# **pythonGraph Overview**

The pythonGraph module lets developers output graphic images to a window.  In addition, it provides functionality to interact with the user via the mouse and via individual keystrokes on the keyboard.

pythonGraph is available *via* the Python Package Management, pip. Alternatively, you can download the full source code at <https://github.com/USAFA-CompSci110/pythonGraph>.

Getting Started

To get started, import the pythonGraph library in your python file, as shown below:



Then, open a pythonGraph window by calling the open\_window function and specifying the dimensions of the window.  A successfully opened window will appear with a white background.

pythonGraph utilizes a coordinate system where **the origin (0, 0) is at the top-left hand corner**.  When the program requests the mouse's position or the location of a click, these will be given using the same coordinate system.

The picture below shows the coordinate layout for a pythonGraph window opened with open\_window(400,300).

|  |
| --- |
| **Example Code** |
|  |
|  |
| **Output** |
|  |

Table of Contents

[**pythonGraph Overview** 1](#_Toc16083121)

[Getting Started 1](#_Toc16083122)

[1. Drawing Operations 4](#_Toc16083123)

[1.1 clear\_window 5](#_Toc16083124)

[1.2 draw\_arc 6](#_Toc16083125)

[1.3 draw\_image 7](#_Toc16083126)

[1.4 draw\_rectangle 8](#_Toc16083127)

[1.5 draw\_circle 9](#_Toc16083128)

[1.6 draw\_ellipse 10](#_Toc16083129)

[1.7 draw\_line 11](#_Toc16083130)

[1.8 draw\_pixel 12](#_Toc16083131)

[1.9 draw\_text 13](#_Toc16083132)

[2. Mouse Operations 14](#_Toc16083133)

[2.1 get\_mouse\_x and get\_mouse\_y 15](#_Toc16083134)

[2.2 mouse\_button\_pressed 16](#_Toc16083135)

[2.3 mouse\_button\_down 17](#_Toc16083136)

[2.4 mouse\_button\_released 18](#_Toc16083137)

[3. Keyboard Operations 19](#_Toc16083138)

[3.1 key\_pressed 20](#_Toc16083139)

[3.2 key\_down 21](#_Toc16083140)

[3.3 key\_released 22](#_Toc16083141)

[4. Window Operations 23](#_Toc16083142)

[4.1 open\_window 24](#_Toc16083143)

[4.2 close\_window 25](#_Toc16083144)

[4.3 get\_window\_height and get\_window\_width 26](#_Toc16083145)

[4.4 is\_open 27](#_Toc16083146)

[4.5 set\_window\_title 28](#_Toc16083147)

[4.6 update\_window 29](#_Toc16083148)

[4.7 delay 30](#_Toc16083149)

[5. Color Operations 31](#_Toc16083150)

[5.1 create\_color 32](#_Toc16083151)

[5.2 create\_random\_color 33](#_Toc16083152)

[6. Music Operations 34](#_Toc16083153)

[6.1 play\_sound\_effect 35](#_Toc16083154)

[6.2 play\_music 36](#_Toc16083155)

[6.3 stop\_music 37](#_Toc16083156)

## Drawing Operations

pythonGraph’s drawing routines can output a variety of shapes in a variety of colors.

Before using these operations, please note that:

* open\_window must be called first, otherwise a run-time error will occur.
* You must call update\_window before the result of the drawing routines will be visible on the screen.

Methods Described in this Chapter

* [clear\_window](#_clear_window)
* [draw\_arc](#_draw_arc)
* [draw\_image](#_draw_image)
* [draw\_rectangle](#_draw_rectangle)
* [draw\_circle](#_draw_circle)
* [draw\_ellipse](#_draw_ellipse)
* [draw\_line](#_draw_line)
* [draw\_pixel](#_draw_pixel)
* [draw\_text](#_draw_text)

### clear\_window

|  |
| --- |
| **Usage** |
| clear\_window(color) |
|  |
| **Description** |
| Clears the entire window to a particular color.  **color** can either be a predefined value (refer to pythonGraph.colors) or a custom color created using the **create\_color** function. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_arc

|  |
| --- |
| **Usage** |
| draw\_arc(x1, y1, x2, y2, start\_x, start\_y, end\_x, end\_y, color, width) |
|  |
| **Description** |
| Draws a portion of the ellipse that is inscribed inside the given rectangle:    The parameters **(x1, y1)** and **(x2, y2)** represent the two opposite corners of the rectangle.  The arc begins at the intersection of the ellipse and the line passing through the center of the ellipse and **(start\_x, start\_y)**.  It then proceeds counter-clockwise until it reaches the intersection of the ellipse and the line passing through the center of the ellipse to **(end\_x, end\_y)**.  **color** specifies the arc’s color. This can either be a predefined value (refer to pythonGraph.colors) or a custom color created using the **create\_color** function.  **width** is an optional parameter that specifies the “thickness” of the arc. Otherwise, it uses a default value. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_image

