UI-Testing
Framework Playwright

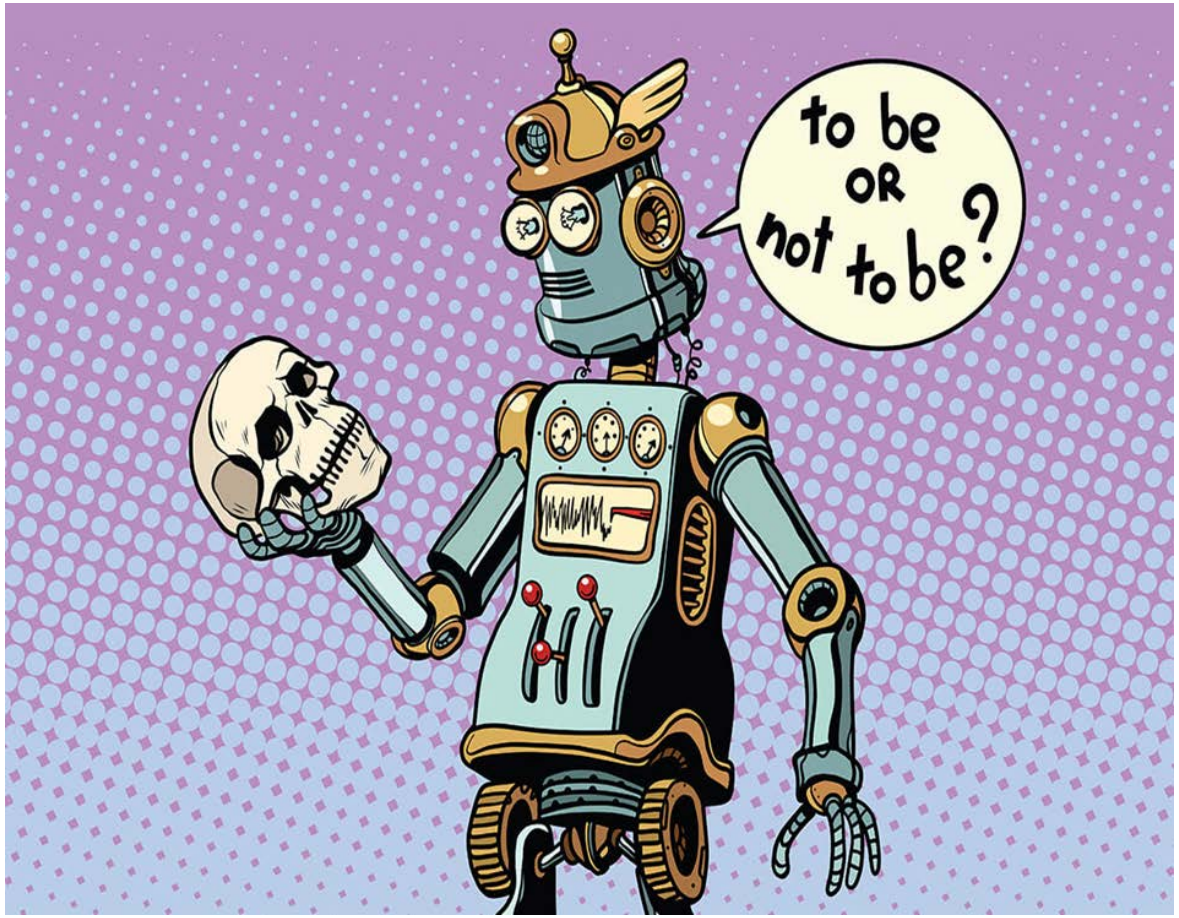
Agenda

1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

Agenda

1. **Neue Frameworks und Tools für das UI-Testing**
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

1. Neue Frameworks und Tools für das UI-Testing



- Gründe für die Auswahl eines neuen Frameworks
- Playwright vs Selenium

1. Neue Frameworks und Tools für das UI-Testing

Playwright vs Selenium

Kriterium	Playwright	Selenium
Programmiersprache	JavaScript, Java, Python und .NET C#	Java, Python, C#, Ruby, Perl, PHP, and JavaScript
Installation	Leicht zu installieren	Leicht zu installieren
Unterstützte Testrunner	Mocha, Jest, Jasmine	Mocha, Jest, Jasmine, Protractor, and WebDriverIO
Installationsbasis	NodeJS	Java, Eclipse IDE, SeleniumStandalone Server, Client Language Bindings und Browser Drivers

1. Neue Frameworks und Tools für das UI-Testing

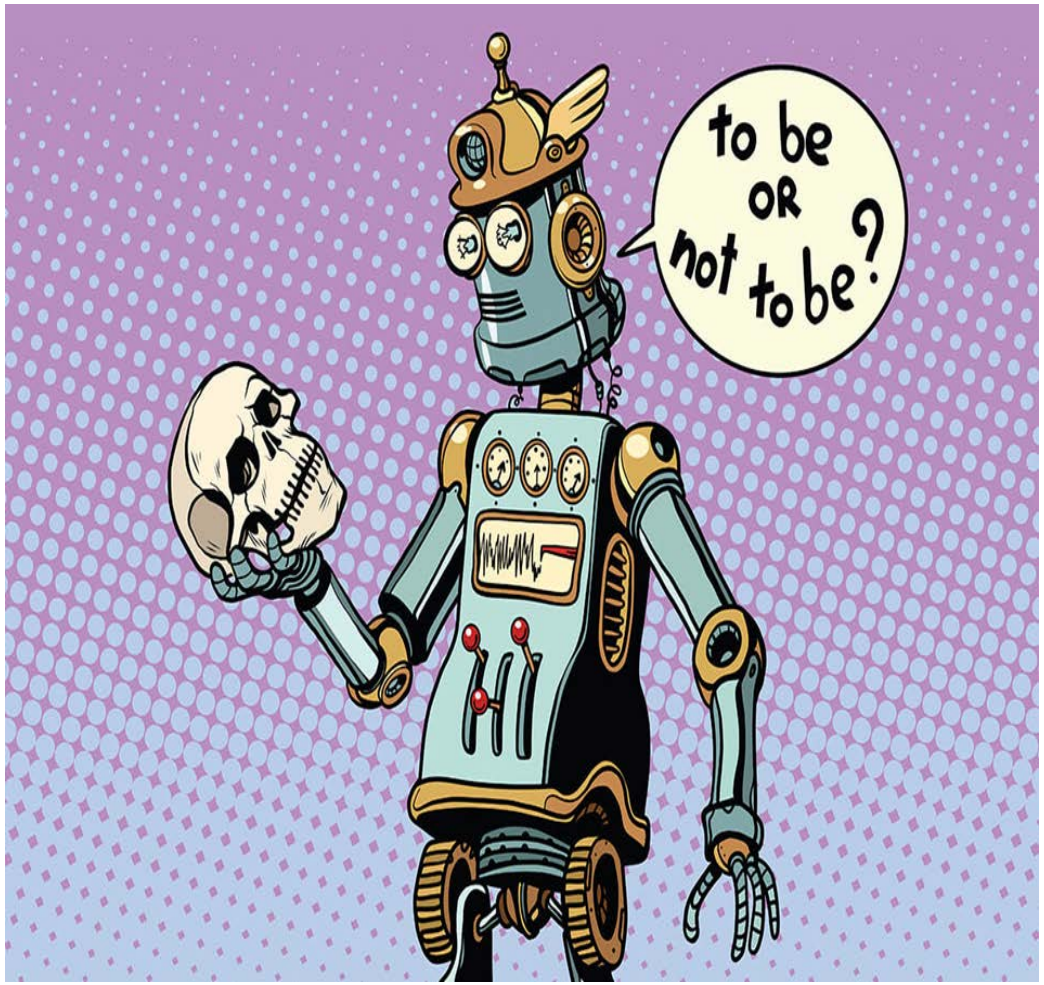
Playwright vs Selenium

Kriterium	Playwright	Selenium
Unterstützte Betriebssysteme	Windows, Linux, and Mac OS	Windows, Linux, and Mac OS
Lizenzmodell	Open Source und kostenfrei	Open Source und kostenfrei
Unterstützte Browser	Chromium, Firefox und WebKit	Chrome, Firefox, IE, Edge, Opera, Safari
Support	Begrenzt	Umfangreich
Hardware Unterstützung	Keine	Vorhanden

Agenda

1. Neue Frameworks und Tools für das UI-Testing
2. **Playwright - Überblick**
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

2. Playwright - Überblick



- Unterstützung von allen Browsern
- Schnelle und zuverlässige Testausführung
- Einstellungen für viele verschiedene Testszenarien
- Einfache Integration in den Softwareentwicklungsprozess
- Einschränkungen

Agenda

1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. **Installation**
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

3. Installation

- NodeJS
- > **npm i -D playwright**
- Evtl. TypeScript Installation

3. Installation

- Installation der Browser in einen vom Benutzer spezifizierten Ordner

Linux/macOS

> **PLAYWRIGHT_BROWSERS_PATH=$HOME/pw-browsers npm i -D playwright**

3. Installation

- Installation der Browser in einen vom Benutzer spezifizierten Ordner

Windows

> set PLAYWRIGHT_BROWSERS_PATH=%USERPROFILE%\pw-browsers

> npm i -D playwright

3. Installation

- Installation – Firewall oder Proxy

Linux/macOS

> HTTPS_PROXY=https://192.168.1.78 npm i -D playwright

3. Installation

- Installation – Firewall oder Proxy

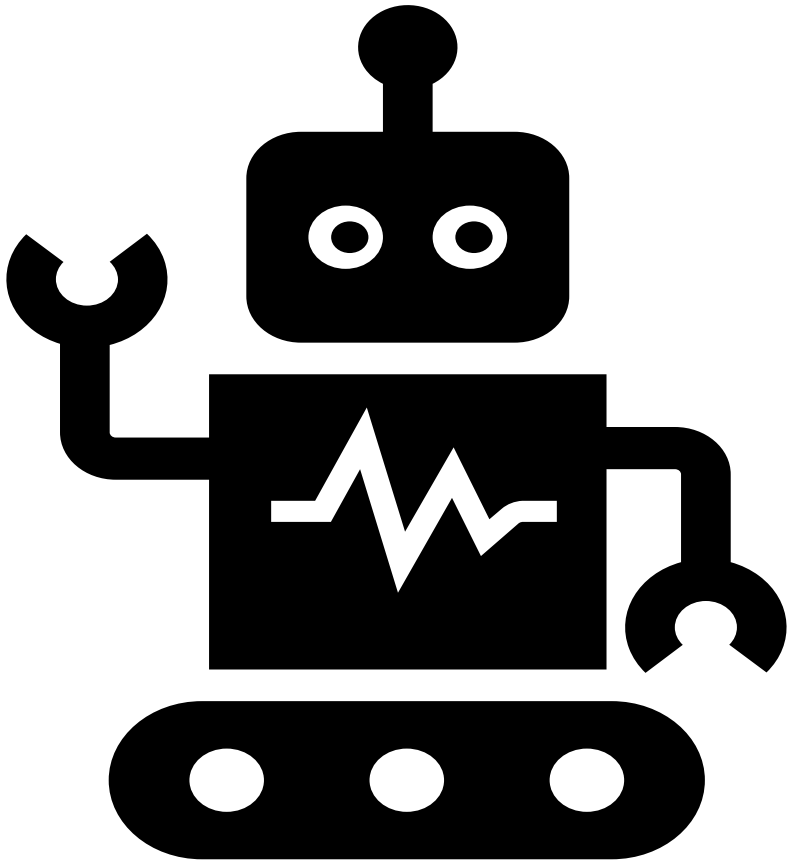
Windows

> set HTTPS_PROXY=https://192.168.1.78
$ npm i -D playwright

Agenda

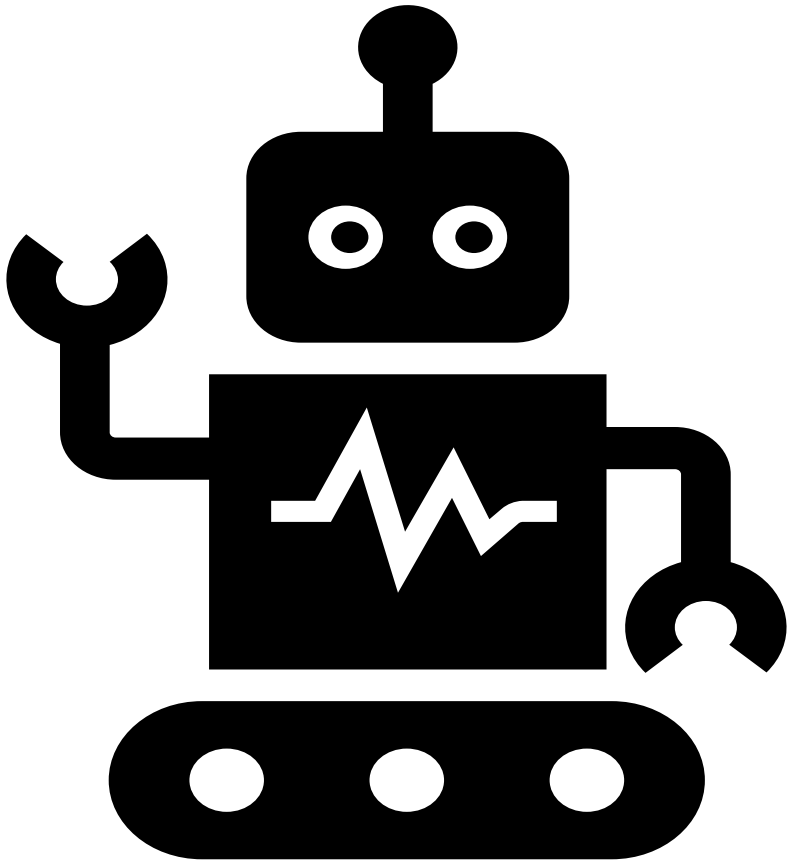
1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. **Basiskonzepte**
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

