HTML & CSS Part 2

=> media query
=> floating
=> positioning
=> normalizers
=> Semantic tags
=> CSS - flexbox

What do you remember from the last lesson?

1. How to select all p element with direct “div” parent?
2. How to select <p> elements that are placed immediately after elements(first sibling)?
3. How to select element with 2 classes, “.list” and “.list_main”?
4. How to select element with class “.list_item” that is inside element with class “.list”?
5. Name 5 pseudo classes?
6. Name 3 pseudo elements?
7. Name 5 responsive units?
8. How to select the third “p” tag?

Media query

“The @media rule is used in media queries to apply different styles for different media types/devices.”

– W3school

Conditions types

=> width and height of the viewport

=> media type

=> orientation (is the tablet/phone in landscape or portrait mode?)

=> resolution

Use **“and”** to concat rules

Media types

=> all(default)

=> print

=> screen

=> speech

Commonly used

Screen

```
@media screen and (min-width: 400px) {
  body {
    background-color: lightgreen;
  }
}
```

Print

```
@media print {
  body {
    background-color: lightgreen;
  }
}
```

Most popular media features

=> min-width

=> max-width

=> orientation

More details – https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

When we use min-width/max-width

=> min-width - when you build mobile first website

=> max-width - when you build desktop first

Float

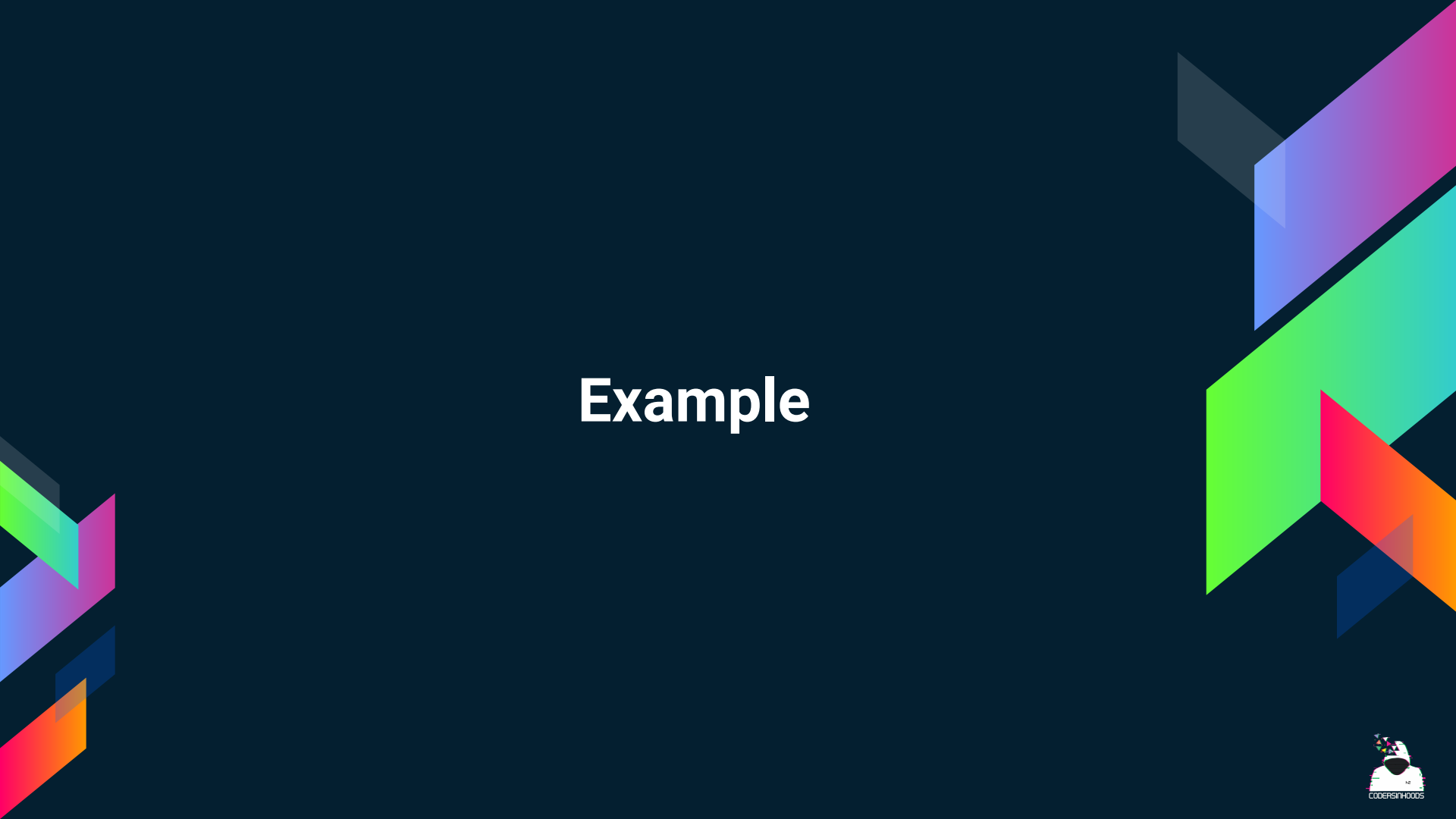
=> left - The element floats to the left of its container

=> right - The element floats to the right of its container

=> none - The element does not float (will be displayed just where it occurs in the text). This is default

=> inherit - The element inherits the float value of its parent

Example



With “float” we will get an issue.

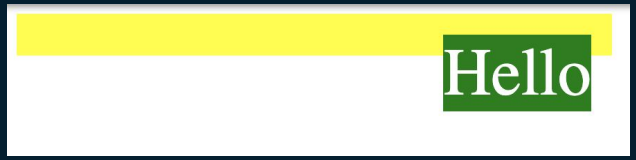
If you apply float(left, right), you remove this element from normal document flow and a parent element loses height

Without “float”

```
1 div {
2   background: yellow;
3   padding: 15px;
4   width: 400px;
5 }
6
7 p {
8   font-size: 3em;
9   color: white;
10  margin: 0;
11  background: green;
12 }
```

With “float”

```
1 div {
2   background: yellow;
3   padding: 15px;
4   width: 400px;
5 }
6
7 p {
8   font-size: 3em;
9   color: white;
10  margin: 0;
11  float: right;
12  background: green;
13 }
```



First solution for siblings

The “clear” property can have one of the following values:

- none - Allows floating elements on both sides. This is default
- left - No floating elements allowed on the left side
- right- No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- inherit - The element inherits the clear value of its parent

Second solution for nested elements

```
1 div {
2   background: yellow;
3   padding: 15px;
4   width: 400px;
5   overflow: auto;
6 }
7 p {
8   font-size: 3em;
9   color: white;
10  margin: 0;
11  float: right;
12  background: green;
13 }
```

Read more

https://www.w3schools.com/css/css_float.asp

Positioning

“The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).”

=> static

=> relative

=> fixed

=> absolute

=> sticky

Static

All elements positioned static by default.

Static elements positioned according to a normal flow.
(properties top, left, right, bottom are not applicable)

Relative

Elements with a “relative” position are in their normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.

Fixed

Elements with position “fixed” has position relative to viewport.

Absolute

Elements with position “absolute” are positioned relative to the closest non-static parent element.

What is the difference between Relative and Absolute?

Sticky

Elements with position “sticky” based on user scroll position.

Sticky elements toggle between relative and fixed. They are positioned relative until a given offset position is met in the viewport, then they behave like elements with position fixed. (top and bottom are applicable)

Normalizers

https://necolas.github.io/normalize.css/

Bonus

Pixel perfect - https://cantunsee.space/

Semantic tags - *tags with a meaning*

Most popular

=> <header>

=> <nav>

=> <section>

=> <article>

=> <aside>

=> <footer>

More

=> <details>

=> <figcaption>

=> <figure>

=> <main>

=> <mark>

=> <summary>

<header> – element specifies a header for a document or section. Should be used as a container for introductory content.

<nav> – element defines a set of navigation links

<section> – element defines a section in a document(grouping similar content)

<article> – element specifies independent, self-contained content(self-sufficient content)

<aside> – element defines some content aside from the content it is placed in (like a sidebar)

<footer> – element specifies a footer for a document or section.

** You can find examples in git folder: /lesson_5/examples/tags*

<details> – tag specifies additional details that the user can view or hide. Can be used for collapsible widget etc.

<summary> – tag defines a visible heading for the <details> element

<figure> – tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.

<figcaption> – tag defines a caption for a <figure> element

<main> – tag specifies the main content of a document

<mark> – tag defines marked text

** You can find examples in git folder: /lesson_5/examples/tags*

Keep in mind

=> use ****** *not* ***<i>*** to emphasize text

=> use ****** *not* ***<b>*** to make your text bold

=> use ***<button>*** *not* ***<a>*** when you need a button

=> use ***<form>*** *not* *** *** when you are creating a form

=> use **<label>** for each **<input>**

=> use **<fieldset>** not ** ** to group sections in a form

Tabindex

The tabindex global attribute indicates if its element can be focused, and if/where it

