**Структура веб-страницы**

Язык CSS используется для описания внешнего вида страницы, которая написана с помощью языка разметки HTML. Если мы хотим сказать браузеру, что какую-то кнопку мы хотим сделать зелёной, мы сначала должны задать правило, которое позволяет выделить эту кнопку среди других элементов. Например, это можно сделать с помощью CSS-селектора. Селектор однозначно выделит кнопку среди других элементов html-страницы, а стиль color задаст кнопке цвет green. Синтаксис такого выражения в CSS-файле выглядит так:

button {

color: green;

}

button - это селектор, выражение color: green задает цвет элементу button.

Знание правил написания CSS-селекторов понадобится, чтобы искать элементы на HTML-странице и взаимодействовать с ними.

Также понадобится знание трёх особенностей языка HTML:

1. Страница на языке HTML состоит из элементов, начало и конец элемента задаются с помощью специальных пометок – **тегов** (**tag**);
2. У тегов есть **атрибуты**, которые определяют свойства элементов;
3. Страница на языке HTML имеет **иерархическую** структуру.

**1. Tag – элемент или тег**

Начало и конец элементов обрамляются специальными словами – **тегами (tag)**. Задача тегов – обозначить, какой именно тип информации на странице они представляют (картинка, текст, блок, ссылка и так далее). Например:

<p>Здесь написан текст.</p>

В первом примере элемент обозначен открывающим (<p>) и закрывающим (</p>) тегами, между тегами помещён текст, который увидит пользователь. Некоторым тегам закрывающий тег не нужен, достаточно только открывающего. Например, тег:

<br>

Это тег для переноса строки, внутри него никакого текста или вложенных элементов быть не может.

**2. Attribute – атрибуты или свойства элемента**

Элементам можно задавать различные стандартные свойства, а также создавать собственные свойства или атрибуты. Атрибуты обычно имеют вид: имя="значение". Иногда знак "=" и значение могут быть опущены или значение может быть пустой строкой: имя="".

В примере ниже помимо тегов и текста добавлен еще атрибут**href**, который задаёт ссылку для тега <a>, на которую будет вести текст внутри тега.

<a href="/about">Посмотреть подробнее</a>

**Атрибут class**

В третьем примере появляется еще один важный атрибут – **класс (class)**. Элементу задано два класса: **simple-text** и **main**. Классы чаще всего используют для задания правил вёрстки с помощью CSS:

<div class="simple-text main">Здесь написан текст.</div>

**Атрибут name**

Свойство **name**, например, используется для задания якоря (закладки) в html-странице. Этот атрибут тоже можно использовать в качестве селектора для поиска элемента:

<a name="top"></a>

<p>Длинный текст</p>

...

<p><a href="#top">Наверх</a></p>

**Атрибут id**

В этом примере появился еще атрибут **id**– уникальный указатель на элемент. Значение id в общем случае не должно повторяться в пределах страницы:

<button id="save\_button">Сохранить</button>

!Важно. В современных JavaScript-фреймворках id-атрибуты чаще всего генерируются динамически самим фреймворком, поэтому они изменяются каждый раз при перезагрузке страницы и совершенно нечитабельны, например: вы увидите что-то вроде id="u\_ps\_0\_0\_n" или id="avadspffd". В таких случаях  придется пользоваться другими селекторами или использовать собственные data-атрибуты, о которых расскажем далее. Названия классов также могут генерироваться автоматически. Поэтому предлагаем вам простое правило: если увидите нечеловекочитаемое значение атрибута или если значение атрибута меняется при перезагрузке страницы, то не используйте его.

**3. Иерархия в HTML-документах**

HTML-документ часто сравнивают с моделью семейного древа, в котором есть родители, дети, братья, предки и потомки. Такое сравнение допустимо благодаря возможности вкладывать одни элементы в другие. Вложенные (дочерние) элементы могут, в свою очередь, содержать свои дочерние элементы, следовательно, глубина вложения элементов может быть любой.

