**От автора:** Привет. Меня зовут Борис и я дистанционный репетитор по информатике. Помогаю школьникам готовится к экзаменам, обучаю программированию и созданию сайтов. Я просмотрел много книг для начинающих программистов, но ни одну не могу порекомендовать своим ученикам: в некоторых мало примеров, в других странная, на мой взгляд, очередность изложения материала или нет самого интересного и т.д. Все это послужило причиной создания данной книги. Она нужна мне, она нужна моим ученикам. Возможно, она будет полезна и для вас.

**Цель книги:** Вы держите в руках учебник-самоучитель по программированию с большим количеством практических заданий, небольшими вставками теоретического материала и массой ссылок на другие ресурсы: книги, сайты.

**Кому будет полезна книга**: Книга ориентирована на школьников, которые хотят узнать, что такое программирование. Книга будет полезна родителям, которые догадываются, что программирование - это не те скучные задачи, которые дает школа.

Возраст, с которого можно читать книгу: 12 лет.

Книга будет полезна и взрослым, которые хотят помочь своему ребенку разобраться в программировании.

**Какие навыки потребуются:** Понимание (20-40) английских слов (см. справочник), некоторые знания по математике (понятия круг, треугольник, площадь, количество, угол, градусы), умение работать с клавиатурой (см. справочник), т.е. ничего особенного не требуется: достаточно знаний, которыми обладают школьники ко третьему классу, и настойчивости, которую приобретают далеко не все. Может быть, именно занятия программированием помогут сформировать вашу настойчивость? В любом случае, настойчивость потребуется.

**Структура книги**: Первая часть предназначена для начинающих, поэтому все действия объясняются очень подробно: что нажать, как запустить, что сделать с ошибками. Ко второй части следует переходить только после полного освоения первой, т.к. объяснение многих деталей в ней опущено (я полагаю, что это было усвоено в первой части). И, наконец, третья часть предназначена для уже опытных учеников, которым нужны интересные задачи на направления. Среди заданий для самостоятельной работы вы найдете задания на написание программ, задание на поиск ошибок в имеющейся программе, задания на поиск информации в Интернете (или справочной системе) (см. справочник о том, как и где искать информацию), задания на логическое мышление. Задачи на написание программ, в свою очередь, можно разделить на задачи на повторение (сделать в точности то, что уже делали), «просто задания» и (ВНЕЗАПНО!) … **нерешаемые задания**. Последние никак выделены не будут. Т.е. неизвестно заранее, решаемая задача или нет. Чаще всего «нерешаемая задача» просто требует новых знаний, но иногда – это совсем нерешаемая задача. Так бывает в жизни, не вижу причины, почему так не может быть в обучении.

**Как работать с книгой**: Делать задания. Задавать вопросы на форуме: progras.ru/first\_step. Искать информацию. Повторять задания. Еще раз повторять задания. Пожалуй, это стоит объяснить подробнее: не важно, выполняете вы задание для самостоятельной работы или здание, разобранное в книге, в любом случае его стоит сделать несколько раз. Если, выполняя задание, вам пришлось подсмотреть решение (или заглянуть в справочник), это значит, что нужно выполнить это задание еще раз, сразу или через некоторое время. Это очень важно! От этого зависит то, насколько хорошо вы будете понимать следующие темы. Все ученики, которые сошли с дистанции, сошли только из-за того, что не стали отрабатывать задания до стопроцентного результата. И, с другой стороны, все, кто не стал отрабатывать - все сошли. Я предлагаю сотрудничество: я подбираю разнообразные и интересные (по возможности) задания, а вы их добросовестно выполняете. Договорились?

**Выделение:**

Вот так будет выделен код программ

1. А это – задания, которые нужно выполнять. Нумерация сквозная. Будем считать, что первое задание – это прочитать данное введение

Промежуточные итоги в конце каждой главы. Если вы не понимаете что-либо из написанного в этом блоке, то стоит еще раз выполнить задания или обратиться на форум за помощью.

Держите под рукой справочник (см. справочник – ссылка на справочник).

Задачи на логику вы найдете в приложении (ссылка на задачи на логику)

Глава первая

О выборе языка программирования

Начинать можно с любого языка: с машинных кодов (как я начинал), Python, Ruby, Perl, С++, Basic, Pascal, Java, JavaScript, даже PHP.

Вопрос лишь в том, сколько придется потратить время на изучение нюансов языка, а сколько - на само программирование. Имея достаточную мотивацию можно “продраться” через любые “дебри”. Но стоит ли оно того? Я считаю, что первый язык программирования должен быть прост, понятен, красив. В тоже время обладать богатыми возможностями, чтобы можно было сделать что-нибудь интересное сразу. Неплохо, если это будет настоящий язык, а не учебный. Под такое описание идеально подходит Python. (см. справочник: о выборе языка)

Я давно использую Python в качестве первого языка программирования. До этого пробовал VBA+Excel (вполне достойная связка), еще раньше - Pascal/Delphi.

Итак, преимущества языка Python: мощный современный язык, обладающий высокой дружественностью к начинающим.

**Установка, настройка, запуск первой программы**

инструкции по установке .. тысячи их. (см. справочник: установка)  
возможные ошибки при запуске, важность пробелов и отступов

Глава вторая

Теперь у вас есть Черепашка!

Первые программы мы будем создавать с помощью специального графического модуля turtle (черепашка). Этот модуль создан специально для начинающих изучать программирование.   
Простая программа: подключить модуль Черепашка и показать окно:

from turtle import \*  
done()

Чтобы запустить программу нажмите F5 (см. справочник: работа с редактором)

Нарисуем линию:

from turtle import \*  
fd(100)  
done()

fd(100) это тоже самое, что и forward(100): команда Черепашке двигаться вперед на 100 шагов. В результате должна получится прямая линия.

А что получится в результате такой программы? Прежде, чем запустить, попробуйте это представить.

rt(90) – тоже самое, что и right(90)

from turtle import \*

rt(90)  
fd(100)  
done()

Что произошло? Почему рисунок именно такой?

Еще одна программа:

from turtle import \*  
fd(100)  
rt(90)  
fd(100)  
done()

Для того, чтобы перемещаться, не оставляя следа, черепашке нужно поднять перо командой up().

from turtle import \*  
fd(50)

up()  
fd(20)

down()  
fd(50)

done()

Задания для самостоятельного выполнения:

1. Нарисовать прямую линию горизонтально
2. Нарисовать две параллельные линии горизонтально
3. Нарисовать пунктирную линию (три штриха)
4. Нарисовать крест
5. Нарисовать квадрат
6. Нарисовать прямоугольник
7. Нарисовать зигзаг (90 градусов)
8. Нарисовать зигзаг как у знака «опасные повороты»
9. Нарисовать равносторонний треугольник
10. Нарисовать окружность (см. справочник команд Черепашки)
11. Решить 4-6 задачу из списка задач на логику

Черепашка узнает новые команды

Команды Черепашки ограничены тем, что заложили в нее создатели. Но как быть, если нам требуется нарисовать много прямоугольников? Мы создадим новую команду, которую назовем sq (от слова square):

from turtle import \*  
def sq():  
 fd(100)  
 rt(90)  
 fd(100)

rt(90)  
 fd(100)

rt(90)  
 fd(100)

done()

Что произошло после запуска данной программы? Подсказка: «ничего» - неправильный ответ. Правильный ответ должен звучать так: программа запустилась без ошибок, показала окно с Черепашкой, ничего не нарисовала. Слова словами, но ведь ничего не нарисовано! И это правильно, ведь команды рисовать не было, а было объяснение новой команды “sq”. Чтобы воспользоваться этой командой нужно написать sq(). Обратите внимание на скобки: без скобок программа запустится и не покажет ошибок, однако нарисовано ничего не будет. На самом деле def – это создание функции (другие названия: подпрограмма, вспомогательный алгоритм, процедура). Пока мы работаем с Черепашкой это будет удобно называть «созданием новой команды».

from turtle import \*  
def sq():  
 fd(100)  
 rt(90)  
 fd(100)

rt(90)  
 fd(100)

rt(90)  
 fd(100)  
sq()

done()

Теперь нарисуем два прямоугольника рядом, на расстоянии 50.

from turtle import \*  
def sq():  
 fd(100)  
 rt(90)  
 fd(100)

rt(90)  
 fd(100)

rt(90)  
 fd(100)  
sq()  
fd(50)  
sq()

done()

Думаю, будет лучше поднять перо после рисования. И опустить – перед рисованием:

from turtle import \*  
def sq():

down()  
 fd(100)  
 rt(90)  
 fd(100)

rt(90)  
 fd(100)

rt(90)  
 fd(100)

up()  
sq()

done()

Для красоты добавим заливку цветом:

from turtle import \*  
def sq():

color('red','yellow')

begin\_fill()  
 down()

fd(100)

rt(90)  
 fd(100)

rt(90)  
 fd(100)

rt(90)

fd(100)

up()  
 end\_fill()

sq()

fd(50)  
sq()

done()

1. Нарисовать 4 квадрата (80х80) в линию по горизонтали, с расстоянием между квадратами, равным 20
2. Создать команду «Прямоугольник» (rect)
3. Нарисовать 6 прямоугольников 80х60 в два ряда
4. Создать команду «Треугольник» (равносторонний)
5. Нарисовать квадрат 50х50, 30х30, 90х90, 10х10, 40х40

При выполнении последнего задания у вас должна была появится мысль, что здесь что-то не так, что должен быть какой-либо другой удобный способ рисования разных квадратов. Конечно же, он существует: для этого нужно использовать параметр:

from turtle import \*  
def sq(a):  
 color('red','yellow')

begin\_fill()  
 down()

fd(a)

rt(90)  
 fd(a)

rt(90)

fd(a)  
 rt(90)

fd(a)  
 up()

end\_fill()

sq(100)  
fd(50)

sq(20)

done()

