Ten Ideas in Programming

A Minimal Introduction

Kickstart Course

Based on work by Morten Misfeldt, Morten Aagaard Schultz & Karl Emil Kjær Bistrup

Adapted by Daniel Spikol

Why Learn These 10 Ideas?

Two Main Goals:

1. **Practical Starting Point** - Write your first code
2. **Understanding Programs** - Read and understand simple code

The 10 Core Ideas Overview

1. **Data** - Everything digital starts here
2. **Functions** - Actions that do things
3. **Variables** - Storage containers
4. **Sequential Processes** - Step-by-step actions
5. **Conditionals** - Making decisions
6. **Loops** - Repetition power
7. **Models** - Representing real-world ideas
8. **Abstraction** - Breaking down complexity
9. **Functional Thinking** - Organizing with functions
10. **Data Structures** - Organizing data efficiently

Idea 1: Data

Everything digital begins with data!

Data can be:

- Numbers (42, 3.14)
- Text ("Hello World!")
- Images 🖼️
- Sensor readings 📡

Data is stored in structures that computers can manipulate and humans can understand.

Idea 2: Functions

Like a "function machine" from math class!

A function:

- Performs an **action** when called
- Can take **inputs**
- Can provide **outputs**
- Does something specific

```
function sayHello(name) {
    return "Hello, " + name + "!";
}
```

Idea 3: Variables

Storage containers for your data

Variables:

- Store values
- Can be updated
- Have names you choose
- Enable dynamic behavior

```
let fishPosition = 100;
let eyeSize = 15;
let playerName = "Alex";
```

> *Key insight: You use variables to store data, and functions to process that data!*

Idea 4: Sequential Processes

Programs run step-by-step

Each step can:

- Add or subtract numbers
- Check conditions
- Call functions
- Transform data

```
Step 1: Get user input
Step 2: Process the data
Step 3: Display the result
```

Idea 5: Conditionals (if/then)

Making decisions in code

Programs need to handle different situations:

```
if (temperature > 30) {
    showMessage("It's hot! 🌞");
} else {
    showMessage("Nice weather! 😊");
}
```

Conditionals guide decision-making and help programs respond to different cases.

Idea 6: Loops (Iteration)

The power of repetition!

Loops let you:

- Repeat tasks multiple times
- Process lists of data
- Continue until a condition is met

```
for (let i = 0; i < 5; i++) {
    drawFish(i * 100);  // Draw 5 fish
}
```

Idea 7: Models

Representing real-world systems

Models help us:

- Simulate real processes
- Make predictions
- Test ideas safely
- Communicate complex concepts

Example: A weather simulation, a game character, or a budget calculator

Idea 8: Abstraction & Decomposition

Breaking down complexity

Two key skills:

1. **Decomposition** - Break big problems into smaller pieces
2. **Abstraction** - Hide unnecessary details

Instead of "draw a complex aquarium scene"

Break it down to: "draw fish" + "draw plants" + "draw bubbles"

Idea 9: Functional Thinking

Organizing code with functions

Think in terms of:

- **Inputs** - What information does it need?
- **Process** - What does it do?
- **Outputs** - What does it produce?

```
// Input: fishX position
// Process: Draw shapes
// Output: A fish on screen
function make_fish(fishX) { ... }
```

Idea 10: Data Structures

Organizing data efficiently

Common structures:

- **Lists/Arrays** - Ordered collections `[1, 2, 3]`
- **Objects** - Named properties `{name: "Fish", size: 10}`
- **Tables** - Rows and columns
- **Trees** - Hierarchical data

The right structure makes your program faster and easier to understand!

Five Key Practices

How We Actually Program

1. **Specify** - Explain what you want
2. **Test & Debug** - Find and fix mistakes
3. **Organize Thinking** - Data, Computation, Interaction
4. **Document & Reuse** - Comment and modularize
5. **Collaborate** - Work with users

Practice 1: Specification

Using Pseudocode

Write your program in plain English first!

```
FUNCTION make_fish with parameter fishX:
    Draw oval body at (fishX, 200)
    Draw triangle tail on the left
    Draw circle eye on the right
END FUNCTION
```

This helps you think through the logic before coding!

Practice 2: Test & Debug

Everyone makes mistakes - it's normal!

The debugging process:

1. Try something small first
2. Test if it works
3. Find what's wrong (isolate the problem)
4. Fix it
5. Test again

> *"Making mistakes and fixing them is a normal and important part of programming"*

Practice 3: Three Layers of Thinking

Organize programs into:

1. **Data Layer** - What information?
   - User's name: "Morten"
2. **Computation Layer** - What processing?
   - Extract first name from full name
3. **Interaction Layer** - What user sees?
   - Type name → See "Hello, Morten!"

Practice 4 & 5: Document & Collaborate

Make your code understandable!

Documentation:

```
// Draw the fish eye
// Position: 30 pixels right of center
eyeSize = 15;  // Standard eye size
```

Collaboration:

- Build for real people
- Watch how they use it
- Listen to their needs
- Improve based on feedback

Remember: Programs are for people!