Лабораторная работа №6. Создание графического интерфейса пользователя с помощью языка программирования Python

На лекции были рассмотрены методы объекта canvas, формирующие на нем геометрические примитивы и текст. Однако это лишь часть методов холста. В другую условную группу можно выделить методы, изменяющие свойства уже существующих объектов холста (например, геометрических фигур). И тут возникает вопрос: как обращаться к уже созданным фигурам? Ведь если при создании было прописано что-то вроде canvas.create_oval(30,10,130,80) и таких овалов, квадратов и др. на холсте очень много, то как к ним обращаться?

Для решения этой проблемы в tkinter для объектов холста можно использовать идентификаторы и теги, которые затем передаются другим методам. У любого объекта может быть как идентификатор, так и тег. Использование идентификаторов и тегов немного различается.

Рассмотрим несколько методов изменения уже существующих объектов с использованием при этом идентификаторов. Для начала создадим холст и три объекта на нем. При создании объекты "возвращают" свои идентификаторы, которые можно связать с переменными (oval, rect и trial в примере ниже) и потом использовать их для обращения к конкретному объекту.

```
c = Canvas(width=460,height=460,bg='grey80')
c.pack()
oval = c.create_oval(30,10,130,80)
rect = c.create_rectangle(180,10,280,80)
trian = c.create_polygon(330,80,380,10,430,80, fill='grey80',
outline="black")
```

Если вы выполните данный скрипт, то увидите на холсте три фигуры: овал, прямоугольник и треугольник.

Далее можно использовать методы-"модификаторы" указывая в качестве первого аргумента идентификатор объекта. Метод move перемещает объект на по оси X и Y на расстояние указанное в качестве второго и третьего аргументов. Следует понимать, что это не координаты, а смещение, т. е. в примере ниже прямоугольник опустится вниз на 150 пикселей. Метод itemconfig изменяет указанные свойства объектов, coords изменяет координаты (им можно менять и размер объекта).

```
c.move(rect,0,150)
```

```
c.itemconfig(trian,outline="red",width=3)
c.coords(oval,300,200,450,450)
```

Если запустить скрипт, содержащий две приведенные части кода (друг за другом), то мы сразу увидим уже изменившуюся картину на холсте: прямоугольник опустится, треугольник приобретет красный контур, а эллипс сместится и сильно увеличится в размерах. Обычно в программах изменения должны наступать при каком-нибудь внешнем воздействии. Пусть по щелчку левой кнопкой мыши прямоугольник передвигается на два пикселя вниз (он будет это делать при каждом щелчке мышью):

```
def mooove(event):
     c.move(rect,0,2)
...
c.bind('<Button-1>',mooove)
```

Теперь рассмотрим как работают теги. В отличие от идентификаторов, которые являются уникальными для каждого объекта, один и тот же тег может присваиваться разным объектам. Дальнейшее обращение к такому тегу позволит изменить все объекты, в которых он был указан. В примере ниже эллипс и линия содержат один и тот же тег, а функция color изменяет цвет всех объектов с тегом group1. Обратите внимание, что в отличие от имени идентификатора (переменная), имя тега заключается в кавычки (строковое значение).

```
oval = c.create_oval(30,10,130,80,tag="group1")
c.create_line(10,100,450,100,tag="group1")
…
def color(event):
     c.itemconfig('group1',fill="red",width=3)
…
c.bind('<Button-3>',color)
```

Еще один метод, который стоит рассмотреть, это delete, который удаляет объект по указанному идентификатору или тегу. В tkinter существуют зарезервированные теги: например, all обозначает все объекты холста. Так в примере ниже функция clean просто очищает холст.

```
def clean(event):
     c.delete('all')
...
c.bind('<Button-2>',clean)
```

Метод tag_bind позволяет привязать событие (например, щелчок кнопкой мыши) к определенному объекту. Таким образом, можно реализовать обращение к различным областям холста с помощью одного

и того же события. Пример ниже это наглядно иллюстрирует: изменения на холсте зависят от того, где произведен щелчок мышью.

```
from tkinter import *

c = Canvas(width=460,height=100,bg='grey80')
c.pack()

oval = c.create_oval(30,10,130,80,fill="orange")
c.create_rectangle(180,10,280,80,tag="rect",fill="lightgreen")
trian                                                          =
c.create_polygon(330,80,380,10,430,80,fill='white',outline="black")

def oval_func(event):
     c.delete(oval)
     c.create_text(30,10,text="Здесь был круг",anchor="w")
def rect_func(event):
     c.delete("rect")
     c.create_text(180,10,text="Здесь был\nпрямоугольник",anchor="nw")
def triangle(event):
      c.create_polygon(350,70,380,20,410,70,fill='yellow',outline="blac
k")

c.tag_bind(oval,'<Button-1>',oval_func)
c.tag_bind("rect",'<Button-1>',rect_func)
c.tag_bind(trian,'<Button-1>',triangle)

mainloop()
```

Модифицированный пример, в котором реализовано мигание овала с помощью потоков:

```
from tkinter import *
import time
import _thread

c = Canvas(width=460,height=100,bg='grey80')
c.pack()

oval = c.create_oval(30,10,130,80,fill="orange")
c.create_rectangle(180,10,280,80,tag="rect",fill="lightgreen")
trian                                                          =
c.create_polygon(330,80,380,10,430,80,fill='white',outline="black")

def oval_func(event):
     c.delete(oval)
     _thread.start_new_thread(f1, ())
def rect_func(event):
     c.delete("rect")
     c.create_text(180,10,text="Здесь был\nпрямоугольник",anchor="nw")
def triangle(event):
     c.create_polygon(350,70,380,20,410,70,
         fill='yellow',outline="black")
def f1():
    cnt = 3
```

```
    while cnt > 0:
        oval = c.create_oval(30,10,130,80,fill="red")
        time.sleep(1)
        c.delete(oval)
        time.sleep(1)
        cnt -= 1
    _thread.exit()

c.tag_bind(oval,'<Button-1>',oval_func)
c.tag_bind("rect",'<Button-1>',rect_func)
c.tag_bind(trian,'<Button-1>',triangle)

mainloop()
```

Справка по модулю Time.

Time - модуль для работы со временем в Python.

time.clock() - в Unix, возвращает текущее время. В Windows, возвращает время, прошедшее с момента первого вызова данной функции.

time.sleep(сек) - приостановить выполнение программы на заданное количество секунд.

time.time() - время, выраженное в секундах с начала эпохи.

Задание

1. Спишите скрипты, рассмотренные в данном уроке. Выполните их.
2. Реализуйте движение (анимацию) геометрической фигуры по холсту.
3. Реализуйте работающий светофор.