## Belajar Javascript Dasar

**3.1 Valuriabel**

A black rectangular object with white lines

Description automatically generated

|  |  |
| --- | --- |
| console**.**log("------ Variables ------")**;**  *// Var*  **for** (var i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  console**.**log(i)**;**  *// Var*  function looping() {  **for** (var i **=** 0**;** i **<** 5**;** i**++**) {          console**.**log(i)**;**      }  }  looping()**;**  console**.**log(i)**;**  *// Let*  **for** (let i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  console**.**log(i)**;**  *// Let*  **for** (let i **=** 0**;** i **<** 5**;** i**++**) {      console**.**log(i)**;**  }  **if** (true) {      let i **=** 100**;**      console**.**log("i ="**,**i)**;**  }  *// Const*  const phi **=** 3.14**;**  *//phi = 3.147;*  console**.**log(phi)**;** | 1. **Menggunakan var dalam Loop Pertama:**   Sebuah loop for menggunakan variabel i dengan kata kunci var untuk menginisialisasi, mengupdate, dan mengakses nilai i.  Loop ini mencetak nilai i dari 0 hingga 4.  Setelah loop selesai, nilai i tetap dapat diakses di luar loop karena deklarasinya menggunakan var. Oleh karena itu, pernyataan console.log(i) di luar loop akan mencetak nilai i, yang pada saat itu bernilai 5.   1. **Menggunakan var dalam Fungsi looping:**   Sebuah fungsi bernama looping juga menggunakan variabel i dengan kata kunci var untuk melakukan loop dan mencetak nilai i dari 0 hingga 4.  Meskipun variabel i dalam fungsi looping memiliki cakupan (scope) yang berbeda dari variabel i di luar fungsi, keduanya tidak saling mempengaruhi. Variabel i dalam fungsi looping bersifat lokal untuk fungsi tersebut.  Setelah fungsi looping selesai dijalankan, pernyataan console.log(i) di luar fungsi akan mencetak nilai i yang ada di luar fungsi, yaitu 5.   1. **Menggunakan let dalam Loop Kedua:**   Sebuah loop for menggunakan variabel i dengan kata kunci let. Variabel i yang dideklarasikan dengan let memiliki cakupan yang lebih terbatas dibandingkan dengan var.  Loop ini mencetak nilai i dari 0 hingga 4.  Setelah loop selesai, pernyataan console.log(i) di luar loop akan menghasilkan ReferenceError karena variabel i dideklarasikan dengan let dan hanya memiliki cakupan dalam blok loop.   1. **Menggunakan let dalam Loop Ketiga:**   Ini adalah loop terpisah yang juga menggunakan variabel i dengan kata kunci let. Variabel i ini berbeda dari variabel i yang digunakan dalam loop sebelumnya.  Loop ini mencetak nilai i dari 0 hingga 4.  Variabel i dalam loop ini juga hanya memiliki cakupan dalam blok loop, sehingga pernyataan console.log(i) di luar loop akan menghasilkan ReferenceError jika dijalankan.   1. **Menggunakan let dalam Blok If:**   Di dalam blok if, ada deklarasi variabel i menggunakan let, dan nilainya diatur menjadi 100.  Variabel i ini hanya memiliki cakupan dalam blok if. Oleh karena itu, pernyataan console.log("i =", i) dalam blok if akan mencetak nilai 100, tetapi variabel i di luar blok if tidak terpengaruh dan tetap tidak terdefinisi.   1. **Menggunakan const untuk Variabel phi:**   Variabel phi dideklarasikan dengan kata kunci const dan diinisialisasi dengan nilai 3.14.  Karena variabel phi dideklarasikan dengan const, nilainya tidak dapat diubah setelah inisialisasi. Jika Anda mencoba untuk mengubah nilai phi dengan pernyataan phi = 3.147;, itu akan menghasilkan error (TypeError). |

## 3.2 Class

console**.**log("------ Class ------")**;**

class Car {

    constructor(**name**) {

*this***.***brand* **=** name**;**

    }

    present() {

**return** "I have a " **+** *this***.***brand***;**

    }

}

let myCar **=** **new** Car("Ford")**;**

console**.**log(myCar**.**present())**;**

1. Deklarasi Kelas Car:

Kode dimulai dengan mendeklarasikan sebuah kelas dengan nama Car menggunakan kata kunci class.

