# Qiskit

Elements for building a quantum future

Igor Sokolov
IBM Research-Zurich

# Installation summary

https://qiskit.org/documentation/install.htmls

1. Python 3.7
2. PyCharm (Optional, any IDE would work)
3. Visual Studio for Windows users
4. Anaconda, then in Terminal:
    1. conda create –n qiskit python=3.7
    2. source activate qiskit or activate qiskit (on Windows)
5. Terminal: pip install qiskit
6. Terminal: pip install matplotlib (plot data)
7. Terminal: pip install jupyter
8. Terminal: source activate qiskit
9. Terminal: python -m ipykernel install --user --name qiskit --display-name ”qiskit”

This presentation on: https://github.com/Brogis1/2020_SCS_Qiskit_Workshop

# Installing Qiskit

## Requirements

https://www.python.org/downloads/

Qiskit supports Python 3.5 or later. However, both Python and Qiskit are evolving ecosystems, and sometimes when new releases occur in one or the other, there can be problems with compatibility.

If you're having trouble installing or using Qiskit after updating Python, check the Qiskit Package metadata for **Programming Language** to see which specific versions of Python are supported.

We recommend installing Anaconda, a cross-platform Python distribution for scientific computing. Jupyter, included in Anaconda, is recommended for interacting with Qiskit.

https://www.anaconda.com/distribution/

Qiskit is tested and supported on the following 64-bit systems:

- Ubuntu 16.04 or later
- macOS 10.12.6 or later
- Windows 7 or later

Using Qiskit on Windows requires VC++ runtime components. We recommend one of the following:

- Microsoft Visual C++ Redistributable for Visual Studio 2017
- Microsoft Visual C++ Redistributable for Visual Studio 2015

# Install

We recommend using Python virtual environments to cleanly separate Qiskit from other applications and improve your experience.

The simplest way to use environments is by using the `conda` command, included with Anaconda. A Conda environment allows you to specify a specific version of Python and set of libraries. Open a terminal window in the directory where you want to work.

Create a minimal environment with only Python installed in it.

```
conda create -n name_of_my_env python=3
```

```
source activate name_of_my_env
```

Or, if you're using Windows

1. Install Anaconda
2. Search for Anaconda Prompt
3. Open Anaconda Prompt

Use the following commands

```
conda create -n name_of_my_env python=3
```

```
activate name_of_my_env
```

Next, install the Qiskit package, which includes Terra, Aer, Ignis, and Aqua.

```
pip install qiskit
```

**Note**

Starting with Qiskit 0.13.0 pip 19 or newer is needed to install qiskit-aer from precompiled binary on Linux. If you do not have pip 19 installed you can run `pip install -U pip` to upgrade it. Without pip 19 or newer this command will attempt to install qiskit-aer from sdist (source distribution) which will try to compile aer locally under the covers.

If the packages installed correctly, you can run `conda list` to see the active packages in your virtual environment.

**Note**

When upgrading from Qiskit < 0.7 to the latest version, uninstall the old version of Qiskit with `pip uninstall qiskit` and then install the latest version.

There are optional dependencies that are required to use all the visualization functions available in Qiskit. You can install these optional dependencies by with the following command

```
pip install qiskit-terra[visualization]
```

After you've installed and verified the Qiskit packages you want to use, import them into your environment with Python to begin working.

```
import qiskit
```

# Superb IDE for Python to navigate though the Qiskit code

```
driver = PySCFDriver(atom=molecule,
                     unit=UnitsType.ANGSTROM,
                     charge=0,
                     spin=0,
                     basis='sto3g')
```

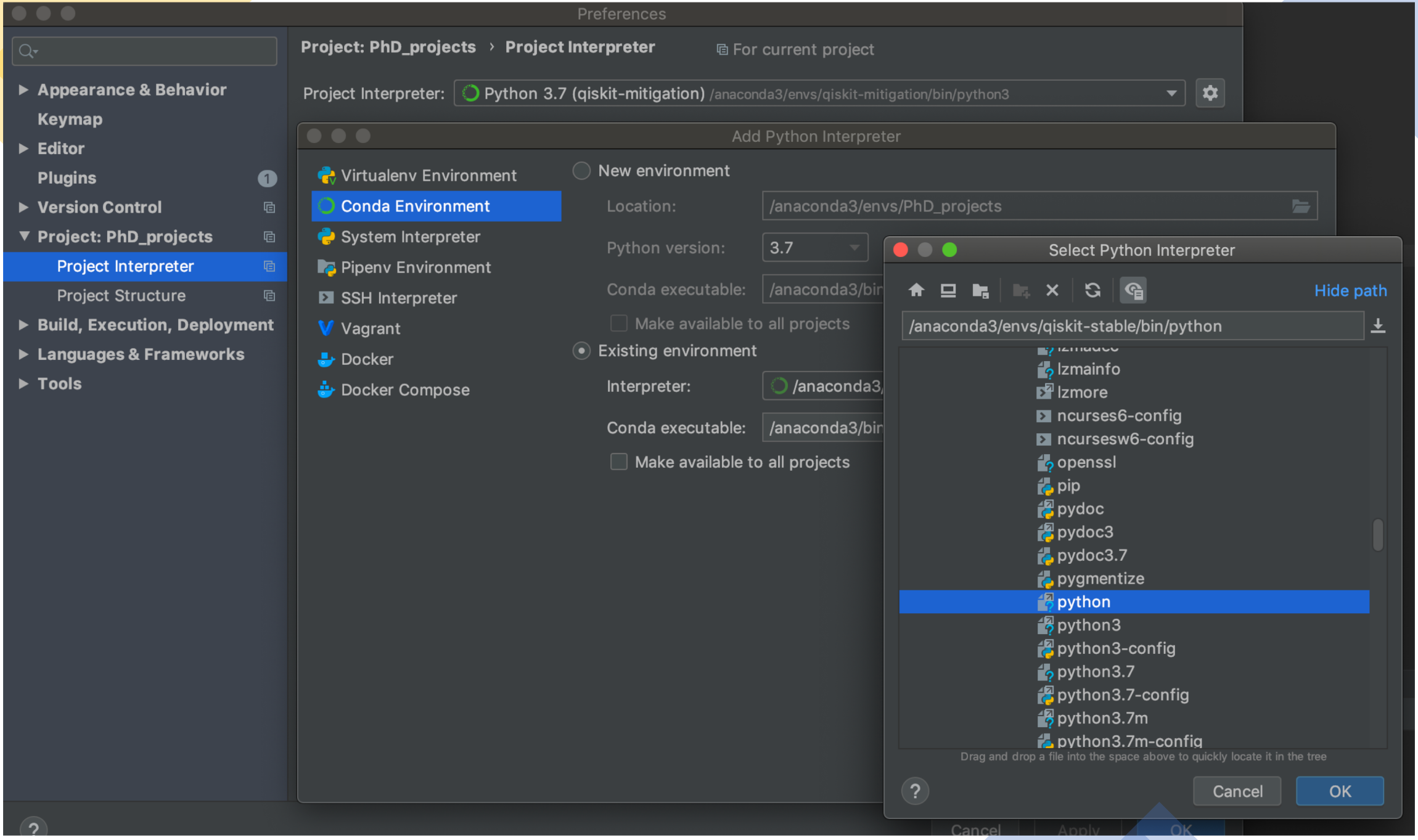
Use on any object in the code:


+ Left Click

Goes to the class definition (path and so on)



```
class PySCFDriver(BaseDriver):
    """Python implementation of a PySCF driver."""

    CONFIGURATION = {
        "name": "PYSCF",
        "description": "PYSCF Driver",
        "input_schema": {
            "$schema": "http://json-schema.org/draft-07/schema#",
            "id": "pyscf_schema",
            "type": "object",
            "properties": {
                "atom": {
                    "type": "string",
                    "default": "H 0.0 0.0 0.0; H 0.0 0.0 0.735"
                },
```


