# Miney

*Release 0.2.2*

**Robert Lieback**

**Mar 06, 2021**

# GETTING STARTED


Miney provides an Python interface to Minetest.

First goal is to have fun with a sandbox for Python.

**Do whatever you like:**

- Play and fiddle around while learning python
- Visualize data in unusual ways
- Automate things with bots
- Connect minetest to external services like twitter, ebay or whatsoever
- Do whatever you want!

**Important:** For the best way to get everything running, take a look at the *Quickstart* page.

**Warning:** Miney is currently in beta, so it's usable but the API may change at any point.

CHAPTER

# ONE

# WHY PYTHON?


Some marketing text from the Python website:

> Python is powerful... and fast;
> plays well with others;
> runs everywhere;
> is friendly & easy to learn;
> is Open.
>
> These are some of the reasons people who use Python would rather not use anything else.

And it's popular! And cause of that it has a giant package index filled by over 400.000 users!

CHAPTER

TWO

# WHY MINETEST?

Why not Minecraft? Minetest is free. Not only you don't have to pay for Minetest (consider to donate!), it's also open source! That's a big point, if you try to use this for example in a classroom.

Also modding for minecraft isn't that easy, cause there is no official API or an embedded scripting language like Lua in Minetest. Mods have to be written in Java, but have to be recompiled on every Minecraft update. Cause of that many attempt for APIs appeared and disappeared in recent years.

In contrast Minetest modding in Lua is easy: no compilation, a official API and all game logic is also in Lua.

CHAPTER

THREE

# TABLE OF CONTENTS

## 3.1 Quickstart

Welcome in the block sandbox!

Blockgames like Minecraft or Minetest give you the ideal playground for creative playing and building just like a real sandbox. But other than real sandboxes, you can work on very large worlds together with your friends over the internet. And you can use (very simplified) physics, save the progress and many more.

But what about learning programming while expressing your creativity? Why not automate things? Or build even greater things?

### 3.1.1 Installation

#### Windows

- Download the latest precompiled Miney distribution: https://github.com/miney-py/miney_distribution/releases
- Start the miney-launcher.exe and click on "Quickstart". This will open Minetest directly into a game and IDLE, the IDE shipped with python.

#### Linux

Tested under lubuntu 20.04LTS

$ sudo apt-get install minetest fonts-crosextra-caladea fonts-crosextra-carlito minetest-mod-moreblocks minetest-mod-moreores minetest-mod-pipeworks minetest-server minetestmapper

$ sudo apt-get install luajit lua-socket lua-cjson idle3 python3-pip

$ pip3 install miney

Then install the mineysocket mod in minetest

$ cd ~/.minetest/mods

$ git clone https://github.com/miney-py/mineysocket.git

Don't forget to enable the mods in the configuration tab for your new game!

#### MacOS

Untested

### 3.1.2 First lines of code

The first lines of code with miney should be the import statement and the creation of the miney object "mt". This will connect miney to your already running Minetest.

```
import miney

mt = miney.Minetest()
```

**Important:** Whenever you see a object "mt" in the documentation, it was created with this line!

## 3.2 API

#### Objects

### 3.2.1 Minetest

This is the starting point for this library. With creating a Minetest object you also connect to minetest.

In this object are all functions that targets minetest itself. There is also some properties inside, to get other objects like players or nodes.

**Example**

```
>>> from miney import Minetest
>>>
>>> mt = Minetest()
>>>
>>> # We set the time to midday.
>>> mt.time_of_day = 0.5
>>>
>>> # Write to the servers log
>>> mt.log("Time is set to midday ...")
```

**class** miney.**Minetest**([*server*, *playername*, *password*[, *port*]])

The Minetest server object. All other objects are accessable from here. By creating an object you connect to Minetest.

**Parameters aren't required, if you run miney and minetest on the same computer.**

*If you connect over LAN or Internet to a Minetest server with installed mineysocket, you have to provide a valid playername with a password:*

```
>>> mt = Minetest("192.168.0.2", "MyNick", "secret_password")
```

Account creation is done by starting Minetest and connect to a server with a username and password. https://wiki.minetest.net/Getting_Started#Play_Online

**Parameters**

- **server** (*str*) – IP or DNS name of an minetest server with installed apisocket mod
- **playername** (*str*) – A name to identify yourself to the server.
- **password** (*str*) – Your password
- **port** (*int*) – The apisocket port, defaults to 29999

