

# Ch.1: Computing with formulas

Hans Petter Langtangen<sup>1,2</sup>

<sup>1</sup>Simula Research Laboratory

<sup>2</sup>University of Oslo, Dept. of Informatics

Aug 21, 2016

## Why program?

*Everybody in this country should learn how to program a computer... because it teaches you how to think.* Steve Jobs, 1955-2011.

## The teaching strategy is example-based

- Present a case (example)
- Present the complete program
- Dissect and discuss every line
- Simulate programs by hand (be the computer!)

## The learning strategy is about doing exercises

- Study and try to understand examples
- Program a lot!
- This course has many compulsory exercises
- The course curriculum is defined through exercises

## Chapter 1 is about evaluating formulas

### Why?

- Everybody understands the problem
- Many fundamental concepts are introduced
  - variables
  - arithmetic expressions
  - objects
  - printing text and numbers

### Formulas and arithmetics are fundamental...

A physicist, a biologist and a mathematician were at a cafe when across the street two people entered a house. Moments later three people came out. The physicist said, “Hmm, that must be a measurement error.” The biologist wondered, “It must be reproduction!” And the mathematician said, “If someone goes into the house, it will be empty again.”

### Evaluating a mathematical formula

#### Height of a ball in vertical motion.

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

where

- $y$  is the height (position) as function of time  $t$
- $v_0$  is the initial velocity at  $t = 0$
- $g$  is the acceleration of gravity

Task: given  $v_0$ ,  $g$  and  $t$ , compute  $y$ .

### Use a calculator? A program is much more powerful!

**What is a program?** A sequence of instructions to the computer, written in a programming language, somewhat like English, but very much simpler - and very much stricter.

This course teaches the Python language.

**Our first example program:** Evaluate  $y(t) = v_0 t - \frac{1}{2}gt^2$  for  $v_0 = 5$ ,  $g = 9.81$  and  $t = 0.6$ :

$$y = 5 \cdot 0.6 - \frac{1}{2} \cdot 9.81 \cdot 0.6^2$$

The complete Python program:

```
print 5*0.6 - 0.5*9.81*0.6**2
```

## How to write and run the program

- A program is plain text, written in a *plain text editor*
- Use Gedit, Emacs, Vim, Spyder, or IDLE (*not* MS Word!)

**Step 1.** Write the program in a text editor, here the line

```
print 5*0.6 - 0.5*9.81*0.6**2
```

**Step 2.** Save the program to a file (say) `ball1.py`. (`.py` denotes Python.)

**Step 3.** Move to a terminal window and go to the folder containing the program file.

**Step 4.** Run the program:

```
Terminal> python ball1.py
```

The program prints out 1.2342 in the terminal window.

## In this course we probably use computers differently from what you are used to

- When you use a computer, you always run some programs
- The computer cannot do anything without being precisely told what to do, and humans write and use programs to tell the computer what to do
- Most people are used to double-click on a symbol to run a program - in this course we give commands in a terminal window because that is more efficient if you work intensively with programming
- Hard math problems suddenly become straightforward by writing programs

## A short program can calculate any integral

You cannot calculate this integral by hand:

$$\int_{-\infty}^1 e^{-x^2} dx.$$

A little program can compute this and “all” other integrals:

```
from numpy import *

def integrate(f, a, b, n=100):
    """
    Integrate f from a to b,
    using the Trapezoidal rule with n intervals.
    """
    x = linspace(a, b, n+1)    # Coordinates of the intervals
    h = x[1] - x[0]            # Interval spacing
    I = h*(sum(f(x)) - 0.5*(f(a) + f(b)))
    return I

# Define my special integrand
def my_function(x):
    return exp(-x**2)

minus_infinity = -20 # Approximation of minus infinity
I = integrate(my_function, minus_infinity, 1, n=1000)
print 'Value of integral:', I

The program computes an approximation with error  $10^{-12}$  within
0.1 s ( $n = 10^6$ )!
```

## Computers are very picky about grammar rules and typos

Look at

```
print 5*0.6 - 0.5*9.81*0.6**2
write 5*0,6 - 0,5*9,81*0,6^2
```

Would you consider these two lines to be equal?

