

```
--:! {turtle: []: (😊) } <- Turtle Operations Overlay Library -> muse/docs/lib/turtle.md
```

--😊 turtle: *Replaces game definitions, unifies operations to all directions: north, east, south, west, up, down.* -> turtle

--:+ *Provides low level item finding, naming and turtle inventory utilities; out-of-game simulated blocking.*

--:> direction: *Four compass points and verticals* ->
"north" | "east" | "south" | "west" | "up" | "down"

--:# **Turtle operations north, east, south, west, up, down**

--:# **Operation dictionaries keyed by direction, values are generally functions of no arguments calling which return a boolean.**

--:> turtle.attacks: *Attack in direction and return attack success.* -> **[:direction:]: (): ^:, ":"?**

--:> turtle.compares: *Check block in direction has the same ID as selected slot* -> **[:direction:]: (): same: ^:**

--:> turtle.detects: *Check block in direction is solid: not air, mob, liquid or floater.* ->
[:direction:]: (): ^:

--:> turtle.digs: *Try to dig block in direction and call suck().* -> **[:direction:]: (side: ":"?): ^:, ":"?**

--:+ *Sucked items go to inventory. If a hoe is used to attempt to "dig" a dirt block, it will be tilled instead.*

--:+ *Tilling is also possible if the space in front of the turtle is empty but dirt exists below that point.*

--:> turtle.drops: *Drop count [or all] items in selected slot to inventory.* -> **[:direction:]: (count: #:?): ^:, ":"?**

--:+ *Returned function drops and returns **false** if there's inventory on the side specified by direction which is full.*

--:> turtle.inspects: *If true, get detail block information in direction.* -> **[:direction:]: (): ^:, detail?**

--:> turtle.puts: *Attempt placing block of the selected slot in direction.* -> **[:direction:]: (text: ":"?): ^:, ":"?**

--:+ *Collects water or lava if the currently selected slot is an empty bucket. Text is used for placed sign.*

--:+ *Value of **turtle.puts[:direction:]** is a function of one optional argument calling which returns a boolean.*

--:> turtle.sucks: *Move count [or all] from direction to inventory.* -> **[:direction:]: (count: #:?): ^:, ":"?**

--:+ *Move from ground or first non empty slot of adjacent inventory enabled block to selected or next turtle slot.*

--:+ *Value of **turtle.sucks[:direction:]** is function of one optional argument calling which returns a boolean.*

--:# Function References

--:: turtle.find(targets: ":"[]) -> *Selects found slot.* -> **detail?**

--:: turtle.select(slot: #😊 -> *Attempts to select the specified slot.* -> **selected: ^:**

--:: turtle.item(slot: #:?) -> *Detail of specified or currently selected slot.* -> **nil | detail**

--:# Item name and turtle status utilities

--:: turtle.inventory() -> *Returns current turtle inventory as turtle detail table.* -> **detail[]**

--:- items -> *Returns items in turtle inventory as string.*

--:: turtle.check(targets: ":"[], :detail:) -> *Tries to match each target against **detail.name**.* -> **matched: ^`**

--:# *Categories provide names for sets of minecraft items.*

--:> ore: *Minecraft* ->

"minecraft:coal_ore" | "minecraft:iron_ore" | "minecraft:lapis_ore" | "minecraft:gold_ore" | "minecraft:diamond_ore" | "minecraft:redstone_ore" | "minecraft:emerald_ore" | "minecraft:nether_quartz_ore" | "minecraft:prismarine" | "minecraft:obsidian"

--:> ores: *Category* -> **ore[]**

--:> minecraft: *For Language Server* -> ":"

--:> group: *Materials* -> **"fuel" | "ore" | "fill" | "dirt" | "stone" | "fence" | "test"**

--:# **Fence material specified by short name (e.g. **oak**) along points specified by **range****

--:> fencings: *Wooden materials* -> **"birch" | "acacia" | "bamboo" | "cherry" | "chrimson" | "dark oak" | "mangrove" | "oak"**

--:: turtle.category(name: "😊 -> *Names in category or fencings matching **name** or {"minecraft:":..name}.* -> **":"[]**

--:- fueling -> *Returns energy available in turtle slots.*

--:: turtle.fuel() -> *Total energy actually available in turtle slots plus turtle fuel level.* -> **fuelTotal: #:**

--:: turtle.unblock(direction: ":", limit: #:?) -> *Retrys (default **_G.Muse.attempts**) dig to limit or bedrock.* -> **"done", nil | "undug" &!**

--:+ *Returns "done, "undug" if dig attempt was for air, water, or lava. Raises error for bedrock or dig limit reached.*

--:: turtle.digTo(xyzf:, limit: #:?) -> *Unblocking move.* -> **code: ":", remaining: #:, xyzf: ":" &: &!**

--:+ *Try to move to position, dig to unblock if needed, catch (table) and raise error(string) for "lost" or "empty".*

2025-11-25

--: + Also catch and raise error (string) if attempt to dig to unblock failed for bedrock or other reason.

--: + Normally return just what a successful move would: "done", 0 remaining, current position.

--: turtle.digAround(orientation: ":", name: ":", diggings: "[:]") -> Unblocking dig. -> "done" &: &!

--: + Dig (unblocking) in diggings directions, catch failure and raise error(string) re-orienting in original orientation.

--: turtle.block(blocked: ^:) -> Out-of-game debug: sets blocking for simulating turtle being blocked.

-> blocked: ^:

--: turtle.blocking(^:) -> Isolate global to control blocking for out-of-game debug. -> ^: