```
Module openql.openql
--------------------
‘OpenQL‘ is a C++/Python framework for high-level quantum programming. The framework provides
a compiler for compiling and optimizing quantum code. The compiler produces the intermediate q
uantum assembly language in cQASM (Common QASM) and the compiled eQASM (executable QASM) for v
arious target platforms. While the eQASM is platform-specific, the quantum assembly code (QASM
) is hardware-agnostic and can be simulated on the QX simulator.

Functions
---------

get_option(option_name)
    Returns value of any of the following OpenQL options:
         Opt. Name   :   Default     : Possible values
        ’log_level’  : LOG_NOTHING   : ’LOG_{NOTHING/CRITICAL/ERROR/WARNING/INFO/DEBUG}’
        ’output_dir’ : ’test_output’ : <output directory>
        ’scheduler’  : ’ASAP’        : ’ASAP/ALAP’
        ’use_default_gates’ : ’yes’  : ’yes/no’
        ’optimize’   :   ’no’        : ’yes/no’
        ’decompose_toffoli’ :  ’no’  : ’yes/no’

    Parameters
    ----------
    arg1 : str
        Option name

    Returns
    -------
    str
        Option value

get_version()
    Returns OpenQL version

    Parameters
    ----------
    None

    Returns
    -------
    str
        version number as a string

print_options()
    Prints a list of available OpenQL options with their values.

set_option(option_name, option_value)
    Sets any of the following OpenQL options:
         Opt. Name   :   Default     : Possible values
        ’log_level’  : LOG_NOTHING   : ’LOG_{NOTHING/CRITICAL/ERROR/WARNING/INFO/DEBUG}’
        ’output_dir’ : ’test_output’ : <output directory>
        ’scheduler’  : ’ASAP’        : ’ASAP/ALAP’
        ’use_default_gates’ : ’yes’  : ’yes/no’
        ’optimize’   :   ’no’        : ’yes/no’
        ’decompose_toffoli’ :  ’no’  : ’yes/no’

    Parameters
    ----------
    arg1 : str
        Option name
    arg2 : str
        Option value
```

```
vectord_swigregister(...)

vectorf_swigregister(...)

vectori_swigregister(...)

vectorui_swigregister(...)

Classes
-------
CReg
    Classical register class.

    Ancestors (in MRO)
    ------------------
    openql.openql.CReg
    builtins.object

    Class variables
    ---------------
    creg

    thisown

    Static methods
    --------------
    __init__(self)
        Constructs a classical register which can be source/destination for classical operatio
ns.

        Parameters
        ----------
        None

        Returns
        -------
        CReg
            classical register object

    Instance variables
    ------------------
    creg

    thisown
        The membership flag

Kernel
    Kernel class which contains various quantum instructions.

    Ancestors (in MRO)
    ------------------
    openql.openql.Kernel
    builtins.object

    Class variables
    ---------------
    creg_count

    kernel

    name

    platform
```

```
    qubit_count

    thisown

    Static methods
    --------------
    __init__(self, name, platform, qubit_count, creg_count=0)
        Constructs a Kernel object.

        Parameters
        ----------
        arg1 : str
            name of the Kernel
        arg2 : Platform
            target platform for which the kernel will be compiled
        arg3 : int
            qubit count
        arg4 : int
            classical register count

    barrier(self, *args, **kwargs)
        inserts explicit barrier on specified qubits. wait with duration '0'
            is also equivalent to appliying barrier on specified list of qubits.
            If no qubits are specified, then barrier is applied on all the qubits.

        Parameters
        ----------
        arg1 : []
            list of qubits

    classical(self, *args)
        adds classical operation kernel.

        Parameters
        ----------
        arg1 : CReg
            destination register for classical operation.
        arg2 : Operation
            classical operation.

    clifford(self, id, q0)
        Applies clifford operation of the specified id on the qubit.

        Parameters
        ----------
        arg1 : int
            clifford operation id
        arg2 : int
            target qubit

        The ids and the corresponding operations are:
        0 : ['I']
        1 : ['Y90', 'X90']
        2 : ['mX90', 'mY90']
        3 : ['X180']
        4 : ['mY90', 'mX90']
        5 : ['X90', 'mY90']
        6 : ['Y180']
        7 : ['mY90', 'X90']
        8 : ['X90', 'Y90']
        9 : ['X180', 'Y180']
        10: ['Y90', 'mX90']
```

```
        11: ['mX90', 'Y90']
        12: ['Y90', 'X180']
        13: ['mX90']
        14: ['X90', 'mY90', 'mX90']
        15: ['mY90']
        16: ['X90']
        17: ['X90', 'Y90', 'X90']
        18: ['mY90', 'X180']
        19: ['X90', 'Y180']
        20: ['X90', 'mY90', 'X90']
        21: ['Y90']
        22: ['mX90', 'Y180']
        23: ['X90', 'Y90', 'mX90']

    cnot(self, q0, q1)
        Applies controlled-not operation.

        Parameters
        ----------
        arg1 : int
            control qubit
        arg2 : int
            target qubit

    conjugate(self, k)
        generates conjugate version of the kernel from the input kernel.

        Parameters
        ----------

        arg1 : ql::Kernel
            input kernel. Except measure, Kernel to be conjugated.

        Returns
        -------
        None

    controlled(self, k, control_qubits, ancilla_qubits)
        generates controlled version of the kernel from the input kernel.

        Parameters
        ----------

        arg1 : ql::Kernel
            input kernel. Except measure, Kernel to be controlled may contain any of the defau
lt gates as well custom gates which are not specialized for a specific qubits.

        arg2 : []
            list of control qubits.

        arg3 : []
            list of ancilla qubits. Number of ancilla qubits should be equal to number of cont
rol qubits.

        Returns
        -------
        None

    cphase(self, q0, q1)
        Applies controlled-phase operation.

        Parameters
        ----------
```

```
        arg1 : int
            control qubit
        arg2 : int
            target qubit

    cz(self, q0, q1)

    display(self)
        inserts QX display instruction (so QX specific).

        Parameters
        ----------
        None

        Returns
        -------
        None

    gate(self, *args)
        adds custom/default gates to kernel.

        Parameters
        ----------
        arg1 : str
            name of gate
        arg2 : []
            list of qubits
        arg3 : CReg
            classical destination register for measure operation.

    get_custom_instructions(self)
        Returns list of available custom instructions.

        Parameters
        ----------
        None

        Returns
        -------
        []
            List of available custom instructions

    hadamard(self, q0)
        Applies hadamard on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    identity(self, q0)
        Applies identity on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    measure(self, q0)
        measures input qubit.

        Parameters
        ----------
```

```
        arg1 : int
            input qubit

    mrx90(self, q0)
        Applies mrx90 on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    mry90(self, q0)

    prepz(self, q0)

    rx(self, q0, angle)

    rx180(self, q0)
        Applies rx180 on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    rx90(self, q0)
        Applies rx90 on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    ry(self, q0, angle)

    ry180(self, q0)
        Applies ry180 on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    ry90(self, q0)

    rz(self, q0, angle)

    s(self, q0)
        Applies x on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    sdag(self, q0)
        Applies sdag on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit
```

```
    t(self, q0)

    tdag(self, q0)

    toffoli(self, q0, q1, q2)
        Applies controlled-controlled-not operation.

        Parameters
        ----------
        arg1 : int
            control qubit
        arg2 : int
            control qubit
        arg3 : int
            target qubit

    wait(self, qubits, duration)
        inserts explicit wait on specified qubits. wait with duration '0'
            is equivalent to barrier on specified list of qubits. If no qubits
            are specified, then wait/barrier is applied on all the qubits.

        Parameters
        ----------
        arg1 : []
            list of qubits
        arg2 : int
            duration in ns

    x(self, q0)

    y(self, q0)
        Applies y on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    z(self, q0)
        Applies z on the qubit specified in argument.

        Parameters
        ----------
        arg1 : int
            target qubit

    Instance variables
    ------------------
    creg_count

    kernel

    name

    platform

    qubit_count

    thisown
        The membership flag

Operation
    Operation class representing classical operations.
```

```
    Ancestors (in MRO)
    ------------------
    openql.openql.Operation
    builtins.object

    Class variables
    ---------------
    operation

    thisown

    Static methods
    --------------
    __init__(self, *args)
        Constructs an Operation object (used for initializing with immediate values).

        Parameters
        ----------
        arg1 : int
            immediate value

    Instance variables
    ------------------
    operation

    thisown
        The membership flag

Platform
    Platform class specifiying the target platform to be used for compilation.

    Ancestors (in MRO)
    ------------------
    openql.openql.Platform
    builtins.object

    Class variables
    ---------------
    config_file

    name

    platform

    thisown

    Static methods
    --------------
    __init__(self, *args)
        Constructs a Platform object.

        Parameters
        ----------
        arg1 : str
            name of the Platform
        arg2 : str
            name of the configuration file specifying the platform

    get_qubit_number(self)
        returns number of qubits in the platform.

        Parameters
```

```
        ----------
        None

        Returns
        -------
        int
            number of qubits

    Instance variables
    ------------------
    config_file

    name

    platform

    thisown
        The membership flag

Program
    Program class which contains one or more kernels.

    Ancestors (in MRO)
    ------------------
    openql.openql.Program
    builtins.object

    Class variables
    ---------------
    creg_count

    name

    platform

    program

    qubit_count

    thisown

    Static methods
    --------------
    __init__(self, *args)
        Constructs a program object.

        Parameters
        ----------
        arg1 : str
            name of the program
        arg2 : Platform
            instance of an OpenQL Platform
        arg3 : int
            number of qubits the program will use
        arg4 : int
            number of classical registers the program will use (default: 0)

    add_do_while(self, *args)
        Adds specified sub-program to a program which will be repeatedly executed while specif
ied condition is true.

        Parameters
        ----------
```

```
        arg1 : Program
            program to be executed repeatedly
        arg2: Operation
            classical relational operation (<, >, <=, >=, ==, !=)

    add_for(self, *args)
        Adds specified sub-program to a program which will be executed for specified iteration
s.

        Parameters
        ----------
        arg1 : Program
            sub-program to be executed repeatedly
        arg2: int
            iteration count

    add_if(self, *args)
        Adds specified sub-program to a program which will be executed if specified condition
is true. This allows nesting of operations.

        Parameters
        ----------
        arg1 : Program
            program to be executed
        arg2: Operation
            classical relational operation (<, >, <=, >=, ==, !=)

    add_if_else(self, *args)
        Adds specified sub-programs to a program. First sub-program will be executed if specif
ied condition is true. Second sub-program will be executed if specified condition is false.

        Parameters
        ----------
        arg1 : Program
            program to be executed when specified condition is true (if part).
        arg2 : Program
            program to be executed when specified condition is false (else part).
        arg3: Operation
            classical relational operation (<, >, <=, >=, ==, !=)

    add_kernel(self, k)
        Adds specified kernel to program.

        Parameters
        ----------
        arg1 : kernel
            kernel to be added

    add_program(self, p)

    compile(self)
        Compiles the program.

        Parameters
        ----------
        None

    get_sweep_points(self)
        Returns sweep points for an experiment.

        Parameters
        ----------
        None
```

```
        Returns
        -------
        []
            list of sweep points

    microcode(self)
        Returns program microcode
        Parameters
        ----------
        None

        Returns
        -------
        str
            microcode

    print_interaction_matrix(self)

    qasm(self)
        Returns program QASM
        Parameters
        ----------
        None

        Returns
        -------
        str
            qasm

    set_sweep_points(self, *args)
        Sets sweep points for an experiment.

        Parameters
        ----------
        arg1 : []
            list of sweep points

    write_interaction_matrix(self)

    Instance variables
    ------------------
    creg_count

    name

    platform

    program

    qubit_count

    thisown
        The membership flag
```