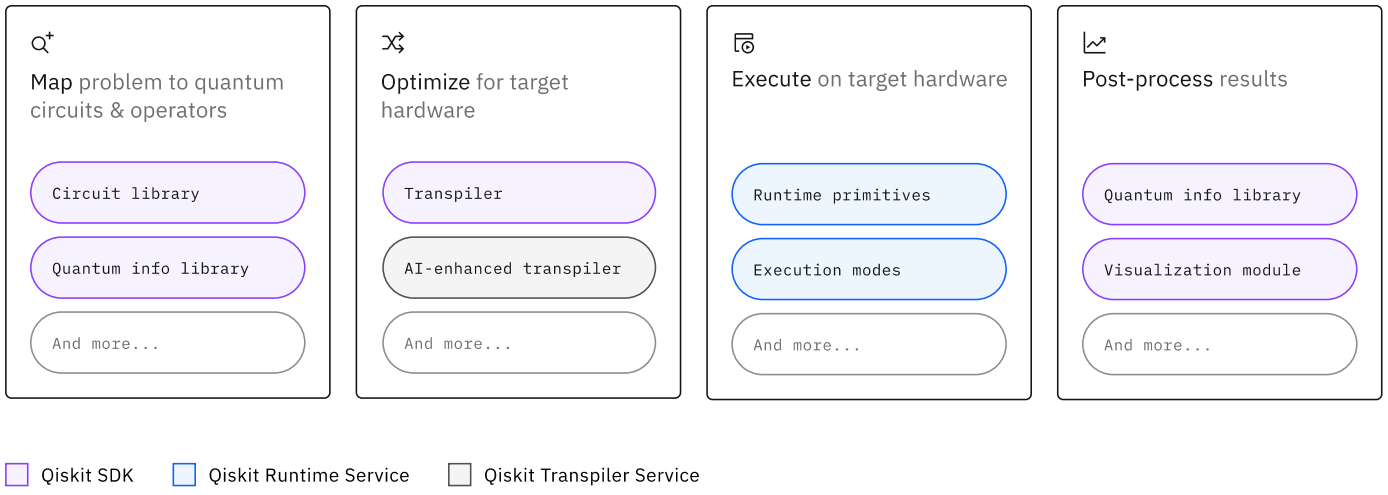
Introduction to Qiskit

The name "Qiskit" is a general term referring to a collection of software for executing programs on quantum computers. Most notably among these software tools is the open-source Qiskit SDK, and the runtime environment (accessed using Qiskit Runtime) through which you can execute workloads on IBM® quantum processing units (QPUs). As quantum technology evolves, so does Qiskit, with new capabilities released every year that expand this core collection of quantum software.

In addition, many open-source projects are part of the broader Qiskit ecosystem. These software tools are not part of Qiskit itself, but rather interface with Qiskit and can provide valuable additional functionality.



IBM is committed to the responsible development of quantum computing. Learn more and review our responsible quantum principles in the Responsible quantum computing topic.

The Qiskit SDK

The Qiskit SDK (package name qiskit) is an open-source SDK for working with quantum computers at the level of extended (static, dynamic, and scheduled) quantum circuits, operators, and primitives. This library is the core component of Qiskit; it is the largest package under the Qiskit name with the broadest suite of tools for quantum computation, and many other components interface with it.

Some of the most useful features of the Qiskit SDK include:

- **Circuit-building tools** (qiskit.circuit) - For initializing and manipulating registers, circuits, instructions, gates, parameters, and control flow objects.
- **Circuit library** (qiskit.circuit.library) - A vast range of circuits, instructions, and gates - key building blocks for circuit-based quantum computations.
- **Quantum info library** (qiskit.quantum_info) - A toolkit for working with quantum states, operators and channels, using exact calculations (no sampling noise). Use this module to specify input observables and analyze fidelity of outputs from primitives queries.
- **Transpiler** (qiskit.transpiler) - For transforming and adapting quantum circuits to suit specific device topology, and optimizing for execution on real quantum processing units (QPUs).
- **Primitives** (qiskit.primitives) - The module that contains the base definitions and reference implementations of the Sampler and Estimator primitives, from which different quantum hardware providers can derive their own implementations. See more information about the Qiskit Runtime primitives in the documentation.

Benchmarking and the Benchpress package

Benchmarking is important for comparing the relative performance of quantum software across different stages of a development workflow. Benchmarking tests for quantum software might, for example, look at the speed and quality of building, manipulating, and transpiling circuits. IBM Quantum is committed to delivering the most performant SDK possible, and to that end, the Qiskit SDK is benchmarked using over 1,000 tests developed by leading universities, national labs, and researchers at IBM. The benchmarking suite used for these tests, named Benchpress, is now available as an open-source package. You can now use the Benchpress package to perform your own analysis of quantum SDK performance.

