# Django Basics

|  |  |  |
| --- | --- | --- |
| Topic | Command | Info |
| Define Varlaibles | Let varaiable = 0 |  |
| Query DOM | Document.querySelector(“h1”),innerHTML = “blabla” | QuerySelector looks for the h1 Element in HTML and change the innerHTML property to “Goodbye”. You can also assign the query selector to a variable: const heading = document.querySelectr(“h1”). Const means it wont change, while let knows it can/will change.  There also exists querySelectorAll which returns a list of arrays |
| Strict equality | A === B | Strict equality means same value and same type |
| Printf in JS | Alert(`hello ${name}`) | Printf(“hello {name}”). Here we use backticks `` as indicator for formatted string |
| DOM manip. | Document.querySelector(“button”).onlick = myfunc | Assign functions to variables aswell. Note here you send the function itself and not call it like myfunc()! |
| DomContent Load | Document.addEventListener(‘DOMContentLoaded’, func(){do something}) | Look at console which you can see from inspect. In JS the script is read from top to bottom, therefore querySelector might not find variables if they are assigned later in the script. For that we can call our functions after the DOM has finished loading. |
| querySelector | Document.querySelector(“tag”) or (“#id”) or (“.class”) | You can use the same syntax you use in CSS for querySelector |
| Data in HTML | <button data-color=”red”>Red </button> | In HTML you can save send data via data-name = a. This data can be accessed in CSS via button.dataset.color in this case 🡪 conventions: use data-name in HTML and datasets.name in JS |
| Forms in JS | Document.queryselector(‘form”).onsubmit = function() {do something Return False} | Return in JS makes sure the form information is not passed in as a POST method |
| For Loop | Document.querySelector(“test”).foreach(function(a){}) |  |
| Function notation | (input) => { do something } | Javascript has introduced the arrow notation for functions. This is the same as function(input) {do something}. Note that when you have a input you don’t necessarily need () |
| Select Dropdown | Doc.querySelector(“select”).onchange = func |  |
| This |  | Refers to the thing that received the event, for example a select dropdown |
| Events in JS | .onclick, .onmouseover, .onkeydown, … | There are a lot of events that can be changed using JS |
| Add new Elem in Elem | Document.querySelector(“ul”).append(li) | Adds a new HTML element (here li ) inside the selected element (here ul) |
| Intervals | SetInterval(somefunction, 1000) | Calls the function every x miliseconds |
| Localstorage | localStorage.getItem(key)  localStorage.setitem(key, value) | Localstorage is used to save session information in the web browser such that it can continue if we reopen the browser |
| API | Fetch(url).  then( response => response.json()  .then(data => { do something} 🡪 asynchronous process | API’s are a well defined- structured way using JSON that is used by services on the internet to communicate to each other, where JSON is a dictionarylike syntax. Fetch send an HTTP request to your url and **then** once the response is there interpet it as a JSON data and **then** do something with this data |
| History API | History.pushState({key: value},”Title-ignore”,”section${section}”)  Window.onpopstate = function(event){event.state.section) | Use history API to remember where on the page we are on one-page style webpages. The first element is used to store key-value pairs in the history, the second one is title which is ignored by most browsers and 3rd one is the url name that shows up 🡪 in this case url/section1 i.e.  Back in history is also available with the event handler with the window.onpopstate |
| Window | Window.innerWidth/Height  Window.scrollY  Window.onscroll | Window.scrollY 🡪 how far has been scrolled.  Document.body.offsetHeight 🡪 height of document  Window.onscroll is an event that listens to scrolling  Infinte scroll implementation windown.onscroll () => {  If (window.innerWidth + window.scrolly >= Document.body.offsetHeight): do something} |
| Animation + CSS | @keyframes animname {  0%{do sth.}  50% {do sth else}  100%{ do sth again}} +  H1 {  Animation-name: animname …} | 1st define the animation name in your header tag or similar as well as basic properties like name, duration and fill mode (usually forwards) play-state, 2nd write what the animation should do at specific points during execution |
| Animation + JS | H1.style.animationPlayState = ‘paused’/’running’ |  |
| Events in JS | Document.addEventListener(‘click’, event => {}) | This event listener listens for click events and takes in the optional event parameter. Most events have access to it. From here we can see what the target of the click was (was was clicked) with event.target. To remove it or its parent element we can use element.parentElement.remove() |
| React | Include scripts React, ReactDOM, Babel (convert JSX to JS) | React is a declarative language with htm-like syntax similar to Django/jinja2. React essentially has components which change based on an underlying state using logic. When adding React code to html in the script tags add type = “text/babel” which translates JSX to JS which the browser can understand |
| React Component | Class App extends React.Component {  render() {  return do something}; }  ReactDOM.render(<App />, doc.querySelector(#app)) | React components are organized into a JS class, like in python, meaning that the JS classes represent react components. Every React component needs a render function, which describes what to render to the user. After defining the React compoenent, insert it into HTML using the ReactDOM |
| React Props | In class: Hello, {this.props.propname} +  when calling: <ReactCompt propname=”name”/> | Props are used to pass in arguments/values to react in order to parameterize the components |
| React States | constructor(props) {  super(props);  this.state = {  key: value  };  }  In render: i.e. <h1>Hello {this.state.name} </h1>  Set state: this.setState(state => ({  count: state.count +1 })); | States are defined in React in the constructor which sets the component initial state. The super(props); line makes sure that setup is done correctly or sth, ignore. In the state section we define our initial state as key-value pairs and we can call those state values in the return function like in jinja2 using {}  This.setState updates the state in JS, who’s input is its old state |
|  |  |  |
| More Events | onKeyPress 🡪 this.state.key  onChange 🡪 | Event handles in React 🡪 Note different notation (capitalized word beginning). Also remember eventhandlers also have access to the event input |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |