МФТИ

Коллекции и их протоколы

Докладчик: Евграфов Михаил

Общие протоколы

Протокол контейнера

Container

`__contains__`

Реализация контейнера

```
from typing import Any, Iterable, Hashable

class MyContainer:
    _container: set

    def __init__(
        self, iterable: Iterable[Hashable]
    ) -> None:
        self._container = set(iterable)

    def __contains__(self, obj: Any) -> bool:
        return obj in self._container
```

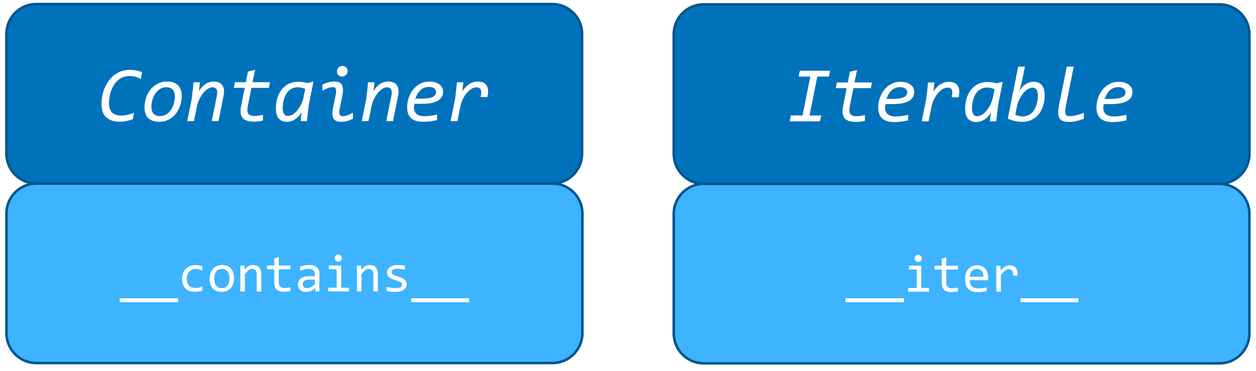
Допустимые действия

```
>>> from collections.abc import Container

>>> my_container = MyContainer([1, 2, 3])
>>> isinstance(my_container, Container)
>>> print(1 in my_container)
>>> print(5 in my_container)
True
True
False
```

Недопустимые действия

```
>>> my_container = MyContainer([1, 2, 3])
>>> for i in my_container:
        ...
TypeError: 'MyContainer' object is not iterable

>>> len(my_container)
...
TypeError: object of type 'MyContainer' has no len()

>>> my_container[1]
...
TypeError: 'MyContainer' object is not subscriptable
```

Протокол итерируемого объекта



Реализация итерируемого объекта

```
from typing import Any, Iterable, Iterator

class MyIterable:
    _iterable: list[Any]

    def __init__(self, iterable: list[Any]) -> None:
        self._iterable = list(iterable)

    def __iter__(self) -> Iterator[Any]:
        return iter(self._iterable)
```

Допустимые действия

```
>>> my_iterable = MyIterable([1, 2, 3])
>>> print(isinstance(my_iterable, Iterable))
True
>>> for elem in my_iterable:
...     print(elem)
1
2
3
>>> print(type(iter(my_iterable)).__name__)
list_iterator
```

Недопустимые действия

```
>>> my_iterable = MyIterable([1, 2, 3])
>>> my_iterable[0]
...
TypeError: 'MyIterable' object is not subscriptable
>>> len(my_iterable)
...
TypeError: object of type 'MyIterable' has no len()
>>> 5 not in my_iterable
True
```

?????

