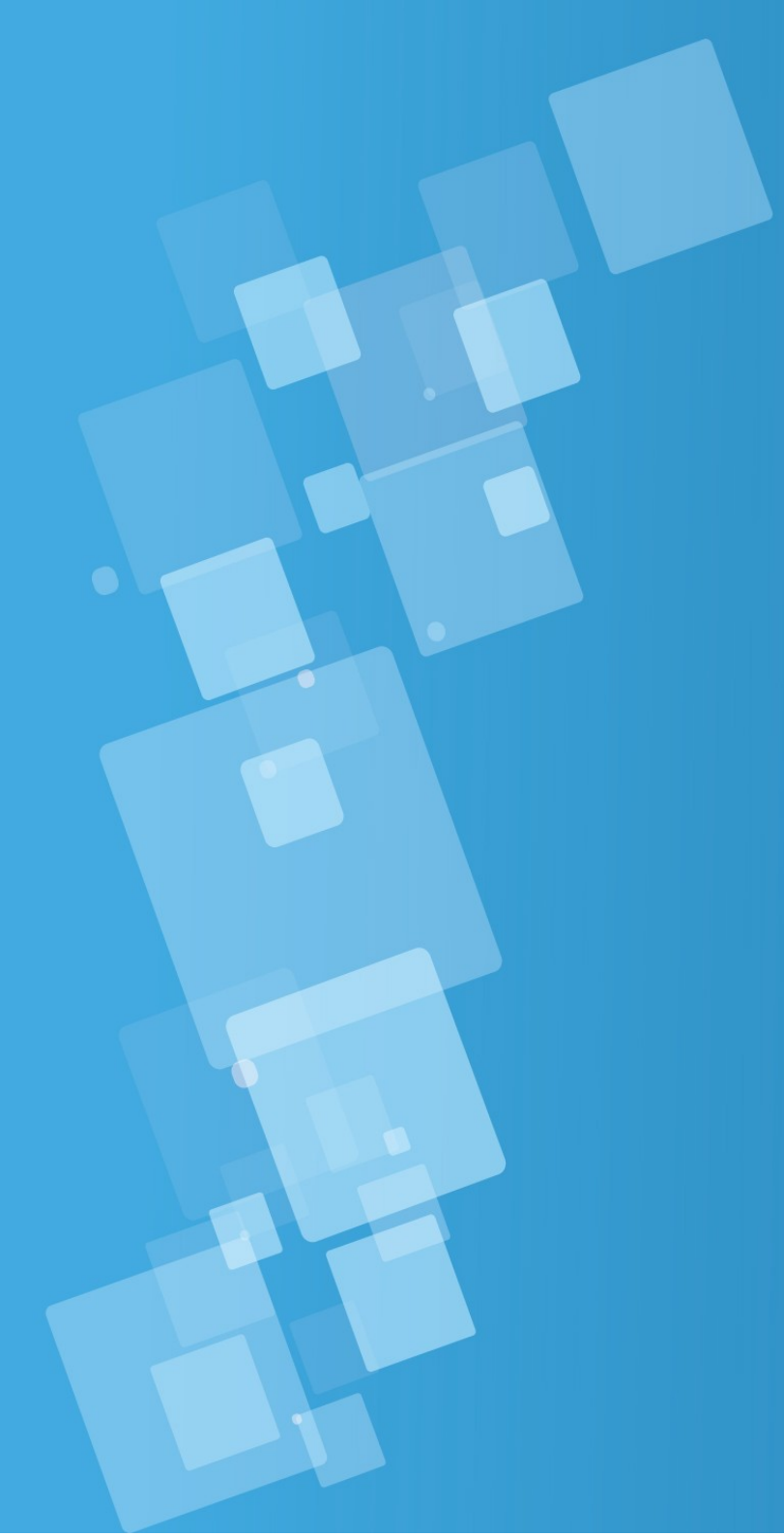


Type hinting

Pydantic

Примечание.

Среда выполнения Python не поддерживает анатации к типам и функциям. Они могут использоваться сторонними инструментами, такими как средства проверки типов, IDE, линтеры и т. д.

Подсказка типа(type hinting) — это формальное решение для явного указания типа значения в вашем коде Python.

