

---

**fake.py**

***Release 0.6.9***

**Artur Barseghyan <artur.barseghyan@gmail.com>**

**May 14, 2024**



# CONTENTS

<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Prerequisites</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>7</b>
3.1	pip . . . . .	7
3.2	Download and copy . . . . .	7
<b>4</b>	<b>Documentation</b>	<b>9</b>
<b>5</b>	<b>Usage</b>	<b>11</b>
5.1	Generate data . . . . .	11
5.2	Generate files . . . . .	12
5.3	Factories/dynamic fixtures . . . . .	13
5.4	Customize . . . . .	15
<b>6</b>	<b>Tests</b>	<b>17</b>
<b>7</b>	<b>Differences with alternatives</b>	<b>19</b>
<b>8</b>	<b>License</b>	<b>21</b>
<b>9</b>	<b>Support</b>	<b>23</b>
<b>10</b>	<b>Author</b>	<b>25</b>
<b>11</b>	<b>Project documentation</b>	<b>27</b>
11.1	Recipes . . . . .	28
11.2	Creating images . . . . .	40
11.3	Creating PDF . . . . .	41
11.4	Creating DOCX . . . . .	43
11.5	Factories . . . . .	44
11.6	Customization . . . . .	61
11.7	Security Policy . . . . .	65
11.8	Contributor guidelines . . . . .	66
11.9	Contributor Covenant Code of Conduct . . . . .	68
11.10	Release history and notes . . . . .	70
11.11	Package . . . . .	74
11.12	Indices and tables . . . . .	84
	<b>Python Module Index</b>	<b>85</b>



Minimalistic, standalone alternative fake data generator with no dependencies.

`fake.py` is a standalone, portable library designed for generating various random data types for testing.

It offers a simplified, dependency-free alternative for creating random texts, (person) names, URLs, dates, file names, IPs, primitive Python data types (such as *uuid*, *str*, *int*, *float*, *bool*) and byte content for multiple file formats including *PDF*, *DOCX*, *PNG*, *SVG*, *BMP*, and *GIF*.

The package also supports file creation on the filesystem and includes factories (dynamic fixtures) compatible with `Django`, `TortoiseORM`, `Pydantic` and `SQLAlchemy`.



## FEATURES

- Generation of random texts, (person) names, emails, URLs, dates, IPs, and primitive Python data types.
- Support for various file formats (*PDF*, *DOCX*, *TXT*, *PNG*, *SVG*, *BMP*, *GIF*) and file creation on the filesystem.
- Basic factories for integration with [Django](#), [Pydantic](#), [TortoiseORM](#) and [SQLAlchemy](#).





## PREREQUISITES

Python 3.8+



## INSTALLATION

### 3.1 pip

```
pip install fake.py
```

### 3.2 Download and copy

`fake.py` is the sole, self-contained module of the package. It includes tests too. If it's more convenient to you, you could simply download the `fake.py` module and include it in your repository.

Since tests are included, it won't have a negative impact on your test coverage (you might need to apply tweaks to your coverage configuration).



## DOCUMENTATION

- Documentation is available on [Read the Docs](#).
- For various ready to use code examples see the [Recipes](#).
- For tips on how to use the factories see the [Factories](#).
- For customization tips see the [Customization](#).
- For tips on PDF creation see [Creating PDF](#).
- For tips on DOCX creation see [Creating DOCX](#).
- For tips on images creation see [Creating images](#).
- For guidelines on contributing check the [Contributor guidelines](#).
- For various implementation examples, see the [Examples](#).



## 5.1 Generate data

### 5.1.1 Person names

```
from fake import FAKER

FAKER.first_name() # str
FAKER.first_names() # list[str]
FAKER.last_name() # str
FAKER.last_names() # list[str]
FAKER.name() # str
FAKER.names() # list[str]
FAKER.username() # str
FAKER.usernames() # list[str]
```

### 5.1.2 Random texts

```
from fake import FAKER

FAKER.slug() # str
FAKER.slugs() # list[str]
FAKER.word() # str
FAKER.words() # list[str]
FAKER.sentence() # str
FAKER.sentences() # list[str]
FAKER.paragraph() # str
FAKER.paragraphs() # list[str]
FAKER.text() # str
FAKER.texts() # list[str]
```

### 5.1.3 Internet

```
from fake import FAKER

FAKER.email() # str
FAKER.url() # str
FAKER.image_url() # str
FAKER.ipv4() # str
```

### 5.1.4 Filenames

```
from fake import FAKER

FAKER.file_name() # str
```

### 5.1.5 Primitive data types

```
from fake import FAKER

FAKER.pyint() # int
FAKER.pybool() # bool
FAKER.pystr() # str
FAKER.pyfloat() # float
FAKER.uuid() # uuid.UUID
```

### 5.1.6 Dates

```
from fake import FAKER

FAKER.date() # datetime.date
FAKER.date_time() # datetime.datetime
```

## 5.2 Generate files

### 5.2.1 As bytes

```
from fake import FAKER

FAKER.pdf() # bytes
FAKER.docx() # bytes
FAKER.png() # bytes
FAKER.svg() # bytes
FAKER.bmp() # bytes
FAKER.gif() # bytes
```



### 5.2.2 As files on the file system

```
from fake import FAKER

FAKER.pdf_file() # str
FAKER.docx_file() # str
FAKER.png_file() # str
FAKER.svg_file() # str
FAKER.bmp_file() # str
FAKER.gif_file() # str
FAKER.txt_file() # str
```

## 5.3 Factories/dynamic fixtures

This is how you could define factories for Django's built-in Group and User models.

```
from django.contrib.auth.models import Group, User
from fake import (
    DjangoModelFactory,
    FACTORY,
    PostSave,
    PreSave,
    trait,
)

class GroupFactory(DjangoModelFactory):
    """Group factory."""

    name = FACTORY.word()

    class Meta:
        model = Group
        get_or_create = ("name",)

def set_password(user: User, password: str) -> None:
    """Helper function for setting password for the User."""
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    """Helper function for adding the User to a Group."""
    group = GroupFactory(name=name)
    user.groups.add(group)

class UserFactory(DjangoModelFactory):
    """User factory."""

    username = FACTORY.username()
```

(continues on next page)

(continued from previous page)

```

first_name = FACTORY.first_name()
last_name = FACTORY.last_name()
email = FACTORY.email()
date_joined = FACTORY.date_time()
last_login = FACTORY.date_time()
is_superuser = False
is_staff = False
is_active = FACTORY.pybool()
password = PreSave(set_password, password="test1234")
group = PostSave(add_to_group, name="Test group")

class Meta:
    model = User
    get_or_create = ("username",)

@trait
def is_admin_user(self, instance: User) -> None:
    """Trait."""
    instance.is_superuser = True
    instance.is_staff = True
    instance.is_active = True

```

And this is how you could use it:

```

# Create just one user
user = UserFactory()

# Create 5 users
users = UserFactory.create_batch(5)

# Create a user using `is_admin_user` trait
user = UserFactory(is_admin_user=True)

# Create a user with custom password
user = UserFactory(
    password=PreSave(set_password, password="another-password"),
)

# Add a user to another group
user = UserFactory(
    group=PostSave(add_to_group, name="Another group"),
)

# Or even add user to multiple groups at once
user = UserFactory(
    group_1=PostSave(add_to_group, name="Another group"),
    group_2=PostSave(add_to_group, name="Yet another group"),
)

```

## 5.4 Customize

Make your own custom providers and utilize factories with them.

```
import random
import string

from fake import Faker, Factory, provider

class CustomFaker(Faker):

    @provider
    def postal_code(self) -> str:
        number_part = "".join(random.choices(string.digits, k=4))
        letter_part = "".join(random.choices(string.ascii_uppercase, k=2))
        return f"{number_part} {letter_part}"

FAKER = CustomFaker()
FACTORY = Factory(FAKER)
```

Now you can use it as follows (make sure to import your custom instances of FAKER and FACTORY):

```
FAKER.postal_code()

from fake import ModelFactory

class AddressFactory(ModelFactory):

    # ... other definitions
    postal_code = FACTORY.postal_code()
    # ... other definitions

    class Meta:
        model = Address
```



## TESTS

Run the tests with unittest:

```
python -m unittest fake.py
```

Or pytest:

```
pytest
```



## DIFFERENCES WITH ALTERNATIVES

`fake.py` is `Faker` + `factory_boy` + `faker-file` in one package, radically simplified and reduced in features, but without any external dependencies (not even `Pillow` or `dateutil`).

`fake.py` is modeled after the famous `Faker` package. Its' API is highly compatible, although drastically reduced. It's not multilingual and does not support postal codes or that many RAW file formats. However, you could easily include it in your production setup without worrying about yet another dependency.

On the other hand, `fake.py` factories look quite similar to `factory_boy` factories, although again - drastically simplified and reduced in features.

The file generation part of `fake.py` are modelled after the `faker-file`. You don't get a large variety of file types supported and you don't have that much control over the content of the files generated, but you get dependency-free valid files and if that's all you need, you don't need to look further.

However, at any point, if you discover that you "need more", go for `Faker`, `factory_boy` and `faker-file` combination.





**LICENSE**

MIT



## SUPPORT

For security issues contact me at the e-mail given in the *Author* section.

For overall issues, go to [GitHub](#).



---

CHAPTER  
TEN

---

AUTHOR

Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>



## PROJECT DOCUMENTATION

Contents:

### Table of Contents

- *fake.py*
  - *Features*
  - *Prerequisites*
  - *Installation*
    - \* *pip*
    - \* *Download and copy*
  - *Documentation*
  - *Usage*
    - \* *Generate data*
      - *Person names*
      - *Random texts*
      - *Internet*
      - *Filenames*
      - *Primitive data types*
      - *Dates*
    - \* *Generate files*
      - *As bytes*
      - *As files on the file system*
    - \* *Factories/dynamic fixtures*
    - \* *Customize*
  - *Tests*
  - *Differences with alternatives*
  - *License*
  - *Support*

- *Author*
- *Project documentation*

## 11.1 Recipes

### Imports and initialization

```
from fake import FAKER
```

---

#### first\_name

Returns a random first name.

```
from fake import FAKER  
  
FAKER.first_name()
```

---

#### last\_name

Returns a random last name.

```
from fake import FAKER  
  
FAKER.last_name()
```

---

#### name

Returns a random full name.

```
from fake import FAKER  
  
FAKER.name()
```

---

#### word

Returns a random word.

```
from fake import FAKER  
  
FAKER.word()
```

---

#### words

Returns a list of nb random words.



```
from fake import FAKER  
  
FAKER.words()
```

Arguments:

- nb (type: int, default value: 5) is an optional argument.

Example with arguments (returns a list of 10 words):

```
from fake import FAKER  
  
FAKER.words(nb=10)
```

### sentence

Returns a random sentence with nb\_words number of words.

```
from fake import FAKER  
  
FAKER.sentence()
```

Arguments:

- nb\_words (type: int, default value: 5) is an optional argument.

Example with arguments (returns a sentence of 10 words):

```
from fake import FAKER  
  
FAKER.sentence(nb_words=10)
```

### sentences

Returns nb number of random sentences.

```
from fake import FAKER  
  
FAKER.sentences()
```

Arguments:

- nb (type: int, default value: 3) is an optional argument.

Example with arguments (returns a list of 10 sentences):

```
from fake import FAKER  
  
FAKER.sentences(nb=10)
```

### paragraph

Returns a random paragraph with nb\_sentences number of sentences.

```
from fake import FAKER  
  
FAKER.paragraph()
```

Arguments:

- `nb_sentences` (type: `int`, default value: 5) is an optional argument.

Example with arguments (returns a paragraph of 10 sentences):

```
from fake import FAKER  
  
FAKER.paragraph(nb_sentences=10)
```

---

### **paragraphs**

Returns `nb` number of random paragraphs.

```
from fake import FAKER  
  
FAKER.paragraphs()
```

Arguments:

- `nb` (type: `int`, default value: 3) is an optional argument.

Example with arguments (returns a list of 10 paragraphs):

```
from fake import FAKER  
  
FAKER.paragraphs(nb=10)
```

---

### **text**

Returns random text with up to `nb_chars` characters.

```
from fake import FAKER  
  
FAKER.text()
```

Arguments:

- `nb_chars` (type: `int`, default value: 200) is an optional argument.

Example with arguments (returns a 1000 character long text):

```
from fake import FAKER  
  
FAKER.text(nb_chars=1_000)
```

---

### **texts**

Returns `nb` number of random texts.

```
from fake import FAKER  
  
FAKER.texts()
```

Arguments:

- nb (type: int, default value: 3) is an optional argument.

Example with arguments (returns a list of 10 texts):

```
from fake import FAKER  
  
FAKER.texts(nb=10)
```

---

### file\_name

Returns a random file name with the given extension.

```
from fake import FAKER  
  
FAKER.file_name()
```

Arguments:

- extension (type: str, default value: txt) is an optional argument.

Example with arguments (returns a filename with “png” extension):

```
from fake import FAKER  
  
FAKER.file_name(extension="png")
```

---

### email

Returns a random email with the specified domain.

```
from fake import FAKER  
  
FAKER.email()
```

Arguments:

- domain (type: str, default value: example.com) is an optional argument.

Example with arguments (returns an email with “gmail.com” domain):

```
from fake import FAKER  
  
FAKER.email(domain="gmail.com")
```

---

### url

Returns a random URL.

```
from fake import FAKER  
  
FAKER.url()
```

Arguments:

- protocols (type: Optional[Tuple[str]], default value: None) is an optional argument.
- tlds (type: Optional[Tuple[str]], default value: None) is an optional argument.
- suffixes (type: Optional[Tuple[str]], default value: None) is an optional argument.

Returns a valid random image URL.

```
from fake import FAKER  
  
FAKER.image_url()
```

Arguments:

- width (type: int, default value: 800) is a required argument.
- height (type: int, default value: 600) is an required argument.
- service\_url (type: Optional[str], default value: None) is an optional argument.

Example with arguments (alternative dimensions):

```
from fake import FAKER  
  
FAKER.image_url(width=640, height=480)
```

---

### **pyint**

Returns a random integer between min\_value and max\_value.

```
from fake import FAKER  
  
FAKER.pyint()
```

Arguments:

- min\_value (type: int, default value: 0) is an optional argument.
- max\_value (type: int, default value: 9999) is an optional argument.

Example with arguments (returns an integer between 0 and 100):

```
from fake import FAKER  
  
FAKER.pyint(min_value=0, max_value=100)
```

---

### **pybool**

Returns a random boolean value.

```
from fake import FAKER  
  
FAKER.pybool()
```

### **pystr**

Returns a random string of `nb_chars` length.

```
from fake import FAKER  
  
FAKER.pystr()
```

Arguments:

- `nb_chars` (type: `int`, default value: `20`) is an optional argument.

Example with arguments (returns a string of 64 characters):

```
from fake import FAKER  
  
FAKER.pystr(nb_chars=64)
```

### **pyfloat**

Returns a random float between `min_value` and `max_value`.

```
from fake import FAKER  
  
FAKER.pyfloat()
```

Arguments:

- `min_value` (type: `float`, default value: `0.0`) is an optional argument.
- `max_value` (type: `float`, default value: `100.00`) is an optional argument.

Example with arguments (returns a float between 0 and 100):

```
from fake import FAKER  
  
FAKER.pyfloat(min_value=0.0, max_value=100.0)
```

### **pydecimal**

Returns a random decimal, according to given `left_digits` and `right_digits`.

```
from fake import FAKER  
  
FAKER.pydecimal()
```

Arguments:

- `left_digits` (type: `int`, default value: `5`) is an optional argument.
- `right_digits` (type: `int`, default value: `2`) is an optional argument.

- `positive` (type: `bool`, default value: `True`) is an optional argument.

Example with arguments:

```
from fake import FAKER  
FAKER.pydecimal(left_digits=1, right_digits=4, positive=False)
```

---

### **ipv4**

Returns a random IPv4 address.

```
from fake import FAKER  
FAKER.ipv4()
```

---

### **date**

Generates a random date.

```
from fake import FAKER  
FAKER.date()
```

---

Arguments:

- `start_date` (type: `str`, default value: `-7d`) is a optional argument.
- `end_date` (type: `str`, default value: `+0d`) is an optional argument.

Example with arguments, generate a random date between given `start_date` and `end_date`:

```
from fake import FAKER  
FAKER.date(start_date="-1d", end_date="+1d")
```

---

### **date\_time**

Generates a random datetime.

```
from fake import FAKER  
FAKER.date_time()
```

---

Arguments:

- `start_date` (type: `str`, default value: `-7d`) is an optional argument.
- `end_date` (type: `str`, default value: `+0d`) is an optional argument.

Example with arguments, generate a random date between given `start_date` and `end_date`:

```
from fake import FAKER

FAKER.date_time(start_date="-1d", end_date="+1d")
```

## pdf

Generates a content (bytes) of a PDF document.

```
from fake import FAKER

FAKER.pdf()
```

Arguments:

- `nb_pages` (type: `int`, default value: 1) is an optional argument.
- `texts` (type: `List[str]`, default value: `None`) is an optional argument.
- `generator` (type: `Union[Type[TextPdfGenerator], Type[GraphicPdfGenerator]]`, default value: `GraphicPdfGenerator`) is an optional argument.
- `metadata` (type: `MetaData`, default value: `None`) is an optional argument.

**Note:** `texts` is valid only in case `TextPdfGenerator` is used.

**Note:** Either `nb_pages` or `texts` shall be provided. `nb_pages` is by default set to 1, but if `texts` is given, the value of `nb_pages` is adjusted accordingly.

Examples with arguments.

Generate a content (bytes) of a PDF document of 100 pages with random graphics:

```
from fake import FAKER

FAKER.pdf(nb_pages=100)
```

Generate a content (bytes) of a PDF document of 100 pages with random texts:

```
from fake import FAKER
from fake import TextPdfGenerator

FAKER.pdf(nb_pages=100, generator=TextPdfGenerator)
```

If you want to get insights of the content used to generate the PDF (texts), pass the `metadata` argument.

```
from fake import FAKER
from fake import MetaData, TextPdfGenerator

metadata = MetaData()
FAKER.pdf(nb_pages=100, generator=TextPdfGenerator, metadata=metadata)

print(metadata.content) # Inspect `metadata`
```

## image

Generates a content (bytes) of a image of the specified format and colour.

```
from fake import FAKER

FAKER.image() # Supported formats are `png`, `svg`, `bmp` and `gif`
```

Arguments:

- `image_format` (type: str, default value: `png`) is an optional argument.
- `size` (type: Tuple[int, int], default value: `(100, 100)`) is an optional argument.
- `color` (type: Tuple[int, int, int], default value: `(0, 0, 255)`) is an optional argument.

Example with arguments.

```
from fake import FAKER

FAKER.image(
    image_format="svg", # SVG format
    size=(640, 480), # 640px width, 480px height
    color=(0, 0, 0), # Fill rectangle with black
)
```

## docx

Generates a content (bytes) of a DOCX document.

```
from fake import FAKER

FAKER.docx()
```

Arguments:

- `nb_pages` (type: int, default value: 1) is an optional argument.
- `texts` (type: List[str], default value: None) is an optional argument.

---

**Note:** Either `nb_pages` or `texts` shall be provided. `nb_pages` is by default set to 1, but if `texts` is given, the value of `nb_pages` is adjusted accordingly.

---

Examples with arguments.

Generate a content (bytes) of a DOCX document of 100 pages with random texts:

```
from fake import FAKER

FAKER.docx(nb_pages=100)
```

If you want to get insights of the content used to generate the DOCX (texts), pass the `metadata` argument.



```

from fake import FAKER
from fake import MetaData

metadata = MetaData()
FAKER.docx(nb_pages=100, metadata=metadata)

print(metadata.content)  # Inspect ``metadata``

```

### pdf\_file

Generates a PDF file.

```

from fake import FAKER

FAKER.pdf_file()

```

Arguments:

**Note:** Accepts all arguments of pdf + the following:

- storage (type: BaseStorage, default value: None) is an optional argument.
- basename (type: str, default value: None) is an optional argument.
- prefix (type: str, default value: None) is an optional argument.

Examples with arguments.

Generate a PDF document of 100 pages with random graphics:

```

from fake import FAKER

FAKER.pdf_file(nb_pages=100)

```

Generate a PDF document of 100 pages with random texts:

```

from fake import FAKER
from fake import TextPdfGenerator

FAKER.pdf_file(nb_pages=100, generator=TextPdfGenerator)

```

If you want to get insights of the content used to generate the PDF (texts), pass the `metadata` argument.

```

from fake import FAKER
from fake import MetaData, TextPdfGenerator

metadata = MetaData()
FAKER.pdf_file(nb_pages=100, generator=TextPdfGenerator, metadata=metadata)

print(metadata.content)  # Inspect ``metadata``

```

### png\_file

Generates a PNG file.

```
from fake import FAKER

FAKER.png_file()
```

Arguments:

---

**Note:** Accepts all arguments of `png` + the following:

---

- `storage` (type: `BaseStorage`, default value: `None`) is an optional argument.
- `basename` (type: `str`, default value: `None`) is an optional argument.
- `prefix` (type: `str`, default value: `None`) is an optional argument.

