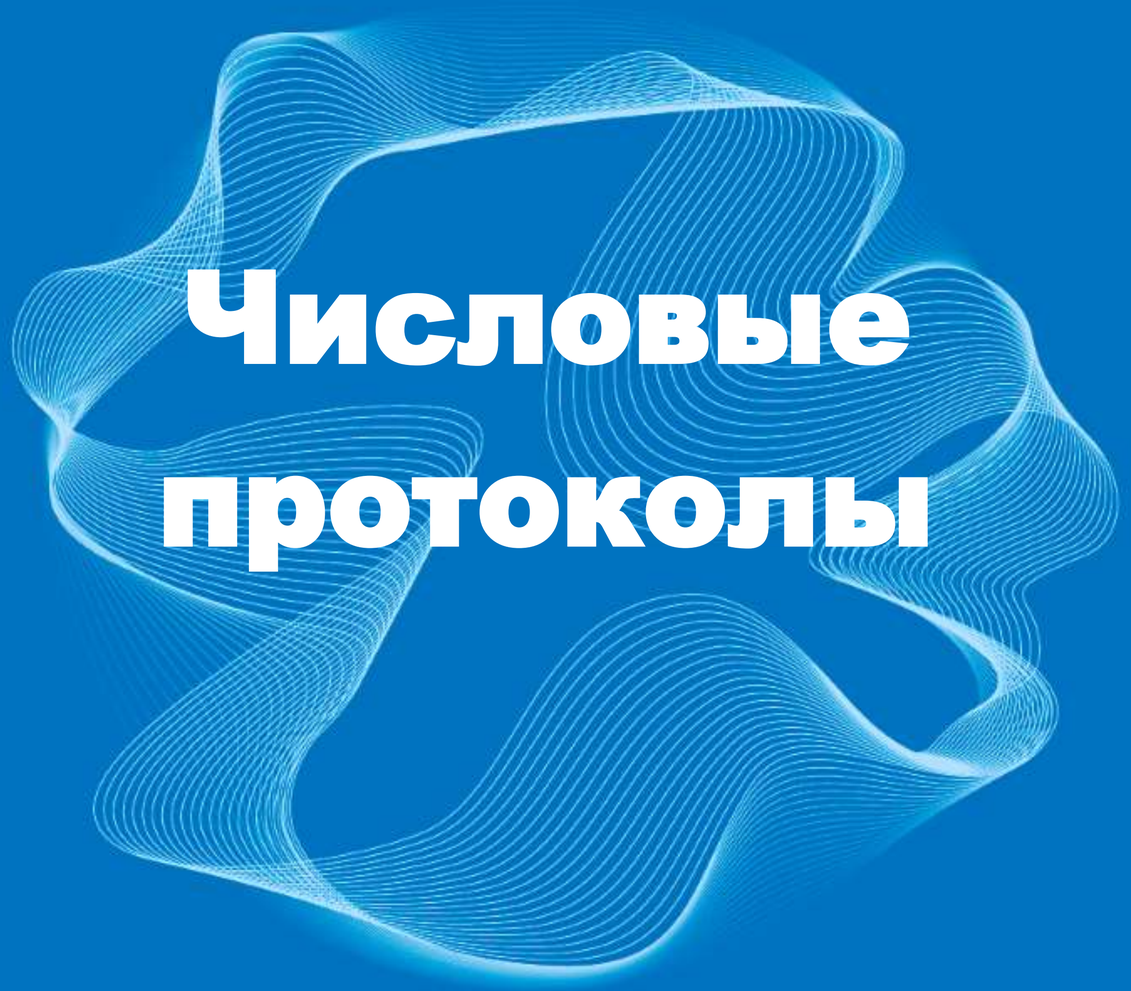
МФТИ



Числовые протоколы

Докладчик: Евграфов Михаил

Протоколы

Динамическая типизация

```
def put_together(
    lhs: object, rhs: object
) -> object:
    return lhs + rhs

>>> print(
...     f"numbers: {put_together(3, 2)};",
...     f"strings: {put_together('Saul ', 'Goodman')};",
...     sep="\n",
... )
numbers: 5;
strings: Saul Goodman;
```

Подтипирование

```
import time

class SleepyHead:
    def sleep(self, sleep_time: int) -> None:
        time_start = time.time()

        while time.time() - time_start < sleep_time:
            print(".", end="")
            time.sleep(1)

        print("")
```

Подтипирование

```
class ProgressSleepyHead(SleepyHead):
    def sleep(self, sleep_time: int) -> None:
        last_string_len = 0
        time_start = time.time()
        time_delta = 0

        while time_delta < sleep_time:
            sleep_percent = min(round(time_delta / sleep_time * 100), 100)
            string = f"sleep: {sleep_percent}%\r"
            last_string_len = len(string)
            print(" " * last_string_len + "\r", end="")
            print(string, end="")
            time.sleep(0.1)
            time_delta = time.time() - time_start

        string = " " * last_string_len + f"\rsleep: 100%\r"
        print(string)
```

