**Урок No16. Классы и объекты**

1.

# Задание No1

# Создайте класс Касса, который хранит текущее количество денег в кассе, у

# него есть методы:

# ● top\_up(X) - пополнить на X

# ● count\_1000() - выводит сколько целых тысяч осталось в кассе

# ● take\_away(X) - забрать X из кассы, либо выкинуть ошибку, что не

# достаточно денег

#

# https://github.com/A-l-E-v/PySynergy/blob/main/U-16/cash.py

#

# функция печати тысяч в кассах и точных сумм

def total():

print (f'В кассе 0 сейчас {Kassa0.count\_1000()} тысяч, точная сумма: {Kassa0.amount}.')

print (f'В кассе 1 сейчас {Kassa1.count\_1000()} тысяч, точная сумма: {Kassa1.amount}.')

print (f'В кассе 2 сейчас {Kassa2.count\_1000()} тысяч, точная сумма: {Kassa2.amount}.')

print()

# пустой класс ошибки

class NotEnoughMoney(ValueError):

pass

class Kassa():

# инициализация начального баланса

def \_\_init\_\_ (self, amount=0):

self.amount = amount

# пополнение на X

def top\_up(self,X):

self.amount += X

# пересчёт тысяч в кассе

def count\_1000(self):

return self.amount//1000

# выдача суммы X

def take\_away(self,X):

if self.amount >= X:

self.amount -=X

else:

# return False

# поднимаем ошибку изъятия денег

raise ValueError(NotEnoughMoney)

# проверим работу трёх касс

Kassa0 = Kassa() # пустая касса

Kassa1 = Kassa (150000)

Kassa2 = Kassa (7500000)

total()

# пополним три кассы на 1 тыс

Kassa0.top\_up(1000)

Kassa1.top\_up(1000)

Kassa2.top\_up(1000)

total()

# пробуем снять по 2 тысячи из каждой кассы

try:

Kassa0.take\_away(2000)

except ValueError:

print ('В кассе 0 недостаточно денег.')

print()

try:

Kassa1.take\_away(2000)

except ValueError:

print ('В кассе 1 недостаточно денег.')

print()

try:

Kassa2.take\_away(2000)

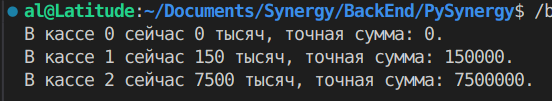
except ValueError:

print ('В кассе 2 недостаточно денег.')

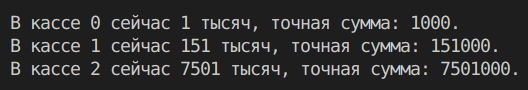
print()

total()

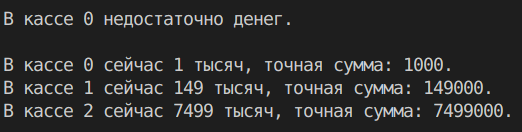
Проверим работу, создав 3 кассы:



Пополним три кассы на 1000:



Попробуем забрать по 2 тысячи из каждой кассы:



2.

# Задание No2

# Создайте класс Черепашка, который хранит позиции x и y черепашки, а также

# s - количество клеточек, на которое она перемещается за ход

# у этого класс есть методы:

# go\_up() - увеличивает y на s

# go\_down() - уменьшает y на s

# go\_left() - уменьшает x на s

# go\_right() - увеличивает y на s

# evolve() - увеличивает s на 1

# degrade() - уменьшает s на 1 или выкидывает ошибку, когда s может

# стать ≤ 0

# ● count\_moves(x2, y2) - возвращает минимальное количество действий, за

# которое черепашка сможет добраться до x2 y2 от текущей позиции

#

# https://github.com/A-l-E-v/PySynergy/blob/main/U-16/turtle.py

#

# для округления в бОльшую сторону

import math

# функция печати координат и перемещения трёх черепах

def t\_print():

print()

print (f'Черепашка 1 находит в ({t1.x}, {t1.y}). Скорость: {t1.s} кл/шаг.')

print (f'Черепашка 2 находит в ({t2.x}, {t2.y}). Скорость: {t2.s} кл/шаг.')

print (f'Черепашка 3 находит в ({t3.x}, {t3.y}). Скорость: {t3.s} кл/шаг.')

