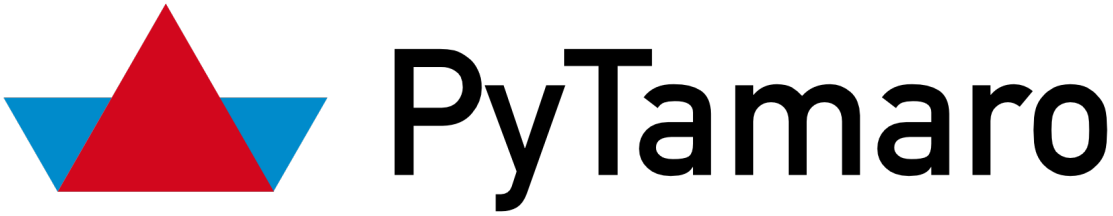
PyTamaro

Release 0.2

Luca Chiodini

Feb 22, 2023

INDEX:



Welcome to PyTamaro's documentation!

CHAPTER

ONE

HOW TO INSTALL PYTAMARO

1.1 Using *pip*

After having configured a Python interpreter along with the *pip* packet manager, you can easily install PyTamaro:

```
pip install pytamaro
```

1.2 Using Thonny

If you are using Thonny as IDE, you can install PyTamaro using the GUI:

1. Select the "Tools" menu, then choose "Manage packages"
2. Type "pytamaro" in the search field and click the "Search on PyPI" button
3. Select "PyTamaro" in the search results, then click the "Install" button.

CHAPTER

TWO

EXAMPLES AS JUPYTER NOTEBOOKS

Over time, we are creating a curated collection of ideas and examples using PyTamaro. This GitHub repository contains the sources of the examples as Jupyter notebooks.

You can try them out directly in your browser thanks to Binder.

Eager to try? Check out this exercise about creating flags!

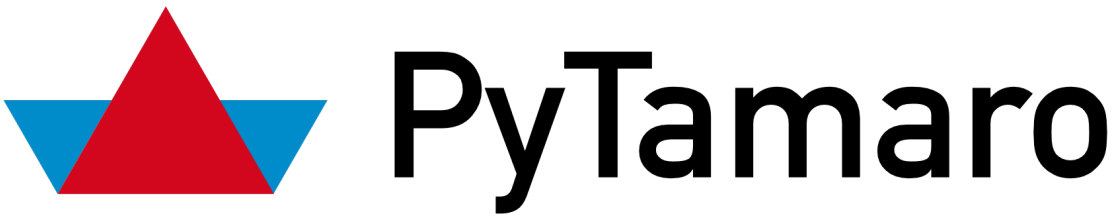
CHAPTER

THREE

API DOCUMENTATION

The API documentation is available in three languages:

3.1 English Version



3.1.1 Primitive shapes and text

Functions to create primitive graphics (shapes and text)

circular_sector(*radius: float*, *angle: float*, *color:* Color) → *Graphic*

Creates a circular sector belonging to a circle of the given radius, filled with a color.

A circular sector is a portion of a circle enclosed between two radii and an arc. Considering a circle as a clock, the first radius is supposed to "point" towards 3 o'clock. The *angle* determines the position of the second radius, computed starting from the first one in the clockwise direction.

Parameters

- **radius** – radius of the circle from which the circular sector is taken, in pixel
- **angle** – central angle, in degrees
- **color** – the color to be used to fill the circular sector

Returns

the specified circular sector as a graphic

`ellipse`(*width: float*, *height: float*, *color:* Color) → *Graphic*

Creates an ellipse with the given width and height, filled with a color.

When width and height are the same, the ellipse becomes a circle with a diameter equal to the provided size.

Parameters

- **`width`** – width of the ellipse, in pixel
- **`height`** – height of the ellipse, in pixel
- **`color`** – the color to be used to fill the circle

Returns
the specified circle as a graphic

`empty_graphic`() → *Graphic*

Creates an empty graphic. When an empty graphic is composed with any other graphic, it behaves as a neutral element: the result is always identical to the other graphic.

An empty graphic cannot be shown nor saved.

Returns
an empty graphic (width and height 0 pixels)

`rectangle`(*width: float*, *height: float*, *color:* Color) → *Graphic*

Creates a rectangle of the given size, filled with a color.

Parameters

- **`width`** – width of the rectangle, in pixel
- **`height`** – height of the rectangle, in pixel
- **`color`** – the color to be used to fill the rectangle

Returns
the specified rectangle as a graphic

`text`(*content: str*, *font: str*, *points: float*, *color:* Color) → *Graphic*

Creates a graphic with the text rendered using the specified font, size and color.

When the indicated True-Type Font is not found in the system, a very basilar font that is always available is used instead. The resulting graphic has the minimal size that still fits the whole text.

Parameters

- **`content`** – the text to render
- **`font`** – the name of the font (e.g., “arial” on Windows, “Arial” on macOS)
- **`points`** – size in typographic points (e.g., 16)
- **`color`** – the color to be used to render the text

Returns
the specified text as a graphic

`triangle`(*side: float*, *color:* Color) → *Graphic*

Creates an equilateral triangle pointing upwards of the given side, filled with a color.

Parameters

- **`side`** – length of the side of the triangle, in pixel
- **`color`** – the color to be used to fill the triangle

Returns
the specified triangle as a graphic

3.1.2 Operations

Functions to do operations on graphics (mainly, to combine them).

