Подсказка по аннотации типов

В Python для аннотации есть специальный модуль, который называется **typing**. Здесь мы рассмотрим, какие классы импортируются из него и используются в разработке чаще всего.

1. Базовые коллекции

При использовании базовых коллекций, можно использовать записи вида “friends: list”. Однако когда мы знаем какой тип данных будет содержать наша коллекция, для аннотации используют соответствующие классы List, Dict и Tuple

```
from typing import List, Dict, Tuple
```

При использовании этих классов в квадратных скобках указывается элементы какого типа хранятся в коллекции, то есть, например, List[<тип элемента из списка>]

Пример с List:

```
def __init__(self, name: str, age: int, friends: List[Person]) -> None:
```

Пояснение: friends - это список, который содержит объекты класса Person.

Пример с Dict:

```
def __init__(self, name: str, age: int, friends: Dict[int, Person]) -> None:
```

Пояснение: friends - это словарь, в котором ключ - это целое число, значение - объект класса Person

2. Любой тип данных

Если передаваемый в функцию параметр (или возвращаемое значение) может быть вообще каким-угодно типом данных (кроме None), то здесь используется класс Any

```
from typing import Any
```

Пример:

```
def print_something(elem: Any) -> None:
    print(elem)
```

3. Объединение

При аннотации типов часто необходимо указать сразу несколько типов, которые могут приниматься в качестве параметра или могут возвращаться из функции (или быть в качестве значения в словаре). Для этого используется класс Union

```
from typing import Union
```

Пример метода:

```
def to_dict(self) -> Dict[str, Union[str, int, List[dict]]]:
    return {
        "name": self.__name,
        "age": self.__age,
        "friends": [i_person.to_dict() for i_person in self.__friends]
    }
```

Пояснение: здесь метод to_dict возвращает словарь. Ключ - строка, а значения идут в таком порядке: строка (имя), целое число (возраст) и список (друзья). При этом в списке в качестве значений хранятся словари.

Заметим, что в конце dict указан с маленькой буквы - это сделано для того, чтобы не описывать рекурсивно одно и тоже в виде *List[Dict[str, Union[str, int, List[Dict[.....]]]]]*

4. Объединение с None

Если нам нужно указать несколько типов, и при этом одним из этих типов является None, то чаще всего используется класс Optional

```
from typing import Optional
```

Optional[...] - это сокращение конструкции Union[..., None]. То есть, например, вместо *Union[None, Any]* стоит писать *Optional[Any]*.

Пример:

```
class Point:
    def __init__(self, value: Optional[Any], another: Optional['Point']) -> None:
```

Здесь мы обратим внимание на две вещи. Во-первых, вместо *value: Any* мы использовали *value: Optional[Any]*, то есть value может содержать None, либо что-то другое.

Во-вторых, в записи *Optional['Point']* объект класса взят в одинарные кавычки. Если их убрать, то PyCharm пожалуется на то, что Point ещё не объявлен, а мы его уже используем. Иногда такая запись также имеет место быть.

5. Генераторы

Наконец, напомним, что если функция реализует ленивые вычисления и возвращает генерируемое значение, то здесь используется библиотека *collections.abc* и класс Iterable

```
from collections.abc import Iterable
```

Пример:

```
def fibonacci(number: int) -> Iterable[int]:
```

P.S. Конечно же, в этой подсказке рассматриваются не все аннотации. Более подробная информация о модуле typing и различных классах, которые в нём содержатся, можно почитать в официальной документации: https://docs.python.org/3/library/typing.html#typing