2. CSS Interview Questions

>> Basic CSS Questions (8 questions)

1. What is CSS, and how does it work with HTML?

CSS বা Cascading Style Sheets হলো একটি ভাষা যা দিয়ে HTML ডকুমেন্টের ডিজাইন ও লেআউট কন্ট্রোল করা হয়। এটি মূলত ওয়েব পেজের রঙ, ফন্ট, মার্জিন, প্যাডিং, অবস্থান, ইত্যাদি স্টাইলিং করতে ব্যবহৃত হয়।

HTML ডকুমেন্টে আমরা কন্টেন্ট (যেমনঃ টেক্সট, ইমেজ, লিংক) তৈরি করি, কিন্তু সেই কন্টেন্টের চেহারা বা স্টাইলিং কেমন হবে তা CSS দিয়ে ঠিক করা হয়। HTML ও CSS একসাথে কাজ করে যাতে আপনার ওয়েব পেজ দেখতে সুন্দর ও আকর্ষণীয় হয়। উদাহরণস্বরূপ, HTML দিয়ে আপনি একটি heading তৈরি করতে পারেন, কিন্তু সেই heading এর রঙ লাল বা ফন্ট সাইজ বড় করতে হলে CSS ব্যবহার করতে হবে।

2. What is the difference between inline, internal, and external CSS?

CSS-এর তিনটি প্রধান প্রকার হলো **Inline CSS**, **Internal CSS**, এবং **External CSS**। প্রতিটি প্রকারের নিজস্ব ব্যবহারযোগ্য ক্ষেত্র ও সুবিধা আছে। এগুলোর মধ্যে পার্থক্য হলো:

1. Inline CSS:

- **কোড**: HTML ট্যাগের ভিতরে `style` অ্যাট্রিবিউটের মাধ্যমে স্টাইল দেয়া হয়।
- **ব্যবহার**: শুধুমাত্র নির্দিষ্ট HTML উপাদানকে (element) স্টাইল করার জন্য।
- **উদাহরণ**:

```
<h1 style="color: red; font-size: 24px;">Hello, World!</h1
>
```

- **সুবিধা**: দ্রুত এবং ছোটখাট স্টাইলিং পরিবর্তনের জন্য ভালো।
- **অসুবিধা**: কোডের পুনরাবৃত্তি বেড়ে যায় এবং HTML ও CSS মিশে যায়, ফলে কোড জটিল হয়।

2. Internal CSS:

- **কোড**: HTML ডকুমেন্টের `<head>` সেকশনের মধ্যে `<style>` ট্যাগ ব্যবহার করে লেখা হয়।
- **ব্যবহার**: পুরো HTML ডকুমেন্টের জন্য নির্দিষ্ট স্টাইল প্রযোজ্য করতে।
- **উদাহরণ**:

```
<style>
    h1 {
        color: blue;
        font-size: 28px;
    }
</style>
```

- **সুবিধা**: পুরো পেজে নির্দিষ্ট স্টাইল প্রযোজ্য করা যায়।
- **অসুবিধা**: অন্য HTML পেজে এই CSS ব্যবহার করা যায় না।

3. External CSS:

- **কোড**: আলাদা `.css` ফাইল তৈরি করে সেটি HTML ডকুমেন্টের `<head>` সেকশনে `<link>` ট্যাগের মাধ্যমে যুক্ত করা হয়।
- **ব্যবহার**: একই স্টাইল একাধিক HTML পেজে প্রয়োগ করতে।

- **উদাহরণ**:

```
<link rel="stylesheet" href="styles.css">
```

- **সুবিধা**: ওয়েবসাইটের একাধিক পেজে একই স্টাইল প্রযোজ্য করা যায় এবং HTML থেকে CSS আলাদা থাকে, ফলে কোড পরিচ্ছন্ন ও মেইনটেইনেবল হয়।
- **অসুবিধা**: এক্সটার্নাল CSS ফাইল লোড হতে সময় নেয়, যা পেজ লোড টাইমে কিছুটা প্রভাব ফেলতে পারে।

3. Explain the CSS box model. What properties does it consist of?

CSS বক্স মডেল হলো একধরনের কাঠামো যা প্রতিটি HTML উপাদানকে একটি বক্স হিসাবে দেখায়। এই বক্স মডেলকে কাজে লাগিয়ে আমরা বিভিন্ন HTML উপাদানের আকার, স্পেসিং, এবং অবস্থান নির্ধারণ করতে পারি। প্রতিটি উপাদানকে একটি বক্স হিসেবে কল্পনা করলে সেটি চারটি মূল অংশ নিয়ে গঠিত: **Content**, **Padding**, **Border**, এবং **Margin**।

CSS বক্স মডেলের অংশগুলো:

1. **Content**:
    - এটি মূল উপাদান যেখানে টেক্সট বা ছবি থাকে।
    - এর দৈর্ঘ্য এবং প্রস্থ পরিবর্তন করে কন্টেন্টের আকার নিয়ন্ত্রণ করা যায়।
2. **Padding**:
    - এটি কন্টেন্ট ও বর্ডারের মধ্যে থাকা ফাঁকা জায়গা।
    - প্যাডিং এর মাধ্যমে কন্টেন্টের চারপাশে জায়গা তৈরি করা যায়।
    - উদাহরণ: যদি প্যাডিং বাড়ানো হয়, তাহলে কন্টেন্ট ও বর্ডারের মধ্যে ফাঁকা জায়গা বাড়ে।
3. **Border**:
    - এটি প্যাডিং ও মার্জিনের মধ্যে থাকা সীমারেখা যা বক্সের চারপাশে থাকে।
    - বর্ডার এর প্রস্থ, রং এবং স্টাইল পরিবর্তন করা যায়।
4. **Margin**:

- এটি বর্ডারের বাইরের জায়গা যা একটি উপাদানের চারপাশে ফাঁকা স্থান তৈরি করে।
- মার্জিনের মাধ্যমে উপাদানগুলির মধ্যে জায়গা বা দূরত্ব বৃদ্ধি করা যায়।
- উদাহরণ: মার্জিন বাড়ালে উপাদান ও অন্য উপাদানের মধ্যে দূরত্ব বাড়ে।

উদাহরণ:

একটি উপাদানের বক্স মডেল এইভাবে সাজানো থাকে:

```
| Margin |
| Border |
| Padding |
| Content |
```

বক্স মডেলের সম্পূর্ণ প্রপার্টিগুলি:

- **width** এবং **height** - কন্টেন্টের আকার।
- **padding** - কন্টেন্ট ও বর্ডারের মধ্যবর্তী জায়গা।
- **border** - কন্টেন্ট ও প্যাডিং এর চারপাশের সীমারেখা।
- **margin** - বর্ডারের বাইরের ফাঁকা জায়গা।

বক্স মডেল বুঝতে পারলে উপাদানের লেআউট ও স্পেসিং সহজে নিয়ন্ত্রণ করা যায়, যা CSS ডিজাইনে গুরুত্বপূর্ণ ভূমিকা রাখে।

4. What is specificity in CSS, and how does it affect styling?

CSS-এ **specificity** হলো একটি মেকানিজম যা বলে দেয় কোন CSS সিলেক্টরের স্টাইল প্রয়োগ হবে, যখন একাধিক সিলেক্টর একই HTML উপাদানের উপর প্রযোজ্য হয়। এটি মূলত **CSS-এর স্টাইলিং সংঘর্ষ** (conflict) সমাধান করতে সহায়তা করে।

Specificity কিভাবে কাজ করে:

CSS-এ চারটি ভিন্ন ভিন্ন স্তরের specificity থাকে, যেগুলো ব্রাউজার গুরুত্ব অনুসারে মূল্যায়ন করে। এগুলো হচ্ছে:

1. **Inline styles** (HTML ট্যাগে `style` অ্যাট্রিবিউট) - সবচেয়ে বেশি স্পেসিফিক।
2. **ID selectors** (যেমন `#header`) - দ্বিতীয় সর্বোচ্চ স্পেসিফিক।
3. **Class selectors**, **attribute selectors**, এবং **pseudo-classes** (যেমন `.menu`, `[type="text"]`, `:hover`) - তৃতীয় স্তরের স্পেসিফিক।
4. **Type selectors** এবং **pseudo-elements** (যেমন `div`, `h1`, `::before`) - সবচেয়ে কম স্পেসিফিক।

Specificity এর পয়েন্ট সিস্টেম:

CSS-এর স্পেসিফিসিটির জন্য একটি পয়েন্ট ভিত্তিক সিস্টেম কাজ করে।

- **Inline Style**: 1000 পয়েন্ট।
- **ID Selector**: 100 পয়েন্ট।
- **Class, Attribute, Pseudo-class**: 10 পয়েন্ট।
- **Type Selector এবং Pseudo-element**: 1 পয়েন্ট।

উদাহরণ:

ধরা যাক, নিচের CSS কোডগুলির মধ্যে একটি নির্দিষ্ট `p` উপাদানের স্টাইল কীভাবে প্রযোজ্য হবে:

```
p {
    color: blue; /* 1 পয়েন্ট */
}

.content p {
    color: green; /* 10 + 1 = 11 পয়েন্ট */
}

#main .content p {
    color: red; /* 100 + 10 + 1 = 111 পয়েন্ট */
}
```

```
<p style="color: yellow;">Hello World</p> <!-- Inline Style: 1000 পয়েন্ট -->
```

এই ক্ষেত্রে, `p` উপাদানটি **yellow** রঙে প্রদর্শিত হবে, কারণ Inline Style এর স্পেসিফিসিটি সর্বোচ্চ (1000 পয়েন্ট)।

Specificity এর প্রভাব:

- **বেশি specificity** থাকা সিলেক্টর প্রথমে প্রয়োগ হবে।
- যদি একই specificity থাকে, তাহলে ডকুমেন্টের নিচে থাকা সিলেক্টর প্রয়োগ হবে।

Specificity ব্যবহারের টিপস:

- **ক্লাস** ব্যবহার করে স্টাইল তৈরি করুন, যাতে specificity বেশি না হয়।
- **ID এর ব্যবহার সীমিত রাখুন**, কারণ এটি specificity বাড়ায়।
- CSS-এর নির্ভুলতা এবং সুনির্দিষ্টতা বাড়াতে specificity বোঝা গুরুত্বপূর্ণ, যাতে অপ্রত্যাশিত স্টাইলিং সংঘর্ষ এড়িয়ে চলা যায়।

5. How do CSS selectors work, and what are some common types of selectors?

CSS সিলেক্টর এমন এক প্রকার প্যাটার্ন বা পদ্ধতি যা দিয়ে HTML ডকুমেন্টের নির্দিষ্ট উপাদানগুলোকে চিহ্নিত করা যায়। CSS সিলেক্টর ব্যবহার করে আমরা নির্দিষ্ট HTML উপাদানের উপর স্টাইল প্রয়োগ করতে পারি। বিভিন্ন ধরনের সিলেক্টর রয়েছে, যা আমাদের প্রয়োজন অনুযায়ী নির্দিষ্ট উপাদানগুলোকে টার্গেট করতে সহায়তা করে।

সাধারণ CSS সিলেক্টর ও তাদের প্রকারভেদ:

1. **Universal Selector ( )**:
    - ডকুমেন্টের সব HTML উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```css
* {
    margin: 0;
    padding: 0;
}
```

2. **Element (Type) Selector**:
    - নির্দিষ্ট HTML ট্যাগ বা উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```css
p {
    color: blue;
}
```

3. **Class Selector (`.className`)**:
    - একই ক্লাসের সাথে যুক্ত একাধিক উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```css
.header {
    font-size: 20px;
    color: green;
}
```

4. **ID Selector (`#idName`)**:
    - নির্দিষ্ট ID যুক্ত একটি নির্দিষ্ট উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```css
#main {
    background-color: yellow;
}
```

5. **Attribute Selector (`[attribute=value]`)**:
    - নির্দিষ্ট অ্যাট্রিবিউট থাকা উপাদানের জন্য স্টাইল প্রযোজ্য করতে।

- উদাহরণ:

```
input[type="text"] {
    border: 1px solid black;
}
```

6. **Pseudo-class Selector**:
   - নির্দিষ্ট অবস্থার উপর ভিত্তি করে উপাদানের স্টাইল নিয়ন্ত্রণ করতে। যেমন, `:hover` বা `:focus`।
   - উদাহরণ:

```
a:hover {
    color: red;
}
```

7. **Pseudo-element Selector**:
   - উপাদানের নির্দিষ্ট অংশে স্টাইল প্রয়োগ করতে, যেমন `::before` এবং `::after`।
   - উদাহরণ:

```
p::first-line {
    font-weight: bold;
}
```

8. **Descendant Selector (space)**:
   - একটি উপাদানের ভিতরে থাকা অন্য উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
   - উদাহরণ:

```
div p {
    color: purple;
}
```

9. **Child Selector (`>`)**:

- শুধুমাত্র নির্দিষ্ট প্যারেন্ট উপাদানের সরাসরি চাইল্ড উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
- উদাহরণ:

```
div > p {
    font-size: 16px;
}
```

10. **Adjacent Sibling Selector (`+`)**:
    - নির্দিষ্ট উপাদানের পরবর্তী সহোদর উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```
h1 + p {
    color: gray;
}
```

11. **General Sibling Selector (`~`)**:
    - নির্দিষ্ট উপাদানের পরবর্তী সকল সহোদর উপাদানের জন্য স্টাইল প্রযোজ্য করতে।
    - উদাহরণ:

```
h1 ~ p {
    color: orange;
}
```

সারাংশ:

CSS সিলেক্টর ব্যবহার করে আমরা নির্দিষ্ট উপাদান বা উপাদানগুলোর গ্রুপকে টার্গেট করে তাদের উপর স্টাইল প্রয়োগ করতে পারি। বিভিন্ন প্রকার সিলেক্টরের সাহায্যে প্রয়োজনমতো কাস্টম স্টাইলিং করা সম্ভব হয়, যা ওয়েবসাইট ডিজাইনের নিয়ন্ত্রণ ও সুনির্দিষ্টতা বজায় রাখতে সহায়ক।

6. What are pseudo-classes and pseudo-elements? Give examples of each.

Pseudo-classes এবং **pseudo-elements** উভয়ই CSS-এর গুরুত্বপূর্ণ অংশ, যা নির্দিষ্ট উপাদানের বিশেষ অবস্থায় বা নির্দিষ্ট অংশে স্টাইল প্রয়োগ করতে সাহায্য করে। তবে তাদের উদ্দেশ্য ও ব্যবহার পদ্ধতি ভিন্ন।

1. Pseudo-classes:

Pseudo-classes ব্যবহার করে উপাদানের একটি নির্দিষ্ট অবস্থায় বা কন্ডিশনে স্টাইল প্রয়োগ করা যায়। এটি সরাসরি HTML স্ট্রাকচারে পরিবর্তন আনে না, বরং নির্দিষ্ট অবস্থায় উপাদানটি কেমন দেখাবে তা নির্ধারণ করে।

উদাহরণ:

- `:hover` : মাউস উপাদানের উপর অবস্থান করলে স্টাইল প্রয়োগ করে।

