</>

Irene Learns to Code

A Complete Course Workbook — 7 Modules · 35 Slides

KEEP GOING 💪

Made with care, one slide at a time

What's Inside

Seven modules, in order, from "what is a computer" all the way to writing your first real program.

LESSON 1 · SLIDE 1

Hey Irene 👋

Before we write a single line of code, let's slow down and understand what a computer actually *is*. Not the marketing version — the real, simple version.

No pressure here. If a sentence doesn't click, read it twice. That's completely normal, even for people who've been coding for years.

LESSON 1 · SLIDE 2

Hardware vs Software

Think of a computer like a body. **Hardware** is the physical body — the brain (called the CPU), the memory (RAM), the storage (hard drive), the screen, the keyboard.

Software is the mind — the instructions telling that body what to actually do. Without software, hardware is just an expensive paperweight.

LESSON 1 · SLIDE 3

It's All Just On and Off

Here's the wild part: deep down, a computer only understands two things — **on** and **off**. We call these 1s and 0s.

Everything you've ever seen on a screen — photos, music, this very page — is built from billions of tiny on/off switches flipping, incredibly fast. Programming is how we control that flipping without doing it by hand.

LESSON 1 · SLIDE 4

Input → Process → Output

Every single thing a computer does follows one pattern, like cooking a meal:

Input (the ingredients) → **Process** (the cooking) → **Output** (the meal on your plate).

You type a message (input) → the computer processes it → it appears on your friend's screen (output). That's it. That's the whole game, repeated in a million different ways.

💡 **Quick note:** Keep this pattern in your back pocket — you'll spot it everywhere once you start coding.

LESSON 1 · SLIDE 5

Quick Recap ✔

You now know: hardware is the body, software is the mind, everything reduces to on/off switches, and every task follows Input → Process → Output.

That's genuinely most of what you need to stop feeling intimidated by "computer stuff." Next up: what programming actually means.

LESSON 2 · SLIDE 1

Giving Instructions

Programming is simply writing a precise list of instructions for a computer to follow, step by step. That's the whole definition — everything else is detail.

A "program" is just that list, saved so the computer can run it whenever you want.

LESSON 2 · SLIDE 2

Why We Need Languages

Computers only truly understand 1s and 0s — that's exhausting for humans to write directly. So we invented **programming languages**: friendlier, more human ways to write instructions, which then get translated into 1s and 0s behind the scenes.

Think of a programming language as a translator standing between you and the machine.

LESSON 2 · SLIDE 3

So Many Languages, Why?

You've probably heard names like Python, JavaScript, or Java. They're all just different "dialects" for talking to computers — some are better for websites, some for apps, some for data.

We're going to think in **Python** for these examples, because it reads almost like plain English and is one of the friendliest languages to start with.

LESSON 2 · SLIDE 4

Computers Are Very Literal

Imagine giving directions to a robot friend who does *exactly* what you say, with zero common sense. If you say "turn left" and forget to say "stop after 10 steps," it just keeps turning left forever.

That's what coding often feels like at first — being extremely precise with a very obedient, very literal helper.

💡 **Quick note:** This is why bugs happen: not because you're bad at this, but because computers do exactly what you typed, not what you meant.

LESSON 2 · SLIDE 5

Quick Recap ✔

Programming = giving precise instructions. Languages = the translator between you and the machine. We'll think in Python because it's readable. And computers need exact instructions, not vague ones.

Next: the very first building block of real code — variables.

LESSON 3 · SLIDE 1

What's a Variable?

A **variable** is just a labeled box that stores a piece of information, so the computer can use it — or change it — later.

Imagine sticking a label that says "name" on a box, and putting the word "Irene" inside. Now, any time your program needs that name, it just looks in the "name" box.

LESSON 3 · SLIDE 2

Different Kinds of Data

Boxes can hold different **types** of things:

🔢 **Numbers** — like your age.
🔤 **Text (called a "string")** — like your name.
✅ **True/False (called a "boolean")** — like "is it raining? yes or no."

The computer treats each type a little differently, the same way you'd treat a fragile box differently from a box of books.

LESSON 3 · SLIDE 3

What This Looks Like in Code

Here's actual Python — read it out loud, it's almost sentence-like:

```
name = "Irene"
age = 19
is_learning_to_code = True
```

💡 **Quick note:** "name" now holds the text "Irene". "age" holds the number 19. "is_learning_to_code" holds True — because, well, you are.

LESSON 3 · SLIDE 4

Why Bother With Variables?

Without variables, you'd have to retype "Irene" every single time your program needed her name. With a variable, you set it once and reuse it everywhere — and if it ever needs to change, you only change it in one place.

Variables are how a computer "remembers" anything at all.

LESSON 3 · SLIDE 5

Quick Recap ✔

Variables are labeled boxes for storing information. They can hold numbers, text, or true/false values. They let your program remember and reuse things instead of repeating itself.

Next: teaching your program to make decisions.

LESSON 4 · SLIDE 1

Meet the Traffic Light

Real life is full of decisions: *if* it's raining, take an umbrella. *Otherwise*, don't bother. Code makes decisions the exact same way, using something called a **conditional**.

Think of it like a traffic light: green means go do this, red means do something else instead.

LESSON 4 · SLIDE 2

If / Else in Code

Here's that umbrella idea, written in Python:

```
is_raining = True

if is_raining:
    print("Take an umbrella!")
else:
    print("Enjoy the sunshine!")
```

💡 **Quick note:** Read "if" as "in the case that" and "else" as "otherwise." That's genuinely all it means.

