Python Intermediate

Piotr Kardaś, 2021

Piotr Kardaś

Software Engineer

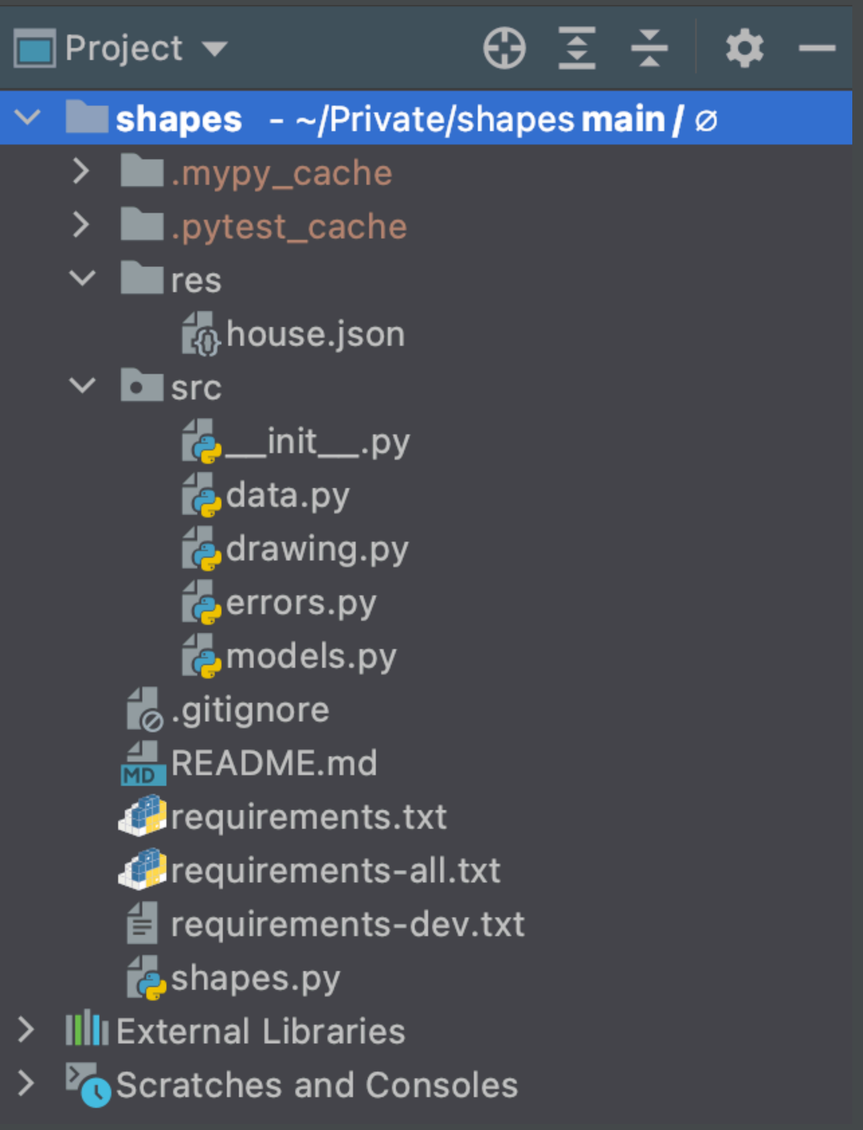
- 3 years of professional experience
- AirHelper since 2019
- Interests: Machine Learning, Programming books, Gym