# пустой класс ошибки перемещения черепахи на s позиций с 0 скоростью

class sZero(ValueError):

pass

class Turtle():

# конструктор координат и перемещения s

def \_\_init\_\_(self, x = 0, y = 0, s = 1):

self.x = x

self.y = y

self.s = s

def go\_up(self):

self.y += self.s

def go\_down(self):

self.y -= self.s

def go\_left(self):

self.x -= self.s

def go\_right(self):

self.x += self.s

def evolve(self):

self.s +=1

def degrade (self):

self.s -=1

if self.s > 0:

return

else:

# ставим черепашке минимальную скорость

self.s = 1

raise ValueError(sZero)

# функция подсчёта шагов

def count\_moves(self, x2, y2):

self.x2 = x2

self.y2 = y2

# необходимые перемещения по обеим координатам

x21 = abs(self.x2-self.x)

y21 = abs(self.y2-self.y)

print (f'По координате Х позиций: {x21}')

print (f'По координате Y позиций: {y21}')

# количество шагов - общий путь делим на скорость и округляем в бОльшую сторону

steps = math.ceil((x21+y21)/self.s)

# черепашка находится рядом с заданными координатами, но может не хватить 1 шага

if steps == 0:

print ('Черепашка почти прибыла!')

steps = 1

return steps

# создадим 3 черепах, первая по умолчанию находится в начале координат, остальные в заданных точках

t1 = Turtle()

t2 = Turtle(10,10,1)

t3 = Turtle (-2,-3,2)

t\_print()

print()

print ('Переместим 1-ую черепашку налево вниз, 2-ую направо вверх, 3-ю направо вниз')

t1.go\_left()

t1.go\_down()

t2.go\_right()

t2.go\_up()

t3.go\_right()

t3.go\_down()

t\_print()

print()

print ('Ускорим 1-ую черепашку налево вверх, замедлим 2-ую налево вниз, замедлим 3-ю налево вниз')

t1.evolve()

t1.go\_left()

t1.go\_down()

try:

t2.degrade()

except ValueError:

print ('Черепашка 2 уже на минимальной скорости!')

t2.go\_right()

t2.go\_up()

try:

t3.degrade()

except ValueError:

print ('Черепашка 3 уже на минимальной скорости!')

t3.go\_right()

t3.go\_down()

t\_print()

# соберём черепашек вместе

print()

print (f'Черепашка 1 может попасть в (4, -3) за {t1.count\_moves(4, -3)} шаг(а)ов.')

print ()

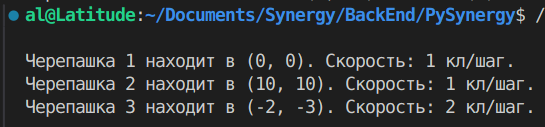
print (f'Черепашка 2 может попасть в (4, -3) за {t2.count\_moves(4, -3)} шаг(а)ов.')

print ()

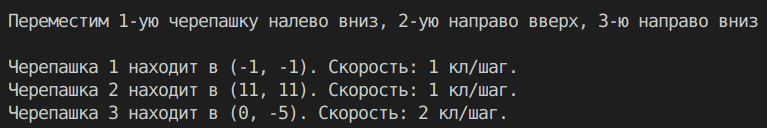
print (f'Черепашка 3 может попасть в (4, -3) за {t3.count\_moves(4, -3)} шаг(а)ов.')

Испытаю программу:

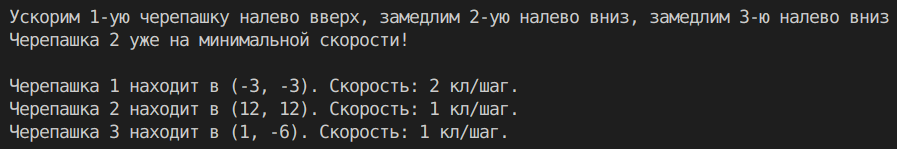
Начальная расстановка черепашек и задание скорости каждой:



Пусть каждая черепашка сделает по 2 шага — 1 по вертикали и 1 по горизонтали:



Попробуем ускорить первую черепаху и замедлим 2-ую и 3-ю и снова сделаем по 2 шага по осям X и Y:



Посчитаем какое количество шагов нужно сделать каждой черепашке, чтобы им собраться вместе в одной точке (4, -3) пройдя свой маршрут по вертикали и по горизонтали:

