Statistics – Data visualization

It's good to know how to calculate the minimum, maximum, average and quartiles of a series. It's even better to visualize them all on the same graph!

Activity 1 (Basic statistics)**.**

Goal: calculate the main characteristics of a series of data: minimum, maximum, mean and standard deviation.

In this activity `mylist` refers to a list of numbers (integer or floating point numbers).

1. Write your own function `mysum(mylist)` which calculates the sum of the elements of a given list. Compare your result with the `sum()` function described below which already exists in `Python`. Especially for an empty list, check that your result is 0.

   `python: sum()`

   Use: `sum(mylist)`
   Input: a list of numbers
   Output: a number

   Example: `sum([4,8,3])` returns `15`

   You can now use the function `sum()` *in your programs!*
2. Write a `mean(mylist)` function that calculates the average of the items in a given list (and returns 0 if the list is empty).
3. Write your own `minimum(mylist)` function that returns the smallest value of the items in a given list. Compare your result with the `Python` `min()` function described below (which can also calculate the minimum of two numbers).

   `python: min()`

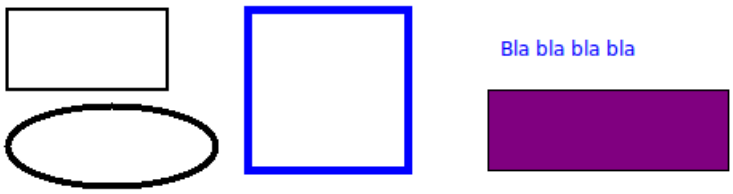
   Use: `min(mylist)` or `min(a,b)`
   Input: a list of numbers or two numbers
   Output: a number

   Example:
   - `min(12,7)` returns `7`
   - `min([10,5,9,12])` returns `5`

   You can now use the `min()` *function and of course also the* `max()` *function in your programs!*

4. The ***variance*** of a data series $(x_1, x_2, \ldots, x_n)$ is defined as the average of the squares of deviations from the mean. That is to say:
$$v = \frac{1}{n}\big((x_1 - m)^2 + (x_2 - m)^2 + \cdots + (x_n - m)^2\big)$$
where m is the average of $(x_1, x_2, \ldots, x_n)$.
Write a `variance(mylist)` function that calculates the variance of the elements in a list.
For example, for the series $(6, 8, 2, 10)$, the average is $m = 6.5$, the variance is
$$v = \frac{1}{4}\big((6-6.5)^2 + (8-6.5)^2 + (2-6.5)^2 + (10-6.5)^2\big) = 8.75.$$
5. The ***standard deviation*** of a series $(x_1, x_2, \ldots, x_n)$ is the square root of the variance:
$$\sigma = \sqrt{v}$$
where v is the variance. Program a `standard_deviation(mylist)` function. With the example above we find $\sigma = \sqrt{v} = \sqrt{8.75} = 2.95\ldots$
6. Here are the average monthly temperatures (in Celsius degrees) in London and Chicago.
```
temp_london = [4.9,5,7.2,9.7,13.1,16.6,18.7,18.2,15.5,11.6,7.7,5.6]
temp_chicago = [-5,-2.7,2.8,9.2,15.2,20.7,23.5,22.6,18.4,12.1,4.8,-1.9]
```
Calculate the average temperature over the year in London and then in Chicago. Calculate the standard deviation of the temperatures in London and then in Chicago. What conclusions do you draw from this?

Lesson 1 (Graphics with tkinter).
To display this:



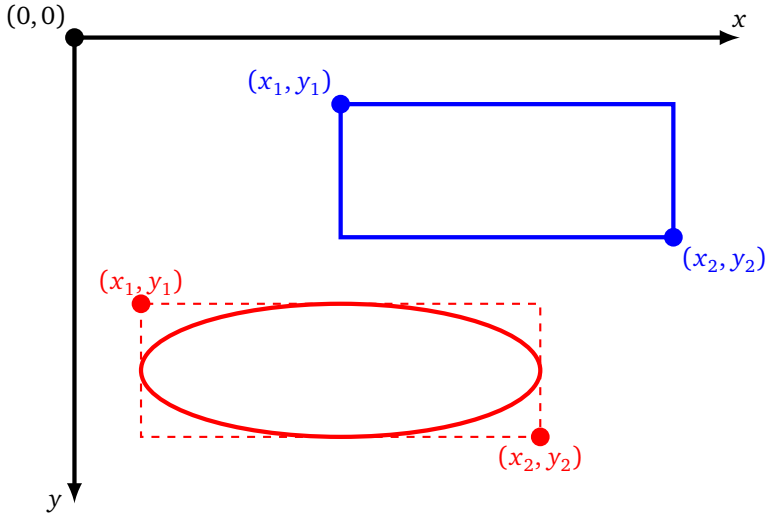
The code is:

```
# 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()
```

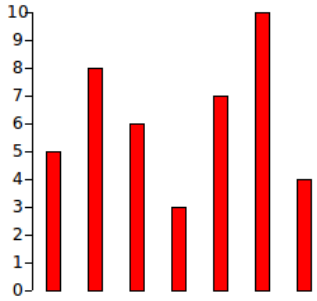
Some explanations:

- 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 (x, y) coordinates of the point from which you want to display the text.

