Class 17: CSS Grid Layout

- Review: User Interviews
- CSS Grid Layout
- Line-Based Item Placement
- Named Template Areas
- Item Alignment
- Office Hours: Project, Milestone 1 (or 2)

Copyright 2026, Kyle J. Harms

Review: User Interviews

1. Plan interview questions
2. Practice/pilot interview
3. Recruit participants
4. Obtain consent to interview
5. Interview users
6. Analyze interview data – are the goals *relatively* uniform?
7. If necessary, conduct another round of user interviews

Review: Effective Interview Questions

- Ask participants to **recall** specific events/memories.
  - “ Recall the last festival or community event you attended. Tell me a bit about it. ”
- Ask **follow-up** questions.
  - “ I see, you visited Porchfest this year. Tell me how you learned about it? ”
- Keep questions **neutral**.
  - “ Describe any challenges you faced at Porchfest. ”

Review: Ineffective Interview Questions

- Avoid **leading** questions.

  → Assumes that some information is useful.

- Avoid asking the participant to **speculate**.

  → Humans are bad at speculating. Ask them how they did *x* **today**.

- Avoid asking users **what they want**.

  → Users don't know what they want. Explore their behavior instead.

- Avoid **closed** questions (yes/no answer).

  → Closed questions don't provide information. Provide space to talk.

- Avoid asking **design** related questions.

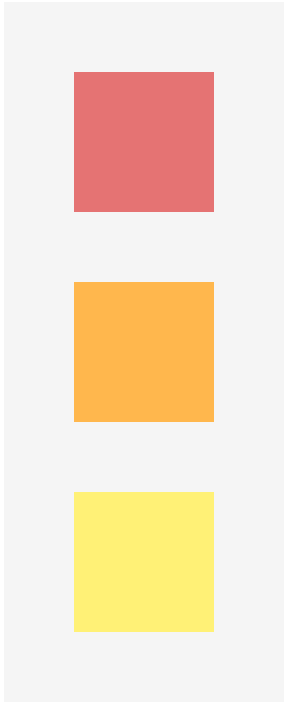
  → You're the designer! That's your job. Don't ask design questions!

CSS Grid Layout

Review: Flexbox

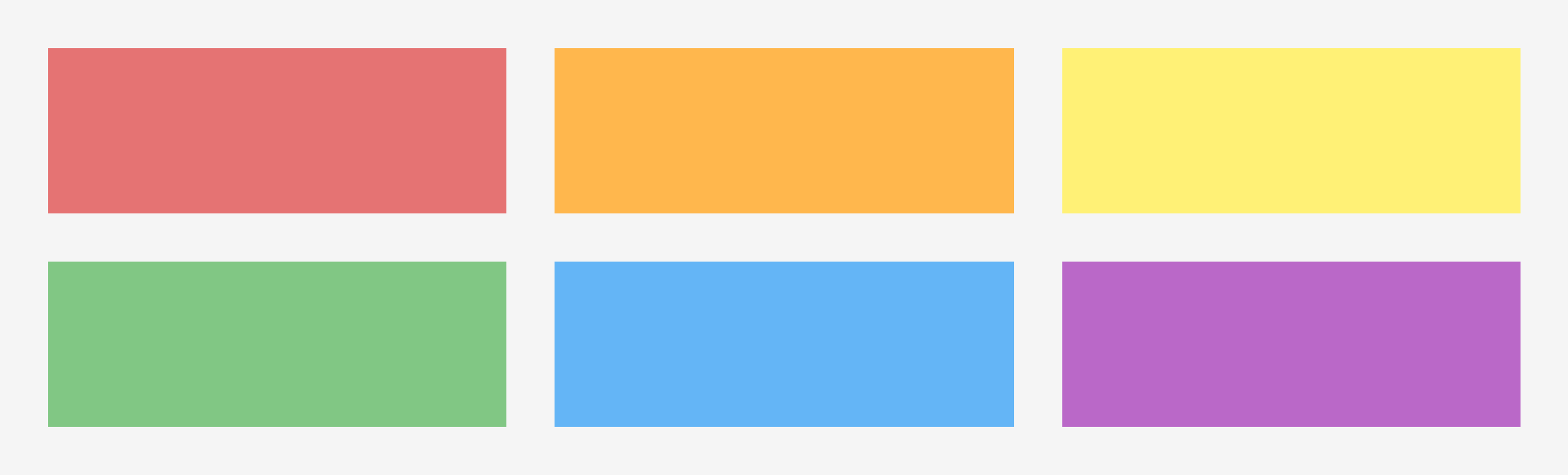
A *nestable* **one-dimensional** layout **container** model.