- Humans may say *yes*, computers always *no*
- The second line has no meaning as a Python program
- **write** is not a legal Python word in this context, comma has another meaning than in math, and the hat is not exponentiation
- We have to be extremely accurate with how we write computer programs!
- It takes time and experience to learn this

## Programming opens up a new life

*People only become computer programmers if they're obsessive about details, crave power over machines, and can bear to be told day after day exactly how stupid they are.* G. J. E. Rawlins

## Store numbers in variables to make a program more readable

From mathematics you are used to variables, e.g.,

$$v_0 = 5, \quad g = 9.81, \quad t = 0.6, \quad y = v_0 t - \frac{1}{2} g t^2$$

We can use variables in a program too, and this makes the last program easier to read and understand:

```
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2
print y
```

This program spans several lines of text and use variables, otherwise the program performs the same calculations and gives the same output as the previous program

## There is great flexibility in choosing variable names

- In mathematics we usually use one letter for a variable
- The name of a variable in a program can contain the letters a-z, A-Z, underscore \_ and the digits 0-9, but cannot start with a digit
- Variable names are case-sensitive (e.g., a is different from A)

```
initial_velocity = 5
accel_of_gravity = 9.81
TIME = 0.6
VerticalPositionOfBall = initial_velocity*TIME - \
    0.5*accel_of_gravity*TIME**2
print VerticalPositionOfBall
```

(Note: the backslash allows an instruction to be continued on the next line)

Good variable names make a program easier to understand!

## Some words are reserved in Python

Certain words have a special meaning in Python and cannot be used as variable names. These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, and `yield`.

## Comments are useful to explain how you think in programs

```
# program for computing the height of a ball
# in vertical motion
v0 = 5    # initial velocity
g = 9.81  # acceleration of gravity
t = 0.6   # time
y = v0*t - 0.5*g*t**2 # vertical position
print y
```

### Note:

- Everything after `#` on a line is a comment and ignored by Python
- Comments are used to explain what the computer instructions mean, what variables mean, how the programmer reasoned when she wrote the program, etc.
- Bad comments say no more than the code: `a = 5 # set a to 5`

## Comments are not always ignored....

Normal rule: Python programs, including comments, can only contain characters from the English alphabet.

```
hilsen = 'æKjre Åsmund!' # er æ og Å lov i en streng?
print hilsen
```

leads to an error:

```
SyntaxError: Non-ASCII character ...
```

Remedy: put this special comment line as the first line in your program

```
# -*- coding: utf-8 -*-
```

Or stick to English everywhere in a program

## The printf syntax gives great flexibility in formatting text with numbers

Output from calculations often contain text and numbers, e.g.,

```
At t=0.6 s, y is 1.23 m.
```

We want to control the formatting of numbers: no of decimals, style: 0.6 vs 6E-01 or 6.0e-01. So-called *printf formatting* is useful for this purpose:

```
t = 0.6; y = 1.2342
print 'At t=%g s, y is %.2f m.' % (t, y)
```

The printf format has “slots” where the variables listed at the end are put:  
 $\%g \leftarrow t, \%.2f \leftarrow y$

## Examples on different printf formats

<code>%g</code>	most compact formatting of a real number
<code>%f</code>	decimal notation (-34.674)
<code>%10.3f</code>	decimal notation, 3 decimals, field width 10
<code>%.3f</code>	decimal notation, 3 decimals, minimum width
<code>%e</code> or <code>%E</code>	scientific notation (1.42e-02 or 1.42E-02)
<code>%9.2e</code>	scientific notation, 2 decimals, field width 9
<code>%d</code>	integer
<code>%5d</code>	integer in a field of width 5 characters
<code>%s</code>	string (text)
<code>%-20s</code>	string, field width 20, left-adjusted
<code>%%</code>	the percentage sign % itself

(See the the book for more explanation and overview)

## Using printf formatting in our program

Triple-quoted strings (""" ) can be used for multi-line output, and here we combine such a string with printf formatting:

```
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2

print """
At t=%f s, a ball with
initial velocity v0=%.3E m/s
is located at the height %.2f m.
""" % (t, v0, y)
```

Running the program:

```
Terminal> python ball_print2.py
```

```
At t=0.600000 s, a ball with  
initial velocity v0=5.000E+00 m/s  
is located at the height 1.23 m.
```

## Some frequently used computer science terms

- Program or code or application
- Source code (program text)
- Code/program snippet
- Execute or run a program
- Algorithm (recipe for a program)
- Implementation (writing the program)
- Verification (does the program work correctly?)
- Bugs (errors) and debugging