Example with arguments.

```
from fake import FAKER

FAKER.png_file(
    basename="png_file", # Basename
    size=(640, 480), # 640px width, 480px height
    color=(0, 0, 0), # Fill rectangle with black
)
```

---

### `svg_file`

Generates an SVG file.

```
from fake import FAKER

FAKER.svg_file()
```

Arguments:

---

**Note:** Accepts all arguments of `svg` + the following:

---

- `storage` (type: `BaseStorage`, default value: `None`) is an optional argument.
- `basename` (type: `str`, default value: `None`) is an optional argument.
- `prefix` (type: `str`, default value: `None`) is an optional argument.

Example with arguments.

```
from fake import FAKER

FAKER.svg_file(
    basename="svg_file", # Basename
    size=(640, 480), # 640px width, 480px height
    color=(0, 0, 0), # Fill rectangle with black
)
```

---

### `bmp_file`

Generates a BMP file.

```
from fake import FAKER  
  
FAKER.bmp_file()
```

Arguments:

**Note:** Accepts all arguments of `bmp` + the following:

- `storage` (type: `BaseStorage`, default value: `None`) is an optional argument.
- `basename` (type: `str`, default value: `None`) is an optional argument.
- `prefix` (type: `str`, default value: `None`) is an optional argument.

Example with arguments.

```
from fake import FAKER  
  
FAKER.bmp_file(  
    basename="bmp_file", # Basename  
    size=(640, 480), # 640px width, 480px height  
    color=(0, 0, 0), # Fill rectangle with black  
)
```

## gif\_file

Generates a GIF file.

```
from fake import FAKER  
  
FAKER.gif_file()
```

Arguments:

**Note:** Accepts all arguments of `gif` + the following:

- `storage` (type: `BaseStorage`, default value: `None`) is an optional argument.
- `basename` (type: `str`, default value: `None`) is an optional argument.
- `prefix` (type: `str`, default value: `None`) is an optional argument.

Example with arguments.

```
from fake import FAKER  
  
FAKER.gif_file(  
    basename="gif_file", # Basename  
    size=(640, 480), # 640px width, 480px height  
    color=(0, 0, 0), # Fill rectangle with black  
)
```

### txt\_file

Generates a TXT file.

```
from fake import FAKER  
  
FAKER.txt_file()
```

Arguments:

---

**Note:** Accepts all arguments of `text` + the following:

---

- `storage` (type: `BaseStorage`, default value: `None`) is an optional argument.
- `basename` (type: `str`, default value: `None`) is an optional argument.
- `prefix` (type: `str`, default value: `None`) is an optional argument.

Example with arguments.

```
from fake import FAKER  
  
FAKER.txt_file(  
    basename="txt_file", # Basename  
    nb_chars=10_000, # 10_000 characters long  
)
```

## 11.2 Creating images

Creating images for testing could be a challenging job. The goal of this library is to help you out with basic tasks. You can easily generate very basic graphic images, but no custom shapes. Paper size is A4.

If you don't like how image files are generated by this library, you can check the [faker-file](#) package, which can produce complex images.

### 11.2.1 Supported image formats

Currently, 4 image formats are supported:

- PNG.
- SVG.
- BMP.
- GIF.

## 11.2.2 Generating images as bytes

See the following full functional snippet for generating a PNG image.

```
from fake import FAKER

png_bytes = FAKER.png()
```

*See the full example here*

The generated PNG image will be an image filled with a given color of a size 100x100 px.

If you want image of a different size or color, provide size (Tuple[int, int]) and color (Tuple[int, int, int]) arguments along. See the example below:

```
png_bytes = FAKER.png(size=(500, 500), color=(127, 127, 127))
```

*See the full example here*

## 11.2.3 Generating files

Generate a PNG image.

```
png_file = FAKER.png_file()
```

*See the full example here*

With size and color tweaks:

```
png_file = FAKER.png_file(size=(500, 500), color=(127, 127, 127))
```

*See the full example here*

All other formats (SVG, BMP and GIF) work in exact same way.

## 11.3 Creating PDF

PDF is certainly one of the most complicated formats out there. And certainly one of the formats most of the developers will be having trouble with, as there are many versions and dialects.

The goal of this library is to help you out with basic tasks. You can easily generate PDFs with 100 pages (paper size is A4), having a little text on each. Or you can generate PDFs with images. Currently, you can't have both at the same time.

If you don't like how PDF files are generated by this library, you can check the [faker-file](#) package, which can produce complex PDF documents.

### 11.3.1 Building PDF with text

#### If you need bytes

```
from fake import FAKER, TextPdfGenerator

pdf_bytes = FAKER.pdf(generator=TextPdfGenerator)
```

*See the full example [here](#)*

The generated PDF will consist of a single page with little text on it.

If you want to control number of pages created, you could:

- Pass the list of texts to be used in `texts` argument.
- Pass the number of pages to be created in `nb_pages` argument.

---

See the example below for `texts` tweak:

```
texts = FAKER.sentences()
pdf_bytes = FAKER.pdf(texts=texts, generator=TextPdfGenerator)
```

*See the full example [here](#)*

---

See the example below for `nb_pages` tweak:

```
pdf_bytes = FAKER.pdf(nb_pages=100, generator=TextPdfGenerator)
```

*See the full example [here](#)*

#### If you need files

```
pdf_file = FAKER.pdf_file(generator=TextPdfGenerator)
```

*See the full example [here](#)*

---

With `texts` tweak:

```
texts = FAKER.sentences()
pdf_file = FAKER.pdf_file(texts=texts, generator=TextPdfGenerator)
```

*See the full example [here](#)*

---

With `nb_pages` tweak:

```
pdf_file = FAKER.pdf_file(nb_pages=100, generator=TextPdfGenerator)
```

*See the full example [here](#)*

## 11.3.2 Building PDF with graphics

### If you need bytes

```
from fake import FAKER, GraphicPdfGenerator

pdf_bytes = FAKER.pdf(generator=GraphicPdfGenerator)
```

*See the full example [here](#)*

The generated PDF will consist of a single page with a colored square on it.

If you want PDF with more pages, provide the `nb_pages` argument.

---

See the example below for `nb_pages` tweak:

```
pdf_bytes = FAKER.pdf(nb_pages=100, generator=GraphicPdfGenerator)
```

*See the full example [here](#)*

### If you need files

```
pdf_file = FAKER.pdf_file(generator=GraphicPdfGenerator)
```

*See the full example [here](#)*

---

With `nb_pages` tweak:

```
pdf_file = FAKER.pdf_file(nb_pages=100, generator=GraphicPdfGenerator)
```

*See the full example [here](#)*

## 11.4 Creating DOCX

The goal of this library is to help you out with basic tasks. You can easily generate DOCX files with 100 pages (paper size is A4), having a little text on each.

If you don't like how DOCX files are generated by this library, you can check the [faker-file](#) package, which can produce complex DOCX documents.

### 11.4.1 If you need bytes

```
from fake import FAKER

docx_bytes = FAKER.docx()
```

*See the full example [here](#)*

The generated DOCX will consist of a single page with little text on it.

If you want to control number of pages created, you could:

- Pass the list of texts to be used in `texts` argument.
  - Pass the number of pages to be created in `nb_pages` argument.
- 

See the example below for `texts` tweak:

```
texts = FAKER.sentences()  
docx_bytes = FAKER.docx(texts=texts)
```

*See the full example here*

---

See the example below for `nb_pages` tweak:

```
docx_bytes = FAKER.docx(nb_pages=100)
```

*See the full example here*

---

### 11.4.2 If you need files

```
docx_file = FAKER.docx_file()
```

*See the full example here*

---

With `texts` tweak:

```
texts = FAKER.sentences()  
docx_file = FAKER.docx_file(texts=texts)
```

*See the full example here*

---

With `nb_pages` tweak:

```
docx_file = FAKER.docx_file(nb_pages=100)
```

*See the full example here*

---

## 11.5 Factories

- `pre_save` is a method decorator that will always run before the instance is saved.
- `post_save` is a method decorator that will always run after the instance is saved.
- `trait` decorator runs the code if set to `True` in factory constructor.
- `PreSave` is like the `pre_save` decorator of the `ModelFactory`, but you can pass arguments to it and have a lot of flexibility. See a working example (below) of how set a user password in Django.
- `PostSave` is like the `post_save` decorator of the `ModelFactory`, but you can pass arguments to it and have a lot of flexibility. See a working example (below) of how to assign a user to a Group after user creation.



- `LazyAttribute` expects a callable, will take the instance as a first argument, runs it with extra arguments specified and sets the value as an attribute name.
- `LazyFunction` expects a callable, runs it (without any arguments) and sets the value as an attribute name.
- `SubFactory` is for specifying relations (typically - ForeignKeys).

### 11.5.1 Django example

Filename: `article/models.py`

```
from django.conf import settings
from django.db import models
from django.utils import timezone

class Article(models.Model):
    title = models.CharField(max_length=255)
    slug = models.SlugField(unique=True)
    content = models.TextField()
    headline = models.TextField()
    category = models.CharField(max_length=255)
    image = models.ImageField(null=True, blank=True)
    pub_date = models.DateField(default=timezone.now)
    safe_for_work = models.BooleanField(default=False)
    minutes_to_read = models.IntegerField(default=5)
    author = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.CASCADE
    )
```

See the full example [here](#)

Filename: `article/factories.py`

```
import random
from functools import partial

from django.conf import settings
from django.contrib.auth.models import Group, User
from django.utils import timezone
from fake import (
    FACTORY,
    DjangoModelFactory,
    FileSystemStorage,
    LazyAttribute,
    LazyFunction,
    PostSave,
    PreSave,
    SubFactory,
    post_save,
    pre_save,
    trait,
)
```

(continues on next page)

(continued from previous page)

```

from article.models import Article

# For Django, all files shall be placed inside `MEDIA_ROOT` directory.
# That's why you need to apply this trick - define a
# custom `FileSystemStorage` class and pass it to the file factory as
# `storage` argument.
STORAGE = FileSystemStorage(root_path=settings.MEDIA_ROOT, rel_path="tmp")
CATEGORIES = (
    "art",
    "technology",
    "literature",
)

class GroupFactory(DjangoModelFactory):
    """Group factory."""

    name = FACTORY.word()

    class Meta:
        model = Group
        get_or_create = ("name",)

def set_password(user: User, password: str) -> None:
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    group = GroupFactory(name=name)
    user.groups.add(group)

class UserFactory(DjangoModelFactory):

    username = FACTORY.username()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    date_joined = FACTORY.date_time(tzinfo=timezone.get_current_timezone())
    last_login = FACTORY.date_time(tzinfo=timezone.get_current_timezone())
    is_superuser = False
    is_staff = False
    is_active = FACTORY.pybool()
    password = PreSave(set_password, password="test1234")
    group = PostSave(add_to_group, name="TestGroup1234")

    class Meta:
        model = User
        get_or_create = ("username",)

