ASSIGNMENT-4

1) How to automatically set the height of div to take the height of parent?

A, Using Flex

For the parent div:

style="display: flex;"

For child div:

style="align-items: stretch;"

B, Using table.

Style=” display: table”;

Table-layout: fixed;

C, Using Grid

Style=” display: grid”;

grid-template-columns: 1fr 1fr 1fr;

2) What is the difference in using px, em, pt, vh, vw? Which is the best measuring unit to use? Explain.

A) **Absolute units**

Absolute units are fixed in its size, you can't discuss how long a centimetre is. If you have a case where the exact length is required, you should use absolute units (for example for components which should not be resized). They can also be useful if you want to define restrictions to avoid that areas become too wide or too narrow. Absolute units do not change according to screen size, direction or other variations.

| Unit | Description |  |
| --- | --- | --- |
| cm | centimetres | 1 cm = 1 cm |
| mm | millimetres | 10 mm = 1 cm |
| in | inches | 1 in = 96px = 2.54 cm |
| px | pixels | 1 px = 1/96th of 1 in |
| pt | points | 1 pt = 1/72 of 1 in |
| pc | pica | 1pc = 12 pt |

**Viewport units**

Viewport units represent a percentage of the current browser viewport.  
The difference to percentage units is, that viewport units always being calculated as the percentage of the browser's viewport size. Whereas percentage units inherit the size of their parent element.

| Unit | Description |
| --- | --- |
| vw | 1% of the viewport's width (50% means the half of the viewport width) |
| vh | 1% of the viewport's height (50% means the half of the viewport height) |
| vmin | 1% of viewport's smaller (vw or vh) dimension |
| vmax | 1% of viewport's larger (vw or vh) dimension |

vmin and vmax can change whilst the browser window is resized or the orientation of the mobile phone is changed.  
vmin is the minimum between the viewport's height or width in percentage depending on which is smaller.

vmax is the maximum between the viewport's height or width in percentage depending on which is bigger.

**Relative units**

As opposed to absolute units like pixels, points or centimetres, you can also define sizes in relative units like percentage, em or rem.  
Relative units also [comply with accessibility standards](https://www.w3.org/WAI/WCAG21/Techniques/css/C28.html).  
In most browsers, the default font size is 16px, you can use this value as a basis for calculations (e.g. 16px equals 1em, 1rem or 100%).

| Unit | Description |
| --- | --- |
| % | percentage |
| em | font size of the element (e.g. 2.5em means the font is 2.5 times bigger than the normal font) |
| rem | font size of the root element of the document |
| ch | width of the "0" character, in mono space fonts, where all characters have the same width, 1ch equals 1 character |
| ex | x-height of the current font, measured at the height of the lower case x |

**What is the difference between em and rem?**

The difference lies in the inheritance. The rem value is based on the root element (html). Every child element uses the html font size as their calculation basis.

em on the other hand, is based on the font size of the parent element.

rem makes the calculation of the font size much easier. With nested elements or even multiple nested elements (e.g. lists), the font size no longer has to be calculated in relation to the font size of the parent element. rem always calculates the font size in relation to the html tag.

**The best measuring unit while using fonts is rem while the best unit for setting height and width is %.**

**Because, rem refers to the parent element for sizing; % refers to the viewport height or viewport width while setting element width or height respectively.**

3) How to draw geometrical shapes using CSS and SVG?

A) Using SVG:

To insert a shape, you create an element in the document. Different elements correspond to different shapes and take different parameters to describe the size and position of those shapes. Some are slightly redundant in that they can be created by other shapes, but they're all there for your convenience and to keep your SVG documents as short and as readable as possible. All the basic shapes are shown in the image to the right. The code to generate that looks something like:



<?xml version="1.0" standalone="no"?>

<svg width="200" height="250" version="1.1" xmlns="http://www.w3.org/2000/svg">

<rect x="10" y="10" width="30" height="30" stroke="black" fill="transparent" stroke-width="5"/>

<rect x="60" y="10" rx="10" ry="10" width="30" height="30" stroke="black" fill="transparent" stroke-width="5"/>

<circle cx="25" cy="75" r="20" stroke="red" fill="transparent" stroke-width="5"/>

<ellipse cx="75" cy="75" rx="20" ry="5" stroke="red" fill="transparent" stroke-width="5"/>

<line x1="10" x2="50" y1="110" y2="150" stroke="orange" stroke-width="5"/>

<polyline points="60 110 65 120 70 115 75 130 80 125 85 140 90 135 95 150 100 145"

stroke="orange" fill="transparent" stroke-width="5"/>

<polygon points="50 160 55 180 70 180 60 190 65 205 50 195 35 205 40 190 30 180 45 180"

stroke="green" fill="transparent" stroke-width="5"/>

</svg>

Rectangle

The [rect](https://developer.mozilla.org/en-US/Web/SVG/Element/rect" \o "en-US/Web/SVG/Element/rect) element draws a rectangle on the screen. There are 6 basic attributes that control the position and shape of the rectangles on screen. The one on the right has its rx and ry parameters set, giving it rounded corners. If they're not set, they default to 0.

<rect x="10" y="10" width="30" height="30"/>

<rect x="60" y="10" rx="10" ry="10" width="30" height="30"/>

**x**

The x position of the top left corner of the rectangle.

**y**

The y position of the top left corner of the rectangle.

**width**

The width of the rectangle

**height**

The height of the rectangle

**rx**

The x radius of the corners of the rectangle

**ry**

The y radius of the corners of the rectangle

Circle

The [circle](https://developer.mozilla.org/en-US/Web/SVG/Element/circle) element draws a circle on the screen. It takes 3 basic parameters to determine the shape and size of the element.

<circle cx="25" cy="75" r="20"/>

**r**

The radius of the circle.

**cx**

The x position of the center of the circle.

**cy**

The y position of the center of the circle.

Ellipse

An [ellipse](https://developer.mozilla.org/en-US/Web/SVG/Element/ellipse) is a more general form of the circle element, where you can scale the x and y radius (commonly refferred to as the semimajor and semiminor axis in maths) of the circle separately.

<ellipse cx="75" cy="75" rx="20" ry="5"/>

**rx**

The x radius of the ellipse.

**ry**

The y radius of the ellipse.

**cx**

The x position of the center of the ellipse.

**cy**

The y position of the center of the ellipse.

Line

The [line](https://developer.mozilla.org/en-US/Web/SVG/Element/line) element takes the positions of two points as parameters and draws a straight line between them.

<line x1="10" x2="50" y1="110" y2="150"/>

**x1**

The x position of point 1.

**y1**

The y position of point 1.

**x2**

The x position of point 2.

**y2**

The y position of point 2.

Polyline

A [polyline](https://developer.mozilla.org/en-US/Web/SVG/Element/polyline) is a group of connected straight lines. Since the list of points can get quite long, all the points are included in one attribute:

<polyline points="60 110, 65 120, 70 115, 75 130, 80 125, 85 140, 90 135, 95 150, 100 145"/>

A list of points, each number separated by a space, comma, EOL, or a line feed character. Each point must contain two numbers, an x coordinate and a y coordinate. So the list (0,0), (1,1) and (2,2) could be written: "0 0, 1 1, 2 2".

Polygon

A [polygon](https://developer.mozilla.org/en-US/Web/SVG/Element/polygon) is similar to a polyline in that it is composed of straight line segments connecting a list of points. For polygons though, the path automatically connects the last point with the first, creating a closed shape. Note that a rectangle is a type of polygon, so a polygon can be used to create a <rect/> element in cases where you need a little more flexibility.

<polygon points="50 160, 55 180, 70 180, 60 190, 65 205, 50 195, 35 205, 40 190, 30 180, 45 180"/>

A list of points, each number separated by a space, comma, EOL, or a line feed character. Each point must contain two numbers, an x coordinate and a y coordinate. So the list (0,0), (1,1) and (2,2) could be written: "0 0, 1 1, 2 2". The drawing then closes the path, so a final straight line would be drawn from (2,2) to (0,0).

Path

A [path](https://developer.mozilla.org/en-US/Web/SVG/Element/path) is probably the most general shape that can be used in SVG. Using a path element, you can draw rectangles (with or without rounded corners), circles, ellipses, polylines, and polygons. Basically any of the other types of shapes, bezier curves, quadratic curves, and many more. For that reason, [the next section](https://developer.mozilla.org/en-US/Web/SVG/Tutorial/Paths) in this tutorial will be focused on paths, but for now, note that there is a single parameter used to control its shape.

<path d="M20,230 Q40,205 50,230 T90,230" fill="none" stroke="blue" stroke-width="5"/>

Using CSS:

inset()

The inset() type defines a rectangle, which may not seem very useful as simply floating an item will give you a rectangular shape around it. However the inset() types enables the definition of offsets, thus pulling the content in over the shape.

Therefore inset() takes four values for the top, right, bottom and left values plus a final value for border-radius. The below CSS creates a rectangular shape inset from the reference box of the floated element 20 pixels from the top and bottom and 10 pixels from the left and right, with a border-radius value of 10 pixels.

.shape {

float: left;

shape-outside: inset(20px 10px 20px 10px round 10px);

}

circle()

The circle() value for shape-outside can accept two possible arguments. The first is the shape-radius.

Both circle() and ellipse() values for shape-outside are specified as accepting this argument of <shape-radius>. This argument can be a length or percentage but can also be one of the keywords closest-side or farthest-side.

The keyword **closest-side** uses the length from the centre of the shape to the closest side of the reference box. For circles, this is the closest side in any dimension. For ellipses, this is the closest side in the radius dimension.

The keyword **farthest-side** uses the length from the centre of the shape to the farthest side of the reference box. For circles, this is the farthest side in any dimension. For ellipses, this is the farthest side in the radius dimension.

The second argument is a position. If omitted this will be set to centre. However, you can use any valid position here to indicate the position of the centre of the circle.

Our circle therefore accepts one radius value, which may be a length, a percentage or the closest-side or farthest side keyword then optionally the keyword **at** followed by a position value.

img {

float: left;

shape-outside: circle(50% at 60%);

}

ellipse()

An ellipse is essentially a squashed circle and so ellipse() acts in a very similar way to circle() except that we have to specify two radii x and y in that order.

These may then be followed by position values as with circle() to move the centre of the ellipse around. In the example below we have an ellipse with an x radius of 40%, a y radius of 50% and the position being left.

Example:

.shape {

float: left;

shape-outside: ellipse (40% 50% at left);

margin: 20px;

width: 100px;

height: 200px;

}

polygon()

The final Basic Shape is the most complex and enables the creation of many side shapes by way of creating a polygon(). This shape accepts three or more pairs of values (a polygon must at least draw a triangle).

Example:

.shape {

float: left;

shape-outside: polygon(0px 0px, 0px 189px, 100.48% 94.71%, 200px 120px, 80.67% 37.17%);

width: 200px;

height: 200px;

}