Лямбда-выражения 

Теория

Лямбда-выражения в языке Python представляют небольшие анонимные функции, которые определяются с помощью оператора lambda. Формальное определение лямбда-выражения:

Пример 1

```
lambda [параметры] : инструкция

test = lambda: print("MPT")

test()   # MPT

# Мы можем вызвать это лямбда-выражение как обычную функцию. Фактически оно
аналогично следующей функции:
```

Пример 2 Параметры в lambda

```
# Передаем один параметр
square = lambda n: n * n

print(square(24))    # 576
print(square(14))    # 196

# Передаем два параметра
min = lambda q,w: q - w

print(min(24,10))    # 14
print(min(14,-2))    # 16

# В данном случае лямбда-выражение принимает один параметр - n. Справа от
двоеточия идет возвращаемое значение - n* n. Это лямбда-выражение аналогично
следующей функции:
```

Пример 3 возвращение лямбда-выражений из функций

```
# Если функция имеет несколько параметров, то необязательные параметры должны идти
после обязательных. Например:

def Op(choice):
    if choice == 1:
```

```
        return lambda a, b: a + b

number = Op(1)
print(number(10,2))

# Результат вывода  "12"
```

Преобразование типов

Основные типы

```
# Для преобразования типов Python предоставляет ряд встроенных функций:

int(): преобразует значение в целое число

float(): преобразует значение в число с плавающей точкой

str(): преобразует значение в строку
```

Примеры

```
# int()
a = int(15)     # a = 15
b = int(3.7)    # b = 3

# float()
a = float(15)       # a = 15.0
b = float(3.7)      # b = 3.7

# str()
a = str(False)      # a = "False"
b = str(True)       # b = "True"
```

Классы и обьекты 🏛

```
Создание класса

Создадим в нашем проекте папку и назовем её (fullClass.py) где будут создаваться
классы
```

Теория

При определении методов любого класса следует учитывать, что все они должны принимать в качестве первого параметра ссылку на текущий объект, который согласно условностям называется self. Через эту ссылку внутри класса мы можем обратиться к функциональности текущего объекта. Но при самом вызове метода этот параметр не учитывается.

Если метод должен принимать другие параметры, то они определяются после параметра self, и при вызове подобного метода для них необходимо передать значения:

Методы классов

```
# Методы класса фактически представляют функции, которые определенны внутри класса
и которые определяют его поведение. Например, определим класс Person с одним
методом:

class Employee:       # определение класса Employee
     def hello_emp(self):
        print("Hello")


Max = Employee()
Max.hello_emp()    # Hello

# Параметры методов

class Employee:       # определение класса Employee
     def hello_emp(self, name, group):
        print(f"Привет {name} из группы {group}")


Max = Employee()
Max.hello_emp("Max", "P50-3-252") # Привет Max из группы P50-3-252

# Использование self

class Employee:       # определение класса Employee
    def hello_emp(self, name, group):
        print(f"Привет {name} из группы {group}")

    def say_hello(self):
        self.hello_emp("Max","P50-3-252")

Max = Employee()
Max.say_hello() # Привет Max из группы P50-3-252
```

Конструкторы

Теория

Для создания объекта класса используется конструктор. Так, выше когда мы создавали объекты класса Employee, мы использовали конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы:

Однако мы можем явным образом определить в классах конструктор с помощью специального метода, который называется __init__() (по два прочерка с каждой стороны). К примеру, изменим класс Employee, добавив в него конструктор:

```
class Employee:       # определение класса Employee

    # Конструктор
    def __init__(self):
        print("Чао")

    def hello_emp(self, name, group):
        print(f"Привет {name} из группы {group}")

    def say_hello(self):
        self.hello_emp("Max","P50-3-252")

Max = Employee()    # Чао
Max.say_hello() # Привет Max из группы P50-3-252

# Теперь при создании объекта: Max = Employee() будет производится вызов
конструктора __init__() из класса Person, который выведет на консоль строку
"Создание объекта Person".
```

