# Data Classes

**Пример — когда нужны dataclass**
У нас есть некоторая функция, которая обращается к ресурсу httpbin.org и возвращает тело ответа в словаре:

```
import asyncio
import aiohttp

async def req():
    async with aiohttp.ClientSession() as session:
        async with session.get('http://httpbin.org/get') as resp:
            return await resp.json()

asyncio.run(req_basic_auth())
```

Без запуска кода или без документации портала httpbin нельзя определить, что возвращает API. Для понимания кода было бы очень удобно знать структуру ответа API, а в идеале иметь объект в Python, который её описывает.

Для таких целей в Python есть dataclass. Они позволяют удобно описывать и структурировать данные без оперирования словарями.
Перед использованием dataclass следует ознакомиться с синтаксисом typing в Python.

**Пример — как можно описать структуру ответа метода API/get**

```
# неструктурированный ответ API
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Host': 'httpbin.org',
'User-Agent': 'Python/3.9 aiohttp/3.7.4', 'X-Amzn-Trace-Id': 'Root=1-617aba18-145c6e6e52f91146667366666'},
'origin': '95.84.229.15', 'url': 'http://httpbin.org/get'}

# описанный ответ с помощью dataclass
@dataclass
class Headers:
    accept: str
    accept_encoding: str
    host: str
    user_agent: str
    x_amzn_trace_id: Optional[str] = None

@dataclass
class GetResponse:
    args: dict
    headers: Headers
    origin: str
    url: str
```

Код можно переписать следующим образом:

```
async def req() -> GetResponse:
    async with aiohttp.ClientSession() as session:
        async with session.get('http://httpbin.org/get') as resp:
            data = await resp.json()
            headers_dict = data.get('headers', {})
            headers = Headers(
                accept=headers_dict['Accept'],
                accept_encoding=headers_dict['Accept-Encoding'],
                host=headers_dict['Host'],
                user_agent=headers_dict['User-Agent'],
                x_amzn_trace_id=headers_dict['X-Amzn-Trace-Id']
            )
            res = GetResponse(
                args=data['args'],
                headers=headers,
                origin=data['origin'],
                url=data['url']
            )
            return res
```

Код стал гораздо понятнее. Теперь без дополнительных материалов можно узнать, какие данные возвращает API. Но при этом мы добавили себе дополнительную работу по описанию ответа в dataclass и по созданию объекта GetResponse из словаря data (mapping, или маппинга). А это достаточно рутинная работа.

От описания ответа никуда не денешься, но для процесса маппинга и валидации есть библиотека https://marshmallow.readthedocs.io/en/stable/. А для нее есть дополнительная библиотека, которая связывает marshmallow и dataclass. Неожиданно: она называется marshmallow_dataclass.

Marshmallow — библиотека, которая позволяет проверять данные на корректность. Для этого опишите ожидаемую структуру, и она проверит корректность переданных данных:

```
from marshmallow import Schema, fields

class UserSchema(Schema):
    name = fields.Str()
    email = fields.Email()
    created_at = fields.DateTime()

user_data = {
    "created_at": "2014-08-11T05:26:03.869245",
    "email": "ken@yahoo.com",
    "name": "Ken",
}
schema = UserSchema()
result = schema.load(user_data)
print(result)
```

Библиотека marshmallow-dataclass позволяет не описывать отдельно схемы (UserSchema), а использовать dataclass.

Так выглядит код после добавления marshmallow-dataclass:

```
from dataclasses import field
from typing import Optional, ClassVar, Type

from marshmallow_dataclass import dataclass
from marshmallow import Schema, EXCLUDE

@dataclass
class Headers:
    accept: str = field(metadata={'data_key': 'Accept'})
    accept_encoding: str = field(metadata={'data_key': 'Accept-Encoding'})
    host: str = field(metadata={'data_key': 'Host'})
    user_agent: str = field(metadata={'data_key': 'User-Agent'})
    x_amzn_trace_id: Optional[str] = field(default=None, metadata={'data_key': 'X-Amzn-Trace-Id'})

    class Meta:
        unknown = EXCLUDE

@dataclass
class GetResponse:
    args: dict
    headers: Headers
    origin: str
    url: str

    Schema: ClassVar[Type[Schema]] = Schema

    class Meta:
        unknown = EXCLUDE
```

Пояснения:

- нужно использовать dataclass из библиотеки **marshmallow_dataclass**, а не стандартный: **from marshmallow_dataclass import dataclass**
- в class Meta описываются параметры для marshmallow схемы
- конструкция **field(metadata={'data_key': 'Accept-Encoding'})** используется тогда, когда название поля в исходных данных нельзя использовать как название переменной в Python. Например, как в случае с **Accept-Encoding**
- строку **Schema: ClassVar[Type[Schema]] = Schema** нужно добавлять, чтобы было удобно вызывать процесс конвертации данных: **GetResponse.Schema().load(data)**. В результате мы сразу получим объект **GetResponse**. Но можно обойтись без этого синтаксиса и создать схему руками:

```
@dataclass
class GetResponse:
    args: dict
    headers: Headers
    origin: str
    url: str

    class Meta:
        unknown = EXCLUDE

GetResponseSchema = marshmallow_dataclass.class_schema(GetResponse)
GetResponseSchema().load(data)

# <>

@dataclass
class GetResponse:
    args: dict
    headers: Headers
    origin: str
    url: str

    Schema: ClassVar[Type[Schema]] = Schema

    class Meta:
        unknown = EXCLUDE

GetResponse.Schema().load(data)
```

При использовании dataclass финальный код функции req() сократится до такого:

```
async def req() -> GetResponse:
    async with aiohttp.ClientSession() as session:
        async with session.get('http://httpbin.org/get') as resp:
            data = await resp.json()
            res = GetResponse.Schema().load(data)
            print(res)
            return res

# >>> print(res)
# GetResponse(args={}, headers=Headers(accept='*/*', accept_encoding='gzip, deflate', host='httpbin.org',
user_agent='Python/3.9 aiohttp/3.7.4', x_amzn_trace_id='Root=1-617abf84-104fabc2421d172302744b83'),
origin='95.84.229.15', url='http://httpbin.org/get')
```

Результат: мы описали формат ответа от API сервиса httpbin.org, что позволит в дальнейшей разработке проще и быстрее дорабатывать код.