Лабораторная работа № 4: Объектно-ориентированное программирование

1. Создайте файл `lab_04_01.py`:

```
import time

class Ticket:

    def __init__(self, date, name,deadline):
        self.createDate = date
        self.owner = name
        self.deadline = deadline

    def __del__(self):
        print("Delete ticket: ", time.asctime(self.createDate))

    def display(self):
        print("Ticket:")
        print(" createDate: ",time.asctime(self.createDate))
        print(" owner: ",self.owner)
        print(" deadline: ",time.asctime(self.deadline))

# создание объект класса
ticket1 = Ticket(time.localtime(),"Ivan Ivanov", \
                    time.strptime("17.12.2017", "%d.%m.%Y"))
# вызов метода
ticket1.display()
# получение значения атрибута
print("Owner: ", ticket1.owner)
print("Owner(getattr): ", getattr(ticket1,"owner"))
# проверка наличия атрибута
print("hasattr: ", hasattr(ticket1,"owner"))
setattr(ticket1,"owner","Alexei  Petrov")  #  установка  значения
атрибута
print("Owner(setattr): ", ticket1.owner)
# delattr(ticket1,"owner") # удаление значения атрибута
# print("delattr: ", ticket1.owner)

# del ticket1 # удаление объекта
# print(ticket1)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
Ticket:
 createDate:  Thu Mar  9 14:53:55 2017
 owner:  Ivan Ivanov
 deadline:  Sun Dec 17 00:00:00 2017
Owner:  Ivan Ivanov
Owner(getattr):  Ivan Ivanov
hasattr:  True
```

```
Owner(setattr):  Alexei Petrov
Delete ticket:  Thu Mar  9 14:53:55 2017
```

Классы и объекты классов

Python является объектно-ориентированным языком программирования. В объектно-ориентированном программировании основными понятиями являются «класс» и «объект» класса. Класс представляет собой категорию, к которой можно отнести множество объектов, обладающих характерными признаками и выполняющих одинаковые действия. Объявление класса происходит с использованием ключевого слова `class` и указанием имени класса (имена классов начинаются с большой буквы). Класс может содержать атрибуты и методы (функции, описывающие действия его объектов). В языке Python объявление класса происходит следующим образом:

```
class Ticket:

    def __init__(self, date, name, deadline):
        pass

    def __del__(self):
        pass

    def display(self):
        pass
```

В качестве первого аргумента методов выступает ссылка на текущий объект класса `self` (аналогичный `this` в большинстве объектно-ориентированных языках программирования). При вызове методов в передаваемых аргументах объект класса не указывается, например:

```
t = Ticket(date, name, deadline)
```

где `t` – объект класса `Ticket`.

Существует два базовых метода, предусмотренных для всех создаваемых классов:

- `__init__` – конструктор класса.
- `__del__` – деструктор класса

Атрибуты класса задаются внутри его методов. Атрибуты, заданные вне методов, являются общими для всех объектов класса и могут изменять свои значения в случае их редактирования.

Вызов методов класса осуществляется с указанием имени метода через точку после имени объекта, от которого вызывается метод. Также существуют методы, предоставляющие возможности работы с атрибутами классов:

- `hasattr(obj,attr)` – проверка наличия атрибута `attr` у объекта `obj`
- `setattr(obj,attr,value)` – установка значения `value` атрибута `attr` у объекта `obj`
- `getattr(obj,attr)` – получение значения атрибута `attr` у объекта `obj`
- `delattr(obj,attr)` – удаление атрибута `attr` у объекта `obj`

Методы класса объявляются как функции внутри класса.

Работа с подключаемой библиотекой

Для подключения внешних библиотек в языке программирования Python используется ключевое слово `import`. Подключение самостоятельной библиотеки или модуля происходит следующим образом:

```
import name1, name2, …
```

где `name1, name2, …` – имена подключаемых библиотек/модулей.

Для подключения библиотеки из какого-либо пакета используется следующий формат записи:

```
from package import name1, name2, …
```

где `package` – имя пакета, `name1, name2, …` – имена подключаемых библиотек/модулей.

Для сокращения названий подключаемых библиотек можно указывать собственные названия после ключевого слова `as`:

```
from package import name1 as N
```

где `N` – собственное название подключаемой библиотеки.

Для работы с датами и временем используется библиотека `time`. Основными функциями данной библиотеки являются:

- `time()` – получение времени в системе Unix-время
- `localtime()` – получение текущего времени машины в виде кортежа
- `asctime(timeT)` – автоматическое приведение времени, представленного кортежем (`timeT`), к формату:
  `Thu Mar  9 14:53:55 2017`
- `strptime(str, format)` – создание кортежа с данными времени по строке `str`, представленной в формате `format`
- `strftime(timeT, format)` – представление кортежа с данными времени `timeT` в виде строки в формате `format`

Со списком форматов для функций `strptime()` и `strftime()`, а также списком остальных функций библиотеки можно ознакомиться по ссылке:
`https://docs.python.org/3/library/time.html`

2. Модифицируйте код программы `lab_04_01.py`. Раскомментируйте строки:
`'# delattr(ticket1,"owner") # удаление значения атрибута`
`# print("delattr: ", ticket1.owner)'`.
Объясните поведение программы. Исправьте программу с помощью стандартных функций так, чтобы ошибка не возникала.

3. Модифицируйте код программы `lab_04_01.py`. Раскомментируйте строки:

```
'# del ticket1 # удаление объекта
# print(ticket1)'.
```
Объясните поведение программы.

4. Дополните код программы `lab_04_01.py`. Получите текущее время сервера (или компьютера) и выведите его на экран в аналогичном формате: `9 Mar 2017 14:53:55`.

5. Дополните код программы `lab_04_01.py`. Создайте объект типа `time` по следующей строке с использованием соответствующей функции библиотеки `time`: `17.07.2017 10:53:00`.

6. Создайте файл `lab_04_02.py`:

```
class Worker:
    'doc class Worker'
    count = 0

    def __init__(self,name,surname):
        self.name = name
        self.surname = surname
        Worker.count += 1

    def display(self):
        print("Worker:")
        print("{} {}".format(self.name,self.surname))

w1 = Worker("Ivan", "Ivanov")
print("w1.count: ", w1.count)
w2 = Worker("Alexei", "Petrov")
print("w2.count: ", w2.count)
print("w1.count: ", w1.count)
print("Worker.count: {0} \n".format(Worker.count))
print("Worker.__name__: ", Worker.__name__)
print("Worker.__dict__: ", Worker.__dict__)
print("Worker.__doc__: ", Worker.__doc__)
print("Worker.__bases__: ", Worker.__bases__)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
w1.count:  1
w2.count:  2
w1.count:  2
Worker.count: 2

Worker.__name__:  Worker
Worker.__dict__:       {'count':    2,    'display':    <function
Worker.display  at  0x7fbc0afb5a60>,  '__module__':  'builtins',
```

```
'__dict__': <attribute '__dict__' of 'Worker' objects>,
'__init__': <function Worker.__init__ at 0x7fbc0afb5ae8>,
'__weakref__': <attribute '__weakref__' of 'Worker' objects>,
'__doc__': 'doc class Worker'}
Worker.__doc__:  doc class Worker
Worker.__bases__:  (<class 'object'>,)
```

Общие атрибуты класса

Общие атрибуты класса доступны всем объектам данного класса. Значения, установленные для общих атрибутов класса, могут быть впоследствии отредактированы, что повлечет за собой их изменения во всех объектах класса. Общие атрибуты задаются вне методов класса:

```
class Worker:
    count = 0
```

Доступ к общим атрибутам класса может быть получен как от объектов класса, так и из самого класса, например:

```
w1.count
Worker.count
```

Описание класса (документация) добавляется в виде строки сразу после начального объявления класса:

```
class Worker:
    'doc class Worker'
    …
```

Для получения информации о классе используются следующие базовые атрибуты:

- `__name__` – получение имени класса в виде строки
- `__dict__` – получение сведений о всех доступных методах и общих атрибутах класса
- `__doc__` – получение строки с описанием (документацией) класса
- `__bases__` – классы, от которых был унаследован данный класс. По умолчанию все классы наследуются от общего класса, представляющего объекты (`object`)

7. Дополните код программы `lab_04_02.py`. Создайте класс `Animal`, обозначив для него в конструкторе атрибуты `name`, `age` и общий атрибут `id`, который будет увеличиваться на единицу при создании каждого нового объекта класса. Определите также метод `display()`, осуществляющий вывод на экран информации по объекту класса в следующем формате, подставив нужные значения:

```
Animal id:
 Name: name
 Age: age
```

Создайте три объекта класса `Animal`, заполнив их произвольными значениями `name` и `age`. Осуществите вывод информации об объектах на экран с помощью функции `display()`.

