Q# 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`

Derived Types

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

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* `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* `=` *value*
Update mutable symbol (variable)	`set` *name* `=` *value*

Arrays

Allocation	`mutable` *name* `= new` *Type*`[`*length*`]`
Length	`Length(`*name*`)`
i-th element (index is 0-based)	*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 { ... }`

Return a value	`return` *value*
Throw an exception	`fail "`*Exception message*`"`

Debugging

Print a string	`Message("Hello Quantum!")`
Print an interpolated string	`Message($"Value = {`*val*`}")`
Assert that qubit is in $\|0\rangle$ or $\|1\rangle$	`AssertQubit (expected : Result, q : Qubit)`
Print the state of the simulator	`DumpMachine()`

Qubits and Operations on Qubits

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

Pauli gates	`X` : $\|0\rangle \to \|1\rangle$, $\|1\rangle \to \|0\rangle$ `Y` : $\|0\rangle \to i\|1\rangle$, $\|1\rangle \to -i\|0\rangle$ `Z` : $\|0\rangle \to \|0\rangle$, $\|1\rangle \to -\|1\rangle$
Hadamard	`H` : $\|0\rangle \to \|+\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$, $\|1\rangle \to \|-\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$
Controlled-NOT	`CNOT : ((control : Qubit, target : Qubit) => ())` $\|00\rangle \to \|00\rangle$, $\|01\rangle \to \|01\rangle$, $\|10\rangle \to \|11\rangle$, $\|11\rangle \to \|10\rangle$
Measure 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 axis	`Rx : (Double, Qubit) => ()`
	`Ry : (Double, Qubit) => ()`
	`Rz : (Double, Qubit) => ()`

Reset qubit to $\|0\rangle$	`Reset : Qubit => ()`
Reset qubits to $\|0..0\rangle$	`ResetAll : Qubit[] => ()`

Resources

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/`