

A list is a way to group elements into a single object. After defining a list, you can retrieve each item of the list one by one, but also add new ones...

Lesson 1 (List (1)).

A **list** is a series of elements. This can be a list of integers, for example `[5, -7, 12, 99]`, or a list of strings, for example `["March", "April", "May"]` or objects of different types `[3.14, "pi", 10e-3, "x", True]`.

- **Construction of a list.** A list is defined by elements between square brackets:
 - `mylist1 = [5,4,3,2,1]` a list of 5 integers,
 - `mylist2 = ["Friday", "Saturday", "Sunday"]` a list of 3 strings,
 - `mylist3 = []` the empty list (very useful if you intend to complete the list later).
- **Get an item.** To get an item from the list, simply write `mylist[i]` where *i* is the rank of the desired item.

Beware! The trap is that you start counting from the rank 0.

For example after the instruction `mylist = ["A", "B", "C", "D", "E", "F"]` then

- `mylist[0]` returns "A"
- `mylist[1]` returns "B"
- `mylist[2]` returns "C"
- `mylist[3]` returns "D"
- `mylist[4]` returns "E"
- `mylist[5]` returns "F"

"A"	"B"	"C"	"D"	"E"	"F"
-----	-----	-----	-----	-----	-----

rank: 0 1 2 3 4 5

- **Add an element.** To add an item at the end of a list, just use the command `mylist.append(element)`. For example if `primes = [2,3,5,7]` then `primes.append(11)` adds 11 to the list, if you then execute the instruction `primes.append(13)` then the list `primes` becomes `[2,3,5,7,11,13]`.
- **Example of construction.** Here is how to build the list that contains the first ten squares:

```
list_squares = []           # Start from the empty list
for i in range(10):
    list_squares.append(i**2) # Add squares one by one
```

At the end `list_squares` is:

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

Lesson 2 (List (2)).

- **Length of a list.** The length of a list is the number of elements it contains. The command `len(mylist)` returns the length. The list `[5,4,3,2,1]` is 5 elements long, the list `["Friday","Saturday","Sunday"]` has length 3, the empty list `[]` has length 0.

- **Browse a list.** Here is the easiest way to scan a list (and in this case, to display each item):

```
for item in mylist:
    print(item)
```

- **Browse a list (again).** Sometimes you need to know the index of the elements. Here is another way to do it (which here displays the index and the element).

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

- To get a list from `range()` you have to write:

```
list(range(n))
```

- It's a bad idea to name your list "list" because this word is already used by Python.

Activity 1 (Simple or compound interests).

Goal: create two lists to compare two types of interests.

1. **Simple interest.** We have an amount of S_0 . Each year this investment earns interest based on the initial amount.

For example, with an initial amount of $S_0 = 1000$ and simple interest of $p = 10\%$. The interest is 100. So after one year, I have a sum of $S_1 = 1100$, after two years $S_2 = 1200\dots$

Program a `simple_interest(S0,p,n)` function that returns the list of amounts for the n first years. For example `simple_interest(1000,10,3)` returns `[1000, 1100, 1200, 1300]`.

2. **Compound interest.** An amount of S_0 brings in compound interest. This time the interest is calculated each year on the basis of the sum of the previous year, i.e. according to the formula:

$$I_{n+1} = S_n \times \frac{p}{100}$$

Program a function `compound_interest(S0,p,n)` which returns the list of amounts of the n first years. For example `compound_interest(1000,10,3)` returns `[1000, 1100, 1210, 1331]`.

3. I have the choice between a simple interest investment of 10% or a compound interest investment of 7%. What is the most advantageous solution depending on the duration of the placement?

Lesson 3 (List (3)).

- **Concatenate two lists.** If you have two lists, you can merge them by the operator "+". For example with `mylist1 = [4,5,6]` and `mylist2 = [7,8,9]`

```
mylist1 + mylist2 is [4,5,6,7,8,9].
```

- **Add an item at the end.** The operator "+" provides another method to add an item to a list:

```
mylist = mylist + [element]
```

For example `[1,2,3,4] + [5]` is `[1,2,3,4,5]`. Attention! The element to be added must be surrounded by square brackets. It is an alternative method to `mylist.append(element)`.

- **Add an element at the beginning.** With:

```
mylist = [element] + mylist
```

the item is added at the beginning of the list. For example `[5] + [1,2,3,4]` is `[5,1,2,3,4]`.

- **Slicing lists.** You can extract a whole part of the list at once: `mylist[a:b]` returns the sublist of items with indices a to $b - 1$.

