Mimesis Documentation

Release 5.1.1

Isaak Uchakaev

Jan 06, 2022

CONTENTS

Mimesis is a high-performance fake data generator for Python, which provides data for a variety of purposes in a variety of languages.

The fake data could be used to populate a testing database, create fake API endpoints, create JSON and XML files of arbitrary structure, anonymize data taken from production and etc.

Mimesis is Open Source and licensed under the MIT License.

CHAPTER

ONE

CONTENTS

1.1 About Mimesis

1.1.1 What Mimesis is?

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.

1.1.2 What Mimesis is Not?

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.

1.1.3 What is the fake data?

Fake data is a type of data that does not contain any useful or sensitive data, but serves to reserve space where real data is nominally present. Fake data can be used as a placeholder for both testing and operational purposes. For testing, dummy data can also be used as stubs.

1.1.4 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.

1.1.5 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.

1.1.6 Features

The key features are:

- **Easy**: Designed to be easy to use and learn.
- **Multilingual**: Supports data for a lot of languages.
- **Performance**: The fastest data generator available for Python.
- **Data variety**: Supports a lot of data providers for a variety of purposes.
- **Country-specific data providers**: Provides data specific only for some countries.
- **Extensibility**: You can create your own data providers and use them with Mimesis.
- **Generic data provider**: The simplified access to all the providers from a single object.
- **Zero hard dependencies**: Does not require any modules other than the Python standard library.
- **Schema-based generators**: Provides an easy mechanism to generate data by the schema of any complexity.

1.1.7 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.

1.1.8 Performance

Below you can see the result of performance comparison of Mimesis and Faker:

Generating 10k full names

Library	Method name	Iterations	Uniqueness	Runtime (in seconds)
Mimesis	*full_name()*	10 000	9988 (99.88%)	0.137
Faker	**Faker.name()**	10 000	9363 (93.63%)	1.758

Generating 100k full names

Library	Method name	Iterations	Uniqueness	Runtime (in seconds)
Mimesis	*full_name()*	100 000	98 265 (98.27%)	1.344
Faker	**Faker.name()**	100 000	71 067 (71.07%)	17.375

Generating 1 million full names

Library	Method name	Iterations	Uniqueness	Runtime (in seconds)
Mimesis	*full_name()*	1 000 000	847 645 (84.76%)	13.685
Faker	**Faker.name()**	1 000 000	330 166 (33.02%)	185.945

1.2 Getting Started

1.2.1 Installation

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

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

Use the following command to install Mimesis in Jupyter Notebook:

```
(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
```

1.2.2 Basic Usage

A minimal basic usage example looks something like this:

```
from mimesis import Person
from mimesis.locales import Locale
from mimesis.enums import Gender
person = Person(Locale.EN)

person.full_name(gender=Gender.FEMALE)
# Output: 'Antonetta Garrison'

person.full_name(gender=Gender.MALE)
# Output: '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 the object `Locale` which provides locale codes (its own attributes) and must be used as a parameter for locale-depend data providers.
3. We import object *Gender* which we are used as a parameter for the *full_name()*.
4. Next we generate random female full name.
5. The same as above, but for male.

1.2.3 Locale

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
from mimesis.locales import Locale

de = Address(locale=Locale.DE)
ru = Address(locale=Locale.RU)

de.region()
# Output: 'Brandenburg'

ru.federal_subject()
# Output: ' '

de.address()
# Output: 'Mainzer Landstraße 912'

>>> ru.address()
# Output: '.  125'
```

See the table below for more details.

Supported locales

Mimesis currently includes support for 34 different locales:

Code	Associated attribute	Name	Native Name
cs	`Locale.CS`	Czech	Česky
da	`Locale.DA`	Danish	Dansk
de	`Locale.DE`	German	Deutsch
de-at	`Locale.DE_AT`	Austrian german	Deutsch
de-ch	`Locale.DE_CH`	Swiss german	Deutsch
el	`Locale.EL`	Greek	
en	`Locale.EN`	English	English
en-au	`Locale.EN_AU`	Australian English	English
en-ca	`LocALE.EN_CA`	Canadian English	English
en-gb	`Locale.EN_GB`	British English	English

continues on next page

Table 1 – continued from previous page

Code	Associated attribute	Name	Native Name
es	`Locale.ES`	Spanish	Español
es-mx	`Locale.ES_MX`	Mexican Spanish	Español
et	`Locale.ET`	Estonian	Eesti
fa	`Locale.FA`	Farsi	
fi	`Locale.FI`	Finnish	Suomi
fr	`Locale.FR`	French	Français
hu	`Locale.HU`	Hungarian	Magyar
is	`Locale.IS`	Icelandic	Íslenska
it	`Locale.IT`	Italian	Italiano
ja	`Locale.JA`	Japanese	
kk	`Locale.KK`	Kazakh	
ko	`Locale.KO`	Korean	
nl	`Locale.NL`	Dutch	Nederlands
nl-be	`Locale.NL_BE`	Belgium Dutch	Nederlands
no	`Locale.NO`	Norwegian	Norsk
pl	`Locale.PL`	Polish	Polski
pt	`Locale.PT`	Portuguese	Português
pt-br	`Locale.PT_BR`	Brazilian Portuguese	Português Brasileiro
ru	`Locale.RU`	Russian	
sk	`Locale.SK`	Slovak	Slovensky
sv	`Locale.SV`	Swedish	Svenska
tr	`Locale.TR`	Turkish	Türkçe
uk	`Locale.UK`	Ukrainian	
zh	`Locale.ZH`	Chinese	

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.locales import Locale

person = Person(locale=Locale.EN)
person.full_name()
# Output: 'Ozie Melton'

with person.override_locale(Locale.RU):
    person.full_name()

# Output: ' '

person.full_name()
# Output: 'Waldo Foster'
```

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

```
from mimesis import Generic
from mimesis.locales import Locale
```

(continues on next page)

(continued from previous page)

```
generic = Generic(locale=Locale.EN)
generic.text.word()
# Output: 'anyone'

with generic.text.override_locale(Locale.FR):
    generic.text.word()

# Output: 'mieux'

generic.text.word()
# Output: 'responsibilities'
```

1.2.4 Data 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.

Warning: Data providers are **heavy objects** since each instance of provider keeps in memory all the data from the provider's JSON file so you **should not** construct too many providers.

You can read more about the heaviness of providers in this issue.

1.2.5 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
from mimesis.locales import Locale
g = Generic(locale=Locale.ES)

g.datetime.month()
# Output: 'Agosto'

g.code.imei()
# Output: '353918052107063'

g.food.fruit()
# Output: 'Limón'
```

1.2.6 Seeded Data

Note: Keep in mind that some methods of some providers cannot be used with seeded providers since their crypto secure nature.

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

```
from mimesis import Person
from mimesis.locales import Locale

person = Person(locale=Locale.TR, seed=0xFF)
person.full_name()
# Output: 'Gizem Tekand'
```

1.2.7 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
from mimesis.locales import Locale
person = Person(locale=Locale.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.locales import Locale
from mimesis.builtins import BrazilSpecProvider

generic = Generic(locale=Locale.PT_BR)
generic.add_provider(BrazilSpecProvider)
generic.brazil_provider.cpf()
# Output: '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()
# Output: '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()
# Output: '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)
# Output: ''

ru.patronymic(gender=Gender.MALE)
# Output: ''
```

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

1.2.8 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 import Generic
from mimesis.locales import Locale
from mimesis.providers.base import BaseProvider


class SomeProvider(BaseProvider):
    class Meta:
        name = "some_provider"

    @staticmethod
    def hello() -> str:
        return "Hello!"


class Another(BaseProvider):
    def __init__(self, seed, message: str) -> None:
        super().__init__(seed=seed)
        self.message = message

    def bye(self) -> str:
        return self.message
```

(continues on next page)

(continued from previous page)

```
generic = Generic(locale=Locale.DEFAULT)
generic.add_provider(SomeProvider)
generic.add_provider(Another, message="Bye!")

generic.some_provider.hello()
# Output: 'Hello!'

generic.another.bye()
# Output: 'Bye!'
```

You can also add multiple providers:

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

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

```
class InvalidProvider:
    @staticmethod
    def hello() -> str:
        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.

1.2.9 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.enums import Gender
from mimesis.locales import Locale
```

(continues on next page)

(continued from previous page)

```
from mimesis.schema import Field, Schema

_ = Field(locale=Locale.EN)
schema = Schema(schema=lambda: {
    "pk": _("increment"),
    "uid": _("uuid"),
    "name": _("text.word"),
    "version": _("version", pre_release=True),
    "timestamp": _("timestamp", posix=False),
    "owner": {
        "email": _("person.email", domains=["test.com"], key=str.lower),
        "token": _("token_hex"),
        "creator": _("full_name", gender=Gender.FEMALE),
    },
})
schema.create(iterations=3)
```

Output:


```
[
  {
    "pk": 1,
    "uid": "c1b2fda1-762b-4c0b-aef7-e995e19758b6",
    "name": "lesbian",
    "version": "3.0.6-alpha.9",
    "timestamp": "2016-12-07T13:26:54Z",
    "owner": {
      "email": "tewing1841@test.com",
      "token": "09960ce907dee56a3c4a6730b7e1ff6ad9620b878c68ff978bfe296da09c1b4b",
      "creator": "Travis Burton"
    }
  },
  {
    "pk": 2,
    "uid": "b0f33a7e-0e3e-4bf0-92df-3ba869add555",
    "name": "disney",
    "version": "2.6.0-alpha.11",
    "timestamp": "2017-02-11T10:45:27Z",
    "owner": {
      "email": "cyprus1904@test.com",
      "token": "a087fadffce394141d3e93c895e4da6db906a60fd0886bad909dc179861b4650",
      "creator": "Dot Anderson"
    }
  },
  {
    "pk": 3,
    "uid": "19b782f0-abd3-468c-9fe2-a82d47212d0c",
    "name": "mar",
    "version": "4.7.0-beta.4",
    "timestamp": "2003-08-22T08:22:24Z",
    "owner": {
      "email": "artiller1822@test.com",
```


(continues on next page)

(continued from previous page)

```
      "token": "d35edc15e74c101e3c2fb6a9b8b74bf40ed21d45b984cc5516105f3853e375e9",
      "creator": "Enda Martinez"
    }
  }
]
```

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.locales import Locale
from mimesis import builtins

custom_providers = (
     builtins.RussiaSpecProvider,
     builtins.NetherlandsSpecProvider,
)
_ = Field(Locale.EN, providers=custom_providers)

_('snils')
# Output: '239-315-742-84'

_('bsn')
# Output: '657340522'
```

1.3 Tips and Tricks

1.3.1 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
from mimesis.locales import Locale

person = Person(Locale.RU)
datetime = Datetime(Locale.RU)
text = Text(Locale.RU)
code = Code(Locale.RU)
```

Correct:

```
from mimesis import Generic
from mimesis.locales import Locale
generic = Generic(locale=Locale.EN)

generic.person.username()
# Output: 'sherley3354'
```

(continues on next page)

(continued from previous page)

```
generic.datetime.date()
# Output: '14-05-2007'
```

Still correct:

```
from mimesis import Person
from mimesis.locales import Locale

p_en = Person(Locale.EN)
p_sv = Person(Locale.SV)
```

Also correct:

```
from mimesis import Person

person = Person(Locale.EN)
with person.override_locale(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()*.

1.3.2 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. Below you can see an abstract example of a Django command which uses Mimesis to bootstrap database.

Good:

```
~ python manage.py fill_fake_db --count=1000 --locale=de
```

Very bad:

```
~ python manage.py fill_fake_db --count=600000 --locale=de
```

1.3.3 Romanization of Cyrillic data

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

Example of usage for romanization of Russian full name:

```
from mimesis.shortcuts import romanize
from mimesis.locales import Locale

romanize(" ", locale=Locale.RU)
# Output: 'Veronika Denisova'
```

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

1.3.4 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.locales import Locale
from mimesis.enums import Gender

_ = Field(Locale.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.

1.3.5 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=1)
    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=1)
        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"
    }
  },
]
```


1.3.6 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=1)
    return jsonify(dummy_data)
```

Response:


```
[
  {
    "id": "f2b326e3-4ce7-1ae9-9e6d-34a28fb70106",
    "name": "Johnny",
    "surname": "Waller",
    "email": "vault1907@live.com",
    "age": 47,
    "username": "CaterpillarsSummational_1995",
    "occupation": "Scrap Dealer",
    "address": {
      "street": "Tonquin",
      "city": "Little Elm",
      "zipcode": "30328"
    }
  },
]
```

1.3.7 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

TWO

API REFERENCE

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

2.1 API Reference

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

2.1.1 Random

Implements various helpers which are used in the various data providers.

This module contains custom `Random()` class where implemented a lot of methods which are not included in standard `random.Random`, but frequently used in this project.

class mimesis.random.**Random**(*x=None*)

Custom class for the possibility of extending.

The class is a subclass of the class `random.Random` from the module random of the standard library, which provides the custom methods.

custom_code(*mask='@###'*, *char='@'*, *digit='#'*)

Generate custom code using ascii uppercase and random integers.

Parameters

- **mask** (`str`) – Mask of code.
- **char** (`str`) – Placeholder for characters.
- **digit** (`str`) – Placeholder for digits.

Return type `str`

Returns Custom code.

generate_string(*str_seq*, *length=10*)

Generate random string created from string sequence.

Parameters

- **str_seq** (`str`) – String sequence of letters or digits.
- **length** (`int`) – Max value.

Return type `str`

Returns Single string.

randints(*amount=3*, *a=1*, *b=100*)
Generate list of random integers.

Parameters

- **amount** (int) – Amount of elements.
- **a** (int) – Minimum value of range.
- **b** (int) – Maximum value of range.

Return type List[int]

Returns List of random integers.

Raises **ValueError** – if amount less or equal to zero.

randstr(*unique=False*, *length=None*)
Generate random string value.

This method can be especially useful when you need to generate only unique values in your provider. Just pass parameter unique=True.

Basically, this method is just a simple wrapper around uuid.UUID.

Parameters

- **unique** (bool) – Generate only unique values.
- **length** (Optional[int]) – Length of string. Default range is [a, b].

Return type str

Returns Random string.

uniform(*a*, *b*, *precision=15*)
Get a random number in the range [a, b) or [a, b] depending on rounding.

Parameters

- **a** (float) – Minimum value.
- **b** (float) – Maximum value.
- **precision** (int) – Round a number to a given precision in decimal digits, default is 15.

Return type float

static urandom(*size=8*)
Return a bytes object containing random bytes.

Parameters **size** (int) – The size of byte object.

Return type bytes

Returns Bytes.

mimesis.random.**get_random_item**(*enum*, *rnd=None*)
Get random item of enum object.

Parameters

- **enum** (Any) – Enum object.
- **rnd** (Optional[*Random*]) – Custom random object.

Return type Any

Returns Random item of enum.

2.2 Builtin Data Providers

2.2.1 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

2.2.2 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 0405420694

2.2.3 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

2.2.4 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.

2.2.5 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.

2.2.6 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.

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.

2.2.7 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

2.3 Shortcuts

This module is provide internal util functions.

mimesis.shortcuts.**luhn_checksum**(*num*)
Calculate a checksum for num using the Luhn algorithm.

Parameters **num** (str) – The number to calculate a checksum for as a string.

Return type str

Returns Checksum for number.

mimesis.shortcuts.**romanize**(*string*, *locale*)
Romanize a given string.

Supported locales are: Locale.RU Locale.UK Locale.KK

Parameters

- **string** (str) – Cyrillic string.
- **locale** (Union[*Locale*, str]) – Locale.

Return type str

Returns Romanized string.

2.4 Custom Exceptions

Custom exceptions which used in Mimesis.

exception mimesis.exceptions.**FieldError**(*name=None*)

exception mimesis.exceptions.**LocaleError**(*locale*)
Raised when a locale isn't supported.

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

exception mimesis.exceptions.**SchemaError**
Raised when schema is unsupported.

2.5 Base Providers

2.5.1 BaseProvider

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

__init__(*, *seed=None*, ***kwargs*)
Initialize attributes.

Keep in mind, that locale-independent data providers will work only with keyword-only arguments since version 5.0.0.

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

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

Return type None

validate_enum(*item*, *enum*)
Validate enum parameter of method in subclasses of BaseProvider.

Parameters

- **item** (Any) – Item of enum object.
- **enum** (Any) – Enum object.

Return type Any

Returns Value of item.

Raises *NonEnumerableError* – if item not in enum.

2.5.2 BaseDataProvider

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

__init__(*locale=Locale.EN*, *seed=None*)
Initialize attributes for data providers.

Parameters

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

extract(*keys*, *default=None*)
Extracts nested values from JSON file by list of keys.

Parameters

- **keys** (List[str]) – List of keys (order extremely matters).
- **default** (Optional[Any]) – Default value.

Return type Any

Returns Data.

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*)
Context manager which allows overriding current locale.

Temporarily overrides current locale for locale-dependent providers.

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

Return type Generator[*BaseDataProvider*, None, None]

Returns Provider with overridden locale.

2.6 Generic Providers

2.6.1 Generic

class mimesis.**Generic**(*locale=Locale.EN*, *seed=None*)
Class which contain all providers at one.

class Meta
Class for metadata.

__init__(*locale=Locale.EN*, *seed=None*)
Initialize attributes lazily.

add_provider(*cls*, ***kwargs*)
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

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

Overrides method *BaseProvider.reseed()*.

Parameters **seed** (Union[None, int, float, str, bytes, bytearray]) – Seed for random.

Return type None

2.7 Locale-Dependent Providers

2.7.1 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.

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[str, Union[str, float]]

Returns Dict with coordinates.

country()
Get the country of the current locale.

Allow_random Return a random country name.

Return type str

Returns The Country.

country_code(*code=CountryCode.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 **code** (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.

2.7.2 Finance

class mimesis.**Finance**(**args*, ***kwargs*)
Class for generating finance data.

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.

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.

Return type str

Returns Currency symbol.

price(*minimum=500*, *maximum=1500*)
Generate random price.

Parameters

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

Return type float

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 float

Returns Price in BTC.

stock_exchange()
Returns stock exchange name.

Return type str

Returns Returns exchange name.

stock_name()
Returns stock name.

Return type str

Returns Stock name.

stock_ticker()
Returns random stock ticker.

Return type str

Returns Ticker.

2.7.3 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.

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.timedelta for more details.

Parameters

- **date_start** (datetime) – Begin of the range.
- **date_end** (datetime) – End of the range.
- **kwargs** (Any) – 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=2022*)
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=2022*, *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** (Any) – 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** (Any) – 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** (Any) – Kwargs for *datetime()*.

Return type Union[str, int]

Returns Timestamp.

timezone(*region=None*)
Get a random timezone.

Parameters **region** (Optional[*TimezoneRegion*]) – Timezone region.

Return type str

Returns Timezone.

week_date(*start=2017*, *end=2022*)
Get week number with year.

Parameters

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

Return type str

Returns Week number.

year(*minimum=1990*, *maximum=2022*)
Generate a random year.

Parameters

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

Return type int

Returns Year.

2.7.4 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.

2.7.5 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.

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.

blood_type()

Get a random blood type.

Return type str

Returns Blood type (blood group).

Example A+

email(*domains=None*, *unique=False*)

Generate a random email.

Parameters

- **domains** (Optional[Sequence[str]]) – List of custom domains for emails.
- **unique** (bool) – Makes email addresses unique.

Return type str

Returns Email address.

Raises **ValueError** – if «unique» is True and the provider was seeded.

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.

Return type str

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 meters.

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.

sex(**args*, ***kwargs*)
An alias for method self.gender().

See docstrings of method self.gender() for details.

Parameters

- **args** (Any) – Positional arguments.
- **kwargs** (Any) – Keyword arguments.

Return type Union[str, int]

Returns Sex

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(*mask=None*, *drange=(1800, 2100)*)
Generate username by template.

You can create many different usernames using masks.

- **C** stands for capitalized username.
- **U** stands for uppercase username.
- **l** stands for lowercase username.
- **d** stands for digits in username.

You can also use symbols to separate the different parts of the username: **.** **_** **-**

Parameters

- **mask** (Optional[str]) – Mask.
- **drange** (Tuple[int, int]) – Digits range.

Raises ValueError – If template is not supported.

Return type str

Returns Username as string.

Example:

```
>>> username(mask='C_C_d')
Cotte_Article_1923
>>> username(mask='U.l.d')
ELKINS.wolverine.2013
>>> username(mask='l_l_d', drange=(1900, 2021))
plasmic_blockader_1907
```

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.

2.7.6 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.

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

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

Return type str

Returns DNA sequence.

Example GCTTTAGACC

measure_unit(*name=None*, *symbol=False*)
Get unit name from International System of Units.

Parameters

- **name** (Optional[*MeasureUnit*]) – Enum object UnitName.
- **symbol** (bool) – Return only symbol

Return type str

Returns Unit.

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

Parameters

- **sign** (Optional[*MetricPrefixSign*]) – Sing of prefix (positive/negative).
- **symbol** (bool) – Return the symbol of the prefix.

Return type str

Returns Metric prefix for SI measure units.

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

Example mega

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

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

Return type str

Returns RNA sequence.

Example AGUGACACAA

2.7.7 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[str]

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]

2.8 Locale-Independent Providers

2.8.1 BinaryFile

class mimesis.**BinaryFile**(**args*, ***kwargs*)
Class for generating binary data

class Meta
Class for metadata.

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

Parameters

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

audio(*, *file_type=AudioFile.MP3*)
Generates audio file of given format and returns it as bytes.

Note: This method accepts keyword-only arguments.

Parameters **file_type** (*AudioFile*) – File extension.

Return type bytes

Returns File as a sequence of bytes.

compressed(*, *file_type=CompressedFile.ZIP*)
Generates compressed file of given format and returns it as bytes.

Note: This method accepts keyword-only arguments.

Parameters **file_type** (*CompressedFile*) – File extension.

Return type bytes

Returns File as a sequence of bytes.

document(*, *file_type=DocumentFile.PDF*)
Generates document of given format and returns it as bytes.

Note: This method accepts keyword-only arguments.

Parameters **file_type** (*DocumentFile*) – File extension.

Return type bytes

Returns File as a sequence of bytes.

image(*, *file_type=ImageFile.PNG*)
Generates image of given format and returns it as bytes.

Note: This method accepts keyword-only arguments.

Parameters **file_type** (*ImageFile*) – File extension.

Return type bytes

Returns File as a sequence of bytes.

video(*, *file_type=VideoFile.MP4*)
Generates video file of given format and returns it as bytes.

Note: This method accepts keyword-only arguments.

Parameters **file_type** (*VideoFile*) – File extension.

Return type bytes

Returns File as a sequence of bytes.

2.8.2 Code

class mimesis.**Code**(**args*, ***kwargs*)
A class, which 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 code with needed value of the enum object *EANFormat*.

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

Return type str

Returns EAN.

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

imei()
Generate a random IMEI.

Return type str

Returns IMEI.

isbn(*fmt=None*, *locale=Locale.EN*)
Generate ISBN for current locale.

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

Parameters

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

Return type str

Returns ISBN.

Raises *NonEnumerableError* – if code 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.

2.8.3 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** (Any) – Arguments.
- **kwargs** (Any) – Keyword arguments.

2.8.4 Cryptographic

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

class Meta
Class for metadata.

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

Parameters **seed** – Seed.

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

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

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

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

Return type str

Returns Hash.

Raises *NonEnumerableError* – When algorithm is unsupported.

mnemonic_phrase()
Generate BIP-39-compatible mnemonic phrase.

Return type str

Returns Mnemonic phrase.

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).

Return type str

Returns URL-safe token.

uuid()
Generate UUID4 string.

Return type str

Returns UUID4 as string.

static uuid_object()
Generate UUID4 object.

Return type UUID

Returns UUID4 object.

2.8.5 Development

class mimesis.**Development**(*, *seed=None*, ***kwargs*)
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

2.8.6 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** (Any) – Arguments.
- **kwargs** (Any) – 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

2.8.7 Hardware

class mimesis.**Hardware**(*, *seed=None*, ***kwargs*)
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.

2.8.8 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** (Any) – Arguments.
- **kwargs** (Any) – 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 List[str]

Returns The list of hashtags.

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

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

hostname(*tld_type=None*, *subdomains=None*)
Generate a random hostname without scheme.

Parameters

- **tld_type** (Optional[*TLDType*]) – TLDType.
- **subdomains** (Optional[List[str]]) – List of subdomains (make sure they are valid).

Return type str

Returns Hostname.

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()
Generate a random IPv4 address as string.

Example 19.121.223.58

Return type str

ip_v4_object()
Generate random ipaddress.IPv4Address object.

Return type IPv4Address

Returns ipaddress.IPv4Address object.

ip_v4_with_port(*port_range=PortRange.ALL*)
Generate a random IPv4 address as string.

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

Return type str

Returns IPv4 address as string.

Example 19.121.223.58:8000

`ip_v6()`
Generate a random IPv6 address as string.

Return type `str`

Returns IPv6 address string.

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

`ip_v6_object()`
Generate random `ipaddress.IPv6Address` object.

Return type `IPv6Address`

Returns `ipaddress.IPv6Address` object.

`mac_address()`
Generate a random MAC address.

Return type `str`

Returns Random MAC address.

Example 00:16:3e:25:e7:f1

`port(port_range=PortRange.ALL)`
Generate random port.

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

Return type `int`

Returns Port number.

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

Example 8080

`query_parameters(length=None)`
Generate arbitrary query parameters as a dict.

Parameters **length** (`Optional[int]`) – Length of dictionary (key/value pair).

Return type `Dict[str, str]`

Returns Dict of query parameters.

`query_string(length=None)`
Generate arbitrary query string of given length.

Parameters **length** (`Optional[int]`) – Length of query string.

Return type `str`

Returns Query string.

`slug(parts_count=None)`
Generate a random slug of given parts count.

Parameters **parts_count** (`Optional[int]`) – Slug's parts count.

Return type `str`

Returns Slug.

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

See «Random search term» on https://source.unsplash.com/ for more details.

Note: This method required an active HTTP connection if you want to get a writable object.

Parameters

- **width** (Union[int, str]) – Width of the image.
- **height** (Union[int, str]) – Height of the image.
- **keywords** (Union[List[str], Set[str], Tuple[str, ...], None]) – List of search keywords.
- **writable** (bool) – Return image as sequence ob bytes.

Return type Union[str, bytes]

Returns Link to the image.

tld(**args*, ***kwargs*)
Generates random top level domain.

An alias for *top_level_domain()*

Return type str

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

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

Return type str

Returns Top level domain.

Raises ***NonEnumerableError*** – if tld_type not in *enums.TLDType*.

uri(*scheme=URLScheme.HTTPS*, *port_range=None*, *tld_type=None*, *subdomains=None*, *query_params_count=None*)
Generate a random URI.

Parameters

- **scheme** (Optional[*URLScheme*]) – Scheme.
- **port_range** (Optional[*PortRange*]) – PortRange enum object.
- **tld_type** (Optional[*TLDType*]) – TLDType.
- **subdomains** (Optional[List[str]]) – List of subdomains (make sure they are valid).
- **query_params_count** (Optional[int]) – Query params.

Return type str

Returns URI.

url(*scheme=URLScheme.HTTPS*, *port_range=None*, *tld_type=None*, *subdomains=None*)
Generate random URL.

Parameters

- **scheme** (Optional[*URLScheme*]) – Scheme.
- **port_range** (Optional[*PortRange*]) – PortRange enum object.
- **tld_type** (Optional[*TLDType*]) – TLDType.
- **subdomains** (Optional[List[str]]) – List of subdomains (make sure they are valid).

Return type str

Returns URL.

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

2.8.9 Numeric

class mimesis.**Numeric**(**args*, ***kwargs*)
A provider for generating numeric data.

class Meta
Class for metadata.

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

Keep in mind, that locale-independent data providers will work only with keyword-only arguments since version 5.0.0.

Parameters **seed** – Seed for random. When set to *None* the current system time is used.

complex_number(*start_real=0.0*, *end_real=1.0*, *start_imag=0.0*, *end_imag=1.0*, *precision_real=15*, *precision_imag=15*)
Generate a random complex number.

Parameters

- **start_real** (float) – Start real range.
- **end_real** (float) – End real range.
- **start_imag** (float) – Start imaginary range.
- **end_imag** (float) – End imaginary range.
- **precision_real** (int) – Round a real part of number to a given precision.
- **precision_imag** (int) – Round the imaginary part of number to a given precision.

Return type complex

Returns Complex numbers.

complexes(*start_real=0*, *end_real=1*, *start_imag=0*, *end_imag=1*, *precision_real=15*, *precision_imag=15*, *n=10*)
Generate a list of random complex numbers.

Parameters

- **start_real** (float) – Start real range.

- **end_real** (float) – End real range.
- **start_imag** (float) – Start imaginary range.
- **end_imag** (float) – End imaginary range.
- **precision_real** (int) – Round a real part of number to a given precision.
- **precision_imag** (int) – Round the imaginary part of number to a given precision.
- **n** (int) – Length of the list.

Return type List[complex]

Returns A list of random complex numbers.

decimal_number(*start=- 1000.0*, *end=1000.0*)
Generate random decimal number.

Parameters

- **start** (float) – Start range.
- **end** (float) – End range.

Return type Decimal

Returns decimal.Decimal object.

decimals(*start=0.0*, *end=1000.0*, *n=10*)
Generate a decimal number as decimal.Decimal objects.

Parameters

- **start** (float) – Start range.
- **end** (float) – End range.
- **n** (int) – Length of the list.

Return type List[Decimal]

Returns A list of random decimal numbers.

float_number(*start=- 1000.0*, *end=1000.0*, *precision=15*)
Generate random float number in range [start, end].

Parameters

- **start** (float) – Start range.
- **end** (float) – End range.
- **precision** (int) – Round a number to a given precision in decimal digits, default is 15.

Return type float

Returns Float.

floats(*start=0*, *end=1*, *n=10*, *precision=15*)
Generate a list of random float numbers.

Parameters

- **start** (float) – Start range.
- **end** (float) – End range.
- **n** (int) – Length of the list.

- **precision** (int) – Round a number to a given precision in decimal digits, default is 15.

Return type List[float]

Returns The list of floating-point numbers.

increment(*accumulator=None*)

Generate incremental number.

Each call of this method returns incremented number.

Example:

```
>>> self.increment()
1
>>> self.increment()
2
>>> self.increment(accumulator="abc")
1
>>> self.increment()
3
>>> self.increment(accumulator="abc")
2
```

Parameters **accumulator** (Optional[str]) – Accumulator name.

Return type int

Returns Integer.

integer_number(*start=- 1000*, *end=1000*)

Generate random integer from start to end.

Parameters

- **start** (int) – Start range.
- **end** (int) – End range.

Return type int

Returns Integer.

integers(*start=0*, *end=10*, *n=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.
- **n** (int) – Length of list.

Return type List[int]

Returns List of integers.

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

matrix(*m=10*, *n=10*, *num_type=NumType.FLOAT*, ***kwargs*)
Generate m x n matrix with random numbers.

This method works with variety of types, so you can pass method-specific ***kwargs*.

See code for more details.

Parameters

- **m** (int) – Number of rows.
- **n** (int) – Number of columns.
- **num_type** (*NumType*) – NumType enum object.
- **kwargs** (Any) – Other method-specific arguments.

Return type Union[List[int], List[float], List[complex], List[Decimal]]

Returns A matrix of random numbers.

2.8.10 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.

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

2.8.11 Payment

class mimesis.**Payment**(**args*, ***kwargs*)
Class that provides data related to payments.

class Meta
Class for metadata.

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

Parameters

- **args** (Any) – Arguments.
- **kwargs** (Any) – Keyword arguments.

bitcoin_address()
Generate a random bitcoin address.

Return type str

Returns Bitcoin address.

Example 3EktnHQD7RiAE6uzMj2ZifT9YgRrkSgzQX

cid()
Generate a random CID.

Return type str

Returns CID code.

Example 7452

credit_card_expiration_date(*minimum=16*, *maximum=25*)
Generate a random expiration date for credit card.

Parameters

- **minimum** (int) – Date of issue.

- **maximum** (int) – Maximum of expiration_date.

Return type str

Returns Expiration date of credit card.

Example 03/19.

credit_card_network()
Generate a random credit card network.

Return type str

Returns Credit card network

Example MasterCard

credit_card_number(*card_type=None*)
Generate a random credit card number.

Parameters **card_type** (Optional[*CardType*]) – Issuing Network. Default is Visa.

Return type str

Returns Credit card number.

Raises **NotImplementedError** – if card_type not supported.

Example 4455 5299 1152 2450

credit_card_owner(*gender=None*)
Generate credit card owner.

Parameters **gender** (*Gender's enum object.*) – Gender of credit card owner.

Return type Dict[str, str]

Returns

cvv()
Generate a random CVV.

Return type str

Returns CVV code.

Example 069

ethereum_address()
Generate a random Ethereum address.

..note: The address will look like Ethereum address, but keep in mind that it is not the valid address.

Return type str

Returns Ethereum address.

Example 0xe8ece9e6ff7dba52d4c07d37418036a89af9698d

paypal()
Generate a random PayPal account.

Return type str

Returns Email of PapPal user.

Example wolf235@gmail.com

2.8.12 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.

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.

manufacturer()
Get a random card manufacturer.

Return type str

Returns A car manufacturer

Example Tesla.

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[*Locale*]) – Registration code for locale (country).

Return type str

Returns Vehicle registration code.

2.9 Schema-based Generators

2.9.1 BaseField

class mimesis.schema.**BaseField**(*locale=Locale.EN*, *seed=None*, *providers=None*)

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

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

See *perform* for more details.

perform(*name=None*, *key=None*, ***kwargs*)

Performs the value of the field by its name.

It takes any string which represents the name of any method of any supported data provider and the `**kwargs` of this method.

Note: Some data providers have methods with the same names and in such cases, you can explicitly define that the method belongs to data-provider `name='provider.name'` otherwise it will return the data from the first provider which has a method `name`.

You can apply a *key function* to the result returned by the method, bt passing a parameter **key** with a callable object which returns the final result.

Parameters

- **name** (Optional[str]) – Name of the method.
- **key** (Optional[Callable[[Any], Any]]) – A key function (or any other callable object) which will be applied to result.
- **kwargs** (Any) – Kwargs of method.

Return type Any

Returns Value which represented by method.

Raises ValueError – if provider not supported or if field not defined.

2.9.2 Field

class mimesis.schema.**Field**(*locale=Locale.EN*, *seed=None*, *providers=None*)

Greedy field.

The field whcih evaluates immediately.

Example:

```
>>> _ = Field()
>>> _('username')
Dogtag_1836
```

2.9.3 Schema

class mimesis.schema.**Schema**(*schema*)
Class which return list of filled schemas.

create(*iterations=1*)
Creates a list of a fulfilled schemas.

Note: This method evaluates immediately, so be careful on creating large datasets otherwise you're risking running out of memory.

If you need a lazy version of this method, see *iterator()*

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

Return type List[Dict[str, Any]]

Returns List of fulfilled schemas.

iterator(*iterations=1*)
Fulfills schema in a lazy way.

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

Return type Iterator[Dict[str, Any]]

Returns List of fulfilled schemas.

loop()
Fulfills a schema **infinitely** in a lazy way.

This method can be useful when you have some dynamic conditions in depend on which the generation must be interrupted.

If you accepting all risks below and want to suppress the warnings then use warnings.catch_warnings

---

Note: Since data *mimesis* provides are limited, frequent calls of this method can cause data duplication.

Note: Before using this method, ask yourself: **Do I really need this**?

In most cases, the answer is: Nah, *iterator()* is enough.

> **Warning:** Do not use this method without interrupt conditions, otherwise, you're risking running out of memory.

> **Warning:** **Never** (seriously) call *list()*, *tuple()* or any other callable which tries to evaluate the whole lazy object on this method — infinite called infinite for a reason.

Return type Iterator[Dict[str, Any]]

Returns An infinite iterator with fulfilled schemas.

2.10 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.

class mimesis.enums.**Algorithm**(*value*)
Provides algorithms which available.

MD5 = 'md5'

SHA1 = 'sha1'

SHA224 = 'sha224'

SHA256 = 'sha256'

SHA384 = 'sha384'

SHA512 = 'sha512'

class mimesis.enums.**AudioFile**(*value*)
An enumeration.

AAC = 'aac'

MP3 = 'mp3'

class mimesis.enums.**CardType**(*value*)
Provides credit card types.

An argument for *credit_card_number()*.

AMERICAN_EXPRESS = 'American Express'

MASTER_CARD = 'MasterCard'

VISA = 'Visa'

class mimesis.enums.**CompressedFile**(*value*)
An enumeration.

GZIP = 'gz'

ZIP = 'zip'

class mimesis.enums.**CountryCode**(*value*)
Provides types of country codes.

An argument for *country_code()*.

A2 = 'a2'

A3 = 'a3'

FIFA = 'fifa'

IOC = 'ioc'

NUMERIC = 'numeric'

class mimesis.enums.**DocumentFile**(*value*)
An enumeration.

DOCX = 'docx'

PDF = 'pdf'

PPTX = 'pptx'

XLSX = 'xlsx'

class mimesis.enums.**EANFormat**(*value*)
Provides formats of EAN.

An argument for *ean()*.

EAN13 = 'ean-13'

EAN8 = 'ean-8'

class mimesis.enums.**FileType**(*value*)
Provides file types.

AUDIO = 'audio'

COMPRESSED = 'compressed'

DATA = 'data'

EXECUTABLE = 'executable'

IMAGE = 'image'

SOURCE = 'source'

TEXT = 'text'

VIDEO = 'video'

class mimesis.enums.**Gender**(*value*)
Represents genders.

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

FEMALE = 'female'

MALE = 'male'

class mimesis.enums.**ISBNFormat**(*value*)
Provides formats of ISBN.

An argument for *isbn()*.

ISBN10 = 'isbn-10'

ISBN13 = 'isbn-13'

class mimesis.enums.**ImageFile**(*value*)
An enumeration.

GIF = 'gif'

JPG = 'jpg'

PNG = 'png'

class mimesis.enums.**Locale**(*value*)
This class provides access to the supported locales from one place.

```
CS = 'cs'
DA = 'da'
DE = 'de'
DEFAULT = 'en'
DE_AT = 'de-at'
DE_CH = 'de-ch'
EL = 'el'
EN = 'en'
EN_AU = 'en-au'
EN_CA = 'en-ca'
EN_GB = 'en-gb'
ES = 'es'
ES_MX = 'es-mx'
ET = 'et'
FA = 'fa'
FI = 'fi'
FR = 'fr'
HU = 'hu'
IS = 'is'
IT = 'it'
JA = 'ja'
KK = 'kk'
KO = 'ko'
NL = 'nl'
NL_BE = 'nl-be'
NO = 'no'
PL = 'pl'
PT = 'pt'
PT_BR = 'pt-br'
RU = 'ru'
SK = 'sk'
SV = 'sv'
TR = 'tr'
UK = 'uk'
ZH = 'zh'
```

classmethod values()

Return type List[str]

class mimesis.enums.**MeasureUnit**(*value*)

Provide unit names.

An argument for *measure_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')

class mimesis.enums.**MetricPrefixSign**(*value*)

Provides prefix signs.

An argument for `metric_prefix()`().

NEGATIVE = 'negative'

POSITIVE = 'positive'

class mimesis.enums.**MimeType**(*value*)

Provides common mime types.

An argument for *mime_type()*.

APPLICATION = 'application'

AUDIO = 'audio'

IMAGE = 'image'

MESSAGE = 'message'

TEXT = 'text'

VIDEO = 'video'

class mimesis.enums.**NumType**(*value*)
Provide number types.

An argument for *matrix()*.

COMPLEX = 'complexes'

DECIMAL = 'decimals'

FLOAT = 'floats'

INTEGER = 'integers'

class mimesis.enums.**PortRange**(*value*)
Represents port ranges.

An argument for *port()*.

ALL = (1, 65535)

EPHEMERAL = (49152, 65535)

REGISTERED = (1024, 49151)

WELL_KNOWN = (1, 1023)

class mimesis.enums.**TLDType**(*value*)
Provides top level domain types.

An argument for *top_level_domain()*.

CCTLD = 'cctld'

GEOTLD = 'geotld'

GTLD = 'gtld'

STLD = 'stld'

UTLD = 'utld'

class mimesis.enums.**TimezoneRegion**(*value*)
Provides regions of timezones.

An argument for *timezone()*.

AFRICA = 'Africa'

AMERICA = 'America'

ANTARCTICA = 'Antarctica'

ARCTIC = 'Arctic'

ASIA = 'Asia'

ATLANTIC = 'Atlantic'

AUSTRALIA = 'Australia'

EUROPE = 'Europe'

INDIAN = 'Indian'

PACIFIC = 'Pacific'

class mimesis.enums.**TitleType**(*value*)
Represents title types.

An argument for *title()*.

ACADEMIC = 'academic'

TYPICAL = 'typical'

class mimesis.enums.**URLScheme**(*value*)
Provides URL schemes.

An argument for some methods of *Internet()*.

FTP = 'ftp'

HTTP = 'http'

HTTPS = 'https'

SFTP = 'sftp'

WS = 'ws'

WSS = 'wss'

class mimesis.enums.**VideoFile**(*value*)
An enumeration.

MOV = 'mov'

MP4 = 'mp4'

2.11 Glossary

locale Locale which represent country-specific data for locale-depend data providers.

See *Locale*

locale-dependent provider A provider which depends in external json files with localized data.

locale-independent provider A provider without external dependencies (can be used for any locale).

provider A class which provides various data generators.

CHAPTER

THREE

ADDITIONAL INFORMATION

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

3.1 License

3.2 Contributors

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

3.2.1 Maintainers

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

3.2.2 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)

3.3 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.

3.4 Contributing Guidelines

The source code and issue tracker are hosted on GitHub. *Mimesis* is tested against Python 3.8 through 3.10 on **GitHub Actions** and **AppVeyor**.

Test coverage is monitored with Codecov.

3.4.1 Dependencies

We use `poetry` to manage dependencies. So, please do not use `virtualenv` or `pip` directly. Before going any further, please, take a moment to read the official documentation about `poetry` to know some basics.

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

```
~  pip install poetry
```

3.4.2 Installing dependencies

Please, note that `poetry` will automatically create a `virtualenv` for this project. It will use you current `python` version. To install all existing dependencies run:

```
poetry install
```

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

```
poetry shell
```

3.4.3 Adding new dependencies

To add a new dependency you can run:

- `poetry add --dev pytest` to install `pytest` as a development dependency

3.4.4 Code Style

Every contributor must follow the PEP8 code style. We're using `black` for formatting code.

3.4.5 Annotating

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

Example of annotated function:

```
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
```

3.4.6 Documenting

Always add docstrings for your modules, classes, methods and functions.

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

3.4.7 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:

```
poetry run pytest
```

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

The tests are randomly shuffled by `pytest-randomly`. To rerun the tests with the previous seed use:

```
) poetry run pytest --randomly-seed=last
```

If you want to specify a seed ahead of time use:

```
) poetry run pytest --randomly-seed=$int
```

3.4.8 Type checking

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

```
poetry run mypy mimesis/ tests/
```

3.4.9 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.

3.4.10 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.

3.4.11 New Locale

Add following files to the directory `mimesis/data/{LOCALE_CODE}/`:

`address.json`:

```
{
  "address_fmt": "{st_num} {st_name} {st_sfx}",
  "city": [
    "Test"
  ],
  "continent": [
    "Test"
  ],
  "country": {
    "current_locale": "Test",
    "name": [
      "Test"
    ]
  },
  "postal_code_fmt": "#####",
  "state": {
    "abbr": [
      "Test"
    ],
    "name": [
      "Test"
    ]
  },
  "street": {
    "name": [
      "Test"
    ],
    "suffix": [
      "Test"
    ]
  }
}
```

`builtin.json`:

```
{
  "any": {
    "structure": [
      "which",
      "you",
      "need"
    ]
  }
}
```

business.json:

```
{
  "company": {
    "name": [
      "Test"
    ],
    "type": {
      "abbr": [
        "Test"
      ],
      "title": [
        "Test"
      ]
    }
  },
  "currency-code": "Test",
  "price-format": "# Test",
  "numeric-decimal": ".",
  "numeric-thousands": ",",
  "numeric-frac-digits": 2
}
```

datetime.json:

```
{
  "day": {
    "abbr": [
      "Test"
    ],
    "name": [
      "Test"
    ]
  },
  "formats": {
    "date": "%m/%d/%Y",
    "time": "%H:%M:%S"
  },
  "month": {
    "abbr": [
      "Test"
    ],
    "name": [
```

(continues on next page)

(continued from previous page)

```
      "Test"
    ]
  },
  "periodicity": [
    "Test"
  ]
}
```

food.json:

```
{
  "dishes": [
    "Test"
  ],
  "drinks": [
    "Test"
  ],
  "fruits": [
    "Test"
  ],
  "spices": [
    "Test"
  ],
  "vegetables": [
    "Test"
  ]
}
```

person.json:

```
{
  "academic_degree": [
    "Test"
  ],
  "gender": [
    "Test"
  ],
  "language": [
    "Test"
  ],
  "names": {
    "female": [
      "Test"
    ],
    "male": [
      "Test"
    ]
  },
  "__COMMENT_NATIONALITY__": "Optional -> nationality: {female: [], male: []}",
  "nationality": [
    "Test"
  ],
```

(continues on next page)

(continued from previous page)

```
  "occupation": [
    "Test"
  ],
  "political_views": [
    "Test"
  ],
  "__COMMENT_SURNAMES__": "Optional -> surnames: {female: [], male: []}",
  "surnames": [
    "Test"
  ],
  "title": {
    "female": {
      "typical": [
        "Test"
      ],
      "academic": [
        "Test"
      ]
    },
    "male": {
      "typical": [
        "Test"
      ],
      "academic": [
        "Test"
      ]
    }
  },
  "university": [
    "Test"
  ],
  "views_on": [
    "Test"
  ],
  "worldview": [
    "Test"
  ],
  "telephone_fmt": [
    "###-###-####",
    "(###) ###-####",
    "1-###-###-####"
  ]
}
```

text.json:

```
{
  "alphabet": {
    "uppercase": [
      "Test"
    ],
    "lowercase": [
```

(continues on next page)

(continued from previous page)

```
      "Test"
    ]
  },
  "answers": [
    "Yes",
    "No",
    "Maybe"
  ],
  "color": [
    "Test"
  ],
  "level": [
    "low",
    "moderate",
    "high",
    "very high",
    "extreme",
    "critical"
  ],
  "quotes": [
    "Test"
  ],
  "text": [
    "Test"
  ],
  "words": {
    "bad": [
      "Test"
    ],
    "normal": [
      "Test"
    ]
  }
}
```

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.

3.4.12 Releases

We use **GitHub Actions** 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.

3.4.13 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

FOUR

CHANGELOG

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

4.1 Version 5.2.1

Removed:

- Removed all params of `mnemonic_phrase()`

4.2 Version 5.1.1

Added:

- Added parameter `region` for `Datetime().timezone()` and enum object `enums.TimezoneRegion`

4.3 Version 5.1.0

Fixed:

- Fix mechanism of reseeding of the internal providers of `Generic` (See #1115).

Removed:

- Removed inappropriate words from `mimesis.data.int.USERNAMES`.

4.4 Version 5.0.0

Warning: This release contains some breaking changes in API.

Python compatibility:

Mimesis 5.0 supports Python 3.8, 3.9, and 3.10.

The Mimesis 4.1.3 is the last to support Python 3.6 and 3.7.

Reworked:

- A method `Person().username()`, now it accepts a parameters `mask` and `drange`.

Renamed:

- Renamed `enums.UnitName` to `enums.MeasureUnit`
- Renamed `enums.PrefixSign` to `enums.MetricPrefixSign`
- Renamed `Business()` to `Finance()`
- Renamed `BaseDataProvider.pull` to `BaseDataProvider._load_datafile`
- Renamed `mimesis.providers.numbers.Numbers` to `mimesis.providers.numeric.Numeric`
- Renamed `fmt` argument of `Address().country_code()` to `code`

Fixed:

- Fix inheritance issues for `Generic`, now it inherits `BaseProvider` instead of `BaseDataProvider`
- Fix locale-independent provider to make them accepts keyword-only arguments
- Fix DenmarkSpecProvider CPR to generate valid CPR numbers.
- Fix `.cvv()` to make it return string
- Fix `.cid()` to make it return string
- Fix `.price()` of `Finance` to make it return float.

Added:

- Added method `hostname()` for `Internet` data provider
- Added support of `**kwargs` for a method `add_provider` of `Generic()` provider
- Added enum `Locale` to `mimesis.enums` and `mimesis.locales`
- Added `measure_unit` and `metric_prefix` methods for the `Science` provider.
- Added `.iterator()` for `schema.Schema`
- Added methods `.slug()` and `ip_v4_with_port()` for `Internet()`
- Added `increment()` method for `Numbers()`
- Added methods `.stock_ticker()`, `.stock_name()` and `.stock_exchange()` for `Finance()`
- Added `BinaryFile` data provider which provides binary data files, such as `.mp3`, `.mp4`, `.png`, etc.

Removed:

- Removed module `decorators`. Use `shortcuts.romanize` to romanize Cyrillic strings.
- Removed `as_object` parameter for `.uuid()`. Now it returns string by default, if you need uuid4 object then use `.uuid_object()`
- Removed invalid names and surnames from `person.json` for `ru` locale
- Removed data provider `UnitSystem()`, use `Science()` instead
- Removed data provider `Structure()`, use `schema.Schema` instead
- Removed builtin provider `GermanySpecProvider`
- Removed data provider `Clothing`, use `Numbers` instead
- Removed method `copyright()` of `Finance()`
- Removed method `network_protocol()` of `Internet()`
- Removed params `with_port` and `port_range` for `ip_v4()` of `Internet()`. Use `ip_v4_with_port()` instead.
- Removed methods `sexual_orientation`, `social_media_profile` and `avatar` of the `Person()` provider.

- Removed a bunch of useless custom exceptions and replaced them with `FieldError`.
- Removed completely useless `chemical_element` and `atomic_number` methods of `Science` data provider and made it locale-independent.

4.5 Version 4.1.3

Added:

- Added `py.typed` file to the package
- Added `Python 3.9` support

4.6 Version 4.1.2

Fix:

- Fixed type hint issue for `schema.Schema` (#928)

4.7 Version 4.1.1

Fix:

- Fixed issue with non-unique uuid

4.8 Version 4.1.0

Added:

- Added method `manufacturer()` for class `Transport()`
- Added `sk` (Slovak) locale support
- Added new parameter `unique` for method `Person().email()`
- Added new parameter `as_object` for method `Cryptographic().uuid()`

Updated:

- Updated parameter `end` for some methods of provider `Datetime()` (Fix #870)
- Updated `.price()` to make it supported locales (Fix #875)

Rename:

- Renamed `decorators.romanized` to `decorators.romanize`
- Renamed `Random.schoice` to `Random.generate_string`
- Renamed `BaseDataProvider.pull` to `BaseDataProvider._pull`

Removed:

- Removed the deprecated `download_image()` function from the `shortcuts` module, use your own custom downloader instead.
- Removed parameter `version` for method `Cryptographic().uuid()`

4.9 Version 4.0.0

> **Warning:** This release (4.0.0) contains some insignificant but breaking changes in API, please be careful.

Added:

- Added an alias `first_name(*args, **kwargs)` for the method `Person().name()`
- Added an alias `sex(*args, **kwargs)` for the method `Person().gender()`
- Added method `randstr()` for class `Random()`
- Added method `complexes()` for the provider `Numbers()`
- Added method `matrix` for the provider `Numbers()`
- Added method `integer_number()` for the provider `Numbers()`
- Added method `float_number()` for the provider `Numbers()`
- Added method `complex_number()` for the provider `Numbers()`
- Added method `decimal_number()` for the provider `Numbers()`
- Added method `ip_v4_object()` and `ip_v6_object` for the provider `Internet()`. Now you can generate IP objects, not just strings.
- Added new parameter `port_range` for method `ip_v4()`
- Added new parameter `separator` for method `Cryptographic().mnemonic_phrase()`

Fixed:

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

Updated:

- Updated names and surnames for locale `ru`
- The `floats()` function in the `Numbers` provider now accepts arguments about the range of the generated float numbers and the rounding used. By default, it generates a list of `n` float numbers instead of a list of 10^n elements.
- The argument `length` of the function `integers` is renamed to `n`.

Removed:

- Removed the `rating()` method from the `Numbers` provider. It can be replaced with `float_number()`.
- Removed the `primes()` method from the `Numbers` provider.
- Removed the `digit()` method from the `Numbers` provider. Use `integer_number()` instead.
- Removed the `between()` method from the `Numbers` provider. Use `integer_number()` instead.
- Removed the `math_formula()` method from the `Science` provider.
- Removed `rounding` argument from `floats()`. Now it's `precision`.

4.10 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.

4.11 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()`

4.12 Version 3.1.0

Fixed:

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

4.13 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

4.14 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`

4.15 Version 2.0.1

Removed:

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

4.16 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

4.17 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

4.18 Version 1.0.4

Added:

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

Fixed:

- Fixed issue with `seed` #325

Optimizations:

- Optimized method `username()`

4.19 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

4.20 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`

4.21 Version 1.0.1

Updated:

- Fixed #304

4.22 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 `generate_string` 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

FIVE

INDICES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z