Q# Tutorial Quick Reference

Working with Tutorials

Command Line Basics

Change directory	`cd dirname`
Go to home	`cd ~`
Go up one directory	`cd ..`
Make new directory	`mkdir dirname`
Open current directory in VS Code	`code .`

Building Project and Running Tests

Change directory to project directory	`cd project-dir`
Build solution	`dotnet build`
Run all unit tests	`dotnet test`

VS Code Keyboard Shortcuts

Open command palette	Ctrl + Shift + P ⌘ + ⇧ + P
Show terminal	Ctrl + ` ⌘ + `
Open folder	Ctrl + K then Ctrl + O ⌘ + K then ⌘ + O

Documentation

Quantum Development Kit	https://docs.microsoft.com/quantum
Q# Language Reference	https://docs.microsoft.com/en-us/quantum/quantum-qr-intro
Q# Library Reference	https://docs.microsoft.com/en-us/qsharp/api/

Q# Language

Primitive Types

64-bit integers	`Int`
Double-precision floats	`Double`
Booleans	`Bool` `true` or `false`
Qubits	`Qubit`
Pauli operators	`Pauli` `PauliI`, `PauliX`, `PauliY`, or `PauliZ`
Measurement results	`Result` `Zero` or `One`
Sequences of integers	`Range` e.g.: 1..10 or 5..-1..0
Strings	`String`

Q# Language (continued)

Derived Types

Arrays	`elementType[]`
Tuples	`(type0, type1, ...)`
Functions	`inputType -> outputType` e.g.: `ArcTan2 : (Double, Double) -> Double`
Operations	`inputType => outputType : variants` e.g.: `H : (Qubit => () : Adjoint, Controlled)`

Function and Operation Definitions

Functions	`function Name(in0 : type0, ...) : returnType { // function body }`
Operations	`operation Name(in0 : type0, ...) : returnType { body { ... } adjoint { ... } controlled { ... } adjoint controlled { ... } }`

Calling w/ Functors

Adjoint call	`(Adjoint Name)(operationInputs)`
Controlled call	`(Controlled Name)(controlQubits, operationInputs)`

Variables

Declare immutable	`let name = value`
Declare mutable	`mutable name = value`
Update mutable	`set name = value`

Arrays

Allocation	`mutable name = new Type[length]`
Length	`Length(name)`
ith element (0-based indexing)	`name[i]`
Array literal	`[value0; value1; ...]` e.g.: `[true; false; true]`
Slicing (subarray)	`let name = name[start..end]`

Control Flow

For loop	`for (ind in range) { ... }` e.g.: `for (i in 0..N-1) { ... }`
Repeat-until-success loop	`repeat { ... } until condition fixup { ... }`
Conditional statement	`if cond1 { ... } elif cond2 { ... } else { ... }`

Qubits and Operations on Qubits

Qubits

Allocation	`using (name = Qubit[length] { // Qubits in name start in $\|0\rangle$. ... // Qubits must be returned to $\|0\rangle$. }`

Standard Library Functions and Operations

Pauli operations	`I : (Qubit => ())`
	`X : (Qubit => ())`
	`Y : (Qubit => ())`
	`Z : (Qubit => ())`

Hadamard	`H : (Qubit => ())`
Controlled-NOT	`CNOT : ((control : Qubit, target : Qubit) => ())`
Measure one qubit in Pauli Z basis	`M : Qubit => Result`
Perform joint measurement of qubits in given Pauli bases	`Measure : (Pauli[], Qubit[]) => Result`
Rotate about given Pauli axis	`R : (Pauli, Double, Qubit) => ()`

Rotate about Pauli X, Y, Z axes	`Rx : (Double, Qubit) => ()`
	`Ry : (Double, Qubit) => ()`
	`Rz : (Double, Qubit) => ()`

Reset qubit to $\|0\rangle$	`Reset : Qubit => ()`
Reset qubits to $\|0..0\rangle$	`ResetAll : Qubit[] => ()`
Apply a gate to each qubit	`ApplyToEach : ((Qubit => ()), Qubit[]) => ()`
Display a log message	`Message : String -> ()`