"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[0:2]` returns `["A", "B"]`
- `mylist[4:7]` returns `["E", "F", "G"]`

Once again, it is important to remember that the index of a list starts at 0 and that slicing `mylist[a:b]` stops at the rank $b - 1$.

Activity 2 (Manipulate lists).

Goal: program small routines that manipulate lists.

1. Program a `rotate(mylist)` function that shifts all the elements of a list by one index (the last element becoming the first). The function returns a new list.
For example, `rotate([1,2,3,4])` returns the list `[4,1,2,3]`.
2. Program an `inverse(mylist)` function that inverts the order of the elements in a list.
For example, `inverse([1,2,3,4])` returns the list `[4,3,2,1]`.
3. Program a `delete_rank(mylist,rank)` function that returns a list of all elements, except the one at the given index.
For example, `delete_rank([8,7,6,5,4],2)` returns the list `[8,7,5,4]` (item 6 that was at index 2 is deleted).
4. Program a `delete_element(mylist,element)` function returning a list that contains all items except those equal to the specified element.
For example, `delete_element([8,7,4,6,5,4],4)` returns the list `[8,7,6,5]` (all items equal to 4 have been deleted).

Lesson 4 (Manipulate lists).

You can now use the Python functions which do some of these operations.

- **Invert a list.** Here are three methods:
 - `mylist.reverse()` modifies the list in place (i.e. `mylist` is now reversed, the command returns nothing);
 - `list(reversed(mylist))` returns a new list;
 - `mylist[::-1]` returns a new list.
- **Delete an item.** The command `mylist.remove(element)` deletes the first occurrence found (the list is modified). For example if `mylist = [2,5,3,8,5]` the call `mylist.remove(5)` modifies the list to become `[2,3,8,5]` (the first 5 has disappeared).
- **Delete an element (again).** The command `del mylist[i]` deletes the element of rank i (the

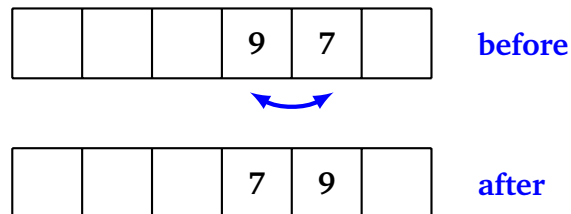
list is modified).

Activity 3 (Bubble sort).

Goal: order a list from the smallest to the largest element.

The bubble sort is a simple way to order a list, here it will be from the smallest to the largest element. The principle is as follows:

- We go through the list from the beginning. As soon as you encounter two consecutive elements in the wrong order, you exchange them.
- At the end of the first pass, the largest element is at the end and it will not move anymore.
- We restart from the beginning (until the penultimate element), this time the last two elements are well placed.
- We continue this way. There is a total of $n - 1$ passages if the list is of length n .



Here is the bubble sort algorithm:

Algorithm.

- – Input: a list ℓ of n numbers
- – Output: the ordered list from the smallest to the largest
- For i ranging from $n - 1$ to 0:
 - For j ranging from 0 to $i - 1$:
 - If $\ell[j + 1] < \ell[j]$ then exchange $\ell[j]$ and $\ell[j + 1]$.
- Return the list ℓ .

Program the bubble sort algorithm into a `bubble_sort(mylist)` function that returns the ordered list of elements. For example `bubble_sort([13,11,7,4,6,8,12,6])` returns the list `[4,6,6,7,8,11,12,13]`.

Hints.

- Begin by defining `new_mylist = list(mylist)` and work only with this new list.
- For the index i to run backwards from $n - 1$ to 0, you can use the command:

```
for i in range(n-1, -1, -1):
```

Indeed `range(a, b, -1)` corresponds to the decreasing list of integers i satisfying $a \geq i > b$ (as usual the right bound is not included).

Lesson 5 (Sorting).

You can now use the `sorted()` function from Python which orders lists.

python: sorted()

Use: `sorted(mylist)`

Input: a list

Output: the ordered list of elements

Example: `sorted([13,11,7,4,6,8,12,6])` returns the list
`[4,6,6,7,8,11,12,13]`.

Attention! There is also a `mylist.sort()` method that works a little differently. This command returns nothing, but on the other hand the list `mylist` is now ordered. We are talking about a modification *in place*.

Activity 4 (Arithmetic).

Goal: improve some of the “Arithmetic – While loop – I” chapter functions.