Qiskit Runtime

Qiskit Runtime is a cloud-based service for executing quantum computations on IBM Quantum™ hardware. The qiskit-ibm-runtime package is a client for that service, and is the successor to the Qiskit IBM Provider. The Qiskit Runtime service streamlines quantum computations and provides optimal implementations of the Qiskit primitives for IBM Quantum hardware. To get started with Qiskit Runtime primitives, visit the documentation.

With Qiskit Runtime you can choose to run your quantum programs on IBM Quantum hardware through the IBM Quantum Platform or IBM Cloud®. See more information on selecting an IBM Quantum channel in the documentation.

Qiskit Runtime is designed to use additional classical and quantum compute resources, including techniques such as error suppression and error mitigation, to return a higher-quality result from executing quantum circuits on quantum processors. Examples include dynamical decoupling for error

suppression, and readout mitigation and zero-noise extrapolation (ZNE) for error mitigation. Learn how to configure these options on the Configure error mitigation page.

Qiskit Runtime also includes three types of execution modes for running your quantum program on IBM hardware: *Job*, *Session*, and *Batch*, each of which have different use cases and implications for the quantum job queue. A Job is a single query to a primitive that can be run over a specified number of shots. Sessions allow you to efficiently run multiple jobs in iterative workloads on quantum computers. Batch mode allows you to submit all your jobs at once for parallel processing.

Is Qiskit Runtime open-source?

The short answer is, *not all of it*. The Qiskit Runtime service software that handles the technicalities of running your quantum program on an IBM Quantum device (including any error mitigation and suppression) is **not** open-source. However, the Qiskit Runtime client (the interface for users to access the Qiskit Runtime service), the Qiskit SDK running on the server side, and some of the software used for error mitigation, **are** open-source. To get involved with the Qiskit open-source efforts, visit our GitHub organization at github.com/Qiskit and github.com/Qiskit-Extensions.

Qiskit Serverless

Creating utility-scale quantum applications generally requires a variety of compute resource requirements. Qiskit Serverless (qiskit-ibm-catalog.QiskitServerless) provides a simple interface to run workloads across quantum-classical resources. This includes deploying programs to IBM Quantum Platform and running workloads remotely, as well as easy resource management for multi-cloud and quantum-centric supercomputing use cases. See more information in the Qiskit Serverless documentation about how to use this collection of tools to:

- Parallelize classical tasks, such as pre-processing and post-processing
- Persist long-running workloads in the cloud, even if your laptop is turned off
- Deploy reusable programs in the cloud

Qiskit Functions

Qiskit Functions (qiskit-ibm-catalog.QiskitFunctionsCatalog) are abstracted services designed to accelerate algorithm discovery and application prototyping. Explore the Qiskit Functions Catalog, including:

- **Circuit functions**: Services that include transpilation, error suppression, error mitigation, and post-processing techniques that take abstract circuits and desired measurement observables as input. With Circuit functions, users can discover new algorithms and applications without needing to manage transpilation or quantum hardware performance.

- **Application functions**: Services that include entire quantum workflows, from mapping classical to quantum, optimizing for hardware, execution on hardware, and post-processing. Users can prototype industry applications with domain-familiar inputs and outputs.

Premium Plan members can access IBM-provided functions right away, or purchase licenses for the partner-provided functions directly from those partners.

Qiskit Transpiler as a Service

The Qiskit Transpiler Service (package name qiskit-ibm-transpiler) is a new experimental service that provides remote transpilation capabilities on the cloud to IBM Quantum Premium Plan users. In addition to the local Qiskit SDK transpiler capabilities, your transpilation tasks can benefit from both IBM Quantum cloud resources and AI-powered transpiler passes using this service. To learn more about how to integrate cloud-based transpilation into your Qiskit workflow you can check out the documentation.

Qiskit addons

Qiskit addons are a collection of research capabilities for utility-scale algorithm discovery. These capabilities build upon Qiskit's performant foundation of tools for creating and running quantum algorithms. Addons are modular software components that plug into a workflow to scale or design new quantum algorithms. To learn more about the set of available Qiskit addons and how to get started using them, visit the documentation.

