**Тест 💎**

1. Почему в большинстве ситуаций стоит использовать flexbox-позиционирование?

Flexbox предназначен для **создания гибких макетов**. С помощью этой технологии можно очень просто и гибко расставить элементы в контейнере, распределить доступное пространство между ними, и выровнять их тем или иным способом даже если они не имеют конкретных размеров.

Flexbox позволяет создать адаптивный дизайн намного проще, чем с использованием других видов позиционирования.

1. Самостоятельно изучите способ позиционирования через display: table и ответьте на вопрос, для каких ситуаций оно лучше всего подходит?
   1. С его помощью можно получить истинное выравнивание по вертикали элементов с динамической высотой.
   2. Динамическое горизонтальное выравнивание по центру
   3. С помощью display: table можно сделать прилипающий подвал с динамической высотой.
2. Какими способами можно сделать горизонтальное выравнивание по центру? Минимум 3 варианта, можно больше

* **Свойство text-align (используется, но реже).** Добавить класс *center* со свойством *text-align: center* к родительскому блоку и отобразим дочерний блок, как *inline-block*.

## **Свойство margin: auto (актуальное).** Данное свойство мы прописываем дочернему блоку. В сокращенной записи верхнему и нижнему отступу мы можем оставить нулевое значение или указать нужные размеры согласно макету для верстки. Но нас интересует значение *auto*, как раз оно и задает автоматический отступ с левой и с правой стороны от вложенного блока. Это значит, что пространство с обеих сторон должно рассчитываться поровну. Этот способ выравнивания и по сей день является самым популярным у верстальщиков при горизонтальном выравнивании.

## **Свойство position: absolute (устаревшее).** Когда к дочернему блоку применяется *position: absolute*, то оно перестает влиять на родителя. Поэтому для родителя мы задаем *position: relative*, чтобы позиционироваться от него. А ребенку задается *position: absolute*, смещаем его на 50% влево и задаем отступ на половину его ширины. Не рекомендую использовать этот способ, так как он противоречит принципу независимого переиспользования блоков.

## **Свойство flexbox (популярное). Flexbox** - самый простой и современный способ **горизонтального выравнивания по центру**. Для родителя добавляются два флексовых свойства.

* + <div class="wrap flexbox">  
        <div class="block"></div>  
    </div>  
      
    .flexbox {     display: flex;  
        justify-content: center;  
      }

1. Какие есть оси во флекс-верстке и как задается их направление?

В Flexbox имеются две оси. Первая ось называется **главной** (по умолчанию она направлена слева направо). Вторая - **поперечная** (по умолчанию направлена сверху вниз), **она всегда перпендикулярно главной**. Главная ось задаёт основное направление flex-элементов во flex-контейнере, а поперечная ось определяет их направление при переносе на новую линию.

По умолчанию элементы во flex-контейнере располагаются вдоль направления главной оси (т.е. слева направо) на одной линии.

Направление главной оси можно изменить, осуществляется это с помощью CSS-свойства flex-direction.

flex-direction: row;

/\* row (слева направо) - по умолчанию

row-reverse (справа налево)

column (сверху вниз)

column-reverse (снизу вверх) \*/

С помощью этого свойства можно сделать так, чтобы flex-элементы располагались не рядами (rows), а колонками (columns). Осуществляется это с помощью значения column или column-reverse.

1. Разберитесь, как работает свойство margin: auto во флекс-верстке, приведите пример использования

C его помощью можно отцентровать элемент по вертикали и горизонтали так, чтобы центровка сохранялась при изменении размеров элемента или контейнера - задать контейнеру раскладку флексбокса, а дочернему флекс-элементу margin: auto. В этом случае флекс-элемент уменьшит свой размер под содержимое и отцентруется по вертикали и горизонтали.

    <style>

        body {

            display: flex;

        }

        p {

            margin: auto;

        }

    </style>

</head>

<body>

    <p> <img  src="assets/images/running.png" alt=""></p>

</body>

1. В чем преимущества box-sizing?

box-sizing позволяет нам точно понять, как вычисляются размеры элемента.

Вообще говоря, лучшее значение box-sizing для применения — это border-box. Это значение намного упрощает нам расчет размера блочного элемента. Если мы хотим, чтобы весь элемент был 400 пикселей в ширину, он останется 400 пикселей в ширину, независимо от того, какие значения padding или border мы к нему добавляем.

