# Idena Client

**Arash Fatahzade**

**May 04, 2021**

# CONTENTS:

This is Python wrapper for Idene Node.

Please checkout API refrence for avaliable APIs.

CHAPTER

ONE

# API REFERENCE

This page contains auto-generated API reference documentation[1].

## 1.1 idena

### 1.1.1 Subpackages

#### idena.apis

#### Submodules

#### idena.apis.account

#### Module Contents

#### Functions

| | |
|---|---|
| *create_account*(password) | |
| *get_accounts*() | |
| *lock_account*(address) | |
| *unlock_account*(address, password = None, time = None) | |

idena.apis.account.**create_account**(*password*)

**Parameters** **password** (*None*) –

**Return type** str

idena.apis.account.**get_accounts**()

**Return type** List[str]

idena.apis.account.**lock_account**(*address*)

**Parameters** **address** (*str*) –

[1] Created with sphinx-autoapi

**Return type** None

idena.apis.account.**unlock_account**(*address*, *password=None*, *time=None*)

**Parameters**

- **address** (*str*) –
- **password** (*str*) –
- **time** (*int*) –

**Return type** None

## idena.apis.blockchain

### Module Contents

### Functions

| | |
|---|---|
| *check_sync*() | |
| *get_address_pending_transactions*(address) | |
| *get_address_transactions*(address, count = 5, token = None) | |
| *get_block_by_hash*(hash) | |
| *get_block_by_height*(height) | |
| *get_burnt_coins*() | |
| *get_fee_rate*() | |
| *get_last_block*() | |
| *get_mempool*() | |
| *get_raw_tx*(nonce = None, epoch = None, from_ = None, to = None, amount = None, maxFee = None, payload = None, tips = None, useProto = None, type = types.TxType.SendTx) | |
| *get_transaction*(hash) | |
| *get_transaction_receipt*(hash) | |
| *send_raw_tx*(tx) | |

idena.apis.blockchain.**check_sync**()

**Return type** *idena.types.Sync*

idena.apis.blockchain.**get_address_pending_transactions**(*address*)

**Parameters** **address** (*str*) –

**Return type** *idena.types.AddressTransactions*

idena.apis.blockchain.**get_address_transactions**(*address*, *count=5*, *token=None*)

**Parameters**

- **address** (*str*) –
- **count** (*int*) –
- **token** (*str*) –

**Return type** *idena.types.AddressTransactions*

idena.apis.blockchain.**get_block_by_hash**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** *idena.types.Block*

idena.apis.blockchain.**get_block_by_height**(*height*)

**Parameters** **height** (*int*) –

**Return type** *idena.types.Block*

idena.apis.blockchain.**get_burnt_coins**()

**Return type** Optional[*idena.types.BurntCoins*]

idena.apis.blockchain.**get_fee_rate**()

**Return type** decimal.Decimal

idena.apis.blockchain.**get_last_block**()

**Return type** *idena.types.Block*

idena.apis.blockchain.**get_mempool**()

**Return type** Optional[str]

idena.apis.blockchain.**get_raw_tx**(*nonce=None*, *epoch=None*, *from_=None*, *to=None*, *amount=None*, *maxFee=None*, *payload=None*, *tips=None*, *useProto=None*, *type=types.TxType.SendTx*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **from_** (*str*) –
- **to** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **payload** (*str*) –
- **tips** (*decimal.Decimal*) –
- **useProto** (*bool*) –
- **type** (idena.types.TxType) –

**Return type** str

idena.apis.blockchain.**get_transaction**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** *idena.types.Transaction*

idena.apis.blockchain.**get_transaction_receipt**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** Optional[*idena.types.TxReceipt*]

idena.apis.blockchain.**send_raw_tx**(*tx*)