Давайте разберёмся с терминологией данного древа:

* потомок элемента X – элемент любой степени вложенности внутри элемента X;
* ребёнок или дочерний элемент — прямой потомок (т.е. элемент на первом уровне вложенности);
* предок элемента Y – любой элемент X, который включает в себя элемент Y;
* родитель — это прямой предок (т.е. элемент, который расположен выше строго на 1 уровень);
* братский или соседний элемент – элемент X, который расположен на одном уровне иерархии с элементом Y. Элементы X и Y имеют одного родителя.

Теперь попробуем закрепить терминологию на примере. Возьмём такой пример HTML кода:

<div class="news">

<div class="title">

<p class="title\_text">Selenium 4alpha is out now</p>

</div>

<div class="text\_block">

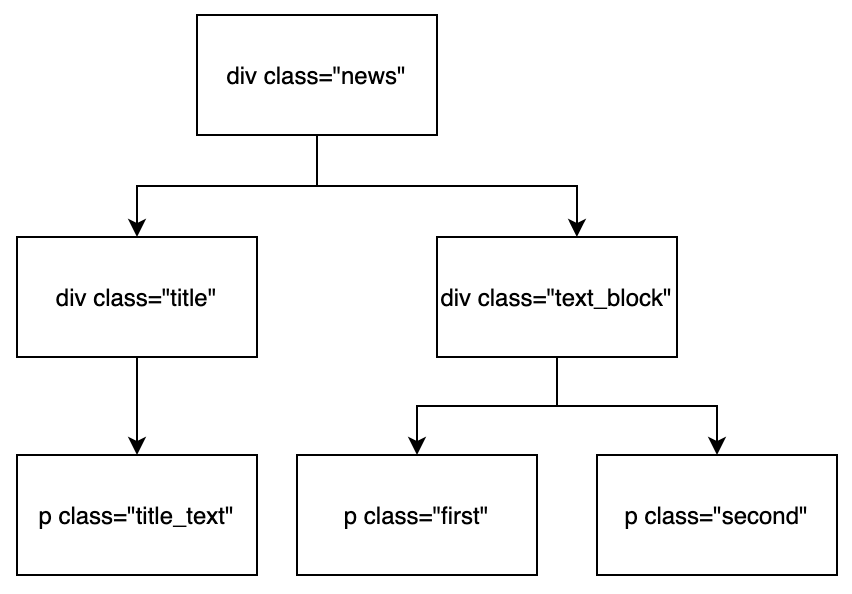
<p class="first">Selenium has put out its first alpha version of Selenium </p>

<p class="second"> Learning Selenium Using Python </p>

</div>

</div>

Теперь нарисуем блок-схему этого HTML кода, где каждый блок будет соответствовать одному HTML элементу.



* **Какие элементы являются потомками элемента, имеющего class="news"?** *Это все остальные элементы в этом html-коде: элемент с class="title", элемент с class="title\_text" и другие.*
* **Какие элементы можно назвать дочерними (child) элементами для элемента с class="news"?** *Только элементы с class= "title" и class="text\_block".*
* **Какие элементы являются предками элемента с class="first"?** *Это элементы с class="text\_block" и class="news".*
* **Какой элемент является родителем элемента с class="first"?** *Только элемент с class="text\_block".*
* **Какой элемент мы можем назвать братским или соседним элементом для элемента, имеющего class="first"?** *Это элемент с class="second".*

Эти знания помогут применять мощный механизм поиска элементов с помощью CSS-селекторов.

1. **Атрибуты, которые влияют на отображение элемента**

Некоторые атрибуты влияют на отрисовку элемента на странице, а другие не влияют напрямую, но могут использоваться в JavaScript-коде или быть нужными только для локации элемента в тестах.