**property chat**
Object with chat functions.

**Example**

```
>>> mt.chat.send_to_all("My chat message")
```

**Returns** *miney.Chat*: chat object

**log**(*line: str*)
Write a line in the servers logfile.

**Parameters** **line** – The log line

**Returns** None

**property lua**
Functions to run Lua inside Minetest.

**Returns** *miney.Lua*: Lua related functions

**property node**
Manipulate and get information's about nodes.

**Returns** *miney.Node*: Node manipulation functions

**on_event**(*name: str*, *callback: callable*) → None
Sets a callback function for specific events.

**Parameters**

- **name** – The name of the event
- **callback** – A callback function

**Returns** None

**property player**
Get a single players object.

**Examples**

Make a player 5 times faster:

```
>>> mt.player.MyPlayername.speed = 5
```

Use a playername from a variable:

```
>>> player = "MyPlayername"
>>> mt.player[player].speed = 5
```

Get a list of all players

```
>>> list(mt.player)
[<minetest player "MineyPlayer">, <minetest player "SecondPlayer">, ...]
```

**Returns** *miney.Player*: Player related functions

**receive**(*result_id: str = None*, *timeout: float = None*) → Union[str, bool]
Receive data and events from minetest.

With an optional result_id this function waits for a result with that id by call itself until the right result was received. **If lua.run() was used this is not necessary, cause miney already takes care of it.**

With the optional timeout the blocking waiting time for data can be changed.

**Example to receive and print all events**

```
>>> while True:
>>>     print("Event received:", mt.receive())
```

**Parameters**

- **result_id** (*str*) – Wait for this result id
- **timeout** (*float*) – Block further execution until data received or timeout in seconds is over.

**Return type** Union[str, bool]

**Returns** Data from mineysocket

**send**(*data: Dict*)
Send json objects to the miney-socket.

**Parameters** **data** –

**Returns**

**property settings**
Receive all server settings defined in "minetest.conf".

**Returns** A dict with all non-default settings.

**property time_of_day**
Get and set the time of the day between 0 and 1, where 0 stands for midnight, 0.5 for midday.

**Returns** time of day as float.

**property tool**
All available tools in the game, sorted by categories. In the end it just returns the corresponding minetest tool string, so *mt.tool.default.axe_stone* returns the string 'default:axe_stone'. It's a nice shortcut in REPL, cause with auto completion you have only pressed 2-4 keys to get to your type.

**Examples** Directly access a tool:

```
>>> mt.tool.default.pick_mese
'default:pick_mese'
```

Iterate over all available types:

```
>>> for tool_type in mt.tool:
>>>     print(tool_type)
default:shovel_diamond
default:sword_wood
default:shovel_wood
... (there should be around 34 different tools)
>>> print(len(mt.tool))
34
```

Get a list of all types:

```
>>> list(mt.tool)
['default:pine_tree', 'default:dry_grass_5', 'farming:desert_sand_
↪soil', ...
```

Add a diamond pick axe to the first player's inventory:

```
>>> mt.player[0].inventory.add(mt.tool.default.pick_diamond, 1)
```

**Return type** `ToolIterable`

**Returns** `ToolIterable` object with categories. Look at the examples above for usage.

### 3.2.2 Lua

Lua related functions

**class** `miney.`**Lua**(*mt: miney.minetest.Minetest*)

Lua specific functions.

**dumps**(*data*) → str

Convert python data type to a string with lua data type.

**Parameters** **data** – Python data

**Returns** Lua data

**run**(*lua_code: str*, *timeout: float = 10.0*)

Run load code on the minetest server.

**Parameters**

- **lua_code** – Lua code to run
- **timeout** – How long to wait for a result

**Returns** The return value. Multiple values as a list.

**run_file**(*filename*)

Loads and runs Lua code from a file. This is useful for debugging, cause Minetest can throws errors with correct line numbers. It's also easier usable with a Lua capable IDE.

**Parameters** **filename** –

**Returns**

### 3.2.3 Chat

Get controls of the chat.

**class** `miney.`**Chat**(*mt: miney.minetest.Minetest*)

Chat functions.

**send_to_all**(*message: str*) → None

Send a chat message to all connected players.

**Parameters** **message** – The chat message

**Returns** None

### 3.2.4 Player

Change properties of a single player, like there view, speed or gravity.

**class** miney.**Player**(*minetest: miney.minetest.Minetest*, *name*)

A player of the minetest server.

**property fly**

Get and set the privilege to fly to this player. Press K to enable and disable fly mode. As a shortcut you can set fly to a number instead if *True* to also changes the players speed to this number.

**Returns**

**property gravity**

Get or set the players gravity. Default is 1.

**Returns** Float

**property hp**

Get and set the number of hitpoints (2 * number of hearts) between 0 and 20. By setting his hitpoint to zero you instantly kill this player.

**Returns**

**inventory:** *miney.Inventory*

Manipulate player's inventory.

**Example to add 99 dirt to player "IloveDirt"'s inventory**

```
>>> import miney
>>> mt = miney.Minetest()
>>> mt.player.IloveDirt.inventory.add(mt.node.type.default.dirt, 99)
```

**Example to remove 99 dirt from player "IhateDirt"'s inventory**

```
>>> import miney
>>> mt = miney.Minetest()
>>> mt.player.IhateDirt.inventory.remove(mt.node.type.default.dirt,␣
↪99)
```

**property is_online**

Returns the online status of this player.

**Returns** True or False

**property jump**

Get or set the players jump height. Default is 1.

**Returns** Float

**property look**

Get and set look in radians. Horizontal angle is counter-clockwise from the +z direction. Vertical angle ranges between -pi/2 (~-1.563) and pi/2 (~1.563), which are straight up and down respectively.

**Returns** A dict like {'v': 0.34, 'h': 2.50} where h is horizontal and v = vertical

**property look_horizontal**

Get and set yaw in radians. Angle is counter-clockwise from the +z direction.

**Returns** Pitch in radians

**property look_vertical**
Get and set pitch in radians. Angle ranges between -pi/2 (~-1.563) and pi/2 (~1.563), which are straight up and down respectively.

**Returns** Pitch in radians

**property position**
Get the players current position.

**Returns** A dict with x,y,z keys: {"x": 0, "y":1, "z":2}

**property speed**
Get or set the players speed. Default is 1.

**Returns** Float

### 3.2.5 Node

Manipulate and get information's about nodes.

Nodes are defined as dicts in the form

```
>>> {'param1': 15, 'name': 'air', 'param2': 0, 'x': -158.67400138786, 'y': 3.
↪5000000521541, 'z': -16.144999935403}
```

The keys "param1" and "param2" are optional storage variables.

**class** miney.**Node**(*mt: miney.minetest.Minetest*)
Manipulate and get information's about nodes.

**Node manipulation is currently tested for up to 25.000 nodes, more optimization will come later**

**get**(*position: dict*, *position2: dict = None*, *relative: bool = True*, *offset: dict = None*) → Union[dict, list]
Get the node at given position. It returns a dict with the node definition. This contains the "x", "y", "z", "param1", "param2" and "name" keys, where "name" is the node type like "default:dirt".

If also position2 is given, this function returns a list of dicts with node definitions. This list contains a cuboid of definitions with the diagonal between position and position2.

You can get a list of all available node types with `type`.

**Parameters**

- **position** – A dict with x,y,z keys
- **position2** – Another point, to get multiple nodes as a list
- **relative** – Return relative or absolute positions
- **offset** – A dict with "x", "y", "z" keys. All node positions will be added with this values.

**Returns** The node type on this position

**set**(*nodes: Union[dict, list]*, *name: str = None*, *offset: dict = None*) → None
Set a single or multiple nodes at given position to another node type (something like mt.node.type.default.apple). You can get a list of all available nodes with `type`

**A node is defined as a dict with these keys:**

- "x", "y", and "z" keys to define the absolute position

- "name" for a the node type like "default:dirt" (you can also get that from mt.node.type.default.dirt). Dicts without name will be set as "air"
- some other optional minetest parameters

**The nodes parameter can be a single dict with the above parameters or a list of these dicts for bulk spawning.**

**Examples** Set a single node over :

```
>>> mt.node.set(mt.player[0].nodes, mt.node)
```

**Parameters**

- **nodes** – A dict or a list of dicts with node definitions
- **name** – a type name like "default:dirt" as string or from `type`. This overrides

node names defined in the `nodes` dict :param offset: A dict with "x", "y", "z" keys. All node positions will be added with this values.

**property type**

All available node types in the game, sorted by categories. In the end it just returns the corresponding minetest type string, so *mt.node.types.default.dirt* returns the string 'default:dirt'. It's a nice shortcut in REPL, cause with auto completion you have only pressed 2-4 keys to get to your type.

**Examples** Directly access a type:

```
>>> mt.node.type.default.dirt
'default:dirt'
```

Iterate over all available types:

```
>>> for node_type in mt.node.type:
>>>     print(node_type)
default:pine_tree
default:dry_grass_5
farming:desert_sand_soil
... (there should be over 400 different types)
>>> print(len(mt.node.type))
421
```

Get a list of all types:

```
>>> list(mt.node.type)
['default:pine_tree', 'default:dry_grass_5', 'farming:desert_sand_
↪soil', ...
```

Add 99 dirt to player "IloveDirt"'s inventory:

```
>>> mt.player.IloveDirt.inventory.add(mt.node.type.default.dirt, 99)
```

**Return type** `TypeIterable`

**Returns** `TypeIterable` object with categories. Look at the examples above for usage.

### 3.2.6 Inventory

Lua related functions

**class** miney.**Inventory**(*minetest: miney.minetest.Minetest*, *parent: object*)
Inventories are places to store items, like Chests or player inventories.

