**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**ИНСТИТУТ ЦИФРЫ**

**ОТЧЁТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«ООП»

Студентки 2 курса, ФИТ-211 группы

**Колесник Полины Олеговны**

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Руководитель:

Доцент Зимин А. И.

Работа защищена

« »

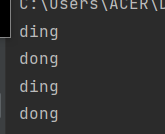
“ ” 2023 г.

Кемерово 2023 г.

**ОТЧЁТ О ПРОДЕЛАННОЙ РАБОТЕ**

**1 задание**

class BigBell:  
 zvuc = 0  
  
 def sound(self):  
 if not self.zvuc:  
 print('ding')  
 self.zvuc = 1  
 else:  
 print('dong')  
 self.zvuc = 0  
  
  
bell = BigBell()  
bell.sound()  
bell.sound()  
bell.sound()  
bell.sound()



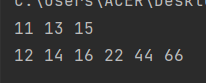
**2 задание**

class Balance:  
 def \_\_init\_\_(self):  
 self.number\_left = 0  
 self.number\_right = 0  
  
 def add\_right(self, number):  
 self.number\_right += number  
  
 def add\_left(self, number):  
 self.number\_left += number  
  
 def result(self):  
 if self.number\_right == self.number\_left:  
 return "="  
  
 elif self.number\_right > self.number\_left:  
 return "R"  
  
 return "L"  
  
  
balance = Balance()  
balance.add\_right(10)  
balance.add\_left(9)  
balance.add\_left(2)  
print(balance.result())



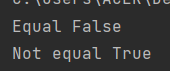
**3 задание**

class Selector:  
 def \_\_init\_\_(self, value):  
 self.value = value  
  
 def get\_odds(self):  
 odds = []  
 for i in self.value:  
 if i % 2 != 0:  
 odds.append(i)  
 return odds  
  
 def get\_evens(self):  
 evens = []  
 for i in self.value:  
 if i % 2 == 0:  
 evens.append(i)  
 return evens  
  
  
values = [11, 12, 13, 14, 15, 16, 22, 44, 66]  
selector = Selector(values)  
odds = selector.get\_odds()  
evens = selector.get\_evens()  
print(' '.join(map(str, odds)))  
print(' '.join(map(str, evens)))



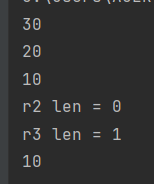
**4 задание**

class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
  
 def \_\_eq\_\_(self, other):  
 return other.x == self.x and other.y == self.y  
  
  
p1 = Point(1, 2)  
p2 = Point(5, 6)  
  
if p1 == p2:  
 print('Equal True')  
else:  
 print('Equal False')  
  
if p1 != p2:  
 print('Not equal True')  
else:  
 print('Not equal False')



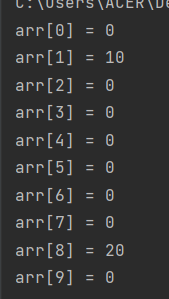
**5 задание**

class ReversedList:  
 def \_\_init\_\_(self, lst):  
 self.lst = lst[::-1]  
  
 def \_\_getitem\_\_(self, item):  
 return self.lst[item]  
  
 def \_\_len\_\_(self):  
 return len(self.lst)  
  
  
r1 = ReversedList([10, 20, 30])  
  
for i in range(len(r1)):  
 print(r1[i])  
  
r2 = ReversedList([])  
print(f'r2 len = {len(r2)}')  
  
r3 = ReversedList([10])  
print(f'r3 len = {len(r3)}')  
print(r3[0])



**6 задание**

class SparseArray:  
 def \_\_init\_\_(self):  
 self.arr = {}  
  
 def \_\_getitem\_\_(self, item):  
 return self.arr.get(item, 0)  
  
 def \_\_setitem\_\_(self, item, value):  
 self.arr[item] = value  
  
  
arr = SparseArray()  
arr[1] = 10  
arr[8] = 20  
for i in range(10):  
 print('arr[{}] = {}'.format(i, arr[i]))