```
a:hover {
    color: red;
}
```

- `:focus` : কোনো উপাদান ফোকাস হলে (যেমন, input বক্সে ক্লিক করলে) স্টাইল প্রয়োগ করে।

```
input:focus {
    border: 2px solid blue;
}
```

- `:nth-child(n)` : নির্দিষ্ট অবস্থানে থাকা চাইল্ড উপাদানে স্টাইল প্রয়োগ করতে।

```
li:nth-child(2) {
    color: green;
}
```

2. Pseudo-elements:

Pseudo-elements ব্যবহার করে একটি উপাদানের নির্দিষ্ট অংশে স্টাইল প্রয়োগ করা যায়। এটি উপাদানের কন্টেন্টের একটি নির্দিষ্ট অংশ নির্বাচন করে, যেমন টেক্সটের প্রথম লাইন বা উপাদানের আগে বা পরে কন্টেন্ট যুক্ত করা।

উদাহরণ:

- `::before`: উপাদানের আগে কন্টেন্ট যোগ করতে।

```
p::before {
    content: "Notice: ";
    color: red;
}
```

- `::after`: উপাদানের পরে কন্টেন্ট যোগ করতে।

```
p::after {
    content: " - Read More";
    color: blue;
}
```

- `::first-line`: টেক্সটের প্রথম লাইনে স্টাইল প্রয়োগ করতে।

```
p::first-line {
    font-weight: bold;
    font-size: 1.2em;
}
```

সারাংশ:

- **Pseudo-classes**: উপাদানের নির্দিষ্ট অবস্থায় (যেমন `:hover`, `:focus`) প্রয়োগ করা হয়।
- **Pseudo-elements**: উপাদানের নির্দিষ্ট অংশে (যেমন `::before`, `::first-line`) প্রয়োগ করা হয়।

এই ধরনের সিলেক্টরগুলির মাধ্যমে আমাদের স্টাইলিং আরও ডাইনামিক ও সুনির্দিষ্ট করতে সাহায্য করে, যা ওয়েবসাইটের ব্যবহারযোগ্যতা এবং নান্দনিকতা বাড়ায়।

7. Explain the concept of inheritance in CSS.

CSS-এ **inheritance** এমন একটি কনসেপ্ট, যেখানে নির্দিষ্ট কিছু প্রপার্টি প্যারেন্ট (parent) উপাদান থেকে চাইল্ড (child) উপাদানে স্বয়ংক্রিয়ভাবে প্রযোজ্য হয়। অর্থাৎ, কিছু CSS প্রপার্টি চাইল্ড উপাদান তার প্যারেন্ট উপাদান থেকে উত্তরাধিকারসূত্রে গ্রহণ করে। এর ফলে, পুরো ওয়েবসাইটে কনসিস্টেন্ট এবং কম কোড লিখে স্টাইলিং করা সহজ হয়।

কিভাবে Inheritance কাজ করে:

CSS-এ সব প্রপার্টি ইনহেরিটেড হয় না। কিছু প্রপার্টি প্যারেন্ট উপাদান থেকে চাইল্ডে চলে যায়, আবার কিছু প্রপার্টি চলে যায় না।

Inherited প্রপার্টির উদাহরণ:

- **color** - যদি প্যারেন্টে টেক্সটের রং নির্ধারণ করা হয়, তবে তা চাইল্ড উপাদানে প্রযোজ্য হয়।
- **font-family** - প্যারেন্টের ফন্ট স্টাইল চাইল্ডে প্রযোজ্য হয়।
- **text-align** - প্যারেন্টের টেক্সট এলাইনমেন্ট চাইল্ড উপাদানে প্রযোজ্য হয়।

Non-inherited প্রপার্টির উদাহরণ:

- **border** - বর্ডার প্যারেন্ট থেকে চাইল্ডে যায় না।
- **margin** ও **padding** - এই প্রপার্টিগুলি চাইল্ড উপাদানে প্রযোজ্য হয় না।
- **width** ও **height** - এগুলোও ইনহেরিট হয় না।

Inheritance নির্ধারণ করার জন্য `inherit` কিওয়ার্ড:

কিছু প্রপার্টি ইনহেরিট না হলেও, `inherit` কিওয়ার্ড ব্যবহার করে আমরা কোনো চাইল্ড উপাদানকে তার প্যারেন্টের নির্দিষ্ট প্রপার্টি ইনহেরিট করতে বলতে পারি।

উদাহরণ:

```
div {
    color: blue;
}

p {
    color: inherit; /* প্যারেন্ট div থেকে color প্রপার্টি ইনহেরিট করবে
```

```
*/
}
```

Inheritance এর সুবিধা:

1. **কোড সহজ ও সংক্ষিপ্ত করা** - একাধিক উপাদানে একই স্টাইল দেয়ার জন্য আলাদা আলাদা স্টাইল লিখতে হয় না।
2. **Consistency** - ইনহেরিট করার মাধ্যমে পুরো ওয়েবসাইটে একই স্টাইল বজায় রাখা সহজ হয়।
3. **Maintenance সহজ হয়** - প্যারেন্টে পরিবর্তন করলেই চাইল্ডে তা আপডেট হয়, ফলে মেইনটেইন সহজ হয়।

সারসংক্ষেপ:

CSS-এর ইনহেরিটেন্স কনসেপ্ট ব্যবহার করে চাইল্ড উপাদান সহজে প্যারেন্ট থেকে প্রপার্টি গ্রহণ করতে পারে। এর মাধ্যমে কোডিং সহজ ও সুনির্দিষ্ট হয় এবং ওয়েবসাইটের স্টাইলিং আরো সুষ্ঠু ও সংগঠিত থাকে।

8. What is the `!important` rule in CSS, and when should you use it?

CSS-এ `!important` হলো একটি বিশেষ কিওয়ার্ড যা কোনো নির্দিষ্ট স্টাইলকে অন্যান্য সব স্টাইলের ওপরে প্রাধান্য দেয়ার জন্য ব্যবহৃত হয়। এটি প্রয়োগ করলে ঐ নির্দিষ্ট প্রপার্টি সব স্পেসিফিসিটি ও সোর্স অর্ডারকে অগ্রাহ্য করে, এবং সেই প্রপার্টি অপরিবর্তিত থাকে যতক্ষণ না অন্য `!important` সহ স্টাইল সেট করা হয়।

কিভাবে `!important` কাজ করে:

যখন একটি CSS প্রোপার্টিতে `!important` যুক্ত করা হয়, তখন সেই প্রপার্টি সকল অন্যান্য স্টাইলের তুলনায় সর্বোচ্চ অগ্রাধিকার পায়, এমনকি যদি স্পেসিফিসিটি কমও থাকে।

উদাহরণ:

```css
p {
    color: red !important; /* এই স্টাইলটি প্রাধান্য পাবে */
}

p {
    color: blue; /* এটি কাজ করবে না, কারণ এর উপরে !important প্রয়োগ করা হয়েছে */
}
```

এই ক্ষেত্রে `p` ট্যাগের রঙ সবসময় **লাল** হবে, কারণ `!important` সেট করা আছে।

`!important` এর সঠিক ব্যবহার:

`!important` এর ব্যবহার খুব সতর্কতার সাথে করা উচিত। কিছু ক্ষেত্রে এটি প্রয়োজন হতে পারে:

1. **কাস্টমাইজেশন** - তৃতীয় পক্ষের CSS লাইব্রেরি বা ফ্রেমওয়ার্কে স্টাইল ওভাররাইড করার জন্য `!important` ব্যবহার করা যায়।
2. **বাগ ফিক্সিং** - যেখানে স্টাইল সহজে ওভাররাইড করা যাচ্ছে না এবং স্পেসিফিসিটি কনফ্লিক্ট রয়েছে।

`!important` এর ব্যবহার এড়িয়ে চলার কারণ:

- **কোডের জটিলতা বৃদ্ধি পায়** - একাধিক `!important` স্টাইল থাকলে কোনটি প্রয়োগ হবে তা খুঁজে বের করা কঠিন হয়।
- **মেইনটেন্যান্সের সমস্যা** - কোড আপডেট করা বা ডিবাগ করা জটিল হয়ে যায়।
- **ইনহেরিটেন্স এবং স্পেসিফিসিটি সমস্যাগুলো ঠিকমতো বোঝা যায় না** - এতে ওয়েব পেজের স্টাইলিং সমস্যা হতে পারে।

সংক্ষেপে:

- `!important` একটি উচ্চ-প্রাধান্য দেয়া কিওয়ার্ড, যা নির্দিষ্ট স্টাইলকে অগ্রাধিকার দেয়।
- এটি প্রয়োজন ছাড়া ব্যবহার এড়িয়ে চলা উচিত, কারণ এটি কোডের জটিলতা বাড়ায় এবং মেইনটেইনেবল কোড লেখায় বাধা দেয়।

>>Mostly Used CSS Properties (7 questions)

1. How does the `display` property work, and what are the different display values?

CSS-এর **display** প্রপার্টি একটি উপাদানের লেআউট কেমন হবে তা নির্ধারণ করে। এর সাহায্যে উপাদানকে ব্লক, ইনলাইন বা অন্য কনটেক্সটে সাজানো যায়। বিভিন্ন ধরনের **display** ভ্যালু রয়েছে, যা উপাদান কিভাবে প্রদর্শিত হবে তা নির্ধারণ করে।

display প্রপার্টির সাধারণ ভ্যালুগুলি:

1. **display: block;**
   - উপাদানটি একটি নতুন লাইন নিয়ে থাকে, অর্থাৎ এটি পুরো লাইনের প্রস্থ গ্রহণ করে।
   - ব্লক লেভেল উপাদানগুলি সাধারণত কাস্টম উচ্চতা, প্রস্থ, মার্জিন, এবং প্যাডিং নেয়।
   - উদাহরণ: `div`, `h1`, `p`

```
div {
    display: block;
}
```

2. **display: inline;**
   - উপাদানটি একই লাইনে থাকে, এর উচ্চতা এবং প্রস্থ কাস্টমাইজ করা যায় না।
   - মার্জিন এবং প্যাডিং শুধুমাত্র অনুভূমিকভাবে কাজ করে।
   - উদাহরণ: `span`, `a`, `strong`

```
span {
    display: inline;
```

```
}
```

3. **display: inline-block;**
   - এটি ব্লক এবং ইনলাইন দুই ধরনের বৈশিষ্ট্য ধারণ করে। উপাদানটি ইনলাইন উপাদানের মত একই লাইনে থাকে, তবে উচ্চতা এবং প্রস্থ কাস্টমাইজ করা যায়।
   - মার্জিন এবং প্যাডিং উভয় দিকেই প্রযোজ্য হয়।
   - উদাহরণ:

```
button {
    display: inline-block;
}
```

4. **display: none;**
   - উপাদানটি সম্পূর্ণভাবে লেআউট থেকে সরিয়ে ফেলা হয় এবং সেটি কোনো স্থান দখল করে না।
   - এটি সাধারণত উপাদান লুকাতে বা দেখাতে ব্যবহার করা হয়।

```
.hidden {
    display: none;
}
```

5. **display: flex;**
   - উপাদানকে একটি ফ্লেক্স কন্টেইনারে পরিণত করে এবং ফ্লেক্স আইটেমগুলিকে একটি লাইনে বা কলামে সাজায়।
   - ফ্লেক্সের সাহায্যে জটিল লেআউট সহজে তৈরি করা যায়।

```
.container {
    display: flex;
}
```

6. **display: grid;**

- উপাদানকে গ্রিড কন্টেইনারে পরিণত করে এবং গ্রিড আইটেমগুলিকে সারি ও কলামে সাজানোর সুবিধা দেয়।
- গ্রিড সিস্টেম জটিল লেআউট সহজে তৈরি করে।

```
.grid-container {
    display: grid;
}
```

7. **display: table;**
    - উপাদানকে HTML টেবিলের মতো করে প্রদর্শন করে। এটি সাধারণত টেবিল লেআউট তৈরি করতে ব্যবহৃত হয়।

```
.table {
    display: table;
}
```

8. **display: list-item;**
    - এটি উপাদানকে লিস্ট আইটেমের মতো প্রদর্শন করে এবং সাধারণত `<li>` ট্যাগে ব্যবহৃত হয়।

```
li {
    display: list-item;
}
```

সারাংশ:

CSS-এর **display** প্রপার্টি বিভিন্ন ধরনের লেআউট স্টাইল নির্ধারণ করতে সহায়তা করে। প্রয়োজন অনুযায়ী সঠিক display ভ্যালু ব্যবহার করলে ওয়েবসাইটের কন্টেন্ট এবং ডিজাইন আরও সহজে কাস্টমাইজ করা যায়।

2. What is the `position` property, and how do different position values (static, relative, absolute,

fixed, sticky) behave?

CSS-এর **position** প্রপার্টি কোনো উপাদানের পজিশন কন্ট্রোল করার জন্য ব্যবহৃত হয়। এটি একটি উপাদান কীভাবে পেজে থাকবে বা এর অবস্থান কী হবে তা নির্ধারণ করে। **position** প্রপার্টির বিভিন্ন ভ্যালু আছে, প্রতিটি ভিন্ন ভিন্ন ভাবে কাজ করে।

position প্রপার্টির ভ্যালুসমূহ এবং তাদের বৈশিষ্ট্য:

1. **position: static;**
   - ডিফল্ট পজিশনিং। উপাদানটি নরমাল ফ্লো অনুসারে রেন্ডার হয় এবং এটির `top`, `right`, `bottom`, `left` প্রপার্টি কাজ করে না।
   - সাধারণত HTML ডকুমেন্টে উপাদান যেভাবে থাকে, এটি সেই অবস্থানে প্রদর্শিত হয়।

```
.box {
    position: static;
}
```

2. **position: relative;**
   - উপাদানটি ডিফল্ট পজিশনের উপর ভিত্তি করে থাকে, তবে `top`, `right`, `bottom`, `left` প্রপার্টি ব্যবহার করে সেটি নড়ানো যায়।
   - এটি মূলত উপাদানটিকে স্থানান্তরিত করতে ব্যবহার করা হয়, কিন্তু মূল অবস্থানের জন্য জায়গা ধরে রাখে।