4. Basiskonzepte



- Browser
- Pages und Frames
- Element Selectors
- Auto-waiting
- Execution context
- Evaluation argument

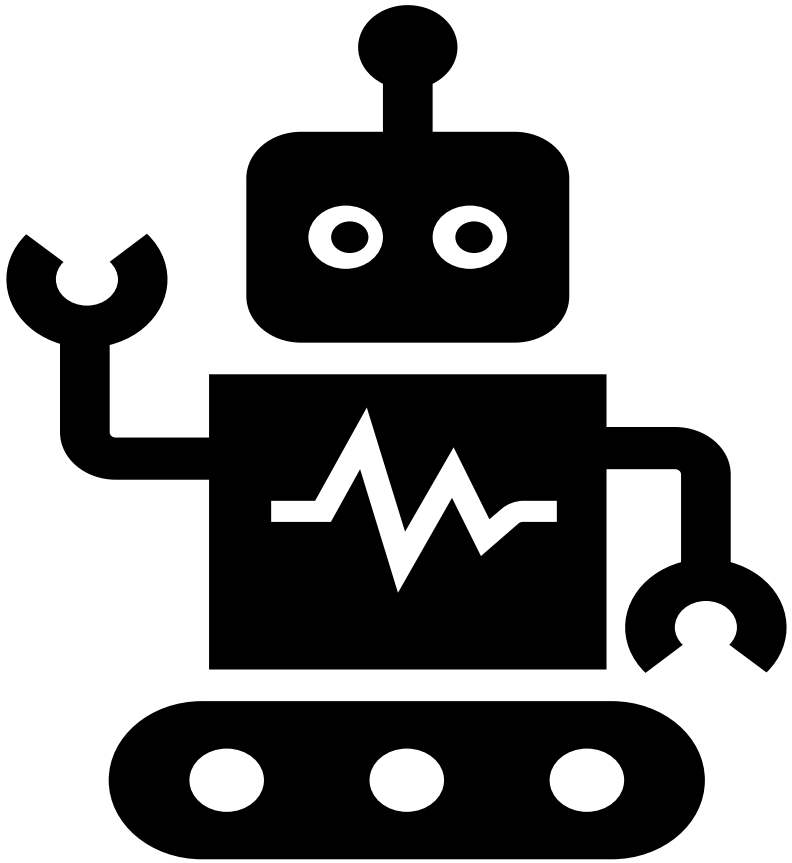
4. Basiskonzepte – Browser – JavaScript-Beispiel



- Abgeleitet von der Klasse **EventEmitter** (Node.js)
- Browser-Start

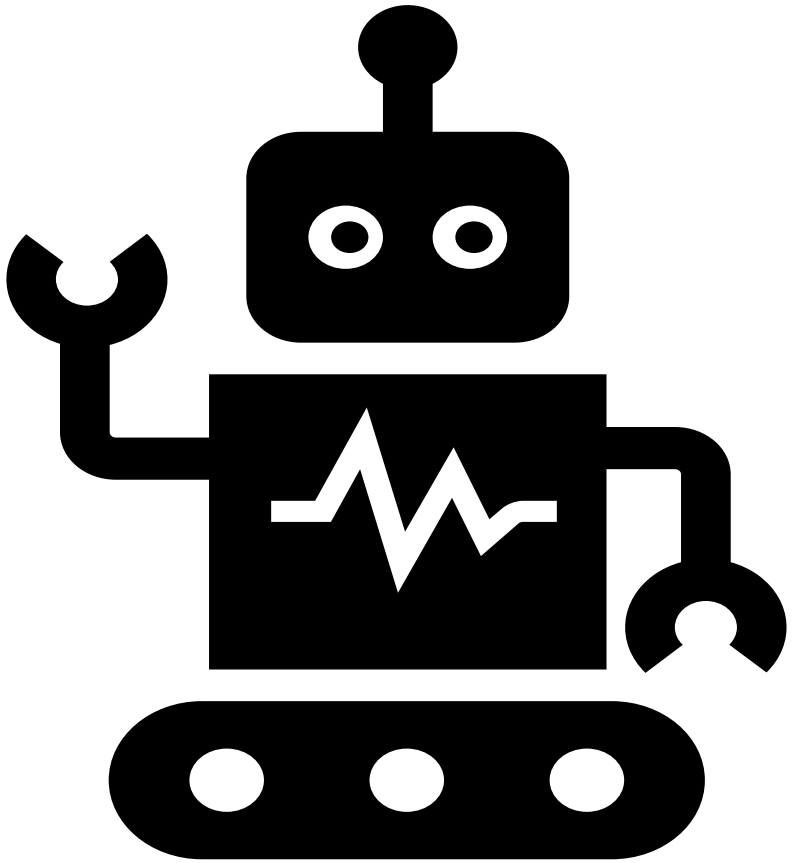
```
const { chromium } = require('playwright');
const browser = await chromium.launch({ headless:
    false });
await browser.close();
```

4. Basiskonzepte – Browser – TypeScript-Beispiel



```
import {chromium, Browser, Page} from ‘playwright’

.............
describe('Regressiontest – Für TACON’, () => {
    let browser: Browser
    let page: Page


    const isHeadless = true
    const windowSize = {width: 500, height: 800}
.............

}
```

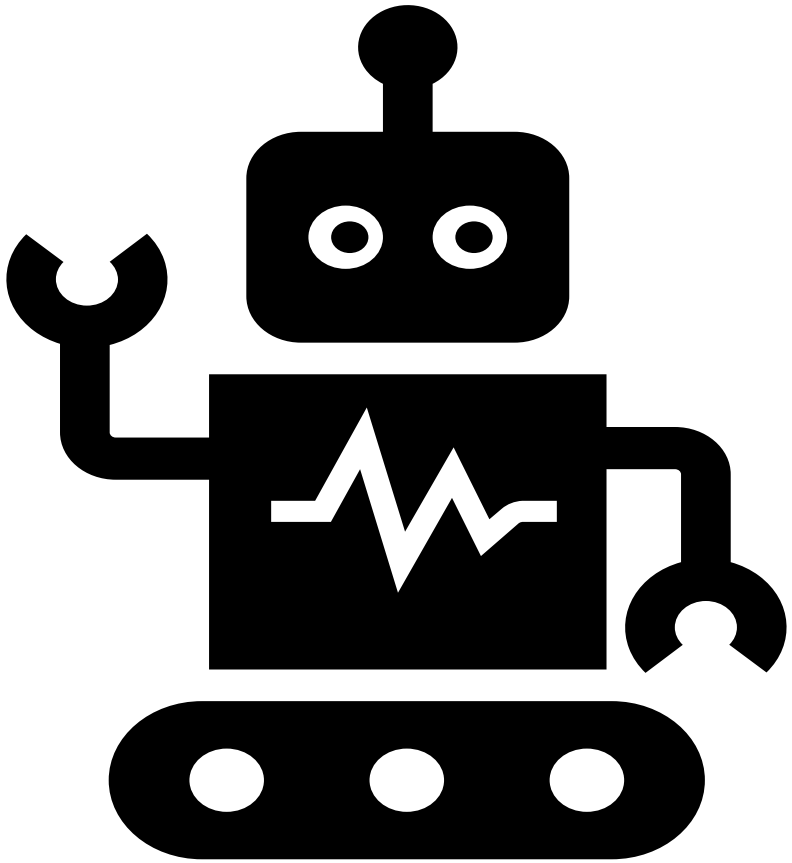
4. Basiskonzepte – Page und Frame



- Beispielcode

```
const browser = await firefox.launch();
const page = await browser.newPage();
await page.goto('https://time.is/de/Paris');
await browser.close();
```

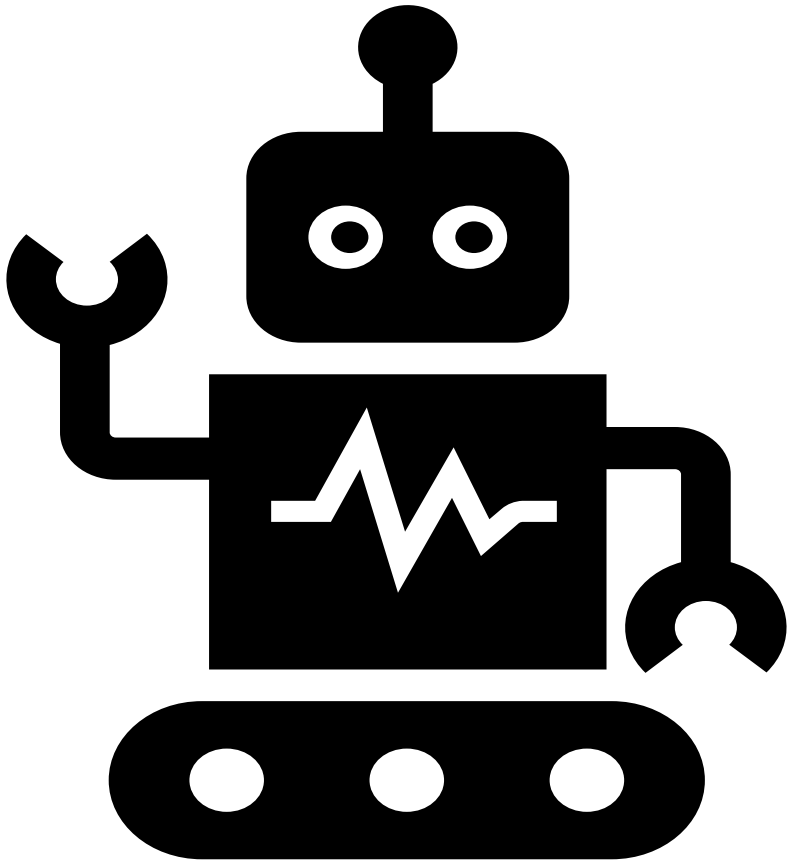
4. Basiskonzepte – Page und Frame



- Abgeleitet von der Klasse **EventEmitter** (Node.js)
- Kann Ereignisse senden:
  - **on**
  - **once**
  - **removeListener**

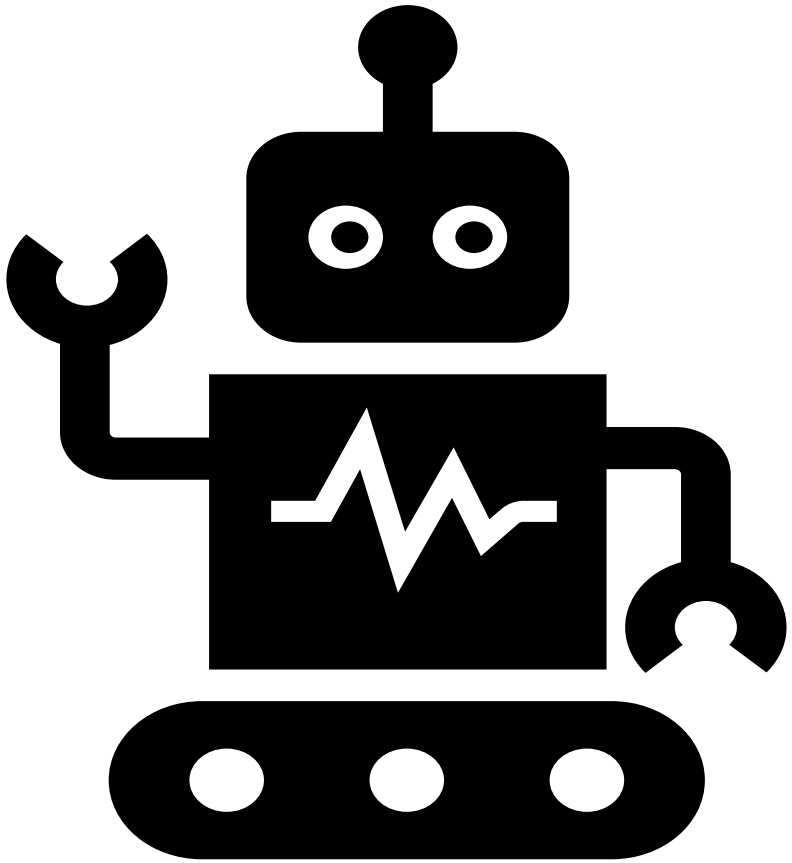
```
function logRequest(interceptedRequest) {
  console.log('Eine Anfrage wurde gesendet:',
    interceptedRequest.url());
}
page.on('request', logRequest);
// Subscription wird später enfernt...
page.removeListener('request', logRequest);
```

4. Basiskonzepte – Page und Frame



page.**once**('load', () => console.log(‚Die Seite wurde geladen!‘));