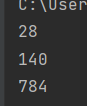
**9 задание**

class Triangle:  
 def \_\_init\_\_(self, ab, bc, ca):  
 self.ab = ab  
 self.bc = bc  
 self.ca = ca  
  
 def perimeter(self):  
 return self.ca + self.ab + self.bc  
  
  
class EquilateralTriangle(Triangle):  
 def \_\_init\_\_(self, line):  
 super().\_\_init\_\_(line, line, line)  
  
  
tr = EquilateralTriangle(6)  
print(tr.perimeter())  
  
tr2 = Triangle(1, 2, 3)  
print(tr2.perimeter())



**10 задание**

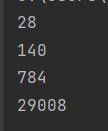
class Summator:  
 def transform(self, n):  
 return n  
  
 def sum(self, N):  
 summ = 0  
 for i in range(N + 1):  
 summ += self.transform(i)  
 return summ  
  
  
class SquareSummator(Summator):  
 def \_\_init\_\_(self):  
 pass  
  
 def transform(self, n):  
 return n \*\* 2  
  
  
class CubeSummator(Summator):  
 def \_\_init\_\_(self):  
 pass  
  
 def transform(self, n):  
 return n \*\* 3  
  
  
s1 = Summator()  
print(s1.sum(7))  
s2 = SquareSummator()  
print(s2.sum(7))  
s3 = CubeSummator()  
print(s3.sum(7))



**11 задание**

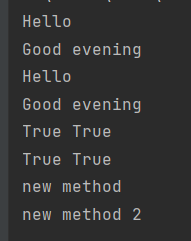
class Summator:  
 def transform(self, n):  
 return n  
  
 def sum(self, N):  
 summ = 0  
 for i in range(N + 1):  
 summ += self.transform(i)  
 return summ  
  
  
class PowerSummator(Summator):  
 def \_\_init\_\_(self, b):  
 self.b = b

def transform(self, n):  
 return n \*\* self.b  
  
  
class SquareSummator(PowerSummator):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_(2)  
  
  
class CubeSummator(PowerSummator):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_(3)  
  
  
s1 = Summator()  
print(s1.sum(7))  
s2 = SquareSummator()  
print(s2.sum(7))  
s3 = CubeSummator()  
print(s3.sum(7))  
s4 = PowerSummator(5)  
print(s4.sum(7))



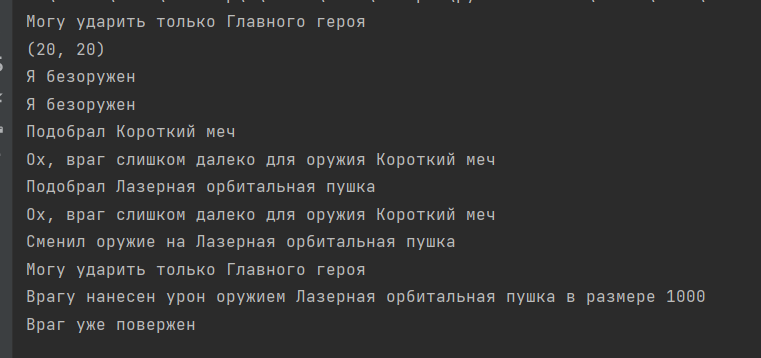
**12 задание**

class A:  
 def \_\_str\_\_(self):  
 return 'A.\_\_str\_\_method'  
  
 def hello(self):  
 print('Hello')  
  
  
class B:  
 def \_\_str\_\_(self):  
 return 'B.\_\_str\_\_method'  
  
 def good\_evening(self):  
 print('Good evening')  
  
  
class C(A, B):  
 pass  
  
  
class D(B, A):  
 pass  
  
  
# C = C()  
# C.hello()  
# C.good\_evening()  
#  
# d = D()  
# d.hello()  
# d.good\_evening()  
#  
# print(C)  
# print(d)  
  
  
def new\_method(arg):  
 return "new method"  
  
  
def new\_method2(arg):  
 return 'new method 2'  
  
  
A.\_\_str\_\_ = new\_method  
B.\_\_str\_\_ = new\_method2  
c = C()  
c.hello()  
c.good\_evening()  
  
d = D()  
d.hello()  
d.good\_evening()  
print(isinstance(c, A), isinstance(c, B))  
print(isinstance(d, A), isinstance(d, B))  
print(c)  
print(d)