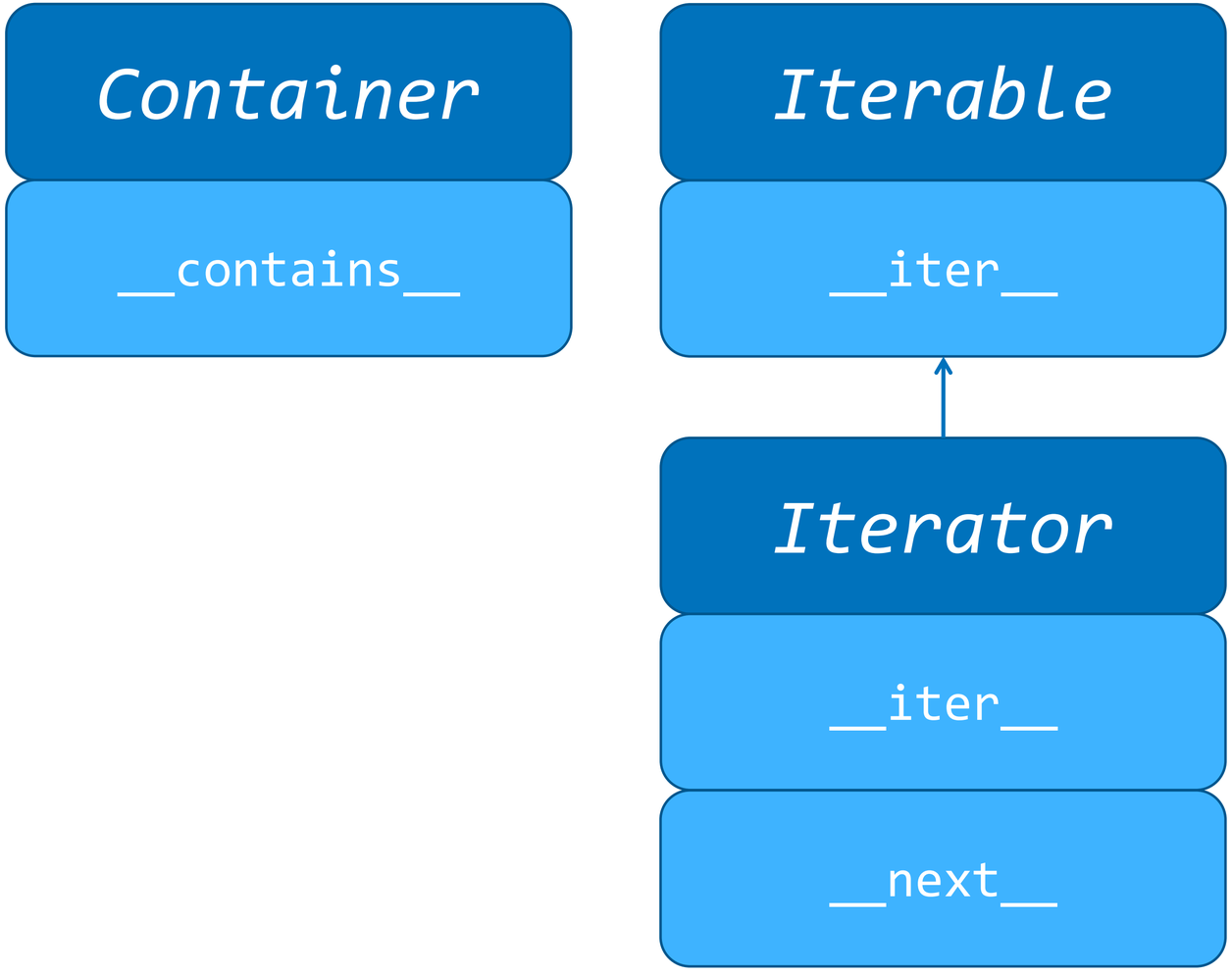
Связь __iter__ и __contains__

```
class MyIterable:
    ...
    def __iter__(self) -> Iterator[Any]:
        print("call iter")
        return iter(self._iterable)

>>> print(isinstance(my_iterable, Container))
>>> 1 in my_iterable
False
call iter
True
```