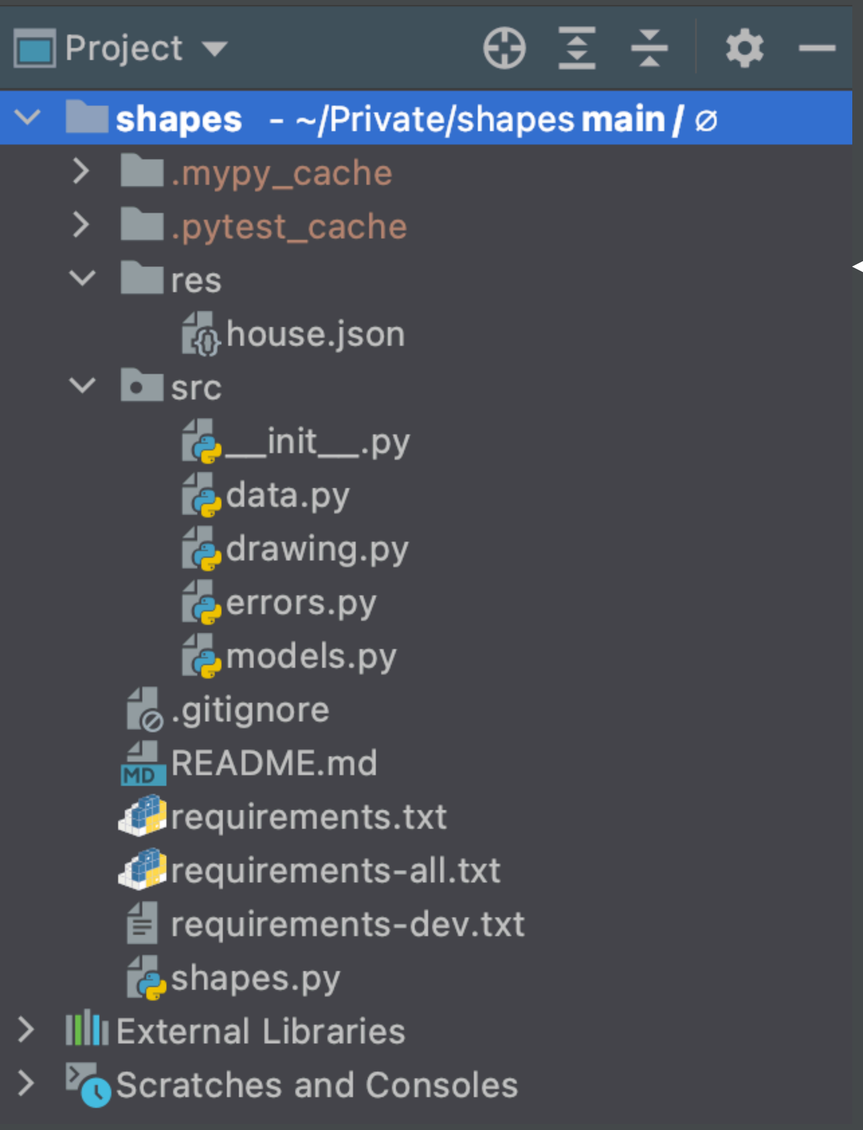
pkardas

Plan for today

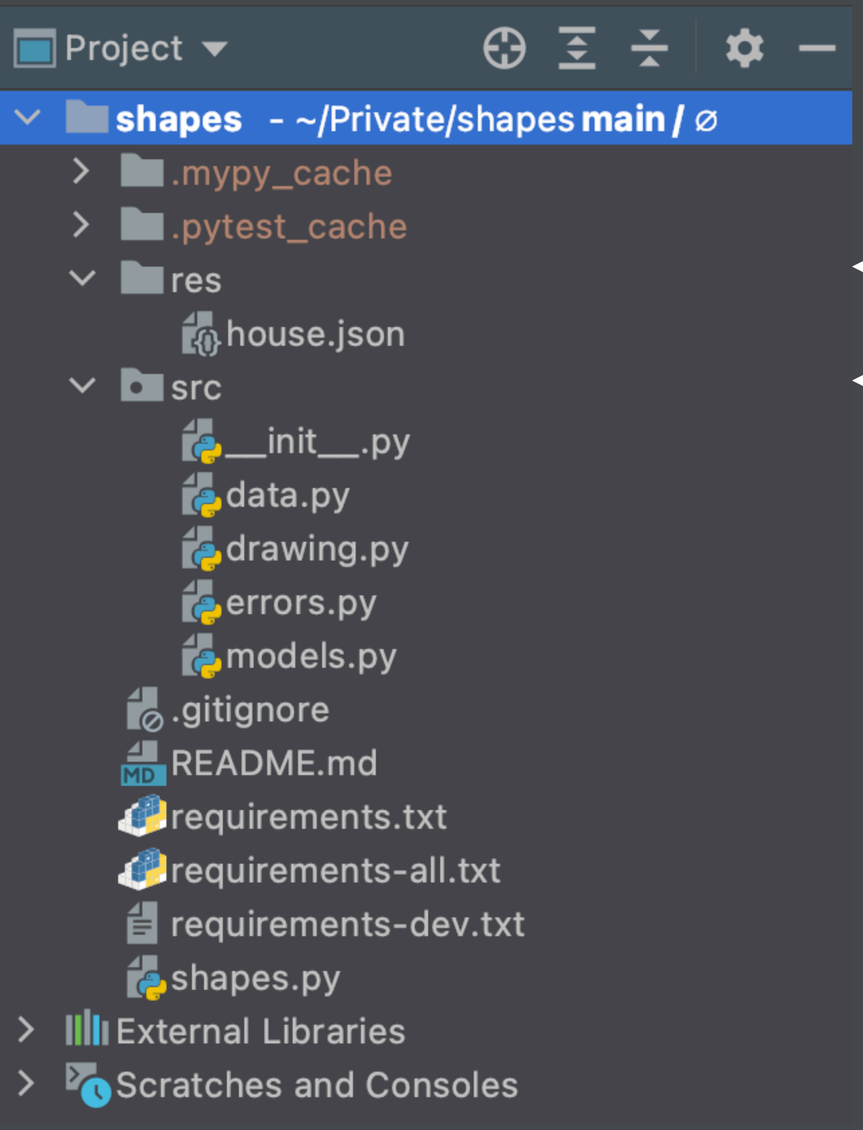
You have joined Awesome Drawings Inc. as a Software Engineer.

You will be responsible for extending company's core application - `shapes`.


Project
shapes - ~/Private/shapes main / ⌀
.mypy_cache
.pytest_cache
res
house.json
src
__init__.py
data.py
drawing.py
errors.py
models.py
.gitignore
README.md
requirements.txt
requirements-all.txt
requirements-dev.txt
shapes.py
External Libraries
Scratches and Consoles