Kelas Car memiliki sebuah konstruktor (constructor) yang menerima satu parameter name. Konstruktor ini digunakan untuk menginisialisasi properti brand dari objek yang akan dibuat berdasarkan kelas ini.

Kelas Car juga memiliki metode (fungsi) bernama present yang akan mengembalikan sebuah string yang berisi teks "I have a " diikuti oleh merek mobil yang diberikan saat objek dibuat.

1. Membuat Objek myCar:

Setelah mendeklarasikan kelas Car, kode selanjutnya membuat sebuah objek bernama myCar dengan menggunakan konstruktor kelas Car.

Objek myCar dibuat dengan merek "Ford", sehingga properti brand-nya akan memiliki nilai "Ford".

1. Mengakses dan Mencetak Informasi Mobil:

Setelah objek myCar dibuat, kode kemudian mencetak informasi tentang mobil tersebut dengan menggunakan metode present yang ada dalam kelas Car.

Hasil cetakan akan menjadi "I have a Ford" karena objek myCar memiliki properti brand yang bernilai "Ford" dan metode present menggabungkannya dengan teks lain.

Kode tersebut merupakan contoh penggunaan kelas dalam JavaScript. Dalam kode tersebut, kita mendefinisikan kelas Car yang memiliki properti merek mobil (brand) dan metode present untuk mengembalikan informasi tentang mobil tersebut dalam bentuk string. Setelah mendefinisikan kelas Car, kita membuat objek myCar dengan merek "Ford" menggunakan konstruktor kelas tersebut. Kemudian, kita menggunakan metode present pada objek myCar untuk mencetak informasi "I have a Ford". Ini menggambarkan cara penggunaan kelas dalam JavaScript untuk mengorganisasi dan mengelola data dan perilaku yang terkait dengan objek tertentu, sehingga memungkinkan pengembang untuk membuat objek-objek dengan struktur dan fungsi yang serupa secara efisien.

## Inheritance

Inheritance merupakan teknik untuk pewarisan

A screenshot of a computer

Description automatically generated

console**.**log("------ Inheritance ------")**;**

class Gadget {

    constructor(**name**) {

*this***.***name* **=** name**;**

    }

    present() {

**return** "I have an " **+** *this***.***name***;**

    }

}

class Handphone *extends* *Gadget* {

    constructor(**name,** **mod**) {

*super*(name)**;**

*this***.***model* **=** mod**;**

    }

    show() {

**return** *this***.**present() **+** ", it is " **+** *this***.***model***;**

    }

}

let handphone **=** **new** Handphone("iphone"**,** "14 Pro Max")**;**

console**.**log(handphone**.**show())**;**

Kode di atas menggambarkan konsep pewarisan (inheritance) dalam pemrograman berorientasi objek menggunakan JavaScript. Dalam contoh ini, kita memiliki dua kelas, yaitu **Gadget dan Handphone**. **Kelas Gadget** memiliki **properti name** dan **metode present** yang mengembalikan informasi tentang gadget**. Kelas Handphone** adalah **turunan** dari kelas Gadget yang tambahkan properti model dan metode show untuk menampilkan informasi yang lebih spesifik tentang handphone.

Ketika objek handphone dibuat dengan merek "iphone" dan model "14 Pro Max" menggunakan konstruktor kelas Handphone, kita dapat menggunakan metode show pada objek tersebut untuk mencetak informasi lengkap tentang handphone tersebut, seperti "I have an iphone, it is 14 Pro Max". Hal ini menunjukkan bagaimana pewarisan memungkinkan kita untuk menggabungkan dan memperluas fungsionalitas dari kelas yang ada untuk menciptakan hierarki kelas yang lebih terperinci dan spesifik.