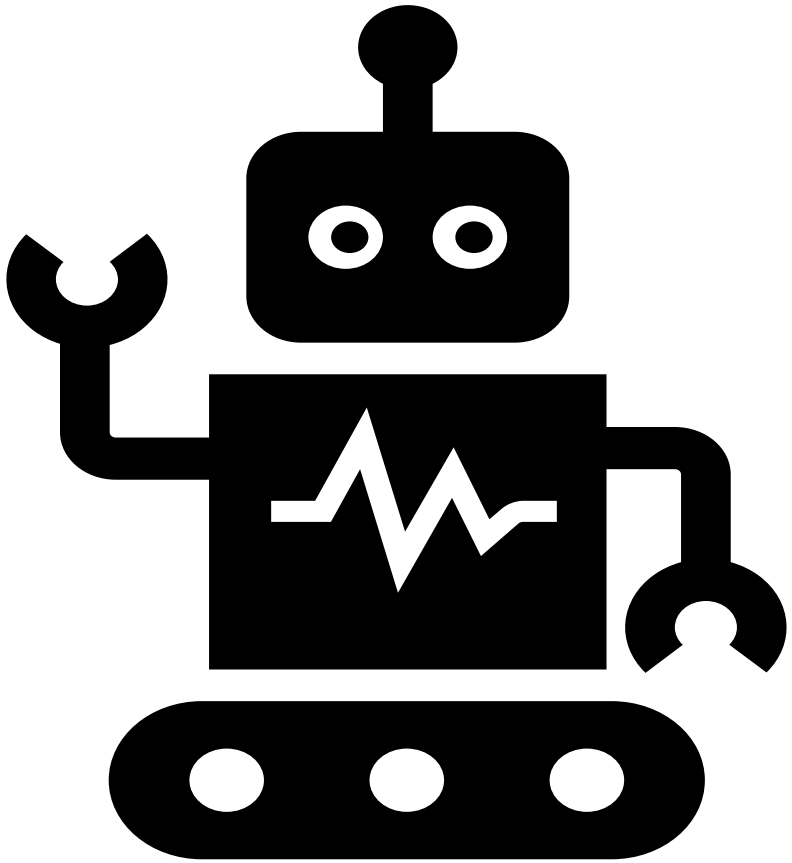
4. Basiskonzepte – Page und Frame



Bespiele für on-Subscriptions

- page.on('close')
- page.on('console')
- page.on('crash')
- page.on('dialog')
- page.on('domcontentloaded')
- page.on('download')
- page.on('filechooser')
- page.on('frameattached')
- page.on('framedetached')
- page.on('framenavigated')
- page.on('load')
- page.on('pageerror')
- page.on('popup')
- page.on('request')

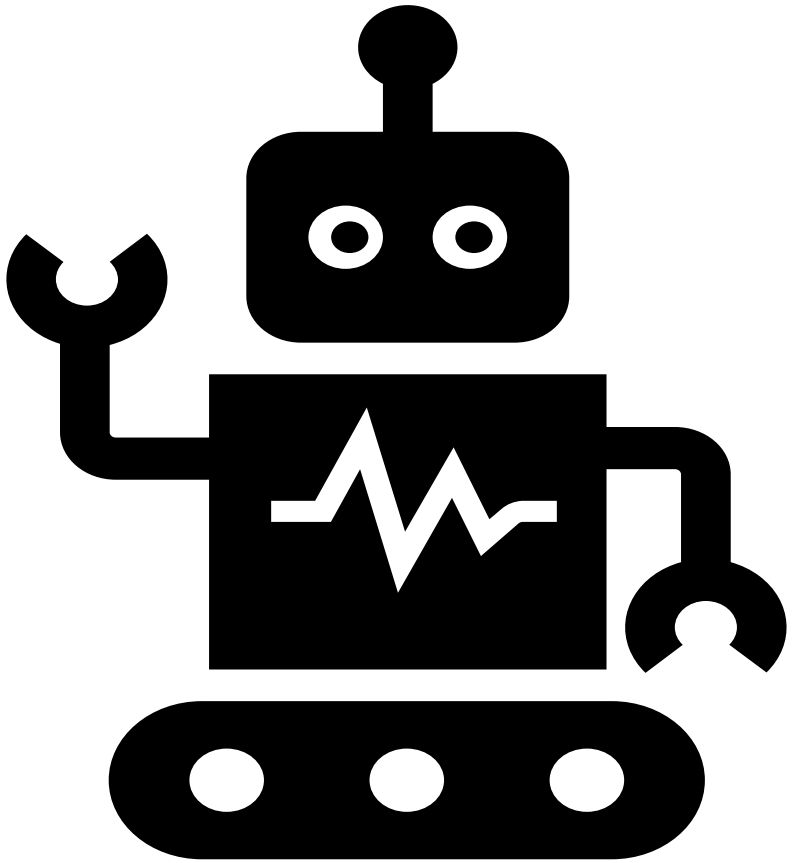
4. Basiskonzepte – Page und Frame



Am häufigsten benutzt

- page.**title**()
- page.**isVisible**(selector[, options])
- page.**waitForSelector**(selector[, options])
- page.**click**(selector[, options])

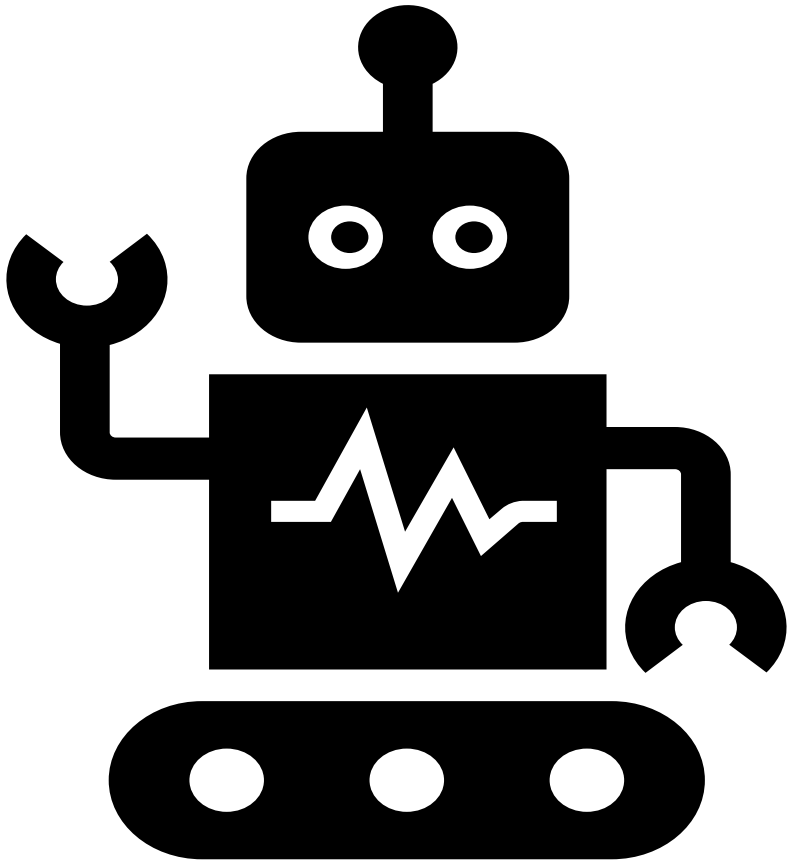
4. Basiskonzepte – Page und Frame



Subscriptions für Frame

- page.on('**frameattached**')
- page.on('**framenavigated**')
- page.on('**framedetached**')

4. Basiskonzepte – Element Selectors



Text Selector

await page.click('text=Log in‘);

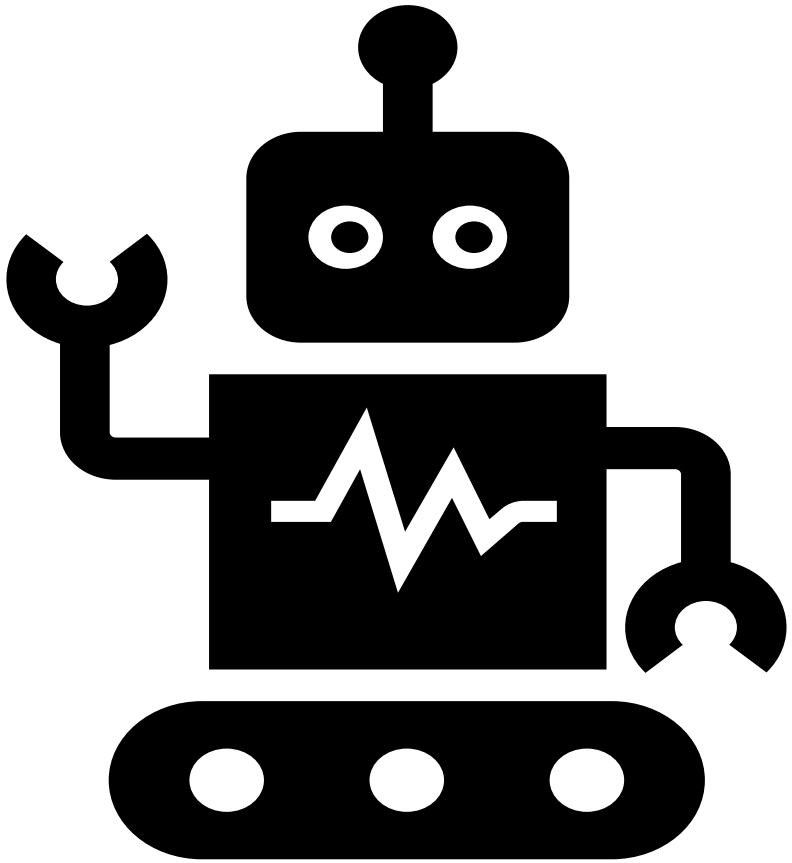
CSS Selector

await page.click('button');

CSS Selector mit Attribut

await page.click('[data-test=login-button]');

4. Basiskonzepte – Element Selectors



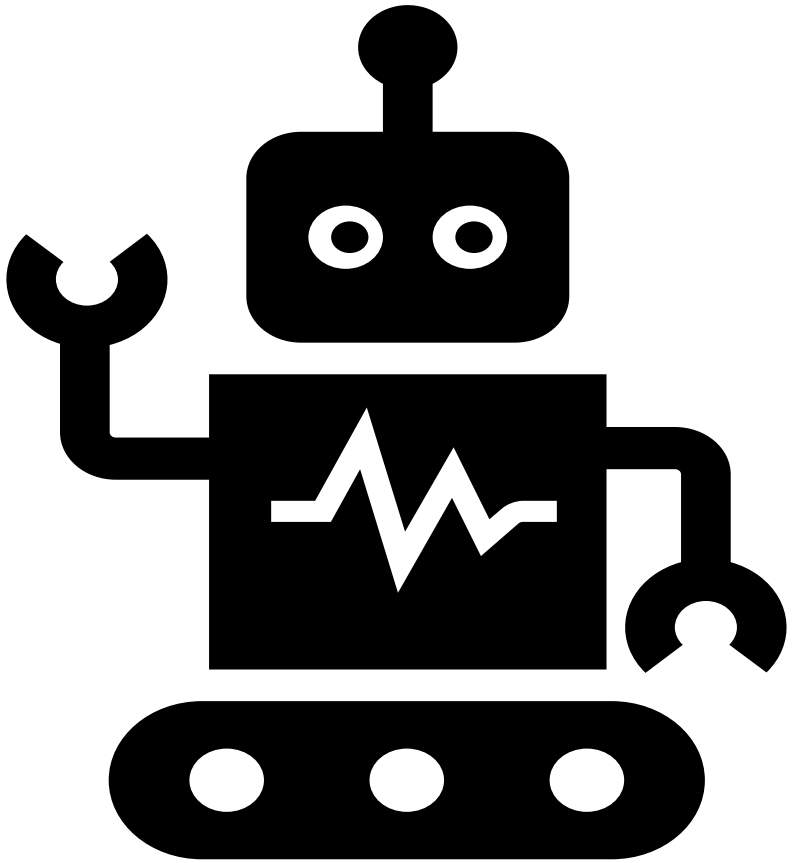
Kombination der Text- und CSS-Selectors

```
await page.click('article:has-text("Playwright")‘);
```

Verschachtelte CSS-Selectors

```
await page.click('.item-description:has(.item-promo-banner)');
```

4. Basiskonzepte – Element Selectors



Sichtbare Elemente mit CSS Selectors

```
await page.click('.login-button:visible’);
```

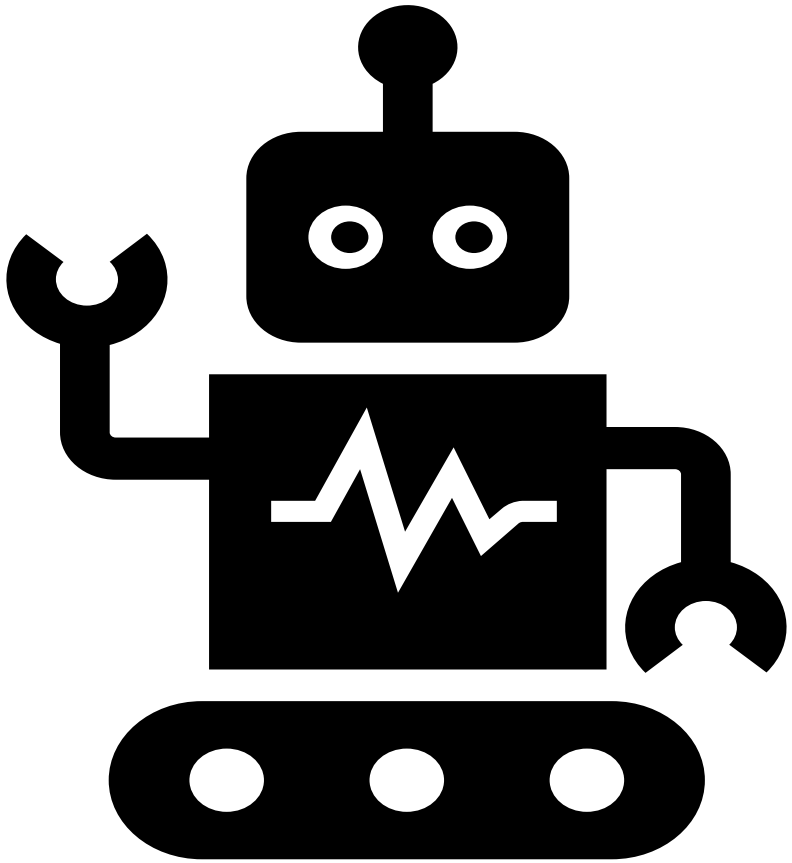
n-th Selectors

```
await page.click(':nth-match(:text(“Campus"), 3)’);
```