**Parameters** **tx** (*str*) –

**Return type** str

## idena.apis.contract

### Module Contents

### Functions

| | |
|---|---|
| *call_contract*(from_ = None, contract = None, method = None, amount = None, maxFee = None, args = None, broadcastBlock = None) | |
| *deploy_contract*(from_ = None, codeHash = None, amount = None, maxFee = None, args = None) | |
| *estimate_call*(from_ = None, contract = None, method = None, amount = None, maxFee = None, args = None, broadcastBlock = None) | |
| *estimate_deploy*(from_ = None, codeHash = None, amount = None, maxFee = None, args = None) | |
| *estimate_terminate*(from_ = None, contract = None, maxFee = None, args = None) | |
| *get_stake*(contract) | |
| *read_contract_data*(contract = None, key = None, format = None) | |
| *read_events*(contract = None) | |
| *readonly_call_contract*(contract = None, method = None, format = None, args = None) | |
| *subscribe_to_contract_event*(contract = None, event = None) | |
| *terminate_contract*(from_ = None, contract = None, maxFee = None, args = None) | |
| *unsubscribe_from_contract_event*(contract = None, event = None) | |

idena.apis.contract.**call_contract**(*from_=None*, *contract=None*, *method=None*, *amount=None*, *maxFee=None*, *args=None*, *broadcastBlock=None*)

**Parameters**

- **from_** (*str*) –
- **contract** (*str*) –
- **method** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –
- **broadcastBlock** (*int*) –

**Return type** str

idena.apis.contract.**deploy_contract**(*from_=None*, *codeHash=None*, *amount=None*, *maxFee=None*, *args=None*)

**Parameters**

- **from_** (*str*) –
- **codeHash** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –

**Return type** str

idena.apis.contract.**estimate_call**(*from_=None*, *contract=None*, *method=None*, *amount=None*, *maxFee=None*, *args=None*, *broadcastBlock=None*)

**Parameters**

- **from_** (*str*) –
- **contract** (*str*) –
- **method** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –
- **broadcastBlock** (*int*) –

**Return type** *idena.types.ContractTxReceipt*

idena.apis.contract.**estimate_deploy**(*from_=None*, *codeHash=None*, *amount=None*, *maxFee=None*, *args=None*)

**Parameters**

- **from_** (*str*) –
- **codeHash** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –

**Return type** *idena.types.ContractTxReceipt*

idena.apis.contract.**estimate_terminate**(*from_=None*, *contract=None*, *maxFee=None*, *args=None*)

**Parameters**

- **from_** (*str*) –
- **contract** (*str*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –

**Return type** *idena.types.ContractTxReceipt*

idena.apis.contract.**get_stake**(*contract*)

**Parameters** **contract** (*str*) –

**Return type** *idena.types.Stake*

idena.apis.contract.**read_contract_data**(*contract=None*, *key=None*, *format=None*)

**Parameters**

- **contract** (*str*) –
- **key** (*str*) –
- **format** (*str*) –

**Return type** Any

idena.apis.contract.**read_events**(*contract=None*)

**Parameters** **contract** (*str*) –

**Return type** None

idena.apis.contract.**readonly_call_contract**(*contract=None*, *method=None*, *format=None*, *args=None*)

**Parameters**

- **contract** (*str*) –
- **method** (*str*) –
- **format** (*str*) –
- **args** (*List[idena.types.DynamicArg]*) –

**Return type** Any

idena.apis.contract.**subscribe_to_contract_event**(*contract=None*, *event=None*)

**Parameters**

- **contract** (*str*) –
- **event** (*str*) –

**Return type** None

idena.apis.contract.**terminate_contract**(*from_=None*, *contract=None*, *maxFee=None*, *args=None*)

**Parameters**

- **from_** (*str*) –

- **contract** (*str*) –
- **maxFee** (*decimal.Decimal*) –
- **args** (*List[idena.types.DynamicArg]*) –

**Return type** str

idena.apis.contract.**unsubscribe_from_contract_event**(*contract=None*, *event=None*)