Computer science meaning of terms is often different from the human language meaning

## A program consists of statements

```
a = 1      # 1st statement (assignment statement)  
b = 2      # 2nd statement (assignment statement)  
c = a + b  # 3rd statement (assignment statement)  
print c    # 4th statement (print statement)
```

Normal rule: one statement per line, but multiple statements per line is possible with a semicolon in between the statements:

```
a = 1; b = 2; c = a + b; print c
```

**Assignment statements evaluate right-hand side and assign the result to the variable on the left-hand side**



```
myvar = 10
myvar = 3*myvar    # = 30
```

## Syntax is the exact specification of instructions to the computer

Programs must have correct syntax, i.e., correct use of the computer language grammar rules, and no misprints!

```
myvar = 5.2
prinnt Myvar
```

```
    prinnt Myvar
           ^
```

```
SyntaxError: invalid syntax
```

Only the first encountered error is reported and the program is stopped (correct the error and continue with next error)

*Programming demands significantly higher standard of accuracy. Things don't simply have to make sense to another human being, they must make sense to a computer.* Donald Knuth, computer scientist, 1938-

## Blanks (whitespace) can be used to nicely format the program text

Blanks may or may not be important in Python programs. These statements are equivalent (blanks do not matter):

```
v0=3
v0 = 3
v0= 3
v0 = 3
```

Here blanks do matter:

```
counter = 1
while counter <= 4:
    counter = counter + 1    # correct (4 leading blanks)
```

```
while counter <= 4:
    counter = counter + 1      # invalid syntax
```

(more about this in Ch. 2)

A program takes some known *input* data and computes some *output* data

```
v0 = 3;  g = 9.81;  t = 0.6
position = v0*t - 0.5*g*t*t
velocity = v0 - g*t
print 'position:', position, 'velocity:', velocity
```

- Input: `v0`, `g`, and `t`
- Output: `position` and `velocity`

An operating system (OS) is a set of programs managing hardware and software resources on a computer

- Linux, Unix (Ubuntu, RedHat, Suse, Solaris)
- Windows (95, 98, NT, ME, 2000, XP, Vista, 7, 8)
- Macintosh (old Mac OS, Mac OS X)
- Mac OS X  $\approx$  Unix  $\approx$  Linux  $\neq$  Windows
- Typical OS commands are quite similar:
  - Linux/Unix: `mkdir folder; cd folder; ls`
  - Windows: `mkdir folder; cd folder; dir`
- Python supports cross-platform programming, i.e., a program is independent of which OS we run the program on

## Evaluating a formula for temperature conversion

Given  $C$  as a temperature in Celsius degrees, compute the corresponding Fahrenheit degrees  $F$ :

$$F = \frac{9}{5}C + 32$$

Program:

```
C = 21
F = (9/5)*C + 32
print F
```

Execution:

```
Terminal> python c2f_v1.py
53
```

**We must always check that a new program calculates the right answer**

**Using a calculator:**  $9/5$  times 21 plus 32 is 69.8, not 53.

## The error is caused by (unintended) integer division

- $9/5$  is not 1.8 but 1 in most computer languages (!)
- If  $a$  and  $b$  are integers,  $a/b$  implies integer division: the largest integer  $c$  such that  $cb \leq a$
- Examples:  $1/5 = 0$ ,  $2/5 = 0$ ,  $7/5 = 1$ ,  $12/5 = 2$
- In mathematics,  $9/5$  is a real number (1.8) - this is called float division in Python and is the division we want
- One of the operands ( $a$  or  $b$ ) in  $a/b$  must be a real number ("float") to get float division
- A float in Python has a dot (or decimals): `9.0` or `9.` is float
- No dot implies integer: `9` is an integer
- `9.0/5` yields 1.8, `9/5.` yields 1.8, `9/5` yields 1

Corrected program (with correct output 69.8):

```
C = 21
F = (9.0/5)*C + 32
print F
```

## Everything in Python is an object

Variables refer to objects:

```
a = 5          # a refers to an integer (int) object
b = 9          # b refers to an integer (int) object
c = 9.0        # c refers to a real number (float) object
d = b/a        # d refers to an int/int => int object
e = c/a        # e refers to float/int => float object
s = 'b/a=%g' % (b/a) # s is a string/text (str) object
```

We can convert between object types:

```
a = 3          # a is int
b = float(a)   # b is float 3.0
c = 3.9        # c is float
d = int(c)     # d is int 3
d = round(c)   # d is float 4.0
d = int(round(c)) # d is int 4
d = str(c)     # d is str '3.9'
e = '-4.2'     # e is str
f = float(e)   # f is float -4.2
```

## Arithmetic expressions are evaluated as you have learned in mathematics

- Example:  $\frac{5}{9} + 2a^4/2$ , in Python written as `5/9 + 2*a**4/2`
- Same rules as in mathematics: proceed term by term (additions/subtractions) from the left, compute powers first, then multiplication and division, in each term
- `r1 = 5/9 (=0)`
- `r2 = a**4`
- `r3 = 2*r2`
- `r4 = r3/2`
- `r5 = r1 + r4`
- Use parenthesis to override these default rules - or use parenthesis to explicitly tell how the rules work:  
`(5/9) + (2*(a**4))/2`

## Standard mathematical functions are found in the `math` module

- What if we need to compute  $\sin x$ ,  $\cos x$ ,  $\ln x$ , etc. in a program?
- Such functions are available in Python's `math` module
- In general: lots of useful functionality in Python is available in modules - but modules must be *imported* in our programs

Compute  $\sqrt{2}$  using the `sqrt` function in the `math` module:

```
import math
r = math.sqrt(2)
# or
from math import sqrt
r = sqrt(2)
# or
from math import *    # import everything in math
r = sqrt(2)
```

## Another example on computing with functions from `math`

Evaluate

$$Q = \sin x \cos x + 4 \ln x$$

for  $x = 1.2$ .

```
from math import sin, cos, log
x = 1.2
Q = sin(x)*cos(x) + 4*log(x)    # log is ln (base e)
print Q
```

## Computers have inexact arithmetics because of rounding errors

Let us compute  $1/49 \cdot 49$  and  $1/51 \cdot 51$ :

```
v1 = 1/49.0*49
v2 = 1/51.0*51
print '%.16f %.16f' % (v1, v2)
```

Output with 16 decimals becomes

```
0.9999999999999999 1.0000000000000000
```

- Most real numbers are represented inexactly on a computer (17 digits)

- Neither  $1/49$  nor  $1/51$  is represented exactly, the error is typically  $10^{-16}$
- Sometimes such small errors propagate to the final answer, sometimes not, and sometimes the small errors accumulate through many mathematical operations
- Lesson learned: real numbers on a computer and the results of mathematical computations are only approximate

## Test that a calculation is correct

What is printed?

```
a = 1; b = 2;
computed = a + b
expected = 3
correct = computed == expected
print 'Correct:', correct
```

Change to  $a = 0.1$  and  $b = 0.2$  (expected = 0.3). What is now printed? Why? How can the comparison be performed?

## Answer to exercise on previous slide: use equality test with tolerance!

```
>>> a = 0.1; b = 0.2; expected = 0.3
>>> a + b == expected
False

>>> print '%.17f\n%.17f\n%.17f\n%.17f' % (0.1, 0.2, 0.1 + 0.2, 0.3)
0.10000000000000001
0.20000000000000001
0.30000000000000004
0.29999999999999999

>>> a = 0.1; b = 0.2; expected = 0.3
>>> computed = a + b
>>> diff = abs(expected - computed)
>>> tol = 1E-15
>>> diff < tol
```

## Another example involving math functions

The  $\sinh x$  function is defined as

$$\sinh(x) = \frac{1}{2} (e^x - e^{-x})$$

We can evaluate this function in three ways:

1. `math.sinh`
2. combination of two `math.exp`
3. combination of two powers of `math.e`

```
from math import sinh, exp, e, pi
x = 2*pi
r1 = sinh(x)
r2 = 0.5*(exp(x) - exp(-x))
r3 = 0.5*(e**x - e**(-x))
print '%.16f %.16f %.16f' % (r1,r2,r3)
```

Output: r1 is 267.7448940410164369, r2 is 267.7448940410164369, r3 is 267.7448940410163232 (!)