`above`(*top_graphic:* Graphic, *bottom_graphic:* Graphic) → *Graphic*

Composes two graphics placing the first on the top and the second on the bottom. The two graphics are aligned horizontally on their centers.

Parameters

- **top_graphic** – graphic to place on the top
- **bottom_graphic** – graphic to place on the bottom

Returns
the resulting graphic after placing the two graphics one above the other

`beside`(*left_graphic:* Graphic, *right_graphic:* Graphic) → *Graphic*

Composes two graphics placing the first on the left and the second on the right. The two graphics are aligned vertically on their centers.

Parameters

- **left_graphic** – graphic to place on the left
- **right_graphic** – graphic to place on the right

Returns
the resulting graphic after placing the two graphics one besides the other

`compose`(*foreground_graphic:* Graphic, *background_graphic:* Graphic) → *Graphic*

Composes two graphics keeping the first one in the foreground and the second one in background, aligning them using their pin positions.

The pin location used to compose becomes the pin location of the resulting graphic.

Parameters

- **foreground_graphic** – graphic to keep in the foreground
- **background_graphic** – graphic to keep in the background

Returns
the resulting graphic after combining the two provided ones

`graphic_height`(*graphic:* Graphic) → int

Returns the height of a graphic, in pixel.

Parameters
graphic – graphic to calculate the height of

Returns
height of the graphic

`graphic_width`(*graphic:* Graphic) → int

Returns the width of a graphic, in pixel.

Parameters
graphic – graphic to calculate the width of

Returns
width of the graphic

`overlay`(*foreground_graphic: Graphic*, *background_graphic: Graphic*) → *Graphic*

Overlays two graphics keeping the first one in the foreground and the second one in background, aligning them on their centers.

Parameters

- **foreground_graphic** – graphic to keep in the foreground
- **background_graphic** – graphic to keep in the background

Returns
the resulting graphic after overlaying the two provided ones

`pin`(*horizontal_place: str*, *vertical_place: str*, *graphic: Graphic*) → *Graphic*

Changes the pinning position of a graphic, returning a new graphic with the same content but with an updated pinning position.

The new pinning position is determined by the parameters *horizontal_place* and *vertical_place*.

Parameters

- **horizontal_place** – one of "left", "middle" or "right" that respectively indicate to move the new pinning positon to the left border, the (horizontal) center, or the right border of the graphic
- **vertical_place** – one of "top", "middle" or "bottom" that respectively indicate to move the new pinning positon to the top border, the (vertical) center, or the bottom border of the graphic
- **graphic** – original graphic

Returns
a new graphic with an updated pinning position

`rotate`(*degrees: float*, *graphic: Graphic*) → *Graphic*

Rotates an graphic by a given amount of degrees counterclockwise around its pinning position.

Small rounding errors (due to approximations to the nearest pixel) may occur.

Parameters

- **degrees** – amount of degrees the graphic needs to be rotated
- **graphic** – the graphic to rotate

Returns
the original graphic rotated around its pinning position

3.1.3 Output

Functions to do I/O with graphics, such as showing or saving them.

`save_gif`(*filename: str*, *graphics: List[Graphic]*, *duration: int = 40*, *loop: bool = True*)

Save a sequence of graphics as an aminated GIF.

Graphics are sequentially reproduced (normally at 25 frames per second) in a loop (unless specificied otherwise).

Parameters

- **filename** – name of the file to create (without the extension)

- **graphics** – list of graphics to be saved as a GIF
- **duration** – duration in milliseconds for each frame (defaults to 40 milliseconds, which leads to 25 frames per second)
- **loop** – whether the GIF should loop indefinitely (defaults to true)

save_graphic(*filename: str*, *graphic:* Graphic, *debug: bool = False*)

Save a graphic as a PNG file. Empty graphics cannot be saved and produce no effect when saved using this function.

When *debug* is *True*, adorns the visualization with useful information for debugging: a red border around the bounding box and a yellowish cross around the pinning position.

Parameters

- **filename** – name of the file to create (without the extension)
- **graphic** – graphic to be saved
- **debug** – can be optionally set to *True* to overlay debugging information

show_graphic(*graphic:* Graphic, *debug: bool = False*)

Show a graphic in a window. Empty graphics cannot be shown and produce no effect when shown using this function.

When *debug* is *True*, adorns the visualization with useful information for debugging: a red border around the bounding box and a yellowish cross around the pinning position.

Parameters

- **graphic** – graphic to be shown
- **debug** – can be optionally set to *True* to overlay debugging information

3.1.4 Colors

Color type, functions to produce colors, and constants for important colors.

class Color

Represents a color. A color also has a degree of opacity, from completely transparent (like the color *transparent*) to completely opaque (like the color *red*).

hsl_color(*hue: float*, *saturation: float*, *lightness: float*, *opacity: float = 1.0*) → *Color*

Returns a color with the provided hue (H), saturation (S), lightness (L) and a certain degree of opacity (alpha, A).

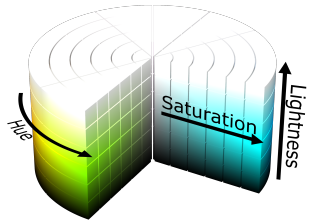


Fig. 1: HSL cylinder: SharkD via Wikimedia Commons

Parameters

- **hue** – hue of the color [0-360]
- **saturation** – saturation of the color [0-1]
- **lightness** – the amount of white or black applied [0-1]. Fully saturated colors have a lightness value of 1/2.
- **opacity** – opacity (alpha) of the color, where 0 means fully transparent and 1 fully opaque. By default, all colors are fully opaque.

Returns
a color with the provided HSLA components

hsv_color(*hue: float*, *saturation: float*, *value: float*, *opacity: float = 1.0*) → *Color*

Returns a color with the provided hue (H), saturation (S), value (V) and a certain degree of opacity (alpha, A).

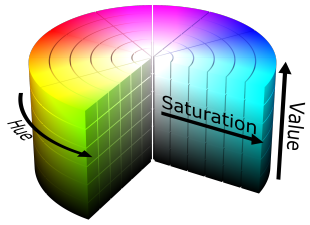


Fig. 2: HSV cylinder (SharkD via Wikimedia Commons)

Parameters

- **hue** – hue of the color [0-360]
- **saturation** – saturation of the color [0-1]
- **value** – the amount of light that is applied [0-1]
- **opacity** – opacity (alpha) of the color, where 0 means fully transparent and 1 fully opaque. By default, all colors are fully opaque.

Returns
a color with the provided HSVA components.

rgb_color(*red: int*, *green: int*, *blue: int*, *opacity: float = 1.0*) → *Color*

Returns a color with the provided components for red (R), green (G) and blue (B) and a certain degree of opacity (alpha, A).

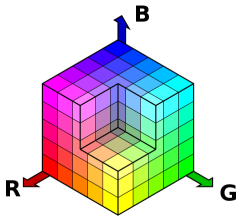


Fig. 3: RGB cube (SharkD via Wikimedia Commons)

Parameters

- **red** – red component [0-255]

- **green** – green component [0-255]
- **blue** – blue component [0-255]
- **opacity** – opacity (alpha) of the color, where 0 means fully transparent and 1 fully opaque. By default, all colors are fully opaque.

Returns
a color with the provided RGBA components

Names of notable colors because they are at the vertices of the RGB cube, plus the fully-transparent one.

black
Black color

blue
Blue color

cyan
Cyan color

green
Green color

magenta
Magenta color

red
Red color

transparent
Fully-transparent color

white
White color

yellow
Yellow color

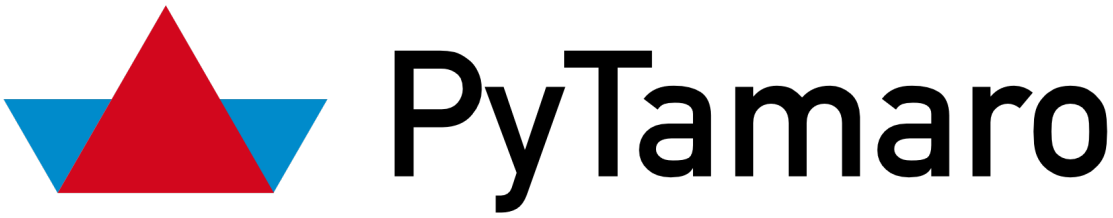
Welcome to PyTamaro's documentation! Use the menu on the left to browse the available functions.

class Graphic
A graphic (image) with a position for pinning.

The pinning position is used in the following operations:

- rotation (to determine the center of rotation)
- graphic composition (two graphics get composed aligning their pinning position).

3.2 Versione Italiana



3.2.1 Forme primitive e testo

Funzioni per creare grafiche primitive (forme e testo)

ellisse(*larghezza: float*, *altezza: float*, *colore: Colore*) → Grafica

Crea un ellisse delle dimensioni indicate, riempito con un colore.

Quando larghezza e altezza coincidono, l'ellisse diventa un cerchio di diametro pari alla dimensione indicata.

Parameters

- **larghezza** – larghezza dell'ellisse, in pixel
- **altezza** – altezza dell'ellisse, in pixel
- **colore** – colore da usare per riempire l'ellisse

Returns

una grafica con l'ellisse specificato

grafica_vuota() → Grafica

Crea una grafica vuota. Quando una grafica vuota viene composta con ogni altra grafica, si comporta da elemento neutro: il risultato è sempre uguale all'altra grafica.

Una grafica vuota non può essere visualizzata né salvata.

Returns

una grafica vuota (larghezza e altezza 0 pixel)

rettangolo(*larghezza: float*, *altezza: float*, *colore: Colore*) → Grafica

Crea un rettangolo delle dimensioni indicate, riempito con un colore.

Parameters

- **larghezza** – larghezza del rettangolo, in pixel
- **altezza** – altezza del rettangolo, in pixel
- **colore** – colore da usare per riempire il rettangolo

Returns
una grafica con il rettangolo specificato

`settore_circolare`(*raggio: float*, *angolo: float*, *colore: Colore*) → Grafica

Crea un settore circolare appartenente a un cerchio del raggio indicato, riempito con un colore.

Un settore circolare è una porzione di cerchio racchiusa tra due raggi e un arco. Considerando il cerchio come un orologio, il primo raggio "punta" in direzione delle ore 3. L'*angolo* determina la posizione del secondo raggio, calcolata a partire dalla posizione del primo in senso orario.

Parameters

- **raggio** – raggio del cerchio da cui è preso il settore circolare, in pixel
- **angolo** – angolo al centro, in gradi
- **colore** – colore da usare per riempire il settore circolare

Returns
una grafica con il settore circolare specificato

`testo`(*contenuto: str*, *font: str*, *punti: float*, *colore: Colore*) → Grafica

Crea una grafica con il testo renderizzato usando font, dimensione e colore indicati.

