Main Functions and Members

None of the movement commands are *blocking*, meaning that if you tell the robot to walk for two seconds, your program will keep running while that's happening rather than wait for the walk action to finish. This can have both useful and undesirable side effects, so just bear this in mind. Secondly, Marty *queues* actions, so if you send a walk, kick and then another walk they will all happen one-after the other, *not* at the same time or overwriting each-other.

```
martypy.Marty(url='socket://192.168.0._', client_types=dict(),
default_lifelike=True, *args, **kwargs)
```

Class constructor for a Marty client instance, with a default URL given.

When you create a `Marty` instance,
the `enable_safeties(True)` and `enable_motors(True)` commands are sent to the Robot. If the kwarg `default_lifelike` is True, lifelike behaviours will also be enabled.

`*args` and `**kwargs` are passed on to the client type, which will be chosen depending on the protocol specified the URL. Currently the natively supported client types are `socket`, `serial`, `ros` and `test`.

For instance, the `debug` keyword argument can be useful for showing what the client is sending, and other stuff over the Socket API when using the `socket` client type.

For more info on extending the available client types via `client_types` see here.

```
hello()
```

Moves to the zero position and wiggles the eyebrows. Be careful with this, as if the Robot doesn't know where it it, it will move as quickly as it can to the zero positions, which can knock the robot over. It's best used as the first command you send when you turn the Robot on, where it should be close to the zero pose.

```
stop(stop_type=None)
```

Stop the robot moving. `stop_type` is a str which should be a key in the `Marty.STOP_TYPE` dict. If it is none (the default) then `'clear and stop'` will be assumed. Other options are:

clear queue
clear movement queue only (so finish the current movement)

clear and stop
clear movement queue and servo queues (freeze in-place)

clear and disable
clear everything and disable motors

clear and zero
clear everything, and make robot return to zero

pause
pause, but keep servo and movequeue intact and motors enabled

pause and disable
as above, but disable motors too

```
move_joint(joint_id, position, move_time)
```

Move a specific joint, selected by `joint_id` (0 to 8) to `position` (-128 to 127) taking `move_time` milliseconds

```
lean(direction, amount, move_time)
```

Lean over in a direction, taken from `SIDE_CODES` taking `move_time` milliseconds

```
walk(num_steps=2, start_foot='auto', turn=0, step_length=40,
move_time=1500)
```

Instructs the robot to start walking, with defaults set for all parameters. `move_time` is in milliseconds (1/1000 of a second), `step_length` is *roughly* millimetres. `turn` is an int8 in the range -128 to 127. 0 makes the robot walk straight ahead, negatives move to the Robot's *right*, positives to the *left* (i.e. positive Y-Axis direction).

```
eyes(angle, move_time=100)
```

Move the eyes to `angle` position, taking `move_time` milliseconds

```
kick(side='right', twist=0, move_time=2000)
```

Kick with the foot on `side` (again from `SIDE_CODES`) taking `move_time` milliseconds. The `twist` (-128 to 127) argument adds a knee twist to the kick. 0 is straight ahead.

```
arms(left_angle, right_angle, move_time)
```

Move the arms to each respective angle, taking `move_time` milliseconds

```
celebrate(move_time=4000)
```

Do a little celebration, taking `move_time` milliseconds. The default is sensible, though this is hilarious/adorable with a move time around 1000.

```
circle_dance(side, move_time)
```

Makes Marty do a little dance in a circular motion. `side` should be a str from `SIDE_CODES`.

```
sidestep(side, steps, step_length, move_time)
```

Walk sideways to `side` (from `SIDE_CODES`) with *roughly* millimetre `step_length` taking `move_time`

```
play_sound(freq_start, freq_end, duration)
```

Play a tone that linearly interpolates between the Frequency `freq_start` in Hz to `freq_end`, taking `duration` milliseconds to play.

```
get_battery_voltage()
```

Returns a `float` that is the reading of the battery voltage from the board.

```
get_accelerometer(axis)
```

Returns a `float` reading of the `axis = 'x' or 'y' or 'z'` from the board. Note the axes are marked on the control board, see here also for conventions and orientation.

The readings are in Gs, i.e. `1` represents an acceleration of $9.81 ms^{-2}$ in that direction.

```
get_motor_current(motor_id)
```

Returns a `float` of the current detected on the `motor_id`'s channel by the control board

```
pinmode_gpio(gpio, mode)
```

Configure a GPIO pin's function. `mode` should be a str from `GPIO_PIN_MODES`.

```
digitalread_gpio(gpio)
```

Returns the `HIGH`/`LOW` state of a GPIO pin (0 to 8) as `True` or `False`

```
write_gpio(gpio, value)
```

Write a value to a GPIO port. Acceptable value types depend on the GPIO configuration.

```
enable_motors(enable=True, clear_queue=True)
```

Toggles whether the motors are enabled (allowed to move) or disabled, where they can freely move. This is called just before the `Marty` constructor completes.

Enable motors also *unpauses* the movement queue (see stop) so you probably also want to send `stop('clear queue')` before enabling motors to prevent a jump. This is done by default with the `clear_queue` argument.

```
lifelike_behaviour(enable=True)
```

If enabled, Marty will perform a short action every minute or so to remind you that it's on. This is disabled by default, but we'd recommend turning it on.

If Lifelike Behaviours are on, they will **still** happen even if motors are disabled. To prevent them from happening you need to explicitly disable Lifelike Behaviours.

```
SIDE_CODES
```

A str, one of `'left'`, `'right'`, `'forward'` and `'backward'`. These are relative to Marty's facing direction.

```
STOP_TYPE
```

A str, one of `'clear queue'`, `'clear and stop'`, `'clear and disable'`, `'clear and zero'`, `'pause'` and `'pause and disable'`

`GPIO_PIN_MODES`

A str, one of `'digital in'`, `'analog in'` and `'digital out'`.