Python

```python
# Variables
full_name = "Jane Hacker"
pi = 3.14

# lists ("Arrays" in JS)
names = ["John", "Paul", "G"]

# dicts (similar to "Objects")
translation = {
    "ola": "Hello",
    "oi": "hi",
}

# For loops
for name in names:
    print("name:", name)

# While loops
x = 0
while x < 3:
    print("X:", x)
    x = x + 1

# If-statements
if full_name == "Jane":
    print("Hi, Jane!")
elif full_name == "Alice":
    print("Hey Alice")
else:
    print("Don't know you")

# Looping through dict
for k, v in translation.items():
    print(key, value)


# Functions
def greeter(name):
    print("Hi", name)
greeter("Bob")


# Lambda function
dst = lambda x, y: x*x + y*y

# Conjunctions
if age < 18 and drink == "beer":
    print("Too young kiddo")
if age > 18 or drink == "soda":
    print("Great choice")


# Class syntax
class User(BaseUser):
    def __init__(self, name):
        self.name = name
        self.logged_in = False
    def log_in(self):
        self.logged_in = True
user = User("janeqhacker")
```

JavaScript

```javascript
// Variables
let fullName = "Jane Hacker";
const pi = 3.14;

// Arrays ("lists" in Py)
let names = ["John", "Paul", "G"];

// Objects (similar to "dicts")
let translation = {
    ola: "Hello",
    oi: "Hi",
};

// For loop
for (let name of names) {
    console.log("name:", name);
}
// While loops
let x = 0;
while (x < 3) {
    console.log("X:", x);
    x++;
}
// If-statements
if (fullName === "Jane") {
    console.log("Hi, Jane!");
} else if (fullName === "Alice") {
    console.log("Hey Alice");
} else {
    console.log("Don't know you");
}
// Looping through object
for (const [k, v] of
    Object.entries(translation)) {
  console.log(key, value);
}
// Functions
function greeter(name) {
    console.log("Hi", name);
}
greeter("Bob")

// Lambda function
const dst = (x, y) => x*x + y*y;

// Conjunctions
if (age < 18 && drink === "beer") {
    console.log("Too young kiddo");
}
if (age > 18 || drink === "soda") {
    console.log("Great choice");
}
// Class syntax
class User extends BaseUser {
    constructor(name) {
        this.name = name;
        this.loggedIn = false;
    }
    logIn() {
        this.loggedIn = true;
    }
}
let user = new User("janeqhacker");
```

Collection operations

```javascript
// Destructuring object
let {ola, oi} = translation;
// ...and the reverse
const age = 3;
let info = {fullName, age};

// Combine obj, arrays with splat
const addedTranslations = {
    ...translation,
    gato: "cat",
};
const extra = [...names, "Mary"];
```

Legacy Syntax

```javascript
// Loop through properties
for (var i in arr) {}
// C-style: Loop through numbers
for (var i = 0; i < 100; i++) {}
var a = 3; // function-scoped
a = 3; // globally scoped
```

Async & Callbacks

```javascript
// Callbacks (common in node.js)
fs.readFile("file.txt",
(err, data) => {
    if (err) throw err;
    console.log(data);
});

// Promise (common in browser)
fetch("http://site.com/api.json")
.then(response => response.json())
.then(data => {
    console.log("Resp:", data);
})
.catch(err => {
    console.log("error", err);
});
```

asynchronous Instead of pausing ("blocking") for a slow operation, the asynchronous approach is to put-off starting the slow operation, then call a function ("callback") at a later time to signal it's done

callback A function passed as an argument to be called later when an event is triggered

promise Another popular way to do callbacks, with a `.then` syntax

Variable Declaration

let Declare a variable (block scoped)

const Like `let`, cannot be reassigned.