Подтипирование

```
def run_sleeper(sleeper_type):
    sleeper = sleeper_type()
    sleeper.sleep(5)
```

```
>>> run_sleeper(SleepyHead)
>>> run_sleeper(ProgressSleepyHead)
.....
sleep: 100%
```

Подтипирование: резюме

```
class SleepyHead:
    def sleep(self, sleep_time: int) -> None:
        ...

class ProgressSleepyHead(SleepyHead):
    def sleep(self, sleep_time: int) -> None:
        ...

def run_sleeper(sleeper_type):
    sleeper = sleeper_type()
    sleeper.sleep(5)

>>> run_sleeper(SleepyHead)                    OK
>>> run_sleeper(ProgressSleepyHead)            OK
```

Структурное подтипирование

```
class SleepyHead:
    def sleep(self, sleep_time: int) -> None:
        ...

class ProgressSleepyHead:  # нет наследования
    def sleep(self, sleep_time: int) -> None:
        ...

def run_sleeper(sleeper_type):
    sleeper = sleeper_type()
    sleeper.sleep(5)

>>> run_sleeper(SleepyHead)                    OK
>>> run_sleeper(ProgressSleepyHead)            OK
```

Пример протокола #1

```
from types import TracebackType

class MyContext:
    def __enter__(self) -> None:
        ...

    def __exit__(
        self,
        exc_type: type[Exception],
        exc_value: Exception,
        exc_tb: TracebackType,
    ) -> None:
        ...

with MyContext():
    pass
```

Пример протокола #2

```
from typing import Sized

def print_len(obj: Sized) -> None:
    print(
        f"object type: {type(obj).__name__}:",
        f"object len: {len(obj)};",
    )

>>> objects = ([1, 2], {"a": 3})
>>> for obj in objects:
...     print_len(obj)
object type: list: object len: 2;
object type: dict: object len: 1;
```

А точно протокол?

```
>>> print(list.__mro__)
>>> print(dict.__mro__)
>>> "__len__" in dir(object)
(<class 'list'>, <class 'object'>)
(<class 'dict'>, <class 'object'>)
False
```

Числовые протоколы

Пользовательский числовой тип

```
class MyBasicNumber:
    _real: float

    def __init__(self, number: float) -> None:
        self._real = float(number)

    def __repr__(self) -> int:
        return f"MyBasicNumber(number={self._real})"
```

real и imag

```
class MyBasicNumber:
    ...
    @property
    def real(self) -> float:
        return self._real

    @property
    def imag(self) -> float:
        return 0.0

>>> my_basic_num = MyBasicNumber(3.14)
>>> print(my_basic_num.real, my_basic_num.imag)
3.14 0.0
```

real и imag: обоснование

```
>>> numbers: list[complex] = [
...     1 + 1j, 3.14, 42, MyBasicNumber(2.72)
... ]

>>> for num in numbers:
...     print(
...         f"number type: {type(num).__name__}; ",
...         f"real: {num.real}, imag: {num.imag}",
...     )
number type: complex;  real: 1.0, imag: 1.0
number type: float;  real: 3.14, imag: 0.0
number type: int;  real: 42, imag: 0
number type: MyBasicNumber;  real: 2.72, imag: 0.0
```

Комплексное сопряжение

```
class MyBasicNumber:
    ...

    def conjugate(self) -> "MyBasicNumber":
        return MyBasicNumber(self._real)
```

Обоснование сопряжения

```
>>> numbers: list[complex] = [
...     1 + 1j, 3.14, 42, MyBasicNumber(2.72)
... ]

>>> for num in numbers:
...     conjugate = num.conjugate()
...     print(
...         f"number type: {type(num).__name__}; ",
...         f"conjugate: {conjugate}",
...     )
number type: complex;  conjugate: (1-1j)
number type: float;  conjugate: 3.14
number type: int;  conjugate: 42
number type: MyBasicNumber;  conjugate: MyBasicNumber(number=2.72)
```

Логические операции

```
class MyBasicNumber:
    ...

    def __bool__(self) -> bool:
        print("bool cast")
        return self._real != 0

    def __eq__(self, other: Complex) -> bool:
        print("eq comparision")
        return self.real == other.real

    def __ne__(self, other: Complex) -> bool:
        print("ne comparision")
        return self.real != other.real
```

__bool__

```
>>> number = MyBasicNumber(2.72)
>>> print(bool(number), number.__bool__())
bool cast
bool cast
True True
```