Quando il font True-Type indicato non è disponibile nel sistema, al suo posto viene usato un font estremamente basilare e sempre disponibile. La grafica risultante ha la dimensione minima in modo da racchiudere l'intero testo.

Parameters

- **contenuto** – il testo di cui fare rendering
- **font** – il nome del font (ad esempio "arial" su Windows, "Arial" su macOS)
- **punti** – dimensione in punti tipografici (ad esempio 16)
- **colore** – colore da usare per fare il rendering del testo

Returns
una grafica con il testo specificato

`triangolo`(*lato: float*, *colore: Colore*) → Grafica

Crea un triangolo equilatero del lato indicato con la punta verso l'alto, riempito con un colore.

Parameters

- **lato** – lunghezza del lato del triangolo, in pixel
- **colore** – colore da usare per riempire il triangolo

Returns
una grafica con il triangolo specificato

3.2.2 Operazioni

Funzioni per operazioni con grafiche (principlamente per combinarle).

accanto(*grafica_sinistra: Grafica*, *grafica_destra: Grafica*) → Grafica

Compone due grafiche affiancandole, posizionando la prima a sinistra e la seconda a destra. Le due grafiche vengono allineate verticalmente al centro.

Parameters

- **grafica_sinistra** – grafica da posizionare a sinistra
- **grafica_destra** – grafica da posizionare a destra

Returns
grafica risultante dall'affiancamento orizzontale delle due grafiche fornite

altezza_grafica(*grafica: Grafica*) → int

Ritorna l'altezza di una grafica, in pixel.

Parameters
grafica – grafica di cui calcolare l'altezza

Returns
altezza della grafica

componi(*grafica_primopiano: Grafica*, *grafica_secondopiano: Grafica*) → Grafica

Compone due grafiche tenendo la prima in primo piano e la seconda sullo sfondo, allineandole usando le loro posizioni di fissaggio.

La posizione di fissaggio usata per comporre diventa la posizione di fissaggio della grafica risultante.

Parameters

- **grafica_primopiano** – grafica da tenere in primo piano
- **grafica_secondopiano** – grafica da tenere sullo sfondo

Returns
grafica risultante dalla composizione delle due fornite

fissa(*posizione_orizzontale: str*, *posizione_verticale: str*, *grafica: Grafica*) → Grafica

Cambia la posizione di fissaggio di una grafica, ritornando una nuova grafica con lo stesso contenuto ma una posizione di fissaggio aggiornata.

La nuova posizione di fissaggio è determinata dai parametri *posizione_orizzontale* e *posizione_orizzontale*.

Parameters

- **posizione_orizzontale** – uno tra "sinistra", "centro", "destra" per muovere la nuova posizione di fissaggio rispettivamente al bordo sinistro, al centro (orizzontalmente), o al bordo destro della grafica
- **posizione_verticale** – uno tra "alto", "centro", "basso" per muovere la nuova posizione di fissaggio rispettivamente al bordo superiore, al centro (verticalmente), o al bordo inferiore della grafica
- **grafica** – grafica originale

Returns
una nuova grafica con una posizione di fissaggio aggiornata

larghezza_grafica(*grafica: Grafica*) → int

Ritorna la larghezza di una grafica, in pixel.

Parameters
grafica – grafica di cui calcolare la larghezza

Returns
larghezza della grafica

ruota(*gradi: float*, *grafica: Grafica*) → Grafica

Ruota una grafica di un certo numero di gradi in senso antiorario attorno alla sua posizione di fissaggio.

È possibile che si verifichino piccoli errori di arrotondamento (a causa dell'approssimazione al pixel più vicino).

Parameters

- **gradi** – numero di gradi di cui ruotare la grafica
- **grafica** – grafica da ruotare

Returns
la grafica originale ruotata attorno alla sua posizione di fissaggio

sopra(*grafica_alto: Grafica*, *grafica_basso: Grafica*) → Grafica

Compone due grafiche affiancandole verticalmente, posizionando la prima in alto e la seconda in basso. Le due grafiche vengono allineate orizzontalmente al centro.

Parameters

- **grafica_sopra** – grafica da posizionare sopra
- **grafica_sotto** – grafica da posizionare sotto

Returns
grafica risultante dall'affiancamento verticale delle due grafiche fornite

sovrapponi(*grafica_primopiano: Grafica*, *grafica_secondopiano: Grafica*) → Grafica

Sovrappone due grafiche tenendo la prima in primo piano e la seconda sullo sfondo, allineandole sui loro centri.

Parameters

- **grafica_primopiano** – grafica da tenere in primo piano
- **grafica_secondopiano** – grafica da tenere sullo sfondo

Returns
grafica risultante dalla sovrapposizione delle due fornite

3.2.3 Output

Funzioni per I/O con grafiche, come visualizzarle oppure salvarle.

salva_gif(*nome_file: str*, *grafiche: list[Grafica]*, *durata: int = 40*, *loop: bool = True*)

Salva una sequenza di grafiche come una GIF animata.

Le grafiche vengono riprodotte sequenzialmente (normalmente a 25 frame al secondo) a ciclo continuo.

