Класс и его атрибуты

Предположим, требуется смоделировать банковский счёт с поддержкой операций снятия и пополнения счёта. Один способ достичь этого — использовать глобальную переменную для хранения баланса.

```
balance = 0

def deposit(amount):
    global balance
    balance += amount

def withdraw(amount):
    global balance
    balance -= amount

deposit(10)
withdraw(5)
```

В этом примере объявлена глобальная переменная balance и две функции с параметром amount. В функциях сначала вызывается оператор global для возможности изменения глобальной переменной, затем на величину amount изменяется balance, и наконец, возвращается обновлённое значение переменной balance.

Данный пример моделирует только один банковский счёт, но если требуется работать с несколькими банковскими счётами можно использовать словарь для хранения данных.

```
def make_account():
    return {'balance': 0}

def deposit(account, amount):
    account['balance'] += amount

def withdraw(account, amount):
    account['balance'] -= amount

a = make_account()
deposit(a, 50)
withdraw(a, 10)
```

Лучшим вариантом для описания банковского счёта будет являться использование объектно-ориентированного подхода и создание класса, который описывается при помощи слова class.

```
class BankAccount(object):
    pass

ba = BankAccount()
```

При описании BankAccount указываем родительский класс object и при помощи оператора pass ничего пока не делаем. Создание экземпляра класса похоже на вызов функции. Возвращаемое значение является ссылкой на объект BankAccount. Расширим описание класса путём добавления атрибутов, содержащих номер, баланс.

```
class BankAccount(object):

    number = 0
    balance = 0
```

Ссылки на атрибуты используют стандартный синтаксис в формате имя_объекта.имя_атрибута.

```
ba = BankAccount()
ba.number = 1
ba.balance = 100
```

Такая же нотация применяется для доступа к значениям атрибутов объекта.

```
print(ba.balance)
```

Обычно классы создают экземпляры, имеющие определённые начальные значения. Для этого класс может содержать специальный метод инициализации __init__(self). При этом описание атрибутов из класса можно убрать. Параметр self — сам объект, к которому применяется метод. Ссылки на атрибуты в методах всегда должны производится относительно имени self.

```
class BankAccount(object):

    def __init__(self, number, balance):
        self.number = number
        self.balance = balance
```

Для создания объекта BankAccount с __init__(self).

```
ba = BankAccount(1, 100)
```

Другой тип ссылок на атрибуты экземпляра класса — это метод. Методы семантически похожи на функции, в которые можно передавать аргументы и возвращать результат. Таким образом, методы описываются внутри класса и они явным образом вызываются у сущности класса.

```
class BankAccount(object):

    def __init__(self, number, balance):
        self.number = number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

ba = BankAccount(1, 100)
ba.deposit(50)
print(ba.balance)
```

Отличительная особенность методов состоит в том, что в качестве первого аргумента функции передаётся объект. Например, вызов x.f() полностью эквивалентен вызову MyClass.f(x). В общем случае, вызов метода со списком из n аргументов эквивалентен вызову соответствующей функции со списком аргументов, созданным за счёт вставки объекта, вызвавшего метод, перед первым аргументом.

Когда запускается модуль Python в виде, то код в этом модуле будет исполнен в момент его импортирования, но значением __name__ будет строка «__main__». Это значит, что добавляя следующий код в конец модуля

```
if __name__ == '__main__':
```

можно сделать возможным запуск модуля (файла) в качестве сценария, и в качестве

импортируемого модуля. Это возможно, поскольку разбирающий командную строку код выполняется только при исполнении модуля как основного (main) файла. Если модуль импортируется, код не будет выполнен. Такой приём часто используется, чтобы предоставить удобный пользовательский интерфейс к модулю или для тестирования (выполнение модуля в качестве сценария запускает набор тестов).

```
class BankAccount(object):

    def __init__(self, number, balance):
        self.number = number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

if __name__ == '__main__':
    ba = BankAccount(1, 100)
    ba.deposit(50)
    print(ba.balance)
```

Задания:

- составьте классы, согласно своего варианта индивидуального задания, с необходимыми атрибутами для хранения значений;
- добавьте методы в классы для получения данных из атрибутов и устанавливающих значения в них;
- используя конструкцию if __name__ == '__main__': проверьте корректность работы с составленными классами.