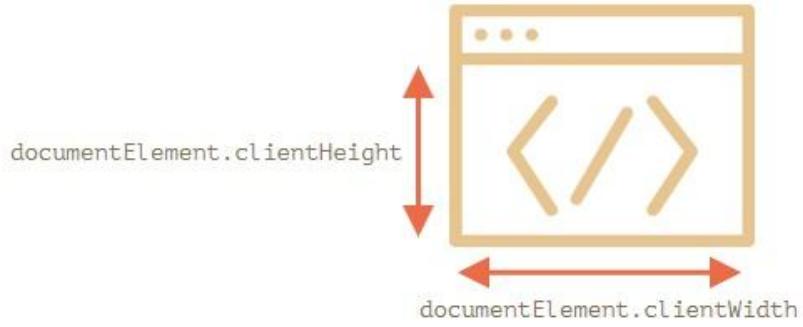
Размеры и прокрутка окна

Как узнать ширину и высоту окна браузера? Как получить полную ширину и высоту документа, включая прокрученную часть? Как прокрутить страницу с помощью JavaScript?

Для большинства таких запросов мы можем использовать корневой элемент документа `document.documentElement`, который соответствует тегу `<html>`. Однако есть дополнительные методы и особенности, которые необходимо учитывать.

Ширина/высота окна

Чтобы получить ширину/высоту окна, можно взять свойства `clientWidth/clientHeight` из `document.documentElement`:



Например, эта кнопка показывает высоту вашего окна:

alert(document.documentElement.clientHeight)

Не `window.innerWidth/Height`
Браузеры также поддерживают свойства `window.innerWidth/innerHeight`. Вроде бы, похоже на то, что нам нужно. Почему же не использовать их?

Если есть полоса прокрутки, и она занимает какое-то место, то свойства `clientWidth/clientHeight` указывают на ширину/высоту документа без неё (за её вычетом). Иными словами, они возвращают высоту/ширину видимой части документа, доступной для содержимого.

А `window.innerWidth/innerHeight` включают в себя полосу прокрутки.

Если полоса прокрутки занимает некоторое место, то эти две строки выведут разные значения:

```
alert( window.innerWidth ); // полная ширина окна
alert( document.documentElement.clientWidth ); // ширина окна за вычетом полосы прокрутки
```

В большинстве случаев нам нужна *доступная* ширина окна: для рисования или позиционирования. Полоса прокрутки «отъедает» её часть. Поэтому следует использовать `documentElement.clientHeight/Width`.

`DOCTYPE` – это важно
Обратите внимание, что геометрические свойства верхнего уровня могут работать немного иначе, если в HTML нет `<!DOCTYPE HTML>`. Возможны странности.

В современном HTML мы всегда должны указывать DOCTYPE.

Ширина/высота документа

Теоретически, т.к. корневым элементом документа является `documentElement`, и он включает в себя всё содержимое, мы можем получить полный размер документа как `documentElement.scrollWidth/scrollHeight`.

Но именно на этом элементе, для страницы в целом, эти свойства работают не так, как предполагается. В Chrome/Safari/Opera, если нет прокрутки, то `documentElement.scrollHeight` может быть даже меньше, чем `documentElement.clientHeight`! С точки зрения элемента это невозможная ситуация.

Чтобы надёжно получить полную высоту документа, нам следует взять максимальное из этих свойств:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);

alert('Полная высота документа с прокручиваемой частью: ' + scrollHeight);
```

Почему? Лучше не спрашивайте. Эти несоответствия идут с древних времён. Глубокой логики здесь нет.

Получение текущей прокрутки

Обычные элементы хранят текущее состояние прокрутки в `elem.scrollLeft/scrollTop`.

Что же со страницей? В большинстве браузеров мы можем обратиться к `documentElement.scrollLeft/Top`, за исключением основанных на старом WebKit (Safari), где есть ошибка (5991), и там нужно использовать `document.body` вместо `document.documentElement`.

К счастью, нам совсем не обязательно запоминать эти особенности, потому что текущую прокрутку можно прочитать из свойств `window.pageXOffset/pageYOffset`:

```
alert('Текущая прокрутка сверху: ' + window.pageYOffset);
alert('Текущая прокрутка слева: ' + window.pageXOffset);
```

Эти свойства доступны только для чтения.

Прокрутка: scrollTo, scrollBy, scrollIntoView

Важно:
Для прокрутки страницы из JavaScript её DOM должен быть полностью построен.

Например, если мы попытаемся прокрутить страницу из скрипта в `<head>`, это не сработает.

Обычные элементы можно прокручивать, изменяя `scrollTop/scrollLeft`.

Мы можем сделать то же самое для страницы в целом, используя `document.documentElement.scrollTop/Left` (кроме основанных на старом WebKit (Safari), где, как сказано выше, `document.body.scrollTop/Left`).

Есть и другие способы, в которых подобных несовместимостей нет: специальные методы `window.scrollBy(x,y)` и `window.scrollTo(pageX,pageY)`.

- Метод `scrollBy(x,y)` прокручивает страницу *относительно её текущего положения*. Например, `scrollBy(0,10)` прокручивает страницу на `10px` вниз.

  Кнопка ниже демонстрирует это:

  window.scrollBy(0,10)

- Метод `scrollTo(pageX,pageY)` прокручивает страницу *на абсолютные координаты* (`pageX,pageY`). То есть, чтобы левый-верхний угол видимой части страницы имел данные координаты относительно левого верхнего угла документа. Это всё равно, что поставить `scrollLeft/scrollTop`. Для прокрутки в самое начало мы можем использовать `scrollTo(0,0)`.

  window.scrollTo(0,0)

  Эти методы одинаково работают для всех браузеров.

scrollIntoView

Для полноты картины давайте рассмотрим ещё один метод: elem.scrollIntoView(top).

Вызов `elem.scrollIntoView(top)` прокручивает страницу, чтобы `elem` оказался вверху. У него есть один аргумент:

- если `top=true` (по умолчанию), то страница будет прокручена, чтобы `elem` появился в верхней части окна. Верхний край элемента совмещён с верхней частью окна.
- если `top=false`, то страница будет прокручена, чтобы `elem` появился внизу. Нижний край элемента будет совмещён с нижним краем окна.

  Кнопка ниже прокрутит страницу так, что она сама окажется вверху:

  this.scrollIntoView()

  А следующая кнопка прокрутит страницу так, что она сама окажется внизу

  this.scrollIntoView(false)

Запретить прокрутку

Иногда нам нужно сделать документ «непрокручиваемым». Например, при показе большого диалогового окна над документом – чтобы посетитель мог прокручивать это окно, но не документ.

Чтобы запретить прокрутку страницы, достаточно установить `document.body.style.overflow = "hidden"`.

Попробуйте сами:

```
document.body.style.overflow = 'hidden'
```

`document.body.style.overflow = ''`

Первая кнопка останавливает прокрутку, вторая возобновляет её.

Аналогичным образом мы можем «заморозить» прокрутку для других элементов, а не только для `document.body`.

Недостатком этого способа является то, что сама полоса прокрутки исчезает. Если она занимала некоторую ширину, то теперь эта ширина освободится, и содержимое страницы расширится, текст «прыгнет», заняв освободившееся место.

Это выглядит немного странно, но это можно обойти, если сравнить `clientWidth` до и после остановки, и если `clientWidth` увеличится (значит полоса прокрутки исчезла), то добавить `padding` в `document.body` вместо полосы прокрутки, чтобы оставить ширину содержимого прежней.

Итого

Размеры:

- Ширина/высота видимой части документа (ширина/высота области содержимого): `document.documentElement.clientWidth/Height`
- Ширина/высота всего документа с прокрученной частью:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);
```

Прокрутка:

- Прокрутку окна можно получить так: `window.pageYOffset/pageXOffset`.
- Изменить текущую прокрутку:
  - `window.scrollTo(pageX,pageY)` – абсолютные координаты,
  - `window.scrollBy(x,y)` – прокрутка относительно текущего места,
  - `elem.scrollIntoView(top)` – прокрутить страницу так, чтобы сделать `elem` видимым (выровнять относительно верхней/нижней части окна).

Координаты

Чтобы передвигать элементы по экрану, нам следует познакомиться с системами координат.

Большинство соответствующих методов JavaScript работают в одной из двух указанных ниже систем координат:

1. **Относительно окна браузера** – как `position:fixed`, отсчёт идёт от верхнего левого угла окна.

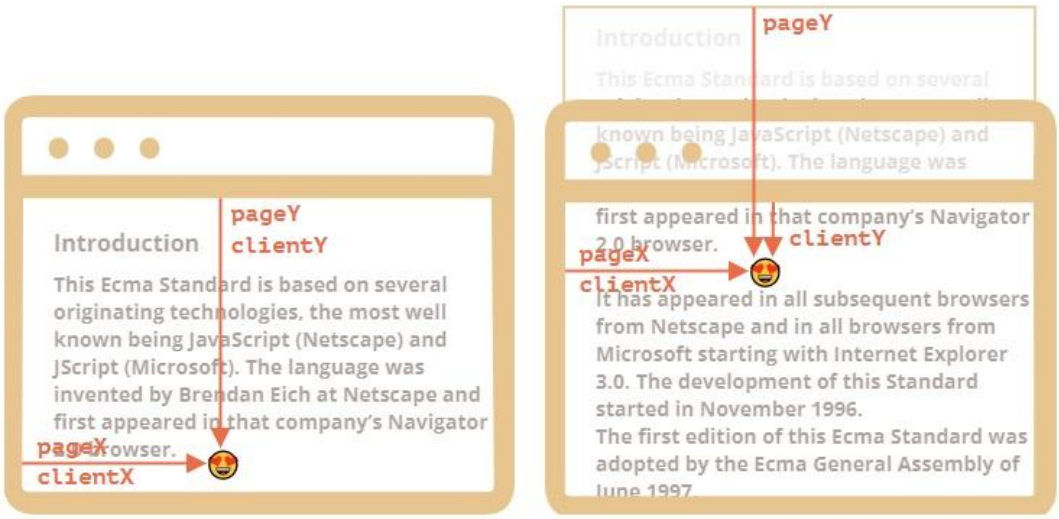
- мы будем обозначать эти координаты как `clientX/clientY`, причина выбора таких имён будет ясна позже, когда мы изучим свойства событий.

2. **Относительно документа** – как `position:absolute` на уровне документа, отсчёт идёт от верхнего левого угла документа.

- мы будем обозначать эти координаты как `pageX/pageY`.

Когда страница полностью прокручена в самое начало, то верхний левый угол окна совпадает с левым верхним углом документа, при этом обе этих системы координат тоже совпадают. Но если происходит прокрутка, то координаты элементов в контексте окна меняются, так как они двигаются, но в то же время их координаты относительно документа остаются такими же.

На приведённой картинке взята точка в документе и показаны её координат до прокрутки (слева) и после (справа):



При прокрутке документа:

- `pageY` – координата точки относительно документа осталась без изменений, так как отсчёт по-прежнему ведётся от верхней границы документа (сейчас она прокручена наверх).
- `clientY` – координата точки относительно окна изменилась (стрелка на рисунке стала короче), так как точка стала ближе к верхней границе окна.

Координаты относительно окна: getBoundingClientRect

Метод `elem.getBoundingClientRect()` возвращает координаты в контексте окна для минимального по размеру прямоугольника, который заключает в себе элемент `elem`, в виде объекта встроенного класса DOMRect.

Основные свойства объекта типа `DOMRect`:

- `x/y` – X/Y-координаты начала прямоугольника относительно окна,
- `width/height` – ширина/высота прямоугольника (могут быть отрицательными).

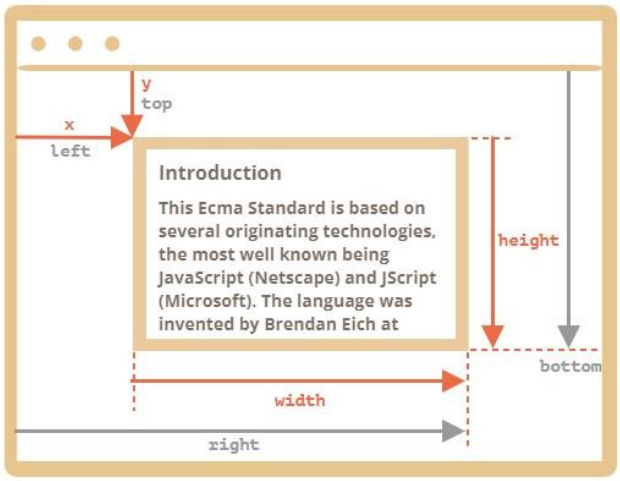
Дополнительные, «зависимые», свойства:

- `top/bottom` – Y-координата верхней/нижней границы прямоугольника,
- `left/right` – X-координата левой/правой границы прямоугольника.

Кликните на кнопку, чтобы увидеть её координаты относительно окна:

Если вы прокрутите страницу, то расположение кнопки в окне поменяется, и, соответственно, её координаты в контексте окна тоже (при вертикальной прокрутке – `y/top/bottom`).

