Понятие объекта и класса

Простые объекты

Объектно-ориентированный *подход предполагает, что в работа программы определяется взаимодействием различных объектов в ответ на обращение к каждому из них.*

При процедурном программировании каждая задача в соответствии с постановкой и условиями разбивалась на отдельные алгоритмы, которые могли быть использованы последовательно один или много раз. При этом в процессе работы каждого алгоритма изменяются значения тех или иных переменных, которые служат для обмена данными между алгоритмами.

Можно представить, что все переменные, которые используются в алгоритмах - это простейшие объекты, к которым обращается <<высший разум>> через действия алгоритма.

В Python сущность объектов определяется через описание класса объектов. Класс позволяет породить хотя бы один объект. Если удалось получить хотя бы один объект, можно получить сколько угодно других (пока позволяют ресурсы).

При этом все объекты имеют свой собственный набор атрибутов (переменных и данных) и методов (функций).
Атрибут — это переменные и их значения, которые характеризуют объект (объект "окружность" характеризуют координаты центра окружности и ее радиус).
Метод — это функция, которая принадлежит непосредственно объекту и имеет доступ ко всем атрибутам этого объекта. (У объекта "окружность" это могут быть вычисления площади круга, длины окружности, функции по изменению атрибутов объекта (увеличить в k раз, подвинуть и тп).

Объект обладает:

1. идентичностью,
2. набором атрибутов, значения которых определяет **состояние** объекта,
3. и поведением, которое определяется методами, присущими объекту.

Совокупность объектов, обладающих одинаковым набором атрибутов и поведением определяет общий для них ***класс или понятие****.*

В Python сущность объектов определяется через описание класса подобных объектов. Класс позволяет породить хотя бы один реальный объект. Если удалось породить хотя бы один объект, то можно получить сколько угодно других объектов (если позволяют ресурсы)

Рассмотрим простейшую задачу.

Задача.

Время задается часами и минутами, которые отображаются на 24-часовом циферблате. Первоначально хотелось бы иметь такой инструмент, который:

1. для новых часов устанавливал бы их в 00:00
2. отображал бы время, указанное на 12-часовом циферблате в показаниях этих часов.

Понятно, что нам нужен объект, у которого есть атрибут <<час>> и атрибут <<минуты>>.

Процедурный и объектно-ориентированный подход

Процедурный подход хорош для небольших (до 500 строк) проектов. Объектно-ориентированно программировать можно и на ассемблере. Но трудно. Удобнее программировать на языке, где есть для этого возможности.

Посмотрим, что есть в питоне для объектно-ориентированного подхода.

Для задания окружности нужно задать координаты центра окружности x, y и ее радиус r. Для этого можно использовать обычный кортеж.

```
circle = (1, 2, 15)
```

Проблемы:

- не очевидно, где тут радиус. Может быть (x, y, r), а может мы задавали (r, x, y)
- если есть функции `distance_from_origin(x, y)` и `edge_distance_from_origin(x, y, radius)`, при обращении к ним нам нужно распаковывать кортеж (лишние операции):

```
distance = distance_from_origin(*circle[:2])
distance = edge_distance_from_origin(*circle)
```

Решение: именованный кортеж (named tuple):

```
import collections
Circle = collections.namedtuple("Circle", "x y radius")
circle = Circle(13, 84, 9)
distance = distance_from_origin(circle.x, circle.y)
```

Проблемы:

- можно создать кортеж с отрицательным радиусом (нет проверки значений) (при процедурном подходе эти проверки либо не делаются, либо требуют написания слишком много кода);
- tuple неизменяем, named tuple изменяем через метод `collections.namedtuple_replace()` (но код ужасает)

```
circle = circle._replace(radius=12)
```

Решение: взять коллекцию с изменяемыми значениями, например, list или dict.

```
circle = [1, 2, 15]
circle = dict(x=36, y=77, radius=8)  # тоже нет контроля данных (r<0); можно
передать в функцию, которая не должна работать с
```

- list
  - нет доступа к circle['x'],
  - можно сделать circle.sort()
- dict
  - еще нет защиты от отрицательного радиуса,
  - еще можно передать в функцию, которая не работает с окружностями (а, например, ждет прямоугольник).