Примеры атрибутов, которые повлияют на отображение и поведение элемента на странице:

<h1 style="color: blue;"> Заголовок будет синим, т.к. цвет задан в атрибуте style </h1>

<p hidden> Атрибут hidden скрывает элемент на странице, элемент не будет показываться </p>

<button disabled> Кнопка с атрибутом disabled будет заблокирована </button>

А еще список атрибутов можно расширять: это значит, что разработчик может создать свой собственный атрибут и присвоить ему любые значения. Что это значит для тестировщика? Это значит, что можно договориться с разработчиками о специальном атрибуте, который можно будет использовать в своих тестах для поиска нужных элементов и который не будет изменяться при исправлении верстки сайта. Это добавит стабильности вашим тестам. Правда, есть несколько ограничений:

* веб-сайт должен использовать стандарт HTML5 (большинство современных сайтов соответствует этому требованию)
* использовать можно только латинские буквы, и символы дефис (-), двоеточие (:) и подчёркивание (\_)

Также принято названия таких атрибутов начинать со слова: "**data"**, например, "data-button".

Что еще важно знать про атрибуты элементов?

Некоторые атрибуты являются универсальными, они могут относиться к любому тегу и любому типу элементов. Например, **hidden** (т.е. скрытым) можно сделать любой элемент. Некоторые же атрибуты ассоциированы строго с определенным тегом, например, для картинки, которая задается тегом **img**, обязательно нужно указывать атрибут **src**.

**Образовательная рекомендация:** если вы не совсем понимаете, что означает тег или атрибут элемента, попробуйте погуглить. Каждый кусочек этих знаний так или иначе поможет вам в будущем.

## ПОИСК элементов на странице

Любая страница в интернете представляет собой html-файл, в котором с помощью языка разметки HTML описана её структура.

Практически все сайты используют язык JavaScript, который позволяет сделать веб-страницу интерактивной, т.е. реагировать на действия пользователя, запрашивать у пользователя данные и возвращать их. Также все сайты используют CSS (Cascading Style Sheets — каскадные таблицы стилей), который используется для вёрстки (красивого оформления страниц).

Умея описывать путь к элементу на странице, можно найти такой элемент и выполнить с ним необходимые действия, например, отправить текст в текстовое поле или нажать на правильную кнопку.

Рассмотрим несколько способов поиска элементов внутри страницы:

* поиск с помощью CSS-селекторов, когда путь к элементу описывается через синтаксис CSS. Селектор — это описание пути к элементу на странице.
* поиск с помощью указания значений тегов или атрибутов элементов: ID, class, и т.д.
* поиск с помощью языка запросов XPath.

Поиск с помощью CSS-селекторов является наиболее удобным способом, т.к. он покрывает практически все возможные ситуации, и CSS-селекторы выглядят более читабельными. В реальности в разных случаях может понадобиться использовать и другие методы поиска.

## Поиск элементов с помощью CSS-селекторов

Ниже приведены части элементов HTML-страницы, по которым можно найти элемент:

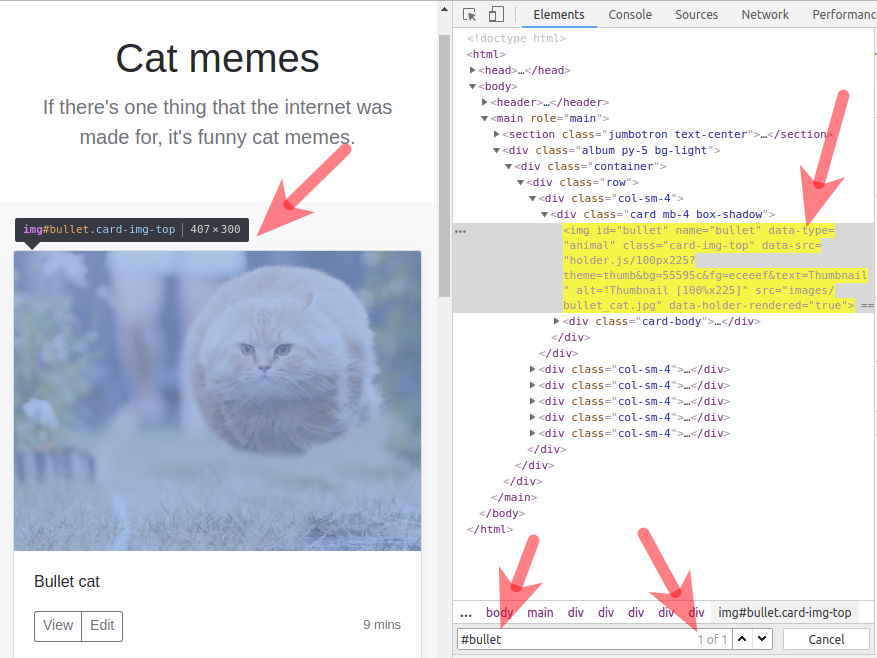
* id
* tag
* значение атрибута
* name
* class