Type hinting

```
def greeting(name: str, num: int) -> str:
    return 'Hello ' + name * num

print(greeting(" a", 3))
```

Рекомендации PEP8

- Использовать двоеточие после определения переменной а затем один пробел `text: str`
- Использовать пространство в один пробел между оператором присваения: `align: bool = True`
- Использовать пространство в один пробел между операторо стрелка: def headline(...) -> str

Синонимы для типов

Синоним типа определяется путем присвоения типа синониму. В этом примере Vector и list[float] будут рассматриваться как взаимозаменяемые синонимы:

```
Vector = list[float]

def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]

# typechecks; a list of floats qualifies as a Vector.
new_vector = scale(2.0, [1.0, -4.2, 5.4])
```

Создание своих типов

```
from typing import NewType

UserId = NewType('UserId', int)
some_id = UserId(524313)

def get_user_name(user_id: UserId)  -> str:
    return "pass"

# typechecks
user_a = get_user_name(UserId(42351))

# does not typecheck; an int is not a UserId
user_b = get_user_name(-1)


# 'output' is of type 'int', not 'UserId'

output = UserId(23413) + UserId(54341)
```

Функции обратного вызова

```
from collections.abc import Callable

def feeder(get_next_item: Callable[[], str]) -> None:
    # Body

def async_query(on_success: Callable[[int], None],
                on_error: Callable[[int, Exception], None]) -> None:
    # Body
async def on_update(value: str) -> None:
    # Body

callback: Callable[[str], Awaitable[None]] = on_update
```

А что такое функция обратного вызова ?

Callback function (функция обратного вызова) – это функция, переданная в другую функцию (при вызове) в качестве значения аргумента, которая будет вызвана в теле другой функции с заранее известными аргументами.

Пример 1 collback function

```
def filter_(callback, iterable):
    result = []
    for i in iterable:
        if callback(i):
            result.append(i)
    return result

def txt_files(value):
    return value.endswith('.txt')
lst = [ '1.txt', '2.html', '', '3.mp3', '8.txt']
txt = filter_(txt_files, lst)
print (txt)
# => ['1.txt', '8.txt']
```

Тип Any

```
from typing import Any

a: Any = None
a = []          # OK
a = 2           # OK

s: str = ''
s = a           # OK !!!

def foo(item: Any) -> int:
    # Typechecks; 'item' could be any type,
    # and that type might have a 'bar' method
    item.bar()
    ...
```

Обратите внимание, что при присвоении значения типа Any более точному типу проверка типов не выполняется.

Тип NoReturn

```
from typing import NoReturn

def stop() -> NoReturn:
    raise RuntimeError('no way')
```

Специальный тип, указывающий, что функция никогда не возвращает значение. Например:

Тип Literal

```
from typing import Any, Literal

def validate_simple(data: Any) -> Literal[True]:  # always returns True
    ...
    return  True

MODE = Literal['r', 'rb', 'w', 'wb']
def open_helper(file: str, mode: MODE) -> str:
    ...
    return  'text'

open_helper('/some/path', 'r')  # Passes type check
open_helper('/other/path', 'typo')  # Error in type checker
```

Анатация классов

```
class User: ...
class BasicUser(User): ...
class ProUser(User): ...
class TeamUser(User): ...

# Accepts User, BasicUser, ProUser, TeamUser, ...
def make_new_user(user_class: User) -> User:
    # ...
    return user_class()
```

Заключение

Подсказки типов – это фомальность и она не влечет за собой каких либо последствий со стороны среды выполнения.

Документация:

https://docs.python.org/3/library/typing.html

Конанический путь проверки

Проверка объекта на принадлежность классу.

```
obj = 'text'
if isinstance(obj, str):
    print(True)
```

Или

```
if not type(o)is str: raise TypeError
```

Или

```
if not issubclass(type(obj), str): return
```

https://pydantic-docs.helpmanual.io/

Pydantic - это библиотека, с помощью которой можно парсить данные и выполнять валидацию.

Модель Pydantic

```
from pydantic import BaseModel
# Создаем модели Pydentic

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


external_json = """{ "id": 123,  "name": "Peter" } """

user = User.parse_raw(external_json)


#либо

external_json = { "id": 123,  "name": "Peter" }

user = User(**external_json)
```

Структуры в C++

Создание структуры:

```
struct person {
    unsigned short age;
    char name[255];
    char sex;
};
```

Объявление переменных:

```
struct person student, people[52], *man;
```

Пример 2

```
from datetime import datetime
from typing import List, Optional
from pydantic import BaseModel


class User(BaseModel):
    id: int
    name = 'John Doe'
    signup_ts: Optional[datetime] = None
    friends: List[int] = []


external_data = {
    'id': '123',
    'signup_ts': '2019-06-01 12:22',
    'friends': [1, 2, '3'],
}
user = User(**external_data)
print(user.id)
```

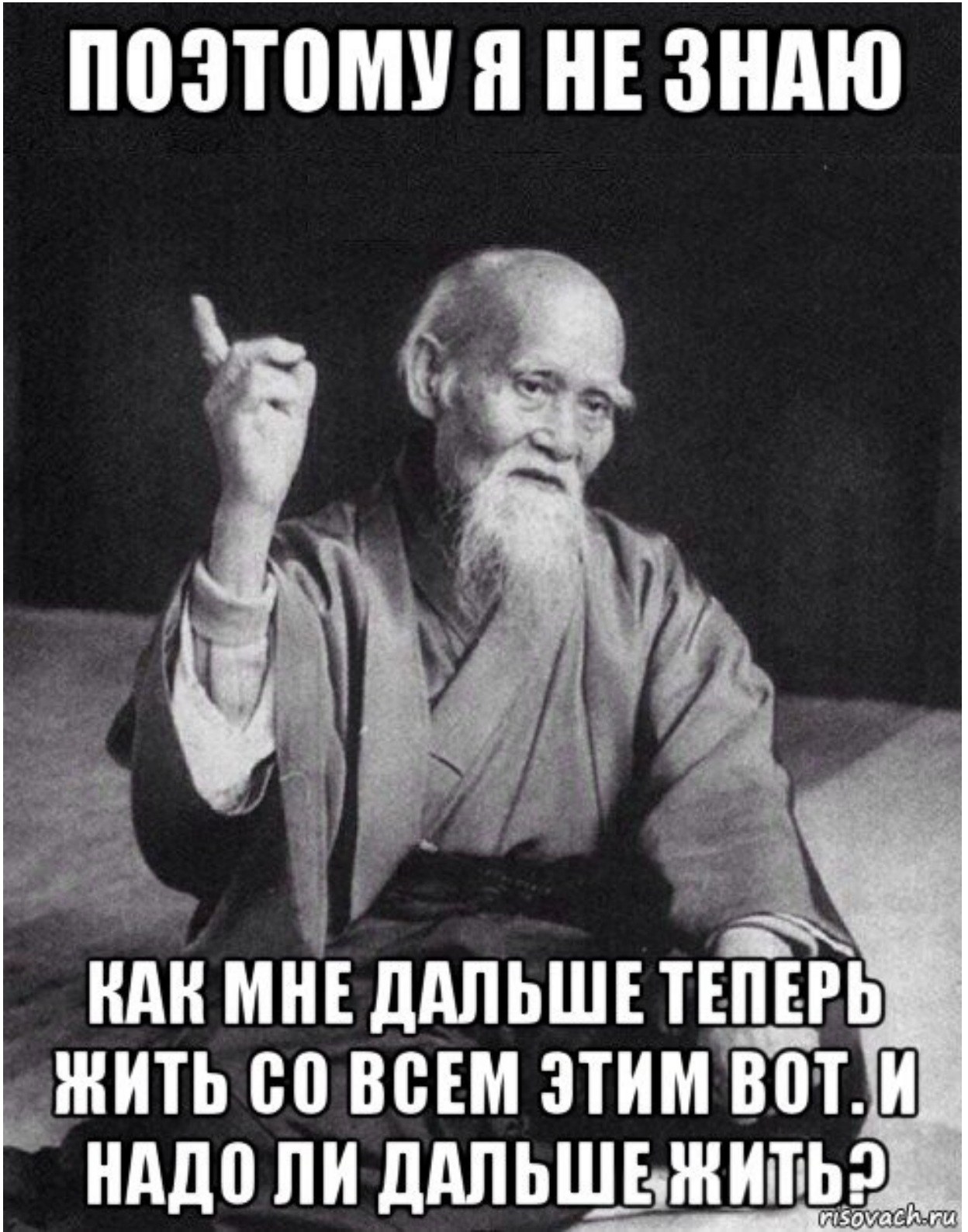
Определение

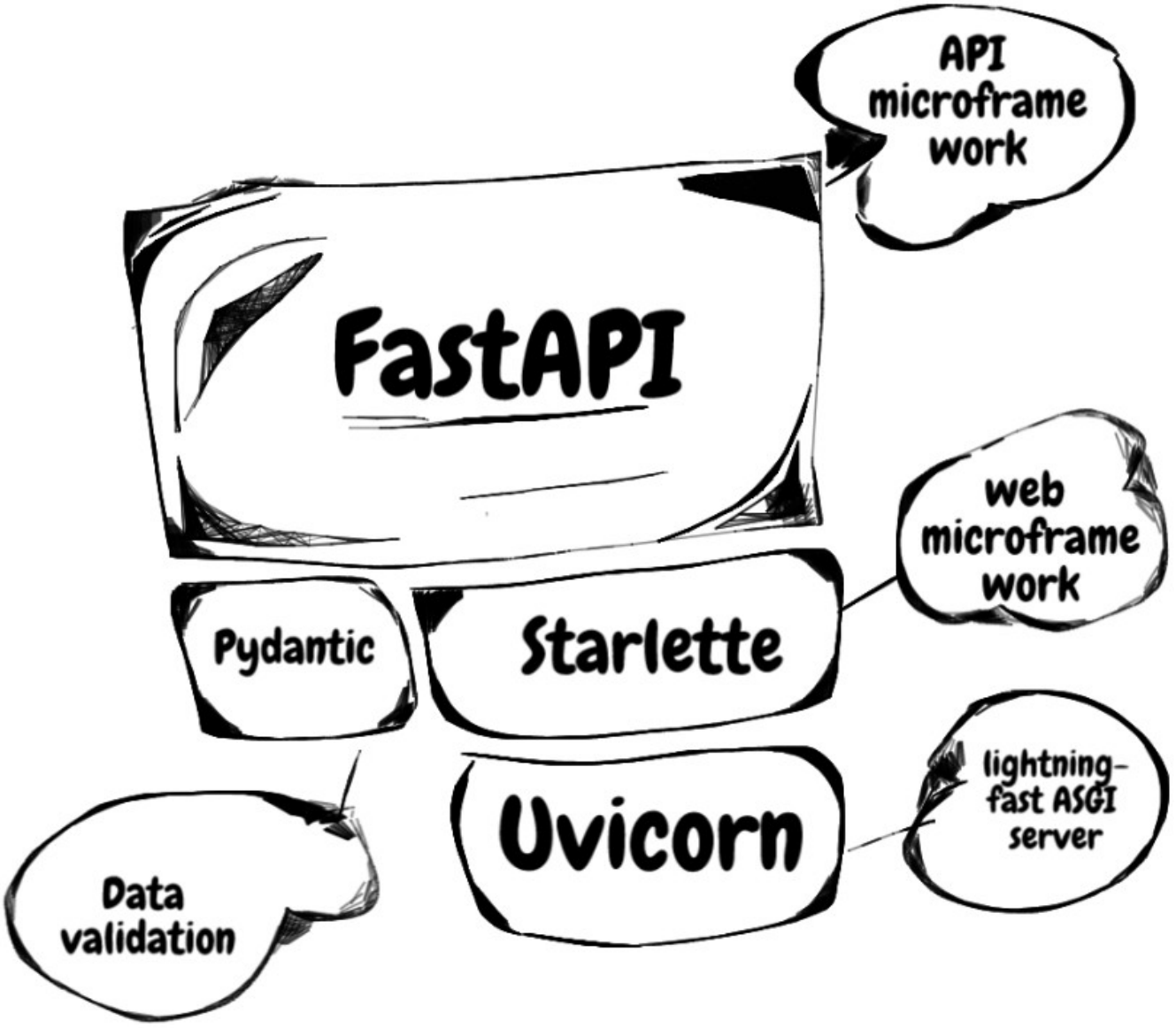
Валидация данных (англ. Data validation) — это процесс проверки данных различных типов по критериям корректности и полезности для конкретного применения. Валидация данных проводится, как правило, после выполнения операций ETL и для подтверждения корректности результатов работы моделей машинного обучения (предиктов).

```python
from pydantic import BaseModel, ValidationError, validator

class UserModel(BaseModel):
    name: str
    username: str
    password1: str
    password2: str

    @validator('name')
    def name_must_contain_space(cls, v):
        if ' ' not in v:
            raise ValueError('must contain a space')
        return v.title()

    @validator('password2')
    def passwords_match(cls, v, values, **kwargs):
        if 'password1' in values and v != values['password1']:
            raise ValueError('passwords do not match')
        return v

    @validator('username')
    def username_alphanumeric(cls, v):
        assert v.isalnum(), 'must be alphanumeric'
        return v

user = UserModel(
    name = 'samuel colvin',
    username = 'scolvin',
    password1 = 'zxcvbn',
    password2 = 'zxcvbn',
)
```

Запуск с ошибкой

```
try:
    UserModel(
        name = 'samuel',
        username = 'scolvin',
        password1 = 'zxcvbn',
        password2 = 'zxcvbn2',
    )
except ValidationError as e:
    print(e)
    """
    2 validation errors for UserModel
    name
      must contain a space (type = value_error)
    password2
      passwords do not match (type=value_error)
    """
```


ПОЭТОМУ Я НЕ ЗНАЮ
КАК МНЕ ДАЛЬШЕ ТЕПЕРЬ ЖИТЬ СО ВСЕМ ЭТИМ ВОТ. И НАДО ЛИ ДАЛЬШЕ ЖИТЬ?
risovach.ru


API microframe work
FastAPI
web microframe work
Pydantic
Starlette
Uvicorn
lightning-fast ASGI server
Data validation