|  |
| --- |
| **Usage** |
| draw\_image(filename, x, y, width, height) |
|  |
| **Description** |
| Draws an image in the pythonGraph window.  **filename** refers to the name of the file (*e.g.,* “image.png”) to be drawn. You can use any BMP, JPEG, or PNG file. **The image file should be in the same folder as your python script.**  **x** and **y** specify the upper-left coordinate where the image is to be drawn.  **width** and **height** represent the desired dimensions of the image. pythonGraph will try to scale the image to fit within these dimensions. |
|  |
| **Example** |
| For this example, assume that the file “falcon.png” exists. |
|  |
| **Output** |
|  |

### draw\_rectangle

|  |
| --- |
| **Usage** |
| draw\_rectangle(x1, y1, x2, y2, color, filled, width) |
|  |
| **Description** |
| Draws a rectangle on the screen.  (x1, x2) is any corner of the rectangle  (x2, y2) is the opposite corner of the rectangle  **color** specifies the rectangle’s color. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function.   filled can be either True or False, depending on whether or not the rectangle should be filled in or not, respectively.  width is an optional parameter that specifies the width of the rectangle’s border. If this value is not provided, a default value will be used. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_circle

|  |
| --- |
| **Usage** |
| draw\_circle(x, y, radius, color, filled, width) |
|  |
| **Description** |
| Draws a circle at (x, y) with the specified radius  **color** specifies the circle’s color. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function.   filled can be either True or False, depending on whether or not the circle should be filled in or not, respectively.  width is an optional parameter that specifies the width of the circle’s border. If this value is not provided, a default value will be used. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_ellipse

|  |
| --- |
| **Usage** |
| draw\_ellipse(x1, y1, x2, y2, color, filled, width) |
|  |
| **Description** |
| Draws an ellipse inscribed in the rectangle whose two diagonally opposite corners,  (x1, y1), (x2, y2) are given:    **color** specifies the ellipse’s color. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function.   filled can be True or False, depending on whether or not the ellipse is filled in or not, respectively.  width is an optional parameter that specifies the width of the ellipse’s border. If this value is not provided, a default value will be used. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_line

|  |
| --- |
| **Usage** |
| draw\_line(x1, y1, x2, y2, color, width=2) |
|  |
| **Description** |
| Draws a line segment from (x1, y1) to (x2, y2) in the given color:  **color** specifies the line’s color. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function.   width is an optional parameter that specifies the width of the line. If this value is not provided, a default value will be used. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_pixel

|  |
| --- |
| **Usage** |
| draw\_pixel(x, y, color) |
|  |
| **Description** |
| Changes the color of a single pixel at location (x, y).  **color** specifies the color for the pixel. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### draw\_text

|  |
| --- |
| **Usage** |
| draw\_text(text, x, y, color, font\_size) |
|  |
| **Description** |
| Writes the specified text string to the pythonGraph window.  text represents the string to be written. This can either be a single string (e.g., “Hello World”) or a concatenated string (e.g., “Bob is “ + str(5) + “ years old!”)  (x,y) denotes the coordinate of the top left corner of the string  **color** specifies the color for the pixel. This can be:   * a predefined value (refer to pythonGraph.colors), * a string representing the desired color (“BLUE”), * or a custom color created using the **create\_color** function.   font\_size is an optional parameter that specifies the size of the text, in pixels. If this value is not provided, a default value will be used. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

## Mouse Operations

pythonGraph can determine the current location of the mouse. It can also determine whether or not a mouse click has occurred.

Before using these operations, please note that:

* open\_window must be called first, otherwise a run-time error will occur.
* The window must be in focus. If the pythonGraph window is not on top, the user may have to click on it once before the application will respond to user mouse clicks.

Methods Described in this Chapter

* [get\_mouse\_x](#_get_mouse_x_and_get_mouse_y)
* [get\_mouse\_y](#_get_mouse_x_and_get_mouse_y)
* [mouse\_button\_pressed](#_mouse_button_pressed)
* [mouse\_button\_down](#_mouse_button_down)
* [mouse\_button\_released](#_mouse_button_released)

### get\_mouse\_x and get\_mouse\_y

|  |
| --- |
| **Usage** |
| get\_mouse\_x()  get\_mouse\_y() |
|  |
| **Description** |
| These functions output the current x or y coordinate of the mouse. |
|  |
| **Example** |
| The following lines of code will store the mouse’s current x and y coordinate in x\_coordinate, and y\_coordinate, respectively. |
|  |

