Семинар 14

Введение в программирование на Python

Папулин С.Ю.
papulin_hse@mail.ru

2015

Семинар 13

Структуры данных

- Дерево
- Граф (продолжение)
  - Путь по матрицам смежности
  - Алгоритм Dijkstra
- Пояснения к семинарским задачам

План семинара 14

- Объектно-ориентированное программирования (ООП) в Python

Объектно-ориентированное программирования в Python

Объявление класса в Python

class *Имя_класса*(*[Наследуемый_класс]*):
    атрибуты и методы

Класс и экземпляр класса в Python

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        self.__firstName = fName
        self.__secondName = sName
        self.__phone = tel
        self.__email = e_mail

    def getFullInfo(self):
        print("First Name: ", self.__firstName)
        print("Second Name: ", self.__secondName)
        print("Phone: ", self.__phone)
        print("Email: ", self.__email)

p1 = Person("George", "Ivanov", "8800232111", "isivanov@mail.com")
p1.getFullInfo()
```

Переменные экземпляра класса

Метод экземпляра класса

Экземпляр класса

Вызов метода экземпляра класса

```
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
```

Класс и экземпляр класса в Python

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        …
    def getFullPersonInfo(self):
        …

def main():
    p1 = Person("George", "Ivanov", "8800232111", "givanov@mail.com")
    p2 = Person("Petr", "Obama", "8800232112", "pobama@mail.com")

    print("-----First person------")
    p1.getFullPersonInfo()
    print("----Second person------")
    p2.getFullPersonInfo()

if __name__ ==  "__main__":
    main()
```

```
-----First person------
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  givanov@mail.com
----Second person------
First Name:  Petr
Second Name:  Obama
Phone:  8800232112
Email:  pobama@mail.com
```

Обращение к методам экземпляра класса

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        …
    def getFullInfo(self):
        …

p1 = Person("George", "Ivanov", "8800232111", "isivanov@mail.com")
```

1
```
p1.getFullInfo()
```

2
```
Person.getFullInfo(p1)
```

→

```
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
```

Основанная программа/подключаемый модуль

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        …
    def getFullInfo(self):
        …

def main():
    p1 = Person("George", "Ivanov", "8800232111", "isivanov@mail.com")
    p1.getFullPersonInfo()

    Person.getFullPersonInfo(p1)

if __name__ == "__main__":
    main()
```

Инкапсуляция в Python – public/private

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        self.firstName = fName
        self.secondName = sName
        self.phone = tel
        self.email = e_mail

    def getFullPersonInfo(self):
        …

def main():
    p1 = Person("George", "Ivanov", "8800232111",
"isivanov@mail.com")

    print("First Name: ", p1.firstName)
    print("Second Name: ", p1.secondName)
    print("Phone: ", p1.phone)
    print("Email: ", p1.email)

if __name__ ==  "__main__":
    main()
```

```
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
```

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        self.__firstName = fName
        self.__secondName = sName
        self.__phone = tel
        self.__email = e_mail

    def getFullPersonInfo(self):
        …

def main():
    p1 = Person("George", "Ivanov", "8800232111",
"isivanov@mail.com")

    print("First Name: ", p1.__firstName)
    print("Second Name: ", p1.__secondName)
    print("Phone: ", p1.__phone)
    print("Email: ", p1.__email)

if __name__ ==  "__main__":
    main()
```

```
Traceback (most recent call last):
AttributeError: 'Person' object has no attribute '__firstName'
```

Инкапсуляция в Python – public/private

```
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        …
    def __getFullPersonInfo(self):
        …


def main():
    p1 = Person("George", "Ivanov", "8800232111", "isivanov@mail.com")
    p1.__getFullPersonInfo()

if __name__ == "__main__":
    main()
```

```
AttributeError: 'Person' object has no attribute '__getFullPersonInfo'
```

Наследование

```python
from Class_person import Person

class Employee(Person):
    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        Person.__init__(self, fName, sName, tel, e_mail)
        self.__position = pos
        self.__salary = sal

    def getFullEmployeeInfo(self):
        self.getFullPersonInfo()
        print("Position: ", self.__position)
        print("Salary: ", self.__salary)

e1 = Employee("George", "Ivanov", "8800232111",
              "isivanov@mail.com", "doctor", "100")
print("---------------------------------")
print("Full information about a person: ")
print("---------------------------------")
e1.getFullPersonInfo()
print("------------------------------------")
print("Full information about an employee: ")
print("------------------------------------")
e1.getFullEmployeeInfo()
```

```
---------------------------------
Full information about a person:
---------------------------------
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
------------------------------------
Full information about an employee:
------------------------------------
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
Position:  doctor
Salary:  100
```

Переменные класса

Переменная класса

```
from Class_person import Person

class Employee(Person):

    budget = 1000

    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        Person.__init__(self, fName, sName, tel, e_mail)
        self.__position = pos
        self.__salary = sal
        Employee.budget -= self.__salary

    def getFullEmployeeInfo(self):
        …

    def getCurrentBudget(self):
        print("Current budget is ", self.budget)

e1 = Employee("George", "Ivanov", "8800232111",
              "isivanov@mail.com", "doctor", 100)

e1.getCurrentBudget()
```

Current budget is 900

```
e2 = Employee("Petr", "Obama", "8800232112",
                "pobama@mail.com", "householder", 1000)

e2.getCurrentBudget()
```

Current budget is -100

Переменные класса

```
from Class_person import Person

class Employee(Person):

    budget = 1000

    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        …
    def getFullEmployeeInfo(self):
        …

    def getCurrentBudget(self):
        print("Current budget is ", self.budget)

e1 = Employee("George", "Ivanov", "8800232111",
            "isivanov@mail.com", "doctor", 100)

e1.getCurrentBudget()

e2 = Employee("Petr", "Obama", "8800232112",
              "pobama@mail.com", "householder", 1000)

e2.getCurrentBudget()

e2.budget = 2000

e2.getCurrentBudget() ?

e1.getCurrentBudget() ?
print("Current budget is ", Employee.budget) ?
```

```
Current budget is 900
Current budget is -100
Current budget is 2000
Current budget is -100
Current budget is  -100
```

Метод класса

ЗАМЕНИМ

```
def getCurrentBudget(self):
        print("Current budget is ",
self.budget)
```

НА

```
@classmethod
def getCurrentBudget(cls):
    print("Current budget is ", cls.budget)
```

Метод класса

ПОЛУЧИМ

```
Current budget is  900
Current budget is  -100
Current budget is  -100
Current budget is  -100
Current budget is  -100
```

1 `Employee.getCurrentBudget()`

2 `e1.getCurrentBudget()`

3 `e2.getCurrentBudget()`

Статический метод

```
from Class_person import Person

class Employee(Person):

    budget = 1000
    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        Person.__init__(self, fName, sName, tel, e_mail)
        self.__position = pos
        self.__salary = sal
        Employee.budget -= Employee.taxes(self.__salary)

    def getFullEmployeeInfo(self):
          …
    @classmethod
    def getCurrentBudget(cls):
          …
    @staticmethod
    def taxes(money):
        taxRate = 0.5
        return money * (1 + taxRate)

e1 = Employee("George", "Ivanov", "8800232111",
            "isivanov@mail.com", "doctor", 100)

Employee.getCurrentBudget()
```

static-метод

Current budget is 850.0

```
e2 = Employee("Petr", "Obama", "8800232112",
              "pobama@mail.com", "householder", 100)

Employee.getCurrentBudget()
```

Current budget is 700.0

Свойства

```python
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        self.__firstName = fName
        self.__secondName = sName
        self.__phone = tel
        self.__email = e_mail

    #Методы экземпляра класса
    def getFistName(self):
        return self.__firstName
    def setFirstName(self, fname):
        self.__firstName = fname

    #Свойства. Способ 1
    @property
    def secondName(self):
        return self.__secondName
    @secondName.setter
    def secondName(self, sname):
        self.__secondName = sname

    #Свойства. Способ 2
    def __getPhone(self):
        return self.__phone
    def __setPhone(self, phNumber):
        self.__phone = phNumber
    phoneNumber = property(__getPhone, __setPhone)
```

```python
p1 = Person("George", "Ivanov",
"8800232111", "givanov@mail.com")

p1.setFirstName("Mike")
print(p1.getFistName())
```

```
Mike
```

```python
p1.secondName = "Petrov"
print(p1.secondName)
```

```
Petrov
```

```python
p1.phoneNumber = "324"
print(p1.phoneNumber)
```

```
324
```

Переопределение (override) методов

```python
class Person:
    def __init__(self, fName, sName, tel, e_mail):
        self.__firstName = fName
        self.__secondName = sName
        self.__phone = tel
        self.__email = e_mail

    def getFirstName(self):
        return self.__firstName

    def getSecondName(self):
        return self.__secondName

    def getFullPersonInfo(self):
        print("First Name: ", self.__firstName)
        print("Second Name: ", self.__secondName)
        print("Phone: ", self.__phone)
        print("Email: ", self.__email)
```

```python
from Class_person import Person

class Employee(Person):
    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        Person.__init__(self, fName, sName, tel, e_mail)
        self.__position = pos
        self.__salary = sal

    #override - перезагрузка метода
    def getFullPersonInfo(self):
        print("First Name: ", self.getFirstName())
        print("Second Name: ", self.getSecondName())
        print("Position: ", self.__position)
        print("Salary: ", self.__salary)

    #Вызов метода экземпляра класса Person. Способ 1
    def getFullPersonInfo_Old_v1(self):
        return Person.getFullPersonInfo(self)

    #Вызов метода экземпляра класса Person. Способ 2
    def getFullPersonInfo_Old_v2(self):
        return super().getFullPersonInfo()
```

