HAB 619 Introduction to Scientific Computing in Sports Science

De Motu Animalium G.Borelli (1680)

#3

SERDAR ARITAN

serdar.aritan@hacettepe.edu.tr

Biyomekanik Araştırma Grubu
www.biomech.hacettepe.edu.tr
Spor Bilimleri Fakültesi
www.sbt.hacettepe.edu.tr
Hacettepe Universitesi, Ankara, Türkiye
www.hacettepe.edu.tr

SymPy : a Python library for symbolic mathematics.

It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python and does not require any external libraries.

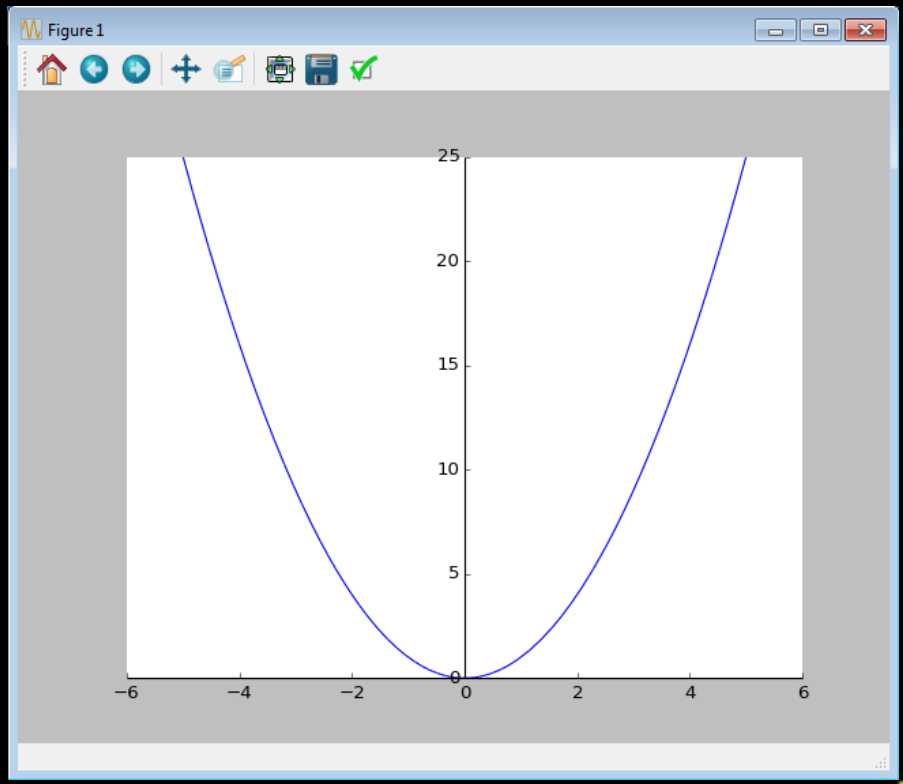
http://docs.sympy.org/latest/guide.html#learning-sympy

SymPy : a Python library for symbolic mathematics.

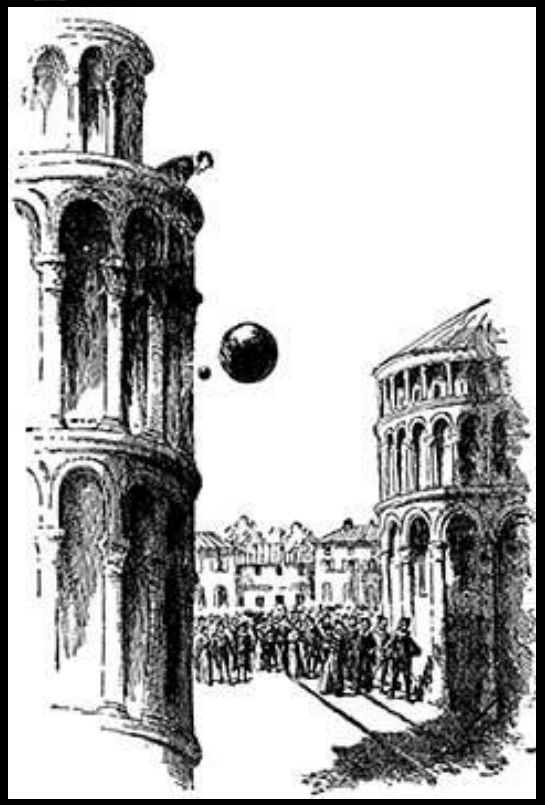
```
>>> from sympy import *
>>> x, y, z = symbols('x y z')
>>> init_printing(use_unicode=True)
>>> Eq(x, y)
x = y
>>> solve(Eq(x**2, 1),  x)
[-1, 1]
>>> solve(Eq(x**2 - 1, 0),  x)
[-1, 1]
>>> solve(x**2 - 1,  x)
[-1, 1]
>>> solve([x - y + 2, x + y - 3], [x, y])
{x: 1/2, y: 5/2}
>>> roots(x**3 - 6*x**2 + 9*x, x)
{0: 1, 3: 2}
```