Неверная реализация

```
class MyIterable:
    ...
    def __iter__(self) -> Iterator[Any]:
        return self._iterable

>>> my_iterable = MyIterable([1, 2, 3])
>>> for elem in my_iterable:
        ...
TypeError: iter() returned non-iterator of type 'list'
```

Протокол итератора



Попытка реализации #1

```
class MyIterable:
    _iterable: list[Any]
    _iter_ptr: int

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)
        self._iter_ptr = -1

    def __iter__(self) -> Iterator[Any]:
        return self

    def __next__(self) -> Any:
        self._iter_ptr += 1
        if self._iter_ptr < len(self._iterable):
            return self._iterable[self._iter_ptr]
        raise StopIteration
```

Проблемы реализации

```
>>> my_iterable = MyIterable([1, 2, 3])
>>> for elem in my_iterable:
...     print(elem)
1
2
3
>>> for elem in my_iterable:
...     print(elem)
# ничего не выведено
```

Попытка реализации #2

```
class MyIterable:
    _iterable: list[Any]
    _iter_ptr: int

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __iter__(self) -> Iterator[Any]:
        self._iter_ptr = -1
        return self

    def __next__(self) -> Any:
        self._iter_ptr += 1
        if self._iter_ptr < len(self._iterable):
            return self._iterable[self._iter_ptr]
        raise StopIteration
```

Проблемы решены?

```
>>> my_iterable = MyIterable([1, 2, 3])
>>> for elem in my_iterable:
...     print(elem)
1
2
3
>>> for elem in my_iterable:
...     print(elem)
1
2
3
```

Да будут новые проблемы!

```
>>> my_iterable = MyIterable([1, 2, 3])
>>> for elem in my_iterable:
...     for elem_inner in my_iterable:
...         print(elem_inner)
1
2
3
```

Попытка реализации #3

```
class MyIterable:
    _iterable: list[Any]
    _iter_ptr: int

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)
        self._iter_ptr = -1

    def __iter__(self) -> Iterator[Any]:
        return MyIterable(self._iterable)

    def __next__(self) -> Any:
        self._iter_ptr += 1
        if self._iter_ptr < len(self._iterable):
            return self._iterable[self._iter_ptr]
        raise StopIteration
```

Ну теперь-то решены?

```
>>> my_iterable = MyIterable([1, 2])

>>> for elem in my_iterable:
...     print(elem, sep=" ")

>>> for elem in my_iterable:
...     for elem_inner in my_iterable:
...         print(elem_inner, sep=" ")
1 2 1 2 1 2
```

Итератор и объектная модель

```
>>> my_iterable = MyIterable([1, 2])
>>> print(next(my_iterable))
1
>>> my_list = [1, 2]
>>> print(next(my_list))
...
TypeError: 'list' object is not an iterator
```

Финальная реализация

```
class MyIterator:
    _iterable: list[Any]
    _iter_ptr: int

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)
        self._iter_ptr = -1

    def __iter__(self) -> "MyIterator":
        return MyIterator(self._iterable)

    def __next__(self) -> Any:
        self._iter_ptr += 1
        if self._iter_ptr < len(self._iterable):
            return self._iterable[self._iter_ptr]
        raise StopIteration
```

Финальная реализация

```
class MyIterable:
    _iterable: list[Any]

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __iter__(self) -> Iterator[Any]:
        return MyIterator(self._iterable)
```

