Variáveis e Estruturas do QDK

In [1]:
```
%version
```

Component	Version
iqsharp	0.11.2006.403
Jupyter Core	1.3.60623.0
.NET Runtime	.NETCoreApp,Version=v3.1

Tipos de Variáveis

- Variáveis Imutáveis:

> O valor desse tipo de variável não pode ser alterado dentro do seu escopo. Seu uso é incentivado porque permite que o compilador execute mais otimizações. Uma associação imutável consiste na palavra-chave "let", seguida por uma tupla de símbolo ou símbolo, um sinal de igual(=), uma expressão para associar os símbolos a ela e um ponto e vírgula de terminação.

In [4]:
```
operation VarImutavel() : Int {
    let x = 10;
    // set x = 4; // Operação inválida
    return x;
}
```

Out[4]:
- VarImutavel

In [5]:
```
%simulate VarImutavel
```

Out[5]: 10

- Variáveis Mutáveis:

> Como alternativa à criação de uma variável com "let", a "mutable" criará uma variável mutável que pode ser revinculada depois de ser inicialmente criada usando o "set".

In [6]:
```
operation VarMutavel() : Int{
    mutable x = 10;
    set x = 4;
    return x;
}
```

Out[6]:
- VarMutavel

In [7]:
```
%simulate VarMutavel
```

Out[7]: 4

Tipos Primitivos

Temos 10 tipos primitivos: Int, BigInt, Double, Bool, Range, String, Unit, Qubit, Pauli, Result

TIPO	Descrição
Int	Números Inteiros com Sinal de 64Bits
BigInt	Inteiro assinado de tamanho arbitrário
Double	Número de ponto flutuante de precisão dupla
Bool	Valor booleano que pode ser true ou false
Range	Representa uma sequência de números inteiros (start..step..stop)
String	Sequência de caracteres.
Unit	Tipo que permite apenas um valor ( )
Qubit	Representa um bit quântico ou qubit
Pauli	Representa os operadores de Pauli de qubit único
Result	Representa o resultado de uma medida (One ou Zero)

Alguns exemplos de declaração e impressão, alguns tipos não foi possível exibí-los:

In [8]:
```
open Microsoft.Quantum.Convert; // Namespace de conversão de tipos

operation TiposPrimitivos() : Unit{
    let inteiro = 10;
    Message("Int: " + IntAsString(inteiro));

    let flutuante = 3.141592;
    Message("Double: " + DoubleAsString(flutuante));

    let boleano = true;
    Message("Bool: " + BoolAsString(boleano));

    let palavra = "Hello World!!!";
    Message("String: " + palavra);

    let bit = One;
    Message("Result: " + BoolAsString(ResultAsBool(bit)));

}
```

Out[8]:
- TiposPrimitivos

In [9]:
```
%simulate TiposPrimitivos
```

```
Int: 10
Double: 3,141592
Bool: True
String: Hello World!!!
Result: True
```

Out[9]: ()

Tipos Definidos pelo Usuário

É possível contruir o seu própio tipo de variável. Como no exemplo:

In [10]:
```
newtype coord = (Int, Int);
operation NovosTipos() : coord{
    let p1 = coord(1,2);
    return p1;
}
```

Out[10]:
- coord
- NovosTipos

In [11]:
```
%simulate NovosTipos
```

Out[11]:
```
coord((1, 2))
```

Temos também os tipos nomeados:

In [12]:
```
newtype complex = (Re: Double, Im: Double);
```

Out[12]:
- complex

In [13]:
```
open Microsoft.Quantum.Convert;
function PrintCp(value : complex) : Unit{
    Message("Re:" + DoubleAsString(value::Re) + " Im:" + DoubleAsSt
ring(value::Im));
}
```

Out[13]:
- PrintCp

In [14]:
```
operation ComplexNumber() : Unit{
    let value = complex(1.0,0.0);
    PrintCp(value);
}
```

Out[14]:
- ComplexNumber

In [15]:
```
%simulate ComplexNumber
```

```
Re:1 Im:0
```

Out[15]:
```
()
```

Estruturas de Decisão

São simples como em qualquer outra liguagem.

> Temos os sinais de comparação semelhantes a linguagem Python: >, <, ==, !=

In [16]:
```
operation estruturasdecisao() : Unit{
    let v = 5;

    if(v > 10){
        Message("Valor Maior que 10");
    }
    elif(v < 10){
        Message("Valor Menor que 10");
    }
    else{ // v == 10
        Message("Valor igual a 10");
    }

    if(v != 10){
        Message("Valor diferente de 10");
    }
}
```

Out[16]:
- estruturasdecisao

In [17]:
```
%simulate estruturasdecisao
```

```
Valor Menor que 10
Valor diferente de 10
```

Out[17]: ()

Estruturas de Repetição

For

In [18]:
```
open Microsoft.Quantum.Convert;
operation For() : Unit{
    let v = 20;
    for(i in 1..2..v){
        Message(IntAsString(i));
    }
}
```

Out[18]:
- For

In [19]:
```
%simulate For
```

```
1
3
5
7
9
11
13
15
17
19
```

Out[19]: ()

While

In [20]:
```
operation While() : Unit{
    let v = 20;
    mutable i = 1;

    while(i < v){
        set i = i + 1;
    }
}
```

```
/snippet_.qs(5,5): error QS4013: While-loops cannot be used in ope
rations. Avoid conditional loops in operations if possible, and us
e a repeat-until-success pattern otherwise.
```

In [21]:
```
open Microsoft.Quantum.Convert;
function WhileRep() : Unit{
    let v = 20;
    mutable i = 1;

    while(i < v){
        Message(IntAsString(i));
        set i = i + 2;
    }
}
```

Out[21]:
- WhileRep

In [22]:
```
%simulate WhileRep
```

```
1
3
5
7
9
11
13
15
17
19
```

Out[22]: ()

In [ ]:
```

```