CSS Grid Layout

A **two-dimensional** layout **container** model.



Grid Requirements

Container: The parent element that contains the grid items.

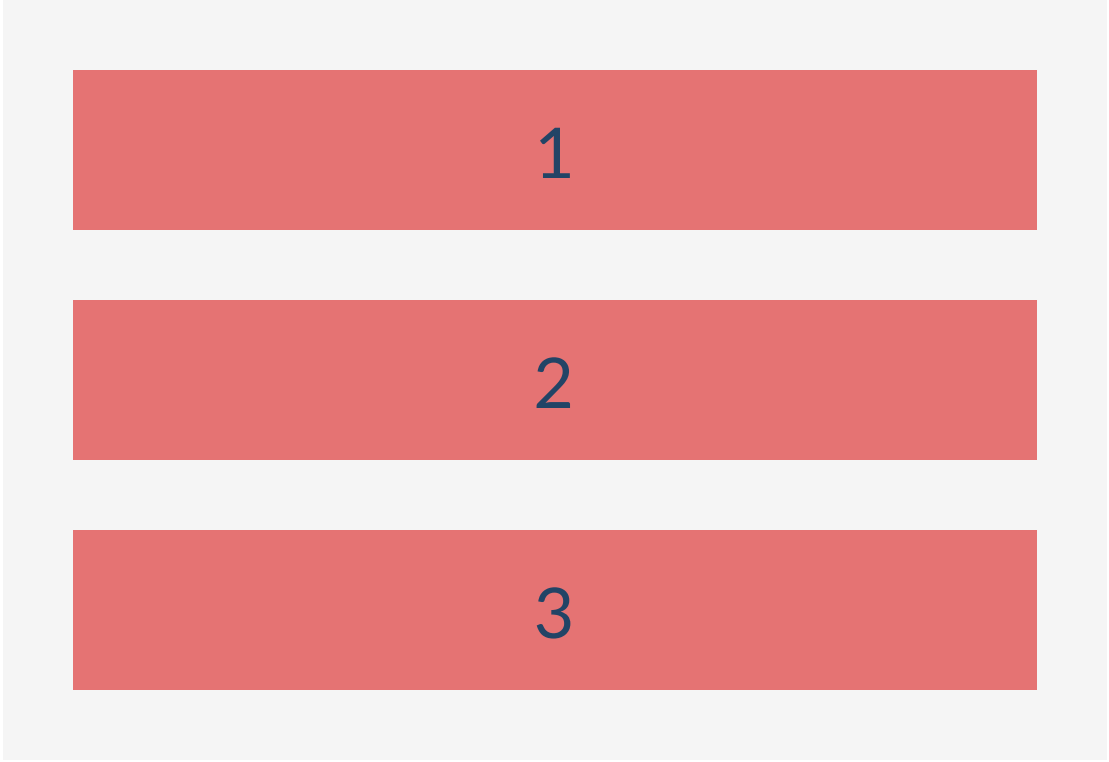
Grid Items: The child elements that are placed in the grid.
(*Direct* children, not grandchildren.)

Grid Container: `display: grid`

Set element as a **grid container**:

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

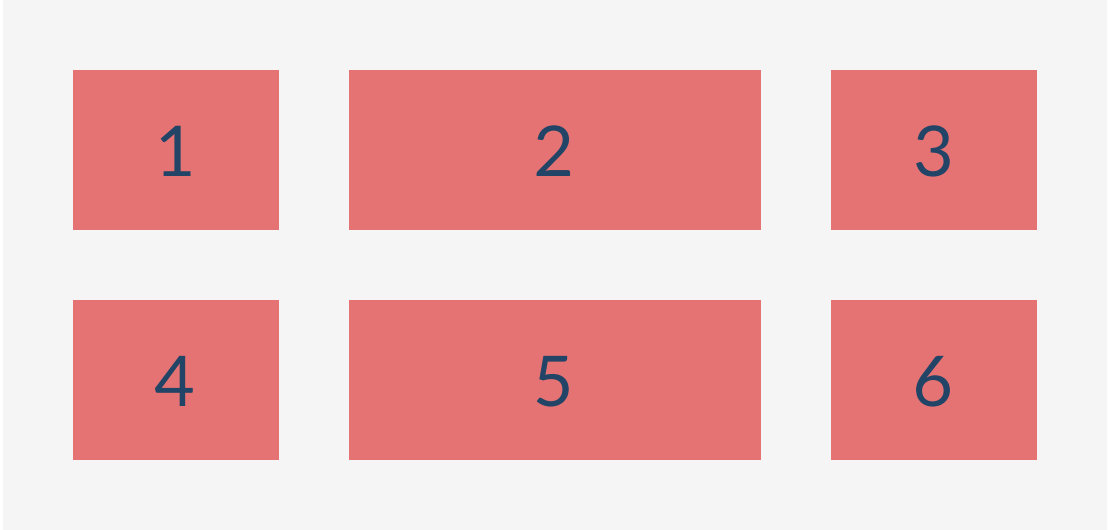
```html
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```



Grid Columns: `grid-template-columns`

```css
.grid-container {
  display: grid;
  /* 3 columns */
  grid-template-columns: 1fr 2fr 1fr;
}
```

```html
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```



Unit: `fr`

`fr` : A *fraction* of the remaining space in the grid container.

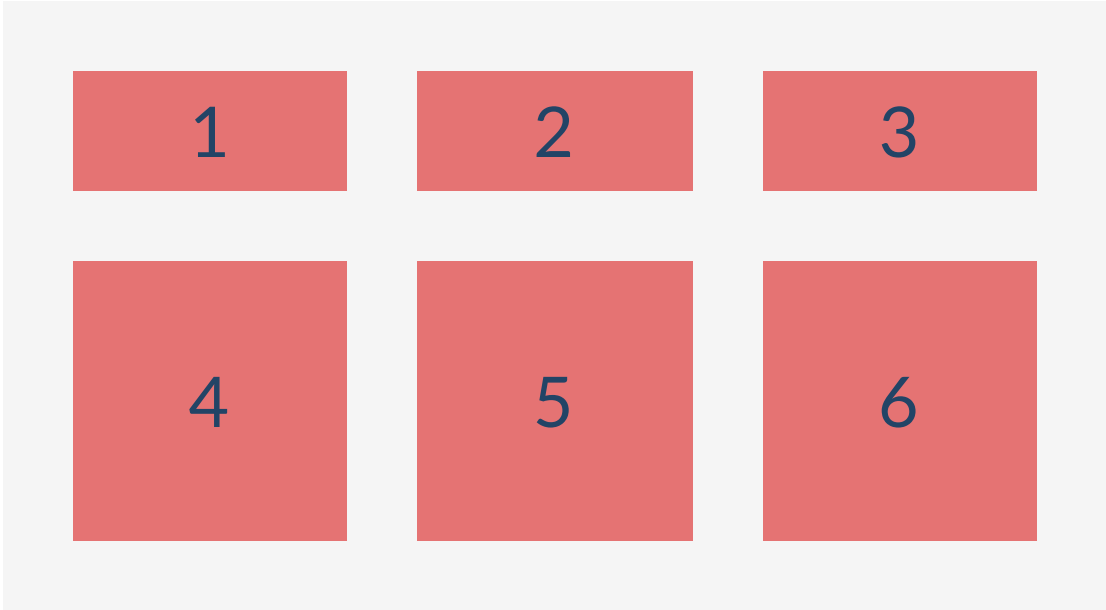
Example: `1fr 2fr 1fr` → (4 parts) ¼, ½, ¼

You may mix units: `grid-template-columns: 200px 1fr 200px;`
(fixed width side columns + middle columns fill remaining space)

Grid Rows: `grid-template-rows`

```css
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 60px 140px;
}
```

```html
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```



Gutters: `gap`

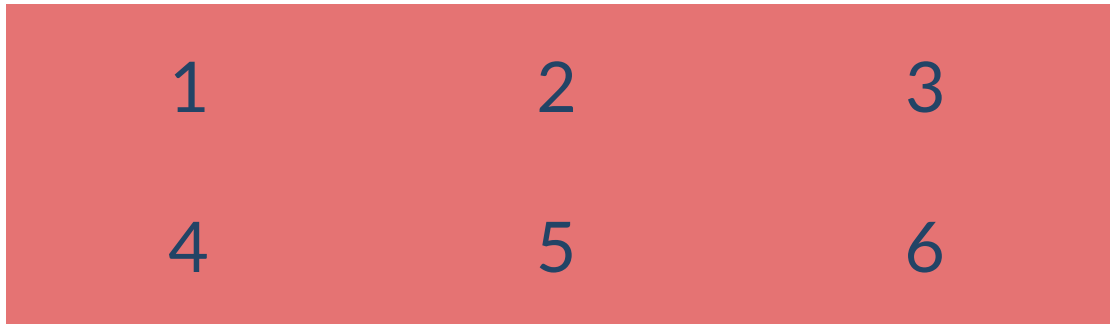
The space *between* rows and columns.

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 1em;
}
```

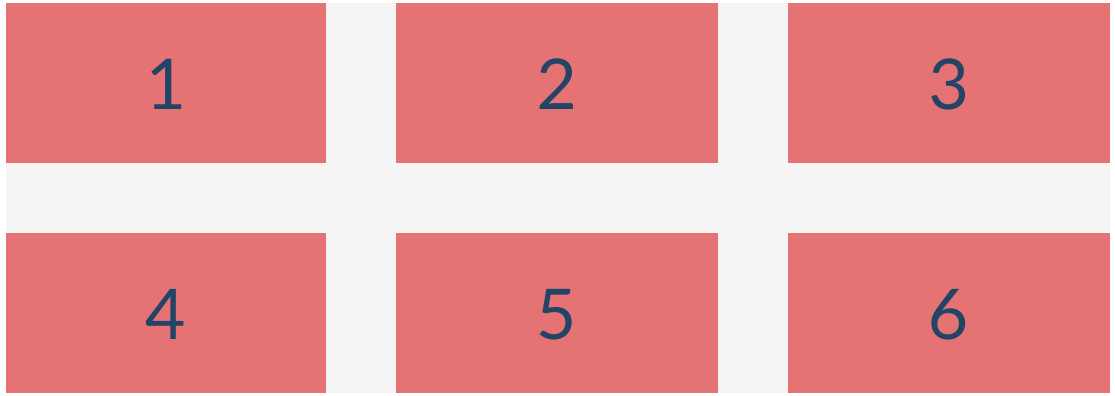
Different row and column gutters:

```
row-gap: 2em;
column-gap: 1em;
```

Default `gap: 0;`



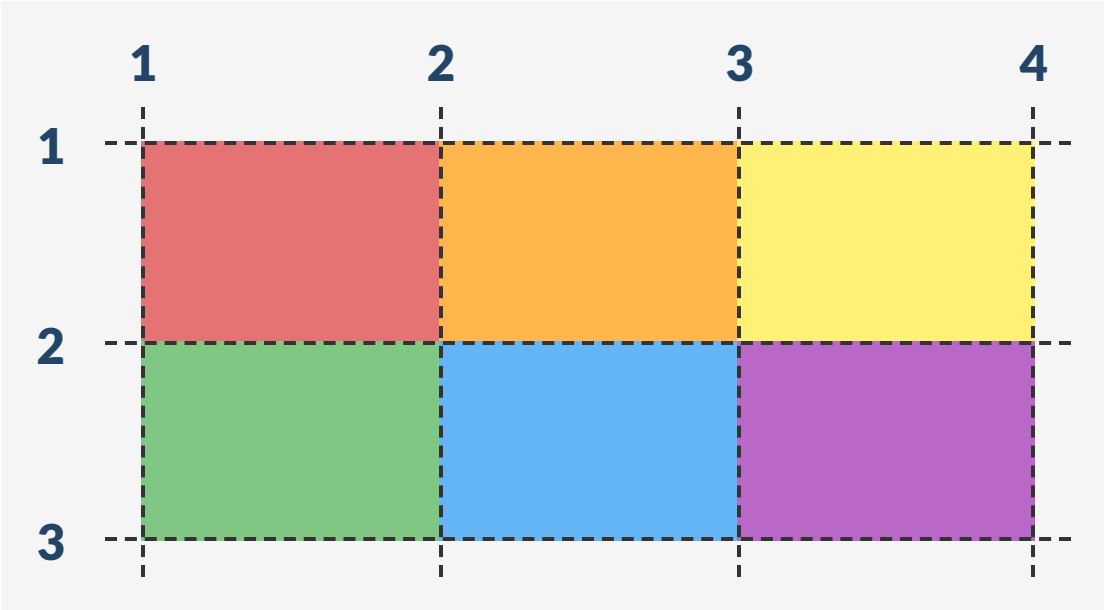
`gap: 1em;`



Grid Lines

Grid **lines** are the edges between rows and columns.