**Parameters**

- **contract** (*str*) –
- **event** (*str*) –

**Return type** None

### idena.apis.dna

## Module Contents

## Functions

| | |
|---|---|
| *activate_invite*(nonce = None, epoch = None, key = None) | |
| *activate_invite_to_random_address*(nonce = None, epoch = None, key = None) | |
| *become_offline*(nonce = None, epoch = None) | |
| *become_online*(nonce = None, epoch = None) | |
| *burn*(nonce = None, epoch = None, from_ = None, amount = None, maxFee = None, key = None) | |
| *change_profile*(info = None, nickname = None, maxFee = None) | |
| *export_key*(password) | |
| *get_balance*(address) | |
| *get_ceremony_intervals*() | |
| *get_coinbase_address*() | |
| *get_current_process*() | |
| *get_epoch*() | |
| *get_identities*() | |
| *get_identity*(address) | |
| *get_profile*(address = None) | |

continues on next page

Table 4 – continued from previous page

| |
|---|
| *isValidationReady*() |
| *send_invite*(nonce = None, epoch = None, to = None, amount = None) |
| *send_tx*(nonce = None, epoch = None, from_ = None, to = None, amount = None, maxFee = None, payload = None, tips = None, useProto = None, type = types.TxType.SendTx) |

### Attributes

| |
|---|
| *change_god_address* |
| *kill_identity* |
| *kill_invitee* |
| *send_dna* |

idena.apis.dna.**change_god_address**

idena.apis.dna.**kill_identity**

idena.apis.dna.**kill_invitee**

idena.apis.dna.**send_dna**

idena.apis.dna.**activate_invite**(*nonce=None*, *epoch=None*, *key=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **key** (*str*) –

**Return type** *idena.types.Invite*

idena.apis.dna.**activate_invite_to_random_address**(*nonce=None*, *epoch=None*, *key=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **key** (*str*) –

**Return type** *idena.types.ActivateInviteToRandomAddr*

idena.apis.dna.**become_offline**(*nonce=None*, *epoch=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –

**Return type** str

idena.apis.dna.**become_online**(*nonce=None*, *epoch=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –

**Return type** str

idena.apis.dna.**burn**(*nonce=None*, *epoch=None*, *from_=None*, *amount=None*, *maxFee=None*, *key=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **from_** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **key** (*str*) –

**Return type** str

idena.apis.dna.**change_profile**(*info=None*, *nickname=None*, *maxFee=None*)

**Parameters**

- **info** (*str*) –
- **nickname** (*str*) –
- **maxFee** (*decimal.Decimal*) –

**Return type** *idena.types.ChangeProfile*

idena.apis.dna.**export_key**(*password*)

**Parameters** **password** (*str*) –

**Return type** str

idena.apis.dna.**get_balance**(*address*)

**Parameters** **address** (*str*) –

**Return type** *idena.types.Balance*

idena.apis.dna.**get_ceremony_intervals**()

**Return type** *idena.types.CeremonyIntervals*

idena.apis.dna.**get_coinbase_address**()

**Return type** str

idena.apis.dna.**get_current_process**()

**Return type** *idena.types.State*

idena.apis.dna.**get_epoch**()

**Return type** *idena.types.Epoch*

idena.apis.dna.**get_identities**()

**Return type** List[*idena.types.Identity*]

idena.apis.dna.**get_identity**(*address*)

**Parameters** **address** (*str*) –

**Return type** *idena.types.Identity*

idena.apis.dna.**get_profile**(*address=None*)

**Parameters** **address** (*str*) –

**Return type** *idena.types.Profile*

idena.apis.dna.**isValidationReady**()

**Return type** bool

idena.apis.dna.**send_invite**(*nonce=None*, *epoch=None*, *to=None*, *amount=None*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **to** (*str*) –
- **amount** (*decimal.Decimal*) –

**Return type** *idena.types.Invite*

idena.apis.dna.**send_tx**(*nonce=None*, *epoch=None*, *from_=None*, *to=None*, *amount=None*, *maxFee=None*, *payload=None*, *tips=None*, *useProto=None*, *type=types.TxType.SendTx*)

**Parameters**

- **nonce** (*int*) –
- **epoch** (*int*) –
- **from_** (*str*) –
- **to** (*str*) –
- **amount** (*decimal.Decimal*) –
- **maxFee** (*decimal.Decimal*) –
- **payload** (*str*) –
- **tips** (*decimal.Decimal*) –
- **useProto** (*bool*) –
- **type** (idena.types.TxType) –

