Computing with Formulas

INF1100 Lectures, Chapter 1:
Hans Petter Langtangen
Simula Research Laboratory
University of Oslo, Dept. of Informatics

Programming a mathematical formula

We will learn programming through examples
The first examples involve programming of formulas
Here is a formula for the position of a ball in vertical motion, starting at $y = 0$ at time $t = 0$:

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

$y$ is the height (position) as function of time $t$
$v_0$ is the initial velocity (at $t = 0$)
$g$ is the acceleration of gravity
Computational task: given $v_0$, $y$ and $t$, compute $y$

The program

What is a program?
A sequence of instructions to the computer, written in a programming language, which is somewhat like English, but very much simpler -- and very much stricter! In this course we shall use the Python language.

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

\[ y = 5 \cdot 0.6 - \frac{1}{2} \cdot 9.81 \cdot (0.6)^2 \]

Python program for doing this calculation:

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

How to write and run the program

A (Python) program is plain text
First we need to write the text in a plain text editor
Use Gedit, Emacs or IDLE (not MS Word or OpenOffice!)

Write the program line
```
print 5*0.6 - 0.5*9.81*0.6**2
```

Save the program to a file (say) `ball_numbers.py`
Python programs are (usually) stored files ending with `.py`

Go to a terminal window
Go to the folder containing the program (text file)
Give this operating system command:
```
Unix/OSS> python ball_numbers.py
```

The program prints out 1.2342 in the terminal window

About programs and programming

Would you consider these two lines to be "equal"?

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

Humans will say "yes", computers "no"

The second line has no meaning as a Python program
`write` is not a legal Python word in this context, and the hat does not imply $0.6^2$

We have to be extremely accurate with how we write computer programs!
It takes time and experience to learn this

"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

Computers are very picky about grammar rules and typos

Storing numbers in variables

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:

```python
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

Names of variables

In mathematics we usually use one letter for a variable
In a program it is smart to use one-letter symbols, words or abbreviation of words as names of variables

The name of a variable can contain the letters a-z, A-Z, underscore _ and the digits 0-9, but the name cannot start with a digit

Variable names are case-sensitive (e.g., `a` is different from `A`)

Example on other variable names in our last program:

```python
initial_velocity = 5
accel_of_gravity = 9.81
TIME = 0.6
VerticalPositionOfBall = initial_velocity*TIME - \frac{1}{2}*accel_of_gravity*TIME**2
print VerticalPositionOfBall
```

(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
- There are many rules about programming and Python, we learn them as we go along with examples

Comments are useful to explain how you think in programs

- Program with comments:
  ```python
  # program for computing the height of a ball
  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
  ```
- Everything after a # on a line is ignored by the computer and is known as a comment where we can write whatever we want
- Comments are used to explain what the computer instructions mean, what variables mean, how the programmer reasoned when she wrote the program, etc.

Comments are not always ignored....

- Normal rule: Python programs, including comments, can only contain characters from the English alphabet
- Norwegian characters, hilsen = “Kjære Åsmund!” # we ø og Å lov i en streng? print hilsen
  - will normally lead to an error:
  ```python
  SyntaxError: Non-ASCII character ...
  ```
- Remedy: put this line as the first line in your program:
  ```python
  # -*- coding: latin-1 -*-
  ```
  (this special comment line is not ignored - Python reads it...)
- Another remedy: stick to English everywhere in a program

"printf-style" formatting of text and numbers

- Output from calculations often contain text and numbers, e.g.
  ```python
  At t=0.6 s, y is 1.23 m.
  ```
- We want to control the formatting of numbers
  ```python
  (no of decimals, style: %g, %s, %d, %e, %E, %f, %g)
  ```
- So-called "printf formatting is useful for this purpose:
  ```python
  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 = t, %2f = y

Examples on different printf formats

- ```python
  Examples:
  
  %g  most compact formatting of a real number
  %.1f, %.2f, %.3f  decimal notation, 1, 2, 3 decimals, field width 10
  %.3f  decimal notation, 3 decimals, minimum width
  %<.3f  scientific notation, 1.42E-02 or 1.42E-02
  %.2e  scientific notation, 2 decimals, field width 9
  %d  integer
  %5d integer in a field of width 5 characters
  %<d  integer in a field of width 5 characters
  %s  string (text)
  %<s  string, field width 20, left-adjusted
  
  See the the book for more explanation and overview
  ```

Example on printf formatting in our program

- ```python
  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

# -*- coding: latin-1 -*-
print "At t=\%g s, a ball with initial velocity v0=%.3E m/s is located at the height %.2f m.

# program for computing the height of a ball
v0 = 5

t = 0.6
y = v0*t - 0.5*g*t**2
print 'At t=%f s, y is %.2f m.' % (t, y)

# program for computing the height of a ball
v0 = 5

t = 0.6
y = v0*t - 0.5*g*t**2
print 'At t=%f s, y is %.2f m.' % (t, y)
```
### Syntax

- Programs must have correct syntax, i.e., correct use of the computer language grammar rules, and no misprints
- This is a program with two syntax errors:
  ```python
  myvar = 5.2
  print Myvar
  ```
  (print is an unknown instruction, Myvar is a non-existing variable)
- Python reports syntax errors:
  ```python
  print Myvar
  SyntaxError: invalid syntax
  ```
- Only the first encountered error is reported and the program is stopped (correct 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

### Blanks (whitespace)

- Blanks may or may not be important in Python programs
- These statements are equivalent (blanks do not matter):
  ```python
  v = 0
  v = 3
  v = 3
  ```
  (the last is the preferred formatting style of assignments)
- Here blanks do matter:
  ```python
  while counter <= 10:
    counter = counter + 1
    # correct (4 leading blanks)
  ```
- (more about this in Ch. 2)