Lines are **numbered**, starting at **1**.



Note:

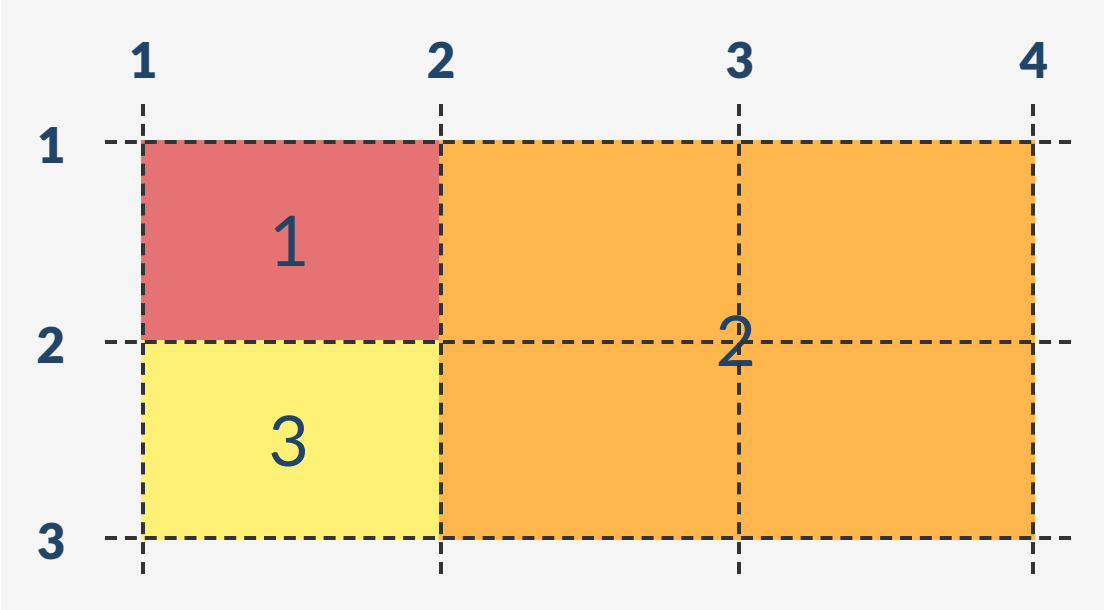
3 columns → 4 column lines

2 rows → 3 row lines

Line-Based Item Placement

```css
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
.feature {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
```