```

(continues on next page)

(continued from previous page)

```

@post_save
def _send_registration_email(self, instance):
    """Send an email with registration information."""
    # Your code here

@trait
def is_admin_user(self, instance: User) -> bool:
    instance.is_superuser = True
    instance.is_staff = True
    instance.is_active = True

class ArticleFactory(DjangoModelFactory):

    title = FACTORY.sentence()
    slug = FACTORY.slug()
    content = FACTORY.text()
    headline = LazyAttribute(lambda o: o.content[:25])
    category = LazyFunction(partial(random.choice, CATEGORIES))
    image = FACTORY.png_file(storage=STORAGE)
    pub_date = FACTORY.date(tzinfo=timezone.get_current_timezone())
    safe_for_work = FACTORY.pybool()
    minutes_to_read = FACTORY.pyint(min_value=1, max_value=10)
    author = SubFactory(UserFactory)

    class Meta:
        model = Article

```

*See the full example [here](#)*

### Usage example

```

# Create one article
article = ArticleFactory()

# Create 5 articles
articles = ArticleFactory.create_batch(5)

# Create one article with authors username set to admin.
article = ArticleFactory(author__username="admin")

# Using trait
user = UserFactory(is_admin_user=True)

# Using trait in SubFactory
article = ArticleFactory(author__is_admin_user=True)

# Create a user. Created user will automatically have his password
# set to "test1234" and will be added to the group "Test group".
user = UserFactory()

```

(continues on next page)

(continued from previous page)

```
# Create a user with custom password
user = UserFactory(
    password=PreSave(set_password, password="another-pass"),
)

# Add a user to another group
user = UserFactory(
    group=PostSave(add_to_group, name="Another group"),
)

# Or even add user to multiple groups at once
user = UserFactory(
    group_1=PostSave(add_to_group, name="Another group"),
    group_2=PostSave(add_to_group, name="Yet another group"),
)
```

---

## 11.5.2 Pydantic example

Filename: `article/models.py`

```
from datetime import date, datetime
from typing import Optional, Set

from fake import xor_transform
from pydantic import BaseModel, Field

class Group(BaseModel):
    id: int
    name: str

    class Config:
        allow_mutation = False

    def __hash__(self):
        return hash((self.id, self.name))

    def __eq__(self, other):
        if isinstance(other, Group):
            return self.id == other.id and self.name == other.name
        return False

class User(BaseModel):
    id: int
    username: str = Field(..., max_length=255)
    first_name: str = Field(..., max_length=255)
    last_name: str = Field(..., max_length=255)
```

(continues on next page)

(continued from previous page)

```

email: str = Field(..., max_length=255)
date_joined: datetime = Field(default_factory=datetime.utcnow)
last_login: Optional[datetime] = None
password: Optional[str] = Field("", max_length=255)
is_superuser: bool = Field(default=False)
is_staff: bool = Field(default=False)
is_active: bool = Field(default=True)
groups: Set[Group] = Field(default_factory=set)

class Config:
    extra = "allow" # For testing purposes only

def __str__(self):
    return self.username

def set_password(self, password: str) -> None:
    self.password = xor_transform(password)

class Article(BaseModel):
    id: int
    title: str = Field(..., max_length=255)
    slug: str = Field(..., max_length=255, unique=True)
    content: str
    image: Optional[str] = None # Use str to represent the image path or URL
    pub_date: date = Field(default_factory=date.today)
    safe_for_work: bool = False
    minutes_to_read: int = 5
    author: User

    class Config:
        extra = "allow" # For testing purposes only

    def __str__(self):
        return self.title

```

See the full example [here](#)

Filename: `article/factories.py`

```

from pathlib import Path

from fake import (
    FACTORY,
    FileSystemStorage,
    ModelFactory,
    PostSave,
    PreSave,
    SubFactory,
    post_save,
    pre_save,

```

(continues on next page)

(continued from previous page)

```

    trait,
)

from article.models import Article, Group, User

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
MEDIA_ROOT = BASE_DIR / "media"

STORAGE = FileSystemStorage(root_path=MEDIA_ROOT, rel_path="tmp")

class GroupFactory(ModelFactory):
    id = FACTORY.pyint()
    name = FACTORY.word()

    class Meta:
        model = Group
        get_or_create = ("name",)

def set_password(user: User, password: str) -> None:
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    group = GroupFactory(name=name)
    user.groups.add(group)

class UserFactory(ModelFactory):
    id = FACTORY.pyint()
    username = FACTORY.username()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    date_joined = FACTORY.date_time()
    last_login = FACTORY.date_time()
    is_superuser = False
    is_staff = False
    is_active = FACTORY.pybool()
    password = PreSave(set_password, password="test1234")
    group = PostSave(add_to_group, name="TestGroup1234")

    class Meta:
        model = User

    @trait
    def is_admin_user(self, instance: User) -> None:
        instance.is_superuser = True
        instance.is_staff = True
        instance.is_active = True

```

(continues on next page)

(continued from previous page)

```

class ArticleFactory(ModelFactory):
    id = FACTORY.pyint()
    title = FACTORY.sentence()
    slug = FACTORY.slug()
    content = FACTORY.text()
    image = FACTORY.png_file(storage=STORAGE)
    pub_date = FACTORY.date()
    safe_for_work = FACTORY.pybool()
    minutes_to_read = FACTORY.pyint(min_value=1, max_value=10)
    author = SubFactory(UserFactory)

    class Meta:
        model = Article

```

*See the full example [here](#)*

*Used just like in previous example.*

### 11.5.3 TortoiseORM example

*Filename: article/models.py*

```

from datetime import date

from fake import xor_transform
from tortoise import fields
from tortoise.models import Model

class Group(Model):
    """Group model."""

    id = fields.IntField(pk=True)
    name = fields.CharField(max_length=255, unique=True)

class User(Model):
    """User model."""

    id = fields.IntField(pk=True)
    username = fields.CharField(max_length=255, unique=True)
    first_name = fields.CharField(max_length=255)
    last_name = fields.CharField(max_length=255)
    email = fields.CharField(max_length=255)
    password = fields.CharField(max_length=255, null=True, blank=True)
    last_login = fields.DateTimeField(null=True, blank=True)
    is_superuser = fields.BooleanField(default=False)
    is_staff = fields.BooleanField(default=False)

```

(continues on next page)

(continued from previous page)

```
is_active = fields.BooleanField(default=True)
date_joined = fields.DateTimeField(null=True, blank=True)
groups = fields.ManyToManyField("models.Group", on_delete=fields.CASCADE)

def set_password(self, password: str) -> None:
    self.password = xor_transform(password)

class Article(Model):
    """Article model."""

    id = fields.IntField(pk=True)
    title = fields.CharField(max_length=255)
    slug = fields.CharField(max_length=255, unique=True)
    content = fields.TextField()
    image = fields.TextField(null=True, blank=True)
    pub_date = fields.DateField(default=date.today)
    safe_for_work = fields.BooleanField(default=False)
    minutes_to_read = fields.IntField(default=5)
    author = fields.ForeignKeyField("models.User", on_delete=fields.CASCADE)
```

*See the full example here*

---

Filename: article/factories.py

```
from pathlib import Path

from fake import (
    FACTORY,
    FileSystemStorage,
    PostSave,
    PreSave,
    SubFactory,
    TortoiseModelFactory,
    post_save,
    pre_save,
    run_async_in_thread,
    trait,
)

from article.models import Article, Group, User

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
MEDIA_ROOT = BASE_DIR / "media"

STORAGE = FileSystemStorage(root_path=MEDIA_ROOT, rel_path="tmp")

class GroupFactory(TortoiseModelFactory):
    """Group factory."""
```

(continues on next page)



(continued from previous page)

```

name = FACTORY.word()

class Meta:
    model = Group
    get_or_create = ("name",)

def set_password(user: User, password: str) -> None:
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    group = GroupFactory(name=name)

    async def _add_to_group():
        await user.groups.add(group)
        await user.save()

    run_async_in_thread(_add_to_group())

class UserFactory(TortoiseModelFactory):
    """User factory."""

    username = FACTORY.username()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    date_joined = FACTORY.date_time()
    last_login = FACTORY.date_time()
    is_superuser = False
    is_staff = False
    is_active = FACTORY.pybool()
    password = PreSave(set_password, password="test1234")
    group = PostSave(add_to_group, name="TestGroup1234")

    class Meta:
        model = User
        get_or_create = ("username",)

    @trait
    def is_admin_user(self, instance: User) -> None:
        instance.is_superuser = True
        instance.is_staff = True
        instance.is_active = True

class ArticleFactory(TortoiseModelFactory):
    """Article factory."""

    title = FACTORY.sentence()

```

(continues on next page)

(continued from previous page)

```
slug = FACTORY.slug()
content = FACTORY.text()
image = FACTORY.png_file(storage=STORAGE)
pub_date = FACTORY.date()
safe_for_work = FACTORY.pybool()
minutes_to_read = FACTORY.pyint(min_value=1, max_value=10)
author = SubFactory(UserFactory)

class Meta:
    model = Article
```

*See the full example [here](#)*

*Used just like in previous example.*

---

### 11.5.4 Dataclasses example

*Filename: article/models.py*

```
from dataclasses import dataclass, field
from datetime import date, datetime
from typing import Optional, Set

from fake import xor_transform


@dataclass(frozen=True)
class Group:
    id: int
    name: str


@dataclass
class User:
    id: int
    username: str
    first_name: str
    last_name: str
    email: str
    date_joined: datetime = field(default_factory=datetime.utcnow)
    last_login: Optional[datetime] = None
    password: Optional[str] = None
    is_superuser: bool = False
    is_staff: bool = False
    is_active: bool = True
    groups: Set[Group] = field(default_factory=set)

    def __str__(self):
        return self.username
```

(continues on next page)

(continued from previous page)

```

def set_password(self, password: str) -> None:
    self.password = xor_transform(password)

@dataclass
class Article:
    id: int
    title: str
    slug: str
    content: str
    author: User
    image: Optional[str] = None # Use str to represent the image path or URL
    pub_date: date = field(default_factory=date.today)
    safe_for_work: bool = False
    minutes_to_read: int = 5

    def __str__(self):
        return self.title

```

See the full example [here](#)

Filename: `article/factories.py`

```

from pathlib import Path

from fake import (
    FACTORY,
    FileSystemStorage,
    ModelFactory,
    PostSave,
    PreSave,
    SubFactory,
    post_save,
    pre_save,
    trait,
)

from article.models import Article, Group, User

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
MEDIA_ROOT = BASE_DIR / "media"

STORAGE = FileSystemStorage(root_path=MEDIA_ROOT, rel_path="tmp")

class GroupFactory(ModelFactory):
    id = FACTORY.pyint()
    name = FACTORY.word()

    class Meta:

```

(continues on next page)

(continued from previous page)

```

        model = Group
        get_or_create = ("name",)

def set_password(user: User, password: str) -> None:
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    group = GroupFactory(name=name)
    user.groups.add(group)

class UserFactory(ModelFactory):
    id = FACTORY.pyint()
    username = FACTORY.username()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    date_joined = FACTORY.date_time()
    last_login = FACTORY.date_time()
    is_superuser = False
    is_staff = False
    is_active = FACTORY.pybool()
    password = PreSave(set_password, password="test1234")
    group = PostSave(add_to_group, name="TestGroup1234")

    class Meta:
        model = User

    @trait
    def is_admin_user(self, instance: User) -> None:
        instance.is_superuser = True
        instance.is_staff = True
        instance.is_active = True

class ArticleFactory(ModelFactory):
    id = FACTORY.pyint()
    title = FACTORY.sentence()
    slug = FACTORY.slug()
    content = FACTORY.text()
    image = FACTORY.png_file(storage=STORAGE)
    pub_date = FACTORY.date()
    safe_for_work = FACTORY.pybool()
    minutes_to_read = FACTORY.pyint(min_value=1, max_value=10)
    author = SubFactory(UserFactory)

    class Meta:
        model = Article

```

*See the full example [here](#)*

*Used just like in previous example.*

### 11.5.5 SQLAlchemy example

*Filename: config.py*

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker

# SQLAlchemy
DATABASE_URL = "sqlite:///test_database.db"
ENGINE = create_engine(DATABASE_URL)
SESSION = scoped_session(sessionmaker(bind=ENGINE))
```

*See the full example [here](#)*

*Filename: article/models.py*

```
from datetime import datetime

from fake import xor_transform
from sqlalchemy import (
    Boolean,
    Column,
    DateTime,
    ForeignKey,
    Integer,
    String,
    Table,
    Text,
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

Base = declarative_base()

# Association table for the many-to-many relationship
user_group_association = Table(
    "user_group",
    Base.metadata,
    Column("user_id", Integer, ForeignKey("users.id")),
    Column("group_id", Integer, ForeignKey("groups.id")),
)

class Group(Base):
    """Group model."""
```

(continues on next page)

```

__tablename__ = "groups"

id = Column(Integer, primary_key=True)
name = Column(String(255), unique=True)

class User(Base):
    """User model."""

    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    username = Column(String(255), unique=True)
    first_name = Column(String(255))
    last_name = Column(String(255))
    email = Column(String(255))
    date_joined = Column(DateTime, default=datetime.utcnow)
    last_login = Column(DateTime, nullable=True)
    password = Column(String(255), nullable=True)
    is_superuser = Column(Boolean, default=False)
    is_staff = Column(Boolean, default=False)
    is_active = Column(Boolean, default=True)

    # Many-to-many relationship
    groups = relationship(
        "Group", secondary=user_group_association, backref="users"
    )

    # One-to-many relationship
    articles = relationship("Article", back_populates="author")

    def set_password(self, password: str) -> None:
        self.password = xor_transform(password)

class Article(Base):
    """Article model."""

    __tablename__ = "articles"

    id = Column(Integer, primary_key=True)
    title = Column(String(255))
    slug = Column(String(255), unique=True)
    content = Column(Text)
    image = Column(Text, nullable=True)
    pub_date = Column(DateTime, default=datetime.utcnow)
    safe_for_work = Column(Boolean, default=False)
    minutes_to_read = Column(Integer, default=5)
    author_id = Column(Integer, ForeignKey("users.id"))

    author = relationship("User", back_populates="articles")

```

See the full example [here](#)

Filename: *article/factories.py*

```

from pathlib import Path

from fake import (
    FACTORY,
    FileSystemStorage,
    PostSave,
    PreSave,
    SQLAlchemyModelFactory,
    SubFactory,
    post_save,
    pre_save,
    trait,
)

from article.models import Article, Group, User
from config import SESSION

# Storage config. Build paths inside the project like this: BASE_DIR / 'subdir'
BASE_DIR = Path(__file__).resolve().parent.parent
MEDIA_ROOT = BASE_DIR / "media"
STORAGE = FileSystemStorage(root_path=MEDIA_ROOT, rel_path="tmp")

def get_session():
    return SESSION()

class GroupFactory(SQLAlchemyModelFactory):
    """User factory."""

    name = FACTORY.word()

    class Meta:
        model = Group
        get_or_create = ("name",)

    class MetaSQLAlchemy:
        get_session = get_session

def set_password(user: User, password: str) -> None:
    user.set_password(password)

def add_to_group(user: User, name: str) -> None:
    session = get_session()
    # Check if the group already exists
    group = session.query(Group).filter_by(name=name).first()

    # If the group doesn't exist, create a new one

```

(continues on next page)

(continued from previous page)

```

if not group:
    group = Group(name=name)
    session.add(group)
    session.commit() # Commit to assign an ID to the new group

# Add the group to the user's groups using append
if group not in user.groups:
    user.groups.append(group)
    session.commit() # Commit the changes

class UserFactory(SQLAlchemyModelFactory):
    """User factory."""

    username = FACTORY.username()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    date_joined = FACTORY.date_time()
    last_login = FACTORY.date_time()
    is_superuser = False
    is_staff = False
    is_active = FACTORY.pybool()
    password = PreSave(set_password, password="test1234")
    group = PostSave(add_to_group, name="TestGroup1234")

    class Meta:
        model = User
        get_or_create = ("username",)

    class MetaSQLAlchemy:
        get_session = get_session

    @trait
    def is_admin_user(self, instance: User) -> None:
        instance.is_superuser = True
        instance.is_staff = True
        instance.is_active = True

class ArticleFactory(SQLAlchemyModelFactory):
    """Article factory."""

    title = FACTORY.sentence()
    slug = FACTORY.slug()
    content = FACTORY.text()
    image = FACTORY.png_file(storage=STORAGE)
    pub_date = FACTORY.date()
    safe_for_work = FACTORY.pybool()
    minutes_to_read = FACTORY.pyint(min_value=1, max_value=10)
    author = SubFactory(UserFactory)

```

(continues on next page)



(continued from previous page)

```
class Meta:
    model = Article

class MetaSQLAlchemy:
    get_session = get_session
```

See the full example [here](#)

Used just like in previous example.

## 11.6 Customization

- The `fake.FAKER` is an instance of the `fake.Faker` class.
- The `fake.FACTORY` is an instance of the `fake.Factory` class, initialized with `fake.FAKER` instance.

The `Faker` class is easy to customize. See the following example:

Filename: `custom_fake.py`

```
import random
import string

from fake import Faker, Factory, provider

# Custom first names dictionary
FIRST_NAMES = [
    "Anahit",
    "Ani",
    "Aram",
    "Areg",
    "Artur",
    "Astghik",
    "Atom",
    "Barsegh",
    "Gaiane",
    "Gor",
    "Hakob",
    "Hasmik",
    "Levon",
    "Lilit",
    "Mariam",
    "Nare",
    "Narek",
    "Nune",
    "Raffi",
    "Shant",
    "Tatev",
    "Tigran",
    "Vahan",
    "Vardan",
```

(continues on next page)

(continued from previous page)

```

]

# Custom last names dictionary
LAST_NAMES = [
    "Amatouni",
    "Avagyan",
    "Danielyan",
    "Egoyan",
    "Gevorgyan",
    "Gnouni",
    "Grigoryan",
    "Hakobyan",
    "Harutyunyan",
    "Hovhannisyan",
    "Karapetyan",
    "Khachatryan",
    "Manukyan",
    "Melikyan",
    "Mkrtchyan",
    "Petrosyan",
    "Sahakyants",
    "Sargsyan",
    "Saroyan",
    "Sedrakyan",
    "Simonyan",
    "Stepanyan",
    "Ter-Martirosyan",
    "Vardanyan",
]

# Custom words dictionary
WORDS = [
    "time", "person", "year", "way", "day", "thing", "man", "world",
    "life", "hand", "part", "child", "eye", "woman", "place", "work",
    "week", "case", "point", "government", "company", "number", "group",
    "problem", "fact", "be", "have", "do", "say", "get", "make", "go",
    "know", "take", "see", "come", "think", "look", "want", "give",
    "use", "find", "tell", "ask", "work", "seem", "feel", "try", "leave",
    "call", "good", "new", "first", "last", "long", "great", "little",
    "own", "other", "old", "right", "big", "high", "different", "small",
    "large", "next", "early", "young", "important", "few", "public",
    "bad", "same", "able", "to", "of", "in", "for", "on", "with", "as",
    "at", "by", "from", "up", "about", "into", "over", "after",
    "beneath", "under", "above", "the", "and", "a", "that", "I", "it",
    "not",
]

STREET_NAMES = [
    "Bosweg",
    "Groningerweg",
    "Jasmijnstraat",
    "Noordstraat",

```

(continues on next page)

(continued from previous page)

```

    "Ooststraat",
    "Oranjestraat",
    "Prinsengracht",
    "Ringweg",
    "Weststraat",
    "Zonnelaan",
    "Zuidstraat",
]

CITIES = [
    "Amsterdam",
    "Delft",
    "Den Haag",
    "Groningen",
    "Leiden",
    "Nijmegen",
]

REGIONS = [
    "Friesland",
    "Groningen",
    "Limburg",
    "Utrecht",
]

class CustomFaker(Faker):
    """Custom Faker class."""

    def load_names(self) -> None:
        """Override default first- and last-names dictionaries."""
        self._first_names = FIRST_NAMES
        self._last_names = LAST_NAMES

    def load_words(self) -> None:
        """Override default words dictionary."""
        self._words = WORDS

    @provider
    def address_line(self) -> str:
        """Generate a random Dutch address line like 'Oranjestraat 1'.

        :return: A randomly generated Dutch address line as a string.
        """
        # Generate components of the address
        street = random.choice(STREET_NAMES)
        house_number = random.randint(1, 200)
        suffixes = [""] * 10 + ["A", "B", "C"] # Optional suffixes
        suffix = random.choice(suffixes)

        # Combine components into a Dutch address format
        return f"{street} {house_number}{suffix}"

```

(continues on next page)

(continued from previous page)

```
@provider
def city(self) -> str:
    return random.choice(CITIES)

@provider
def region(self) -> str:
    return random.choice(REGIONS)

@provider
def postal_code(self) -> str:
    """Generate a random Dutch postal code in the format '1234 AB'.