Термины

- **Модель** — упрощенное описание

- **Объект** -
  - Переменные (характеризуют состояние объекта)
  - Методы (могут состояния менять)

- **Класс** – способ задания *ОДНОТИПНЫХ* объектов
  - прототип объекта (его переменные)
  - метод создания из прототипа конкретного объекта

Еще раз "на пальцах":

- Класс - **шаблон** объекта,
- Объект - **экземпляр** класса.

Одинаковое значение:

- **Класс** или **тип** или **тип данных** - шаблон создания данных. Описывает атрибуты и методы работы с ними.
- **Объект** или **экземпляр [класса]**
- **атрибуты** или **поля** класса. Это переменные и константы (атрибуты данных) и *вызываемые* атрибуты (функции).
- **метод** класса или **функция** класса

5 - это объект класса int
'Hello' - это объект класса str

Создаем свой класс

- Для создания класса используют ключевое слово **class**
- поля и методы пишут с отступом;
- имя класса принято писать с большой буквы.

Класс записывается так (будем использовать сегодня):

```
class ИмяКласса:
    поля и методы класса
```

или так (будем использовать, когда начнем говорить о наследовании):

```
class ИмяКласса(базовые классы через запятую):
    поля и методы класса
```

*На самом деле, когда базовый класс не указывается явно, это класс **object**.*

Самый простой класс (мы использовали для создания своего исключения):

```
class MyOwnException(Exception):    # мой класс исключений наследуется от базового класса Exception
    pass                            # ничего дополнительного или особого он не делает
```

Пример класса, описывающего окружности:

```
from math import PI
class Circle:
    # конструктор класса, вызывается когда создаем новый объект класса
    def __init__(self, x=0, y=0, r=0):
        self.x = x                      # все переменные объекта указываются
в конструкторе
        self.y = y
        self.r = r

    # методы объекта:
    def area(self):                     # первый аргумент всегда self
        return PI * self.r * self.r  # доступ к аргументам - только через
self.

    def perimetr(self):                 # первый аргумент всегда self
        return 2*PI * self.r            # доступ к аргументам - только через
self.

    def zoom(self, k):                  # увеличим окружность в k раз
        self.r *= k

    def is_crossed(self, c):            # пересекается или нет эта окружность
с окружностью c?
        d2 = (self.x - c.x)**2 + (self.y - c.y)**2
        r2 = (self.r + c.r)**2
        return d2 <= r2

# тут можно уже создавать объекты класса и их использовать
```

- `__init__(self, другие аргументы через запятую)` - конструктор класса (не совсем так, там еще есть ***new***, но об этом подробнее в наследовании и переопределении методов).
- **self** - ссылка на себя, через нее доступаемся к атрибутам (полям и методам)

Создание объекта класса

```
c = Circle(1, 2, 3)
```

- Создан объект класса Circle с центром окружности в (1, 2) и радиусом 3.
- переменная ***c*** **ссылается** на этот объект.

Доступ к полям и методам

ссылка_на_объект.поле

ссылка_на_объект.метод

```
c = Circle()
c.x = 1
c.y = 2
c.r = 3
a = c.area()    # у объекта, на который ссылается c, вызвали метод area
```

Объект и ссылка (повторение - мать учения)

```
c = Circle()
c.x = 1
c.y = 2
c.r = 3
a = c.area()    # у объекта, на который ссылается c, вызвали метод area

d = Circle(5, 6, 2.5)
a = c.area() + d.area()
```

Нарисовать на доске картинку про объекты и ссылки на них.

c = d - это присвоение ссылок,

Что будет напечатано?

```
c = Circle()
d = Circle()
c.x = 1
d.x = 6
print('c.x = ', c.x)
print('d.x = ', d.x)

c = d
print('c.x = ', c.x)
print('d.x = ', d.x)
```

Method overloading

Можно ли создать в питоне методы класса с одинаковыми именами?

Нет. В python так не пишут.

Вы можете использовать значения по умолчанию, *args и **kargs - в питоне есть другие механизмы для того же результата.

Можно использовать декоратор @overload, но о декораторах расскажем позже.

Можно задать их

-- TatyanaDerbysheva - 04 Nov 2017