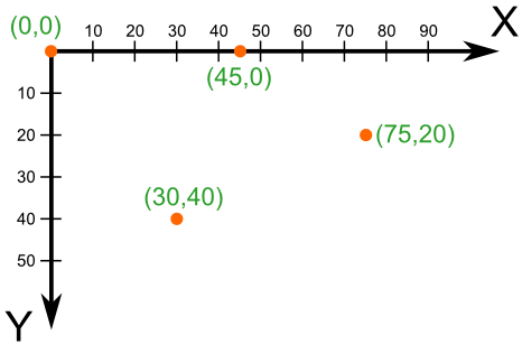
3. СОЗДАНИЕ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ И АНИМАЦИИ

Для создания графических изображений в модуле **tkinter** используется класс **Canvas**. Объектом этого класса является так называемый холст, на который и наносятся изображения. Для прорисовки различных изображений используются соответствующие методы класса **Canvas**.

При создании холста необходимо указать его размеры в виде ширины и высоты. Для размещения выводимого объекта в графическом окне необходимо задать координаты этого объекта в системе координат окна. Как и во всех других графических приложениях, в модуле **tkinter** используется система координат, в которой ось абсцисс направлена слева направо, а ось ординат — сверху вниз. При этом в качестве начала координат выступает левый верхний угол графического окна.



3.1. СОЗДАНИЕ ГРАФИЧЕСКИХ ПРИМИТИВОВ

На первом этапе создания любого графического изображения необходимо создать объект класса **Canvas**. Для создания объекта холста используется конструктор **Canvas()**. В качестве аргументов конструктору необходимо передать идентификатор родительского окна (как правило, это главное окно приложения) и размеры холста (ширину и высоту). Как и большинство конструкторов, конструктор класса **Canvas** имеет и другие аргументы, для которых установлены значения по умолчанию (например, цвет фона).

Простейшим графическим примитивом является линия. Для прорисовки линии в классе **Canvas** предназначен метод **create_line()**.

Обязательными аргументами этого метода являются координаты начала (x_1, y_1), затем – конца (x_2, y_2) линии, задаваемые в пикселях. Остальные аргументы не являются обязательными, поскольку содержат значения по умолчанию. В качестве таких аргументов, в частности, могут выступать:

- **fill** — определяет цвет линии;
- **activefill** — определяет цвет линии при наведении на неё курсора;
- **width** — определяет толщину линии;
- **arrow** — задает наличие стрелки у линии; если стрелка должна находиться в начале линии, то **arrow** = **FIRST**, если в конце — то **arrow** = **LAST**;
- **arrowshape** — определяет форму стрелки, задаваемую строкой из трех чисел;
- **dash** — задает пунктирную линию в виде кортежа из двух чисел: первое число определяет количество заполняемых пикселей, второе число — количество пропускаемых пикселей.

Пример задачи: написать программу, которая будет рисовать в графическом окне линии разного типа.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Линии')
#Создаем холст
c = Canvas(window, width=200, height = 200,bg='light blue')
c.pack()
#Создаем сплошную линию
c.create_line(20, 15, 150, 45, width = 5, fill ='white')
#Создаем линию со стрелкой
```

```
    c.create_line(100,   180,   100,   60,   fill='red',width=3,   arrow=LAST,
dash=(10,2),arrowshape="10 20 10")
    window.mainloop()
```

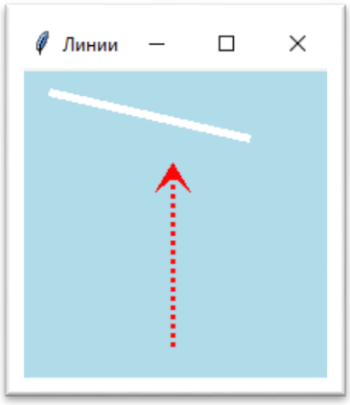


Рис. 1

Для рисования прямоугольников в модуле **tkinter** используется метод **create_rectangle()**. Обязательными аргументами метода являются координаты левого верхнего и правого нижнего углов. Остальные аргументы не являются обязательными. Кроме необязательных аргументов, совпадающих с необязательными аргументами метода **create_line()**, метод **create_rectangle()** содержит необязательный аргумент **outline**, который задает цвет рамки прямоугольника. Необходимо отметить, что в случае прямоугольника некоторые аргументы имеют смысловое значение, отличное от смыслового значения этих аргументов в методе **create_line()**. В частности, аргумент **width** определяет толщину линии рамки, аргумент **fill** — цвет заполнения прямоугольника (цвет заливки).

Пример задачи: написать программу, которая будет рисовать в графическом окне различные прямоугольники. Для одного из прямоугольников реализовать эффект изменения его рамки из сплошной линии на пунктирную при наведении мыши на этот прямоугольник.

Решение:

```
    from tkinter import *
    #Создаем окно
    window = Tk()
    window.title('Прямоугольники')
    #Создаем холст
    c = Canvas(window, width=400, height=300,bg='light blue')
    c.pack()
    #Создаем прямоугольники
    c.create_rectangle(20, 20, 280, 100, outline ='red',width=4)
    c.create_rectangle(100,        150,        300,        200,fill='yellow',
outline='green',width=3, activedash=(5, 4))
    window.mainloop()
```

Результат работы программы представлен на рисунке 2.

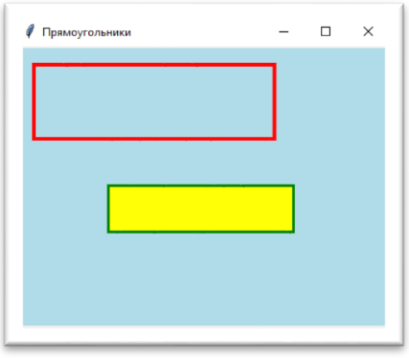


Рис. 2

Для рисования ломаных линий и многоугольников произвольной формы в модуле **tkinter** используется метод **create_polygon()**. В качестве аргументов данного метода необходимо задать координаты всех точек линии или многоугольника.

Пример задачи: написать программу, которая будет рисовать в графическом окне различные многоугольники произвольной формы, например, ромб, трапецию, параллелепипед.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Многоугольники произвольной формы')
#Создаем холст
c = Canvas(window, width=450, height=250)
c.pack()
#Создаем фигуры произвольной формы
#Рисуем ромб
c.create_polygon((180, 10), (100, 90), (260, 90),fill = 'yellow', outline
= 'red')
#Рисуем трапецию
c.create_polygon((40,     110),     (160,     110),     (190,180),     (10,
180),fill='orange', outline='green', width=3)
#Рисуем параллелепипед
c.create_polygon((300, 230), (300, 180), (400,180), (400, 230), fill='red',
outline = 'blue')
c.create_polygon((300, 180), (330, 150), (430,150), (400, 180), fill='red',
outline = 'blue')
c.create_polygon((400, 230), (400, 180), (430,150), (430, 200), fill='red',
outline = 'blue')
c.create_line((300, 230), (330, 200), fill ='blue', dash = (2,2))
c.create_line((330, 150), (330, 200), fill ='blue', dash = (2,2))
c.create_line((330, 200), (430, 200), fill ='blue', dash = (2,2))
window.mainloop()
```

Результат работы программы представлен на рисунке 3.

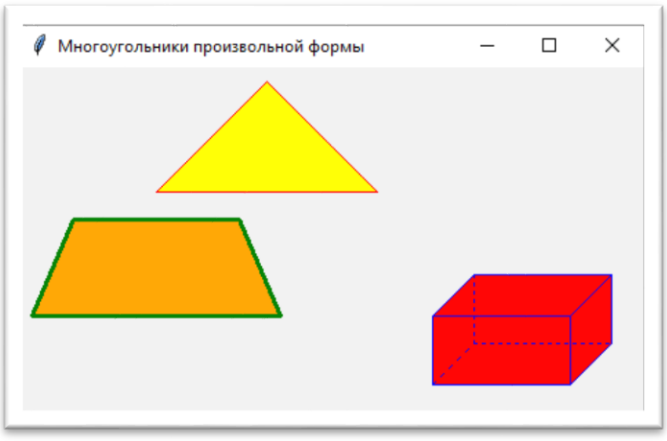


Рис. 3

Для рисования эллипсов и кругов в модуле **tkinter** используется метод **create_oval()**. В качестве аргументов метода необходимо задать координаты гипотетического прямоугольника, в который вписывается эллипс. Круг в данном случае представляет собой частный случай эллипса. Для его создания в качестве описывающего прямоугольника необходимо задать квадрат.

Пример задачи: написать программу, которая будет рисовать в графическом окне круг и эллипс.

Решение:

```
from tkinter import *
#Создаем окно
```

```
window = Tk()
window.title('Эллипс и круг')
#Создаем холст
c = Canvas(window, width=250, height=200,bg='light blue')
c.pack()
#Рисуем круг
c.create_oval((50, 10), (150, 110), fill='red',outline ='white')
#Рисуем эллипс
c.create_oval(10, 120, 190, 190, fill='yellow',outline='white')
window.mainloop()
```

Результат работы программы представлен на рисунке 4.

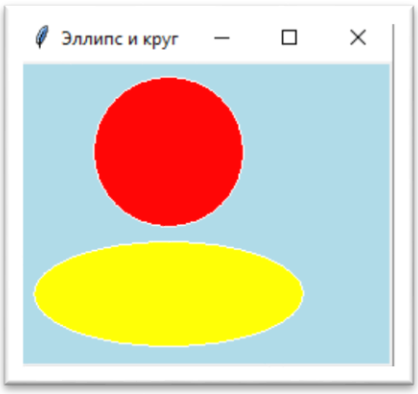


Рис. 4

Для создания более сложных фигур в модуле **tkinter** используется метод **create_arc()**. В зависимости от значения его аргумента **style** можно построить сектор (строится по умолчанию), сегмент (**style = CHORD**) или дугу (**style = ARC**). По аналогии с методом **create_oval()** в качестве аргументов метода необходимо задать координаты прямоугольника, в который вписана окружность (или эллипс), из которой «вырезается» сектор, сегмент или дуга. Значение аргумента **start** определяет начальное положение фигуры, значение аргумента **extent** задает угол поворота (можно задавать как положительные значения (отсчет против часовой стрелки), так и отрицательные значения (отсчет по часовой стрелке)).

Пример задачи: написать программу, которая будет рисовать в графическом окне дугу размером 50º из точки 140º, сегмент размером 90º из точки 240º и два сектора размерами 60º из точки 0º и 30º из точки 160º. В качестве фона для большей наглядности нарисовать круг серого цвета.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Сектора, дуги и сегменты')
#Создаем холст
c = Canvas(window, width=350, height=200,bg='light blue')
c.pack()
#Рисуем круг
c.create_oval((10, 10), (190, 190), fill = 'lightgrey',outline = 'white')
#Рисуем сектор размером 60 градусов из точки 0 градусов
c.create_arc((10, 10), (190, 190), start = 0,extent = 60, fill='blue')
#Рисуем сектор размером 30 градусов из точки 160 градусов
c.create_arc((10, 10), (190, 190), start = 160,extent = 30, fill='red')
#Рисуем дугу размером 50 градусов из точки 140 градусов
c.create_arc((10, 10), (190, 190), start = 140,extent = -50, style =
ARC,outline='darkgreen', width = 5)
#Рисуем сегмент размером 90 градусов из точки 240 градусов
c.create_arc((10, 10), (190, 190), start = 240,extent = 90, style = CHORD,
fill ='orange')
window.mainloop()
```

Результат работы программы представлен на рисунке 5.

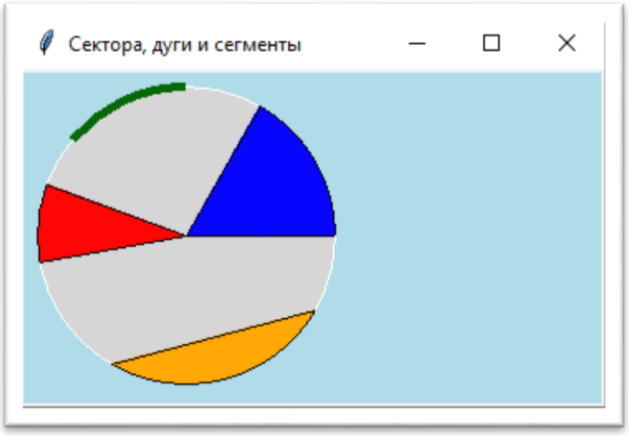


Рис. 5

Для размещения текста в графическом окне в модуле **tkinter** используется метод **create_text()**. Обязательными аргументами метода являются координаты местоположения выводимого текста и сам текст (аргумент **text**). В качестве необязательных аргументов метода могут выступать:

- **justify** — определяет способ выравнивания текста относительно самого себя (**CENTER** — по центру, **LEFT** — по левой границе,
- **RIGHT** — по правой границе); по умолчанию в задаваемой точке графического окна располагается центр текстовой надписи;
- **fill** — задает цвет текста;
- **font** — задает тип шрифта (в виде строки);
- **anchor** (якорь) — используется для привязки текста к конкретному месту графического окна. Для значения этого аргумента используются обозначения сторон света. Например, для размещения по указанной координате левой границы текста необходимо задать для якоря значение **W** (от *англ.* **west** — запад). Возможные другие значения аргумента **anchor**: **N** (север — вверху), **E** (восток — справа), **S** (юг — внизу), **W** (запад — слева), **NE** (северо-восток — вверху справа), **SE** (юго-восток — внизу справа), **SW** (юго-запад — внизу слева), **NW** (северо-восток — вверху слева).

Если сторона привязки текста задается двумя буквами, то первая буква определяет вертикальную привязку (вверх или вниз от заданной координаты), а вторая — горизонтальную (влево

Пример задачи: написать программу, которая будет выводить на экран круговую диаграмму, отображающую структуру инвестиционного портфеля, состоящего из двух активов: акций Лукойла (доля в портфеле составляет 20%) и акций Сбербанка (доля в портфеле составляет 80%). Кроме самой диаграммы необходимо вывести поясняющий текст.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Структура инвестиционного портфеля')
#Создаем холст
c = Canvas(window, width=500, height=200)
c.pack()
#Рисуем сектор размером 72 градуса из точки 0 градусов
c.create_arc((10, 10), (190, 190), start = 0, extent= 72, fill='light blue')
#Рисуем сектор размером 288 градусов из точки 72 градуса
c.create_arc((10, 10), (190, 190), start = 72,extent = 288, fill='yellow')
#Рисуем два прямоугольника
c.create_rectangle((250,100),(300,120), fill ='light blue')
c.create_rectangle((250,150),(300,170), fill ='yellow')
#Добавляем текст
c.create_text((350,50), text = "Круговая диаграмма", justify = CENTER,fill = "darkgreen", font = "Comic 20 bold")
```

```
    c.create_text((325,110), text = "ВТБ - 20%",anchor = W,fill = "darkgreen",
font = "Comic 10 bold")
    c.create_text((325,160),  text  =  "Сбербанк  -  80%",anchor  =  W,fill  =
"darkgreen", font = "Comic 10 bold")
    window.mainloop()
```

Результат работы программы представлен на рисунке 6.

Рис. 6

3.2. ИДЕНТИФИКАЦИЯ ГРАФИЧЕСКИХ ОБЪЕКТОВ. ИДЕНТИФИКАТОРЫ И ТЕГИ

В модуле **tkinter** существует два способа идентификации графических объектов, расположенных в графическом окне, — идентификаторы и теги. Идентификаторы являются уникальными для каждого объекта, т. е. не могут существовать два разных объекта с одинаковыми идентификаторами. Теги не являются уникальными, т. е. один и тот же тег может присутствовать в атрибутах разных объектов. Таким образом, использование тегов позволяет делать группировки объектов по тегам и, как следствие, при необходимости изменять свойства всей группы одновременно. При этом каждый отдельный графический объект может иметь как идентификатор, так и тег.

В качестве идентификаторов создаваемых графических объектов можно использовать числовые значения, которые возвращают все методы по созданию объектов. Эти идентификаторы можно присвоить каким-то переменным, которые в дальнейшем можно использовать для работы с этими объектами.

Пример задачи: реализовать возможность перемещения прямоугольника в пределах графического окна с помощью стрелок на клавиатуре. Для идентификации соответствующих клавиш клавиатуры использовать их идентификаторы (**<Up>**, **<Down>**, **<Left>**, **<Right>**).

Для реализации перемещения использовать метод **move()**. Для передачи объекта-прямоугольника в метод **move()** использовать его идентификатор, например, **rect**.

Решение:

```
    from tkinter import *
    #Создаем окно
    window = Tk()
    window.title('Идентификаторы объектов')
    #Создаем холст
    can = Canvas(width=400, height=200,bg="white")
    can.focus_set()
    can.pack()
    #Создаем прямоугольник
    rect = can.create_rectangle(150,75,250,125,fill = 'light blue')
    can.bind('<Up>',
             lambda event: can.move(rect, 0, -10))#Смещение вверх на 10
пикселей
    can.bind('<Down>',
             lambda event: can.move(rect, 0, 10))#Смещение вниз на 10 пикселей
    can.bind('<Left>',
             lambda event: can.move(rect, -25, 0))#Смещение влево на 25
пикселей
```

```
    can.bind('<Right>',
             lambda event: can.move(rect, 0, -25))#Смещение вправо на 25
пикселей
    window.mainloop()
```

Результат работы программы представлен на рисунке 7.

Рис.7

В данном примере прямоугольник двигается по холсту с помощью стрелок на клавиатуре. Когда создавался прямоугольник, его идентификатор был присвоен переменной r*ect* Метод **move** объекта Canvas принимает идентификатор и смещение по осям.

Свойства созданных объектов можно изменять по ходу работы программы. Для этого в модуле **tkinter** используется метод **itemconfig()**.

Для изменения местоположения объекта используется метод **coord()**. Если при вызове этого метода в качестве аргумента задать ему только идентификатор или тег объекта, то метод вернет текущие координаты объекта.

Пример задачи: реализовать возможность изменения цвета и размера прямоугольника при нажатии клавиши **<Enter>**.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Идентификаторы объектов')
#Создаем холст
can = Canvas(window, width=400, height=200)
can.focus_set()
can.pack()
#Создаем прямоугольник
rect = can.create_rectangle((150,75),(250,125),fill = 'blue')
#Функция изменяет цвет и размер прямоугольника
def change(event):
    can.itemconfig(rect, fill='green')
    can.coords(rect, 50, 50, 300, 150)
can.bind('<Return>', change)
window.mainloop()
```

Результат работы программы представлен на рисунке 8.

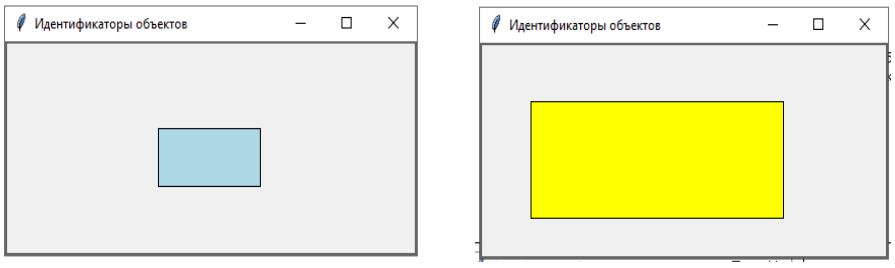


Рис. 8

В отличие от идентификаторов один и тот же тег можно присвоить различным объектам. Фактически теги позволяют осуществлять группировку объектов в программе, что позволяет, например, одновременно изменять свойства объектов, принадлежащих к одной группе (имеющих одинаковый тег). Принципиальное отличие тега от идентификатора с точки типов данных заключается в том, что идентификатор объекта представляет собой целочисленную переменную, в то время как тег должен представлять собой строку.

Пример задачи: реализовать с помощью тега возможность одновременного изменения цвета всех объектов при нажатии левой кнопки мыши.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Использование тегов')
#Создаем холст
can = Canvas(window, width=400, height=200)
can.focus_set()
can.pack()
#Создаем группу фигур под одним тегом group_1
circ = can.create_oval((150,75),(250,125), fill ='yellow', tag = "t_1")
rect = can.create_rectangle((50,25),(150,75), fill= 'green',tag = "t_1")
#Функция изменяет цвет всех фигур из группы с тегом group_1
def change_color(event):
    can.itemconfig('t_1', fill='light blue')
can.bind('<Button-1>', change_color)
window.mainloop()
```

Результат работы программы представлен на рисунке 9.

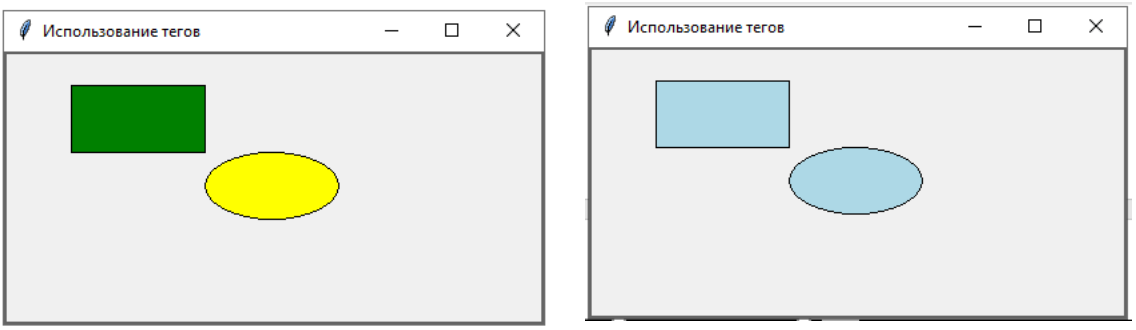


Рис. 9

Удаление объектов из графического окна осуществляется с помощью метода **delete()**. В качестве аргумента методу необходимо передать идентификатор удаляемого объекта или его тег. Для удаления из графического окна всех созданных объектов в качестве аргумента данному методу необходимо передать константу **ALL**.

С помощью метода **tag_bind()** можно осуществить привязку определенного события (например, нажатия кнопки мыши) к определенному графическому объекту в графическом окне. Такой подход позволяет использовать одно и то же событие для обращения к различным частям окна.

Пример задачи: реализовать возможность замены изображения графического объекта на его текстовое название при щелчке левой кнопки мыши по выбранному объекту.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Использование идентификаторов')
#Создаем холст
can = Canvas(window, width=400, height=200)
can.pack()
#Создаем фигуры
circ = can.create_oval((150,75),(250,125), fill ='yellow')
```

```
    rect = can.create_rectangle((50,25),(150,75), fill= 'green')
    trial  =  can.create_polygon((350,20),  (310,80),(390,80),fill  =  "blue",
outline = "yellow")
    #Функции меняют изображение фигур на текст
    def change_circ(event):
        can.delete(circ)
    can.create_text((200,100), text='Эллипс',fill = "red")
    def change_rect(event):
        can.delete(rect)
    can.create_text((100,50),text='Прямоугольник', fill = "red")
    def change_trial(event):
        can.delete(trial)
    can.create_text((350,40), text='Треугольник',fill = "red")
    can.tag_bind(circ, '<Button-1>', change_circ)
    can.tag_bind(rect, '<Button-1>', change_rect)
    can.tag_bind(trial, '<Button-1>', change_trial)
    window.mainloop()
```

Результат работы программы представлен на рисунке 10.

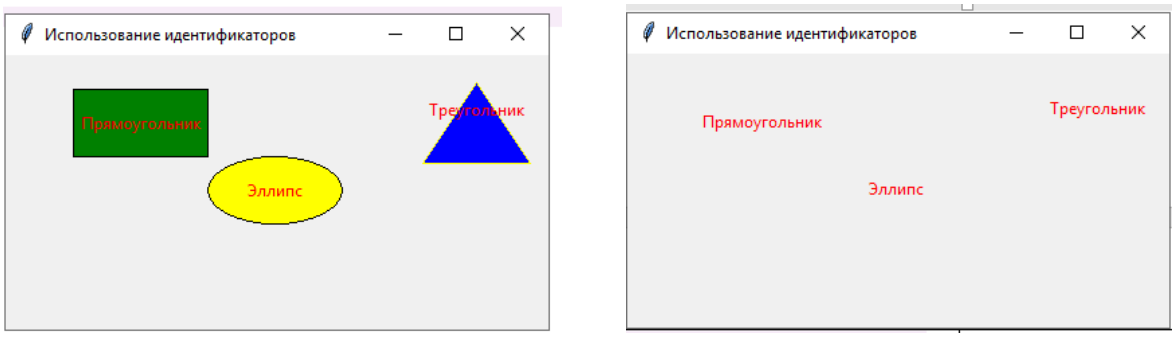


Рис. 10

3.3. СОЗДАНИЕ АНИМАЦИИ

С помощью методов класса **Canvas** в графическом окне можно реализовывать эффект движения изображений объектов.

К основным методам, используемым для создания анимации, относятся методы **move()** и **after()**. Первый из них реализует перемещение объекта в заданную точку графического окна, а второй —вызов функции, переданной ему в качестве второго аргумента через заданные интервалы времени, задаваемые его первым аргументом.

В качестве вспомогательных функций при создании анимации, как правило, используются рассмотренная выше функция **coords()**, возвращающая список текущих координат объекта, а также функции **winfo_reqwidth()** и **winfoheight()**, возвращающие ширину и высоту графического окна соответственно.

Пример задачи: реализовать простую анимацию, представляющую собой перемещение круга от левой границы графического окна до его правой границы. По достижении правой границы объект (круг) должен исчезать из окна.

Решение:

```
from tkinter import *
#Создаем окно
window = Tk()
window.title('Простая анимация')
#Создаем холст
can = Canvas(window, width=400, height=200, bg='yellow')
can.pack()
width = can.winfo_reqwidth()-4 #Вычисляем ширину холста
#Создаем круг
ball = can.create_oval((0,100),(80,180), fill ='green')
print(can.coords(ball)[0],can.coords(ball)[1],can.coords(ball)[2])
```

```
#Функция для реализации движения
def motion():
    can.move(ball, 5, 0)
    if can.coords(ball)[2] < width:
        window.after(10, motion)
    else:
        can.delete(ball)
motion()
window.mainloop()
```

Результат работы программы представлен на рисунке 11.

Рис. 11

Комментарии к программе:

Для получения значения ширины окна используется метод **winfo_reqwidth()**. От полученного значения отнимаются четыре пикселя, которые занимают две границы холста (левую и правую).

Метод **move()** увеличивает координату **x** объекта **ball**, переданного ему в качестве первого аргумента, на один пиксель.

Метод **coords()** возвращает список текущих координат объекта **ball**, переданного ему в качестве аргумента. Третий элемент этого списка содержит значение координаты **x** объекта **ball**.

Метод **after()** в качестве аргументов получает интервал времени (в данном случае это 10 миллисекунд) и функцию, которую нужно вызывать через этот интервал (функция **motion()**).

3.4. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ ИЗ ФАЙЛОВ

Модуль **tkinter** позволяет работать с файлами изображений в форматах GIF и PGM/PPM, которые могут быть выведены на виджетах **Label**, **Text**, **Button** и **Canvas**. Для этих целей используется класс **Photoimage**.

Для создания объекта-изображения, как и в случае со всеми остальными виджетами, используется конструктор **Photoimage()** этого класса. В качестве аргумента конструктору необходимо указать имя файла с изображением в виде **file = ‘имя файла’**.

Для уменьшения размера изображения можно воспользоваться методом **subsample()**, передав ему в качестве аргументов параметры дискретизации по горизонтали и по вертикали в виде **x = значение** и **y = значение**. Например, значения **x = 2**, **y = 2** приведут к отбрасыванию каждого второго пикселя — результатом будет уменьшение изображения в два раза по сравнению с оригиналом.

Для увеличения размера изображения в классе **Photoimage** существует и обратный метод **zoom()**, увеличивающий размер изображения в соответствии с переданными ему аргументами **x** и **y**.

После создания объекта-изображения его можно добавлять на виджеты с помощью передачи аргумента **image** в соответствующих конструкторах.

У объектов-виджетов **Text** существует метод **image_create()**, с помощью которого изображение встраивается в текстовое поле. Данный метод принимает два аргумента: первый — для определения позиции размещения (например, **‘1.0’** указывает первую строку и первый символ), а второй представляет собой ссылку на само изображение в виде аргумента **image**.

Объекты класса **Canvas** имеют аналогичный метод

create_image(), тоже принимающий два аргумента, только первый аргумент, отвечающий за расположение изображения, представлен в виде пары координат **(x, y)**, которые определяют координаты точки графического окна, куда помещается изображение.

Замечание. Несмотря на то что методы **image_create()** класса **Text** и **create_image()** класса **Canvas** очень похожи, они все-таки отличаются друг от друга.

Пример задачи: реализовать размещение графического изображения из внешнего файла на различных виджетах.

Решение:

```
from tkinter import *
window = Tk()
window.title( 'Работа с изображением' )
#Создаем объект-изображение
img = PhotoImage( file = '1.gif' )
#Уменьшаем полученное изображение в два раза
small_img = PhotoImage.subsample( img , x = 2 ,y = 2 )
#Создаем метку с изображением
label = Label( window , image = img , bg ='yellow' )
#Создаем кнопку с изображением
btn = Button( window , bg = 'red', image =small_img )
#Создаем текстовое поле с изображением
txt = Text( window , width = 25 , height = 7, fg ='darkgreen' )
txt.image_create( '1.0' , image = small_img )
txt.insert( '1.1', 'Tkinter!' )
#Создаем холст с изображением
can = Canvas( window , width = 100 , height = 100, bg = 'black' )
can.create_image( ( 50 , 50 ), image = small_img )
can.create_line( 0 , 0 , 100 , 100, width = 25 ,fill = 'yellow' )
#Размещаем созданные виджеты в окне приложения
label.pack( side = TOP )
btn.pack( side = LEFT , padx = 10 )
txt.pack( side = LEFT )
can.pack( side = LEFT, padx = 10 )
window.mainloop()
```

Результат работы программы представлен на рисунке 12.

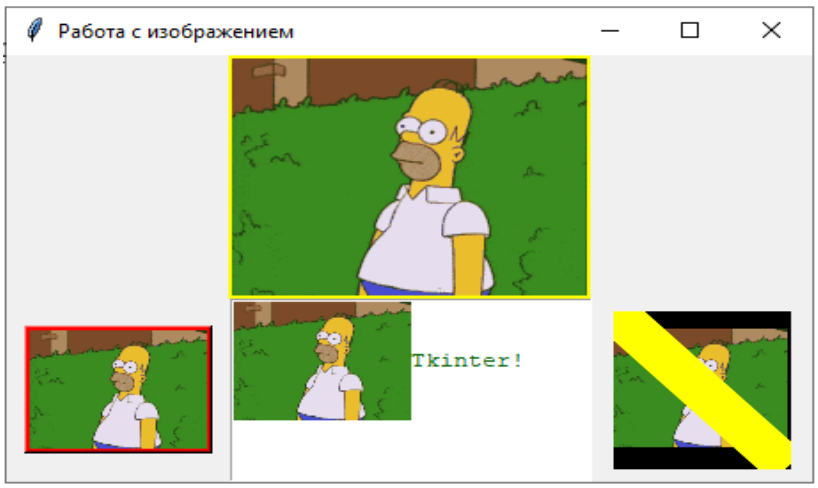


Рис. 12

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

1.Составьте программу с графическим пользовательским интерфейсом, которая будет отображать на экране структуру инвестиционного портфеля клиента в виде круговой диаграммы.

Возможный результат работы программы представлен на рисунке 13.

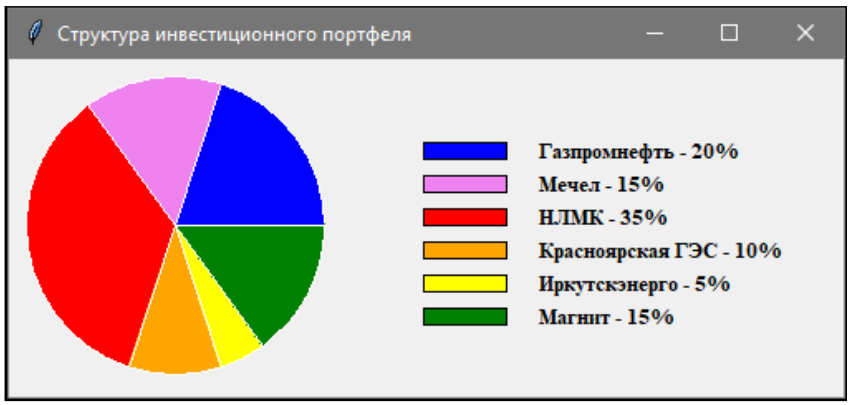


Рис. 13

2. Составьте программу с графическим пользовательским интерфейсом, которая будет отображать на экране структуру инвестиционного портфеля клиента в виде столбчатой.

Возможный результат работы программы представлен на рисунке 14.

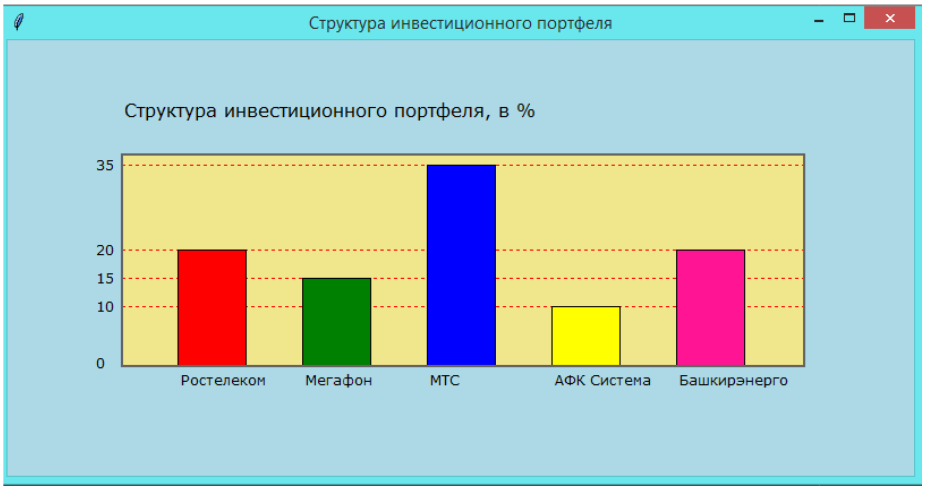


Рис. 14

3. Составьте программу с графическим пользовательским интерфейсом и элементами анимации. Тема произвольная.