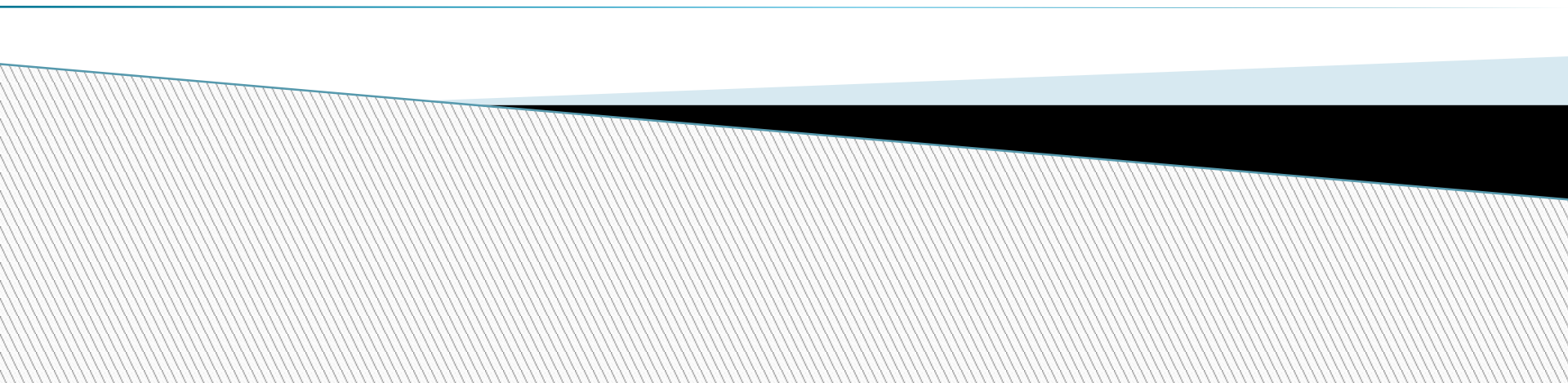


Основы программирования на Python 3

ООП, Классы в Python



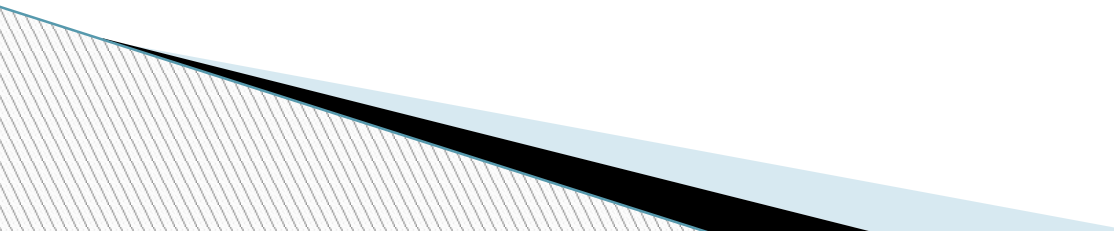
Классы - зачем и как?

- ▶ Удобная, понятная человеку структуризация кода и данных
- ▶ Переиспользование кода

Простейший синтаксис

```
class A:  
    pass
```

Рекомендации по наименованию PEP-8:
UseTitleNames



Python - все есть объект

- ▶ Все сущности в Python3, в том числе пользовательские классы наследуются от базового класса **object**

```
class A:  
    pass
```

```
class B(object):  
    pass
```

```
issubclass(A, object)
```

```
True
```

```
issubclass(B, object)
```

```
True
```



Что могут содержать классы

- ▶ Атрибуты данных

```
class A:  
    a = 1
```

A.a

1

- Атрибуты-методы

```
class A:  
    def f():  
        return 1
```

A.f()

1

Экземпляры класса

- ▶ Класс - описание множества объектов.
Например, "человек":

```
class Human:  
    pass
```

Теперь можно сделать "экземпляры", то есть конкретных людей:

```
Вася = Human()  
Петя = Human()  
Ирина = Human()
```

Конструктор класса `__init__`

- ▶ Было бы грустно, если бы все экземпляры `Human` были одинаковыми.
- ▶ На этапе проектирования следует решить, каким атрибутами будет обладать каждый экземпляр класса `Human`

```
class Human:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

Конструктор класса `__init__`

```
Вася = Human('Вася', age=21)
Петя = Human(name='Петя', age=13)
print(Вася.name, Вася.age)
print(Петя.name, Петя.age)
```

- Атрибуты всегда известны при проектировании класса
- Конструктор **всегда** вызывается при создании экземпляра
- Архитектурно правильно задавать **новые** атрибуты только в конструкторе

Конструктор `__init__`

- ▶ Параметры, передаваемые в конструктор не обязательно должны быть присвоены атрибутам класса. Это обычные переменные