```
.box {
    position: relative;
    top: 10px;
    left: 15px;
}
```

3. **position: absolute;**
   - উপাদানটি তার নিকটস্থ পজিশনড (non-static) প্যারেন্টের তুলনায় স্থানান্তরিত হয়। যদি কোনো পজিশনড প্যারেন্ট না থাকে, তবে এটি পুরো ডকুমেন্ট বডির তুলনায় অবস্থান নেবে।
   - উপাদানটি নরমাল ফ্লো থেকে বাদ পড়ে, তাই এটি অন্য উপাদানকে প্রভাবিত করে না।

```css
.box {
    position: absolute;
    top: 20px;
    right: 30px;
}
```

4. **position: fixed;**
    - উপাদানটি ভিউপোর্টের তুলনায় নির্দিষ্ট করা হয়, অর্থাৎ স্ক্রোল করার সময়ও এটি একই জায়গায় থাকে।
    - এটি নরমাল ফ্লো থেকে বাদ পড়ে, তাই অন্য উপাদানকে প্রভাবিত করে না।
    - সাধারণত নেভিগেশন বার বা নোটিফিকেশন পপ-আপের জন্য ব্যবহৃত হয়।

```css
.box {
    position: fixed;
    bottom: 0;
    right: 0;
}
```

5. **position: sticky;**
    - উপাদানটি স্ক্রোল করা পর্যন্ত নরমাল ফ্লোতে থাকে, কিন্তু একটি নির্দিষ্ট পজিশনে পৌঁছালে ভিউপোর্টে আটকে যায় এবং সেখানে স্থির থাকে।
    - `top`, `right`, `bottom`, বা `left` ভ্যালু ব্যবহার করে আটকে যাওয়ার পজিশন নির্ধারণ করা যায়।
    - সাধারণত নেভিগেশন মেনু বা হেডারের জন্য ব্যবহৃত হয়।

```css
.box {
    position: sticky;
    top: 10px;
}
```

সারাংশ:

- **static**: ডিফল্ট পজিশনিং, কোনো অবস্থান পরিবর্তন হয় না।
- **relative**: নিজ অবস্থানের সাপেক্ষে স্থানান্তরিত হয়, কিন্তু নরমাল ফ্লোতে থাকে।
- **absolute**: নিকটস্থ পজিশনড প্যারেন্ট বা বডির তুলনায় স্থানান্তরিত হয়, নরমাল ফ্লো থেকে বাদ পড়ে।
- **fixed**: ভিউপোর্টের তুলনায় স্থির থাকে এবং স্ক্রোল করলেও পরিবর্তন হয় না।
- **sticky**: স্ক্রোলের সাথে নড়ে কিন্তু একটি নির্দিষ্ট পজিশনে গিয়ে আটকে যায়।

এই ভিন্ন ভিন্ন পজিশন ভ্যালুগুলি বিভিন্ন উপাদানের জন্য প্রয়োজনীয় লেআউট তৈরি করতে এবং সুনির্দিষ্টভাবে কন্টেন্ট প্রদর্শন করতে ব্যবহার করা হয়।

3. How do the `padding`, `margin`, and `border` properties work in CSS?

CSS-এ **padding**, **margin**, এবং **border** প্রপার্টিগুলি একটি উপাদানের লেআউট এবং স্থান নির্ধারণে অত্যন্ত গুরুত্বপূর্ণ। এগুলি উপাদানের চারপাশে বিভিন্ন স্তরের স্থান তৈরি করে এবং এটি কীভাবে প্রদর্শিত হবে তা নিয়ন্ত্রণ করে।

1. Padding:

- **Padding** হলো উপাদানের কন্টেন্ট এবং বর্ডারের মধ্যে স্থান। এটি উপাদানের ভিতরের স্থান তৈরি করে এবং কন্টেন্টের চারপাশে "ফাঁকা" জায়গা রাখে।
- এটি সর্বদা কন্টেন্টের সাথে যুক্ত থাকে এবং এর আকার কন্টেন্টের আকারের সাথে যুক্ত থাকে।
- **padding** প্রপার্টির মান পিক্সেল, ইম, শতাংশ ইত্যাদিতে নির্ধারণ করা যায়।

উদাহরণ:

```
.box {
    padding: 20px; /* সব দিকেই 20px padding */
}
```

2. Border:

- **Border** হলো উপাদানের কন্টেন্ট এবং মার্জিনের মধ্যে একটি রেখা। এটি উপাদানের সীমানা তৈরি করে।

- বর্ডারের বিভিন্ন প্রকার (যেমন, সলিড, ড্যাশড, ডটেড) এবং বিভিন্ন প্রস্থ, রঙ নির্ধারণ করা যায়।

উদাহরণ:

```
.box {
    border: 2px solid black; /* 2px প্রস্থের সলিড কালো বর্ডার */
}
```

3. Margin:

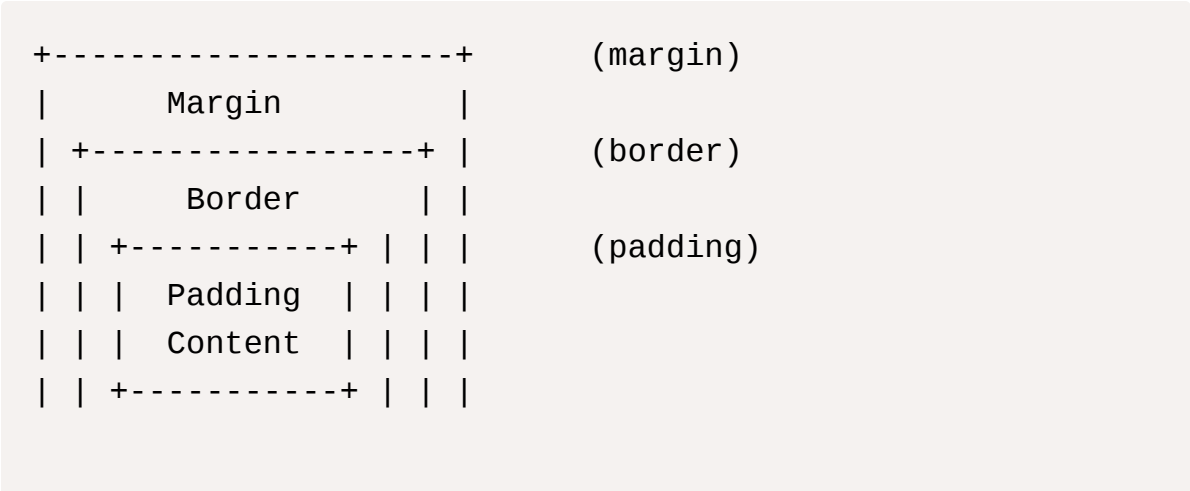
- **Margin** হলো উপাদানের বাইরের স্থান, যা অন্য উপাদানের সাথে দূরত্ব নির্ধারণ করে। এটি উপাদানের বাইরের অংশে "ফাঁকা" জায়গা তৈরি করে।
- মার্জিনস কন্টেন্ট বা বর্ডারের বাইরের অংশের জন্য কাজ করে এবং এটি অন্যান্য উপাদানের সাথে ব্যবধান তৈরি করে।
- **margin** প্রপার্টির মানও পিক্সেল, ইম, শতাংশ ইত্যাদিতে নির্ধারণ করা যায়।

উদাহরণ:

```
.box {
    margin: 15px; /* সব দিকেই 15px margin */
}
```

Padding, Border, Margin এর মধ্যে সম্পর্ক:

- **Box Model**: padding, border, এবং margin একসাথে একটি উপাদানের বক্স মডেল তৈরি করে। এটি উপাদানের মোট প্রস্থ এবং উচ্চতা নির্ধারণে সাহায্য করে।



```
| +------------------+ |
+----------------------+
```

সারাংশ:

- **Padding**: উপাদানের কন্টেন্ট এবং বর্ডারের মধ্যে স্থান। এটি কন্টেন্টের ভিতরে "ফাঁকা" জায়গা তৈরি করে।
- **Border**: উপাদানের কন্টেন্ট এবং মার্জিনের মধ্যে রেখা। এটি উপাদানের সীমানা তৈরি করে।
- **Margin**: উপাদানের বাইরের স্থান। এটি অন্যান্য উপাদানের সাথে দূরত্ব নির্ধারণ করে।

এই প্রপার্টিগুলির সঠিক ব্যবহার করে ওয়েব পেজের কন্টেন্টের দৃশ্যমানতা এবং স্থান সঠিকভাবে নিয়ন্ত্রণ করা যায়।

4. Explain the `flex` property. How do `flex-grow`, `flex-shrink`, and `flex-basis` work?

CSS-এর **flex** প্রপার্টি একটি ফ্লেক্স কন্টেইনারের মধ্যে ফ্লেক্স আইটেমগুলির সাইজ এবং বিতরণ নিয়ন্ত্রণ করতে ব্যবহৃত হয়। এটি লেআউট ডিজাইনকে আরও সুষম এবং প্রতিক্রিয়াশীল (responsive) করতে সাহায্য করে। **flex** প্রপার্টিটি তিনটি প্রধান অংশ নিয়ে গঠিত: **flex-grow**, **flex-shrink**, এবং **flex-basis**।

flex-grow

- **flex-grow** একটি ফ্লেক্স আইটেমকে তার প্যারেন্ট ফ্লেক্স কন্টেইনারের মধ্যে যে পরিমাণ স্থান পাওয়া যায়, সেই অনুযায়ী প্রসারিত হতে দেয়।
- এর মান **0** থেকে **∞** এর মধ্যে হতে পারে। যদি একটি ফ্লেক্স আইটেমের **flex-grow** এর মান **0** হয়, তবে এটি অতিরিক্ত স্থান গ্রহণ করবে না। যদি এটি **1** হয়, তবে এটি যে পরিমাণ অতিরিক্ত স্থান পাওয়া যায়, তা গ্রহণ করবে।

উদাহরণ:

```css
.container {
    display: flex;
}
```

```css
.item1 {
    flex-grow: 1; /* এটি অতিরিক্ত স্থান গ্রহণ করবে */
}

.item2 {
    flex-grow: 2; /* এটি item1 এর চেয়ে দ্বিগুণ অতিরিক্ত স্থান গ্রহণ কর
বে */
}
```

flex-shrink

- **flex-shrink** একটি ফ্লেক্স আইটেমের সঙ্কুচিত (shrink) হওয়ার ক্ষমতা নিয়ন্ত্রণ করে যখন ফ্লেক্স কন্টেইনারের ভিতরে স্থান কমে যায়।
- এর মানও **0** থেকে **∞** এর মধ্যে হতে পারে। যদি **flex-shrink** এর মান **0** হয়, তবে আইটেমটি সংকুচিত হবে না। যদি এটি **1** হয়, তবে এটি অটোমেটিক্যালি সংকুচিত হবে, অন্য আইটেমগুলির তুলনায় যা বেশি সঙ্কুচিত হতে পারে।

উদাহরণ:

```css
.container {
    display: flex;
}

.item1 {
    flex-shrink: 1; /* এটি সংকুচিত হতে পারে */
}

.item2 {
    flex-shrink: 0; /* এটি সংকুচিত হবে না */
}
```

flex-basis

- **flex-basis** নির্ধারণ করে একটি ফ্লেক্স আইটেমের প্রাথমিক আকার (বেস আকার)। এটি আইটেমটির আসল আকার আগে থেকেই কিভাবে থাকবে তা নির্দেশ করে।

- এর মান পিক্সেল, শতাংশ, অথবা অটো হতে পারে। **flex-basis** মূলত আইটেমটির মূল সাইজ বোঝায় যা অন্য সব প্রপার্টির সাথে কার্যকর হয়।

উদাহরণ:

```
.container {
    display: flex;
}

.item1 {
    flex-basis: 100px; /* প্রাথমিক আকার 100px */
}

.item2 {
    flex-basis: 200px; /* প্রাথমিক আকার 200px */
}
```

সারসংক্ষেপ:

flex প্রপার্টিটি ফ্লেক্স কন্টেইনারের মধ্যে ফ্লেক্স আইটেমগুলির আচরণ নিয়ন্ত্রণ করে:

- **flex-grow**: অতিরিক্ত স্থান নেওয়ার ক্ষমতা নির্ধারণ করে।
- **flex-shrink**: সংকুচিত হওয়ার ক্ষমতা নির্ধারণ করে।
- **flex-basis**: প্রাথমিক আকার নির্ধারণ করে।

এই তিনটি প্রপার্টি মিলিয়ে একটি সুষম এবং প্রতিক্রিয়াশীল লেআউট তৈরি করতে সাহায্য করে। Flexbox ব্যবহারের মাধ্যমে বিভিন্ন ডিভাইসে এবং স্ক্রীনের আকারে কন্টেন্ট সুন্দরভাবে সাজানো যায়।

5. What is the difference between `width`, `min-width`, and `max-width`?

CSS-এ **width**, **min-width**, এবং **max-width** প্রপার্টিগুলি একটি উপাদানের প্রস্থ নিয়ন্ত্রণ করার জন্য ব্যবহৃত হয়, কিন্তু এগুলোর উদ্দেশ্য এবং ব্যবহার ভিন্ন।

1. Width:

- **width** প্রপার্টি নির্ধারণ করে একটি উপাদানের সাধারণ প্রস্থ। এটি নির্দিষ্ট পিক্সেল, শতাংশ, ইম, অথবা অন্য যে কোনো ইউনিটে সেট করা যেতে পারে।
- যদি **width** একটি নির্দিষ্ট মান সেট করা হয়, তবে উপাদানটি সেই মান অনুসারে প্রস্থ নেবে।

উদাহরণ:

```
.box {
    width: 300px; /* উপাদানটির প্রস্থ 300px হবে */
}
```

2. Min-width:

- **min-width** প্রপার্টি একটি উপাদানের জন্য সর্বনিম্ন প্রস্থ নির্ধারণ করে। এর মানে হলো, উপাদানটি কখনই এই প্রস্থের কম হবে না।
- যখন কন্টেন্ট বা কন্টেইনারের আকার পরিবর্তিত হয়, তবে উপাদানটি এই নির্ধারিত প্রস্থে থাকবেই, যদিও প্রস্থের অন্য প্রপার্টি দ্বারা সেট করা হয়েছে।

উদাহরণ:

```
.box {
    min-width: 200px; /* উপাদানটির প্রস্থ কখনই 200px এর কম হবে না
*/
}
```

3. Max-width:

- **max-width** প্রপার্টি একটি উপাদানের জন্য সর্বাধিক প্রস্থ নির্ধারণ করে। এর মানে হলো, উপাদানটি এই প্রস্থের বেশি হবে না।
- যখন কন্টেন্ট বা কন্টেইনারের আকার পরিবর্তিত হয়, তবে উপাদানটি এই নির্ধারিত প্রস্থের মধ্যে থাকবে এবং এর প্রস্থ এই সীমার মধ্যে থাকবে।

উদাহরণ:

```
.box {
    max-width: 500px; /* উপাদানটির প্রস্থ কখনই 500px এর বেশি হবে না
```

```
*/
}
```

সারসংক্ষেপ:

- **width**: উপাদানের প্রস্থ নির্ধারণ করে।
- **min-width**: উপাদানের জন্য সর্বনিম্ন প্রস্থ নির্ধারণ করে; উপাদানটি এই মানের নিচে যাবে না।
- **max-width**: উপাদানের জন্য সর্বাধিক প্রস্থ নির্ধারণ করে; উপাদানটি এই মানের উপরে যাবে না।

এই প্রপার্টিগুলির সঠিক ব্যবহার নিশ্চিত করে যে উপাদানগুলি বিভিন্ন স্ক্রীনে এবং ডিভাইসে সঠিকভাবে প্রদর্শিত হবে।

6. How does the `z-index` property work, and when would you use it?

CSS-এর **z-index** প্রপার্টি একটি উপাদানের স্তরের গভীরতা (stacking order) নির্ধারণ করতে ব্যবহৃত হয়। এটি একটি উপাদানকে অন্য উপাদানের উপরে বা নিচে অবস্থান করতে সাহায্য করে, বিশেষ করে যখন উপাদানগুলি একে অপরের উপরে অবস্থান করে (overlapping)।

z-index প্রপার্টির কাজ:

1. **Stacking Context**:
   - **z-index** শুধুমাত্র পজিশনড উপাদানের (যেমন, `position: relative;`, `position: absolute;`, `position: fixed;`, বা `position: sticky;` ব্যবহার করা হলে) উপর কাজ করে। যদি একটি উপাদানে **z-index** নির্ধারণ করা না হয়, তবে এটি তার প্যারেন্ট উপাদানের stacking context অনুসারে আচরণ করবে।
2. **Numeric Value**:
   - **z-index** এর মান সাধারণত পূর্ণসংখ্যা হতে হয়, যেমন 1, 2, 3, ইত্যাদি। উচ্চতর মানের উপাদানগুলি কম মানের উপাদানের উপরে প্রদর্শিত হবে।
   - যদি দুটি উপাদানের z-index সমান হয়, তবে তারা তাদের ডকুমেন্টের কার্যকরী অর্ডার অনুসারে স্তরে অবস্থান করবে; যার প্রথমে ডিফাইন করা হয় সেটি উপরে থাকবে।

উদাহরণ:

```css
.box1 {
    position: absolute;
    z-index: 1; /* নিম্ন স্তরে */
}

.box2 {
    position: absolute;
    z-index: 2; /* উচ্চ স্তরে, box1 এর উপরে প্রদর্শিত হবে */
}
```

কিভাবে z-index ব্যবহার করবেন:

- **Overlay Elements**: যখন আপনি একটি লেয়ারিং ইফেক্ট তৈরি করতে চান, যেমন মডাল উইন্ডো, টুলটিপ, বা নেভিগেশন মেনু, তখন **z-index** ব্যবহার করা হয়।
- **Custom Layouts**: যেকোনো কাস্টম লেআউট যেখানে বিভিন্ন উপাদান একে অপরের উপরে ও নিচে থাকতে পারে।

কখন z-index ব্যবহার করবেন:

- যখন একাধিক উপাদান একে অপরের উপরে অবস্থান করছে এবং আপনি সঠিক স্তর নিশ্চিত করতে চান।
- যখন একটি উপাদান (যেমন একটি পপআপ বা মডাল) অপর একটি উপাদানের উপরে প্রদর্শিত হওয়া প্রয়োজন।
- যখন অস্বাভাবিক লেআউট ডিজাইন করা হচ্ছে এবং উপাদানগুলি overlapping করতে হচ্ছে।

সতর্কতা:

- **z-index** প্রপার্টি শুধুমাত্র পজিশনড উপাদানের জন্য কার্যকর, তাই এটি ব্যবহারের আগে নিশ্চিত করুন যে উপাদানটি যথাযথভাবে পজিশনড করা আছে।
- অতিরিক্ত **z-index** মানের ব্যবহার আপনার CSS কোডকে জটিল করে তুলতে পারে, তাই এটি সঠিকভাবে এবং সংক্ষিপ্তভাবে ব্যবহার করা উচিত।

সারাংশ:

z-index প্রপার্টি একটি উপাদানের স্তরের গভীরতা নির্ধারণ করতে সাহায্য করে, যা overlapping উপাদানগুলির মধ্যে সঠিকভাবে সাজানো এবং প্রদর্শনের জন্য গুরুত্বপূর্ণ। সঠিকভাবে ব্যবহার করা

হলে, এটি ডিজাইনে পরিস্কারতা এবং ব্যবহারযোগ্যতা বাড়াতে পারে।

7. What is the difference between the `opacity` and `visibility` properties?

CSS-এ **opacity** এবং **visibility** প্রপার্টিগুলি একটি উপাদানের দৃশ্যমানতা নিয়ন্ত্রণ করতে ব্যবহৃত হয়, তবে তাদের কাজ এবং প্রভাব ভিন্ন।

1. Opacity:

- **opacity** প্রপার্টি একটি উপাদানের স্বচ্ছতা নিয়ন্ত্রণ করে। এর মান 0 থেকে 1 এর মধ্যে হতে পারে, যেখানে 0 মানে সম্পূর্ণ অদৃশ্য (transparent) এবং 1 মানে সম্পূর্ণ দৃশ্যমান (opaque)।
- যখন opacity কমানো হয়, উপাদানটি অদৃশ্য হলেও এর স্থান এবং প্রভাব ডকুমেন্টের ফ্লোতে বজায় থাকে। এটি অন্যান্য উপাদানের উপর স্থান ধরে রাখে।

উদাহরণ:

```
.box {
    opacity: 0.5; /* উপাদানটি 50% স্বচ্ছ, অর্ধেক দৃশ্যমান */
}
```

2. Visibility:

- **visibility** প্রপার্টি একটি উপাদানের দৃশ্যমানতা নিয়ন্ত্রণ করে। এর দুটি মূল মান রয়েছে: `visible` এবং `hidden`। `visible` মানে উপাদান দৃশ্যমান, আর `hidden` মানে উপাদানটি অদৃশ্য, কিন্তু এটি এখনও ডকুমেন্টের ফ্লোতে অবস্থান ধরে রাখে।
- `hidden` অবস্থায়, উপাদানটি দৃশ্যমান নয়, কিন্তু স্থান বরাদ্দ বজায় থাকে। এটি অন্যান্য উপাদানের উপর স্থান ধরে রাখে।

উদাহরণ:

```
.box {
    visibility: hidden; /* উপাদানটি অদৃশ্য, কিন্তু স্থান ধরে রাখে */
}
```

প্রধান পার্থক্য:

বৈশিষ্ট্য	Opacity	Visibility
প্রভাব	উপাদানের স্বচ্ছতা নিয়ন্ত্রণ করে।	উপাদানের দৃশ্যমানতা নিয়ন্ত্রণ করে।
মান	0 (অদৃশ্য) থেকে 1 (দৃশ্যমান) পর্যন্ত।	`visible` এবং `hidden`।
স্থান	স্থান বজায় থাকে, তবে দৃশ্যমানতা কমে।	স্থান বজায় থাকে, কিন্তু দৃশ্যমান নয়।
ইনটারঅ্যাকশন	ইউজার ইনপুট গ্রহণ করতে পারে (যদিও দেখা যাচ্ছে না)।	ইউজার ইনপুট গ্রহণ করতে পারে না।

সারসংক্ষেপ:

- **opacity** প্রপার্টি একটি উপাদানকে স্বচ্ছ করে এবং এর মান 0 থেকে 1 এর মধ্যে পরিবর্তিত হয়। এটি দৃশ্যমানতা কমানোর পাশাপাশি ইনপুট গ্রহণ করার ক্ষমতাও বজায় রাখে।
- **visibility** প্রপার্টি একটি উপাদানকে সম্পূর্ণরূপে অদৃশ্য করে এবং এর মান `visible` এবং `hidden` এর মধ্যে সীমাবদ্ধ। এটি অদৃশ্য অবস্থায় স্থান ধরে রাখে, কিন্তু ইনপুট গ্রহণ করতে পারে না।

এই প্রপার্টিগুলি ব্যবহার করে, আপনি আপনার ডিজাইনে নির্দিষ্ট প্রভাব তৈরি করতে পারেন।

>> CSS Layouts & Responsiveness (8 questions)

1. What is the difference between `flexbox` and `CSS Grid`, and when should you use each?

Flexbox এবং **CSS Grid** উভয়ই CSS-এর লেআউট মডিউল, তবে তাদের উদ্দেশ্য, কাজ করার পদ্ধতি এবং ব্যবহারের ক্ষেত্রে কিছু মৌলিক পার্থক্য রয়েছে। নিচে এই দুটি প্রযুক্তির মধ্যে পার্থক্য এবং

কোন পরিস্থিতিতে কী ব্যবহার করবেন তা আলোচনা করা হলো।

Flexbox:

- **কাজের পদ্ধতি**: Flexbox একমাত্র একমাত্র ডাইমেনশনে (অর্থাৎ, এক্স বা ওয়াই) কাজ করে। এটি মূলত একলাইন লেআউটের জন্য ডিজাইন করা হয়েছে এবং উপাদানগুলিকে একে অপরের সাথে স্থান দেওয়ার জন্য ব্যবহার করা হয়।
- **মুখ্য উদ্দেশ্য**: এটি প্রধানত একটি বা একাধিক লাইনের মধ্যে উপাদানগুলির বিতরণ এবং অ্যালাইনমেন্টের জন্য ব্যবহার হয়। এটি আর্কষণীয়ভাবে স্থানীয় বা ভিন্ন আকারের উপাদানগুলির ব্যবস্থাপনায় সহায়ক।
- **উপাদানের সংহতি**: Flexbox উপাদানগুলি খুব সহজে আকার পরিবর্তন করতে পারে এবং এটি স্বয়ংক্রিয়ভাবে স্থান পূরণ করে, যা উপাদানগুলির মধ্যে স্পেসিং সঠিকভাবে সেট করতে সহায়ক।

ব্যবহার উদাহরণ:

- ন্যাভিগেশন মেনু
- ফর্ম লেআউট
- কার্ড ডিজাইন

CSS Grid:

- **কাজের পদ্ধতি**: CSS Grid একাধিক ডাইমেনশনে (দুই ডাইমেনশন) কাজ করে, অর্থাৎ এটি কলাম এবং রো উভয়ই সন্নিবেশ করতে সক্ষম। এটি পুরো লেআউটকে গ্রিডের আকারে তৈরি করে এবং খুবই শক্তিশালী।
- **মুখ্য উদ্দেশ্য**: এটি একটি জটিল লেআউট তৈরি করতে ব্যবহৃত হয় যেখানে আপনি বিভিন্ন উপাদানকে নির্দিষ্ট জায়গায় রাখতে পারেন। এটি একটি ফ্লেক্সিবল এবং অত্যন্ত নিয়ন্ত্রণযোগ্য লেআউট তৈরি করতে সহায়ক।
- **উপাদানের সংহতি**: Grid ব্যবহারে, আপনি গ্রিডের মধ্যে উপাদানগুলি সঠিকভাবে কোথায় স্থান দেবেন তা নির্ধারণ করতে পারেন, এবং এটি খুব বেশি উপাদান বা জটিল লেআউটের জন্য উপযুক্ত।

ব্যবহার উদাহরণ:

- ওয়েব পেজের সম্পূর্ণ লেআউট
- ড্যাশবোর্ড ডিজাইন

- গ্যালারী লেআউট

কখন কী ব্যবহার করবেন:

- **Flexbox ব্যবহার করুন**:
  - যখন আপনি একটি একক রো বা কলামের লেআউট তৈরি করতে চান।
  - যখন উপাদানগুলির মধ্যে স্থান এবং অ্যালাইনমেন্ট নিয়ন্ত্রণ করতে চান।
  - যখন আপনার উপাদানগুলির আকার পরিবর্তনযোগ্য, কিন্তু তাদের প্রধানত এক লাইনে সাজাতে চান।
- **CSS Grid ব্যবহার করুন**:
  - যখন আপনার একটি সম্পূর্ণ লেআউট তৈরি করার প্রয়োজন।
  - যখন বিভিন্ন কলাম এবং রোতে উপাদানগুলিকে সঠিকভাবে সাজানোর প্রয়োজন।
  - যখন আপনি একটি জটিল লেআউট তৈরি করতে চান যেখানে উপাদানগুলির মধ্যে সংযোগ থাকতে পারে।

সারসংক্ষেপ:

Flexbox এবং **CSS Grid** উভয়ই আলাদা উদ্দেশ্যে ডিজাইন করা হয়েছে। Flexbox সাধারণত একটি একক দিকের লেআউটের জন্য এবং উপাদানগুলির বিতরণ এবং অ্যালাইনমেন্টের জন্য ব্যবহৃত হয়, যেখানে CSS Grid একটি জটিল এবং বহু দিকের লেআউট তৈরি করতে সহায়ক। আপনার প্রকল্পের প্রয়োজনীয়তার উপর ভিত্তি করে সঠিক টুলটি নির্বাচন করুন।

2. Explain how to create a responsive layout using media queries.

CSS-এ **media queries** ব্যবহার করে একটি রেসপন্সিভ লেআউট তৈরি করা যায়। এটি ডিভাইসের স্ক্রীনের আকার এবং রেজোলিউশনের উপর ভিত্তি করে বিভিন্ন স্টাইল প্রয়োগ করার অনুমতি দেয়। নিচে মিডিয়া কোয়েরি ব্যবহার করে একটি রেসপন্সিভ লেআউট তৈরি করার পদক্ষেপগুলো তুলে ধরা হলো:

1. বেস স্টাইল সেট আপ করুন:

প্রথমে, আপনার CSS-এ একটি বেস লেআউট তৈরি করুন যা সকল ডিভাইসের জন্য কাজ করবে। এখানে, আপনি সাধারণভাবে লেআউট, টাইপোগ্রাফি, রঙ এবং অন্যান্য মৌলিক স্টাইল নির্ধারণ করবেন।

উদাহরণ:

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

.container {
    display: flex;
    flex-direction: column;
    padding: 20px;
}

.item {
    background-color: lightblue;
    margin: 10px;
    padding: 20px;
    border: 1px solid #ccc;
}
```