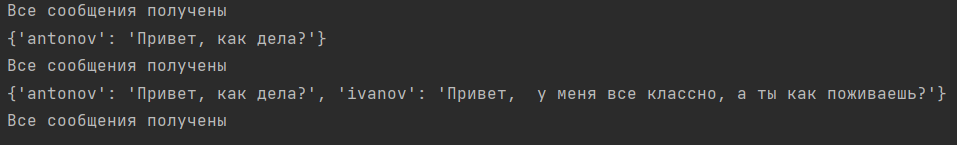
**13 задание**

import math  
  
  
class Weapon: # оружие  
 def \_\_init\_\_(self, name, damage, range):  
 self.name = name  
 self.damage = damage  
 self.range = range  
  
 def hit(self, actor, target): # удар, target - препятствие, ну или злодей  
 if target.is\_alive():  
 if math.sqrt((target.pos\_x - actor.pos\_x) \*\* 2 + (target.pos\_y - actor.pos\_y) \*\* 2) > self.range:  
 print(f'Ох, враг слишком далеко для оружия {self.name}')  
 else:  
 print(f'Врагу нанесен урон оружием {self.name} в размере {self.damage}')  
 target.hp -= self.damage  
 # между врагом и человеком есть расстояние....  
 # рассчитать можно через теорему Пифагора их расстояние друг от друга...  
 # наверное...  
 else:  
 print('Враг уже повержен')  
  
 def \_\_str\_\_(self):  
 return self.name  
  
  
class BaseCharacter: # базовые персонажи  
 def \_\_init\_\_(self, pos\_x, pos\_y, hp):  
 self.pos\_x = pos\_x  
 self.pos\_y = pos\_y  
 self.hp = hp  
  
 def move(self, delta\_x, delta\_y): # Передвижение  
 self.pos\_x += delta\_x  
 self.pos\_y += delta\_y  
  
 def is\_alive(self):  
 if self.hp > 0:  
 return True  
 else:  
 return False  
  
 def get\_damage(self, amount): # в каком размере  
 if self.is\_alive():  
 self.hp -= amount  
  
 def get\_coords(self):  
 return self.pos\_x, self.pos\_y  
  
  
class BaseEnemy(BaseCharacter):  
 def \_\_init\_\_(self, pos\_x, pos\_y, weapon, hp):  
 super().\_\_init\_\_(pos\_x, pos\_y, hp)  
 self.weapon = weapon  
  
 def hit(self, target):  
 if isinstance(target, MainHero):  
 self.weapon.hit(self, target)  
 else:  
 print('Могу ударить только Главного героя')  
  
 def \_\_str\_\_(self):  
 return f'Враг на позиции ({self.pos\_x}, {self.pos\_y}) с оружием {self.weapon}'  
  
  
class MainHero(BaseCharacter):  
 def \_\_init\_\_(self, pos\_x, pos\_y, name, hp):  
 super().\_\_init\_\_(pos\_x, pos\_y, hp)  
 self.name = name  
 self.weapons = []  
 self.current\_weapon = 0  
  
 def hit(self, target):  
 if self.weapons:  
 if isinstance(target, BaseEnemy):  
 self.weapons[self.current\_weapon].hit(self, target)  
 else:  
 print('Могу ударить только Главного героя')  
 else:  
 print('Я безоружен')  
  
 def add\_weapon(self, weapon):  
 if isinstance(weapon, Weapon):  
 self.weapons.append(weapon)  
 print(f'Подобрал {weapon}')  
 else:  
 print('Это не оружие')  
  
 def next\_weapon(self):  
 if len(self.weapons) == 1:  
 print('У меня только одно оружие')  
 elif len(self.weapons) > 1:  
 self.current\_weapon += 1  
 if self.current\_weapon == len(self.weapons):  
 self.current\_weapon = 0  
 print(f'Сменил оружие на {self.weapons[self.current\_weapon]}')  
 else:  
 print('Я безоружен')  
  
 def heal(self, amount):  
 if self.hp < 200:  
 self.hp += amount  
 print(f'Полечился, теперь здоровья {self.hp}')  
  