## Arrow Function

Fungsi panah (arrow function) adalah fitur dalam JavaScript yang memungkinkan Anda mendefinisikan fungsi dengan sintaks yang lebih singkat dan mudah dibaca.

## 4.1 JSX (Javascript XML)

Basic rules

|  |
| --- |
| myElement = (  <p>Learn Javascript</p>  <p>Learn React</p> ); //wrong |

Pada contoh code ditas merupakan contoh yang salah karena dalam rules JSX mestinya kode diberikan pengelompokan beberapa elemen di dalam satu element root seperti code dibawah ini .

|  |
| --- |
| // Basic Rules  myElement = (  <div>  <p>Learn Javascript</p>  <p>Learn React</p>  </div> ); // correct |

Pada contoh kedua, kelompokkan kedua elemen <p> di dalam satu elemen <div>. Ini adalah cara yang benar dalam JSX. Elemen <div> berperan sebagai elemen root utama yang mengelilingi elemen-elemen lainnya. Dengan demikian, Anda mematuhi aturan dasar JSX yang memerlukan satu elemen root.

4.2 Class Component

|  |
| --- |
| // Class Component  class Practice1 extends React.Component {  render() {  return <h3>Practice: ReactJS</h3>  }  }  let myElement = <Practice1 />; |

A screen shot of a computer program

Description automatically generated

Pada code sebelumnya belum ada code untuk merender component react kedalam DOM setelah di berikan component react dapat terender ke dalam dom.

|  |
| --- |
| ReactDOM**.**render(myElement**,** document**.**getElementById('root'))**;** |

Baris kode ini mengatakan kepada React untuk merender komponen yang disimpan dalam myElement ke dalam elemen DOM yang memiliki id "root" pada halaman web. Dengan demikian, setelah komponen React diberikan kepada ReactDOM dengan perintah ini, komponen tersebut dapat di-render ke dalam DOM dan akan ditampilkan di halaman web dengan mengambil referensi ke elemen dengan id "root".

Prop

A white background with black text

Description automatically generated

|  |
| --- |
| *// Props Children in Class Component*  class Practice3 *extends* React**.***Component* {      render() {  **return** <h3 *id***=**{*this***.***props***.***module*}>Practice: {*this***.***props***.***children*}</h3>**;**      }  }  *// Buat elemen React dengan komponen Practice3 dan prop children="ReactJS"*  const myElement **=** <Practice3 *module***=**"01">ReactJS</Practice3>**;**  *// Merender elemen React ke dalam elemen dengan id "root"*  ReactDOM**.**createRoot(document**.**getElementById('root'))**.**render(myElement)**;** |

4.3 Functional Component

A white background with black text

Description automatically generated

|  |
| --- |
| *// Functional Component*  const ReactTitle **=** () **=>** {  **return** <h3>Currently learning ReactJS</h3>  }  const JavascriptTitle **=** () **=>** {  **return** <h3>Currently learning Javascript</h3>  }  const MyComponent **=** () **=>** {  **return** (          <div>              <ReactTitle />              <hr />              <JavascriptTitle/>          </div>      )**;**  }  let myElement **=** <MyComponent />  ReactDOM**.**render(<MyComponent />**,** document**.**getElementById('root'))**;** |

**ReactTitle dan JavascriptTitle** merupakan dua functional components. Mereka adalah komponen React sederhana yang hanya mengembalikan elemen-elemen JSX. ReactTitle mengembalikan elemen <h3> dengan teks "Currently learning ReactJS," dan JavascriptTitle mengembalikan elemen <h3> dengan teks "Currently learning Javascript."

**MyComponent** juga merupakan functional component. Ini adalah komponen yang lebih kompleks yang menggabungkan dua komponen sebelumnya, yaitu ReactTitle dan JavascriptTitle, dalam sebuah elemen <div>. Dalam komponen ini, Anda menggunakan <hr /> (garis horizontal) untuk memisahkan kedua judul.