```
class Human:
    def __init__(self, name, age):
        self.name = name
        if age > 20:
            self.age = 'Старик'
        else:
            self.age = 'Молодой'
```

- `__двойные__` подчеркивания с двух сторон указывают на то, что метод "магический"

Пользовательские методы объекта.

self

- ▶ Параметр **self** - служебный и всегда указывает на экземпляр класса. Нельзя вызвать функцию экземпляра не имея экземпляра

```
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def is_old(self):
        return self.age >= 20
```

```
>>> h1 = Human('Вася', 10)
>>> h2 = Human('Иван Никанорович', 56)
>>> Human.is_old()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: is_old() missing 1 required positional argument:
'self'
>>> h1.is_old(), h2.is_old()
(False, True)
```

"Магические" методы

```
▶ class Human:
    def __init__(self, name):
        self.name = name

    def __add__(self, other):
        return Human(name='{}
        {}'.format(self.name, other.name))

    def __str__(self):
        return self.name

▶ >>> h1 = Human('Вася')
>>> h2 = Human('Маша')
>>> h3 = h1 + h2
>>> print(h3)
'Вася Маша'
```

"Магические" методы

- ▶ `__add__` Сложение
- ▶ `__str__` (и `__unicode__` для совместимости)
Преобразование к строке
- ▶ `__sub__` Вычитание
- ▶ `__mul__` Умножение
- ▶ `__divmod__`, `__truediv__` Деление
- ▶ `__cmp__` Сравнение
- ▶ `__call__` Вызов "как функции"
- ▶ И другие

Методы класса и статические методы

- ▶ Иногда нужно, чтобы у всех экземпляров вызывался один метод
- ▶ Либо вызывать метод **без** создания экземпляров

```
class Human:  
    @classmethod  
    def class_name(cls):  
        print('Человечество')
```

```
    @staticmethod  
    def some_method():  
        """
```

*Просто метод, который имеет что-то общее с
логикой human,
поэтому его не целесообразно выносить в отдельную
функцию*
 """

Наследование

- ▶ Самая "мякотка" классов

```
class Woman:  
    def __init__(self, name, age,  
dress_color):  
        self.name = name  
        self.age = age  
        self.dress_color =  
dress_color
```

- Неудобно, код дублируется

Наследование. `super()`

```
▶ class Woman(Human):  
    def __init__(self, name, age,  
dress_color):  
        self.dress_color = dress_color
```

▶ Все еще плохо – `name` и `age` не задаются

```
class Woman(Human):  
    def __init__(self, name, age,  
dress_color):  
        super().__init__(name, age)  
        self.dress_color = dress_color
```

Наследование. `super()`

- ▶ `super()` - получение объекта вида "а что если бы элемент этого класса был элементом родительского"
- ▶ Результатом выполнения `super()` является класс-родитель, методы которого "привязаны" к текущему объекту
- ▶ Поэтому можно вызывать метод `__init__()` без `self`
- ▶ Раньше методу `super()` явно передавались класс и экземпляр

Множественное наследование

```
class A:  
    def x(self):  
        print(1)  
  
    def y(self):  
        print(3)
```

```
class B:  
    def x(self):  
        print(2)
```

```
class C(A, B):  
    pass
```

- Наследование "слева направо" - то что слева - имеет приоритет и перезапишет то что справа

Исключения - зачем?

- ▶ Защита программы от непредвиденного поведения в рантайме
- ▶ Безусловная передача информации об ошибке

Синтаксис

```
▶ try:
    a = int('Это точно не инт')
except ValueError as e:
    print('Что-то пошло не так! %s' %
e)
else:
    print('Все пошло как надо'))
finally:
    print('В конце я всегда уведомляю
хозяина')
```

Синтаксис

- ▶ Обязателен хотя бы один блок `except` или `finally`
- ▶ Можно отлавливать сразу несколько типов ошибок:

- ▶ **try:**

```
    a = 1 / 0
```

```
except (ZeroDivisionError,  
TypeError):
```

```
    print( 'Нехорошо делить на 0! ' )
```

Безусловная передача ошибки

```
▶ a = int(input())
```

```
    if a < 0:  
        raise Exception('Недопустимы  
отрицательные числа')
```

- ▶ Ошибка будет передаваться по стеку вызовов функций, пока не будет отловлена или программа не завершится аварийно
- ▶ Соблазнительно для передачи внутренних ошибок, но следует использовать аккуратно – вызов дорогой

Широкий диапазон ошибок - плохо

- ▶ Все ошибки в Python наследуются от базового класса Exception
 - ▶ Как следствие, except Exception позволит поймать любую ошибку
 - ▶ Например, незапланированный RuntimeError или KeyboardInterrupt
- 