Все примеры по поиску элементов на странице рассматриваются на примере сайта <http://suninjuly.github.io/cats.html>

**Поиск по ID**

У элемента с картинкой есть атрибут id="bullet", а значит, однозначно можно найти её с помощью селектора **#bullet** (знак **#** означает, что ищем по **id** со значением **bullet**).

*Можно проверить правильность подобранного селектора непосредственно в браузере в консоли разработчика. Откройте консоль разработчика и перейдите в ней на вкладку Elements. Затем нажмите ctrl+F и в открывшейся внизу поисковой строке введите селектор. Если селектор написан правильно, то вы увидите подсвеченный элемент на веб-странице, а также элемент будет подсвечен жёлтым цветом в html-коде. Еще в поисковой строке вы увидите количество найденных элементов. Желательно писать точные селекторы, которые позволяют найти ровно один элемент.*



**Поиск по tag**

Чтобы найти элемент по тегу, просто напишите название тега в поисковой строке, как делали это при поиске по id (только без знака **#**), например, **h1**. Поиск по **h1** найдёт элемент с названием страницы. Поиск по тегам не очень удобен, т.к. разработчики используют небольшое количество тегов для разметки страниц, и скорее всего, одному тегу будет соответствовать множество элементов.

**Поиск по значению атрибута**

Можно найти элемент, указав название атрибута и его значение. Например, можно переписать поиск по **id** в следующем виде **[id="bullet"]** вместо **#bullet**.

Лучше использовать вариант с квадратными скобками при поиске значения атрибута для тех атрибутов, у которых нет собственных коротких команд поиска. Например, найдите элемент h1 по значению его атрибута value: **[value="Cat memes"]**

**Поиск по name**

Этот вариант поиска является разновидностью поиска по значению атрибута и записывается так же: **[name="bullet-cat"]**. Этот вариант довольно часто используется, так как выделяется как отдельный вид поиска элементов в Selenium WebDriver.

**Поиск по class**

Поиск по классу можно записать в виде **[class="jumbotron-heading"]**, так как **class** тоже является атрибутом элемента. Но раз уж классы используются практически в каждой странице при задании стилей страниц, то для них также имеется свой короткий вариант поиска: **.jumbotron-heading**. То есть пишут значение класса и предваряют его точкой.

Допустим, у элемента **article** задано больше одного класса, как на странице <http://suninjuly.github.io/cats.html>:

<article id="moto" class="lead text-muted" title="one-thing" name="moto">If there's one thing that the internet was made for, it's funny cat memes.</article>

Вариант **[class="lead"]** не найдет этот элемент, так как он ищет по точному совпадению. Чтобы найти элемент, нужно будет написать **[class="lead text-muted"]**, порядок классов при этом важен. **[class="text-muted lead"]** — уже не найдет искомый элемент.

Вариант **.lead** при этом позволит найти данный элемент, так как он ищет простое вхождение класса в элемент. Для уточнения селектора можно задать также оба класса, для этого нужно добавить второй класс к строке поиска без пробела и предварить его точкой: **.lead.text-muted**. Порядок классов в отличие от первого способа здесь не важен — **.text-muted.lead** так же найдет нужный элемент. Рекомендуем пользоваться вторым способом поиска классов, так как он является более гибким.

Еще одно важное замечание. Поиск по классу чувствителен к регистру, то есть **.Lead** уже не найдет нужный элемент.