Parameters

- **nome_file** – nome del file da creare (senza estensione)
- **grafiche** – lista di grafiche da salvare come GIF

- **durata** – durata in millisecondi di ciascun frame (default a 40 millisecondi, ovvero 25 frame al secondo)
- **loop** – determina se la GIF debba riprodursi in loop indefinitamente (default a True)

salva_grafica(*nome_file: str*, *grafica: Grafica*, *debug: bool = False*)

Salva una grafica come file PNG.

Una grafica vuota non può essere salvata; quindi chiamare questa funzione con essa non produce alcun affetto.

Quando *debug* è *True*, adorna la visualizzazione con informazioni utili per debugging: un bordo rosso attorno alla bounding box e una croce giallastra attorno al punto di fissaggio.

Parameters

- **nome_file** – nome del file da creare (senza estensione)
- **grafica** – grafica da visualizzare
- **debug** – può facoltativamente essere impostato a *True* per sovrapporre informazioni di debug

visualizza_grafica(*grafica: Grafica*, *debug: bool = False*)

Visualizza una grafica in una nuova finestra.

Una grafica vuota non può essere mostrata; quindi chiamare questa funzione con essa non produce alcun affetto.

Quando *debug* è *True*, adorna la visualizzazione con informazioni utili per debugging: un bordo rosso attorno alla bounding box e una croce giallastra attorno al punto di fissaggio.

Parameters

- **grafica** – grafica da visualizzare
- **debug** – può facoltativamente essere impostato a *True* per sovrapporre informazioni di debug

3.2.4 Colori

Tipo *Colore*, funzioni per produrre colori e costanti per colori notevoli.

Colore

Rappresenta un colore. Un colore ha anche un grado di opacità, da completamente trasparente (come il colore *trasparente*) a completamente opaco (come il colore *rosso*).

colore_hsl(*tonalita: float*, *saturazione: float*, *luce: float*, *opacita: float = 1.0*) → Colore

Ritorna un colore con la tonalità (H), saturazione (S) e luce (L) indicati, e un certo grado di opacità (alpha, A).

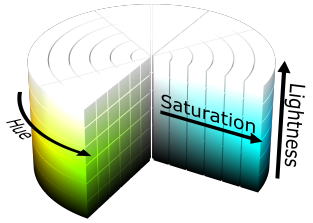


Fig. 4: Cilindro HSL: SharkD via Wikimedia Commons

Parameters

- **tonalita** – tonalità del colore [0-360]

- **saturazione** – saturazione del colore [0-1]
- **luce** – quantità di bianco o nero applicata [0-1]. Colori completamente saturi hanno un valore di luce di 1/2.
- **opacita** – opacità (alpha) del colore, dove 0 significa completamente trasparente e 1 completamente opaco. Di default, tutti i colori sono completamente opachi.

Returns
un colore con le componenti HSLA indicate

colore_hsv(*tonalita: float*, *saturazione: float*, *valore: float*, *opacita: float = 1.0*) → Colore

Ritorna un colore con la tonalità (H), saturazione (S) e valore (V) indicati, e un certo grado di opacità (alpha, A).

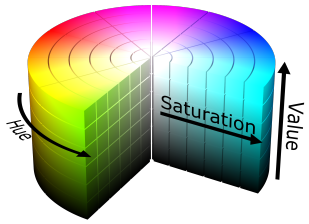


Fig. 5: Cilindro HSV (SharkD via Wikimedia Commons)

Parameters

- **tonalita** – tonalità del colore [0-360]
- **saturazione** – saturazione del colore [0-1]
- **valore** – quantità di luce applicata [0-1]
- **opacita** – opacità (alpha) del colore, dove 0 significa completamente trasparente e 1 completamente opaco. Di default, tutti i colori sono completamente opachi.

Returns
un colore con le componenti HSVA indicate

colore_rgb(*rosso: int*, *verde: int*, *blu: int*, *opacita: float = 1.0*) → Colore

Ritorna un colore con le componenti indicate per rosso (R), verde (G) e blu (B) e un certo grado di opacità (alpha, A).

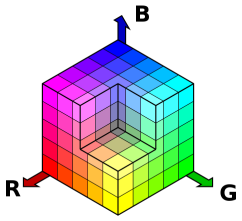


Fig. 6: Cubo RGB (SharkD via Wikimedia Commons)

Parameters

- **rosso** – componente rosso [0-255]
- **verde** – componente verde [0-255]

- **blu** – componente blu [0-255]
- **opacita** – opacità (alpha) del colore, dove 0 significa completamente trasparente e 1 completamente opaco. Di default, tutti i colori sono completamente opachi.

Returns
un colore con le componenti RGBA indicate

Nomi di colori notevoli essendo ai vertici del cubo RGB. In aggiunta, il colore completamente trasparente.

`bianco: Colore`
Colore bianco

`blu: Colore`
Colore blu

`ciano: Colore`
Colore ciano

`giallo: Colore`
Colore giallo

`magenta: Colore`
Colore magenta

`nero: Colore`
Colore nero

`rosso: Colore`
Colore rosso

`trasparente: Colore`
Colore completamente trasparente

`verde: Colore`
Colore verde

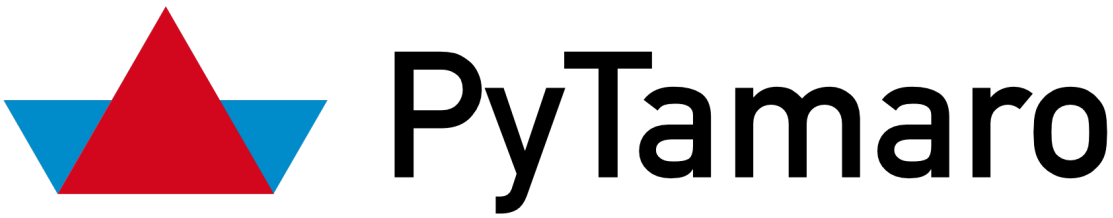
Benvenuti nella documentazione di Pytamaro! Esplorate le funzioni disponibile usando il menu a sinistra.

`Grafica`
Una grafica (immagine) con una posizione per fissare.

La posizione per fissare viene usata nelle seguenti operazioni:

- rotazione (per determinare il centro di rotazione)
- composizione di grafiche (due grafiche vengono composte allineando le loro posizioni di fissaggio).

3.3 Deutsche Version



3.3.1 Primitive Grafiken

Funktionen zum Erzeugen primitiver Grafiken (Figuren und Texte)

dreieck(*seite: float*, *farbe: Farbe*) → Grafik

Erzeugt ein gleichseitiges Dreieck mit der gegebenen Seitenlänge und einer nach oben zeigenden Ecke, gefüllt in der gegebenen Farbe.

Parameters

- **seite** – Seitenlänge des Dreiecks, in Pixel
- **farbe** – Füllfarbe des Dreiecks

Returns
eine Grafik mit dem gegebenen Dreieck

ellipse(*breite: float*, *hoehe: float*, *farbe: Farbe*) → Grafik

Erzeugt eine Ellipse mit der gegebenen Breite und Höhe, gefüllt in der gegebenen Farbe.

Wenn Breite und Höhe gleich gross sind wird die Ellipse zum Kreis mit dem entsprechenden Durchmesser.

Parameters

- **breite** – Breite der Ellipse, in Pixel
- **hoehe** – Höhe der Ellipse, in Pixel
- **farbe** – Füllfarbe der Ellipse

Returns
eine Grafik mit dem gegebenen Rechteck

kreis_sektor(*radius: float*, *winkel: float*, *farbe: Farbe*) → Grafik

Erzeugt einen Kreissektor mit dem gegebenen Radius, der den gegebenen Winkel umspannt, gefüllt in der gegebenen Farbe.

Ein Kreissektor ist ein Teil eines Kreises begrenzt durch zwei Radien und einen Bogen. Wenn man den Kreis als Uhr betrachtet dann zeigt der erste Radius in Richtung 3 Uhr. Der Winkel bestimmt die Position des zweiten Radius, ausgehend vom ersten Radius im Uhrzeigersinn.

Parameters

- **radius** – Kreisradius, in Pixel

- **winkel** – Winkel des Sektors, in Grad
- **farbe** – Füllfarbe des Kreissektors

Returns
eine Grafik mit dem gegebenen Kreissektor

leere_grafik() → Grafik

Erzeugt eine leere Grafik. Wenn eine leere Grafik mit einer anderen Grafik kombiniert wird verhält sie sich als neutrales Element: das Ergebnis der Komposition ist einfach gleich der anderen Grafik.

Eine leere Grafik kann weder angezeigt noch gespeichert werden.

Returns
eine leere Grafik (Breite und Höhe sind 0 Pixel)

rechteck(*breite: float*, *hoehe: float*, *farbe: Farbe*) → Grafik

Erzeugt ein Rechteck mit der gegebenen Breite und Höhe, gefüllt in der gegebenen Farbe.

Parameters

- **breite** – die Breite des Rechtecks, in Pixel
- **hoehe** – die Höhe des Rechtecks, in Pixel
- **farbe** – Füllfarbe des Rechtecks

Returns
eine Grafik mit dem gegebenen Rechteck

text(*inhalt: str*, *schriftart: str*, *punkte: float*, *farbe: Farbe*) → Grafik

Erzeugt einen Text in der gegebenen Schriftart und Schriftgrösse, gefüllt in der gegebenen Farbe.

Falls für die gegebene Schriftart auf dem System keine True-Type Schrift zur Verfügung steht, wird eine einfache Standardschriftart verwendet. Die resultierende Grafik hat die minimale Grösse, die den gesamten Text umschliesst.

Parameters

- **inhalt** – der Text, der dargestellt werden soll
- **schriftart** – der Name der Schriftart (zum Beispiel “arial” auf Windows, “Arial” auf macOS)
- **punkte** – Schriftgrösse in typografischen Punkten (zum Beispiel 16)
- **farbe** – Farbe, in der der Text dargestellt werden soll

Returns
eine Grafik bestehend aus dem gegebenen Text

3.3.2 Operationen

Funktionen für Operationen mit Grafiken (hauptsächlich für deren Komposition).

drehe(*grad: float*, *grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die einer Rotation der gegebenen Grafik um ihre Fixierungsposition im Gegenuhrzeigersinn um den gegebenen Winkel entspricht.

Es kann wegen der Approximation auf die nächstgelegenen Pixel zu kleinen Rundungsfehlern kommen.

Parameters

- **grad** – Drehwinkel, in Grad im Gegenuhrzeigersinn

- **grafik** – zu rotierende Grafik

Returns
die neue, rotierte Grafik

fixiere(*horizontale_position: str*, *vertikale_position: str*, *grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die der gegebenen Grafik mit einer anderen Fixierungsposition entspricht.

Die neue Fixierungsposition wird mit den Parametern *horizontale_position* und *vertikale_position* bestimmt.