SymPy : a Python library for symbolic mathematics.

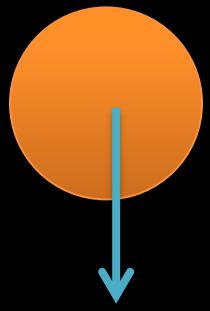
```
>>> from sympy import symbols
>>> from sympy.plotting import plot
>>> x = symbols('x')
>>> plot(x**2, (x, -5, 5))
```



Free Fall of an Object: An Experiment by Galileo

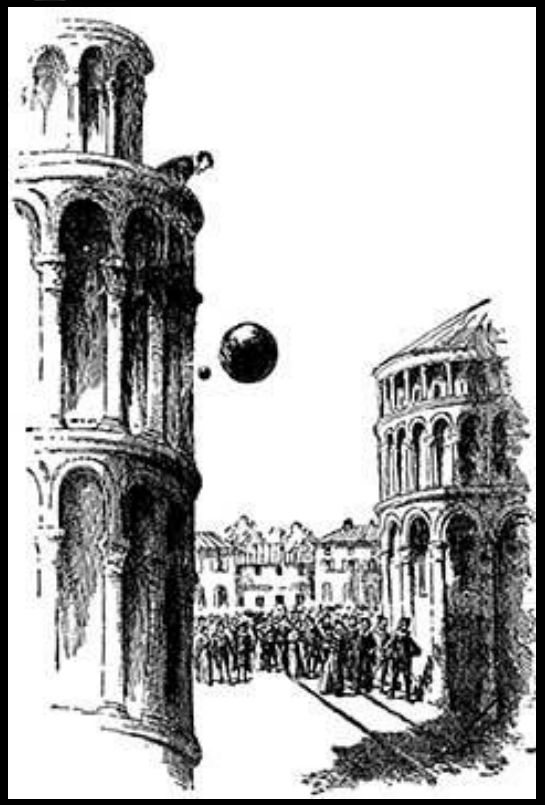


Galileo didn't know calculus (*because Newton and Leibniz hadn't discovered it yet*) so he couldn't derive the equation mathematically. Since we do know calculus (SymPy) we know that acceleration is the variation of velocity with time.



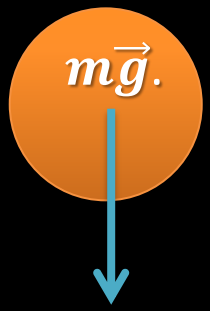
$$\vec{F}_{gravity} = m\vec{g}.$$

Free Fall of an Object: An Experiment by Galileo



We assume no drag/air resistance

$$\vec{F}_{gravity} = m\vec{g}$$



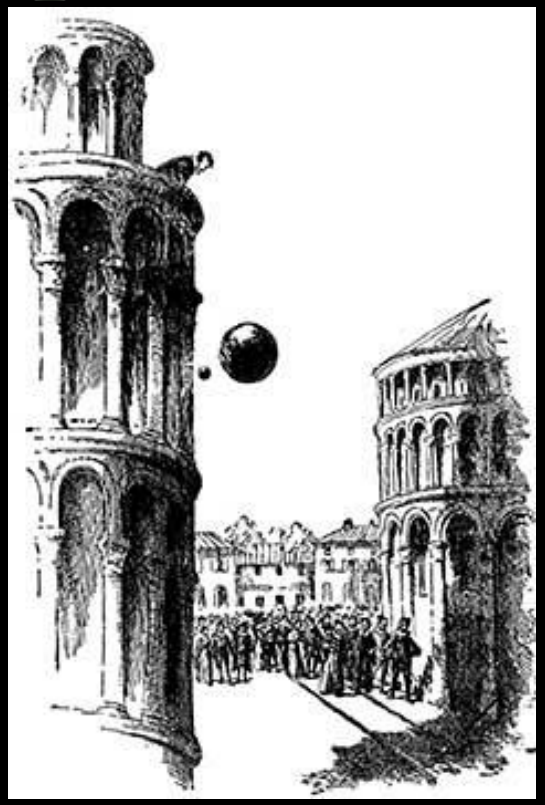
$$-\vec{g} = \frac{dv}{dt} = \frac{d^2x}{dt^2}$$