Вот картинка с результатами вызова `elem.getBoundingClientRect()`:



Как вы видите, `x/y` и `width/height` уже точно задают прямоугольник. Остальные свойства могут быть легко вычислены на их основе:

- `left = x`
- `top = y`
- `right = x + width`
- `bottom = y + height`

Заметим:

- Координаты могут считаться с десятичной частью, например `10.5`. Это нормально, ведь браузер использует дроби в своих внутренних вычислениях. Мы не обязаны округлять значения при установке `style.left/top`.
- Координаты могут быть отрицательными. Например, если страница прокручена так, что элемент `elem` ушёл вверх за пределы окна, то вызов `elem.getBoundingClientRect().top` вернёт отрицательное значение.

Зачем вообще нужны зависимые свойства? Для чего существуют `top/left`, если есть `x/y`?

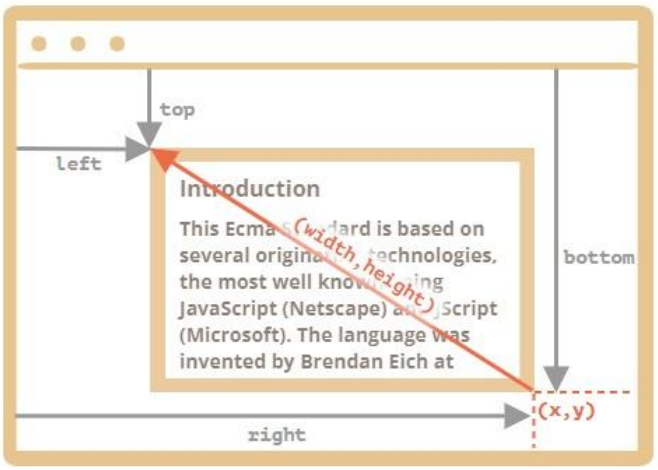
С математической точки зрения, прямоугольник однозначно задаётся начальной точкой `(x,y)` и вектором направления `(width,height)`.

Так что дополнительные зависимые свойства существуют лишь для удобства.

Что же касается `top/left`, то они на самом деле не всегда равны `x/y`. Технически, значения `width/height` могут быть отрицательными. Это позволяет задать «направленный» прямоугольник, например, для выделения мышью с отмеченным началом и концом.

То есть, отрицательные значения `width/height` означают, что прямоугольник «растет» влево-вверх из правого угла.

Вот прямоугольник с отрицательными `width` и `height` (например, `width=-200`, `height=-100`):



Как вы видите, свойства `left/top` при этом не равны `x/y`.

Впрочем, на практике результат вызова `elem.getBoundingClientRect()` всегда возвращает положительные значения для ширины/высоты. Здесь мы упомянули отрицательные `width/height` лишь для того, чтобы вы поняли, зачем существуют эти с виду дублирующие свойства.

Internet Explorer и Edge: не поддерживают `x/y`

Internet Explorer и Edge не поддерживают свойства `x/y` по историческим причинам.

Таким образом, мы можем либо сделать полифил (добавив соответствующие геттеры в `DomRect.prototype`), либо использовать `top/left`, так как это всегда одно и то же при положительных `width/height`, в частности – в результате вызова `elem.getBoundingClientRect()`.

Координаты right/bottom отличаются от одноимённых CSS-свойств

Есть очевидное сходство между координатами относительно окна и CSS `position:fixed`.

Но в CSS свойство `right` означает расстояние от правого края, и свойство `bottom` означает расстояние от нижнего края окна браузера.

Если взглянуть на картинку выше, то видно, что в JavaScript это не так. Все координаты в контексте окна считаются от верхнего левого угла, включая `right/bottom`.

elementFromPoint(x, y)

Вызов `document.elementFromPoint(x, y)` возвращает самый глубоко вложенный элемент в окне, находящийся по координатам `(x, y)`.

Синтаксис:

```
let elem = document.elementFromPoint(x, y);
```

Например, код ниже выделяет с помощью стилей и выводит имя тега элемента, который сейчас в центре окна браузера:

```
let centerX = document.documentElement.clientWidth / 2;
let centerY = document.documentElement.clientHeight / 2;
```

```
let elem = document.elementFromPoint(centerX, centerY);

elem.style.background = "red";
alert(elem.tagName);
```

Поскольку используются координаты в контексте окна, то элемент может быть разным, в зависимости от того, какая сейчас прокрутка.

Для координат за пределами окна метод `elementFromPoint` возвращает `null`

Метод `document.elementFromPoint(x,y)` работает, только если координаты `(x,y)` относятся к видимой части содержимого окна.

