Университет Сириус Колледж

Объектно-ориентированное программирование Лекция 6: Дженерики и `Protocol` для явных интерфейсов

Статическая типизация в Python. Дженерики (`typing.Generic`, `TypeVar`) для создания обобщенных классов и функций. `Protocol` (PEP 544) как явный способ определения структурных интерфейсов.

Университет Сириус Колледж

```python
def get_total_price(price: int, quantity: int):
    return price * quantity

print(get_total_price(100, 5))  # -> 500
```

Университет Сириус Колледж

```python
def get_total_price(price: int, quantity: int):
    return price * quantity

print(get_total_price("100", 5)) # -> "1001001001001001"
print(get_total_price("сто", "пять")) # -> TypeError
```

Университет Сириус Колледж

```python
def get_total_price(price: int, quantity: int):
    return price * quantity

print(get_total_price("100", 5)) # -> "100100100100100"
print(get_total_price("сто", "пять")) # -> TypeError

# mypy_error: Argument 1 to "get_total_price"
# has incompatible type "str"; expected "int"
```

Университет Сириус Колледж

Статическая типизация в Python

Python является языком с динамической типизацией, однако начиная с версии 3.5 поддерживает *аннотации типов*.

Статическая типизация достигается с помощью внешних инструментов: **MyPy**, **Pyright**, **Pylance** и др.

Аннотации позволяют описывать:

- типы переменных и атрибутов
- сигнатуры функций и методов
- структуры сложных типов данных
- интерфейсы взаимодействия объектов

Статическая типизация повышает надёжность, читаемость, предсказуемость и удобство рефакторинга.

Университет Сириус Колледж

Обобщённое программирование в Python

Обобщённое программирование (generic programming) позволяет описывать алгоритмы и структуры данных, не зависящие от конкретного типа.

Цели:

- Исключение дублирования кода.
- Создание универсальных структур.
- Повышение выразительности типовой системы.

Инструменты:

- `typing.TypeVar`
- `typing.Generic`
- Параметризованные типы (`list[int]`, `dict[str, float]`)

Университет Сириус Колледж

`TypeVar`: параметризация типов

`TypeVar` используется для объявления *типового параметра*, который выступает как переменная типа. Он позволяет создавать функции и классы, работающие с различными типами, сохраняя при этом строгую типизацию.

- `TypeVar` обозначает **неизвестный, но согласованный тип**.
- Все места, где используется данный `TypeVar`, должны относиться к *одному и тому же фактическому типу* при вызове функции или создании объекта.
- Конкретный тип выводится автоматически анализатором типов на основании переданных аргументов.

Университет Сириус Колледж

`TypeVar`: параметризация типов

```python
from typing import TypeVar

T = TypeVar("T")
```

`T` — это placeholder для реального типа.

Университет Сириус Колледж

`TypeVar`: параметризация типов

Без `TypeVar` сигнатура обобщённой функции потеряла бы точность. Например:

```
1  def identity(x):
2      return x
```

Университет Сириус Колледж

`TypeVar`: параметризация типов

Без `TypeVar` сигнатура обобщённой функции потеряла бы точность. Например:

```
1  def identity(x):
2      return x
```

Статический анализатор не может вывести, что тип результата связан с типом аргумента.

Университет Сириус Колледж

`TypeVar`: параметризация типов

Без `TypeVar` сигнатура обобщённой функции потеряла бы точность. Например:

```
1  def identity(x):
2      return x
```

Статический анализатор не может вывести, что тип результата связан с типом аргумента.

```
1  def identity(x: T) -> T:
2      return x
```

Университет Сириус Колледж

`TypeVar`: параметризация типов

Без `TypeVar` сигнатура обобщённой функции потеряла бы точность. Например:

```
1  def identity(x):
2      return x
```

Статический анализатор не может вывести, что тип результата связан с типом аргумента.

```
1  def identity(x: T) -> T:
2      return x
```

Такие функции называются **обобщенными**. Теперь анализатор знает: *тип результата всегда совпадает с типом аргумента*.