XPath Selectors

```
await page.click('xpath=//greenbutton');
```

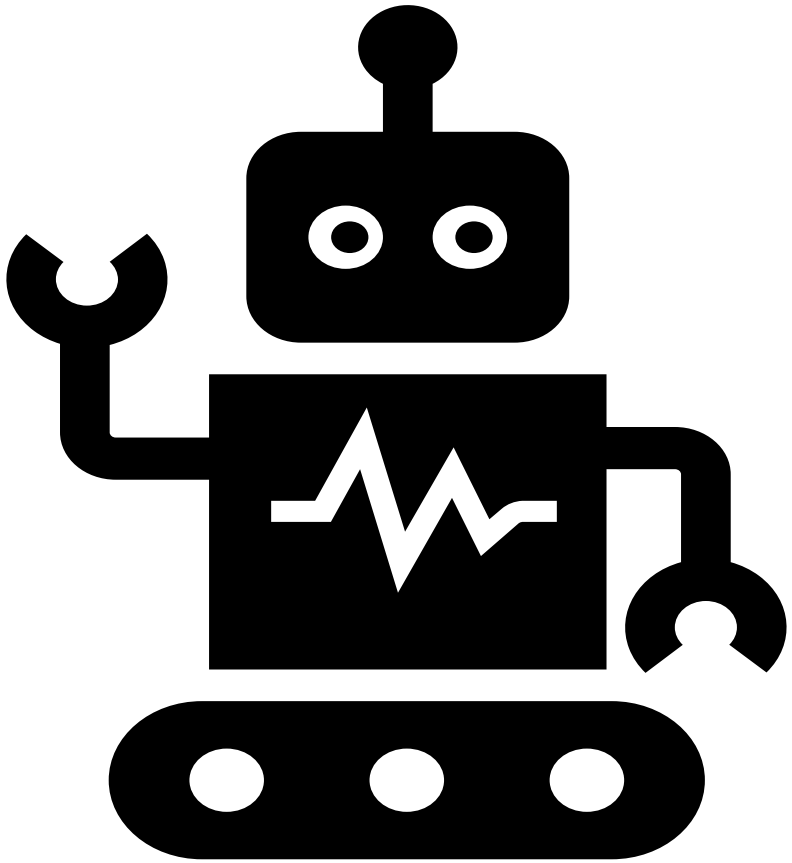
4. Basiskonzepte – Element Selectors



Selectors für Shadow DOM Elemente

```
<article>
  <div>Nicht Shadow Bereicht - light</div>
  <div slot='myslot'>Shadow Beispiel</div>
  #shadow-root
    <div class='in-the-shadow'>
      <span class='content'>
        #shadow-root
          <li id='target'>Tief im Schatten versteckt</li>
      </span>
    </div>
    <slot name='myslot'></slot>
</article>
```

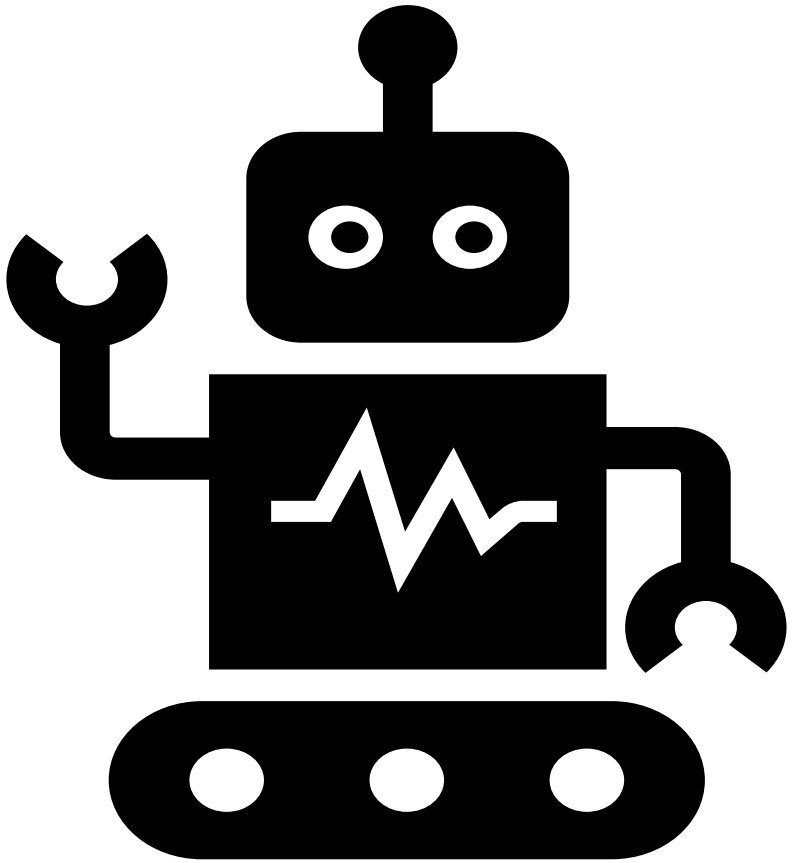
4. Basiskonzepte – Element Selectors



Selectors für Shadow DOM Elemente

```
<article>
  <div>Nicht Shadow Bereich - light</div>
  <div slot='myslot'>Shadow Beispiel</div>
  #shadow-root
    <div class='in-the-shadow'>
      <span class='content'>
        #shadow-root
          <li id='target'>Tief im Schatten versteckt</li>
      </span>
    </div>
    <slot name='myslot'></slot>
</article>
```

4. Basiskonzepte – Element Selectors

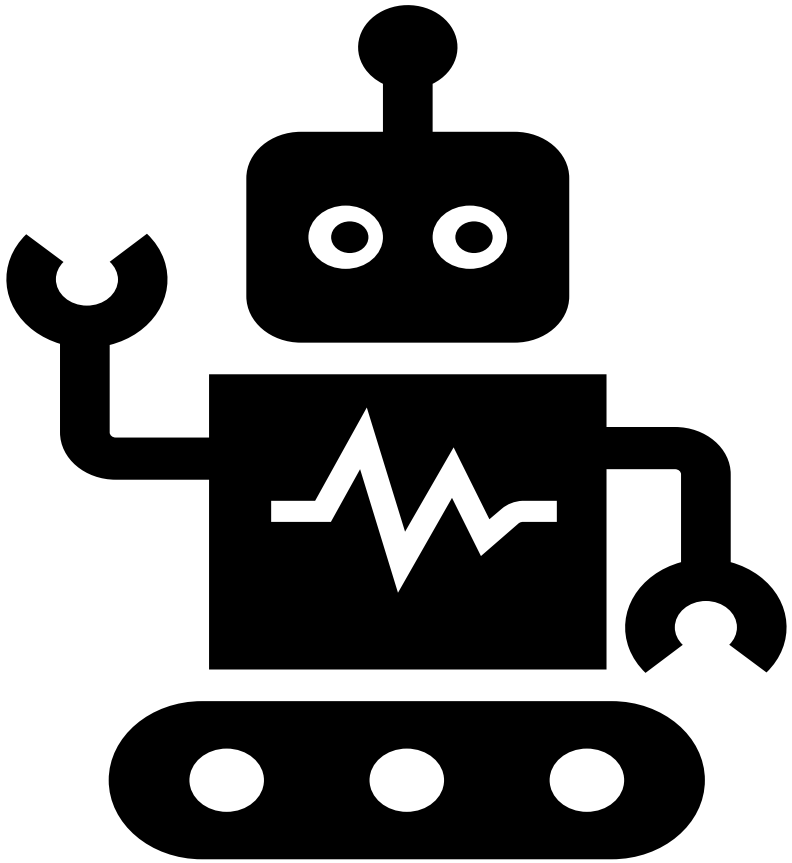


Selectors für Shadow DOM Elemente

<**div**> Nicht Shadow Bereich - light </**div**>

- **article div**
- **:light(article div)**

4. Basiskonzepte – Element Selectors

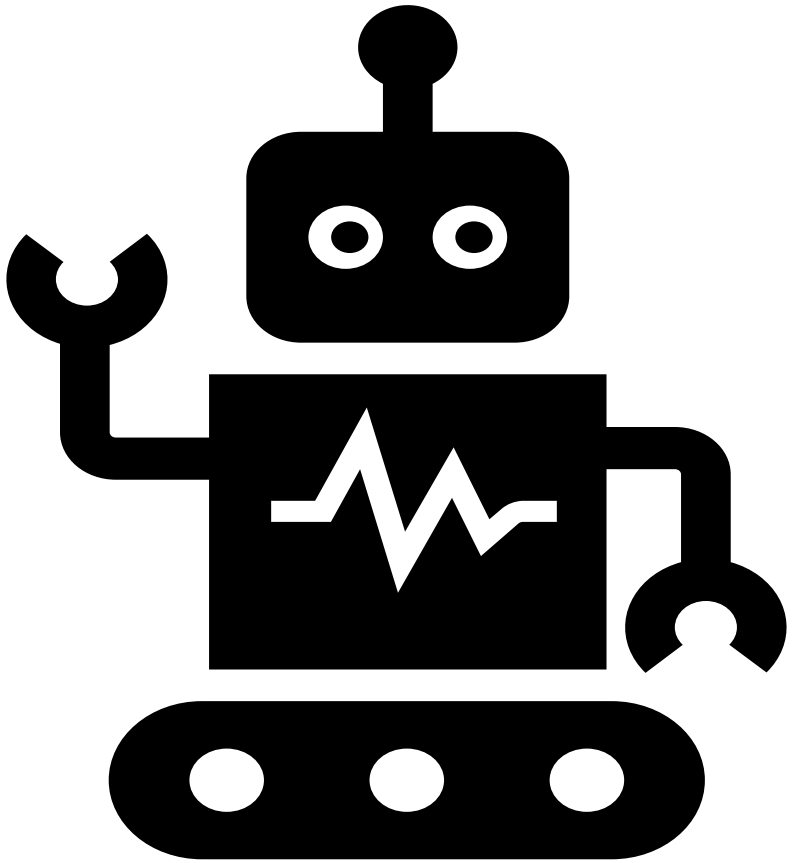


Selectors für Shadow DOM Elemente

<**li** id='target'>Tief im Schatten versteckt</**li**>

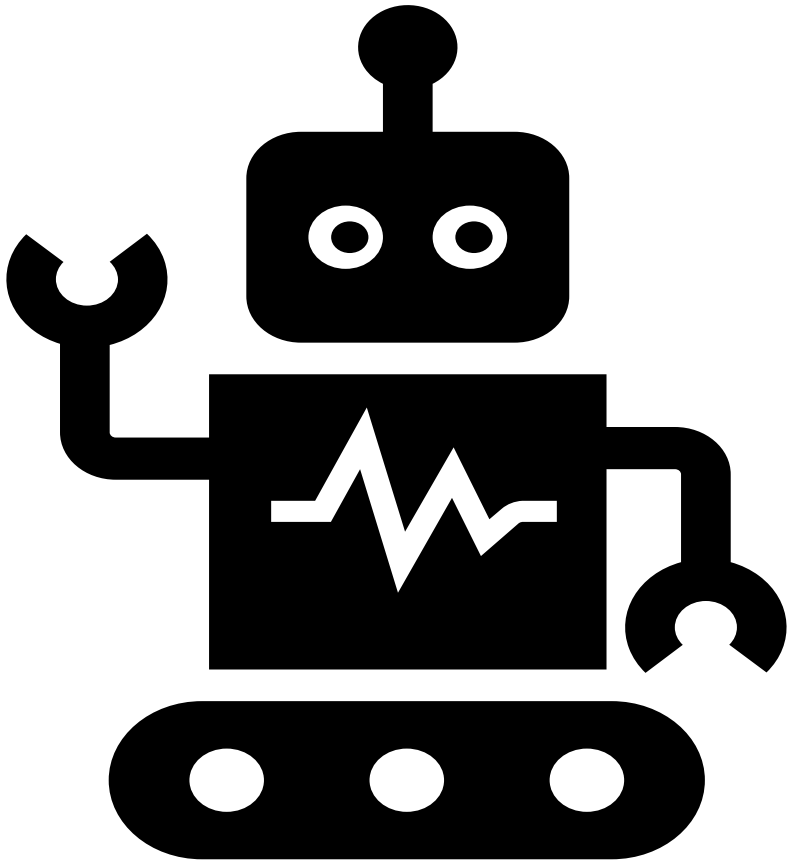
- **article li#target**

4. Basiskonzepte – Auto-waiting



- page.click(selector[, options])
- page.fill(selector, value[, options])

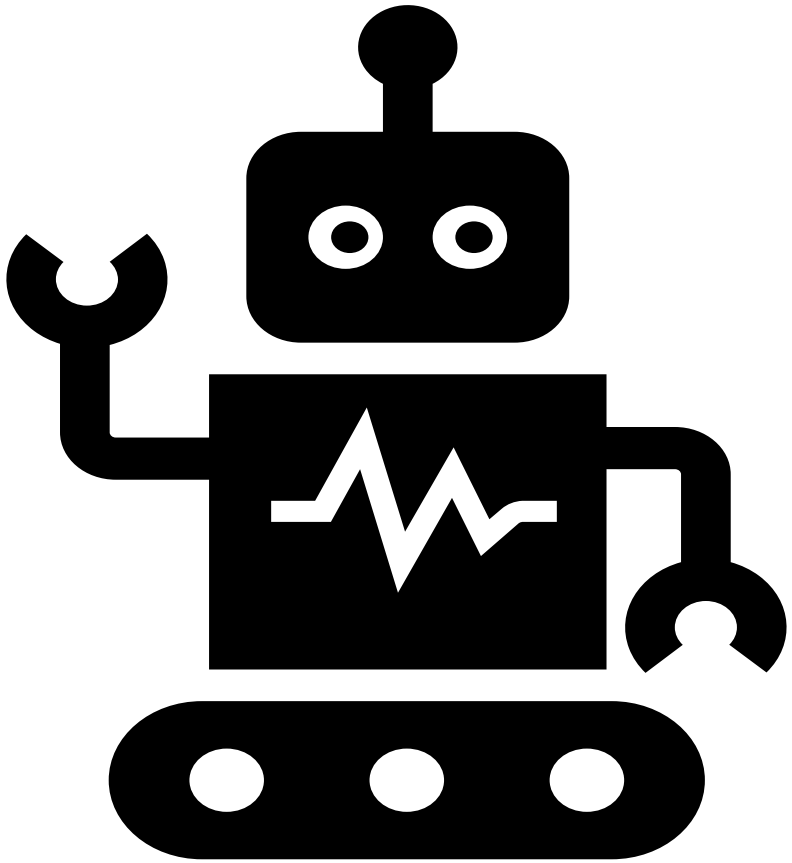
4. Basiskonzepte – Auto-waiting



page.click(selector[, options])

