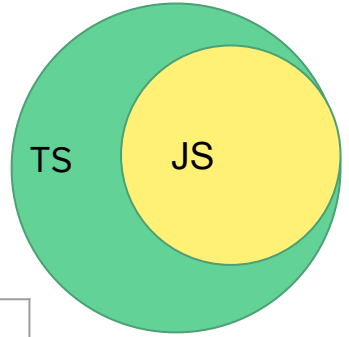
Declarative vs. Imperative Programming in JS/TS

Joey Merline, Katie Pikturna, Jake Smith, Gence Yalcin, Ali Youssef

Contents

Javascript (JS) vs Typescript (TS)



TS introduced on top of JS to aid in development of large-scale web applications

JS (est 1995)	**TS** (est 2012)
High level, JIT, dynamically typed	Transcompiles to JS, strongly typed
OOP (without classes), lightweight	OOP features, modules, functions
Various programming styles, can switch between imperative and declarative	Decorators extend class/method functionality in a declarative way

```
// Imperative
const arrayContainsAnotherArray = (needle, haystack) => {
  for(let i = 0; i < needle.length; i++) {
    if(haystack.indexOf(needle[i]) === -1)
      return false;
  }
  return true;
}
```

```
// Declarative
const arrayContainsOtherArray = (needle=[], haystack=[]) =>
    needle.every(el => haystack.includes(el));
```

Declarative vs. Imperative Programming

- Declarative Programming:
  - A programming paradigm that expresses the logic of computation without describing the control flow:
    - I have some number n of fruits, and for each fruit I would like to slice them in half
- Imperative Programming:
  - A programming paradigm that changes the state of the program using statements.
    - I have some number n of fruits. Assign a number to each fruit. Loop from 0 to 10. Get that fruit based on the number the fruit is. Slice that fruit in half.

```
7   const fruits = [
8       new Fruit("apple"),
9       new Fruit("banana"),
10      new Fruit("orange")
11  ];
```

```
13  // Imperative
14  for (var i = 0; i < fruits.length; i++) {
15      fruits[0].sliceInHalf()
16  }
```

```
18  // Declarative
19  fruits.forEach(fruit => fruit.sliceInHalf());
```

Pros and Cons of Declarative Programming in JS/TS

Pros:

- Shorter and more efficient code compared to imperative
- Allows you to work with solution states, to let programs figure things out for you. (Ex. Upsert)
- Reduces side effects by discouraging variables in favor of constructs like pipelines.

Cons:

- Can be hard for external developers to understand
- Unfamiliarity with the conceptual model (solution states)

Imperative

```
const isEven = x => x % 2 === 0
const input = [ 1, 2, 3, 4, 5 ]
const output = []
for (let i = 0; i < input.length; i++) {
  if (isEven(input[i])) {
    output.push(input[i])
  }
}

output //=> [ 2, 4, 6 ]
```

Declarative

```
const isEven = x => x % 2 === 0
const input = [ 1, 2, 3, 4, 5 ]
const output = input.filter(isEven)

output //=> [ 2, 4, 6 ]
```

Pros and Cons of Imperative Programming in JS/TS

Pros

- Easy to understand because of additional lines of code
- Easier to learn how to program
- Characteristics of specific applications can be taken into account (more control)

Cons

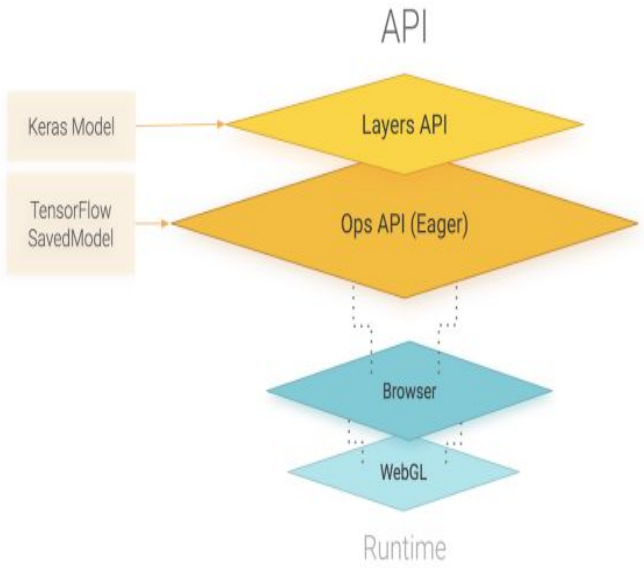
- The additional lines of code take longer to write/read
- Can be a waste of time for experienced developers
- MORE confusing in a complex project
- Higher risk of errors

Combining Imperative and Declarative Programming

Separates low-level and high-level elements of an application.

- Low-level: Imperative programming
- High-level: Declarative programming

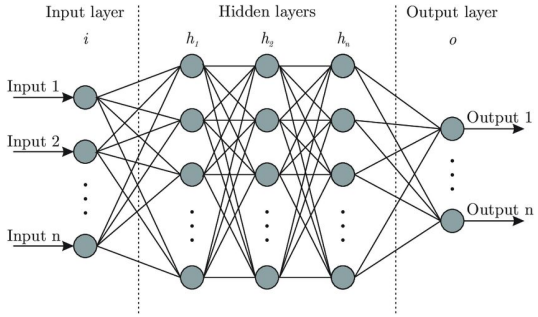
  Allows developers to use an certain elements of application without needing to interact with the specifics of its implementation directly.



High and low level API's for TensorFlow.

Example: TensorFlow.js

Low-level:



Low-level: Handles the actual process behind the neural network.

High-level:

TF.js

```
// A sequential model is a container which you can add layers to.
const model = tf.sequential();

// Add a dense layer with 1 output unit.
model.add(tf.layers.dense({units: 1,
                           inputShape: [1],
                           activation: 'softmax'
                          }));

// Specify the loss type and optimizer for training.
model.compile({loss: 'meanSquaredError', optimizer: 'adam'});

// Generate some synthetic data for training.
const xs = tf.tensor2d([[1], [2], [3], [4], [5], [6], [7]], [7, 1]);
const ys = tf.tensor2d([[1], [3], [5], [7], [9], [11], [13]], [7, 1]);

// Train the model.
model.fit(xs, ys, {epochs: 1000});

// After the training, perform inference.
const pred = model.predict(tf.tensor2d([[8]], [1, 1]));
pred.print();
```

High-Level: Developers can simply call functions that are handled by the low-level framework.

Discussion Questions

Did you ever hear of declarative/imperative programming styles before this class? If yes, where? If no, do you think the styles should be emphasized more in a CS curriculum?

While implementing modules in JS, what use cases would warrant a declarative over an imperative approach? Is it best to stick with one or the other, or do some implementations warrant a hybrid approach?