2. মিডিয়া কোয়েরি যোগ করুন:

এরপর, মিডিয়া কোয়েরি ব্যবহার করে বিভিন্ন স্ক্রীনের আকারে স্টাইল পরিবর্তন করুন। একটি মিডিয়া কোয়েরি নিম্নলিখিতভাবে দেখতে হয়:

```
@media (condition) {
    /* CSS rules */
}
```

3. স্ক্রীন সাইজ অনুযায়ী স্টাইল পরিবর্তন করুন:

বিভিন্ন ডিভাইসের জন্য স্টাইলিং পরিবর্তন করতে মিডিয়া কোয়েরি ব্যবহার করুন। যেমন, একটি সাধারণ উদাহরণে আপনি ট্যাবলেট এবং মোবাইল ডিভাইসের জন্য আলাদা লেআউট তৈরি করতে পারেন।

উদাহরণ:

```
/* মোবাইল ডিভাইসের জন্য */
@media (max-width: 600px) {
    .container {
        flex-direction: column;
    }

    .item {
        margin: 5px 0;
    }
}

/* ট্যাবলেট ডিভাইসের জন্য */
@media (min-width: 601px) and (max-width: 900px) {
    .container {
        flex-direction: row;
        flex-wrap: wrap;
    }

    .item {
        flex: 1 1 calc(50% - 20px); /* দুইটি আইটেম এক লাইনে */
    }
}

/* ডেস্কটপের জন্য */
@media (min-width: 901px) {
    .container {
        flex-direction: row;
    }

    .item {
```

```
            flex: 1 1 calc(33% - 20px); /* তিনটি আইটেম এক লাইনে */
        }
    }
```

4. স্ক্রীন রেজোলিউশনের ভিত্তিতে নিয়ন্ত্রণ:

উপরের উদাহরণে, মিডিয়া কোয়েরি মোবাইল (600px এর নিচে), ট্যাবলেট (601px থেকে 900px), এবং ডেস্কটপ (901px এর উপরে) এর জন্য স্টাইল পরিবর্তন করছে।

5. সর্বশেষ রেসপন্সিভ লেআউট:

এখন, আপনার তৈরি করা মিডিয়া কোয়েরিগুলি বিভিন্ন ডিভাইসে সঠিকভাবে কাজ করবে, নিশ্চিত করে যে আপনার ওয়েবসাইট বা অ্যাপ্লিকেশন সকল স্ক্রীনে সঠিকভাবে প্রদর্শিত হচ্ছে।

সারসংক্ষেপ:

- মিডিয়া কোয়েরি ব্যবহার করে রেসপন্সিভ ডিজাইন তৈরি করতে, প্রথমে একটি বেস স্টাইল তৈরি করুন এবং পরে বিভিন্ন স্ক্রীন সাইজের জন্য আলাদা মিডিয়া কোয়েরি ব্যবহার করে কাস্টম স্টাইল প্রয়োগ করুন।
- মিডিয়া কোয়েরি ব্যবহার করে, আপনি নিশ্চিত করতে পারেন যে আপনার লেআউট বিভিন্ন ডিভাইসে সঠিকভাবে কাজ করছে এবং ব্যবহারকারীর অভিজ্ঞতাকে উন্নত করছে।

3. How do you create a centered element in CSS? Provide multiple ways.

CSS-এ একটি কেন্দ্রীভূত উপাদান তৈরি করার বিভিন্ন পদ্ধতি রয়েছে। নিচে কয়েকটি সাধারণ এবং কার্যকর উপায় বর্ণনা করা হলো:

1. Flexbox ব্যবহার করে:

Flexbox একটি সহজ এবং কার্যকর পদ্ধতি উপাদানকে কেন্দ্রীভূত করতে। এখানে একটি উদাহরণ দেওয়া হলো:

```
.container {
    display: flex;
```

```
    justify-content: center; /* অনুভূমিক কেন্দ্রীভূত */
    align-items: center; /* উল্লম্ব কেন্দ্রীভূত */
    height: 100vh; /* পুরো viewport উচ্চতা */
}

.box {
    width: 200px;
    height: 100px;
    background-color: lightblue;
}
```

```
<div class="container">
    <div class="box">Centered Box</div>
</div>
```

2. Grid Layout ব্যবহার করে:

CSS Grid ব্যবহার করেও উপাদান কেন্দ্রীভূত করা যায়। উদাহরণ:

```
.container {
    display: grid;
    place-items: center; /* অনুভূমিক ও উল্লম্বভাবে কেন্দ্রীভূত */
    height: 100vh; /* পুরো viewport উচ্চতা */
}

.box {
    width: 200px;
    height: 100px;
    background-color: lightgreen;
}
```

```
<div class="container">
    <div class="box">Centered Box</div>
</div>
```

3. মার্জিন ব্যবহার করে:

মার্জিন ব্যবহার করে কেন্দ্রীভূত করা একটি প্রচলিত পদ্ধতি। এখানে `margin: auto` ব্যবহার করে উপাদানকে কেন্দ্রীভূত করা হয়েছে:

```
.box {
    width: 200px;
    height: 100px;
    background-color: lightcoral;
    margin: auto; /* অনুভূমিক কেন্দ্রীভূত */
}
```

```
<div class="container" style="height: 100vh;">
    <div class="box">Centered Box</div>
</div>
```

4. পজিশনিং ব্যবহার করে:

`position` প্রপার্টি ব্যবহার করেও একটি উপাদানকে কেন্দ্রীভূত করা যায়। নিচে একটি উদাহরণ:

```
.container {
    position: relative;
    height: 100vh; /* পুরো viewport উচ্চতা */
}

.box {
    position: absolute;
    top: 50%; /* কেন্দ্রের দিকে */
    left: 50%; /* কেন্দ্রের দিকে */
    transform: translate(-50%, -50%); /* সঠিকভাবে কেন্দ্রীভূত */
    width: 200px;
    height: 100px;
    background-color: lightyellow;
}
```

```html
<div class="container">
    <div class="box">Centered Box</div>
</div>
```

5. টেক্সট কেন্দ্রীভূত করা:

যদি আপনি শুধুমাত্র টেক্সট কেন্দ্রীভূত করতে চান, তাহলে `text-align` প্রপার্টি ব্যবহার করতে পারেন:

```css
.container {
    text-align: center; /* টেক্সটকে কেন্দ্রীভূত */
}

.box {
    display: inline-block; /* ব্লক এলিমেন্টকে inline-block করুন */
    width: 200px;
    height: 100px;
    background-color: lightpink;
}
```

```html
<div class="container">
    <div class="box">Centered Box</div>
</div>
```

সারসংক্ষেপ:

উপাদানকে কেন্দ্রীভূত করতে CSS-এ বিভিন্ন পদ্ধতি রয়েছে, যেমন Flexbox, Grid, মার্জিন, পজিশনিং, এবং টেক্সট কেন্দ্রীভূত করার প্রক্রিয়া। আপনি আপনার প্রয়োজন অনুযায়ী যেকোনো একটি পদ্ধতি বেছে নিতে পারেন।

4. How does CSS Grid work, and what are `grid-template-rows` and `grid-template-columns`?

CSS Grid একটি শক্তিশালী লেআউট মডিউল যা ডেভেলপারদের জন্য একটি ডুয়াল-ডাইমেনশনাল লেআউট তৈরি করা সহজ করে। এটি ব্যবহারকারীদেরকে একটি গ্রিডের মাধ্যমে উপাদানগুলিকে সাজানোর জন্য প্রয়োজনীয় নিয়ন্ত্রণ প্রদান করে, যা কলাম এবং রো উভয়ই নিয়ে কাজ করে।

CSS Grid এর মূল ধারণা:

1. **Grid Container**: একটি গ্রিড কন্টেইনার হল একটি উপাদান যা গ্রিড হিসেবে কাজ করবে। এটি `display: grid;` প্রপার্টি দিয়ে সেট করা হয়।
2. **Grid Items**: কন্টেইনারের অভ্যন্তরে থাকা উপাদানগুলি হল গ্রিড আইটেম।
3. **Grid Lines**: গ্রিডের রেখাগুলি যা রো এবং কলামের মধ্যে বিভাজক হিসেবে কাজ করে।
4. **Grid Areas**: গ্রিডের মধ্যে এক বা একাধিক গ্রিড আইটেমের ক্ষেত্র।

grid-template-rows এবং grid-template-columns:

`grid-template-rows` এবং `grid-template-columns` হল দুইটি গুরুত্বপূর্ণ প্রপার্টি যা গ্রিডের রো এবং কলামের আকার নির্ধারণ করতে ব্যবহৃত হয়।

1. grid-template-rows:

`grid-template-rows` প্রপার্টি গ্রিড কন্টেইনারের জন্য রো-এর উচ্চতা নির্ধারণ করে। আপনি এর মান হিসাবে একাধিক উচ্চতা নির্দেশ করতে পারেন।

উদাহরণ:

```
.container {
    display: grid;
    grid-template-rows: 100px 200px 150px; /* তিনটি রো: 100px, 200px, 150px উচ্চতা */
}
```

2. grid-template-columns:

`grid-template-columns` প্রপার্টি গ্রিড কন্টেইনারের জন্য কলামের প্রস্থ নির্ধারণ করে। এর মান হিসাবে বিভিন্ন প্রস্থ নির্দেশ করতে পারেন।

উদাহরণ:

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr; /* তিনটি কলাম: প্রথম এবং তৃতীয় 1fr, দ্বিতীয় 2fr */
}
```

এখানে `fr` (fraction) ইউনিট ব্যবহার করা হয়েছে, যা উপলব্ধ স্থানকে ভাগ করে দেয়।

উদাহরণ:

নিচে একটি সম্পূর্ণ উদাহরণ দেওয়া হলো যেখানে CSS Grid ব্যবহার করা হয়েছে:

```
<div class="container">
    <div class="item1">Item 1</div>
    <div class="item2">Item 2</div>
    <div class="item3">Item 3</div>
    <div class="item4">Item 4</div>
</div>
```

```
.container {
    display: grid;
    grid-template-columns: 1fr 2fr; /* 2টি কলাম */
    grid-template-rows: 100px 200px; /* 2টি রো */
    gap: 10px; /* রো ও কলামের মধ্যে ফাঁক */
}

.item1 {
    grid-column: 1; /* প্রথম কলামে */
    grid-row: 1; /* প্রথম রোতে */
}

.item2 {
    grid-column: 2; /* দ্বিতীয় কলামে */
    grid-row: 1; /* প্রথম রোতে */
```

```
}

.item3 {
    grid-column: 1 / span 2; /* উভয় কলাম জুড়ে */
    grid-row: 2; /* দ্বিতীয় রোতে */
}

.item4 {
    grid-column: 1; /* প্রথম কলামে */
    grid-row: 2; /* দ্বিতীয় রোতে */
}
```

সারাংশ:

- **CSS Grid** একটি ডুয়াল-ডাইমেনশনাল লেআউট তৈরি করতে সাহায্য করে, যেখানে উপাদানগুলোকে সহজেই কলাম এবং রোতে সাজানো যায়।
- `grid-template-rows` এবং `grid-template-columns` প্রপার্টিগুলি গ্রিডের রো এবং কলামের উচ্চতা ও প্রস্থ নির্ধারণ করতে ব্যবহৃত হয়, যা একটি দক্ষ এবং নিয়ন্ত্রিত লেআউট তৈরিতে সহায়ক।

CSS Grid ব্যবহার করে আপনি খুব সহজে এবং কার্যকরভাবে জটিল লেআউট তৈরি করতে পারেন।

5. How does the `flex-direction` property work in Flexbox, and what are its values?

Flexbox এ `flex-direction` প্রপার্টি উপাদানগুলির (ফ্লেক্স আইটেম) বিন্যাসের দিক নির্ধারণ করে। এটি গ্রিডের অভ্যন্তরীণ উপাদানগুলোকে কিভাবে সাজানো হবে তার একটি নির্ধারক। `flex-direction` প্রপার্টির মাধ্যমে আপনি প্রধান অক্ষ (main axis) এবং ক্রস অক্ষ (cross axis) নির্ধারণ করতে পারেন।

flex-direction এর মানসমূহ:

`flex-direction` প্রপার্টির চারটি মূল মান রয়েছে:

1. **row (ডিফল্ট)**:
    - এটি উপাদানগুলোকে অনুভূমিকভাবে (বামে থেকে ডানে) সাজায়।
    - প্রধান অক্ষ হল অনুভূমিক (x-axis) এবং ক্রস অক্ষ হল উল্লম্ব (y-axis)।

```
.container {
    display: flex;
    flex-direction: row; /* উপাদানগুলো অনুভূমিকভাবে সাজানো হবে
*/
}
```

2. **row-reverse**:
    - এটি উপাদানগুলোকে অনুভূমিকভাবে (ডানে থেকে বামে) সাজায়, অর্থাৎ উল্টো দিক থেকে।
    - প্রধান অক্ষ এখনও অনুভূমিক, কিন্তু উপাদানগুলোর সাজানো উল্টো।

```
.container {
    display: flex;
    flex-direction: row-reverse; /* উপাদানগুলো ডানে থেকে বামে
সাজানো হবে */
}
```

3. **column**:
    - এটি উপাদানগুলোকে উল্লম্বভাবে (শীর্ষ থেকে তল) সাজায়।
    - প্রধান অক্ষ হবে উল্লম্ব এবং ক্রস অক্ষ হবে অনুভূমিক।

```
.container {
    display: flex;
    flex-direction: column; /* উপাদানগুলো উল্লম্বভাবে সাজানো হবে
*/
}
```

4. **column-reverse**:
    - এটি উপাদানগুলোকে উল্লম্বভাবে (তল থেকে শীর্ষে) সাজায়, অর্থাৎ উল্টো দিক থেকে।

- প্রধান অক্ষ এখনও উল্লম্ব, কিন্তু উপাদানগুলোর সাজানো উল্টো।

```
.container {
    display: flex;
    flex-direction: column-reverse; /* উপাদানগুলো তল থেকে শীর্ষে সাজানো হবে */
}
```

উদাহরণ:

```
<div class="container">
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
</div>
```

```
.container {
    display: flex;
    flex-direction: row; /* বা row-reverse, column, column-reverse */
}

.item {
    background-color: lightblue;
    margin: 10px;
    padding: 20px;
}
```

সারসংক্ষেপ:

- `flex-direction` প্রপার্টি ফ্লেক্স কন্টেইনারের উপাদানগুলোর বিন্যাসের দিক নির্ধারণ করে।
- এর চারটি মান আছে: `row`, `row-reverse`, `column`, এবং `column-reverse`।
- এই প্রপার্টি ফ্লেক্সবক্স লেআউট তৈরি করতে সাহায্য করে, যা বিভিন্ন স্ক্রীন সাইজের জন্য প্রয়োজনীয়।

6. What is a viewport, and how do you make elements responsive to viewport changes?

Viewport হলো আপনার ব্রাউজারের দৃশ্যমান অঞ্চল, যেখানে ওয়েবপেজের বিষয়বস্তু প্রদর্শিত হয়। এটি ব্রাউজারের উইন্ডোর আকারের সাথে সম্পর্কিত এবং বিভিন্ন ডিভাইসে (মোবাইল, ট্যাবলেট, ডেস্কটপ) এর আকার পরিবর্তিত হতে পারে। ভিউপোর্টের আকার পরিবর্তনের সাথে সাথে ডিজাইনও পরিবর্তিত হওয়া উচিত, যাতে ব্যবহারকারীরা সব ডিভাইসে একটি ভাল অভিজ্ঞতা পান।

Responsive Design:

এখন, ভিউপোর্টের পরিবর্তনের প্রতি উপাদানগুলোকে প্রতিক্রিয়া জানাতে নিম্নলিখিত পদ্ধতিগুলি ব্যবহার করা হয়:

1. Viewport Meta Tag:

মোবাইল ডিভাইসের জন্য প্রথমে একটি ভিউপোর্ট মেটা ট্যাগ যুক্ত করা উচিত। এটি ডিভাইসের স্ক্রীন সাইজ অনুযায়ী পেজের স্কেল এবং প্রস্থ নির্ধারণ করে।

```
<meta name="viewport" content="width=device-width, initial-sc
ale=1.0">
```

2. Flexible Layouts:

ফ্লেক্সিবল লেআউট তৈরিতে `percentage` অথবা `vw` (viewport width) এবং `vh` (viewport height) ইউনিট ব্যবহার করা যেতে পারে।

উদাহরণ:

```
.container {
    width: 80%; /* 80% ভিউপোর্টের প্রস্থ */
    height: 50vh; /* 50% ভিউপোর্টের উচ্চতা */
}
```

3. Media Queries:

মিডিয়া কোয়েরি ব্যবহার করে বিভিন্ন স্ক্রীন সাইজের জন্য আলাদা স্টাইল প্রয়োগ করা হয়। এটি বিভিন্ন ডিভাইসের জন্য কাস্টমাইজেশন সম্ভব করে।

উদাহরণ:

```
@media (max-width: 600px) {
    .container {
        background-color: lightblue; /* মোবাইল স্ক্রীনের জন্য */
    }
}

@media (min-width: 601px) {
    .container {
        background-color: lightgreen; /* ডেস্কটপ স্ক্রীনের জন্য */
    }
}
```

4. Flexbox এবং Grid:

Flexbox এবং CSS Grid ব্যবহারের মাধ্যমে উপাদানগুলোকে স্বয়ংক্রিয়ভাবে উপযুক্তভাবে সাজানো যায়।

Flexbox উদাহরণ:

```
.container {
    display: flex;
    flex-wrap: wrap; /* উপাদানগুলোকে স্বয়ংক্রিয়ভাবে লাইন ব্রেক করতে
দেয় */
}
```

Grid উদাহরণ:

```
.container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1f
```

```
r)); /* স্বয়ংক্রিয়ভাবে কলাম সংখ্যা বাড়িয়ে দেয় */
}
```

5. Responsive Images:

ছবিগুলোকে রেসপন্সিভ করতে `max-width: 100%` এবং `height: auto` ব্যবহার করা হয়।

উদাহরণ:

```
img {
    max-width: 100%; /* চিত্রের প্রস্থ ভিউপোর্টের প্রস্থের সমান হবে */
    height: auto; /* চিত্রের উচ্চতা স্বয়ংক্রিয়ভাবে পরিবর্তিত হবে */
}
```

সারসংক্ষেপ:

- **Viewport** হল ব্রাউজারে দৃশ্যমান অঞ্চলের আকার।
- উপাদানগুলোকে ভিউপোর্ট পরিবর্তনের প্রতি প্রতিক্রিয়া জানাতে, ভিউপোর্ট মেটা ট্যাগ, ফ্লেক্সিবল লেআউট, মিডিয়া কোয়েরি, Flexbox এবং CSS Grid, এবং রেসপন্সিভ ছবি ব্যবহার করা হয়।
- এই পদ্ধতিগুলি একটি উন্নত এবং ব্যবহারকারী-বান্ধব অভিজ্ঞতা তৈরি করে।

7. How can you create a sticky footer that stays at the bottom of the page?

একটি **sticky footer** তৈরি করার জন্য যা পৃষ্ঠার তলায় থাকে, আপনি CSS-এ কিছু সহজ নিয়ম প্রয়োগ করতে পারেন। নিচে এর জন্য একটি সাধারণ পদ্ধতি বর্ণনা করা হলো:

1. HTML Structure:

প্রথমে একটি HTML স্ট্রাকচার তৈরি করুন যেখানে একটি কন্টেন্ট এবং একটি ফুটার থাকবে।

```
<!DOCTYPE html>
<html lang="bn">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initia
l-scale=1.0">
    <title>Sticky Footer Example</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="content">
        <h1>পৃষ্ঠার বিষয়বস্তু</h1>
        <p>এখানে আপনার বিষয়বস্তু যাবে।</p>
    </div>
    <footer class="footer">
        <p>এটি একটি স্টিকি ফুটার।</p>
    </footer>
</body>
</html>
```

2. CSS Styling:

এখন CSS ব্যবহার করে ফুটারটিকে স্টিকি করা হবে। নিচে CSS কোড দেওয়া হলো:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

html, body {
    height: 100%; /* পুরো উচ্চতা নিন */
}

.body {
    display: flex;
    flex-direction: column; /* উল্লম্বভাবে সাজানো */
    min-height: 100vh; /* সর্বনিম্ন উচ্চতা 100vh */
}
```

```css
.content {
    flex: 1; /* কন্টেন্টের জন্য স্থিতিস্থাপকতা প্রদান করে */
    padding: 20px;
}

.footer {
    background-color: #333; /* ফুটারের পটভূমি রঙ */
    color: white; /* ফুটারের টেক্সটের রঙ */
    text-align: center; /* কেন্দ্রিত টেক্সট */
    padding: 10px 0; /* উপরে এবং নিচে প্যাডিং */
}
```

3. ব্যাখ্যা:

- **সিলেক্টর**: সব উপাদানের মার্জিন এবং প্যাডিং শূন্য করে দেয় এবং `box-sizing` কে `border-box` সেট করে।
- `html, body`: এই উভয়কে 100% উচ্চতা দেয় যাতে পুরো viewport ব্যবহার করা যায়।
- `flex`: `.body` ক্লাসে `display: flex;` এবং `flex-direction: column;` ব্যবহার করা হয়েছে, যা উপাদানগুলোকে উল্লম্বভাবে সাজায়।
- `min-height: 100vh;`: এটি নিশ্চিত করে যে `.body` সর্বনিম্ন 100vh (viewport উচ্চতা) পর্যন্ত থাকবে।
- `flex: 1;`: এটি `.content` ক্লাসকে ফুটারের উপরে যতটা সম্ভব স্থান নিতে দেয়।
- `.footer`: এটি ফুটারের স্টাইলিংয়ের জন্য।

কাজের ফলাফল:

এই কোডের ফলে, আপনার ফুটার পৃষ্ঠার তলায় থাকবে, এবং পৃষ্ঠার বিষয়বস্তু যথেষ্ট বেশি হলে এটি স্ক্রোল করবে, কিন্তু বিষয়বস্তু কম হলে ফুটার সবসময় দৃশ্যমান থাকবে।

সম্পূর্ণ কোড:

এখন আপনি একটি HTML ফাইল এবং একটি CSS ফাইল তৈরি করে উপরের কোডগুলো যুক্ত করতে পারেন এবং এটি আপনার ব্রাউজারে পরীক্ষা করতে পারেন।

এই পদ্ধতিতে আপনি সহজেই একটি স্টিকি ফুটার তৈরি করতে পারেন যা পৃষ্ঠার তলায় থাকবে।

8. Explain how you would approach creating a mobile-first design.

Mobile-first design একটি ডিজাইন কৌশল যা প্রথমে মোবাইল ডিভাইসের জন্য ডিজাইন করা হয় এবং তারপর ধাপে ধাপে বড় স্ক্রীনে সম্প্রসারণ করা হয়। এই পদ্ধতির লক্ষ্য হল ব্যবহারকারীর অভিজ্ঞতাকে অগ্রাধিকার দেওয়া এবং মোবাইল ব্যবহারকারীদের জন্য একটি কার্যকর এবং ব্যবহারযোগ্য ডিজাইন তৈরি করা। এখানে একটি মোবাইল-ফার্স্ট ডিজাইন তৈরি করার জন্য আমি কীভাবে অগ্রসর হব তা বর্ণনা করা হলো:

1. গবেষণা এবং পরিকল্পনা:

- **লক্ষ্য নির্ধারণ**: প্রথমে, আপনার ডিজাইনের উদ্দেশ্য এবং লক্ষ্য শ্রোতাকে চিহ্নিত করুন।
- **ব্যবহারকারী গবেষণা**: আপনার লক্ষ্য ব্যবহারকারীদের প্রয়োজনীয়তা এবং আচরণ বোঝার জন্য গবেষণা করুন।

2. কন্টেন্ট ও ফিচারPrioritization:

- **কন্টেন্ট শ্রেণীবদ্ধ করুন**: মোবাইল ডিভাইসে কন্টেন্টের ভলিউম সীমিত হতে পারে, তাই সবচেয়ে গুরুত্বপূর্ণ কন্টেন্ট এবং ফিচারগুলো চিহ্নিত করুন।
- **হায়ারার্কি তৈরি করুন**: মোবাইল ভিউয়ের জন্য কন্টেন্টের একটি স্পষ্ট হায়ারার্কি তৈরি করুন যাতে ব্যবহারকারীরা দ্রুত তথ্য খুঁজে পেতে পারে।

3. ডিজাইন ও প্রোটোটাইপিং:

- **লেআউট ডিজাইন**: প্রথমে মোবাইল ডিভাইসের জন্য একটি লেআউট ডিজাইন করুন। এটি সাধারণত একক কলামের লেআউট এবং সহজ নেভিগেশন ব্যবস্থার সাথে করা হয়।

```
<!-- উদাহরণ HTML -->
<nav>
    <ul>
        <li>হোম</li>
        <li>আমাদের  সম্পর্কে</li>
        <li>যোগাযোগ</li>
    </ul>
```

```
</nav>
<section>
    <h1>স্বাগতম!</h1>
    <p>এটি আমাদের মোবাইল ফার্স্ট ডিজাইন।</p>
</section>
```

- **ভিজ্যুয়াল প্রোটোটাইপ**: ডিজাইন টুল (যেমন Figma, Adobe XD) ব্যবহার করে মোবাইল ডিজাইনের একটি প্রোটোটাইপ তৈরি করুন।

4. স্টাইলিং:

- **CSS**: মোবাইল জন্য CSS লিখুন এবং ডিফল্ট স্টাইল তৈরি করুন। এর জন্য, সিম্পল, ক্লিন এবং স্পষ্ট ডিজাইন প্রয়োজন।

```
body {
    font-size: 16px; /* সহজ পড়ার জন্য */
}

nav ul {
    display: flex;
    flex-direction: column; /* মোবাইলে উপরে থেকে নিচে সাজানো */
}
```

5. মিডিয়া কোয়েরি ব্যবহার:

- মোবাইল ডিজাইন করার পরে, বড় স্ক্রীনের জন্য মিডিয়া কোয়েরি যোগ করুন। এটি ব্রাউজারের আকার পরিবর্তনের সঙ্গে সঙ্গে CSS নিয়ম পরিবর্তন করতে সহায়ক হবে।

```
@media (min-width: 768px) {
    nav ul {
        flex-direction: row; /* বড় স্ক্রীনে অনুভূমিক সাজানো */
    }
}
```

6. টেস্টিং:

- মোবাইল এবং ডেস্কটপ উভয় ডিভাইসে ডিজাইনটি পরীক্ষা করুন। ব্যবহারকারীর অভিজ্ঞতা এবং ফাংশনালিটি যাচাই করতে ব্যবহারকারীদের অন্তর্ভুক্ত করুন।

7. অভিজ্ঞতা উন্নত করুন:

- ব্যবহারকারীর ফিডব্যাকের ভিত্তিতে ডিজাইন এবং ফিচারগুলি সংশোধন করুন। এটি নিশ্চিত করতে হবে যে মোবাইল ব্যবহারকারীদের জন্য অভিজ্ঞতা সেরা হয়।

8. পারফরম্যান্স অপটিমাইজেশন:

- মোবাইল ডিভাইসে লোডিং সময় কমাতে ইমেজ অপ্টিমাইজ করুন এবং অপ্রয়োজনীয় স্ক্রিপ্টগুলি কমিয়ে আনার চেষ্টা করুন।

সারসংক্ষেপ:

- **মোবাইল-ফার্স্ট ডিজাইন** একটি পদ্ধতি যেখানে মোবাইল ব্যবহারকারীদের অভিজ্ঞতাকে প্রথম অগ্রাধিকার দেওয়া হয়।
- গবেষণা, কন্টেন্টের প্রাধান্য, ডিজাইন, মিডিয়া কোয়েরি, টেস্টিং, এবং পারফরম্যান্স অপটিমাইজেশনের মাধ্যমে একটি কার্যকর ডিজাইন তৈরি করা যায়।
- এই পদ্ধতির ফলে ব্যবহারকারীরা মোবাইল ডিভাইসে একটি আরও সমৃদ্ধ এবং কার্যকরী অভিজ্ঞতা লাভ করবে।

>> Advanced CSS Questions (7 questions)

1. What do you understand by the universal sector?

Universal sector বলতে সাধারণত এমন একটি অংশ বোঝায় যা বিভিন্ন ক্ষেত্র বা শিল্পের জন্য প্রযোজ্য এবং একই সময়ে বিভিন্ন চাহিদা বা পরিস্থিতিতে ব্যবহার করা যায়। এটি একটি অর্থনৈতিক বা সামাজিক কাঠামো যেখানে বিভিন্ন সেক্টর বা কার্যকলাপের মধ্যে সম্পর্ক থাকে এবং সমন্বয় সাধন করা হয়।