Если любая из координат представляет собой отрицательное число или превышает размеры окна, то возвращается `null`.

Вот типичная ошибка, которая может произойти, если в коде нет соответствующей проверки:

```
let elem = document.elementFromPoint(x, y);
// если координаты ведут за пределы окна, то elem = null
elem.style.background = ''; // Ошибка!
```

Применение для fixed позиционирования

Чаще всего нам нужны координаты для позиционирования чего-либо.

Чтобы показать что-то около нужного элемента, мы можем вызвать `getBoundingClientRect`, чтобы получить его координаты элемента, а затем использовать CSS-свойство `position` вместе с `left/top` (или `right/bottom`).

Например, функция `createMessageUnder(elem, html)` ниже показывает сообщение под элементом `elem`:

```
let elem = document.getElementById("coords-show-mark");

function createMessageUnder(elem, html) {
  // создаём элемент, который будет содержать сообщение
  let message = document.createElement('div');
  // для стилей лучше было бы использовать css-класс здесь
  message.style.cssText = "position:fixed; color: red";

  // устанавливаем координаты элементу, не забываем про "px"!
  let coords = elem.getBoundingClientRect();

  message.style.left = coords.left + "px";
  message.style.top = coords.bottom + "px";

  message.innerHTML = html;

  return message;
}
```

```
// Использование:
// добавим сообщение на страницу на 5 секунд
let message = createMessageUnder(elem, 'Hello, world!');
document.body.append(message);
setTimeout(() => message.remove(), 5000);
```

Кликните кнопку, чтобы увидеть пример в действии:

Кнопка с id=«coords-show-mark», сообщение появится под ней

Код можно изменить, чтобы показывать сообщение слева, справа, снизу, применять к нему CSS-анимации и так далее. Это просто, так как в нашем распоряжении имеются все координаты и размеры элемента.

Но обратите внимание на одну важную деталь: при прокрутке страницы сообщение уплывает от кнопки.

Причина весьма очевидна: сообщение позиционируется с помощью `position:fixed`, поэтому оно остаётся всегда на том же самом месте в окне при прокрутке страницы.

Чтобы изменить это, нам нужно использовать другую систему координат, где сообщение позиционировалось бы относительно документа, и свойство `position:absolute`.

Координаты относительно документа

В такой системе координат отсчёт ведётся от левого верхнего угла документа, не окна.

В CSS координаты относительно окна браузера соответствуют свойству `position:fixed`, а координаты относительно документа – свойству `position:absolute` на самом верхнем уровне вложенности.

Мы можем воспользоваться свойствами `position:absolute` и `top/left`, чтобы привязать что-нибудь к конкретному месту в документе. При этом прокрутка страницы не имеет значения. Но сначала нужно получить верные координаты.

Не существует стандартного метода, который возвращал бы координаты элемента относительно документа, но мы можем написать его сами.

Две системы координат связаны следующими формулами:

- `pageY` = `clientY` + высота вертикально прокрученной части документа.
- `pageX` = `clientX` + ширина горизонтально прокрученной части документа.

Функция `getCoords(elem)` берёт координаты в контексте окна с помощью `elem.getBoundingClientRect()` и добавляет к ним значение соответствующей прокрутки:

```
// получаем координаты элемента в контексте документа
function getCoords(elem) {
  let box = elem.getBoundingClientRect();

  return {
    top: box.top + pageYOffset,
    left: box.left + pageXOffset
```

```
  };
}
```

Если бы в примере выше мы использовали её вместе с `position:absolute`, то при прокрутке сообщение оставалось бы рядом с элементом.

Модифицированная функция `createMessageUnder`:

```
function createMessageUnder(elem, html) {
  let message = document.createElement('div');
  message.style.cssText = "position:absolute; color: red";

  let coords = getCoords(elem);

  message.style.left = coords.left + "px";
  message.style.top = coords.bottom + "px";

  message.innerHTML = html;

  return message;
}
```

Итого

Любая точка на странице имеет координаты:

1. Относительно окна браузера – `elem.getBoundingClientRect()`.
2. Относительно документа – `elem.getBoundingClientRect()` плюс текущая прокрутка страницы.

Координаты в контексте окна подходят для использования с `position:fixed`, а координаты относительно документа – для использования с `position:absolute`.

Каждая из систем координат имеет свои преимущества и недостатки. Иногда будет лучше применить одну, а иногда – другую, как это и происходит с позиционированием в CSS, где мы выбираем между `absolute` и `fixed`.