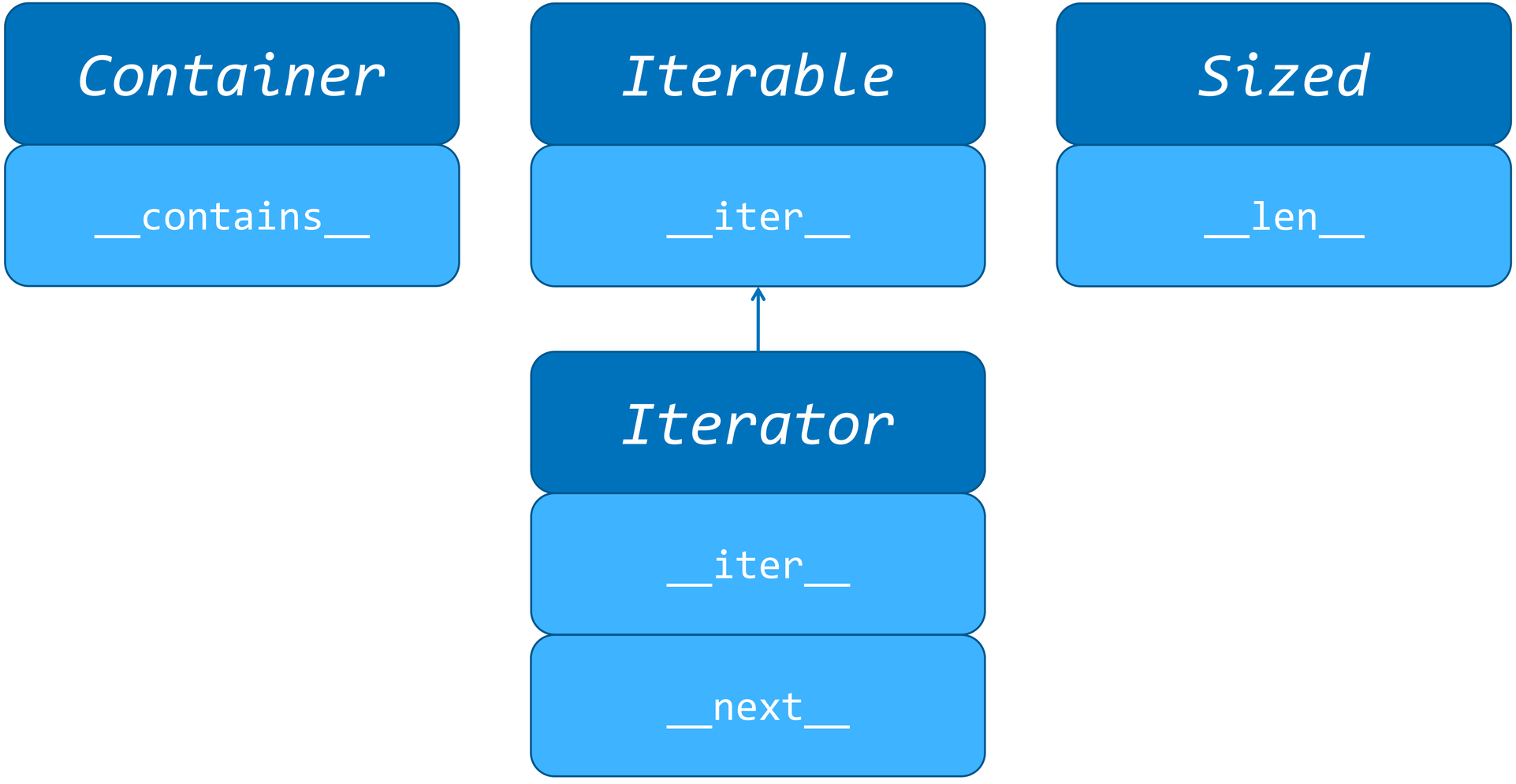
Финальная реализация

```
>>> my_iterable = MyIterable([1, 2])
>>> for _ in my_iterable:
...     for elem in my_iterable:
...         print(elem, end=" ")
1 2 1 2
>>> next(my_iterable)
...
TypeError: 'MyIterable' object is not an iterator
```

StopIteration и секрет цикла for

```
>>> my_iterable = MyIterable([1, 2])
>>> my_iterator = iter(my_iterable)
>>> while True:
...     try:
...         elem = next(my_iterator)
...         print(elem)
...     except StopIteration:
...         break
...
1
2
# for elem in my_iterable:
#     print(elem)
```

