

07 - Eventos I

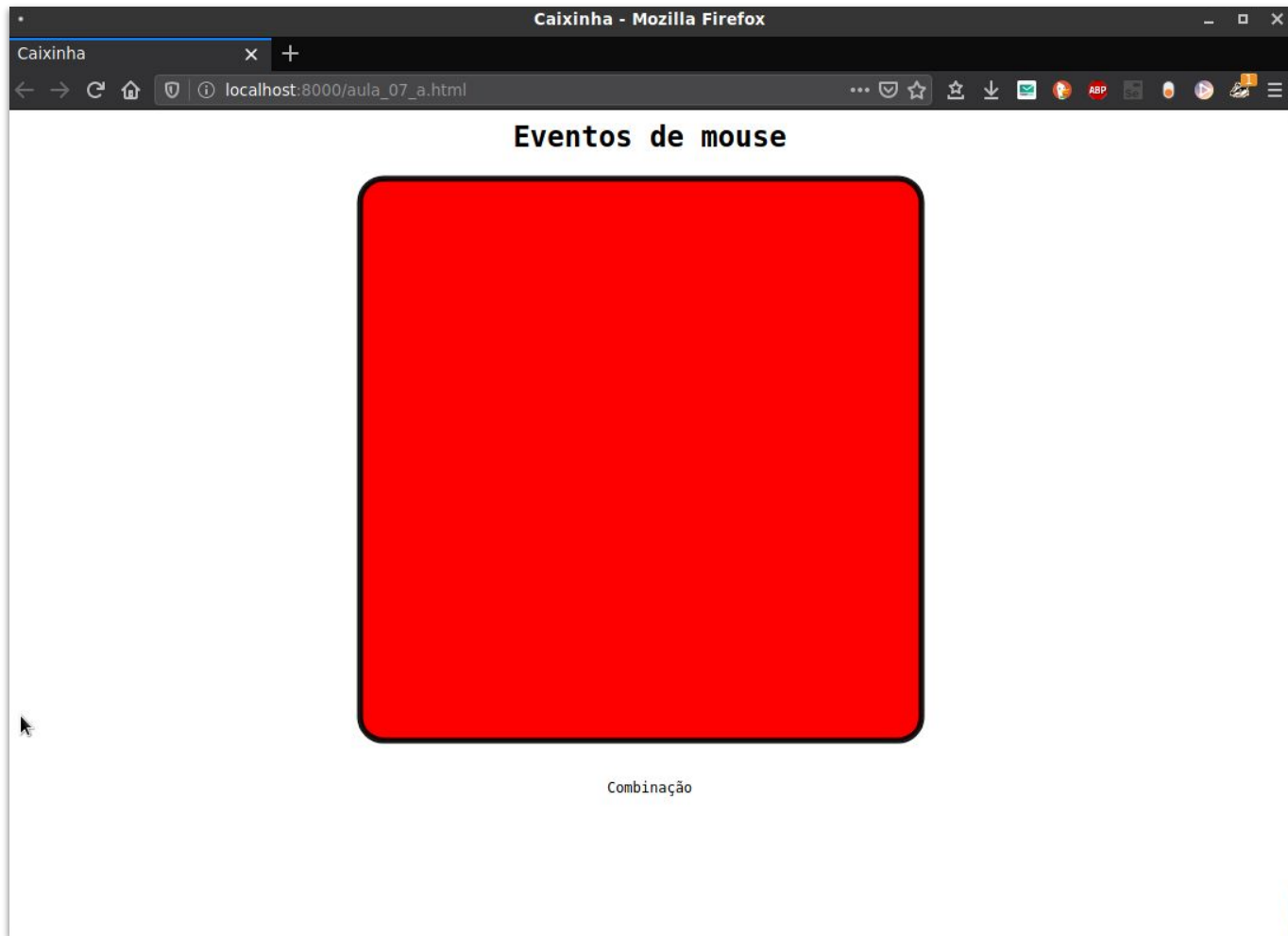
Roteiro

- DOM
- CSSOM
- Eventos
- Ouvinte de eventos



Conhecendo nosso
caso nas aulas de
eventos





```
<body onload="brython(1)">

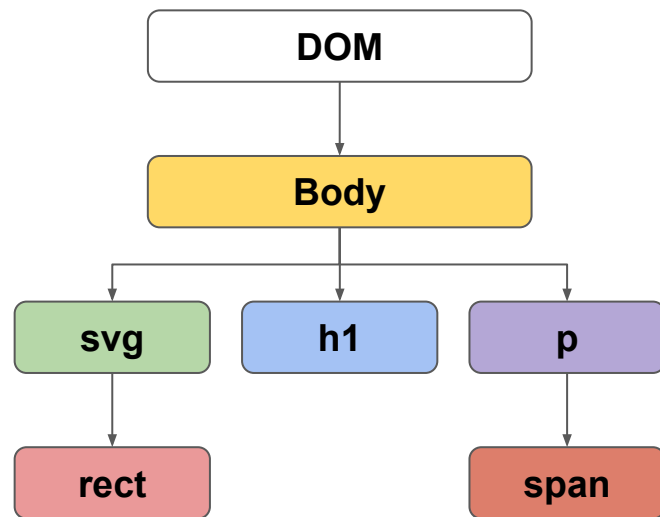
  <h1>Eventos de mouse</h1>

  <svg>
    <rect id="caixa" x="5" y="5" rx="20" ry="20"/>
  </svg>

  <br>

  <p>Combinação <span></span></p>

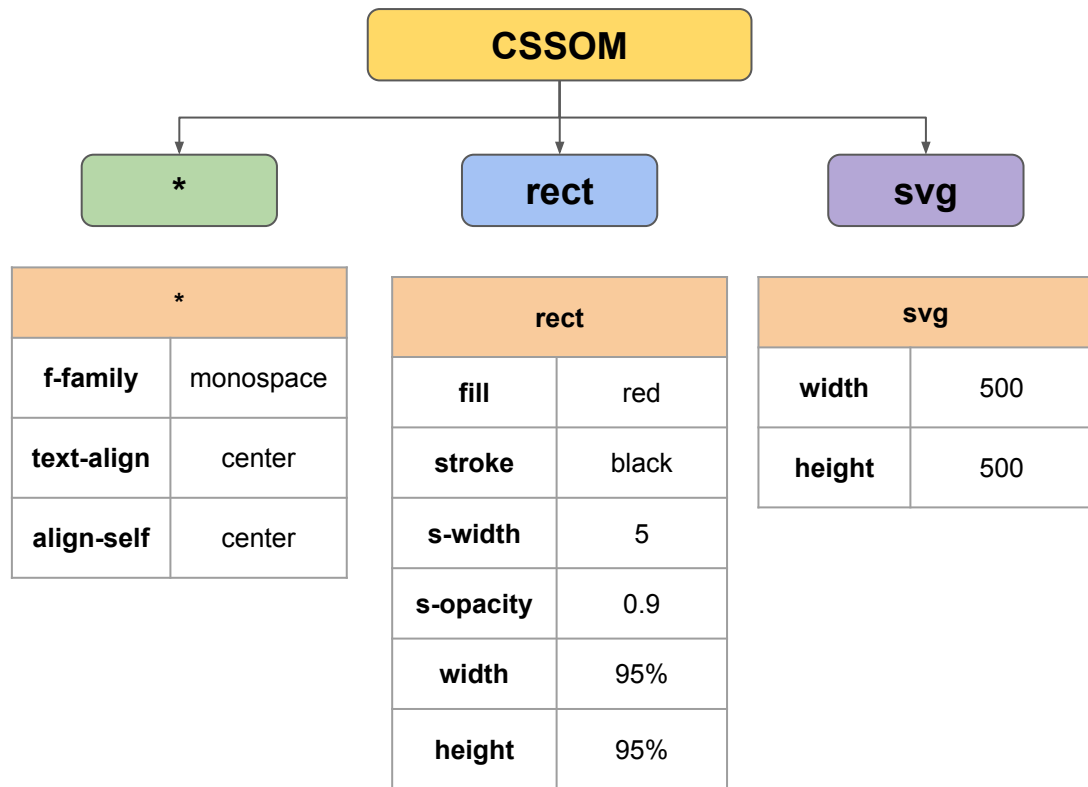
</body>
```



```
* {  
  font-family: monospace;  
  text-align: center;  
  align-self: center;  
}
```

```
rect {  
  fill:red;  
  stroke:black;  
  stroke-width:5;  
  stroke-opacity:0.9;  
  width: 95%;  
  height: 95%;  
}
```

```
svg {  
  width: 500;  
  height: 500;  
}
```



Vamos acessar

https://selenium.dunossauro.live/aula_07_a



Eventos

<https://developer.mozilla.org/en-US/docs/Web/API/event>



Eventos

Eventos são ***super poderes*** que um Web Element adquiere para ter para interagir com o usuário. É como pensar em elementos com comportamento dinâmico.

Event é uma API do browser e tem várias interfaces que herdam suas características para finalidades específicas.



Eventos

- DOM
 - Inserir elementos
 - Remover elementos
- CSSOM
 - Alterar estilo
 - Adicionar estilo
- Salvar cookies
- Checar conexão com a internet
- Salvar mensagens do DB do browser
-

API de eventos

Eventos tem duas características importantes:

- Tipo (type) [nome]
- Alvo (target)

bubbles, cancelable, currentTarget, defaultPrevented, timestamp, trusted, ...



Mostrar lista de eventos

<https://developer.mozilla.org/en-US/docs/Web/Events>



Linguagens de script

- **JavaScript**
- **TypeScript**
- Elm

Alternativas em bibliotecas Python

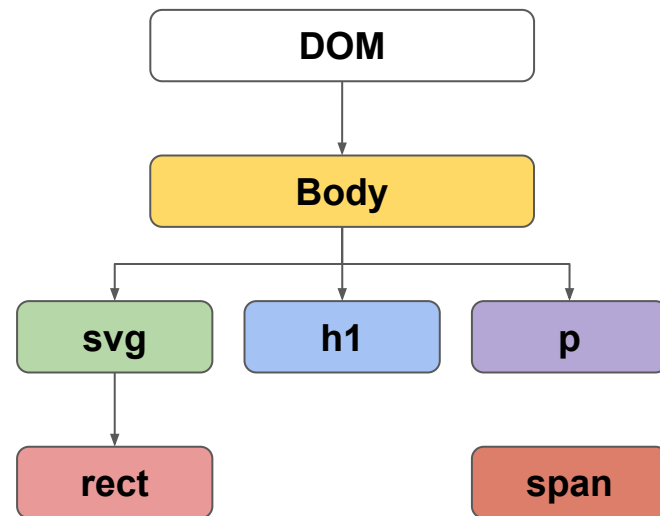
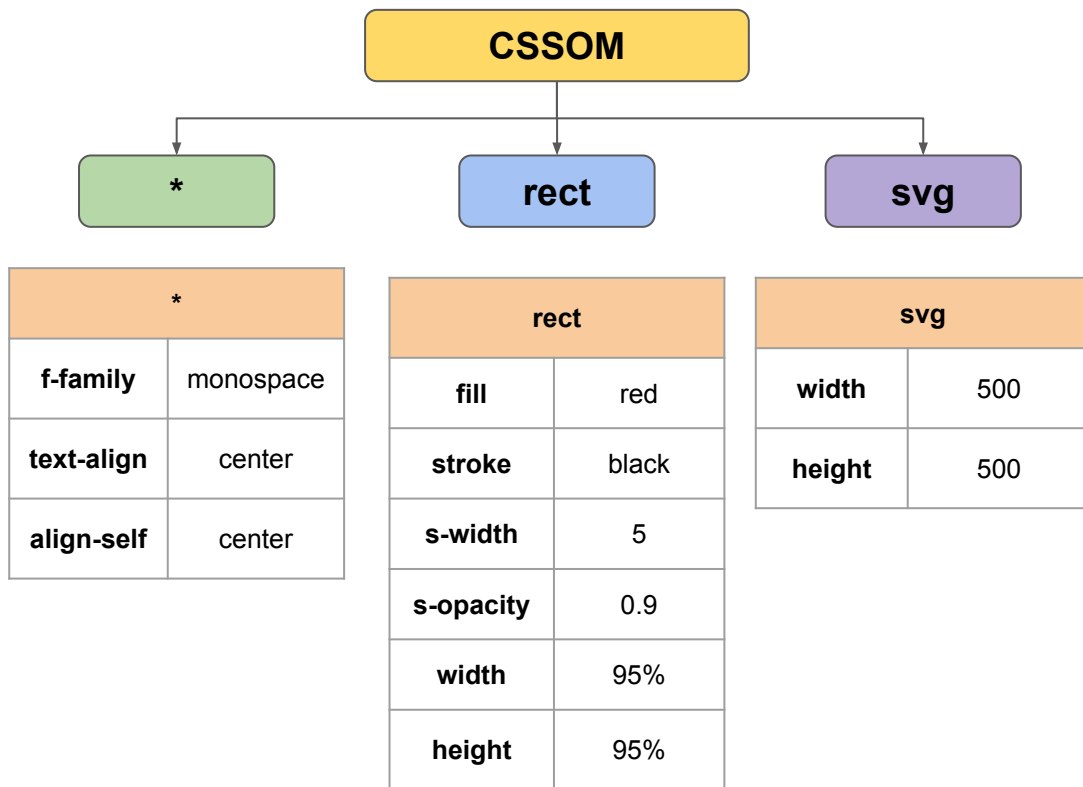
- Brython *
- Pypy.js
- Batavia
- Transcript



