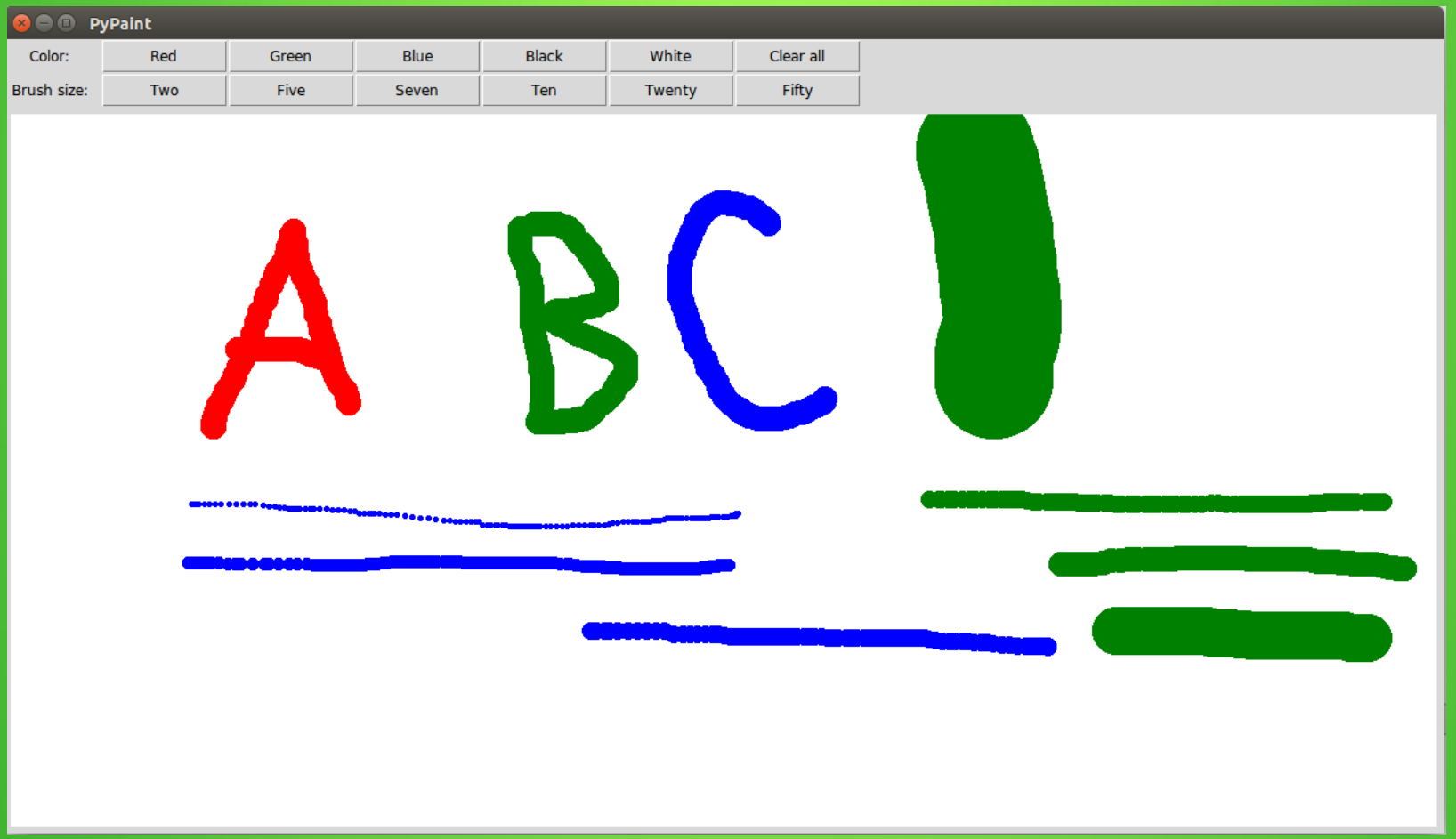
Python 3.4

Tkinter
Урок 12
Простой графический редактор

Постановка задачи

Цель работы — тренировка в создании GUI, использовании макетов компоновки в tkinter, передаче дополнительных аргументов в функцию-обработчик нажатия кнопки и использовании lambda-функции в Python.

Для этого примера удобнее будет использовать классовый подход к созданию GUI.



Определим класс Paint

```
1  from tkinter import *
2
3  class Paint(Frame):
4      def __init__(self, parent):
5          Frame.__init__(self, parent)
6          self.parent = parent
7
8
9  def main():
10     root = Tk()
11     root.geometry("1920x1080+300+300")
12     app = Paint(root)
13     root.mainloop()
14
15 if __name__ == "__main__":
16     main()
```

Запустив этот код мы получим простенькое окно, с которым будем работать дальше.

Определим метод setUI

Теперь напишем для класса Paint метод setUI, в котором будет задаваться расположение всех кнопок, меток и самого поля для рисования. У нас будет два ряда кнопок, первый ряд с кнопками устанавливающими цвет, второй ряд устанавливает размер кисти для рисования. Под ними будет идти поле для рисования.

Не забудем добавить вызов этого метода в __init__, чтобы все работало.

self.setUI()

```python
def setUI(self):

        self.parent.title("Pythonicway PyPaint")  # Устанавливаем название окна
        self.pack(fill=BOTH, expand=1)  # Размещаем активные элементы на родительском ок

        self.columnconfigure(6, weight=1) # Даем седьмому столбцу возможность растягивати
        self.rowconfigure(2, weight=1) # То же самое для третьего ряда

        self.canv = Canvas(self, bg="white")  # Создаем поле для рисования, устанавливае
        self.canv.grid(row=2, column=0, columnspan=7,
                       padx=5, pady=5, sticky=E+W+S+N)  # Прикрепляем канвас методом gri

        color_lab = Label(self, text="Color: ") # Создаем метку для кнопок изменения цве
        color_lab.grid(row=0, column=0, padx=6) # Устанавливаем созданную метку в первый

        red_btn = Button(self, text="Red", width=10) # Создание кнопки:  Установка текст
        red_btn.grid(row=0, column=1) # Устанавливаем кнопку первый ряд, вторая колонка

        # Создание остальных кнопок повторяет ту же логику, что и создание
        # кнопки установки красного цвета, отличаются лишь аргументы.

        green_btn = Button(self, text="Green", width=10)
        green_btn.grid(row=0, column=2)

        blue_btn = Button(self, text="Blue", width=10)
        blue_btn.grid(row=0, column=3)

        black_btn = Button(self, text="Black", width=10)
        black_btn.grid(row=0, column=4)

        white_btn = Button(self, text="White", width=10)
        white_btn.grid(row=0, column=5)


        size_lab = Label(self, text="Brush size: ") # Создаем метку для кнопок изменения
        size_lab.grid(row=1, column=0, padx=5)
        one_btn = Button(self, text="Two", width=10)
        one_btn.grid(row=1, column=1)

        two_btn = Button(self, text="Five", width=10)
        two_btn.grid(row=1, column=2)

        five_btn = Button(self, text="Seven", width=10)
        five_btn.grid(row=1, column=3)

        seven_btn = Button(self, text="Ten", width=10)
        seven_btn.grid(row=1, column=4)

        ten_btn = Button(self, text="Twenty", width=10)
        ten_btn.grid(row=1, column=5)

        twenty_btn = Button(self, text="Fifty", width=10)
        twenty_btn.grid(row=1, column=6, sticky=W)
```

Пояснения

`w.columnconfigure(N, option=value, ...)`

In the grid layout inside widget *w*, configure column *N* so that the given *option* has the given `value`. For options, see the table below.

`w.rowconfigure(N, option=value, ...)`

In the grid layout inside widget *w*, configure row *N* so that the given *option* has the given `value`. For options, see the table below.

Here are the options used for configuring column and row sizes.

Table 2. Column and row configuration options for the `.grid()` geometry manager

`minsize`	The column or row's minimum size in pixels. If there is nothing in the given column or row, it will not appear, even if you use this option.
`pad`	A number of pixels that will be added to the given column or row, over and above the largest cell in the column or row.

`weight`	To make a column or row stretchable, use this option and supply a value that gives the relative weight of this column or row when distributing the extra space. For example, if a widget `w` contains a grid layout, these lines will distribute three-fourths of the extra space to the first column and one-fourth to the second column:
	`w.columnconfigure(0, weight=3)`
	`w.columnconfigure(1, weight=1)`
	If this option is not used, the column or row will not stretch.

