

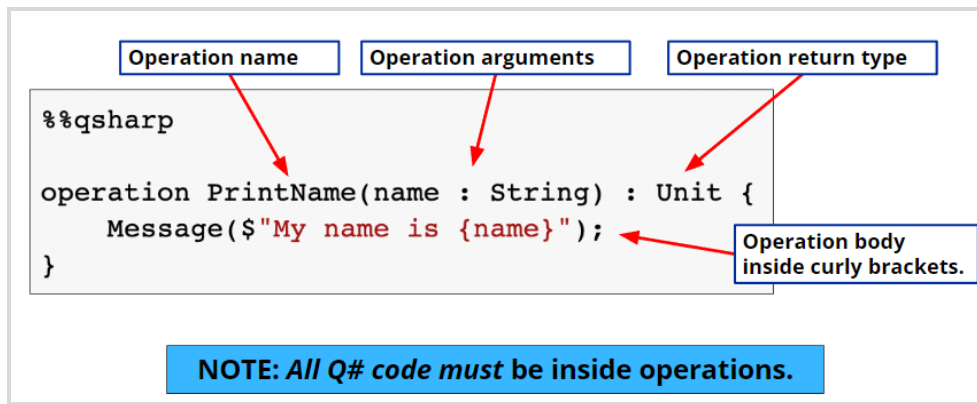
Useful Q# Commands

Operations

1. Declaring an operation in Python:

```
HelloWorld: any = None
```

2. Operations in Q#:



The diagram shows a Q# operation definition with labels pointing to its components:

```
%%qsharp
operation PrintName(name : String) : Unit {
    Message($"My name is {name}");
}
```

- Operation name:** Points to `PrintName`.
- Operation arguments:** Points to `name : String`.
- Operation return type:** Points to `: Unit`.
- Operation body inside curly brackets:** Points to the `Message` statement.

NOTE: All Q# code must be inside operations.

Loops

```
for i in 0 .. Length(array) - 1 {
    Message($" {array[i]}")
}
```

If-Statements

```
if arr[i] == true {
    Message($" {i}");
}
```



© 2023 The Coding School

Data Types

| Data Type | Declaration |
|-----------------------------|---|
| Immutable Primitives | <code>let a = 3;</code> |
| Mutable Primitives | <code>mutable a = 3;</code> <code>set a = a + 1; //reassign a</code> |
| Immutable Arrays | <code>let a = [3, size = 2];</code> |
| Mutable Arrays | <code>mutable a = [3, size = 2];</code> <code>set a w/= 0 <- 2; //reassign a[0] to 2</code> |
| Single Qubit | <code>use q = Qubit();</code> |
| Qubit Arrays | <code>use qs = Qubit[5];</code> |

Qubit operations

| Operation | Single Qubit | Multiple Qubits |
|--------------------|--|--------------------------|
| Gates | <code>H(q);</code> <code>Z(q);</code> <code>X(q);</code> | <code>CX(q0, q1);</code> |
| Measurement | <code>M(q);</code> | <code>MultiM(qs);</code> |

Useful Python Commands

- Local (Microsoft) simulation:

```
HelloWorld.simulate()
```

- Print all targets available:

```
print("Your available targets:")  
for target in targets:  
    print(target.id)
```

- Pick a target:

```
qsharp.azure.target("ionq.simulator")
```

- Submit your code to a remote simulator

```
result = qsharp.azure.execute(ImplementCircuit, shots = 100,  
    jobName = "Exercise #3.5", timeout = 5000)
```

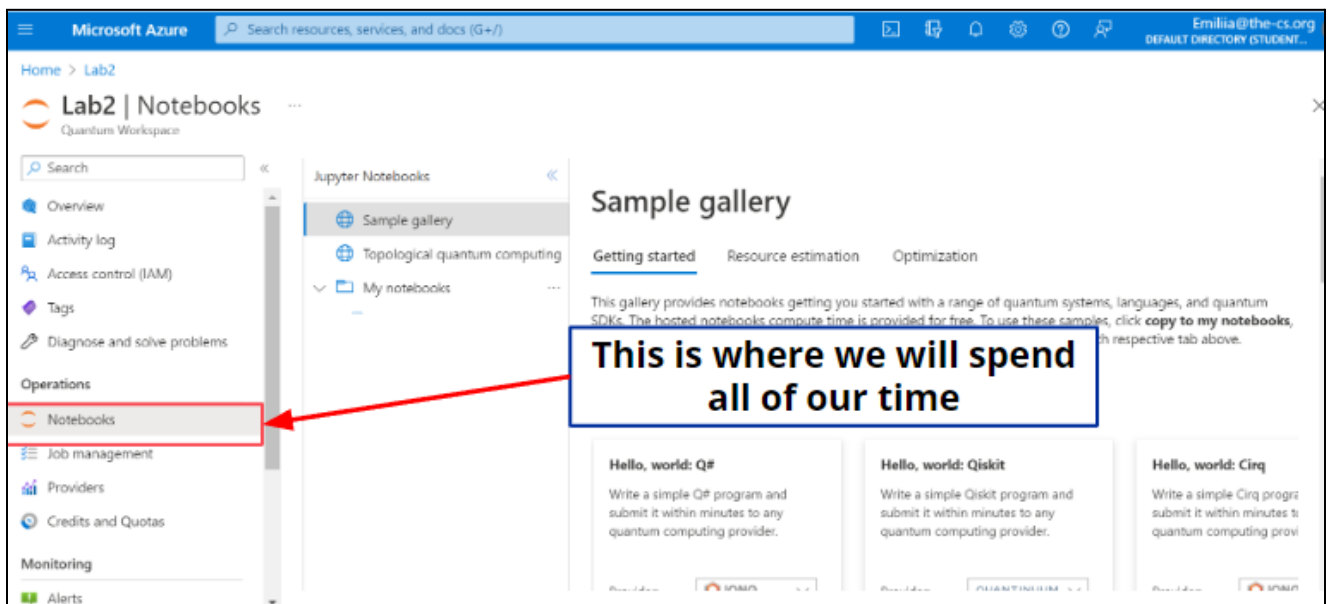
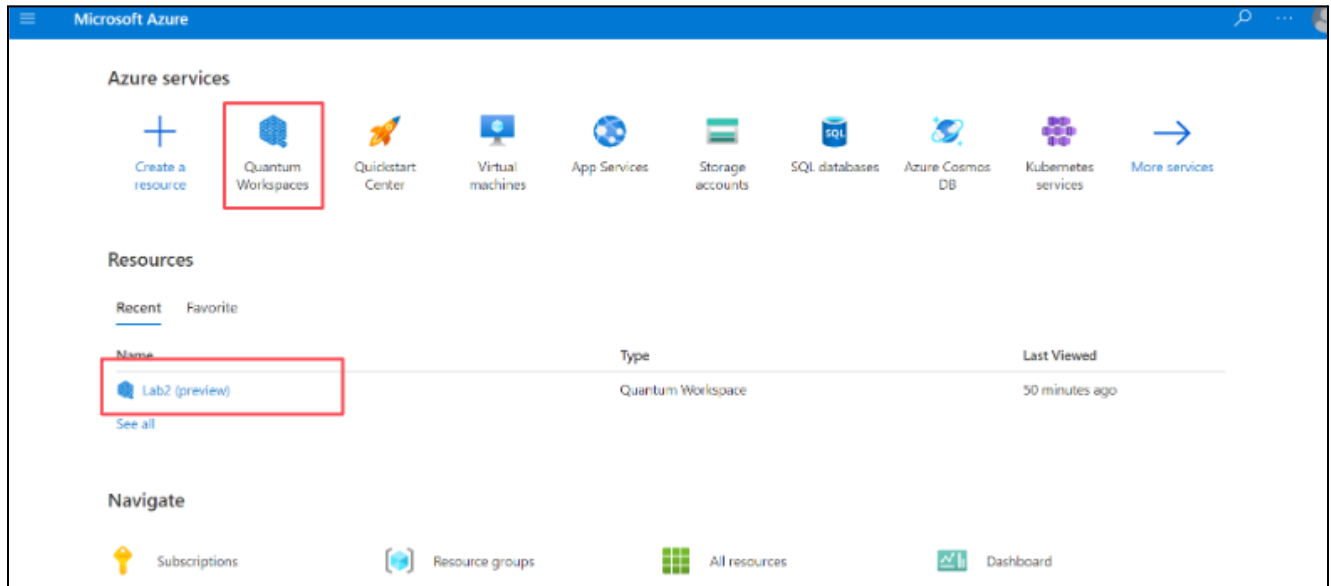
- Visualizing Results:

```
pyplot.bar(result.keys(), result.values())  
  
pyplot.title("Result")  
pyplot.xlabel("Measurement")  
pyplot.ylabel("Probability")  
pyplot.xticks(rotation = 90)  
  
pyplot.show()
```



© 2023 The Coding School

Navigating Azure





© 2023 The Coding School

Microsoft Azure

Search resources, services, and docs (G+/I)

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Operations

Notebooks

Job management

Providers

Credits and Quotas

Monitoring

Alerts

Jupyter Notebooks

Sample gallery

Topological quantum computing

My notebooks

Sample gallery

Getting started

Resource estimation

Optimization

New Notebook

Upload Notebooks

Refresh

Run your first quantum job

Hello, world: Q#

Hello, world: Qiskit

Hello, world: Cirq

Every lab, you will upload a **notebook** we provided for you here.

Microsoft Azure

Search resources, services, and docs (G+/I)

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Operations

Notebooks

Job management

Providers

Credits and Quotas

Monitoring

Jupyter Notebooks

Sample gallery

Topological quantum computing

My notebooks

QXQ_MSFT_Lab_2

Run all

Save

Table of contents

Find

Clear all outputs

Hosted · Kernel idle · CPU100% RAM 4% Last saved a few seconds ago

Python 3 (ipykernel)

Lab #2: Quantum Computing on Azure Quantum with Q# and Jupyter Notebooks

Microsoft Winter School, February 5th 2023

Summary

In today's lab, we are going to build a random number generator while learning about quantum computing and Q#.

Note that only a quantum computer can generate **truly random** numbers. Classical computers are only able to create what we call **pseudorandom** numbers, because, technically, one can predict which number a classical computer will generate (given enough information).

Minimize sidebars with these buttons



© 2023 The Coding School

Microsoft Azure

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Run all Save Table of contents Find Clear all outputs Kernel Restart kernel Interrupt kernel Variables

Hosted · Kernel idle CPU 0% RAM 3% Last saved 21 minutes ago Python 3 (ipykernel)

Lab #2: Quantum Computing on Azure Quantum with Q# and Jupyter Notebooks

Microsoft Winter School, February 5th 2023

Summary

...d a random number generator while learning ...
... can generate **truly random** numbers. Class ...
... number a classical computer will generate (giv ...
... we call **pseudorandom** numbers, because,

Runs all the code cells in order

Shows the outline of your lab

Microsoft Azure

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Run all Save Table of contents Find Clear all outputs Kernel Restart kernel Interrupt kernel Variables

Hosted · Kernel idle CPU 0% RAM 3% Last saved 24 minutes ago Q# Python 3 (ipykernel)

Lab #2: Quantum Computing on Azure Quantum with Q# and Jupyter Notebooks

Microsoft Winter School, February 5th 2023

Summary

In today's lab, we are going to build a random ...
Note that only a quantum computer can gener ...
technically, one can predict which number a cla ...
... that we call **pseudorandom** numbers, because,

We will only be using Python 3 kernel



© 2023 The Coding School

Microsoft Azure

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Run all Save Table of contents Find Clear all outputs Kernel Restart kernel Interrupt kernel Variables

Hosted · Kernel idle CPU 0% RAM 3% Last saved 21 minutes ago Python 3 (ipykernel)

Lab #2: Quantum Computing on Azure Quantum with Q# and Jupyter Notebooks

Microsoft Winter School, February 5th 2023

If something weird happens (e.g. Kernel error), either click **Restart kernel** or **Reload your page (in your browser)**. You will, unfortunately have to re-run your code cells when you do that.

Microsoft Azure

Home > Lab2

Lab2 | Notebooks

Quantum Workspace

Run all Save Table of contents Find Clear all outputs Kernel Restart kernel Interrupt kernel Variables

Hosted · Kernel idle CPU 0% RAM 3% Last saved 37 minutes ago Python 3 (ipykernel)

+ Code + Markdown

from matplotlib import pyplot

import os

You can create your own **code cells** or **text (Markdown) cells** with these. They show up whenever you move your mouse in between of cells.
To edit a Markdown cell - double click.



© 2023 The Coding School

The screenshot shows a Jupyter Notebook interface. At the top, there's a toolbar with buttons for 'Run all', 'Save', 'Table of contents', 'Find', 'Clear all outputs', 'Kernel', 'Restart kernel', 'Interrupt kernel', and 'Variables'. Below the toolbar, a status bar indicates 'Hosted · Kernel idle CPU 0% RAM 3%' and 'Last saved 37 minutes ago'. The main area has tabs for '+ Code' and '+ Markdown'. A code cell is selected, containing the following Python code:

```
from matplotlib import pyplot

import qsharp
import qsharp.azure

# Connect to available targets
targets = qsharp.azure.connect(
    resourceId="/subscriptions/81a08292-c017-4356-956a-0ab12fe1cbd0/resourceGro
    location="eastus")
```

Below the code cell, the text 'Preparing Q# environment...' is visible. A red box highlights the code cell, and a red arrow points from a text box to it.

This is a **code cell**. Click on it, then press **Shift+Enter** to run it. The **output** (if there is any) will appear below.