8. Дополните код программы `lab_04_02.py`. По выведенной на прошлом шаге информации об объектах объясните поведение атрибута `id`.

9. Модифицируйте код программы `lab_04_02.py`. Измените название атрибута `id` на `count`. Модифицируйте программу так, чтобы, определив атрибут `id`, он являлся уникальным для каждого объекта, изменяясь на единицу с увеличением количества объектов класса.

10. Создайте файл `lab_04_03.py`:

```
class Geometric:
    def calculateArea(self):
        print("Calculating area")

class Square(Geometric):
    def __init__(self,a):
        self.side = a
    def _perimeter(self):
        print("Perimeter of Square {}: {}\n".format(self.side,
self.side*4))
    def calculateArea(self):
        print("Area of Square {}: {}\n".format(self.side,
pow(self.side,2)))

geom = Geometric()
geom.calculateArea()
sq = Square(5)
sq.calculateArea()
sq._perimeter()

print("Check subclass: ", issubclass(Square,Geometric))
print("Check instance sq->Square: ", isinstance(sq,Square))
print("Check instance sq->Geometric: ",
isinstance(sq,Geometric))
print("Check instance sq->dict: ", isinstance(sq,dict))

print("Geometric.__bases__: ", Geometric.__bases__)
print("Square.__bases__: ", Square.__bases__)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Calculating area
Area of Square 5: 25

Perimeter of Square 5: 20

Check subclass:  True
Check instance sq->Square:  True
```

```
Check instance sq->Geometric:  True
Check instance sq->dict:  False
Geometric.__bases__:  (<class 'object'>,)
Square.__bases__:  (<class 'Geometric'>,)
```

Наследование

Наследование между классами в языке программирования Python реализуется следующим образом:

```
class Ancestor:
    def func(self):
        pass

class Descendant(Ancestor):
    def func(self):
        pass
```

Имя родительского класса (`Ancestor`) указывается в скобках при объявлении дочернего (`Descendant(Ancestor)`).

Для показания приватности метода в начале его названия добавляется символ подчеркивания (_). Однако, следует отметить, что в Python не предусмотрено полностью приватных атрибутов и методов: доступ к ним в любом случае может быть получен из основного кода программы. Для того, чтобы метод/атрибут казался более закрытым, допускается добавление двух символов подчеркивания перед названием метода (__), но в этом случае доступ к методу/атрибуту может быть получен с использованием имени класса (`_Class__method`).

Переопределение методов аналогично многим объектно-ориентированным языкам программирования осуществляется путем указания функции родительского класса с тем же именем и аргументами внутри дочернего класса.

Для наследования атрибутов родительского класса, указанных в конструкторе родительского класса, необходимо внутри конструктора дочернего класса первоначально вызвать конструктор родительского класса, например:

```
class Ancestor:
    def __init__(self,param):
        self.param = param
    def func(self):
        pass

class Descendant(Ancestor):
    def __init__(self, param, number):
        Ancestor.__init__(self, param)
        self.number = number
    def func(self):
        pass
```

Здесь:
`Ancestor.__init__(self, param)` – строка вызова конструктора родительского класса.
Также может быть использована следующая запись:
`super(Descendant, self).__init__(param)`

Для проверки наследования класса (`Descendant`) от известного родительского класса (`Ancestor`) используется функция `issubclass(Descendant, Ancestor)`. Для проверки принадлежности объекта (`obj`) какому-либо классу (`Class`) используется функция `isinstance(obj,Class)`. Также для проверки наследования классов может быть использован атрибут `__bases__`.

11. Дополните код программы `lab_04_03.py`. Создайте класс `Circle`, унаследованный от класса `Geometric`, для которого определите атрибут `radius`. Сделайте атрибут `radius` приватным (с использованием двойного подчеркивания). Переопределите унаследованный метод `calculateArea()`, рассчитав площадь окружности (значение числа Пи доступно в библиотеке `math`).