    :return: A randomly generated Dutch postal code as a string.
    """
    number_part = "".join(random.choices(string.digits, k=4))
    letter_part = "".join(random.choices(string.ascii_uppercase, k=2))
    return f"{number_part} {letter_part}"
```

```
FAKER = CustomFaker()
FACTORY = Factory(FAKER)
```

The `postal_code` is the provider method and shall be decorated with `@provider` decorator.

You can now use both `FAKER` and `FACTORY` as you would normally do.

---

Filename: *models.py*

```
from dataclasses import dataclass
from datetime import date

@dataclass
class Address:
    id: int
    address_line: str
    postal_code: str
    city: str
    region: str

    def __str__(self) -> str:
        return self.address_line

@dataclass
class Person:
    id: int
    first_name: str
    last_name: str
    email: str
    dob: date
```

(continues on next page)

(continued from previous page)

```
address: Address

def __str__(self) -> str:
    return self.username
```

Filename: factories.py

```
from fake import ModelFactory, SubFactory, post_save, pre_save

from models import Address, Person
from custom_fake import FACTORY

class AddressFactory(ModelFactory):
    id = FACTORY.pyint()
    address_line = FACTORY.address_line()
    postal_code = FACTORY.postal_code()
    city = FACTORY.city()
    region = FACTORY.region()

    class Meta:
        model = Address

class PersonFactory(ModelFactory):
    id = FACTORY.pyint()
    first_name = FACTORY.first_name()
    last_name = FACTORY.last_name()
    email = FACTORY.email()
    dob = FACTORY.date()
    address = SubFactory(AddressFactory)

    class Meta:
        model = Person
```

## 11.7 Security Policy

### 11.7.1 Reporting a Vulnerability

#### Do not report security issues on GitHub!

Please report security issues by emailing Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>.

## 11.7.2 Supported Versions

### Make sure to use the latest version.

The two most recent `fake.py` release series receive security support.

For example, during the development cycle leading to the release of `fake.py` 0.17.x, support will be provided for `fake.py` 0.16.x.

Upon the release of `fake.py` 0.18.x, security support for `fake.py` 0.16.x will end.

Version	Supported	
0.6.x	Yes	
0.5.x	Yes	
< 0.5	No	

## 11.8 Contributor guidelines

### 11.8.1 Developer prerequisites

#### pre-commit

Refer to [pre-commit](#) for installation instructions.

TL;DR:

```
pip install pipx --user # Install pipx
pipx install pre-commit # Install pre-commit
pre-commit install # Install pre-commit hooks
```

Installing [pre-commit](#) will ensure you adhere to the project code quality standards.

### 11.8.2 Code standards

[black](#), [isort](#), [ruff](#) and [doc8](#) will be automatically triggered by [pre-commit](#). Still, if you want to run checks manually:

```
make black
make doc8
make isort
make ruff
```

### 11.8.3 Requirements

Requirements are compiled using [pip-tools](#).

```
make compile-requirements
```

### 11.8.4 Virtual environment

You are advised to work in virtual environment.

TL;DR:

```
python -m venv env
pip install -e .[all]
```

### 11.8.5 Documentation

Check [documentation](#).

### 11.8.6 Testing

Check [testing](#).

If you introduce changes or fixes, make sure to test them locally using all supported environments. For that use `tox`.

```
tox
```

In any case, GitHub Actions will catch potential errors, but using `tox` speeds things up.

### 11.8.7 Pull requests

You can contribute to the project by making a [pull request](#).

For example:

- To fix documentation typos.
- To improve documentation (for instance, to add new recipe or fix an existing recipe that doesn't seem to work).
- To introduce a new feature (for instance, add support for a non-supported file type).

#### Good to know:

- This library consists of a single `fake.py` module. That module is dependency free, self-contained (includes all tests) and portable. Do not submit pull requests splitting the `fake.py` module into small parts.
- Some tests contain simplified implementation of existing libraries (Django ORM, TortoiseORM, SQLAlchemy). If you need to add integration tests for existing functionality, you can add the relevant code and requirements to the examples, along with tests. Currently, all integration tests are running in the CI against the latest version of Python.

#### General list to go through:

- Does your change require documentation update?
- Does your change require update to tests?

- Does your change rely on third-party package or a cloud based service? If so, please consider turning it into a dedicated standalone package, since this library is dependency free (and will always stay so).

**When fixing bugs (in addition to the general list):**

- Make sure to add regression tests.

**When adding a new feature (in addition to the general list):**

- Make sure to update the documentation (check whether the [installation](#), [features](#), [recipes](#) and [quick start](#) require changes).

## 11.8.8 Questions

Questions can be asked on GitHub [discussions](#).

## 11.8.9 Issues

For reporting a bug or filing a feature request use GitHub [issues](#).

**Do not report security issues on GitHub.** Check the [support](#) section.

# 11.9 Contributor Covenant Code of Conduct

## 11.9.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

## 11.9.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting



### 11.9.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

### 11.9.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

### 11.9.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at [artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com). All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

### 11.9.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

#### 1. Correction

**Community Impact:** Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

**Consequence:** A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

#### 2. Warning

**Community Impact:** A violation through a single incident or series of actions.

**Consequence:** A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

### 3. Temporary Ban

**Community Impact:** A serious violation of community standards, including sustained inappropriate behavior.

**Consequence:** A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

### 4. Permanent Ban

**Community Impact:** Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence:** A permanent ban from any sort of public interaction within the community.

## 11.9.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at [https://www.contributor-covenant.org/version/2/0/code\\_of\\_conduct.html](https://www.contributor-covenant.org/version/2/0/code_of_conduct.html).

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>. Translations are available at <https://www.contributor-covenant.org/translations>.

## 11.10 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

<code>major.minor[.revision]</code>
-------------------------------------

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

### 11.10.1 0.6.9

2024-05-10

- Minor fixes in `pdf_file` and `docx_file` providers.
- Minor fixes in docs.

### 11.10.2 0.6.8

2024-05-06

- Minor fixes in docs.

### 11.10.3 0.6.7

2024-01-17

- Add uuids, first\_names, last\_names, names, usernames and slugs plural providers (return List).

### 11.10.4 0.6.6

2024-01-15

- Add image\_url provider.

### 11.10.5 0.6.5

2023-12-18

- Improve docs.
- MyPy fixes.

### 11.10.6 0.6.4

2023-12-16

- Add PreSave and PostSave.

### 11.10.7 0.6.3

2023-12-13

- Add LazyAttribute and LazyFunction.
- Improve package portability (tests).
- Improve tests.

### 11.10.8 0.6.2

2023-12-11

- Add SQLAlchemyModelFactory.

### **11.10.9 0.6.1**

2023-12-10

- Allow to load registered `Faker` instance by `uid` or `alias`.
- Improve test coverage.

### **11.10.10 0.6**

2023-12-09

- Add optional argument `alias` to the `Faker` class.
- Improve multiple `Faker` instances.
- Add `generic_file` provider.

### **11.10.11 0.5**

2023-12-08

- Make `fake.Faker` and `fake.Factory` classes more customizable.
- Introduce `provider` decorator to decorate provider methods.
- Documentation improvements.

### **11.10.12 0.4.1**

2023-12-07

- Added `pydecimal`.
- Make `date_time` timezone aware.
- Added documentation on how to customize.

### **11.10.13 0.4**

2023-12-06

- Streamline on how to use traits, pre- and post-save hooks.

### **11.10.14 0.3.1**

2023-12-04

- Improve Tortoise ORM factory.
- Add traits.
- Improve documentation.

### 11.10.15 0.3

2023-12-03

- Added factories.
- Added mechanism to clean-up (remove) the created test files.
- Improved documentation.

### 11.10.16 0.2

2023-12-01

- Add factories.
- Improve docs.
- Add uuid, slug and username generators.
- Change date\_between to date.
- Change date\_time\_between to date\_time.

### 11.10.17 0.1.3

2023-11-28

- Added pdf\_file, docx\_file, png\_file, svg\_file, bmp\_file, gif\_file support.
- Added storages.

### 11.10.18 0.1.2

2023-11-26

- Adding texts support.
- Improve tests and documentation.

### 11.10.19 0.1.1

2023-11-26

- Adding DOCX support.
- Fixes in documentation.

## 11.10.20 0.1

2023-11-25

- Initial beta release.

## 11.11 Package

### 11.11.1 fake module

<https://github.com/barseghyanartur/fake.py/>

**class** fake.AuthorshipData

Bases: object

```
first_names: Set[str] = {'Andre', 'Andrew', 'Andrey', 'Anthony', 'Barry', 'Ben',
                          'Benjamin', 'Christian', 'Collin', 'David', 'Donald', 'Eric', 'Ezio', 'George',
                          'Gregor', 'Guido', 'Guilherme', 'Jack', 'Jacques', 'Jiwon', 'Ka-Ping', 'Keith',
                          'Kenneth', 'Lars', 'Marc-Andre', 'Martin', 'Michael', 'Mike', 'Nadeem', 'Nick',
                          'Paul', 'Piers', 'Skip', 'Steen', 'Steven', 'Thomas', 'Victor', 'Vinay', 'Zooko'}
```

```
last_names: Set[str] = {'Ascher', 'Baxter', 'Bland', 'Boutsioukis', 'Dalke',
                        'Dart', 'Diederich', 'Dragon De Monsyne', 'Edds', 'Felt', 'Frechet', 'Gertzfield',
                        'Gust', 'Heimes', 'J', 'Kippes', 'Larson', 'Lauder', 'Lemburg', 'Lingl', 'Lumholt',
                        'McGuire', 'Melotti', 'Montanaro', 'O'Whielacronx", 'Peterson', 'Petrov', 'Polo',
                        'Reitz', 'Roberge', 'Sajip', 'Seo', 'Stinner', 'Stuftt and individual contributors',
                        'Vawda', 'Warsaw', 'Winter', 'Wouters', 'Yee', 'van Rossum', 'von Loewis'}
```

**class** fake.BaseStorage(\*args, \*\*kwargs)

Bases: object

Base storage.

**abstract** abspath(filename: Any) → str

Return absolute path.

**abstract** exists(filename: Any) → bool

Check if file exists.

**abstract** generate\_filename(extension: str, prefix: Optional[str] = None, basename: Optional[str] = None) → Any

Generate filename.

**abstract** relpath(filename: Any) → str

Return relative path.

**abstract** unlink(filename: Any) → None

Delete the file.

**abstract** write\_bytes(filename: Any, data: bytes) → int

Write bytes.

**abstract** write\_text(filename: Any, data: str, encoding: Optional[str] = None) → int

Write text.

---

```
class fake.DjangoModelFactory(**kwargs)
```

Bases: *ModelFactory*

Django ModelFactory.

```
classmethod create(**kwargs)
```

```
classmethod save(instance)
```

Save the instance.

```
class fake.DocxGenerator(faker: Faker)
```

Bases: object

DocxGenerator - generates a DOCX file with text.

Usage example:

```
from pathlib import Path
from fake import FAKER

Path("/tmp/example.docx").write_bytes(FAKER.docx(nb_pages=100))
```

```
create(nb_pages: Optional[int] = None, texts: Optional[List[str]] = None, metadata: Optional[MetaData]
       = None) → bytes
```

```
class fake.Factory(faker: Optional[Faker] = None)
```

Bases: object

Factory.