Университет Сириус Колледж

Пример использования TypeVar

```
2
3  T = TypeVar('T')
4
5  def get_first_item(items: List[T]) -> T:
6      return items[0]
7
8  numbers = [10, 20, 30]
9  first_num = get_first_item(numbers)
10 # MyPy знает, что first_num имеет тип int, а не Any!
11 print(first_num + 5) # OK
12
13 strings = ["a", "b", "c"]
14 first_str = get_first_item(strings)
15 # MyPy знает, что first_str имеет тип str
16 print(first str.upper()) # OK
```

Университет Сириус Колледж

Пример использования TypeVar

```python
from typing import TypeVar, List

T = TypeVar('T')

def get_first_item(items: List[T]) -> T:
    return items[0]

numbers = [10, 20, 30]
first_num = get_first_item(numbers)
# MyPy знает, что first_num имеет тип int, а не Any!
print(first_num + 5) # OK

strings = ["a", "b", "c"]
first_str = get_first_item(strings)
# MyPy знает, что first_str имеет тип str
```

Университет Сириус Колледж

Пример использования TypeVar

```python
from typing import TypeVar, List

T = TypeVar('T')

def get_first_item(items: List[T]) -> T:
    return items[0]

numbers = [10, 20, 30]
first_num = get_first_item(numbers)
# MyPy знает, что first_num имеет тип int, а не Any!
print(first_num + 5) # OK

strings = ["a", "b", "c"]
first_str = get_first_item(strings)
# MyPy знает, что first_str имеет тип str
```

Университет Сириус Колледж

Пример использования TypeVar

```python
T = TypeVar('T')

def get_first_item(items: List[T]) -> T:
    return items[0]

numbers = [10, 20, 30]
first_num = get_first_item(numbers)
# MyPy знает, что first_num имеет тип int, а не Any!
print(first_num + 5) # OK

strings = ["a", "b", "c"]
first_str = get_first_item(strings)
# MyPy знает, что first_str имеет тип str
print(first_str.upper()) # OK
```

Университет Сириус Колледж

Пример использования TypeVar

```python
T = TypeVar('T')

def get_first_item(items: List[T]) -> T:
    return items[0]

numbers = [10, 20, 30]
first_num = get_first_item(numbers)
# MyPy знает, что first_num имеет тип int, а не Any!
print(first_num + 5) # OK

strings = ["a", "b", "c"]
first_str = get_first_item(strings)
# MyPy знает, что first_str имеет тип str
print(first_str.upper()) # OK
```

Университет Сириус Колледж

Ограничения типового параметра. Ограничение сверху (bound)

Используется, когда требуется, чтобы параметр типа был *подтипом* определённого класса.

```
TNum = TypeVar("TNum", bound=float)
```

Это означает: допускаются `float` и любые его подтипы (например, `numpy.float64`).

Использование:

```
def halve(x: TNum) -> TNum:
    return x / 2
```

Университет Сириус Колледж

Ограничения типового параметра. Ограничения перечислением (constrained)

Определяет строгий набор допустимых типов:

```
1  TComparable = TypeVar("TComparable", int, float, str)
```

Здесь допускаются только перечисленные типы.

Использование подходит, например, для функций сравнения или сортировки.

Университет Сириус Колледж

`TypeVar` vs обычные аннотации

Аннотация без TypeVar	Аннотация с TypeVar
тип результата не связан с типом аргумента	типы связаны, поведение строго описано
слабая выразительность	высокая выразительность
часто приходится указывать `Any`	точная типизация

Университет Сириус Колледж

Автоматический вывод типа

```python
def pair(a: T, b: T) -> tuple[T, T]:
    return (a, b)

pair(1, 2)        # T -> int
pair("a", "b")    # T -> str
pair(1, "b")      # ошибка статической типизации
```

Анализатор выводит типы для `T` и проверяет их согласованность.

Университет Сириус Колледж

Обобщённые классы и `Generic`

`Generic` — это базовый класс из модуля `typing`, предназначенный для объявления **обобщённых (параметризованных) классов**, которые работают с типами, передаваемыми в виде *типовых параметров* (`TypeVar`).

