Advanced Python. Introduction to OOP.

Терминология

- Тип, класс
- Объект, экземпляр (instance)
- Атрибут
- Метод
- Инкапсуляция
- Наследование
- Полиморфизм

Наследование

- Наследование или специализация
- Базовый класс, суперкласс
- Дочерний класс
- Все классы в Python явно или неявно унаследованы от базового класса **object**

Наследование (2)

- Переопределение
- Динамическое связывание, полиморфизм
- Duck typing

Нет в Python

- **Перегрузка методов** (method overloading). Может быть реализовано самостоятельно - isinstance()
- **Управление доступом**. Специальные имена переменных.

Объявление класса

Для объявления класса используется ключевое слово **class.**

```
class ClassName:
   ...
```

При наследовании базовый класс (классы) указываются после имени класса:

```
class ClassName(BaseClass):
   ...
```

Объявление класса (2)

Классы без явного указания родительского класса неявно наследуются от **object.** Эти объявления эквивалентны:

```
class ClassName:
    …

class ClassName(object):
    …
```

Атрибуты, инициализация

Для инициализации класса используется специальный метод **__init__**

```
class Point
  def __init__(self, x=0, y=0):
    self.x = x
    self.y = y
```

Методы

- В качестве первого аргумента метод принимает экземпляр класса.
- По соглашению имя аргумента - **self**

```
class Point:
  ...
  def __str__(self):
    return 'Point' + str((self.x, self.y))
```

Создание объекта класса

Для создания экземпляра (объекта) класса используется синтаксис, аналогичный синтаксису вызова функции. При этом вызывается специальные методы, конструктор __new__ и инициализатор __init__

```
p1 = Point()
p2 = Point(3, 4)
p3 = Point(x=1, y=2)
```

Специальные методы

- __init__
- __new__
- __str__
- __repr__
- __eq__
- __next__, __iter__ (генераторы, итераторы)
- __call__ (функторы)

Доступ к родительскому классу

Для получения доступа к реализации метода родительского класса, используется функция **super()**

```
class Parent:
  def test(self):
    print('Parent')

class Child(Parent):
  def test(self):
    super().test()
    print('Child')
```

Множественное наследование

В Python класс может быть унаследован от более чем одного базового класса

```
class A:
   ...

class B:
   ...

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

Множественное наследование (2)

Порядок поиска метода для вызова определяется процедурой **MRO** (method resolution order)

Чтобы посмотреть MRO для класса, можно воспользоваться “магической” переменной **__mro__** или методом **mro():**

C.__mro__
C.mro()

Абстрактные классы

- Абстрактный класс определяет методы, которые должны быть реализованы в дочерних классах
- В Python для реализации абстрактных классов используется пакет abc
- Абстрактный класс должен быть унаследован от abc.ABC, или использован метакласс abc.ABCMeta
- Абстрактные методы должны быть задекорированы abc.abstractmethod

Статические и класс- методы

- @staticmethod - метод класса, обычная функция без специальных аргументов, которая ничего не знает про свой класс
- @classmethod - метод класса, который получает класс (не экземпляр!) в качестве первого аргумента

Property

- getters/setters - специальное соглашение для методов, осуществляющим доступ к атрибутам класса
- @property - функция/декоратор упрощающая реализацию и использование этой идиомы
- Плохо совместима с наследованием

Исключения

Перехват и обработка исключений

Простые формы:

```
try:
   statements*
except:
  …

try:
   statements*
finally:
  …
```

Перехват и обработка исключений

Полная форма:

```
try:
   statements*
except Exception1 as e1:
   …
except Exception2 as e2:
  …
else:
 …
finally:
 …
```

Создание и возбуждение исключений

Для создания исключения необходимо объявить наследника класса Exception:

```
class MyException(Exception):
  pass
```

Для возбуждения исключений используется инструкция **raise:**

```
raise MyException
```