**add**(*item: str*, *amount: int = 1*) → None
Add an item to an inventory. Possible items can be obtained from *type*.

**Parameters**

- **item** – item type
- **amount** – item amount

**Returns** None

**remove**(*item: str*, *amount: int = 1*) → None
Remove an item from an inventory. Possible items can be obtained from mt.node.type.

**Parameters**

- **item** – item type
- **amount** – item amount

**Returns** None

### 3.2.7 Exceptions

**exception** miney.exceptions.**AuthenticationError**
Authentication error.

**exception** miney.exceptions.**DataError**
Malformed data received.

**exception** miney.exceptions.**LuaError**
Error during Lua code execution.

**exception** miney.exceptions.**LuaResultTimeout**
The answer from Lua takes to long.

**exception** miney.exceptions.**MinetestRunError**
Error: minetest was not found.

**exception** miney.exceptions.**NoValidPosition**

**exception** miney.exceptions.**PlayerInvalid**

**exception** miney.exceptions.**PlayerOffline**

**exception** miney.exceptions.**SessionReconnected**
We had to reconnect and reauthenticate.

### Indices and tables

- genindex
- modindex
- search

## 3.3 Helper functions

miney.**doc**() → None
Open the documention in the webbrower. This is just a shortcut for IDLE or the python interactive console.

**Returns** None

miney.**is_miney_available**(*ip: str = '127.0.0.1'*, *port: int = 29999*, *timeout: int = 1.0*) → bool
Check if there is a running miney game available on an optional given host and/or port. This functions pings mineysocket and waits **timeout** seconds for a pong.

**Parameters**

- **ip** – Optional IP or hostname
- **port** – Optional port
- **timeout** – Optional timeout

**Returns** True or False

miney.**run_miney_game**()
Run minetest with the miney world. Miney will look for the minetest executable in common places for itself, but it's also possible to provide the path as parameter or as environment variable MINETEST_BIN.

**Returns** None

miney.**run_minetest**(*minetest_path: str = None*, *show_menu: bool = True*, *world_path: str = 'Miney'*, *seed: str = '746036489947438842'*) → None
Run minetest. Miney will look for the minetest executable in common places for itself, but it's also possible to provide the path as parameter or as environment variable 'MINETEST_BIN'.

**Parameters**

- **minetest_path** – Path to the minetest executable
- **show_menu** – Start in the world or in the menu
- **world_path** – Optional world path
- **seed** – Optional world seed

**Returns** None


## 3.4 Technical Background

This page provides an inside view of how Miney works.

Miney's basic idea is, to use Minetest with Python.

Minetest's main programming language (besides C++) is Lua and it provides an mighty Lua-API for mod programming. But Lua isn't the ideal programming language to start programming and mod programming isn't fun, if you just want to play around with a sandbox. So we need something like an interface that is accessible by Python.

### 3.4.1 The interface

For this we've written the Mineysocket mod as a regular Lua mod. This mod opens a network port and receives JSON encoded commands. The most important command is the "lua" command, where it just executes the received Lua code and sends any return value back.

Miney is using this lua command to execute Lua code inside Minetest.

---

**Note: And you can use Miney without knowing any Lua or even seeing a single line of Lua code.**

---

### 3.4.2 Mineysocket, Windows and the Miney distribution

Python is the language with batteries included and it ships with a very complete library for nearly everything. In contrast Lua has the batteries explicitly excluded, so there are nearly no libraries and it misses also a network library.

So we need a Lua extension for networking, thats luasocket. And an extension for JSON, thats lua-cjson

Under Linux this should be no big deal, just install these packages (most distributions provide them) and you are ready to go.

### Windows

It isn't that easy for Minetest on Windows. The Minetest binary's where compiled with Visual Studio and the extension has to be linked against minetest also with the same version of Visual Studio. So the best way under windows is, to compile Minetest and the Lua extensions by yourself with the same Visual Studio.

And when we already do this, why not replace Lua with lua-jit for more speed?

And when we've done all this, why not also bundle a actual python interpreter? And why not preinstall Miney in this interpreter? Now it would be nice to have a comfortable launcher.

We've done all this for windows and we call it Miney Distibution.

Miney version: 0.2.2

# PYTHON MODULE INDEX

## m

# INDEX

## T