1. Изменить команду «Квадрат» так, чтобы можно было рисовать квадрат с произвольного размера
2. Изменить команду «Прямоугольник» так, чтобы можно было рисовать прямоугольник произвольного размера (потребуется два параметра. Объявлять: def rect(a,b), вызывать: rect(100,80)
3. Изменить команду «Треугольник», чтобы можно было рисовать треугольник произвольного размера
4. Нарисовать 10 треугольников разного размера, вложенных друг в друга
5. Игральный кубик: нарисовать квадрат, внутри которого – одну окружность (в центре, как у )
6. Игральный кубик: нарисовать квадрат, внутри которого – две окружности
7. Решить 7-10 задачу из списка задач на логику

Созданные функции не удаляйте – они еще нам потребуются

Генератор случайных чисел

В процессе обучения мы широко будем использовать генератор случайных чисел. Это будет полезно как в играх, например для случайной расстановки бомб, так и для обычных тренировочных задач. Чтобы получить случайное число, надо подключить модуль random и воспользоваться функцией randrange(), которая получает в качестве аргумента (см. справочник на то, что такое функция, что такое аргумент, результат) максимально возможное значение (почти). randrange(10) вернет случайное число в промежутке от 0 до 9, а print(a) позволит вывести это число на экран (в консоль). Запустите эту программу несколько раз:

import random  
a = random.randrange(10)  
print(a)

Нельзя сказать, что random.randrange слишком удобно печатать – уж слишком длинно. Мы можем упростить себе жизнь, если подключим только функцию randrange:

from random import randrange  
a = randrange(10)  
print(a)

Стало лучше, но можно еще короче!

from random import randrange as rnd  
a = rnd(10)  
print(a)

Теперь попробуем нарисовать квадрат случайного размера:

from turtle import \*  
def sq(a):  
 color('red','yellow')  
 begin\_fill()  
 down()   
 fd(a)  
 rt(90)  
 fd(a)  
 rt(90)  
 fd(a)  
 rt(90)  
 fd(a)  
 up()  
 end\_fill()  
a = rnd(10,100)  
sq(a)  
done()

На этот раз функция rnd (randrange) получает два агрумента: минимальное значение и максимальное значение. Не слишком-то интересно рисовать квадраты очень маленького размера, не так ли?

1. Нарисовать окружность, размер выбрать случайным образом от 20 до 100
2. Нарисовать окружность в случайном месте
3. Нарисовать 10 окружностей в случайных местах, случайно выбранным размером
4. Нарисовать 10 окружностей в случайных местах, цветом, и размером, выбранным случайным образом
5. Решить 10-13 задачу из списка задач на логику

Цикл for. Повторение участка кода

Для того, чтобы не писать одно и тоже по нескольку раз, придумали циклы. Что будет выполнено в результате работы такой программы:

from turtle import \*  
for x in range(6):  
 circle(30)  
 fd(20)  
done()

1. Нарисовать 4 окружности подряд, не соединенные линией
2. Нарисовать 10 окружностей, не соединенных линией

Кстати, на мой взгляд Черепашка двигается слишком медленно. Нельзя ли ее как-нибудь ускорить?

1. Найти команду, которая позволяет увеличить скорость движения Черепашка (если такая есть)

Теперь посмотрим на распространённую ошибку:

from turtle import \*  
for x in range(6):  
circle(30)  
fd(20)  
done()

Программа не запускается, верно? Дело в том, что тело цикла (см. справочник), должно быть сдвинуто вправо.

И еще один вариант:

from turtle import \*  
for x in range(6):  
 circle(30)  
fd(20)  
done()

В этом случае программа будет работать, но что она сделает? Почему?

Продолжим работу с циклом for: нарисуем 12 окружностей, расположенных в случайном месте

from turtle import \*  
from random import randrange as rnd  
speed(100)  
up()  
for z in range(12):  
 x = rnd(-200,200)  
 y = rnd(-200,300)  
 goto(x,y)  
 down()  
 circle(30)  
 up()  
done()

Неплохо, но … скучновато… Может быть, добавим цвет?

from turtle import \*  
from random import randrange as rnd  
speed(100)  
up()  
for z in range(12):  
 x = rnd(-200,200)  
 y = rnd(-200,300)  
 goto(x,y)  
 down()  
 begin\_fill()

circle(30)  
 end\_fill()

up()

done()

Ой, вот это «добавили цвет»! Для изменения цвета линий и заливки воспользуемся командой color():

…  
from random import randrange as rnd  
color('red','yellow')  
speed(100)  
…

И добавим еще больше случайности:

from turtle import \*

from random import randrange as rnd  
color('red','yellow')

speed(100)

up()

for z in range(12):

x = rnd(-200,200)

y = rnd(-200,300)

goto(x,y)

down()  
 begin\_fill()

circle(rnd(10,50))  
 end\_fill()

up()

done()

Следующий шаг – это сделать случайный цвет.

Список – как способ хранения данных

Сначала мы создадим список возможных цветов:

colors = ['red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

Теперь, безо всякой случайности, выберем оранжевый и синий цвет (нумерация идет с нуля):

оранжевый - colors[1]

синий – colors[5]

Убедимся, что это работает:

from turtle import \*

from random import randrange as rnd  
colors = ['red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

color(colors[5],colors[1])

speed(100)

up()

for z in range(12):

x = rnd(-200,200)

y = rnd(-200,300)

goto(x,y)

down()  
 begin\_fill()

circle(rnd(10,50))  
 end\_fill()

up()

done()

Остался один шаг – выбирать случайный номер. Для этого нужно знать длину списка. Можно написать так: rnd(10), но так делать не стоит. Ведь мы можем захотеть удалить или добавить цвет. Придется переделывать код программы, что не очень удобно. Лучше мы узнаем длину списка: len(colors), и это значение подставим в функцию rnd: rnd(len(colors)). Теперь по номеру возьмем цвет:

from turtle import \*

from random import randrange as rnd

colors = ['red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

color(colors[rnd(len(colors))],colors[rnd(len(colors))])

speed(100)

up()

for z in range(12):

x = rnd(-200,200)

y = rnd(-200,300)

goto(x,y)

down()

begin\_fill()

circle(rnd(10,50))

end\_fill()

up()

done()

Работает, но … цвет одинаковый! При каждом запуске – свой, но одинаковый у всех окружностей. Так не пойдет – нужно перенести выбор цвета внутрь цикла:

from turtle import \*

from random import randrange as rnd

colors = ['red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

speed(100)

up()

for z in range(12):

color(colors[rnd(len(colors))],colors[rnd(len(colors))])

x = rnd(-200,200)

y = rnd(-200,300)

goto(x,y)

down()

begin\_fill()

circle(rnd(10,50))

end\_fill()

up()

done()

Ну вот так намного веселее. Особенно, если добавить еще кружков.

1. Нарисовать 10 окружностей в случайном месте, случайным радиусом, случайным цветом
2. Нарисовать 20 квадратов в случайном месте, случайным размером, случайным цветом
3. Нарисовать случайное количество (не менее 10 и не более 30) окружностей в случайном месте, случайным цветом
4. Нарисовать 10 прямоугольников в случайном месте, случайного цвета, случайного размера
5. \*Создать команду «Звезда» для рисования пятиконечной звезды
6. \*Нарисовать 10 звезд случайным цветом в случайном месте, случайного цвета
7. \*Изменить команду «Звезда», чтобы можно было рисовать звезду с произвольным количеством лучей
8. \*Нарисовать 10 звезд случайным цветом, в случайном месте, случайного размера, со случайным количеством лучей
9. \*\*Что умеет функция random.choice?

Если задачи со \*звездочкой не получатся – не страшно. Они требуют знаний геометрии на уровне 8-9 класса. Чуть позже я дам решение.

Еще немного задач

1. Решить 14-17 задачу из списка задач на логику
2. Дано задание написать функцию рисования узора, изображенного на рисунке . Функция должна принимать параметр - размер узора. Программист ошибся, помогите ему исправить ошибки:

from turtle import \*

def zz(a):

down()

fd(45)

lt(45)

fd(90)

rt(90)

fd(90)

lt(45)

fd(45)

up()

zz(1000)

done()

1. Программист написал программу для рисования квадрата. Помогите исправить ошибки

from turtle import \*

def sq():

down()

fd(50)

lt(90)

fd(50)

lt(90)

fd(50)

lt(90)

fd(50)

up()

sq

done()

1. Программист хотел написать программу для рисования 10 окружностей в ряд, но у него получилась только одна. Помогите найти и исправить ошибку:

from turtle import \*

for z in range(8):

down()

circle(20)

up()

fd(40)

done()

1. Программист хотел написать программу для рисования узора. Как должен выглядеть узор он объяснить вам не смог, но просит помочь понять, почему программа не запускается:

from turtle import \*

speed(100)

for z in range(60)

fd(5)

circle(120)

rt(6)

done()

1. Программист нашел в справочнике программу и не может понять, какая связь между числами, которые появляются на экране. Помогите ему разобраться.

from random import randrange as rnd

for z in range(15):

x = rnd(-100,100)

print(x)

1. Программист нашел пример, но не может понять, как он работает. Помогите ему разобраться и объясните, что делает данная программа:

from random import randrange as rnd

a = [10,30,40,100]

for z in range(10):

num = rnd(4)

print(a[num])

Промежуточные итоги

Создание новых команд (функций)

Использование параметров в функциях

Случайные числа

Повторение кода несколько раз (цикл for)

Создание списка возможных цветов и выбор случайного цвета из этого списка.

print() для вывода информации на экран (в консоль)

Проверьте себя: вы должны понимать код, который написан ниже.