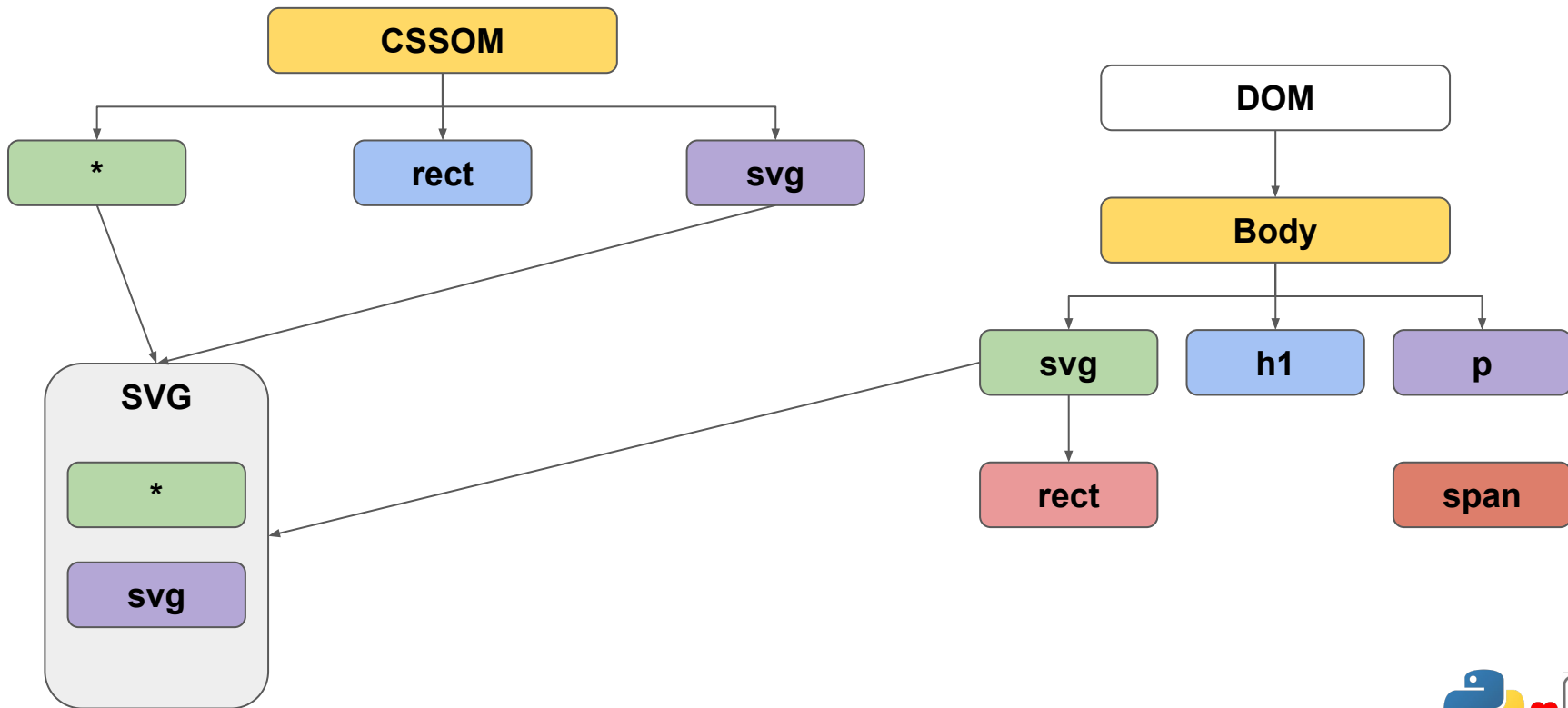
Pintando telas



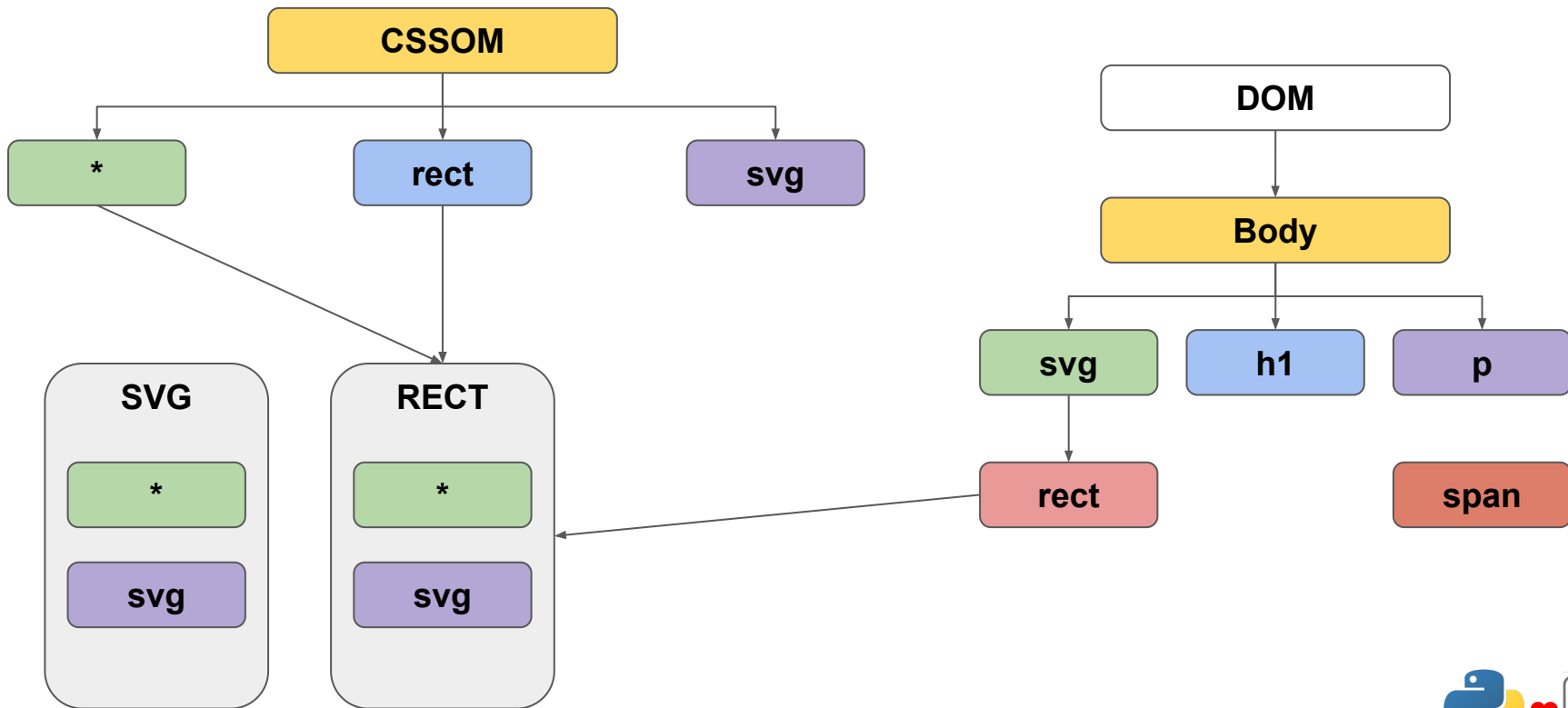
Fase de pintura



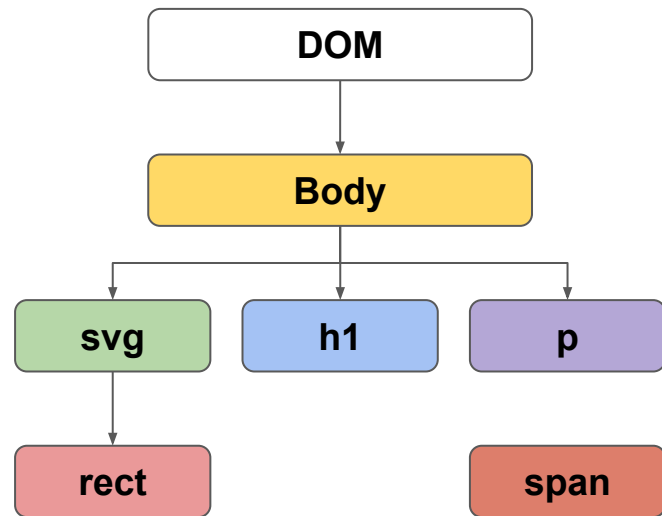
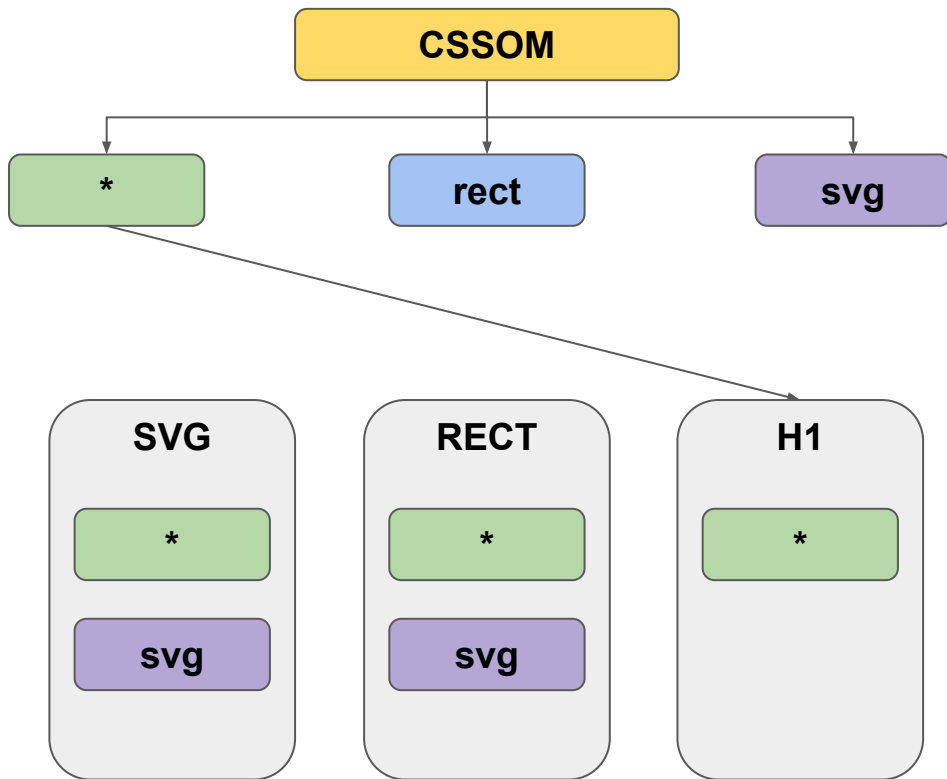
Convergência