Атрибуты объекта

Атрибуты хранят состояние объекта. Для определения и установки атрибутов внутри класса можно применять слово self. Например, определим следующий класс Employee:

```
class Employee:       # определение класса Employee

    """ Конструктор """
    def __init__(self, name, group):
        self.name = name
        self.group = group
        self.age = 15

        print(f"Привет {name} из группы {group} возраст {self.age}")
```

```
    # def hello_emp(self, name, group):
    #     print(f"Привет {name} из группы {group}")

    # def say_hello(self):
    #     self.hello_emp("Max","P50-3-252")

Max = Employee("Max","P50-3-252")    # Чао
# Получение значений по атрибутам
print(Max.name) # Max
# Изменение значение по имени
Max.age = 22
print(Max.age) # 22
print(Max.group) # P50-3-252
```

Для обращения к атрибутам объекта внутри класса в его методах также применяется слово self:

Пример 1

```
class Employee:       # определение класса Employee

    # Конструктор
    def __init__(self, name, group):
        self.n = name
        self.g = group
        self.a = 15
    # def hello_emp(self, name, group):
    #     self.name = name
    #     print(f"Привет {name} из группы {group}")

    def say_hello(self):
        self.a = 22
        print(f"Привет {self.n} из группы {self.g} возраст {self.a}")

Max = Employee("Max","P50-3-252")
print(Max.say_hello())              # Привет Max из группы P50-3-252 возраст 22
```

Пример 2 Использование Class в другом файле .py

```
# Импорт файла "classTest" из папки "Class"
from Class import method
from Class import classTest

Max = classTest.Employee("Max","P50-3-252")
print(Max.say_hello())              # Привет Max из группы P50-3-252 возраст 22
```

Пример 3 Строковое представление объекта

```
class Employee:       # определение класса Employee

    # Конструктор
    def __init__(self, name, group):
        self.n = name
        self.g = group
        self.a = 15

    def __str__(self) -> str:
        return f"Привет {self.n} из группы {self.g} возраст {self.a}"

Max = Employee("Max","P50-3-252")
print(Max)   #Привет Max из группы P50-3-252 возраст 15
```

Наследование

```
# Наследование
class Employee:       # определение класса Employee

    # Конструктор
    def __init__(self, name, group):
        self.n = name
        self.g = group
        self.a = 15

    # def __str__(self) -> str:
    #     return "Привет"

class Work(Employee):
    def __init__(self, name, group):
        # связывает потомка с родителем
        super().__init__(name, group) # обращение к методу __init__ в классе
Employee
        print(f"Привет {self.n} из группы {self.g} возраст {self.a}")

Max = Work("Max","MPT")    # Привет Max из группы MPT возраст 15
```

Аннотации свойств

```
Для создания свойства-геттера (get) над свойством ставится аннотация @property.

Для создания свойства-сеттера (set) над свойством устанавливается аннотация
имя_свойства_геттера.setter.
```

Перепишем класс Person с использованием аннотаций:

Проверка типа объекта

При работе с объектами бывает необходимо в зависимости от их типа выполнить те или иные операции. И с помощью встроенной функции isinstance() мы можем проверить тип объекта. Эта функция принимает два параметра:

Первый параметр представляет объект, а второй - тип, на принадлежность к которому выполняется проверка.

```
isinstance(object, type)
```

```
class Person:

    def __init__(self, name):
        self.__name = name   # имя человека

    # get получаем имя человека
    @property
    def Name(self):
        return self.__name

    def do_nothing(self):
        print(f"{self.Name} ничего не делает")


#  класс работника
class Employee(Person):

    def work(self):
        print(f"{self.Name} работает")


#  класс студента
class Student(Person):

    def study(self):
        print(f"{self.Name} студент")

#  проверяем к какому классу принадлежит тот или иной человек
def proverka(person):
    if isinstance(person, Student):
        person.study()
```

```
    elif isinstance(person, Employee):
        person.work()
    elif isinstance(person, Person):
        person.do_nothing()


tom = Employee("Tom")
bob = Student("Bob")
sam = Person("Sam")

proverka(tom)    # Tom работает
proverka(bob)    # Bob студент
proverka(sam)    # Sam ничего не делает
```

Задание 🏠

5

```
# Разделить операции +,-,/,* на классы в которых будут функции из прошлых лекций, посчитать в них сумму и вернуть ответ в другом файле .py (доработка первой практической с учетом ввода колличесва чисел)
```

4

```
# Разделить операции +,-,/,* на классы в которых будут функции из прошлых лекций, посчитать в них сумму и вернуть ответ в другом файле .py (ввод двух чисел, использовать lambda)
```

3

```
# Повторить все как в файле"
```