- in DOM
- sichtbar
- aktiv
- bewegt sich nicht
- nicht verdeckt durch die anderen Elemente

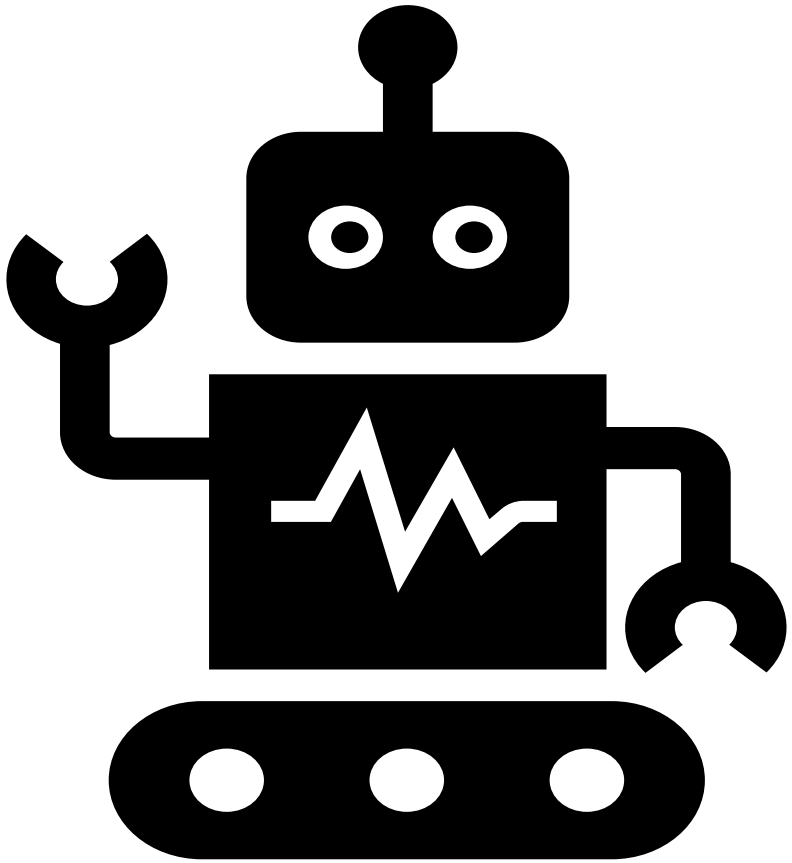
4. Basiskonzepte – Auto-waiting



page.fill(selector, value[, options])

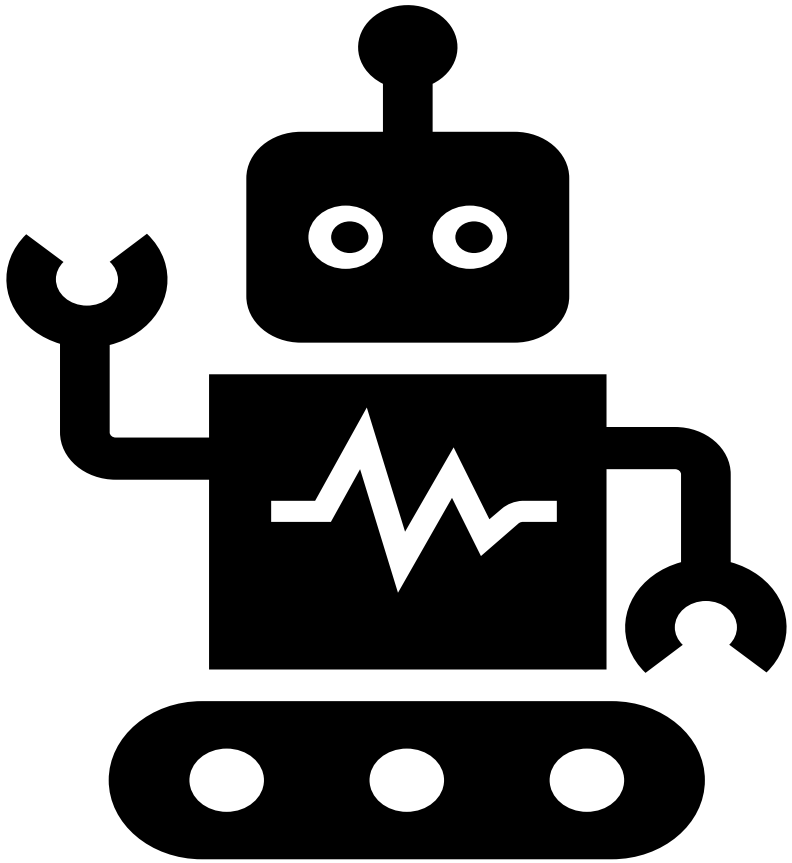
- in DOM
- sichtbar
- aktiv
- nicht verdeckt durch die anderen Elemente
- editierbar

4. Basiskonzepte – Execution Context



- Playwright Scripts – Playwright Umgebung
- Page Scripts – Browser Page Umgebung
- page.evaluate(pageFunction[, arg])

4. Basiskonzepte – Execution Context

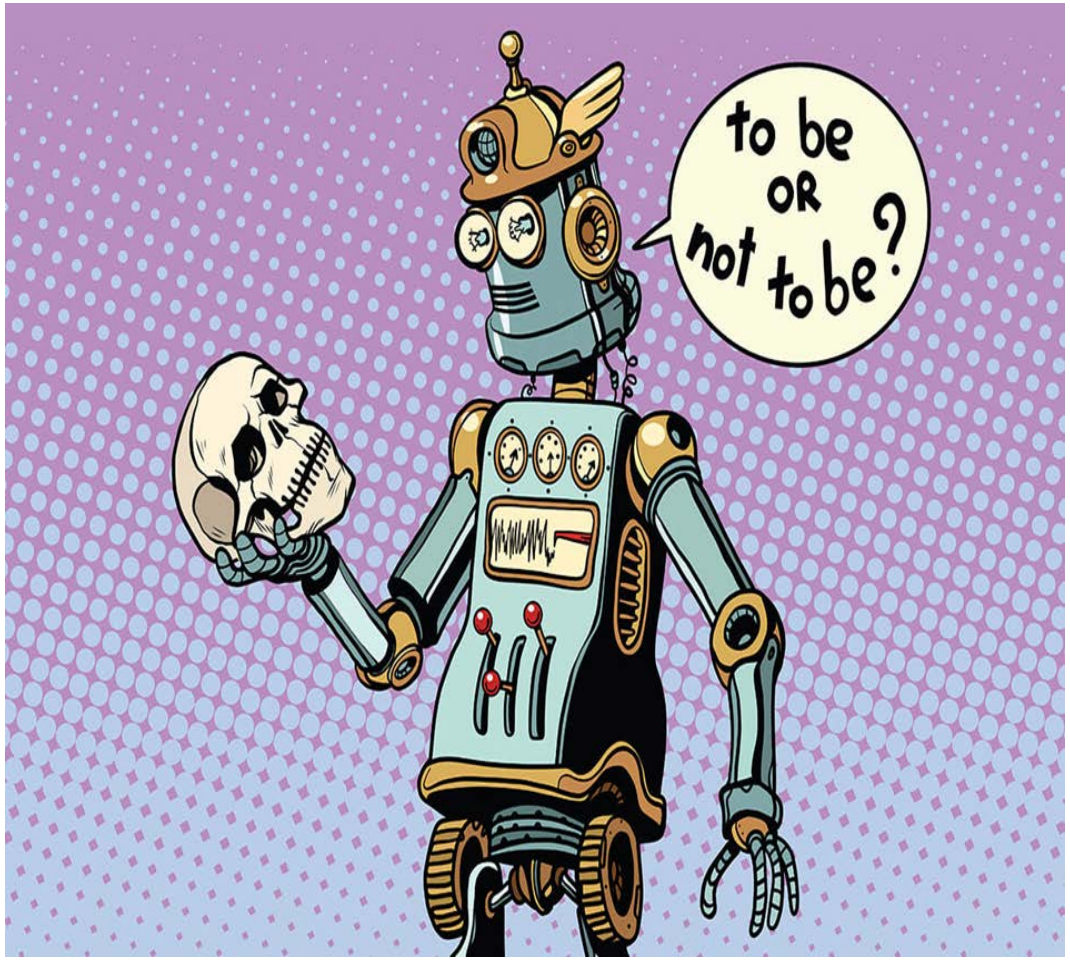


```
const data = { text: 'some data', value: 1 };
  / / Pass |data| as a parameter.
const result = await page.evaluate(data => {
   window.myApp.use(data);
}, data);
```

Agenda

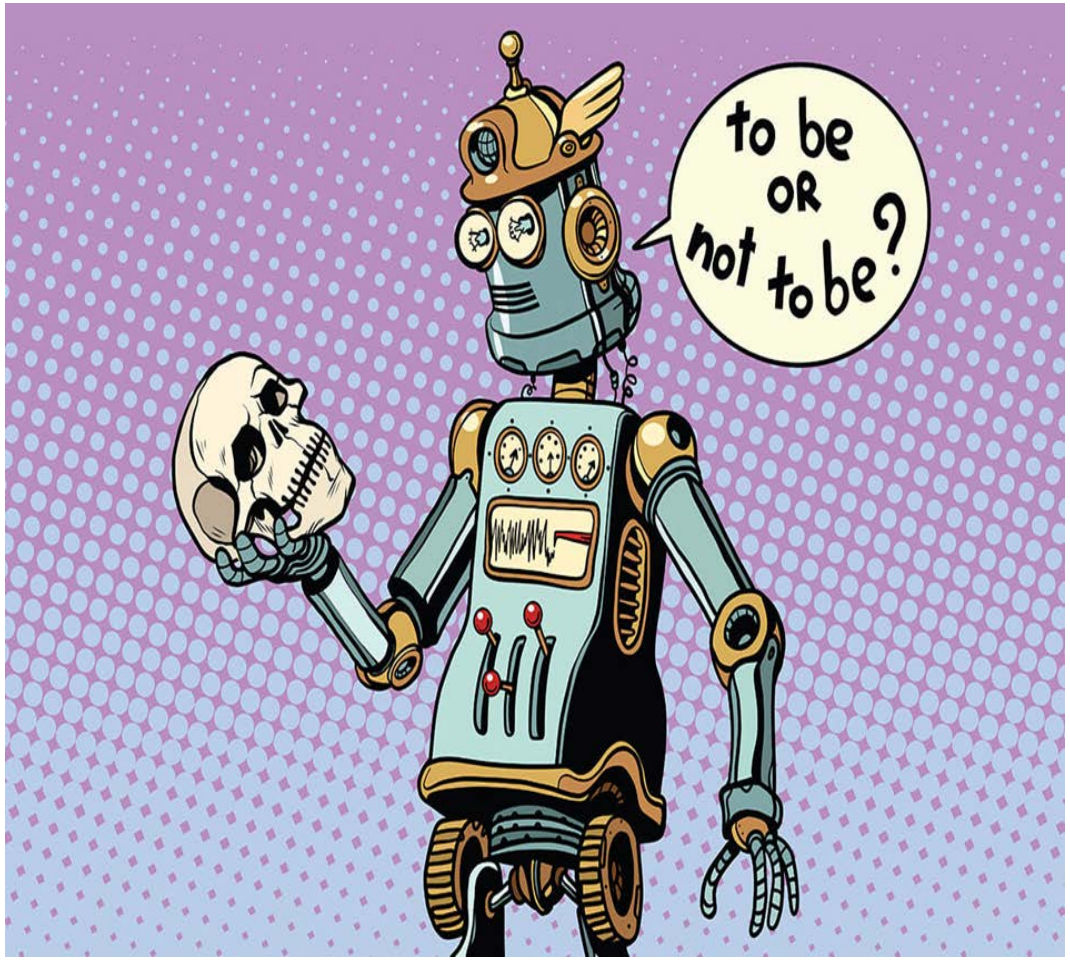
1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. **Authentifizierung**
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

