Welcome to Faker’s documentation!

Faker is a Python package that generates fake data for you. Whether you need to bootstrap your database, create good-looking XML documents, fill-in your persistence to stress test it, or anonymize data taken from a production service, Faker is for you.

Faker is heavily inspired by PHP Faker, Perl Faker, and by Ruby Faker.

```
_|_|_|_|          _|
_|        _|_|_|  _|  _|      _|_|    _|  _|_|
_|_|_|  _|    _|  _|_|      _|_|_|_|  _|_|
_|      _|    _|  _|  _|    _|        _|
_|        _|_|_|  _|    _|    _|_|_|  _|
```

version v18.6.2 | Python Tests passing | coverage 96% | license MIT

Compatibility

Starting from version `4.0.0`, `Faker` dropped support for Python 2 and from version `5.0.0` only supports Python 3.7 and above. If you still need Python 2 compatibility, please install version `3.0.1` in the meantime, and please consider updating your codebase to support Python 3 so you can enjoy the latest features `Faker` has to offer. Please see the extended docs for more details, especially if you are upgrading from version `2.0.4` and below as there might be breaking changes.

This package was also previously called `fake-factory` which was already deprecated by the end of 2016, and much has changed since then, so please ensure that your project and its dependencies do not depend on the old package.

Basic Usage

Install with pip:

```
pip install Faker
```

Use `faker.Faker()` to create and initialize a faker generator, which can generate data by accessing properties named after the type of data you want.

```
from faker import Faker
fake = Faker()

fake.name()
# 'Lucy Cechtelar'

fake.address()
# '426 Jordy Lodge
#  Cartwrightshire, SC 88120-6700'

fake.text()
# 'Sint velit eveniet. Rerum atque repellat voluptatem quia rerum. Numquam excepturi
#  beatae sint laudantium consequatur. Magni occaecati itaque sint et sit tempore. Nesciunt
#  amet quidem. Iusto deleniti cum autem ad quia aperiam.
#  A consectetur quos aliquam. In iste aliquid et aut similique suscipit. Consequatur qui
#  quaerat iste minus hic expedita. Consequuntur error magni et laboriosam. Aut aspernatur
#  voluptatem sit aliquam. Dolores voluptatum est.
#  Aut molestias et maxime. Fugit autem facilis quos vero. Eius quibusdam possimus est.
#  Ea quaerat et quisquam. Deleniti sunt quam. Adipisci consequatur id in occaecati.
#  Et sint et. Ut ducimus quod nemo ab voluptatum.'
```

Each call to method `fake.name()` yields a different (random) result. This is because faker forwards `faker.Generator.method_name()` calls to `faker.Generator.format(method_name)`.

```
for _ in range(10):
  print(fake.name())

# 'Adaline Reichel'
# 'Dr. Santa Prosacco DVM'
# 'Noemy Vandervort V'
# 'Lexi O'Conner'
# 'Gracie Weber'
# 'Roscoe Johns'
# 'Emmett Lebsack'
# 'Keegan Thiel'
# 'Wellington Koelpin II'
# 'Ms. Karley Kiehn V'
```

Pytest fixtures

`Faker` also has its own `pytest` plugin which provides a `faker` fixture you can use in your tests. Please check out the *pytest fixture docs* to learn more.

Providers

Each of the generator properties (like `name`, `address`, and `lorem`) are called “fake”. A faker generator has many of them, packaged in “providers”.

```python
from faker import Faker
from faker.providers import internet

fake = Faker()
fake.add_provider(internet)

print(fake.ipv4_private())
```

Check the extended docs for a list of bundled providers and a list of community providers.