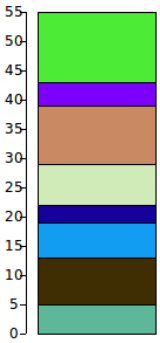


Activity 2 (Graphics)**.**

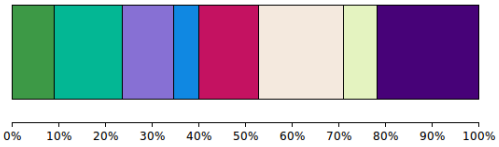
Goal: visualize data by different types of graphs.



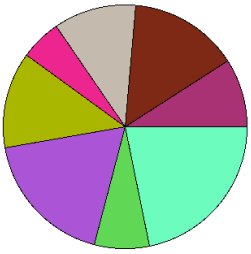
Bar graphics



Cumulative graph



Percentage graphics



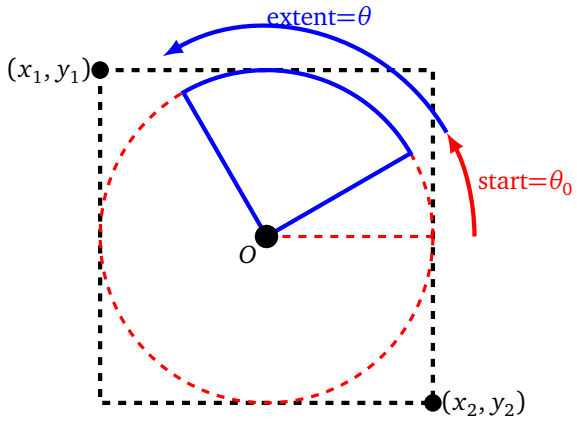
Pie chart

1. **Bar graphics.** Write a `bar_graphics(mylist)` function that displays the values of a list as vertical bars.
   Hints.
   - First of all, don't worry about drawing the vertical axis of the coordinates with the figures.
   - You can define a variable `scale` that allows you to enlarge your rectangles, so that they have a size adapted to the screen.
   - If you want to test your graph with a random list, here is how to build a random list of 10 integers between 1 and 20:

     ```
     from random import *
     mylist = [randint(1,20) for i in range(10)]
     ```

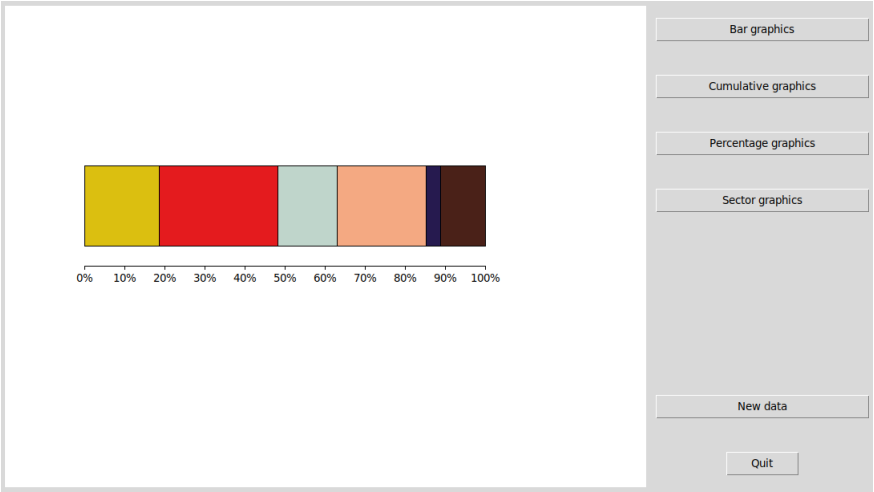
2. **Cumulative graph.** Write a `cumulative_graphics(mylist)` function that displays the values of a list in the form of rectangles one above the other.
3. **Graphics with percentage.** Write a `percentage_graphics(mylist)` function that displays the values of a list in a horizontal rectangle of fixed size (for example 500 pixels) and is divided into sub-rectangles representing the values.
4. **Pie chart.** Write a `sector_graphics(mylist)` function that displays the values of a list as a pie chart (a fixed size disk divided into sectors representing the values).
   The `tkinter create_arc()` function, which allows you to draw arcs of circles, is not very intuitive. Imagine that we draw a circle, by specifying the coordinates of the corners of a square that surrounds it, then by specifying the starting angle and the angle of the sector (in degrees).

   ```
   canvas.create_arc(x1,y1,x2,y2,start=start_angle,extent=my_angle)
   ```



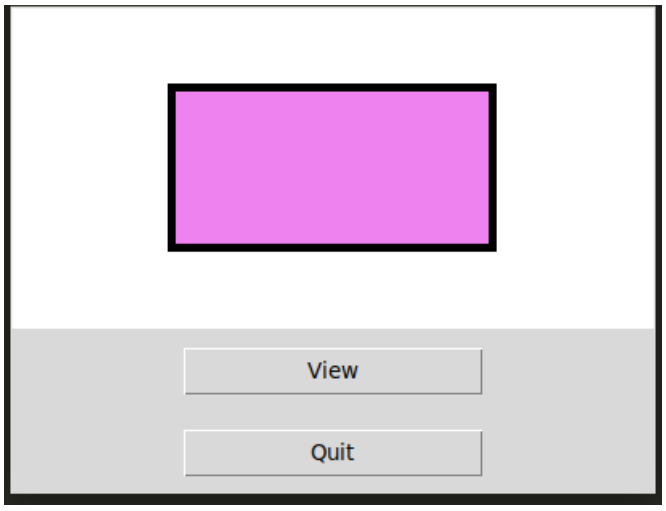
The option `style=PIESLICE` displays a sector instead of an arc.

5. **Bonus.** Gather your work into a program that allows the user to choose the diagram he wants by clicking on buttons, and also the possibility to get a new random series of data. *To create and manage buttons with* `tkinter`*, see the lesson below.*



Lesson 2 (Buttons with tkinter)**.**

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","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`, associates the click on the button with a change of color.
- Attention! You have to give the name of the function without brackets: `command=my_functcion` 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.

Activity 3 (Median and quartiles)**.**

Goal: calculate the median and quartiles of some data.

1. Program a function `median(mylist)` which calculates the median value of the items in a given list. By definition, half of the values are less than or equal to the median, the other half are greater than or equal to the median.
   Background. We note n the length of the list and we assume that the list is ordered (from the smallest to the largest element).
   - **Case n odd.** The median is the value of the list at the index $\frac{n-1}{2}$. Example with `mylist = [12,12,14,15,19]`:
     - the length of the list is $n = 5$ (indices range from 0 to 4),
     - the middle value at index 2,

  - the median is the value `mylist[2]`, so it is 14.
- **Case n even.** The median is the average between the value of the list at index $\frac{n}{2}-1$ and index $\frac{n}{2}$. Example with `mylist = [10,14,19,20]`:
  - the length of the list is $n=4$ (indices range from 0 to 3),
  - the middle indices are 1 and 2,
  - the median is the average between `mylist[1]` and `mylist[2]`, so it is $\frac{14+19}{2}=16.5$.

2. The results of a class are collected in the following form of a number of students per grade:

   `grade_count = [0,0,1,2,5,2,3,5,4,1,2]`

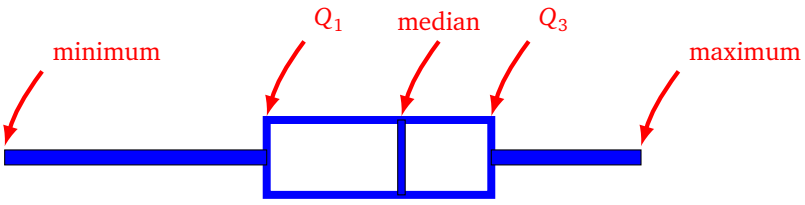
   The index i range is from 0 to 10. And the value at index i indicates the number of students who received the grade i. For example here, 1 student got the grade 2, 2 students got the grade 3, 5 students got 4, ... Write a `grades_to_list(grade_count)` function that takes the list of numbers of students for each grade as input and returns the list of all grades. For our example the function must return `[2,3,3,4,4,4,4,4,5,5,6,6,6,7,...]`.
   Deduce a function that calculates the median of a class's scores from the numbers of students for each grade.
3. Write a function `quartiles(mylist)` that calculates the quartiles Q_1, Q_2, Q_3 of the items in a given list. The quartiles divide the values into: one quarter below Q_1, one quarter between Q_1 and Q_2, one quarter between Q_2 and Q_3, one quarter above Q_3. For the calculation, we will use the fact that:
   - Q_2 is simply the median of the entire list (assumed ordered),
   - Q_1 is the median of the sublist formed by the first half of the values,
   - Q_3 is the median of the sublist formed by the second half of the values.

For the implementation, it is necessary to consider again whether the length n of the list is even or not. Deduce a function that calculates the quartiles of a class's grades from a list of the numbers of students per grade.