Кроме того, мы можем легко смешивать значения длины. Скажем, мы хотим, чтобы наш блок занимал 40% в ширину. Добавление padding 20 пикселей и border 10 пикселей для всех сторон элемента не сложно и мы по-прежнему можем гарантировать, что фактическая ширина нашего блока останется 40%, несмотря на использование пикселей в другом месте.

1. Чем отличается flex-grow от flex-shrink?

### ****CSS-свойство flex-grow****

**flex-grow** определяет, может ли **начальная ширина flex-элемента увеличиваться (расти)**. Увеличение ширины flex-элемента осуществляется за счёт **свободного пространства линии**. В качестве значения CSS-свойства flex-grow указывается **целое число**. Именно это значение и определяет (если оно больше или равно 1) какую часть свободного пространства flex-элемент заберёт себе.

**flex-shrink** свойство определяет, может ли ширина flex-элемента уменьшиться. Уменьшение ширины flex-элемента будет осуществляться только в том случае, если **ширины линии будет не достаточно для отображения всех flex-элементов**, расположенных в ней. Необходимая ширина рассчитывается на **основании начальной ширины**, которую имеет каждый flex-элемент в ней.

1. Как можно добиться следующего позиционирования элементов:

justify-content: space-between;

1. Какой в итоге будет размер у элемента (можно округлить)?

Длина =135+5+5+2+2=149

Ширина =12+2+2+2=18

1. Самостоятельно разберитесь, зачем нужно свойство order?

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) свойство  **order** определяет порядок, используемый для размещения flex элементов в их flex контейнере. Элементы располагаются в восходящем порядке по значению order. Элементы с одинаковым значением order  располагаются в том порядке, в каком они находятся в исходном коде.

**Важно**: order воздействует только на **визуальный порядок** элементов, но не логический порядок или табуляцию. **order** не должен применяться к невизуальным медиаданным, таким как речь.

/\* Числовые значения, в том числе отрицательные \*/

order: 5;

order: -5;

/\* Глобальные значения \*/

order: inherit;

order: initial;

order: unset;

1. Каким кодом можно сделать такую таблицу?
2. <body>
3. <table border="1" style="width: 50%;">
4. <tr>
5. <td>Columu1</td>
6. <td>Columu2</td>
7. <td>Columu3</td>
8. </tr>
9. <tr>
10. <td rowspan="2">Row 1 Cell 1</td>
11. <td>Row 1 Cell 2</td>
12. <td>Row 1 Cell 3</td>
13. </tr>
14. <tr>
15. <td>Row 1 Cell 3</td>
16. <td>Row 2 Cell 3</td>
17. </tr>
18. <tr>
19. <td colspan="3">Row 3 Cell 1</td>
20. </tr>
21. </table>
22. Изучите материалы и найдите, для каких ситуаций подходит position: fixed?

{Position: fixed} или зафиксированное позиционирование. Который чаще всего используется при создании шапки и футера страницы. Может использоваться для создания блоков с рекламой.

Суть его заключается в том, что он закрепляет элемент в определенной части окна, не позволяя ему исчезнуть с экрана. То есть, если вы будете прокручивать страницу, то зафиксированный элемент всегда будет на экране монитора.

Для таких элементов создается отдельный поток элементов. В первом потоке содержатся элементы, которые не зафиксированы на странице. Во втором — контент имеющий свойство {position: fixed}.

1. Каким способом лучше всего верстать большие блоки текста?

Используя свойство overflow-wrap

Свойство overflow-wrap позволяет сообщить браузеру о том, что он должен разорвать слово, перенеся его на новую строку, в том случае, если слово не помещается в контейнер.

1. .card {
2. **overflow-wrap**: break-word;
3. }

Используя Свойство hyphens

Значение auto CSS-свойства hyphens позволяет сообщить браузеру о том, что он должен самостоятельно принять решение о разделении длинных слов с использованием дефиса и о переносе их на новые строки. Это свойство может принимать и значение manual, что позволяет, используя особые символы, предусмотреть возможность и порядок переноса слова на новую строку в том случае, если в этом возникнет необходимость.

1. .element {
2. **hyphens**: auto;
3. }

Вывод длинных текстов в полях, поддерживающих горизонтальную прокрутку

В некоторых ситуациях непрактично организовывать разрыв слов или их перенос на новые строки с использованием дефиса. Например, если имеется поле, в котором должен выводиться фрагмент JavaScript-кода, такой код будет тяжело читать в том случае, если слова будут выводиться с переносом на новые строки. В подобном случае облегчить работу с текстом можно, предусмотрев возможность горизонтальной прокрутки содержимого поля.

