

# Main functions

## 1. Mathematics

### Classical operations

- $a + b$ ,  $a - b$ ,  $a * b$  classic operations
- $a / b$  “real” division (returns a floating point number)
- $a // b$  Euclidean division quotient (returns an integer)
- $a \% b$  remainder of the Euclidean division, called  $a$  modulo  $b$
- $\text{abs}(x)$  absolute value
- $x ** n$  power  $x^n$
- $4.56\text{e}12$  for  $4.56 \times 10^{12}$

### “math” module

The use of other mathematical functions requires the math module which is called by the command:

```
from math import *
```

- $\text{sqrt}(x)$  square root  $\sqrt{x}$
- $\cos(x)$ ,  $\sin(x)$ ,  $\tan(x)$  trigonometric functions  $\cos x$ ,  $\sin x$ ,  $\tan x$  in radians
- $\pi$  approximate value of  $\pi = 3.14159265\dots$
- $\text{floor}(x)$  integer just below  $x$
- $\text{ceil}(x)$  integer just above  $x$
- $\text{gcd}(a,b)$  gcd of  $a$  and  $b$

### “random” module

The random module generates numbers in a pseudo-random way. It is called by the command:

```
from random import *
```

- $\text{random}()$  on each call, returns a floating number  $x$  at random, satisfying  $0 \leq x < 1$ .
- $\text{randint}(a,b)$  for each call, returns an integer  $n$  at random, satisfying  $a \leq n \leq b$ .
- $\text{choice}(mylist)$  on each call, randomly draws an item from the list.
- $mylist.\text{shuffle}()$  mixes the list (the list is modified).

### Binary notation

- $\text{bin}(n)$  returns the binary notation of the integer  $n$  as a string. Example:  $\text{bin}(17)$  returns '0b10001'.
- To write a number directly in binary notation, simply write the number starting with 0b (without quotation marks). For example 0b11011 is equal to 27.

## 2. Booleans

A boolean is a data that takes either the value `True` or the value `False`.

### Comparisons

The following comparison tests return a boolean.

- `a == b` equality test
- `a < b` strict lower test
- `a <= b` large lower test
- `a > b` or `a >= b` higher test
- `a != b` non-equality test

Do not confuse “`a = b`” (assignment) and “`a == b`” (equality test).

### Boolean operations

- `P and Q` logical “and”
- `P or Q` logical “or”
- `not P` negation

## 3. Strings I

### Strings

- `"A"` or `'A'` one character
- `"Python"` or `'Python'` a string
- `len(string)` the string length. Example: `len("Python")` returns 6.
- `string1 + string2` concatenation.  
Example: `"I love" + "Python"` returns `"I lovePython"`.
- `string[i]` returns the  $i$ -th character of `string` (numbering starts at 0).  
Example with `string = "Python"`, `string[1]` is equal to `"y"`. See the table below.

Letter	P	y	t	h	o	n
Rank	0	1	2	3	4	5

### Number/string conversion

- **String.** `str(number)` converts a number (integer or floating point number) into a string. Examples: `str(7)` returns the string `"7"`; `str(1.234)` returns the string `"1.234"`.
- **Integer.** `int(string)` returns the integer corresponding to the string. Example: `int("45")` returns the integer 45.
- **Floating point number.** `float(string)` returns the floating point number corresponding to the string. Example: `float("3.14")` returns the number 3.14.

### Substrings

- `string[i:j]` returns the substring of characters with from rank  $i$  to rank  $j-1$  of `string`.  
Example: with `string = "This is a string"`, `string[2:7]` returns `"is is"`.
- `string[i:]` returns characters from rank  $i$  until the end of `string`.  
Example: `string[5:]` returns `"is a string"`.
- `string[:j]` returns characters from the beginning to rank  $j-1$  of `string`. Example: `string[:4]` returns `"This"`.

## Format

The `format()` method allows you to format text or numbers. This function returns a string.