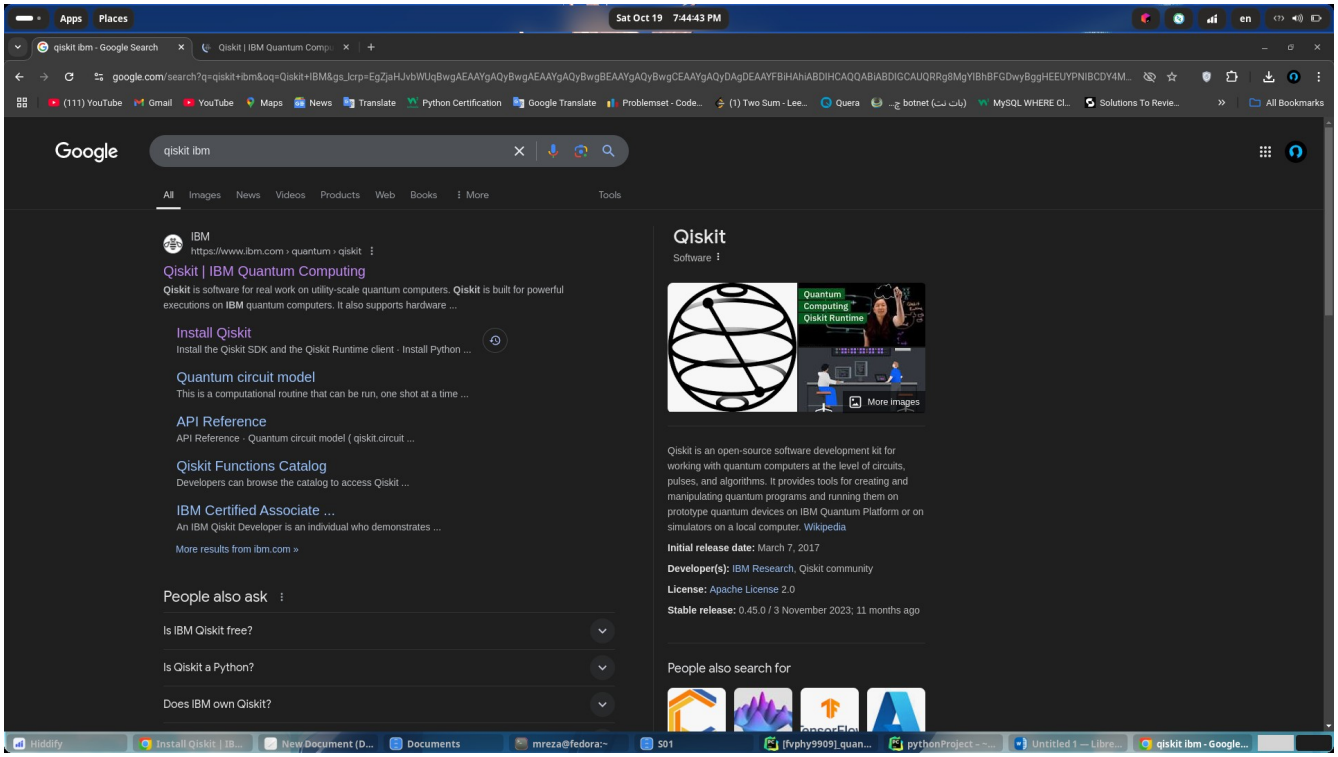
The Qiskit ecosystem

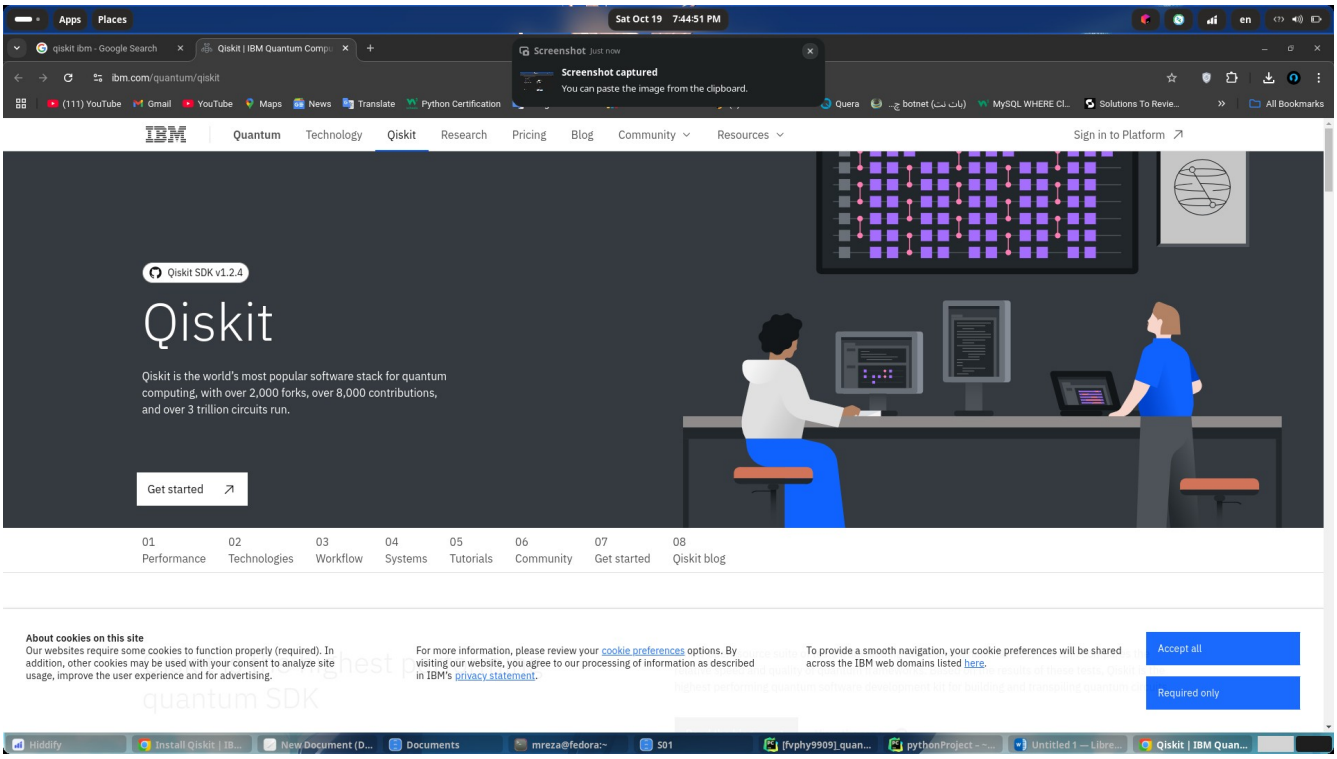
Beyond Qiskit there are many open-source projects that use the "Qiskit" name but are not part of Qiskit itself; rather, they interface with Qiskit and can provide valuable additional functionality to supplement the core Qiskit workflow. Some of these