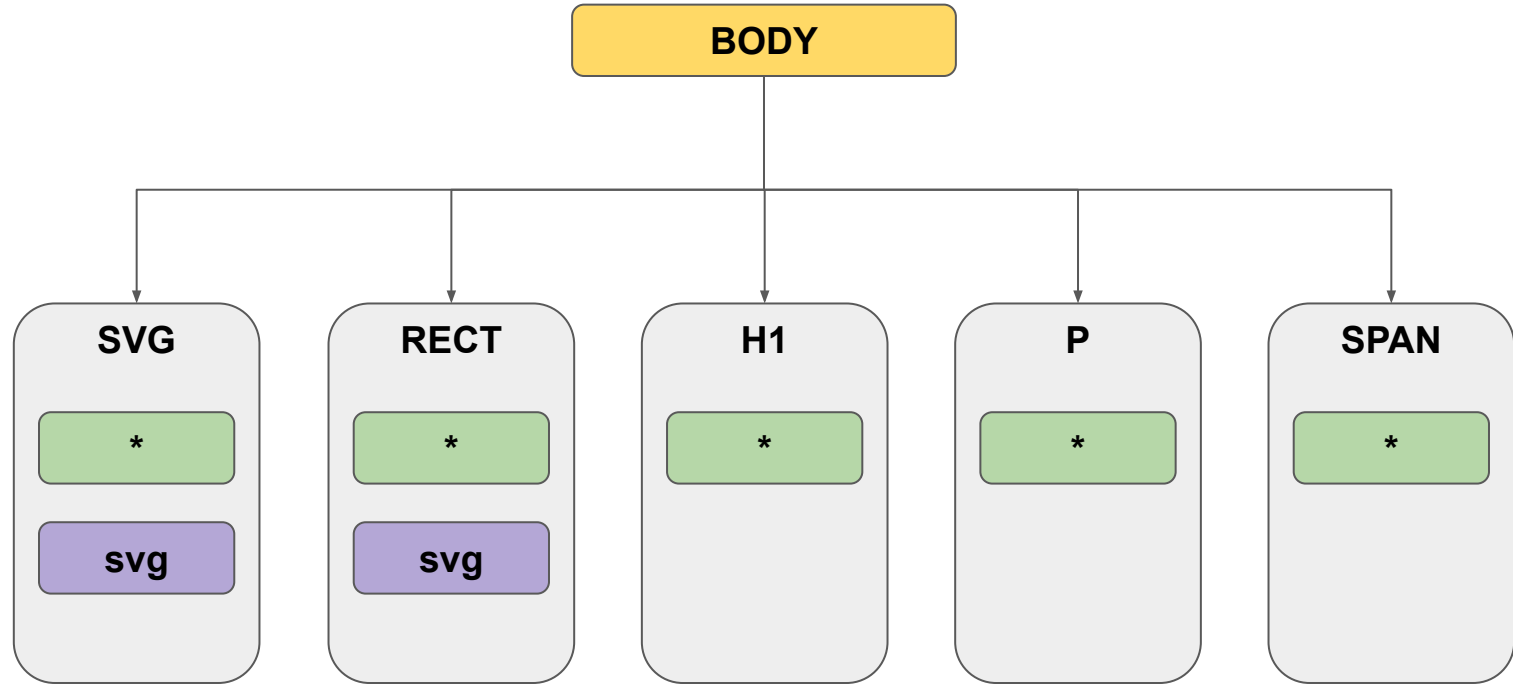
Convergência

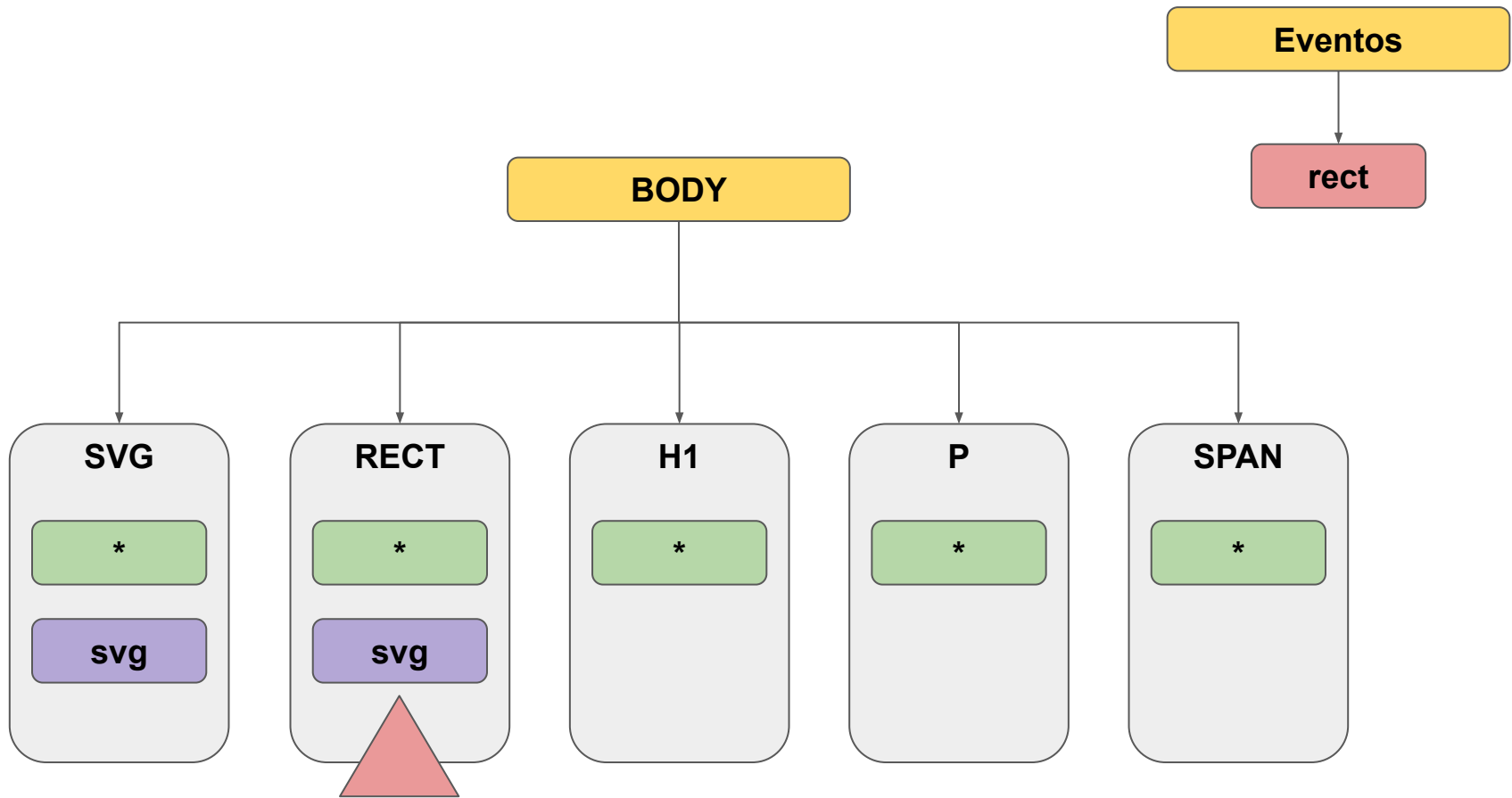


Convergência

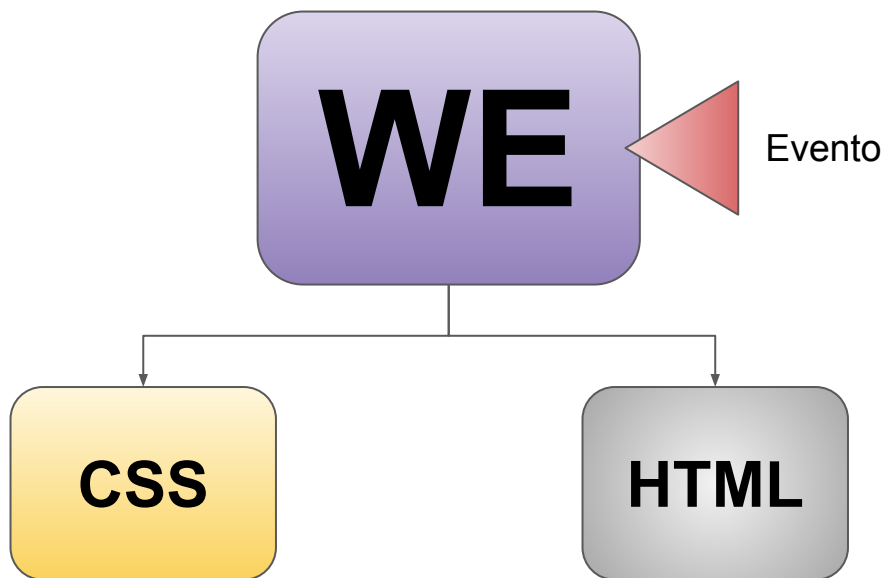


Painted

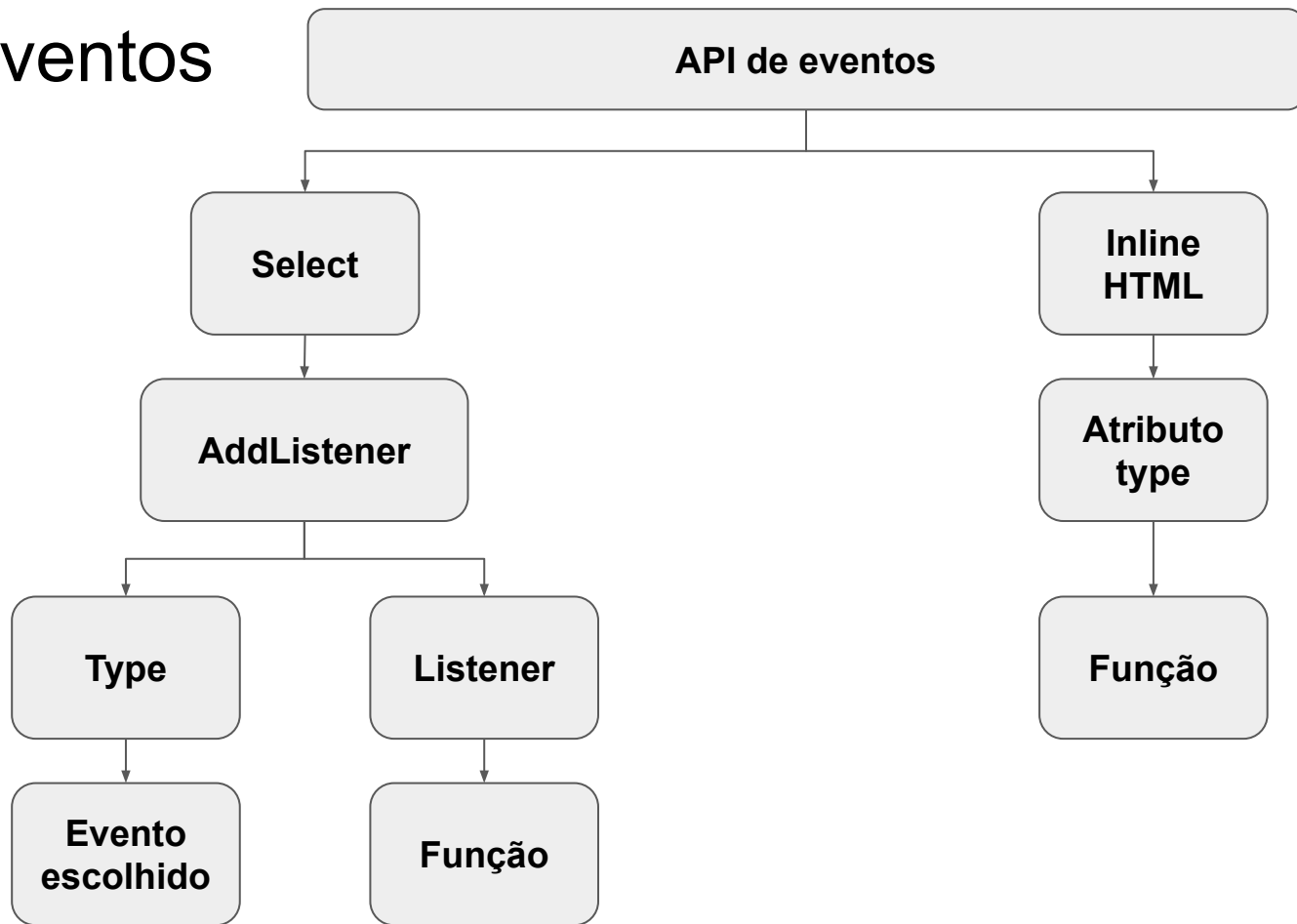




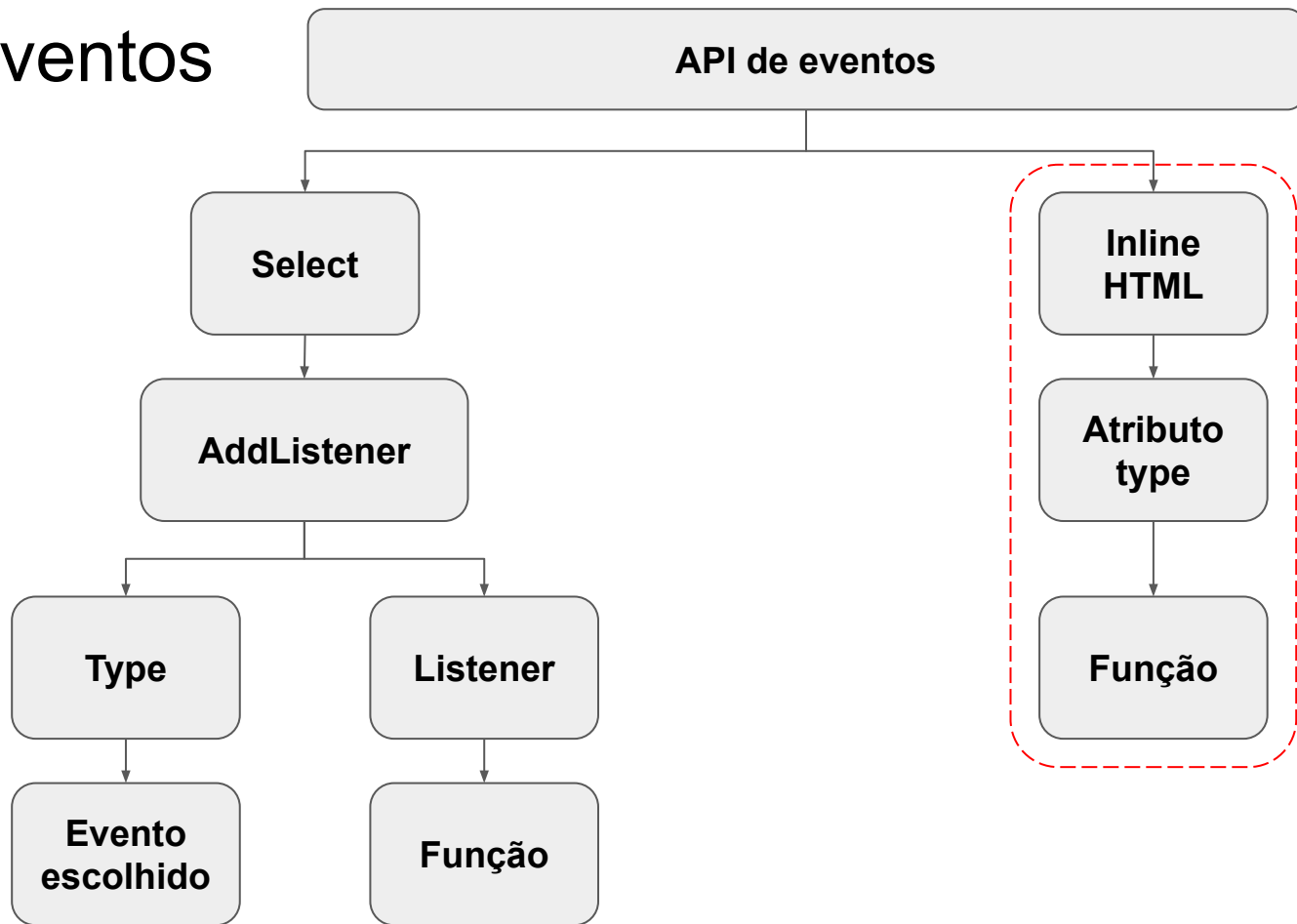
Juntando o que sabemos até agora



Eventos



Eventos



Eventos

API de eventos

Inline
HTML

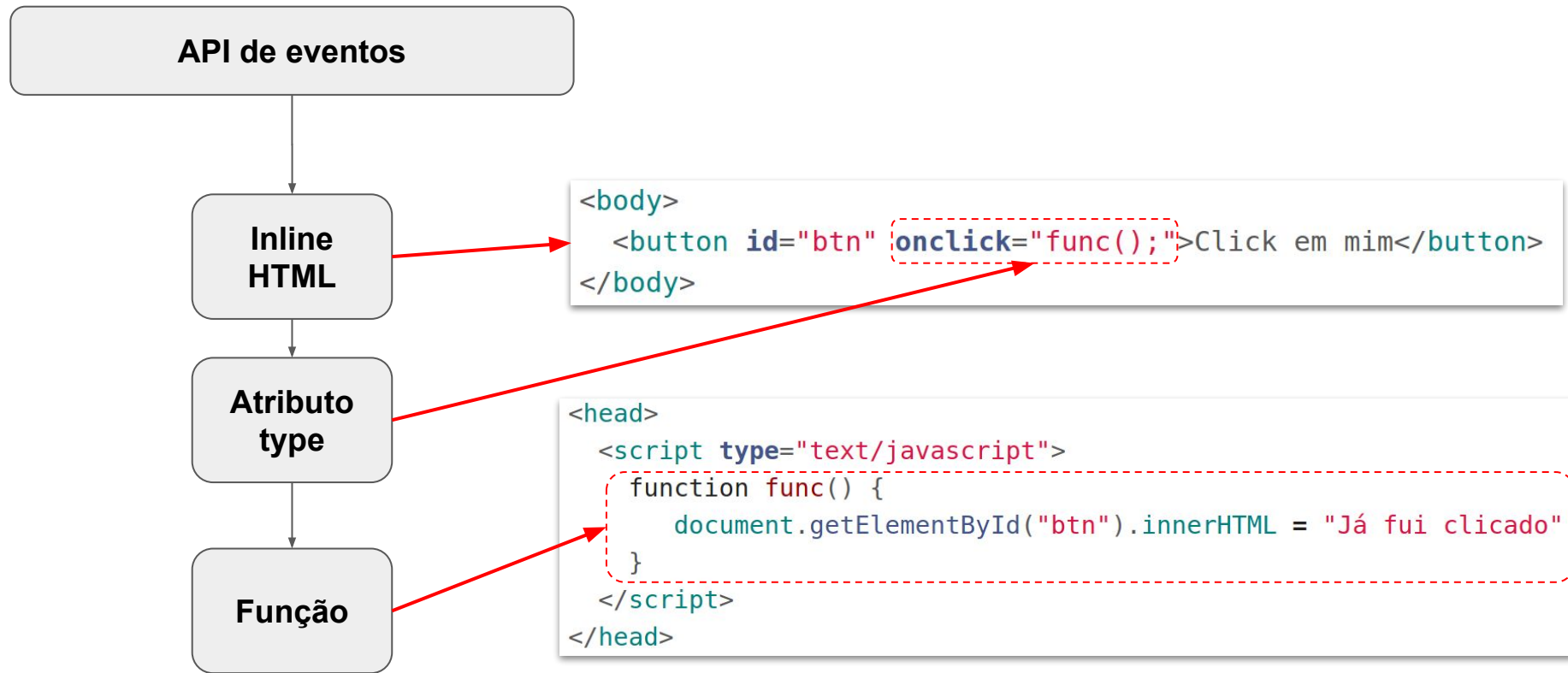
Atributo
type

Função

```
<body>
  <button id="btn" onclick="func();">Click em mim</button>
</body>
```

```
<head>
  <script type="text/javascript">
    function func() {
      document.getElementById("btn").innerHTML = "Já fui clicado"
    }
  </script>
</head>
```


Eventos



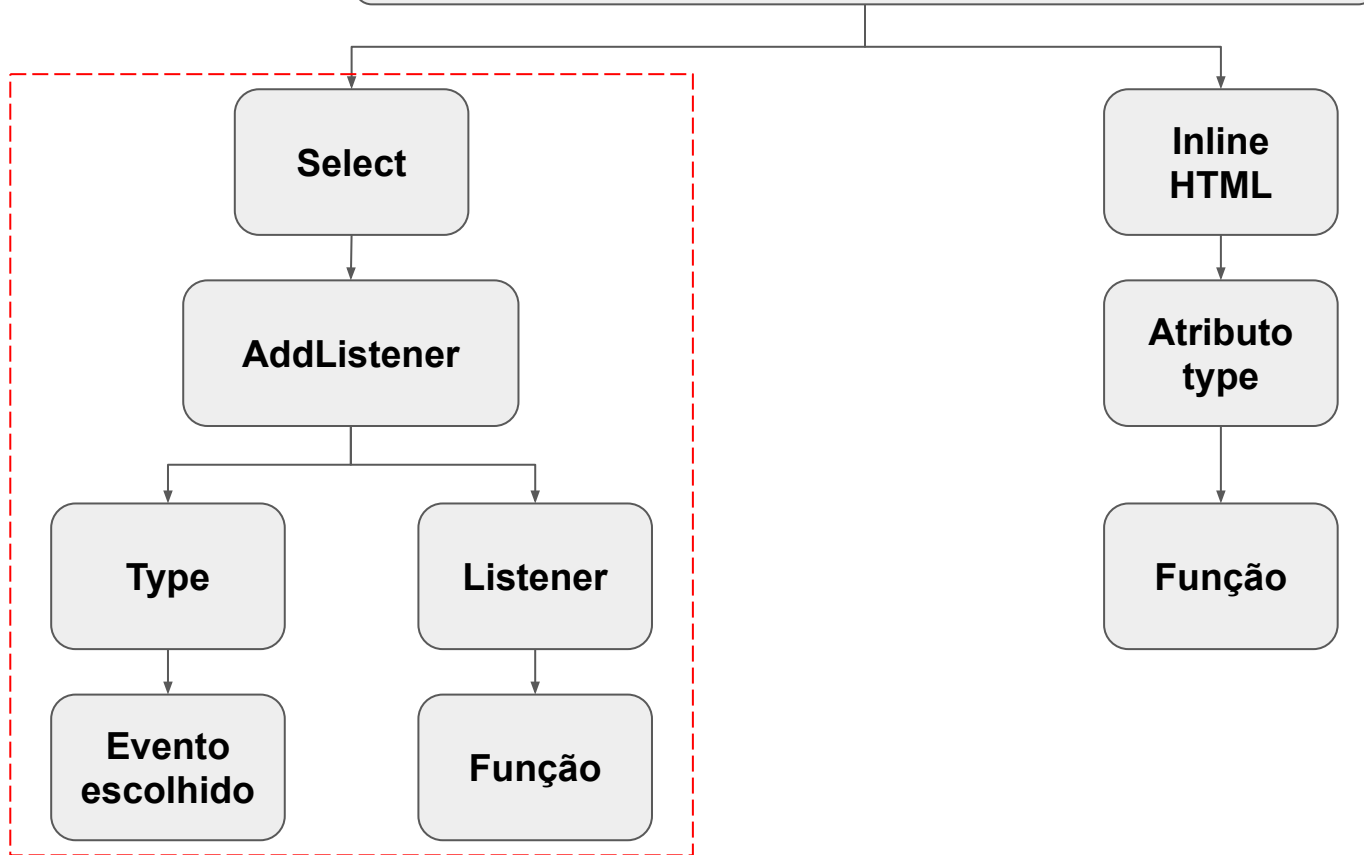
Vamos acessar

https://selenium.dunossauro.live/aula_07_b

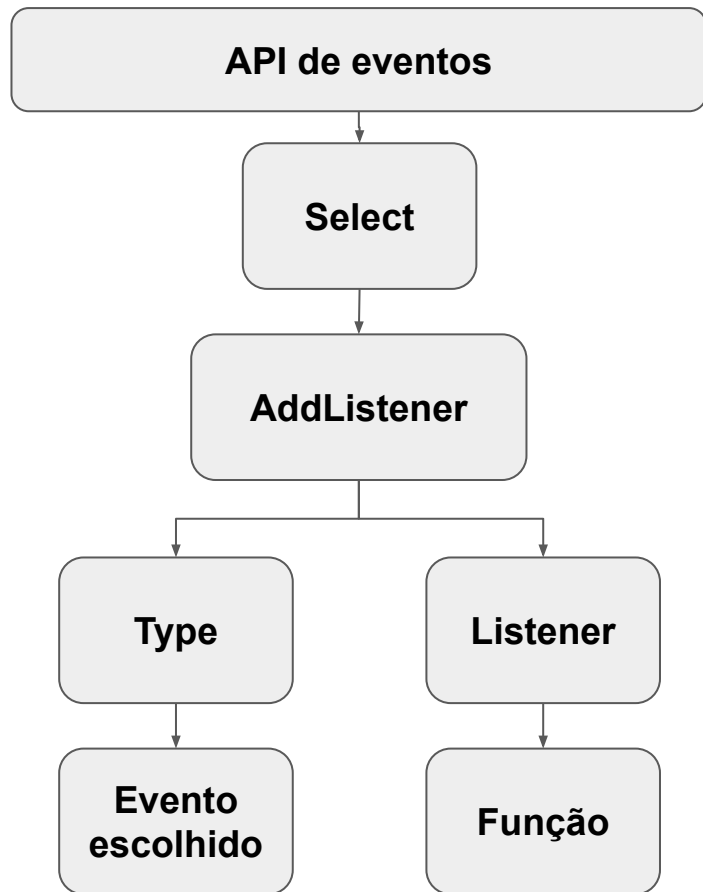


Eventos

API de eventos



Eventos

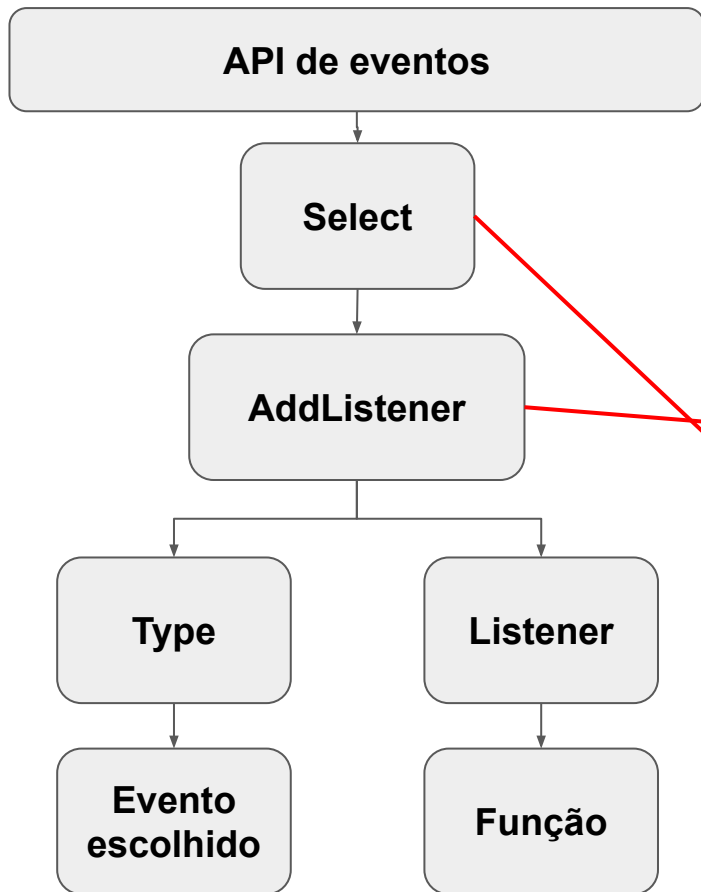


```
<body>
  <button id="btn">Click em mim</button>

  <script type="text/javascript">
    function func() {
      document.getElementById("btn").innerHTML = "Já fui clicado"
    }

    document.getElementById("btn").addEventListener("click", func);
  </script>
</body>
```

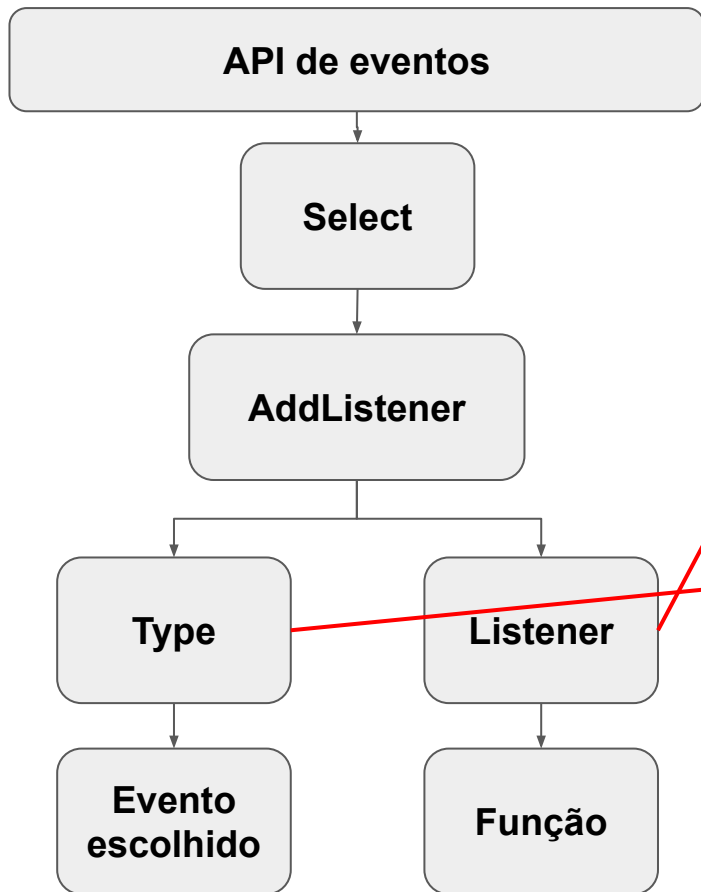
Eventos



```
<body>
  <button id="btn">Click em mim</button>

  <script type="text/javascript">
    function func() {
      document.getElementById("btn").innerHTML = "Já fui clicado"
    }
    document.getElementById("btn").addEventListener("click", func);
  </script>
</body>
```

