# Introduction

This project aims to simplify the programming of quantum computing models and shows how to faster develop interactive examples that will help you learn to formulate and solve problems for the D-Wave system using Dann5 d5o library. The d5o library is an extension of D-Wave Ocean's stack of tools, which will allow you to develop integrated code, and to submit it to run in a quantum annealing computer (QAC), either D-Wave Advantage, 2000Q or Hybrid Solver.

The goal of Dann5 project and d5o framework is to provide QAC developers with programming constructs, such as:

- standard types:
  - **Qbit** – quantum bit, where
    - Qbit $\mathbf{x} \in \{ \mathbf{0, 1, S}(\text{uperposition}) \}$
  - **Qbool** – quantum boolean, where
    - Qbool $\mathbf{x} \in \{ \mathbf{T}(\text{rue}), \mathbf{F}(\text{alse}), \mathbf{S}(\text{uperposition}) \}$
  - **Qbin** – quantum binary, as an array of Qbit's
    - Qbin $\mathbf{x} \equiv \text{Qbit}[\,]_{i=0}^{n} \mathbf{x_i}$ where $\mathbf{x_i} \in \mathbf{Qbit}$
  - **Qwhole** - quantum whole numbers (non-negative integer), where
    - ✓ Qwhole $\boldsymbol{x} \in \left\{ \boldsymbol{n} \middle|_{\boldsymbol{n}=\boldsymbol{U}(nknown)\ if\ \mathbf{n_i = S} \mid \mathbf{n_i} \in \mathbf{Qbit} \,\&\, \mathbf{n_i} \in \mathbf{n}\}}^{\boldsymbol{n} \geq \mathbf{0}} \right\}$
  - **Qnnr** – quantum non-negative rational numbers, where
    - Qnnr $\boldsymbol{x} \in \left\{ \boldsymbol{d} \middle|_{\boldsymbol{d}=\boldsymbol{U}(nknown)\ if\ \mathbf{d_i = S} \mid \mathbf{d_i} \in \mathbf{Qbit} \,\&\, \mathbf{d_i} \in \mathbf{d}\}}^{\boldsymbol{d} \in \boldsymbol{R}(ational)\,\&\,\boldsymbol{d} \geq \mathbf{0}} \right\}$
- definitions:
  - **Qvar** – quantum variable
    - ✓ Qvar **a(3, "a")** ⇛ initialize Q variable *a* with *3 quantum bits* in S(uperpostion) state,
    - ✓ Qvar **b("b", 6)** ⇛ initialize Q variable *b* with *value 6*, i.e. $b_0 = 0$, $b_1 = 1$ and $b_2 = 1$
    - Qvar **c("c", 2.5)** ⇛ initialize Q variable c with *binary value 10.1*, i.e. $c_{-1} = 1$, $c_0 = 0$ and $c_1 = 1$
    - Qvar **a(1.2, "d")** ⇛ initialize Q variable *d* with *3 quantum bits*, as an unknown rational number, i.e. $d_{-2} = S$, $d_{-1} = S$ and $d_0 = S$

- **Qequation** – quantum equation statement
  - ✓ Qequation **eX**(Qvar("**X**", **15**)) ⇛ creates a Q equation *eX* with *result* Q variable *defined as A and set to 15*
  - ✓ **eX = a * b** ⇛ for Q variables *a* and *b* (defined above) assigns *multiplication expression* to *eX* Q equation

- **Qcondition** – quantum condition statement
  - ➢ Qcondition **cA(a > b)** ⇛ creates a Q condition cA that compares (above defined) Q variables a and b
  - ➢ **cA << eX** ⇛ adds the (above defined) eX Q equation to be calculated under condition cA, i.e. *a* is bigger than *b*

- **Qroutine** – quantum routine
  - ➢ Qroutine **program("my Q program")** ⇛ creates a Q routine *program*, which allows correlation of programming Q statements
  - ➢ **program << cA << eB** ⇛ adds the (above defined) *cA* Q condition and a predefined *eB* Q equation into *my Q program*

