankar.details = function(){return `Shankar is ${this.age} years old and
${this.colour}ish colour.
He is an ${this.nationality}`;}

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
console.log(shankar.details());
```

By using object constructor.

```
function shankar1(age,colour,nationality){
    this.age=age;
    this.colour=colour;
    this.nationality=nationality;
    this.details = function() {return `Shankar is ${this.age} years old
and ${this.colour}ish colour.
    He is an ${this.nationality}`;}
}

var shankar = new shankar1(29,"brown","Indian")

console.log(shankar.age);
console.log(shankar.colour);
console.log(shankar.nationality);
console.log(shankar.details());
```

All these object methods will give the same output.