Localization

`faker.Faker` can take a locale as an argument, to return localized data. If no localized provider is found, the factory falls back to the default LCID string for US english, ie: `en_US`.

```python
from faker import Faker
fake = Faker('it_IT')
for _ in range(10):
    print(fake.name())

# 'Elda Palumbo'
# 'Pacifico Giordano'
# 'Sig. Avide Guerra'
# 'Yago Amato'
# 'Eustachio Messina'
# 'Dott. Violante Lombardo'
# 'Sig. Alighieri Monti'
# 'Costanzo Costa'
# 'Nazzareno Barbieri'
# 'Max Coppola'
```

`faker.Faker` also supports multiple locales. New in v3.0.0.

```python
from faker import Faker
fake = Faker(['it_IT', 'en_US', 'ja_JP'])
for _ in range(10):
    print(fake.name())

# 鈴木 陽一
# Leslie Moreno
# Emma Williams
# 渡辺 裕美子
# Marcantonio Galuppi
# Martha Davis
# Kristen Turner
# 中津川 春香
# Ashley Castillo
# 山田 桃子
```

You can check available Faker locales in the source code, under the providers package. The localization of Faker is an ongoing process, for which we need your help. Please don't hesitate to create a localized provider for your own locale and submit a Pull Request (PR).

Optimizations

The Faker constructor takes a performance-related argument called `use_weighting`. It specifies whether to attempt to have the frequency of values match real-world frequencies (e.g. the English name Gary would be much more frequent than the name Lorimer). If `use_weighting` is `False`, then all items have an equal chance of being selected, and the selection process is much faster. The default is `True`.

Command line usage

When installed, you can invoke faker from the command-line:

```
faker [-h] [--version] [-o output]
      [-l {bg_BG,cs_CZ,...,zh_CN,zh_TW}]
      [-r REPEAT] [-s SEP]
      [-i {package.containing.custom_provider otherpkg.containing.custom_provider}]
      [fake] [fake argument [fake argument ...]]
```

Where:

- `faker`: is the script when installed in your environment, in development you could use `python -m faker` instead
- `-h`, `--help`: shows a help message
- `--version`: shows the program's version number
- `-o FILENAME`: redirects the output to the specified filename
- `-l {bg_BG,cs_CZ,...,zh_CN,zh_TW}`: allows use of a localized provider
- `-r REPEAT`: will generate a specified number of outputs
- `-s SEP`: will generate the specified separator after each generated output
- `-i {my.custom_provider other.custom_provider}` list of additional custom providers to use. Note that is the import path of the package containing your Provider class, not the custom Provider class itself.
- `fake`: is the name of the fake to generate an output for, such as `name`, `address`, or `text`
- `[fake argument ...]`: optional arguments to pass to the fake (e.g. the profile fake takes an optional list of comma separated field names as the first argument)

Examples:

```
$ faker address
968 Bahringer Garden Apt. 722
Kristinaland, NJ 09890

$ faker -l de_DE address
Samira-Niemeier-Allee 56
94812 Biedenkopf

$ faker profile ssn,birthdate
{'ssn': '628-10-1085', 'birthdate': '2008-03-29'}

$ faker -r=3 -s=";" name
Willam Kertzmann;
Josiah Maggio;
Gayla Schmitt;
```

How to create a Provider

```python
from faker import Faker
fake = Faker()

# first, import a similar Provider or use the default one
from faker.providers import BaseProvider

# create new provider class
class MyProvider(BaseProvider):
    def foo(self) -> str:
        return 'bar'

# then add new provider to faker instance
fake.add_provider(MyProvider)

# now you can use:
fake.foo()
# 'bar'
```

How to create a Dynamic Provider

Dynamic providers can read elements from an external source.

```python
from faker import Faker
from faker.providers import DynamicProvider

medical_professions_provider = DynamicProvider(
     provider_name="medical_profession",
     elements=["dr.", "doctor", "nurse", "surgeon", "clerk"],
)

fake = Faker()

# then add new provider to faker instance
fake.add_provider(medical_professions_provider)

# now you can use:
fake.medical_profession()
# 'dr.'
```

How to customize the Lorem Provider

You can provide your own sets of words if you don’t want to use the default lorem ipsum one. The following example shows how to do it with a list of words picked from cakeipsum :

```
from faker import Faker
fake = Faker()

my_word_list = [
'danish','cheesecake','sugar',
'Lollipop','wafer','Gummies',
'sesame','Jelly','beans',
'pie','bar','Ice','oat' ]

fake.sentence()
# 'Expedita at beatae voluptatibus nulla omnis.'

fake.sentence(ext_word_list=my_word_list)
# 'Oat beans oat Lollipop bar cheesecake.'
```