Eventos



```
<body>
  <button id="btn">Click em mim</button>

  <script type="text/javascript">
    function func() {
      document.getElementById("btn").innerHTML = "Já fui clicado"
    }

    document.getElementById("btn").addEventListener("click", func);
  </script>
</body>
```

Vamos acessar

https://selenium.dunossauro.live/aula_07_c



Trabalhando com eventos



Os eventos que vamos estudar*

- Foco (focus)
- Mudança (change)
- Mouse
- Drag
- Teclado

<https://developer.mozilla.org/en-US/docs/Web/Events>



Os eventos que vamos estudar*

- Foco (focus)
 - Mudança (change)
 - **Mouse**
 - **Drag**
 - **Teclado**
- } **Aula 08**

<https://developer.mozilla.org/en-US/docs/Web/Events>



Focus x Blur



Focus x Blur

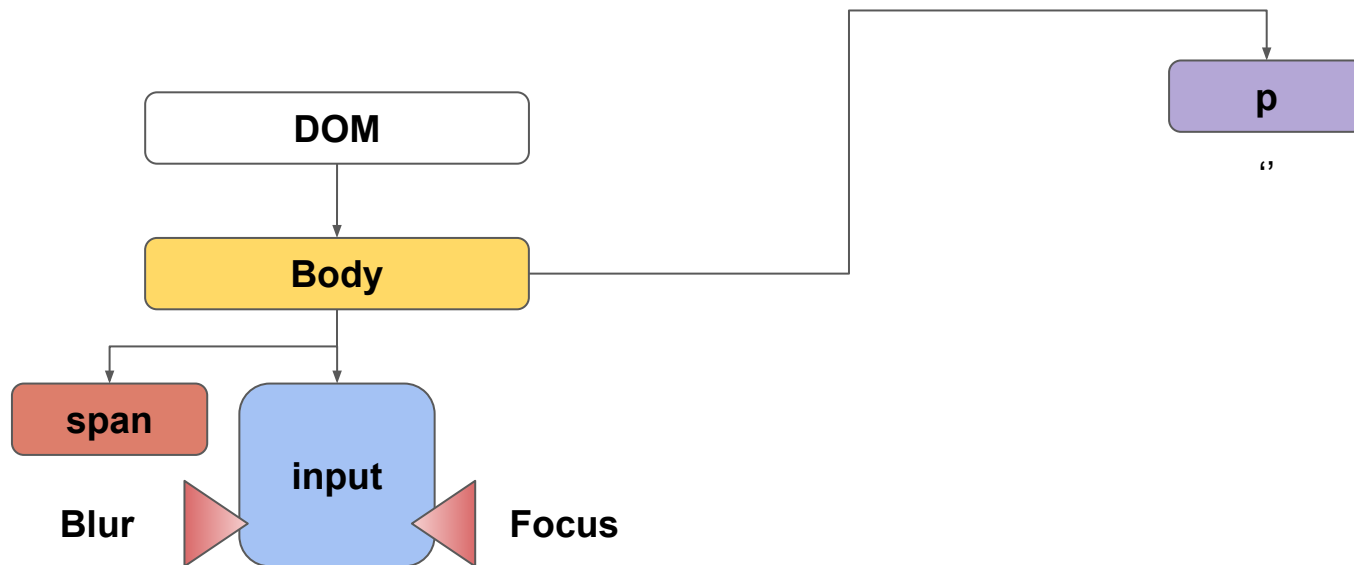
Evento de foco. Quando o elemento é “acessado” ele está em foco. Com isso o evento “Focus” é disparado.

Quando o elemento perde o foco, o evento Blur é desencadeado

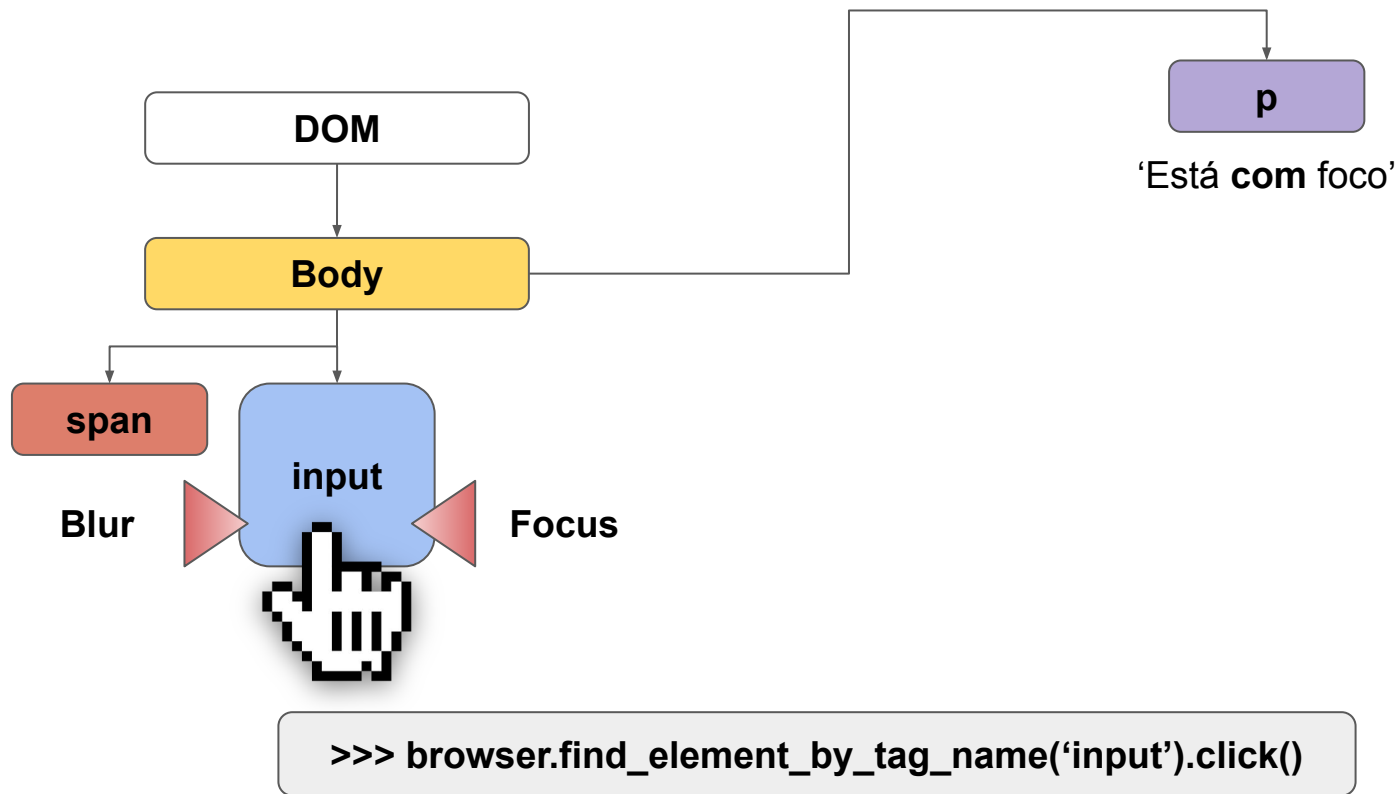
https://selenium.dunossauro.live/aula_07_d



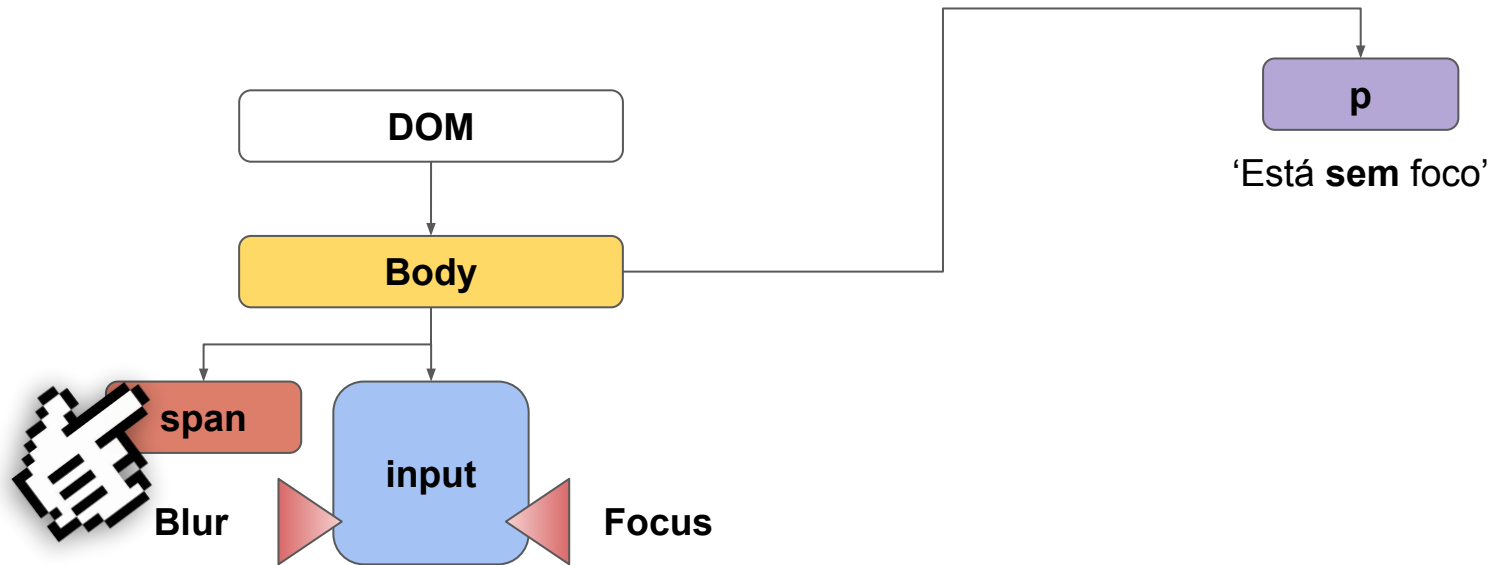
Ilustrando eventos no DOM



Ilustrando eventos no DOM



Ilustrando eventos no DOM



```
>>> browser.find_element_by_tag_name('span').click()
```

