FOLKERT NIEWIJK 16-04-2020

FUNCTIONEEL PROGRAMMEREN WORKSHOP

PARADIGMA'S BINNEN SOFTWARE ONTWIKKELING

- **Welkom**

- **14:00 - 14:30 Presentatie**
- **14:30 - 15:00 Live coding**
- **15:00 - 15:50 Zelf coden**
- **15:50 - 16:00 Afronding**

PARADIGMA'S BINNEN SOFTWARE ONTWIKKELING

- **Introductie**
- **Imperatief**
- **Functioneel**

PARADIGMA'S BINNEN SOFTWARE ONTWIKKELING

Introductie

- **Paradigma**: Een manier van programmeren
- Gebaseerd op een set principes en theorie
- Elk paradigma heeft een andere kijk op hoe je kunt programmeren

- **Idoom**: Een programmeerpatroon binnen een paradigma

IMPERATIEF PROGRAMMEREN

- **Imperatief:** hoe lossen we een probleem op
- Procedureel, object-georienteerd

- **Declaratief**: wat lossen we op
- Functioneel, logica

IMPERATIEF PROGRAMMEREN

- **Gebaseerd op de Turing Machine**
- **Het geheugen slaat de variabelen op - State**
- **Het programma verandert de variabelen - Mutatie**
- **Variabelen worden aangewezen. Controlestructuren opgebouwd.**
- **Volgt het natuurlijk model van hoe computerhardware werkt**

IMPERATIEF PROGRAMMEREN

- **Gebaseerd op de Turing Machine**
- **Het geheugen slaat de variabelen op - State**
- **Het programma verandert de variabelen - Mutatie**
- **Variabelen worden aangewezen. Controlestructuren opgebouwd.**
- **Volgt het natuurlijk model van hoe computerhardware werkt**

IMPERATIEF PROGRAMMEREN

Voorbeeld

```
Class Sum {
   const sum: number = 0; // State
   this.add(value: number) { this.sum += value; } // Mutatie
  this.get() { return sum; }
}

Const sum = new Sum();
sum.add(1);
sum.add(2);
sum.get(); // 3
```

FUNCTIONEEL PROGRAMMEREN

- **Gebaseerd op Lambda Calculus**
- **Programma: een set van functies**
- **Zelfde input geeft altijd dezelfde output - Puurheid**
- **Functie verandert de context niet - Side effects**
- **De input mag niet worden veranderd - Immutability**

FUNCTIONEEL PROGRAMMEREN

Voorbeeld

```
Const sum:number = (values: number[]): number => {
  let total: number = 0;
  values.forEach((value) => total += value);
  return total;
}

// je maakt een waarde aan! Total
```

FUNCTIONEEL PROGRAMMEREN

Voorbeeld

```
// beter:


Const sum: number = (values: numbers[]): number => {
  return values[0] + sum(values.splice(1,values.length-1);
}


// Puurheid: Dezelfde input voor sum() geeft dezelfde output.
```

FUNCTIONEEL PROGRAMMEREN

Imperatief voorbeeld

```
Class Sum {
   const sum: number = 0;
   this.add(value: number) { this.sum += value; }
  this.get() { return sum; }
}

Const sum: number = new Sum();
sum.add(1); Side effect: Elke keer als add wordt aangeroepen verandert het resultaat van get
sum.add(2);
sum.get(); // 3 Puurheid: Elke call van add() verandert het resultaat van get()
```

FUNCTIONEEL PROGRAMMEREN

Side effects

Const getLocalStorage: string = () => localStorage.get("key");

// Side-effect: Je krijgt niet altijd dezelfde waarde terug van deze functie.

FUNCTIONEEL PROGRAMMEREN

Side effects

Const createNewDate: Date = () => new Date();

// Side-effect: Je krijgt niet altijd dezelfde waarde terug van deze functie. De tijd verandert steeds

FUNCTIONEEL PROGRAMMEREN

Side effects

Const getRandomValue: number = () => Math.random();

// Side-effect: Je krijgt niet altijd dezelfde waarde terug van deze functie.

FUNCTIONEEL PROGRAMMEREN

Immutability

Const getLastFromArray: any = (array: any[]) => array.pop();

// Immutability: Je mag de input niet veranderen.

Hier wordt de originele input array veranderd door de code.

FUNCTIONEEL PROGRAMMEREN

Voordelen

- **Input en output zijn altijd hetzelfde dus:**
  - **Makkelijker te begrijpen wat de code doet**
  - **Makkelijker te testen wat de code doet**
  - **Makkelijker om de code te parallelliseren.**

- **Wel**
  - **Als niet-pure functies en side-effects nodig zijn hoe gaan we hier mee om?**

    Hier zijn speciale Functionele Programmeertalen voor geschreven.