```
property faker
```

```
class fake.FactoryMethod(method_name: str, faker: Optional[Faker] = None, **kwargs)
```

Bases: object

```
class fake.Faker(alias: Optional[str] = None)
```

Bases: object

fake.py - simplified, standalone alternative with no dependencies.

---

Usage example:

```
from fake import FAKER

print(FAKER.first_name()) # Random first name
print(FAKER.last_name()) # Random last name
print(FAKER.name()) # Random name
print(FAKER.word()) # Random word from the Zen of Python
print(FAKER.words(nb=3)) # List of 3 random words from Zen of Python
print(FAKER.sentence()) # Random sentence (5 random words by default)
print(FAKER.paragraph()) # Paragraph (5 random sentences by default)
print(FAKER.paragraphs()) # 3 random paragraphs
print(FAKER.text()) # Random text up to 200 characters
print(FAKER.file_name()) # Random filename with '.txt' extension
print(FAKER.email()) # Random email
print(FAKER.url()) # Random URL
```

(continues on next page)

(continued from previous page)

```
print(FAKER.pyint()) # Random integer
print(FAKER.pybool()) # Random boolean
print(FAKER.pystr()) # Random string
print(FAKER.pyfloat()) # Random float
```

---

PDF:

```
from pathlib import Path
from fake import FAKER, TextPdfGenerator, GraphicPdfGenerator

Path("/tmp/graphic_pdf.pdf").write_bytes(
    FAKER.pdf(nb_pages=100, generator=GraphicPdfGenerator)
)

Path("/tmp/text_pdf.pdf").write_bytes(
    FAKER.pdf(nb_pages=100, generator=TextPdfGenerator)
)
```

---

Various image formats:

```
from pathlib import Path
from fake import FAKER

Path("/tmp/image.png").write_bytes(FAKER.png())

Path("/tmp/image.svg").write_bytes(FAKER.svg())

Path("/tmp/image.bmp").write_bytes(FAKER.bmp())

Path("/tmp/image.gif").write_bytes(FAKER.gif())
```

Note, that all image formats accept *size* (default: *(100, 100)*) and *color* (default: *(255, 0, 0)*) arguments.

**bmp**(*size: Tuple[int, int] = (100, 100)*, *color: Tuple[int, int, int] = (0, 0, 255)*) → bytes

Create a BMP image of a specified color.

**Parameters**

- **size** – Tuple of width and height of the image in pixels.
- **color** – Color of the image in RGB format (tuple of three integers).

**Returns**

Byte content of the BMP image.

**Return type**

bytes

**bmp\_file**(*size: Tuple[int, int] = (100, 100)*, *color: Tuple[int, int, int] = (0, 0, 255)*, *storage: Optional[BaseStorage] = None*, *basename: Optional[str] = None*, *prefix: Optional[str] = None*) → *StringValue*



**date**(*start\_date: str = '-7d', end\_date: str = '+0d', tzinfo: timezone = datetime.timezone.utc*) → date

Generate random date between *start\_date* and *end\_date*.

#### Parameters

- **start\_date** – The start date from which the random date should be generated in the shorthand notation.
- **end\_date** – The end date up to which the random date should be generated in the shorthand notation.
- **tzinfo** – The timezone.

#### Returns

A string representing the formatted date.

#### Return type

date

**date\_time**(*start\_date: str = '-7d', end\_date: str = '+0d', tzinfo: timezone = datetime.timezone.utc*) → datetime

Generate a random datetime between *start\_date* and *end\_date*.

#### Parameters

- **start\_date** – The start datetime from which the random datetime should be generated in the shorthand notation.
- **end\_date** – The end datetime up to which the random datetime should be generated in the shorthand notation.
- **tzinfo** – The timezone.

#### Returns

A string representing the formatted datetime.

#### Return type

datetime

**docx**(*nb\_pages: Optional[int] = 1, texts: Optional[List[str]] = None, metadata: Optional[MetaData] = None*) → bytes

**docx\_file**(*nb\_pages: int = 1, texts: Optional[List[str]] = None, metadata: Optional[MetaData] = None, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None*) → *StringValue*

**email**(*domain: str = 'example.com'*) → str

**file\_name**(*extension: str = 'txt'*) → str

**first\_name**() → str

**first\_names**(*nb: int = 5*) → List[str]

**generic\_file**(*content: Union[bytes, str], extension: str, storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None*) → *StringValue*

**static get\_by\_alias**(*alias: str*) → Optional[Faker]

**static get\_by\_uid**(*uid: str*) → Optional[Faker]

**gif**(size: *Tuple[int, int] = (100, 100)*, color: *Tuple[int, int, int] = (0, 0, 255)*) → bytes

Create a GIF image of a specified color.

**Parameters**

- **size** – Tuple of width and height of the image in pixels.
- **color** – Color of the image in RGB format (tuple of three integers).

**Returns**

Byte content of the GIF image.

**Return type**

bytes

**gif\_file**(size: *Tuple[int, int] = (100, 100)*, color: *Tuple[int, int, int] = (0, 0, 255)*, storage: *Optional[BaseStorage] = None*, basename: *Optional[str] = None*, prefix: *Optional[str] = None*) → *StringValue*

**image**(image\_format: *Literal['png', 'svg', 'bmp', 'gif'] = 'png'*, size: *Tuple[int, int] = (100, 100)*, color: *Tuple[int, int, int] = (0, 0, 255)*) → bytes

**image\_url**(width: *int = 800*, height: *int = 600*, service\_url: *Optional[str] = None*) → str  
Image URL.

**ipv4**() → str

**last\_name**() → str

**last\_names**(nb: *int = 5*) → List[str]

**load\_names**() → None

**load\_words**() → None

**name**() → str

**names**(nb: *int = 5*) → List[str]

**paragraph**(nb\_sentences: *int = 5*) → str

**paragraphs**(nb: *int = 3*) → List[str]

**pdf**(nb\_pages: *int = 1*, generator: *~typing.Union[~typing.Type[~fake.TextPdfGenerator], ~typing.Type[~fake.GraphicPdfGenerator]] = <class 'fake.GraphicPdfGenerator'>*, metadata: *~typing.Optional[~fake.MetaData] = None*, \*\*kwargs) → bytes

Create a PDF document of a given size.

**pdf\_file**(nb\_pages: *int = 1*, generator: *~typing.Union[~typing.Type[~fake.TextPdfGenerator], ~typing.Type[~fake.GraphicPdfGenerator]] = <class 'fake.GraphicPdfGenerator'>*, metadata: *~typing.Optional[~fake.MetaData] = None*, storage: *~typing.Optional[~fake.BaseStorage] = None*, basename: *~typing.Optional[str] = None*, prefix: *~typing.Optional[str] = None*, \*\*kwargs) → *StringValue*

**png**(size: *Tuple[int, int] = (100, 100)*, color: *Tuple[int, int, int] = (0, 0, 255)*) → bytes

Create a PNG image of a specified color.

**Parameters**

- **size** – Tuple of width and height of the image in pixels.

- **color** – Color of the image in RGB format (tuple of three integers).

**Returns**

Byte content of the PNG image.

**Return type**

bytes

**png\_file**(*size: Tuple[int, int] = (100, 100), color: Tuple[int, int, int] = (0, 0, 255), storage: Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None*) → *StringValue*

**pybool**() → bool

**pydecimal**(*left\_digits: int = 5, right\_digits: int = 2, positive: bool = True*) → Decimal

Generate a random Decimal number.

**Parameters**

- **left\_digits** – Number of digits to the left of the decimal point.
- **right\_digits** – Number of digits to the right of the decimal point.
- **positive** – If True, the number will be positive, otherwise it can be negative.

**Returns**

A randomly generated Decimal number.

**Return type**

Decimal

**Raises**

ValueError

**pyfloat**(*min\_value: float = 0.0, max\_value: float = 10.0*) → float

**pyint**(*min\_value: int = 0, max\_value: int = 9999*) → int

**pystr**(*nb\_chars: int = 20*) → str

**sentence**(*nb\_words: int = 5*) → str

**sentences**(*nb: int = 3*) → List[str]

**slug**() → str

**slugs**(*nb: int = 5*) → List[str]

**svg**(*size: Tuple[int, int] = (100, 100), color: Tuple[int, int, int] = (0, 0, 255)*) → bytes

Create a SVG image of a specified color.

**Parameters**

- **size** – Tuple of width and height of the image in pixels.
- **color** – Color of the image in RGB format (tuple of three integers).

**Returns**

Byte content of the SVG image.

**Return type**

bytes

```
svg_file(size: Tuple[int, int] = (100, 100), color: Tuple[int, int, int] = (0, 0, 255), storage:
    Optional[BaseStorage] = None, basename: Optional[str] = None, prefix: Optional[str] = None)
    → StringValue

text(nb_chars: int = 200) → str

texts(nb: int = 3) → List[str]

txt_file(nb_chars: Optional[int] = 200, storage: Optional[BaseStorage] = None, basename: Optional[str]
    = None, prefix: Optional[str] = None, text: Optional[str] = None) → StringValue

url(protocols: Optional[Tuple[str]] = None, tlds: Optional[Tuple[str]] = None, suffixes:
    Optional[Tuple[str]] = None) → str

username() → str

usernames(nb: int = 5) → List[str]

uuid() → UUID

uuids(nb: int = 5) → List[UUID]

word() → str

words(nb: int = 5) → List[str]
```

```
class fake.FileRegistry
```

```
    Bases: object
```

```
    Stores list StringValue instances.
```

```
from fake import FAKER, FILE_REGISTRY

txt_file_1 = FAKER.txt_file()
txt_file_2 = FAKER.txt_file()
...
txt_file_n = FAKER.txt_file()

# The FileRegistry._registry would then contain this:
{
    txt_file_1,
    txt_file_2,
    ...,
    txt_file_n,
}

# Clean up created files as follows:
FILE_REGISTRY.clean_up()
```

```
add(string_value: StringValue) → None
```

```
clean_up() → None
```

```
remove(string_value: Union[StringValue, str]) → bool
```

```
search(value: str) → Optional[StringValue]
```

```
class fake.FileSystemStorage(root_path: Optional[Union[str, Path]] = '/tmp', rel_path: Optional[str] = 'tmp', *args, **kwargs)
```

Bases: [BaseStorage](#)

File storage class using pathlib for path handling.

Usage example:

```
from fake import FAKER, FileSystemStorage

storage = FileSystemStorage()
docx_file = storage.generate_filename(prefix="zzz_", extension="docx")
storage.write_bytes(docx_file, FAKER.docx())
```

Initialization with params:

```
from fake import FAKER, FileSystemStorage

storage = FileSystemStorage()
docx_file = FAKER.docx_file(storage=storage)
```

**abspath**(*filename: str*) → str

Return absolute path.

**exists**(*filename: str*) → bool

Check if file exists.

**generate\_filename**(*extension: str, prefix: Optional[str] = None, basename: Optional[str] = None*) → str

Generate filename.

**relpath**(*filename: str*) → str

Return relative path.

**unlink**(*filename: str*) → None

Delete the file.

**write\_bytes**(*filename: str, data: bytes*) → int

Write bytes.

**write\_text**(*filename: str, data: str, encoding: Optional[str] = None*) → int

Write text.