projects are maintained by IBM Quantum teams, whereas others are supported by the broader open-source community. The Qiskit SDK is designed in a modular, extensible way to make it easy for developers to create projects like these that extend its capabilities.

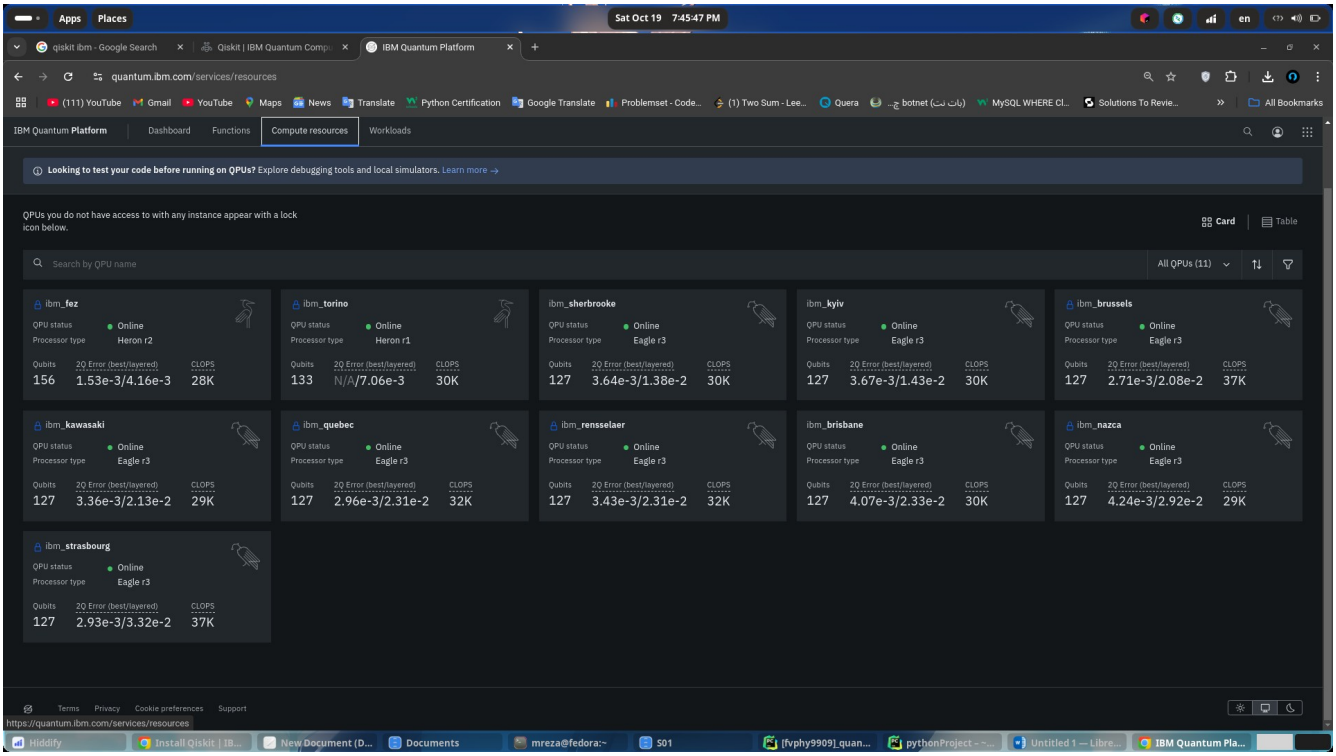
Some popular projects in the Qiskit ecosystem include:

- **Qiskit Aer** (qiskit-aer) - a package for quantum computing simulators with realistic noise models. It provides interfaces to run quantum circuits with or without noise using multiple different simulation methods. Maintained by IBM Quantum.
- **qBraid SDK** (qbraid) - a platform-agnostic quantum runtime framework for both quantum software and hardware providers, designed to streamline the full lifecycle management of quantum jobs—from defining program specifications to job submission and through to the post-processing and visualization of results. Maintained by qBraid.

- **mthree** mthree - a package for implementing M3 (Matrix-free Measurement Mitigation), a measurement mitigation technique that solves for corrected measurement probabilities using a dimensionality reduction step followed by either direct LU factorization or a preconditioned iterative method that nominally converges in O(1) steps, and can be computed in parallel. Maintained by IBM Quantum.







When you log in to the platform and create an account using an email, after creating the account, you set a password for yourself. Then, based on the IBM email, an authentication process is sent to you, and your email is verified. Finally, after email verification, you are given a token and an ID, which you can use to work with IBM quantum machines.

Install Pyhton--->

for linux:

python3 -m venv /path/to/virtual/environment

source /path/to/virtual/environment/bin/activate

Install Qiskit--->

pip install qiskit
pip install qiskit-ibm-runtime

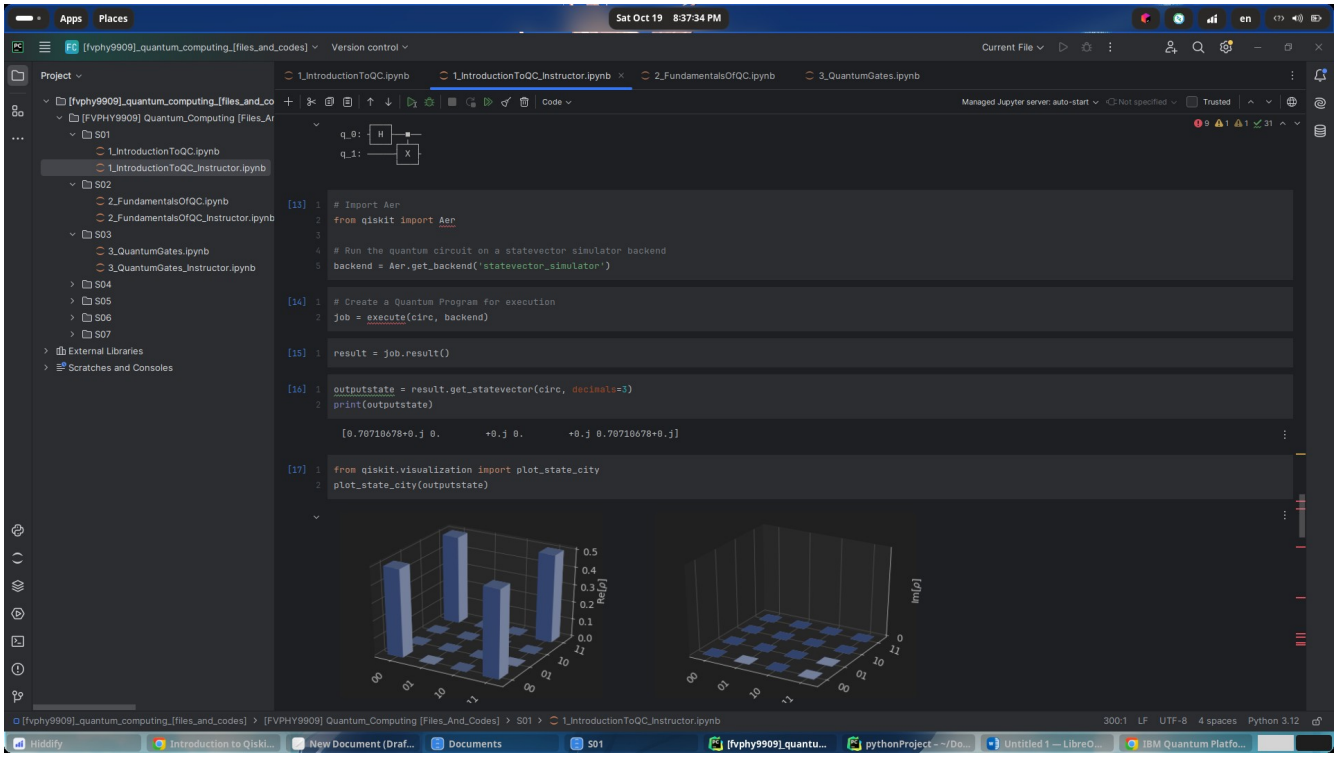
Install jupyter noteBook if You want

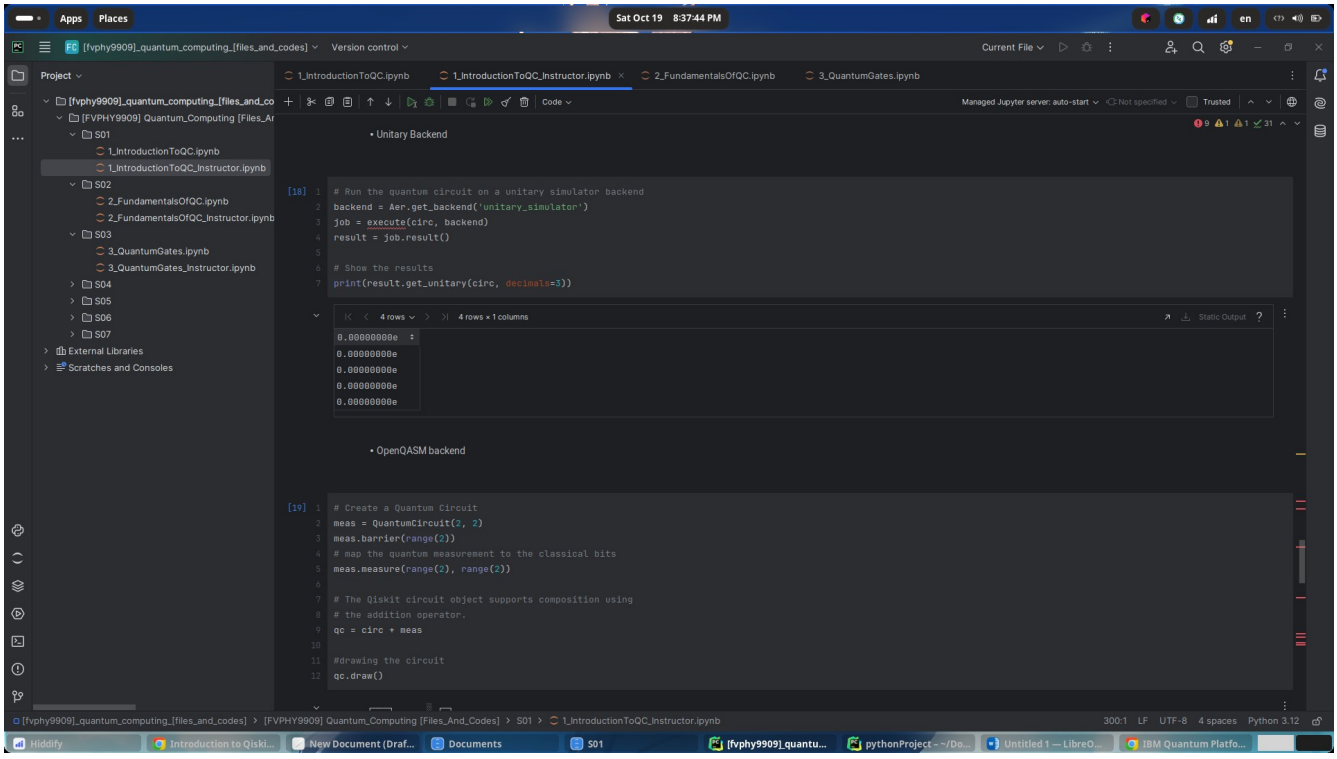
pip install jupyter

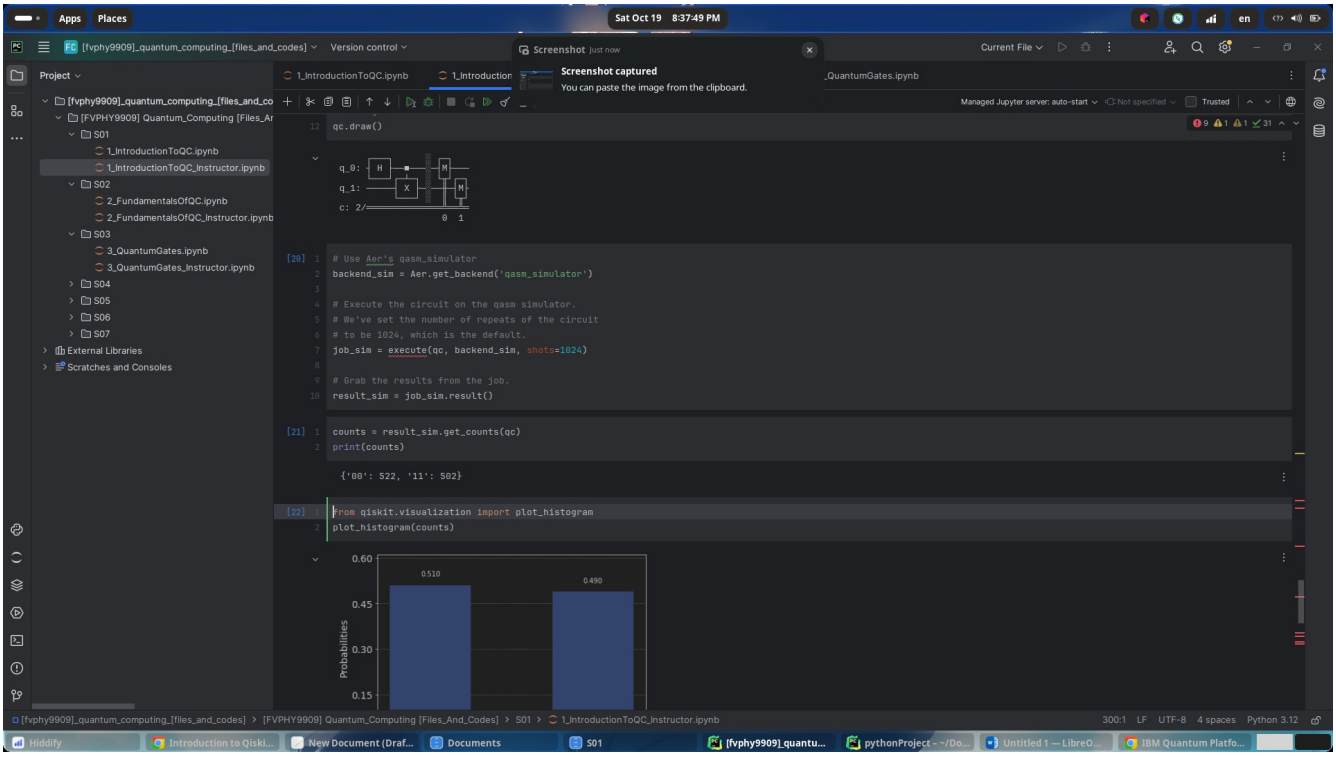
Then open your notebook as follows:

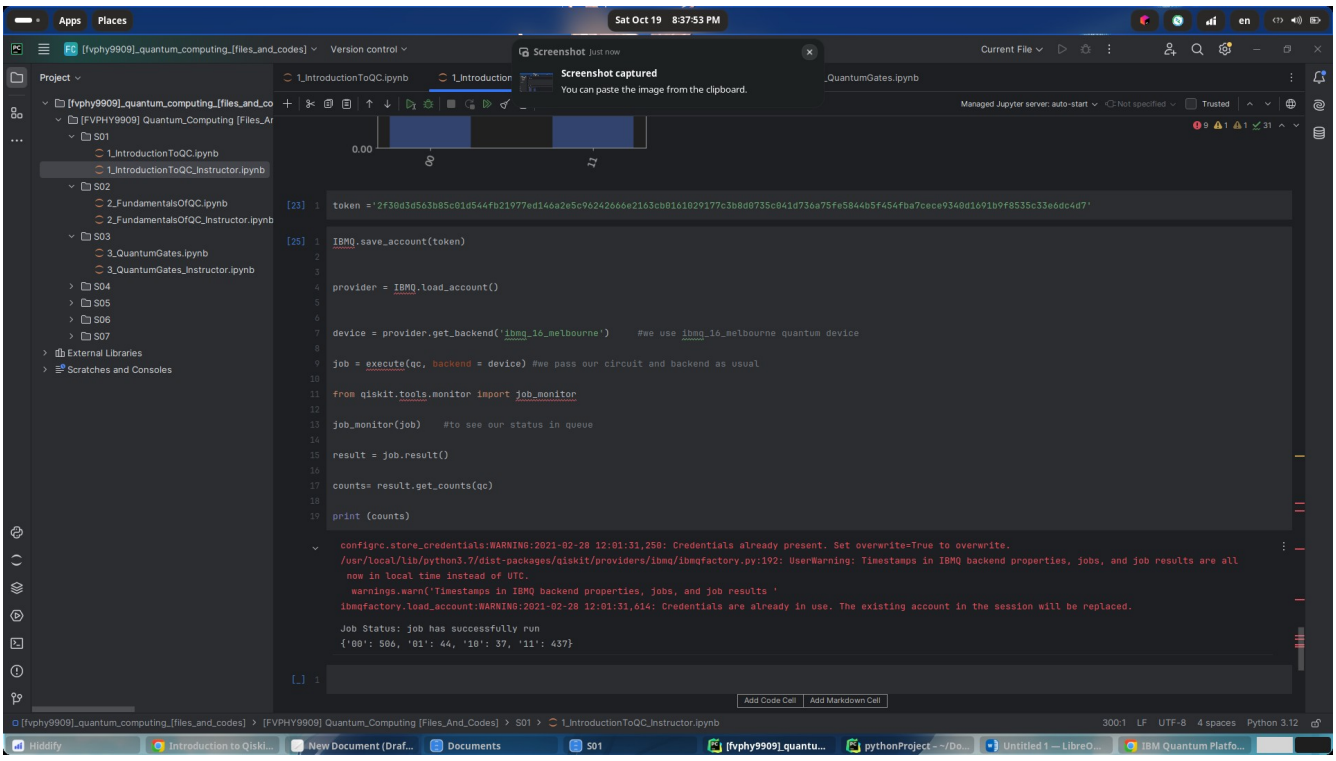
jupyter notebook path/to/notebook.ipynb

Take an Example:






Apps
Places
Sat Oct 19 8:37:49 PM
Screenshot captured
You can paste the image from the clipboard.
qc.draw()
q_0:
q_1:
c: 2/
# Use Aer's qasm_simulator
backend_sim = Aer.get_backend('qasm_simulator')
# Execute the circuit on the qasm simulator.
# We've set the number of repeats of the circuit
# to be 1024, which is the default.
job_sim = execute(qc, backend_sim, shots=1024)
# Grab the results from the job.
result_sim = job_sim.result()
counts = result_sim.get_counts(qc)
print(counts)
{'00': 522, '11': 502}
from qiskit.visualization import plot_histogram
plot_histogram(counts)
0.510
0.490
Probabilities
0.60
0.45
0.30
0.15


Apps
Places
Sat Oct 19 8:37:53 PM
Screenshot captured
You can paste the image from the clipboard.
0.00
token ='2f30d3d563b85c01d544fb21977ed146a2e5c962426666e2163cb0161029177c3b8d0735c041d736a75fe5844b5f454fba7cece9340d1691b9f8535c33e6dc4d7'
IBMQ.save_account(token)
provider = IBMQ.load_account()
device = provider.get_backend('ibmq_16_melbourne')
job = execute(qc, backend = device)
from qiskit.tools.monitor import job_monitor
job_monitor(job)
result = job.result()
counts= result.get_counts(qc)
print (counts)
configrc.store_credentials:WARNING:2021-02-28 12:01:31,250: Credentials already present. Set overwrite=True to overwrite.
/usr/local/lib/python3.7/dist-packages/qiskit/providers/ibmq/ibmqfactory.py:192: UserWarning: Timestamps in IBMQ backend properties, jobs, and job results are all
now in local time instead of UTC.
warnings.warn('Timestamps in IBMQ backend properties, jobs, and job results '
ibmqfactory.load_account:WARNING:2021-02-28 12:01:31,614: Credentials are already in use. The existing account in the session will be replaced.
Job Status: job has successfully run
{'00': 506, '01': 44, '10': 37, '11': 437}
Add Code Cell
Add Markdown Cell