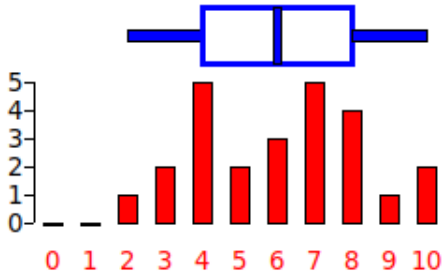
Activity 4 (Box plot).

Goal: draw box plots.

A ***box plot*** is a diagram that represents the main characteristics of a statistical series: minimum, maximum, median and quartiles. The schematic diagram is as follows:



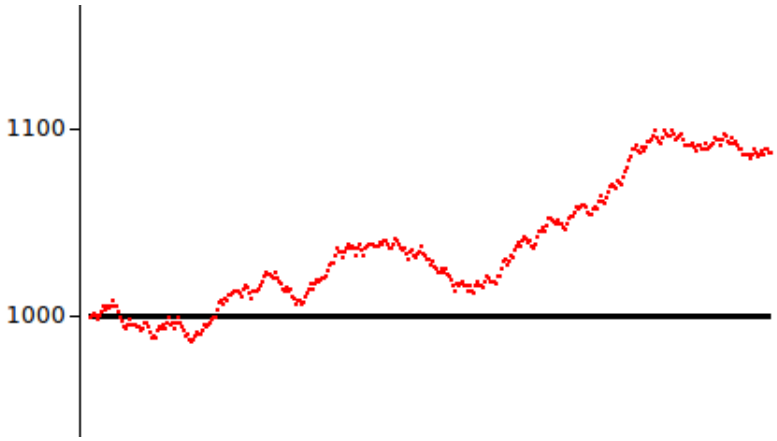
Write a `box_plot(grade_count)` function that draws the box plot of a class's grades from a list of the numbers of students per grade (see previous activity).



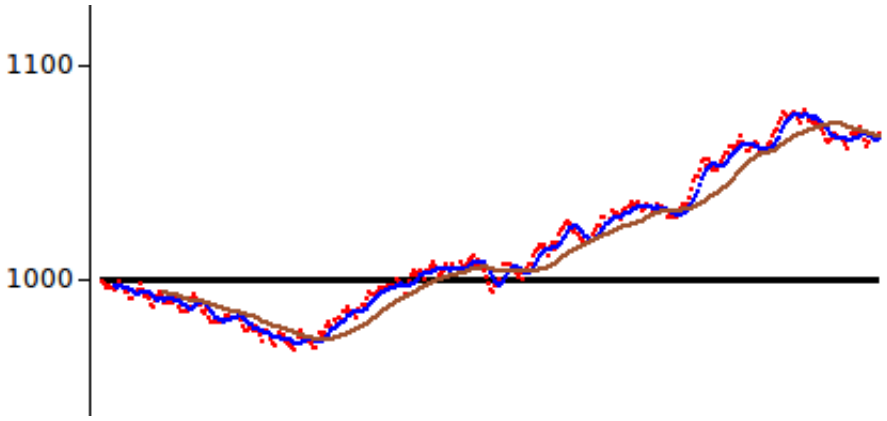
Activity 5 (Moving average)**.**

Goal: calculate moving averages in order to draw "smooth" curves.

1. Simulate the stock market price of the *NISDUQ* index over 365 days. At the beginning, day $j = 0$, the index is equal to 1000. Then the index for a day is determined by adding a random value (positive or negative) to the value of the previous day's index:
   $$\text{index of the day } j = \text{index of the day } (j-1) \ + \ \text{random value}.$$
   For this random value, you can try a formula like:
   ```
   value = randint(-10,12)/3
   ```
   Write an `index_stock_exchange()` function, without parameters, which returns a list of 365 index values using this method.
2. Trace point by point the index curve over a year. (To draw a point, you can display a square with a size of 1 pixel.)



3. *Since the daily index curve is very chaotic, we want to smooth it out in order to make it more readable. For this we can calculate moving averages.*
   The 7-day moving average at the day j, is the average of the last 7 indices. For example: the 7-day moving average for the day $j = 7$ is the average of the day's indices $j = 1, 2, 3, 4, 5, 6, 7$. You can change the duration: for example the 30-day moving average is the average of the last 30 indices. Write a `moving_average(mylist,duration)` function that returns a list of all moving averages in a data set with respect to a fixed time.
4. Trace these data point by point on the same graph: the index curve over a year (shown in red below), the 7-day moving average curve (shown in blue below) and the 30-day moving average curve (shown in brown below). Note that the longer the duration, the more the curve is "smooth". (Of course the 30-day moving average curve only starts from the thirtieth day.)


1100
1000