5. Authentifizierung - Login - github



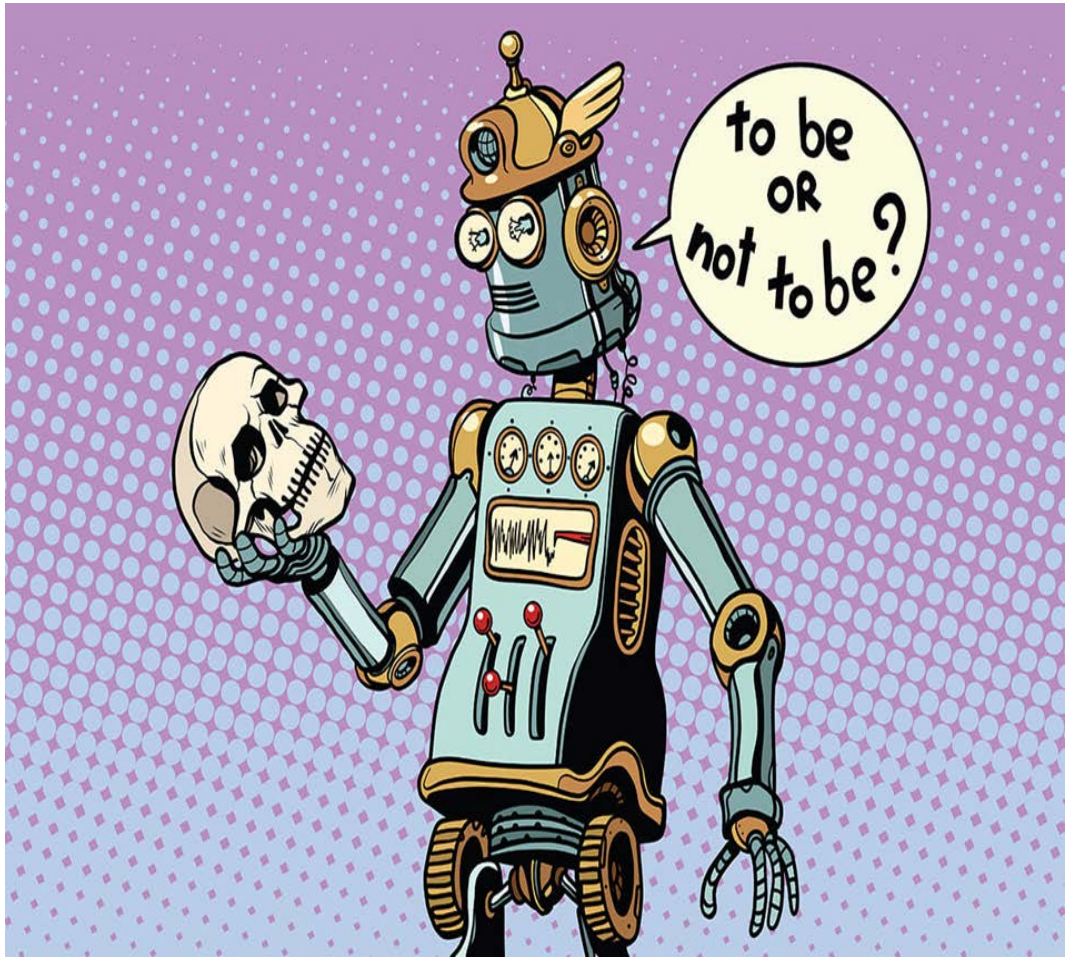

```
const page = await context.newPage();
await page.goto('https://github.com/login');

await page.click('text=Login');
await page.fill('input[name="login"]', USERNAME);
await page.fill('input[name="password"]', PASSWORD);
await page.click('text=Submit');
```

5. Authentifizierung - Login - Beispielanwendung



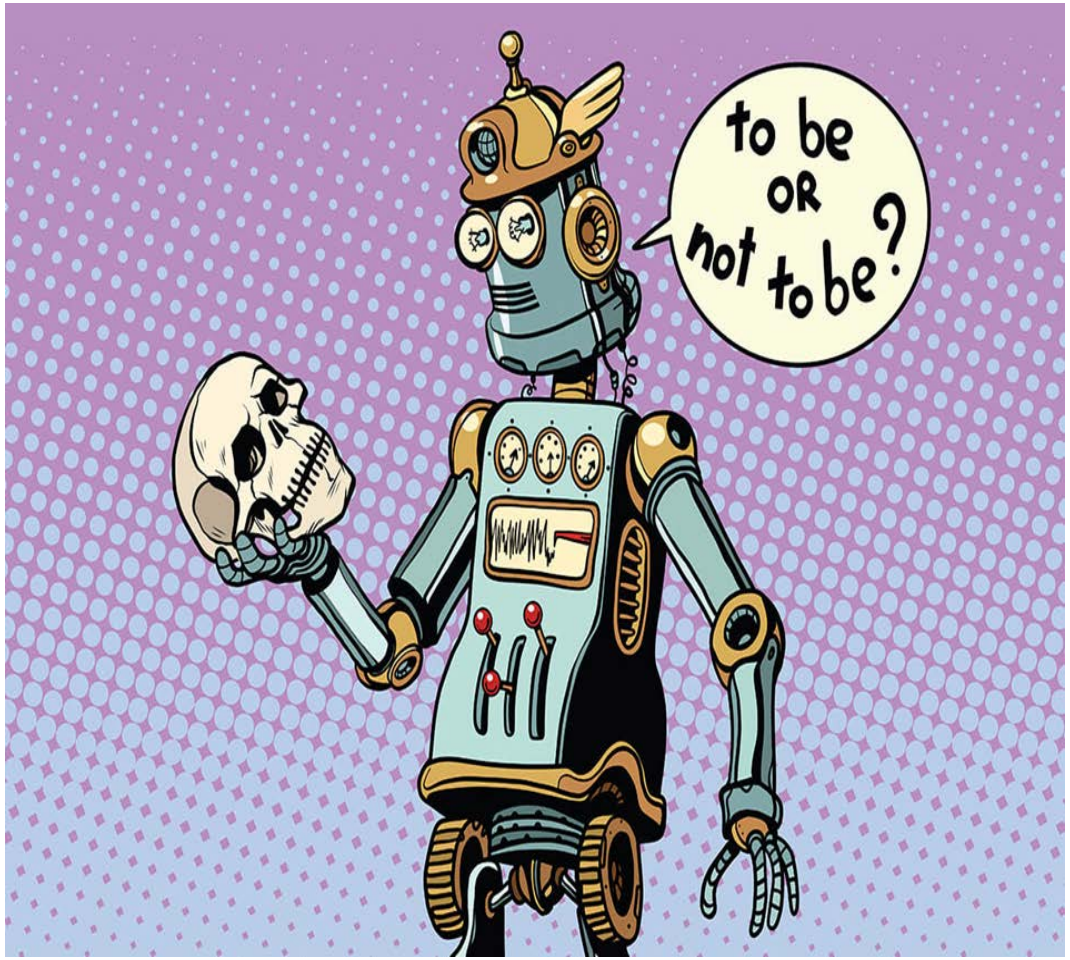

```
expect(await page.title()).toBe(desktopProps.desktopTitle),
await page.waitForSelector(cssSelectors.signInUserNameField),
await page.fill(cssSelectors.signInUserNameField, userName),
await page.waitForSelector(cssSelectors.signInPasswordField),
await page.fill(cssSelectors.signInPasswordField, password),
await page.waitForSelector(cssSelectors.signInSubmitButton)
await page.click(cssSelectors.signInSubmitButton)
```

5. Authentifizierung



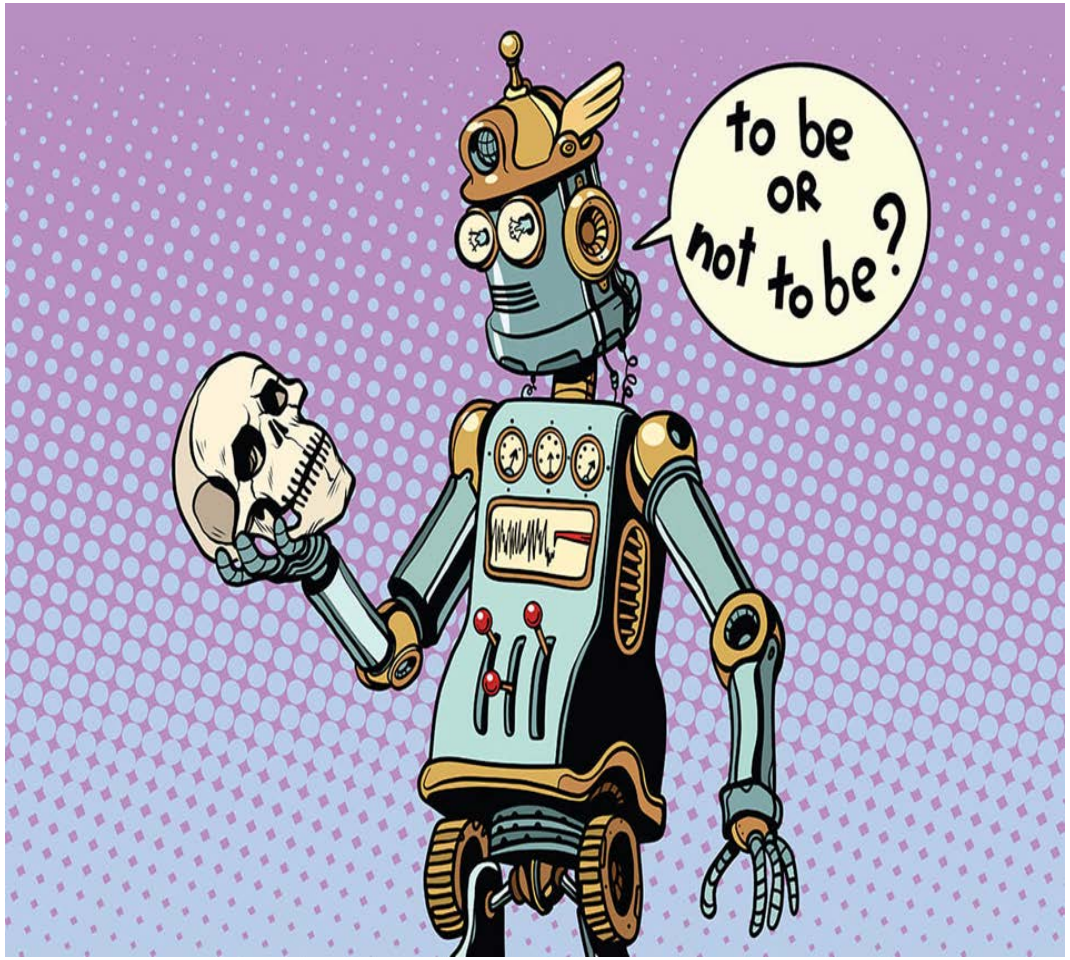
- Ist Anmeldung ein Test oder Testvorbedingung?
- Ist es notwendig Anmeldung jedesmal zu testen?
- Zeitverlust durch häufiges Anmelden

5. Authentifizierung



- Authentifizierungsstatus wiederverwenden
- Cookies
- Local Storage
- browserContext.storageState([options])
- Anmeldung + mehrere Testfälle
- beforeAll/beforeEach

5. Authentifizierung



- // Speichern des Authentifizierungszustandes in eine Environment-Variable
- const storage = await context.storageState();
- process.env.STORAGE = JSON.stringify(storage);

- // Erstellung eines neuen Testkontextes mit dem vorher gespeicherten Zustand
- const storageState = JSON.parse(process.env.STORAGE);
- const context = await browser.newContext({ storageState });

Agenda

1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. **Input und Dialoge**
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

6. Input und Dialoge

- Texteingabe
- Checkboxes und Radiobuttons
- Select-Optionen
- Maus-Klick
- Tippen der Zeichen (Tastatur)
- Keys und Shortcuts
- Datei hochladen
- Elemente im Fokus