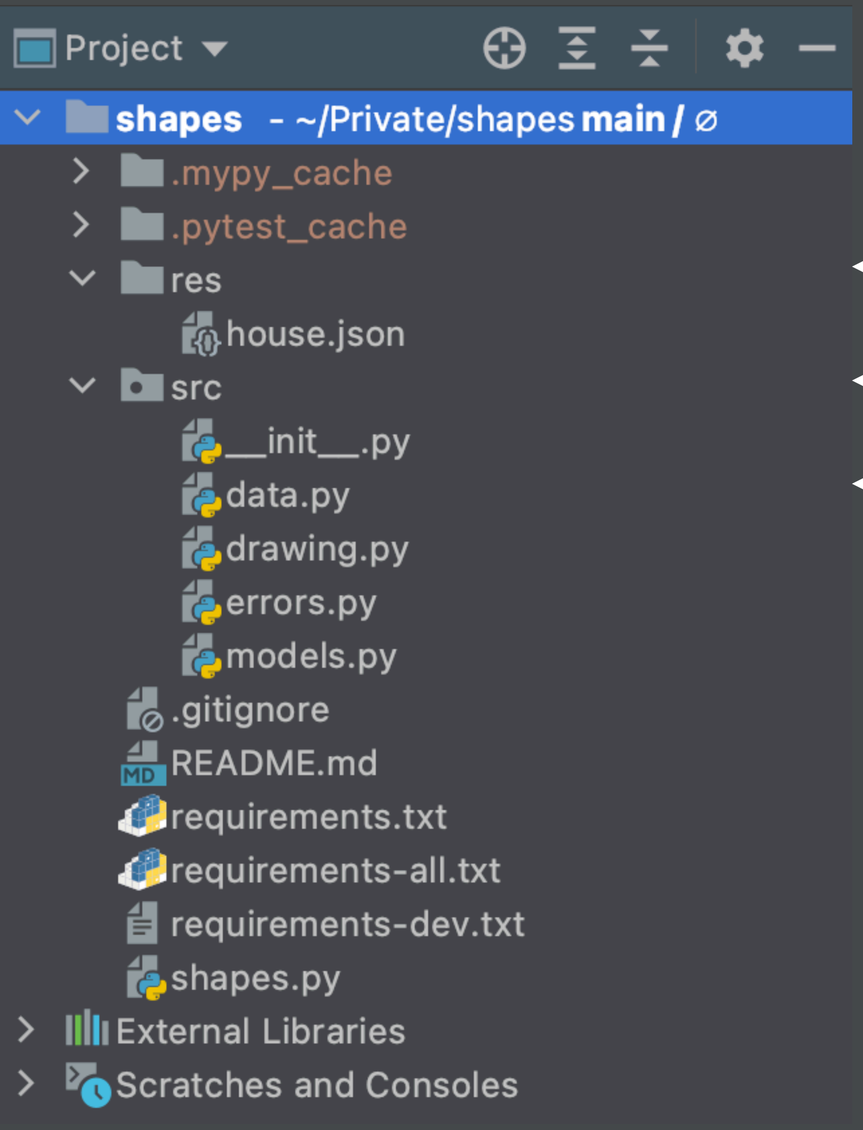


folder with resources (JSON files), that can be used as input for the program



folder with resources (JSON files), that can be used as input for the program

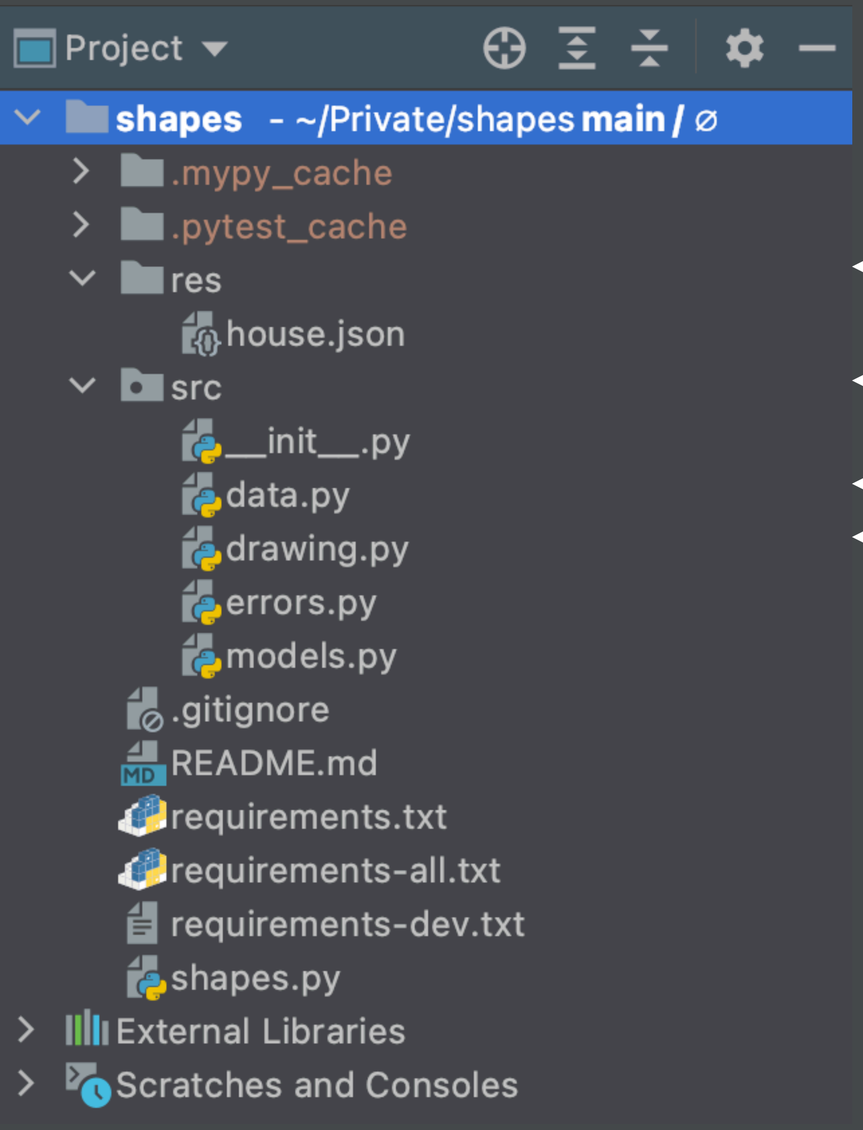
folder with Python code



folder with resources (JSON files), that can be used as input for the program

folder with Python code

everything related to data manipulation (saving, loading)

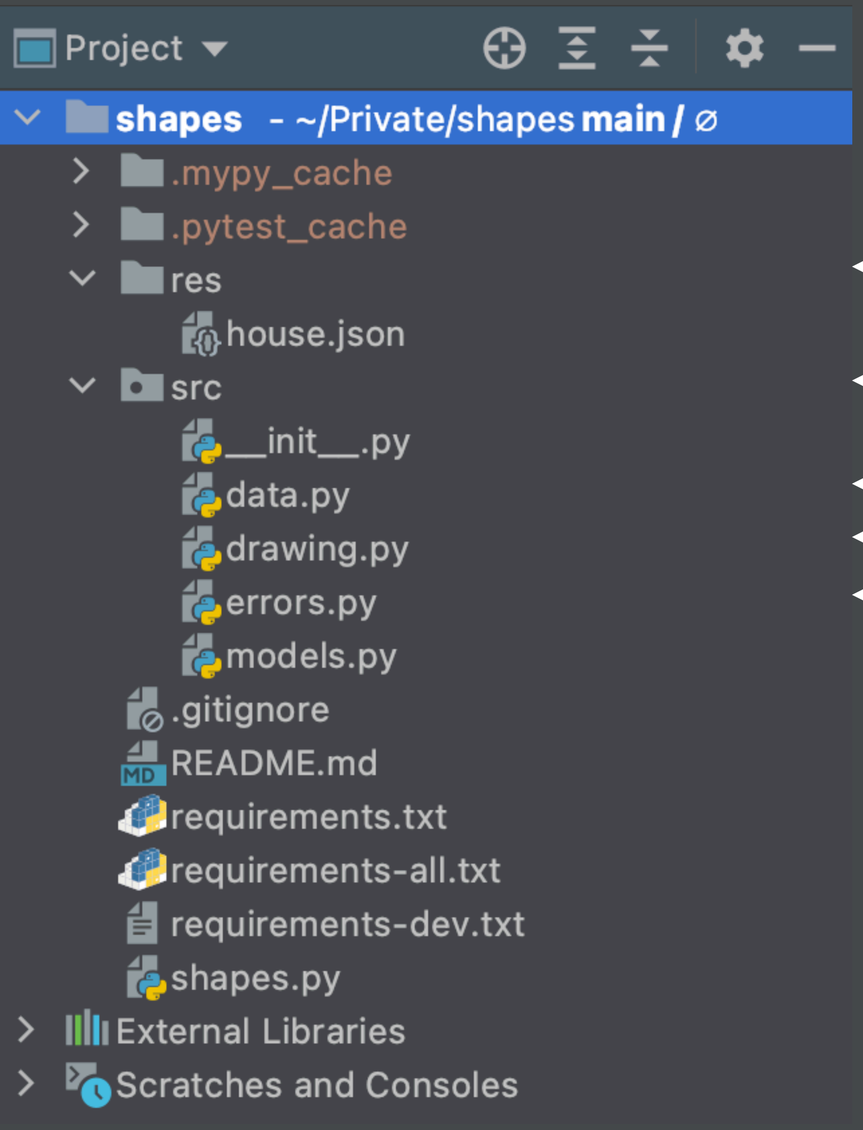


folder with resources (JSON files), that can be used as input for the program

folder with Python code

everything related to data manipulation (saving, loading)

everything related to drawing



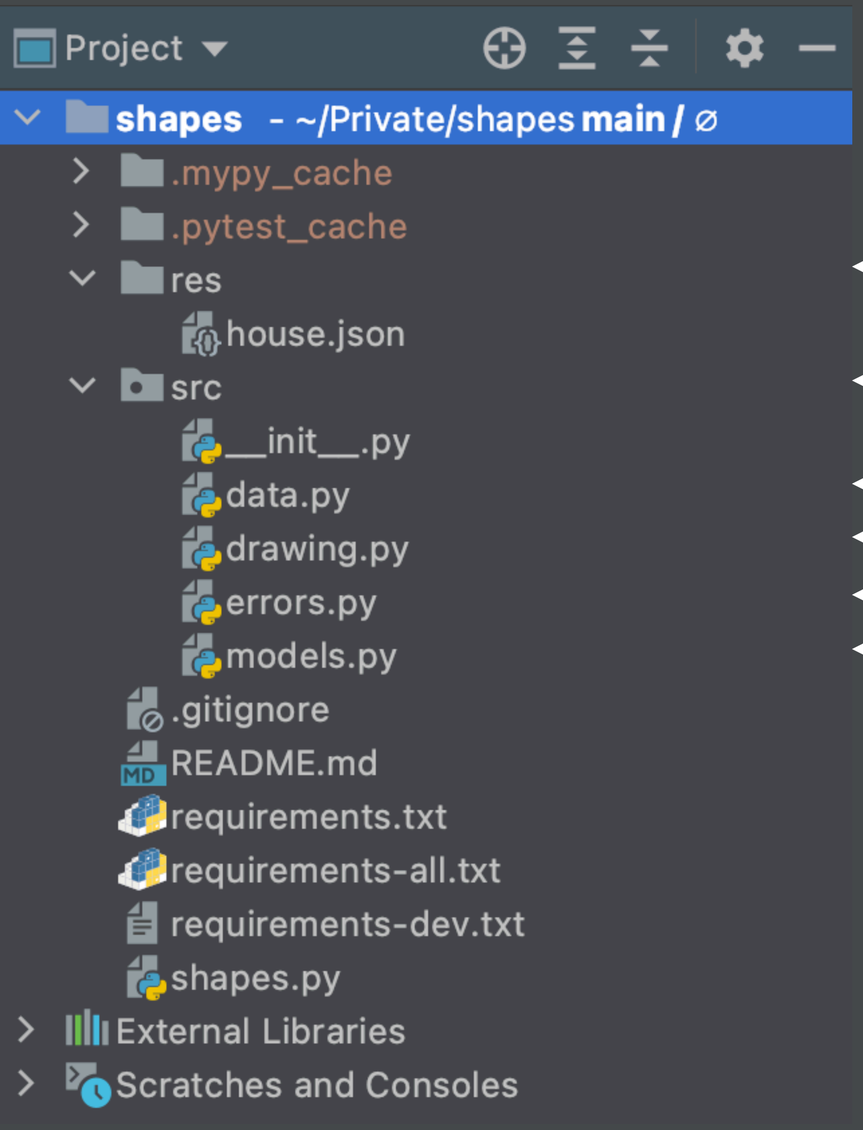
folder with resources (JSON files), that can be used as input for the program

folder with Python code

everything related to data manipulation (saving, loading)

everything related to drawing

custom errors



← folder with resources (JSON files), that can be used as input for the program

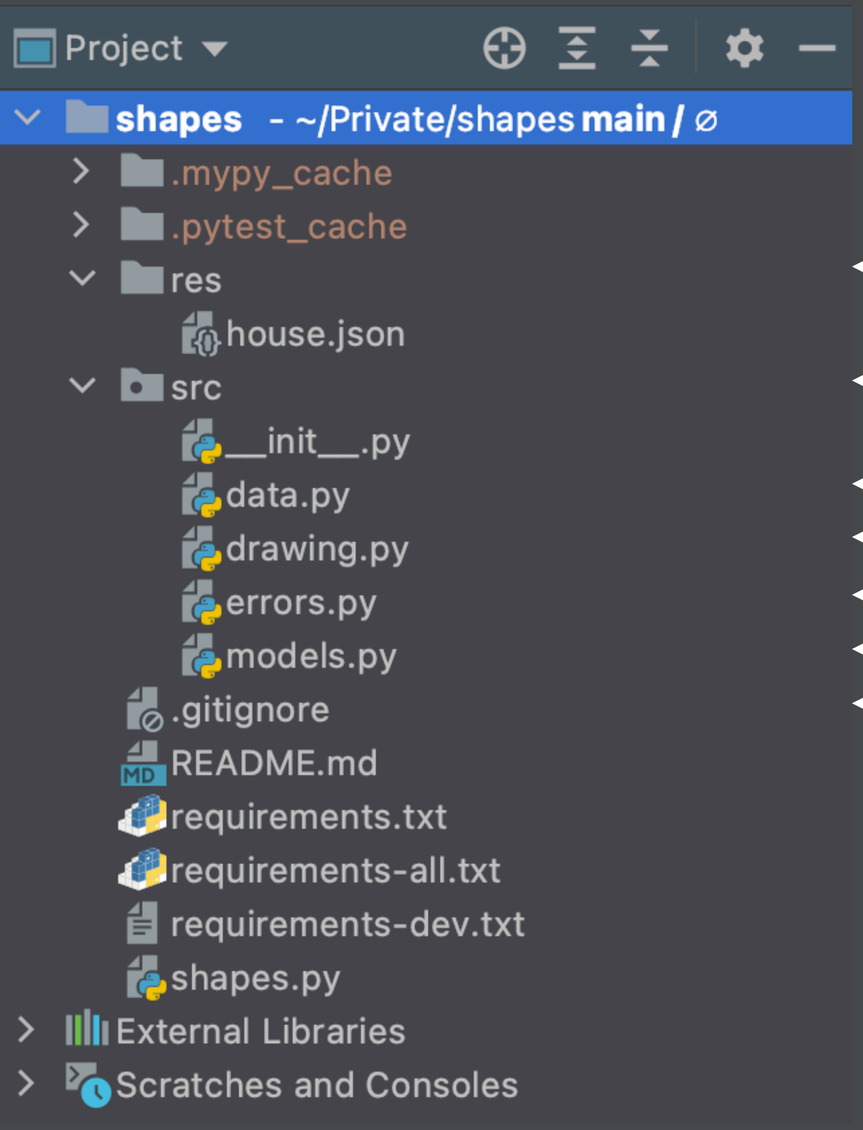
← folder with Python code

← everything related to data manipulation (saving, loading)

← everything related to drawing

← custom errors

← shapes definition



folder with resources (JSON files), that can be used as input for the program

folder with Python code

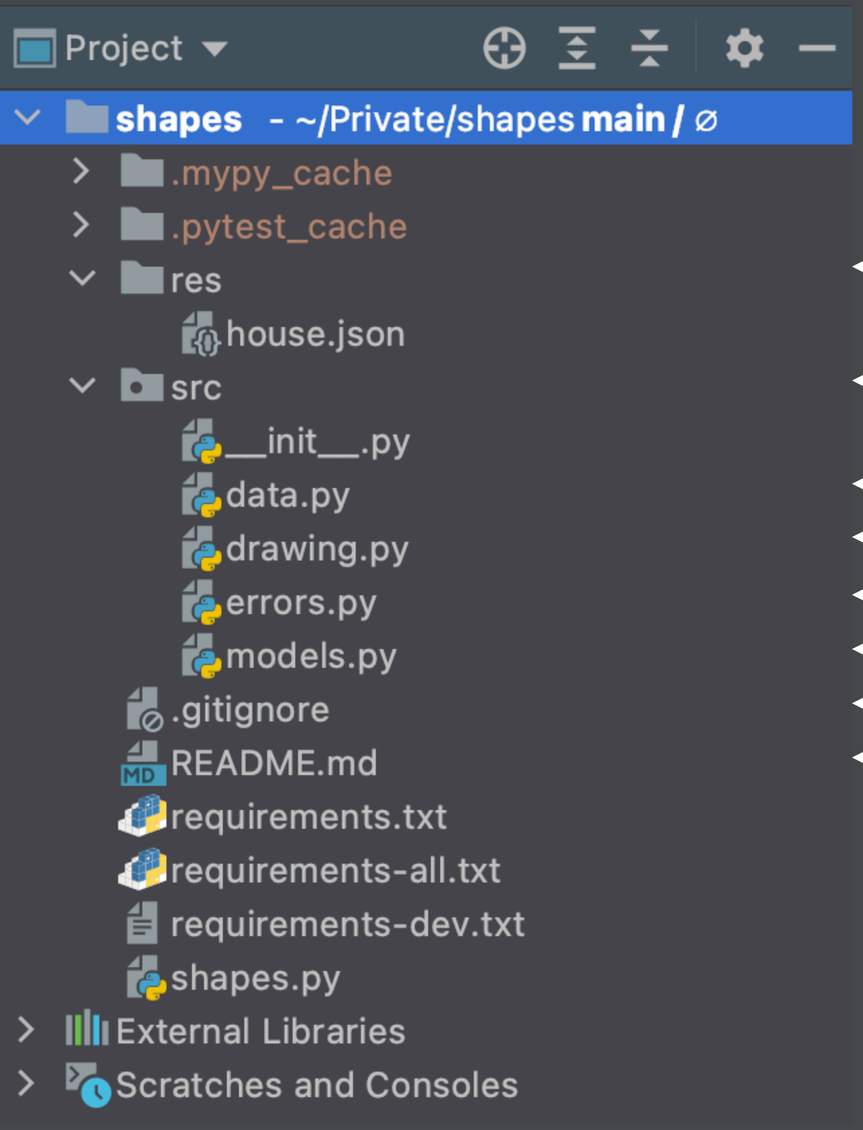
everything related to data manipulation (saving, loading)

everything related to drawing

custom errors

shapes definition

paths that are excluded from version control system



← folder with resources (JSON files), that can be used as input for the program

← folder with Python code

← everything related to data manipulation (saving, loading)