def zz(a):

down()

fd(a)

rt(90)

fd(a)

lt(90)

fd(a)

lt(90)

fd(a)

a = rnd(10,100)

for x in range(12):

circle(20)

fd(50)

colors = [‘red’,’blue’,’yellow’,’green’]

print(colors[rnd(4)])

А вот и обещанные звезды. Не то, чтобы совсем «звезды», но иногда – вполне звезды.

from turtle import \*

from random import randrange as rnd

colors = ['red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

speed(100)

def star(z,n):

a = 360/n

for d in range(n):

fd(z)

rt(180-a)

up()

for z in range(120):

color(colors[rnd(len(colors))],colors[rnd(len(colors))])

x = rnd(-200,200)

y = rnd(-200,300)

goto(x,y)

down()

begin\_fill()

star(50,rnd(1,30))

end\_fill()

up()

done()

Глава третья

Tkinter вместо Черепашки

Пришла пора сменить графическую систему отображения. К Черепашке мы еще вернемся (с ее помощью удобно рисовать фракталы), а пока оставим ее и начнем рисовать в Tkinter. Tkinter – это один из способов создать «окошки и кнопки» на Python. Кроме Tkinter существуют и другие, самые известные: Qt, wxForms. Tkinter удобен тем, что он входит в стандартную поставку Python, а значит доступен всегда (почти всегда). Недостаток – маловато возможностей для построения сложных систем. Создадим простую программу:

from tkinter import \*

root = Tk()

mainloop()

Первая строка подключает модуль tkinter. Вторая строка – создает окно (главное и единственное). Третья – запускает цикл обработки событий (см. справочник: зачем нужен главный событийный цикл).

Немного изменим программу. Прежде, чем ее запустить, подумайте, что должно измениться?

from tkinter import \*

root = Tk()

root.geometry('800x600')

mainloop()

Создадим кнопку:

from tkinter import \*

root = Tk()

root.geometry('800x600')

bt1 = Button(root,width=8,text='Go!')

mainloop()

Ошибок нет, но нет и кнопки! Причина в том, что создать кнопку мало – надо ее размесить на форме. За это отвечает метод pack() (см. справочник – параметры pack)

from tkinter import \*

root = Tk()

root.geometry('800x600')

bt1 = Button(root,width=8,text='Go!')

bt1.pack()

mainloop()

1. Создать кнопку и прижать ее к левой стороне окна
2. Создать две кнопки и прижать их к левой стороне окна
3. Создать две кнопки в верхней части окна. Кнопки расположить горизонтально (вторая справа от первой), в одной строке
4. Решить 18-21 задачу из списка задач на логику

На последней задаче остановимся подробнее: если просто написать

bt1.pack(side='left')

bt2.pack(side='left')

То кнопки будут расположены горизонтально, но не в верхней части окна. Другой вариант:

bt1.pack(side='top')

bt2.pack(side='top')

так же не дает нужного результата: кнопки в верхней части окна, но друг под другом.

Выход из этой ситуации есть и он заключается в использовании Frame. Мы создадим невидимый контейнер, в который поместим кнопки. Прижмем их к левому краю Frame, а сам контейнер – к верхней границе окна:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt1 = Button(fr,width=8,text='1')

bt2 = Button(fr,width=8,text='2')

bt1.pack(side='left')

bt2.pack(side='left')

fr.pack()

mainloop()

Неплохо, но кнопки «слиплись». Это некрасиво. Исправим, использую padx и pady:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt1 = Button(fr,width=8,text='1')

bt2 = Button(fr,width=8,text='2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5)

mainloop()

И снова будем рисовать. Для этого создадим Canvas

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt1 = Button(fr,width=8,text='1')

bt2 = Button(fr,width=8,text='2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

mainloop()

Проверим, можем ли мы рисовать?

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt1 = Button(fr,width=8,text='1')

bt2 = Button(fr,width=8,text='2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

canv.create\_rectangle(30,50,120,80,fill='green')

mainloop()

1. Что означают 30,50,120,80? Как это понимать?
2. Нарисуйте квадрат примерно в центра окна, размером 100

Жмем на кнопки. Возникают события. Обрабатываем!

Можно рисовать сразу при создании окна, как мы это делали. Не самое лучшее решение. Усложним программу еще немного, чтобы потом было проще: создадим функцию, которую будем вызывать по нажатию кнопки «1»: (см. справочник использование bind)

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt1 = Button(fr,width=8,text='1')

bt2 = Button(fr,width=8,text='2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def paint(event):

canv.create\_rectangle(30,50,120,80,fill='green')

bt1.bind('<Button-1>',paint)

mainloop()

Запустите программу, нажмите на кнопку «1», чтобы нарисовать зеленый прямоугольник. Все работает, однако было сделано одно допущение, на которое следует обратить внимание: внутри функции paint не существует переменной canv, но мы можем к ней обратиться, потому что она есть в основной программе. Не всегда так получится сделать: (см. справочник – область видимости переменных).

Доработаем программу: добавим кнопку для очистки Canvas, а также изменим названия кнопок и надписи на них:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt\_paint = Button(fr,width=8,text='paint')

bt\_clear = Button(fr,width=8,text='clear')

bt\_paint.pack(side='left',padx=2)

bt\_clear.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

def paint(event):

canv.create\_rectangle(30,50,120,80,fill='green')

bt\_paint.bind('<Button-1>',paint)

bt\_clear.bind('<Button-1>',clear)

mainloop()

При нажатии на paint – рисуется зеленый прямоугольник, при нажатии на clear – очищается Canvas (холст). В дальнейшем мы будем менять только содержимое функции paint, остальной код останется прежним.

1. Что означает <Button-1>? (см. справочник событий)
2. Что будет, если указать просто <Button>?
3. Что будет, если указать Button-1 без треугольных скобок?
4. Какова роль метода bind? Что он делает?
5. Программист создал программу из примера, но допустил ошибки. Помогите ему разобраться и запустить программу:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt\_paint = Button(fr,width=8,text='paint')

bt\_clear = Button(fr,width=8,text='clear')

bt\_paint.pack(side='left',padx=2)

bt\_clear.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

def paint(event):

canv.create\_rectangle(30,50,120,80,fill='green')

bt\_paint.bind('<Button-1>',paint())

bt\_clear.bind('<Button-1>',clear())

mainloop()

1. Нужно было сделать окно, в котором расположена одна кнопка, которая рисует красную окружность. Программа запускается, но не похоже, чтобы она что-то рисовала. Помогите найти и исправить одну ошибку:

from tkinter import \*

root = Tk()

root.geometry('300x300')

bt\_paint = Button(root,width=8,text='paint')

bt\_paint.pack()

canv = Canvas(root,bg='white')

canv.pack

def paint(event):

canv.create\_rectangle(30,50,120,80,fill='green')

bt\_paint.bind('<Button-1>',paint)

mainloop()

Порисуем?

Задания выполнять в функции paint. Пример некоторых фигур:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt\_paint = Button(fr,width=8,text='paint')

bt\_clear = Button(fr,width=8,text='clear')

bt\_paint.pack(side='left',padx=2)

bt\_clear.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

def paint(event):

canv.create\_rectangle(30,50,120,80,fill='green')

canv.create\_line(10,100,150,10,fill='blue',width=5)

canv.create\_rectangle(200,200,500,500)

canv.create\_oval(200,200,500,500)  
 canv.create\_text(350,350,text='Hello!')

bt\_paint.bind('<Button-1>',paint)

bt\_clear.bind('<Button-1>',clear)

mainloop()

1. Нарисовать окружность с центром в точке 300,200, и радиусом 50 (см. справочник Tkinter Canvas)
2. Нарисовать закрашенную окружность в точке 300,200, и радиусом 50
3. Нарисовать линию
4. Нарисовать красную линию
5. Нарисовать красную линию толщиной 3 из точки 10,300 в точку 300,300
6. В какую сторону «растет» координата Y? Почему?
7. Какие названия цветов можно использовать?
8. Можно ли задать произвольный цвет, не имеющий названия?
9. Какие фигуры еще можно рисовать?
10. Решить 22-24 задачу из списка задач на логику

Вспомним цикл for и rnd

В первой строке добавьте подключение функции randrange из модуля random:

from tkinter import \*

from random import randrange as rnd

root = Tk()

...

1. Нарисовать 10 окружностей в случайном месте и случайного размера
2. Нарисовать 12 окружностей в случайном месте, случайного размера и цвета

Для задания случайного цвета будем использовать тот же список цветов, что и ранее (я добавил еще несколько штук):

colors = ['lightyellow', 'lightgray', 'gray', 'pink', 'violet', 'brown', 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'magenta', 'black', 'gray', 'lightgreen']

С помощью генератора случайных чисел выберем случайный номер в пределах длины списка:

col\_num = rnd(len(colors))

И нарисуем кружок выбранного цвета:

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

Все вместе:

colors = ['lightyellow', 'lightgray', 'gray', 'pink', 'violet', 'brown', 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'magenta', 'black', 'gray', 'lightgreen']

def paint(event):

for z in range(12):

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

col\_num = rnd(len(colors))

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

Оператор выбора

Посчитаем, сколько синих кругов мы нарисовали:

def paint(event):

k = 0

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

r = rnd(5,30)

col\_num = rnd(len(colors))

if colors[col\_num] == 'blue':

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

print(k)

Результат подсчета появится в консоли. За это отвечает уже знакомый **print**, но можно вывести и прямо на canvas:

def paint(event):

k = 0

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

r = rnd(5,30)

col\_num = rnd(len(colors))

if colors[col\_num] == 'red':

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])  
 canv.create\_text(20,20,text=str(k), font = '18')

А теперь посчитаем, сколько было синих или бирюзовых, для этого заменим строку:

if colors[col\_num] == 'red':

на

if colors[col\_num] == 'blue' or colors[col\_num] == 'cyan':

На мой взгляд, писать colors[num\_col] неудобно. Изменим выбора цвета:

def paint(event):

k = 0

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

r = rnd(5,30)

color = colors[rnd(len(colors))]

if color == 'blue' or color == 'cyan':

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)  
 canv.create\_text(20,20,text=str(k), font = '18')

В предыдущем примере мы сначала выбирали номер случайным образом, потом брали элемент списка с этим номером каждый раз, когда нужно было узнать выбранный цвет. Теперь мы сразу берем элемент списка с выбранным номером и запоминаем название цвета.