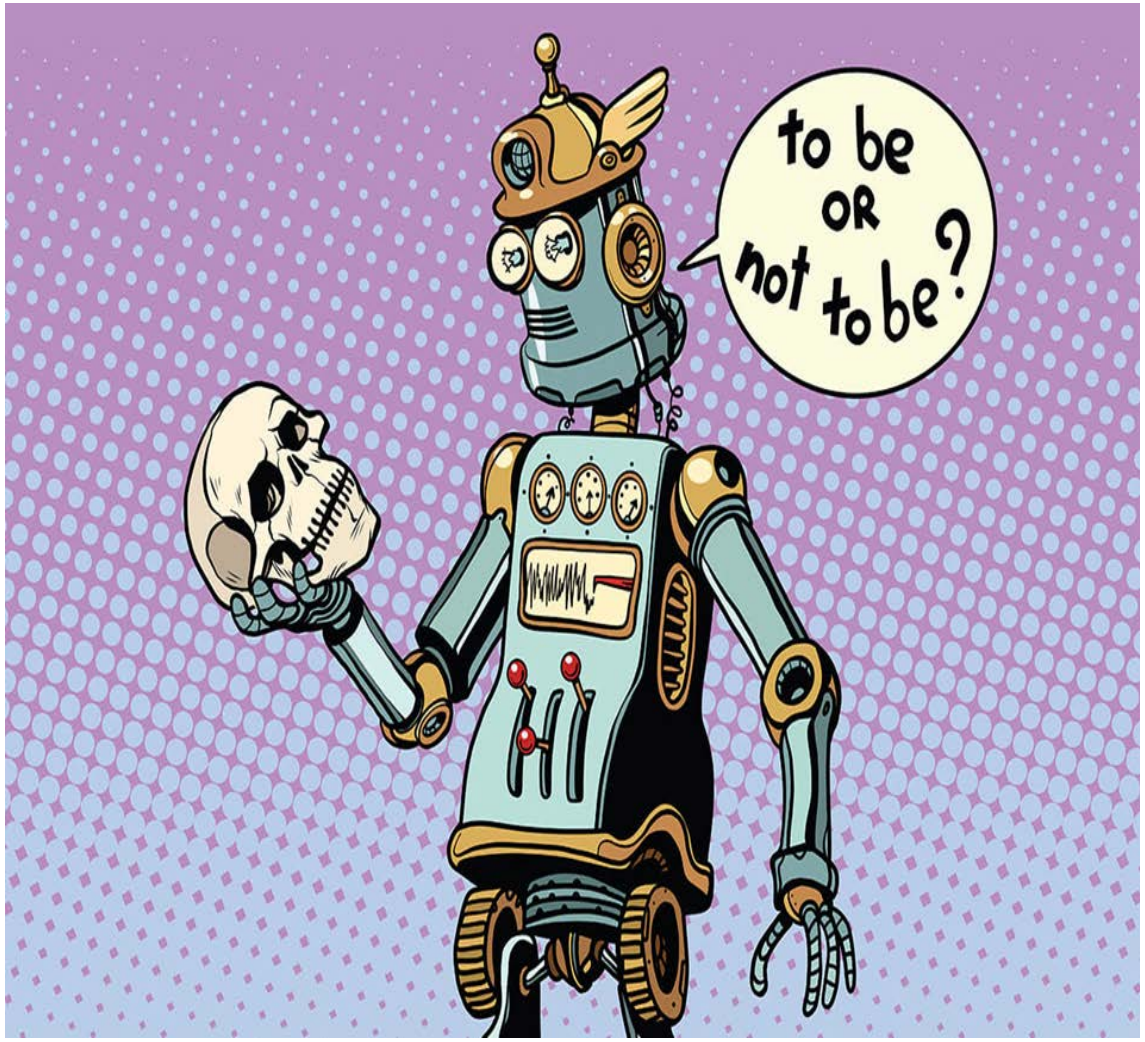
6. Input und Dialoge

- alert()
- confirm()
- prompt()
- der Dialog vor dem Verlassen der Seite

Agenda

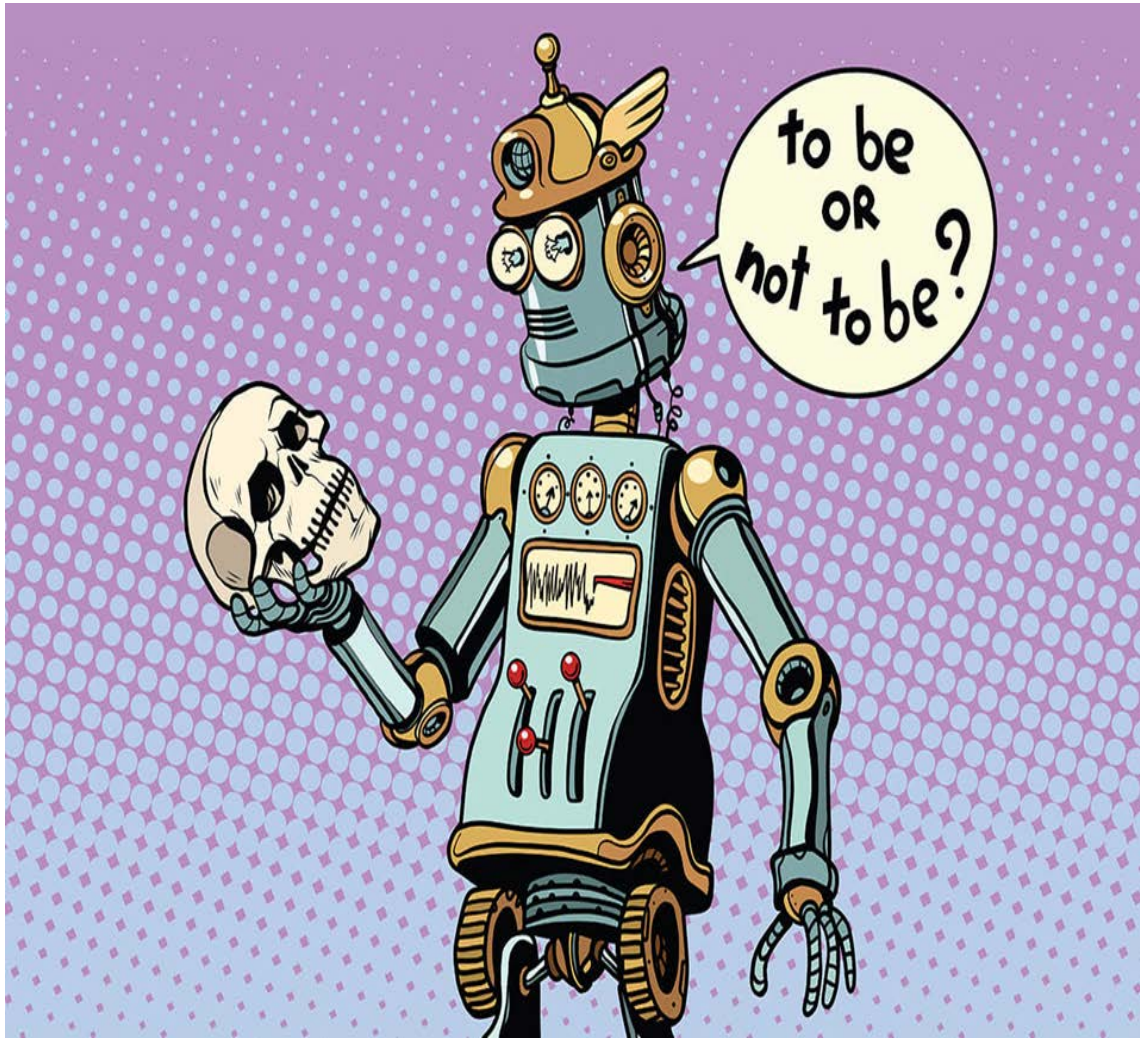
1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. **Assertions**
8. Screenshots und Videos
9. Hardware Emulation
10. Erfahrungsbericht & Demo

7. Assertions



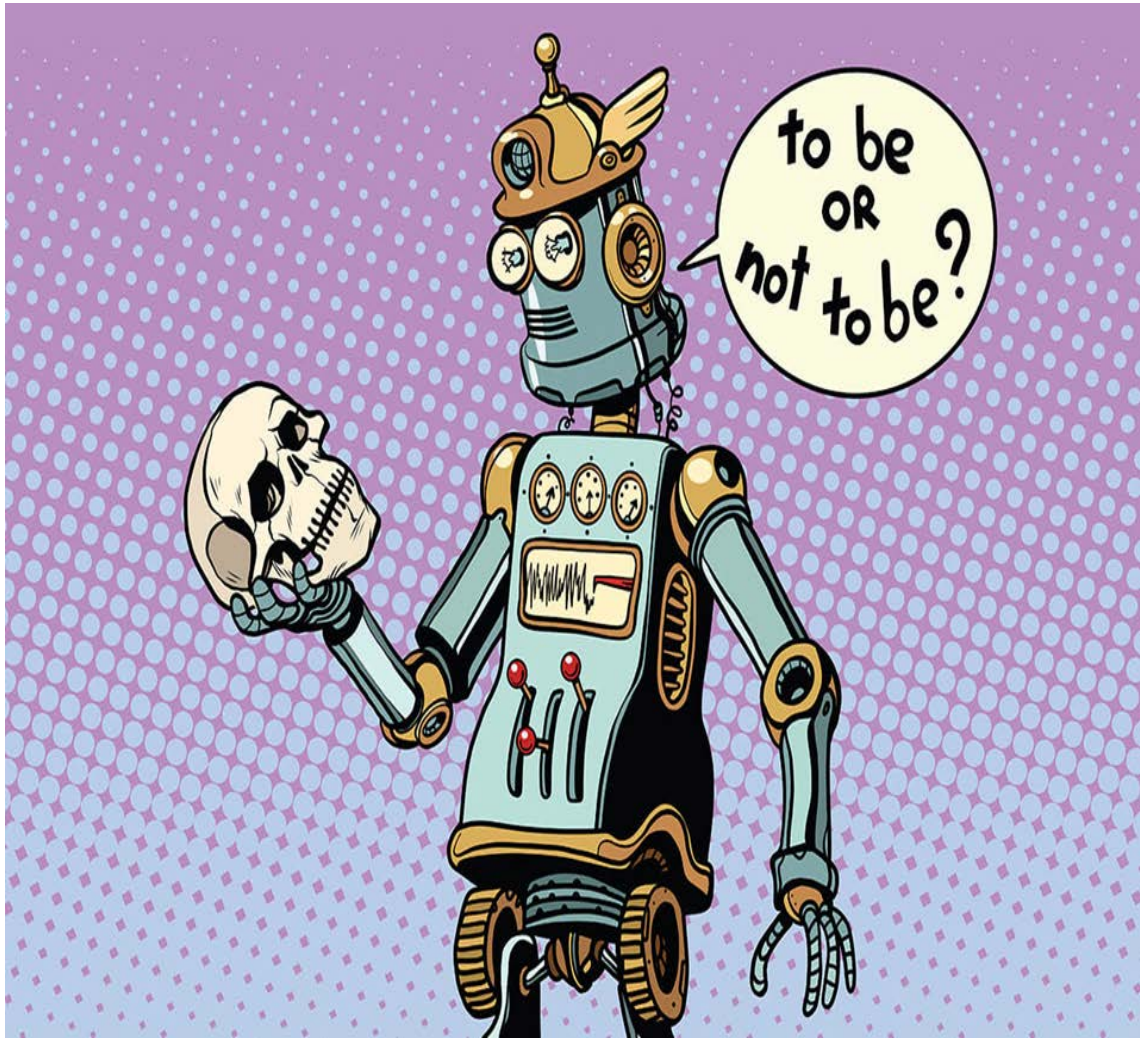
- Text
- Inner text
- Attribut-Werte
- Checkbox-Zustand
- JS – Ausdruck

7. Assertions



- Inner HTML
- Sichbarkeit eines Elements (Visibility)
- Aktiver Zustand (Enabled state)
- Custom assertions

7. Assertions



Custom assertions

```
// Assert value for input element
await page.waitForSelector('#search');
const value = await page.$eval('#search', el =>
    el.value);
expect(value === 'query').toBeTruthy();
```

Agenda

1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. **Screenshots und Videos**
9. Hardware Emulation
10. Erfahrungsbericht & Demo

8. Screenshots und Videos

Screenshot von der ganzen Seite

```
await page.screenshot({
path: 'screenshot.png', fullPage: true
});
```

8. Screenshots und Videos

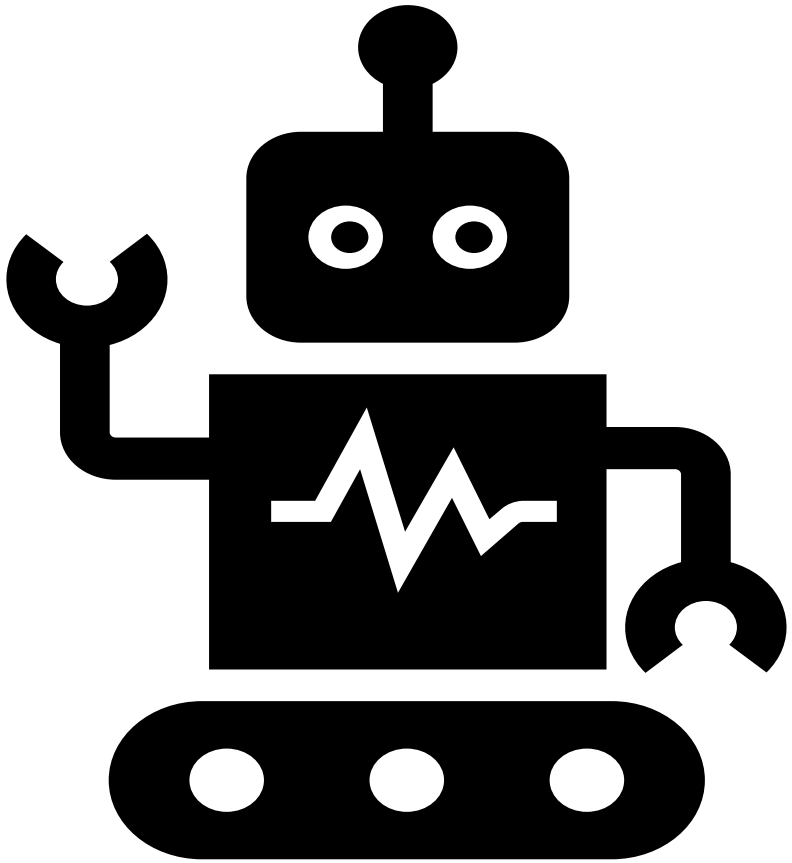
Screenshot von einem Element

```
const elementHandle = await page.$('.header');
await elementHandle.screenshot({
path: 'screenshot.png'
});
```

Agenda

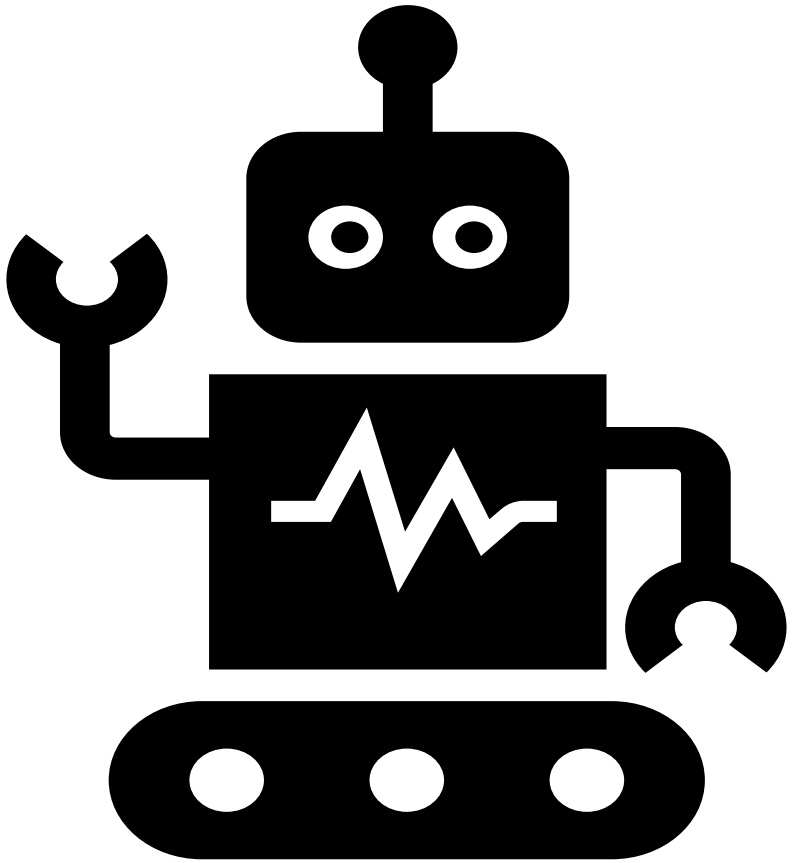
1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. **Hardware Emulation**
10. Erfahrungsbericht & Demo

