Импортируется графическую библиотеку tkinter, необходимые функции библиотеки math, и из библиотеки PIL импортируем ImageGrab -получение скриншота нашего экрана.

In [1]:
```
from tkinter import *
from math import pi, cos, sin, sqrt, atan2
from PIL import ImageGrab
```

Создается класс Paint, который создает окно, на котором будут происходить все дальнейшие действия.

In [ ]:
```
class Paint(Frame):
```

Инициализируются аттрибуты. Во первых, создается Frame, на котором будут располагаться виджеты. Далее задаются **self.n: int** - колличество симметрий, **self.red, self.green, self.blue: int** - значения в RGB цветов, каждый от 0 до 255, **self.parent** - родитель , **self.color** - цвет в поддерживаемом формате, изначально черный, цвет рисуемых фигур, **self.color_canvas** - цвет в нужном формате, изначально белый, цвет canvas, **self.file_name: str** - имя файла, изначально Untitled, **self.type: str** - тип рисуемой фигуры, изначально круги, **self.brush_size: int** - размер кисти, изначально 1, **self.canvas_wigth_center, self.canvas_height_center: int** - координаты центра экрана, изначально (0, 0), **self.set_ui()** - вызов программы, которая располагает виджиты на окне.

In [ ]:
```
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.n = 1  # Колличество симметрий
        self.red, self.green, self.blue = 0, 0, 0  # Цвета в формате RG
B
        self.parent = parent  # Родитель
        self.color = "black"  # Цвет рисуемых фигур
        self.color_canvas = "white"  # Цвет canvas
        self.file_name = 'Untitled'  # Имя файла
```

```
        self.type = 'circle'  # Тип рисуемой фигуры
        self.brush_size = 1  # Размер кисти
        self.canvas_width_center = 0  # Центр по ширине canvas
        self.canvas_height_center = 0  # Центр по высоте canvas
        self.set_ui()  # Вызов функции задания виджитов
```

Функция **set_color_canvas(self)**, получающая значения шкал методом **get()**, задающих цвета в RGB, переводит их в необходимый формат и меняет цвет canvas за счет смены значения **self.color_canvas** и вызыва функции **config**, которая меняет значения цвета для **self.canvas**.

In [ ]:
```
    def set_color_canvas(self):
        self.color_canvas = "#%02x%02x%02x" % (int(self.scale_red.get
()),
                                               int(self.scale_green.get
()),
                                               int(self.scale_blue.get
()))
        # Получаем цвет в нужном нам формате.
        self.canvas.config(bg=self.color_canvas)
```

Далее, идет функция **draw(self, event)**, которая на вход запрашивает **event**, в нашем случае являющийся движением мышки с зажатой левой кнопкой. Функция считывает значения шкалы симметрий и помещает его в **self.n**, и цвета со шкал цветов в формате RGB, переводя их в нужный формат и помещая в значение **self.color** - цвет рисуемых фигур, а также со шкалы размера кисти считывем значение **self.brush_size** - размер кисти, по факту, размер рисуемых фигур. Далее по **self.type** - заданому типу рисуемых фигур (определяется последней нажатой кнопкой типа фигуры) определяется, рисуется круг, треугольник или квадрат.

In [ ]:
```
    def draw(self, event):
        self.n = int(self.scale_symmetry.get())
        self.color = "#%02x%02x%02x" % (int(self.scale_red.get()),
                                        int(self.scale_green.get()),
                                        int(self.scale_blue.get()))
```

```
        self.brush_size = int(self.scale_brush.get())/2
        # Задаем количество симметрий n, цвет для рисования и размер ки
сти.
        if self.type == 'circle':
            self.draw_circle(event)
        if self.type == 'triangle':
            self.draw_triangle(event)
        if self.type == 'rectangle':
            self.draw_rectangle(event)
        # Разделение по типам рисуемой фигуры.
```

Далее задаем функции **draw_triangle**, **draw_circle**, **draw_rectangle**, вызываемые функцией **draw**, и рисующие соответсвенно треугольники, круги и квадраты. Отрисовка происходит по следующей схеме - получаем текущие координаты центра canvas встроенными методами **winfo_width(), winfo_height**, возвращающими ширину и длину canvas, делим полученные величины на два и помещаем их в **self.canvas_width_center, self.canvas_height_center**. Далее рассчитываем координаты **x, y** в системе координат относительно центра canvas, и получаем величину **radius** - радиус--вектор точки **event** в СК относительно центра canvas. Определяем угол **phi** для полярной СК - при значении **y =** 0 угол равен $\frac{\pi}{2}$, иначе через функцию **ata2** библиотеки **math**, возвращающей не значение от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$, как это делает **atan**, а нас интересующее значение от $-\pi$ до π. Далее в цикле по количеству симметрий рассчитываем угол **alpha** симметричной точки в полярной СК, и используя этот угол возвращаемся в СК canvas.

In [ ]:
```
    def draw_triangle(self, event):  # Функция рисования треугольников
        self.canvas_width_center = self.canvas.winfo_width() / 2
        self.canvas_height_center = self.canvas.winfo_height() / 2
        x = event.x - self.canvas_width_center
        y = -event.y + self.canvas_height_center
        # Рассчет для перехода в систему координат в центре self.canva
s.
        radius = sqrt(x ** 2 + y ** 2)
        # Радиус-вектор для перехода в полярную СК.
        if y == 0:
            phi = pi / 2
        else:
```

```
            phi = atan2(x, y)
        # Рассчет угла для перехода в полярную СК.
        for i in range(self.n):  # Рисование симметрично.
            self.parent.update_idletasks()
            # Обновление canvas.
            alpha = phi + i / self.n * 2 * pi
            # Угол в полярной СК в зависимости от количества симметий.
            x_polar = radius * sin(alpha) + self.canvas_width_center
            y_polar = -radius * cos(alpha) + self.canvas_height_center
            # Переход из полярной СК в СК экрана.
            self.canvas.create_polygon(x_polar, y_polar + self.brush_si
ze, x_polar + self.brush_size,
                                       y_polar - self.brush_size, x_pol
ar - self.brush_size,
                                       y_polar - self.brush_size, fill=
self.color, outline=self.color)
            # Рисование треугольника.
```

In [ ]:

```
    def draw_rectangle(self, event):  # Функция для рисования квадрато
в.
        self.canvas_width_center = self.canvas.winfo_width() / 2
        self.canvas_height_center = self.canvas.winfo_height() / 2
        x = event.x - self.canvas_width_center
        y = -event.y + self.canvas_height_center
        # Рассчет для перехода в систему координат в центре self.canva
s.
        radius = sqrt(x ** 2 + y ** 2)
        # Радиус-вектор для перехода в полярную СК.
        if y == 0:
            phi = pi / 2
        else:
            phi = atan2(x, y)
        # Рассчет угла для перехода в полярную СК.
        for i in range(self.n):  # Рисование симметрично.
            self.parent.update_idletasks()
            # Обновление canvas.
            alpha = phi + i / self.n * 2 * pi
            # Угол в полярной СК в зависимости от количества симметий.
            x_polar = radius * sin(alpha) + self.canvas_width_center
```

```
            y_polar = -radius * cos(alpha) + self.canvas_height_center
            # Переход из полярной СК в СК экрана.
            self.canvas.create_rectangle(x_polar - self.brush_size, y_p
olar - self.brush_size,
                                         x_polar + self.brush_size, y_p
olar + self.brush_size,
                                         fill=self.color, outline=self.
color)
            # Рисование квадрата.
```

In [ ]:

```
    def draw_circle(self, event):  # Функция для рисования кругов.
        self.canvas_width_center = self.canvas.winfo_width() / 2
        self.canvas_height_center = self.canvas.winfo_height() / 2
        x = event.x - self.canvas_width_center
        y = -event.y + self.canvas_height_center
        # Рассчет для перехода в систему координат в центре self.canva
s.
        radius = sqrt(x ** 2 + y ** 2)
        # Радиус-вектор для перехода в полярную СК.
        if y == 0:
            phi = pi / 2
        else:
            phi = atan2(x, y)
        # Рассчет угла для перехода в полярную СК.
        for i in range(self.n):  # Рисование симметрично.
            self.parent.update_idletasks()
            # Обновление canvas.
            alpha = phi + i / self.n * 2 * pi
            # Угол в полярной СК в зависимости от количества симметий.
            x_polar = radius * sin(alpha) + self.canvas_width_center
            y_polar = -radius * cos(alpha) + self.canvas_height_center
            # Переход из полярной СК в СК экрана.
            self.canvas.create_oval(x_polar - self.brush_size, y_polar
- self.brush_size,
                                    x_polar + self.brush_size, y_polar
+ self.brush_size,
                                    fill=self.color, outline=self.color
)
            # Рисование квадрата.
```

Функция сохранения файла считывает имя из окна назнвания файла и заносит его имя в значение **self.file_name**. Далее с помощью функции **ImageGrab.grab** делается скриншот canvas и сохраняется изображением **png** в папке, где лежит сама программа.

In [ ]:
```
    def file_save(self):  # Функция сохранения файла png.
        self.file_name = self.e.get()
        # Получение имени файла.
        ImageGrab.grab((self.canvas.winfo_rootx(), self.canvas.winfo_ro
oty(), self.canvas.winfo_width(),
                        self.canvas.winfo_height())).save(self.file_nam
e + '.png')
        # Запечатление и сохранение рисунка в папке приложения.
```

В функции, создающей и размещающей виджеты на окне, в самом начале задается название окна методом **sel.parent.title**. Затем методом **self.pack** размещаются все виджиты на родительском окне, методами **self.columnconfigure, self.rowconfigure** создается необходимая разметка окна из 9 столбцов и 3 рядов, в цикле по всем столбцам задаем методом **self.grid_columnconfigure** задается минимальная ширина для всех столбцовю. Далее создается сам **self.canvas**, задавая его цвет, шкалы **self.scale_red, self.scale_green, self.scale_blue** горизонтальной ориентировки, принимающих значения от 0 до 255 для кодировки RGB, шкалу **self.scale_symmetry**, ориентированную горизонтально и принимающую значения от 1 до 10, задающую количество симметрий, шкалу **self.scale_brush**, ориентированную горизонтально, принимающую значения от 1 до 20 и задающую размер рисуемых фигур, а так же поле **self.e**, куда можно будет вписывать имя файла с рисунком. Далее методом **grid** располагается **self.canvas** в 3 ряду и растягивается на 9 колонок с возможностью растягивания при увеличении размера окна. Далее тем же методом располагаются вышеуказанные элементы, рядом с ними располагая их названия заданные как **label**. Так же создаются кнопки **clean_btn, file_btn, circle_btn, triangle_btn, rectangle_btn** - кнопки, позволяющие полностью очищать экран, сохранять рисунок, и выбирать типы рисуемых фигур соответственно. Для вызова функций и замены значений используется лямбда-функция.

In [ ]:
```
    def set_ui(self):
```

```
        self.parent.title("Симметричное рисование")  # Имя окна
        self.pack(fill=BOTH, expand=1)  # Размещение элементов на родит
ельском окне
        self.columnconfigure(8, weight=1)
        self.rowconfigure(2, weight=1)  # Количество столбцов и строк
        for i in range(9):
            self.grid_columnconfigure(i, minsize=145)  # Минимальная ши
рина столбца

        self.canvas = Canvas(self, bg=self.color_canvas)
        self.scale_red = Scale(self, from_=0, to=255, orient=HORIZONTAL
)
        self.scale_brush = Scale(self, from_=1, to=20, orient=HORIZONTA
L)
        self.e = Entry(self)
        self.scale_symmetry = Scale(self, from_=1, to=10, orient=HORIZO
NTAL)
        self.scale_green = Scale(self, from_=0, to=255, orient=HORIZONT
AL)
        self.scale_blue = Scale(self, from_=0, to=255, orient=HORIZONTA
L)
        # Задаем canvas и шкалы, значения которых необходимо получать в
 вызываемых функциях.
        self.canvas.grid(row=2, column=0, columnspan=9, sticky=E + W +
S + N)
        # Расположение canvas, его размер в столбцах и возможность раст
ягивания.
        self.canvas.bind("<B1-Motion>", self.draw)  # Отслеживание за с
чет движения + нажатия левой кнопки мыши

        canvas_color_setting_btn = Button(self, text="Set canvas color"
,
                                          width=16, command=lambda: sel
f.set_color_canvas())
        canvas_color_setting_btn.grid(row=1, column=7)
        # Кнопка для задания цвета canvas.
        self.scale_red.grid(row=0, column=1)
        label_red = Label(self, text='Red')
        label_red.grid(row=0, column=0)
```

```
        # Расположение шкалы красного и ее названия.
        self.scale_green.grid(row=0, column=3)
        label_green = Label(self, text='Green')
        label_green.grid(row=0, column=2)
        # Расположение шкалы зеленого и ее названия.
        self.scale_blue.grid(row=0, column=5)
        label_blue = Label(self, text='Blue')
        label_blue.grid(row=0, column=4)
        # Расположение шкалы голубого и ее названия.
        symmetry_lab = Label(self, text="Symmetry number")
        symmetry_lab.grid(row=1, column=0, padx=6)
        self.scale_symmetry.grid(row=1, column=1)
        # Расположение шкалы симметрий и ее названия.
        clean_btn = Button(self, text="Clean", width=16, command=lambda
: self.canvas.delete('all'))
        clean_btn.grid(row=1, column=8)
        # Кнопка для полной очистки экрана.
        Label(self, text="File name").grid(row=1, column=2)
        self.e.grid(row=1, column=3)
        self.e.insert(10, "Untitled")
        # Расположение названия и окна для ввода имени файла.
        file_btn = Button(self, text="Save", width=16, command=lambda:
self.file_save())
        file_btn.grid(row=1, column=4)
        # Расположение кнопки для сохранения файла.
        circle_btn = Button(self, text="Draw circles", width=16, comman
d=lambda: self.type == 'circle')
        circle_btn.grid(row=0, column=6)
        # Расположение кнопки, задающей тип фигуры круг.
        triangle_btn = Button(self, text="Draw triangles", width=16, co
mmand=lambda: self.type == 'triangle')
        triangle_btn.grid(row=0, column=7)
        # Расположение кнопки, задающей тип фигуры треугольник.
        rectangle_btn = Button(self, text="Draw rectangles", width=16,
command=lambda: self.type == 'rectangle')
        rectangle_btn.grid(row=0, column=8)
        # Расположение кнопки, задающей тип фигуры квадрат.
        self.scale_brush.grid(row=1, column=5)
        label_brush = Label(self, text='Brush size')
```

```
        label_brush.grid(row=1, column=6)
        # Расположение шкалы размера кисти и ее названия.
```

set_type(n:str) - функция, принимающая на вход тип фигуры и задающая его для рисования

In [ ]:
```
    def set_type(self, n):
        if n == 'circle':
            self.type = 'triangle'
        if n == 'rectangle':
            self.type = 'rectangle'
        if n == 'triangle':
            self.type = 'triangle'
```

В теле программы создается окно - **root**, задаются его размеры через информацию о размере экрана методами **winfo_screenwidth(), winfo_screenheight()**, вызывается на окне класс **Paint**, описанный выше, окно в цикл до закрытия.

In [ ]:
```
if __name__ == '__main__':
    root = Tk()
    root.geometry(str(int(root.winfo_screenwidth()))+'x'+str(int(root.w
info_screenheight())))
    app = Paint(root)
    root.mainloop()
```