12. Создайте файл `lab_04_04.py`. Создайте класс `Encoder`, определив для него методы `encode()` и `decode()`, аргументом которых является строка, а выходными данными – закодированная и декодированная строка соответственно. Создайте классы `HuffmanEncoder` и `LZEncoder`, унаследованные от класса `Encoder`, определив для них атрибут `compressionCoef` в конструкторах классов. Для созданных классов `HuffmanEncoder` и `LZEncoder` переопределите методы `encode()` и `decode()`, реализующие кодирование и декодирование поданных в качестве аргументов строк, используя методы Хаффмана и Лемпеля-Зива соответственно. Определите приватный метод `setCompressionCoef()` (с использованием двойного подчеркивания), осуществляющий расчет коэффициентов сжатия для методов Хаффмана и Лемпеля-Зива в соответствующих классах. Метод `setCompressionCoef()` должен вызываться при работе внутри класса в методе `encode()`. Определите общедоступный метод `getCompressionCoef()`, позволяющий получить значение коэффициента сжатия. Осуществите проверку методов классов на следующей строке:

```
Python is a widely used high-level programming language for general-purpose
programming, created by Guido van Rossum and first released in 1991.
```

13. Создайте файл `lab_04_05.py`. Создайте класс `Person`, определив для него в конструкторе атрибуты `firstname`, `lastname` и `age`, а также метод `display()`, выводящий информацию об объекте класса на экран. Создайте класс `Student`, унаследованный от `Person`, определив для него дополнительные атрибуты `studentID`, являющийся уникальным номером для каждого объекта класса, и `recordBook`, содержащий информацию о количестве пятерок, четверок, троек и двоек студента в виде списка. В классе `Student` дополните метод `display()` выводом значений атрибутов `studentID` и `recordBook` на экран. Создайте три объекта класса `Student` для проверки работы методов и выведите информацию о них на экран.

14. Дополните код программы `lab_04_05.py`. Создайте класс `Professor`, унаследованный от `Person`, определив для него дополнительные атрибуты `professorID`, являющийся уникальным номером для каждого объекта класса, и

degree, содержащий информацию о научной степени в виде строки. В классе Professor дополните метод display() выводом значений атрибутов professorID и degree на экран. Создайте три объекта класса Professor для проверки работы методов и выведите информацию о них на экран.

15. Создайте файл lab_04_06.py. Создайте класс HammingEncoder, конструктор которого принимает на вход количество информационных разрядов dataBits, на основе которых рассчитывает значение количества контрольных разрядов – controlBits. dataBits и controlBits являются атрибутами класса. Реализуйте метод encode(str), служащий для кодирования строки двоичных символов str с использованием кода Хэмминга с установленными параметрами dataBits и controlBits. Реализуйте метод decode(str), служащий для определения кода ошибки закодированной строки str.

16. Создайте файл lab_04_07.py. Создайте следующие классы и реализуйте в них указанные методы:

- Класс Row
  Атрибуты:
  - id – идентификатор строки.
  - collection – список значений переменных для текущего значения функции.
  - value – значение функции.

  Методы:
  - __init__(collection, value) – конструктор класса, принимающий на вход значения collection и value.
- Класс Table
  Атрибуты:
  - rows – список объектов класса Row.
  - rowsNum – количество строк таблицы.

  Методы:
  - __init__(rowsNum)– конструктор класса, принимающий на вход значение rowsNum.
  - addRow(row) – добавление строки (объект row класса Row) в список. Если в списке уже находится строка с таким же идентификатором, должна выдаваться ошибка.
  - setRow(row) – изменение строки (объект row класса Row). Если в списке нет строки с таким же идентификатором, должна выдаваться ошибка.
  - getRow(rowId) – получение строки с идентификатором rowId. Возвращается объект класса Row.
  - display() – вывод таблицы на экран в следующем формате:

```
id  x1  x2   f(x1,x2)
1   0   0  |    1
2   0   1  |    0
3   1   0  |    0
4   1   1  |    1
```

- Класс LogicFunction
  Атрибуты:
  - variablesNum – количество переменных функции
  - table – объект класса Table. Таблица истинности логической функции.

  Методы:

- `__init__(variablesNum,table)` – конструктор класса, принимающий на вход значения `variablesNum` и `table`.
- `getExpression()` – вычисляет и возвращает минимальную формулу логической функции.
- `getTable()` – получение значения `table`.
- `printTable()` – вывод значения `table` на экран.