Mimesis Documentation

Release 3.3.0

Likid Geimfari

Aug 15, 2019

CONTENTS

Mimesis is fast and extremely easy to use Python package, which helps generate big volumes of fake data for a variety of purposes in a variety of languages.

The fake data can be particularly useful during software development and testing. For example, it could be used to populate a testing database, create beautiful JSON and XML files, anonymize data taken from a production service and etc.

CHAPTER

ONE

FEATURES

Mimesis provides a lot of useful features, here are some of them:

- Custom providers
- Generic data provider
- More than 33 locales
- More than 21 data providers
- Data generation by the schema
- Romanization of Cyrillic data
- Built-In country-specific data providers

CHAPTER

TWO

CONTENTS

2.1 Foreword

2.1.1 Advantages

This library offers a number of advantages over other similar libraries, such as Faker:

- Performance. Significantly faster than other similar libraries.
- Completeness. Strives to provide many detailed providers that offer a variety of data generators.
- Simplicity. Does not require any modules other than the Python standard library.

Below you can look how we compared performance:

```
import cProfile

from mimesis import Person
from faker import Faker

person = Person()
faker = Faker()
```

Generate 10k full names

```
# Generating using Mimesis:
cProfile.run('[person.full_name() for _ in range(10000)]')

# Generating using Faker:
cProfile.run('[faker.name() for _ in range(10000)]')
```

Result:

Library	Method name	Iterations	Runtime (second)
Mimesis	*full_name()*	10 000	0.254
Faker	**Faker.name()**	10 000	15.144

Generate 10k last names

```
# Generating using Mimesis:
cProfile.run('[person.last_name() for _ in range(10000)]')

# Generating using Faker:
cProfile.run('[faker.last_name() for _ in range(10000)]')
```

Result:

Library	Method name	Iterations	Runtime (second)
Mimesis	*last_name()*	10 000	0.040
Faker	**Faker.last_name()**	10 000	8.218

2.1.2 What does name mean?

Mimesis (/mamiss/; Ancient Greek: (*mīmēsis*), from (*mīmeisthai*), "to imitate", from (mimos), "imitator, actor") is a critical and philosophical term that carries a wide range of meanings, which include imitation, representation, mimicry, imitatio, receptivity, nonsensuous similarity, the act of resembling, the act of expression, and the presentation of the self.

2.1.3 Why octopus?

Basically, because octopuses are cool guys, but also because of the fantastic mimicry abilities of some families of octopuses. Have you ever hear about Thaumoctopus mimicus? Just read about that guy, because he is a really badass one.

2.1.4 What is the fake data?

Fake data is a kind of data which is used in software development.

That data looks like real data, but it is not.

2.1.5 What Mimesis is, What Mimesis is Not

The problem that **Mimesis** solves and solves it perfectly is generating data. When you need to populate database, create complex structured JSON/XML files, anonymize data taken from productive services then **Mimesis** is this is exactly what you need.

Mimesis is **not object factory** and it was not developed for using with specific database or ORM (such as Django ORM, SQLAlchemy etc.). It does not mean that you can't use it with ORM on the contrary, this will be done very simply, this only means that possibly you'll need third-party libraries to do it, like *mimesis-factory* or another one.

2.2 Getting Started

2.2.1 Installation

Attention: Mimesis works only on Python 3.6 and higher. Developers have not plans related to adding support for old versions of Python.

Within the pre-activated environment, use the following command to install Mimesis:

```
(env)  pip install mimesis
```

Installation using *Pipenv* is pretty same:

```
(env)  pipenv install --dev mimesis
```

If you want to work with the latest Mimesis code before it's released, install or update the code from the master branch:

```
(env)  git clone git@github.com:lk-geimfari/mimesis.git
(env)  cd mimesis/
(env)  make install
```

2.2.2 Basic Usage

A minimal basic usage example looks something like this:

```
>>> from mimesis import Person
>>> from mimesis.enums import Gender
>>> person = Person('en')

>>> person.full_name(gender=Gender.FEMALE)
'Antonetta Garrison'

>>> person.full_name(gender=Gender.MALE)
'Jordon Hall'
```

So what did the code above?

1. First we imported the *Person* provider. An instance of this class will be our provider of personal data.
2. We import object *Gender* which we are used as a parameter for the *full_name()*.
3. Next we generate random female full name.
4. The same as above, but for male.

2.2.3 Providers

Mimesis support over twenty different data providers available, which can produce data related to food, people, computer hardware, transportation, addresses, and more. See *API Reference* for more info.

2.2.4 Generic Provider

When you only need to generate data for a single locale, use the *Generic* provider, and you can access all Mimesis providers from one object.

```
>>> from mimesis import Generic
>>> g = Generic('es')

>>> g.datetime.month()
'Agosto'

>>> g.code.imei()
```

(continues on next page)

(continued from previous page)

```
'353918052107063'

>>> g.food.fruit()
'Limón'

>>> g.science.rna()
'GCTTTAGACC'
```

2.2.5 Locales

You can specify a locale when creating providers and they will return data that is appropriate for the language or country associated with that locale:

```
>>> from mimesis import Address

>>> de = Address('de')
>>> ru = Address('ru')

>>> de.region()
'Brandenburg'

>>> ru.federal_subject()
' '

>>> de.address()
'Mainzer Landstraße 912'

>>> ru.address()
'.  125'
```

Override locale

Sometimes you need only some data from other locale and creating an instance for such cases is not really good, so it's better just temporarily override current locale for provider's instance:

```
>>> from mimesis import Person
>>> from mimesis import locales

>>> person = Person(locales.EN)
>>> person.full_name()
'Ozie Melton'

>>> with person.override_locale(locales.RU):
...     person.full_name()

' '

>>> person.full_name()
'Waldo Foster'
```

You can also use it with *Generic()*:

```
>>> from mimesis import Generic
>>> from mimesis import locales

>>> generic = Generic(locales.EN)
>>> generic.text.word()
'anyone'

>>> with generic.text.override_locale(locales.FR):
...     generic.text.word()

'mieux'

>>> generic.text.word()
'responsibilities'
```

Supported locales

Mimesis currently includes support for 33 different locales:

Code	Name	Native Name
cs	Czech	Česky
da	Danish	Dansk
de	German	Deutsch
de-at	Austrian german	Deutsch
de-ch	Swiss german	Deutsch
el	Greek	
en	English	English
en-au	Australian English	English
en-ca	Canadian English	English
en-gb	British English	English
es	Spanish	Español
es-mx	Mexican Spanish	Español
et	Estonian	Eesti
fa	Farsi	
fi	Finnish	Suomi
fr	French	Français
hu	Hungarian	Magyar
is	Icelandic	Íslenska
it	Italian	Italiano
ja	Japanese	
kk	Kazakh	
ko	Korean	
nl	Dutch	Nederlands
nl-be	Belgium Dutch	Nederlands
no	Norwegian	Norsk
pl	Polish	Polski
pt	Portuguese	Português
pt-br	Brazilian Portuguese	Português Brasileiro
ru	Russian	
sv	Swedish	Svenska
tr	Turkish	Türkçe

Continued on next page

Table 1 – continued from previous page

Code	Name	Native Name
uk	Ukrainian	
zh	Chinese	

2.2.6 Seeded Data

For using seeded data just pass an argument *seed* (which can be *int*, *str*, *bytes*, *bytearray*) to data provider:

```
>>> from mimesis import Person

>>> person = Person('tr', seed=0xFF)
>>> person.full_name()
'Gizem Tekand'
```

2.2.7 Built-in Providers

Most countries, where only one language is official, have data typical only for these particular countries. For example, «CPF» for Brazil (**pt-br**), «SSN» for USA (**en**). This kind of data can cause discomfort and meddle with the order (or at least annoy) by being present in all the objects regardless of the chosen language standard. You can see that for yourselves by looking at the example (the code won't run):

```
>>> from mimesis import Person
>>> person = Person('en')

>>> person.ssn()
>>> person.cpf()
```

We bet everyone would agree that this does not look too good. Perfectionists, as we are, have taken care of this in a way that some specific regional provider would not bother other providers for other regions. For this reason, class providers with locally-specific data are separated into a special sub-package (**mimesis.builtins**) for keeping a common class structure for all languages and their objects.

Here's how it works:

```
>>> from mimesis import Generic
>>> from mimesis.builtins import BrazilSpecProvider

>>> generic = Generic('pt-br')
>>> generic.add_provider(BrazilSpecProvider)
>>> generic.brazil_provider.cpf()
'696.441.186-00'
```

If you want to change default name of built-in provider, just change value of attribute *name*, class *Meta* of the builtin provider:

```
>>> BrazilSpecProvider.Meta.name = 'brasil'
>>> generic.add_provider(BrazilSpecProvider)
>>> generic.brasil.cpf()
'019.775.929-70'
```

Or just inherit the class and override the value of attribute *name* of class *Meta* of the provider (in our case this is *BrazilSpecProvider*) :

```
>>> class Brasil(BrazilSpecProvider):
...
...     class Meta:
...         name = "brasil"
...
>>> generic.add_provider(Brasil)
>>> generic.brasil.cnpj()
'55.806.487/7994-45'
```

Generally, you don't need to add built-it classes to the object *Generic*. It was done in the example with the single purpose of demonstrating in which cases you should add a built-in class provider to the object *Generic*. You can use it directly, as shown below:

```
>>> from mimesis.builtins import RussiaSpecProvider
>>> from mimesis.enums import Gender
>>> ru = RussiaSpecProvider()

>>> ru.patronymic(gender=Gender.FEMALE)
''

>>> ru.patronymic(gender=Gender.MALE)
''
```

See *API Reference* for more info about built-in providers.

2.2.8 Custom Providers

The library supports a vast amount of data and in most cases this would be enough. For those who want to create their own providers with more specific data. This can be done like this:

```
>>> from mimesis.providers.base import BaseProvider

>>> class SomeProvider(BaseProvider):
...     class Meta:
...         name = "some_provider"
...
...     @staticmethod
...     def hello():
...         return 'Hello!'

>>> class Another(BaseProvider):
...     @staticmethod
...     def bye():
...         return "Bye!"

>>> generic.add_provider(SomeProvider)
>>> generic.add_provider(Another)

>>> generic.some_provider.hello()
'Hello!'

>>> generic.another.bye()
'Bye!'
```

You can also add multiple providers:

```
>>> generic.add_providers(SomeProvider, Another)
>>> generic.some_provider.hello()
'Hello!'
>>> generic.another.bye()
'Bye!'
```

If you'll try to add provider which does not inherit `BaseProvider` then you got `TypeError` exception:

```
>>> class InvalidProvider(object):
...     @staticmethod
...     def hello():
...         return 'Hello!'

>>> generic.add_provider(InvalidProvider)
Traceback (most recent call last):
  ...
TypeError: The provider must inherit BaseProvider.
```

All providers must be subclasses of `BaseProvider` because of ensuring a single instance of object `Random`.

Everything is pretty easy and self-explanatory here, therefore, we will only clarify one moment—attribute *name*, class *Meta* is the name of a class through which access to methods of user-class providers is carried out. By default class name is the name of the class in lowercase letters.

2.2.9 Schema and Fields

For generating data by schema, just create an instance of *Field* object, which takes any string which represents the name of data provider in format *provider.method_name* (explicitly defines that the method *method_name* belongs to data-provider *provider*) or *method* (will be chosen the first provider which has a method *method_name*) and the ****kwargs** of the method *method_name*, after that you should describe the schema in lambda function and pass it to the object *Schema* and call method *create()*.

Optionally, you can apply a *key function* to result returned by the method, to do it, just pass the parameter *key* with a callable object which returns final result.

Example of usage:

```
>>> from mimesis.schema import Field, Schema
>>> from mimesis.enums import Gender
>>> _ = Field('en')
>>> description = (
...     lambda: {
...         'id': _('uuid'),
...         'name': _('text.word'),
...         'version': _('version', pre_release=True),
...         'timestamp': _('timestamp', posix=False),
...         'owner': {
...             'email': _('person.email', key=str.lower),
...             'token': _('token_hex'),
...             'creator': _('full_name', gender=Gender.FEMALE),
...         },
...     }
... )
>>> schema = Schema(schema=description)
>>> schema.create(iterations=1)
```

Output:

```
[
  {
    "owner": {
      "email": "aisling2032@yahoo.com",
      "token": "cc8450298958f8b95891d90200f189ef591cf2c27e66e5c8f362f839fcc01370",
      "creator": "Veronika Dyer"
    },
    "name": "pleasure",
    "version": "4.3.1-rc.5",
    "id": "33abf08a-77fd-1d78-86ae-04d88443d0e0",
    "timestamp": "2018-07-29T15:25:02Z"
  }
]
```


By default, *Field* works only with providers which supported by *Generic*, to change this behavior should be passed parameter *providers* with a sequence of data providers:

```
>>> from mimesis.schema import Field
>>> from mimesis import builtins as b

>>> extra = (
...     b.RussiaSpecProvider,
...     b.NetherlandsSpecProvider,
... )
>>> _ = Field('en', providers=extra)

>>> _('snils')
'239-315-742-84'

>>> _('bsn')
'657340522'
```

2.3 Tips and Tricks

2.3.1 Creating objects

If your app requires data in one particular language, it's preferable to use class *Generic()*, giving access to all class providers through a single object, rather than through multiple separate class providers. Using *Generic()* will allow you to get rid of several extra lines of code.

Incorrect:

```
>>> from mimesis import Person, Datetime, Text, Code

>>> person = Person('ru')
>>> datetime = Datetime('ru')
>>> text = Text('ru')
>>> code = Code('ru')
```

Correct:

```
>>> from mimesis import Generic
>>> generic = Generic('ru')
```

(continues on next page)

(continued from previous page)

```
>>> generic.person.username()
'sherley3354'

>>> generic.datetime.date()
'14-05-2007'
```

Still correct:

```
>>> from mimesis import Person

>>> p_en = Person('en')
>>> p_sv = Person('sv')
>>> # ...
```

Also correct:

```
>>> from mimesis import Person

>>> person = Person('en')
>>> with person.override_locale('sv')
>>>     pass
>>> # ...
```

It means that importing class providers separately makes sense only if you limit yourself to the data available through the class you imported, otherwise it's better to use *Generic()*.

2.3.2 Inserting data into database

If you need to generate data and import it into a database we strongly recommend generating data in chunks rather than *600k* at once. Keep in mind the possible limitations of databases, ORM, etc. The smaller the generated data chunks are, the faster the process will go.

Good:

```
>>> User().fill_fake(count=2000, locale='de')
```

Very bad:

```
>>> User().fill_fake(count=600000, locale='de')
```

2.3.3 Importing images

Class *Internet()* boasts of several methods which generate image links (more details here). Links to images locate on remote servers would be enough, however, if you still want to have a number of random images locally, you can download images generated by the respective class *Internet()* methods with the help of function `download_image()` from model utils:

```
>>> from mimesis import Internet
>>> from mimesis.shortcuts import download_image

>>> net = Internet()

>>> url = net.stock_image(width=1920, height=1080, keywords=['love', 'passion'])
>>> download_image(url=url, save_path='/some/path/')
```

2.3.4 Romanization of Cyrillic data

If your locale belongs to the family of Cyrillic languages, but you need latinized locale-specific data, then you can use decorator *romanized()* which help you romanize your data.

Example of usage for romanization of Russian full name:

```
>>> from mimesis.decorators import romanized

>>> @romanized('ru')
... def russian_name():
...     return ' '

>>> russian_name()
'Veronika Denisova'
```

At this moment it works only for Russian (**ru**), Ukrainian (**uk**) and Kazakh (**kk**):

2.3.5 Dummy API Endpoints

You can create dummy API endpoints when you have not data, but need them and know the structure of the endpoint's response.

Let's define the structure of the dummy response.

dummy_endpoints.py:

```
from mimesis.schema import Field, Schema
from mimesis.enums import Gender

_ = Field('en')
dummy_users = Schema(
    lambda: {
        'id': _('uuid'),
        'name': _('name', gender=Gender.MALE),
        'surname': _('surname', gender=Gender.MALE),
        'email': _('email'),
        'age': _('age'),
        'username': _('username', template='UU_d'),
        'occupation': _('occupation'),
        "address": {
            "street": _('street_name'),
            "city": _('city'),
            "zipcode": _('zip_code'),
        },
    }
)
```

Now, you can return unique response with JSON for each request.

2.3.6 Django/DRF Dummy API Endpoint

Basically you need just create simple view, which returns *JsonResponse*:

```
from dummy_endpoints import dummy_users

def users(request):
    dummy_data = dummy_users.create(iterations=100)
    return JsonResponse(dummy_data)
```

For DRF the same, but in terms of DRF:

```
from dummy_endpoints import dummy_users

class Users(APIView):
    def get(self, request):
        data = dummy_users.create(iterations=100)
        return Response(data)
```

Response:

```
[
  {
    "id": "a46313ab-e218-41cb-deee-b9afd755a4dd",
    "name": "Wally",
    "surname": "Stein",
    "email": "artiller1855@yahoo.com",
    "age": 51,
    "username": "SystemicZeuzera_1985",
    "occupation": "Travel Courier",
    "address": {
      "street": "Lessing",
      "city": "Urbandale",
      "zipcode": "03983"
    }
  }
# ...,
# ...,
]
```

2.3.7 Flask Dummy API Endpoint

The same way as above:

```
from dummy_endpoints import dummy_users

@app.route('/users')
def users():
    dummy_data = dummy_users.create(iterations=100)
    return jsonify(dummy_data)
```

Response:

```
[
  {
    "id": "f2b326e3-4ce7-1ae9-9e6d-34a28fb70106",
    "name": "Johnny",
    "surname": "Waller",
    "email": "vault1907@live.com",
    "age": 47,
```

(continues on next page)

(continued from previous page)

```
    "username": "CaterpillarsSummational_1995",
    "occupation": "Scrap Dealer",
    "address": {
      "street": "Tonquin",
      "city": "Little Elm",
      "zipcode": "30328"
    }
  },
# ...,
# ...,
]
```

2.3.8 Integration with third-party libraries

- mimesis-factory - Integration with `factory_boy`.
- pytest-mimesis - is a pytest plugin that provides pytest fixtures for Mimesis providers.

CHAPTER

THREE

API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

3.1 API Reference

This part of the documentation covers all the public interfaces of *Mimesis*.

3.1.1 Builtin Data Providers

BrazilSpecProvider

class mimesis.builtins.**BrazilSpecProvider**(*seed=None*)

Class that provides special data for Brazil (pt-br).

class Meta

The name of the provider.

__init__(*seed=None*)

Initialize attributes.

cnpj(*with_mask=True*)

Get a random CNPJ.

Parameters **with_mask** (bool) – Use cnpj mask (###.###.###-##)

Return type str

Returns Random cnpj.

Example 77.732.230/0001-70

cpf(*with_mask=True*)

Get a random CPF.

Parameters **with_mask** (bool) – Use CPF mask (###.###.###-##).

Return type str

Returns Random CPF.

Example 001.137.297-40

DenmarkSpecProvider

class mimesis.builtins.**DenmarkSpecProvider**(*seed=None*)
Class that provides special data for Denmark (da).

class Meta
The name of the provider.

__init__(*seed=None*)
Initialize attributes.

cpr()
Generate a random CPR number (Central Person Registry).

Return type str

Returns CPR number.

Example 0105865167

GermanySpecProvider

class mimesis.builtins.**GermanySpecProvider**(*seed=None*)
Specific-provider of misc data for Germany.

class Meta
The name of the provider.

__init__(*seed=None*)
Initialize attributes.

noun(*plural=False*)
Return a random noun in German.

Parameters **plural** (bool) – Return noun in plural.

Return type str

Returns Noun.

NetherlandsSpecProvider

class mimesis.builtins.**NetherlandsSpecProvider**(*seed=None*)
Class that provides special data for Netherlands (nl).

class Meta
The name of the provider.

__init__(*seed=None*)
Initialize attributes.

bsn()
Generate a random, but valid Burgerservicenummer.

Return type str

Returns Random BSN.

Example 255159705

burgerservicenummer()
Generate a random, but valid `Burgerservicenummer`.

An alias for self.bsn()

Return type `str`

RussiaSpecProvider

class mimesis.builtins.**RussiaSpecProvider**(*seed=None*)
Class that provides special data for Russia (ru).

class Meta
The name of the provider.

__init__(*seed=None*)
Initialize attributes.

bic()
Generate random `BIC` (Bank ID Code).

Return type `str`

Returns BIC.

Example

44025575.

generate_sentence()
Generate sentence from the parts.

Return type `str`

Returns Sentence.

inn()
Generate random, but valid `INN`.

Return type `str`

Returns INN.

kpp()
Generate random `KPP`.

Return type `str`

Returns 'KPP'.

Example

560058652.

ogrn()
Generate random valid `OGRN`.

Return type `str`

Returns OGRN.

Example

-2147483648.

passport_number()
Generate random passport number.

Return type int

Returns Number.

Example 560430

passport_series(*year=None*)
Generate random series of passport.

Parameters **year** (*int or None*) – Year of manufacture.

Return type str

Returns Series.

Example 02 15.

patronymic(*gender=None*)
Generate random patronymic name.

Parameters **gender** (Optional[*Gender*]) – Gender of person.

Return type str

Returns Patronymic name.

Example .

series_and_number()
Generate a random passport number and series.

Return type str

Returns Series and number.

Example 57 16 805199.

snils()
Generate snils with special algorithm.

Return type str

Returns SNILS.

Example

-2147483648.

UkraineSpecProvider

class mimesis.builtins.**UkraineSpecProvider**(*seed=None*)
Class that provides special data for Ukraine (uk).

class Meta
The name of the provider.

__init__(*seed=None*)
Initialize attributes.

patronymic(*gender=None*)
Generate random patronymic name.

Parameters **gender** (*str or int*) – Gender of person.

Return type str

Returns Patronymic name.

USASpecProvider

class mimesis.builtins.**USASpecProvider**(*seed=None*)

Class that provides special data for USA (en).

class Meta

The name of the provider.

__init__(*seed=None*)

Initialize attributes.

personality(*category='mbti'*)

Generate a type of personality.

Parameters **category** (str) – Category.

Returns Personality type.

Return type str or int

Example ISFJ.

ssn()

Generate a random, but valid SSN.

Return type str

Returns SSN.

Example 569-66-5801

tracking_number(*service='usps'*)

Generate random tracking number.

Supported services: USPS, FedEx and UPS.

Parameters **service** (*str*) – Post service.

Return type str

Returns Tracking number.

PolandSpecProvider

class mimesis.builtins.**PolandSpecProvider**(*seed=None*)

Class that provides special data for Poland (pl).

class Meta

The name of the provider.

__init__(*seed=None*)

Initialize attributes.

nip()

Generate random valid 10-digit NIP.

Return type str

Returns Valid 10-digit NIP

pesel(*birth_date=None*, *gender=None*)
Generate random 11-digit PESEL.

Parameters

- **birth_date** (Optional[datetime]) – Initial birth date (optional)
- **gender** (Optional[*Gender*]) – Gender of person

Return type str

Returns Valid 11-digit PESEL

regon()
Generate random valid 9-digit REGON.

Return type str

Returns Valid 9-digit REGON

3.1.2 Decorators

Decorators for the public API and for internal purpose.

mimesis.decorators.**romanized**(*locale=''*)
Romanize the Cyrillic text.

Transliterate the Cyrillic language from the Cyrillic script into the Latin alphabet.

Note: At this moment it works only for *ru*, *uk*, *kk*.

Parameters **locale** (str) – Locale code.

Return type Callable

Returns Latinized text.

3.1.3 Custom Exceptions

UnsupportedAlgorithm

class mimesis.exceptions.**UnsupportedAlgorithm**
Raised when the user wants to use an unsupported algorithm.

UnsupportedField

class mimesis.exceptions.**UnsupportedField**(*name=None*)
Raises when field is not supported.

UnsupportedLocale

class mimesis.exceptions.**UnsupportedLocale**(*locale=None*)
Raised when a locale isn't supported.

UndefinedField

class mimesis.exceptions.**UndefinedField**
Raises when `field` is None.

UndefinedSchema

class mimesis.exceptions.**UndefinedSchema**
Raised when schema is empty.

UnacceptableField

class mimesis.exceptions.**UnacceptableField**
Raises when the `field` has an unacceptable format.

NonEnumerableError

class mimesis.exceptions.**NonEnumerableError**(*enum_obj*)
Raised when object is not instance of Enum.

3.1.4 Base Providers

BaseProvider

class mimesis.providers.**BaseProvider**(*seed=None*)
This is a base class for all providers.

__init__(*seed=None*)
Initialize attributes.

Parameters **seed** (Union[int, str, bytes, bytearray, None]) – Seed for random. When set to *None* the current system time is used.

Return type None

reseed(*seed=None*)
Reseed the internal random generator.

In case we use the default seed, we need to create a per instance random generator, in this case two providers with the same seed will always return the same values.

Parameters **seed** (Union[int, str, bytes, bytearray, None]) – Seed for random. When set to *None* the current system time is used.

Return type None

BaseDataProvider

class mimesis.providers.**BaseDataProvider**(*locale='en'*, *seed=None*)
This is a base class for all data providers.

__init__(*locale='en'*, *seed=None*)
Initialize attributes for data providers.

Parameters

- **locale** (str) – Current locale.
- **seed** (Union[int, str, bytes, bytearray, None]) – Seed to all the random functions.

Return type None

get_current_locale()
Get current locale.

If locale is not defined then this method will always return en, because en is default locale for all providers, excluding builtins.

Return type str

Returns Current locale.

override_locale(*locale='en'*)
Context manager which allows overriding current locale.

Temporarily overrides current locale for locale-dependent providers.

Parameters **locale** (str) – Locale.

Return type Generator[BaseDataProvider, None, None]

Returns Provider with overridden locale.

pull(*self*, *datafile=''*)
Pull the content from the JSON and memorize one.

Opens JSON file file in the folder data/locale and get content from the file and memorize ones using lru_cache.

Parameters **datafile** (str) – The name of file.

Returns The content of the file.

Raises ***UnsupportedLocale*** – if locale is not supported.

3.1.5 Generic Providers

Generic

class mimesis.**Generic**(**args*, ***kwargs*)
Class which contain all providers at one.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes lazily.

Parameters

- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type None

add_provider(*cls*)
Add a custom provider to Generic() object.

Parameters **cls** (Type[BaseProvider]) – Custom provider.

Return type None

Returns None

Raises **TypeError** – if cls is not class or is not a subclass of BaseProvider.

add_providers(**providers*)
Add a lot of custom providers to Generic() object.

Parameters **providers** (Type[BaseProvider]) – Custom providers.

Return type None

Returns None

3.1.6 Locale-Dependent Providers

Address

class mimesis.**Address**(**args*, ***kwargs*)
Class for generate fake address data.

This object provides all the data related to geographical location.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **locale** – Current locale.

Return type None

address()
Generate a random full address.

Return type str

Returns Full address.

calling_code()
Get a random calling code of random country.

Return type str

Returns Calling code.

city()
Get a random city.

Return type str

Returns City name.

continent(*code=False*)
Get a random continent name or continent code.

Parameters **code** (bool) – Return code of continent.

Return type str

Returns Continent name.

coordinates(*dms=False*)
Generate random geo coordinates.

Parameters **dms** (bool) – DMS format.

Return type dict

Returns Dict with coordinates.

country(*allow_random=False*)
Get the country of the current locale.

Allow_random Return a random country name.

Return type str

Returns The Country.

country_code(*fmt=<CountryCode.A2: 'a2'>*)
Get a random code of country.

Default format is *A2* (ISO 3166-1-alpha2), you can change it by passing parameter fmt with enum object *CountryCode*.

Parameters **fmt** (Optional[*CountryCode*]) – Enum object CountryCode.

Return type str

Returns Country code in selected format.

Raises **KeyError** – if fmt is not supported.

federal_subject(**args*, ***kwargs*)
Get a random region.

An alias for *state()*.

Return type str

latitude(*dms=False*)
Generate a random value of latitude.

Parameters **dms** (bool) – DMS format.

Return type Union[str, float]

Returns Value of longitude.

longitude(*dms=False*)
Generate a random value of longitude.

Parameters **dms** (bool) – DMS format.

Return type Union[str, float]

Returns Value of longitude.

postal_code()
Generate a postal code for current locale.

Return type str

Returns Postal code.

prefecture(**args*, ***kwargs*)
Get a random prefecture.

An alias for *state()*.

Return type str

province(**args*, ***kwargs*)
Get a random province.

An alias for *state()*.

Return type str

region(**args*, ***kwargs*)
Get a random region.

An alias for *state()*.

Return type str

state(*abbr=False*)
Get a random administrative district of country.

Parameters **abbr** (bool) – Return ISO 3166-2 code.

Return type str

Returns Administrative district.

street_name()
Get a random street name.

Return type str

Returns Street name.

street_number(*maximum=1400*)
Generate a random street number.

Parameters **maximum** (int) – Maximum value.

Return type str

Returns Street number.

street_suffix()
Get a random street suffix.

Return type str

Returns Street suffix.

zip_code()
Generate a zip code.

An alias for *postal_code()*.

Return type str

Returns Zip code.

Business

class mimesis.**Business**(**args*, ***kwargs*)
Class for generating data for business.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **locale** – Current locale.

company()
Get a random company name.

Return type str

Returns Company name.

company_type(*abbr=False*)
Get a random type of business entity.

Parameters **abbr** (bool) – Abbreviated company type.

Return type str

Returns Types of business entity.

copyright()
Generate a random copyright.

Return type str

Returns Copyright of company.

cryptocurrency_iso_code()
Get symbol of random cryptocurrency.

Return type str

Returns Symbol of cryptocurrency.

cryptocurrency_symbol()
Get a cryptocurrency symbol.

Return type str

Returns Symbol of cryptocurrency.

currency_iso_code(*allow_random=False*)
Get code of the currency for current locale.

Parameters **allow_random** (bool) – Get a random ISO code.

Return type str

Returns Currency code.

currency_symbol()
Get a currency symbol for current locale.

Returns Currency symbol.

price(*minimum=10.0*, *maximum=1000.0*)
Generate a random price.