Parameters

- **horizontale_position** – "links", "mitte" oder "rechts" um die Fixierungsposition auf den linken Rand, in die Mitte, oder auf den rechten Rand der Grafik zu setzen.
- **vertikale_position** – "oben", "mitte" oder "unten" um die Fixierungsposition auf den oberen Rand, in die Mitte, oder auf den unteren Rand der Grafik zu setzen.
- **grafik** – die ursprüngliche Grafik

Returns
die neue Grafik mit der gegebenen Fixierungsposition

grafik_breite(*grafik: Grafik*) → int

Gibt die Breite (in Pixel) der gegebenen Grafik zurück.

Parameters
grafik – Grafik deren Breite gesucht ist

Returns
Breite der Grafik

grafik_hoehe(*grafik: Grafik*) → int

Gibt die Höhe (in Pixel) der gegebenen Grafik zurück.

Parameters
grafik – Grafik deren Höhe gesucht ist

Returns
Höhe der Grafik

kombiniere(*vordere_grafik: Grafik*, *hintere_grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die aus der Kombination der zwei gegebenen Grafiken besteht. Die erste gegebene Grafik liegt im Vordergrund und die zweite im Hintergrund. Die Grafiken werden so ausgerichtet, dass ihre Fixierungspositionen übereinanderliegen.

Die überlappenden Fixierungspositionen werden zur Fixierungsposition der resultierenden Grafik.

Parameters

- **vordere_grafik** – Grafik im Vordergrund
- **hintere_grafik** – Grafik im Hintergrund

Returns
die zusammengesetzte Grafik

neben(*linke_grafik: Grafik*, *rechte_grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die aus dem Nebeneinanderlegen der zwei gegebenen Grafiken besteht. Die zwei Grafiken sind vertikal zentriert.

Parameters

- **linke_grafik** – linke Grafik (im Westen)

- **rechte_grafik** – rechte Grafik (im Osten)

Returns
die zusammengesetzte Grafik

ueber(*obere_grafik: Grafik*, *untere_grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die aus dem Übereinanderlegen der zwei gegebenen Grafiken besteht. Die zwei Grafiken sind horizontal zentriert.

Parameters

- **obere_grafik** – obere Grafik (im Norden)
- **untere_grafik** – untere Grafik (im Süden)

Returns
die zusammengesetzte Grafik

ueberlagere(*vordere_grafik: Grafik*, *hintere_grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die aus der zentrierten Überlagerung der zwei gegebenen Grafiken besteht. Die erste gegebene Grafik liegt im Vordergrund und die zweite im Hintergrund, und ihre Fixierungspositionen liegen übereinander im Zentrum.

Parameters

- **vordere_grafik** – Grafik im Vordergrund
- **hintere_grafik** – Grafik im Hintergrund

Returns
die zusammengesetzte Grafik

3.3.3 Ausgabe

Funktionen zur Ausgabe (Anzeigen oder Speichern) von Grafiken.

speichere_gif(*datei_name: str*, *grafiken: list[Grafik]*, *dauer: int = 40*, *loop: bool = True*)

Speichere die gegebene Sequenz von Grafiken als animierte GIF-Datei.

Beim Anzeigen des GIFs werden die Grafiken in einer unendlichen Schleife animiert (normalerweise mit 25 Grafiken pro Sekunde).

Parameters

- **datei_name** – Name der zu kreierenden Datei (ohne Erweiterung)
- **grafiken** – Liste der zu speichernden Grafiken
- **dauer** – Dauer jeder Grafik, in Millisekunden (Default: 40 millisekunden, enspricht 25 Grafiken pro Sekunde)
- **loop** – bestimmt ob das GIF in einer unendlichen Schleife abgespielt werden soll (Default: true)

speichere_grafik(*datei_name: str*, *grafik: Grafik*, *debug: bool = False*)

Speichere die gegebene Grafik als PNG-Datei.

Eine leere Grafik kann nicht gespeichert werden; deshalb hat der Aufruf dieser Funktion mit einer leeren Grafik keinen Effekt.

Falls *debug True* ist werden auf der Grafik zusätzliche Informationen dargestellt, welche für das Debugging nützlich sein können. Ein roter Rahmen markiert die Bounding Box der Grafik, und ein gelbliches Kreuz gibt den Fixierpunkt an.

Parameters

- **datei_name** – Name der zu kreierenden Datei (ohne Erweiterung)
- **grafik** – zu speichernde Grafik
- **debug** – kann optional auf *True* gesetzt werden, um über der Grafik Debug-Informationen darzustellen

zeige_grafik(*grafik: Grafik*, *debug: bool = False*)

Zeige die gegebene Grafik in einem neuen Fenster an.

Eine leere Grafik kann nicht angezeigt werden; deshalb hat der Aufruf dieser Funktion mit einer leeren Grafik keinen Effekt.

Falls *debug True* ist werden auf der Grafik zusätzliche Informationen dargestellt, welche für das Debugging nützlich sein können. Ein roter Rahmen markiert die Bounding Box der Grafik, und ein gelbliches Kreuz gibt den Fixierpunkt an.

Parameters

- **grafik** – die anzuzeigende Grafik
- **debug** – kann optional auf *True* gesetzt werden, um über der Grafik Debug-Informationen darzustellen

3.3.4 Farben

Der Typ *Farbe*, Funktionen, die Farben erzeugen und Konstanten für wichtige Farben.