Переопределение (override) методов

```python
from Class_person import Person

class Employee(Person):
    def __init__(self, fName, sName, tel, e_mail, pos, sal):
        Person.__init__(self, fName, sName, tel, e_mail)
        self.__position = pos
        self.__salary = sal

    #override - переопределение метода
    def getFullPersonInfo(self):
        print("First Name: ", self.getFirstName())
        print("Second Name: ", self.getSecondName())
        print("Position: ", self.__position)
        print("Salary: ", self.__salary)

    #Вызов метода экземпляра класса Person. Способ 1
    def getFullPersonInfo_Old_v1(self):
        return Person.getFullPersonInfo(self)

    #Вызов метода экземпляра класса Person. Способ 2
    def getFullPersonInfo_Old_v2(self):
        return super().getFullPersonInfo()
```

```python
e1 = Employee("George", "Ivanov", "8800232111",
              "isivanov@mail.com", "doctor", 100)

print("-----Overridden method-----")
e1.getFullPersonInfo()
print("-----Old method 1-----")
e1.getFullPersonInfo_Old_v1()
print("-----Old method 2-----")
e1.getFullPersonInfo_Old_v2()
```

```
-----Overridden method-----
First Name:  George
Second Name:  Ivanov
Position:  doctor
Salary:  100
-----Old method 1-----
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
-----Old method 2-----
First Name:  George
Second Name:  Ivanov
Phone:  8800232111
Email:  isivanov@mail.com
```

Перегрузка (overload) операторов

```
class Vector:
    def __init__(self, values):
        self.__values = values

    @property
    def values(self):
        return self.__values

    def __add__(self, other):
        newValues = [self.__values[i] + other.values[i] for i in range(len(self.__values))]
        return Vector(newValues)

    def __sub__(self, other):
        newValues = [self.__values[i] - other.values[i] for i in range(len(self.__values))]
        return Vector(newValues)

    def __str__(self):
        return str(self.__values)

v1 = Vector([1, 2, 3])
v2 = Vector([4, 5, 6])
v3 = Vector([7, 8, 9])

v4 = v1 + v2 - v3

print(v4)
```

Свойство для получения значений вектора

Перегрузка оператора сложения

Перегрузка оператора вычитания

Перегрузка строкового представления экземпляра класса

```
[-2, -1, 0]
```

Перегрузка (overload) функций/методов

В Python нет стандартных средств перегрузки функций/методов

Одно имя функции и разное количество аргументов

```
def getAccess(login):
    pass #do something

def getAccess(login, password):
    pass #do something

def getAccess(login, password, secret_numbers):
    pass #do something

#for guest
getAccess("guest")

#for usual user
getAccess("user", "user_password")

#for protected user
getAccess("user", "user_password", 12642845)
```

TypeError: getAccess() missing 2 required positional arguments: 'password' and 'secret_numbers'

В качестве альтернативы можно воспользоваться

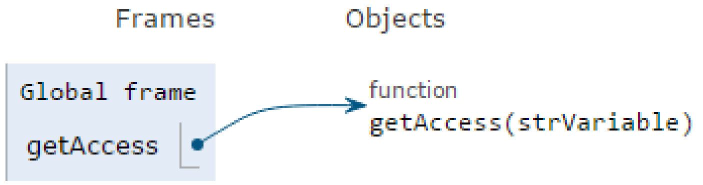
```
def getAccess(login, password = None, secret_word = None):
    pass #do something

#for guest
getAccess("guest")

#for usual user
getAccess("user", "user_password")

#for protected user
getAccess("user", "user_password", 12642845)
```

Перегрузка (overload) функций/методов

Одно имя функции и разные типы аргументов

```
def getAccess(intVariable):
    pass #do something

def getAccess(floatVariable):
    pass #do something

def getAccess(strVariable):
    pass #do something

#for int
getAccess(5437)

#for float
getAccess(3434.45)

#for string
getAccess("secret_word")
```

Только последняя функция getAccess будет использовать



Дополнительные модули для реализации перегрузки функций/методов

from ***overloading*** import ***overload***

from ***multipledispatch*** import ***dispatch***

from ***functools*** import ***singledispatch***

Источники

9. Classes

Классы

Программирование на Python: Часть 6. Классы

The definitive guide on how to use static, class or abstract methods in Python

Property

Перегрузка операторов

Magic Methods and Operator Overloading

- *overloading*
- *multipledispatch*
- *singledispatch*