- **Text**

Test\_ Test\_ Test\_

- '{:10}'.format('Test') left alignment (on 10 characters)
- '{:>10}'.format('Test') right alignment
- '{:~10}'.format('Test') centered

- Integer

456                  □□□456                  000456

- ```
- '{:d}'.format(456)    integer
- '{:6d}'.format(456)   right aligned (on 6 characters)
- '{:06d}'.format(456)  adding leading zeros (on 6 characters)
```

- Floating point number

3.141593      3.14159265      3.1416      003.1416

- ```
- '{:f}'.format(3.14159265653589793)    floating point number
- '{:.8f}'.format(3.14159265653589793)    8 decimal places
- '{:8.4f}'.format(3.14159265653589793)    on 8 characters with 4 numbers after the
decimal point
- '{:08.4f}'.format(3.141592653589793)    adding leading zeros
```

## 4. Strings II

## Encoding

- `chr(n)` returns the character associated with the ASCII/unicode code number *n*. Example: `chr(65)` returns "A"; `chr(97)` returns "a".
- `ord(c)` returns the ASCII/unicode code number associated with the character *c*. Example: `ord("A")` returns 65; `ord("a")` returns 97.

The beginning of the ASCII/unicode table is given below.

33	!	43	+	53	5	63	?	73	I	83	S	93	]	103	g	113	q	123	{
34	"	44	,	54	6	64	@	74	J	84	T	94	^	104	h	114	r	124	
35	#	45	-	55	7	65	A	75	K	85	U	95	_	105	i	115	s	125	}
36	\$	46	.	56	8	66	B	76	L	86	V	96	'	106	j	116	t	126	~
37	%	47	/	57	9	67	C	77	M	87	W	97	a	107	k	117	u	127	-
38	&	48	0	58	:	68	D	78	N	88	X	98	b	108	l	118	v		
39	'	49	1	59	;	69	E	79	O	89	Y	99	c	109	m	119	w		
40	(	50	2	60	<	70	F	80	P	90	Z	100	d	110	n	120	x		
41	)	51	3	61	=	71	G	81	Q	91	[	101	e	111	o	121	y		
42	*	52	4	62	>	72	H	82	R	92	\	102	f	112	p	122	z		

### Upper/lower-case

- `string.upper()` returns a string in uppercase.
- `string.lower()` returns a string in lowercase.

### Search/replace

- `substring in string` returns “true” or “false” depending on if substring appears in string.  
Example: "NOT" in "TO BE OR NOT TO BE" returns True.
- `string.find(substring)` returns the rank at which the substring was found (and -1 otherwise).  
Example: with `string = "ABCDE"`, `string.find("CD")` returns 2.
- `string.replace(substring,new_substring)` replaces each occurrence of the substring by the new substring.  
Example: with `string = "ABCDE"`, `string.replace("CD","XY")` returns "ABXYE".

### Split/join

- `string.split(separator)` separates the string into a list of substrings (by default the separator is the space).

Examples:

- `"To be or not to be.".split()` returns ['To', 'be', 'or', 'not', 'to', 'be.']
- `"12.5;17.5;18".split(";")` returns ['12.5', '17.5', '18']

- `separator.join(mylist)` groups the substrings into a single string by adding the separator between each.

Examples:

- `"".join(["To", "be", "or", "not", "to", "be."])` returns 'Tobeornottobe.'  
Spaces are missing.
- `" ".join(["To", "be", "or", "not", "to", "be."])` returns 'To be or not to be.'  
It's better when the separator is a space.
- `"--".join(["To", "be", "or", "not", "to", "be."])` returns 'To--be--or--not--to--be.'

## 5. Lists I

### Construction of a list

Examples:

- `mylist1 = [5,4,3,2,1]` a list of five integers.
- `mylist2 = ["Friday","Saturday","Sunday"]` a list of three strings.
- `mylist3 = []` the empty list.
- `list(range(n))` list of integers from 0 to  $n-1$ .
- `list(range(a,b))` list of integers from  $a$  to  $b-1$ .
- `list(range(a,b,step))` list of integers from  $a$  to  $b-1$ , with a step given by the integer `step`.

### Get an item

- `mylist[i]` returns the element at rank  $i$ . Be careful, the rank starts at 0.  
Example: `mylist = ["A","B","C","D","E","F"]` then `mylist[2]` returns "C".

Letter	"A"	"B"	"C"	"D"	"E"	"F"
Rank	0	1	2	3	4	5

- `mylist[-1]` returns the last element, `mylist[-2]` returns the second last element...
- `mylist.pop()` removes the last item from the list and returns it.

### Add one element (or more)

- `mylist.append(element)` adds the item at the end of the list. Example: if `mylist = [5,6,7,8]` then `mylist.append(9)` adds 9 to the list, `mylist` is now `[5,6,7,8,9]`.
- `new_mylist = mylist + [element]` provides a new list with an extra element at the end. Example: `[1,2,3,4] + [5]` is `[1,2,3,4,5]`.
- `[element] + mylist` returns a list where the item is added at the beginning. Example: `[5] + [1,2,3,4]` is `[5,1,2,3,4]`.
- `mylist1 + mylist2` concatenates the two lists. Example: with `mylist1 = [4,5,6]` and `mylist2 = [7,8,9]` then `mylist1 + mylist2` is `[4,5,6,7,8,9]`.

**Example of construction.** Here is how to build the list that contains the first squares:

```
list_squares = []                # We start from an empty list
for i in range(10):
    list_squares.append(i**2)     # We add squares one by one
```

At the end `list_squares` is:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

### Browse a list

- `len(mylist)` returns the length of the list. Example: `len([5,4,3,2,1])` returns 5.
- Browse a list (and here display each item):  

```
for element in mylist:
    print(element)
```
- Browse a list using the rank.

```
n = len(mylist)
for i in range(n):
    print(i,mylist[i])
```

## 6. Lists II

### Mathematics

- `max(mylist)` returns the largest element. Example: `max([10,16,13,14])` returns 16.
- `min(mylist)` returns the smallest element. Example: `min([10,16,13,14])` returns 10.
- `sum(mylist)` returns the sum of all elements. Example: `sum([10,16,13,14])` returns 53.

### Slicing lists

- `mylist[a:b]` returns the sublist of elements from rank  $a$  to rank  $b - 1$ .
- `mylist[a:]` returns the list of elements from rank  $a$  until the end.
- `mylist[:b]` returns the list of items from the beginning to rank  $b - 1$ .

Letter	"A"	"B"	"C"	"D"	"E"	"F"	"G"
Rank	0	1	2	3	4	5	6

For example if `mylist = ["A","B","C","D","E","F","G"]` then:

- `mylist[1:4]` returns `["B","C","D"]`.
- `mylist[:2]` is like `mylist[0:2]` and returns `["A","B"]`.
- `mylist[4:]` returns `["E","F","G"]`. It's the same thing as `mylist[4:n]` where `n = len(mylist)`.

### Find the rank of an element

- `mylist.index(element)` returns the first position at which the item was found. Example: with `mylist = [12, 30, 5, 9, 5, 21]`, `mylist.index(5)` returns 2.
- If you just want to know if an item belongs to a list, then the statement :  
`element in mylist`  
returns True or False. Example: with `mylist = [12, 30, 5, 9, 5, 21]`, "9 in mylist" is true, while "8 in mylist" is false.

### Order

- `sorted(mylist)` returns the ordered list of items.  
Example: `sorted([13,11,7,4,6,8,12,6])` returns the list `[4,6,6,7,8,11,12,13]`.
- `mylist.sort()` does not return anything but the list `mylist` is now ordered.

### Invert a list

Here are three methods:

- `mylist.reverse()` modifies the list in place;
- `list(reversed(mylist))` returns a new list;
- `mylist[::-1]` returns a new list.

### Delete an item

Three methods.

- `mylist.remove(element)` deletes the first occurrence found.  
Example: `mylist = [2,5,3,8,5]`, the instruction `mylist.remove(5)` modifies the list which is now `[2,3,8,5]` (the first 5 has disappeared).
- `del mylist[i]` deletes element at rank  $i$  (the list is modified).
- `element = mylist.pop()` removes the last item from the list and returns it.