$$dv = -\vec{g}dt$$

$$v = \int -\vec{g}dt$$

$$v = -\vec{g}t$$

Free Fall of an Object: An Experiment by Galileo



We assume no drag/air resistance

$$\int \boldsymbol{vdt} = \int -\vec{\boldsymbol{g}}\boldsymbol{tdt}$$

$$\boldsymbol{x} = -\frac{\boldsymbol{1}}{\boldsymbol{2}}\vec{\boldsymbol{g}}\boldsymbol{t}^{\boldsymbol{2}}$$

Free Fall of an Object: An Experiment by Galileo

```
from sympy import *
v, x, t, g = symbols('v x t g', real = True)
m= symbols('m', constant = True)
init_printing(use_unicode = True)

v = integrate(-m*g, t)
print('Velocity : ', v)
x = integrate(v, t)
print('Position : ', x)

Velocity :  -g*m*t
Position :  -g*m*t**2/2
```

SymPy : a Python library for symbolic mathematics.

Try to write these equations

$$slope = \frac{y_2 - y_1}{x_2 - x_1}$$

$$resist = \frac{1}{\frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3}}$$

$$factor = 1 + \frac{b}{v} + \frac{c}{v^2}$$

$$center = \frac{38.1972 \cdot (r^3 - s^3) \sin a}{(r^2 - s^2) \cdot a}$$

Input Function

The input function is used in an assignment statement. To call it, a string is passed, which is the prompt that will appear on the screen, and whatever the user types will be stored in the variable named on the left of the assignment statement. To make it easier to read the prompt, put a colon and then a space after the prompt. For example,

```
>>> rad = input('Enter the radius: ')
Enter the radius: 5
rad =
'5'
>>> name = input('What is your name : ')
What is your name : Serdar
name =
'Serdar'
```

Print Function

The simplest output function in Python is print, which is used to display the result of an expression or a string without assigning any value. For example,

```
>>> print('Hello')
Hello
>>> print(4**3)
64
```

Control Statements

We can create “logical” or “conditional” expressions using comparison and Boolean operators. Such expressions always produce a numerical result that is either 1 for true or 0 for false expressions. The comparison operators are ‘<’ ’<=’ ’==’ ‘>=’ ‘>’ and ‘!=’. The Boolean operators are ‘and’ and ‘or’ or, ‘not’ not. You can use parentheses to bracket expressions to force an evaluation order.

```
>>> x = 10
>>> y = 20
>>> print(x < y)                    # displays True
>>> print(x <= 10)                  # displays True
>>> print (x == y)                  # displays False
>>> print ((0 < x) and (y < 30))    # displays True
>>> print((x > 10) or (y > 100))    # displays False
>>> print(not(x > 10))                   # displays True
```

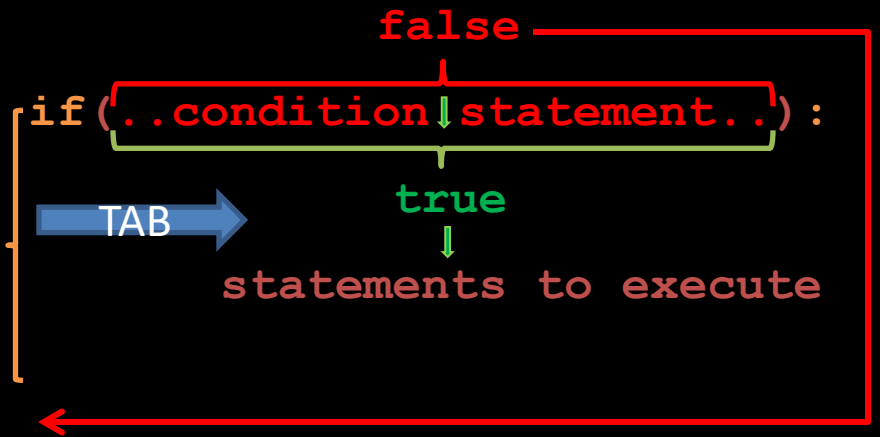
```
>>> 1 < 2     # Less than
True
>>> 2.0 >= 1  # Greater than or equal: mixed-type 1
converted to 1.0
True
>>> 2.0 == 2.0       # Equal value
True
>>> 2.0 != 2.0       # Not equal value
False
>>> X = 2
>>> Y = 4
>>> Z = 6
>>> X < Y < Z # Chained comparisons: range tests
True
>>> X < Y and Y < Z
True
```