**Return type** str

### idena.apis.flip

## Module Contents

## Functions

| | |
|---|---|
| *delete_flip*(hash) | |
| *flip_words*(hash) | |
| *get_flip*(hash) | |
| *get_long_flip_hashes*() | |
| *get_raw_flip*(hash) | |
| *get_short_flip_hashes*() | |
| *submit_flip*(publicHex = None, privateHex = None, pairId = None, hex = None) | |
| *submit_long_answers*(answers) | |
| *submit_short_answers*(answers) | |

idena.apis.flip.**delete_flip**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** str

idena.apis.flip.**flip_words**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** Tuple[int, int]

idena.apis.flip.**get_flip**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** *idena.types.Flip*

idena.apis.flip.**get_long_flip_hashes**()

**Return type** List[*idena.types.FlipHashes*]

idena.apis.flip.**get_raw_flip**(*hash*)

**Parameters** **hash** (*str*) –

**Return type** *idena.types.RawFlip*

idena.apis.flip.**get_short_flip_hashes**()

**Return type** List[*idena.types.FlipHashes*]

idena.apis.flip.**submit_flip**(*publicHex=None*, *privateHex=None*, *pairId=None*, *hex=None*)

**Parameters**

- **publicHex** (*str*) –
- **privateHex** (*str*) –
- **pairId** (*str*) –
- **hex** (*str*) –

**Return type** idena.types.FlipSubmit

idena.apis.flip.**submit_long_answers**(*answers*)

**Parameters** **answers** (idena.types.FlipAnswers) –

**Return type** str

idena.apis.flip.**submit_short_answers**(*answers*)

**Parameters** **answers** (idena.types.FlipAnswers) –

**Return type** str

### idena.apis.net

#### Module Contents

#### Functions

| |
|---|
| *add_peers*(url) |
| *get_ipfs_addr*() |
| *get_peers*() |

idena.apis.net.**add_peers**(*url*)

**Return type** str

idena.apis.net.**get_ipfs_addr**()

**Return type** str

idena.apis.net.**get_peers**()

**Return type** List[*idena.types.Peer*]

## 1.1.2 Submodules

### idena.client

#### Module Contents

#### Functions

| | |
|---|---|
| *init*(rpc_node, api_key) | Initialization |

idena.client.**init**(*rpc_node*, *api_key*)
Initialization

**Args:** rpc_node (str): URL of rpc node api_key (str): API-Key

**Parameters**

- **rpc_node** (*str*) –
- **api_key** (*str*) –

**Return type** None

### idena.exceptions

#### Module Contents

**exception** idena.exceptions.**IdenaException**(*code*, *message*)
Bases: Exception

Common base class for all non-exit exceptions.

**__repr__**(*self*)
Return repr(self).

### idena.types

#### Module Contents

#### Classes

| | |
|---|---|
| *ActivateInviteToRandomAddr* | Typed version of namedtuple. |
| *AddressTransactions* | Typed version of namedtuple. |
| *Answer* | Typed version of namedtuple. |
| *Balance* | Typed version of namedtuple. |
| *BaseTxArgs* | Typed version of namedtuple. |
| *Block* | Typed version of namedtuple. |
| *BurntCoins* | Typed version of namedtuple. |
| *CeremonyIntervals* | Typed version of namedtuple. |
| *ChangeProfile* | Typed version of namedtuple. |
| *ContractTxReceipt* | |
| *DynamicArg* | Typed version of namedtuple. |
| *Epoch* | Typed version of namedtuple. |
| *Flip* | Typed version of namedtuple. |
| *FlipAnswer* | Typed version of namedtuple. |
| *FlipAnswers* | Typed version of namedtuple. |
| *FlipHashes* | Typed version of namedtuple. |
| *FlipWords* | Typed version of namedtuple. |

continues on next page

Table 9 – continued from previous page

| | |
|---|---|
| *Grade* | |
| *Identity* | Typed version of namedtuple. |
| *Invite* | Typed version of namedtuple. |
| *Peer* | Typed version of namedtuple. |
| *Profile* | Typed version of namedtuple. |
| *RawFlip* | Typed version of namedtuple. |
| *SendTxArgs* | docstring |
| *Stake* | Typed version of namedtuple. |
| *State* | Typed version of namedtuple. |
| *Sync* | Typed version of namedtuple. |
| *Transaction* | Typed version of namedtuple. |
| *TxAddr* | Typed version of namedtuple. |
| *TxEvent* | Typed version of namedtuple. |
| *TxReceipt* | Typed version of namedtuple. |
| *TxType* | |