LESSON 4 · SLIDE 3

Comparing Things

Decisions usually involve comparing values. Some comparisons you'll use constantly:

`==` means "is equal to"
`>` means "is greater than"
`<` means "is less than"

Example: `age >= 18` means "is age 18 or older?" — it answers True or False, and your program acts on that answer.

LESSON 4 · SLIDE 4

Stacking Decisions

Sometimes there's more than two paths — like a road with several signs, not just one light. Python lets you stack conditions with `elif` (short for "else if"):

```
score = 75

if score >= 90:
    print("Grade: A")
elif score >= 70:
    print("Grade: B")
else:
    print("Keep practicing!")
```

LESSON 4 · SLIDE 5

Quick Recap ✔

Conditionals let your program choose between paths, just like a traffic light. "if" starts the check, "elif" adds more options, "else" catches everything left over.

Next: getting a computer to repeat something without you typing it a hundred times.

LESSON 5 · SLIDE 1

The Washing Machine Idea

A washing machine doesn't wash one sock at a time and stop — it repeats the same cycle until the whole load is done. That's exactly what a **loop** does in code: repeat an action until some condition is met.

LESSON 5 · SLIDE 2

"for" Loops — Repeat a Set Number of Times

Use a `for` loop when you know how many times you want something to happen — like counting from 1 to 5:

```
for number in range(1, 6):
    print(number)
```

💡 **Quick note:** This prints 1, 2, 3, 4, 5 — one per line — without you writing "print" five separate times.

LESSON 5 · SLIDE 3

"while" Loops — Repeat Until Something Changes

Use a `while` loop when you don't know exactly how many times — you just want to keep going *while* something is true. Like watering a plant while the soil is still dry:

```
soil_is_dry = True

while soil_is_dry:
    print("Watering...")
    soil_is_dry = False  # pretend it's wet now
```

LESSON 5 · SLIDE 4

A Real-World Example

Say you want to send the same birthday message to 10 friends. Instead of writing the "send message" code 10 separate times, a loop does it 10 times for you, automatically — same idea, way less typing, way fewer mistakes.

LESSON 5 · SLIDE 5

Quick Recap ✔

Loops repeat actions so you don't have to. Use "for" when you know the number of repeats, "while" when you're repeating until a condition changes.

Next: packaging up your code so you can reuse it anywhere.

LESSON 6 · SLIDE 1

A Recipe Card You Can Reuse

A **function** is a named, reusable block of instructions — like a recipe card you write once and pull out whenever you need it, instead of re-explaining the recipe from scratch every time.

LESSON 6 · SLIDE 2

Defining a Function

Here's a simple function that greets someone by name:

```
def greet(name):
    return "Hello, " + name + "!"
```

💡 **Quick note:** "def" means "define." "name" here is a placeholder — it becomes whatever you give it when you actually use the function.

LESSON 6 · SLIDE 3

Calling It With Different Inputs

Once defined, you can reuse that same function with any name you like:

```
print(greet("Irene"))
print(greet("Kwame"))

# Output:
# Hello, Irene!
# Hello, Kwame!
```

LESSON 6 · SLIDE 4

Why Functions Matter

Functions keep your code organized and stop you from repeating the same lines over and over. As programs grow bigger, functions are what keep them from turning into a tangled mess — each one handles one clear job.

LESSON 6 · SLIDE 5

Quick Recap ✔

Functions are reusable blocks of instructions you define once and call whenever you need them, with different inputs each time.

You now know variables, decisions, loops, and functions — the four real building blocks of programming. Last stop: putting them all together.

LESSON 7 · SLIDE 1

Let's Build Something

You now have all four building blocks: variables, decisions, loops, and functions. Real programs are just these four things, combined creatively. Let's build a tiny "Guess the Number" idea, in plain English first.

LESSON 7 · SLIDE 2

Plan It in Plain English First

Before touching code, always describe the steps in normal words:

1. The computer secretly picks a number.
2. It asks the player to guess.
3. If the guess is too high or too low, it says so.
4. It keeps asking until the guess is correct.

Notice: step 3 is a *decision*, and step 4 is a *loop*. You already know both.

LESSON 7 · SLIDE 3

Now, the Code

Here's roughly how that plan becomes Python — don't worry about memorizing it, just notice the pieces you already recognize:

```
secret_number = 7
guess = 0

while guess != secret_number:
    guess = int(input("Guess a number: "))
    if guess < secret_number:
        print("Too low!")
    elif guess > secret_number:
        print("Too high!")
    else:
        print("You got it!")
```

💡 **Quick note:** See it? "while" is the loop, "if / elif / else" is the decision, and "guess" / "secret_number" are variables. Nothing new — just combined.

LESSON 7 · SLIDE 4

It Won't Be Perfect — And That's Fine

Every programmer, no matter how experienced, writes code that breaks the first time. That's not failure, that's the actual process — write something, run it, see what breaks, fix it, repeat.

The programmers who seem "naturally good at this" have just made peace with that cycle. You're allowed to, too.

LESSON 7 · SLIDE 5

You Did It 🎉

Seriously — take that in. You now understand what a computer is, what programming means, how to store information, how programs make decisions, how they repeat tasks, how to package logic into functions, and how all of it combines into a real program.

That's not "the basics" — that's the actual foundation everything else in programming is built on. Whenever you're ready, try running real Python at **python.org** or a free site like **replit.com** — no installation needed. Go build something, Irene.

You Did It 🎉

You now understand what a computer is, what programming means, how to store information, how programs make decisions, how they repeat tasks, how to package logic into functions — and how it all combines into a real program.

Whenever you're ready to actually run this code, try **python.org** or a free site like **replit.com** — no installation needed.

#KEEPGOING