Пояснения

5.10. Geometry strings

A *geometry string* is a standard way of describing the size and location of a top-level window on a desktop.

A geometry string has this general form:

```
'wxh±x±y'
```

where:

- The *w* and *h* parts give the window width and height in pixels. They are separated by the character 'x'.
- If the next part has the form +*x*, it specifies that the left side of the window should be *x* pixels from the left side of the desktop. If it has the form -*x*, the right side of the window is *x* pixels from the right side of the desktop.
- If the next part has the form +*y*, it specifies that the top of the window should be *y* pixels below the top of the desktop. If it has the form -*y*, the bottom of the window will be *y* pixels above the bottom edge of the desktop.

Пояснения

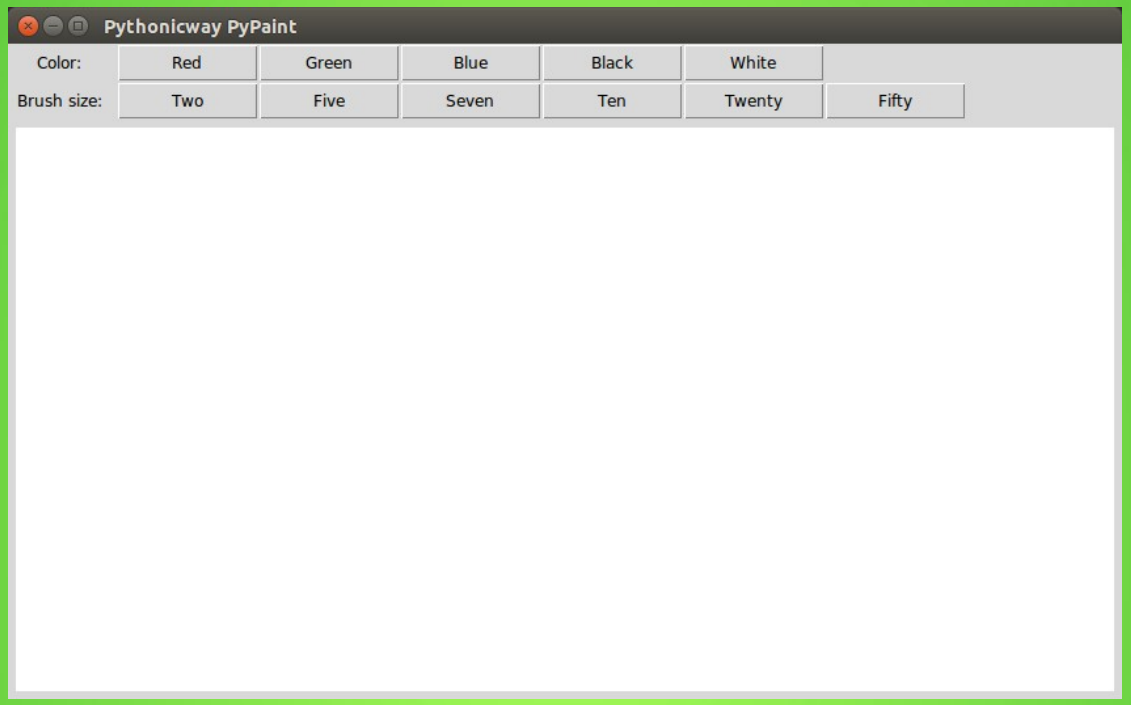
Упаковщик pack() является самым интеллектуальным (и самым непредсказуемым). При использовании этого упаковщика с помощью свойства side нужно указать к какой стороне родительского виджета он должен примыкать. Как правило этот упаковщик используют для размещения виджетов друг за другом (слева направо или сверху вниз). Пример:

```
from tkinter import *
root=Tk()
button1 = Button(text="1")
button2 = Button(text="2")
button3 = Button(text="3")
button4 = Button(text="4")
button5 = Button(text="5")
button1.pack(side='left')
button2.pack(side='top')
button3.pack(side='left')
button4.pack(side='bottom')
button5.pack(side='right')
root.mainloop()
```

При применении этого упаковщика можно указать следующие аргументы:

- side ("left"/"right"/"top"/"bottom") - к какой стороне должен примыкать размещаемый виджет.
- fill (None/"x"/"y"/"both") - необходимо ли расширять пространство предоставляемое виджету.
- expand (True/False) - необходимо ли расширять сам виджет, чтобы он занял всё предоставляемое ему пространство.