Он служит механизмом поддержки *дженериков* в системе статической типизации Python и позволяет создавать классы, которые сохраняют информацию о типах своих элементов.

Ключевые особенности:

1. `Generic` связывает объявленный `TypeVar` с конкретным классом.
2. Специализация типа создаётся записью `Box[int]`, `Box[str]` и т. д.
3. Статический анализатор отслеживает корректность используемых типов.
4. В runtime `Generic` не создаёт новых классов — его назначение находится исключительно в слое типовой системы.

Университет Сириус Колледж

Обобщённые классы и `Generic`

Используются:

- Для создания универсальных структур данных (например, `Stack[T]`, `Graph[T]`).
- Для повышения точности статической типизации.
- Для обеспечения типобезопасности при работе с разными типами данных.
- Для описания API, в которых типы параметров и возвращаемых значений должны быть согласованы.

Университет Сириус Колледж

Обобщённые классы и Generic

```
3  T = TypeVar("T")
4
5  class Stack(Generic[T]):
6      def __init__(self) -> None:
7          self._items: list[T] = []
8
9      def push(self, item: T) -> None:
10         self._items.append(item)
11
12     def pop(self) -> T:
13         return self._items.pop()
14
15
16 ints: Stack[int] = Stack()
17 ints.push(10)
```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python
from typing import Generic, TypeVar

T = TypeVar("T")

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()

```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python
from typing import Generic, TypeVar

T = TypeVar("T")

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()

```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python
from typing import Generic, TypeVar

T = TypeVar("T")

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()

```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python
T = TypeVar("T")

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()


ints: Stack[int] = Stack()
ints.push(10)
```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()


ints: Stack[int] = Stack()
ints.push(10)
ints.push("str")  # ошибка статической типизации
```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python

class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()


ints: Stack[int] = Stack()
ints.push(10)
ints.push("str")  # ошибка статической типизации
```

Университет Сириус Колледж

Обобщённые классы и `Generic`

```python
class Stack(Generic[T]):
    def __init__(self) -> None:
        self._items: list[T] = []

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        return self._items.pop()


ints: Stack[int] = Stack()
ints.push(10)
ints.push("str")  # ошибка статической типизации
```

Университет Сириус Колледж

Интерфейсы в Python: проблема отсутствия ключевого слова

Python не имеет:

- ключевого слова `interface`,
- строгой контрактной связи,
- проверки соответствия интерфейсу на уровне рантайма.

Исторически Python опирается на:

- утиную типизацию,
- структурную типизацию по поведению.

PEP 544 ввёл механизм `Protocol`, обеспечив формальное описание интерфейсов в рамках типовой системы.

Университет Сириус Колледж

`Protocol`: формирование структурного интерфейса

```python
from typing import Protocol

class Serializer(Protocol):
    def serialize(self) -> str:
        ...
```

Характеристики:

- Соответствие определяется структурно (по наличию методов).
- Наследование необязательно.
- Используется только статическими анализаторами.

Университет Сириус Колледж

`Protocol`: формирование структурного интерфейса

Сравнение с `ABC`

`Protocol`	`ABC`
Структурная типизация	Номинальная
Наследование не требуется	Требуется
Нет реализации	Может содержать

Университет Сириус Колледж

Пример использования Protocol

```
1  class Stream(Protocol):
2      def read(self, n: int) -> bytes:
3          pass
4
5  class FileStream:
6      def read(self, n: int) -> bytes:
7          return b"data"
8
9  class MemoryStream:
10     def read(self, n: int) -> bytes:
11         return b"x"
12
13
14 def process_stream(stream: Stream, bytes_to_read: int):
15     """Эта функция работает с ЛЮБЫМ объектом
```

Университет Сириус Колледж

Пример использования `Protocol`

```python
class Stream(Protocol):
    def read(self, n: int) -> bytes:
        pass

class FileStream:
    def read(self, n: int) -> bytes:
        return b"data"

class MemoryStream:
    def read(self, n: int) -> bytes:
        return b"x"


def process_stream(stream: Stream, bytes_to_read: int):
    """Эта функция работает с ЛЮБЫМ объектом
```

Университет Сириус Колледж

Пример использования Protocol

```python
class Stream(Protocol):
    def read(self, n: int) -> bytes:
        pass

class FileStream:
    def read(self, n: int) -> bytes:
        return b"data"

class MemoryStream:
    def read(self, n: int) -> bytes:
        return b"x"


def process_stream(stream: Stream, bytes_to_read: int):
    """Эта функция работает с ЛЮБЫМ объектом
```

Университет Сириус Колледж

Пример использования Protocol

```
        pass

class FileStream:
    def read(self, n: int) -> bytes:
        return b"data"

class MemoryStream:
    def read(self, n: int) -> bytes:
        return b"x"


def process_stream(stream: Stream, bytes_to_read: int):
    """Эта функция работает с ЛЮБЫМ объектом,
    который соответствует протоколу Stream."""
    data = stream.read(bytes_to_read)
```

Университет Сириус Колледж

Пример использования Protocol

```
class MemoryStream:
    def read(self, n: int) -> bytes:
        return b"x"


def process_stream(stream: Stream, bytes_to_read: int):
    """Эта функция работает с ЛЮБЫМ объектом,
    который соответствует протоколу Stream."""
    data = stream.read(bytes_to_read)
    print(f"Прочитано {len(data)} байт: {data}")

file_obj = FileStream()
memory_obj = MemoryStream()

process_stream(file_obj, 4)
```

Университет Сириус Колледж

Пример использования Protocol

```
7           return b"data"
8
9   class MemoryStream:
10      def read(self, n: int) -> bytes:
11          return b"x"
12
13
14  def process_stream(stream: Stream, bytes_to_read: int):
15      """Эта функция работает с ЛЮБЫМ объектом,
16      который соответствует протоколу Stream."""
17      data = stream.read(bytes_to_read)
18      print(f"Прочитано {len(data)} байт: {data}")
19
20  file_obj = FileStream()
21  memory obj = MemoryStream()
```

Университет Сириус Колледж

Пример использования Protocol

```
10      def read(self, n: int) -> bytes:
11          return b"x"
12
13
14  def process_stream(stream: Stream, bytes_to_read: int):
15      """Эта функция работает с ЛЮБЫМ объектом,
16      который соответствует протоколу Stream."""
17      data = stream.read(bytes_to_read)
18      print(f"Прочитано {len(data)} байт: {data}")
19
20  file_obj = FileStream()
21  memory_obj = MemoryStream()
22
23  process_stream(file_obj, 4)
24  process_stream(memory_obj, 5)
```

Университет Сириус Колледж

Протоколы с атрибутами и свойствами

```python
class UserProtocol(Protocol):
    name: str

    @property
    def is_active(self) -> bool:
        ...
```

Подходит любой объект, имеющий необходимые члены.

Университет Сириус Колледж

Обобщённые протоколы

```python
from typing import Protocol, TypeVar

T = TypeVar("T")

class Comparable(Protocol[T]):
    def __lt__(self, other: T) -> bool:
        ...
```

Университет Сириус Колледж

Статическая проверка протоколов

Статический анализатор проверяет:

- наличие методов и атрибутов,
- корректность сигнатур,
- соответствие типовым ограничениям.

Это повышает надёжность и формализует архитектуру.

Университет Сириус Колледж

`Protocol` и утиная типизация

Утиная типизация	Protocol
Проверка только во время выполнения.	Формальный контракт
Ошибки возникают поздно	Проверка до выполнения
	Сохраняет гибкость структурной типизации

Университет Сириус Колледж

Протоколы стандартной библиотеки

PEP 544 вводит протоколы:

- `Iterable`, `Iterator`, `Sequence`
- `Mapping`, `MutableMapping`
- `SupportsInt`, `SupportsFloat`
- `ContextManager`
- и другие

Они активно используются в стандартной библиотеке и сторонних фреймворках.