প্রধান দিকনির্দেশনা:

1. **অর্থনৈতিক সেক্টর**:
    - এটি সাধারণত তিনটি প্রধান অর্থনৈতিক সেক্টরে বিভক্ত: প্রাথমিক (যেমন কৃষি), গৌণ (যেমন শিল্প) এবং তৃতীয়ক (যেমন পরিষেবা)।
    - ইউনিভার্সাল সেক্টর সব ক্ষেত্রের সাপোর্টিং ফাংশনগুলি কভার করে, যা এই তিনটি সেক্টরের মধ্যে সঠিক সমন্বয় ঘটায়।
2. **সামাজিক উন্নয়ন**:
    - সামাজিক কাঠামোর উন্নয়ন এবং বিভিন্ন সেক্টরের মধ্যে সহযোগিতা নিশ্চিত করে।
    - এটি জনগণের মৌলিক চাহিদাগুলি যেমন স্বাস্থ্য, শিক্ষা, এবং সুরক্ষা সংক্রান্ত সেবাগুলি প্রদান করে।
3. **টেকসই উন্নয়ন**:
    - ইউনিভার্সাল সেক্টর টেকসই উন্নয়নের লক্ষ্য পূরণে সহায়ক হয়, যা দীর্ঘমেয়াদী অর্থনৈতিক এবং পরিবেশগত স্থিতিশীলতার দিকে নির্দেশ করে।

উদাহরণ:

- **সামাজিক নিরাপত্তা**: এই সেক্টরটি বিভিন্ন অর্থনৈতিক সেক্টরের মধ্যে সমন্বয় ঘটায়, যেন সকল জনগণ পর্যাপ্ত সেবা পায়।
- **শিক্ষা এবং স্বাস্থ্যসেবা**: এই দুটি সেক্টর একে অপরের উপর নির্ভরশীল এবং তাদের কার্যক্রমের জন্য একটি ইউনিভার্সাল কাঠামো গঠন করে।

সারসংক্ষেপ:

- **ইউনিভার্সাল সেক্টর** এমন একটি কাঠামো যা বিভিন্ন শিল্প এবং অর্থনৈতিক সেক্টরের মধ্যে সম্পর্ক স্থাপন করে এবং সমন্বয় সাধন করে।
- এটি সমাজের মৌলিক চাহিদাগুলিকে পূরণ করার জন্য সহায়ক, যেমন শিক্ষা, স্বাস্থ্য, এবং সামাজিক নিরাপত্তা।
- টেকসই উন্নয়ন নিশ্চিত করতে এবং উন্নত জীবনমানের লক্ষ্যে এই সেক্টরের গুরুত্ব অপরিসীম।

2. What are CSS preprocessors, and how do they work? Give examples like SASS or LESS.

CSS Preprocessors হলো CSS-এর উপর ভিত্তি করে তৈরি করা কিছু টুল যা উন্নত বৈশিষ্ট্য এবং কার্যকারিতা সরবরাহ করে। এগুলি CSS কোড লেখার প্রক্রিয়াকে সহজতর করে, বিশেষ করে বড় এবং জটিল প্রকল্পগুলির জন্য। প্রিপ্রসেসরগুলি CSS কোডের একটি ট্রান্সপাইলার হিসাবে কাজ করে, যা লেখার সময় অতিরিক্ত ফিচারগুলি যোগ করে এবং শেষে স্বাভাবিক CSS কোডে রূপান্তরিত করে।

CSS Preprocessors কী কী সুবিধা দেয়:

1. **ভেরিয়েবলস**: প্রিপ্রসেসরগুলি ভেরিয়েবল ব্যবহার করতে দেয়, যা রঙ, ফন্ট, সাইজ ইত্যাদির মতো পুনরাবৃত্ত মানগুলিকে সংরক্ষণ করে।

```
// SASS উদাহরণ
$primary-color: #3498db;

body {
    color: $primary-color;
}
```

2. **নেস্টেড সিলেক্টরস**: CSS কোডকে আরও পরিষ্কার এবং পাঠযোগ্য করার জন্য নেস্টিং সম্ভব।

```
// SASS উদাহরণ
.nav {
    ul {
        list-style: none;
    }

    li {
        display: inline;
    }
}
```

3. **Mixin**: পুনর্ব্যবহারযোগ্য কোড স্নিপেট তৈরি করার জন্য মিক্সিন ব্যবহার করা হয়।

```
// SASS উদাহরণ
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
```

```
    -moz-border-radius: $radius;
    border-radius: $radius;
}

.box {
    @include border-radius(10px);
}
```

4. **নেস্টেড মিডিয়া কোয়েরি**: মিডিয়া কোয়েরিগুলোকে সহজে নেস্ট করা যায়, যা রেসপন্সিভ ডিজাইনকে সহজ করে।

```
// SASS উদাহরণ
.container {
    width: 100%;

    @media (min-width: 768px) {
        width: 50%;
    }
}
```

সাধারণ CSS Preprocessors:

1. **Sass (Syntactically Awesome Style Sheets)**:
    - Sass হলো একটি জনপ্রিয় CSS প্রিপ্রসেসর যা SCSS (Sassy CSS) এবং Sass (Indented Syntax) দুইটি সিনট্যাক্স সমর্থন করে।
    - এটি ভেরিয়েবল, মিক্সিন, ইনহেরিটেন্স এবং ফাংশন প্রদান করে।
2. **LESS**:
    - LESS হলো আরেকটি CSS প্রিপ্রসেসর যা JavaScript ব্যবহার করে লেখা হয়েছে।
    - LESS-এও ভেরিয়েবল, মিক্সিন এবং নেস্টিং এর সুবিধা পাওয়া যায়।

কিভাবে কাজ করে:

1. **লেখা**: আপনি প্রিপ্রসেসর সিনট্যাক্সে CSS কোড লেখেন (যেমন SCSS বা LESS)।
2. **কম্পাইল**: প্রিপ্রসেসর টুলটি আপনার কোডকে স্ট্যান্ডার্ড CSS কোডে রূপান্তর করে।

3. **লিঙ্ক**: প্রাপ্ত CSS ফাইলটি আপনার HTML ডকুমেন্টের সাথে সংযুক্ত করা হয়।

উদাহরণ:

SASS কোড:

```
$main-color: #333;

.header {
    color: $main-color;

    .title {
        font-size: 2em;
    }
}
```

কম্পাইল হওয়া CSS:

```
.header {
    color: #333;
}

.header .title {
    font-size: 2em;
}
```

সারসংক্ষেপ:

- **CSS Preprocessors** হল টুল যা CSS কোড লেখাকে আরও কার্যকরী এবং সংগঠিত করে।
- **SASS** এবং **LESS** দুটি জনপ্রিয় প্রিপ্রসেসর, যা ভেরিয়েবল, মিক্সিন, নেস্টিং এবং অন্যান্য উন্নত বৈশিষ্ট্য সরবরাহ করে।
- এটি লেখার সময় কোডকে সহজ এবং পরিষ্কার করার জন্য সহায়ক, এবং প্রাপ্ত CSS কোডটি ব্রাউজারে কার্যকরভাবে কাজ করে।

3. Explain CSS custom properties (CSS variables) and their benefits.

CSS Custom Properties, যেগুলো সাধারণত **CSS Variables** হিসেবে পরিচিত, CSS-এ একটি শক্তিশালী ফিচার যা আপনাকে স্টাইল শীটে ভেরিয়েবল তৈরি করতে এবং ব্যবহারের অনুমতি দেয়। এই ভেরিয়েবলগুলো বিভিন্ন স্থানে ব্যবহার করা যায় এবং সহজে পরিবর্তন করা যায়, যা ডিজাইন এবং কন্টেন্ট ম্যানেজমেন্টকে আরও কার্যকরী করে তোলে।

CSS Custom Properties কী?

CSS Custom Properties একটি নির্দিষ্ট সিনট্যাক্সের মাধ্যমে তৈরি হয়। এগুলি সাধারণত একটি ডাবল কোলন (`--`) দিয়ে শুরু হয় এবং কোনো একটি CSS নির্বাচক (যেমন ক্লাস বা আইডি) এর অধীনে ডিফাইন করা হয়। উদাহরণস্বরূপ:

```
:root {
    --primary-color: #3498db;
    --font-size: 16px;
}
```

এখন এই ভেরিয়েবলগুলো পরবর্তীতে স্টাইল শীটে ব্যবহৃত হতে পারে:

```
body {
    color: var(--primary-color);
    font-size: var(--font-size);
}
```

CSS Custom Properties-এর সুবিধা:

1. **পুনরাবৃত্তি কমানো**:
    - একাধিক স্থানে একই মান ব্যবহারের সময়, CSS ভেরিয়েবল ব্যবহার করলে সেই মান পরিবর্তন করতে পারেন একবারে।
2. **সহজভাবে পরিবর্তন**:

- CSS ভেরিয়েবল ব্যবহার করে, আপনি স্টাইল পরিবর্তন করতে পারেন কোন জায়গায় গিয়ে সঠিক ভেরিয়েবল পরিবর্তন করলেই হয়।

3. **ডাইনামিক ভ্যালু:**
   - CSS ভেরিয়েবলগুলি ডাইনামিক। অর্থাৎ, JavaScript ব্যবহার করে আপনি runtime-এ ভেরিয়েবলগুলির মান পরিবর্তন করতে পারেন।
4. **সাপোর্টেড নেস্টিং:**
   - CSS ভেরিয়েবলগুলি CSS-এর কোনও নির্দিষ্ট সিলেক্টরের অধীনে সংজ্ঞায়িত করা যেতে পারে, যা তাদের নেস্টেড ব্যবহারকে সম্ভব করে।
5. **স্কোপিং:**
   - CSS ভেরিয়েবলগুলোর একটি স্কোপ থাকে। এটি একটি সিলেক্টরের মধ্যে সংজ্ঞায়িত হলে, সেটি শুধুমাত্র সেই সিলেক্টরের ভেতরেই প্রযোজ্য হয়।

উদাহরণ:

```
:root {
    --main-bg-color: #f0f0f0;
    --text-color: #333;
}

.container {
    background-color: var(--main-bg-color);
    color: var(--text-color);
}

.button {
    background-color: var(--primary-color);
}
```

JavaScript এর সাথে ব্যবহার:

আপনি JavaScript এর মাধ্যমে CSS Custom Properties-এর মান পরিবর্তন করতে পারেন:

```
document.documentElement.style.setProperty('--primary-color', '#e74c3c');
```

সারসংক্ষেপ:

- **CSS Custom Properties** (বা CSS Variables) হল CSS-এর মধ্যে পরিবর্তনশীল মান যা ডিফাইন করা যেতে পারে এবং পুনর্ব্যবহারযোগ্য।
- এগুলি কোডের পুনরাবৃত্তি কমায়, স্টাইল পরিবর্তনকে সহজ করে, এবং ডাইনামিকভাবে পরিবর্তনের সুযোগ দেয়।
- JavaScript-এর মাধ্যমে এই ভেরিয়েবলগুলোর মান পরিবর্তন করা সম্ভব, যা এটি আরও কার্যকরী করে তোলে।

4. How do CSS animations work, and how can you create a keyframe animation?

CSS Animations হল CSS-এর একটি শক্তিশালী বৈশিষ্ট্য যা উপাদানের ভিজ্যুয়াল পরিবর্তনকে সহজে এবং কার্যকরভাবে পরিচালনা করতে দেয়। এটি একটি উপাদানের গুণাবলী (যেমন অবস্থান, আকার, রঙ ইত্যাদি) পরিবর্তন করতে সাহায্য করে, এবং সেগুলোকে সময়ের সাথে সাথে অ্যানিমেট করে।

CSS Animations-এর প্রধান অংশ:

1. **Animation Properties**: অ্যানিমেশনের জন্য বিভিন্ন প্রপার্টি রয়েছে, যেমন:
   - `animation-name`: অ্যানিমেশনের নাম নির্ধারণ করে।
   - `animation-duration`: অ্যানিমেশন কত সময় ধরে চলবে তা নির্ধারণ করে।
   - `animation-timing-function`: অ্যানিমেশনের গতি নিয়ন্ত্রণ করে (যেমন linear, ease-in, ease-out)।
   - `animation-delay`: অ্যানিমেশন শুরু হওয়ার আগে কত সময় অপেক্ষা করতে হবে।
   - `animation-iteration-count`: অ্যানিমেশন কতবার পুনরাবৃত্তি হবে।

- `animation-direction`: অ্যানিমেশন কখন সামনে এবং কখন পেছনে চলবে তা নির্ধারণ করে।

2. **Keyframes**: এটি অ্যানিমেশন তৈরির জন্য প্রধান উপাদান। `@keyframes` ব্যবহার করে আপনি বিভিন্ন স্টেজে উপাদানের গুণাবলী কীভাবে পরিবর্তিত হবে তা নির্ধারণ করতে পারেন।

কীভাবে CSS Animations কাজ করে:

1. **নির্ধারণ**: প্রথমে `@keyframes` দিয়ে অ্যানিমেশনের জন্য একটি কনফিগারেশন তৈরি করুন।
2. **অ্যানিমেশন প্রপার্টি অ্যাপ্লাই করা**: তারপর, অ্যানিমেশনটির নাম এবং অন্যান্য প্রপার্টি সংশ্লিষ্ট HTML এলিমেন্টে প্রয়োগ করুন।

উদাহরণ:

Step 1: Keyframe Animation তৈরি করা

```
@keyframes slideIn {
    0% {
        transform: translateX(-100%); /* শুরুতে বাম দিকে */
        opacity: 0; /* দেখা যাচ্ছে না */
    }
    100% {
        transform: translateX(0); /* মূল অবস্থানে */
        opacity: 1; /* সম্পূর্ণ দৃশ্যমান */
    }
}
```

Step 2: অ্যানিমেশন প্রপার্টি অ্যাপ্লাই করা

```
.slide {
    animation-name: slideIn; /* অ্যানিমেশনের নাম */
    animation-duration: 1s; /* অ্যানিমেশন 1 সেকেন্ডে সম্পন্ন হবে */
    animation-timing-function: ease; /* অ্যানিমেশনের গতি */
    animation-fill-mode: forwards; /* অ্যানিমেশন শেষ হলে অবস্থা ধ
রে রাখবে */
}
```

Step 3: HTML কোড

```
<div class="slide">স্বাগতম!</div>
```

CSS Animation এর মূল অংশগুলোর সারসংক্ষেপ:

- **Keyframes**: অ্যানিমেশনের বিভিন্ন পর্যায় নির্ধারণ করে।
- **Animation Properties**: অ্যানিমেশন নিয়ন্ত্রণ করতে ব্যবহৃত হয়, যেমন নাম, সময়কাল, গতি এবং পুনরাবৃত্তি সংখ্যা।

সারসংক্ষেপ:

- **CSS Animations** ব্যবহার করে, আপনি উপাদানের স্টাইল পরিবর্তন করতে পারেন সময়ের সাথে সাথে।
- `@keyframes` দিয়ে অ্যানিমেশনের বিভিন্ন পর্যায় নির্ধারণ করা হয়, এবং CSS প্রপার্টি ব্যবহার করে HTML উপাদানে অ্যানিমেশন প্রয়োগ করা হয়।
- এটি একটি সহজ এবং কার্যকর উপায়ে আপনার ওয়েবপেজে ভিজ্যুয়াল আকর্ষণ যোগ করার জন্য ব্যবহার করা হয়।

5. What are CSS transitions, and how are they different from animations?

CSS Transitions এবং **CSS Animations** উভয়ই CSS-এর একটি গুরুত্বপূর্ণ অংশ, যা উপাদানের পরিবর্তনশীল বৈশিষ্ট্যগুলিকে কার্যকরভাবে পরিচালনা করতে সাহায্য করে। তবে, এগুলির মধ্যে কিছু মৌলিক পার্থক্য রয়েছে।

CSS Transitions কী?