Результат



Полный интерфейс есть, но кнопки пока не работают, так как не определены обработчики событий.

```
def setUI(self):

        self.parent.title("Pythonicway PyPaint")  # Устанавливаем название окна
        self.pack(fill=BOTH, expand=1)  # Размещаем активные элементы на родительском окне

        self.columnconfigure(6, weight=1) # Даем седьмому столбцу возможность растягиваться, благодаря чему кнопки не будут разъезжаться при ресайзе
        self.rowconfigure(2, weight=1) # То же самое для третьего ряда

        self.canv = Canvas(self, bg="white")  # Создаем поле для рисования, устанавливаем белый фон
        self.canv.grid(row=2, column=0, columnspan=7,
                       padx=5, pady=5, sticky=E+W+S+N)  # Прикрепляем канвас методом grid. Он будет находится в 3м ряду, первой колонке, и будет занимать 7 колонок, задаем отступы по X и Y в 5 пикселей, и заставляем растягиваться при растягивании всего окна

        color_lab = Label(self, text="Color: ") # Создаем метку для кнопок изменения цвета кисти
        color_lab.grid(row=0, column=0, padx=6) # Устанавливаем созданную метку в первый ряд и первую колонку, задаем горизонтальный отступ в 6 пикселей

        red_btn = Button(self, text="Red", width=10) # Создание кнопки:  Установка текста кнопки, задание ширины кнопки (10 символов)
        red_btn.grid(row=0, column=1) # Устанавливаем кнопку первый ряд, вторая колонка

        # Создание остальных кнопок повторяет ту же логику, что и создание
        # кнопки установки красного цвета, отличаются лишь аргументы.

        green_btn = Button(self, text="Green", width=10)
        green_btn.grid(row=0, column=2)

        blue_btn = Button(self, text="Blue", width=10)
        blue_btn.grid(row=0, column=3)

        black_btn = Button(self, text="Black", width=10)
        black_btn.grid(row=0, column=4)

        white_btn = Button(self, text="White", width=10)
        white_btn.grid(row=0, column=5)
```

```
size_lab = Label(self, text="Brush size: ") # Создаем метку для кнопок изменения размера кисти
 size_lab.grid(row=1, column=0, padx=5)
 one_btn = Button(self, text="Two", width=10)
 one_btn.grid(row=1, column=1)

 two_btn = Button(self, text="Five", width=10)
 two_btn.grid(row=1, column=2)

 five_btn = Button(self, text="Seven", width=10)
 five_btn.grid(row=1, column=3)

 seven_btn = Button(self, text="Ten", width=10)
 seven_btn.grid(row=1, column=4)

 ten_btn = Button(self, text="Twenty", width=10)
 ten_btn.grid(row=1, column=5)

 twenty_btn = Button(self, text="Fifty", width=10)
 twenty_btn.grid(row=1, column=6, sticky=W)
```

Напишем обработчики

```
from tkinter import *


class Paint(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.setUI()
        self.brush_size = 10
        self.color = "black"

    def set_color(self, new_color):
        self.color = new_color

    def set_brush_size(self, new_size):
        self.brush_size = new_size

    def draw(self, event):
        self.canv.create_oval(event.x - self.brush_size,
                              event.y - self.brush_size,
                              event.x + self.brush_size,
                              event.y + self.brush_size,
                              fill=self.color, outline=self.color)


    def setUI(self):
            self.parent.title("PyPaint")  # Устанавливаем название окна
            self.pack(fill=BOTH, expand=1)  # Размещаем активные элементы на родительском окне

            self.columnconfigure(6,
                                 weight=1)  # Даем седьмому столбцу возможность растягиваться, благодаря чему кнопки не б
            self.rowconfigure(2, weight=1)  # То же самое для третьего ряда

            self.canv = Canvas(self, bg="white")  # Создаем поле для рисования, устанавливаем белый фон
            self.canv.grid(row=2, column=0, columnspan=7,
                           padx=5, pady=5,
                           sticky=E + W + S + N)  # Прикрепляем канвас методом grid. Он будет находится в 3м ряду, первой

            self.canv.bind("<B1-Motion>", self.draw)

            color_lab = Label(self, text="Color: ")  # Создаем метку для кнопок изменения цвета кисти
            color_lab.grid(row=0, column=0,
                           padx=6)  # Устанавливаем созданную метку в первый ряд и первую колонку, задаем горизонтальный

            red_btn = Button(self, text="Red",
                             width=10, command=lambda: self.set_color("red"))  # Создание кнопки:  Установка текста кнопк
            red_btn.grid(row=0, column=1)  # Устанавливаем кнопку первый ряд, вторая колонка
```