Parameters

- **minimum** (float) – Max value of price.
- **maximum** (float) – Min value of price.

Return type str

Returns Price.

price_in_btc(*minimum=0*, *maximum=2*)
Generate random price in BTC.

Parameters

- **minimum** (float) – Minimum value of price.
- **maximum** (float) – Maximum value of price.

Return type str

Returns Price in BTC.

Datetime

class mimesis.**Datetime**(**args*, ***kwargs*)
Class for generating data related to the date and time.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **locale** – Current locale.

static bulk_create_datetimes(*date_start*, *date_end*, ***kwargs*)
Bulk create datetime objects.

This method creates list of datetime objects from date_start to date_end.

You can use the following keyword arguments:

- days
- hours
- minutes
- seconds
- microseconds

See datetime module documentation for more: https://docs.python.org/3.7/library/datetime.html#timedelta-objects

Parameters

- **date_start** (datetime) – Begin of the range.
- **date_end** (datetime) – End of the range.
- **kwargs** – Keyword arguments for datetime.timedelta

Return type List[datetime]

Returns List of datetime objects

Raises ValueError: When date_start/date_end not passed and when date_start larger than date_end.

century()
Get a random century.

Return type str

Returns Century.

date(*start=2000*, *end=2019*)
Generate random date object.

Parameters

- **start** (int) – Minimum value of year.
- **end** (int) – Maximum value of year.

Return type date

Returns Formatted date.

datetime(*start=2000*, *end=2035*, *timezone=None*)
Generate random datetime.

Parameters

- **start** (int) – Minimum value of year.
- **end** (int) – Maximum value of year.
- **timezone** (Optional[str]) – Set custom timezone (pytz required).

Return type datetime

Returns Datetime

day_of_month()
Generate a random day of month, from 1 to 31.

Return type int

Returns Random value from 1 to 31.

day_of_week(*abbr=False*)
Get a random day of week.

Parameters **abbr** (bool) – Abbreviated day name.

Return type str

Returns Day of the week.

formatted_date(*fmt=''*, ***kwargs*)
Generate random date as string.

Parameters

- **fmt** (str) – The format of date, if None then use standard accepted in the current locale.
- **kwargs** – Keyword arguments for *date()*

Return type str

Returns Formatted date.

formatted_datetime(*fmt=''*, ***kwargs*)
Generate datetime string in human readable format.

Parameters

- **fmt** (str) – Custom format (default is format for current locale)
- **kwargs** – Keyword arguments for *datetime()*

Return type str

Returns Formatted datetime string.

formatted_time(*fmt=''*)
Generate string formatted time.

Parameters **fmt** (str) – The format of time, if None then use standard accepted in the current locale.

Return type str

Returns String formatted time.

gmt_offset()
Get a random GMT offset value.

Return type str

Returns GMT Offset.

month(*abbr=False*)
Get a random month.

Parameters **abbr** (bool) – Abbreviated month name.

Return type str

Returns Month name.

periodicity()
Get a random periodicity string.

Return type str

Returns Periodicity.

time()
Generate a random time object.

Return type time

Returns datetime.time object.

timestamp(*posix=True*, ***kwargs*)
Generate random timestamp.

Parameters

- **posix** (bool) – POSIX time.
- **kwargs** – Kwargs for *datetime()*.

Return type Union[str, int]

Returns Timestamp.

timezone()
Get a random timezone.

Return type str

Returns Timezone.

week_date(*start=2017*, *end=2018*)
Get week number with year.

Parameters

- **start** (int) – From start.
- **end** (int) – To end.

Return type str

Returns Week number.

year(*minimum=1990*, *maximum=2050*)
Generate a random year.

Parameters

- **minimum** (int) – Minimum value.
- **maximum** (int) – Maximum value.

Return type int

Returns Year.

Food

class mimesis.**Food**(**args*, ***kwargs*)
Class for generating data related to food.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **locale** – Current locale.

dish()
Get a random dish.

Return type str

Returns Dish name.

Example Ratatouille.

drink()
Get a random drink.

Return type str

Returns Alcoholic drink.

Example Vodka.

fruit()
Get a random fruit or berry.

Return type str

Returns Fruit name.

Example Banana.

spices()
Get a random spices or herbs.

Return type str

Returns Spices or herbs.

Example Anise.

vegetable()
Get a random vegetable.

Return type str

Returns Vegetable name.

Example Tomato.

Person

class mimesis.**Person**(**args*, ***kwargs*)
Class for generating personal data.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **locale** – Current locale.
- **seed** – Seed.

Return type None

academic_degree()
Get a random academic degree.

Return type str

Returns Degree.

Example Bachelor.

age(*minimum=16*, *maximum=66*)
Get a random integer value.

Parameters

- **maximum** (int) – Maximum value of age.
- **minimum** (int) – Minimum value of age.

Return type int

Returns Random integer.

Example

23.

avatar(*size=256*)
Generate a random avatar..

Parameters **size** (int) – Size of avatar.

Return type str

Returns Link to avatar.

blood_type()
Get a random blood type.

Return type str

Returns Blood type (blood group).

Example A+

email(*domains=None*)
Generate a random email.

Parameters **domains** (*list or tuple*) – List of custom domains for emails.

Return type str

Returns Email address.

Example foretime10@live.com

first_name(*gender=None*)
Generate a random first name.

..note: An alias for self.name().

Parameters **gender** (Optional[*Gender*]) – Gender's enum object.

Returns First name.

full_name(*gender=None*, *reverse=False*)
Generate a random full name.

Parameters

- **reverse** (bool) – Return reversed full name.
- **gender** (Optional[*Gender*]) – Gender's enum object.

Return type str

Returns Full name.

Example Johann Wolfgang.

gender(*iso5218=False*, *symbol=False*)
Get a random gender.

Get a random title of gender, code for the representation of human sexes is an international standard that defines a representation of human sexes through a language-neutral single-digit code or symbol of gender.

Parameters

- **iso5218** (bool) – Codes for the representation of human sexes is an international standard (0 - not known, 1 - male, 2 - female, 9 - not applicable).
- **symbol** (bool) – Symbol of gender.

Return type Union[str, int]

Returns Title of gender.

Example Male

height(*minimum=1.5*, *maximum=2.0*)
Generate a random height in M (Meter).

Parameters

- **minimum** (float) – Minimum value.
- **maximum** (*float*) – Maximum value.

Return type str

Returns Height.

Example 1.85.

identifier(*mask='##-##/##'*)
Generate a random identifier by mask.

With this method you can generate any identifiers that you need. Simply select the mask that you need.

Parameters **mask** (str) – The mask. Here @ is a placeholder for characters and # is placeholder for digits.

Return type str

Returns An identifier.

Example 07-97/04

language()
Get a random language.

Return type str

Returns Random language.

Example Irish.

last_name(*gender=None*)
Generate a random last name.

..note: An alias for self.surname().

Parameters **gender** (Optional[*Gender*]) – Gender's enum object.

Return type str

Returns Last name.

name(*gender=None*)
Generate a random name.

Parameters **gender** (Optional[*Gender*]) – Gender's enum object.

Return type str

Returns Name.

Example John.

nationality(*gender=None*)
Get a random nationality.

Parameters **gender** (Optional[*Gender*]) – Gender.

Return type str

Returns Nationality.

Example Russian

occupation()
Get a random job.

Return type str

Returns The name of job.

Example Programmer.

password(*length=8*, *hashed=False*)
Generate a password or hash of password.

Parameters

- **length** (int) – Length of password.
- **hashed** (bool) – MD5 hash.

Return type str

Returns Password or hash of password.

Example k6dv2odff9#4h

political_views()
Get a random political views.

Return type str

Returns Political views.

Example Liberal.

sexual_orientation(*symbol=False*)
Get a random (LOL) sexual orientation.

Parameters **symbol** (bool) – Unicode symbol.

Return type str

Returns Sexual orientation.

Example Heterosexuality.

social_media_profile(*site=None*)
Generate profile for random social network.

Return type str

Returns Profile in some network.

Example http://facebook.com/some_user

surname(*gender=None*)
Generate a random surname.

Parameters **gender** (Optional[*Gender*]) – Gender's enum object.

Return type str

Returns Surname.

Example Smith.

telephone(*mask=''*, *placeholder='#'*)
Generate a random phone number.

Parameters

- **mask** (str) – Mask for formatting number.
- **placeholder** (str) – A placeholder for a mask (default is #).

Return type str

Returns Phone number.

Example +7-(963)-409-11-22.

title(*gender=None*, *title_type=None*)
Generate a random title for name.

You can generate random prefix or suffix for name using this method.

Parameters

- **gender** (Optional[*Gender*]) – The gender.
- **title_type** (Optional[*TitleType*]) – TitleType enum object.

Return type str

Returns The title.

Raises ***NonEnumerableError*** – if gender or title_type in incorrect format.

Example PhD.

university()
Get a random university.

Return type str

Returns University name.

Example MIT.

username(*template=None*)
Generate username by template.

Supported template placeholders: (U, l, d)

Supported separators: (-, ., _)

Template must contain at least one "U" or "l" placeholder.

If template is None one of the following templates is used: ('U_d', 'U.d', 'U-d', 'UU-d', 'UU.d', 'UU_d', 'ld', 'l-d', 'Ud', 'l.d', 'l_d', 'default')

Parameters **template** (Optional[str]) – Template.

Return type str

Returns Username.

Raises **ValueError** – If template is not supported.

Example Celloid1873

views_on()
Get a random views on.

Return type str

Returns Views on.

Example Negative.

weight(*minimum=38*, *maximum=90*)
Generate a random weight in Kg.

Parameters

- **minimum** (int) – min value
- **maximum** (int) – max value

Return type int

Returns Weight.

Example

48.

work_experience(*working_start_age=22*)
Get a work experience.

Parameters **working_start_age** (int) – Age then person start to work.

Return type int

Returns Depend on previous generated age.

worldview()
Get a random worldview.

Return type str

Returns Worldview.

Example Pantheism.

Science

class mimesis.**Science**(**args*, ***kwargs*)
Class for generating pseudo-scientific data.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **locale** – Current language.
- **seed** – Seed.

atomic_number()
Generate random atomic number.

Return type int

Returns Atomic number

Example 92

chemical_element(*name_only=True*)
Generate a random chemical element.

Parameters **name_only** (bool) – If False then will be returned dict.

Returns Name of chemical element or dict.

Return type dict or str

Example {'Symbol': 'S', 'Name': 'Sulfur', 'Atomic number': '16'}

dna_sequence(*length=10*)
Generate a random DNA sequence.

Parameters **length** (int) – Length of block.

Return type str

Returns DNA sequence.

Example GCTTTAGACC

math_formula()
Get a random mathematical formula.

Return type str

Returns Math formula.

Example A = (ab)/2.

rna_sequence(*length=10*)
Generate a random RNA sequence.

Parameters **length** (int) – Length of block.

Return type str

Returns RNA sequence.

Example AGUGACACAA

Text

class mimesis.**Text**(**args*, ***kwargs*)
Class for generating text data.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **locale** – Current locale.
- **seed** – Seed.

alphabet(*lower_case=False*)
Get an alphabet for current locale.

Parameters **lower_case** (bool) – Return alphabet in lower case.

Return type list

Returns Alphabet.

answer()
Get a random answer in current language.

Return type str

Returns An answer.

Example No

color()
Get a random name of color.

Return type str

Returns Color name.

Example Red.

hex_color(*safe=False*)
Generate a random hex color.

Parameters **safe** (bool) – Get safe Flat UI hex color.

Return type str

Returns Hex color code.

Example #d8346b

level()
Generate a random level of danger or something else.

Return type str

Returns Level.

Example critical.

quote()
Get a random quote.

Return type str

Returns Quote from movie.

Example "Bond... James Bond."

rgb_color(*safe=False*)
Generate a random rgb color tuple.

Parameters **safe** (bool) – Get safe RGB tuple.

Return type Tuple[int, ...]

Returns RGB tuple.

Example (252, 85, 32)

sentence()
Get a random sentence from text.

Return type str

Returns Sentence.

swear_word()
Get a random swear word.

Return type str

Returns Swear word.

Example Damn.

text(*quantity=5*)
Generate the text.

Parameters **quantity** (int) – Quantity of sentences.

Return type str

Returns Text.

title()
Get a random title.

Return type str

Returns The title.

word()
Get a random word.

Return type str

Returns Single word.

Example Science.

words(*quantity=5*)
Generate lis of the random words.

Parameters **quantity** (int) – Quantity of words. Default is 5.

Return type List[str]

Returns Word list.

Example [science, network, god, octopus, love]

3.1.7 Locale-Independent Providers

Clothing

class mimesis.**Clothing**(*seed=None*)
Class for generate data related to clothing.

class Meta
Class for metadata.

custom_size(*minimum=40*, *maximum=62*)
Generate clothing size using custom format.

Parameters

- **minimum** (int) – Minimum value.
- **maximum** (int) – Maximum value.

Return type int

Returns Clothing size.

european_size()
Generate a random clothing size in European format.

Return type int

Returns Clothing size.

international_size()
Get a random size in international format.

Return type str

Returns Clothing size.

Code

class mimesis.**Code**(**args*, ***kwargs*)
Class that provides methods for generating codes.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **locale** – Current locale.

ean(*fmt=None*)
Generate EAN.

To change EAN format, pass parameter `fmt` with needed value of the enum object *EANFormat*.

Parameters **fmt** (Optional[*EANFormat*]) – Format of EAN.

Return type str

Returns EAN.

Raises ***NonEnumerableError*** – if fmt is not enum EANFormat.

imei()
Generate a random IMEI.

Return type str

Returns IMEI.

isbn(*fmt=None*, *locale='en'*)
Generate ISBN for current locale.

To change ISBN format, pass parameter `fmt` with needed value of the enum object *ISBNFormat*

Parameters

- **fmt** (Optional[*ISBNFormat*]) – ISBN format.
- **locale** (str) – Locale code.

Return type str

Returns ISBN.

Raises ***NonEnumerableError*** – if fmt is not enum ISBNFormat.

issn(*mask='####-####'*)
Generate a random ISSN.

Parameters **mask** (str) – Mask of ISSN.

Return type str

Returns ISSN.

locale_code()
Get a random locale code (MS-LCID).

See Windows Language Code Identifier Reference for more information.

Return type str

Returns Locale code.

pin(*mask='####'*)
Generate a random PIN code.

Parameters **mask** (str) – Mask of pin code.

Return type str

Returns PIN code.

Choice

class mimesis.**Choice**(**args*, ***kwargs*)
Class for generating a random choice from items in a sequence.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **args** – Arguments.
- **kwargs** – Keyword arguments.

Return type None

Cryptographic

class mimesis.**Cryptographic**(**args*, ***kwargs*)
Class that provides cryptographic data.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters **seed** – Seed.

Return type None

hash(*algorithm=None*)
Generate random hash.

To change hashing algorithm, pass parameter algorithm with needed value of the enum object *Algorithm*

Parameters **algorithm** (Optional[*Algorithm*]) – Enum object *Algorithm*.

Return type str

Returns Hash.

Raises ***NonEnumerableError*** – if algorithm is not supported.

mnemonic_phrase(*length=12*)
Generate pseudo mnemonic phrase.

Parameters **length** (int) – Number of words.

Return type str

Returns Mnemonic code.

static token_bytes(*entropy=32*)
Generate byte string containing `entropy` bytes.

The string has `entropy` random bytes, each byte converted to two hex digits.

> **Warning:** Seed is not applicable to this method, because of its cryptographic-safe nature.

Parameters **entropy** (`int`) – Number of bytes (default: 32).

Return type `bytes`

Returns Random bytes.

static token_hex(*entropy=32*)
Return a random text string, in hexadecimal.

The string has *entropy* random bytes, each byte converted to two hex digits. If *entropy* is `None` or not supplied, a reasonable default is used.

> **Warning:** Seed is not applicable to this method, because of its cryptographic-safe nature.

Parameters **entropy** (`int`) – Number of bytes (default: 32).

Return type `str`

Returns Token.

static token_urlsafe(*entropy=32*)
Return a random URL-safe text string, in Base64 encoding.

The string has *entropy* random bytes. If *entropy* is `None` or not supplied, a reasonable default is used.

> **Warning:** Seed is not applicable to this method, because of its cryptographic-safe nature.

Parameters **entropy** (`int`) – Number of bytes (default: 32).

Returns URL-safe token.

uuid(*version=None*)
Generate random UUID.

Parameters **version** (`Optional[int]`) – UUID version.

Return type `str`

Returns UUID

Development

class mimesis.**Development**(*seed=None*)
Class for getting fake data for Developers.

class Meta
Class for metadata.

boolean()
Get a random boolean value.

Return type bool

Returns True of False.

os()
Get a random operating system or distributive name.

Return type str

Returns The name of OS.

Example Gentoo

programming_language()
Get a random programming language from the list.

Return type str

Returns Programming language.

Example Erlang.

software_license()
Get a random software license.

Return type str

Returns License name.

Example The BSD 3-Clause License.

version(*calver=False*, *pre_release=False*)
Generate version number.

Parameters

- **calver** (bool) – Calendar versioning.
- **pre_release** (bool) – Pre-release.

Return type str

Returns Version.

Example 0.2.1

File

class mimesis.**File**(**args*, ***kwargs*)
Class for generate data related to files.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **args** – Arguments.

- **kwargs** – Keyword arguments.

extension(*file_type=None*)
Get a random file extension from list.

Parameters **file_type** (Optional[*FileType*]) – Enum object FileType.

Return type str

Returns Extension of the file.

Example .py

file_name(*file_type=None*)
Get a random file name with some extension.

Parameters **file_type** (Optional[*FileType*]) – Enum object FileType

Return type str

Returns File name.

Example legislative.txt

mime_type(*type_=None*)
Get a random mime type from list.

Parameters **type** – Enum object MimeType.

Return type str

Returns Mime type.

size(*minimum=1*, *maximum=100*)
Get size of file.

Parameters

- **minimum** (int) – Maximum value.
- **maximum** (int) – Minimum value.

Return type str

Returns Size of file.

Example 56 kB

Hardware

class mimesis.**Hardware**(*seed=None*)
Class for generate data related to hardware.

class Meta
Class for metadata.

cpu()
Get a random CPU name.

Return type str

Returns CPU name.

Example Intel® Core i7.

cpu_codename()
Get a random CPU code name.

Return type str

Returns CPU code name.

Example Cannonlake.

cpu_frequency()
Get a random frequency of CPU.

Return type str

Returns Frequency of CPU.

Example 4.0 GHz.

cpu_model_code()
Get a random CPU model.

Return type str

Returns CPU model.

generation()
Get a random generation.

Return type str

Returns Generation of something.

Example 6th Generation.

graphics()
Get a random graphics.

Return type str

Returns Graphics.

Example Intel® Iris™ Pro Graphics 6200.

manufacturer()
Get a random manufacturer.

Return type str

Returns Manufacturer.

Example Dell.

phone_model()
Get a random phone model.

Return type str

Returns Phone model.

Example Nokia Lumia 920.

ram_size()
Get a random size of RAM.

Return type str

Returns RAM size.

Example 16GB.

ram_type()
Get a random RAM type.

Return type str

Returns Type of RAM.

Example DDR3.

resolution()
Get a random screen resolution.

Return type str

Returns Resolution of screen.

Example 1280x720.

screen_size()
Get a random size of screen in inch.

Return type str

Returns Screen size.

Example 13.

ssd_or_hdd()
Get a random value from list.

Return type str

Returns HDD or SSD.

Example 512GB SSD.

Internet

class mimesis.**Internet**(**args*, ***kwargs*)
Class for generating data related to the internet.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **args** – Arguments.
- **kwargs** – Keyword arguments.

content_type(*mime_type=None*)
Get a random HTTP content type.

Return type str

Returns Content type.

Example Content-Type: application/json

emoji()
Get a random emoji shortcut code.

Return type str

Returns Emoji code.

Example

kissing

hashtags(*quantity=4*)
Generate a list of hashtags.

Parameters **quantity** (int) – The quantity of hashtags.

Return type Union[str, list]

Returns The list of hashtags.

Raises ***NonEnumerableError*** – if category is not in Hashtag.

Example ['#love', '#sky', '#nice']

home_page(*tld_type=None*)
Generate a random home page.

