# Welcome!

While everyone settles in, feel free to get started.

# INSTALLING THE QUANTUM DEVELOPMENT KITRUNNING THE QUANTUM KATAS1Install .NET Core SDK1Download the Quantum Katas2Install Visual Studio Code + Extension2Open a kata folder in VS Code3Install Project Templates3Run dotnet test

#### INSTALL GUIDE aka.ms/qdk-install

### kata download aka.ms/quantum-katas

### #qsharp • @MsftQuantum

### Nitrogen fixation

### Carbon capture

### Materials science

Machine learning

an ann an Annailtean Ann Alssaidhean



### Quantum Programming THE QUANTUM DEVELOPMENT KIT

# Christopher GranadeBettina HeimResearch SDESenior SDEQuantum Architectures and Computation Group, Microsoft

chgranad@microsoft.com

beheim@microsoft.com

# Why Quantum Programming?

# Why Quantum Programming?

### Practical Cost Estimates

# **Example:** Quantum Chemistry of Fe<sub>2</sub>S<sub>2</sub>

Used in reactions and energy transport in photosynthesis

# CLASSICAL ALGORITHM

QUANTUM ALGORITHM (2012)

quantum algorithm (2015) **DAY**  Progress made through

- Better quantum algorithms
- More complete cost estimates
- Improved classical statistics

# Why Quantum Programming?

### Practical Cost Estimates

### Debugging Quantum Algorithms

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

-Brian Kernighan

Even harder to debug an algorithm if you can't run it.

Run All 📗 Run... 🔻 📗 Playlist : All Tests 💌 Failed Tests (14) 🔀 T11\_StateFlip\_Test 6 ms 🔀 T12\_BasisChange\_Test 80 ms X T13\_SignFlip\_Test 8 ms 😢 T14\_AmplitudeChange\_Test 9 ms 🔀 T15\_PhaseFlip\_Test 8 ms 😢 T16\_PhaseChange\_Test 20 ms X T17\_BellStateChange1\_Test 5 ms 😢 T18\_BellStateChange2\_Test 17 ms 😢 T19\_BellStateChange3\_Test 6 ms 😢 T21\_TwoQubitGate1\_Test 👘 42 ms 🔀 T22\_TwoQubitGate2\_Test 🚽 28 ms 😢 T23\_TwoQubitGate3\_Test 🔰 10 ms 🔀 T24\_ToffoliGate\_Test 11 ms 🔀 T25\_FredkinGate\_Test 👘 14 ms

**Test Explorer** 

Search

#### Summary

Last Test Run Failed (Total Run Time 0:

 $(\mathbf{X})$ 14 Tests Failed **P** -

# Why Quantum Programming?

### Practical Cost Estimates

### Debugging Quantum Algorithms

Algorithms to Quantum Applications

# Computing at every level

Developing applications takes a full-stack approach, including both classical and quantum logic



# The Quantum Development Kit

Q#

A new language for quantum *algorithms*.

## Open-source libraries/samples

Phase estimation, amplitude amplification, state preparation, and more.

# Powerful dev tools

Extensions for Visual Studio and Visual Studio Code.

Qshar		
File Edit	View Project Build De	leurg ieam iools Architecture iest Riools Analyze Window Help " • Debug • Any CPU • Microsoft-Quantum-Canon • ▶ Microsoft-Quantum-Canon • ♬ ↓ 1 1 1 1 1 1 1 ↓ 1 1 ↓
DatabaseSe	arch.qs 🤏 🗙	
	224	/// # Summary
	225	/// Prepares the start state and boosts the amplitude of the marked
	226	/// subspace by a sequence of reflections about the marked state and
	227	/// the start state.
	228	
	229	/// # Input
	230	/// ## nIterations
	231	/// Number of applications of the Grover iterate (RS $\cdot$ RM).
	232	/// ## markedQubit
	233	/// Qubit that indicates whether database element is marked.
	234	/// ## databaseRegister
	235	/// A register of n qubits initially in the $ 000 angle$ state.
	236	operation QuantumSearch(
	237	nIterations : Int, markedQubit : Qubit,
	238	<pre>databaseRegister: Qubit[]) : ()</pre>
	239	{
	240	body {
	241	<pre>StatePreparationOracle(markedQubit, databaseRegister);</pre>
	242	// Loop over Grover iterates.
	243	<pre>for(idx in 0nIterations - 1){</pre>
	244	ReflectMarked(markedQubit);
	245	ReflectStart(markedQubit, databaseRegister);
	246	}
	247	}
	248	}

# The Quantum Development Kit

.NET Core

Works with C#, F#, and VB.NET.

# Cross-platform

Windows, macOS, and Linux.

# Versatile simulation

Local simulator and cost estimator.