В консоли браузера вы также можете искать по простому совпадению текста в HTML, например, запрос **lead** подсветит текст **lead**. Однако, не стоит пользоваться таким поиском для выбора элементов, так как он слишком общий и не может использоваться в качестве селектора.

**Поиск элементов с помощью составных CSS-селекторов**

Предположим, что не можем найти элемент на странице, используя простой селектор, так как такой селектор находит сразу несколько элементов. Ниже мы привели часть кода простой HTML-страницы, описывающей блог.

Вопрос: как нам найти селектор для подписи у второй картинки? Вот здесь поможет иерархическая структура страницы и возможность комбинировать CSS-селекторы. CSS-селекторы позволяют использовать одновременно любые селекторы, рассмотренные ранее, а также имеют некоторые дополнительные возможности для уточнения поиска.

<div id="posts" class="post-list">

<div id="post1" class="item">

<div class="title">Как я провел лето</div>

<img src="./images/summer.png">

</div>

<div id="post2" class="item">

<div class="title second">Ходили купаться</div>

<img src="./images/bad\_dog.jpg">

</div>

<div id="post3" class="item">

<div class="title">С друзьями</div>

<img src="./images/friends.jpg">

</div>

</div>

**Использование потомков**

Попробуем найти элемент с текстом "Ходили купаться". Для решения этой задачи можно взять элемент, стоящий выше в иерархии нужного элемента, и написать следующий селектор:

#post2 .title

Здесь символ **#** означает, что надо искать элемент с id post2, пробел - что также нужно найти элемент-потомок, а **.**, что элемент-потомок должен иметь класс со значением title.

Элемент .title называется **потомком** элемента #post2. Потомок может находиться на любом уровне вложенности, все элементы с селектором .title также являются и потомками элемента #posts, хотя и расположены от него на два уровня ниже. #posts .title найдет все 3 элемента с классом title.

!Внимание. Символ пробела " " является значащим в CSS-селекторах. Это важный символ, который разделяет описание предка и потомка. Если бы мы записали селектор #post2.title без пробела, то в данном примере не было найдено ни одного элемента. Такая запись означала бы, что мы хотим найти элемент, который одновременно содержит id "post2" и класс "title". Таким образом #post2 .title и #post2.title — это разные селекторы**.**

**Использование дочерних элементов**

Другой способ найти этот элемент:

#post2 > div.title

Здесь указали еще тег элемента divи уточнили, что нужно взять элемент с тегом и классом: div.title, который находится строго на один уровень иерархии ниже чем элемент #post2. Для этого используется символ >.

Элемент #post2 в этом случае называется **родителем** для элементаdiv.title, а элемент div.title называется **дочерним элементом** для элемента #post2. Если символа > нет, то будет выполнен поиск всех элементов div.title на любом уровне ниже первого элемента.

!Внимание. В данном случае символы пробела вокруг символа ">" не несут важного значения в отличие от предыдущего примера, и могут быть опущены. Запись #post2>div.titleаналогична записи #post2 > div.title.

**Использование порядкового номера дочернего элемента**

Еще один способ найти этот элемент:

#posts > .item:nth-child(2) > .title

Псевдо-класс :nth-child(2) — позволяет найти второй по порядку элемент среди дочерних элементов для #posts. Затем с помощью конструкции > .title мы указываем, что нам нужен элемент .title, родителем которого является найденный ранее элемент .item.

**Использование нескольких классов**

Также можно использовать сразу несколько классов элемента, чтобы его найти. Для этого классы записываются подряд через точку: .title.second

## Поиск элементов с помощью XPath

**XPath** (XML Path Language) это язык запросов, который использует древовидную структуру документа. Проверять XPath-запросы можно точно так же как и CSS-селекторы — в консоли разработчика.

**1. XPath запрос всегда начинается с символа / или //**

Символ / аналогичен символу > в CSS-селекторе, а символ // — пробелу. Их смысл:

* el1/el2 — выбирает элементы el2, являющиеся прямыми потомками el1;
* el1//el2 — выбирает элементы el2, являющиеся потомками el1 любой степени вложенности.

