Functional Programming in JavaScript

Your Speaker

Shravan Kumar Kasagoni

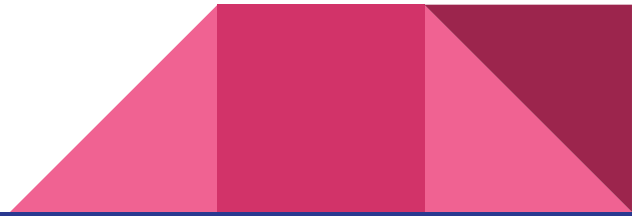
Senior Developer – RealPage

Microsoft MVP – VSDT

Organizer - Microsoft User Group Hyderabad

http://theshravan.net | @techieshravan

http://github.com/techieshravan



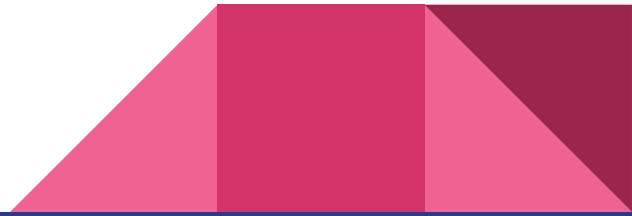
Agenda

1. What is Functional Programming?
2. Functional Programming Principles
3. Why Functional Programming?

What is Functional Programming?

Functional Programming

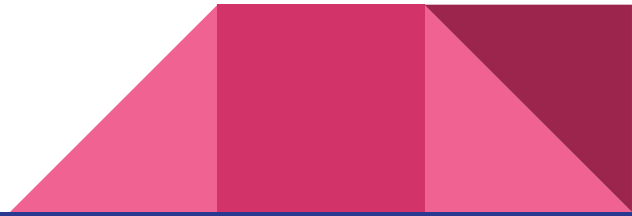
Functional Programming is programming with functions



Pure Functional Programming

Functional programming is a programming paradigm — a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

Functional Programming is programming with mathematical functions



Functional Programming Principles

- Pure Functions
- Immutability
- Composition
- Closure
- Recursion
- List Transformations

```
function sumOfTwoNumbers(x, y) {
  return x + y;
}
```

Always Returns The Same Result

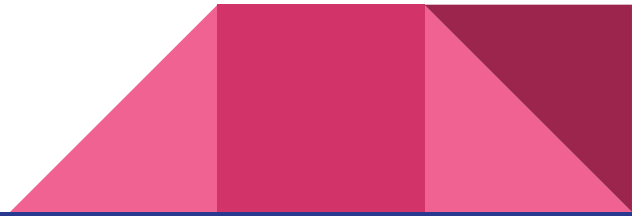
```
function tickElapsedFrom(year) {
  var now = new Date();
  var then = new Date(year, 0, 1);

  return (then.getTime() - now.getTime());
}
```

```
getCurrentProgram(guide: TVGuide, channel: number): Program {

  Schedule schedule = guide.getSchedule(channel);

  Program current = schedule.programAt(new Date());

  return current;
}
```

Method written using TypeScript

What is a pure function?

- For the same input, will always return the same output
- Produces no side effects
- Relies on no external state



How to convert an impure function to a pure function?

- Honest Signature
  - Should have precisely defined inputs and outputs
- Referentially Transparent
  - Doesn't affect or refer to the global state ()
  - Solely depended on its parameters

```
getCurrentProgramAt(guide: TVGuide, channel: number, when: Date): Program {

  Schedule schedule = guide.getSchedule(channel);

  Program current = schedule.programAt(when);

  return current;
}
```

Method written using TypeScript

```ts
class UserProfile {
  user: User;
  address: string;

  updateUser(userId: number, userName: string) {
    this.user = new User(userId, name);
  }
}
```

```ts
class User {
  id: number;
  name: string;

  constructor(id: number, name: string) {
    this.id = id;
    this.name = name;
  }
}
```

Classes written using TypeScript

Immutability

```
class UserProfile {
  user: User;
  address: string;

  updateUser(userId: number, userName: string): User {
    return new User(userId, name);
  }
}

class User {
  id: number;
  name: string;

  constructor(id: number, name: string) {
    this.id = id;
    this.name = name;
  }
}
```

Classes written using TypeScript

```
class Address {
  constructor(address) {
    this.address = address;
  }
}

class Customer {
  constructor(name, address) {
    this.name = name;
    this.address = address;
  }
}

class Repository {
  save(customer) {
  }
}
```

Classes written using ES2015

```
class CustomerService {
  process(customerName, address) {
    this.createAddress(address);
    this.createCustomer(customerName);
    this.saveCustomer();
  }

  createAddress(address) {
    this.address = new Address(address);
  }

  createCustomer(name) {
    this.customer = new Customer(name, this.address);
  }

  saveCustomer() {
    new Repository().save(this.customer);
  }
}
```

```
class CustomerService {
  process(customerName, address) {
    var _address = this.createAddress(address);
    var _customer = this.createCustomer(customerName, _address);
    this.saveCustomer(_customer);
  }

  createAddress(address) {
    return new Address(address);
  }

  createCustomer(name, address) {
    return new Customer(name, address);
  }

  saveCustomer(customer) {
    new Repository().save(customer);
  }
}
```

Classes written using ES2015

Composition

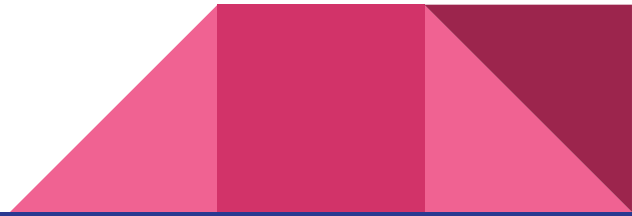
Closure

Recursion

List Transformations

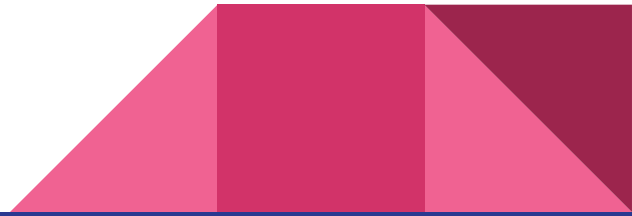
List Transformations

- Array.prototype.forEach()
- Array.prototype.map()
- Array.prototype.filter()
- Array.prototype.reduce()



Why Functional Programming?

- Helps to reduce Code Complexity
- Composable - Functions can Composed to build a complex system if they are pure
- Easy to reason about (predictable)
- Easier to Unit Test



Keep in touch

http://theshravan.net

http://twitter.com/techieshravan

http://github.com/techieshravan