Programming environments: Python, notebooks, SDK choices

Set up a reproducible local environment, then verify a phase-sensitive first circuit.

Core 22 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

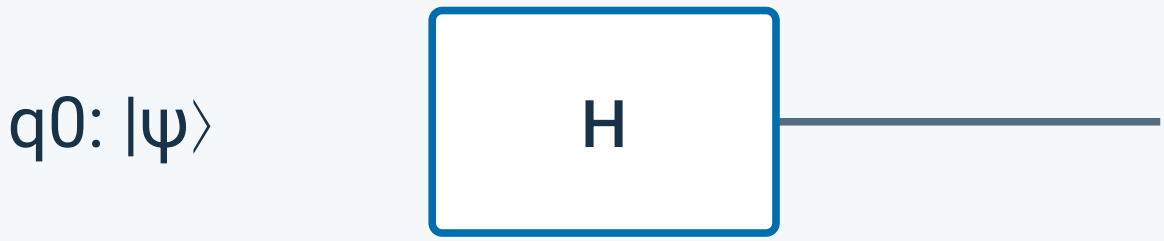
- Create an isolated, pinned Python environment.
- Read amplitudes before measurement.
- Record enough information to reproduce a run.

Begin with a clear question

A quantum program is ordinary software that builds an experiment. Python assembles gates, a simulator or processor executes them, and classical code checks the results. You can begin with a laptop and a tiny circuit. The first goal is a result you can explain and reproduce.

Build the reasoning

Use a separate virtual environment for this book. The notebook bundle includes exact direct dependencies and the full lock from its tested Python 3.13.2 environment. Install from the supplied files, select that environment as your notebook kernel and run the first notebook from top to bottom. An installation is ready when package checks and the notebook assertions pass. Keep a record of the Python and package versions; a notebook file alone does not describe its execution environment.



Follow time from left to right. Qiskit statevector labels put q0 on the right; diagrams put q0 on the top wire.

Worked example

The Hadamard gate applied to zero produces two positive amplitudes. Applied to one, it produces a negative amplitude for one. Their Z probabilities match, while their phases differ. Checking amplitudes makes this difference visible before a histogram can hide it. The code below is a complete local example in the pinned environment.

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

```
import numpy as np
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector

zero_input = QuantumCircuit(1)
zero_input.h(0)
one_input = QuantumCircuit(1)
one_input.x(0)
one_input.h(0)
plus = Statevector.from_instruction(zero_input)
minus = Statevector.from_instruction(one_input)
assert np.allclose(plus.data, [1, 1] / np.sqrt(2))
assert np.allclose(minus.data, [1, -1] / np.sqrt(2))
assert np.allclose(plus.probabilities(), minus.probabilities())
print(plus.data, minus.data)
```

Practical lab

Keep source notebooks free of output when you want to predict first; execute a copy to retain evidence. Use a seed for sampled results and tolerances for floating-point arithmetic. The statevector calculation above needs neither credentials nor a remote account. Notebook 08 later introduces an explicit, optional hardware handoff after local validation.

Run the corresponding notebook from the software practice collection:

- Download Circuits, amplitudes and the phase you cannot see in a histogram

Keep a short execution record: predicted result, observed result, environment, inputs and the check that would reject a plausible defect. Explain a failure before changing the expected value.

Chapter summary

- Isolate and record the execution environment.
- Use amplitudes to check phase-sensitive behaviour.
- Run the local assertions before extending the experiment.

Exercises

1. Replace H by Y on the zero input. Predict the complex amplitude for one.
2. Change one part of the worked program to create a plausible fault. State which assertion should fail and why.

> Worked answers
>
> 1. Y|0⟩ = i|1⟩, so the second amplitude is i and the probability of one is 1.
> 2. A useful answer identifies a specific changed gate, parameter, mapping or model and links it to an independent expected behaviour. Keep the intended specification fixed while changing the implementation; save the failing case as a regression test.

Check your understanding

PAUSE & THINK

Two notebooks produce the same Z histogram. What should you check before calling their states equivalent?

- ☐ The complex amplitudes, allowing a common global phase
- ☐ Only the notebook filenames
- ☐ Only whether the bars have the same colour

Answer: The complex amplitudes, allowing a common global phase. Compare relative phases as well as magnitudes.

Sources

IBM's bit-ordering guide, Statevector API and Operator API describe the SDK conventions and equivalence tools. The notebook bundle records its pinned environment and local execution evidence.