### Attributes

| | |
|---|---|
| *FlipSubmit* | |

idena.types.**FlipSubmit**

**class** idena.types.**ActivateInviteToRandomAddr**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**address :str**

**hash :str**

**key :str**

**class** idena.types.**AddressTransactions**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**token :Optional[str]**

**transactions :Optional[List[Transaction]]**

**class** idena.types.**Answer**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**left = 1**

**none = 0**

**right = 2**

**class** idena.types.**Balance**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**balance :decimal.Decimal**

**nonce :int**

**stake :decimal.Decimal**

**class** idena.types.**BaseTxArgs**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**epoch :int**

**nonce :int**

**class** idena.types.**Block**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**coinbase :str**

**flags :List[str]**

**hash :str**

**height :int**

**identityRoot :str**

**ipfsCid :str**

**isEmpty :bool**

**offlineAddress :str**

**parentHash :str**

**root :str**

**timestamp :int**

**transactions :List[str]**

**class** idena.types.**BurntCoins**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**address :str**

**amount :decimal.Decimal**

**key :str**

**class** idena.types.**CeremonyIntervals**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**FlipLotteryDuration :float**

**LongSessionDuration :float**

**ShortSessionDuration :float**

**class** idena.types.**ChangeProfile**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**hash :str**

**txHash :str**

**class** idena.types.**ContractTxReceipt**

**contract :str**

**error :str**

**gasCost :decimal.Decimal**

**gasUsed :int**

**success :bool**

**txFee :decimal.Decimal**

**txHash :str**

**class** idena.types.**DynamicArg**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**format :str**

**index :int**

**value :str**

**class** idena.types.**Epoch**
Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**currentPeriod :str**

**currentValidationStart :datetime.datetime**

**epoch :int**

**nextValidation :datetime.datetime**

**class** idena.types.**Flip**
Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**hex :str**

**privateHex :str**

**class** idena.types.**FlipAnswer**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**answer :Answer**

**grade :Grade**

**hash :str**

**wrongWords :List[bool]**

**class** idena.types.**FlipAnswers**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**answers :List[FlipAnswer]**

**class** idena.types.**FlipHashes**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**available :bool**

**extra :bool**

**hash :str**

**ready :bool**

**class** idena.types.**FlipWords**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**id :int**

**used :bool**

**words :List[int]**

**class** idena.types.**Grade**

**gradeA = 5**

**gradeB = 4**

**gradeC = 3**

**gradeD = 2**

**gradeNone = 0**

**gradeReported = 1**

**class** idena.types.**Identity**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**address :str**

**age :int**

**availableFlips :int**

**code :bytes**

**flipKeyWordPairs :Optional[List[FlipWords]]**

**flips :Optional[List[str]]**

**generation :int**

**invitees :Optional[List[TxAddr]]**

**invites :int**

**lastValidationFlags :Optional[List[str]]**

**madeFlips :int**

**online :bool**

**penalty :decimal.Decimal**

**profileHash :str**

**pubkey :str**

**requiredFlips :int**

**stake :decimal.Decimal**

**state :str**

**totalQualifiedFlips :int**

**totalShortFlipPoints :float**

**class** idena.types.**Invite**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**hash :str**

**key :str**

**receiver :str**

**class** idena.types.**Peer**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**addr :str**

**id :str**

**class** idena.types.**Profile**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**info :bytes**

**nickname :str**

**class** idena.types.**RawFlip**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**privateHex :str**

**publicHex :str**

**class** idena.types.**SendTxArgs**

Bases: *BaseTxArgs*

