Pydantic

CI passing | coverage 96%

pypi v2.9.2 | conda | conda-forge v2.9.2 | downloads/month 284M

license MIT

Documentation for version: v2.9.2.

Pydantic is the most widely used data validation library for Python.

Fast and extensible, Pydantic plays nicely with your linters/IDE/brain. Define how data should be in pure, canonical Python 3.8+; validate it with Pydantic.

Migrating to Pydantic V2

Using Pydantic V1? See the Migration Guide for notes on upgrading to Pydantic V2 in your applications!

Pydantic Example

```python
from datetime import datetime
from typing import Tuple

from pydantic import BaseModel


class Delivery(BaseModel):
    timestamp: datetime
    dimensions: Tuple[int, int]


m = Delivery(timestamp='2020-01-02T03:04:05Z', dimensions=['10', '20'])
print(repr(m.timestamp))
#> datetime.datetime(2020, 1, 2, 3, 4, 5, tzinfo=TzInfo(UTC))
print(m.dimensions)
#> (10, 20)
```

Why is Pydantic named the way it is?

The name "Pydantic" is a portmanteau of "Py" and "pedantic." The "Py" part indicates that the library is associated with Python, and "pedantic" refers to the library's meticulous approach to data validation and type enforcement.

Combining these elements, "Pydantic" describes our Python library that provides detail-oriented, rigorous data validation.

We're aware of the irony that Pydantic V1 was not strict in its validation, so if we're being pedantic, "Pydantic" was a misnomer until V2 😉.

Why use Pydantic?

- **Powered by type hints** — with Pydantic, schema validation and serialization are controlled by type annotations; less to learn, less code to write, and integration with your IDE and static analysis tools. Learn more…
- **Speed** — Pydantic's core validation logic is written in Rust. As a result, Pydantic is among the fastest data validation libraries for Python. Learn more…
- **JSON Schema** — Pydantic models can emit JSON Schema, allowing for easy integration with other tools. Learn more…
- **Strict** and **Lax** mode — Pydantic can run in either strict mode (where data is not converted) or lax mode where Pydantic tries to coerce data to the correct type where appropriate. Learn more…
- **Dataclasses**, **TypedDicts** and more — Pydantic supports validation of many standard library types including `dataclass` and `TypedDict`. Learn more…
- **Customisation** — Pydantic allows custom validators and serializers to alter how data is processed in many powerful ways. Learn more…
- **Ecosystem** — around 8,000 packages on PyPI use Pydantic, including massively popular libraries like *FastAPI*, *huggingface*, *Django Ninja*, *SQLModel*, & *LangChain*. Learn more…
- **Battle tested** — Pydantic is downloaded over 70M times/month and is used by all FAANG companies and 20 of the 25 largest companies on NASDAQ. If you're trying to do something with Pydantic, someone else has probably already done it. Learn more…

Installing Pydantic is as simple as: `pip install pydantic`

Pydantic examples

To see Pydantic at work, let's start with a simple example, creating a custom class that inherits from `BaseModel`:

Validation Successful

```python
from datetime import datetime

from pydantic import BaseModel, PositiveInt


class User(BaseModel):
    id: int  1
    name: str = 'John Doe'  2
    signup_ts: datetime | None  3
    tastes: dict[str, PositiveInt]  4


external_data = {
    'id': 123,
    'signup_ts': '2019-06-01 12:22',  5
    'tastes': {
        'wine': 9,
        b'cheese': 7,  6
        'cabbage': '1',  7
    },
}

user = User(**external_data)  8

print(user.id)  9
#> 123
print(user.model_dump())  10
"""
{
    'id': 123,
    'name': 'John Doe',
    'signup_ts': datetime.datetime(2019, 6, 1, 12, 22),
    'tastes': {'wine': 9, 'cheese': 7, 'cabbage': 1},
}
"""
```

1. `id` is of type `int`; the annotation-only declaration tells Pydantic that this field is required. Strings, bytes, or floats will be coerced to integers if possible; otherwise an exception will be raised.
2. `name` is a string; because it has a default, it is not required.
3. `signup_ts` is a `datetime` field that is required, but the value `None` may be provided; Pydantic will process either a Unix timestamp integer (e.g. `1496498400`) or a string representing the date and time.
4. `tastes` is a dictionary with string keys and positive integer values. The `PositiveInt` type is shorthand for `Annotated[int, annotated_types.Gt(0)]`.
5. The input here is an ISO 8601 formatted datetime, but Pydantic will convert it to a `datetime` object.
6. The key here is `bytes`, but Pydantic will take care of coercing it to a string.

7. Similarly, Pydantic will coerce the string `'1'` to the integer `1`.
8. We create instance of `User` by passing our external data to `User` as keyword arguments.
9. We can access fields as attributes of the model.
10. We can convert the model to a dictionary with `model_dump()`.

If validation fails, Pydantic will raise an error with a breakdown of what was wrong:

Validation Error

```
# continuing the above example...

from datetime import datetime
from pydantic import BaseModel, PositiveInt, ValidationError


class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]


external_data = {'id': 'not an int', 'tastes': {}}  1

try:
    User(**external_data)  2
except ValidationError as e:
    print(e.errors())
    """
    [
        {
            'type': 'int_parsing',
            'loc': ('id',),
            'msg': 'Input should be a valid integer, unable to parse string as
an integer',
            'input': 'not an int',
            'url': 'https://errors.pydantic.dev/2/v/int_parsing',
        },
        {
            'type': 'missing',
            'loc': ('signup_ts',),
            'msg': 'Field required',
            'input': {'id': 'not an int', 'tastes': {}},
            'url': 'https://errors.pydantic.dev/2/v/missing',
        },
    ]
    """
```

1. The input data is wrong here — `id` is not a valid integer, and `signup_ts` is missing.
2. Trying to instantiate `User` will raise a `ValidationError` with a list of errors.

Who is using Pydantic?

Hundreds of organisations and packages are using Pydantic. Some of the prominent companies and organizations around the world who are using Pydantic include:

For a more comprehensive list of open-source projects using Pydantic see the list of dependents on github, or you can find some awesome projects using Pydantic in awesome-pydantic.

Was this page helpful?