1. **Чи є різниця між виконанням JavaScript в браузері та в середовищі Node.js?**
   Так, є деяка різниця між виконанням JavaScript у браузері та в середовищі Node.js.

   Найбільш очевидна різниця полягає в тому, що у браузері JavaScript використовується для взаємодії зі сторінкою та її елементами, такими як DOM-елементи та події миші та клавіатури. У Node.js, з іншого боку, JavaScript використовується для програмування на стороні сервера та розробки інструментів командного рядка, таких як npm.

   Інші різниці між браузерним та середовищем Node.js включають:

   Обмеження доступу до браузерних API у Node.js. Наприклад, у браузері є API для доступу до DOM-елементів та обробки подій миші та клавіатури, але вони не підтримуються у Node.js.

   Різні глобальні об'єкти та функції, що доступні в браузері та Node.js. Наприклад, у браузері глобальний об'єкт window, а у Node.js глобальний об'єкт global.

   Різні модулі та бібліотеки, які доступні у браузері та Node.js. Наприклад, у браузері можна використовувати бібліотеку jQuery для зручної роботи з DOM-елементами, а у Node.js є багато модулів для роботи з файловою системою, мережевими запитами та іншими завданнями, які зазвичай пов'язані з серверною розробкою.

   Таким чином, хоча JavaScript в браузері та Node.js базуються на одній мові, вони мають деякі різні характеристики та функціональність, що важливо враховувати при розробці програмного забезпечення.

2. **Назвіть основні типи в JavaScript та поясніть їх**
   JavaScript має декілька основних типів, а саме:

   <u>Примітивні типи даних:</u>

   Числа (Numbers): цілі числа та числа з плаваючою крапкою. Приклад: 42, 3.14.

   Рядки (Strings): послідовності символів, що обмежуються лапками (одинарні або подвійні). Приклад: "Hello, world!", 'JavaScript is awesome'.

   Булеві значення (Booleans): два можливих значення - true та false. Використовуються для умовних операторів та логічних виразів. Приклад: true, false.

   Null: спеціальне значення, яке вказує на відсутність значення. Приклад: null.

   Undefined: значення, що вказує на те, що змінна не має присвоєного значення. Приклад: undefined.

   Symbol: унікальні ідентифікатори, що використовуються в об'єктах.

   <u>Об'єктні типи даних:</u>

   Об'єкти (Objects): структури даних, які складаються з властивостей (пари ключ-значення).
   Приклад: { name: "John", age: 30 }.

   Масиви (Arrays): відсортовані колекції значень, до яких можна отримувати доступ за індексом. Приклад: [1, 2, 3, 4, 5].

   Функції (Functions): блоки коду, які можуть бути викликані для виконання певної дії та повертати значення. Приклад: function add(x, y) { return x + y; }.

Крім того, JavaScript має декілька спеціальних типів даних, таких як NaN (Not a Number), який використовується для позначення некоректних математичних операцій, та Infinity та -Infinity, які вказують на безмежність та від'ємну безмежність відповідно.

3. **Як працює замикання (closure) в javascript?**

Замикання (closure) - це механізм мови JavaScript, який дозволяє зберігати локальні змінні та параметри функції в пам'яті, навіть після завершення її виконання. Замикання дозволяє створювати функції, які зберігають контекст виклику, включаючи значення змінних, що були визначені в більш високорівневій функції.

Коли функція в JavaScript повертається як значення або передається як аргумент іншій функції, вона може бути викликана пізніше з тим самим контекстом, який містить локальні змінні та параметри з попереднього виклику. Це дозволяє функції зберігати стан та поводитись як об'єкти.

Приклад замикання в JavaScript:

```
function outerFunction() {
  var outerVariable = 10;

  function innerFunction() {
    console.log(outerVariable);
  }

  return innerFunction;
}

var innerFunc = outerFunction();
innerFunc(); // виведе 10
```

У цьому прикладі функція innerFunction має доступ до змінної outerVariable, яка була визначена в більш високорівневій функції outerFunction. Після виклику функції outerFunction її змінна outerVariable залишається доступною в пам'яті завдяки механізму замикання. Функція outerFunction повертає innerFunction як значення, і змінна innerFunc отримує посилання на цю функцію. Після того, як innerFunc була викликана, вона має доступ до змінної outerVariable, яка була збережена в пам'яті завдяки механізму замикання.
Замикання можуть бути корисними в багатьох сценаріях, таких як обробка подій, асинхронне програмування та керування доступом до даних та ресурсів.

4. **Назви основні стандартні бібліотеки Node.js**

Node.js має широку стандартну бібліотеку, яка надає багато різноманітних інструментів та модулів, що допомагають розробникам писати високоякісний код. Основні стандартні бібліотеки Node.js можна класифікувати в такі категорії:

Робота з файловою системою - fs

Робота з мережевими протоколами - http, https, dns, net, tls

Робота з каталогами та шляхами - path

Робота з процесами та системою - child_process, os, process, util

Робота з потоками введення/виведення - readline, stream

Робота з рядками - querystring, string_decoder

Робота з часом та датами - timers, date

Робота з криптографією - crypto, tls

Ці бібліотеки дозволяють розробникам Node.js створювати програми для різних задач та додавати до них додаткові функції, які допомагають у роботі з даними, мережевими протоколами, шифруванням, роботою з потоками та багато іншого.

5. Які є способи імпортувати модулі в Node.js?

У Node.js є різні способи імпортування модулів залежно від версії Node.js, яку ви використовуєте, та формату модуля, який використовується модулем, який ви хочете імпортувати. Нижче наведено способи імпорту модулів у Node.js:

- За допомогою функції require(): Це найпоширеніший спосіб імпорту модулів в Node.js. Функція require() - це вбудована функція в Node.js, яка дозволяє імпортувати модулі. Ви можете використовувати її для імпорту вбудованих модулів або модулів, встановлених за допомогою npm.

```
const http = require('http');
```

- Використання інструкції import: Оператор import - це новий синтаксис для імпорту модулів в Node.js, і він вимагає використання формату модулів ES. Щоб використовувати його, вам потрібно ввімкнути прапорець "experimental-modules" під час запуску вашого скрипту.

```
import http з 'http';
```

- Використання функції import(): Це динамічний спосіб імпорту модулів в Node.js. Можна використовувати його для завантаження модулів умовно або на вимогу.

```
import('http').then(http => {
// зробити щось з модулем http
});
```

- Використання module.exports: Якщо ви пишете модуль, ви можете експортувати його функціональність за допомогою об'єкта module.exports. Потім інші модулі можуть імпортувати ваш модуль за допомогою функції require().

```
// greet.js
module.exports = {
  greet(ім'я) {
    console.log(`Hello, ${name}!`);
  }
};

// index.js
const greet = require('./greet');
greet('World'); // Hello, World!
```

6. **Як пов’язані Google Chrome / Chromium та Node.js?**

Google Chrome / Chromium і Node.js побудовані на JavaScript-engine V8 - це JavaScript-engine з відкритим вихідним кодом, який компілює JavaScript-код у машинний код для швидшого виконання.

Node.js - це середовище виконання JavaScript, яке використовує двигун V8 для виконання JavaScript-коду на стороні сервера. Він надає набір API для операцій з файловою системою, мережевого зв'язку та інших завдань, які недоступні в середовищі браузера.

Google Chrome / Chromium, з іншого боку, - це веб-браузер, який використовує V8 для виконання JavaScript-коду на стороні клієнта. Він надає користувацький інтерфейс для відображення веб-сторінок, виконання коду JavaScript та взаємодії з веб-сервісами.

Google, розробник Google Chrome / Chromium, також є одним з основних учасників розробки Node.js. Google зробив свій внесок у розвиток Node.js, надаючи ресурси, код та сприяючи його впровадженню.

Node.js також отримав користь від інновацій, впроваджених в Google Chrome / Chromium, таких як протокол WebSocket, який вперше був реалізований в Google Chrome / Chromium до того, як був інтегрований в Node.js.

Таким чином, Google Chrome / Chromium і Node.js побудовані на V8 і були розроблені компанією Google. У той час як Google Chrome / Chromium фокусується на рендерингу веб-сторінок і виконанні JavaScript-коду на стороні клієнта, Node.js надає середовище виконання для виконання JavaScript-коду на стороні сервера.

7. **Як можна дозволити імпортувати дані з поточного модуля Node.js?**

Щоб дозволити імпорт даних з поточного модуля Node.js, вам потрібно визначити дані, які ви хочете експортувати з модуля за допомогою об'єкта module.exports.

```
// myModule.js
const myData = {
  name: "John",
  вік: 30
};
module.exports = myData;
```

У вищенаведеному коді ми визначаємо об'єкт myData і присвоюємо його модулю module.exports. Це робить myData доступним для імпорту в інші модулі.

Щоб імпортувати myData в інший модуль, ви можете використовувати функцію require, наприклад, так:

```
// anotherModule.js
const myData = require('./myModule');
console.log(myData.name); // Виведення: John
console.log(myData.age); // Виведення: 30
```

У вищенаведеному коді ми використовуємо функцію require для імпорту об'єкта myData з файлу myModule.js. Після цього ми можемо отримати доступ до властивостей myData у файлі anotherModule.js.