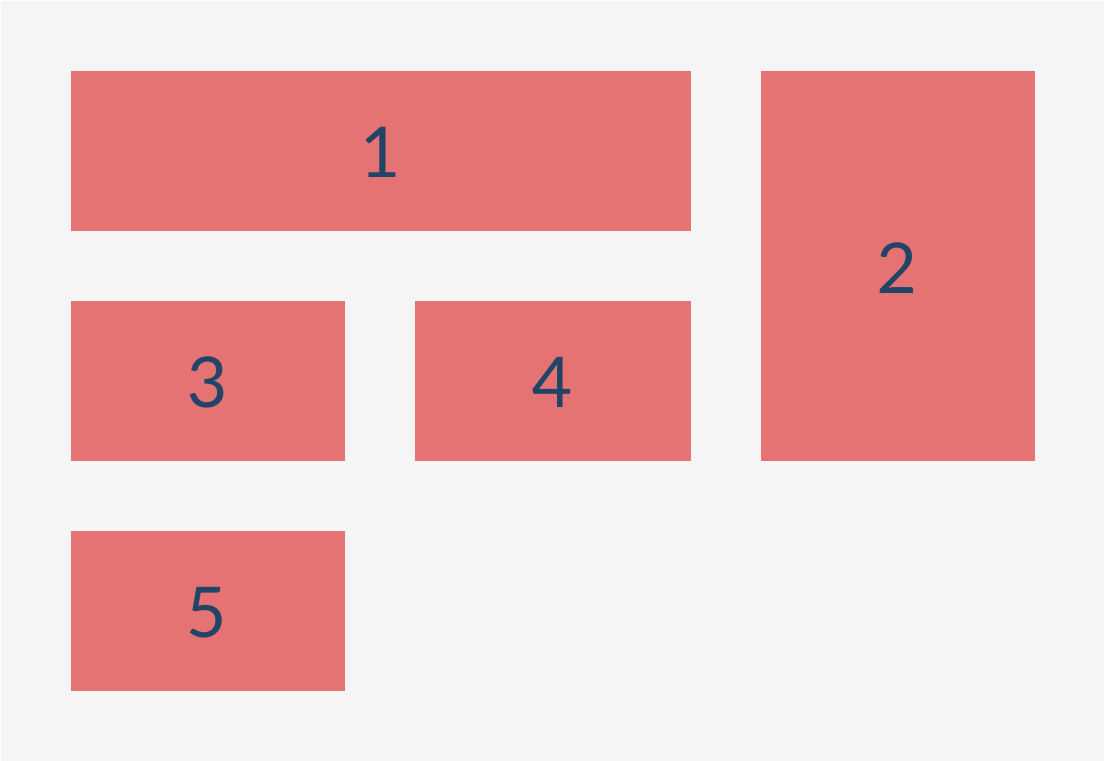
```html
<div class="grid-container">
  <div>1</div>
  <div class="feature">2</div>
  <div>3</div>
</div>
```



Spanning Rows & Columns

```css
.grid-container {
  display: grid;
  gap: 1em;
  grid-template-columns: 1fr 1fr 1fr;
}
.wide {
  grid-column: span 2;
}
.tall {
  grid-row: span 2;
}
```

```html
<div class="grid-container">
  <div class="wide">1</div>
  <div class="tall">2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
</div>
```



Activity: Grid (Spanning) Practice

Working with a peer, complete Part I on the handout.

Use the following HTML structure for the activity:

```
<body>
  <header>...</header>
  <nav>...</nav>
  <main>...</main>
  <aside>...</aside>
  <footer>...</footer>
</body>
```

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
.wide {
  grid-column: span 2;
}
.tall {
  grid-row: span 2;
}
```

```
<div class="grid-container">
  <div class="wide">1</div>
  <div class="tall">2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
</div>
```

Example: Responsive Spanning Columns

```
body {
  display: grid;
}

@media (min-width: 768px) {
  body {
    grid-template-columns: 1fr 3fr 1fr;
  }

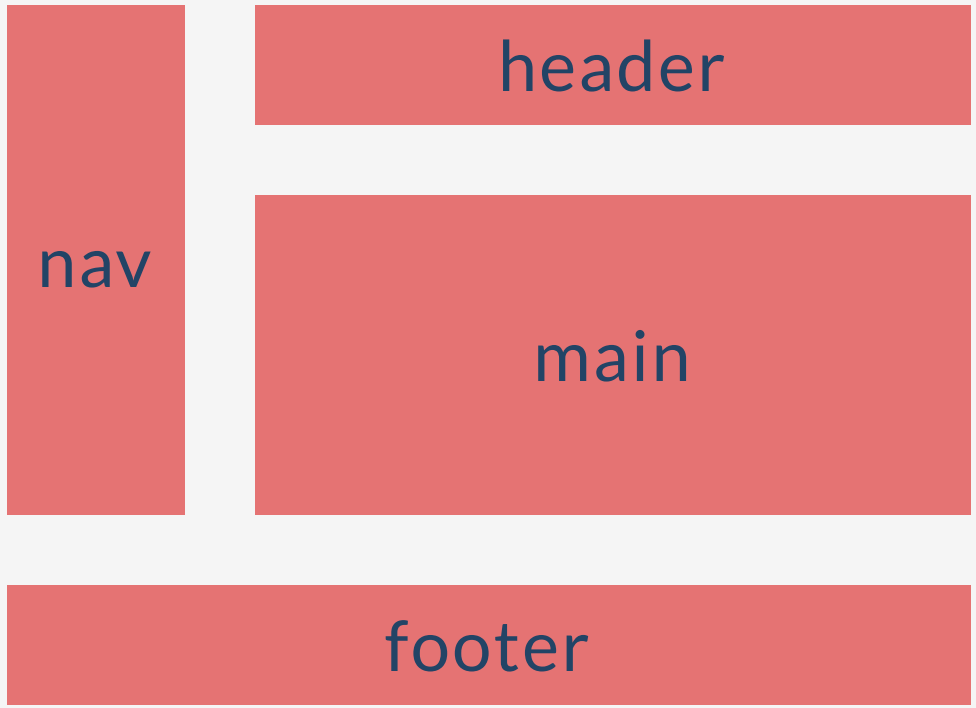
  header,
  footer {
    grid-column: span 3;
  }
}
```

Named Template Areas

`grid-area` **names** a grid item. Layout `grid-areas` using `grid-template-areas`.

```
.grid-container {
  display: grid;
  gap: 1em;
  grid-template-columns: 1fr 4fr;
  grid-template-areas:
    'nav    header'
    'nav    main'
    'footer footer';
}

header { grid-area: header; }
nav    { grid-area: nav; }
main   { grid-area: main; }
footer { grid-area: footer; }
```



Activity: Named Template Areas Practice

Working with a peer, complete Part II (back) on the handout.

Use the following HTML structure for the activity:

```
<body>
  <header>...</header>
  <nav>...</nav>
  <main>...</main>
  <aside>...</aside>
  <footer>...</footer>
</body>
```

Example:

```
.grid-container {
  display: grid;
  grid-template-columns: 1fr 4fr;
  grid-template-areas:
    'nav    header'
    'nav    main'
    'footer footer';
}

header { grid-area: header; }
nav    { grid-area: nav; }
main   { grid-area: main; }
footer { grid-area: footer; }
```

Example: Responsive Named Template Areas

```css
body {
  display: grid;
  grid-template-areas:
    "header"
    "nav"
    "main"
    "aside"
    "footer";
}

header { grid-area: header; }
nav    { grid-area: nav; }
main   { grid-area: main; }
aside  { grid-area: aside; }
footer { grid-area: footer; }
```

```css
@media (min-width: 768px) {
  body {
    grid-template-columns: 1fr 3fr 1fr;
    grid-template-areas:
      "header header header"
      "nav    main   aside"
      "footer footer footer";
  }
}
```

Aligning Grid Items

By default, grid items **stretch** to fill their cell.

Align items *inside* their cells:

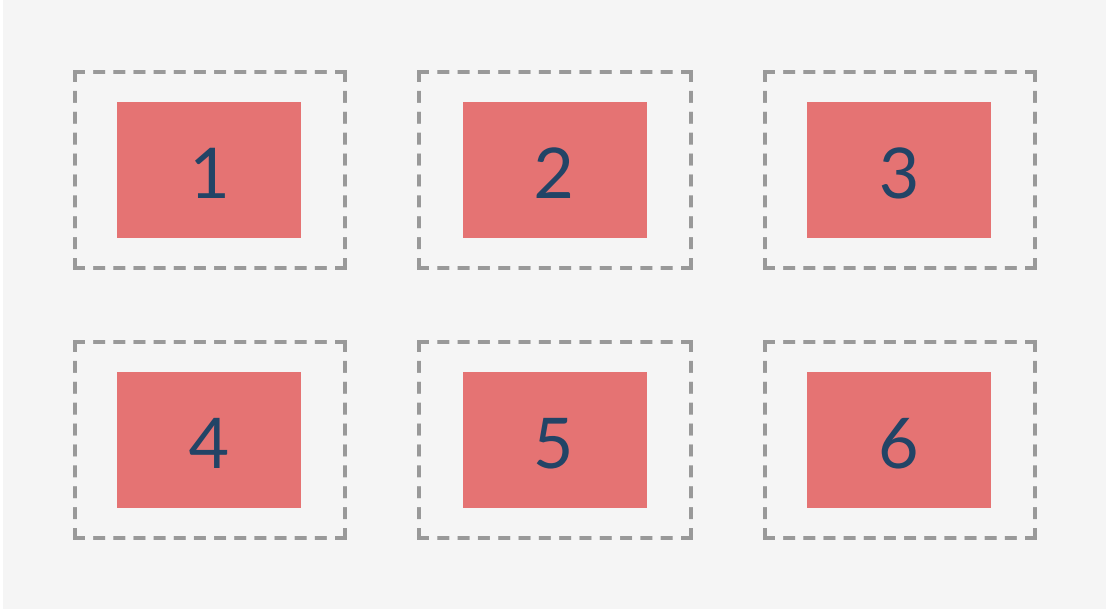
Property	Direction	Values	Default
`justify-items`	horizontal (↔)	`start`, `center`, `end`	`stretch`
`align-items`	vertical (↕)	`start`, `center`, `end`	`stretch`

Tip: `place-items: center;` sets both.

Aligning Items: Container

Set on the **container**; applies to **all** items:

```
.grid-container {
  display: grid;
  gap: 1em;
  grid-template-columns: 1fr 1fr 1fr;
  justify-items: center;
  align-items: center;
}
```

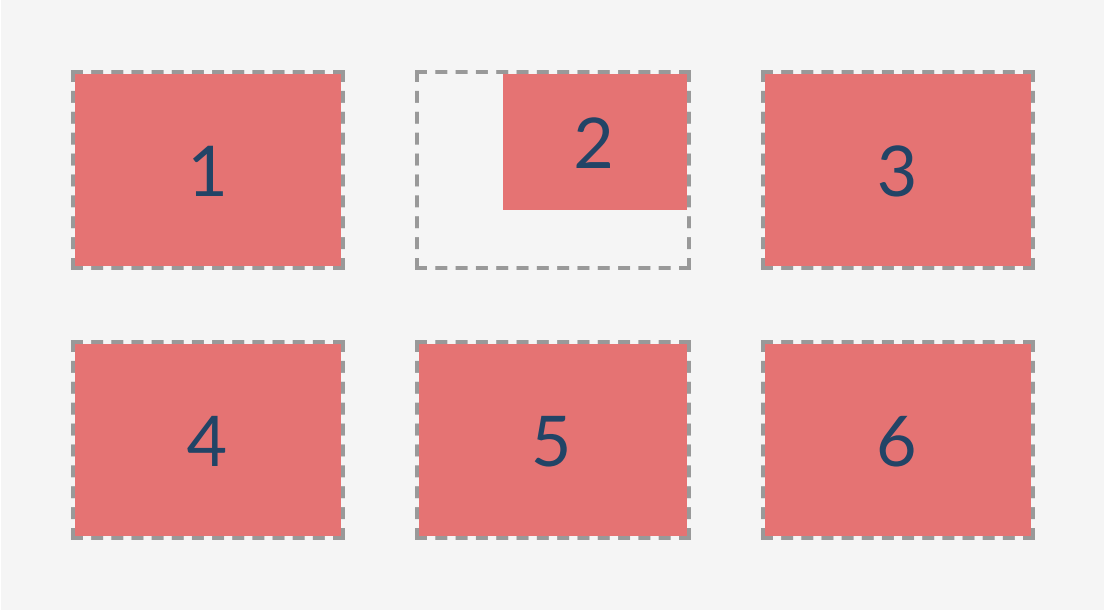


Aligning Items: One Item

Set on the **item**:

```
.corner {
  justify-self: end;
  align-self: start;
}
```

```
<div class="grid-container">
  <div>1</div>
  <div class="corner">2</div>
  <div>3</div>
  ...
</div>
```



Nesting Layouts

Like Flexbox, only the **direct children** of the container are grid items.

You can nest flexbox and grid containers to create more complex layouts:

- A grid item can be a **grid** container.
- A grid item can be a **flexbox** container.
- A flex item can be a **grid** container.

Flexbox or Grid?

Flexbox: One dimension. Content decides the size.
(e.g., a row of nav links, a card's contents)

Grid: Two dimensions. The layout decides the size.
(e.g., gallery of cards)

Most sites use **both**.

Caution: Project – Grid

The project requires **nested flexboxes** for credit.

Grid use is optional; no credit is provided for CSS grid (no grid homework).

Make sure you meet the nested flexbox requirements if you do decide to use grid too.

Milestone 2

In milestone 2, you will implement a **professional narrow** screen version of your planned design.

Responsive methods are prohibited for this milestone; you may not implement a responsive website (milestone 3 is the responsive implementation).

Office Hours

Project, Milestone 1 (or 2)

What's Next

Project, Milestone 1 due today

Wednesday: Designing Usable Website Interactivity

Friday Sections (10/30): Project 2 – Peer Feedback

Project, Milestone 2 due Monday (11/2)