How to use with Factory Boy

Factory Boy already ships with integration with `Faker`. Simply use the `factory.Faker` method of `factory_boy`:

```
import factory
from myapp.models import Book

class BookFactory(factory.Factory):
    class Meta:
        model = Book

    title = factory.Faker('sentence', nb_words=4)
    author_name = factory.Faker('name')
```

Accessing the *random* instance

The `.random` property on the generator returns the instance of `random.Random` used to generate the values:

```
from faker import Faker
fake = Faker()
fake.random
fake.random.getstate()
```

By default all generators share the same instance of `random.Random`, which can be accessed with `from faker.generator import random`. Using this may be useful for plugins that want to affect all faker instances.

Unique values

Through use of the `.unique` property on the generator, you can guarantee that any generated values are unique for this specific instance.

```
from faker import Faker
fake = Faker()
names = [fake.unique.first_name() for i in range(500)]
assert len(set(names)) == len(names)
```

Calling `fake.unique.clear()` clears the already seen values. Note, to avoid infinite loops, after a number of attempts to find a unique value, Faker will throw a `UniquenessException`. Beware of the birthday paradox, collisions are more likely than you'd think.

```
from faker import Faker

fake = Faker()
for i in range(3):
     # Raises a UniquenessException
     fake.unique.boolean()
```

In addition, only hashable arguments and return values can be used with `.unique`.

Seeding the Generator

When using Faker for unit testing, you will often want to generate the same data set. For convenience, the generator also provide a `seed()` method, which seeds the shared random number generator. Seed produces the same result when the same methods with the same version of faker are called.

```
from faker import Faker
fake = Faker()
Faker.seed(4321)

print(fake.name())
# 'Margaret Boehm'
```

Each generator can also be switched to its own instance of `random.Random`, separate to the shared one, by using the `seed_instance()` method, which acts the same way. For example:

```
from faker import Faker
fake = Faker()
fake.seed_instance(4321)

print(fake.name())
# 'Margaret Boehm'
```

Please note that as we keep updating datasets, results are not guaranteed to be consistent across patch versions. If you hardcode results in your test, make sure you pinned the version of `Faker` down to the patch number.

If you are using `pytest`, you can seed the `faker` fixture by defining a `faker_seed` fixture. Please check out the *pytest fixture docs* to learn more.

Tests

Run tests:

```
$ tox
```

Write documentation for the providers of the default locale:

```
$ python -m faker > docs.txt
```

Write documentation for the providers of a specific locale:

```
$ python -m faker --lang=de_DE > docs_de.txt
```

Contribute

Please see CONTRIBUTING.

License

Faker is released under the MIT License. See the bundled LICENSE file for details.

Credits

- FZaninotto / PHP Faker
- Distribute
- Buildout
- modern-package-template

Contents

- Locale da_DK
- Locale de
- Locale de_AT
- Locale de_CH
- Locale de_DE
- Locale dk_DK
- Locale el_CY
- Locale el_GR
- Locale en
- Locale en_AU
- Locale en_CA
- Locale en_GB
- Locale en_IE
- Locale en_IN
- Locale en_NZ
- Locale en_PH
- Locale en_TH
- Locale en_US
- Locale es
- Locale es_AR
- Locale es_CA
- Locale es_CL
- Locale es_CO
- Locale es_ES
- Locale es_MX
- Locale et_EE
- Locale fa_IR
- Locale fi_FI
- Locale fil_PH
- Locale fr_BE
- Locale fr_CA
- Locale fr_CH
- Locale fr_FR
- Locale fr_QC
- Locale ga_IE
- Locale he_IL
- Locale hi_IN
- Locale hr_HR
- Locale hu_HU
- Locale hy_AM
- Locale id_ID
- Locale it_CH
- Locale it_IT
- Locale ja_JP
- Locale ka_GE

Indices and tables

- Index

- Module Index
- Search Page