**ReactDOM** untuk merender MyComponent ke dalam DOM. ReactDOM adalah bagian dari React yang digunakan untuk menghubungkan komponen React dengan elemen DOM dalam halaman web Anda. Dalam kode tersebut, komponen MyComponent akan dirender ke dalam elemen dengan id 'root' pada halaman web Anda.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| *SEBELUM* | *SESUDAH* |
| const Practice = (props) => <h3>Practice: {props.course}</h3>; myElement = (  <div>  <Practice course="Javascript"/>  <Practice course="ReactJS"/>  <Practice course="React Native"/>  </div> );  const myApp = myElement;  ReactDOM.createRoot(document.getElementById('root')).render(myApp); | *// Functional Component dengan props*  const Practice **=** (**props**) **=>** <h3>Practice: {props**.***course*}</h3>**;**  *// Mendeklarasikan myElement sebagai variabel dengan const atau let*  const myElement **=** (      <div>          <Practice *course***=**"Javascript"/>          <Practice *course***=**"ReactJS"/>          <Practice *course***=**"React Native"/>      </div>  )**;**  *// Render myElement ke dalam DOM*  ReactDOM**.**createRoot(document**.**getElementById('root'))**.**render(myElement)**;** |

Pada kode yang sebelumnya Masalahnya terletak pada penggunaan variabel myElement dalam kode pertama. Di kode pertama, Anda tidak mendeklarasikan variabel myElement dengan benar sebelum menggunakannya. Seharusnya Anda mendeklarasikan myElement sebagai variabel dengan menggunakan const atau let

4.4

4.5 State

Tampilan code 1

A screenshot of a computer

Description automatically generated

Tampilan code 2

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Kode 1 | Kode2 |
| *// Cat Component*  class Cat *extends* React**.***Component* {  *// State Declaration*      state **=** {          isHungry**:** true      }**;**      render() {  **return** (              <div>                  <p>I am {*this***.***props***.***name*}, and I am {*this***.***state***.***isHungry* **?** " hungry" **:** " full"}!</p>                  <button  *onClick***=**{() **=>** {  *this***.**setState({ isHungry**:** false })**;**                      }}  *disabled***=**{**!***this***.***state***.***isHungry*}                  >                      {*this***.***state***.***isHungry* **?** "Pour me some milk, please!" **:** "Thank you!"}                  </button>              </div>          )**;**      }  }**;**  ReactDOM**.**render(<Cat *name***=**"Whiskers" />**,** document**.**getElementById('root'))**;** | class Cat *extends* React**.***Component* {      state **=** {          isHungry**:** true      }**;**      render() {  **return** (              <div>                  <p>I am {*this***.***props***.***name*}, and I am {*this***.***state***.***isHungry* **?** " hungry" **:** " full"}!</p>                  <button  *onClick***=**{() **=>** {  *this***.**setState({ isHungry**:** false })**;**                      }}  *disabled***=**{**!***this***.***state***.***isHungry*}                  >                      {*this***.***state***.***isHungry* **?** "Pour me some milk, please!" **:** "Thank you!"}                  </button>              </div>          )**;**      }  }  class Cafe *extends* React**.***Component* {      render() {  **return** (              <div>                  <Cat *name***=**"Munkustrap" />                  <Cat *name***=**"Spot" />              </div>          )**;**      }  }  const myElement **=** <Cafe />**;**  *// Render komponen Cafe ke dalam elemen dengan ID "root"*  ReactDOM**.**createRoot(document**.**getElementById('root'))**.**render(myElement)**;** |

Kode pertama hanya merender satu komponen Cat dengan nama "Whiskers" langsung ke dalam elemen dengan ID "root" menggunakan ReactDOM.render. Ini berarti hanya satu kucing dengan nama "Whiskers" akan ditampilkan di halaman web.

Kode kedua memiliki dua komponen Cat dengan nama "Munkustrap" dan "Spot" yang dirender melalui komponen Cafe. Dalam kode ini, kita membuat sebuah komponen Cafe yang berisi dua komponen Cat yang berbeda. Kemudian, kita merender komponen Cafe ke dalam elemen dengan ID "root" menggunakan ReactDOM.createRoot.