```
class fake.GraphicPdfGenerator(faker: Faker)
```

Bases: object

Graphic PDF generatr.

Usage example:

```
from pathlib import Path
from fake import FAKER, GraphicPdfGenerator

Path("/tmp/graphic_example.pdf").write_bytes(
    FAKER.pdf(nb_pages=100, generator=GraphicPdfGenerator)
)
```

```
create(nb_pages: int = 1, image_size: Tuple[int, int] = (100, 100), image_color: Tuple[int, int, int] = (255, 0, 0), **kwargs) → bytes

image_color: Tuple[int, int, int]

image_size: Tuple[int, int]

nb_pages: int

class fake.LazyAttribute(func)
    Bases: object

class fake.LazyFunction(func)
    Bases: object

class fake.MetaData
    Bases: object

    add_content(content: Union[str, List[str]]) → None

    content: Optional[str]

class fake.ModelFactory(**kwargs)
    Bases: object

    ModelFactory.

    class Meta
        Bases: object

        get_or_create = ('id',)

    classmethod create(**kwargs)

    classmethod create_batch(count, **kwargs)

    classmethod save(instance)
        Save the instance.

class fake.PostSave(func, *args, **kwargs)
    Bases: object

    execute(instance)

class fake.PreSave(func, *args, **kwargs)
    Bases: object

    execute(instance)

class fake.SQLAlchemyModelFactory(**kwargs)
    Bases: ModelFactory

    SQLAlchemy ModelFactory.

    classmethod create(**kwargs)

    classmethod save(instance)
        Save the instance.
```

```
class fake.StringValue(value, *args, **kwargs)
```

Bases: str

**data:** Dict[str, Any]

```
class fake.SubFactory(factory_class, **kwargs)
```

Bases: object

```
class fake.TextPdfGenerator(faker: Faker)
```

Bases: object

Text PDF generatr.

Usage example:

```
from pathlib import Path
from fake import FAKER, TextPdfGenerator

Path("/tmp/text_example.pdf").write_bytes(
    FAKER.pdf(nb_pages=100, generator=TextPdfGenerator)
)
```

**create**(nb\_pages: Optional[int] = None, texts: Optional[List[str]] = None, metadata: Optional[MetaData] = None, \*\*kwargs) → bytes

**nb\_pages:** int

**texts:** List[str]

```
class fake.TortoiseModelFactory(**kwargs)
```

Bases: *ModelFactory*

Tortoise ModelFactory.

**classmethod create**(\*\*kwargs)

**classmethod save**(instance)

Save the instance.

**fake.fill\_dataclass**(dataclass\_type: Type) → Any

Fill dataclass with data.

**fake.fill\_pydantic\_model**(object\_type: Type) → Any

**fake.post\_save**(func)

**fake.pre\_save**(func)

**fake.provider**(func)

**fake.run\_async\_in\_thread**(coroutine: Coroutine) → Awaitable

Run an asynchronous coroutine in a separate thread.

**Parameters**

**coroutine** – An asyncio coroutine to be run.

**Returns**

The result of the coroutine.

**Return type**

Awaitable

`fake.trait(func)`

`fake.xor_transform(val: str, key: int = 10) → str`

Simple, deterministic string encoder/decoder.

Usage example:

```
val = "abcd"
encoded_val = xor_transform(val)
decoded_val = xor_transform(encoded_val)
```

## 11.12 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)



## PYTHON MODULE INDEX

f

fake, [74](#)



## A

abspath() (*fake.BaseStorage* method), 74  
 abspath() (*fake.FileSystemStorage* method), 81  
 add() (*fake.FileRegistry* method), 80  
 add\_content() (*fake.MetaData* method), 82  
 AuthorshipData (class in *fake*), 74

## B

BaseStorage (class in *fake*), 74  
 bmp() (*fake.Faker* method), 76  
 bmp\_file() (*fake.Faker* method), 76

## C

clean\_up() (*fake.FileRegistry* method), 80  
 content (*fake.MetaData* attribute), 82  
 create() (*fake.DjangoModelFactory* class method), 75  
 create() (*fake.DocxGenerator* method), 75  
 create() (*fake.GraphicPdfGenerator* method), 81  
 create() (*fake.ModelFactory* class method), 82  
 create() (*fake.SQLAlchemyModelFactory* class method), 82  
 create() (*fake.TextPdfGenerator* method), 83  
 create() (*fake.TortoiseModelFactory* class method), 83  
 create\_batch() (*fake.ModelFactory* class method), 82

## D

data (*fake.StringValue* attribute), 83  
 date() (*fake.Faker* method), 76  
 date\_time() (*fake.Faker* method), 77  
 DjangoModelFactory (class in *fake*), 74  
 docx() (*fake.Faker* method), 77  
 docx\_file() (*fake.Faker* method), 77  
 DocxGenerator (class in *fake*), 75

## E

email() (*fake.Faker* method), 77  
 execute() (*fake.PostSave* method), 82  
 execute() (*fake.PreSave* method), 82  
 exists() (*fake.BaseStorage* method), 74  
 exists() (*fake.FileSystemStorage* method), 81

## F

Factory (class in *fake*), 75  
 FactoryMethod (class in *fake*), 75  
 fake  
     module, 74  
 Faker (class in *fake*), 75  
 faker (*fake.Factory* property), 75  
 file\_name() (*fake.Faker* method), 77  
 FileRegistry (class in *fake*), 80  
 FileSystemStorage (class in *fake*), 80  
 fill\_dataclass() (in module *fake*), 83  
 fill\_pydantic\_model() (in module *fake*), 83  
 first\_name() (*fake.Faker* method), 77  
 first\_names (*fake.AuthorshipData* attribute), 74  
 first\_names() (*fake.Faker* method), 77

## G

generate\_filename() (*fake.BaseStorage* method), 74  
 generate\_filename() (*fake.FileSystemStorage* method), 81  
 generic\_file() (*fake.Faker* method), 77  
 get\_by\_alias() (*fake.Faker* static method), 77  
 get\_by\_uid() (*fake.Faker* static method), 77  
 get\_or\_create (*fake.ModelFactory.Meta* attribute), 82  
 gif() (*fake.Faker* method), 77  
 gif\_file() (*fake.Faker* method), 78  
 GraphicPdfGenerator (class in *fake*), 81

## I

image() (*fake.Faker* method), 78  
 image\_color (*fake.GraphicPdfGenerator* attribute), 82  
 image\_size (*fake.GraphicPdfGenerator* attribute), 82  
 image\_url() (*fake.Faker* method), 78  
 ipv4() (*fake.Faker* method), 78

## L

last\_name() (*fake.Faker* method), 78  
 last\_names (*fake.AuthorshipData* attribute), 74  
 last\_names() (*fake.Faker* method), 78  
 LazyAttribute (class in *fake*), 82  
 LazyFunction (class in *fake*), 82

load\_names() (*fake.Faker method*), 78  
load\_words() (*fake.Faker method*), 78

## M

MetaData (*class in fake*), 82  
ModelFactory (*class in fake*), 82  
ModelFactory.Meta (*class in fake*), 82  
module  
    fake, 74

## N

name() (*fake.Faker method*), 78  
names() (*fake.Faker method*), 78  
nb\_pages (*fake.GraphicPdfGenerator attribute*), 82  
nb\_pages (*fake.TextPdfGenerator attribute*), 83

## P

paragraph() (*fake.Faker method*), 78  
paragraphs() (*fake.Faker method*), 78  
pdf() (*fake.Faker method*), 78  
pdf\_file() (*fake.Faker method*), 78  
png() (*fake.Faker method*), 78  
png\_file() (*fake.Faker method*), 79  
post\_save() (*in module fake*), 83  
PostSave (*class in fake*), 82  
pre\_save() (*in module fake*), 83  
PreSave (*class in fake*), 82  
provider() (*in module fake*), 83  
pybool() (*fake.Faker method*), 79  
pydecimal() (*fake.Faker method*), 79  
pyfloat() (*fake.Faker method*), 79  
pyint() (*fake.Faker method*), 79  
pystr() (*fake.Faker method*), 79

## R

relpath() (*fake.BaseStorage method*), 74  
relpath() (*fake.FileSystemStorage method*), 81  
remove() (*fake.FileRegistry method*), 80  
run\_async\_in\_thread() (*in module fake*), 83

## S

save() (*fake.DjangoModelFactory class method*), 75  
save() (*fake.ModelFactory class method*), 82  
save() (*fake.SQLAlchemyModelFactory class method*),  
    82  
save() (*fake.TortoiseModelFactory class method*), 83  
search() (*fake.FileRegistry method*), 80  
sentence() (*fake.Faker method*), 79  
sentences() (*fake.Faker method*), 79  
slug() (*fake.Faker method*), 79  
slugs() (*fake.Faker method*), 79  
SQLAlchemyModelFactory (*class in fake*), 82  
StringValue (*class in fake*), 82

SubFactory (*class in fake*), 83  
svg() (*fake.Faker method*), 79  
svg\_file() (*fake.Faker method*), 79

## T

text() (*fake.Faker method*), 80  
TextPdfGenerator (*class in fake*), 83  
texts (*fake.TextPdfGenerator attribute*), 83  
texts() (*fake.Faker method*), 80  
TortoiseModelFactory (*class in fake*), 83  
trait() (*in module fake*), 83  
txt\_file() (*fake.Faker method*), 80

## U

unlink() (*fake.BaseStorage method*), 74  
unlink() (*fake.FileSystemStorage method*), 81  
url() (*fake.Faker method*), 80  
username() (*fake.Faker method*), 80  
usernames() (*fake.Faker method*), 80  
uuid() (*fake.Faker method*), 80  
uuids() (*fake.Faker method*), 80

## W

word() (*fake.Faker method*), 80  
words() (*fake.Faker method*), 80  
write\_bytes() (*fake.BaseStorage method*), 74  
write\_bytes() (*fake.FileSystemStorage method*), 81  
write\_text() (*fake.BaseStorage method*), 74  
write\_text() (*fake.FileSystemStorage method*), 81

## X

xor\_transform() (*in module fake*), 84