# Comprehensive documentation

石 🚦 Q# standard libraries - ; 🗙	+ <						
$\to$ $\circlearrowright$   $\ominus$ https://doc	:s.microsoft.com/en-us/quantum/libraries/algorithms?view=qsharp-previ 🖽 🛠 🛛 左 🖒 ·						
	Quantum Phase Estimation						
Filter by title	One particularly important application of the quantum Fourier transform is to learn the						
> Quantum computing ^ concepts	eigenvalues of unitary operators, a problem known as <i>phase estimation</i> . Consider a unitary $U$ and a state $ \phi\rangle$ such that $ \phi\rangle$ is an eigenstate of $U$ with unknown eigenvalue						
Installation and validation	77   / /   / /						
Quickstart - your first quantum program	$U\ket{\phi}=\phi\ket{\phi}$ .						
<ul> <li>Managing quantum machines and drivers</li> </ul>	If we only have access to $U$ as an oracle, then we can learn the phase $\phi$ by utilizing that $Z$ rotations applied to the target of a controlled operation propagate back onto the						
> Quantum development techniques	control.						
✓ Q# standard libraries	Suppose that $V$ is a controlled application of $U$ , such that						
OSS licensing	$V(\ket{0}\otimes\ket{\phi})=\ket{0}\otimes\ket{\phi}$						
The prelude	$ ext{ and } V(\ket{1} \otimes \ket{\phi}) = e^{i\phi} \ket{1} \otimes \ket{\phi}.$						
Higher-order control flow	Then, by linearity,						
Data structures and modeling	$V(\ket{+}\otimes\ket{\phi}) = rac{(\ket{0}\otimes\ket{\phi})+e^{i\phi}(\ket{1}\otimes\ket{\phi})}{\sqrt{2}}.$						
Quantum algorithms	$\sqrt{2}$						
Characterization	We can collect terms to find that						
Error correction	$\ket{0}+e^{i\phi}\ket{1}$						
Applications	$V(\ket{+}\otimes\ket{\phi})=rac{\ket{0}+arepsilon+arepsilon+arepsilon_1}{\sqrt{2}}\otimes\ket{\phi}$						
Testing and debugging $\checkmark$	$= egin{array}{c} {\mathbf v} {\mathbf z} \ = (R_1(\phi) \ket{+}) \otimes \ket{\phi},$						

# Why Q# Works

# High-level programming

First-class quantum operations, partial applications, type parameters, and more.

# Rich Standard Libraries

# Classical logic

Easy to implement hybrid quantum/classical algorithms and adaptive measurement protocols.

💐 Revers	ibleLogicSynth	esis.qs -	- Libraries - Vi	sual Stu	ıdio Cod	e								-		$\times$
<u>F</u> ile <u>E</u> dit	<u>S</u> election <u>V</u>	<u>/</u> iew <u>G</u>	o <u>D</u> ebug _	<u>F</u> asks <u>H</u>	<u>H</u> elp											
D	≡ Rever	sibleL	ogicSynthe	esis.qs	<b>×</b>							•		0		
	398		operatio	on Hi	ddens	ShiftP	roble	em(per	m : In	t[],	shift	: : Int	) :	Int	{	
Ω	399		body	/ {												
	400			let	n = E	BitSiz	e(Len	ngth(p	erm));							
00	401			muta	able n	result	:= 0;									
Y	402															
Ŭ	403			usin	n <mark>g</mark> (qu	ubits	= Qub	oit[2	* n])	{						
	404				let S	Superp	os =	Apply	ToEach	A(H,	_);					
<b>S</b>	405				let S	Shift	= App	lyShi	ft(shi	ft, _	);					
	406				let S	Synth	= Per	mutat	ion0ra	cle(p	erm,	TBS, _	);			
	407				let F	PermX	= App	olyToS	ubregi	sterA	(Synt	h, Seq	uenc	e(0,	n -	1),
					_);											
	408				let F	PermY	= App	olyToS	ubregi	sterA	(Synt	h, Seq	uenc	e(n,	2 *	n
					- 1),	, _);										
	409															
	410				With(	(BindA	([Sup	perpos	; Shif	t; Pe	rmY])	, Inne	Pro	duct	,	
					qubit	ts);										
	411				With(	((Adjo	int P	PermX)	, Inne	rProd	uct,	qubits	;);			
	412				Super	rpos(q	ubits	5);								
	413															
	414				set r	result	: = Me	easure	Intege	r(Lit	tleEn	ndian(q	ubit	s));		[
	415			}												
1	416															
ဖို cgra	nade/temp	ዋ	Libraries	រាំ 1	1 -	<b>蒹</b> 7	<b>1</b>	Ln 3	78, Col 6	Spac	:es: 4	UTF-8	CRLF	Q#	۲	<b>1</b>

The Quantum Development Kit Community

### Feedback

quantum.uservoice.com

### Documentation

github.com/MicrosoftDocs/quantum-docs-pr

#### Open Source Contributions github.com/Microsoft/Quantum

**Social Media** #qsharp • @MsftQuantum



# Visual Studio Code



Open source for Windows 10, macOS, and Linux.

Q#, C#, F#, C++, Python, Julia, LaTeX, Bash, PowerShell, TypeScript and more.