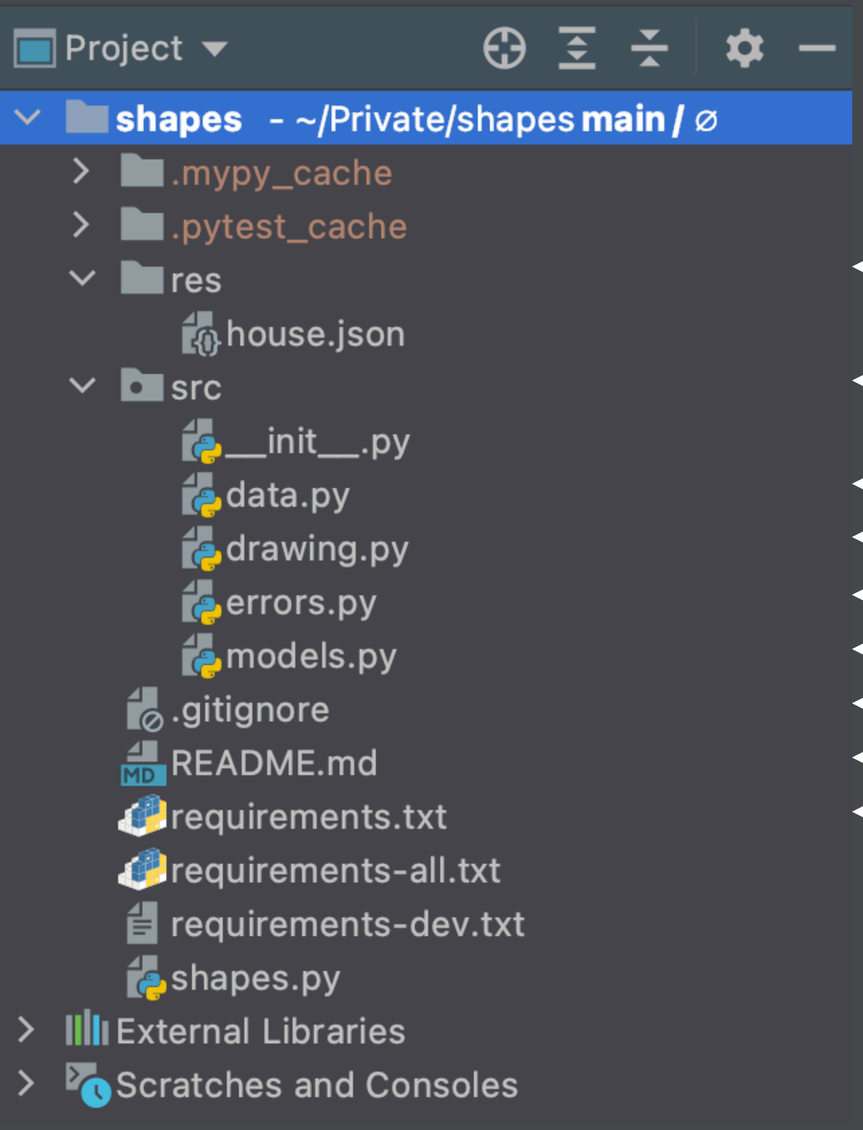
← everything related to drawing

← custom errors

← shapes definition

← paths that are excluded from version control system

← repository description and list of tasks



← folder with resources (JSON files), that can be used as input for the program

← folder with Python code

← everything related to data manipulation (saving, loading)

← everything related to drawing

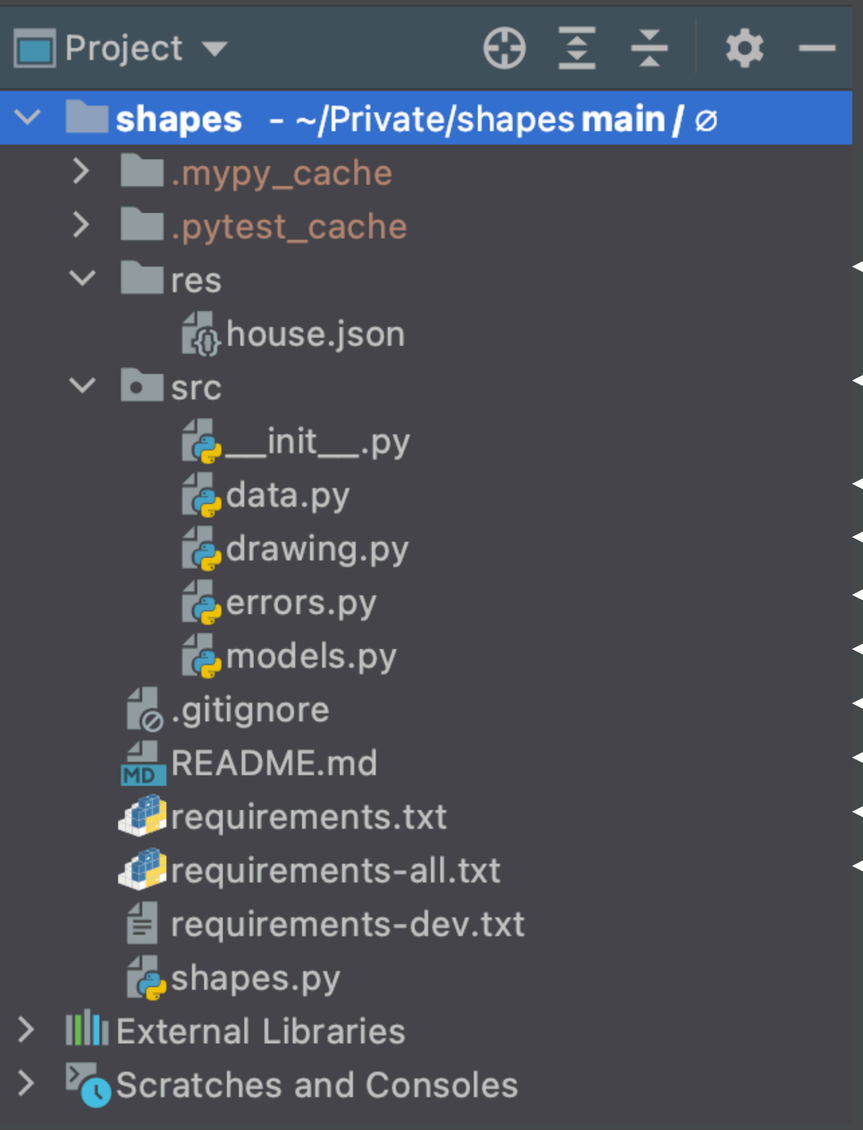
← custom errors

← shapes definition

← paths that are excluded from version control system

← repository description and list of tasks

← dependencies that are used by the program during runtime



← folder with resources (JSON files), that can be used as input for the program

← folder with Python code

← everything related to data manipulation (saving, loading)