Протокол объекта с размером



Реализация объекта с размером

```
class MySized:
    _iterable: list[Any]

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __len__(self) -> int:
        return len(self._iterable)

>>> my_sized = MySized(range(10))
>>> print(len(my_sized))
10
```

Недопустимые действия

```
>>> my_sized = MySized(range(10))
>>> for elem in my_sized:
        ...
TypeError: 'MySized' object is not iterable

>>> 1 in my_sized
...
TypeError: argument of type 'MySized' is not iterable

>>> my_sized[0]
...
TypeError: 'MySized' object is not subscriptable
```

Ограничения

```
class MySized:
    _iterable: list[Any]

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __len__(self) -> int:
        return -1

>>> my_sized = MySized(range(10))
>>> print(len(my_sized))
...
ValueError: __len__() should return >= 0
```

__len__ и __bool__

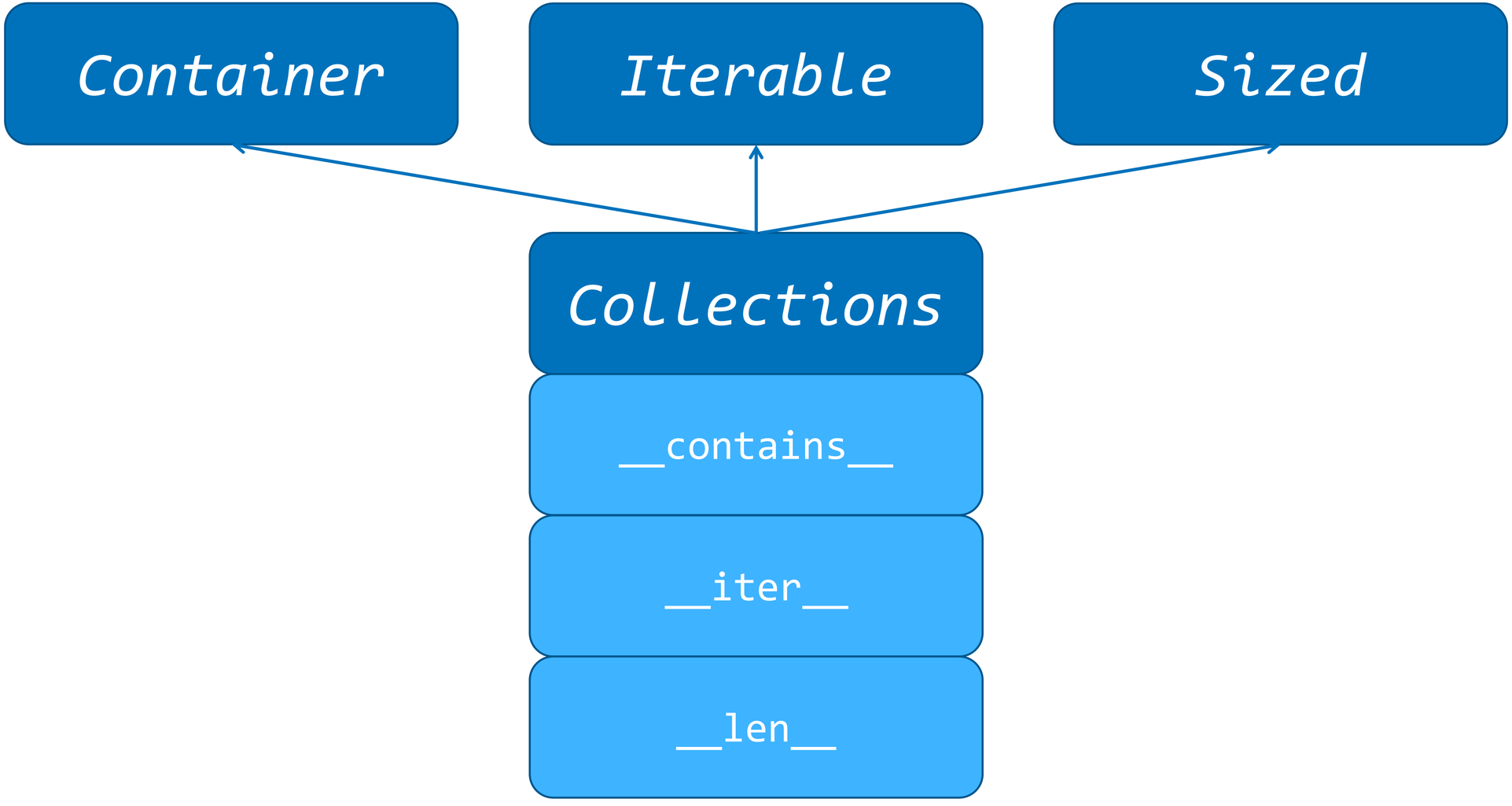
```
class MySized:
    _iterable: list[Any]

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __len__(self) -> int:
        print("len")
        return len(self._iterable)
```