Change



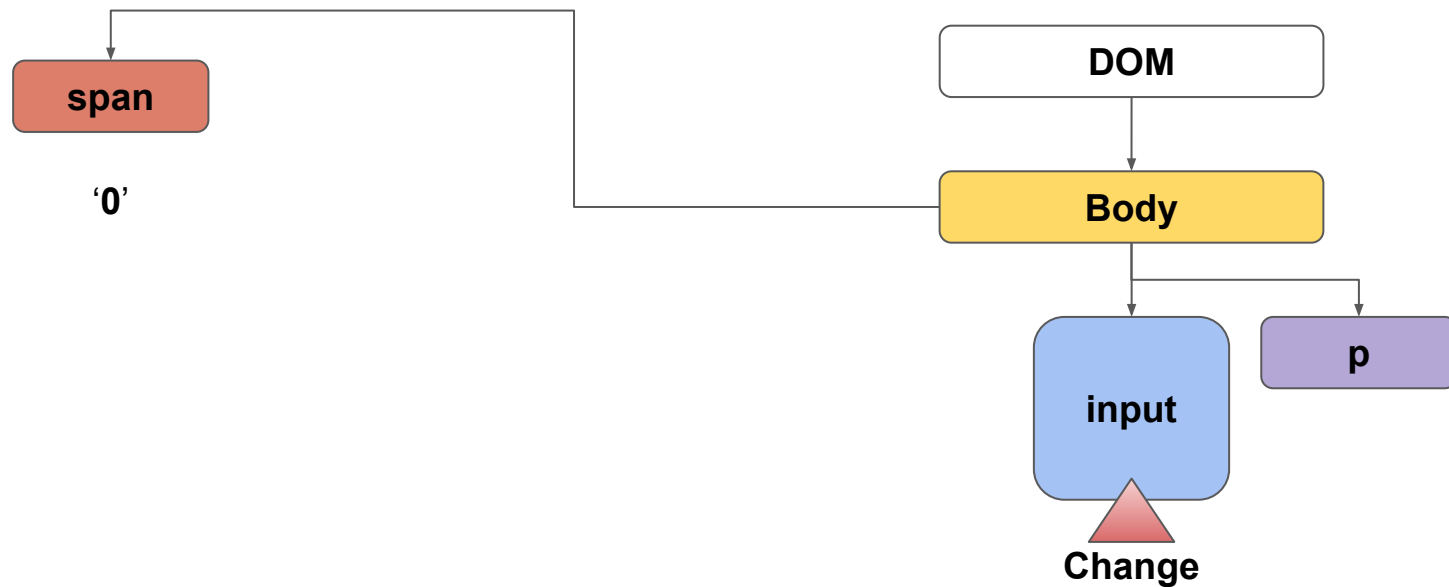
Change

O evento de mudança [change] é desencadeado quando um elemento perde o foco [blur]. O elemento será analisado e caso alguma mudança tenha ocorrido durante o período de foco, ele será disparado.

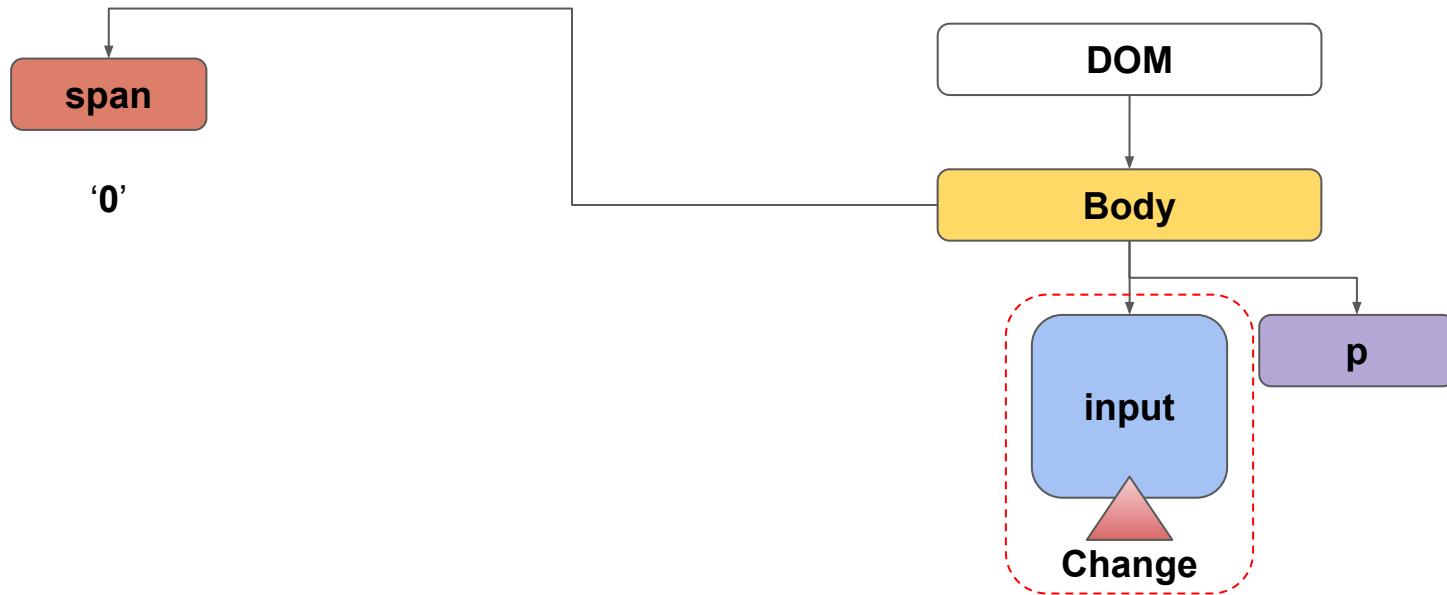
https://selenium.dunossauro.live/aula_07_d



Ilustrando eventos no DOM



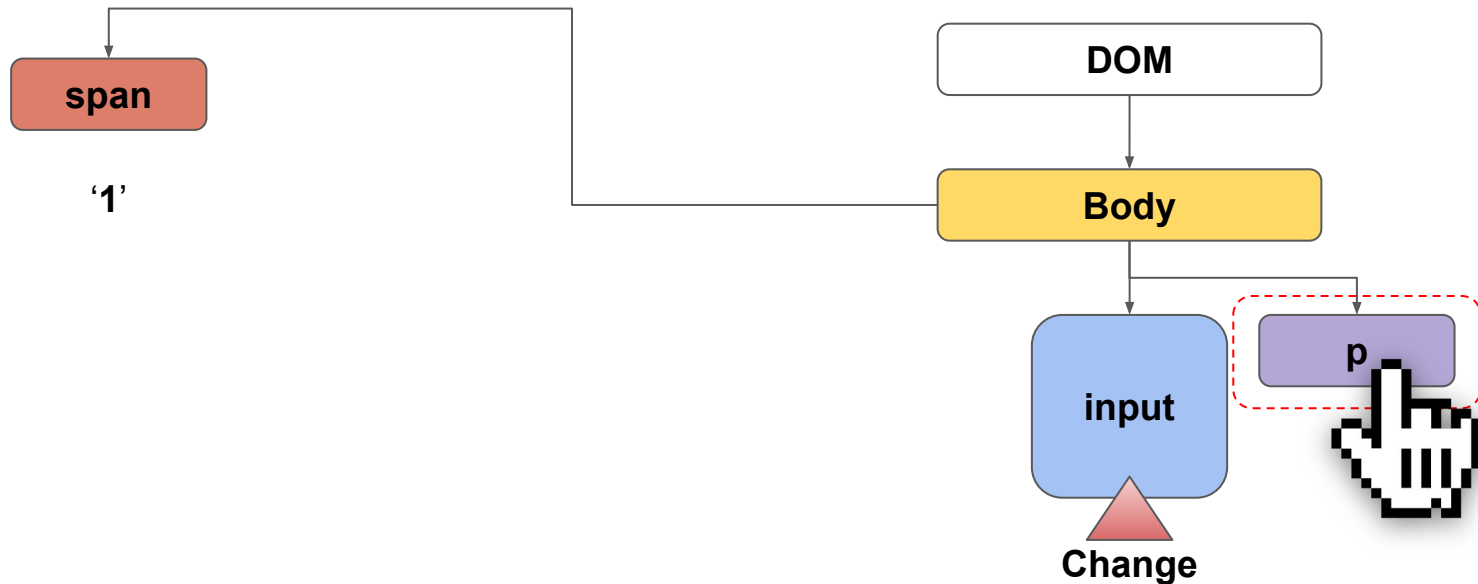
Ilustrando eventos no DOM



```
>>> browser.find_element_by_tag_name('input').send_keys('eduardo')
```



Ilustrando eventos no DOM



```
>>> browser.find_element_by_tag_name('input').send_keys('eduardo')  
>>> browser.find_element_by_tag_name('p').click()
```



Vamos codar um tikin


https://selenium.dunossauro.live/aula_07_d



Para não esquecer



Financiamento



Live de Python

Python explicado de maneira simples, didática e gratuita


[Perfil](#) [Mural\(2\)](#) [Apoiadores](#)




Meu nome é Eduardo Mendes eu toco semanalmente lives no youtube ensinando python.

R\$ **1.010** arrecadados por mês


Meta: R\$ 1.500 por mês (67.3% alcançada)

 44 pessoas apoiando


[Apoiar agora](#)


 continua por mês 

Compartilhe:







livedepython

<https://picpay.me/livedepython>

Eventos + Selenium

Parte 1



Observando os eventos

A biblioteca do selenium conta com um mecanismo para observar eventos. Um `EventListener` [Escutador de eventos]. A implementação dele é baseada em um padrão de projeto chamado **Template Method**.

A ideia principal é fazer uma ação “antes” e/ou “depois” de alguma determinada ação do WebDriver



Observando os eventos

“Antes” e “Depois”, geralmente são conhecidos como **Hooks**.

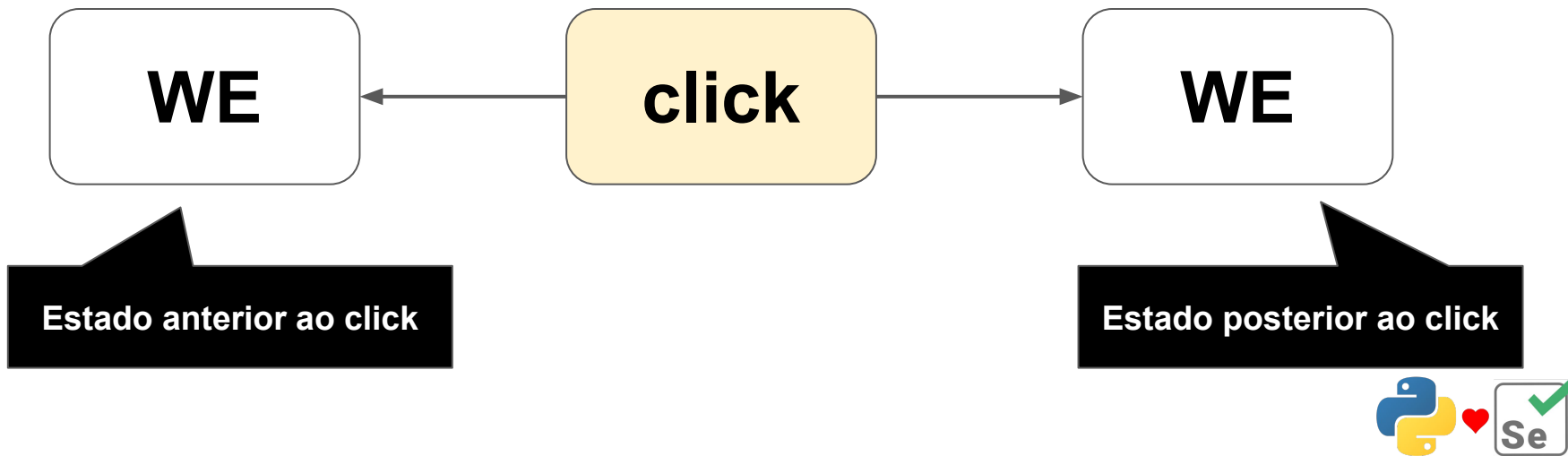
Vamos usar o método ‘click’, como exemplo.