Разница состоит в том, что в XPath, когда мы начинаем запрос с символа /, мы должны указать элемент, являющийся корнем нашего документа. Корнем всегда будет элемент с тегом <html>. Пример: /html/body/header

Можно начинать запрос и с символа //. Это будет означать, что хотим найти всех потомков корневого элемента без указания корневого элемента. В этом случае, для поиска того же хедера, можем выполнить запрос //header, так как других заголовков нет.

Важно! Такой поиск может быть неоднозначным. Например, запрос //div вернет все элементы с тегом <div>. Избегайте неоднозначных ситуаций, они плохо влияют на здоровье автотестов.

**2. Символ [ ] — это команда фильтрации**

Если по запросу найдено несколько элементов, то будет произведена фильтрация по правилу, указанному в скобках.

Правил фильтрации очень много:

* по любому **атрибуту**, будь то id, class, title (или любой другой). Например, хотим найти картинку с летящим котом, для этого можно выполнить запрос

//img[@id='bullet']

* по **порядковому номеру**. Допустим, хотим выбрать вторую по порядку карточку с котом. Для этого найдем элемент с классом "row" и возьмем его второго потомка:

//div[@class="row"]/div[2]

* по **полному совпадению текста.** Да, XPath — это единственный способ найти элемент по внутреннему тексту. Если хотим найти блок текста с котом-Лениным, можно воспользоваться XPath селектором //p[text()="Lenin cat"]. Такой селектор вернет элемент, только если текст полностью совпадет. Здесь важно сказать, что не всегда поиск по тексту — это хорошая практика, особенно в случае мультиязычных сайтов.
* по **частичному совпадению** текста или атрибута. Для этого нужна функция contains. Запрос //p[contains(text(), "cat")] вернет все абзацы текста, которые содержат слово cat. Точно так же можно искать по частичному совпадению других атрибутов, это удобно, если у элемента несколько классов. Посмотрите на код навбара сайта с котами. Его можно найти селектором //div[contains(@class, "navbar")]
* в фильтрации еще можно использовать булевы операции (and, or, not) и некоторые простые арифметические выражения (но вообще не стоит, наверное). Допустим, мы хотим найти картинку обязательно с data-type "animal" и именем "bullet-cat", для этого подойдет запрос: //img[@name='bullet-cat' and @data-type='animal']

**3. Символ \* — команда выбора всех элементов**

* Например можем найти текст в заголовке запросом //div/\*[@class="jumbotron-heading"]. Это может быть удобно, когда мы не знаем точно тег элемента, который ищем.

**4. Поиск по классу в XPath регистрозависим**

Также как и в случае поиска по CSS-селектором будьте внимательными к регистру при поиске по классам:

//div/\*[@class="Jumbotron-heading"] не найдет элемент на странице.

Что важно знать про XPath, чтобы пользоваться им безболезненно:

* Не используйте селекторы вида //div[1]/div[2]/div[3] без крайней нужды: по такому селектору невозможно с первого раза понять, что за элемент вы ищете. А когда структура страницы хоть немного изменится, то ваш селектор с большой вероятностью перестанет работать;
* Если есть возможность использовать CSS-селекторы: сlass, id или name — лучше использовать их вместо поиска по XPath;
* Можно искать по полному или частичному совпадению текста или любого атрибута;
* Можно использовать булевы операции и простую арифметику;
* Можно удобно перемещаться по структуре документа (переходить к потомкам и к родителям);
* Подойдет, когда у сайта всё плохо с атрибутами и нет возможности достучаться до разработчиков;
* Есть мнение, что поиск по XPath в среднем медленнее, чем по css. Но достоверно это неизвестно;
* Не стоит использовать разные расширения для браузеров по поиску XPath: они подбирают нечитабельные и переусложненные селекторы. Лучше потратить немного времени и разобраться в синтаксисе самостоятельно, тем более, что он не очень сложный.