1. \*Попробуйте реализовать выбор из списка цветов с помощью функции choice из модуля random
2. or – это логический оператор. Какие еще логические операторы бывают? См. справочник.

Будем рисовать красные круги случайным радиусом, а остальные – радиусом 20:

def paint(event):

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

color = colors[rnd(len(colors))]

if color == 'red':

r = rnd(10,40)

else:

r = 20

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

В данном случае мы видим выбор из двух вариантов. Если условие, записанное в if верно, то блок else не выполняется. А если неверно, то – выполняется.

Рассмотрим еще один пример. Не запуская, попробуйте определить, что он делает:

def paint(event):

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

color = colors[rnd(len(colors))]

if color == 'red':

r = rnd(10,40)

else:

if color == 'pink':

r = 50

else:

r = 10

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

А что, если у нас для каждого цвета есть свой вариант? Так мы быстро запутаемся в «лесенке» if… К счастью, Python позволяет организовать множественный выбор более простым способом:

def paint(event):

for z in range(30):

x = rnd(100,700)

y = rnd(100,500)

color = colors[rnd(len(colors))]

if color == 'red':

r = rnd(10,40)

elif color == 'pink':

r = 50

elif color == 'magenta':

r = 100

else:

r = 20

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

Итак, оператор выбора позволяет писать сложные условия (вы же заглянули в справочник, не так ли?).

Проверьте себя: вы должны понимать примеры, приведенные ниже:

if color == ‘red’:

if color != ‘red’:

if color == ‘red’ or color ==’blue’:

if color = ‘red’ and r > 20:

if 10 < r < 30:

if color ==’red’:

r = 30

else:

r = 50

if color == ‘blue’ or ‘cyan’:

r = 10

elif color == ‘red’ or color == ‘pink’:

r= 20

else:

r – rnd(10,50)

А теперь немного практики:

1. Нарисовать 20 случайных кругов, посчитать, сколько нарисовано красных
2. Нарисовать 20 случайных кругов, посчитать, сколько было нарисовано зеленых и светло-зеленых
3. Нарисовать 20 случайных кругов, посчитать, сколько было нарисовано маленьких (радиус меньше 20) красных и больших синих (радиус больше 50)
4. \*Нарисовать 20 случайных кругов, посчитать общую площадь красных (на то, что они частично перекрываются не обращать внимание: считать сумму площадей) Площадь круга: math.pi\*r\*\*2
5. Нарисовать 20 случайных кругов. Если фигура левее центра – красным, если правее – синим.
6. Решить 25-28 задачу на логику.
7. Нужно было посчитать количество зеленых кругов, но программист торопился и допустил две ошибки. Исправьте их:

def paint(event):

for z in range(30):

k = 0

x = rnd(100,700)

y = rnd(100,500)

color = colors[rnd(len(colors))]

r = rnd(10,40)

if color == 'red':

k = 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

print(k)

1. Нужно было посчитать количество маленьких, но нашего друга опять постигла неудача: программа выдает странный результат. Он ждет вашей помощи:

def paint(event):

for z in range(30):

k = 0

x = rnd(100,700)

y = rnd(100,500)

color = colors[rnd(len(colors))]

r = rnd(10,40)

if r <= 15:

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

print(k)

1. Программист написал программу, которая должна кидать круги, радиусом 10, в экран, если попадает не далее, чем два радиуса от центра, то красным цветом, если не далее четырех радиусов от центра – то синим, если не более 8-ми радиусов от центра, то зеленым и желтым в противном случае. Программист утверждает, что все условия написаны правильно, ведь по-отдельности они работают. Помогите найти ошибку и доделать программу.

from tkinter import \*

from random import randrange as rnd

import math

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt\_paint = Button(fr,width=8,text='paint')

bt\_clear = Button(fr,width=8,text='clear')

bt\_paint.pack(side='left',padx=2)

bt\_clear.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

def paint(event):

xc = 400

yc = 300

for z in range(55):

x = rnd(50,750)

y = rnd(50,500)

r = 15

color = 'yellow'

if (x-2\*r) < xc < (x+2\*r) and (y-2\*r) < yc < (y+2\*r):

color = 'red'

if (x-4\*r) < xc < (x+4\*r) and (y-4\*r) < yc < (y+4\*r):

color = 'blue'

if (x-8\*r) < xc < (x+8\*r) and (y-8\*r) < yc < (y+8\*r):

color = 'green'

canv.create\_oval(x-r,y-r,x+r,y+r,fill=color,width=0)

bt\_paint.bind('<Button-1>',paint)

bt\_clear.bind('<Button-1>',clear)

mainloop()

Цикл for – это не только повторение несколько раз

1. Нарисовать 10 окружностей с центром в точке 400, 300 с увеличивающимся радиусом (первая – 10, вторая – 20 и т.д.)
2. Нарисовать 10 окружностей в ряд
3. Нарисовать 10 окружностей в ряд, с увеличивающимся радиусом
4. Решить 25-27 задачу из списка задач на логику

Нарисуем 10 окружностей с увеличивающимся радиусом:

def paint(event):

for z in range(12):

x = 400

y = 300

r = z\*10

canv.create\_oval(x-r,y-r,x+r,y+r)

А можно и по-другому:

def paint(event):

for r in range(10,100,10):

x = 400

y = 300

canv.create\_oval(x-r,y-r,x+r,y+r)

Первое аргумент в range – это начальное значение, второй – конечное, третий – шаг. Таким образом r принимает значение 10, потом 20 и т.д. до 100. Но до 100 ли?

1. \*Сколько окружностей нарисовано? Сколько должно быть? Почему?

Нарисуем 12 окружностей в ряд:

def paint(event):

for i in range(10):

x = 20+i\*60

y = 300

r = i\*3

col\_num = rnd(len(colors))

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

1. \*Нарисовать в ряд 10 окружностей с увеличивающимся радиусом так, чтобы они касались друг друга
2. Нарисовать 5 рядов по 10 окружностей (радиуса 20, чтобы поместились)
3. Нарисовать 5 рядов по 10 окружностей так, чтобы в каждом был только один цвет заливки, но выбран он должен быть случайным образом. Т.е. при каждом запуске – новые цвета. Но для каждого ряд – цвет один.
4. Нарисовать 5 рядов окружностей. В первом 3, во втором 6, в третьем 9 и т.д.
5. Нарисовать 5 рядов по 10 окружностей или квадратов. Выбирать фигуру случайным образом

Для того, чтобы выполнить эти задания нам потребуется использовать цикл в цикле:

def paint(event):

for y in range(100,400,50):

for x in range(100,600,30):

r = 10

col\_num = rnd(len(colors))

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

Еще один пример для разбора:

def paint(event):

for y in range(100,400,50):

col\_num = rnd(len(colors))

for x in range(100,y+200,30):

r = 10

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

Используем случайное число для выбора фигуры:

def paint(event):

for y in range(100,400,50):

for x in range(100,y+200,30):

col\_num = rnd(len(colors))

r = 10

if rnd(2) == 1:

canv.create\_oval(x-r,y-r,x+r,y+r,fill = colors[col\_num])

else:

canv.create\_rectangle(x-r,y-r,x+r,y+r,fill = colors[col\_num])

Как вы можете видеть, появилось новое слово: **else.** Код, написанный в блоке else выполнится в том случае, если условие под if – ложно (см. справочник: оператор ветвления). Также обратите внимание на два знака «равно» в строке if rnd(2) == 1. (см. справочник операторы)

1. Решить 29-31 задачу на логику
2. Разобрались сами – поможем другому. Программисту нужно было написать программу для рисования десяти квадратов, вложенных друг в друга, но получается только один:

from tkinter import \*

root = Tk()

root.geometry('800x600')

fr = Frame(root)

bt\_paint = Button(fr,width=8,text='paint')

bt\_clear = Button(fr,width=8,text='clear')

bt\_paint.pack(side='left',padx=2)

bt\_clear.pack(side='left',padx=2)

fr.pack(pady=5)

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

def paint(event):

x = 300

y = 200

for r in range(10,100,10):

canv.create\_rectangle(x-r,y-r,x+r,y+r, fill = 'lightgreen')

bt\_paint.bind('<Button-1>',paint)

bt\_clear.bind('<Button-1>',clear)

mainloop()

1. Еще одна программа (часть программы), которая работает неправильно. Нужно было нарисовать 10 рядов по 10 окружностей:

def paint(event):

r = 10

for x in range(30,350,22):

for y in range(30,350,22):

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

Проверьте себя: вы должны понимать, почему на экране появится число 100:

k = 0

for x in range(10):

for y in range(10):

k += 1

print(k)

Проверьте себя: вы должны понимать, почему на экране появится число 10:

for x in range(10):

k = 0

for y in range(10):

k += 1

print(k)

Проверьте себя: вы должны понимать, почему на 10 раз экране появится число 10:

for x in range(10):

k = 0

for y in range(10):

k += 1

print(k)

Повторять, пока не получится. Цикл while

Выводить на экран случайные круги, пока не появится красный:

def paint(event):

while color != 'red':

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

Однако, вместо результата мы видим ошибку. Python сообщает нам, что прежде, чем мы будем использовать color, мы должны поместить туда какое-либо значение. Исправим ошибку, заодно добавим вызов clear() в начало paint():

def paint(event):

color = ''

clear(0)

while color != 'red':

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

Обратите внимание на то, что clear() вызывается с аргументом 0: clear(0). На самом деле с таким же успехом можно использовать любое значение. Можете это проверить. Дело в том, что функция clear описана так, что она должна получать значение: clear(event), мы не можем вызывать ее без аргумента. Когда clear вызывается при наступлении события, то она получает информацию о событии (в скором времени мы на это посмотрим), но если мы решили вызвать ее сами, то … нужно тоже что-нибудь передать, ведь clear ждет этого.

Вернемся к while: этот цикл будет работать, пока истинно условие, записанное в его заголовке. В данном случае, пока цвет не окажется красным.

Усложним задачу: выводить на экран случайные круги, пока не будет нарисовано два красных:

def paint(event):

k = 0

clear(0)

while k < 2:

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,30)

color = colors[rnd(len(colors))]

if colors[col\_num] == 'red':

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

Еще больше усложним задачу, и будем рисовать круги, пока не будет нарисовано два красных и два синих:

def paint(event):

k\_red = 0

k\_blue = 0

clear(0)

while k\_red < 2 or k\_blue < 2:

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,30)

color = colors[rnd(len(colors))]

if color == 'red':

k\_red += 1

if color == 'blue':

k\_blue += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

1. Рисовать случайные круги, пока не будет нарисован зеленый круг
2. Рисовать случайные круги, пока не будет нарисован зеленый или светло-зеленый круг
3. Рисовать случайные круги, пока не будет получен круг, радиусом больше 28
4. Рисовать случайные круги, пока не будет нарисовано три зеленых круга
5. Рисовать случайные круги, пока не будет нарисовано три зеленых или три синих круга
6. Рисовать случайные круги, пока не будет нарисовано три зеленых и три синих круга
7. Рисовать случайные круги, пока не будет нарисовано два красных подряд
8. Рисовать случайные круги, пока не будет нарисовано три красных круга. Если за это время будет нарисован хотя бы один зеленый, то начинать считать красные сначала
9. Рисовать случайные круги. После того, как будет нарисовано 3 красных – красные и розовые больше не рисовать. После того, как будет нарисовано 3 синих – не рисовать синие и голубые. После того, как будет нарисовано 3 зеленых – не рисовать зеленые и светло-зеленые. Закончить, когда будет нарисовано 7 серых или светло-серых кругов.
10. Решить 32-34 задачу из списка задач на логику

Программисту нужно было написать программу, которая рисует круги, пока не будет нарисован большой и красный круг. Но программа не запускается. Помогите найти ошибку.

colors = ['violet', 'brown', 'red', 'orange', 'yellow', 'green','blue']

def paint(event):

clear(0)

k = 0

while color != 'red' and r < 50:

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

k += 1

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

canv.create\_text(x,y, text = str(k))

Промежуточные итоги

Проверьте себя: вы должны понимать приведенный ниже код:

Ошибка:

def paint:

print(‘test’)

На экране появится число 10:

def test(a):

print(a)  
test(10)

Ошибка:

def test(a):

print(a)

test()

Ошибка:

def test():

a = 10

print(a)

test(10)

На экране будет число 12:

def test(a=12):

print(a)

test(a)

На экране будет число 12:

k = 0

for z in range(12):

k += 1

print(k)

Окружности «идут» вниз и вправо:

def paint(event):

clear(0)

x = 100

y = 100

r = 20

for z in range(12):

x += 30

y += 30

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

canv.create\_text(x,y, text = str(z))

Окружности в ряд, радиус увеличивается

def paint(event):

clear(0)

x = 100

y = 300

r = 10

for z in range(12):

r += 5

x += 30

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'green')

Цикл остановится, когда будет нарисован круг, радиусом больше или равно 50:

def paint(event):

clear(0)

r = 0

k = 0

while r < 50:

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

k += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'green')

canv.create\_text(x,y, text = str(k))

Будет нарисовано три красных круга, после чего цикл остановится:

colors = ['violet', 'brown', 'red', 'orange', 'yellow', 'green','blue']

def paint(event):

clear(0)

r = 1

k = 0

while k < 3:

x = rnd(100,700)

y = rnd(100,500)

r = rnd(10,70)

color = colors[rnd(len(colors))]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

if color =='red':

k += 1

canv.create\_text(x,y, text = str(k))

Будет выведен «+»

a = 13

b = 12

if a > b:

print(‘+’)

else:

print(‘-’)

Будет выведен «+»

a = 13

if a > 10 and a < 20:

print(‘+’)

else:

print(‘-’)

Будет выведен «+»

a = 13

if 10 < a < 20:

print(‘+’)

else:

print(‘-’)

Будет выведен «+»

a = 13

if not (a <= 10 or a >= 20):

print(‘+’)

else:

print(‘-’)

Будет выведен «+», потом «-»

a = 13

if a > 10:

print(‘+’)

else:

print (‘-’)

if a > 20:

print(‘+’)

else:

print(‘-’)

Будет выведен «+++»

a = 13

if a > 30:

print('+')

elif a > 20:

print('++')

elif a > 10:

print('+++')

else:

print('-')

Теперь вы знаете основные операторы и умеете ими пользоваться (def, for, while, if..else), понимаете, что такое переменная и чем отличается список от переменной. Наш вводный курс подходит к концу и настала пора создать простую игру!

Глава четвертая

Мышка оставляет след. Работаем с информацией о событии (event).

Рассмотрим такой пример:

from tkinter import \*

from random import randrange as rnd, choice

import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=600,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

def paint(event):

clear(0)

x = event.x

y = event.y

r = rnd(10,70)

color = choice(colors)

canv.create\_oval(x-r,y-r,x+r,y+r,fill = color)

canv.bind('<Button-1>',paint)

mainloop()

Что происходит?

canv.bind('<Button-1>',paint)

связывает событие «Button-1», т.е. щелчок левой кнопкой мыши на Canvas и вызов функции paint, т.е. теперь paint будет вызываться не тогда, когда мы нажмем кнопку в окне, а когда мы будем щелкать мышкой в разных местах Canvas, а event.x, event.y поможет нам узнать координаты щелчка.

Измените одну строку, но не запускайте сразу. Motion – это событие, которое возникает при перемещении мыши. Как вы думаете, что произойдет? Проверьте ваше предположение, запустив программу.

canv.bind('<Motion>',paint)

А теперь удалите вызов clear(0) в начале функции paint. И опять подумайте, к чему это должно привести и только потом запускайте.

Еще немного поиграем с мышью: в следующем примере слишком много нового, чтобы его было просто понять. Мы будем разбирать это позже, поэтому не переживайте, если некоторая часть кода не будет понятна. Это забавно и стоит того, чтобы попробовать прямо сейчас.

from tkinter import \*

from random import randrange as rnd

import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=550,bg='white')

canv.pack()

def clear(event):

canv.delete(ALL)

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

tail = []

def paint(event):

global tail

x = event.x

y = event.y

tail += [(x,y)]

tail = tail[-20:]

r = 20

clear(0)

for t in tail:

x = t[0]

y = t[1]

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'lightgreen')

canv.bind('<Motion>',paint)

mainloop()

Забавно, не так ли? Но черную окантовку окружности лучше убрать:

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'lightgreen', width=0)

И еще один прием:

from tkinter import \*

from random import randrange as rnd

import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=550,bg='lightblue')

canv.pack()

def clear(event):

canv.delete(ALL)

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

tail = []

def paint(event):

global tail

x = event.x

y = event.y

tail += [(x,y)]

tail = tail[-30:]

r = 2

clear(0)

for t in tail:

x = t[0]

y = t[1]

r += 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'yellow', width=0)

canv.bind('<Motion>',paint)

mainloop()

Наоборот тоже забавно:

tail = []

def paint(event):

global tail

x = event.x

y = event.y

tail += [(x,y)]

tail = tail[-30:]

r = 30

clear(0)

for t in tail:

x = t[0]

y = t[1]

r -= 1

canv.create\_oval(x-r,y-r,x+r,y+r,fill = 'yellow', width=0)

1. Что означает global? (см. справочник – область видимости)
2. Что делает tail = tail[-30:]? (см. справочник - списки)

Создаем первую игру: поймай шарик

Суть игры проста: в случайном месте появляется на короткое время шарик и мы должны щелкнуть в нем мышкой.