Preferences
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Project: PhD_projects
Project Interpreter
Project Structure
Build, Execution, Deployment
Languages & Frameworks
Tools
Project: PhD_projects › Project Interpreter
For current project
Project Interpreter: Python 3.7 (qiskit-mitigation) /anaconda3/envs/qiskit-mitigation/bin/python3
Add Python Interpreter
Virtualenv Environment
Conda Environment
System Interpreter
Pipenv Environment
SSH Interpreter
Vagrant
Docker
Docker Compose
New environment
Location: /anaconda3/envs/PhD_projects
Python version: 3.7
Conda executable: /anaconda3/bir
Make available to all projects
Existing environment
Interpreter: /anaconda3/
Conda executable: /anaconda3/bir
Make available to all projects
Select Python Interpreter
Hide path
/anaconda3/envs/qiskit-stable/bin/python
lzmainfo
lzmore
ncurses6-config
ncursesw6-config
openssl
pip
pydoc
pydoc3
pydoc3.7
pygmentize
python
python3
python3-config
python3.7
python3.7-config
python3.7m
python3.7m-config
Drag and drop a file into the space above to quickly locate it in the tree
Cancel
OK
Cancel
Apply
OK

## Checking Which Version is Installed

Since the Qiskit package includes a constellation of different elements, simply printing the version by running `qiskit.__version__` can be misleading as it returns only the version for the `qiskit-terra` package. This is because the `qiskit` namespace in Python doesn't come from the Qiskit package, but instead is part of the `qiskit-terra` package.

```
import qiskit
qiskit.__version__
```

```
'0.11.1'
```

To see the versions of all the Qiskit elements in your environment you can use the `__qiskit_version__` attribute. For example, running the following command will return a dictionary that includes the versions for each of the installed Qiskit packages.

```
qiskit.__qiskit_version__
```

```
{'qiskit-terra': '0.11.1',
 'qiskit-aer': '0.3.4',
 'qiskit-ignis': '0.2.0',
 'qiskit-ibmq-provider': '0.4.5',
 'qiskit-aqua': '0.6.2',
 'qiskit': '0.14.1'}
```

**Tip**

If you're filing an issue or need to share your installed Qiskit versions for something, use the `__qiskit_version__` attribute.

# Installation summary

1. Python 3.7
2. PyCharm (Optional, any IDE would work)
3. Visual Studio for Windows users
4. Anaconda, then in Terminal:
    1. conda create –n qiskit python=3.7
    2. source activate qiskit or activate qiskit (on Windows)
5. Terminal: pip install qiskit
6. Terminal: pip install matplotlib (plot data)
7. Terminal: pip install jupyter
8. Terminal: source activate qiskit
9. Terminal: python -m ipykernel install --user --name qiskit --display-name ”qiskit”

# Access IBM Quantum Systems

IBM Quantum offers several real quantum computers and high-performance classical computing simulators through its IBM Quantum Experience with Qiskit. Follow these steps to set up your Qiskit environment to send jobs to IBM Quantum systems.

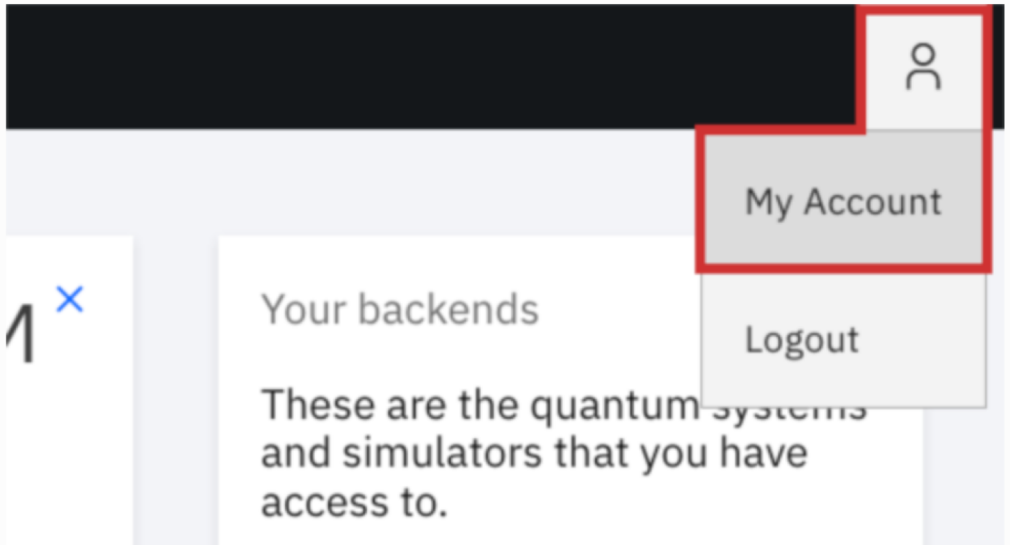
**Note**

With the release of Qiskit 0.11, if you had previously saved your IBM Quantum credentials locally, you might need to update your IBM Quantum Experience credentials so that you can use the new IBM Quantum Experience V2. See Updating your IBM Quantum Experience Credentials.

To configure your account, you create a local configuration file which includes an API key

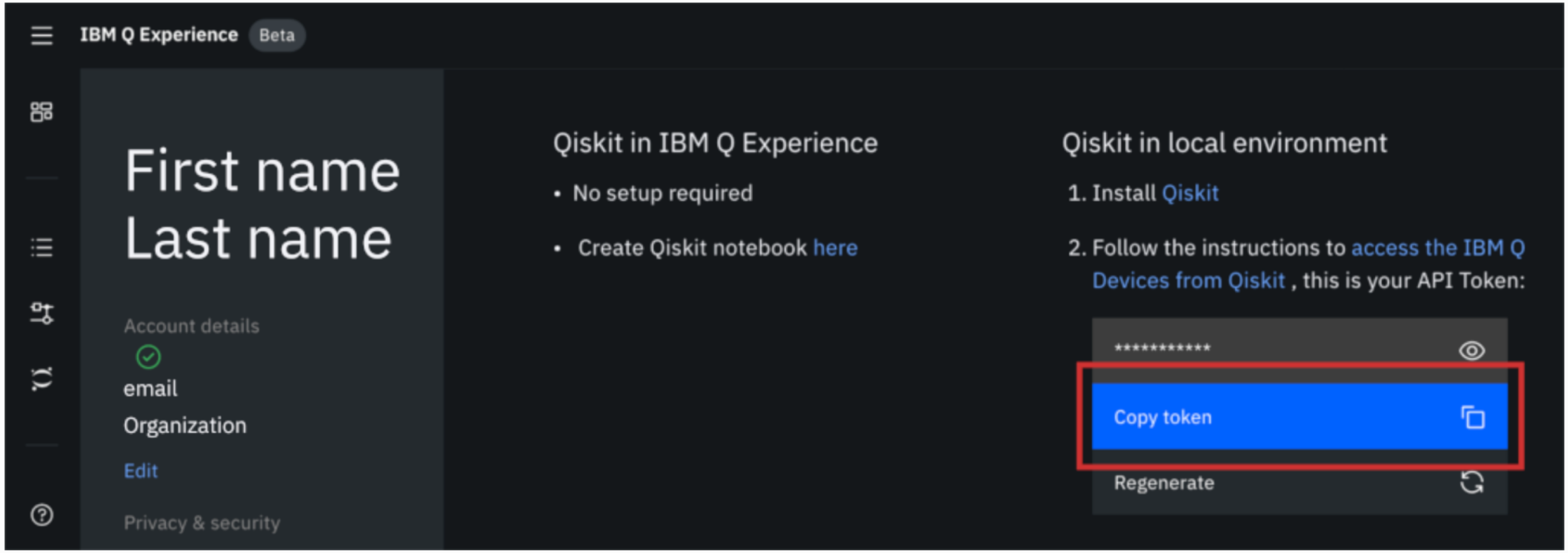
**1.** Create a free IBM Quantum Experience account.

**2.** Navigate to **My Account** to view your account settings.




https://quantum-computing.ibm.com/

**3.** Click on **Copy token** to copy the token to your clipboard. Temporarily paste this API token into your favorite text editor so you can use it later to create an account configuration file.




**4.** Run the following commands to store your API token locally for later use in a configuration file called `qiskitrc`. Replace `MY_API_TOKEN` with the API token value that you stored in your text editor.

```
from qiskit import IBMQ
IBMQ.save_account('MY_API_TOKEN')
```

For more details, such as how to manage multiple IBM Quantum account credentials, refer to this tutorial titled The IBM Quantum Account.

# The IBM Q Account

In Qiskit we have an interface for backends and jobs that is useful for running circuits and extending to third-party backends. In this tutorial, we will review the core components of Qiskit's base backend framework, using the IBM Q account as an example.

The interface has four main components: the account, providers, backends, and jobs:

- **account**: Gives access to one or more 'providers' based on the account's permissions.
- **provider**: Provides access to quantum devices and simulators, collectively called 'backends', and additional services tailored to a specific backend instance.
- **backend**: A quantum device or simulator capable of running quantum circuits or pulse schedules.
- **job**: A local reference to a collection of quantum circuits or pulse schedules submitted to a given backend.

## Table of contents

# The Account

The Qiskit `IBMQ` account object is the local reference for accessing your IBM Q account, and all of the providers, backends, etc, that are available to you.

The `IBMQ` account has functions for handling administrative tasks. The credentials can be saved to disk, or used in a session and never saved.

- `enable_account(TOKEN)`: Enable your account in the current session.
- `save_account(TOKEN)`: Save your account to disk for future use.
- `load_account()`: Load account using stored credentials.
- `disable_account()`: Disable your account in the current session.
- `stored_account()`: List the account stored to disk.
- `active_account()`: List the account currently in the session.
- `delete_account()`: Delete the saved account from disk.

In order to access quantum devices, simulators, or other services, you must specify the source of these items by selecting a provider. To see all the providers available do the following:

```
from qiskit import IBMQ

IBMQ.load_account() # Load account from disk
IBMQ.providers()    # List all available providers
```

```
[<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>]
```

where we have assumed that the user has stored their IBMQ account information locally ahead of time using `IBMQ.save_account(TOKEN)`.

# The Provider

Providers accessed via the `IBMQ` account provide access to a group of different backends (for example, backends available through the IBM Q Experience or IBM Q Network quantum cloud services).

A provider inherits from `BaseProvider` and implements the methods:

- `backends()`: Returns all backend objects known to the provider.
- `get_backend(NAME)`: Returns the named backend.

Using the public provider instance from above:

```
provider = IBMQ.get_provider(hub='ibm-q')
provider.backends()
```

```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

Selecting a backend is done by name using the `get_backend(NAME)` method:

```
backend = provider.get_backend('ibmq_16_melbourne')
backend
```

```
<IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='main')>
```

```
# In[10]:

from qiskit import Aer

backend = Aer.get_backend('statevector_simulator')
```

# Where is the code and tutorials ?

# Qiskit

An open-source framework for working with noisy quantum computers at the level of pulses, circuits, and algorithms.

https://qiskit.org qiskit@qiskit.org

https://github.com/Qiskit

Repositories 23 | Packages | People 111 | Teams 13

## Pinned repositories

**qiskit**

Qiskit is an open-source framework for working with noisy quantum computers at the level of pulses, circuits, and algorithms.

Python ★ 806 327

**qiskit-iqx-tutorials**

A collection of Jupyter notebooks showing how to use Qiskit that is synced with the IBM Q Experience

Jupyter Notebook ★ 1.2k 694

**qiskit-terra**

Terra provides the foundations for Qiskit. It allows the user to write quantum circuits easily, and takes care of the constraints of real hardware.

Python ★ 2.7k 1k

**qiskit-aer**

Aer is a high performance simulator for quantum circuits that includes noise models

Python ★ 122 146

**qiskit-aqua**

Quantum Algorithms & Applications in Python.

Python ★ 307 223

**qiskit-ignis**

Ignis provides tools for quantum hardware verification, noise characterization, and error correction.

Python ★ 73 85

Qiskit / qiskit-iqx-tutorials

Watch 121 | Star 1.2k | Fork 694

Code | Issues 19 | Pull requests 15 | Actions | Security | Insights

A collection of Jupyter notebooks showing how to use Qiskit that is synced with the IBM Q Experience

quantum-computing | tutorial | quantum-programming-language | qiskit

**TO DOWNLOAD**

1,447 commits | 3 branches | 0 packages | 4 releases | 100 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

nonhermitian Merge pull request #864 from dtmcclure/patch-1 ... Latest commit bd9174e 20 days ago

| | | |
|---|---|---|
| .github | Qiskit-tutorials to Qiskit tutorials, drop s from requests (#796) | 5 months ago |
| qiskit | get gate lengths directly from backend | 21 days ago |
| utils | Update test_tutorials.py docstring to match NOTEBOOK_PATH's default v... | 4 months ago |

Qiskit / qiskit-iqx-tutorials

Watch 121

Code | Issues 19 | Pull requests 15 | Actions | Security | Insights

Branch: master | qiskit-iqx-tutorials / qiskit / advanced / aqua / chemistry / | Create new file

manoelmarques Embed header images — Latest

| | |
|---|---|
| .. | |
| H2 | Restructure and new header (#758) |
| LiH | Restructure and new header (#758) |
| dissociation_profile_of_molecule.ipynb | Embed header images |
| index.ipynb | Embed header images |
| programmatic_approach.ipynb | Embed header images |

# *Qiskit Chemistry, Programmatic Approach*

The latest version of this notebook is available on https://github.com/Qiskit/qiskit-tutorial.

---

## Contributors

Richard Chen[1], Antonio Mezzacapo[1], Marco Pistoia[1], Stephen Wood[1]

## Affiliation

- [1]IBMQ

## Introduction

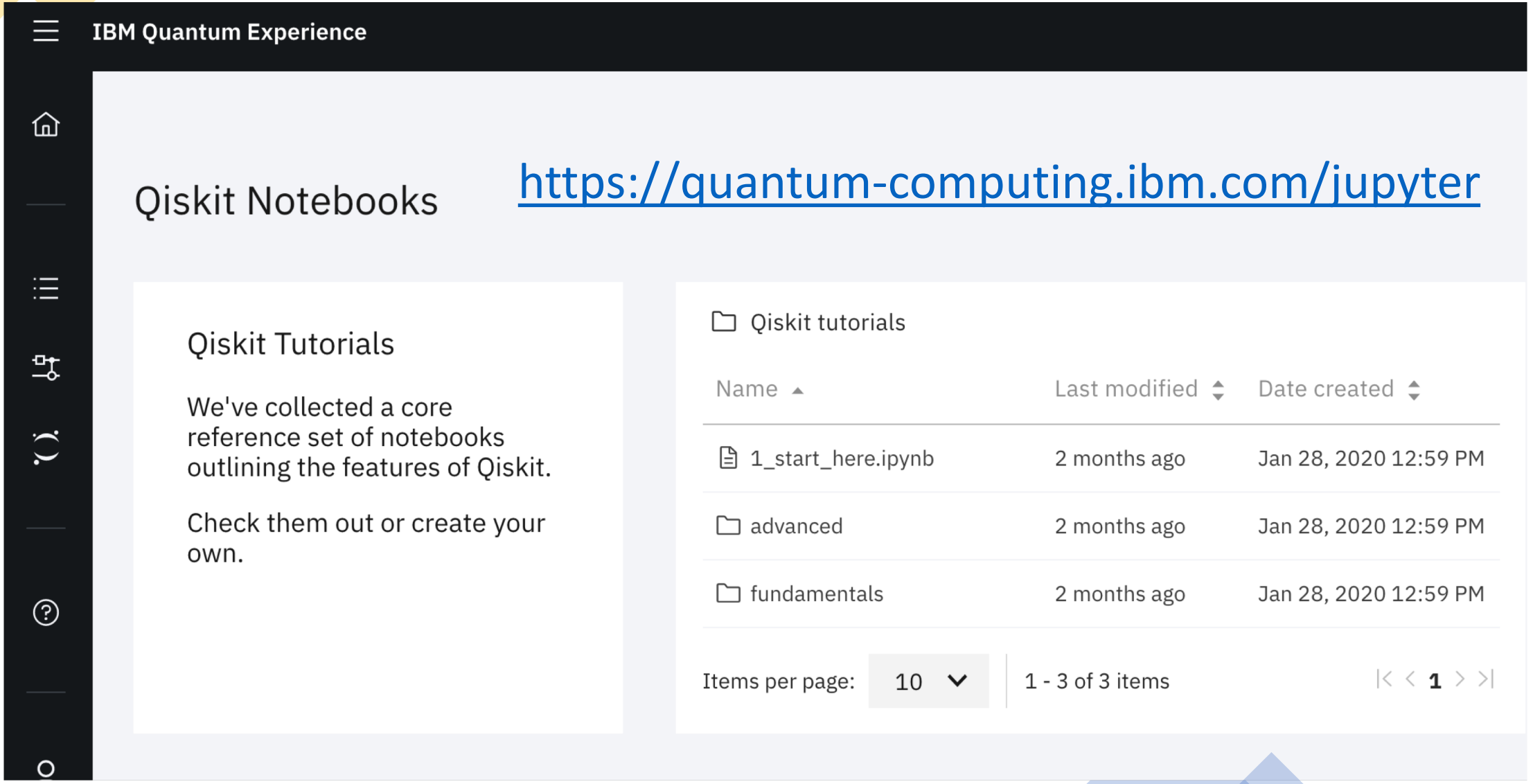
This notebook illustrates how to use Qiskit Chemistry's programmatic APIs.

In this notebook, we decompose the computation of the ground state energy of a molecule into 4 steps:

1. Define a molecule and get integrals from a computational chemistry driver (PySCF in this case)
2. Construct a Fermionic Hamiltonian and map it onto a qubit Hamiltonian
3. Instantiate and initialize dynamically-loaded algorithmic components, such as the quantum algorithm VQE, the optimizer and variational form it will use, and the initial_state to initialize the variational form
4. Run the algorithm on a quantum backend and retrieve the results

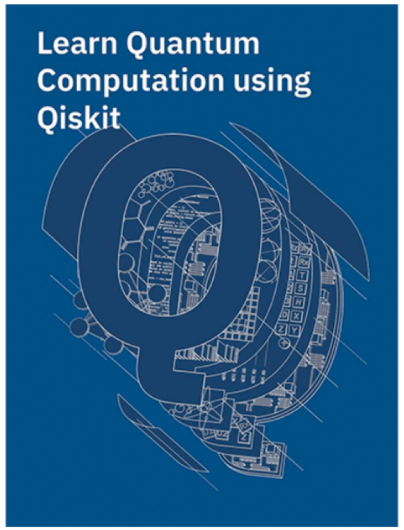
1]:
```
# import common packages
import numpy as np