Привязка функции рисования

```
from tkinter import *


class Paint(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.setUI()
        self.brush_size = 10
        self.color = "black"

    def set_color(self, new_color):
        self.color = new_color

    def set_brush_size(self, new_size):
        self.brush_size = new_size

    def draw(self, event):
        self.canv.create_oval(event.x - self.brush_size,
                              event.y - self.brush_size,
                              event.x + self.brush_size,
                              event.y + self.brush_size,
                              fill=self.color, outline=self.color)


    def setUI(self):
            self.parent.title("PyPaint")  # Устанавливаем название окна
            self.pack(fill=BOTH, expand=1)  # Размещаем активные элементы на родительском окне

            self.columnconfigure(6,
                                 weight=1)  # Даем седьмому столбцу возможность растягиваться, благодаря чему кнопки не б
            self.rowconfigure(2, weight=1)  # То же самое для третьего ряда

            self.canv = Canvas(self, bg="white")  # Создаем поле для рисования, устанавливаем белый фон
            self.canv.grid(row=2, column=0, columnspan=7,
                           padx=5, pady=5,
                           sticky=E + W + S + N)  # Прикрепляем канвас методом grid. Он будет находится в 3м ряду, первой

            self.canv.bind("<B1-Motion>", self.draw)

            color_lab = Label(self, text="Color: ")  # Создаем метку для кнопок изменения цвета кисти
            color_lab.grid(row=0, column=0,
                           padx=6)  # Устанавливаем созданную метку в первый ряд и первую колонку, задаем горизонтальный

            red_btn = Button(self, text="Red",
                             width=10, command=lambda: self.set_color("red"))  # Создание кнопки:  Установка текста кнопк
            red_btn.grid(row=0, column=1)  # Устанавливаем кнопку первый ряд, вторая колонка

            # Создание остальных кнопок повторяет ту же логику, что и создание
```

self.canv.bind("<B1-Motion>", self.draw) — рисование при нажатой кнопке мыши

Изменение цвета

```
def set_color(self, new_color):
    self.color = new_color
```

```
red_btn = Button(self, text="Red",
                 width=10, command=lambda: self.set_color("red"))  # Создание кнопки:  Установка текста к
red_btn.grid(row=0, column=1)  # Устанавливаем кнопку первый ряд, вторая колонка
```

Button(self, text="Red", width=10, command=lambda: self.set_color("red"))

