Intro to Programming, Part 1

Richard Kelley

AI 109

Why Programming in an AI Course?

- AI systems do not replace programs
- Models run *inside* code
- Programming is how we:
  - Express ideas precisely
  - Automate reasoning
  - Control AI systems
- Today: *first contact*, not mastery

What Is a Program?

- A program is a set of **precise instructions**
- Computers are:
  - Fast
  - Literal
  - Unforgiving of ambiguity
- Same input -> same output
- No “understanding,” only execution

The Programming Environment

- Programs can run in different environments
- Common forms:
  - ***GUI* programs** (desktop and mobile apps)
    - **Browser programs** (web pages and web apps)
  - ***Terminal* programs** (command line tools, scripts)
- Same core ideas, different ways of interacting with the computer.
- Today we use a **browser program** to learn.
  - The ***computer language*** we will use is called ***JavaScript***.
  - I'm going to use the Brave browser, but this will also work with Chrome, Firefox.
  - This will also let you start building web programs, which will eventually use AI.

The Browser Console as a Programming Environment

- The browser includes a built-in programming tool called the ***console***.
  - No installation required.
  - Immediate feedback.
- This process is repeated until the user quits:
  - Type some code.
  - Run the code.
  - Observe the result.
- The technical terms for these steps are ***read***, ***evaluate***, ***print***.
- The technical term for “repeating” is a loop, so we call the console a “read-evaluate-print loop” or ***REPL***.

Comments

- We can write notes to ourselves that the computer won't try to execute.
- These are called ***comments***.
- In JavaScript there are two kinds:
  - Single line comments start with //
  - Multi-line comments start with /* and end with */

Expressions and Evaluation

- An ***expression*** produces a ***value.***
- The computer ***evaluates*** expressions to produce values.
- Examples:
  - Arithmetic: The *expression* 1 + 1 evaluates to the value 2.
  - “Logic” tests
    - && means “and”.
    - So the expression true && false evaluates to false.
  - Text manipulation
    - The expression “Hello, ” + “world!” evaluates to “Hello, world!”
- The console shows results directly.

Values Have *Types*

- Different kinds of values:
  - Numbers
  - Text (***strings***)
  - True / false (***booleans***)
- The computer treats them differently
- ***Type errors*** are common — and informative.
  - What should be the value of the expression `1 + “Hello!”` ?
  - What is it?

Types of Expressions

- Arithmetic Expressions
  - These evaluate to numbers.
- Boolean Expressions
  - These evaluate to ***true*** or ***false***.
- ***String*** expressions
  - These evaluate to ***strings*** – which are pieces of text.
- And more…

Arithmetic Expressions

- Addition
  - 5 + 4
- Subtraction
  - 5 - 3
- Multiplication
  - 5 * 3
- Division
  - 6 / 2
- ***Modulo*** (remainder).
  - 5 % 2 (which is 1).

Simple Boolean Expressions

- ***And***

Expression	Value
true && true	true
true && false	false
false && true	false
false && false	false

- ***Or***

Expression	Value
true || true	true
true || false	true
false || true	true
false || false	false

Other Boolean Expressions

- Strict Inequalities
  - 1 < 4
  - 1 > 4
  - 1 < 1
- “Weak” Inequalities (“less than or equal”)
  - 1 <= 5
  - 1 <= 1
- Equality
  - 1 === 2
  - true === false

Expressions vs. Statements

- Some code *produces values.*
  - Code that produces a value is an expression.
- Other code *performs actions.*
  - Code that performs an action is called a ***statement***.
- Understanding this explains console behavior.
- This distinction appears in all languages.
  - In JavaScript, it is typical (and recommended) to end simple statements with a **semicolon**.
  - But we don’t put a semicolon after curly braces.

Variables: Naming State

- ***Variables*** store values (they are like boxes for data). `let x = 5;`
- Variables have ***names***
  - Always start with a letter.
  - May otherwise contain letters, numbers, underscores.
- ***Assignment*** changes state over time
  - Assignment is a *statement* not an expression. `x = 6;`
- Key distinction:
  - Assignment ≠ equality. Equals sign is deceptive.
- Programs execute **top to bottom.**

Two Powerful Ideas

- Computers are dumb – at the lowest level they only follow rules encoded by computer programs.
- But they have two abilities that make them powerful:
  - ***Conditionals*** allow programs to “make decisions”
  - ***Loops*** allow programs to perform actions repeatedly.
- Because computers are so fast, conditionals and loops can let them do amazing things.

Conditionals: Making Decisions

```
if (<condition>) {
  console.log(“true”);
} else {
  console.log(“false”);
}
```

- Programs can ***branch***
- ***if / else statement*** chooses between options
- Decisions are based on boolean expressions
- This is how programs “decide”
- No understanding — just rules.
- The two ***branches*** of the statement are wrapped in curly braces.
  - We call them the ***if branch*** and the ***else branch***.

Loops: Repetition at Scale

- Computers excel at repetition
- ***Loops*** automate repeated work
- Loop variables change over time
- ***for loops*** are a kind of statement.

```
for (let i = 0; i < 5; i = i + 1) {
  console.log(i);
}
```

For Loops

- Parts of a loop
  - The keyword “for”
  - Parentheses
    - Code that runs once at the start of the loop
    - A test that is run at the end of every loop iteration
    - Code that runs after every test (usually to update something)
  - The body of the loop, surrounded by braces.

```
for (let i = 0; i < 5; i = i + 1) {
  console.log(i);
}
```

Arrays: Collections of Data

- ***Arrays*** store multiple values
- Ordered, indexed collections
  - Starting from 0.
- Core operations:
  - ***Access*** `arr[0]`
  - ***Length*** `arr.length`
  - ***Iteration***
- This is the foundation of data processing

```
let arr = [1,2,3];
```

Functions: Abstraction

- ***Functions*** package behavior
- Inputs -> outputs
- Functions let us:
  - Reuse logic
  - Hide details
  - Think at a higher level
- Black-box thinking is essential

```
function square(x) {
  return x * x;
}
```

Functions

- Functions have
  - A name
  - A set of ***arguments***
  - A ***body***
- Functions may have outputs.
  - The keyword ***return*** indicates an output.

```
function square(x) {
  return x * x;
}
```

Functions Have Contracts

- Functions expect certain inputs
- Incorrect inputs lead to errors or nonsense
- The computer does not “guess intent”
- This is why precision matters
- Reading function behavior is a skill

Errors Are Normal

- ***Syntax errors***: code is malformed
- ***Runtime errors***: something went wrong during execution
- ***Logic errors***: code runs but is wrong
- ***Debugging*** is:
  - Reading code
  - Testing code
  - Thinking about
- Humans write broken code
  - Mariner 1 Spacecraft, 1962 (missing hyphen, $192 million)
  - Pentium FDIV bug, 1994 (incorrect division operator, $1.1 billion)

What Programming Is Not

- Not memorizing syntax
- Not typing speed
- Not innate talent
- Not math-only
- It is structured thinking made executable

How This Fits the Course

- Programming enables:
  - Search
  - Machine learning pipelines
  - AI agents
- We start small
- Complexity builds gradually
- The skills compound

What Comes Next

- More practice with code
- More structure
- Data and ***algorithms***
- Using AI tools *with* understanding
- You are not expected to be fluent yet

If you’re curious to learn more…

https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Your_first_website