sourcemind

JAVASCRIPT LANGUAGE SPECIFICATIONS

JavaScript (often shortened to **JS**) is a lightweight, interpreted, object-oriented language with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well. It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles.

Interpreted vs Compiled Programming Languages

Compiled Languages

C, C++, Erlang,

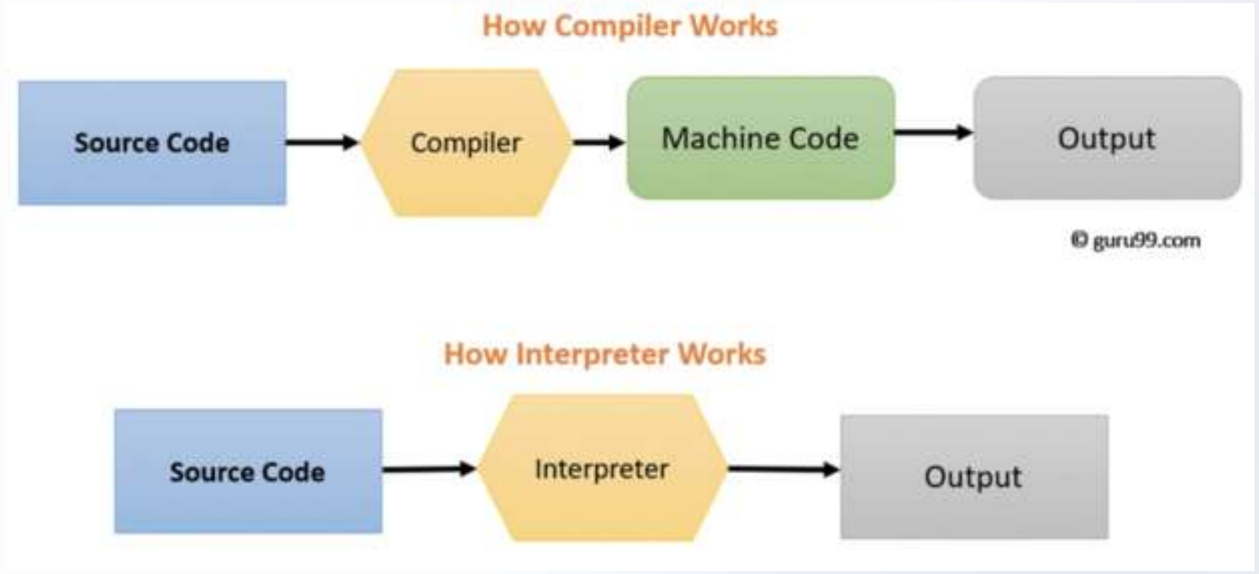
Haskell, Rust, and Go.

Compiled languages are converted directly into machine code that the processor can execute. As a result, they tend to be faster and more efficient to execute than interpreted languages.
They also give the developer more control over hardware aspects, like memory management and CPU usage.

Interpreted Languages

PHP, Ruby, Python, JavaScript

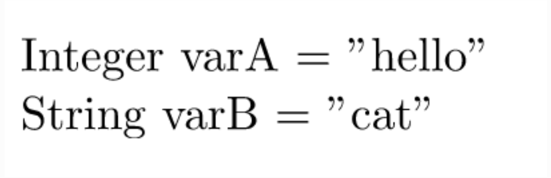
Interpreters run through a program line by line and execute each command.
Your translator friend can then convey that change to you as it happens.
Interpreted languages were once significantly slower than compiled languages.
But, with the development of just-in-time compilation, that gap is shrinking.



sourcemind

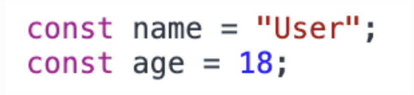
Dynamic typing vs. static typing

In statically typed programming languages, type checking occurs at compile time. At compile time, source code in a specific programming language is converted to a machine-readable format. This means that before source code is compiled, the type associated with each and every single variable must be known.

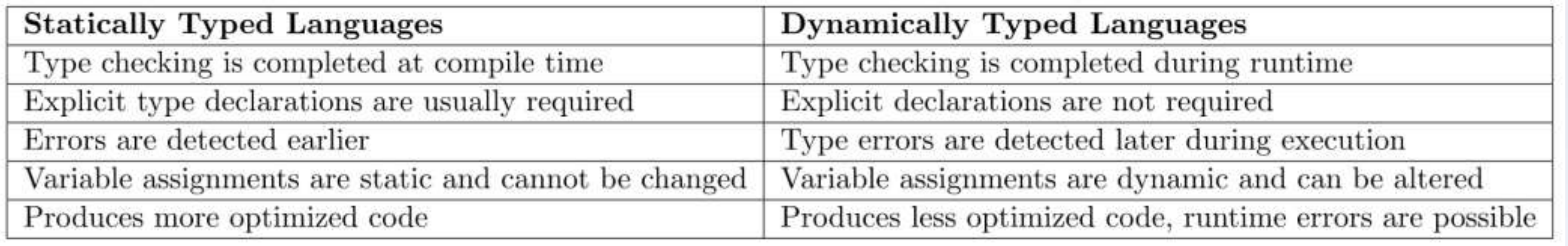


```
Integer varA = "hello"
String varB = "cat"
```

Conversely, **in dynamically typed languages, type checking takes place at runtime or execution time**. This means that variables are checked against types only when the program is executing. Some examples of programming languages that belong to this category are Python, JavaScript, Lisp, PHP, Ruby, Perl, Lua, and Tcl.



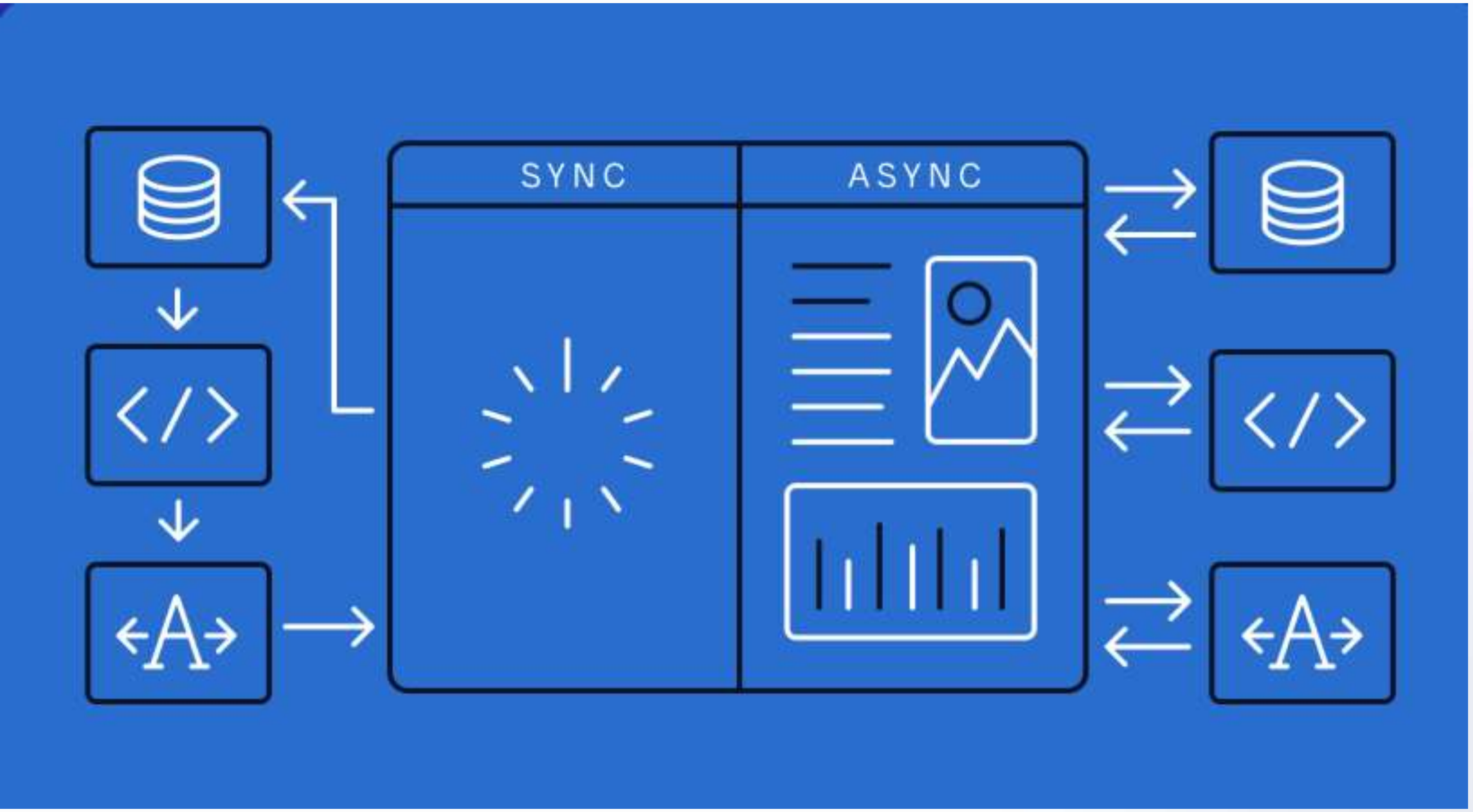
```
const name = "User";
const age = 18;
```



Statically Typed Languages	Dynamically Typed Languages
Type checking is completed at compile time	Type checking is completed during runtime
Explicit type declarations are usually required	Explicit declarations are not required
Errors are detected earlier	Type errors are detected later during execution
Variable assignments are static and cannot be changed	Variable assignments are dynamic and can be altered
Produces more optimized code	Produces less optimized code, runtime errors are possible

sourcemind

Asynchronous vs. Synchronous Programming



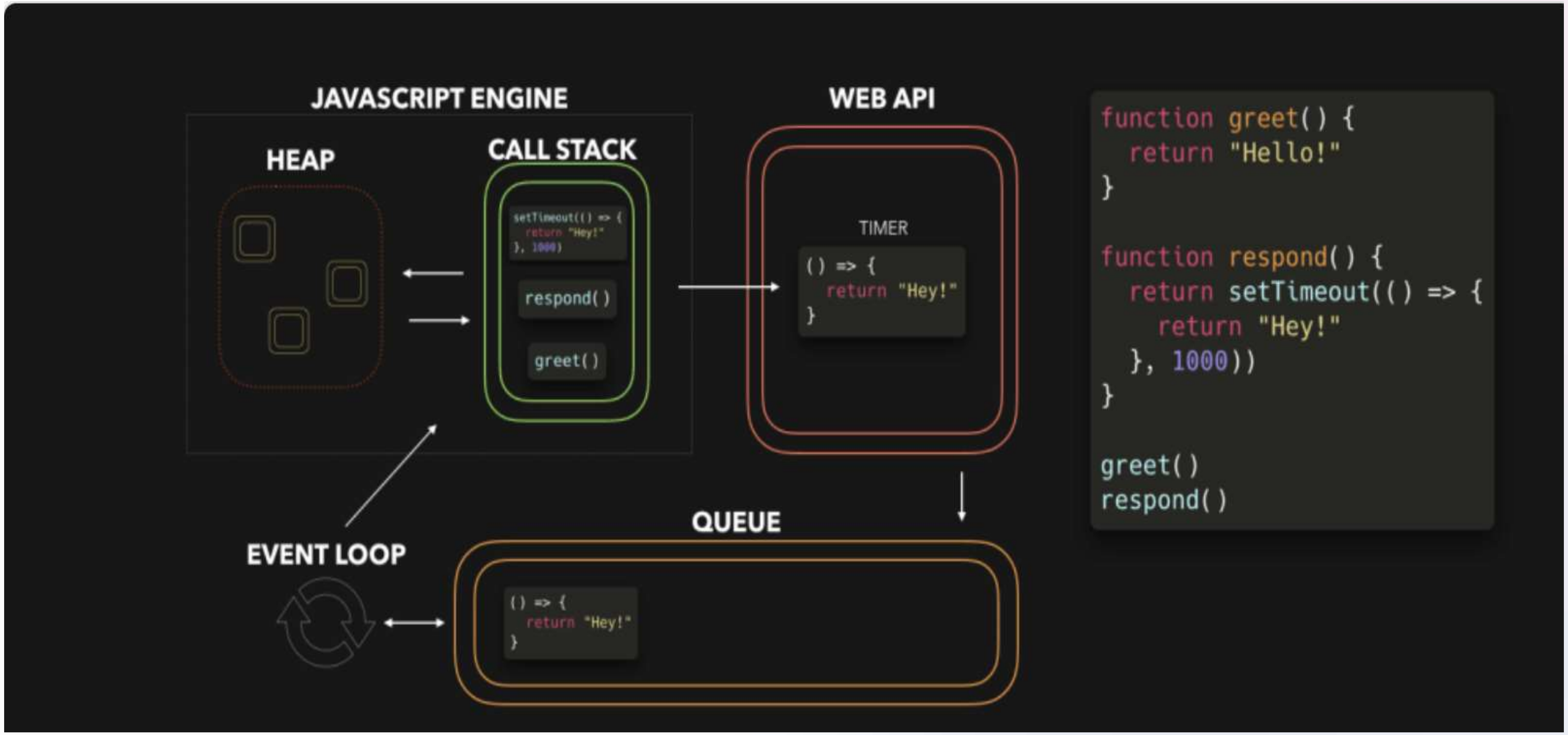
JavaScript is a **synchronous**, blocking, single-threaded language.
That just means that only one operation can be in progress at a time.
That's not the entire story, though!

Example

```javascript
console.log("1");
setTimeout(() => console.log("2"));
const promise = new Promise((res) => res());
promise.then(() => console.log("3"));
console.log("4");

```

sourcemind

Event Loop



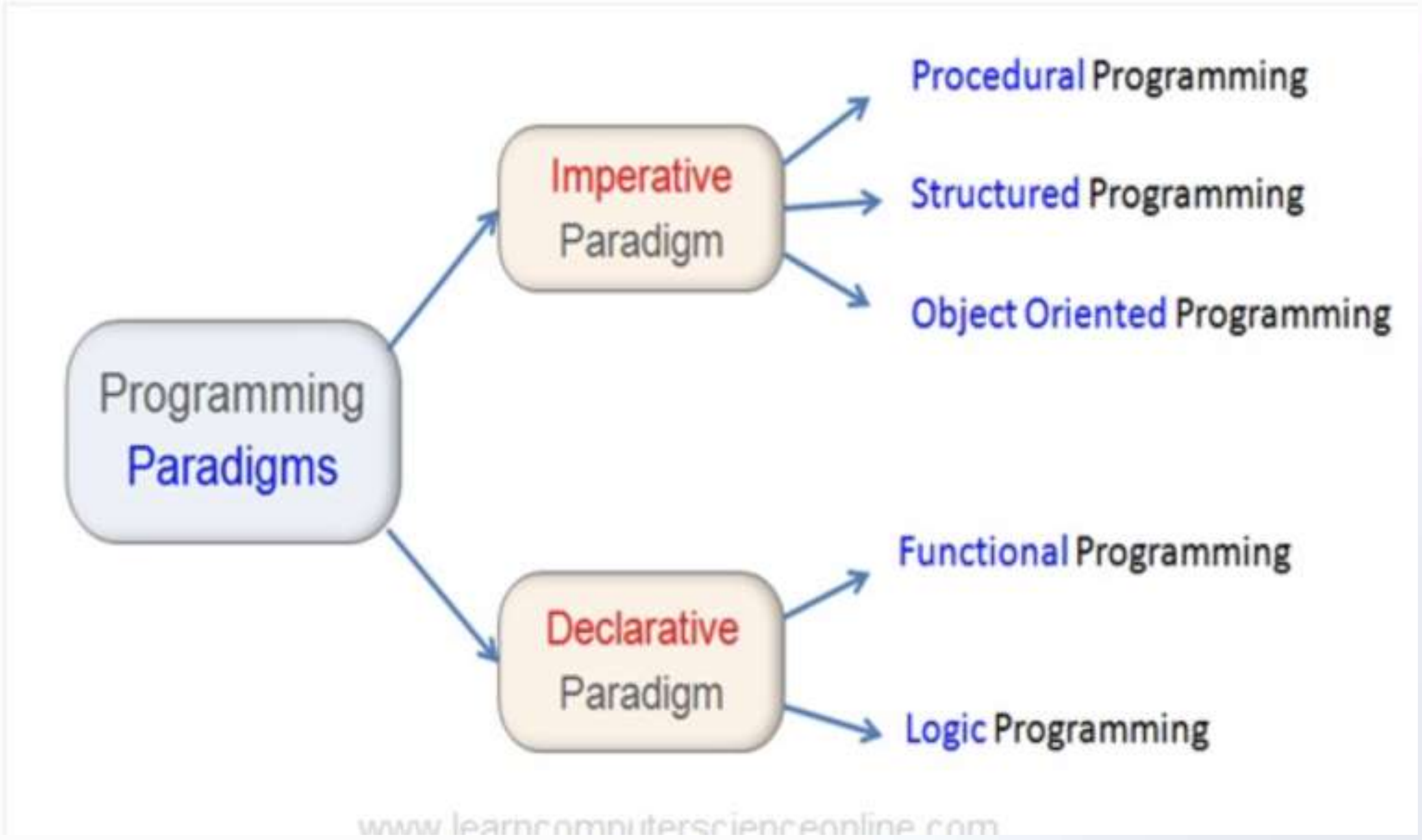
```javascript
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

Link => https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif

sourcemind

Programming Paradigms

The programming paradigm is all about the writing style and organizing the program code in a specific way. Each paradigm advocates a specific way to organize the program code.



Imperative programming languages differ from declarative languages on one fundamental point:
Imperative programming focuses on the "hov
declarative programming on the "what".

Imperative Programming

Imperative programming consists of sets of detailed instructions that are given to the computer to execute in a given order.
It's called "imperative" because as programmers we dictate exactly what the computer has to do, in a very specific way.

Imperative programming focuses on describing *how* a program operates, step by step.

Procedural Programming

Procedural programming is a programming paradigm built around the idea that programs are sequences of instructions to be executed. They focus heavily on splitting up programs into named sets of instructions called procedures, analogous to functions.

Structured Programming

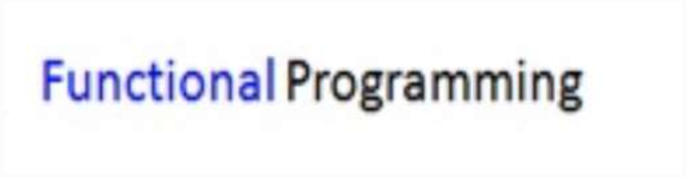
Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for) block structures, and subroutines.

Object Oriented Programming

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

Declarative Programming

Declarative programming is **a method to abstract away the control flow for logic required for software to perform an action, and instead involves stating what the task or desired outcome is**. Declarative programming is a high-level programming concept, which is the opposite of imperative programming.



Functional Programming

Functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are trees of expressions that map values to other values, rather than a sequence of imperative statements which update the running state of the program.

Logic Programming

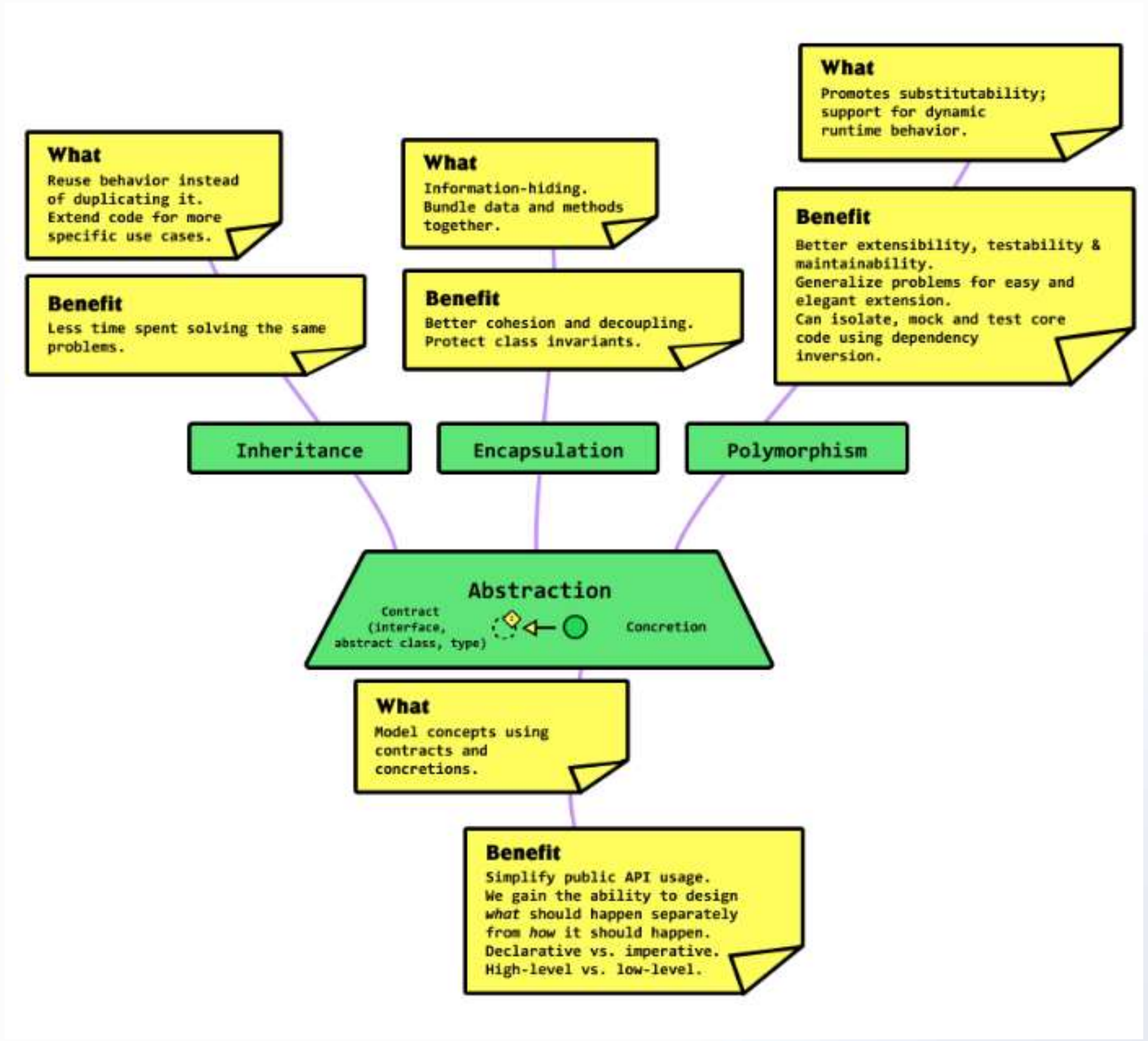
Logic programming is a programming paradigm which is largely based on formal logic.
Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. Major logic programming language families include Prolog, answer set programming (ASP) and Datalog. In all of these languages, rules are written in the form of clauses:

Functional Programming

The principles of Functional Programming

- Purity.
- Immutability.
- Disciplined state.
- First class functions and high order functions.
- Type systems.
- Referential transparency.

OOP


What
Reuse behavior instead of duplicating it. Extend code for more specific use cases.
Benefit
Less time spent solving the same problems.
What
Information-hiding. Bundle data and methods together.
Benefit
Better cohesion and decoupling. Protect class invariants.
What
Promotes substitutability; support for dynamic runtime behavior.
Benefit
Better extensibility, testability & maintainability. Generalize problems for easy and elegant extension. Can isolate, mock and test core code using dependency inversion.
Inheritance
Encapsulation
Polymorphism
Abstraction
Contract (interface, abstract class, type)
Concretion
What
Model concepts using contracts and concretions.
Benefit
Simplify public API usage. We gain the ability to design *what* should happen separately from *how* it should happen. Declarative vs. imperative. High-level vs. low-level.

sourcemind

Tasks

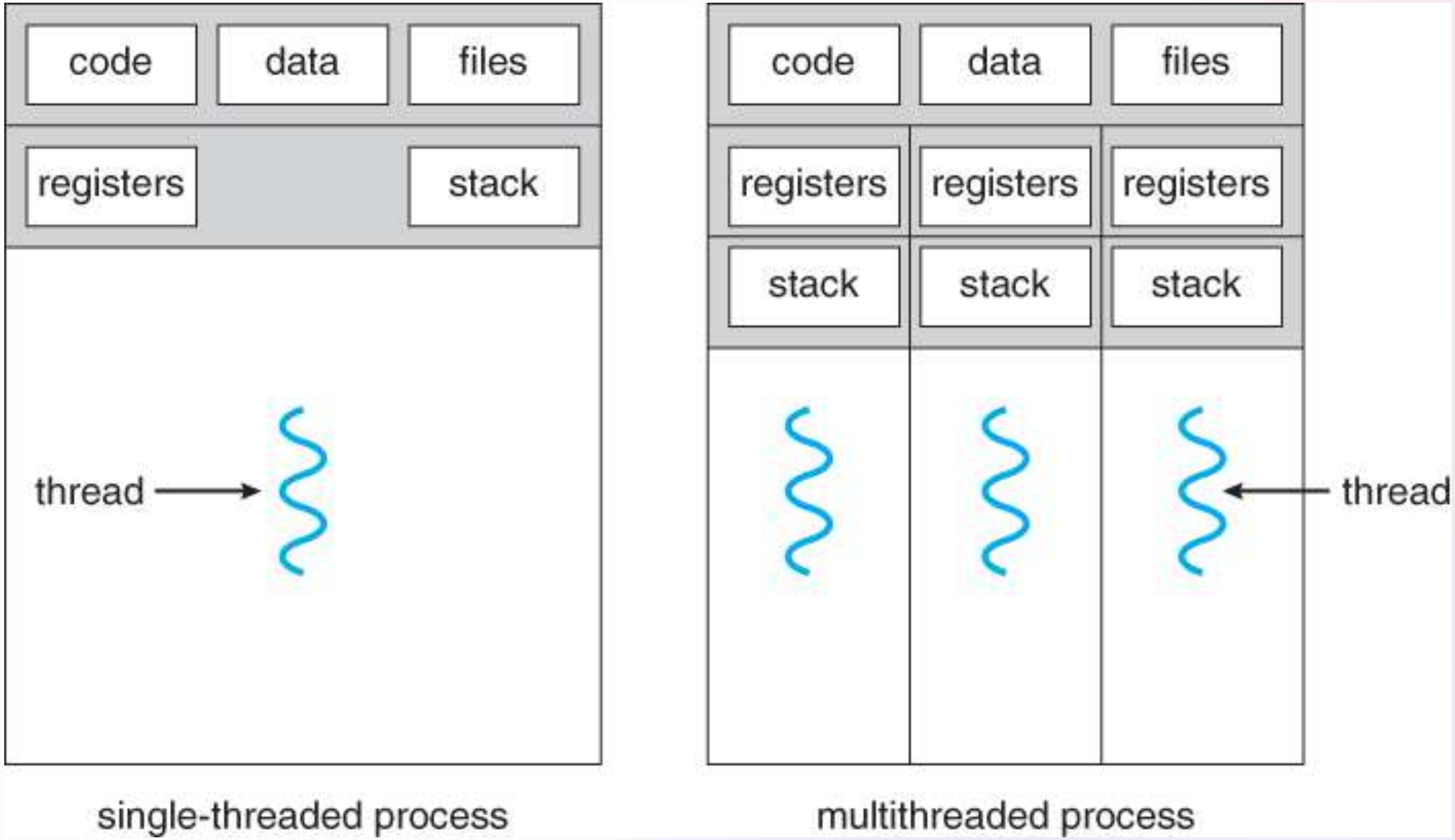
1. Create a pure function in JS

2. Create a High Order function

3. Create a Cat class which is inhertited from Animal class.

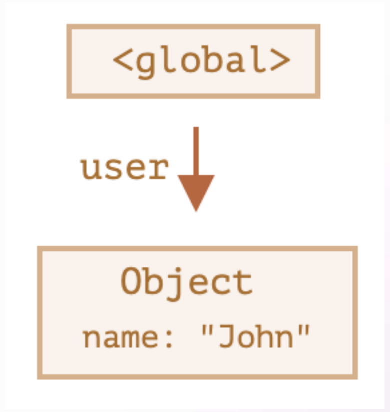
4. Creata a functoin which receive a string and returning SECRET_INFO + string value (SECRET_INFO must be encapsulated)

sourcemind

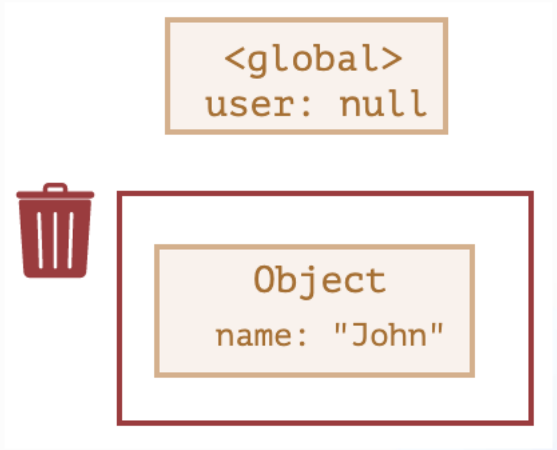

code
data
files
registers
stack
thread
single-threaded process
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded process

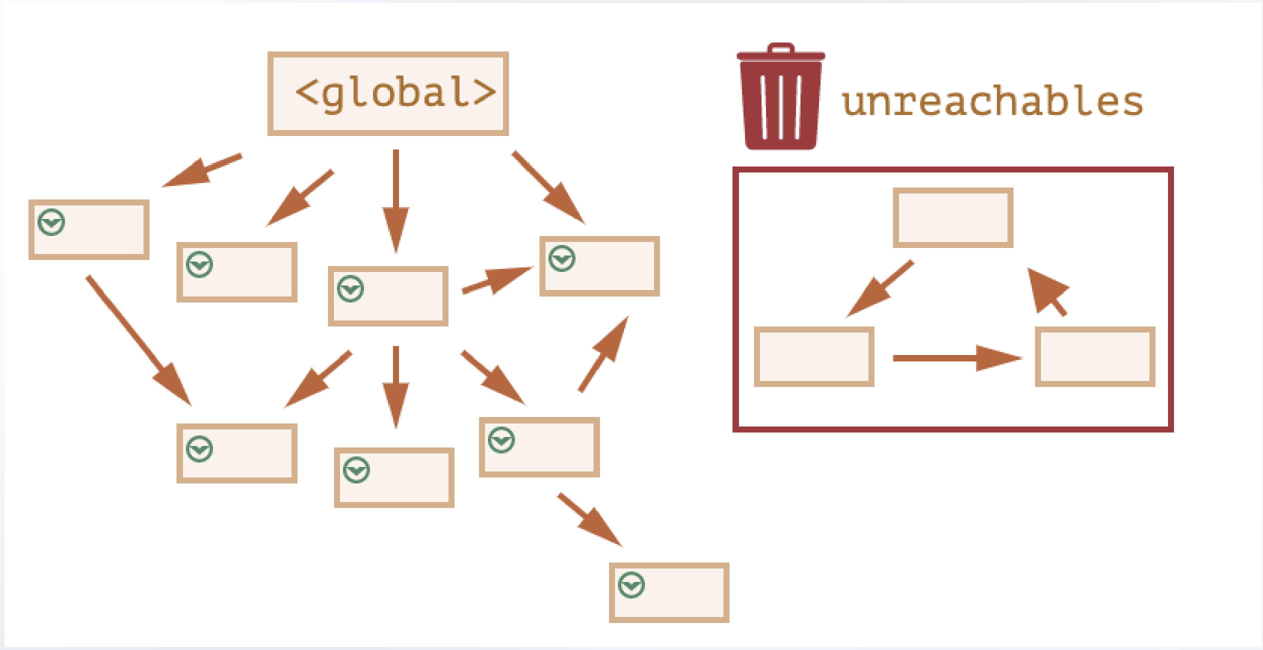
Garbage collection

```javascript
// user has a reference to the object
let user = {
  name: "John"
};
```



```javascript
user = null;
```





sourcemind

JavaScript Hoisting

Hoisting is JavaScript's default behavior of moving declarations to the top.

In JavaScript, a variable can be declared after it has been used.
In other words; a variable can be used before it has been declared.

```
console.log(a);
var a = 5;
```

This will result in a `ReferenceError`:

```
carName = "Volvo";
let carName;
```

sourcemind