--:! {core: []: (:smiley:} <- **Core Functions Library** -> muse/docs/lib/core.md
--:neutral_face: core: *Strings, session state, cloning, error handling, reporting, UI, math, iterators, lowest level turtle support.* -> core

--:# **Managing state: clone table, generate closure for session (non-persistent) state, cache loads**

--:: core.clone(source: {:}|any) -> *Deep copy source table or return source if not table.* -> `{:}|any`

--:> closing: *Returns value or sets it and optional table entry to non `nil value`.* -> `(value: any): value: any`

--:: core.state(table: {:}?, key: ":"?) -> *Returns closure over closure variable* -> `closing`

--:# **Table Utilities: merging tables and finding common items in a pair of tables**

--:: core.merge(...: {:}) -> *Merge any number of flat tables into one, allowing repeats.* -> `{:}`

--:: core.match(tableA: any[], tableB: any[]) -> *Find first matching item in pair of item tables.* -> `nil | any`

--:# **Making Strings: both instantiable strings and simple single quoted strings for printing**

--:: core.serialize(input: any) -> *Executable string to instantiate input.* -> `"return ..":" &!`

--:: core.string(...: any) -> *Makes string from any inputs, simplifies single entry tables.* -> `":"`

--:: core.xyzf(:xyzf:) -> *Returns specially formatted string for `xyzf`.* -> `":"`

--:> xyzf: *Position and facing as table* -> `{x: #:, y: #:, z: #:, facing: ":"}`

--:# **Handling errors and reporting operations**

--:: core.pass(ok: ^:, ...: any) -> *Pass input but report string if not ok.* -> ok: `true|false, result: ...|":", any?`

--:: core.where() -> *GPS location if available.* -> `x: #:?, y: #:?, z: #:?`

--:# **Logging and Quit Control Globals**

--:- quit message -> *Set `quit` flag to message; next `core.status` throws `error` to abort operations.*

--:> core.log: *Closure variable* -> `{level: closing, file: closing, handle: closing}`

--:: core.status(level: #:, ...: any) -> *If level less than (elimination) threshold, then report rest as string.* -> `nil`
--:+ *If player, status report is printed and potentially logged. Otherwise sent to player using Muse Status (MS) protocol.*
--:+ *If for in-game turtle with GPS and the dead reckoning and GPS disagree, include that in report.*

--:: core.report(level: #:, ...: any) -> *If level less than `status` threshold, report `rest` as string.* -> `nil`

--:: core.logging(arguments: :[level: #:, filename: ":"]) -> *Set threshold level [and local log file] for status reports* -> `nil`

--:: core.record(message: ":blush: -> *Appends (status) message to log file on player.* -> `nil & !`

--:: core.trace(err: any) -> *Reports traceback for xpcalls.* -> `err: any`

--:# **User interface utilities**

--:: core.completer(completions: {:}) -> *Register command completions for shell* -> `(:)`

--:: core.echo(...: any) -> *For testing; just returns its arguments.* -> ...: `any`

--:- echo arguments ... -> *For testing: just returns its arguments.*

--:: core.optionals(string: ":"?, number: #:?, ...: any) -> *Optional number and/or string.* -> `string: ":"?, number: #:?, ...: any`

--:# **Math utilities**

--:: core.vectorPairs(start: bounds, addend: xyz, number: #:, partial: bounds?) -> *Make plots.* -> `bounds[]`
--:+ *Addend is used to create a vector pair to be added cumulatively beginning with start bounds for result.*
--:+ *The number n is the number of bounds in result where each bound is offset by addend from the prior bounds.*
--:+ *Optionally the partial bounds are included as the first bounds in the result.*

--:> bounds: *Vector pair defining a rectangular solid* -> `:[xyz, xyz]`

--:> xyz: *Minecraft coordinates: +x: east, +y: up, +z: south* -> :[x: #:, y: #:, z: #:]

--:: core.orient(vectors: xyzMap, face: ":"?, rotate: ":"??) -> *Three dimensional rotation* -> `xyzMap`
--:+ *Turn from up north to face, default for no face is to rotate -90 degrees.*

--:> xyzMap: *Table of vectors either an array or dictionary* -> `xyz[] | [core.faces]: xyz`

--:> core.faces: *Key for composed function dictionary* -> "north"|"south"|"east"|"west"|"up"|"down"|"rotate"

--:: core.round(n: #:smiley: -> *Next integer down if below half fraction* -> `#:`

--:# **Example iterator, restartable at index**

--:: core.inext(table: {:}, index: #:smiley: -> *Iterator over table beginning at index.* -> `(:), {:}, #:`

--:# **Lowest level turtle and mock turtle support used by several libraries including lib/motion**

--:: core.findItems(targets: ":"[]) -> *Selects found slot.* -> `detail?`

--:> detail: *Defined by Computercraft* -> `{name: detail.name, count: detail.count, damage: detail.damage}`

--:> detail.name: *Prepended by the mod name* `"minecraft:"`. -> `":"`

--:> detail.count: *Available in inventory* -> `#:`

--:> detail.damage: *Distinguishing value* -> `#:`

--:: core.sleep(#:?) -> *Mocks sleep as null operation out of game.* -> `nil`

--:: core.getComputerID(id: #:?) -> *Out of game returns id; id ignored in game.* -> `id: #:`

--:: core.getComputerLabel(label: ":"?) -> *Out of game returns label; label ignored in game.* -> `label: ":"`

--:: core.setComputerLabel(label: ":blush: -> *Sets (out-of game global) label* -> `label: ":"`