1. **Prime factors.** Program a `prime_factors(n)` function that returns a list of all the prime factors of an integer $n \geq 2$. For example, for $n = 12936$, its decomposition into prime factors is $n = 2^3 \times 3 \times 7^2 \times 11$, the function returns `[2, 2, 2, 3, 7, 7, 11]`.

Hints. Consult the “Arithmetic – While loop – I” chapter. The core of the algorithm is as follows:

As long as $d \leq n$:

If d is a divisor of n , then:

add d to the list,

n becomes n/d .

Otherwise increment d by 1.

2. **List of prime numbers.** Write a `list_primes(n)` function that returns the list of all prime numbers less than n . For example `list_primes(100)` returns the list:

`[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]`

To do this, you will program an algorithm that is a simple version of the sieve of Eratosthenes:

Algorithm.

- – Input: an integer $n \geq 2$.
- – Output: the list of prime numbers $< n$.
- Initialize `mylist` with a list that contains all integers from 2 to $n - 1$.
- For d ranging from 2 to $n - 1$:
 - For k in `mylist`:
 - If d divides k and $d \neq k$, then remove the element k from `mylist`.
- Return `mylist`.

Hints.

- Start from `mylist = list(range(2,n))`.
- Use `mylist.remove(k)`.

Explanations. Let’s see how the algorithm works with $n = 30$.

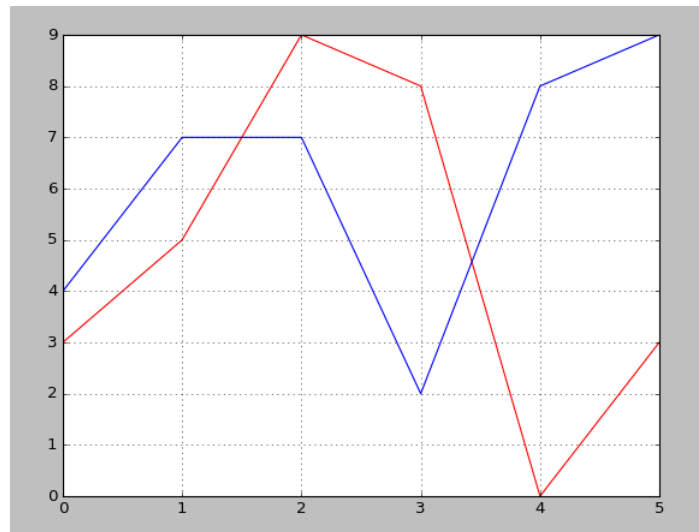
- At the beginning the list is

`[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29]`

- We start with $d = 2$, we eliminate all the numbers divisible by 2, unless it is the number 2: so we eliminate 4, 6, 8, ..., the list is now [2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29].
- We continue with $d = 3$, we eliminate multiples of 3 (except 3), after these operations the list is: [2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29].
- With $d = 4$, we eliminate multiples of 4 (but there are no more).
- With $d = 5$ we eliminate multiples of 5 (here we just eliminate 25), the list becomes [2, 3, 5, 7, 11, 13, 17, 19, 23, 29].
- We continue (here nothing happens anymore).
- At the end, the list is [2, 3, 5, 7, 11, 13, 17, 19, 23, 29].

Lesson 6 (Plot a list).

With the `matplotlib` module it is very easy to visualize the elements of a list of numbers.



```
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()
```

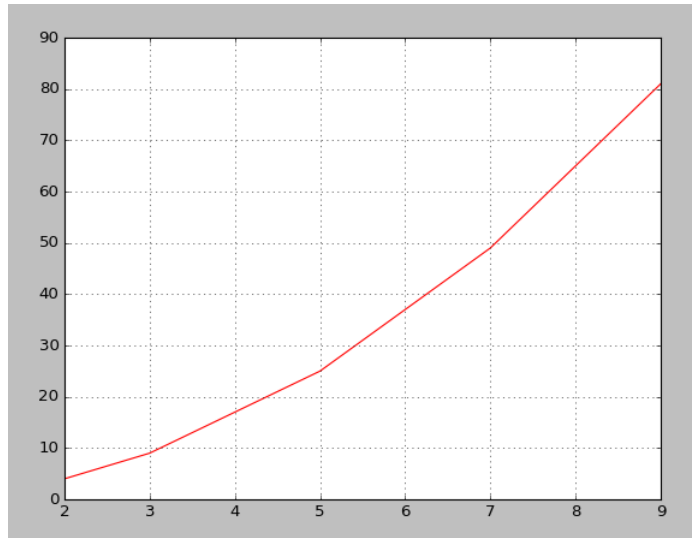
Explanations.

- The module is named `matplotlib.pyplot` and is given the new simpler name of `plt`.
- Attention! The `matplotlib` module is not always installed by default with Python.
- `plt.plot(mylist)` traces the points of a list (in the form of (i, l_i)) that are linked by segments.
- `plt.grid()` draws a grid.
- `plt.show()` displays everything.

To display points (x_i, y_i) you must provide the list of x -values then the list of y -values:

```
plt.plot(mylist_x,mylist_y,color="red")
```

Here is an example of a graph obtained by displaying coordinate points of the type (x, y) with $y = x^2$.

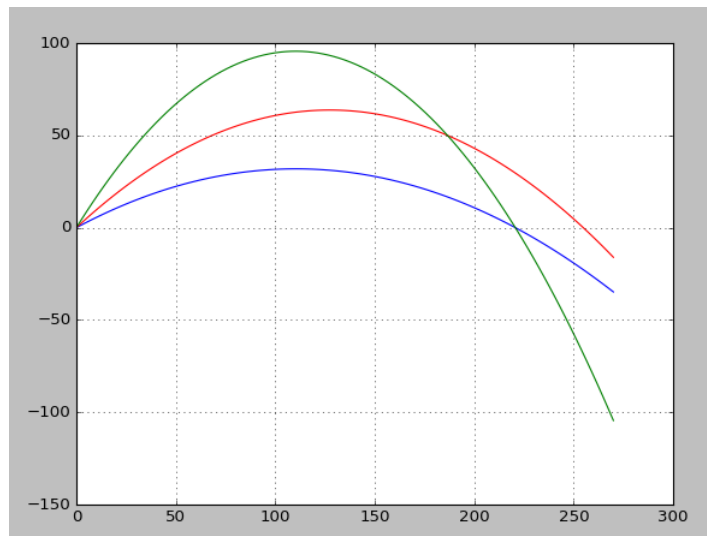


```
import matplotlib.pyplot as plt

mylist_x = [2, 3, 5, 7, 9]
mylist_y = [4, 9, 25, 49, 81]
plt.plot(mylist_x,mylist_y,color="red")
plt.grid()
plt.show()
```

Activity 5 (Ballistics).

Goal: visualize the firing of a cannonball.

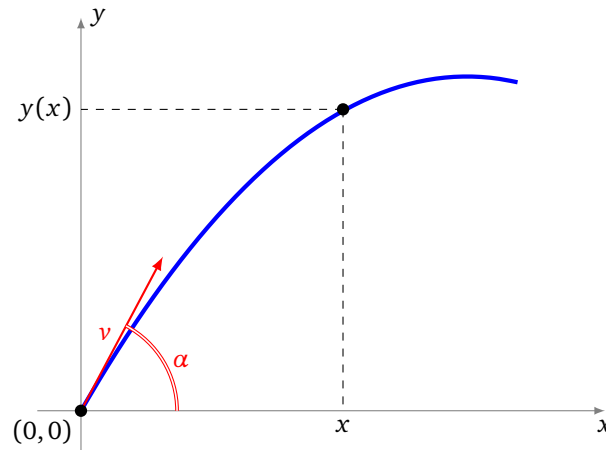


A cannonball has been fired from the point $(0, 0)$. The trajectory equation is given by the formula:

$$y(x) = -\frac{1}{2}g \frac{1}{v^2 \cos^2(\alpha)} x^2 + \tan(\alpha)x$$

where

- α is the angle of the shot,
- v is the initial speed,
- g is the gravitational constant: we will take $g = 9.81$.



1. Program a `parabolic_shot(x, v, alpha)` function which returns the value $y(x)$ given by the formula.

Hint. Be careful with the units for the angle α . If for example you choose that the unit for the angle is degrees, then to apply the formula with Python you must first convert the angles to radians:

$$\alpha_{\text{radian}} = \frac{2\pi}{360} \alpha_{\text{degree}}$$

2. Program a `list_trajectory(xmax, n, v, alpha)` function that calculates the list of y -values of the $n + 1$ points of the trajectory whose x -values are regularly spaced between 0 and x_{max} .

Method. For i ranging from 0 to n :

- calculate $x_i = i \cdot \frac{x_{\text{max}}}{n}$,
- calculate $y_i = y(x_i)$ using the trajectory formula,
- add y_i to the list.

3. For $v = 50$, $x_{\text{max}} = 270$ and $n = 100$, display different trajectories according to the values of α . What angle α allows to reach the point $(x, 0)$ at ground level as far away from the shooting point as possible?