Задачи

Найдите координаты точек относительно окна браузера

важность: 5

В ифрейме ниже располагается документ с зелёным «полем».

Используйте JavaScript, чтобы найти координаты углов, обозначенных стрелками.

В документе уже реализована функциональность, когда при клике на любом месте показываются соответствующие координаты.

Ваш код должен при помощи DOM получить четыре пары координат:

1. верхний левый, внешний угол (это просто).
2. нижний правый, внешний угол (тоже просто).
3. верхний левый, внутренний угол (чуть сложнее).
4. нижний правый, внутренний угол (есть несколько способов, выберите один).

Координаты, вычисленные вами, должны совпадать с теми, которые возвращаются по клику мыши.

P.S. Код должен работать, если у элемента другие размеры или есть рамка, без привязки к конкретным числам.

Покажите заметку рядом с элементом

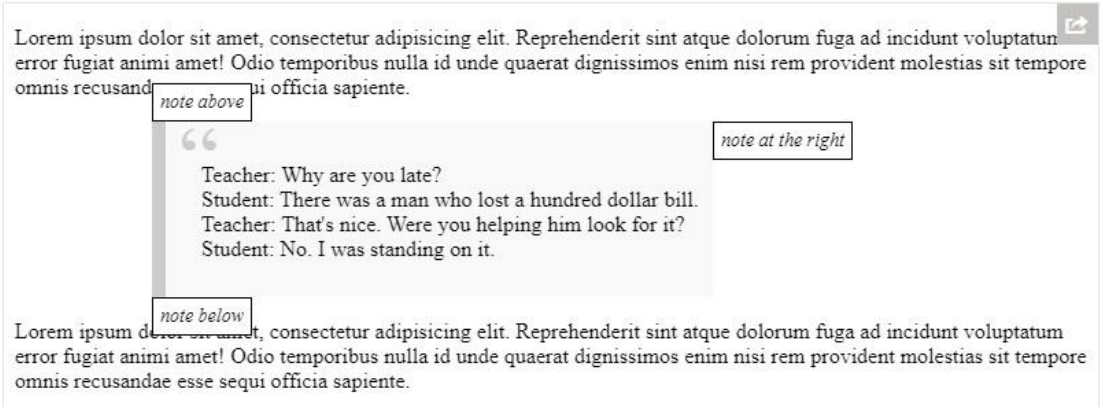
важность: 5

Создайте функцию `positionAt(anchor, position, elem)`, которая позиционирует элемент `elem` в зависимости от значения свойства `position` рядом с элементом `anchor`.

Аргумент `position` – строка с одним из 3 значений:

- `"top"` – расположить `elem` прямо над `anchor`
- `"right"` – расположить `elem` непосредственно справа от `anchor`
- `"bottom"` – расположить `elem` прямо под `anchor`

Она используется внутри функции `showNote(anchor, position, html)`, которая уже есть в исходном коде задачи. Она создаёт и показывает элемент-«заметку» с текстом `html` на заданной позиции `position` рядом с элементом `anchor`.



Покажите заметку около элемента (абсолютное позиционирование)

важность: 5

Измените код решения предыдущего задания так, чтобы элемент заметки использовал свойство `position:absolute` вместо `position:fixed`.

Это предотвратит расхождение элементов при прокрутке страницы.

Используйте решение предыдущего задания для начала. Чтобы проверить решение в условиях с прокруткой, добавьте стиль элементу `<body style="height: 2000px">`.

Расположите заметку внутри элемента (абсолютное позиционирование)

важность: 5

Усовершенствуйте решение предыдущего задания Покажите заметку около элемента (абсолютное позиционирование): научите функцию `positionAt(anchor, position, elem)` вставлять `elem` внутрь `anchor`.

Новые значения для аргумента `position`:

- `top-out`, `right-out`, `bottom-out` – работают так же, как раньше, они вставляют `elem` сверху/справа/снизу `anchor`.
- `top-in`, `right-in`, `bottom-in` – вставляют `elem` внутрь `anchor`: приклеивают его к верхнему/правому/нижнему краю.

Например:

```
// показывает заметку поверх цитаты
positionAt(blockquote, "top-out", note);

// показывает заметку внутри цитаты вблизи верхнего края элемента
positionAt(blockquote, "top-in", note);
```

Для начала возьмите решение задания Покажите заметку около элемента (абсолютное позиционирование).