← everything related to drawing

← custom errors

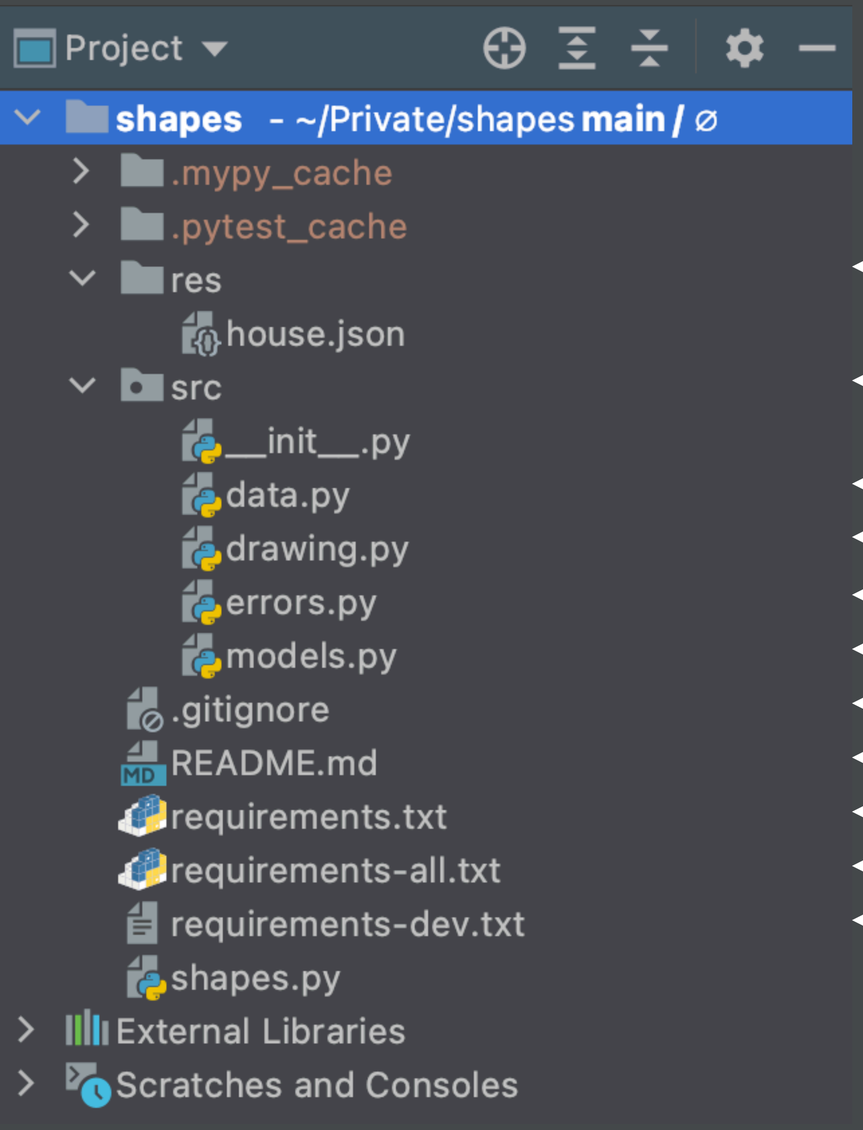
← shapes definition

← paths that are excluded from version control system

← repository description and list of tasks

← dependencies that are used by the program during runtime

← requirements.txt + requirements-dev.txt (joined together)



← folder with resources (JSON files), that can be used as input for the program

← folder with Python code

← everything related to data manipulation (saving, loading)

← everything related to drawing

← custom errors

← shapes definition

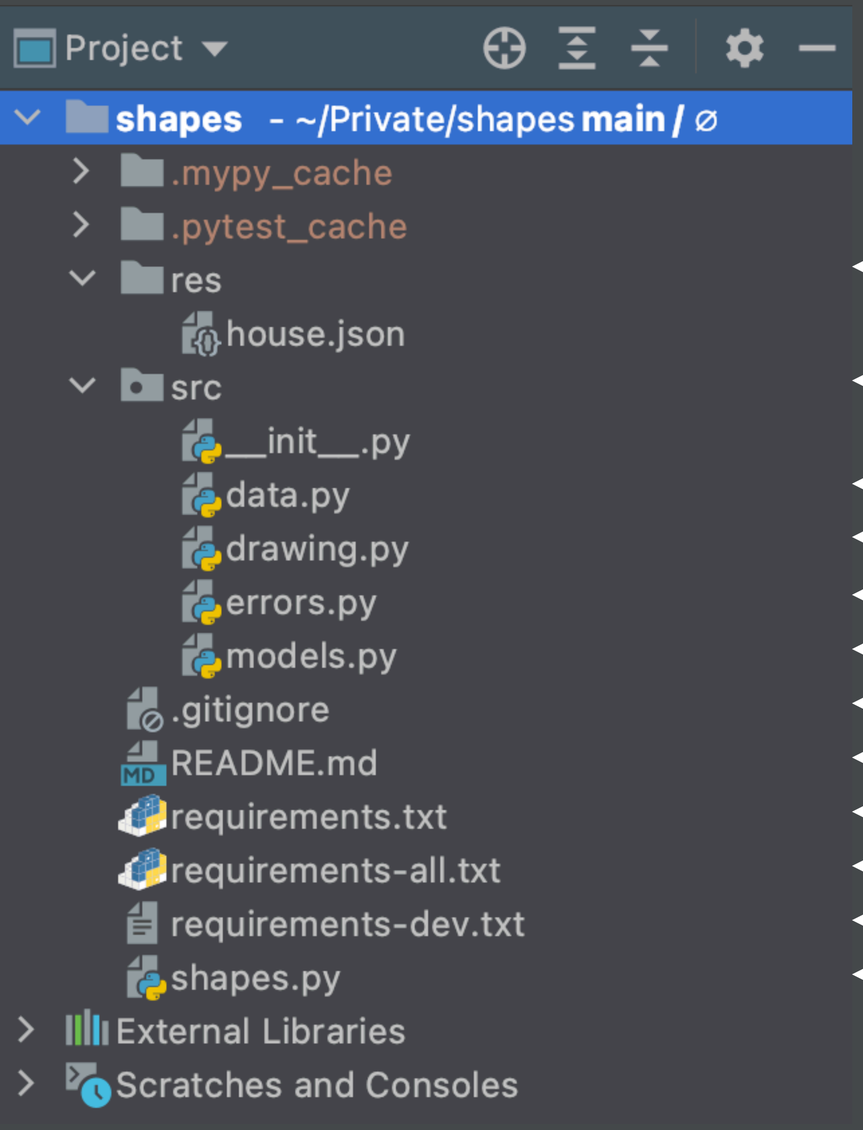
← paths that are excluded from version control system

← repository description and list of tasks

← dependencies that are used by the program during runtime

← requirements.txt + requirements-dev.txt (joined together)

← dependencies that are used during development



← folder with resources (JSON files), that can be used as input for the program

← folder with Python code

← everything related to data manipulation (saving, loading)

← everything related to drawing

← custom errors

← shapes definition

← paths that are excluded from version control system

← repository description and list of tasks

← dependencies that are used by the program during runtime

← requirements.txt + requirements-dev.txt (joined together)

← dependencies that are used during development

← script that is starting the application

TASK 1

Theory

Type Hints

```
surname: str
age: int
```

python.org/dev/peps/pep-0484

Theory

Type Hints

```
surname: str
age: int


from typing import List

numbers: List[int]
```

python.org/dev/peps/pep-0484

Theory

Type Hints

```
surname: str
age: int


from typing import List

numbers: List[int]


def foo(numbers: List[int]) → int:
    pass
```

python.org/dev/peps/pep-0484

Theory

The standard way of creating classes

```
class Person:
    def __init__(self, name: str, age: int) → None:
        self.name = name
        self.age = age


person = Person("James", 49)
```

Theory

@dataclass

```
from dataclasses import dataclass


@dataclass
class Person:
    name: str
    age: int


person = Person("James", 49)
```

python.org/dev/peps/pep-0484

Theory

Pydantic - dataclass on steroids

```
from pydantic import BaseModel


class Person(BaseModel):
    name: str
    age: int


person = Person(name="James", age=49)
```

Theory

Pydantic - dataclass on steroids

but why?

TASK 2

🐍 TASK 3

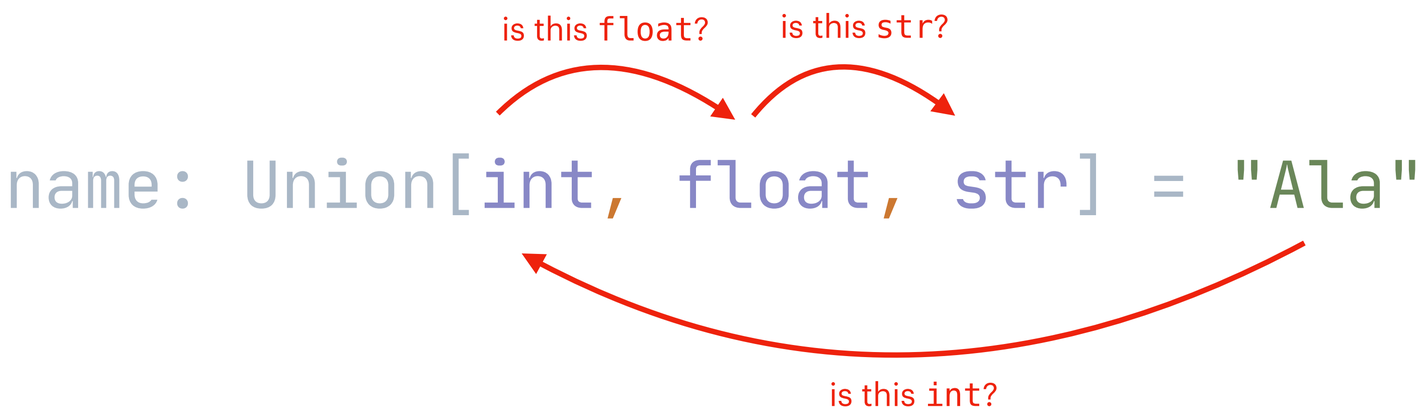
Theory

Union[...]

```
from typing import Union

name: Union[int, float, str] = "123"
name: Union[int, float, str] = 123
name: Union[int, float, str] = 123.00
```

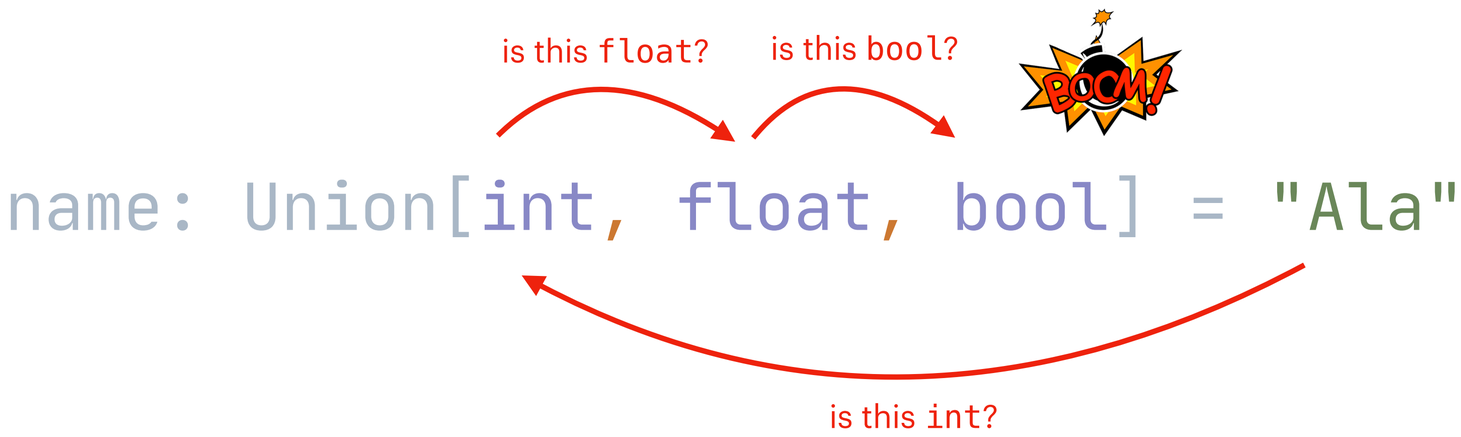
Theory

Union[...] and Pydantic



Theory

Union[...] and Pydantic



TASK 4

Theory

Optional[...]

```
from typing import Optional

name: Optional[int] = "James"
name: Optional[int] = None
```

Optional can be expressed as Union:

```
name: Union[int, None]
```

en.wikipedia.org/wiki/Option_type

TASK 5

Theory

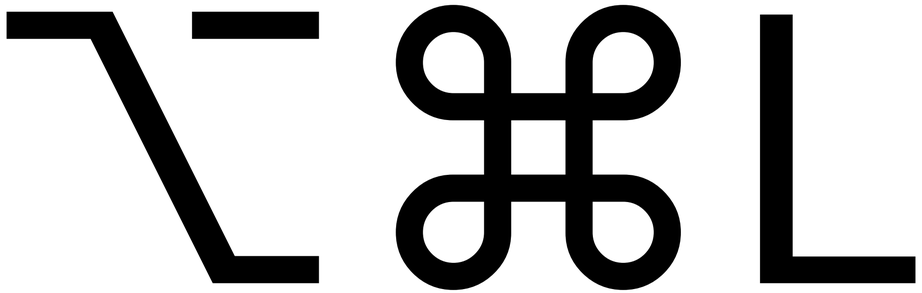
flake8

```
name: str =                    ":D"
```

E222 multiple spaces after operator

Theory

Reformat code



resources.jetbrains.com/storage/products/pycharm/docs/PyCharm_ReferenceCard_mac.pdf

Theory

flake8

```
name: str = ":D"
```

Theory

mypy

```
name: str = 123
```

error: Incompatible types in assignment (expression has type "int", variable has type "str")

Theory

mypy

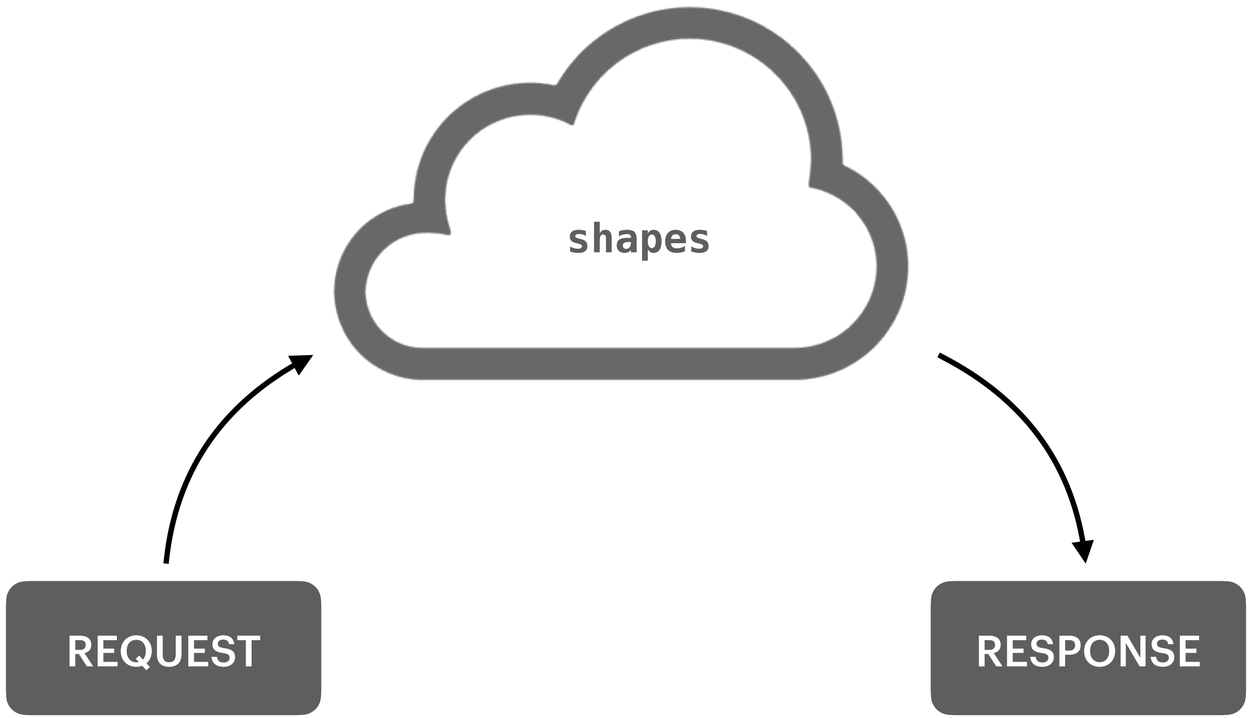
```
name: str = "Josh"
```

TASK 6

🐍 TASK 7*

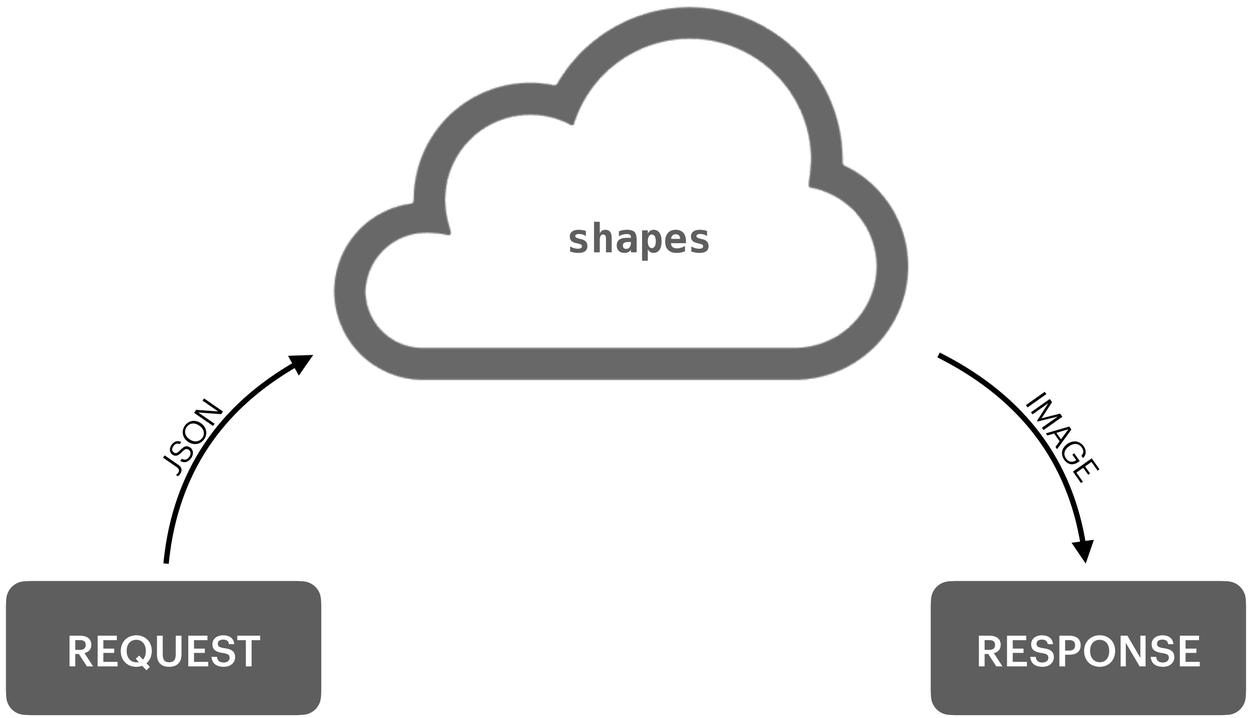
Theory

SaaS - Software as a Service



Theory

SaaS - Software as a Service



Theory

REST

Representational **S**tate **T**ransfer

REST is a design philosophy that builds upon the principles of HTTP.

Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, Martin Kleppmann

Theory

HTTP methods

GET	Requests a representation of the specified resource. Requests using GET should only retrieve data.
HEAD	Asks for a response identical to that of a GET request, but without the response body.
POST	Used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
PUT	Replaces all current representations of the target resource with the request payload.
DELETE	Deletes the specified resource.
CONNECT	Establishes a tunnel to the server identified by the target resource.
OPTIONS	Used to describe the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.
PATCH	Used to apply partial modifications to a resource.

developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Theory

HTTP methods

GET	**Requests a representation of the specified resource. Requests using GET should only retrieve data.**
HEAD	Asks for a response identical to that of a GET request, but without the response body.
POST	**Used to submit an entity to the specified resource, often causing a change in state or side effects on the server.**
PUT	**Replaces all current representations of the target resource with the request payload.**
DELETE	Deletes the specified resource.
CONNECT	Establishes a tunnel to the server identified by the target resource.
OPTIONS	Used to describe the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.
PATCH	Used to apply partial modifications to a resource.

Theory

FastAPI

```python
from typing import Optional

from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

TASK 8

🐍 TASK 9*

Theory

pytest

```
def add(a: int, b: int) → int:
    return a + b
```

Theory

pytest

```python
def add(a: int, b: int) -> int:
    return a + b


def test_add():
    assert add(1, 1) == 2
```

Theory

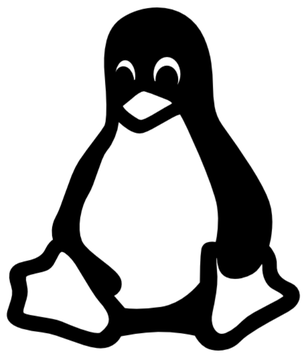
pytest

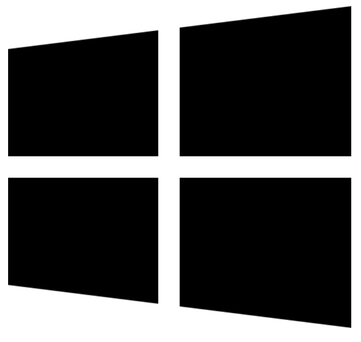
```
def add(a: int, b: int) → int:
    return a + b


def test_add():
    assert add(1, 1) == 2


import pytest


@pytest.mark.parametrize(
    "arg_0, arg_1, expected_result", [
        (0,   0,  0),
        (1,   1,  2),
        (1,  -1,  0),
        (-1, -1, -2),
    ]
)
def test_add(arg_0, arg_1, expected_result):
    assert add(arg_0, arg_1) == expected_result
```

TASK 10*

Theory

Docker

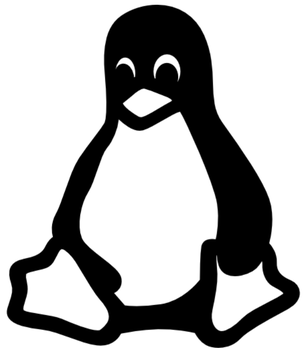


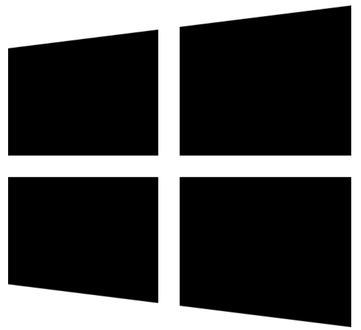


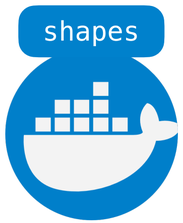
application installation might differ between operating systems

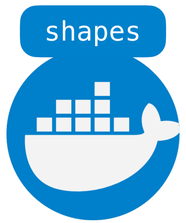
Theory

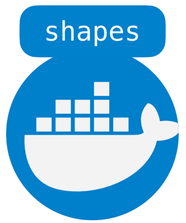
Docker











application installation looks exactly the same across different operating systems

Next steps 👟 🐍

Further application development.

Better error handling, database integration, AWS deployment.

Q&A