– MDN

** You can find examples in git folder: /lesson_5/examples/tags*

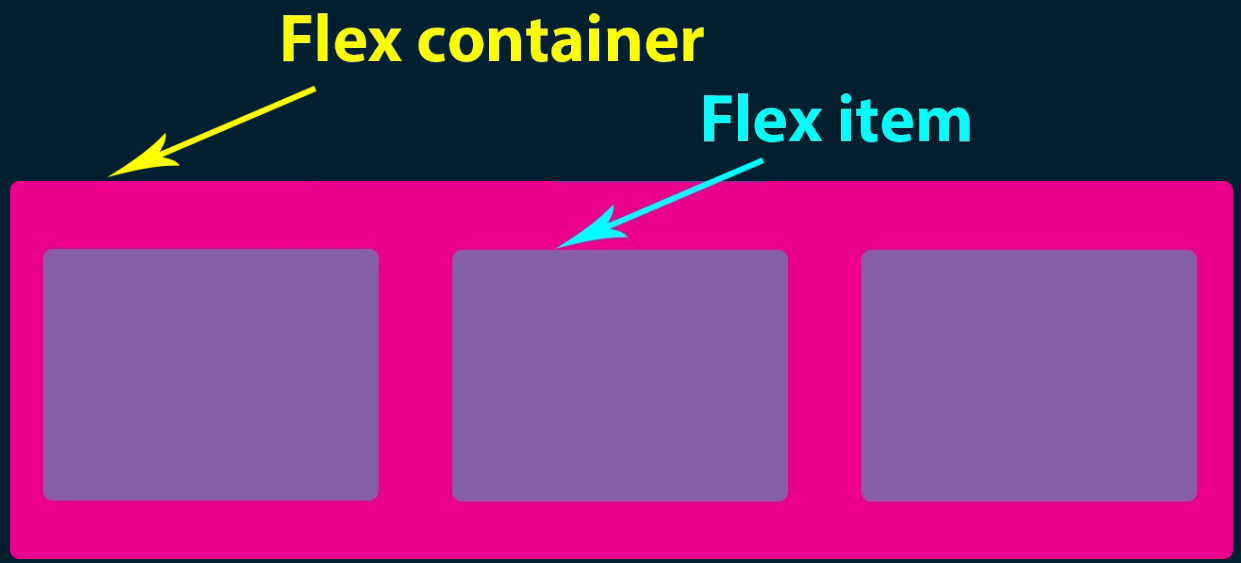
FLEXBOX

Flexbox is “content first”, that means, that you have a content and you want to fit it.
(CSS Grid is layout first - when you have layout and then you put content).

Flexbox has 2 types of elements:

=> flex container

=> flex item



To define a “flex container” all we need to do is to set *“display: flex”*

“flex container” properties:

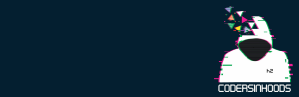
- => *Flex-direction*
- => *Flex-wrap*
- => *Flex-flow*
- => *Justify-content*
- => *Align-items*
- => *Align-content*

“flex item” properties:

- => *Order*
- => *Flex-grow*
- => *Flex-shrink*
- => *Flex-basis*
- => *Flex*
- => *Align-self*

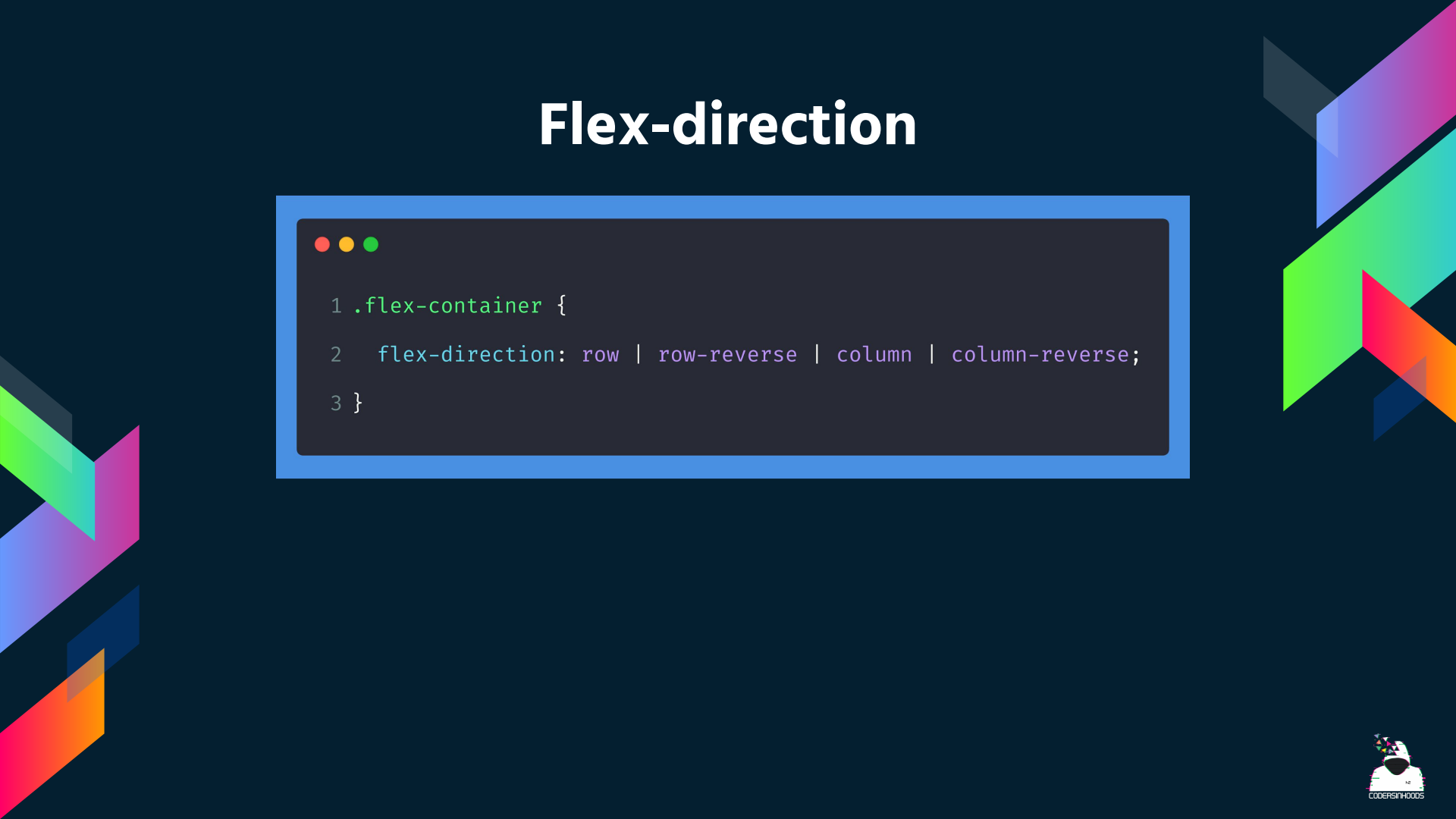
** You can find examples in git folder to this lesson*

