Porting to KMK
Porting a board to KMK is quite simple, and follows this base format.

```python
import board

from kmk.kmk_keyboard import KMKKeyboard as _KMKKeyboard
from kmk.scanners import DiodeOrientation
{EXTENSIONS_IMPORT}

class KMKKeyboard(_KMKKeyboard):
{REQUIRED}
    extensions = []

```

REQUIRED
This is designed to be replaced with the defining pins of your keyboard. Rows, columns and the diode direction (if any), should be defined like this
```python
    row_pins = [board.p0_31, board.p0_29, board.p0_02, board.p1_15]
    col_pins = [board.p0_22, board.p0_24, board.p1_00, board.p0_11, board.p1_04]
    diode_orientation = DiodeOrientation.COL2ROW
```

Additional pins for extensions
KMK includes built in extensions for RGB and split keyboards, and powersave. If these are applicable on your keyboard/microcontroller, the pins should be added here. Refer to the instructions on the respective extensions page on how to add them. If not adding any extensions, leave this as an empty list as shown.

Coord mapping
If your keyboard is not built electrically as a square (though most are), you can provide a mapping directly. An example of this is the [Corne](https://github.com/foostan/crkbd). That has 12 columns for 3 rows, and 6 columns for the bottom row. Split keyboards count as the total keyboard, not per side, the right side being offset by the number of keys on the left side, as if the rows were stacked.
That would look like this:
```python
coord_mapping = [
 0,  1,  2,  3,  4,  5,  24, 25, 26, 27, 28, 29,
 6,  7,  8,  9, 10, 11,  30, 31, 32, 33, 34, 35,
12, 13, 14, 15, 16, 17,  36, 37, 38, 39, 40, 41,
            21, 22, 23,  42, 43, 44,
]
```

Note: Not all numbers are necessarily used ! The keyboard assumes `number of line * number of rows` keys. Some of the possible keys might not be used. For example a keyboard with 60 keys might have 8 rows, 8 cols, allowing 64 total combinations -- hence 64 keys. 4 numbers will then not be used for keys in the `coord_mapping` (might be anyone of them depending of the wiring).

Find your `coord_mapping`
The following code will help you setup your `coord_mapping` by having every key send its corresponding number. Use it after your pins and module definition to define both `keyboard.coord_mapping` and `keyboard.keymap`.

```python
from kmk.handlers.sequences import simple_key_sequence
from kmk.keys import KC

# *2 for split keyboards, which will typically manage twice the number of keys
```

```
# of one side. Having this N too large will have no impact (maybe slower boot..)
N = len(keyboard.col_pins) * len(keyboard.row_pins) * 2

keyboard.coord_mapping = list(range(N))

layer = []

for i in range(N):
    c, r = divmod(i, 100)
    d, u = divmod(r, 10)
    layer.append(
        simple_key_sequence(
            (
                getattr(KC, 'N' + str(c)),
                getattr(KC, 'N' + str(d)),
                getattr(KC, 'N' + str(u)),
                KC.SPC,
            )
        )
    )
keyboard.keymap = [layer]

if __name__ == '__main__':
    keyboard.go()
```

Keymaps
Keymaps are organized as a list of lists. Keycodes are added for every key on
each layer. See [keycodes](keycodes.md) for more details on what keycodes are
available. If using layers or other extensions, also refer to the extensions
page for additional keycodes.
```python
from kb import KMKKeyboard
from kmk.keys import KC

keyboard = KMKKeyboard()

keyboard.keymap = [
    [KC.A, KC.B],
    [KC.C, KC.D],
]

if __name__ == '__main__':
    keyboard.go()
```

More information
More information on keymaps can be found in the [config and keymap](config_and_keymap.md)
documentation.