1. .code {
2. **overflow-x**: auto;
3. }
4. Как рассчитывается размер flex-контейнера?

### Свойство flex-basis “Базовый размер”

Свойство flex-basis — задает **базовый размер** элемента,**размер элемента вдоль главной оси**. Это **начальный** или**исходный размер** вдоль главной оси. Значение по умолчанию —auto.

Если главная ось направлена горизонтально, то главный размер — это ширина (width), а поперечный размер — это высота (height). Если главная ось направлена вертикально, то все наоборот.

Если flex-basis не задан или его значение равно auto, то базовый размер берется из width или height. Свойство flex-basis принимает те же значения, что и width и heigh, но является «сильнее» и переопределит либо ширину, либо высоту в зависимости от направления главной оси.

Базовый размер — это исходный размер флекс-элементов **до применения flex-grow**.

### Свойство flex-grow “Коефициент растягивания”

Свойство flex-grow — задает **коефициент растягивания** (будет ли растягиваться элемент на свободное пространство), принимает неотрицательные числовые значения (по умолчанию —0). Значение flex-grow: 0 означает, что элемент **не будет растягиватьcz** на свободное место во флекс-контейнере. Значение flex-grow: > 0 (больше нуля), означает, что элемент**будет растягиваться**, «захватывая» оставшееся свободное место. Если сразу у нескольких флекс-элементов значение flex-grow больше нуля, то они будут **делить свободное место между собой**. Свободное место будет добавляться флекс-элементам пропорционально значениям. При этом важно не само значение коэффициента, а его соотношение с коэффициентами остальных элементов, например: 1 и 2 = 1:2, 30 и 40 = 3:4 и т.д.

Рассчитать flex-grow по формуле можно так:

1. СМ (Свободное место) = Ширина контейнера — Сумма базовых размеров элементов **flex-basis**
2. ДСМ (Доля свободного места) = СМ / Сумма **flex-grow** всех элементов
3. Итоговый размер = Базовый размер **flex-basis** + (ДСМ \* **flex-grow**)

При flex-basis: 0 свободное место будет занимать всю ширину флекс-контейнера, и коэффициенты растягивания будут другими.

Использовать flex-basis: 0 и flex-grow для точного управления относительными размерами не стоит. Лучше использовать flex-basis: %.

На размер оставшегося свободного места влияет не только flex-basis, но и рамки, и отступы. Если flex-basis явно задано нулевое значение, то min-width на размер свободного места влиять не будет, так как ограничения размеров к флекс-элементам применяются уже после перераспределения свободного места.

### Свойство flex-shrink “Коефициент сжатия”

Свойство flex-shrink — отвечает за **сжатие** флекс-элементов. Флекс-элементы стараются быть максимально «гибкими» и не выпадать из контейнера, поэтому у flex-shrink имеет значение по умолчанию 1. Свойство принимает неотрицательные числовые значения.

Если flex-shrink: > 0, то элемент **будет уменьшаться**, «впитывая» часть отрицательного пространства, если оно существует. Если flex-shrink: 0, то элемент **уменьшаться не будет**.

Свойство flex-shrink работает схожим образом со свойством flex-grow и задает пропорции, в которых флекс-элементы «впитывают» отрицательное пространство. Чем больше значение свойства у элемента, тем больше отрицательного пространства он «впитает» и тем сильнее сожмется. Доля отрицательного пространства, которую «впитает» элемент, зависит от соотношения коэффициента сжатия элемента с коэффициентами других элементов и соотношения базового размера элемента с базовыми размерами других элементов.

Рассчитать flex-shrink по формуле можно так:

1. ОП (Отрицательное пространство) = Ширина контейнера — Сумма базовых размеров элементов **flex-basis**
2. СПБР (Cумма произведений базовых размеров) = (Базовый размер1 \* flex-shrink1) + (Базовый размер2 \* flex-shrink2) + … + (Базовый размерn \* flex-shrinkn)
3. НКС (Нормированный коэффициент сжатия) = (Базовый размер \* flex-shrink) / СПБР
4. Итоговый размер = Базовый размер — (НКС \* ОП)

Элементы сжимаются в пределах своих базовых размеров, внутренние отступы и рамки не сжимаются. min-width применяется к элементам после этапа перераспределения свободного места или отрицательного пространства.