## Python can be used interactively as a calculator and to test statements

- So far we have performed calculations in Python *programs*
- Python can also be used interactively in what is known as a *shell*
- Type `python`, `ipython`, or `idle` in the terminal window
- A Python shell is entered where you can write statements after `>>>` (IPython has a different prompt)

```
Terminal> python
Python 2.7.6 (r25:409, Feb 27 2014, 19:35:40)
...
>>> C = 41
>>> F = (9.0/5)*C + 32
>>> print F
105.8
>>> F
105.8
```

Previous commands can be recalled and edited

## Python has full support for complex numbers

- $2 + 3i$  in mathematics is written as `2 + 3j` in Python

```

>>> a = -2
>>> b = 0.5
>>> s = complex(a, b) # make complex from variables
>>> s
(-2+0.5j)
>>> s*w          # complex*complex
(-10.5-3.75j)
>>> s/w          # complex/complex
(-0.25641025641025639+0.28205128205128205j)
>>> s.real
-2.0
>>> s.imag
0.5

```

See the book for additional info

## Python can also do symbolic computing

- Numerical computing: computation with numbers
- Symbolic computing: work with formulas (as in trad. math)

```

>>> from sympy import *
>>> t, v0, g = symbols('t v0 g')
>>> y = v0*t - Rational(1,2)*g*t**2
>>> dydt = diff(y, t) # 1st derivative
>>> dydt
-g*t + v0
>>> print 'acceleration:', diff(y, t, t) # 2nd derivative
acceleration: -g
>>> y2 = integrate(dydt, t)
>>> y2
-g*t**2/2 + t*v0

```

## SymPy can do a lot of traditional mathematics

```

>>> y = v0*t - Rational(1,2)*g*t**2
>>> roots = solve(y, t) # solve y=0 wrt t
>>> roots
[0, 2*v0/g]

>>> x, y = symbols('x y')
>>> f = -sin(x)*sin(y) + cos(x)*cos(y)
>>> simplify(f)
cos(x + y)
>>> expand(sin(x+y), trig=True) # requires a trigonometric hint
sin(x)*cos(y) + sin(y)*cos(x)

```



## Summary of Chapter 1 (part 1)

- Programs must be accurate!
- Variables are names for objects
- We have met different object types: `int`, `float`, `str`
- Choose variable names close to the mathematical symbols in the problem being solved
- Arithmetic operations in Python: term by term (+/-) from left to right, power before `*` and `/` - as in mathematics; use parenthesis when there is any doubt
- Watch out for unintended integer division!

## Summary of Chapter 1 (part 2)

Mathematical functions like  $\sin x$  and  $\ln x$  must be imported from the `math` module:

```
from math import sin, log
x = 5
r = sin(3*log(10*x))
```

Use `printf` syntax for full control of output of text and numbers!

Important terms: object, variable, algorithm, statement, assignment, implementation, verification, debugging

## Programming is challenging

- *You think you know when you can learn,  
are more sure when you can write,  
even more when you can teach,  
but certain when you can program*
- *Within a computer, natural language is unnatural*
- *To understand a program you must become both the machine  
and the program*

Alan Perlis, computer scientist, 1922-1990.

### Summarizing example: throwing a ball (problem)

We throw a ball with velocity  $v_0$ , at an angle  $\theta$  with the horizontal, from the point  $(x = 0, y = y_0)$ . The trajectory of the ball is a parabola (we neglect air resistance):

$$y = x \tan \theta - \frac{1}{2v_0} \frac{gx^2}{\cos^2 \theta} + y_0$$

- Program tasks:
  - initialize input data  $(v_0, g, \theta, y_0)$
  - import from `math`
  - compute  $y$
- We give  $x, y$  and  $y_0$  in m,  $g = 9.81\text{m/s}^2$ ,  $v_0$  in km/h and  $\theta$  in degrees - this requires conversion of  $v_0$  to m/s and  $\theta$  to radians

### Summarizing example: throwing a ball (solution)

Program:

```
g = 9.81      # m/s**2
v0 = 15       # km/h
theta = 60    # degrees
x = 0.5       # m
y0 = 1        # m

print ""v0      = %.1f km/h
theta = %d degrees
y0     = %.1f m
x      = %.1f m"" % (v0, theta, y0, x)

# convert v0 to m/s and theta to radians:
v0 = v0/3.6
from math import pi, tan, cos
theta = theta*pi/180

y = x*tan(theta) - 1/(2*v0)*g*x**2/((cos(theta))**2) + y0

print 'y      = %.1f m' % y
```