Вначале создадим появляющиеся шарики:

from tkinter import \*

from random import randrange as rnd, choice   
import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=550,bg='lightblue')

canv.pack()

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

def new\_ball():

canv.delete(ALL)

x = rnd(100,700)

y = rnd(100,500)

r = rnd(30,50)

canv.create\_oval(x-r,y-r,x+r,y+r,fill = choice(colors), width=0)

root.after(1000,new\_ball)

def click(event):

pass

root.after(1000,new\_ball)

canv.bind('<Button-1>',click)

mainloop()

Теперь определим, что щелчок произошел на самом круге, для этого нужно знать координаты круга, его радиус и координаты щелчка. Изменим программу так, чтобы можно было работать с нужными данными:

from tkinter import \*

from random import randrange as rnd, choice

import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=550,bg='lightblue')

canv.pack()

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

x = 0

y = 0

r = 0

def new\_ball():

global x,y,r

canv.delete(ALL)

x = rnd(100,700)

y = rnd(100,500)

r = rnd(30,50)

canv.create\_oval(x-r,y-r,x+r,y+r,fill = choice(colors), width=0)

root.after(1000,new\_ball)

def click(event):

print(x,y,r)

root.after(1000,new\_ball)

canv.bind('<Button-1>',click)

mainloop()

Использование global – это не самое лучшее решение. Для данной задачи больше подходит использование ООП (объектно-ориентированного подхода), но об этом мы поговорим в следующей части. А пока – продолжим. Изменим функцию click, добавив вывод координат щелчка:

def click(event):

print(x,y,r)  
 print (event.x,event.y)

Осталось проверить, не лежит ли точка event.x,event.y дальше, чем r от точки x,y:

def click(event):

print (abs(x-event.x) < r and abs(y-event.y)<r)

abs – функция, которая возвращает модуль. Т.е. мы проверяем, расстояние (не важно в бОльшую или меньшую сторону) больше чем радиус или нет? Чтобы считать очки придется создать еще одну глобальную переменную, также добавим вывод набранных очков:

from tkinter import \*

from random import randrange as rnd, choice

import time

root = Tk()

root.geometry('800x600')

canv = Canvas(root,width=800,height=550,bg='lightblue')

canv.pack()

colors = ['lightyellow','lightgray','gray','pink','violet','brown','red','orange','yellow','green','cyan','blue','magenta','black','gray','lightgreen']

x = 0

y = 0

r = 0

points =0

def new\_ball():

global x,y,r

canv.delete(ALL)

canv.create\_text(20,20,text=str(points), font = 'Arial 20')

x = rnd(100,700)

y = rnd(100,500)

r = rnd(30,50)

canv.create\_oval(x-r,y-r,x+r,y+r,fill = choice(colors), width=0)

root.after(1000,new\_ball)

def click(event):

global points

if abs(x-event.x) < r and abs(y-event.y) < r:

points += 1

root.after(1000,new\_ball)

canv.bind('<Button-1>',click)

mainloop()

Чтобы нельзя было «накручивать» очки, щелкая много раз по кругу, пока он не исчез, нужно сразу после удачного попадания удалять его с экрана. Заодно и обновлять информацию об очках.

def click(event):

global points

if abs(x-event.x) < r and abs(y-event.y)<r:

points += 1

canv.delete(ALL)

canv.create\_text(20,20,text=str(points), font = 'Arial 20')

Введем подсчет промахов:

x = 0

y = 0

r = 0

points = 0

miss = 0

def new\_ball():

global x,y,r

canv.delete(ALL)

canv.create\_text(20,20,text=str(points)+'/'+str(miss), font = 'Arial 20')

x = rnd(100,700)

y = rnd(100,500)

r = rnd(30,50)

canv.create\_oval(x-r,y-r,x+r,y+r,fill = choice(colors), width=0)

root.after(1000,new\_ball)

def click(event):

global points, miss

if abs(x-event.x) < r and abs(y-event.y)<r:

points += 1

canv.delete(ALL)

canv.create\_text(20,20,text=str(points)+'/'+str(miss), font = 'Arial 20')

else:

miss += 1

Промахи появляются не сразу и это нехорошо. Придется еще усложнить программу, чтобы удалять объекты не все сразу, а по отдельности. Для этого нужно дать имена как кругу, так и объекту «text».

def new\_ball():

global x, y, r, res

canv.delete(ALL)

res = canv.create\_text(60,20,text=str(points)+'/'+str(miss), font = 'Arial 20')

x = rnd(100,700)

y = rnd(100,500)

r = rnd(30,50)

target = canv.create\_oval(x-r,y-r,x+r,y+r,fill = choice(colors), width=0)

root.after(1000,new\_ball)

def click(event):

global points, miss, res

if abs(x-event.x) < r and abs(y-event.y)<r:

points += 1

canv.delete(ALL)

res = canv.create\_text(60,20,text=str(points)+'/'+str(miss), font = 'Arial 20')

else:

miss += 1

canv.delete(res)

res = canv.create\_text(60,20,text=str(points)+'/'+str(miss), font = 'Arial 20')

Если честно, то есть другой, более удобный способ определять по какому графическому объекту щелкнули. Советую посмотреть справочник по Tkinter. Мы рассмотрим некоторые из этих возможностей во второй части книги.

Что стоит сделать еще:

1. Изменить подсчет очков, в зависимости от цвета или радиуса круга
2. Изменить подсчет очков, в зависимости от точности попадания. Ближе к центру круга – больше очков
3. Добавить случайности во время появления круга
4. Добавить подсчет упущенных кругов
5. Заканчивать игру по времени, по очкам или по количеству выстрелов
6. Хранить таблицу лучших результатов
7. Сопровождать звуков попадание и промах
8. Усложнить управление, добавив инерцию в прицел
9. Заменить курсор мыши на изображение прицела
10. Менять изображение прицела при наведении на цель

Я предлагаю поработать на этими заданиями самостоятельно. Жду ваших вопросов и результатов на форуме progras.ru/first\_step

Во второй части мы рассмотрим основы объектно-ориентированного программирования (ООП), сделаем еще несколько игр и простых программ, порешаем «классические задачи на программирование».

Третья часть посвящена более глубокому знакомству с языком Python.

В четвертой части решение «классических учебных» задач, таких, которые встречаются на экзаменах и олимпиадах.

В пятой части мы уйдем в Интернет и начнем создавать web-приложения.

В шестой, заключительной, будем создавать сайт с помощью Django.

Во второй части появятся задания для выполнения в группе (паре, тройке). Если никто из ваших друзей не хочет вместе с вами изучать программирование, то приходите на форум progras.ru/first\_step – ищите там. Работать в команде интереснее и эффективнее, чем одному.

Часть вторая.

Глава пятая. Игра «Пушка»

Создадим игру пушка. Суть игры состоит в том, чтобы выстрелить мячиком и попасть в мишень. Первый этап – научиться создавать анимацию.

Анимация

Анимация – это последовательная смена кадров. Чем чаще меняются кадры и чем меньше они различаются, тем плавнее получится анимация. Создадим шарик, который просто «едет» вправо:

from tkinter import \*

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

x = 40

y = 450

r = 20

b = canv.create\_oval(x-r,y-r,x+r,y+r)

for z in range(100):

x += 5

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

mainloop()

Но все происходит слишком быстро, чтобы можно было увидеть движение. Мы видим только конечный результат. Чтобы увидеть процесс, добавим задержку:

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

x = 40

y = 450

r = 20

b = canv.create\_oval(x-r,y-r,x+r,y+r)

for z in range(100):

x += 5

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

mainloop()

Неплохо, если не считать мерцание. С мерцанием мы бороться не будем, просто смиримся с ним. Для создания игр больше подходит pygame, но наша основная задача на данный момент – это не создание качественных игр, а создание игр как обучение программированию.Поэтому мы ограничимся tkinter’ом и не будем обращать внимание на мерцание.

А теперь сделаем так, чтобы шарик менял скорость:

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

x = 40

y = 450

r = 20

v = 30

b = canv.create\_oval(x-r,y-r,x+r,y+r)

for z in range(100):

x += v

v \*= 0.9

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

mainloop()

1. Чем умножение в формуле v \*= 0.9 удобнее, чем деление на 10? Можно было бы написать v /= 10. Чем это чревато?
2. Чем умножение в той же формуле удобнее вычитания? Можно было бы написать v -= 0.7. Чем это хуже?
3. Зачем нужно строка canv.update()? Чтобы будет, если ее убрать?
4. Что передается первым параметром у метода coords: canv.coords(b,x-r,y-r,x+r,y+r)?

v \*=0.9 – это уменьшение скорости на 10%.

Но из пушки мячик должен вылетать под углом. Не все сразу и для начала запустим шарик вверх:

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

x = 40

y = 450

r = 20

v = 30

b = canv.create\_oval(x-r,y-r,x+r,y+r)

for z in range(100):

y -= v

v \*= 0.9

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

mainloop()

В случае вертикального движения v \*= 0.9 плохое решение, ведь нам не нужно, чтобы мячик «завис в воздухе»?

Исправим это:

…  
for z in range(100):

#x += v

#vx \*= 0.9

vy += 1.2

y += vy

canv.coords(b,x-r,y-r,x+r,y+r)

1. Почему при vy \*= 0.9 замедление оканчивается остановкой, а при vy += 1.2 – движением в обратную сторону?
2. Почему у «растет» не вверх, а вниз?

Когда шарик долетит до нижней части экрана – он должен остановится:

for z in range(100):

#x += v

#vx \*= 0.9

if y <= 500:

vy += 1.2

y += vy

canv.coords(b,x-r,y-r,x+r,y+r)

Теперь просто включим x:

x = 40

y = 450

r = 20

vx = 30

vy = -30

b = canv.create\_oval(x-r,y-r,x+r,y+r)

for z in range(100):

x += vx

vx \*= 0.97

if y <= 500:

vy += 1.2

y += vy

canv.coords(b,x-r,y-r,x+r,y+r)

Неплохо, но на мой взгляд, стоит изменить скорость по x, чтобы полет был более естественным:

x = 10

y = 450

r = 8

vx = 15

vy = -20

b = canv.create\_oval(x-r,y-r,x+r,y+r,fill='black')

for z in range(100):

if y <= 500:

vy += 1.2

y += vy

x += vx

vx \*= 0.99

else:

vy \*= -0.3

vx \*= 0.4

y += vy

x += vx

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

Еще я сделал так, чтобы в конце полета шарик не останавливался, а «начинал новый полет» со значениями скорости 40%, чтобы он немного попрыгал после приземления.