Важность типа данных

```
class MyBasicNumber:
    ...

    def __bool__(self) -> bool:
        return self._real


>>> number = MyBasicNumber(2.72)
>>> bool(number)
...
TypeError: __bool__ should return bool, returned float
```

__eq__

```
>>> number = MyBasicNumber(2.72)
>>> print(
...     number == MyBasicNumber(2.72),
...     number == 3.14,
...     sep="\n",
... )
eq comparision
eq comparision
True
False
```

__ne__

```
>>> number = MyBasicNumber(2.72)
>>> print(
...       number != MyBasicNumber(2.72),
...       number != 3.14,
...       sep="\n",
... )
ne comparision
ne comparision
False
True
```

Дефолтные __eq__ и __ne__

```
class DumbClass:
    pass

>>> dump_class = DumbClass()
>>> print(
...     dump_class == DumbClass(),
...     dump_class == dump_class,
...     dump_class != DumbClass(),
...     sep="\n",
... )
False
True
True
```

Выражение __ne__ через __eq__

```
class DumbClass:
    def __eq__(self, _: object) -> bool:
        print("eq comparision")
        return True

>>> dump_class = DumbClass()
>>> print(
...     dump_class == DumbClass(),
...     dump_class != DumbClass(),
...     sep="\n",
... )
eq comparision
eq comparision
True
False
```

__abs__ и __neg__

```
class MyBasicNumber:
    ...

    def __abs__(self) -> float:
        print("call abs")
        return abs(self.real)

    def __neg__(self) -> float:
        print("call neg")
        return -self.real
```

__abs__ и __neg__

```
>>> number = MyBasicNumber(-3.14)
>>> print(
...     abs(number),
...     -number,
...     sep="\n",
... )
call abs
call neg
3.14
3.14
```

Аддитивные операции

```
class MyBasicNumber:
    ...

    def __add__(self, other: Complex) -> float:
        if not isinstance(other, (Complex, MyBasicNumber)):
            return NotImplemented

        return self.real + other.real

    def __sub__(self, other: Complex) -> float:
        if not isinstance(other, (Complex, MyBasicNumber)):
            return NotImplemented

        return self.real - other.real
```

Аддитивные операции

```
>>> number = MyBasicNumber(42)
>>> print(
...     number - MyBasicNumber(2.72),
...     number - 3.14,
...     number - 5,
...     sep=", "
... )
39.28, 38.86, 37.0
```

Аддитивные операции: проблема

```
>>> number = MyBasicNumber(42)
>>> print(number - MyBasicNumber(2.72))
>>> print(3.14 - number)
...
TypeError: unsupported operand type(s) for -:
'float' and 'MyBasicNumber'
```

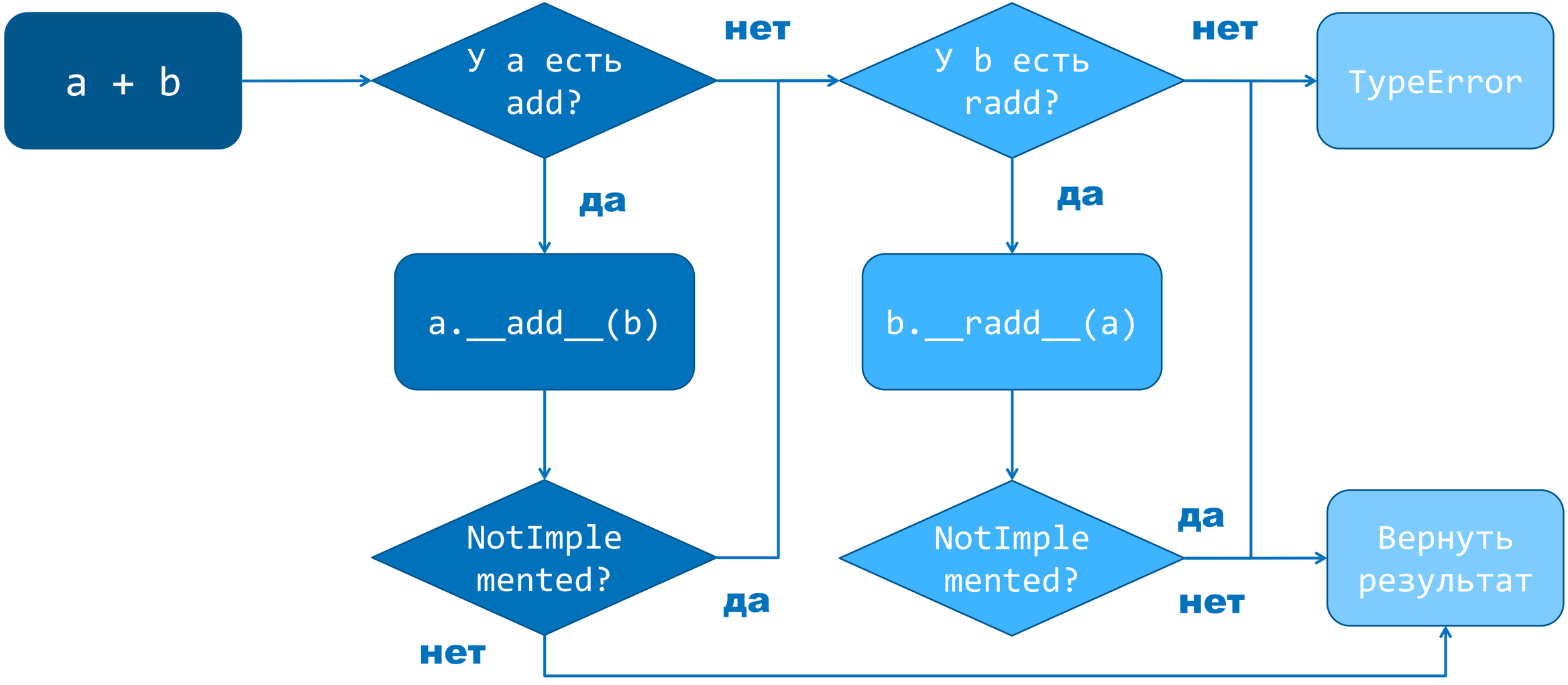
Отраженные аддитивные операции

```
class MyBasicNumber:
    ...

    def __radd__(self, other: Complex) -> float:
        return self + other

    def __rsub__(self, other: Complex) -> float:
        return -self + other
```