Observando os eventos

“Antes” e “Depois”, geralmente são conhecidos como **Hooks**.

Vamos usar o método ‘click’, como exemplo.



EventListener

https://selenium-python.readthedocs.io/api.html#module-selenium.webdriver.support.abstract_event_listener



EventListener

O objetivo do EventListener é observar o estado do WD em todos os momentos. Antes e depois de uma ação ser executada.

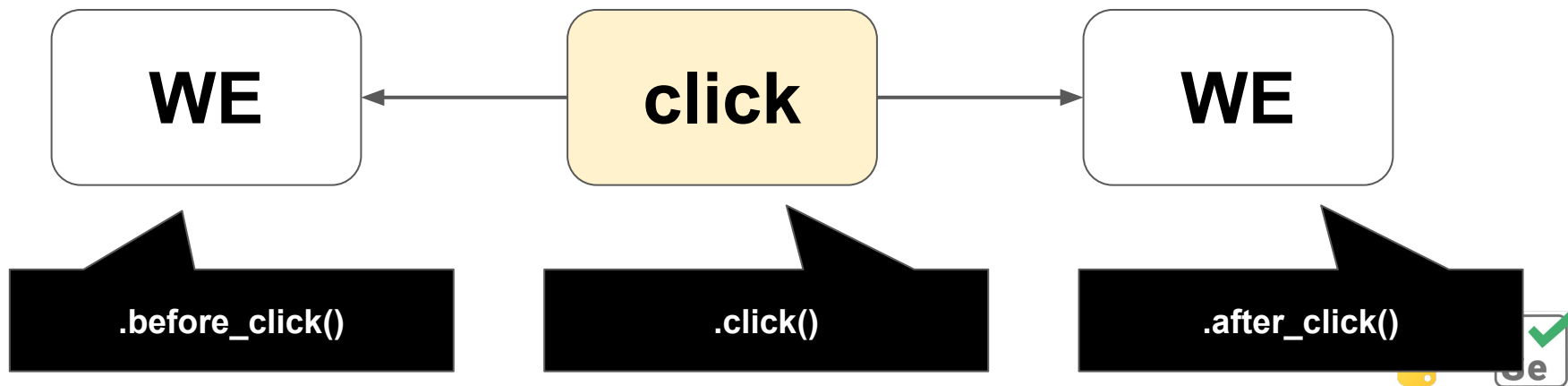
Por exemplo, todas as vezes em que o 'click' for chamado podemos observar o estado do dom, de um único elemento, fazer logs, etc...



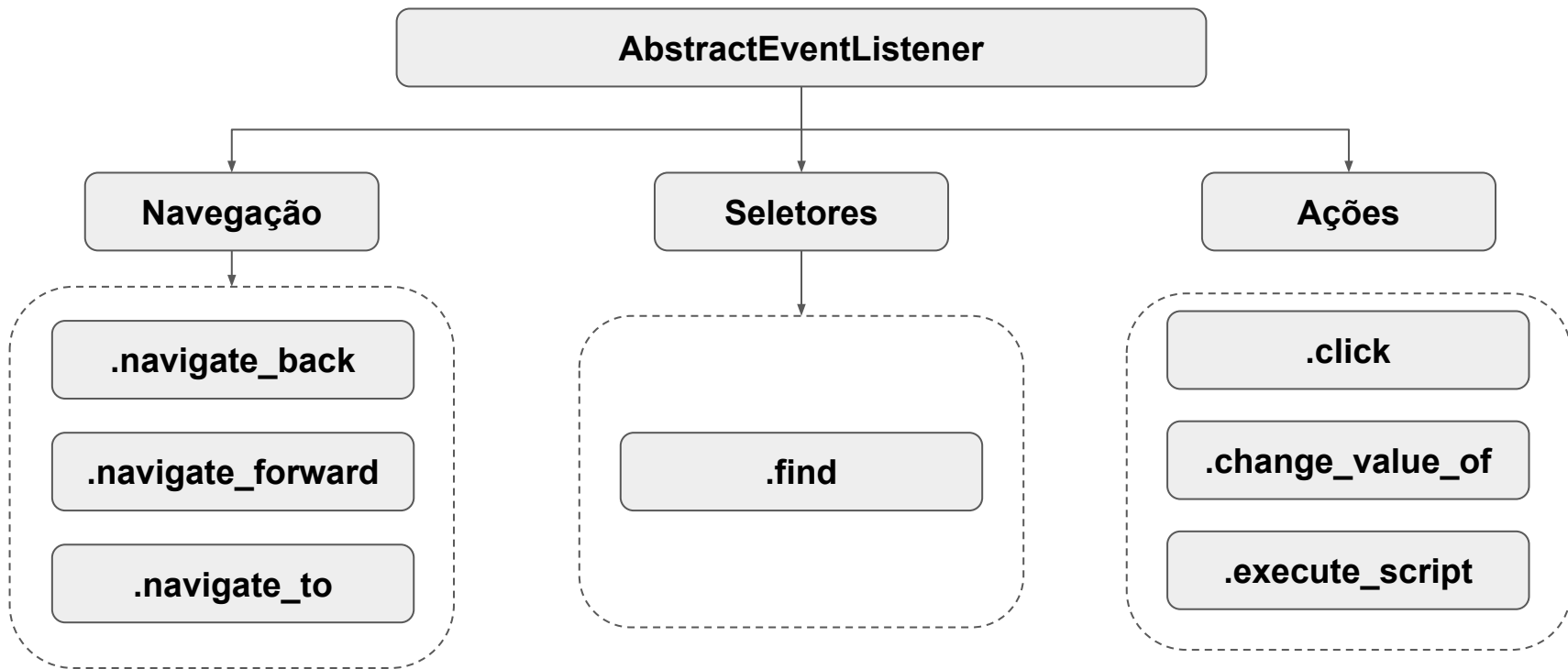
EventListener

O objetivo do EventListener é observar o estado do WD em todos os momentos. Antes e depois de uma ação ser executada.

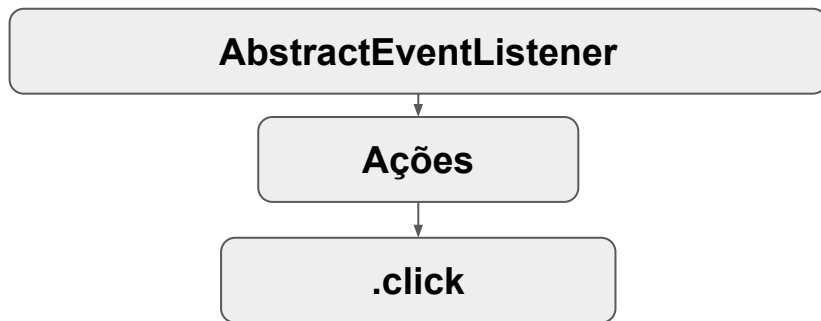
Por exemplo, todas as vezes em que o 'click' for chamado podemos observar o estado do dom, de um único elemento, fazer logs, etc...



Uma breve olhada

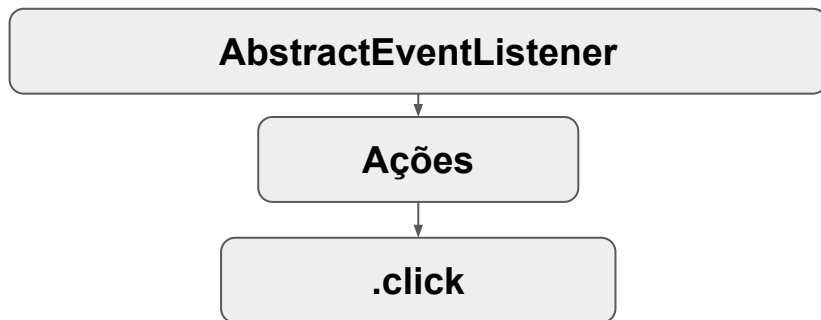


Uma breve olhada



```
class Escuta(AbstractEventListener):  
    def before_click(self, elemento, driver):  
        print(  
            f'Tag {elemento.tag_name}: Texto: {elemento.text}: Antes de clicar'  
        )  
  
    def after_click(self, elemento, driver):  
        print(  
            f'Tag {elemento.tag_name}: Texto: {elemento.text}: Depois de clicar'  
        )
```


Uma breve olhada



```
class Escuta(AbstractEventListener):  
    def before_click(self, elemento, driver):  
        print(  
            f'Tag {elemento.tag_name}: Texto: {elemento.text}: Antes de clicar'  
        )  
  
    def after_click(self, elemento, driver):  
        print(  
            f'Tag {elemento.tag_name}: Texto: {elemento.text}: Depois de clicar'
```

Vamos codar um tikin

https://selenium.dunossauro.live/aula_07_d



Event Firing

Disparador de eventos



Event Firing

O disparador de eventos é uma “*burocracia*” do selenium para usar um Listener.

Ele constrói um *wrapper* do webdriver e dispara os eventos para o Listener.



Event Firing

```
from selenium.webdriver import Firefox
from selenium.webdriver.support.events import (
    EventFiringWebDriver,
    AbstractEventListener,
)

class MeuOuvinte(AbstractEventListener):
    def after_change_value_of(self, elemento, driver):
        ...

    def before_click(self, elemento, driver):
        ...

    def after_click(self, elemento, driver):
        ...

f = Firefox()

new_driver = EventFiringWebDriver(f, MeuOuvinte())
```

O disparador de eventos é uma “*burocracia*” do selenium para usar um Listener.

Ele constrói um *wrapper* do webdriver e dispara os eventos para o Listener.



Event Firing

```
from selenium.webdriver import Firefox
from selenium.webdriver.support.events import (
    EventFiringWebDriver,
    AbstractEventListener,
)

class MeuOuvinte(AbstractEventListener):
    def after_change_value_of(self, elemento, driver):
        ...

    def before_click(self, elemento, driver):
        ...

    def after_click(self, elemento, driver):
        ...

f = Firefox()

new_driver = EventFiringWebDriver(f, MeuOuvinte())
```

O disparador de eventos é uma “*burocracia*” do selenium para usar um Listener.

Ele constrói um *wrapper* do webdriver e dispara os eventos para o Listener.



Vamos codar um tikin

https://selenium.dunossauro.live/aula_07_d



Exercícios

https://selenium.dunossauro.live/exercicio_07

Refazer:

https://selenium.dunossauro.live/exercicio_03

* usar o EventListener para printar a o after de
navegação e de clicks



Lições para casa:

- Assistir a Live de Python #61 - Introdução a orientação a objetos
 - Se lidar com classes for um problema para você
- Assistir a Live de Python #68 - Interfaces de ABCs
 - Se quiser entender o que é abstrato
- Assistir a Live de Python #115 - Introdução aos padrões de projeto
 - Vamos usar esse conceitos na aula de page objects
- Assistir a Live de Python #117 - Padrão template method
 - Se quiser entender como funciona o AbstractEventListener