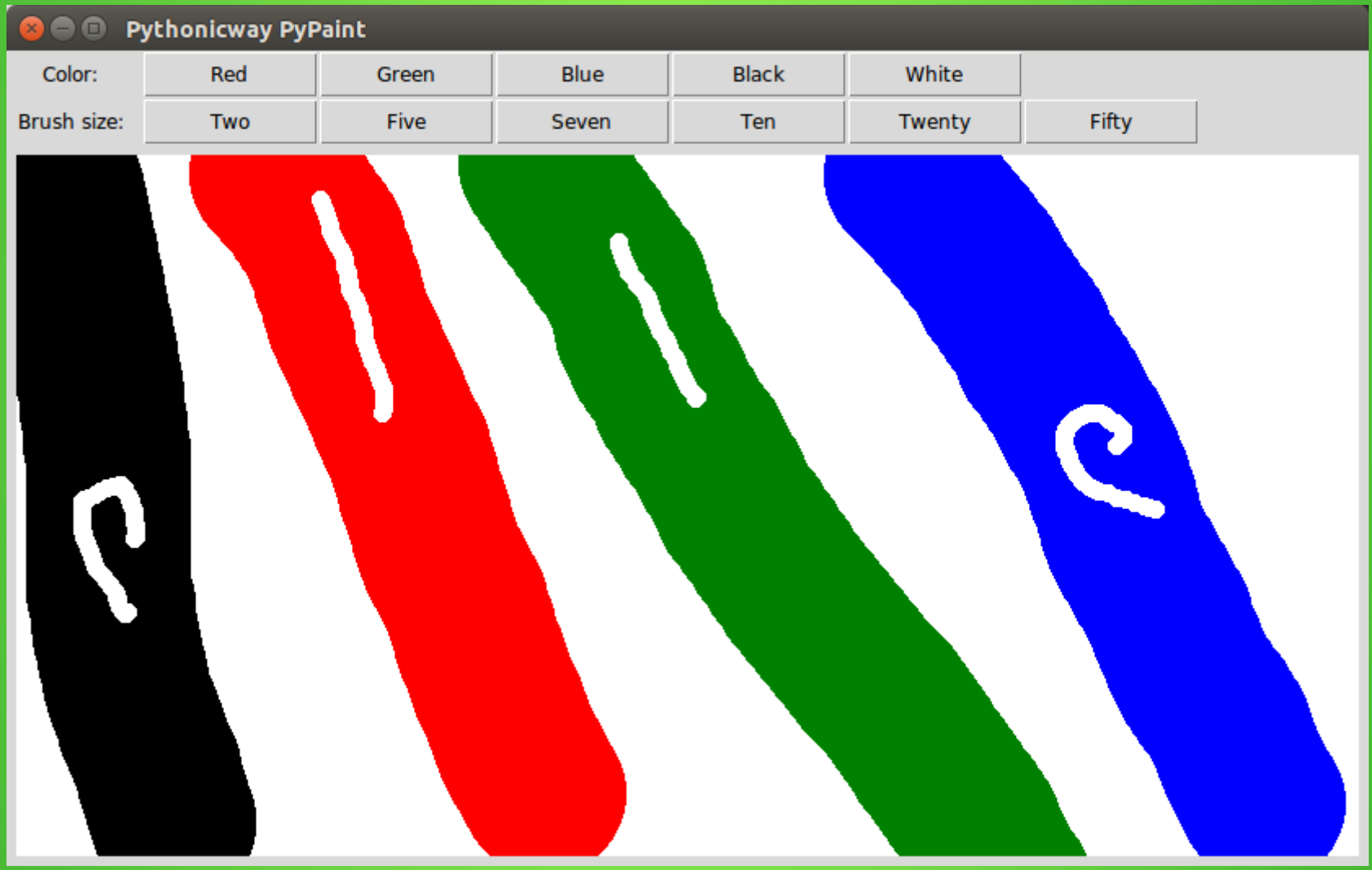
Используем lambda-функцию для передачи параметра в метод set_color()

Изменение толщины линии

```
class Paint(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent
        self.setUI()
        self.brush_size = 10
        self.color = "black"

    def set_color(self, new_color):
        self.color = new_color

    def set_brush_size(self, new_size):
        self.brush_size = new_size
```

```
size_lab = Label(self, text="Brush size: ")  # Создаем метку для кнопок изменения размера кисти
size_lab.grid(row=1, column=0, padx=5)
one_btn = Button(self, text="Two", width=10, command=lambda: self.set_brush_size(2))
one_btn.grid(row=1, column=1)

two_btn = Button(self, text="Five", width=10, command=lambda: self.set_brush_size(5))
two_btn.grid(row=1, column=2)

five_btn = Button(self, text="Seven", width=10, command=lambda: self.set_brush_size(7))
five_btn.grid(row=1, column=3)

seven_btn = Button(self, text="Ten", width=10, command=lambda: self.set_brush_size(10))
seven_btn.grid(row=1, column=4)

ten_btn = Button(self, text="Twenty", width=10, command=lambda: self.set_brush_size(20))
ten_btn.grid(row=1, column=5)

twenty_btn = Button(self, text="Fifty", width=10, command=lambda: self.set_brush_size(50))
twenty_btn.grid(row=1, column=6, sticky=W)
```