Отраженные аддитивные операции

```
>>> number = MyBasicNumber(42)
>>> print(
...       number - MyBasicNumber(2.72),
...       3.14 - number,
...       8 + number,
...       sep=", ",
... )
39.28, -38.86, 50.0
```

Принцип работы бинарных операторов



__mul__ и __rmul__

```
class MyBasicNumber:
    ...

    def __mul__(self, other: Complex) -> float:
        if not isinstance(other, (Complex, MyBasicNumber)):
            return NotImplemented

        return self.real * other.real

    def __rmul__(self, other: Complex) -> float:
        return self * other
```

__mul__ и __rmul__

```
>>> number = MyBasicNumber(2)
>>> print(
...     number * MyBasicNumber(5),
...     number * 5,
...     5 * number,
...     sep=", ",
... )
10.0, 10.0, 10.0
```

__truediv__ и __rtruediv__

```
class MyBasicNumber:
    ...

    def __truediv__(self, other: Complex) -> float:
        return self * 1 / other.real

    def __rtruediv__(self, other: Complex) -> float:
        return 1 / (self / other)
```

__truediv__ и __rtruediv__

```
>>> number = MyBasicNumber(10)
>>> print(
...     number / MyBasicNumber(5),
...     number / 5,
...     5 / number,
...     sep=", ",
... )
2.0, 2.0, 0.5
```

__pow__ и __rpow__

```
class MyBasicNumber:
    ...
    def __pow__(
        self, other: Complex, mod: Optional[Complex] = None
    ) -> float:
        if not isinstance(other, (Complex, MyBasicNumber)):
            return NotImplemented

        return pow(self.real, other.real, mod)

    def __rpow__(
        self, other: Complex, mod: Optional[Complex] = None
    ) -> float:
        if not isinstance(other, (Complex, MyBasicNumber)):
            return NotImplemented

        return pow(other.real, self.real, mod)
```

__pow__ и __rpow__

```
>>> number = MyBasicNumber(2)
>>> print(
...     number ** MyBasicNumber(5),
...     number ** 5,
...     5 ** number,
...     sep=", ",
... )
32.0, 32.0, 25.0
```

Преобразования типа

```
class MyBasicNumber:
    ...

    def __complex__(self) -> complex:
        return complex(real=self.real, imag=self.imag)

    def __float__(self) -> float:
        return self.real

    def __int__(self) -> int:
        return int(self.real)
```

Преобразования типа

```
>>> number = MyBasicNumber(2)
>>> print(
...     f"complex: {complex(number)}",
...     f"float: {float(number)}",
...     f"int: {int(number)}",
...     sep="\n",
... )
complex: (2+0j)
float: 2.0
int: 2
```

Дальнейшие пути развития

- **Логические операторы порядка:** `<`, `<=`, `>`, `>=`
- **Битовые операторы:** `&`, `|`, `^`, `~`, `<<`, `>>`
- **Округления:** `round`, `floor`, `ceil`, `trunc`
- **Модульное и целочисленное деление**
- **Составное присваивание**

Семинар