  
weapon1 = Weapon("Короткий меч", 5, 1)  
weapon2 = Weapon("Длинный меч", 7, 2)  
weapon3 = Weapon("Лук", 3, 10)  
weapon4 = Weapon("Лазерная орбитальная пушка", 1000, 1000)  
princess = BaseCharacter(100, 100, 100)  
archer = BaseEnemy(50, 50, weapon3, 100)  
armored\_swordsman = BaseEnemy(10, 10, weapon2, 500)  
archer.hit(armored\_swordsman)  
armored\_swordsman.move(10, 10)  
print(armored\_swordsman.get\_coords())  
main\_hero = MainHero(0, 0, "Король Артур", 200)  
main\_hero.hit(armored\_swordsman)  
main\_hero.next\_weapon()  
main\_hero.add\_weapon(weapon1)  
main\_hero.hit(armored\_swordsman)  
main\_hero.add\_weapon(weapon4)  
main\_hero.hit(armored\_swordsman)  
main\_hero.next\_weapon()  
main\_hero.hit(princess)  
main\_hero.hit(armored\_swordsman)  
main\_hero.hit(armored\_swordsman)



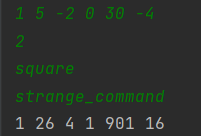
**14 задание**

class MailClient:  
 def \_\_init\_\_(self, server, user):  
 self.server = server  
 self.user = user  
  
 def receive\_mail(self):  
 if self.server:  
 self.server = []  
 return "Все сообщения получены"  
  
 def send\_mail(self, server1, user1, message):  
 server1[user1] = message  
 return server1  
  
  
server = []  
user1 = 'ivanov'  
client1 = MailClient(server, user1)  
print(client1.receive\_mail())  
server1 = {}  
server.append(server1)  
user2 = 'antonov'  
client2 = MailClient(server, user2)  
message1 = 'Привет, как дела?'  
print(client1.send\_mail(server1, user2, message1))  
print(client2.receive\_mail())  
message2 = 'Привет, у меня все классно, а ты как поживаешь?'  
server.append(server1)  
print(client2.send\_mail(server1, user1, message2))  
print(client1.receive\_mail())



**15 задание**

class Transformation:  
 def \_\_init\_\_(self, numbers):  
 self.numbers = numbers  
  
 def new\_action(self, command):  
 if command == 'make\_negative':  
 for i in range(len(self.numbers)):  
 number = int(self.numbers[i])  
 if number > 0:  
 number \*= -1  
 self.numbers[i] = number  
  
 elif command == 'square':  
 for i in range(len(self.numbers)):  
 number = int(self.numbers[i])  
 number \*\*= 2  
 self.numbers[i] = number  
  
 elif command == 'strange\_command':  
 for i in range(len(self.numbers)):  
 number = int(self.numbers[i])  
 if number % 5 == 0:  
 number += 1  
 self.numbers[i] = number  
  
 return self.numbers  
  
  
numbers = input().split()  
count = int(input())  
  
for i in range(count):  
 transformation = Transformation(numbers)  
 command = input()  
 numbers = transformation.new\_action(command)  
  
print(\*numbers)   
# 1 5 -2 0 30 -4  
# 2  
# square  
# strange\_command



**16 задание**

class Transformation:  
 def \_\_init\_\_(self):  
 self.functions = {}  
  
 def f\_x(self, x):  
 return x  
  
 def sqrt\_fun(self):  
 return self.f\_x(x) \*\* 0.5  
  
 def define(self, name\_f, f1, operator, f2):  
 def f():  
 if operator == '+':  
 return f1() + f2()  
 elif operator == '-':  
 return f1() - f2()  
 elif operator == '\*':  
 return f1() \* f2()  
 else:  
 return f1() / f2()  
  
 self.functions[name\_f] = f()  
  
 def calculate(self, f, \*spisok):  
 for i in spisok:  
 self.functions[f](i)  
  
  
n = int(input())  
name\_f = []  
for i in range(n):  
 string = input()  
 string = string.split()  
 if string[0] == 'define':  
 name\_f.append(string[1])  
 x = string[2]  
 operator = string[3]  
 f2 = string[4]  
 transformation = Transformation()  
 transformation.define(name\_f, x, operator, f2)  
  
# 2  
# calculate x -2 0 1 2  
# calculate sqrt\_fun 0 1 2 3 4