### List comprehension

- Let's start from a list, for example `mylist = [1,2,3,4,5,6,7,6,5,4,3,2,1]`.
- `list_doubles = [ 2*x for x in mylist ]` returns a list that contains the doubles of the items of `mylist`. So this is the list `[2,4,6,8,...]`.
- `liste_squares = [ x**2 for x in mylist ]` returns the list of squares of the items in the list `mylist`. So this is the list `[1,4,9,16,...]`.
- `partial_list = [ x for x in mylist if x > 2 ]` extracts from the list only the elements greater than 2. So this is the list `[3,4,5,6,7,6,5,4,3]`.

### List of lists

Example:

```
array = [ [2,14,5], [3,5,7], [15,19,4], [8,6,5] ]
```

corresponds to the table:

		index $j$		
		→		
		$j=0$	$j=1$	$j=2$
index $i$	$i=0$	2	14	5
	$i=1$	3	5	7
	$i=2$	15	19	4
	$i=3$	8	6	5

Then `array[i]` returns the sublist of rank  $i$ , and `array[i][j]` returns the element located in the sublist number  $i$ , at rank  $j$  of this sublist. For example:

- `array[0]` returns the sublist `[2,14,5]`.
- `array[1]` returns the sublist `[3,5,7]`.
- `array[0][0]` returns the integer 2.
- `array[0][1]` returns the integer 14.
- `array[2][1]` returns the integer 19.

A table of  $n$  rows and  $p$  columns.

- `array = [[0 for j in range(p)] for i in range(n)]` initializes an array and fills it with 0.
- `array[i][j] = 1` modifies a value in the table (the one at the location  $(i,j)$ ).

## 7. Input/output

### Display

- `print(string1,string2,string3,...)` displays strings or objects. Example: `print("Value =",14)` displays `Value = 14`. Example: `print("Line 1 \n Line 2")` displays on two lines.
- **Separator.** `print(...,sep="...")` changes the separator (by default the separator is the space character). Example: `print("Bob",17,13,16,sep="; ")` displays `Bob; 17; 13; 16`.
- **End of line.** `print(...,end="...")` changes the character placed at the end (by default it is the line break character `\n`). Example `print(17,end="")` then `print(76)` displays `1776` on a single line.

### Keyboard entry

`input()` pauses the program and waits for the user to send a message on the keyboard (ended by pressing the “Enter” key). The message is a string.

Here is a small program that asks for the user’s first name and age and displays a message like “Hello Kevin” then “You are a minor/adult” according to age.

```
first_name = input ("What's your name? ")
print("Hello",first_name)
```

```
age_str = input("How old are you? ")
age = int(age_str)
```

```
if age >= 18:
    print("You're an adult!")
else:
    print("You're a minor!")
```

## 8. Files

### Order

- `fi = open("my_file.txt","r")` opening in reading ("r" for read).
- `fi = open("my_file.txt","w")` opening in writing ("w" for write). The file is created if it does not exist, if it existed the previous content is first deleted.
- `fi = open("my_file.txt","a")` opening for writing, the data will be written at the end of the current data ("a" for append).
- `fi.write("one line")` write to the file.
- `fi.read()` reads the whole file (see below for another method).
- `fi.readlines()` reads all the lines (see below for another method).
- `fi.close()` file closing.

### Write lines to a file

```
fi = open("my_file.txt","w")
```

```
fi.write("Hello world!\n")
```

```
line = "Hi there.\n"
```



```
fi.write(line)
```

```
fi.close()
```

### Read lines from a file

```
fi = open("my_file.txt","r")
```

```
for line in fi:  
    print(line)
```

```
fi.close()
```

### Read a file (official method)

```
with open("my_file.txt","r") as fi:  
    for line in fi:  
        print(line)
```

## 9. Turtle

The turtle module is called by the command:

```
from turtle import *
```

### Main commands

- `forward(length)` advances a number of steps
- `backward(length)` goes backwards
- `right(angle)` turns to the right (without advancing) at a given angle in degrees
- `left(angle)` turns left
- `setheading(direction)` points in a direction (0 = right, 90 = top, -90 = bottom, 180 = left)
- `goto(x,y)` moves to the point (x,y)
- `setx(newx)` changes the value of the abscissa
- `sety(newy)` changes the value of the ordinate
- `down()` sets the pen down
- `up()` sets the pen up
- `width(size)` changes the thickness of the line
- `color(col)` changes the color: "red", "green", "blue", "orange", "purple"...
- `position()` returns the (x,y) position of the turtle
- `heading()` returns the direction angle to which the turtle is pointing
- `towards(x,y)` returns the angle between the horizontal and the segment starting at the turtle and ending at the point (x,y)
- `speed("fastest")` maximum travel speed
- `exitonclick()` ends the program as soon as you click

### Several turtles

Here is an example of a program with two turtles.

```

turtle1 = Turtle()    # with capital 'T'!
turtle2 = Turtle()

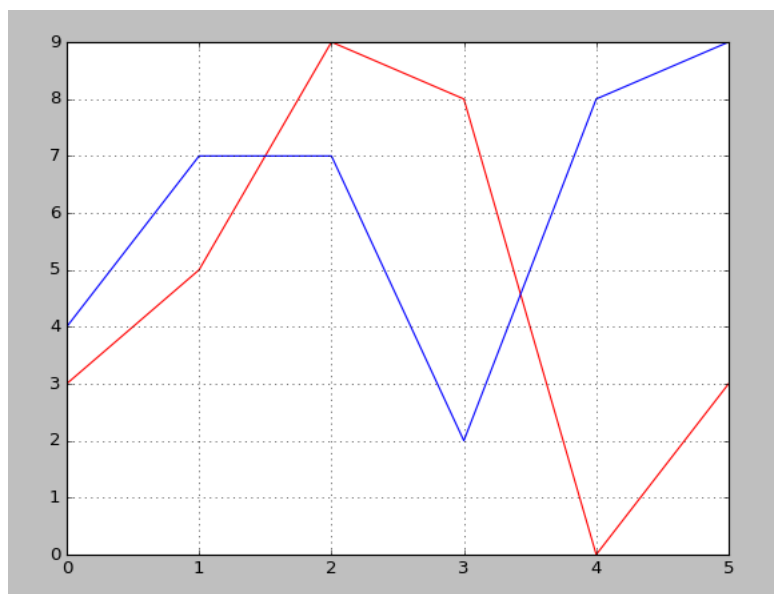
turtle1.color('red')
turtle2.color('blue')

turtle1.forward(100)
turtle2.left(90)
turtle2.forward(100)

```

## 10. Matplotlib

With the matplotlib module it is very easy to draw a list. Here is an example.



```

import matplotlib.pyplot as plt

mylist1 = [3,5,9,8,0,3]
mylist2 = [4,7,7,2,8,9]

plt.plot(mylist1,color="red")
plt.plot(mylist2,color="blue")
plt.grid()
plt.show()

```

### Main functions.

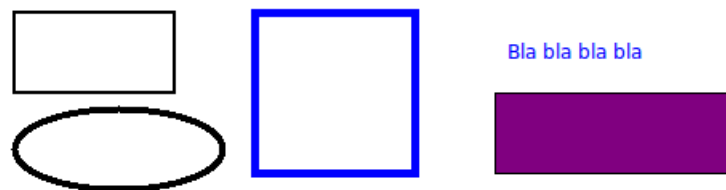
- `plt.plot(mylist)` traces the points of a list (in the form  $(i, \ell_i)$ ) that are joined by segments.
- `plt.plot(list_x, list_y)` traces the points of a list (of the form  $(x_i, y_i)$  where  $x_i$  browses the first list and  $y_i$  the second).
- `plt.scatter(x, y, color='red', s=100)` displays a point at  $(x, y)$  (of a size  $s$ ).
- `plt.grid()` draws a grid.
- `plt.show()` displays everything.
- `plt.close()` exits the display.

- `plt.xlim(xmin,xmax)` defines the interval for the  $x$ .
- `plt.ylim(ymin,ymax)` defines the interval for the  $y$ .
- `plt.axis('equal')` imposes an orthonormal basis.

## 11. Tkinter

### 11.1. Graphics

To display this:



The code is:

```
# tkinter window
root = Tk()

canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)

# A rectangle
canvas.create_rectangle(50,50,150,100,width=2)

# A rectangle with thick blue edges
canvas.create_rectangle(200,50,300,150,width=5,outline="blue")

# A rectangle filled with purple
canvas.create_rectangle(350,100,500,150,fill="purple")

# An ellipse
canvas.create_oval(50,110,180,160,width=4)

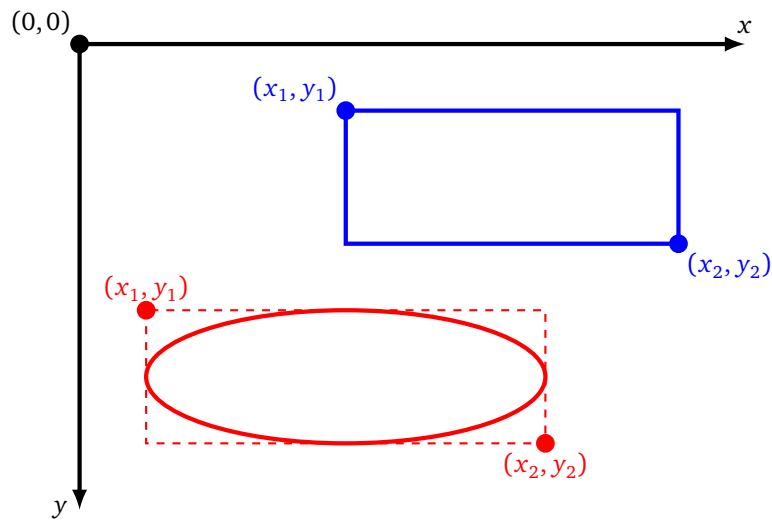
# Some text
canvas.create_text(400,75,text="Bla bla bla bla",fill="blue")

# Launch of the window
root.mainloop()
```

Some explanations:

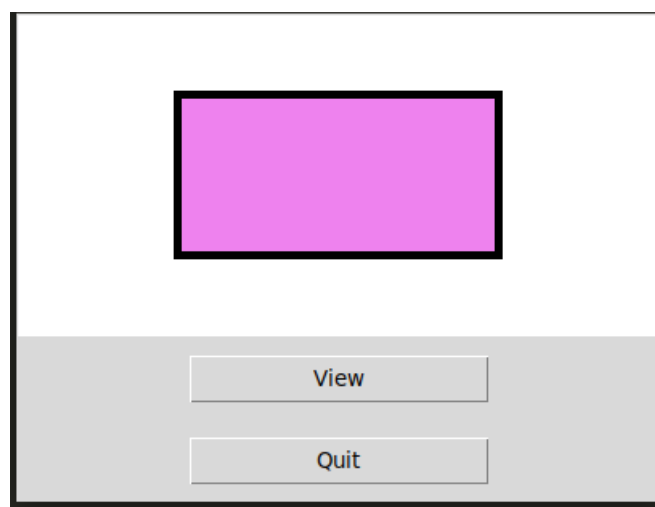
- The `tkinter` module allows us to define variables `root` and `canvas` that determine a graphic window (here width 800 and height 600 pixels). Then describe everything you want to add to the window. And finally the window is displayed by the command `root.mainloop()` (at the very end).

- Attention! The window's graphic marker has its y-axis pointing downwards. The origin  $(0,0)$  is the top left corner (see figure below).
- Command to draw a rectangle: `create_rectangle(x1,y1,x2,y2)`; just specify the coordinates  $(x_1, y_1)$ ,  $(x_2, y_2)$  of two opposite vertices. The option `width` adjusts the thickness of the line, `outline` defines the color of this line, `fill` defines the filling color.
- An ellipse is traced by the command `create_oval(x1,y1,x2,y2)`, where  $(x_1, y_1)$ ,  $(x_2, y_2)$  are the coordinates of two opposite vertices of a rectangle framing the desired ellipse (see figure). A circle is obtained when the corresponding rectangle is a square!
- Text is displayed by the command `canvas.create_text(x,y,text="My text")` specifying the coordinates  $(x, y)$  of the point from which you want to display the text.