__len__ и __bool__

```
>>> my_sized = MySized(range(10))
>>> print(bool(my_sized))
len
True

>>> my_sized = MySized([])
>>> print(bool(my_sized))
len
False
```

Протоколы коллекций

Протокол коллекции



Реализация коллекции

```
from typing import Any, Iterator, Iterable

class MyCollection:
    _iterable: list[Any]

    def __init__(self, iterable: Iterable) -> None:
        self._iterable = list(iterable)

    def __contains__(self, elem: Any) -> bool:
        return elem in self._iterable

    def __iter__(self) -> Iterator[Any]:
        return iter(self._iterable)

    def __len__(self) -> int:
        return len(self._iterable)
```

Допустимые действия

```
>>> my_collection = MyCollection(range(2))
>>> for elem in my_collection:
...     print(elem)
0
1
>>> print(2 in my_collection)
False
>>> print(len(my_collection))
2
```

Чтение элементов

```
from typing import Any, Iterator, Iterable

class MyCollection:
    ...
    def __getitem__(self, key: int) -> Any:
        if not isinstance(key, int):
            raise TypeError

        if 0 <= key < len(self._iterable):
            return self._iterable[key]
```

Чтение элементов

```
>>> my_collection = MyCollection(range(2))
>>> print(my_collection[1])
1
>>> print(my_collection[30])
None
>>> print(my_collection["32"])
...
TypeError:
```

Проблемы реализации

```
class MyCollection:
    ...
    def __init__(self, iterable: Iterable) -> None:
        ...
    def __contains__(self, elem: Any) -> bool:
        ...
    def __len__(self) -> int:
        ...
    def __getitem__(self, key: int) -> Any:
        print("getitem")
        ...
```

Проблемы реализации

```
>>> my_collection = MyCollection(range(2))
>>> for elem in my_collection:  # бесконечный цикл
...     print(elem)
getitem
0
getitem
1
getitem
None
...
```

Правильное чтение элементов

```
from typing import Any, Iterator, Iterable

class MyCollection:
    ...
    def __getitem__(self, key: int) -> Any:
        if not isinstance(key, int):
            raise TypeError

        if 0 <= key < len(self._iterable):
            return self._iterable[key]

        raise IndexError(key)
```

Правильное чтение элементов

```
>>> my_collection = MyCollection(range(2))
>>> print(my_collection[0])
0
>>> for elem in my_collection:
...     print(elem)
0
1
```

Перезапись элементов

```
from typing import Any, Iterator, Iterable

class MyCollection:
    ...
    def __setitem__(self, key: int, value: Any) -> None:
        if not isinstance(key, int):
            raise TypeError

        if len(self._iterable) <= key < 0:
            raise IndexError(key)

        self._iterable[key] = value
```

Перезапись элементов

```
>>> my_collection = MyCollection(range(2))
>>> print(my_collection[0])
0
>>> my_collection[0] = 42
>>> print(my_collection[0])
42
```

Семинар