### mouse\_button\_pressed

|  |
| --- |
| **Usage** |
| mouse\_button\_pressed(which\_button) |
|  |
| **Description** |
| Returns True if the specified mouse button is clicked, and False otherwise. This function will only return True once per mouse click.  which\_button can be one of the following values:   * + “LEFT”   + “RIGHT”   + “CENTER”   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will print a string when the left mouse button is clicked: |

### mouse\_button\_down

|  |
| --- |
| **Usage** |
| mouse\_button\_down(which\_button) |
|  |
| **Description** |
| Returns True if the specified mouse button is held down, and False otherwise. Unlike mouse\_button\_pressed, this function will keep returning True for as long as the button is held down.  which\_button can be one of the following values:   * + “LEFT”   + “RIGHT”   + “CENTER”   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will print a string when the left mouse button is pressed: |

### mouse\_button\_released

|  |
| --- |
| **Usage** |
| mouse\_button\_released(which\_button) |
|  |
| **Description** |
| Returns True if the specified mouse button is released, and False otherwise.  which\_button can be one of the following values:   * + “LEFT”   + “RIGHT”   + “CENTER”   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will print a string when the left mouse button is released: |

## Keyboard Operations

These functions allow pythonGraph to determine if a keystroke has occurred.

Before using these operations, please note that:

* open\_window must be called first, otherwise a run-time error will occur.
* The window must be in focus. If the pythonGraph window is not on top, the user may have to click on it once before the application will respond to user keyboard.

Methods Described in this Chapter

* [key\_pressed](#_key_pressed)
* [key\_down](#_key_down)
* [key\_released](#_key_released)

### key\_pressed

|  |
| --- |
| **Usage** |
| key\_pressed(which\_key) |
|  |
| **Description** |
| Returns True if the specified key is pressed, and False otherwise. This function will only return True once per keyboard press.  which\_key is a string that represents the key that we want to check. For example:   * + ‘a’   + ‘f1’   + ‘left’   + ‘escape’   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will print a string when the ‘a’ key is pressed:    The following code snippet will print a string when the up arrow key is pressed: |

### key\_down

|  |
| --- |
| **Usage** |
| key\_down(which\_button) |
|  |
| **Description** |
| Returns True if the specified key is held down, and False otherwise. Unlike key\_pressed, this function will keep returning True for as long as the key is held down.  which\_key is a string that represents the key that we want to check. For example:   * + ‘a’   + ‘f1’   + ‘left’   + ‘escape’   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will repeatedly print a string for as long as the ‘a’ button is pressed: |

### key\_released

|  |
| --- |
| **Usage** |
| key\_released(which\_button) |
|  |
| **Description** |
| Returns True if the specified key is released, and False otherwise.  which\_key is a string that represents the key that we want to check. For example:   * + ‘a’   + ‘f1’   + ‘left’   + ‘escape’   If the window is not on top, the user may have to click on it once before this function will be called. |
|  |
| **Example** |
| The following code snippet will print a string when ‘a’ key is released: |

## Window Operations

The following functions allow pythonGraph to open, close, and update the graphics window.

Methods Described in this Chapter

* [open\_window](#_open_window)
* [close\_window](#_close_window)
* [get\_window\_height](#_get_window_height_and_get_window_wi)
* [get\_window\_width](#_get_window_height_and_get_window_wi)
* [is\_open](#_is_open)
* [set\_window\_title](#_set_window_title)
* [update\_window](#_update_window)

### open\_window

|  |
| --- |
| **Usage** |
| open\_window(width, height) |
|  |
| **Description** |
| Creates a graphics window of the specified width and height (in pixels).  Important Notes:   * You can only have one pythonGraph window open at a time. If you attempt to open a second, an error will occur. * The width and height dimensions cannot be negative |
|  |
| **Example** |
| The following code snippet will open a 400 x 300 pixel window: |

### close\_window

|  |
| --- |
| **Usage** |
| close\_window(width, height) |
|  |
| **Description** |
| Closes the pythonGraph window. A run-time error will occur if the graphics window is not open. |
|  |
| **Example** |
|  |

### get\_window\_height and get\_window\_width

|  |
| --- |
| **Usage** |
| get\_window\_height() get\_window\_width() |
|  |
| **Description** |
| Returns the height and width, respectively, of the window. |
|  |
| **Example** |
| The following snippet will store the window’s height and width in the variables h and w, respectively. |

### is\_open

|  |
| --- |
| **Usage** |
| is\_open() |
|  |
| **Description** |
| Returns True if the pythonGraph window is currently open, and False otherwise. |
|  |
| **Example** |
| The following code snippet will output the state of the window. |

### set\_window\_title

|  |
| --- |
| **Usage** |
| set\_window\_title(title) |
|  |
| **Description** |
| Changes the title of the pythonGraph window. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### update\_window

|  |
| --- |
| **Usage** |
| update\_window() |
|  |
| **Description** |
| Updates the visual contents of the pythonGraph window. All of the draw functions called prior to this will now appear on the screen.  Every pythonGraph program should call this function at least once per frame (see example below). Without this call, the application will freeze. |
|  |
| **Example** |
|  |
|  |
| **Output** |
|  |

### delay

|  |
| --- |
| **Usage** |
| delay(time) |
|  |
| **Description** |
| Pauses the application for the specified amount of time (in milliseconds).  This function is typically called during an animation loop in order to allow the image to stay on the screen long enough for the user to see it.  The time parameter expects a positive integer. |
|  |
| **Example** |
|  |

## Color Operations

pythonGraph comes with a predefined set of colors, as well as methods to easily generate custom and/or random colors as needed.

**Predefined colors:**

The Official and “Friendly” names can be used interchangeably.

|  |  |
| --- | --- |
| Official Name | **String Name (Case Insensitive)** |
| pythonGraph.colors.BLACK | “BLACK” |
| pythonGraph.colors.BLUE | “BLUE” |
| pythonGraph.colors.BROWN | “BROWN” |
| pythonGraph.colors.CYAN | “CYAN” |
| pythonGraph.colors.GRAY | “GRAY” |
| pythonGraph.colors.GREEN | “GREEN” |
| pythonGraph.colors.LIGHT\_BLUE | “LIGHT\_BLUE” |
| pythonGraph.colors.LIGHT\_CYAN | “LIGHT\_CYAN” |
| pythonGraph.colors.LIGHT\_GRAY | “LIGHT\_GRAY” |
| pythonGraph.colors.LIGHT\_GREEN | “LIGHT\_GREEN” |
| pythonGraph.colors.LIGHT\_MAGENTA | “LIGHT\_MAGENTA” |
| pythonGraph.colors.LIGHT\_RED | “LIGHT\_RED” |
| pythonGraph.colors.MAGENTA | “MAGENTA” |
| pythonGraph.colors.RED | “RED” |
| pythonGraph.colors.WHITE | “WHITE” |
| pythonGraph.colors.YELLOW | “YELLOW” |
| pythonGraph.colors.ORANGE | “ORANGE” |

Methods Described in this Chapter

* [create\_color](#_create_color)
* [create\_random\_color](#_create_random_color)

### create\_color

|  |
| --- |
| **Usage** |
| create\_color(red, green, blue) |
|  |
| **Description** |
| Returns a color with the specified red, green, and blue combination.  red, green, and blue are all integer values between 0-255. Refer to <https://www.colorspire.com/rgb-color-wheel/> to see how combinations of these three colors can be used to create other colors. |
|  |
| **Example** |
|  |

### create\_random\_color

|  |
| --- |
| **Usage** |
| create\_random\_color() |
|  |
| **Description** |
| Returns a color with a random red, green, and blue combination. |
|  |
| **Example** |
|  |

## Music Operations

pythonGraph provides limited functions to play sound effects and background music. A sound effect is defined as a short sound clip (< 1s). Background music, in contrast, can range from seconds to minutes, and can be set to be played once or on a continuous loop.

WAV and MP3 files are currently supported.

Methods Described in this Chapter

* [play\_sound\_effect](#_play_sound_effect)
* [play\_music](#_play_music)
* [stop\_music](#_stop_music)

### play\_sound\_effect

|  |
| --- |
| **Usage** |
| play\_sound\_effect(filename) |
|  |
| **Description** |
| Plays the specified sound file once, if a channel is available.  The filename parameter specifies where the file to be played is located on the computer. Typically, your sound effect files should be in the same folder as your python application.  This method supports WAV and MP3 files. The larger the file, the longer it will take for the application to load and play it. |
|  |
| **Example** |
| This snippet will play the sound “sound.mp3”, assuming that the file is in the same folder. |

### play\_music

|  |
| --- |
| **Usage** |
| play\_music(filename, loop) |
|  |
| **Description** |
| Plays the specified music file, if a channel is available.  The filename parameter specifies where the file to be played is located on the computer. Typically, your music files should be in the same folder as your python application.  The loop parameter is optional, and specifies whether or not to play the music on a continuous loop. By default, this value is set to True.  This method supports WAV and MP3 files. The larger the file, the longer it will take for the application to load and play it. |
|  |
| **Example** |
| This snippet will play the sound “music.mp3”, assuming that the file is in the same folder. |

### stop\_music

|  |
| --- |
| **Usage** |
| stop\_music() |
|  |
| **Description** |
| Stops any music that is currently playing. This function can be safely called, even if music is not playing. |
|  |
| **Example** |
|  |