from qiskit import Aer

# lib from Qiskit Aqua
from qiskit.aqua import QuantumInstance
from qiskit.aqua.algorithms import VQE, ExactEigensolver
from qiskit.aqua.operators import Z2Symmetries
from qiskit.aqua.components.optimizers import COBYLA

# lib from Qiskit Aqua Chemistry
from qiskit.chemistry import FermionicOperator
from qiskit.chemistry.drivers import PySCFDriver, UnitsType
from qiskit.chemistry.components.variational_forms import UCCSD
from qiskit.chemistry.components.initial_states import HartreeFock
```

# Access directly all tutorials online if troubles with downloading






# Learn Quantum Computing using Qiskit




## Great book to learn QC

https://qiskit.org/textbook/preface.html

Greetings from the Qiskit Community team! We initiated this open-source textbook in collaboration with IBM Research as a university quantum algorithms/computation course supplement based on Qiskit.

*Traditional Quantum Computation Course*

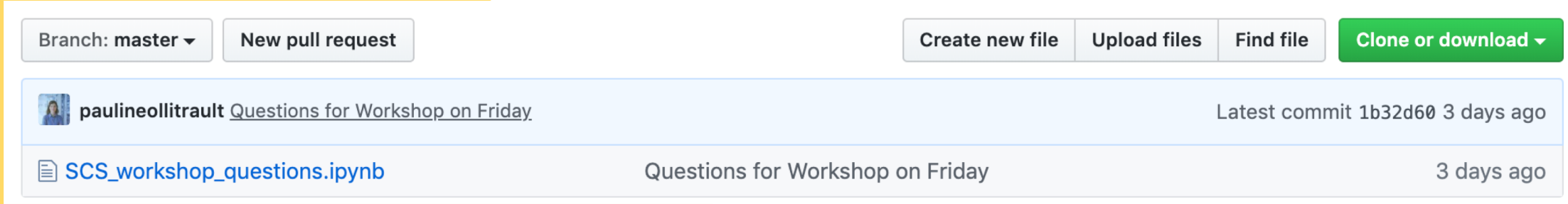
Linear Algebra Quantum Mechanics → Quantum Algorithms → Quantum Hardware

+

*Learn Quantum Computation using Qiskit Textbook*

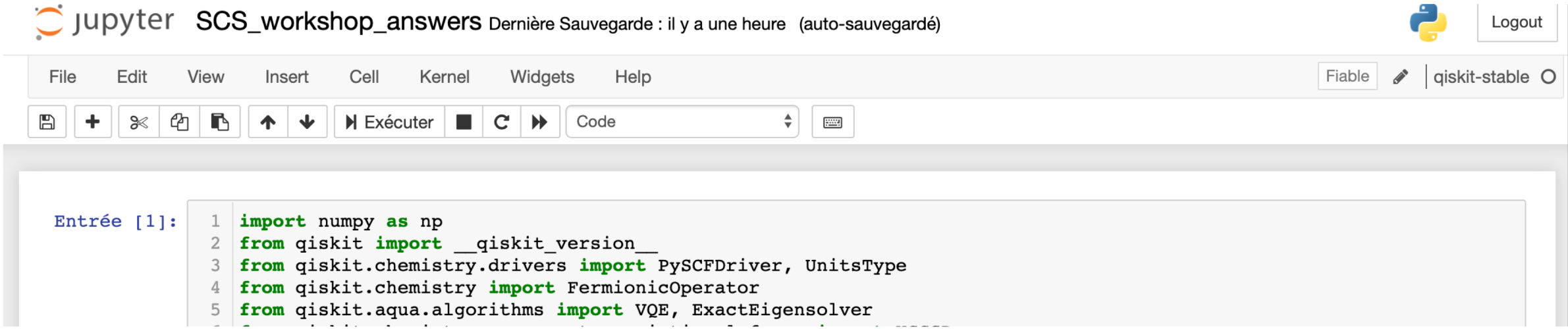
Python Qiskit → Quantum Programming → Quantum Algorithms on Today's Hardware

# We got questions for you !
# Go to the link and download




https://github.com/paulineollitrault/2020_SCS_Qiskit_Workshop

Create a jupyter notebook (good for making plots), in Terminal type: jupyter notebook




Use PyCharm to play with code and see where functions of qiskit are defined

# All summary

https://qiskit.org/documentation/install.html

GitHub with all of Qiskit

https://github.com/Qiskit

Online Tutorials

https://quantum-computing.ibm.com/jupyter

Questions

https://github.com/paulineollitrault/2020_SCS_Qiskit_Workshop

Book on QC and Qiskit

https://qiskit.org/textbook/preface.html

1. Python 3.7
2. PyCharm (Optional, any IDE would work)
3. Visual Studio for Windows users
4. Anaconda, then in Terminal:
    1. conda create –n qiskit python=3.7
    2. source activate qiskit or activate qiskit (on Windows)
5. Terminal: pip install qiskit
6. Terminal: pip install matplotlib (plot data)
7. Terminal: pip install jupyter
8. Terminal: source activate qiskit
9. Terminal: python -m ipykernel install --user --name qiskit --display-name ”qiskit”

This presentation:

https://github.com/Brogis1/2020_SCS_Qiskit_Workshop