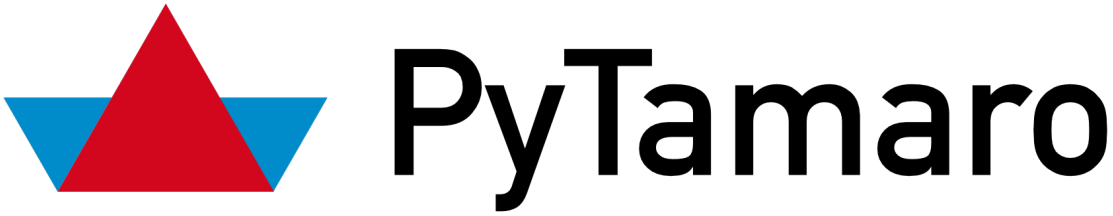
PyTamaro

Release 0.6.1

LuCE Research Lab

Dec 05, 2023

CONTENTS



This site contains the API documentation of the PyTamaro Python library.

For more information about the library, including examples of activities and installation instructions, please visit the project's website at https://pytamaro.si.usi.ch.

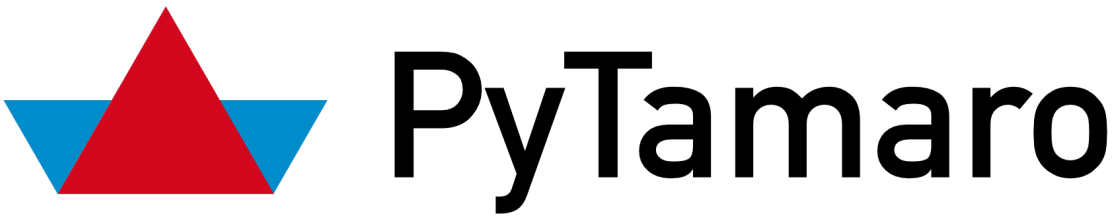
CHAPTER

ONE

API DOCUMENTATION

The API documentation is available in four languages:

1.1 English Version



1.1.1 Primitive shapes and text

Functions to create primitive graphics (shapes and text). Unless specified otherwise, the initial pinning position is at the center of the graphic's bounding box.

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 counterclockwise direction. An angle of 360 degrees corresponds to a full circle.

The pinning position is at the center of the circle from which the circular sector is taken.

Parameters

- **radius** – radius of the circle from which the circular sector is taken
- **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
- **`height`** – height of the ellipse
- **`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.

Returns
an empty graphic (width and height 0)

`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
- **`height`** – height of the rectangle
- **`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.

The pinning position is horizontally aligned on the left and vertically on the baseline of the 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`(*side1: float*, *side2: float*, *angle: float*, *color:* Color) → *Graphic*

Creates a triangle specifying two sides and the angle between them, filled with a color. The first side extends horizontally to the right. The angle specifies how much the second side is rotated, counterclockwise, from the first one.

For all triangles, except obtuse ones, the bottom-left corner of the resulting graphic concides with the vertex of the triangle for which the angle is specified.

The pinning position is the centroid of the triangle.

Parameters

- **side1** – length of the first, horizontal side of the triangle
- **side2** – length of the second side of the triangle
- **angle** – angle between the two sides, in degrees
- **color** – the color to be used to fill the triangle

Returns
the specified triangle as a graphic

1.1.2 Operations

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

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

Creates a new graphic by placing the two graphics one above the other. The two graphics are horizontally centered.

The pinning position of the new graphic is at its center.

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*

Creates a new graphic by placing the two graphics one besides the other. The two graphics are vertically centered.

The pinning position of the new graphic is at its center.

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*

Creates a new graphic by composing the two provided graphics. The first graphic is kept in the foreground, the second one in the background. The graphics are aligned by superimposing their pinning positions.

The pinning position used to compose becomes the pinning position of the resulting graphic.

Parameters

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

Returns
the resulting composed graphic

graphic_height(*graphic:* Graphic) → int

Returns the height of a graphic.

Parameters
graphic – graphic to calculate the height of

Returns
height of the graphic

graphic_width(*graphic:* Graphic) → int

Returns the width of a graphic.

Parameters
graphic – graphic to calculate the width of

Returns
width of the graphic

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

Creates a new graphic by overlaying the two provided graphics, keeping the first one in the foreground and the second one in background. The two graphics are overlaid on their centers.

The pinning position of the new graphic is at its center.

Parameters

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

Returns
the resulting graphic after overlaying the two provided ones

pin(*point:* Point, *graphic:* Graphic) → *Graphic*

Creates a new graphic that corresponds to the provided graphic, with a new pinning position.

Each graphic is contained in a rectangular bounding box. There are 9 notable points, corresponding to the four corners of this rectangle, the middle points of the four edges and the center of the rectangle. These points can be referred to using these names: *top_left*, *top_right*, *bottom_left*, *bottom_right*, *top_center*, *center_right*, *bottom_center*, *center_left* and *center*.

Parameters

- **point** – the point indicating the new pinning position
- **graphic** – original graphic

Returns
a new graphic with the specified pinning position

rotate(*angle: float*, *graphic:* Graphic) → *Graphic*

Creates a new graphic by rotating counterclockwise the provided graphic around its pinning position by the given angle. A negative angle corresponds to a clockwise rotation.

Parameters

- **angle** – angle of counterclockwise rotation, in degrees
- **graphic** – the graphic to rotate

Returns
a new, rotated graphic

1.1.3 Output

Functions for output (show or save) of graphics.

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

Save a sequence of graphics as an animation (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, including the extension '.gif'
- **graphics** – list of graphics to be saved as an animation
- **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 to a file. Two file formats are supported: PNG (raster graphics) and SVG (vector graphics). The extension of the filename (either ".png" or ".svg") determines the format.

Graphics with no area cannot be saved in the PNG format.

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 (with the extension)
- **graphic** – graphic to be saved
- **debug** – can be optionally set to *True* to overlay debugging information

show_animation(*graphics: List[*Graphic*]*, *duration: int = 40*, *loop: bool = True*)

Show a sequence of graphics as an animation (GIF).

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

Parameters

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

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

Show a graphic. Graphics with no area cannot be shown.

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

1.1.4 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).

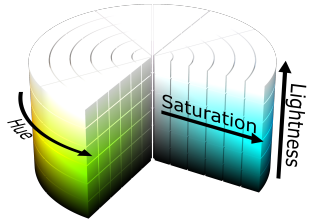


Fig. 1: HSL cylinder: SharkD via Wikimedia Commons

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).

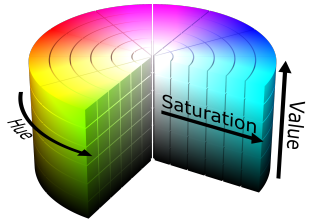


Fig. 2: HSV cylinder (SharkD via Wikimedia Commons)

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).

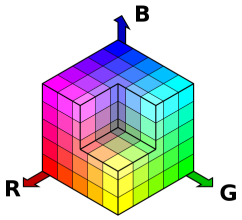


Fig. 3: RGB cube (SharkD via Wikimedia Commons)

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

1.1.5 Points

Point type.

class Point
Represents a point on a plane.

Names of notable points, that can be used as pinning positions for a graphic.

bottom_center: *Point*
The middle point of the bottom edge of the bounding box

bottom_left: *Point*
The bottom left corner of the bounding box

bottom_right: *Point*
The bottom right corner of the bounding box

center: *Point*
The center point of the bounding box

center_left: *Point*
The middle point of the left edge of the bounding box

center_right: *Point*
The middle point of the right edge of the bounding box

top_center: *Point*
The middle point of the top edge of the bounding box

top_left: *Point*
The top left corner of the bounding box

top_right: *Point*
The top right corner of the bounding box

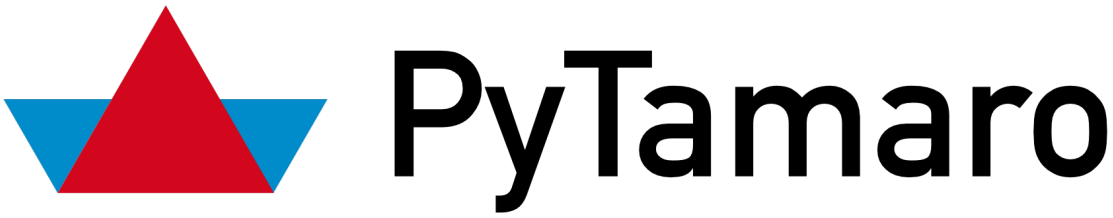
Welcome to PyTamaro's documentation! Use the menu on the left to browse the available functions.

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).

1.2 Versione Italiana



1.2.1 Forme primitive e testo

Funzioni per creare grafiche primitive (forme e testo). Tranne quando specificato diversamente, la posizione di fissaggio iniziale è al centro del rettangolo di delimitazione (bounding box) della grafica.

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
- **altezza** – altezza dell'ellisse
- **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.

Returns
una grafica vuota (larghezza e altezza 0)

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

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

Parameters

- **larghezza** – larghezza del rettangolo
- **altezza** – altezza del rettangolo
- **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 antiorario. Un angolo di 360 gradi corrisponde a un cerchio completo.

La posizione di fissaggio è al centro del cerchio da cui è preso il settore circolare.

Parameters

- **raggio** – raggio del cerchio da cui è preso il settore circolare
- **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.

La posizione di fissaggio è allineata orizzontalmente a sinistra e verticalmente sulla linea di base (baseline) del 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(*lato1: float*, *lato2: float*, *angolo: float*, *colore: Colore*) → Grafica

Crea un triangolo specificando due lati e l’angolo tra essi compreso, riempito con un colore. Il primo lato si estende orizzontalmente verso destra. L’angolo specifica di quanto il secondo lato è ruotato, in senso antiorario, rispetto al primo.

Per tutti i triangoli, eccetto quelli ottusi, il punto in basso a sinistra della grafica risultante coincide con il vertice del triangolo di cui si è specificato l’angolo.

La posizione di fissaggio è il centroide del triangolo.

Parameters

- **lato1** – lunghezza del primo lato (orizzontale) del triangolo
- **lato2** – lunghezza del secondo lato del triangolo
- **angolo** – angolo compreso tra i due lati, in gradi
- **colore** – colore da usare per riempire il triangolo

Returns
una grafica con il triangolo specificato

1.2.2 Operazioni

Funzioni per operazioni con grafiche (principlamente per combinarle).

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

Crea una nuova grafica affiancando orizzontalmente le due grafiche fornite. Le due grafiche vengono centrate verticalmente.

La posizione di fissaggio della grafica risultante è nel suo 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.

Parameters
grafica – grafica di cui calcolare l’altezza

Returns
altezza della grafica

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

Crea una nuova grafica componendo le due grafiche fornite. La prima grafica viene tenuta in primo piano, la seconda sullo sfondo. Le grafiche vengono allineate superimponendo le loro posizioni di fissaggio.

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

Parameters

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

Returns
la grafica risultante composta

fissa(*punto: Point*, *grafica: Grafica*) → Grafica

Crea una nuova grafica che corrisponde alla grafica fornita, con una nuova posizione di fissaggio.

Ogni grafica è racchiusa in un rettangolo di delimitazione (bounding box). Ci sono 9 punti notevoli, corrispondenti ai quattro angoli di questo rettangolo, ai punti centrali dei quattro lati e al centro del rettangolo. Ci si può riferire a questi punti usando i nomi *alto_sinistra*, *alto_destra*, *basso_sinistra*, *basso_destra*, *alto_centro*, *centro_destra*, *basso_centro*, *centro_sinistra* e *centro*.

Parameters

- **punto** – il punto indicante la nuova posizione di fissaggio
- **grafica** – grafica originale

Returns
una nuova grafica con una posizione di fissaggio aggiornata

larghezza_grafica(*grafica: Grafica*) → int

Ritorna la larghezza di una grafica.

Parameters
grafica – grafica di cui calcolare la larghezza

Returns
larghezza della grafica

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

Crea una nuova grafica ruotando dell'angolo indicato, in senso antiorario, una grafica attorno alla sua posizione di fissaggio. Un angolo negativo corrisponde a una rotazione in senso orario.

Parameters

- **angolo** – angolo di rotazione in senso antiorario, in gradi
- **grafica** – grafica da ruotare

Returns
una nuova grafica, ruotata

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

Crea una nuova grafica posizionando le due grafiche fornite una sopra l'altra. Le due grafiche vengono centrate orizzontalmente.

La posizione di fissaggio della grafica risultante è nel suo 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

Crea una nuova grafica sovrapponendo le due grafiche fornite, tenendo la prima in primo piano e la seconda sullo sfondo. Le due grafiche vengono sovrapposte sui loro centri.

La posizione di fissaggio della grafica risultante è nel suo centro.

Parameters

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

Returns
grafica risultante dalla sovrapposizione delle due fornite

1.2.3 Output

Funzioni per output di grafiche (visualizzare o salvare).

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

Salva una sequenza di grafiche come un'animazione (GIF).

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

Parameters

- **nome_file** – nome del file da creare (inclusa l'estensione '.gif')
- **grafiche** – lista di grafiche da salvare come animazione
- **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 in un file. Due formati di file sono supportati: PNG (grafica raster) e SVG (grafica vettoriale). L'estensione del nome del file (o ".png" o ".svg") determina il formato.

Una grafica priva di area non può essere salvata nel formato PNG.

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 (con l'estensione)
- **grafica** – grafica da visualizzare
- **debug** – può facoltativamente essere impostato a *True* per sovrapporre informazioni di debug

`visualizza_animazione`(*grafiche: list[Grafica]*, *durata: int = 40*, *loop: bool = True*)

Visualizza una sequenza di grafiche come un'animazione (GIF).

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

Parameters

- **grafiche** – lista di grafiche da salvare come animazione

- **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)

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

Visualizza una grafica. Grafiche prive di area non possono essere visualizzate.

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

1.2.4 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).

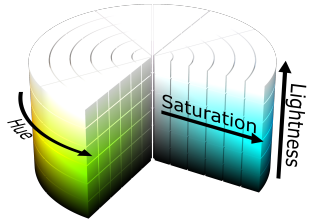


Fig. 4: Cilindro HSL: SharkD via Wikimedia Commons

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).

Parameters

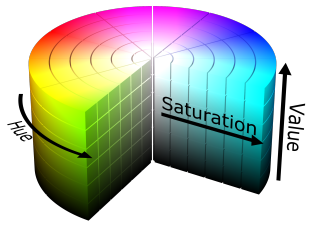


Fig. 5: Cilindro HSV (SharkD via Wikimedia Commons)

- **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).

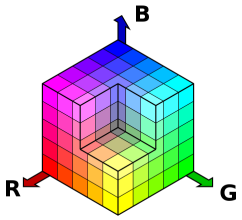


Fig. 6: Cubo RGB (SharkD via Wikimedia Commons)

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

1.2.5 Punti

Tipo *Punto*.

Punto
Rappresenta un punto nel piano.

Nomi di punti notevoli, che possono essere usati come posizioni di fissaggio per una grafica.

alto_centro: *Point*
Il punto centrale del lato superiore del rettangolo di delimitazione della grafica

alto_destra: *Point*
Il vertice in alto a destra del rettangolo di delimitazione della grafica

alto_sinistra: *Point*
Il vertice in alto a sinistra del rettangolo di delimitazione della grafica

basso_centro: *Point*
Il punto centrale del lato inferiore del rettangolo di delimitazione della grafica

basso_destra: *Point*
Il vertice in basso a destra del rettangolo di delimitazione della grafica

basso_sinistra: *Point*
Il vertice in basso a sinistra del rettangolo di delimitazione della grafica

centro: *Point*
Il punto centrale del rettangolo di delimitazione della grafica

centro_destra: *Point*
Il punto centrale del lato destro del rettangolo di delimitazione della grafica

centro_sinistra: *Point*
Il punto centrale del lato sinistro del rettangolo di delimitazione della grafica

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 di fissaggio 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).

1.3 Deutsche Version

1.3.1 Primitive Grafiken

Funktionen zum Erzeugen primitiver Grafiken (Figuren und Texte). Falls nicht anders angegeben befindet sich die Fixierposition in der Mitte des Begrenzungsrahmens der erzeugten Grafik.

dreieck(*seite1: float*, *seite2: float*, *winkel: float*, *farbe: Farbe*) → Grafik

Erzeugt ein Dreieck mit den gegebenen Seitenlängen und dem gegebenen Winkel, gefüllt in der gegebenen Farbe. Die erste Seite verläuft horizontal nach rechts. Der Winkel gibt an, wie viel die zweite Seite im Gegenuhrzeigersinn von der ersten Seite abweicht.

Für alle Dreiecke, ausser für stumpfe Dreiecke, liegt die untere linke Ecke des Begrenzungsrahmens auf dem Eckpunkt des Dreiecks, für das der Winkel angegeben ist.

Die Fixierposition liegt auf dem Schwerpunkt des Dreiecks.

Parameters

- **seite1** – Länge der ersten, horizontalen Seite
- **seite2** – Länge der zweiten Seite
- **winkel** – Winkel von der ersten zu der zweiten Seiten, in Grad
- **farbe** – Farbe 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
- **hoehe** – Höhe der Ellipse
- **farbe** – Füllfarbe der Ellipse

Returns
eine Grafik mit der gegebenen Ellipse

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 Gegenuhrzeigersinn. Ein Winkel von 360 Grad entspricht einem ganzen Kreis.

Die Fixierposition liegt in der Mitte des Kreises, aus dem der Kreissektor ausgeschnitten wurde.

Parameters

- **radius** – Kreisradius
- **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.

Returns
eine leere Grafik (Breite und Höhe sind 0)

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
- **hoehe** – die Höhe des Rechtecks
- **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.

Die Fixierposition liegt auf der linken Kante des Begrenzungsrahmens, auf der Höhe der Grundlinie des Textes.

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

1.3.2 Operationen

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

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

Erzeugt eine neue Grafik, die einer Rotation der gegebenen Grafik um ihre Fixierposition im Gegenuhrzeigersinn um den gegebenen Winkel entspricht. Negative Winkel entsprechen einer Rotation um Uhrzeigersinn.

Parameters

- **winkel** – Drehwinkel, in Grad im Gegenuhrzeigersinn
- **grafik** – zu rotierende Grafik

Returns
die neue, rotierte Grafik

fixiere(*punkt:* Point, *grafik: Grafik*) → Grafik

Erzeugt eine neue Grafik, die der gegebenen Grafik mit einer neuen Fixierposition entspricht.

Jede Grafik liegt in einem rechteckigen Begrenzungsrahmen. Der Rahmen definiert 9 nennenswerte Punkte: die vier Ecken, die Mittelpunkte der vier Kanten und die Mitte des Rahmens. Die Namen dieser Punkte sind: *oben_links*, *oben_mitte*, *oben_rechts*, *mitte_links*, *mitte*, *mitte_rechts*, *unten_links*, *unten_mitte* und *unten_rechts*.

Parameters

- **point** – ein Punkt welcher die neue Fixierposition bestimmt
- **graphic** – die ursprüngliche Grafik

Returns
eine neue Grafik mit der gegebenen Fixierposition

grafik_breite(*grafik: Grafik*) → int

Gibt die Breite der gegebenen Grafik zurück.

Parameters
grafik – Grafik deren Breite gesucht ist

Returns
Breite der Grafik

grafik_hoehe(*grafik: Grafik*) → int

Gibt die Höhe 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 Fixierpositionen übereinanderliegen.

Die überlappenden Fixierpositionen werden zur Fixierposition 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.

Die Fixierposition der neuen Grafik liegt in deren Mitte.

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.

Die Fixierposition der neuen Grafik liegt in deren Mitte.

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.

Die Fixierposition der neuen Grafik liegt in deren Mitte.

Parameters

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

Returns
die zusammengesetzte Grafik

1.3.3 Ausgabe

Funktionen zur Ausgabe (Anzeigen oder Speichern) von Grafiken.

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

Speichere eine Sequenz von Grafiken als Animation (GIF).

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 (mit Erweiterung '.gif')
- **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 in einer Datei.

Zwei Dateiformate werden unterstützt: PNG (Rastergrafik) und SVG (Vektorgrafik). Die Dateinamenerweiterung (entweder ".png" oder ".svg") bestimmt das Dateiformat.

Grafiken ohne Fläche können nicht als PNG-Datei gespeichert werden.

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

Parameters

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

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

Zeige eine Sequenz von Grafiken als Animation (GIF) an.

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

Parameters

- **grafiken** – Liste der anzuzeigenden 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)

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

Zeige die gegebene Grafik an.

Eine Grafik ohne Fläche kann nicht angezeigt werden.

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

Parameters

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

1.3.4 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).

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)

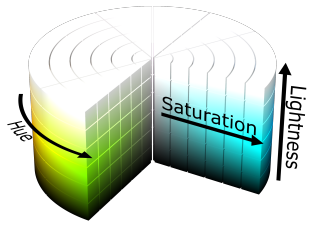


Fig. 7: HSL Zylinder: SharkD via Wikimedia Commons

- **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).

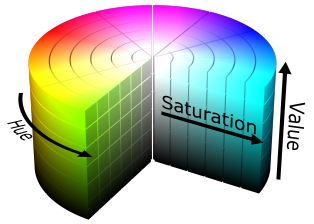


Fig. 8: HSV Zylinder (SharkD via Wikimedia Commons)

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).

Parameters

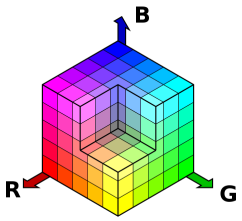


Fig. 9: RGB Würfel (SharkD via Wikimedia Commons)

- **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.

1.3.5 Punti

Typ *Punkt*.

Punkt
Repräsentiert einen Punkt in der Ebene.

Namen nennenswerter Punkte, die als Fixierpositionen für eine Grafik verwendet werden können

mitte: *Point*
Der Mittelpunkt des Begrenzungsrahmens

mitte_links: *Point*
Der Mittelpunkt der linken Kante des Begrenzungsrahmens

mitte_rechts: *Point*
Der Mittelpunkt der rechten Kante des Begrenzungsrahmens

oben_links: *Point*
Die obere linke Ecke des Begrenzungsrahmens

oben_mitte: *Point*
Der Mittelpunkt der oberen Kante des Begrenzungsrahmens

oben_rechts: *Point*
Die obere rechte Ecke des Begrenzungsrahmens

unten_links: *Point*
Die untere linke Ecke des Begrenzungsrahmens

unten_mitte: *Point*
Der Mittelpunkt der unteren Kante des Begrenzungsrahmens

unten_rechts: *Point*
Die untere rechte Ecke des Begrenzungsrahmens

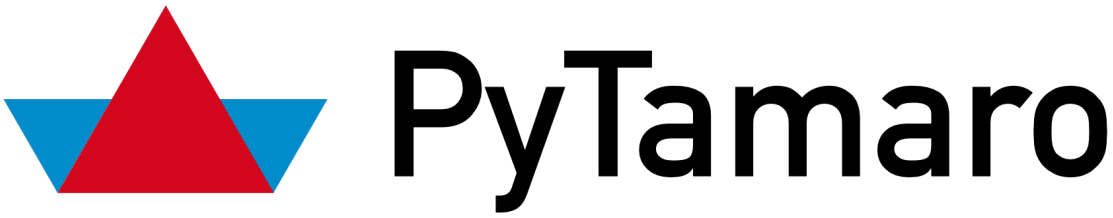
Willkommen in der Pytamaro Dokumentation! Erkunden Sie die verfügbaren Funktionen mit dem Menu auf der linken Seite.

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).

1.4 Version Française



1.4.1 Formes primitives et texte

Fonctions pour créer des graphiques primitifs (formes et texte). À moins d'être spécifié autrement, le point d'ancrage initial est le centre du cadre de délimitation (bounding box) du graphique.

ellipse(*largeur: float*, *hauteur: float*, *couleur:* Color) → *Graphic*
Créé une ellipse avec les dimensions indiquées et rempli avec la couleur donnée.

Lorsque la largeur et la hauteur coïncident, l'ellipse devient un cercle dont le diamètre est égal à la taille indiquée.

Parameters

- **largeur** – largeur de l'ellipse
- **hauteur** – hauteur de l'ellipse
- **couleur** – couleur à utiliser pour remplir l'ellipse

Returns
un graphique avec l'ellipse spécifié

graphique_vide() → *Graphic*
Créé un graphique vide. Quand un graphique vide est composé avec n'importe quel autre graphique, il se comporte comme l'élément neutre: le résultat est toujours égal à l'autre graphique.

Returns
un graphique vide (largeur et hauteur 0)

`rectangle`(*largeur: float*, *hauteur: float*, *couleur:* Color) → *Graphic*

Créé un rectangle de dimensions indiquées et rempli avec la couleur donnée.

Parameters

- **largeur** – largeur du rectangle
- **hauteur** – hauteur du rectangle
- **couleur** – couleur qui remplira le rectangle

Returns
un graphique avec le rectangle spécifié

`secteur_circulaire`(*rayon: float*, *angle: float*, *couleur:* Color) → *Graphic*

Crée un secteur circulaire appartenant à un cercle du rayon indiqué, rempli d'une couleur.

Un secteur circulaire est une portion de cercle comprise entre deux rayons et un arc. Si l'on considère le cercle comme une horloge, le premier rayon "pointe" dans la direction de 3 heures. L'*angle* détermine la position du deuxième rayon, calculée à partir de la position du premier rayon dans le sens inverse des aiguilles d'une montre. Un angle de 360 degrés correspond à un cercle complet.

Le point d'ancrage se trouve au centre du cercle à partir duquel le secteur circulaire est pris.

Parameters

- **rayon** – rayon du cercle duquel est pris le secteur circulaire
- **angle** – angle au centre, en degrés
- **couleur** – couleur à utiliser pour remplir le secteur circulaire

Returns
un graphique avec le secteur circulaire spécifié

`texte`(*contenu: str*, *police: str*, *points: float*, *couleur:* Color) → *Graphic*

Crée un graphique avec le texte rendu à l'aide de la police, de la taille et de la couleur spécifiées.

Lorsque la police True-Type indiquée n'est pas disponible dans le système, une police très basilaire toujours disponible est utilisée à la place. Le graphique qui en résulte a la taille minimale nécessaire pour contenir l'ensemble du texte.

La point d'ancrage est alignée horizontalement sur la gauche et verticalement sur la ligne de base du texte.

Parameters

- **contenu** – le texte à présenter
- **font** – le nom de la police (par exemple, "arial" sous Windows, "Arial" sous macOS)
- **points** – la taille en points typographiques (par exemple, 16)
- **couleur** – la couleur à utiliser pour le rendu du texte

Returns
le texte spécifié sous forme de graphique

`triangle`(*cote1: float*, *cote2: float*, *angle: float*, *couleur:* Color) → *Graphic*

Crée un triangle en spécifiant deux côtés et l'angle qui les sépare, remplis d'une couleur. Le premier côté s'étend horizontalement vers la droite. L'angle spécifie la rotation du deuxième côté, dans le sens inverse des aiguilles d'une montre, par rapport au premier.

Pour tous les triangles, à l'exception des triangles obtus, le point inférieur gauche du graphique résultant coïncide avec le sommet du triangle dont l'angle a été spécifié.

Le point d'ancrage est le centroïde du triangle.

Parameters

- **cote1** – longueur du premier côté (horizontal) du triangle
- **cote2** – longueur du second côté du triangle
- **angle** – angle compris entre les deux côté, en degrés
- **couleur** – couleur à utiliser pour remplir le secteur circulaire

Returns

un graphique avec le triangle spécifié

1.4.2 Operations

Fonctions pour faire des opérations sur des graphiques (principalement pour les combiner).

ancre(*point:* Point, *graphique:* Graphic) → *Graphic*

Créé un nouveau graphique qui correspond au graphique fourni avec un nouveau point d'ancrage.

Chaque graphique est compris dans un cadre de délimitation. Il y a 9 points d'ancrages particuliers, qui correspondent aux quatre coins du cadre (rectangle), le milieu de chaque côté ainsi que le centre du cadre. Les points peuvent être désignés avec les noms suivants: *haut_gauche*, *haut_droite*, *bas_gauche*, *bas_droite*, *haut_centre*, *centre_droite*, *bas_centre*, *centre_gauche* et *centre*.

Parameters

- **point** – le point indiquant le nouveau point d'ancrage
- **graphique** – le graphique originel

Returns

un nouveau graphique avec le nouveau point d'ancrage

au_dessus(*graphique_haut:* Graphic, *graphique_bas:* Graphic) → *Graphic*

Créé un graphique en plaçant les deux graphiques fournis l'un au-dessus de l'autre. Les deux graphiques sont centrés horizontalement.

Le point d'ancrage du nouveau graphique est en son centre.

Parameters

- **graphique_haut** – le graphique à placer au dessus
- **graphique_bas** – le graphique à placer en dessous

Returns

le graphique résultant après avoir placé les deux graphiques l'un au-dessus de l'autre

compose(*graphique_premier_plan:* Graphic, *graphique_arriere_plan:* Graphic) → *Graphic*

Créé un nouveau graphique en composant les deux graphiques fournis. Le premier graphique est maintenu au premier plan, le second en arrière plan. Les graphiques sont alignés en superposant les points d'ancrage.

Le point d'ancrage utilisé pour la composition sera celui du graphique résultant.

Parameters

- **graphique_premier_plan** – le graphique au premier plan

- **graphique_premier_plan** – le graphique en arrière plan

Returns
le graphique qui résulte de la composition

cote_a_cote(*graphique_gauche:* Graphic, *graphique_droite:* Graphic) → *Graphic*

Créé un graphique en plaçant les deux graphiques fournis côte à côte. Les deux graphiques sont centrés verticalement.

Le point d'ancrage du nouveau graphique est en son centre.

Parameters

- **graphique_gauche** – le graphique à placer à gauche
- **graphique_droite** – le graphique à placer à droite

Returns
le graphique résultant après avoir placé les deux graphiques l'un à côté de l'autre

hauteur_graphique(*graphique:* Graphic) → int

Retourne la hauteur du graphique.

Parameters
graphique – graphique duquel calculer la hauteur

Returns
la hauteur du graphique

largeur_graphique(*graphique:* Graphic) → int

Retourne la largeur du graphique.

Parameters
graphique – graphique duquel calculer la largeur

Returns
la largeur du graphique

pivote(*angle: float*, *graphique:* Graphic) → *Graphic*

Crée un nouveau graphique en pivotant dans le sens inverse des aiguilles d'une montre le graphique fourni autour de son point d'ancrage selon l'angle donné. Un angle négatif correspond à une rotation dans le sens des aiguilles d'une montre.

Parameters

- **angle** – angle dans le sens inverse des aiguilles d'une montre, en degrés
- **graphique** – le graphique à pivoter

Returns
un nouveau graphique pivoté

superpose(*graphique_premier_plan:* Graphic, *graphique_arriere_plan:* Graphic) → *Graphic*

Créé un nouveau graphique en superposant les deux graphiques fournis, en gardant le premier au premier plan et en mettant le second en arrière plan. Les graphiques sont superposés par rapport à leur centre.

Le point d'ancrage du nouveau graphique est en son centre.

Parameters

- **graphique_premier_plan** – le graphique au premier plan
- **graphique_arriere_plan** – le graphique en arrière plan

Returns
le graphique résultant de la superposition des deux graphiques fournis

1.4.3 Output

Fonctions pour output de graphiques (visualiser ou sauvegarder).

montre_animation(*graphiques: list[*Graphic*]*, *duree: int = 40*, *en_boucle: bool = True*)

Montre une séquence de graphiques en tant qu'animation (GIF).

Les graphiques sont reproduit de manière séquentielle (normalement à 25 images par secondes) en boucle (à moins que ça ne soit indiqué autrement).

Parameters

- **graphiques** – la liste des graphiques à sauvegarder en tant qu'animation.
- **duree** – durée en millisecondes entre chaque image (par défaut est égale à 40 millisecondes, ce qui amène à avoir 25 images par secondes)
- **en_boucle** – si le GIF doit tourner en boucle indéfiniment (par défaut est *True*)

montre_graphique(*graphique:* Graphic, *debug: bool = False*)

Montre un graphique. Les graphiques n'ayant pas de surface ne peuvent pas être montrés.

Quand *debug* est *True*, la visualisation est ornée d'informations utile pour le débogage: une bordure rouge autour du cadre de délimitation et une croix jaunâtre autour du pion d'ancrage.

Parameters

- **graphique** – le graphique à montrer
- **debug** – (optionnel) peut être assigné à *True* pour superposer les informations de débogage

sauvegarde_animation(*nom_fichier: str*, *graphiques: list[*Graphic*]*, *duree: int = 40*, *en_boucle: bool = True*)

Sauvegarde une séquence de graphiques en tant qu'animation (GIF).

Les graphiques sont reproduit de manière séquentielle (normalement à 25 images par secondes) en boucle (à moins que ça ne soit indiqué autrement).

Parameters

- **nom_fichier** – le nom du fichier à créer (contenant l'extension ".gif")
- **graphiques** – la liste des graphiques à sauvegarder en tant qu'animation.
- **duree** – durée en millisecondes entre chaque image (par défaut est égale à 40 millisecondes, ce qui amène à avoir 25 images par secondes)
- **en_boucle** – si le GIF doit tourner en boucle indéfiniment (par défaut est *True*)

sauvegarde_graphique(*nom_fichier: str*, *graphique:* Graphic, *debug: bool = False*)

Sauvegarde un graphique dans un fichier. Deux formats de fichiers sont supporter: PNG (graphiques tramés) et SVG (graphiques vectoriels). L'extension du fichier (".png" ou ".svg") détermine le format.

Les graphiques avec aucune surface ne peuvent pas être sauvegardés au format PNG.

Quand *debug* est *True*, la visualisation est ornée d'informations utile pour le débogage: une bordure rouge autour du cadre de délimitation et une croix jaunâtre autour du pion d'ancrage.

Parameters

- **nom_fichier** – le nom du fichier à créer (avec l'extension)

- **graphique** – le graphique à sauvegarder
- **debug** – (optionnel) peut être assigné à *True* pour superposer les informations de débogage

1.4.4 Couleurs

Type *Couleur*, ainsi des fonctions pour produire des couleurs et constantes représentants des couleurs particulières.

Couleur

Represente une couleur. Une couleur a aussi un taux d'opacité, de complètement transparent (come la couleur *transparent*) a complètement opaque (come la couleur *rouge*).

couleur_hsl(*teinte: float*, *saturation: float*, *luminosite: float*, *opacite: float = 1.0*) → *Color*

Retourne une couleur avec la teinte (H), saturation (S) et luminosité (V) données, ainsi que le taux d'opacité (alpha, A).

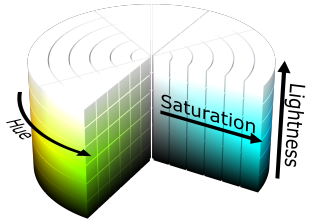


Fig. 10: Cylindre HSL: SharkD via Wikimedia Commons

Parameters

- **teinte** – teinte de la couleur [0-360]
- **saturation** – saturation de la couleur [0-1]
- **luminosite** – quantité de blanc ou noire appliquée [0-1]. Les couleurs complètement saturé ont un valeur de luminosité de 1/2.
- **opacite** – opacité (alpha) de la couleur, où 0 est complètement transparent et 1 complètement opaque. Par défaut, toutes les couleurs sont complètement opaques.

Returns

une couleur avec les composantes HSLA données.

couleur_hsv(*teinte: float*, *saturation: float*, *valeur: float*, *opacite: float = 1.0*) → *Color*

Retourne une couleur avec la teinte (H), saturation (S) et valeur (V) données, ainsi que le taux d'opacité (alpha, A).

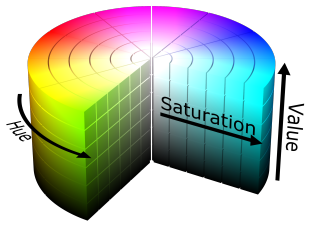


Fig. 11: Cylindre HSV (SharkD via Wikimedia Commons)

Parameters

- **teinte** – teinte de la couleur [0-360]
- **saturation** – saturation de la couleur [0-1]
- **valeur** – quantité de lumière appliquée [0-1] Les couleurs complètement saturé ont une quantité de lumière de 1.
- **opacite** – opacité (alpha) de la couleur, où 0 est complètement transparent et 1 complètement opaque. Par défaut, toutes les couleurs sont complètement opaques.

Returns

une couleur avec le composantes HSVA données

couleur_rgb(*rouge: int*, *vert: int*, *bleu: int*, *opacite: float = 1.0*) → *Color*

Retourne une couleur avec les composantes indiquant le rouge (R), vert (G) et bleu (B) et un certain taux d'opacité (alpha, A).

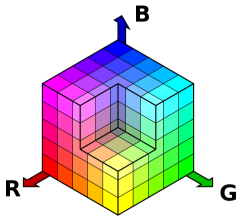


Fig. 12: Cube RGB (SharkD via Wikimedia Commons)

Parameters

- **rouge** – composante rouge [0-255]
- **vert** – composante verte [0-255]
- **bleu** – composante bleu [0-255]
- **opacite** – opacité (alpha) de la couleur, où 0 est complètement transparent et 1 complètement opaque. Par défaut, toutes les couleurs sont complètement opaques.

Returns

une couleur avec les composantes RGBA données

Noms de couleurs particulières qui sont aux coins du cube RGB. Ainsi que la couleur complètement transparent.

blanc: *Color*

Couleur blanc

bleu: *Color*

Couleur bleu

cyan: *Color*

Couleur cyan

jaune: *Color*
Couleur jaune

magenta: *Color*
Couleur magenta

noir: *Color*
Couleur noire

rouge: *Color*
Couleur rouge

transparent: *Color*
Couleur complètement transparente

vert: *Color*
Couleur verte

1.4.5 Points

Type *Point*.

class Point
Represents a point on a plane.

Noms de points particuliers, qui peuvent être utilisé comme des points d'ancrage pour des graphiques.

bas_centre: *Point*
Le point centrale du côté inférieur du cadre de délimitation du graphique

bas_droite: *Point*
Le coin en bas à droite du cadre de délimitation du graphique

bas_gauche: *Point*
Le coin en bas à gauche du cadre de délimitation du graphique

centre: *Point*
Le point centrale du cadre de délimitation du graphique

centre_droite: *Point*
Le point centrale du côté droit du cadre de délimitation du graphique

centre_gauche: *Point*
Le point centrale du côté gauche du cadre de délimitation du graphique

haut_centre: *Point*
Le point centrale du côté supérieur du cadre de délimitation du graphique

haut_droite: *Point*
Le coin en haut à droite du cadre de délimitation du graphique

haut_gauche: *Point*
Le coin en haut à gauche du cadre de délimitation du graphique

Bienvenue dans la documentation de Pytamaro ! Explorez les fonctions disponibles à l'aide du menu de gauche.

Graphique
Un graphique (image) avec un point à ancrer.

Le point d'ancrage est utilisé pour les opérations suivantes:

- rotation (pour déterminer le centre de rotation)
- composition de graphiques (deux graphiques sont composés en alignant leur point d'ancrage).

PYTHON MODULE INDEX

p

INDEX

H

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z