--:! {move: []: (:smiley:, step: []: (:smiley:: (:smiley: } <- **Move and Step Function Libraries** -> muse/docs/lib/motion.md
--:neutral_face: motion: *Libraries to move turtles and move turtles by steps allowing operations at each step.* -> motion, move, step
--:+ move: **Position setting, tracking, and reporting by dead reckoning checked by fuel consumption.**
--:+ step: **Iterators (closures) for moving block by block, potentially doing operations at each block.**

--:# *Provide fuel level check to validate a dead reckoning move, can track movement for retracing move as a trail.*
--:+ *Report error conditions `"blocked"`, `"lost"` (for apparent but invalid movement), `"empty"` (for no fuel).*
--:+ *Throw some errors as tables rather than strings to allow for attempted recovery operations.*

--:# **State variables for turtle motion:** (maintained across programs within session, not persistent beyond that).

--:> situation: *Dead reckoning* -> `{position:position, facing:facing, fuel: situation.fuel, level: situation.level}`

--:> situation.fuel: *Simulated fuel level checked against reported fuel to validate dead reckoning* -> `#:`

--:> situation.level: *For tracking* -> `"same"|"rise"|"fall"`

--:> position: *Computercraft co-ordinates (+x east, +y up, +z south)* -> `{x: #:, y: #:, z: #:}`

--:> facing: *For movement in four NESW cardinal directions* -> `"north"|"east"|"south"|"west"`

--:> situations: *Tracking history* -> `situation[]`

--:## **Some Utilities: position reporting and setting:**

--:: move.get(:situation:?) -> *Default current situation.* -> `x: #:, y: #:, z: #:, facing: ":", fuel: #:, level: ":"`

--:: move.track(enable: ^:) -> *Set tracking condition* -> `enable: ^:`

--:: move.set(x: #:, y: #:, z: #:, f: facing?, fuel: #:??, level: ":"???) -> *Set position, optionally rest of situation.* -> `nil`

--:: move.situations(:situations:) -> *Set `_G.Muse.situations` to situations.* -> situations

--:: move.clone() -> *Clone current situation* -> situation

--:: move.clones() -> *Deep copy `_G.Muse.situations`.* -> situations

--:: move.at(:situation:?) -> *(Current) situation xyzf.* -> `xyzf`

--:: move.ats(:situation:?) -> *(Current) situation position and facing string (`""` in game if not turtle).* -> `xyzf: ":"`

--:: move.where(tx: #:?, ty: #:?, tz: #:?, tf: ":"?) -> *Returns GPS results if available.* -> `x: #:, y: #:, z: #:, facing: ":", ^: ok`
--:+ *If no GPS, returns the optional (testing) parameters or, if not supplied, current dead reckoning position in situation.*

--:> recovery: *For some errors* -> `{call: ":", failure: ":", cause: ":", remaining: #:, :xyzf:, :direction:, operation: ":"}`

--:# **Forward! Up! Down! move, step ... again (raising errors, providing for recovery)**

--:# **Tracking Movement: completing movement**

--:# **Exposed APIs for move functions: turn left|right or face cardinal if needed, then repeat count forward**

--:: move.moves(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.left(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.right(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.north(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.east(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.south(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.west(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.up(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.down(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.forward(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:: move.back(count: #:?) -> *Count 0: just turn, 1: default* -> `"done", remaining: #:, xyzf, direction &!recovery`

--:# **Exposed APIs for step functions: turn or face direction if needed then step count forward in that direction**

--:: step.steps(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:> stepping: *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.left(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.right(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.north(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.east(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.south(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.left(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.up(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.down(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.forward(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:: step.back(count: #:?) -> *Iterator (default 1 step)* -> `(): "done", remaining: #:, xyzf, direction &!recovery`

--:# **Move or Step to target xyzf position**

--:: move.to(xyzf: xyzf, first: ":"?) -> *Current situation to x, z, y, and optionally face.* -> `"done", #:, xyzf &!recovery`
--:+ *Optional argument* `first` *is "x", "y", or "z" to select first move in that direction to deal with blockages.*

--:: step.to(:xyzf:, situation:situation?) -> *Step to position from (current) sItuation.* -> `(:): nil &!recovery`
--:+ *Iterate first in x direction to completion, then z, and finally y. Once complete, each iterator is exhausted.*
--:+ *Finally turn to face if supplied. Returned iterator returns* `nil` *when iterators for all directions are exhausted.*