The Boolean operators are and, or, not. Such expressions always produce a numerical result that is either True or False expressions.

and	0 and 0 results False	not (0 and 0) results True
	1 and 0 results False	not (1 and 0) results True
	0 and 1 results False	not (0 and 1) results True
	1 and 1 results True	not (1 and 1) results False

or	0 or 0 results False	not (0 or 0) results True
	1 or 0 results True	not (1 or 0) results False
	0 or 1 results True	not (0 or 1) results False
	1 or 1 results True	not (1 or 1) results False

The conditional statements "if condition statement end" only executes the statements if the condition evaluates to true. The condition should be logical expression evaluating to true or false. For example:




```
if(x < 10):
    print(x)        # only displays x when x < 10


if((x > 0) and (x < 100)):
    print('OK')# displays OK if x between 0 and 100
```

You can put more than one statement between the 'if' and the end of indentation. You can choose between two courses of action with "if *condition statement* else in:

```
if((x > 0) and (x < 100)):
    print('OK')
else:
    print('x contains invalid number')
```

You can build a chain of test with 'elif', as in:

```
if(n <= 0):
    print('n is negative or zero')
elif (n%2):
    print('n is even')
else:
    print('n is odd')
```

if/else branches

Example:

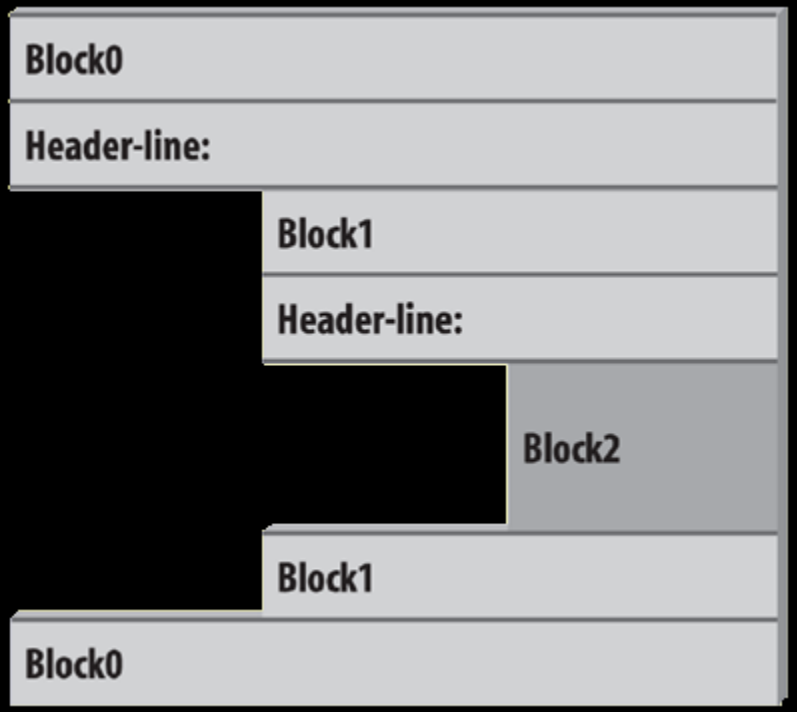
```
# age example 1
age = int(input("How old are you? "))
if age < 13:
        print ("Your age is ", age, "you are a children.")
else:
        print ("Your age is ", age, "you are an adult.")
```

Output:

```
>>> How old are you? 5
your age is 5 you are a children
>>> How old are you? 55
your age is 55 you are a adult
```

Python detects block boundaries automatically, by line *indentation*–that is, the empty space to the left of your code.