Сделаем несколько выстрелов. Чтобы не повторять код, оформим его как функцию, с параметрами vx,vy:

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

def fire(vx,vy):

x = 10

y = 450

r = 8

b = canv.create\_oval(x-r,y-r,x+r,y+r,fill='black')

for z in range(100):

if y <= 500:

vy += 1.2

y += vy

x += vx

vx \*= 0.99

else:

y = 501

vy \*= -0.3

vx \*= 0.4

y += vy

x += vx

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

fire(15,-20)

fire(19,-18)

fire(20,-19)

mainloop()

А теперь будет указывать начальную скорость с помощью мыши:

from random import randrange as rnd

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

def fire(event):

x = 10

y = 450

vx = (event.x-x)/10

vy = (event.y-y)/10

r = 8

b = canv.create\_oval(x-r,y-r,x+r,y+r,fill='black')

for z in range(100):

if y <= 500:

vy += 1.2

y += vy

x += vx

vx \*= 0.99

else:

y = 501

vy \*= -0.3

vx \*= 0.4

y += vy

x += vx

canv.coords(b,x-r,y-r,x+r,y+r)

canv.update()

time.sleep(0.03)

canv.bind('<Button>',fire)

mainloop()

Ну, как вам «застревающие в небе» мячи? Бороться с этим можно двумя способами: созданием потоков (см. справочник – Потоки) и изменением функции fire так, чтобы она занималась просчетом не одного мяча, а всех, которые находятся на поле. Потоки – это довольно сложно, мы посмотрим на них в следующей части. Да и нет необходимости в них в этой задаче, если честно. Потоки используют для более серьезных вещей. А играх намного удобнее использовать обычный список объектов и обрабатывать их по очереди. Я долго думал, как же построить изложение, чтобы подвести к необходимости использования ООП в данной задаче, но все способы оказались муторными и долгими. Поэтому поступим проще: я предлагаю поверить мне на слово. Просто взять и поверить. Переходим к ООП.

Объекты – это экземпляры классов

Первое, что мы сделаем – это поместим всю информацию о шарике внутрь класса ball. Класс – это «шаблон», по которому мы будем создавать объекты (экземпляры класса).

class ball():

def \_\_init\_\_(self):

self.x = 0

self.y = 0

self.r = 0

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

В данном случае id – это имя графического примитива, которое нужно нам, чтобы перемещать и удалять нарисованный кружок.

def вам уже знакомо – это объявление функции. В данном случае эта функция называется метод, потому что связана с классом. Например, есть функция len(a), которая вернет количество элементов любой последовательности: строки, списка, кортежа, словаря. А есть метод a.sort(), который будет сортировать не какой-нибудь, а конкретный список. Методы всегда пишутся через точку после имени объекта.

Поскольку мы пишем шаблон, то не можем знать, как будет называться конкретный экземпляр. Кроме того, предполагается, что их будет много, поэтому ссылка на себя нужна обязательно (см. справочник – объекты). Помните область видимости, глобальные переменные? Когда мы создавали простую функцию, то говорили, что изменения переменных внутри функции будут потеряны после завершения работы функции. Я бы сказал - жили с ней и умрут с ней. Чтобы сохранить изменения мы объявляли переменную глобальной. Примерно так же нужно работать с переменными внутри класса. Если нужна временная переменная, значение которой после работы метода не потребуется, то self использовать не нужно. Можно, но зачем плодить ненужные переменные? Если же требуется сохранить значение, то нужно использовать переменные с префиксом self. (см. справочник – объекты).

Метод \_\_init()\_\_ - особенный. На это указывают два знака подчеркивания до и после имени. Он называется конструктор и вызывается в момент создания экземпляра класса. Вам никогда не придется вызывать его явно, а только так:

b1 = ball()

Создадим другой метод, который будет отвечать за прорисовку изображения. Назовем set\_coords (можно называть как угодно):

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

А еще метод для пересчета координат и проверки удара о землю:

def move(self):

if self.y <= 500:

self.vy += 1.2

self.y += self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y += self.vy

self.x += self.vx

Для проверки создадим два шарика и попробуем их перемещать:

from random import randrange as rnd

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 0

self.color = 'black'

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy += 1.2

self.y += self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y += self.vy

self.x += self.vx

b1 = ball()

b2 = ball()

Не работает? Будем проверять. Первое, что попадает под подозрение – это создание шариков. Чтобы проверить, создаются ли они, мы добавим print(‘!’) в конструктор:

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 0

self.color = 'black'

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

print('!')

В консоли появляются два восклицательных знака. Значит шарики создаются. Но если посмотреть внимательно, то видно, что радиус равен нулю! Исправляем на 10 и проверяем еще раз. Должны появится. Правда, друг на друге.

Теперь задаем начальные значения для выстрела:

b1.vx = 19

b1.vy = -20

b2.vx = 23

b2.vy = -15

Но отрицательные значения смотрятся не очень красиво. Тем и хороши объекты, что можно скрыть реализацию и сделать так:

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y -= self.vy

self.x += self.vx

b1 = ball()

b2 = ball()

b1.vx = 19

b1.vy = 20

b2.vx = 23

b2.vy = 15

Выстрелим же наконец!

Для этого придется создать бесконечный цикл для пересчета координат и отрисовки шариков:

from random import randrange as rnd

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 10

self.color = 'black'

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y -= self.vy

self.x += self.vx

b1 = ball()

b2 = ball()

b1.vx = 19

b1.vy = -20

b2.vx = 23

b2.vy = -15

while 1:

b1.move()

b2.move()

canv.update()

time.sleep(0.03)

Не очень-то работает, верно?

Попробуйте понять, почему. И подумайте, как это можно проверить.

Не торопитесь смотреть учебник дальше.

Используйте print(‘!’) для того, чтобы проверить, что метод срабатывает. Вообще для поиска таких ошибок используют отладчик, но это тема для третей части учебника. Пока будет достаточно диагностического вывода, т.е. print. Выводите строки, выводите значения переменных.

Еще раз посмотрите на то, за что отвечают разные методы. Проверьте, все ли выполняют свою работу.

…  
 def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y -= self.vy

self.x += self.vx

self.set\_coords()  
…

Теперь-то все получится: два шарика вылетают (непонятно откуда) и летят вместе. Именно то, что нужно.

Для хранения неизвестного количества шариков будем использовать список:

balls = []

for z in range(12):

balls += [ball()]

balls[-1].vx = 10+rnd(10)

balls[-1].vy = 10+rnd(10)

Создали список, положили туда 12 элементов и установили у каждого случайные значения vx,vy.

ball[-1] – это обращение к последнему элементу списка.

Теперь заставим их всех лететь:

from random import randrange as rnd

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 10

self.color = 'black'

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.3

self.vx \*= 0.4

self.y -= self.vy

self.x += self.vx

self.set\_coords()

balls =[]

for z in range(12):

balls += [ball()]

balls[-1].vx = rnd(15)+10

balls[-1].vy = rnd(20)+5

while 1:

for b in balls:

b.move()

canv.update()

time.sleep(0.03)

mainloop()

Ну вот теперь, наконец-то, получилось сделать так, чтобы шарики летели все вместе.

Осталось добавить выстрел по щелчку мыши:

from random import randrange as rnd

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 10

self.color = 'black'

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

self.y = 501

self.vy \*= -0.4

self.vx \*= 0.4

self.y -= self.vy

self.x += self.vx

self.set\_coords()

balls = []

def fire(event):

global balls

balls += [ball()]

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

canv.bind('<Button-1>',fire)

while 1:

for b in balls:

b.move()

canv.update()

time.sleep(0.03)

mainloop()

Если вы протестировали этот код тщательно, то заметили, что на второй-третей сотни шариков из отображение начало тормозить. Это связано с тем, что программе приходится обрабатывать большое количество объектов. В pygame для решения это проблемы есть специальный инструмент, позволяющий задач оптимальную частоту кадров. Tkinter является пакетом для создание GUI, а не игр, поэтому у него нет ни двойной буферизации (см. справочник – буферизация) для борьбы с мерцанием, ни возможности управлять количеством кадров. Но мы можем менять задержку, определяя количество шариков и уменьшая задержку с каждым новым десятком:

z = 0.03

while 1:

for b in balls:

b.move()

canv.update()

z = 0.03 - 0.00001\*len(balls)

if z < 0:

z = 0

time.sleep(z)

mainloop()

Чтобы не мучать свою любимую мышку (и себя) – упростим процесс создания новых шариков, заменив событие <Button-1> на <Motion>:

canv.bind('<Motion>',fire)

Однако, нет в мире совершенства. Во всей красе проявляется самый серьезный недостаток Python – невысокая скорость работы. Программа, написанная на С или Pascal смогла бы выдержать создание во много раз большего количества объектов. Это проблема легко компенсируется использованием «движков» для создания игр (см. справочник – создание игр, движки). Обычно на Python не пишут обработку графики, а используют для обработки игровой логики (см. справочник – использование Python в играх). Но мы-то и не собирались крутую игру писать. Создание игр – это всего лишь более красочный способ обучения, не более.

Вернемся к задаче. Проблема есть и ее надо решить. Решим ее простым способом: зачем на просчитывать положение тех шариков, которые уже лежат неподвижно? Просто удалим их из списка. И я опять прошу простить использование глобальных переменных. Так будет проще.

Нам нужно найти элемент в списке и удалить его. Для этого находим номер:

balls.index(self.id)

А потом удаляем:

balls.pop(balls.index(self.id))

Можно и с холста удалить:

canv.delete(self.id)

Все вместе:

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

else:

balls.pop(balls.index(self))

canv.delete(self.id)

self.set\_coords()

А вот такой способ даст небольшую задержку перед удалением:

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

self.set\_coords()

else:

self.y -= self.vy

if self.y > 800:

balls.pop(balls.index(self))

canv.delete(self.id)

1. Почему использование такого способа задержки не дает одинаковое время задержки?
2. Придумайте другой способ (возможно придется ввести еще одно свойство ball)

90% работы сделано!

Осталось еще нарисовать флажок и определять попадание в этот флажок.

