Q# 0.3 Language Quick Reference

Primitive Types

64-bit integers	`Int`
Double-precision floats	`Double`
Booleans	`Bool` e.g.: `true` or `false`
Qubits	`Qubit`
Pauli basis	`Pauli` e.g.: `PauliI`, `PauliX`, `PauliY`, or `PauliZ`
Measurement results	`Result` e.g.: `Zero` or `One`
Sequences of integers	`Range` e.g.: 1..10 or 5..-1..0
Strings	`String`
"Return no information" type	`Unit` e.g.: `()`

Derived Types

Arrays	*elementType*`[]` `(elementType,elementType)[][]`
Tuples	`(`*type0*`, `*type1*`, ...)` e.g.: `(Int, Qubit)`
Functions	*input* `->` *output* e.g.: `ArcCos : (Double) -> Double`
Operations	*input* `=>` *output* `:` *variants* e.g.: `H : (Qubit => Unit : Adj, Ctl)`

Functions, Operations and Types

Define function (classical routine)	`function` *Name*`(`*in0* `:` *type0*`, ...) :` *returnType* `{`
	`  // function body`
	`}`

Define operation (quantum routine)	`operation` *Name*`(`*in0* `:` *type0*`, ...)` `:` *returnType* `{`
	`  body { ... }`
	`  adjoint { ... }`
	`  controlled { ... }`
	`  adjoint controlled { ... }`
	`}`

Define user-defined type	`newtype` *TypeName* `=` *BaseType* e.g.: `newtype TermList = (Int, Int -> (Double, Double))`
Call adjoint operation	`Adjoint` *Name*`(`*parameters*`)`
Call controlled operation	`Controlled` *Name*`(`*controlQubits*`,` *parameters*`)`

Symbols and Variables

Declare immutable symbol	`let` *name* `=` *value*
Declare mutable symbol (variable)	`mutable` *name* `=` *initialValue*
Update mutable symbol (variable)	`set` *name* `=` *newValue*

Apply-and-Reassign	`mutable` *name* `=` *initialValue*
	`for (`*index* `in` *range*`) {`
	`set counter +=` *index*`;`
	`}`

Arrays

Allocation	`mutable` *name* `= new` *Type*`[`*length*`]`
Length	`Length(`*name*`)`
k-th element	*name*`[k]` NB: indices are 0-based
Array literal	`[`*value0*`, `*value1*`, ...]` e.g.: `[true, false, true]`
Slicing (subarray)	*name*`[`*start*`...`*end*`]` *name*`[`*start*`...]` *name*`[...`*end*`]` *name*`[...]`

Control Flow

For loop	`for (`*index* `in` *range*`) {`
	`  // Use integer` *index*
	`}`
	e.g.: `for (i in 0..N-1) { ... }`

Iterate over an array	`for (`*val* `in` *array*`) {`
	`  // Use value` *val*
	`}`
	e.g.: `for (q in register) { ... }`

Repeat-until-success loop	`repeat { ... }`
	`until (`*condition*`)`
	`fixup { ... }`

Conditional statement	`if (`*cond1*`) { ... }`
	`elif (`*cond2*`) { ... }`
	`else { ... }`

Ternary operator	*condition* `?` *caseTrue* `|` *caseFalse*
Return a value	`return` *value*
Stop with an error	`fail "`*Error message*`"`

Conjugations

ApplyWith	`operation` *Name*`(`*in0* `:` *type0*`, ...)`
	`:` *returnType* `{`
	`  within { ... }`
	`  apply { ... }`
	`}`

Debugging

Print a string	`Message("Hello Quantum!")`
Print an interpolated string	`Message($"Value = {`*val*`}")`
Assert that a qubit is in $\|0\rangle$ or $\|1\rangle$ state	`AssertQubit(Zero,` *oneQubit*`)`
Print amplitudes of wave function	`DumpMachine("dump.txt")`

Qubit Allocation

Allocate qubits	`using (`*reg* `= Qubit[`*length*`]) {`
	`  // Qubits in` *reg* `start in` $\|0\rangle$.
	`  ...`
	`  // Qubits must be returned to` $\|0\rangle$.
	`}`

Allocate one qubit	`using (`*one* `= Qubit()) { ... }`

Measure qubit in Pauli Z basis	`M(oneQubit)` yields a `Result` (`Zero` or `One`)
Reset qubit to $\|0\rangle$	`Reset(oneQubit)`
Reset an array of qubits to $\|0..0\rangle$	`ResetAll(register)`

Basic Gates

Pauli gates	`X(qubit)` : $\|0\rangle \mapsto \|1\rangle$, $\|1\rangle \mapsto \|0\rangle$ `Y(qubit)` : $\|0\rangle \mapsto i\|1\rangle$, $\|1\rangle \mapsto -i\|0\rangle$ `Z(qubit)` : $\|0\rangle \mapsto \|0\rangle$, $\|1\rangle \mapsto -\|1\rangle$
Hadamard	`H(qubit)` : $\|0\rangle \mapsto \|+\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$, $\|1\rangle \mapsto \|-\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$
Controlled-NOT	`CNOT(controlQubit, targetQubit)` $\|00\rangle \mapsto \|00\rangle$, $\|01\rangle \mapsto \|01\rangle$, $\|10\rangle \mapsto \|11\rangle$, $\|11\rangle \mapsto \|10\rangle$
Apply several gates (Bell pair example)	`H(qubit1);` `CNOT(qubit1, qubit2);`

Documentation

Quantum Development Kit	`https://docs.microsoft.com/quantum`
Q# Language Reference	`https://docs.microsoft.com/quantum/language/`
Q# Library Reference	`https://docs.microsoft.com/qsharp/api`

Q# Code Repositories

QDK Samples	`https://github.com/Microsoft/Quantum`
QDK Libraries	`https://github.com/Microsoft/QuantumLibraries`
Quantum Katas (tutorials)	`https://github.com/Microsoft/QuantumKatas`

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 .`

Working with Q# Projects

Create new project	`dotnet new console -lang Q# --output project-dir`
Change directory to project directory	`cd project-dir`
Build project	`dotnet build`
Run all unit tests	`dotnet test`