# Configuring KMK

KMK is configured through a rather large, plain-old-Python class called `KMKKeyboard`. Subclasses of this configuration exist which pre-fill defaults for various known keyboards (for example, many QMK, TMK, or ZMK keyboards are supported with a nice!nano, or through our ItsyBitsy to Pro Micro pinout adapter.) This class is the main interface between end users and the inner workings of KMK. Let's dive in!

- Edit or create a file called `main.py` on your `CIRCUITPY` drive. You can also keep this file on your computer (perhaps under `user_keymaps` - please feel free to submit a pull request with your layout definitions!) and copy it over (either manually or, if you're adept with developer tooling and/or a command line, [our Makefile](flashing.md)).

  It's definitely recommended to keep a backup of your configuration somewhere that isn't the microcontroller itself - MCUs die, CircuitPython may run into corruption bugs, or you might just have bad luck and delete the wrong file some day.

- Assign a `KMKKeyboard` instance to a variable (ex. `keyboard = KMKKeyboard()` - note the parentheses).

- Make sure this `KMKKeyboard` instance is actually run at the end of the file with a block such as the following:

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

- Assign pins and your diode orientation (only necessary on handwire keyboards), for example:

```python
import board

from kmk.scanners import DiodeOrientation

col_pins = (board.SCK, board.MOSI, board.MISO, board.RX, board.TX, board.D4)
row_pins = (board.D10, board.D11, board.D12, board.D13, board.D9, board.D6, board.D5, board.SCL)
rollover_cols_every_rows = 4
diode_orientation = DiodeOrientation.COL2ROW
```

The pins should be based on whatever CircuitPython calls pins on your particular board. You can find these in the REPL on your CircuitPython device:

```python
import board
print(dir(board))
```

> Note: `rollover_cols_every_rows` is only supported with
> `DiodeOrientation.COLUMNS`/`DiodeOrientation.COL2ROW`, not
`DiodeOrientation.ROWS`/`DiodeOrientation.ROW2COL`. It is used for boards
> such as the Planck Rev6 which reuse column pins to simulate a 4x12 matrix in
> the form of an 8x6 matrix

- Import the global list of key definitions with `from kmk.keys import KC`. You can either print this out in the REPL as we did with `board` above, or simply look at [our Key documentation](keycodes.md).
  We've tried to keep that listing reasonably up to date, but if it feels like something is missing, you may need to read through `kmk/keys.py` (and then

  open a ticket to tell us our docs are out of date, or open a PR and fix the
  docs yourself!)

- Define a keymap, which is, in Python terms, a List of Lists of `Key` objects.
  A very simple keymap, for a keyboard with just two physical keys on a single
  layer, may look like this:

```python
keyboard.keymap = [[KC.A, KC.B]]
```

- The keymap contains a flat list of `Key` objects for each layer of the keyboard.
  The list of keys in each layer are stored as a single list that follows the
  grid of row and column pins in the keyboard matrix.  This list starts with keys
  in the first row from left to right, then the second row, and so on.
  The row x column matrix structure doesn't appear explicitly
  in the keymap.  Use `KC.NO` to mark grid positions without a physical key.
  For very sparse grids `keyboard.coord_mapping` can be useful to avoid `KC.NO`.

You can further define a bunch of other stuff:

- `keyboard.debug_enabled` which will spew a ton of debugging information to the serial
  console. This is very rarely needed, but can provide very valuable information
  if you need to open an issue.

- `keyboard.tap_time` which defines how long `KC.TT` and `KC.LT` will wait before
  considering a key "held" (see `layers.md`).