Чтобы не замусоривать код, напишем функцию для проверки столкновения:

def bum(ball):

if flag\_coords[0] <= ball.x <= flag\_coords[2] and flag\_coords[1] <= ball.y <= flag\_coords[3]:

return True

else:

return False

Теперь будем вызывать проверку это в основном цикле:

canv.bind('<Button-1>',fire)

z = 0.03

work = 1

while work:

for b in balls:

b.move()

if bum(b):

print ('!')

work = 0

canv.update()

time.sleep(0.03)

mainloop()

Но видеть «залипший экран» неинтересно. Добавим экран «конец игры»:

z = 0.03

work = 1

while work:

for b in balls:

b.move()

if bum(b):

work = 0

canv.update()

time.sleep(0.03)  
canv.delete(ALL)

canv.create\_text(400,300, text = 'Вы уничтожили цель за ' + str(bullet) + 'выстрелов',font = '28')

Неплохо, но слишком резко. Пусть долетят оставшиеся шарики:

z = 0.03

work = 1

while work or balls:

print (len(balls))

for b in balls:

b.move()

if bum(b):

canv.bind('<Button-1>','')

work = 0

canv.update()

time.sleep(0.03)

canv.bind('<Button-1>','')

Это строка лишает возможности выстрелить шариком и новых шариков больше появится не может. Остальные имеют возможность долететь. Каждый долетевший удаляется из списка и список становится все короче. Когда в списке не останется ни одного шарика – игра заканчивается и появляется надпись и результате.

Окончательный результат + пара «пасхальных яиц»

from random import randrange as rnd, choice

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

canv.create\_oval(750,450,766,466,fill='orange')

canv.create\_line(758,466,758,480,width=3)

flag\_coords = (745,445,770,485)

bullet = 0

def bum(ball):

if flag\_coords[0] <= ball.x <= flag\_coords[2] and flag\_coords[1] <= ball.y <= flag\_coords[3]:

return True

else:

return False

class ball():

def \_\_init\_\_(self):

self.x = 20

self.y = 450

self.r = 10

self.color = choice(['blue','green','red','brown'])

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

self.set\_coords()

else:

self.y -= self.vy

if self.y > 800:

balls.pop(balls.index(self))

canv.delete(self.id)

balls = []

def fire(event):

global balls

global bullet

bullet += 1

balls += [ball()]

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

def fire2(event):

global balls

global bullet

bullet += 1

balls += [ball()]

balls[-1].r += 20

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

def fire3(event):

global balls

global bullet

for x in range(10):

bullet += 1

balls += [ball()]

balls[-1].r = 5

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10+(x-10)

canv.bind('<Button-1>',fire)

canv.bind('<Button-3>',fire2)

canv.bind('<Button-2>',fire3)

z = 0.03

work = 1

while work:

for b in balls:

b.move()

if bum(b):

work = 0

canv.update()

time.sleep(0.03)

canv.delete(ALL)

canv.create\_text(400,300, text = 'Вы уничтожили цель за ' + str(bullet) + 'выстрелов',font = '28')

mainloop()

Добавим … пушку!

from random import randrange as rnd, choice

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

canv.create\_oval(750,450,766,466,fill='orange')

canv.create\_line(758,466,758,480,width=3)

flag\_coords = (745,445,770,485)

gun = canv.create\_line(20,450,50,420,width=7)

bullet = 0

def bum(ball):

if flag\_coords[0] <= ball.x <= flag\_coords[2] and flag\_coords[1] <= ball.y <= flag\_coords[3]:

return True

else:

return False

class ball():

def \_\_init\_\_(self,x=40,y=450):

self.x = x

self.y = y

self.r = 10

self.color = choice(['blue','green','red','brown'])

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

self.set\_coords()

else:

self.y -= self.vy

if self.y > 800:

balls.pop(balls.index(self))

canv.delete(self.id)

balls = []

def fire(event):

global balls

global bullet

bullet += 1

balls += [ball(canv.coords(gun)[2],canv.coords(gun)[3])]

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

def fire2(event):

global balls

global bullet

bullet += 1

balls += [ball()]

balls[-1].r += 20

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

def fire3(event):

global balls

global bullet

for x in range(10):

bullet += 1

balls += [ball()]

balls[-1].r = 5

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10+(x-10)

def pointing (event):

canv.coords(gun,20,450,20+(event.x-20)/10,450-(450-event.y)/10)

canv.bind('<Button-1>',fire)

canv.bind('<Button-3>',fire2)

canv.bind('<Button-2>',fire3)

canv.bind('<Motion>',pointing)

z = 0.03

work = 1

while work:

for b in balls:

b.move()

if bum(b):

work = 0

canv.update()

time.sleep(0.03)

canv.delete(ALL)

canv.create\_text(400,300, text = 'Вы уничтожили цель за ' + str(bullet) + 'выстрелов',font = '28')

mainloop()

Добавим рикошет, вывод надписи сразу после уничтожения пушки, и еще кое-что:

from random import randrange as rnd, choice

from tkinter import \*

import time

root = Tk()

fr = Frame(root)

root.geometry('800x600')

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

def bum(ball):

if flag\_coords[0] <= (ball.x + ball.r) and (ball.x - ball.r) <= flag\_coords[2] and flag\_coords[1] <= (ball.y + ball.r) and (ball.y - ball.r) <= flag\_coords[3]:

return True

else:

return False

class ball():

def \_\_init\_\_(self,x=40,y=450):

self.x = x

self.y = y

self.r = 10

self.color = choice(['blue','green','red','brown'])

self.id = canv.create\_oval(self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r,fill=self.color)

self.live = 30

def set\_coords(self):

canv.coords(self.id,self.x-self.r,self.y-self.r,self.x+self.r,self.y+self.r)

def move(self):

if self.y <= 500:

self.vy -= 1.2

self.y -= self.vy

self.x += self.vx

self.vx \*= 0.99

self.set\_coords()

else:

if self.vx\*\*2+self.vy\*\*2 > 10:

self.vy = -self.vy/2

self.vx = self.vx/2

self.y = 499

if self.live < 0:

balls.pop(balls.index(self))

canv.delete(self.id)

else:

self.live -= 1

if self.x > 780:

self.vx = - self.vx/2

self.x = 779

def fire(event):

global balls

global bullet

bullet += 1

balls += [ball(canv.coords(gun)[2],canv.coords(gun)[3])]

balls[-1].vx = (event.x-balls[-1].x)/10

balls[-1].vy = -(event.y-balls[-1].y)/10

def fire2(event):

global balls

global bullet

bullet += 1

balls += [ball()]

balls[-1].r += 20

balls[-1].vx = (event.x-balls[-1].x)/20

balls[-1].vy = -(event.y-balls[-1].y)/20

def fire3(event):

global balls

global bullet

for x in range(10):

bullet += 1

balls += [ball()]

balls[-1].r = 5

balls[-1].vx = (event.x-balls[-1].x)/15 + rnd(5)

balls[-1].vy = -(event.y-balls[-1].y)/15+(x-10) + rnd(5)

def pointing (event):

canv.coords(gun,20,450,20+(event.x-20)/10,450-(450-event.y)/10)

def new\_game(event=''):

canv.delete(ALL)

global gun, flag\_coords, bullet, balls

canv.create\_oval(750,450,766,466,fill='orange')

canv.create\_line(758,466,758,480,width=3)

flag\_coords = (745,445,770,485)

gun = canv.create\_line(20,450,50,420,width=7)

bullet = 0

balls = []

canv.bind('<Button-1>',fire)

canv.bind('<Button-3>',fire2)

canv.bind('<Button-2>',fire3)

canv.bind('<Motion>',pointing)

z = 0.03

work = 1

while work or balls:

for b in balls:

b.move()

if bum(b) and work:

work = 0

canv.bind('<Button-1>','')

canv.bind('<Button-3>','')

canv.bind('<Button-2>','')

canv.create\_text(400,300, text = 'Вы уничтожили цель за ' + str(bullet) + 'выстрелов',font = '28')

canv.update()

time.sleep(0.03)

canv.delete(ALL)

canv.create\_text(400,300, text = 'Вы уничтожили цель за ' + str(bullet) + 'выстрелов',font = '28')

canv.bind('<Button-1>',new\_game)

new\_game()

mainloop()

На это игру «Пушка» мы закончим.

Что еще можно сделать с пушкой? Немало. Позже мы вернемся к пушке и все это доделаем, а пока вы можете попробовать самостоятельно:

Сделать прицеливание с помощью удержания мыши. Дольше держишь – дальше летит.

Сделать прицеливание как в AngryBirds (как из рогатки)

Сделать мишень, которая будет появляться в случайных местах правой части экрана и находится 3-5 секунд на каждом месте.

Создать новый вид шариков: разрывные (при ударе о землю – разделяется на 8-10 мелких частей, резлетающихся в стороны)

Для каждого вида снарядов создать свой класс (см. справочник – наследование)

Добавить пушку противника. И сделать игру пошаговой

Сделать пушку перемещаемой (ограничить в дальности 50-80 точек)

Реализовать ограничение снарядов в количестве

Ограничить время хода 20 секундами и одним выстрелом

Добавить призы (как в червяках). С выбором – стрелять в противника или в приз: новый вид оружия, дальность и скорость перемещения, возможность сделать два выстрела

Добавить фоновое изображение

Создать разрушаемый рельеф

Реализовать «здоровье» в процентах. Чтобы не уничтожать с одного попадания.

Создать новый вид шарика: химическое оружие (при ударе о землю вырывается облако, причиняя кратковременный урон)

Создать «биологическое оружие» (при ударе вырывается облако, причиняя небольшой урон, но повторяющийся при каждом ходе)

Реализовать простой AI (ИИ) для игры с компьютером

…. (продолжение следует)