Nested blocks of code




```
x = 1
if x:
    y =2
    if y:
        print('block2')
    print('block1')
print('block0')
```

This code contains three blocks: the **first** is not indented at all, the **second** is indented four spaces, and the **third** is indented eight spaces.

if/elif/else branches

Example:

```
# age example 2
age = int(input("How old are you? "))
if age < 13:
      print ("Your age is ", age, "you are a
children.")
elif age >13 and age < 20:
      print ("Your age is ", age, "you are a
    teenager.")
else:
      print ("Your age is ", age, "you are an
adult.")
```

What is wrong with this code?

Please write a program that calculates Body Mass Index and gives the result accordingly.

$$\text{BMI} = \frac{mass[kg]}{(height[m])^2}$$

< 18 → underweight
< 18.5 → thin for height
18.6 - 24.9 → healthy weight
25 - 29.9 → overweight
> 30 → obesity

While statements have the following basic structure:

```
>>> while condition:
        action
>>>
```

As long as the condition is true, the while statement will execute the action

Example:

```
>>> x = 1
>>> while x < 4:          # as long as x < 4...
        print( x**2)      # print the square of x
        x = x+1           # increment x by +1

1          # only the squares of 1, 2, and 3 are printed, because
4          # once x = 4, the condition is false
9
>>>
```

Pitfall to avoid:

While statements are intended to be used with changing conditions. If the condition in a while statement does not change, the program will be stuck in an infinite loop until the user hits **ctrl-C**.

Example:

```
>>> x = 1
>>> while x == 1:
        print ('Hello world')
```

Since x does not change, Python will continue to print "Hello world" until interrupted

for loop: Repeats a set of statements over a group of values.

Syntax:

```
for variableName in groupOfValues:
  statements
```

We indent the statements to be repeated with tabs or spaces.

variableName gives a name to each value, so you can refer to it in the statements.

groupOfValues can be a range of integers, specified with the range function.

Example:

```
for x in range(1, 6):
  print (x, "squared is", x **2)
```

Output:

1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25

The range function specifies a range of integers:

`range(start, stop)` the integers between **start** (inclusive) and **stop** (exclusive)

It can also accept a third value specifying the change between values.range(start, stop, step) - the integers between **start** (inclusive) and **stop** (exclusive) by **step**

Example:

```
for x in range(5, 0, -1):
    print(x)
print ("Blastoff!")
```

Output:

```
5
4
3
2
1
Blastoff!
```

Example:

```
>>> for i in range(1,7):
        print(i, i**2, i**3, i**4)
1 1 1 1
2 4 8 16
3 9 27 81
4 16 64 256
5 25 125 625
6 36 216 1296

>>> L = [0,1,2,3]  # or, equivalently, range(4)
>>> for i in range(len(L)):
        L[i] = L[i]**2
>>> L
[0,1,4,9]
>>>
```

Some loops incrementally compute a value that is initialized outside the loop. This is sometimes called a cumulative sum.

```
sum = 0
for i in range(1, 11):
    sum = sum + (i **2)
print ("sum of first 10 squares is", sum)
```

Output:

```
sum of first 10 squares is 385
```

Exercise: Write a Python program that computes the factorial of an integer.

Please write a simple averaging program

Sample output:

```
Lütfen 10 adet sayı giriniz
1.sayıyı giriniz : 1
2.sayıyı giriniz : 2
3.sayıyı giriniz : 3
4.sayıyı giriniz : 4
5.sayıyı giriniz : 5
6.sayıyı giriniz : 6
7.sayıyı giriniz : 7
8.sayıyı giriniz : 8
9.sayıyı giriniz : 9
10.sayıyı giriniz : 10
10 sayının toplamı 55 ve ortalaması : 5.5
```

Write program that asks two integer values n and m from the user, then produces a box that is n wide and m deep, such as the following:

```
Enter a width: 5
Enter a height: 4
*****
*   *
*   *
*****
```

Guessing Numbers : The problem is to guess what number a computer has in mind. You will write a program that randomly generates an integer between 0 and 100, inclusive. For each user input, the program reports whether it is too low or too high, so the user can choose the next input intelligently. Here is a sample run:

```
Guess a magic number between 0 and 100
Enter your guess: 50
Your guess is too high
Enter your guess: 25
Your guess is too low
Enter your guess: 42
Your guess is too high
Enter your guess: 39
Yes, the number is 39
```

Write a Python script that calculates the average value of given numbers. Please write the same code with - (minus) number enterence

This is Output of the script

```
enter your number: 12
enter your number: -1
average is 12.0

enter your number: 12
enter your number: 9
enter your number: 18
enter your number: -1
average is 13.0
```