Parameters **tld_type** (Optional[*TLDType*]) – TLD type.

Return type str

Returns Random home page.

Example http://www.fontir.info

http_method()
Get a random HTTP method.

Return type str

Returns HTTP method.

Example POST

http_status_code()
Get a random HTTP status code.

Return type int

Returns HTTP status.

Example 200

http_status_message()
Get a random HTTP status message.

Return type str

Returns HTTP status message.

Example 200 OK

static image_placeholder(*width=1920*, *height=1080*)
Generate a link to the image placeholder.

Parameters

- **width** (Union[int, str]) – Width of image.
- **height** (Union[int, str]) – Height of image.

Return type str

Returns URL to image placeholder.

ip_v4(*with_port=False*)
Generate a random IPv4 address.

Parameters **with_port** (bool) – Add port to IP.

Return type str

Returns Random IPv4 address.

Example 19.121.223.58

ip_v6()
Generate a random IPv6 address.

Return type str

Returns Random IPv6 address.

Example 2001:c244:cf9d:1fb1:c56d:f52c:8a04:94f3

mac_address()
Generate a random MAC address.

Return type str

Returns Random MAC address.

Example 00:16:3e:25:e7:b1

network_protocol(*layer=None*)
Get a random network protocol form OSI model.

Parameters **layer** (Optional[*Layer*]) – Enum object Layer.

Return type str

Returns Protocol name.

Example AMQP

port(*port_range=<PortRange.ALL: (1, 65535)>*)
Generate random port.

Parameters **port_range** (*PortRange*) – Range enum object.

Return type int

Returns Port number.

Raises ***NonEnumerableError*** – if port_range is not in PortRange.

Example 8080

static stock_image(*width=1920*, *height=1080*, *keywords=None*, *writable=False*)
Generate random stock image (JPEG) hosted on Unsplash.

Note: This method required an active HTTP connection.

Parameters

- **width** (Union[int, str]) – Width of the image.
- **height** (Union[int, str]) – Height of the image.
- **keywords** (Optional[List[str]]) – List of search keywords.
- **writable** (bool) – Return image as sequence ob bytes.

Return type Union[str, bytes]

Returns Link to the image.

top_level_domain(*tld_type=None*)
Return random top level domain.

Parameters **tld_type** (Optional[*TLDType*]) – Enum object DomainType

Return type str

Returns Top level domain.

Raises ***NonEnumerableError*** – if tld_type not in DomainType.

user_agent()
Get a random user agent.

Return type str

Returns User agent.

Example Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Gecko/20100101 Firefox/15.0.1

Numbers

class mimesis.**Numbers**(*seed=None*)
Class for generating numbers.

class Meta
Class for metadata.

between(*minimum=1*, *maximum=1000*)
Generate a random number between minimum and maximum.

Parameters

- **minimum** (int) – Minimum of range.
- **maximum** (int) – Maximum of range.

Return type int

Returns Number.

digit(*to_bin=False*)
Get a random digit.

Parameters **to_bin** (bool) – If True then convert to binary.

Return type Union[str, int]

Returns Digit.

Example

4.

floats(*n=2*)
Generate a list of random float numbers.

Parameters **n** (int) – Raise 10 to the 'n' power.

Return type List[float]

Returns The list of floating-point numbers.

integers(*start=0*, *end=10*, *length=10*)
Generate a list of random integers.

Integers can be negative or positive numbers. .. note: You can use both positive and negative numbers.

Parameters

- **start** (int) – Start.
- **end** (int) – End.
- **length** (int) – Length of list.

Return type List[int]

Returns List of integers.

Example [-20, -19, -18, -17]

static primes(*start=1*, *end=999*)
Generate a list of prime numbers.

Parameters

- **start** (int) – First value of range.
- **end** (int) – Last value of range.

Return type List[int]

Returns A list of prime numbers from start to end.

rating(*maximum=5.0*)
Generate a random rating for something.

Parameters **maximum** (float) – Maximum value (default is 5.0).

Return type float

Returns Rating.

Example 4.7

Path

class mimesis.**Path**(*platform='linux'*, **args*, ***kwargs*)
Class that provides methods and property for generate paths.

class Meta
Class for metadata.

__init__(*platform='linux'*, **args*, ***kwargs*)
Initialize attributes.

Supported platforms: 'linux', 'darwin', 'win32', 'win64'.

Parameters **platform** (str) – Required platform type.

Return type None

dev_dir()
Generate a random path to development directory.

Return type str

Returns Path.

Example /home/sherrell/Development/Python

home()
Generate a home path.

Return type str

Returns Home path.

Example /home

project_dir()
Generate a random path to project directory.

Return type str

Returns Path to project.

Example /home/sherika/Development/Falcon/mercenary

root()
Generate a root dir path.

Return type str

Returns Root dir.

Example /

user()
Generate a random user.

Return type str

Returns Path to user.

Example /home/oretha

users_folder()
Generate a random path to user's folders.

Return type str

Returns Path.

Example /home/taneka/Pictures

Structure

class mimesis.**Structure**(**args*, ***kwargs*)
Class for generating structured data.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **locale** – Current locale.
- **seed** – Seed.

Return type None

css()
Generate a random snippet of CSS.

Return type str

Returns CSS.

css_property()
Generate a random snippet of CSS that assigns value to a property.

Return type str

Returns CSS property.

Examples 'background-color: #f4d3a1'

html()
Generate a random HTML tag with text inside and some attrs set.

Return type str

Returns HTML.

Examples ' Ports are created with the built-in function open_port. '

html_attribute_value(*tag=None*, *attribute=None*)
Generate random value for specified HTML tag attribute.

Parameters

- **tag** (Optional[str]) – An HTML tag.
- **attribute** (Optional[str]) – An attribute of the specified tag.

Return type str

Returns An attribute.

Raises **NotImplementedError** – if tag is unsupported.

Transport

class mimesis.**Transport**(**args*, ***kwargs*)
Class for generating data related to transports.

class Meta
Class for metadata.

__init__(**args*, ***kwargs*)
Initialize attributes.

Parameters

- **locale** – Current locale.
- **seed** – Seed.

Return type None

airplane(*model_mask='###'*)
Generate a dummy airplane model.

Parameters **model_mask** (str) – Mask of truck model. Here '@' is a placeholder of characters and '#' is a placeholder of digits.

Return type str

Returns Airplane model.

Example Boeing 727.

car()
Get a random vehicle.

Return type str

Returns A vehicle.

Example Tesla Model S.

truck(*model_mask='#### @@'*)
Generate a truck model.

Parameters **model_mask** (str) – Mask of truck model. Here '@' is a placeholder of characters and '#' is a placeholder of digits.

Return type str

Returns Dummy truck model.

Example Caledon-966O.

vehicle_registration_code(*locale=None*)
Get vehicle registration code of country.

Parameters **locale** (Optional[str]) – Registration code for locale (country).

Return type str

Returns Vehicle registration code.

UnitSystem

class mimesis.**UnitSystem**(*seed=None*)
Class for generating data related to units.

class Meta
Class for metadata.

prefix(*sign=None*, *symbol=False*)
Get a random prefix for the International System of Units.

Parameters

- **sign** (Optional[*PrefixSign*]) – Sing of number.
- **symbol** (bool) – Return symbol of prefix.

Return type str

Returns Prefix for SI.

Raises ***NonEnumerableError*** – if sign is not supported.

Example mega

unit(*name=None*, *symbol=False*)
Get unit name.

Parameters

- **name** (Optional[*UnitName*]) – Enum object UnitName.

- **symbol** – Return only symbol

Returns Unit.

3.1.8 Schema

AbstractField

class mimesis.schema.**AbstractField**(*locale='en'*, *seed=None*, *providers=None*)

AbstractField is a class for generating data by the name of the method.

Instance of this object takes any string which represents name of any method of any supported data provider (*Generic*) and the `**kwargs` of the method:

```
>>> _ = AbstractField('en', 0xf)
>>> surname = _('surname')
>>> isinstance(surname, str)
True
```

Field

mimesis.schema.**Field**

alias of *mimesis.schema.AbstractField*

Schema

class mimesis.schema.**Schema**(*schema*)

Class which return list of filled schemas.

create(*iterations=1*)

Return filled schema.

Create a list of a filled schemas with elements in an amount of **iterations**.

Parameters **iterations** (int) – Amount of iterations.

Return type List[Dict[str, Any]]

Returns List of willed schemas.

3.1.9 Enums

Implements enums for a lot of methods.

Enums from this module are used in a lot of methods. You should always import enums from this module if you want behavior for the methods that differ from the default behavior.

You should never use your own enums in methods because in this case, there no guarantee that you will get the result which you actually expected.

Below you can see an example of usage enums in methods of data providers.

Example:

```
>>> from mimesis import Person
>>> from mimesis.enums import Gender
>>> person = Person()
>>> name = person.name(gender=Gender.FEMALE)
>>> name in person._data['names']['female']
True
```

class mimesis.enums.**Algorithm**
Provides algorithms which available.

MD5 = 'md5'

SHA1 = 'sha1'

SHA224 = 'sha224'

SHA256 = 'sha256'

SHA384 = 'sha384'

SHA512 = 'sha512'

class mimesis.enums.**CardType**
Provides credit card types.

An argument for `credit_card_number()`.

AMERICAN_EXPRESS = 'American Express'

MASTER_CARD = 'MasterCard'

VISA = 'Visa'

class mimesis.enums.**CountryCode**
Provides types of country codes.

An argument for *country_code()*.

A2 = 'a2'

A3 = 'a3'

FIFA = 'fifa'

IOC = 'ioc'

NUMERIC = 'numeric'

class mimesis.enums.**EANFormat**
Provides formats of EAN.

An argument for *ean()*.

EAN13 = 'ean-13'

EAN8 = 'ean-8'

class mimesis.enums.**FileType**
Provides file types.

AUDIO = 'audio'

COMPRESSED = 'compressed'

DATA = 'data'

EXECUTABLE = 'executable'

IMAGE = 'image'

SOURCE = 'source'

TEXT = 'text'

VIDEO = 'video'

class mimesis.enums.**Gender**
Represents genders.

An argument for a lot of methods which takes argument `gender`.

FEMALE = 'female'

MALE = 'male'

class mimesis.enums.**ISBNFormat**
Provides formats of ISBN.

An argument for *isbn()*.

ISBN10 = 'isbn-10'

ISBN13 = 'isbn-13'

class mimesis.enums.**Layer**
Provides network protocol layers.

An argument for *network_protocol()*.

APPLICATION = 'application'

DATA_LINK = 'data_link'

NETWORK = 'network'

PHYSICAL = 'physical'

PRESENTATION = 'presentation'

SESSION = 'session'

TRANSPORT = 'transport'

class mimesis.enums.**MimeType**
Provides common mime types.

An argument for *mime_type()*.

APPLICATION = 'application'

AUDIO = 'audio'

IMAGE = 'image'

MESSAGE = 'message'

TEXT = 'text'

VIDEO = 'video'

class mimesis.enums.**PortRange**
Represents port ranges.

An argument for *port()*.

ALL = (1, 65535)

EPHEMERAL = (49152, 65535)

REGISTERED = (1024, 49151)

WELL_KNOWN = (1, 1023)

class mimesis.enums.**PrefixSign**
Provides prefix signs.

An argument for `prefix()`().

NEGATIVE = 'negative'

POSITIVE = 'positive'

class mimesis.enums.**SocialNetwork**
Provides most popular social networks.

An argument for `social_media_profile()`().

FACEBOOK = 'facebook'

INSTAGRAM = 'instagram'

TWITTER = 'twitter'

VK = 'vk'

class mimesis.enums.**TLDType**
Provides top level domain types.

An argument for *top_level_domain()*.

CCTLD = 'cctld'

GEOTLD = 'geotld'

GTLD = 'gtld'

STLD = 'stld'

UTLD = 'utld'

class mimesis.enums.**TitleType**
Represents title types.

An argument for *title()*.

ACADEMIC = 'academic'

TYPICAL = 'typical'

class mimesis.enums.**UnitName**
Provide unit names.

An argument for *unit()*.

AMOUNT_OF_SUBSTANCE = ('mole', 'mol')

ANGLE = ('radian', 'r')

ELECTRICAL_CONDUCTANCE = ('siemens', 'S')

ELECTRIC_CAPACITANCE = ('farad', 'F')

ELECTRIC_CHARGE = ('coulomb', 'C')

ELECTRIC_RESISTANCE = ('ohm', '')

ENERGY = ('joule', 'J')

FLUX = ('watt', 'W')

```
FORCE = ('newton', 'N')
FREQUENCY = ('hertz', 'Hz')
INDUCTANCE = ('henry', 'H')
INFORMATION = ('byte', 'b')
MAGNETIC_FLUX = ('weber', 'Wb')
MAGNETIC_FLUX_DENSITY = ('tesla', 'T')
MASS = ('gram', 'gr')
POWER = ('watt', 'W')
PRESSURE = ('pascal', 'P')
RADIOACTIVITY = ('becquerel', 'Bq')
SOLID_ANGLE = ('steradian', '')
TEMPERATURE = ('Celsius', '°C')
THERMODYNAMIC_TEMPERATURE = ('kelvin', 'K')
VOLTAGE = ('volt', 'V')
```

CHAPTER

FOUR

ADDITIONAL INFORMATION

Disclaimer, legal information and other information are here for the interested.

4.1 License

4.2 Contributors

Mimesis is written and maintained by Isaak Uchakaev (Likid Geimfari: lk-geimfari) and various contributors:

4.2.1 Maintainers

- Likid Geimfari (lk-geimfari)
- Sobolev Nikita (sobolevn)
- Emilio Cecchini (ceccoemi)

4.2.2 Patches and Suggestions

- Kevin Schellenberg (wikkiewikkie)

- Casey Weed (Battleroid)
- Alessandro Martini (martini97)
- Amin Alaee (aminalaee)
- Baurzhan Muftakhidinov (crayxt)
- Benjamin Schwarze (benjixx)
- Bill DeRusha (bderusha)
- David Poggi (drpoggi)
- Eliz Kiliç (el)
- Flavio Curella (fcurella)
- FliegendeWurst (FliegendeWurst)
- JLWT90 (jlwt90)
- Jack McMorrow (jackmcmorrow)
- Jakub Wilk (jwilk)
- Jeremy Costava (Costava)
- Jin Yang (redus)
- Json701 (jasonwaiting-dev)
- Jérôme Christ (jeromechrist)
- Michael Crilly (mrcrilly)
- Michael Hand (mipaaa)
- Paul Walters (PaulWaltersDev)
- Philipp Offermann (offermann)
- Sobolev Nikita (sobolevn)
- Rafael Passos (auyer)
- Ranwise (ranwise)
- Sambuddha Basu (sammyshj)
- Thomas Carroll (Uncleleech)
- Tsimpitas Dimitris TsimpDim
- Vladislav Glinsky (cl0ne)
- Yn-Coder (yn-coder)
- Dmytro Zelinskyi (zelds)
- axcel (axce1)
- Ruslan Valerievich (Valerievich)
- Simon (DefaltSimon)
- dy (duckyou)

4.3 Disclaimer

The authors do not assume any responsibility for how you use this library or how you use data generated with it. This library is designed only for developers and only with good intentions. Do not use the data generated with Mimesis for illegal purposes.

4.4 Contributing Guidelines

The source code and issue tracker are hosted on GitHub. *Mimesis* is tested against Python 3.6 through 3.7 on Travis-CI and *AppVeyor_*. Test coverage is monitored with Codecov.

4.4.1 Dependencies

Mimesis does not have any need for third-party tools, but we use a lot of tools in development stage, which you should install on your system if you want to contribute.

We use `pipenv` to manage development dependencies. So, please do not use `virtualenv` or `pip` directly.

Firstly, install `pipenv`, it is recommended to do so with `pip`:

```
~  pip install pipenv
```

Installing all dependencies

Please, note that `pipenv` will automatically create a `virtualenv` for this project. It will use `python_version` specified in `Pipfile`. To install (or renew) all existing dependencies run:

```
pipenv install -d
```

Activating virtualenv

And to activate `virtualenv` created by `pipenv` run:

```
pipenv shell
```

Adding new dependencies

To add a new dependency you can run:

- `pipenv install -d pytest` to install `pytest` as a development dependency

4.4.2 Code Style

Every contributor must follow the PEP8 code style.

4.4.3 Annotating

We use optional static typing (mypy). Every function and method should be annotated.

Example of annotated function:

```python
def plus(a: int = 0, b: int = 0) -> int:
    """Get sum of a and b.

    :param a: First number.
    :param b: Second number.
    :return: Sum of a and b.
    """
    return a + b
```

4.4.4 Documenting

Always add docstrings for your modules, classes, methods and functions. Below you can see a great example of module:

```python
"""Demonstrate high quality docstrings.

Module-level docstrings appear as the first "statement" in a module. Remember,
that while strings are regular Python statements, comments are not, so an
inline comment may precede the module-level docstring.

After importing a module, you can access this special string object through the
``__doc__`` attribute; yes, it's actually available as a runtime attribute,
despite not being given an explicit name! The ``__doc__`` attribute is also
what is rendered when you call ``help()`` on a module, or really any other
object in Python.

You can also document a package using the module-level docstring in the
package's ``__init__.py`` file.

"""


class Example(object):
    """Illustrate class-level docstring.

    Classes use a special whitespace convention: the opening and closing quotes
    are preceded and followed by a blank line, respectively. No other
    docstrings should be preceded or followed by anything but code.

    A blank line at the end of a multi-line docstring before the closing
    quotation marks simply makes it easier for tooling to auto-format
    paragraphs (wrapping them at 79 characters, per PEP8), without the closing
    quotation marks interfering.

    """

    def __init__(self, *args, **kwargs) -> None:
        """Illustrate method-level docstring.

        All public callables should have docstrings, including magic methods
        like ``__init__()``.

        You'll notice that all these docstrings are wrapped in triple double
        quotes, as opposed to just "double quotes", 'single quotes', or
        '''triple single quotes.''' This is a convention for consistency and
        readability.
```

(continues on next page)

(continued from previous page)

```
        ..note:: Note must look like that.

        :param foo: Description of foo.
        :param bar: Description of bar.

        """
        super().__init__(*args, **kwargs)

    def foo(self) -> str:
        """Return 'foo'.

        You can also specify summary with a lot of details about
        how the method works on multiple lines if it's really needed.

        :return: String ``foo``
        """
        return 'foo'


def pi() -> float:
    """Illustrate function-level docstring.

    Note that all docstrings begin with a one-line summary. The summary is
    written in the imperative mood ("do", "use", "find", "return", "render",
    etc) and ends with a period. The method signature is not, in any way,
    duplicated into the comments (that would be difficult to maintain).

    All subsequent paragraphs in a docstring are indented exactly the same as
    the summary line. The same applies to the closing quotation marks.

    """
    return 3.14
```

Comment only things that are not obvious: hacks, optimizations, complex algorithms. Obvious code does not require any additional comments.

4.4.5 Testing

You should write the test which shows that the bug was fixed or that the feature works as expected, run test before you commit your changes to the branch and create PR.

To run tests, simply:

```
make test
```

Check out logs of Travis CI or AppVeyor if tests were failed on creating PR, there you can find useful information.

4.4.6 Type checking

After adding every feature you should run the type checking and make sure that everything is okay. You can do it using make:

```
make type-check
```

4.4.7 Code Review

Contributions will not be merged until they've been code reviewed by one of our reviewers. In the event that you object to the code review feedback, you should make your case clearly and calmly. If, after doing so, the feedback is judged to still apply, you must either apply the feedback or withdraw your contribution.

4.4.8 Questions

The GitHub issue tracker is for bug reports and feature requests. Please do not create issue which does not related to features or bug reports.

4.4.9 New Locale

We have created a directory with a real structure which you can use as great example `mimesis/data/locale_template` if you want to add a new locale.

4.4.10 Releases

We use **Travis CI** for automatically creating releases. The package will be published on PyPi after pushing the new **tag** to the master branch. The new release can be approved or disapproved by maintainers of this project. If the new release was disapproved, then maintainer should justify why the new release cannot be created.

4.4.11 Summary

- Add one change per one commit.
- Always comment your code (only in English!).
- Check your spelling and grammar.
- Run the tests after each commit.
- Make sure the tests pass.
- Make sure that type check is passed.
- If you add any functionality, then you should add tests for it.
- Annotate your code.
- Do not write bad code!

CHAPTER

FIVE

CHANGELOG

Here you can see the full list of changes between each Mimesis release.

5.1 Version 3.4.0

Note: This version is still under development.

Added:

- Added an alias `first_name()` for `Person().name()`

Fixed:

- Fixed issue with invalid email addresses on using custom domains without `@` for `Person().email()`

5.2 Version 3.3.0

Fixed:

- `country()` from the `Address()` provider now by default returns the country name of the current locale.
- Separated Europe and Asia continents in Italian locale.

Removed:

- Removed duplicated names in the countries of `et` locale.

5.3 Version 3.2.0

Added:

- Added built-in provider DenmarkSpecProvider
- Added meta classes for providers for internal usage (see #621.)
- Added support for custom templates in `Person().username()`
- Added `ItalianSpecProvider()`

Fixed:

- Support of seed for custom providers

- `currency_iso_code` from the `Business()` provider now by default returns the currency code of the current locale.

Removed:

- Removed `multiple_choice()` in the `random` module because it was unused and it could be replaced with `random.choices`.
- Removed legacy method `child_count()` from provider `Person()`

5.4 Version 3.1.0

Fixed:

- Fixed `UnsupportedField` on using field `choice`, #619

5.5 Version 3.0.0

> **Warning:** This release (3.0.0) contains some breaking changes in API

> **Warning:** In this release (3.0.0) we've reject support of Python 3.5

Added:

- Added provider `Choice()`
- Added method `formatted_time()` for `Datetime()` provider
- Added method `formatted_date()` for `Datetime()` provider
- Added method `formatted_datetime()` for `Datetime()` provider
- Added support of timezones (optional) for `Datetime().datetime()`
- Added method to bulk create datetime objects: `Datetime().bulk_create_datetimes()`
- Added `kpp` for `RussiaSpecProvider`
- Added `PolandSpecProvider` builtin data provider
- Added context manager to temporarily overriding locale - `BaseDataProvider.override_locale()`
- Added method `token_urlsafe()` for `Cryptographic` provider
- Added 6k+ username words

Updated:

- Updated documentation
- Updated data for `pl` and `fr`
- Updated SNILS algorithm for `RussiaSpecProvider`
- Updated method `Datetime().time()` to return only `datetime.time` object
- Updated method `Datetime().date()` to return only `datetime.date` object

- Completely annotated all functions
- Locale independent providers inherit `BaseProvider` instead of `BaseDataProvider` (it's mean that locale independent providers does not support parameter `locale` anymore)
- Now you can add to Generic only providers which are subclasses of `BaseProvider` to ensure a single instance of `random.Random()` for all providers

Renamed:

- Renamed provider `ClothingSizes` to `Clothing`, so now it can contain any data related to clothing, not sizes only
- Renamed `Science().dna()` to `Science().dna_sequence()`
- Renamed `Science().rna()` to `Science().rna_sequence()`
- Renamed module `helpers.py` to `random.py`
- Renamed module `config.py` to `locales.py`
- Renamed module `utils.py` to `shortcuts.py`
- Renamed `Cryptographic().bytes()` to `Cryptographic.token_bytes()`
- Renamed `Cryptographic().token()` to `Cryptographic.token_hex()`

Removed:

- Removed deprecated argument `fmt` for `Datetime().date()`, use `Datetime().formatted_date()` instead
- Removed deprecated argument `fmt` for `Datetime().time()`, use `Datetime().formatted_time()` instead
- Removed deprecated argument `humanize` for `Datetime().datetime()`, use `Datetime().formatted_datetime()` instead
- Removed deprecated method `Science.scientific_article()`
- Removed deprecated providers `Games`
- Removed deprecated method `Structure().json()`, use `schema.Schema()` and `schema.Field` instead
- Removed deprecated and useless method: `Development().backend()`
- Removed deprecated and useless method: `Development().frontend()`
- Removed deprecated and useless method: `Development().version_control_system()`
- Removed deprecated and useless method: `Development().container()`
- Removed deprecated and useless method: `Development().database()`
- Removed deprecated method `Internet().category_of_website()`
- Removed duplicated method `Internet().image_by_keyword()`, use `Internet().stock_image()` with `keywords` instead
- Removed deprecated JapanSpecProvider (it didn't fit the definition of the data provider)
- Removed deprecated method `Internet().subreddit()`
- Removed `Cryptographic().salt()` use `Cryptographic().token_hex()` or `Cryptographic().token_bytes()` instead

- Removed methods `Person.favorite_movie()`, `Person.favorite_music_genre()`, `Person.level_of_english()` because they did not related to `Person` provider

Fixed:

- Fixed bug with seed
- Fixed issue with names on downloading images
- Fixed issue with `None` in username for `Person().username()`
- Other minor improvements and fix

5.6 Version 2.1.0

Added:

- Added a list of all supported locales as `mimesis/locales.py`

Updated:

- Changed how `Internet` provider works with `stock_image`
- Changed how `random` module works, now exposing global `Random` instance
- Updated dependencies
- Updated `choice` to make it a provider with more output types

Fixed:

- Prevents `ROMANIZED_DICT` from mutating
- Fixed `appveyour` builds
- Fixed `flake8-builtins` checks
- Fixed some `mypy` issues with strict mode
- Fixed number of elements returned by `choice` with `unique=True`

5.7 Version 2.0.1

Removed:

- Removed internal function `utils.locale_info` which duplicate `utils.setup_locale`

5.8 Version 2.0.0

Note: This release (2.0.0) contains some breaking changes and this means that you should update names of classes and methods in your code.

Added:

- Added items `IOC` and `FIFA` for enum object `CountryCode`
- Added support of custom providers for `schema.Field`

- Added support of parameter `dms` for `coordinates, longitude, latitude`
- Added method `Text.rgb_color`
- Added support of parameter `safe` for method `Text.hex_color`
- Added an alias `zip_code` for `Address.postal_code`

Optimizations:

- Significantly improved performance of `schema.Field`
- Other minor improvements

Updated/Renamed:

- Updated method `integers`
- Renamed provider `Personal` to `Person`
- Renamed provider `Structured` to `Structure`
- Renamed provider `ClothingSizes` to `Clothing`
- Renamed json file `personal.json` to `person.json` for all locales
- Renamed `country_iso_code` to `country_code` in `Address` data provider

5.9 Version 1.0.5

Added:

- Added method `RussiaSpecProvider.inn`

Fixed:

- Fixed issue with seed for `providers.Cryptographic.bytes`
- Fixed issue #375

Optimizations:

- Optimized method `Text.hex_color`
- Optimized method `Address.coordinates`
- Optimized method `Internet.ip_v6`

Tests:

- Grouped tests in classes
- Added tests for seeded data providers
- Other minor optimizations and improvements

5.10 Version 1.0.4

Added:

- Added function for multiple choice `helpers.Random.multiple_choice`

Fixed:

- Fixed issue with `seed` #325

Optimizations:

- Optimized method `username()`

5.11 Version 1.0.3

Mover/Removed:

- Moved `custom_code` to `helpers.Random`

Optimizations:

- Optimized function `custom_code` and it works faster by 50%
- Other minor optimizations in data providers

5.12 Version 1.0.2

Added:

- Added method `ethereum_address` for `Payment`
- Added method `get_current_locale` for `BaseProvider`
- Added method `boolean` for `Development` which returns random boolean value
- Added method `integers` for `Numbers`
- Added new built in specific provider `UkraineSpecProvider`
- Added support of `key functions` for the object `schema.Field`
- Added object `schema.Schema` which helps generate data by schema

Fixed:

- Fixed issue `full_name` when method return female surname for male name and vice versa
- Fixed bug with improper handling of attributes that begin with an underscore for class `schema.Field`

Updated:

- Updated method `version` for supporting pre-releases and calendar versioning
- Renamed methods `international`, `european` and `custom` to `international_size`, `european_size` and `custom_size`

5.13 Version 1.0.1

Updated:

- Fixed #304

5.14 Version 1.0.0

This is a first major version of `mimesis` and here are **breaking changes** (including changes related to support for only the latest versions of `Python`, i.e `Python 3.5` and `Python 3.6`), so there is no backwards compatibility with early versions of this library.

Added:

- Added `Field` for generating data by schema
- Added new module `typing.py` for custom types
- Added new module `enums.py` and support of enums in arguments of methods
- Added `category_of_website` and `port` to `Internet` data provider
- Added `mnemonic_phrase` for `Cryptography` data provider
- Added `price_in_btc` and `currency_symbol` to `Business` data provider
- Added `dna`, `rna` and `atomic_number` to `Science` data provider
- Added `vehicle_registration_code` to `Transport` data provider
- Added `schoice` method for `Random`
- Added alias `last_name` for `surname` in `Personal` data provider
- Added alias `province`, `region`, `federal_subject` for `state` in `Address` data provider
- Added annotations for all methods and functions for supporting type hints
- Added new data provider `Payment`
- Added new methods to `Payment`: `credit_card_network`, `credit_card_owner`

Fixed:

- Fixed issue with `primes` in `Numbers` data provider
- Fixed issue with repeated output on using `Code().custom code`
- Other minor fix and improvements

Mover/Removed:

- Moved `credit_card`, `credit_card_expiration_date`, `cid`, `cvv`, `paypal` and `bitcoin` to `Payment` from `Personal`
- Moved `custom_code` to `utils.py` from `providers.code.Code`
- Removed some useless methods
- Removed module `constants`, in view of adding more convenient and useful module `enums`
- Removed non informative custom exception `WrongArgument` and replaced one with `KeyError` and `NonEnumerableError`
- Parameter `category` of method `hashtags` is deprecated and was removed
- Removed all methods from `UnitSystem` and replaced ones with `unit()`.

Updated/Renamed:

- Updated data for `de-at`, `en`, `fr`, `pl`, `pt-br`, `pt`, `ru`, `uk`
- Other minor updates in other languages
- Renamed `currency_iso` to `currency_iso_code in Business` data provider

CHAPTER

SIX

INDICES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z