Результат

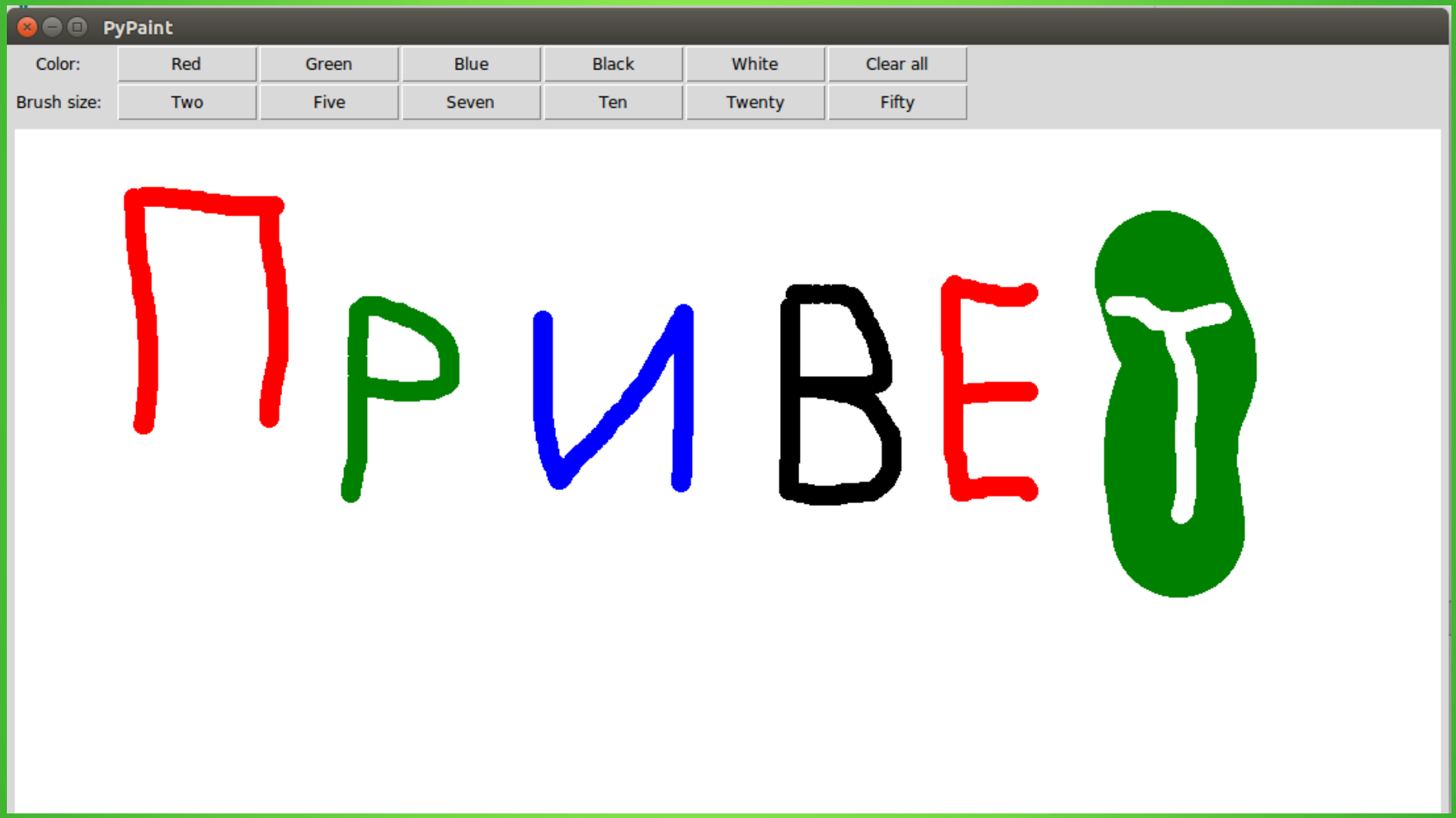

Pythonicway PyPaint
Color:
Red
Green
Blue
Black
White
Brush size:
Two
Five
Seven
Ten
Twenty
Fifty

Стереть рисунок

```
clear_btn = Button(self, text="Clear all", width=10, command=lambda: self.canv.delete("all"))
clear_btn.grid(row=0, column=6, sticky=W)
```

Метод delete() не надо программировать самостоятельно. Он — стандартный для объекта типа Холст (Canvas)

Финал


PyPaint
Color:
Red
Green
Blue
Black
White
Clear all
Brush size:
Two
Five
Seven
Ten
Twenty
Fifty
ПрИВЕТ