9. Hardware Emulation



- Keine Unterstützung für physikalische Hardware
- Hardware Emulation

9. Hardware Emulation



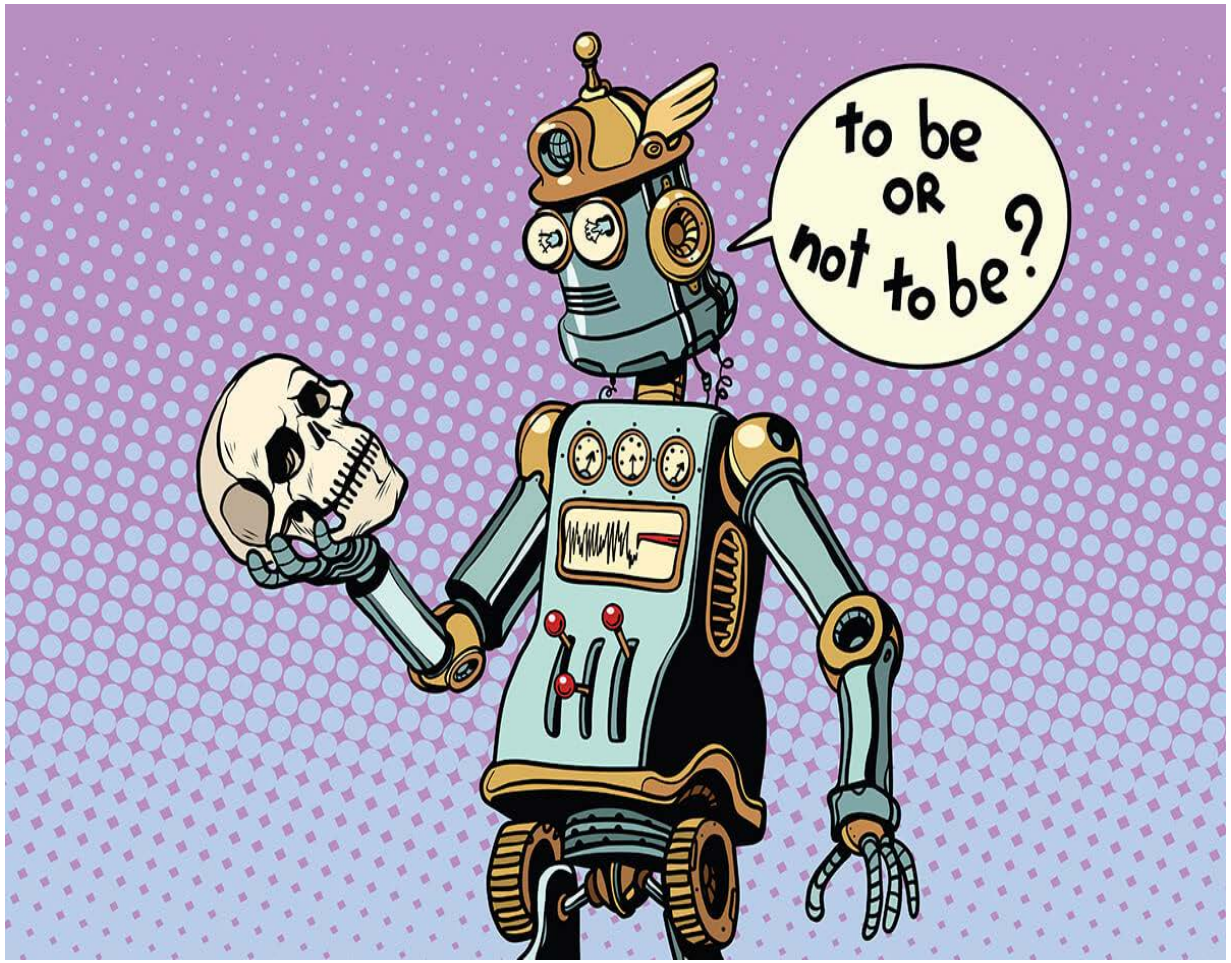
```
const { webkit, devices } = require('playwright');
const iPhone = devices['iPhone 6'];

(async () => {
  const browser = await webkit.launch();
  const context = await browser.newContext({
    ...iPhone
  });
  const page = await context.newPage();
  await page.goto('https://time.is/de/Paris');
  await browser.close();
})();
```

Warum diese Seite als Beispiel genommen wird, wird später erläutert

Agenda

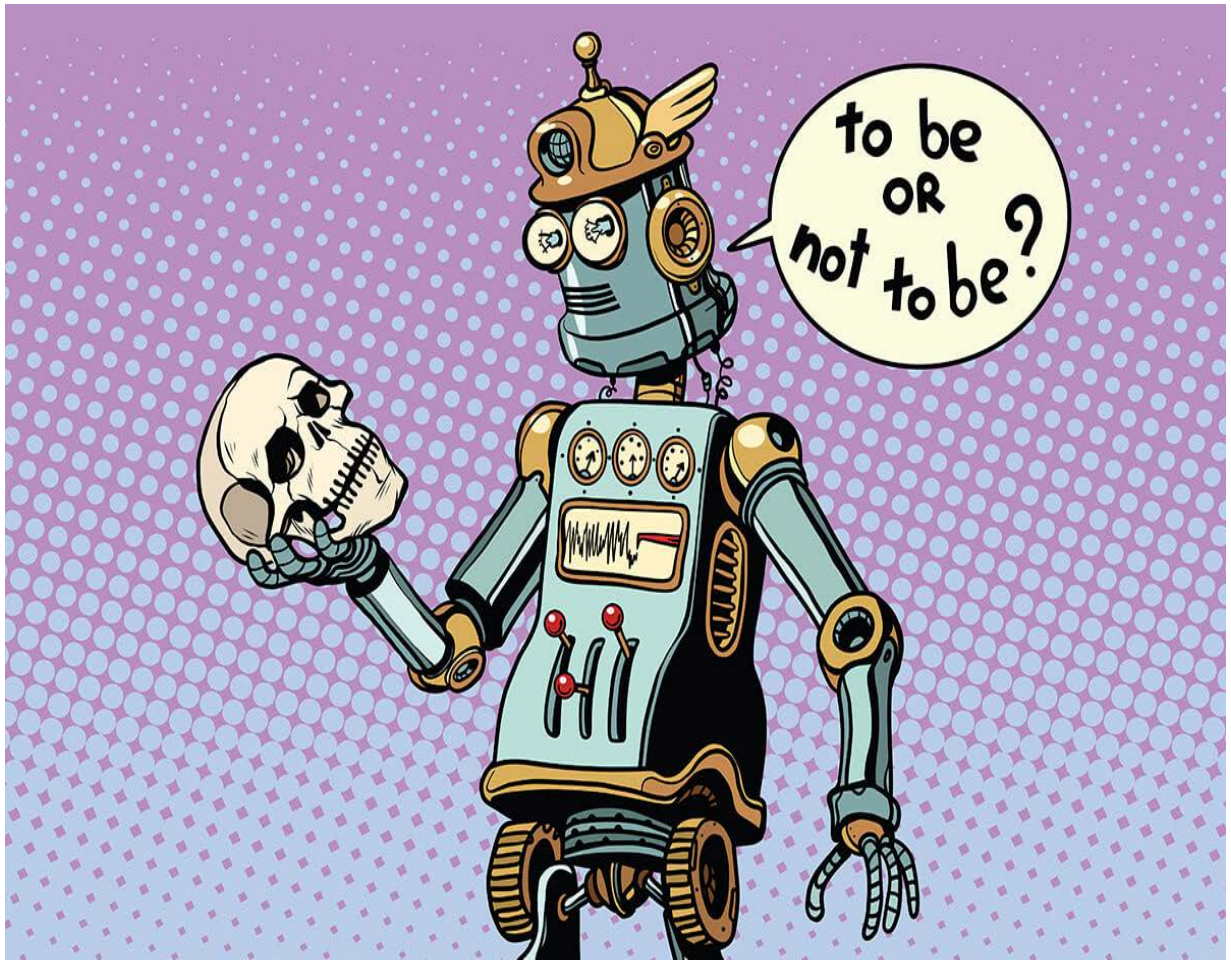
1. Neue Frameworks und Tools für das UI-Testing
2. Playwright - Überblick
3. Installation
4. Basiskonzepte
5. Authentifizierung
6. Input und Dialoge
7. Assertions
8. Screenshots und Videos
9. Hardware Emulation
10. **Erfahrungsbericht & Demo**

10. Erfahrungsbericht und Demo



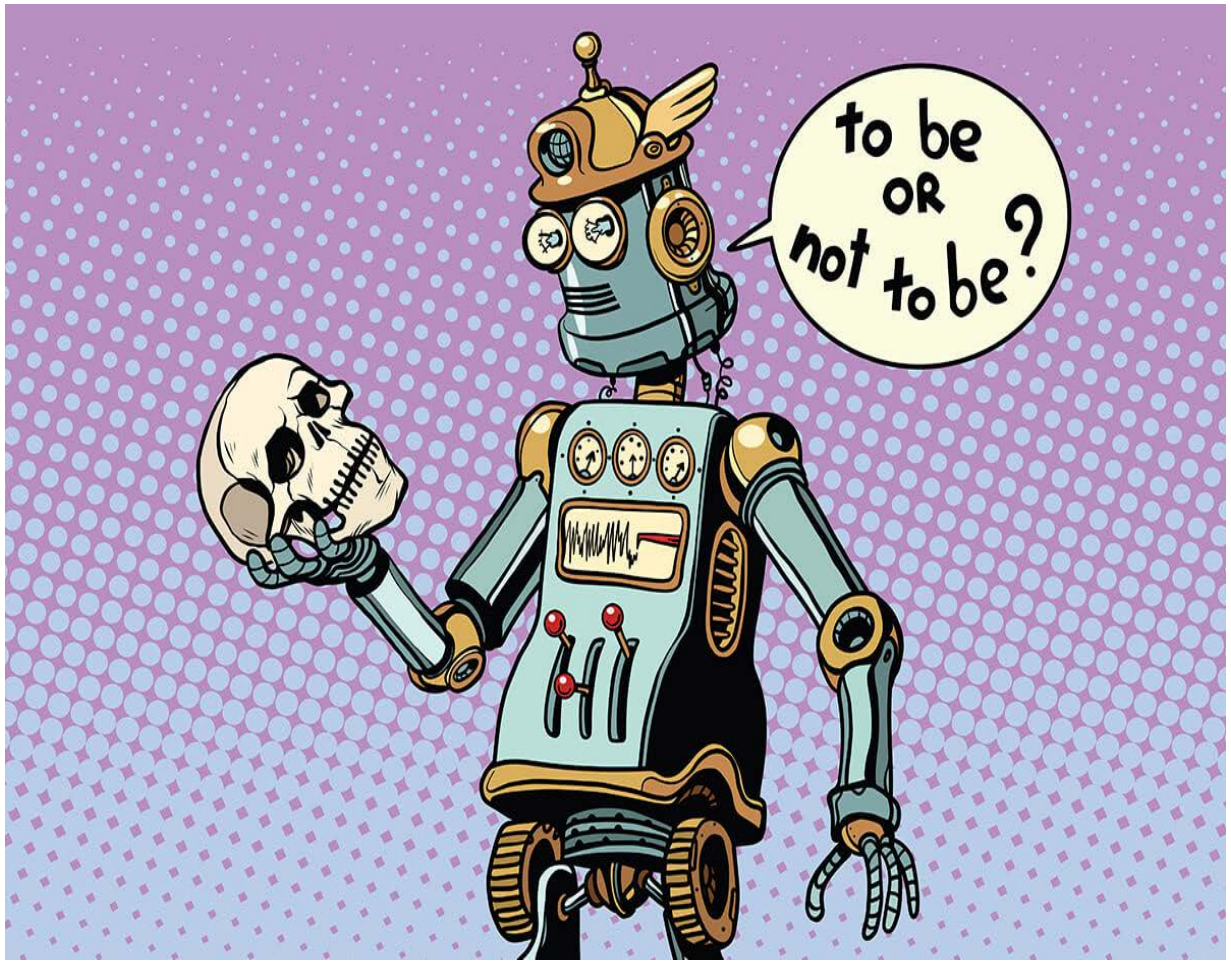
- Asynchrone Navigation
- auto-wait für Elemente
- Debugging Tools
- Möglichkeit, die Aktionen im Slow-Motion-Modus laufen zu lassen
- Problemlösungen für komplexere Ausführungsszenarien

10. Erfahrungsbericht und Demo



- Stubbing und Mocking von Netzwerkanfragen
- Testszenarien über mehrere Seiten, Domänen und iFrames
- Möglichkeit, mit Shadow-Selektoren zu arbeiten
- Möglichkeit, Upload und Download von Dateien zu testen
- APIs, um Netzwerk-Datenverkehr zu beobachten
- XHR und Fetch-Anfragen können getrackt werden

10. Erfahrungsbericht und Demo



- Keine Unterstützung von physikalischen Geräten
- Maximaze Window durch **page.viewportSize()**
- Anbindung an BrowserStack?

Quellen

- https://playwright.dev/
- https://www.martinmcgee.dev/starting-microsoft-playwright/
- https://github.com/microsoft/playwright
- https://www.browserstack.com/guide/playwright-vs-selenium
- https://playwrightsharp.dev/documentation/index.html
- https://dev.to/davert/5-reasons-you-should-not-use-protractor-in-2019-3l4b/comments

Seeing beyond