Farbe

Repräsentiert eine Farbe. Eine Farbe hat auch eine gewisse Opazität, von komplett durchsichtig (wie die Farbe *transparent*), bis komplett undurchsichtig (wie die Farbe *rot*).

hsl_farbe(*farbton: float*, *saettigung: float*, *helligkeit: float*, *opazitaet: float = 1.0*) → Farbe

Erzeugt eine Farbe mit dem gegebenen Farbton (H), der Sättigung (S), dem Helligkeit (L) und der Opazität (Undurchsichtigkeit, alpha, A).

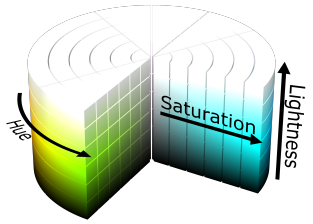


Fig. 7: HSL Zylinder: SharkD via Wikimedia Commons

Parameters

- **farbton** – der Farbton (hue) [0-360] als Farbwinkel, in Grad, auf dem Farbkreis (0 für Rot, 120 für Grün, 240 für Blau)
- **saettigung** – Farbsättigung (saturation) [0-1] (0 = Grau, 0.5 = wenig gesättigte Farbe, 1 = gesättigte, reine Farbe)
- **helligkeit** – der Anteil Schwarz oder Weiss [0-1]. (0 = Schwarz, 0.5 = weder abgedunkelt noch aufgehellt, 1 = Weiss)

- **opazitaet** – die Undurchsichtigkeit (alpha), wobei 0 komplett durchsichtig und 1 komplett undurchsichtig entspricht. Standardmäßig sind alle Farben vollständig undurchsichtig.

Returns
eine Farbe mit den gegebenen HSLA-Komponenten

hsv_farbe(*farbton: float*, *saettigung: float*, *hellwert: float*, *opazitaet: float = 1.0*) → Farbe

Erzeugt eine Farbe mit dem gegebenen Farbton (H), der Sättigung (S), dem Hellwert (V) und der Opazität (Undurchsichtigkeit, alpha, A).

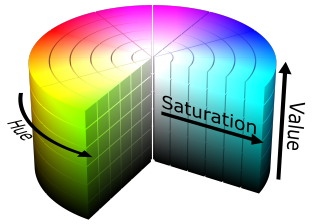


Fig. 8: HSV Zylinder (SharkD via Wikimedia Commons)

Parameters

- **farbton** – der Farbton (hue) [0-360] als Farbwinkel, in Grad, auf dem Farbkreis (0 für Rot, 120 für Grün, 240 für Blau)
- **saettigung** – Farbsättigung (saturation) [0-1] (0 = Grau, 0.5 = wenig gesättigte Farbe, 1 = gesättigte, reine Farbe)
- **hellwert** – Hellwert (value) der Farbe [0-1] (0 = dunkel, 1 = hell)
- **opazitaet** – die Undurchsichtigkeit (alpha), wobei 0 komplett durchsichtig und 1 komplett undurchsichtig entspricht. Standardmäßig sind alle Farben vollständig undurchsichtig.

Returns
eine Farbe mit den gegebenen HSVA-Komponenten

rgb_farbe(*rot: int*, *gruen: int*, *blau: int*, *opazitaet: float = 1.0*) → Farbe

Erzeugt eine Farbe mit den gegebenen Anteilen Rot (R), Grün (G) und Blau (B) und der gegebenen Opazität (Undurchsichtigkeit, alpha, A).

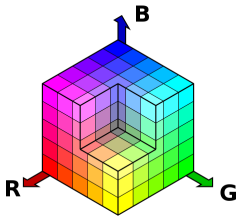


Fig. 9: RGB Würfel (SharkD via Wikimedia Commons)

Parameters

- **rot** – der rote Farbanteil [0-255]
- **gruen** – der grüne Farbanteil [0-255]

- **blau** – der blaue Farbanteil [0-255]
- **opazitaet** – die Undurchsichtigkeit (alpha), wobei 0 komplett durchsichtig und 1 komplett undurchsichtig entspricht. Standardmäßig sind alle Farben vollständig undurchsichtig.

Returns
eine Farbe mit den gegebenen RGBA-Komponenten

Die folgenden nennenswerten Farben sind bereits definiert: Farben an den Ecken des RGB Würfels, komplett durchsichtige Farbe.

blau: Farbe
Die Farbe Blau.

cyan: Farbe
Die Farbe Cyan.

gelb: Farbe
Die Farbe Gelb.

gruen: Farbe
Die Farbe Grün.

magenta: Farbe
Die Farbe Magenta.

rot: Farbe
Die Farbe Rot.

schwarz: Farbe
Die Farbe Schwarz.

transparent: Farbe
Die komplett durchsichtige Farbe.

weiss: Farbe
Die Farbe Weiss.

Willkommen in der Pytamaro Dokumentation! Erkunden Sie die verfügbaren Funktionen mit dem Menu auf der linken Seite.

Grafik
Eine Grafik mit einer Fixierposition.

Die Fixierposition wird von den folgenden Operationen verwendet:

- *drehe* (das Rotationszentrum ist die Fixierposition)
- *kombiniere* (die zwei Grafiken werden so ausgerichtet, dass ihre Fixierpositionen übereinanderliegen).

PYTHON MODULE INDEX

p

INDEX

N

O

P

R

S

T

U

V

W

Y

Z