Multi-paradigm and Meta-programming in the Software Engineering

Copyright © Timur Shemsedinov and HowProgrammingWorks contributors

Kiev, 2015-2022

Index

1. Introduction
   1.1. Approach to learn programming
   1.2. Modeling: abstractions and reuse
   1.3. Algorithm, program, syntax, language
   1.4. Decomposition and separation of concerns
   1.5. Software engineer speciality overview
   1.6. Programming paradigms overview
2. Basic concepts in JavaScript, Python and C syntax
   2.1. Value, identifier, variable and constant, literal, assignment
   2.2. Data types, scalar, reference and structured types
   2.3. Contexts and lexical scope
   2.4. Operator and expression, code block, function, loop, condition
   2.5. Procedural paradigm, call, stack and heap
   2.6. Higher-order function, pure function, side effects
   2.7. Closures, callbacks, wrappers, and events
   2.8. Exceptions and error handling
3. Application state, data structures and collections
   3.1. Stateful and stateless approach
   3.2. Structs and records
   3.3. Array, list, set, tuple
   3.4. Dictionaries, hash table and associative array
   3.5. Stack, queue, deque
   3.6. Trees and Graphs
   3.7. Dataset projections
4. Extended concepts
   4.1. Programming languages and paradigms, technology stack
   4.2. Development environment and debugging
   4.3. Iterations: recursion, iterators, and generators

- 4.4. Application building blocks: files, modules, components
- 4.5. Object, prototype and class
- 4.6. Partial application and currying, pipe and compose
- 4.7. Chaining for methods and functions
- 4.8. Mixins
- 4.9. Dependencies and libraries

5. Widespread programming paradigms
- 5.1. Imperative and declarative approach
- 5.2. Structured and non-structured programming
- 5.3. Procedural programming
- 5.4. Functional programming
- 5.5. Object-oriented programming
- 5.6. Prototype-based programming

6. Antipatterns
- 6.1. Common for all paradigms
- 6.2. Procedural antipatterns
- 6.3. Object-oriented antipatterns
- 6.4. Functional antipatterns

7. Development process
- 7.1. Refactoring
- 7.2. Code review
- 7.3. Computational complexity estimation
- 7.4. Testing: unittests, system and integration testing
- 7.5. Monomorphic code in dynamic languages
- 7.6. Multi-aspect optimizations

8. Advanced concepts
- 8.1. Events, Timers and EventEmitter
- 8.2. Introspection and reflection
- 8.3. Serialization and deserialization
- 8.4. Regular expressions

- 8.5. Memoization
- 8.6. Factory and Poll
- 8.7. Typed arrays
- 8.8. Projections
- 8.9. I/O and Files

9. Concurrent computing basics
    - 9.1. Asynchronous programming
    - 9.2. Parallel programming
    - 9.3. Actor-based programming
10. Advanced programming paradigms
    - 10.1. Generic programming
    - 10.2. Event-driven and reactive programming
    - 10.3. Automata-based programming and state machines
    - 10.4. Language-oriented programming and DSLs
    - 10.5. Data-flow pogramming
    - 10.6. Metaprogramming
    - 10.7. Metamodel dynamic interpretation
11. Databases and persistent storage
12. Software architecture
13. Distributed systems

Consider following:

```
const id = (x) => x;

// Usage

const res = id(5);
console.log({ res });
```