Positions in CSS

Assignment

Solution.

Task 1:

Problem Statement
Write code to position equal-sized (50 X 50px) boxes A, B, C, D and E as follows

1. box A 200px from the left and 200px from the top of the viewport.
2. box B -30px left and -30px above from the centre of box A.
3. box B -30px right and -30px above from the centre of box A .
4. box B -30px left and -30px below from the centre of box A .
5.box B -30px right and -30px below from the centre of box A.

ANS1. VIA THE VSCODE

Task 2:

Problem Statement

Explain the difference between Absolute and Relative positioning.

ANS2.

Certainly! In the context of web development, particularly with CSS (Cascading Style Sheets), absolute and relative positioning are two key techniques used to control the layout and positioning of elements on a web page. Here's a detailed explanation of each:

Absolute Positioning

- **Absolute Positioning** places an element precisely relative to its nearest positioned ancestor. If no such ancestor exists, it is positioned

relative to the initial containing block (usually the `<html>` element or the browser window).

- When an element is given `position: absolute`, it is removed from the normal document flow. This means it does not affect the positioning of subsequent elements, and those elements are also not affected by it.

- You can specify the exact position of the element using properties like `top`, `right`, `bottom`, and `left`.

Example:

```html
<div class="container">
  <div class="absolute-box">I am absolutely positioned</div>
</div>

<style>
  .container {
    position: relative; /* Positioned ancestor */
    width: 200px;
    height: 200px;
    background-color: lightgray;
  }
  .absolute-box {
    position: absolute;
    top: 50px;
    left: 50px;
    width: 100px;
    height: 100px;
    background-color: blue;
  }
</style>
```

In this example, `.absolute-box` is positioned 50 pixels from the top and left of its nearest positioned ancestor, which is `.container`.

Relative Positioning

- **Relative Positioning** moves an element relative to its original position in the normal document flow. It remains in the flow, meaning it still occupies space and affects the positioning of other elements.

- When an element is given `position: relative`, it can be shifted using `top`, `right`, `bottom`, and `left`, but its original space is still preserved in the layout.

Example:

```html
<div class="relative-box">I am relatively positioned</div>

<style>
  .relative-box {
    position: relative;
    top: 20px;
    left: 30px;
    width: 100px;
    height: 100px;
    background-color: green;
  }
</style>
```

In this example, `.relative-box` is moved 20 pixels down and 30 pixels to the right from its original position, but the space it originally occupied is still there in the document flow.

Key Differences

1. **Document Flow**:

- **Absolute**: Removes the element from the document flow.
- **Relative**: Keeps the element in the document flow.

2. **Position Reference**:
- **Absolute**: Positions the element relative to its nearest positioned ancestor or the initial containing block.
- **Relative**: Positions the element relative to its original position.

3. **Impact on Other Elements**:
- **Absolute**: Does not affect the positioning of other elements since it is taken out of the flow.
- **Relative**: Affects the positioning of other elements as it remains in the flow.

4. **Use Cases**:
- **Absolute**: Useful for precisely positioning elements in complex layouts, overlays, or when elements need to be placed outside the normal flow.
- **Relative**: Useful for making small adjustments or offsets to elements within the normal flow, often for minor tweaks.

Understanding these differences helps in choosing the right positioning method based on the layout requirements and desired behavior of elements on the web page.

Task 3:

Problem Statement

Create a card as shown in the picture below. (You can use CSS float property only for layout).

ANS3. Via vs code.

Task 4:

Problem Statement

Create a simple header that sticks to the top of a webpage upon scrolling.

Ans4. ans via code.

Task 5:

Problem Statement

Explain the z-index, with a code example.

Ans5. The `z-index` property in CSS controls the stacking order of elements along the z-axis (which is perpendicular to the screen). Elements with a higher `z-index` value are stacked above elements with a lower value. The `z-index` property only works on elements that have their `position` property set to `absolute`, `relative`, `fixed`, or `sticky`.

Syntax

```css
element {
  z-index: value;
}
```

- **`value`**: Can be an integer (positive, negative, or zero). Higher values stack on top of lower values.

Example

Here's an example to demonstrate the use of the `z-index` property:

HTML:

```html
<div class="box box1">Box 1</div>
```

```
<div class="box box2">Box 2</div>
<div class="box box3">Box 3</div>
```

CSS:
```css
.box {
  width: 100px;
  height: 100px;
  position: absolute;
  color: white;
  padding: 10px;
}

.box1 {
  background-color: red;
  top: 20px;
  left: 20px;
  z-index: 1;
}

.box2 {
  background-color: green;
  top: 50px;
  left: 50px;
  z-index: 3;
}

.box3 {
  background-color: blue;
```

```
  top: 80px;
  left: 80px;
  z-index: 2;
}
```

Explanation

In this example:

- `.box1` (red) has a `z-index` of 1.
- `.box2` (green) has a `z-index` of 3.
- `.box3` (blue) has a `z-index` of 2.

Since `.box2` has the highest `z-index`, it will be on top, followed by `.box3`, and then `.box1` at the bottom.

Visual Representation

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Z-Index Example</title>
  <style>
    .box {
      width: 100px;
      height: 100px;
      position: absolute;
      color: white;
```

```
      padding: 10px;
      font-size: 18px;
    }
    .box1 {
      background-color: red;
      top: 20px;
      left: 20px;
      z-index: 1;
    }
    .box2 {
      background-color: green;
      top: 50px;
      left: 50px;
      z-index: 3;
    }
    .box3 {
      background-color: blue;
      top: 80px;
      left: 80px;
      z-index: 2;
    }
  </style>
</head>
<body>
  <div class="box box1">Box 1</div>
  <div class="box box2">Box 2</div>
  <div class="box box3">Box 3</div>
</body>
</html>
```

Important Notes

1. **Positioning**: The `z-index` property only works on positioned elements (those with `position` set to `relative`, `absolute`, `fixed`, or `sticky`).

2. **Stacking Context**: Elements create a new stacking context in certain conditions, like when they have a `position` of `absolute`, `relative`, or `fixed` and a `z-index` other than `auto`. The stacking order of children elements is determined within their stacking context.

3. **Default `z-index`**: If the `z-index` is not set, it is considered `auto`, and the stacking order is determined by the document's order (later elements stack above earlier ones).

Using `z-index` effectively helps manage the visual layering of elements, ensuring the correct stacking order for your web design needs.