## 11.2. Buttons

It is more ergonomic to display windows where actions are performed by clicking on buttons. Here is the window of a small program with two buttons. The first button changes the color of the rectangle, the second button ends the program.



The code is:

```
from tkinter import *
from random import *
```

```

root = Tk()
canvas = Canvas(root, width=400, height=200, background="white")
canvas.pack(fill="both", expand=True)

def action_button():
    canvas.delete("all")          # Clear all
    colors = ["red","orange","yellow","green","cyan","blue","violet","purple"]
    col = choice(colors)         # Random color
    canvas.create_rectangle(100,50,300,150,width=5,fill=col)
    return

button_color = Button(root,text="View", width=20, command=action_button)
button_color.pack(pady=10)

button_quit = Button(root,text="Quit", width=20, command=root.quit)
button_quit.pack(side=BOTTOM, pady=10)

root.mainloop()

```

Some explanations:

- A button is created by the command `Button`. The `text` option customizes the text displayed on the button. The button created is added to the window by the method `pack`.
- The most important thing is the action associated with the button! It is the option `command` that receives the name of the function to be executed when the button is clicked. For our example `command=action_button`, associate the click on the button with a change of color.
- Attention! You have to give the name of the function without brackets: `command=my_function` and not `command=my_function()`.
- To associate the button with “Quit” and close the window, the argument is `command=root.quit`.
- The instruction `canvas.delete("all")` deletes all drawings from our graphic window.

### 11.3. Text

Here’s how to display text with Python and the graphics window module `tkinter`.

## Text with Python!

The code is:

```

from tkinter import *
from tkinter.font import Font
# tkinter window
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(fill="both", expand=True)
# Font
myfont = Font(family="Times", size=30)
# Some text
canvas.create_text(100,100, text="Text with Python!",

```

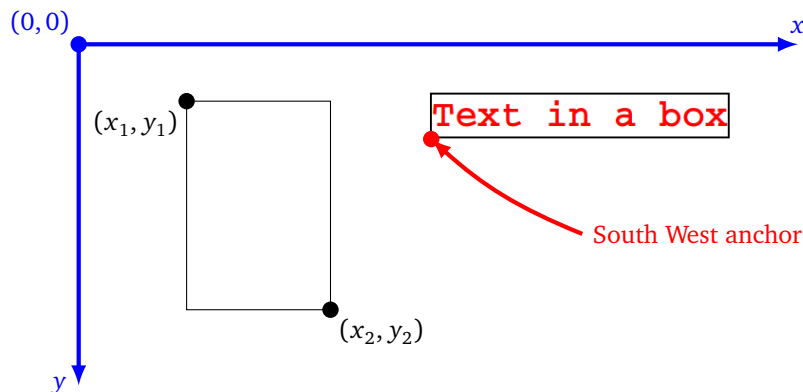
```

anchor=SW, font=myfont, fill="blue")
# Launch the window
root.mainloop()

```

Some explanations:

- `root` and `canvas` are the variables that define a graphic window (here of width 800 and height 600 pixels). This window is launched by the last command: `root.mainloop()`.
- We remind you that for the graphic coordinates, the  $y$ -axis is directed downwards. To define a rectangle, simply specify the coordinates  $(x_1, y_1)$  and  $(x_2, y_2)$  from two opposite vertices (see figure below).
- The text is displayed by the command `canvas.create_text()`. It is necessary to specify the  $(x, y)$  coordinates of the point from which you want to display the text.
- The `text` option allows you to pass the string to display.
- The `anchor` option allows you to specify the text anchor point, `anchor=SW` means that the text box is anchored to the Southwest point (SW) (see figure below).
- The `fill` option allows you to specify the text color.
- The `font` option allows you to define the font (i.e. the style and size of the characters). Here are some examples of fonts, it's up to you to test them:
  - `Font(family="Times", size=20)`
  - `Font(family="Courier", size=16, weight="bold")` in **bold**
  - `Font(family="Helvetica", size=16, slant="italic")` in *italic*