- **Qfunction** – quantum function
  - ➢ Qfunction **myF("my F", a)** ⇛ creates a Q function routine *my F* with Q variable *a* as a return variable
  - ➢ **myF << cA << a + b** ⇛ adds the (above defined) *cA* Q condition and addition Q equation of a and b into *myF*
  - ➢ **eB *= myF** ⇛ updates eB Q equation with a multiplication of its existing expression with a return from myF Q function

- **Qpow, Qmod, Qgcd** – specialized quantum function
  - ✓ Qpow **a_x3(a, 3)** ⇛ creates a power Q function routine *a_x3* with Q variable *a* as a base and exponent 3
  - ➢ Qmod **b_m3(b, 3)** ⇛ creates a modulo Q function routine *b_m3* with Q variable *b* as a dividend and divisor 3
  - ➢ Qgcd **a_b(a, b)** ⇛ creates a greatest common divisor Q function routine *a_b* for Q variables *a* and *b*

- standard operations:
  - **assignment** – of quantum variable or expression to the Q equation
  - **unary operators**: ~ (`inversion`)
  - **binary operators**: +, *, -, /
  - **logical operators**: &, |, ^, nand, nor
  - **comparison operators**: ==, !=, >, >=, <, <=

- D-Wave Ocean integration:
  - **qubo()** – returns Qubo binary quadratic model (BQM)
    - ✓ eX**.qubo()** ⇛ for *eX* Q equation (defined above) returns a *Qubo* dictionary object for execution on an *exact* or *QAC solver*

- **samples()** – receives a dictionary of samples returned by a solver
  - ✓ eX.**samples(lowSamples)** ⇛ sets resulting samples returned from a solver into $eXQ$ equation, which has been used to create Qubo BQM

- **solutions()** – returns solutions of unknown variables
  - ✓ eX.**solutions()** ⇛ for $eXQ$ equation (defined above) with expected result 15 and $b$ set to 6 returns solutions for undefined Q variable $a$

**Note :** ✓ released features, ➢ features in development, ▪ planned features.

For additional information on how to use features of d5o refer to examples.

# Install & Use

## Installation prerequisites

To install and use Dann5 d5o precompiled library

- install Python 3; anaconda3 distribution of the version 3.8 has been used to test released functionality
- install D-Wave Ocean Tools into python environment; follow Installing Ocean Tools
- install pybind11 into the same python environment; follow Installing the library

To compile Dann5 d5o source code in addition to above prerequisites

- install Eigen by following download instructions

## Installation

Dann5 d5o library is in active development on Windows x64 as a primary OS. A Mac OS X/Linux version and package installation will be released soon.

- Install Windows x64 version from this repository into python project or environment
  1. Download **d5o.cp38-win_amd64.pyd** file from debug folder
  2. Create ***Dann5*** as a sub-folder of
     - your python project folder; or
     - site-packages folder for your python environment, e.g. for anaconda distribution, on windows, the path should be:
     *C:\Users\<user_name>\AppData\Local\Continuum\anaconda3\envs\<env_name>\Lib\site-packages*
  3. Copy **d5o.cp38-win_amd64.pyd** into ***Dann5*** sub-folder

- Install package from source code
    1. Simply clone or download the dann5 repository
        - If using the MS Visual Studio dann5.ocean.sln, make sure dann5 and Eigen are sub-folders in the same folder.
    2. Build
        - d5o.cp38-win_amd64.pyd for windows and python x64
        - d5o.cp38-win_amd32.pyd for windows and python x32
        - d5o.so for Mac OS X or Linux
    3. Deploy d5o library into Dann5 folder within your python project or environment as described above in Windows x64 version installation

### Use

After installing d5o library as per instructions above, use released features in your python source code:

```
from dann5.d5o import Qvar, Qequation
a = Qvar(4, "a")
b = Qvar(3, "b")
eA = Qequation(Qvar("A", 15))
eA.assign( a + b )
print(eA.toString(True, -1))
print(eA.qubo(True, -1))
```

## Bugs, issues and contributing

Contributions to this project are welcome.

**Note:** if you have an issue, please send me the code example.