Модуль Graph

Версия 1.3 от 27.07.2018

Общее описание

Модуль graph – это набор функций, который представляет собой «обёртку» для создания учебных графических программ на языке Python на основе виджета Canvas библиотеки Tkinter.

Работа с окном

`mainWindow()`

функция возвращает ссылку на объект главного окна, что позволяет использовать все возможности Tkinter.

`windowSize(width, height)`

установка ширины (width) и высоты (height) рабочей области окна; при вызове без параметров возвращает кортеж (width,height) с текущими размерами:

```
width, height = windowSize()
```

`canvas()`

функция возвращает ссылку на объект области рисования, что позволяет использовать все возможности виджета Canvas пакета Tkinter.

`canvasPos(x, y)`

установка координат (x,y) области рисования; при вызове без параметров возвращает кортеж (x,y) с текущими размерами:

```
x, y = canvasPos()
```

`canvasSize(width, height)`

установка ширины (width) и высоты (height) области рисования; при вызове без параметров возвращает кортеж (width,height) с текущими размерами:

```
width, height = canvasSize()
```

`viewCoords(x1, x2, y1, y2)`

устанавливает для каждой оси диапазоны математической декартовой системе координат, которые отображаются в области рисования; при вызове без параметров возвращается к экранной системе координат.

`pointInView(x, y)`

логическая функция, которая возвращает True, если точка с координатами (x,y) находится в пределах области рисования и False, если точка находится вне области.

`circleInView(x, y, r)`

логическая функция, которая возвращает True, если окружность с центром в точке (x,y) радиуса r находится в пределах области рисования и False, если точка находится вне области.

`run()`

запускает основной цикл обработки сообщений; вызов этой функции должен быть последней строчкой в любой графической программе.

`close()`

завершает основной цикл обработки сообщений; закрывает графическое окно.

Обработчики событий области рисования (Canvas)

`onTimer(fn, time)`

установить функцию `fn`, которая будет вызываться по таймеру каждые `time` миллисекунд.

`onKey(key)`
`onKey(fn)`
`onKey(key, fn)`

установить функцию `fn` как обработчик нажатия клавиши с символьным обозначением `key`; если функция не указана, обработчик нажатия этой клавиши отключается; если не указана клавиша, устанавливается один обработчик на все клавиши; функция `fn` должна принимать один параметр – блок данных о событии; через поле `keycode` этого блока можно получить код нажатой клавиши:

```
def keyPressed(event):
  if event.keycode == VK_LEFT:
    move(obj, x-5, y)
```

Список кодов виртуальных клавиш, определенных в модуле `graph.py`:

```
VK_SPACE = 0x20
VK_PRIOR = 0x21 # PAGE UP key
VK_NEXT  = 0x22 # PAGE DOWN key
VK_END   = 0x23 # END key
VK_HOME  = 0x24 # HOME key
VK_LEFT  = 0x25
VK_UP    = 0x26
VK_RIGHT = 0x27
VK_DOWN  = 0x28
VK_INSERT = 0x2D # INS key
VK_DELETE = 0x2E # DELETE key
VK_BACK   = 0x08 # BACKSPACE key
VK_TAB    = 0x09 # TAB key
VK_RETURN = 0x0D # RETURN key
VK_ESCAPE = 0x1B # ESC key
```

Для того, чтобы выяснить коды и символьные обозначения клавиш, можно использовать такую программу:

```
import tkinter as tk
def __keyPress(event):
  print("Key Press Event:")
  print("  event.char:", event.char)
  print("  event.keysym:", event.keysym)
  print("  event.keycode:", event.keycode)
  print("  event.keysym_num:", event.keysym_num)
root = tk.Tk()
root.bind("<KeyPress>", __keyPress)
tk.mainloop()
```

`onMouseMove(fn)`

установить функцию `fn` как обработчик события движения мыши; если функция не указана, обработчик отключается; функция `fn` должна принимать один параметр – блок данных о событии; через поля этого блока можно получить информацию о положении мыши:

```
def handleMove(event):
  print("Координаты: (%s %s)" % (event.x, event.y))
```

```
        print("Относительно экрана: (%s %s)" %
              (event.x_root, event.y_root))
```

`onMouseDown(fn, btn)`
: установить функцию `fn` как обработчик нажатия кнопки мыши с номером `btn` (`btn=1` для левой кнопки, `btn=2` – для средней и `btn=3` для правой); если номер кнопки мыши не задан, устанавливается обработчик для всех кнопок; если функция не указана, обработчик отключается; функция `fn` должна принимать один параметр – блок данных о событии; через поля этого блока можно получить информацию о положении мыши (см. выше).

`onMouseUp(fn, btn)`
: установить функцию `fn` как обработчик отпускания кнопки мыши с номером `btn` (`btn=1` для левой кнопки, `btn=2` – для средней и `btn=3` для правой); если номер кнопки мыши не задан, устанавливается обработчик для всех кнопок; если функция не указана, обработчик отключается; функция `fn` должна принимать один параметр – блок данных о событии; через поля этого блока можно получить информацию о положении мыши (см. выше).

`onMouseClick(fn, btn)`
: установить функцию `fn` как обработчик щелчка кнопкой мыши с номером `btn` (`btn=1` для левой кнопки, `btn=2` – для средней и `btn=3` для правой);в текущей версии равносильно обработчику `onMouseUp`.

`onMouseDblClick(fn, btn)`
: установить функцию `fn` как обработчик двойного щелчка кнопкой мыши с номером `btn` (`btn=1` для левой кнопки, `btn=2` – для средней и `btn=3` для правой); если номер кнопки мыши не задан, устанавливается обработчик для всех кнопок; если функция не указана, обработчик отключается; функция `fn` должна принимать один параметр – блок данных о событии; через поля этого блока можно получить информацию о положении мыши (см. выше).

Команды рисования геометрических фигур

`penSize(width)`
: установка толщины пера; при вызове без параметров функция возвращает текущую толщину пера:

```
width = penSize()
```

`penColor(color)`
`penColor(r, g, b)`
: установка цвета пера; при вызове с одним параметром цвет `color` может быть задан как символьная строка с названием цвета (`"red"`, "green" и т.д.) или как символьная строка с HTML-кодом цвета (`"#FF00GFF"`) или как кортеж `(r,g,b)` со значениями составляющих цвета в модели RGB; при вызове с тремя параметрами они воспринимаются как значения составляющих цвета в модели RGB; при вызове без параметров функция возвращает текущий цвет:

```
color = penColor()
```

`brushColor(color)`
`brushColor(r, g, b)`
: установка цвета заливки; при вызове с одним параметром цвет `color` может быть задан как символьная строка с названием цвета (`"red"`, "green" и т.д.) или как символьная строка с HTML-кодом цвета (`"#FF00GFF"`) или как кортеж `(r,g,b)` со значениями составляющих цвета в модели RGB; при вызове с тремя параметрами

они воспринимаются как значения составляющих цвета в модели RGB; при вызове без параметров функция возвращает текущий цвет:

```
color = brushColor()
```

`randColor()`
: функция возвращает случайный цвет в виде символьной строки с HTML-кодом цвета (`"#FF00GFF"`).

`point(x, y)`
`point(x, y, color)`
: нарисовать точку цвета `c` с координатами `(x,y)`; если цвет не задан, используется текущий цвет лини, установленный ранее с помощью команды `penColor`; функция возвращает ссылку на объект-точку.

`moveTo(pos)`
`moveTo(x, y)`
: переместить исполнителя в точку, заданную координатами `(x,y)` или кортежем `pos=(x,y)`, составленным из этих координат.

`lineTo(pos)`
`lineTo(x, y)`
: нарисовать линию из текущего положения исполнителя в точку, заданную координатами `(x,y)` или кортежем `pos=(x,y)`, составленным из этих координат; цвет линии определяется последней командой `penColor`; функция возвращает ссылку на объект-отрезок.

`line(x1, y1, x2, y2)`
: нарисовать линию между точками с координатами `(x1,y1)` и `(x2,y2)`; цвет линии определяется последней командой `penColor`; функция возвращает ссылку на объект-отрезок.

`polyline(p)`
: нарисовать ломаную линию по точками, заданным как массив кортежей `p` (каждый элемент массива – кортеж `(x,y)` координат очередной точки); цвет линии определяется последней командой `penColor`; функция возвращает ссылку на объект-ломаную.

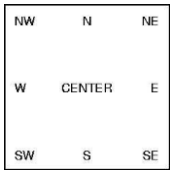
`polygon(points)`
: нарисовать многоугольник с заливкой по точками, заданным как массив кортежей `points` (каждый элемент массива – кортеж `(x,y)` координат очередной точки); цвет контура и заливки определяются последними командами `penColor` и `brushColor`; функция возвращает ссылку на объект-многоугольник.

`rectangle(x1, y1, x2, y2)`
: нарисовать прямоугольник с координатами противолежащих углов `(x1,y1)` и `(x2,y2)`; цвет контура и заливки определяются последними командами `penColor` и `brushColor`; функция возвращает ссылку на объект-прямоугольник.

`circle(x, y, r)`
: нарисовать окружность с заливкой с центром в точке `(x,y)` радиуса `r`; цвет контура и заливки определяются последними командами `penColor` и `brushColor`; функция возвращает ссылку на объект-окружность.

`image(x, y, fileName, anchor = NW, **kwargs):`
: добавить на холст рисунок из файла; опорная точка рисунка находится в точке холста `(x,y)`; рисунок загружается из файла `fileName`; если не подключена библиотека

Pillow (https://pypi.python.org/pypi/Pillow), то загружаются только рисунки формата `.GIF`, если эта библиотека подключена, можно загружать и рисунки других форматов; параметр `anchor` определяет привязку рисунка:



по умолчанию привязка идёт к левому верхнему углу (NW); функции можно передавать и другие именованные параметры метода `create_image` класса `Canvas`; функция возвращает ссылку на объект-рисунок.

Работа с графическими объектами

`coords(obj)`
: функция возвращает массив из 4-х элементов: `[x1,y1,x2,y2]`, которые обозначают координаты противоположных углов прямоугольника, `(x1,y1)` и `(x2,y2)`, в который вписано изображение объекта.

`xCoord(obj)`
: функция возвращает x-координату левого верхнего угла прямоугольника, в который вписано изображение объекта.

`yCoord(obj)`
: функция возвращает y-координату левого верхнего угла прямоугольника, в который вписано изображение объекта.

`changeCoords(obj, pos)`
: установить для объекта `obj` новые координаты `pos=[x1,y1,x2,y2]` – это координаты противоположных углов прямоугольника, `(x1,y1)` и `(x2,y2)`, в который вписано изображение объекта.

`changePenColor(obj, color)`
: установить для объекта `obj` новый цвет контура `color`.

`changeFillColor(obj, color)`
: установить для объекта `obj` новый цвет заливки `color`.

`changeProperty(obj, ...)`
: установить для объекта `obj` новые свойства, например:

```
changeProperty(obj, fill="green")
```

`moveObjectTo(obj, x, y)`
: переместить объект `obj` в точку с координатами `(x,y)` (с этой точкой будет совпадать левый верхний угол объекта).

`moveObjectBy(obj, dx, dy)`
: переместить объект `obj` на вектор `(dx,dy)`.

`deleteObject(obj)`
: удалить объект по ссылке `obj`.

Работа с виджетами (элементами интерфейса)

`label(text, x, y, ...)`

добавить текстовую метку с надписью `text` в точке с координатами `(x,y)`; функция возвращает ссылку на объект-метку; после обязательных параметров можно добавлять любые параметры виджета `Label` (см. описание модуля `Tkinter`).

`button(text, x, y, width, fn, ...)`

добавить кнопку с надписью `text` в точке с координатами `(x,y)`; ширина кнопки (в символах) равна `width`, при щелчке по кнопке вызывается функция `fn`; функция возвращает ссылку на объект-кнопку; после обязательных параметров можно добавлять любые параметры виджета `Button` (см. описание модуля `Tkinter`).