## 11.4. Mouse click

Here is a small program that displays a graphic window. Each time the user clicks (with the left mouse button) the program displays a small square (on the window) and displays “Click at  $x = \dots$ ,  $y = \dots$ ” (on the console).

```

from tkinter import *

# Window
root = Tk()
canvas = Canvas(root, width=800, height=600, background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

# Catch mouse click
def action_mouse_click(event):
    canvas.focus_set()

```

```

x = event.x
y = event.y
canvas.create_rectangle(x,y,x+10,y+10,fill="red")
print("Click at x =",x," y =",y)
return

# Association click/action
canvas.bind("<Button-1>", action_mouse_click)

# Launch
root.mainloop()

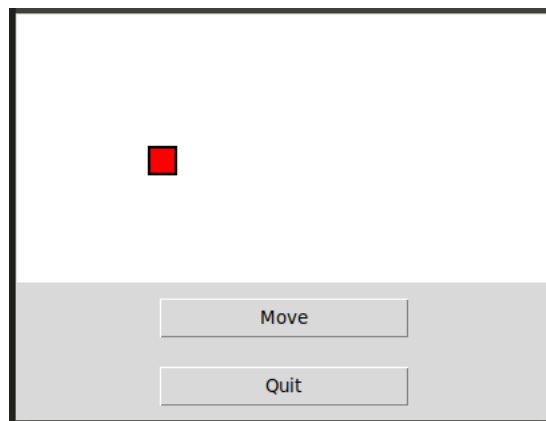
```

Here are some explanations:

- The creation of the window is usual. The program ends with the launch using the command `mainloop()`.
- The first key point is to associate a mouse click to an action, that's what the line does:  
`canvas.bind("<Button-1>", action_mouse_click)`  
Each time the left mouse button is clicked, Python executes the function `action_mouse_click`. (Note that there are no brackets for the call to the function.)
- Second key point: the `action_mouse_click` function retrieves the click coordinates and then does two things here: it displays a small rectangle at the click location and prints the  $(x, y)$  coordinates in the terminal window.
- The coordinates  $x$  and  $y$  are expressed in pixels;  $(0, 0)$  refers to the upper left corner of the window (the area delimited by `canvas`).

## 11.5. Movement

Here is a program that moves a small square and bounces it off the edges of the window.



Here are the main points:

- An object `rect` is defined, it is a global variable, as well as its coordinates  $x_0, y_0$ .
- This object is (a little bit) moved by the function `mymove()` which shifts the rectangle by  $(dx, dy)$ .
- The key point is that this function will be executed again after a short period of time. The command:  
`canvas.after(50, mymove)`  
requests a new execution of the function `mymove()` after a short delay (here 50 milliseconds).
- The repetition of small shifts simulates movement.

```
from tkinter import *
```

```
the_width = 400
the_height = 200

root = Tk()
canvas = Canvas(root, width=the_width, height=the_height, background="white")
canvas.pack(fill="both", expand=True)

# Coordinates and speed
x0, y0 = 100,100
dx = +5 # Horizontal speed
dy = +2 # Vertical speed

# The rectangle to move
rect = canvas.create_rectangle(x0,y0,x0+20,y0+20,width=2,fill="red")

# Main function
def mymove():
    global x0, y0, dx, dy

    x0 = x0 + dx # New abscissa
    y0 = y0 + dy # New ordinate

    canvas.coords(rect,x0,y0,x0+20,y0+20) # Move

    if x0 < 0 or x0 > the_width:
        dx = -dx # Change of horizontal direction
    if y0 < 0 or y0 > the_height:
        dy = -dy # Change of vertical direction

    canvas.after(50,mymove) # Call after 50 milliseconds

    return

# Function for the button
def action_move():
    mymove()
    return

# Buttons
button_move = Button(root,text="Move", width=20, command=action_move)
button_move.pack(pady=10)

button_quit = Button(root,text="Quit", width=20, command=root.quit)
button_quit.pack(side=BOTTOM, pady=10)

root.mainloop()
```