CSS Transitions হল একটি উপায় যা দ্বারা একটি CSS প্রপার্টি পরিবর্তন করতে সময়কাল যুক্ত করা হয়। এটি একটি পাত্র বা উপাদানের গুণাবলীর পরিবর্তন (যেমন রঙ, আকার, অবস্থান) ধীরে ধীরে ঘটে। ট্রানজিশন সাধারণত একটি হোভার ইফেক্ট বা ব্যবহারকারীর অ্যাকশনের উপর ভিত্তি করে ঘটে।

উদাহরণ:

```
.box {
    width: 100px;
    height: 100px;
    background-color: blue;
    transition: background-color 0.5s ease; /* ব্যাকগ্রাউন্ড রঙের প
রিবর্তন 0.5 সেকেন্ডে হবে */
}

.box:hover {
    background-color: red; /* হোভার করলে রঙ পরিবর্তন হবে */
}
```

CSS Animations কী?

CSS Animations হল একটি আরও জটিল প্রক্রিয়া যা `@keyframes` ব্যবহার করে বিভিন্ন পর্যায়ে উপাদানের স্টাইল পরিবর্তন করতে সাহায্য করে। অ্যানিমেশনগুলি একটি নির্দিষ্ট সময়ের মধ্যে একাধিক গুণাবলী পরিবর্তন করতে পারে এবং ব্যবহারকারীর ক্রিয়ার উপর ভিত্তি করে নয়, বরং স্বয়ংক্রিয়ভাবে শুরু হতে পারে।

উদাহরণ:

```
@keyframes move {
    0% {
        transform: translateX(0);
    }
    100% {
        transform: translateX(100px);
    }
}

.box {
    width: 100px;
    height: 100px;
    background-color: blue;
```

```
    animation: move 2s infinite; /* 2 সেকেন্ডে 100px ডানে স্থানান্তর
হবে */
}
```

CSS Transitions এবং Animations-এর মধ্যে প্রধান পার্থক্য:

1. **জটিলতা**:
   - **Transitions**: সাধারণত এক বা দুইটি গুণাবলী পরিবর্তনের জন্য ব্যবহৃত হয় এবং সময়কাল যুক্ত করে।
   - **Animations**: একাধিক গুণাবলীর পরিবর্তন এবং বিভিন্ন পর্যায়ে নিয়ন্ত্রণের জন্য ব্যবহৃত হয়।
2. **চালনা**:
   - **Transitions**: সাধারণত ব্যবহারকারীর ইন্টারঅ্যাকশনের উপর ভিত্তি করে (যেমন হোভার করা) শুরু হয়।
   - **Animations**: স্বয়ংক্রিয়ভাবে শুরু হয় এবং সময়সীমার সাথে একাধিক পরিবর্তন প্রদর্শন করে।
3. **কনফিগারেশন**:
   - **Transitions**: CSS প্রপার্টির সাথে সরাসরি কাজ করে এবং শুধু `transition` প্রপার্টি ব্যবহার করে কনফিগার করা হয়।
   - **Animations**: `@keyframes` ব্যবহার করে এবং এতে বেশ কয়েকটি প্রপার্টি কনফিগার করা যেতে পারে, যেমন গতি, দিক, পুনরাবৃত্তি ইত্যাদি।

সারসংক্ষেপ:

- **CSS Transitions** হল সহজ এবং সরল স্টাইল পরিবর্তনের জন্য, যা সময়ের সাথে একটি গুণাবলী পরিবর্তন করতে দেয়।
- **CSS Animations** হল জটিল এবং কার্যকরী, যা একাধিক গুণাবলী পরিবর্তনের জন্য ব্যবহার হয় এবং এটি স্বয়ংক্রিয়ভাবে বা বিভিন্ন পর্যায়ে কাজ করতে সক্ষম।
- উভয়ই ওয়েব ডিজাইনকে প্রাণবন্ত ও আকর্ষণীয় করে তুলতে সাহায্য করে, তবে তাদের ব্যবহারের উদ্দেশ্য এবং কার্যকারিতায় পার্থক্য রয়েছে।

6. What is the difference between `rem`, `em`, and `px` units?

CSS-এ বিভিন্ন ইউনিট ব্যবহার করা হয়, যার মধ্যে **rem**, **em**, এবং **px** হল তিনটি সাধারণ ইউনিট। প্রতিটি ইউনিটের নিজস্ব বৈশিষ্ট্য এবং ব্যবহার থাকে। নিচে এগুলোর মধ্যে পার্থক্যগুলো আলোচনা করা হলো:

1. px (পিক্সেল)

- **সংজ্ঞা**: পিক্সেল হল একটি স্থির ইউনিট, যা পর্দায় একটি পিক্সেলের আকারকে নির্দেশ করে।
- **বৈশিষ্ট্য**:
  - এটি একটি অবধারিত ইউনিট, অর্থাৎ এটি পর্দার রেজোলিউশনের ওপর ভিত্তি করে পরিবর্তিত হয় না।
  - এটি ডিজাইনগুলির জন্য সঠিক এবং নির্ভরযোগ্য, কিন্তু বিভিন্ন ডিভাইসে স্কেলিংয়ের ক্ষেত্রে অসুবিধা হতে পারে।
- **ব্যবহার**: সাধারণত সুনির্দিষ্ট আকারের এলিমেন্ট এবং মার্জিন, প্যাডিং ইত্যাদির জন্য ব্যবহার করা হয়।

উদাহরণ:

```
.box {
    width: 300px; /* 300 পিক্সেল */
    height: 200px; /* 200 পিক্সেল */
}
```

2. em

- **সংজ্ঞা**: `em` হল একটি রিলেটিভ ইউনিট যা বর্তমান এলিমেন্টের ফন্ট সাইজের ভিত্তিতে কাজ করে।
- **বৈশিষ্ট্য**:
  - যদি একটি এলিমেন্টের ফন্ট সাইজ 16px হয়, তাহলে `1em` সমান হবে 16px।

- যদি আপনি নেস্টেড এলিমেন্ট ব্যবহার করেন, তাহলে `em` ইউনিটটি অভ্যন্তরীণ এলিমেন্টের ফন্ট সাইজের ওপর ভিত্তি করে পরিবর্তিত হবে।

- **ব্যবহার**: সাধারণত ফন্ট সাইজ, প্যাডিং, মার্জিন ইত্যাদির জন্য ব্যবহার করা হয়, যেখানে এলিমেন্টের আকার ফন্ট সাইজের সাথে সম্পর্কিত হতে পারে।

উদাহরণ:

```
.container {
    font-size: 16px; /* মূল ফন্ট সাইজ */
}

.box {
    font-size: 1.5em; /* 1.5 x 16px = 24px */
    padding: 2em; /* 2 x 16px = 32px */
}
```

3. rem (root em)

- **সংজ্ঞা**: `rem` হল একটি রিলেটিভ ইউনিট, কিন্তু এটি মূল (root) এলিমেন্টের ফন্ট সাইজের উপর ভিত্তি করে কাজ করে (সাধারণত `<html>` এলিমেন্ট)।

- **বৈশিষ্ট্য**:
  - `rem` ব্যবহার করে, আপনি নিশ্চিত হন যে এটি সব সময় একই রকম থাকবে, যেহেতু এটি শুধুমাত্র মূল ফন্ট সাইজের ওপর নির্ভর করে।
  - এটি নেস্টিং-এর সমস্যাগুলো এড়ায়, কারণ এটি অভ্যন্তরীণ এলিমেন্টগুলির ফন্ট সাইজ পরিবর্তনের দ্বারা প্রভাবিত হয় না।

- **ব্যবহার**: পুনরাবৃত্তি করতে সুবিধা হয় এবং সাধারণত মার্জিন, প্যাডিং, এবং ফন্ট সাইজের জন্য ব্যবহৃত হয়।

উদাহরণ:

```
html {
    font-size: 16px; /* মূল ফন্ট সাইজ */
}
```

```css
.box {
    font-size: 1.5rem; /* 1.5 x 16px = 24px */
    padding: 2rem; /* 2 x 16px = 32px */
}
```

সারসংক্ষেপ:

- **px**: একটি স্থির ইউনিট, যা ডিজাইনের জন্য সুনির্দিষ্ট।
- **em**: বর্তমান এলিমেন্টের ফন্ট সাইজের ওপর ভিত্তি করে পরিবর্তিত হয়; নেস্টেড এলিমেন্টগুলির জন্য প্রভাবিত হতে পারে।
- **rem**: মূল ফন্ট সাইজের ওপর ভিত্তি করে কাজ করে; নেস্টেড এলিমেন্টগুলির প্রভাব এড়ায়।

এই ইউনিটগুলো ব্যবহার করে, আপনি আপনার ওয়েব ডিজাইনকে আরও নমনীয় এবং প্রতিক্রিয়াশীল করতে পারেন।

7. What is SVG and how can it be used to create custom shapes?

SVG (Scalable Vector Graphics) হল একটি XML ভিত্তিক ফাইল ফরম্যাট যা গ্রাফিক্স এবং অ্যানিমেশন তৈরির জন্য ব্যবহৃত হয়। এটি বিশেষভাবে ভেক্টর গ্রাফিক্স তৈরি করতে ডিজাইন করা হয়েছে, যা মানে হলো যে ছবিগুলো কোনও রেজোলিউশনে স্কেল করা যেতে পারে এবং তারা কখনোই অস্পষ্ট হবে না।

SVG-এর বৈশিষ্ট্য:

1. **স্কেলেবিলিটি**: SVG ফাইলগুলি যেকোনো আকারে স্কেল করা যেতে পারে, এটি সর্বদা পরিষ্কার ও ধারালো থাকে।
2. **সফল পঠনযোগ্যতা**: SVG একটি টেক্সট ফাইল ফরম্যাট, তাই আপনি সহজেই এটিকে এডিট করতে পারেন। এটি ব্রাউজারে সরাসরি রেন্ডার হয়।
3. **ইন্টারেক্টিভিটি**: SVG এ অ্যানিমেশন এবং ইন্টারেক্টিভিটি যুক্ত করা যেতে পারে, যা ব্যবহারকারীর জন্য আকর্ষণীয় অভিজ্ঞতা তৈরি করে।
4. **স্টাইলিং**: CSS এবং JavaScript ব্যবহার করে SVG গ্রাফিক্সকে স্টাইল এবং নিয়ন্ত্রণ করা যায়।

SVG ব্যবহার করে কাস্টম শেপ তৈরি করা:

SVG-এর সাহায্যে কাস্টম শেপ তৈরি করা খুবই সহজ। সাধারণত, SVG গ্রাফিক্সে কিছু প্রাথমিক উপাদান ব্যবহার করা হয়, যেমন:

- `<circle>`: গোলাকার শেপ তৈরি করতে।
- `<rect>`: আয়তাকার শেপ তৈরি করতে।
- `<line>`: রেখা তৈরি করতে।
- `<polygon>`: বিভিন্ন কোণযুক্ত শেপ তৈরি করতে।
- `<path>`: জটিল শেপ তৈরি করতে।

উদাহরণ:

1. একটি সিম্পল গোল তৈরি করা:

```
<svg width="100" height="100">
    <circle cx="50" cy="50" r="40" fill="blue" />
</svg>
```

এই উদাহরণে, একটি নীল গোল তৈরি করা হয়েছে যার কেন্দ্র `(50, 50)` এবং ব্যাস 80 পিক্সেল (রেডিয়াস 40 পিক্সেল)।

2. একটি আয়তাকার শেপ তৈরি করা:

```
<svg width="100" height="100">
    <rect width="100" height="50" fill="green" />
</svg>
```

এখানে, একটি সবুজ আয়তক তৈরি করা হয়েছে যার প্রস্থ 100 পিক্সেল এবং উচ্চতা 50 পিক্সেল।

3. জটিল শেপ তৈরি করা:

```
<svg width="200" height="200">
    <path d="M 10 80 Q 95 10 180 80 T 350 80" fill="transpare
```

```
nt" stroke="black" />
</svg>
```

এই উদাহরণে, একটি জটিল পাথ তৈরি করা হয়েছে যা একটি বেজিয়ার কার্ভ ব্যবহার করছে।

SVG এর সাথে CSS ও JavaScript ব্যবহার:

SVG উপাদানগুলোকে CSS দ্বারা স্টাইল করতে পারেন এবং JavaScript ব্যবহার করে অ্যানিমেশন বা ইন্টারঅ্যাকশন যুক্ত করতে পারেন।

CSS উদাহরণ:

```
circle {
    stroke: black;
    stroke-width: 2;
}
```

JavaScript উদাহরণ:

```
document.querySelector('circle').addEventListener('click', fu
nction() {
    alert('Circle clicked!');
});
```

সারসংক্ষেপ:

- **SVG** হল একটি শক্তিশালী গ্রাফিক্স ফরম্যাট যা স্কেলেবল ভেক্টর গ্রাফিক্স তৈরি করে।
- এটি কাস্টম শেপ তৈরির জন্য বিভিন্ন উপাদান (যেমন circle, rect, path) ব্যবহার করে।
- SVG গ্রাফিক্সকে CSS ও JavaScript এর মাধ্যমে স্টাইল ও নিয়ন্ত্রণ করা যায়, যা ইন্টারঅ্যাক্টিভ এবং অ্যানিমেটেড উপাদান তৈরি করতে সাহায্য করে।