docstring

**amount :decimal.Decimal**

**from_ :str**

**maxFee :decimal.Decimal**

**payload :str**

**tips :decimal.Decimal**

**to :str**

**type :TxType**

**useProto :bool**

**class** idena.types.**Stake**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526

compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**Hash :str**

**Stake :decimal.Decimal**

**class** idena.types.**State**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**name :str**

**class** idena.types.**Sync**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**currentBlock :int**

**genesisBlock :int**

**highestBlock :int**

**syncing :bool**

**wrongTime :bool**

**class** idena.types.**Transaction**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**amount :str**

**blockHash :str**

**epoch :int**

**from_ :str**

**hash :str**

**maxFee :str**

**nonce :int**

**payload :str**

**timestamp :int**

**tips :str**

**to :Optional[str]**

**type :str**

**usedFee :str**

**class** idena.types.**TxAddr**
Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**Address :str**

**TxHash :str**

**class** idena.types.**TxEvent**
Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**data :List[bytes]**

**eventName :str**

**class** idena.types.**TxReceipt**

Bases: NamedTuple

Typed version of namedtuple.

Usage in Python versions >= 3.6:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

The resulting class has extra __annotations__ and _field_types attributes, giving an ordered dict mapping field names to types. __annotations__ should be preferred, while _field_types is kept to maintain pre PEP 526 compatibility. (The field names are in the _fields attribute, which is part of the namedtuple API.) Alternative equivalent keyword syntax is also accepted:

```
Employee = NamedTuple('Employee', name=str, id=int)
```

In Python versions <= 3.5 use:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

**contractAddress :str**

**error :str**

**events :List[TxEvent]**

**from_ :str**

**gasCost :int**

**gasUsed :int**

**success :bool**

**txHash :str**

**class** idena.types.**TxType**

**ActivationTx = 1**

**BurnTx = 12**

**ChangeGodAddressTx = 11**

**ChangeProfileTx = 13**

**EvidenceTx = 8**

**InviteTx = 2**

**KillInviteeTx = 10**

**KillTx = 3**

**OnlineStatusTx = 9**

**SendTx = 0**

**SubmitAnswersHashTx = 5**

**SubmitFlipTx = 4**

**SubmitLongAnswersTx = 7**

**SubmitShortAnswersTx = 6**

- Highly inspired by Web3.py and cookiecutter-pypackage.

CHAPTER

TWO

# INSTALLATION

Idena.py can be installed using `pip` as follows:

```
$ pip install idena
```

For the development, clone the repository then:

```
$ pip install -e .
```

CHAPTER

THREE

# USING IDENA

This library depends on a connection to an Idena node and there are 2 ways to configure them.

## 3.1 Calling *client.init*

```
>>> from idena import client
>>> client.init('http://localhost:9009/', 'api-key')
```

## 3.2 Setting environment variables

Set *IDENA_RPC_NODE* and *IDENA_API_KEY* envars:

```
$ export IDENA_RPC_NODE=http://localhost:9009/
$ export IDENA_API_KEY=api-key
```

CHAPTER

FOUR

# GETTING BLOCKCHAIN INFO

```
>>> client.blockchain.get_last_block()

Block(coinbase='0xbe854231db69ab042073b7ff8309ae3ee265a40f',
   hash='0xa88e6ab305d7ee311ad2de35338cdbf7e664d860709e5a53f0307baeeaa6f968',
   parentHash='0xe324a208892241e0294e5a6334965660375dda7a3ad8d8a42a5f3f2ef2857a22',
   height=2159398,
   timestamp=1606904853,
   root='0xd874709bdd4c6fcd95e2e531cc07a4ce42ab23334dfd12ceb45350535b36664c',
   identityRoot='0x9f3661f19e13d4860f2f2f1610abbbaf86abc8adf1a2781b371189e683745a97',
   ipfsCid=None,
   transactions=None,
   flags=['OfflinePropose'],
   isEmpty=False,
   offlineAddress='0x0df427ad7e1906ab4fcc5fd31118932256f5dc7a')
```

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## i

# INDEX

## Symbols



## A



## B



## C



## D



## E

## F



## G



## H



## I

## K



## L



## M



## N



## O



## P



## R

## S



## T



## U



## V



## W