Flex container properties



Flex-direction

```
.flex-container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```



Flex-wrap

```
1 .flex-container {
2   flex-wrap: nowrap | wrap | wrap-reverse;
3 }
```

nowrap (default) – *all flex items will be on one line*
wrap – *flex items will wrap onto multiple lines, from top to bottom.*
wrap-reverse – *flex items will wrap onto multiple lines from bottom to top.*

Flex-flow

```
.flex-container {
  flex-flow: <‘flex-direction’> || <‘flex-wrap’>
}
```

Justify-content

```
.flex-container {
  justify-content:  flex-start | flex-end | center |
                    space-between | space-around |
                    space-evenly;
}
```

flex-start (default) – *items are packed toward the start line*
flex-end – *items are packed toward the end line*
center – *items are centered along the line*
space-between – *items are evenly distributed in the line; first item is on the start line, last item on the end line*

Space-around – *items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.*
Space-evenly – *items are distributed so that the spacing between any two items (and the space to the edges) is equal.*

Align-items

```css
.flex-container {
    align-items: stretch | flex-start | flex-end | center | baseline;
}
```

stretch (default) – *stretch to fill the container (still respect min-width/max-width)*
flex-start – *cross-start margin edge of the items is placed on the cross-start line*
flex-end – *cross-end margin edge of the items is placed on the cross-end line*
center – *items are centered in the cross-axis*
baseline – *items are aligned such as their baselines align*

Align-content

```
.flex-container {
    align-content:  flex-start | flex-end | center |
                    space-between | space-around | stretch;
}
```

flex-start – *lines packed to the start of the container*
flex-end – *lines packed to the end of the container*
center – *lines packed to the center of the container*
space-between – *lines evenly distributed; the first line is at the start of the container while the last one is at the end*
space-around – *lines evenly distributed with equal space around each line*
stretch (default) – *lines stretch to take up the remaining space*

Flex item properties

Order

```
.flex-item {
    order: <integer>; /* default is 0 */
}
```

Flex-grow

```
1 .flex-item {
2     flex-grow: <number>; /* default 0 */
3 }
```

If all items have flex-grow set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, the remaining space would take up twice as much space as the others

Flex-shrink

```
1 .flex-item {
2     flex-shrink: <number>; /* default 0 */
3 }
```

Flex-basis

```
.flex-item {
    flex-basis: <length> | auto; /* default auto */
}
```

This defines the default size of an element before the remaining space is distributed.

Flex

```
.flex-item {
    flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

This is the shorthand for flex-grow, flex-shrink and flex-basis combined. The second and third parameters (flex-shrink and flex-basis) are optional. Default is 0 1 auto.

Align-self

```
.flex-item {
    align-self: auto | flex-start | flex-end | center | baseline | stretch;
}
```

This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items. Please see the align-items explanation to understand the available values.

Bonus

FlexboxFroggy - https://flexboxfroggy.com/