### Input and output

- A program has some known input data and computes some (on beforehand unknown) output data
- Sample program:
  ```python
  v0 = 3; g = 9.81; t = 0.6
  position = v0*t - 0.5*9.81*t**2
  velocity = -9.81*t
  print 'position', position, 'velocity', velocity
  ```
- Input: v0, g, and t
- Output: position and velocity

### Operating system

- An operating system (OS) is a set of programs managing hardware and software resources on a computer
- Example:
  ```python
  Unix/DOS> emacs myprog.py
  ```
  emacs is a program that needs help from the OS to find the file myprog.py on the computer’s disk
- Linux, Unix (Ubuntu, RedHat, Suse, Solaris)
- Windows (95, 98, NT, ME, 2000, XP, Vista)
- Macintosh (old Mac OS, Mac OS X)
- Mac OS X = Unix = Linux = Windows
- Python supports cross-platform programming, i.e., a program is independent of which OS we run the program on

### New formula: temperature conversion

- Given $C$ as a temperature in Celsius degrees, compute the corresponding Fahrenheit degrees $F$:
  $$F = \frac{9}{5}C + 32$$

- Program:
  ```python
  C = 22
  F = (9/5)*C + 32
  print F
  ```

- Execution:
  ```python
  Unix/DOS> python c2f_v1.py
  ```
  53

- We must always check that a new program calculates the right answer(s); a calculator gives 69.8, not 53

- Where is the error?

### 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 $b \leq a$
- Examples: $1/0 = 0$, $2/3 = 0.75$, $5 = 1/0$, $12/5 = 2$
- 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 $b$) 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):
  ```python
  C = 22
  F = (9/5)*C + 32
  print F
  ```

### Objects

- Everything in Python is an object
- Variables refer to objects:
  ```python
  a = 5  # a refers to an integer (int) object
  b = float(a)  # b refers to a float number (float) object
  c = 9.0  # c refers to a real number (float) object
  d = int(c)  # d is a float number (float) object
  e = c/d  # e refers to float/int => float object
  f = str(e)  # f refers to string/text (str) object
  ```

- We can convert between object types:
  ```python
  a = 3  # a is int
  b = float(a)  # b is float
  c = 3.9  # c is float
  d = int(c)  # d is int
  d = round(c)  # d is float
  d = abs(round(c))  # d is int
  d = str(c)  # d is string
  e = "4.2"  # e is str
  f = float(e)  # f is float
  ```

### How are arithmetic expressions evaluated?

- Example: $\frac{1}{2} + 2\times 4/2$, in Python written as $5/9 + 2\times 4/2$
- The rules are the same as in mathematics: proceed term by term (additions/subtractions) from the left, compute powers first, then multiplication and division, in each term
- Example:
  ```python
  r1 = 5/9
  r2 = 2 + 4
  r3 = 2 + r2
  r4 = 2 + r3
  ```
- Use parentheses to override these default rules – or use parenthesis to explicitly tell how the rules work (smart):
  $$(5/9) + (2\times (4+4))/2$$
**Standard mathematical functions**

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:
  ```python
  import math
  x = math.sqrt(2)
  # or
  from math import sqrt
  x = sqrt(2)
  # or
  from math import *  # import everything in math
  x = sqrt(2)
  ```

- Another example:
  ```python
  from math import sin, cos, log
  x = 1.2
  print sin(x) + cos(x) + 4 * log(x)  # log is ln (base e)
  ```

**A glimpse of round-off errors**

Let us compute \( 1/49 \cdot 49 \) and \( 1/51 \cdot 51 \):

```python
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.

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.

**Another example involving math functions**

The \( \sinh x \) function is defined as

\[
\sinh(x) = \frac{e^x - e^{-x}}{2}
\]

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`

```python
from math import sinh, exp, e
x = 2.5
r1 = sinh(x)
r2 = 0.5*(exp(x) - exp(-x))
r3 = e**x - e**(-x)
```

Output:

```
r1 is 267.7448940410164369
r2 is 267.7448940410164369
r3 is 267.7448940410163232
``` (!)

**Interactive Python shells**

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
```

A Python shell is entered where you can write statements after `>>>` (IPython has a different prompt).

Example:

```python
import numpy
a = numpy.array([1, 2, 3, 4])
b = numpy.array([5, 6, 7, 8])
c = a + b
```

**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 = 2.5
z = sin(3*log(10*x))
```

Use printf syntax for full control of output of text and numbers.

```
>>> a = 5.0, b = -5.0, c = 1.9456, d = 33
>>> print 'a = %a, b = %a, c = %a, d = %a', a, b, c, d
```

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

**Programming is challenging**

Alan Perlis, computer scientist, 1922-1990:

"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."
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):

\[
y = x \tan \theta - \frac{1}{2} \frac{g x^2}{v_0^2 \cos^2 \theta} + y_0
\]

Let us program this formula.

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:

\[
g = 9.81 \quad \# \text{m/s}^2
\]
\[
v_0 = 15 \quad \# \text{km/h}
\]
\[
\theta = 60 \quad \# \text{degrees}
\]
\[
y_0 = 1 \quad \# \text{m}
\]

\[
y = x \tan \theta - \frac{1}{2} \frac{g x^2}{v_0^2 \cos^2 \theta} + y_0
\]

# convert \( v_0 \) to m/s and \( \theta \) to radians:

\[
v_0 = v_0/3.6
\]
\[
\theta = \theta \pi/180
\]

Let us program this formula.

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.