# Visual Studio 2017

QSharpApplication - Microsoft				7 🗗			- م	-	×
<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>P</u> roject <u>B</u> u	ild <u>D</u> ebug Tea <u>m</u> <u>T</u> ools	Ar <u>c</u> hitecture Te <u>s</u> t <u>R</u>	Tools A <u>n</u> alyze	<u>W</u> indow <u>H</u>	elp		Chris Granad	le 🍷	C
o - o 📸 - 🚔 🗳 🐇	ク・ペー Debug ・ Any	y CPU 🔹 🕨 QSha	rpApplication 👻 👼	• • • •		열 📕 위 개 개 🚽			
Operation.gs * X 1 Enamespace QSharpA 2 jopen Microsof 4 jopen Microsof 5 poperation Hel 6 body { 7 limessa 9 limessa 9 limessa 10 11 } 12	<pre>upplication {     t.Quantum.Canon;     t.Quantum.Primitive; .lo() : () {     age("Hello, world!");     </pre>					Solution Explorer Solution Solution CysharpAp Constraints Solution 'QSharpAp Constraints' Solution 'QSharpAp Constraints' Solution 'QSharpAp Solution 'QSharpApplicat P Constraints' Solution 'QSha	、 、 、 に Ctrl+;) plication' (1 pro ion s	ہ ج ہو jject	
Output Compiler Inline Report	Compiler Optimization Report	Error List Code Coverage	e Results						
🗖 Ready	Ln 12 Col 1	Ch 1	INS			↑ Add	o Source Contr	ol 🔺	

World-class C# and Q# debugging on Windows 10, unit testing, performance analysis.

Works with Q#, C#, F#, VB.NET, C++, Python, R, and more.

### Visual Studio Code



I'll be showing Visual Studio Code today, but if you're using Windows 10 and would prefer to use Visual Studio 2017, great! Full documentation for using Visual Studio 2017 is available at aka.ms/quantum-katas.

### Visual Studio 2017

			Chris Granade 🔻 💽

# Getting Started with Q#

# Q# programs are made up of *operations* and *functions*:

#### **OPERATIONS**

Quantum or nondeterministic classical code that can interact with and transform qubits.

#### FUNCTIONS

Purely deterministic classical code (e.g. **Sin**, **Cos**).

The Q# *prelude* includes builtin operations and functions:

- X, Y, Z
- H, S
- R, R1, Rz
- M, Reset
- Message

Complete documentation at: docs.microsoft.com/quantum



# Working with .NET Core

# Code in .NET Core is grouped into **projects**.



Each project compiles to a single cross-platform library or program.

### QSharpApplication/

SOURCE CODE QSharpApplication.csproj Driver.cs Operations.qs

project file classical (C#) source quantum (Q#) source

COMPILED CODE bin/Debug/ QSharpApplication.dll compiled quantum app obj/

### Working with .NET Core Code in .NET Core is grouped into projects.

You may also see a *solution file* (\*.sln).

This collects one or more related projects for loading in Visual Studio 2017. Each project compiles to a single cross-platform library or program.

### **QSharpApplication/**

SOURCE CODE QSharpApplication.csproj Driver.cs Operations.qs

project file classical (C#) source quantum (Q#) source

COMPILED CODE bin/Debug/ QSharpApplication.dll compiled quantum app obj/

# Working with .NET Core

### The **dotnet** tool provides a rich command line interface into .NET Core.



dotnet help
dotnet cmd --help

**dotnet new** Makes a new project.

### dotnet build

Downloads all the packages needed by a project and runs the compiler.

**dotnet run** Runs the project, rebuilding if necessary.

**dotnet test** Runs any unit tests defined in a project.

dotnet add package Installs a new package into the project.

# Working with .NET Core

### The **dotnet** tool provides a rich command line interface into .NET Core.



dotnet help
dotnet cmd --help

**dotnet new** Makes a new project.

### dotnet build

Downloads all the packages needed by a project and runs the compiler.

**dotnet run** Runs the project, rebuilding if necessary.

dotnet test Runs any unit tests defined in a project. dotnet add package

Installs a new package into the project.

# Q# By Example: A Classic Hello

Let's start by seeing how Q# and C# work together to make a complete program.



#### **Int** Signed 64-bit integers.

#### Double

Floating point numbers.

#### **Bool** Either **true** or **false**.

### Result

Either Zero or One.

Pauli Either PauliI, PauliX, PauliY, or PauliZ.

#### Range A sequence of integers.

String Diagnostic message. For any types  $T_0$ ,  $T_1$ ,  $T_2$ , ...,  $T_n$ we can make a tuple:  $(T_0, T_1, T_2, ..., T_n)$ We can also make *arrays* of any type T, denoted T[].

(Result[], (Pauli, Pauli[]))
 (Int, Bool)[]
 (Int[], Double)

# A Bit About Qubits

Q# allows us to interact with quantum data in a variety of ways:

- Prepare fresh qubits in  $|0\rangle$ .
- Transform quantum data with built-in operations.
- Measure qubits to get back classical data.

### Example

We can get random classical bits by:

- Starting with a fresh qubit.
- Using the H operation to prepare superposition.
- Measuring and collapsing the qubit to either |0> or |1>.

# Representing States and Operations Qubit states can be represented as vectors, $|0\rangle \coloneqq \begin{bmatrix} 1\\0 \end{bmatrix}, \quad |1\rangle \coloneqq \begin{bmatrix} 0\\1 \end{bmatrix}.$ Operations then transform qubit states by matrix multiplication.

E.g.:

$$H \coloneqq \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} / \sqrt{2}, \qquad T \coloneqq \begin{bmatrix} 1 & 0 \\ & \frac{i\pi}{4} \end{bmatrix}.$$

### conceptual INTRODUCTION AT aka.ms/quantum-docs/the-qubit

# Q# By Example: QRNGs

Let's make things quantum by generating some random numbers using the  $|+\rangle \coloneqq (|0\rangle + |1\rangle) / \sqrt{2}$  state.

# Example: Qrng

```
operation NextRandomBit() : Result {
  body {
    mutable result = Zero;
    using (qubits = Qubit[1]) {
      H(qubits[0]);
      set result = M(qubits[0]);
      Reset(qubits[0]);
    return result;
```

```
// Prepare in |0\rangle.
// H|0\rangle = |+\rangle.
// Measure \langle 0|.
// Reset to |0\rangle.
```

// Return to
// classical host.

# Example: Qrng

operation NextRandomBit() : Result { body { mutable result = Zero; using (qubits = Qubit[1]) { H(qubits[0]); set result = M(qubits[0]); Reset(qubits[0]); return result;

Let's look closer at what's going on when we call the H operation.

// Prepare in |0>.
// H|0> = |+>.
// Measure <0|.
// Reset to |0>.

// Return to
// classical host.

### States in Q#

The **Qrng** example points at two different ways of thinking about quantum operations:

 $\frac{|\text{INPUT } IS \text{ A STATE}}{|+\rangle} = H |0\rangle$ 

INPUT HAS A STATE
H(qubit);

In Q#, we program with qubits, and let the target machine define how to represent states. Each target machine might define states differently.

- Local simulator: state vector
- Trace simulator: tally counter
- Actual hardware: device ID

By working with qubits, code is reusable across all targets.

# Heisenberg vs Schrödinger Programming

### **Problem:**

How do we describe states when states aren't in Q#?

$$|+\rangle \coloneqq \frac{|0\rangle + |1\rangle}{\sqrt{2}} = H|0\rangle$$

$$\begin{aligned} |\beta_{00}\rangle &\coloneqq \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\ &= \text{CNOT } H |00\rangle \end{aligned}$$

# **Solution:** Think of operations as *programs* which prepare states when given $|0\rangle$ .

Operations are *first-class* in Q#, so we can pass them as values. (More on this later.)

# Allocating and Using Qubits

operation NextRandomBit() : Result { body { mutable result = Zero; using (qubits = Qubit[1]) { H(qubits[0]); set result = M(qubits[0]); Reset(qubits[0]); return result;

This works because we agree to always prepare qubits in the same state, labeled |0>.

// Prepare in  $|0\rangle$ .
//  $H|0\rangle = |+\rangle$ .
// Measure  $\langle 0|$ .
// Reset to  $|0\rangle$ .

// Return to
// classical host.

### Representing Multiple Qubits

For multiple qubits, state vectors have an entry for every bitstring,

$$|00\rangle \coloneqq \begin{bmatrix} 1\\0\\0\\0\\0 \end{bmatrix}, |01\rangle \coloneqq \begin{bmatrix} 0\\1\\0\\0\\0 \end{bmatrix}, |10\rangle \coloneqq \begin{bmatrix} 0\\0\\1\\0\\0 \end{bmatrix}, |11\rangle \coloneqq \begin{bmatrix} 0\\0\\0\\1\\0 \end{bmatrix}.$$

Operations can be written as matrices on these states. E.g.,

$$CNOT \coloneqq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

conceptual introduction at aka.ms/quantum-docs/multiple-qubits

# Q# By Example: Entanglement

We now have everything we need to make and measure pairs of entangled qubits.

# Q# By Example: Teleport I

Q# can help us *think* conceptually about algorithms.

What is a program?

### #include <stdio.h>

main()
{
 printf("hello, world\n");
}

\$ clang hello.c
\$ ./a.out
hello, world

# What is a program?

- A *description* of how to implement a computational task.
- *Interpreted* by a compiler to produce a lower-level program.
- Interpreted by a processor to run the program.

Description and interpretation are inherently about *communication* — hence the need for *languages*.

"We dissect nature along lines laid down by our native language."

—Benjamin Lee Whorf, linguist



"If programmers think in programming languages, they must influence thoughts as much as natural languages do."

—Yukihiro Matsumoto, lead designer of Ruby



### Example: Four Views of Iteration

map(fn, array)

Each expresses same idea, but communicates a different focus: e.g.: architecture-specific implementation, data structures, or composition. We dissect quantum algorithms along lines laid down by our programming languages.

Q# can help us **think** conceptually about algorithms.



# **Case Study:** Conjugating With Unitaries Common quantum algorithm concept: UVU<sup>+</sup>

We want to make it easier to think in terms of this concept: let PhaseIncrement = With(QFT, Increment);

Let's explore the features we need to support conceptual thinking.

### Functors

```
// U<sup>+</sup> is an example of the adjoint functor.
operation EPR(left : Qubit, right : Qubit) : () {
    body {
        H(left);
         (Controlled X)([left], right);
    adjoint auto;
```

// Can undo the operation with (Adjoint EPR).

## Example: With

```
operation WithCA(
    outer : (Qubit[] => () : Adjoint),
    inner : (Qubit[] => () : Adjoint, Controlled),
    target : Qubit[]
) : () {
    body {
                                                     Communicates the
         outer(target);
                                     // U
                                                     algorithmic pattern UVU<sup>+</sup>.
                                     // V
         inner(target);
         (Adjoint outer)(target); // U<sup>+</sup>
                                                     E.g.:
    }
                                                     let PhaseIncrement
    adjoint auto; controlled auto;
                                                        = With(QFT, Increment);
    controlled adjoint auto;
```

**Case Study:** Conjugating With Unitaries We now have a language for clearly expressing the  $UVU^+$  concept.

Let's go further by reducing the *cognitive load* imposed by that expression.

e.g.: Currently, a user has to think about the type **Qubit[]** to use **With**.

### Type-Parameterized Functions and Operations

// Using type parameters, we can express
// that a concept applies across different types.
function Fst<'T, 'U>(pair : ('T, 'U)) : 'T {
 let (first, second) = pair;
 return first;
}

# Function Types

Let 'T and 'U be any two types.

Then 'T -> 'U is a function that takes 'T as an input and returns 'U.

### SINGLETON-TUPLE EQUIVALENCE

'T and ('T) are exactly the same type for all 'T. Thus, all functions take and return a single tuple.

#### **EXAMPLE**

function Map<'T, 'U>(fn : 'T -> 'U, in : 'T[]) : 'U[] {
 let n = Length(arr);
 mutable out = new 'U[n];
 for (idx in 0..n - 1) { set out[idx] = fn(in[idx]); }
 return out;

### Operation Types

Similarly, 'T => 'U is an operation that takes 'T as an input and returns 'U.

Adjoint and Controlled can be specified as well. ('T => () : Adjoint, Controlled)

#### **EXAMPLE**

```
operation ApplyToEach<'T>(op : ('T => ()), targets : 'T[]) : () {
    body {
        for (idx in 0..Length(targets) - 1) { op(targets[idx]); }
      }
    }
}
ApplyToEach(H, register);
```

### Passing States as Operations

operation QuantumSearch(nIterations : Int, markedQubit : Qubit, databaseRegister: Qubit[]) : () {

body {

StatePreparationOracle(markedQubit, databaseRegister);

// Loop over Grover iterates.

for (idx in 0..nIterations - 1) {

ReflectMarked(markedQubit);

ReflectStart(markedQubit, databaseRegister);

We can't directly pass  $|\psi\rangle$ , but we can pass an operation *O* that prepares  $|\psi\rangle$ when given qubits in  $|0\rangle$ .

FULL SAMPLE AT aka.ms/quantum-samples/DatabaseSearch

# Q# By Example: Teleport II

We can make teleport much more reusable with first-class operations.

### Functional Programming

function Add(a : Int, b : Int) : Int {
 return a + b;

let Sum = Fold(Add, 0, \_);
Sum([3; 4; 5]); // Returns 12.

The same features also make it easier to express some kinds of classical processing in terms of composition and combinations of functions.

Map(Add, Zip([1; 2; 3], [10; 11; 12]));
// Returns [11; 13; 15].

# Partial Application

# operation PhaseEst( oracle : (Qubit[] => ()), prep : (Qubit[] => ()), target : Qubit[] ) : Double { ... }

PhaseEst(
 Exp(theta, [PauliZ], \_),
 ApplyToEach(H, \_),
 target

We can make new functions and operations by specifying a subset of the inputs to a function or operation.

This lets us quickly combine existing callables to work well together.

# Partial Application as Control Flow I

We can implement new functional constructs with partial application.

### EXAMPLE

function ComposeImpl<'T, 'U, 'V>
 (outer : ('U -> 'V), inner : ('T -> 'U), target : 'T) : 'V
{ return outer(inner(target)); }

function Compose<'T, 'U, 'V>
 (outer : ('U -> 'V), inner : ('T -> 'U)): ('T -> 'V) {
 return ComposeImpl(outer, inner, \_);

# Partial Application as Control Flow II

Functions can also return partially applied operations, representing *classical* reasoning *about* the instructions given to a quantum device.

#### **EXAMPLE**

operation ApplyTwice<'T>(op : 'T => (), target : 'T) : () {
 body { op(target); op(target); }
}
function SquareOp<'T>(op : 'T => ()) : ('T => ()) {
 return ApplyTwice(op, \_);
 SquareOp has type ('T => ()) -> ('T => ()).

# **Example:** With

```
operation WithCA<'T>(
    outer : ('T => () : Adjoint),
    inner : ('T => () : Adjoint, Controlled),
    target : 'T
) : () {
    body {
        outer(target);
                                   // U
                                                   );
        inner(target);
                                   // V
                                                   Conjugate(U);
        (Adjoint outer)(target); // U<sup>+</sup>
    }
    adjoint auto; controlled auto;
    controlled adjoint auto;
```

Can quickly express ideas like "conjugate the operation U by a Hadamard on each qubit."

let Conjugate = WithCA( ApplyToEach(H, \_),

# Q# By Example: Teleport III

We can quickly combine different operations together using partial application.

### Diagnosing Quantum Programs or How To Keep Moving When Everything Seems Broken

Quantum programming is still programming: mistakes and bugs still happen.

Let's look at a few ways to find and fix bugs!

Step 1

# Read the Docs.

Many bugs come down to mismatched assumptions and conventions.

Checking back with the docs can help.

📑 🐔 📕 Q# standard libraries - ; 🗙	+ ~	×					
$\leftarrow$ $\rightarrow$ $\circlearrowright$ $\mid$ $\stackrel{ ext{ https://docs.l}}{\mapsto}$	microsoft.com/en-us/quantum/libraries/algorithms?view=qsharp-previr 🖽 ☆   🏂 💪 🖄						
	Quantum Phase Estimation						
Filter by title	One particularly important application of the quantum Fourier transform is to learn the						
> Quantum computing ^ concepts	eigenvalues of unitary operators, a problem known as <i>phase estimation</i> . Consider a unitary $U$ and a state $ \phi\rangle$ such that $ \phi\rangle$ is an eigenstate of $U$ with unknown eigenvalue $\phi$ .						
Installation and validation	$U  \phi\rangle = \phi  \phi\rangle$						
quantum program	$\nabla  \psi\rangle - \psi  \psi\rangle$ .						
<ul> <li>Managing quantum machines and drivers</li> </ul>	If we only have access to $U$ as an oracle, then we can learn the phase $\phi$ by utilizing that $Z$ rotations applied to the target of a controlled operation propagate back onto the						
<ul> <li>Quantum development techniques</li> </ul>	control.						
✓ Q# standard libraries	Suppose that $V$ is a controlled application of $U$ , such that						
OSS licensing	$V(\ket{0}\otimes\ket{\phi})=\ket{0}\otimes\ket{\phi}$						
The prelude	$ ext{ and } V(\ket{1} \otimes \ket{\phi}) = e^{i\phi} \ket{1} \otimes \ket{\phi}.$						
Higher-order control flow	Then, by linearity,						
Data structures and modeling	$V(\ket{+}\otimes\ket{\phi}) = rac{(\ket{0}\otimes\ket{\phi})+e^{i\phi}(\ket{1}\otimes\ket{\phi})}{\sqrt{2}}.$						
Quantum algorithms	V 2						
Characterization	We can collect terms to find that						
Error correction	$\ket{0}+e^{i\phi}\ket{1}$						
Applications	$V(\ket{+}\otimes\ket{\phi}) = rac{1+\gamma}{\sqrt{2}}\otimes\ket{\phi}$						
Testing and debugging $\succeq$	$= (R_1(\phi)\ket{+}) \otimes \ket{\phi},$						

operation and function reference aka.ms/qsharp-ref "printf" Debugging

Message : String -> () emits diagnostics to C# driver.

By default, messages are printed to the console.

Especially useful for finding classical mistakes.

### INTERPOLATED STRINGS

**\$"{expr}"** inserts diagnostic information about **expr** into a **String** for use with **Message**.

#### EXAMPLE

```
function Hello() : () {
    let x = 42;
```

Message(\$"Hello, {x}."); // Prints "Hello, 42." to the console.

### Assertions About States

Simulators need not obey the No-Cloning Theorem.

The Assert\* operations executed on a simulator kill a Q# program if it would produce the "wrong" measurement.

### ANTHROPIC PRINCIPLE

Observed effects must be compatible with their being an observer.

#### **EXAMPLE**

u (msg); TeleportMessage (msg, there); (Adjoint u)(there);

AssertQubit(Zero, there);

We used AssertQubit earlier to test the validity of our teleport operation.

# The Choi–Jamiłkowski Isomorphism

Quantum processes are isomorphic to quantum states,  $|i\rangle\langle j| \leftrightarrow |i\rangle|j\rangle$ . Define  $J(\Lambda)$  by acting  $\Lambda$  on half of an entangled pair.



#### EXAMPLE

```
operation IdentityTeleport(q : Qubit[]) : () {
    body { using (aux = Qubit[1]) {
        Teleport(q[0], aux[0]);
        SWAP(q[0], aux[0]);
    } }
}
operation TeleportationTest() : () {
    body {
        // Process assertions are a
        // special case of state assertions.
        AssertOperationsEqualReferenced(
            IdentityTeleport, NoOp, 1);
    }
```

#### **FULL SAMPLE AT** aka.ms/quantum-samples/UnitTesting

# Dumping Diagnostic Information

DumpMachine<'T>: 'T -> () instructs target machines to report their diagnostic information.

The **only** observable effect of functions is their return value.

DumpMachine and Message return (), which has only one valid value ().

This allows for calls to be stripped with no observable consequences.

DumpRegister<'T> :
 ('T, Qubit[]) -> ()
reports diagnostic information
about a subset of qubits.

Information is target machine–dependent:

- QuantumSimulator reports state vectors.
- QCTraceSimulator does not report any additional info.
- Future machines may provide different information entirely (e.g.: internal IDs for **Qubits**).

# Q# By Example: Diagnostics

Let's see **DumpMachine** in action.

# Step-By-Step Debugging in Visual Studio 2017

Visual Studio 2017 goes further, using diagnostic information exposed by target machines to help find bugs.

- Can step between C# and Q# code.
- Locals shows the probability of obtaining Zero for hypothetical measurements.
- Debugger and Test Explorer integrate to help fix unit tests.

<u></u> roject <u>b</u> uild <u>D</u> e				
э - 0 粘 - 🖆 💾 🚰 ツ - 🤇			► <u>C</u> ontinue ▼	🗖 🖕 🔍 📕 🗏 🍹
ocess: [41072] dotnet.exe	▼ 🗷 Lifecycle Events ▼ Thread: [659	80] Worker Thread	- 🔻 🔻 😕 –	
Taska				
IdSKS				
1       □ namespace Microsoft         2       open Microsoft.(         3       open Microsoft.(         4       open Microsoft.(         5	<pre>Quantum.Demos.Teleportation { Quantum.Primitive; Quantum.Canon; Quantum.Extensions.Testing; portArbitraryState (u : (Qubit = register = Qubit[2]) { msg = register [0]; there = register [1]; msg); aportMessage (msg, there); joint u)(there); artQubit(Zero, there); sage ("The anthropic principle lyToEach(Reset, register); </pre>	=> () : Adjoint)) : ( says we teleported a	) {	essfully! :) ");
23 J 24 100 % → < Chris Granade, 32 days ago	r aution, r change 4			
23 J 24 100 % → < Chris Granade, 32 days ago   Locals				
23 J 24 100 % → < Chris Granade, 32 days ago   Locals	Value			
23 J 24 100 % → < Chris Granade, 32 days ago   Locals Name > ♥ register	Value Count = 2			
23 J 24 100 % → < Chris Granade, 32 days ago   Locals Name > @ register ▲ @ msg	Value Count = 2 (q:0)			<ul> <li>– ₽</li> <li>Type</li> <li>Microsoft.Quantum.Simulati</li> <li>Microsoft.Quantum.Simulati</li> </ul>
23 J 24 100 % → < Chris Granade, 32 days ago   Locals Name > @ register ▲ @ msg ↓ ld	Value Count = 2 (q:0) 0			<ul> <li></li></ul>
23 J 24 100 % ← < Chris Granade, 32 days ago   Locals Name	Value Count = 2 (q:0) 0 0			✓ ₽ Type Microsoft.Quantum.Simulati Microsoft.Quantum.Simulati int double
23 J 24 100 % - < Chris Granade, 32 days ago   Locals Name ▷ @ register ▲ @ msg	Value Count = 2 {q:0} 0 {q:1} 1			<ul> <li>✓ ┦</li> <li>Type</li> <li>Microsoft.Quantum.Simulati</li> <li>int</li> <li>double</li> <li>Microsoft.Quantum.Simulati</li> <li>int</li> </ul>

Compiler Inline Report Compiler Optimization Report Call Stack Breakpoints Exception Settings Command Window Immediate Window Output Error List

Let's get hands-on!

Each kata has a **Tasks.qs** file with blanks to fill in with your code.

# Run **dotnet test** to check your answers.

#### RUNNING THE QUANTUM KATAS

Download the Quantum Katas



Open QuantumKatas.vscode-workspace.



Run **Tasks: Run Test Task** from the Command Palette.

# Six katas to choose from — pick one and have fun!

- If you get stuck in Visual Studio Code, search the Command Palette. Ctrl + Shift + P or  $\Re$  + Shift + P
- ReferenceImplementation.qs has the solutions — don't look unless you get *really* stuck!
- Your quick references have a lot of useful information, including function and operation summaries and links to documentation.



Open QuantumKatas.vscode-workspace.



Run Tasks: Run Test Task from the Command Palette.

### Six katas to choose from pick one and have fun!

### **KATA DOWNLOAD AND INSTRUCTIONS** aka.ms/quantum-katas

#### RUNNING THE QUANTUM KATAS



# Six katas to choose from — pick one and have fun!

<b>F</b> I	EXPLORER	>Test ග් 🗖	
-	OPEN EDITORS	Tasks: Run Test Task recently used	
Ω	<b>≣ Tasks.qs</b> Basic	Git: Pop Latest Stash other commands	
~	🔺 QUANTUMKATAS (W	Tasks: Configure Default Test Task	
88	<ul> <li>Basic Gates</li> </ul>	Terminal: Run Selected Text In Active Terminal	
x	README.md	Terminal: Scroll to Bottom	
$\sim$	≣ Tasks.qs	Terminal: Scroll to Top % Home	
8	<ul> <li>Deutsch–Jozsa Alç</li> </ul>	Terminal: Show Next Find Term	
	<ol> <li>README.md</li> </ol>	Terminal: Show Previous Find Term	
ġ,	≡ Tasks.qs	Terminal: Switch Active Terminal	
	<ul> <li>Measurements</li> </ul>	42 // returns the qubit to the original state.	
	<ol> <li>README.md</li> </ol>	43 operation StateFlip (q : Qubit) : ()	
	≡ Tasks.qs	44 {	
	<ul> <li>QEC: Bit-flip Code</li> </ul>	45 body	
	<li>README.md</li>	47 // The Pauli X gate will change the 10) state to the 11) state and	
	≡ Tasks.qs		
	<ul> <li>Superposition</li> </ul>		
	<li>README.md</li>	49 // Then rebuild the project and rerun the tests –	
	≡ Tasks.qs	T11_StateFlip_Test should now pass!	
	<ul> <li>Teleportation</li> </ul>		
	README.md	52 }	
	≣ Tasks.qs	53 adjoint self;	
		54 }	
		$55$ // Task 1.2 Basis change: $ 0\rangle$ to $ \pm\rangle$ and $ 1\rangle$ to $ -\rangle$ (and vice versa)	
**		57 // Input: A qubit in state $ \psi\rangle = \alpha  0\rangle + \beta  1\rangle$ .	
**		58 // Goal: Change the state of the qubit as follows:	
₽ mas	ter 💭 😵 0 🛦 0	Ln 1, Col 1 Spaces: 4 UTF-8 with BOM LF Q#	۲

#### UNNING THE QUANTUM KATAS





Run Tasks: Run Test Task from the Command Palette.

# Six katas to choose from — pick one and have fun!

<u>n</u>	EXPLORER	basi	ى ھ	□ …
	OPEN EDITORS	test Basic Gates	recently used tasks 🔅 ///////	
2 8 8 8 1 1	<ul> <li>OPEN EDITORS</li> <li>Tasks.qs Basic</li> <li>QUANTUMKATAS (W</li> <li>Basic Gates         <ul> <li>README.md</li> <li>Tasks.qs</li> <li>Deutsch-Jozsa Alg</li> <li>README.md</li> <li>Tasks.qs</li> <li>Measurements</li> <li>README.md</li> <li>Tasks.qs</li> <li>Measurements</li> <li>README.md</li> <li>Tasks.qs</li> <li>QEC: Bit-flip Code</li> <li>README.md</li> <li>Tasks.qs</li> </ul> </li> </ul>	test Basic Gates ORKSPACE) 33 34 35 36 37 37 37 39 40 41 42 43 44 45 46 47	<pre>////////////////////////////////////</pre>	
\$	<ul> <li>Frasks.qs</li> <li>Superposition</li> <li>README.md</li> <li>Tasks.qs</li> <li>Teleportation</li> <li>README.md</li> <li>Tasks.qs</li> </ul>	48 49 50 51 52 53 54 55 56 57 58	<pre>// Type X(q); // Type X(q); // Then rebuild the project and rerun the tests - T11_StateFlip_Test should now pass! // } adjoint self; } // Task 1.2. Basis change:  0) to  +) and  1) to  -) (and vice versa) // Input: A qubit in state  ψ) = α  0) + β  1). // Goal: Change the state of the qubit as follows:</pre>	
P mast	ter 🗯 😵 0 🛦 0		Ln 1, Col 1 Spaces: 4 UTF-8 with BOM LF (	ಧ# 🙂

#### RUNNING THE QUANTUM KATAS



# Six katas to choose from — pick one and have fun!

Ŋ	EXPLORER	≣ Tasks.qs	×	<u>р</u>	···· [						
	▲ OPEN EDITORS		///////////////////////////////////////								
$\cap$	<b>Ξ Tasks.qs</b> BasicGates										
	A QUANTUMKATAS (WORKSPACE)										
ŶŶ	<ul> <li>Basic Gates</li> </ul>		// Tack 1 1 State flips 10) to 11) and vice versa								
8	README.md		// Input: A qubit in state $ \psi\rangle = \alpha  0\rangle + \beta  1\rangle$ .								
_	≣ Tasks.qs		// Goal: Change the state of the qubit to $\alpha  1\rangle + \beta  0\rangle$ .								
8	<ul> <li>Deutsch–Jozsa Algorithm</li> </ul>										
	<ol> <li>README.md</li> </ol>										
	≣ Tasks.gs		<pre>// If the qubit is in state  1), change its state to  0).</pre>								
	Measurements		// Note that this operation is set -adjoint; applying it for a second tim								
	(i) README.md		operation StateFlip (g : Qubit) : ()								
	≡ Tasks αs		{								
			body								
			{								
			<pre>// The Pauli X gate will change the  0) state to the  1) state an wise warea</pre>								
			Vice Versa. $(/ Type X(a))$								
			// Then rebuild the project and rerun the tests -								
		PROBLEM	S OUTPUT TERMINAL *** 1. Task - tast (Ba 🕯 🔺 🏛	<u>,</u> п	×						
	≕ Tasks.qs			· ⊔	Â						
	<ul> <li>Teleportation</li> </ul>	at Quan	<pre>tum.Kata.BasicGates.TestSuiteRunner.TestTarget(TestOperation op) in /Users/cg</pre>		le/S						
	<ol> <li>README.md</li> </ol>	ource/Repo	s/QuantumKatas/BasicGates/TestSuiteRunner.cs:Line 38								
	≣ Tasks.qs	Total tests: 14. Passed: 0. Failed: 14. Skipped: 0. Test Run Failed. Test execution time: 1.8839 Seconds The terminal process terminated with exit code: 1									
\$		Terminal w	ill be reused by tasks, press any key to close it.								

#### RUNNING THE QUANTUM KATAS



# Six katas to choose from — pick one and have fun!

### kata download and instructions aka.ms/quantum-katas

C O O A

The Quantum Development Kit Community Many ways to get involved! Feedback quantum.uservoice.com

### Documentation

github.com/MicrosoftDocs/quantum-docs-pr

### **Open Source Contributions**

github.com/Microsoft/Quantum

Social Media #qsharp • @MsftQuantum



Fix link to style guide (#3)

5 months ag