ACADEMIND

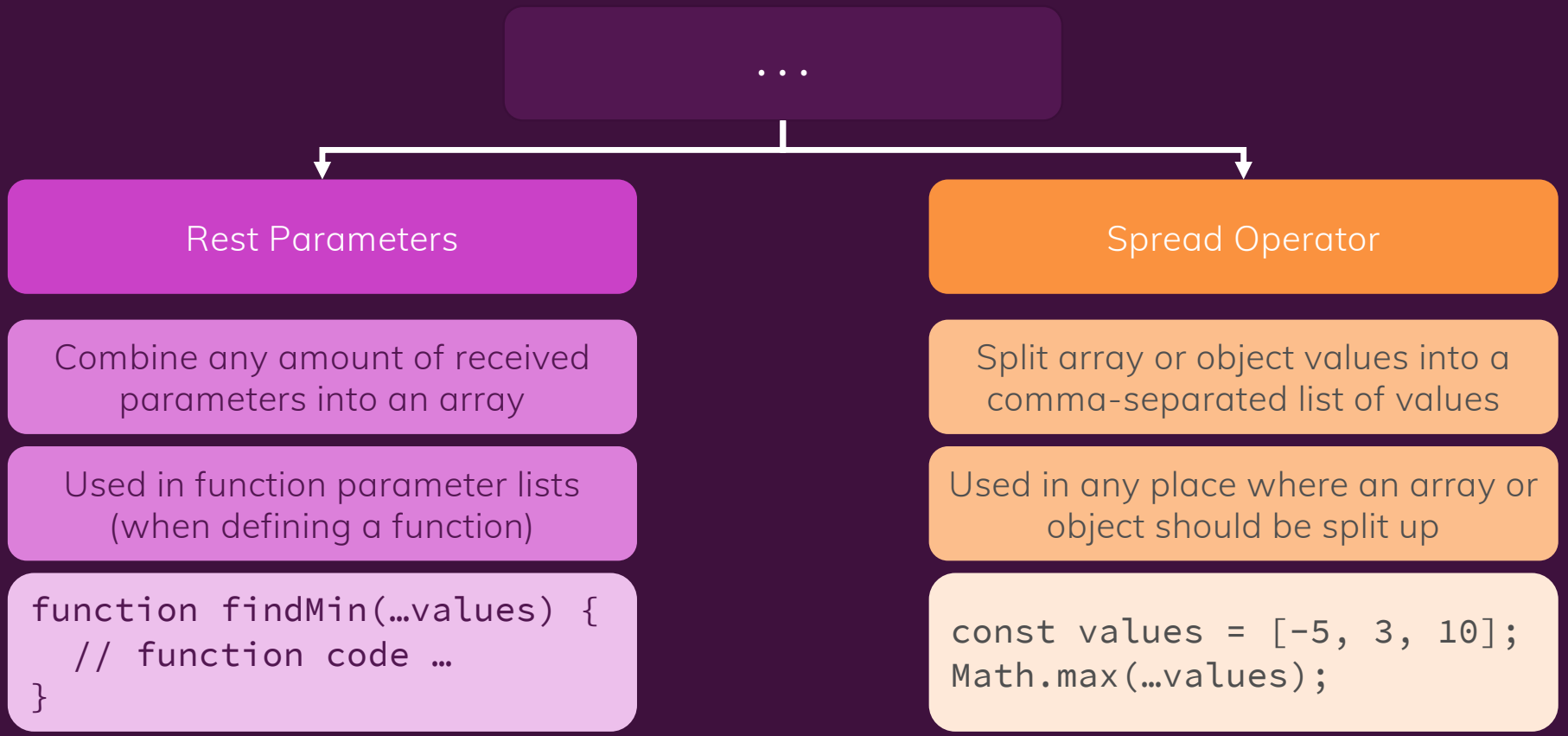
# Special Values / Value Types

Besides “normal” values like 32 (number), ”Hi there!” (string) or [1, 2, 10] (array), JavaScript also has a few “special purpose” values built-in.

| | | |
|---|---|---|
| **undefined** | Represents “no value” (e.g. used if a variable holds no value) | `let age; // no value assigned`<br>`console.log(age); // outputs undefined` |
| **null** *(will be shown later!)* | Also represents “no value”. Use “null” if you explicitly want to set this as a value | `let age = null; // age should not be set`<br>`console.log(age); // outputs null` |
| **NaN** | Not a Number<br>A number was expected but could not be derived. | `let result = 1 + undefined;`<br>`console.log(result); // outputs NaN` |

ACADEMIND

# Rest Parameters & Spread Operator



ACADEMIND

# Primitive & Reference Values

| Primitive Values | Reference Values |
| --- | --- |
| Simple values | More complex values |
| Numbers, strings, booleans, undefined, null | All objects (incl. functions and arrays) |
| Stored in a more basic kind of computer memory | Stored in a different kind of computer memory |
| Don’t occupy a lot of space, hence copying values is “cheap” | Can occupy a lot of space, hence copying values is “expensive” |
| Values themselves are stored in variables or constants | Only addresses of (shared) values are stored in variables or constants |

ACADEMIND

# Nesting Callback Functions

Dummy Code

```
function storeData() {
  fs.readFile('input-data.csv', function(error, data) {
    const cleanedData = cleanData(data);
    storeDatainDatabase(cleanedData, function(error, result) {
      if (result.changedData) {
        confirmDataChange(function(error, done) {
          if (!error && done) {
            res.render('success');
          }
        });
      }
    });
  });
}
```

Lots of dependent asynchronous tasks lead to nested callback functions ➔ **"Callback Hell"**

*(because the code becomes harder to read, understand and maintain)*

ACADEMIND

# Introducing “Promises”

**Promises in Real Life**

Today

You lend me money

In the future

I return the money

In JavaScript, we also have built-in, standardized objects that are called “Promises” that can wrap asynchronous operations

ACADEMIND

# Using Promises To Solve “Callback Hell”

Dummy Code

```
fs.readFile('input-data.csv')
    .then(function(data) {
      const cleanedData = cleanData(data);
      return storeDataInDatabase(cleanedData);
    })
    .then(function(result) {
      if (result.changedData) {
        return confirmDataChange();
      }
    })
    .then(function(done) {
      if (done) {
        res.render('success');
      }
    }
```

Because promises can be returned and chained, less nesting is required to orchestrate multiple, dependent asynchronous tasks