Computing with formulas

Foundation of programming (CK0030)

Francesco Corona

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

FdP

- ☹ **Intro to variables, objects, modules, and text formatting**
- ☹ Programming with WHILE- and FOR-loops, and lists
- ☹ Functions and IF-ELSE tests

- ☹ Data reading and writing
- ☹ Error handling
- ☹ Making modules

- ☹ Arrays and array computing
- ☹ Plotting curves and surfaces

MAR 20, 2019 _ FC _

A formula

Computing with formulas

A formula

Example

The vertical motion of a ball thrown up in the air

We can set up a mathematical model for the motion of the ball

- From Newton's second law of motion

The vertical position of the ball, called y, varies with time t

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

⇝ v_0 is the initial velocity of the ball

⇝ g is the acceleration of gravity

⇝ t is time

The y axis is chosen such that the ball starts from $y = 0$ at $t = 0$

A formula (cont.)

The time for the ball to move upwards and return to the ground again

We are interested in the solutions to equation $y(t) = 0$

$$v_0 t - \frac{1}{2} g t^2 = t\left(v_0 - \frac{1}{2} g t\right) = 0 \rightsquigarrow \begin{cases} t = 0 \\ t = 2\dfrac{v_0}{g} \end{cases} \tag{2}$$

The ball returns to ground level in $2v_0/g$ (seconds)

- We can focus in the interval $t \in [0, 2v_0/g]$

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

A formula (cont.)

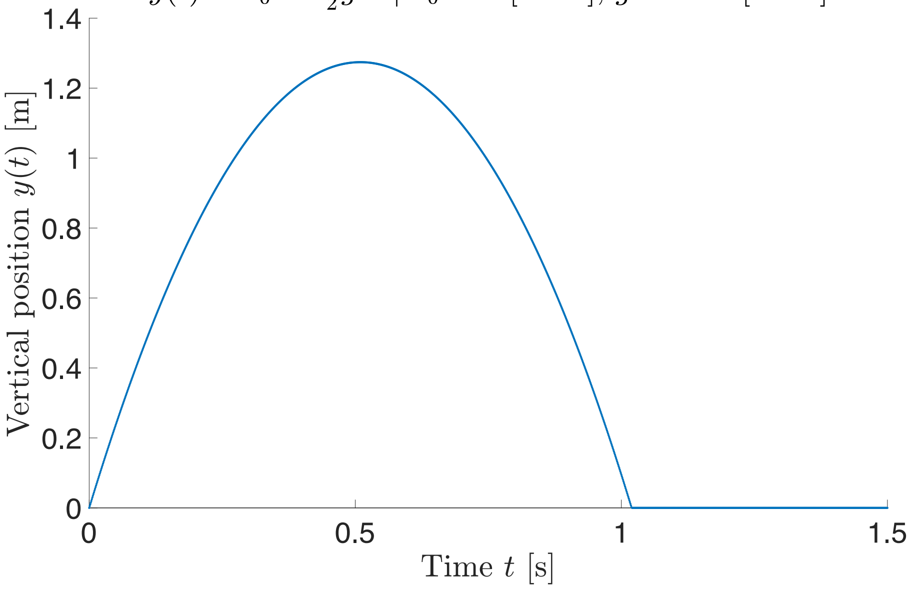
$$y(t) = v_0 t - \frac{1}{2} g t^2$$

We evaluate the formula for some values of v_0 and g

- $v_0 = 5 \text{ ms}^{-1}$
- $g = 9.81 \text{ ms}^{-2}$

We want to compute the ball's height for $t = 0.6$ s

$y(t) = v_0 t - \frac{1}{2} g t^2 \mid v_0 = 5$ [ms^{-1}], $g = 9.81$ [ms^{-2}]



A formula (cont.)

$$y = \underbrace{5}_{v_0} \cdot \underbrace{0.6}_{t} - \frac{1}{2} \cdot \underbrace{9.81}_{g} \cdot \underbrace{0.6^2}_{t^2} \quad (3)$$

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

Remark

The four **standard arithmetic operators**

⇝ +, -, * and /

Exponentiation employs a double asterisk ** notation

The arithmetic expression is easily evaluated and printed

- A one-line Python program

The ball comes back after some time $t = 2v_0/g \approx 1$ [s]

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Programs and programming

A formula

Programs and programming

Our task is to create programs/code and run it

There are three main types of tools for writing Python code

- A plain text editor
- An IPython notebook
- An integrated development environment (IDE) with a text editor

Remark

What you choose depends on how you access Python

There are various possibilities

- Access a plain installation on your own computer
- Access a pre-installed environment (distribution)
- Access Python in a cloud service

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Programs and programming (cont.)

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

- $\rightsquigarrow$ $t = 0.6$s
- $\rightsquigarrow$ $v_0 = 5 \text{ ms}^{-1}$
- $\rightsquigarrow$ $g = 9.81 \text{ ms}^{-2}$

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

This line is a **complete Python program** for evaluating the formula

- *Copy the line in a* **text file**
- *Save the text file with name* `ball1.py`

Programs and programming (cont.)

The action required to run this program depends on the chosen tool

- Terminal window, IPython, Spyder, IPython notebook, ...

Example

```
Terminal> python ball1.py

1.2342
```

After execution of `ball1.py`, the output (`1.2342`) is printed to screen

[*Run me in a terminal, with/without ipython, in spyder, a notebook ...*]

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Programs and programming (cont.)

Suppose that you now want to evaluate the formula for $v_0 = 1$ and $t = 0.1$

1. One must first edit the program text
2. Then, program must be re-executed

Example

$$y = \underbrace{1}_{v_0} \cdot \underbrace{0.1}_{t} - \frac{1}{2} \cdot \underbrace{9.81}_{g} \cdot \underbrace{0.1^2}_{t^2}$$

First edit the program text

```
print 1*0.1 - 0.5*9.81*0.1**2
```

Then, re-execute the program

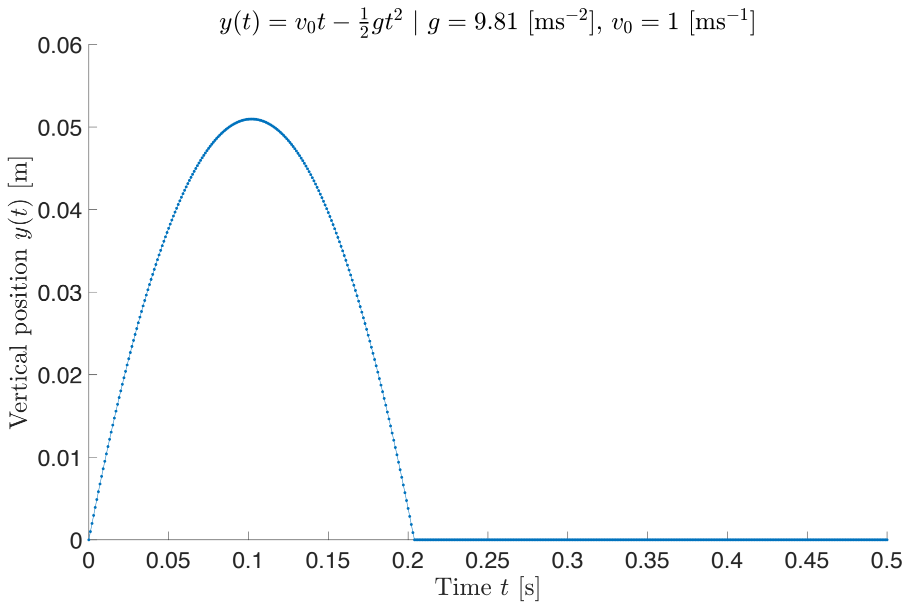
```
Terminal> python ball1.py

0.05095
```

The calculation has been changed

- The output is different
- `0.05095`

A formula (cont.)



Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Programs and programming (cont.)

We had to modify the value of t at two places in our program

Every time we want to evaluate $y(t)$ for different values of t

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

Such modifications could be made much simpler to perform

Programs and programming (cont.)

We must express formulas in terms of symbols called **variables**

- Rather than numerical values

Definition

Variables

*In Python, **variables** are defined by setting a name (here `v0`, `g`, `t`, or `y`) equal to a numerical value or an expression involving already defined variables*

Most programming languages, Python included, can use variables

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Programs and programming (cont.)

Example

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

```python
v0 = 5
g = 9.81
t = 0.6

y = v0*t - 0.5*g*t**2

print y
```

This second program is much easier to read

- Closer to the mathematical notation

⇝ *Store the program text in a file* `ball2.py`

⇝ Running the program outputs `1.2342`

Variables and reserved words

A formula

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Variables and reserved words

Variable names can contain

- Any lower or upper case letter of the alphabet (`A`, `a`, `B`, `b`, ...)
- Numbers from 0 to 9 (but first character cannot be a number)
- Underscore (`_`)

Remark

Python distinguishes between upper and lower case letters

- Variable `X` is different from variable `x`
- Variable `Xx` is different from `xX`
- ...

Variables and reserved words (cont.)

Example

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

```
initial_velocity = 5
acceleration_of_gravity = 9.81
TIME = 0.6

VerticalPositionOfBall = initial_velocity*TIME - \
                         0.5*acceleration_of_gravity*TIME**2

print VerticalPositionOfBall
```

With long variables names, the code for evaluating the formula got long

- We broke it into two lines (the backslash \ at the end of the line)
- Make sure there are no blanks after the backslash

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Variables and reserved words (cont.)

Long names explain well what they represent

Though checking correctness of the formula for `y` became harder

- (Than in the program using `v0`, `g`, `t`, and `y0`)

Variables and reserved words (cont.)

A standard convention is to have variable names with lower case letters

- (Then, words are separated by an underscore)

Example

Whenever the variable represents a mathematical symbol, we use it

- v_0 in mathematics becomes `v0` in the program
- y in mathematics becomes `y` in the program

Resemblance between math symbols and variables names is important

⇝ Easy reading of the code

⇝ Errors detection

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Variables and reserved words (cont.)

Certain words are reserved in Python

- Utilised to build the language

These **reserved words** CAN NOT be used as variable names

- `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `False`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `None`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `True`, `try`, `with`, `while`, `yield`

Remark

To use a reserved word as variable name, add an underscore at the end

- For some quantity λ, use `lambda_`

Variables and reserved words (cont.)

Program files can have a freely chosen name

It is good practice to AVOID names coinciding with **keywords** or **module**

Keywords and module names in Python

- `math.py`, `time.py`, `random.py`, `os.py`, `sys.py`
- `while.py`, `for.py`, `if.py`, `class.py`, `def.py`
- ...

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats

A formula

Comments, text and number formats

Along with code statements, it is always informative to provide **comments**

- To explain the idea behind statements
- Using a natural language

Definition

Comments

In Python, comments start with the # character

Everything after # on a line is ignored when the program is executed

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

Example

```
# Compute 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
```

Comments, text and number formats (cont.)

Remark

By default, **non-English characters** in comments are desabled

- If you use them, Python will complain

```
SyntaxError: Non-ASCII character '\xc3' in file ...
but no encoding declared; see
http://www.python.org/peps/pep-0263.html for details
```

Non-English characters are enabled by using a code line in the beginning

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

- This is a comment that is not ignored by Python

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

As output of our program we simply print a numerical value of y

```
# Compute 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
```

It is often a good idea to write/print a more informative text

For example, consider printing the example text

⇝ `At t=0.6 s, the height of the ball is 1.23 m`

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

Definition

Printf syntax (from function `printf` in the C programming language)

Output from a `print` statement, plus number formatting

The oldest and most widely used technique is printf formatting/syntax

- *The printf syntax is used in a lot of other programming languages*
- *It is easy to learn and very convenient and flexible to work with*
- *The syntax of printf formatting may look awkward*

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

The `print` statement prints a **string**

```
# Compute 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 'At t=%g s, the height of the ball is %.2f m.' % (t, y)  # print y
```

Everything enclosed in quotes (single, `'`, or double `"`) denotes a string

```
print 'At t=%g s, the height of the ball is %.2f m.' % (t, y)
```

The string above (based on our program) is formatted using printf syntax

- The string has '**slots**', starting with a percentage sign
- Variables in the program can be inserted in the slots

The slots and the variables in the example

⇝ `%g` and `%.2f`

⇝ `t` and `g`

Comments, text and number formats (cont.)

```
print 'At t=%g s, the height of the ball is %.2f m.' % (t, y)
```

We have two 'slots', thus two variables must be inserted into the slots

The relevant syntax is to list the variables inside parentheses after the string

- The variables' list is separated from it by a percentage symbol

⇝ `% (t, y)`

The first variable, `t`, goes into the first 'slot' with format specification `%g`

- The percentage sign **marks the slot**
- The following character, `g`, is the chosen **format specification**
- The `g` format instructs the real number to be compactly written

The next variable, `y`, goes into the second 'slot' with format `.2f`

- The `.2f` format instructs the real number is with two decimal digits

⇝ (The `f` in the `.2f` format stands for *floating-point number*)

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

Example

```
v0 = 5
g = 9.81
t = 0.6

y = v0*t - 0.5*g*t**2

print 'At t=%g s, the height of the ball is %.2f m.' % (t, y)
```

```
Terminal> python ball_print_f.py

At t=0.6 s, the height of the ball is 1.23 m
```

Comments, text and number formats (cont.)

There are many available ways to specify formats

`e` writes a number in **scientific notation**

- A number between 1 and 10 followed by a power of 10
- ($1.2432 \cdot 10^{-3}$, as `1.2432e-03`)
- Capital `E` in the exponent is possible: Replace `e` by `E` (`1.2432E-03`)

For **decimal notation** we use letter `f` (as in `%f`)

- The output number appears with digits before and/or after a comma
- (`0.0012432` instead of `1.2432E-03`)

With the `g` format, the output is in scientific notation for large or small numbers and it is in decimal notation otherwise (**compact output**)

- A lower case `g` leads to lower case `e` in scientific notation
- An upper case `G` implies `E` instead of `e` in the exponent

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

It is possible to specify the format in some very sophisticated manner

Example

⇝ `10.4f`

⇝ `14.6E`

The first case: A float written in decimal notation

- 4 decimals in a field of width equal to 10 characters

The second case: A float written in scientific notation

- 6 decimals in a field of width equal to 14 characters

Comments, text and number formats (cont.)

Format	Explaination
`%s`	A string
`%d`	An integer
`%0xd`	An integer in a `x`-width field, padded with leading zeros
`%f`	Decimal notation with six decimals
`%e`	Compact scientific notation, `e` in the exponent
`%E`	Compact scientific notation, `E` in the exponent
`%g`	Compact decimal or scientific notation, with `e`
`%G`	Compact decimal or scientific notation, with `E`
`%xz`	Format `z` right-adjusted in a `x`-width field
`%-xz`	Format `z` left-adjusted in a `x`-width field
`%.yz`	Format `z` with `y` decimals
`%x.yz`	Format `z` with `y` decimals in a `x`-width field
`%%`	The percentage sign

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

Example

```
i = 62
r = 189876545.7654675432

# Print out numbers with quotes "" to see width of field

print '"%d"' % i        # minimum field
print '"%5d"' % i       # field of width 5 characters
print '"%05d"' % i      # pad with zeros

print '"%g"' % r        # r is big number, scientific notation
print '"%G"' % r        # E in the exponent
print '"%e"' % r        # compact scientific notation
print '"%E"' % r        # compact scientific notation
print '"%20.2E"' % r    # 2 decimals, field of width 20
print '"%30g"' % r      # field of width 30 (right-adjusted)
print '"%-30g"' % r     # left-adjust number
print '"%-30.4g"' % r   # 3 decimals

print '%s' % i          # convert i to string automatically
print '%s' % r

# Use %% to print the percentage sign
print '%g %% of %.2f Euro is %.2f Euro' % \
      (5.1, 346, 5.1/100*346)
```

Comments, text and number formats (cont.)

Example

```
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)
```

A **triple-quoted string**, started and ended by three single/double quotes

Triple-quoted strings are used for text that spans several lines

- `t` is printed in the `f` format (by default six decimals)
- `v0` is written in the `.3E` format (three decimals and the number spans as narrow field as possible)
- `y` is two decimals in narrow decimal notation, `.2f`

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

```
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.
```

- `t` is printed in the `f` format (by default six decimals)
- `v0` is written in the `.3E` format (three decimals and the number spans as narrow field as possible)
- `y` is two decimals in narrow decimal notation, `.2f`

Comments, text and number formats (cont.)

Format string syntax

It offers all the functionalities available with the printf format

- (And much more, through a different syntax)

Example

We illustrate this syntax on the one-line output that was used earlier

```
print 'At t={t:g} s, the height of the ball is {y:.2f} m.' \
      .format(t=t, y=y)
```

- Slots are denoted by curly braces (rather than a percentage sign)
- Variable are listed with an optional colon and format specifier
- Variables and their values are listed at the end of the statement
- Slots have names (the sequence of variables is not important)

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Comments, text and number formats (cont.)

At times, we want to write out text that spans several lines

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

We can obtain such an output by using triple-quoted strings

MAP_FC 2019

Comments, text and number formats (cont.)

The **newline character**

We can also use ordinary single-quoted strings and a special character

- The special character indicates where line breaks should occur
- The special character is `\n` (a backslash followed by letter `n`)

Example

```
print """y(t) is
the position of
our ball."""

print 'y(t) is\nthe position of\nour ball'
```

The two `print` statements have identical output

```
y(t) is
the position of
our ball.
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Another formula

Computing with formulas

Another formula

Example

Consider the usual expression for converting a temperature measurement

- From degrees Celsius (C) to its value in degrees Fahrenheit (F)

$$F = \frac{9}{5}C + 32 \tag{4}$$

Given the formula above and a value of C, our goal is to compute F

A first attempt at implementing the formula

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

The parentheses are not strictly needed

Another formula (cont.)

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

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

When run under Python version 2.x, the program prints the value 53

Testing correctness is easy, we evaluate the formula on a calculator

$$\rightsquigarrow \quad \frac{9}{5} \cdot 21 + 32 = 69.8 \neq 53$$

What is wrong? The formula typed in the program looks correct!

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

Integer division

Another formula

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

MAR 20 2019 _ FC _

Integer division

The error is one of the most common errors in mathematical coding

- For a newcomer to programming, this is not obvious at all

In many computer languages, there are two types of divisions

- **Integer division**
- **Float division**

Integer division

Definition

Float division *is what you expect from standard arithmetics*

- $9/5$ *becomes* 1.8 *in decimal notation*

Integer division a/b *with integers* a *and* b *is an integer* c

- *It is the largest integer* c *such that* $bc \leq a$

$\rightsquigarrow$ $9/5$ is 1, as $1 \cdot 5 = 5 \leq 9$ and $2 \cdot 5 = 10 > 9$

$\rightsquigarrow$ $1/5$ is 0, as $0 \cdot 5 \leq 1$ and $1 \cdot 5 > 1$

Computing with formulas

FC
CK0030
2019.1

Integer division (cont.)

Remark

Many computer languages (..., Fortran, C, C++, Java, and Python 2.x) interpret a/b as integer division, if both operands a and b are integers

Suppose that either a or b are real (floating-point) numbers

$\rightsquigarrow$ Then, a/b implies the standard mathematical division

$\rightsquigarrow$ (Float division)

Other languages (..., MATLAB and Python 3.x) interpret a/b as float division even if both operands are integers

- (or complex division if any of the operands is a complex number)

Integer division (cont.)

The issue with our program is in the coding of the formula `(9/5)*C + 32`

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

First, `9/5` is calculated (Python interprets `9` and `5` as integers)

- `9/5` is thus interpreted as a division between two integers
- Python chooses by default integer division, returning `1`

Then, `1` is (normally) multiplied by `C`, giving `21`

- `21` and `32` are added
- `53` is returned

⇝ (Wrong result)

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Objects in Python

Another formula

Objects in Python

Consider a very general assignment statement like `C = 21`

- Python interprets number 21 as an integer

```
C = 21
```

It creates an **int** (for integer) **object** holding the value 21

- The variable `C` acts as **variable name**
- This name labels the **int object** as `C`

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Objects in Python (cont.)

Similarly, in `C = 21.0`, Python recognises number `21.0` as a real number

```
C = 21.0
```

It creates a **float** (for floating-point) **object** holding the value 21.0

- The variable `C` is the **variable name** of this **float object**

Remark

The key issue is that 21 and 21.0 are identical numbers in mathematics

In Python,

- 21 gives an `int object`
- 21.0 gives a `float object`

Objects in Python (cont.)

Remark

Any (Python) assignment statement has the general form

- Variable name, on the left-hand side
- An object, on the right-hand side

```
C = 21
```

```
v0 = 5
g = 9.81
t = 0.6

y = v0*t - 0.5*g*t**2
```


Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Objects in Python (cont.)

At this point, it is not requested to know now what an object exactly is

As initial simplification, one can think of an `int object` as a collection

- It is like a storage box, with some information about an integer
- The information is stored within the computer's memory
- The variable name `C` is used to access this information

There are various object types, some are pre-built some are user-defined

- Objects may contain a lot of data, not just integer/real numbers

Objects in Python (cont.)

Example

```
print 'A text with an integer %d and a float %f' % (2, 2.0)
```

A **str** (for string) **object**, without a name, is first created from 'the text between quotes' and then the `str object` is printed using `print` command

We can alternatively do this in two, sequential, steps

```
s = 'A text with an integer %d and a float %f' % (2, 2.0)
print s
```

Computing with formulas

FC
CK0030
2019.1

A formula
- Programs and programming
- Variables
- Comments, text and numbers formatting

Another formula
- Integer division
- Objects in Python
- Integer division
- Arithmetic operators

Mathematical functions
- Examples
- Rounding errors

Interactive computing
- The shell
- Type conversion
- IPython

Avoiding integer division

Another formula

Avoiding integer division

We must be careful to avoid integer division when coding math formulas

Remark

Python 3.x has no problem with *unintended* integer division

- Only with Python 2.x (and other languages)

There are several ways to avoid integer division with the plain (`/`)

The simplest remedy in Python version 2

```
from __future__ import division
```

This import statement must be present in the beginning of EVERY single file where the `/` operator ALWAYS shall imply float division

An alternative remedy in Python 2

```
Terminal> python -Qnew someprogram.py
```

One can run any Python program `someprogram.py` from the command line with the argument `-Qnew` for the Python interpreter

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Avoiding integer division (cont.)

A more widely used method, common also to other programming languages

⇝ Force one of the operands to be a `float object`

Example

```
F = (9.0/5)*C + 32
F = (9/5.0)*C + 32

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

In the first two lines, one of the operands is written as a decimal number

- This implies a `float object`, and therefore float division results

In the last line, `float(C)*9` means (`float` times `int`)

- This results in a `float object`, and thus float division is implicit

Example

```
F = C*float(9/5) + 32                  # !! It does not work correctly !!
```

Avoiding integer division (cont.)

We can ask our Python to locate *potential* integer divisions in a program

- Python programs can be executed with a `-Qwarnall` argument

It will show a warning every time an integer division expression is found

```
Terminal> python -Qwarnall someprogram.py
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Avoiding integer division (cont.)

Remark

We could have run into problems when we wrote the formula $\frac{1}{2}gt^2$

- We used `0.5*g*t**2`, and that worked well
- If `(1/2)*g*t**2`, term `(1/2)` would be zero

Arithmetic operators
Another formula

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Arithmetic operators

In Python, formulas are evaluated as they are mathematically

- Given an expression, from left to right, term by term
- The terms are separated by plus (`+`) or minus (`-`)

Within terms, power operations (a^b, `a**b`) have priority

- Computed before multiplication/division

Parentheses dictate how a formula is evaluated

Arithmetic operators (cont.)

Example

```
5/9 + 2*a**4/2
```

First, $5/9$ (`5/9`) is evaluated (as integer division, with `0` as result)

a^4 (`a**4`) is evaluated, 2 and a^4 are multiplied (`2*a**4`)

- The result is divided by 2 (`2*a**4/2`)
- The answer is therefore `a**4`

This result is added to the result of `5/9` (`5/9 + a**4`)

Computing with formulas
FC
CK0030
2019.1
A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Arithmetic operators (cont.)

Example

```
5/(9+2)*a**(4/2)
```

First, expression $\dfrac{5}{9+2}$ ($\leadsto$ `5/(9+2)`) is evaluated (integer division, result `0`)

$4/2$ (`4/2`) is computed (integer division, result `2`)

`a**2` (`a**(4/2)`) is calculated

This result is multiplied by the result of `5/(9+2)` (`5/(9+2)*a**(4/2)`)

The answer is thus always `0`

Arithmetic operators (cont.)

It is easy to unintentionally get integer division in formulas

Of course, it is possible to turn integer division off in (any) Python

- Important to be aware of the existence of the concept
- Important to develop programming habits to avoid it

Remark

The concept of integer division appears in many programming languages

- It is better to learn as early as possible how to deal with it
- Python-specific (or else) features does not remove the issue

Computing with formulas

FC
CK0030
2019.1

Mathematical functions

Computing with formulas

Evaluating mathematical functions

Standard mathematical formulas frequently involve common functions

- sin, cos, tan, sinh, cosh, exp, log, ...

On a pocket calculator you have special buttons for such functions

- Similarly, in a language you have ready-made *functions*

Remark

In principle, one could write his/her own program for evaluating

- Say, the $\sin(x)$ function

How to do it efficiently is often a non-trivial task

Experts have worked on such problem for decades

- They implemented their best recipes
- These codes should be (are) re-used

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Evaluating mathematical functions (cont.)

We discuss how to reach sin, cos, and similar functions within Python

Examples

Mathematical functions

Examples, `sqrt` and `sinh`

Example

The height y of a ball in vertical motion, with initial upward velocity v_0

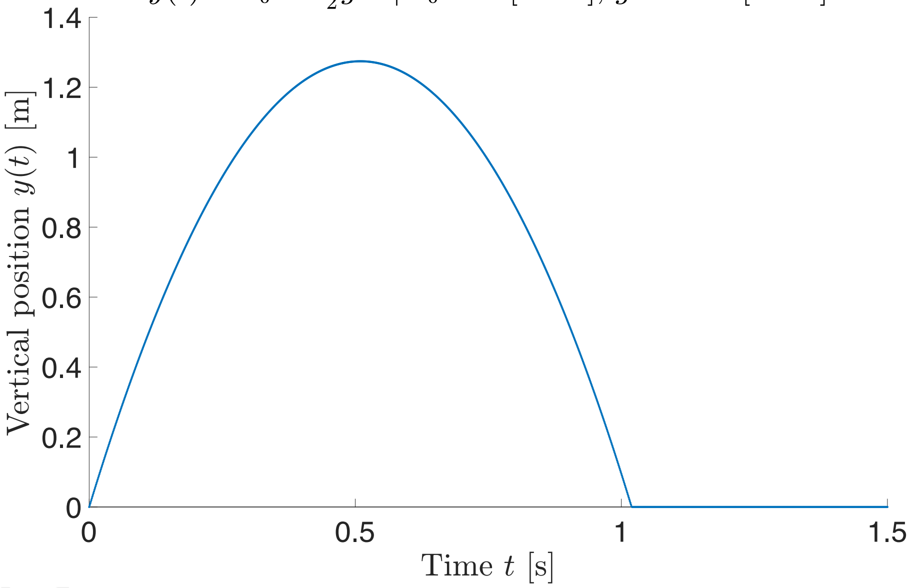
$$y = v_0 t - \frac{1}{2} g t^2$$

In the formula, we are using g for the gravity acceleration and t for time

Examples, `sqrt` and `sinh` (cont.)

How long time does it take for the ball to reach the height y_c?

$y(t) = v_0 t - \frac{1}{2} g t^2 \mid v_0 = 5$ [ms^{-1}], $g = 9.81$ [ms^{-2}]



There are two solutions (t_1 and t_2)

- Once when the ball reaches y_c on its way up (t_1)
- Once when it reaches on its way down (t_2)

Computing with formulas

FC
CK0030
2019.1

A formula
- Programs and programming
- Variables
- Comments, text and numbers formatting

Another formula
- Integer division
- Objects in Python
- Integer division
- Arithmetic operators

Mathematical functions
- Examples
- Rounding errors

Interactive computing
- The shell
- Type conversion
- IPython

Examples, `sqrt` and `sinh` (cont.)

- When $y = y_c$, we have $y_c = v_0 t - \frac{1}{2} g t^2$ and the equation

$$\frac{1}{2} g t^2 - v_0 t + y_c = 0 \tag{5}$$

- A quadratic form[1] that must be solved with respect to t

$$t_1 = \frac{v_0 - \sqrt{v_0^2 - 2 g y_c}}{g}$$
$$t_2 = \frac{v_0 + \sqrt{v_0^2 - 2 g y_c}}{g}$$

For the expressions of t_1 and t_2, we need the square root [$\sqrt{(\cdot)}$]

[1] $ax^2 + bx + c = 0$, $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Examples, sqrt and sinh (cont.)

Remark

Square root and other math functions are available in a **module** called `math`

- sin, cos, sinh, exp, log, ...

To make module functions available, we must first **import the module**

- We must write `import math` in our program

To take the square root of variable a, $\sqrt{a}$, we write `math.sqrt(a)`

Computing with formulas
FC
CK0030
2019.1
A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Examples, sqrt and sinh (cont.)

Example

$$t_{(1|2)} = \frac{v_0 \mp \sqrt{v_0^2 - 2gy_c}}{g}$$

```
v0 = 5
g = 9.81
yc = 0.2

import math
t1 = (v0 - math.sqrt(v0**2 - 2*g*yc))/g
t2 = (v0 + math.sqrt(v0**2 - 2*g*yc))/g

print 'At time t=%g s and %g s, the height is %g m.' % (t1, t2, yc)
```

The output from this program

```
At time t=0.0417064 s and 0.977662 s, the height is 0.2 m.
```

Examples, `sqrt` and `sinh` (cont.)

Definition

The standard way to import a module, `module_name`

⇝ *`import module_name`*

Functions `function_name` are accessed by using `module_name` as prefix

⇝ *`module_name.function_name`*

Example

```
import math
x = math.sqrt(y)
```

Clearly, the use of `math.sqrt(y)` is less pleasing than a plain `sqrt(y)`

Computing with formulas

FC
CK0030
2019.1

A formula
- Programs and programming
- Variables
- Comments, text and numbers formatting

Another formula
- Integer division
- Objects in Python
- Integer division
- Arithmetic operators

Mathematical functions
- **Examples**
- Rounding errors

Interactive computing
- The shell
- Type conversion
- IPython

Examples, `sqrt` and `sinh` (cont.)

Definition

An alternative import syntax allows to skip the module name prefix

- *`from module_name import function_name`*

A specific example of `from module_name import function_name`

⇝ `from math import sqrt exp log sin`

The alternative import syntax allows direct access to `sqrt` (or else)

- Without the `math.` prefix

Examples, sqrt and sinh (cont.)

Example

$$t_{(1|2)} = \frac{v_0 \mp \sqrt{v_0^2 - 2 g y_c}}{g}$$

```
v0 = 5
g = 9.81
yc = 0.2

from math import sqrt                                    # WAS: import math
t1 = (v0 - sqrt(v0**2 - 2*g*yc))/g
t2 = (v0 + sqrt(v0**2 - 2*g*yc))/g
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Examples, sqrt and sinh (cont.)

Definition

All functions `function_name` in module `module_name` can be imported at once

$\rightsquigarrow$ `from module_name import *`

- Importing all (∗) functions from a module is often convenient
- Not recommended to import more functions than needed
- The convenience of a compact import syntax often wins

In the `math module`

- `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`
- `exp`, `log` (base e), `log10` (base 10), `sqrt`
- Numbers (`e`, `pi`, ...)
- ...

Examples, `sqrt` and `sinh` (cont.)

Definition

Modules and functions can be given new names in the import statement

Example

```
import math as m              # m is now the name of the math module

v = m.sin(m.pi)

from math import log as ln
v = ln(5)

from math import sin as s, cos as c, log as ln
v = s(5)*c(5) + ln(5)
```


Examples, `sqrt` and `sinh` (cont.)

Remark

Since in Python everything is an object

Modules, functions, numbers and strings are objects

Variables refer to objects and new variables may refer to them

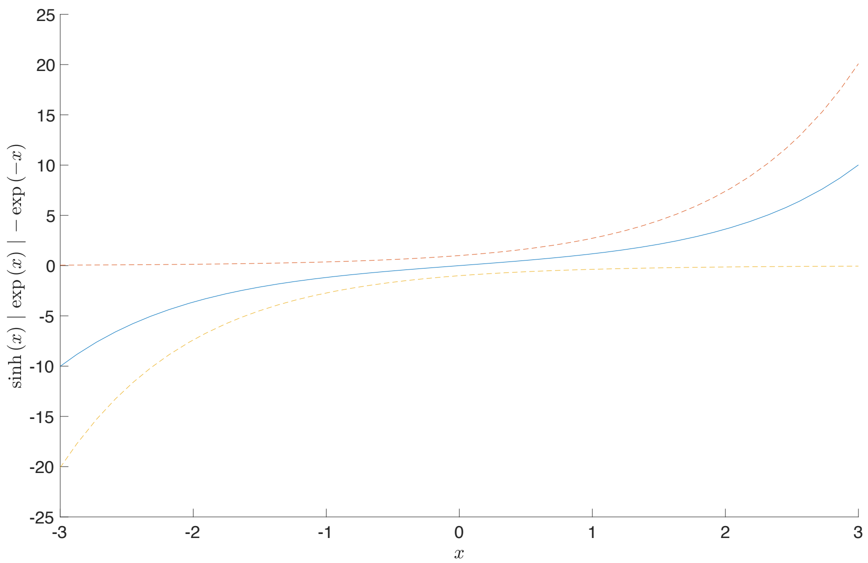
```
m = math
ln = m.log
s = m.sin
c = m.cos
```

Examples, `sqrt` and `sinh` (cont.)

Example

Consider the definition of the hyperbolic function $\sinh(x)$

$$\sinh(x) = \frac{1}{2}\Big(e^{x} - e^{-x}\Big) \tag{6}$$



Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Examples, `sqrt` and `sinh` (cont.)

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

We can evaluate $\sinh(x)$ in three different ways

- By calling `math.sinh`, directly
- By computing the RHS using `math.exp`
- By computing the RHS using `e` and power expressions `math.e**x` and `math.e**(-x)`

Examples, `sqrt` and `sinh` (cont.)

Example

$$\sinh(x) = \frac{1}{2}\Big(e^{x} - e^{-x}\Big), \text{ for } x = 2\pi$$

```python
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 r1, r2, r3
```

All three computations are mathematically equivalent

- Output from `print` displays identical results

```
267.744894041 267.744894041 267.744894041
```

..., SQN!

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Rounding errors

Mathematical functions

Rounding errors

Example

```
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)        # WAS: print r1, r2, r3
```

A print out of `r1`, `r2`, `r3` that displays 16 decimals

```
267.7448940410164369
267.7448940410164369
267.7448940410163232
```

This command shows how `r1`, `r2`, `r3` are different

- But, why is it so?

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Rounding errors (cont.)

MAR_FC 2019

Remark

A computer program calculates its arithmetics using *wannabe* real numbers[2]

True real numbers (Dedekind) may require an infinite number of decimals

- $\rightsquigarrow$ Because of finite storage, the sequence of decimals is truncated
- $\rightsquigarrow$ On computers, it is standard to keep 17 digits in a real number

[2]Let $x \in \mathcal{R}$ and let $\text{fl}(x)$ its (rounded) representation in a computer. We have that $x \neq \text{fl}(x)$ with $\dfrac{|x - \text{fl}(x)|}{|x|} \leq \dfrac{1}{2}\varepsilon_M$ in which the quantity ε_M is called *machine precision*.

Rounding errors (cont.)

Remark

Real numbers on a computer often have a small error

Only a few numbers can be represented exactly

- The rest are approximations

Most arithmetic operations on a computer involve inaccurate real numbers

- This results in inaccurate calculations

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Rounding errors (cont.)

Example

Think of $\frac{1}{49}49 = 1$ and $\frac{1}{51}51 = 1$ when performed in Python

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

```
0.9999999999999999 1.0000000000000000
```

- $1/49$ is not correctly represented in the computer
- $1/51$ also has an inexact representation

$\rightsquigarrow$ (but error does not show too much :/)

Errors in floating-point numbers may propagate through computations

The results are approximations to the exact mathematical values

- Such errors are commonly called **rounding errors**

Rounding errors (cont.)

Remark

Python has ad hoc modules `decimal` and `SymPy` package has module `mpmath`

They allow for real numbers to be represented with adjustable accuracy

- Rounding errors can be made as small as desired

Interactive computing

Computing with formulas

Interactive computing

Python can execute statements and evaluate expressions interactively

The environments where one works interactively are Python **shells**

- The simplest Python 2.x shell is invoked by `python` or `python2`

⇝ (In a terminal)

```
Terminal> python
Python 2.7.9 (default, Jun 29 2016, 13:08:31)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>>
```

Some Python messages are displayed together with a **prompt** > > >

- After that, you can start issuing commands

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Interactive computing (cont.)

Example

The interactive shell as calculator

1. Type `3*4.5-0.5`
2. Press `Return`

```
Terminal> python
Python 2.7.9 (default, Jun 29 2016, 13:08:31)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> 3*4.5-0.5
    13.0
```

The text after the > > > prompt is the **shell input**

The text without the > > > prompt is the result that Python calculates

- The **shell output**

Interactive computing (cont.)

Remark

The shell makes it easy to recover previous input and edit the text

⇝ This helps experimenting with statements and expressions

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

The shell

Interactive computing

The shell

Example

Consider the program for the vertical position of the ball

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

It can be fully re-typed line-by-line in the Python shell

```
>>> v0 = 5
>>> g = 9.81
>>> t = 0.6
>>> y = v0*t - 0.5*g*t**2
>>> print y
    1.2342
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

The shell (cont.)

We can easily calculate the `y` value corresponding to another `v0` value

- Hit the arrow-up key (⇑), to recover previous statements
- Repeat pressing ⇑, until the `v0 = 5` statement shows up
- You can then edit the relative line

```
>>> v0 = 6                                   # It was: v0 = 5
```

- Press `Reurn`, to execute this statement
- To check the new value of `v0` either type `v0` or `print v0`

```
>>> v0
    6

>>> print v0
    6

>>> print y                        # Old, needs be re-computed
    1.2342
```

The shell (cont.)

The next step is to re-compute `y`, with the new `v0` value

- Hit the arrow-up key (⇑) multiple times to recover the statement where `y` is assigned
- Press `Return`
- Write `y` or `print y` to see the result

```
>>> y = v0*t - 0.5*g*t**2
>>> y
    1.8341999999999996

>>> print y
    1.8342
```

We get two slightly different results

- `y` prints out all the decimals stored in the computer (16)
- `print y` prints out `y` with fewer decimals, standard format

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

The shell (cont.)

Remark

Computations on a computer often suffer from rounding errors

- The present calculation is no exception

The correct answer is 1.8342

Rounding errors led to a number that is incorrect

- The error is in the 16th decimal
- The error is $4 \cdot 10^{-16}$

Type conversion

Interactive computing

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Type conversion

Work w/o bothering about the **type of objects** variables refer to

- Yet, we encountered a serious problem with integer division
- Important to be careful about the involved types of objects

The interactive shell is useful for exploring types (the `type` function)

Type conversion (cont.)

Example

Let us create some `int object C`

Let us check its type with `type(C)`

⇝ Function `type`

```
>>> C = 21
>>> type(C)
    <type 'int'>

>>> C
    21
```

We convert the `int object C` to a corresponding `float object`

⇝ Function `float`

```
>>> C = float(C)                              # type conversion
>>> type(C)
    <type 'float'>

>>> C
    21.0
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Type conversion (cont.)

Statement `C = float(C)` creates a new object, `C`

⇝ From the original one, `C`

The new object is also referred to by the name `C`

⇝ It binds it to the same name `C`

After the statement, variable `C` refers to a different object

- Original `int object`, value `21`, becomes unreachable

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

Type conversion (cont.)

Example

We can also convert a `float object` to a corresponding `int object`

⇝ Function `int`

```
>>> C = 20.9
>>> type(C)
    <type 'float'>

>>> D = int(C)                              # type conversion
>>> type(D)
    <type 'int'>

>>> D
    20                                  # decimals are truncated
```

Converting a `float` to an `int` implied stripping off the decimals

Example

Conversion according to rounding rules

⇝ Function `round`

```
>>> round(20.9)
    21.0

>>> int(round(20.9))
    21
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

IPython

Interactive computing

IPython

There are several improvements of the standard Python shell

- **IPython** is the common interactive shell
- You need to have `ipython` installed

Typing `ipython` in a terminal window starts the shell

```
Terminal >  ipython
Python 2.7.9 (default , Jun 29 2016, 13:08:31)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra
              details.

In [1]:
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

iPython (cont.)

Example

Running programs

```
Terminal > ipython
Python 2.7.9 (default , Jun 29 2016, 13:08:31)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra
              details.

In [1]: run ball2.py
        1.2342
```

The command requires that you have `cd`'ed to the folder with `ball2.py`

IPython (cont.)

On Windows you may, as alternative to starting IPython from a DOS or PowerShell window, double click on the IPython icon or use Start menu

- You must move to the folder where your program is located
- If `ball2.py` is in the folder `div` under `My Stuff` of user `me`
- (This is done by the `os.chdir`, change directory, command)

```
In [1]: import os
In [2]: os.chdir(r'C:\Documents and Settings\me\My Stuff\div')
In [3]: run ball2.py
```

- Note the `r` before the quote in the string
- Required to let a backslash (\\) really mean the backslash character

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

IPython (cont.)

Remark

You may frequently have to type the `os.chdir` command in `ipython`

This and other commands can be suitably placed in a **startup file**

- ⇝ A file that is automatically executed when you launch `ipython`
- ⇝ To create one from Terminal, `ipython create profile`

IPython (cont.)

Inside `ipython` you can invoke any operating system command

This allows to navigate the filesystem with Unix/Windows commands (`cd`)

- (Instead of Python's, `os.chdir`)

```
In [1]: cd C:\Documents and Settings\me\My Stuff\div
In [3]: run ball2.py
```

Definition

OS commands

```
In [3]: date
        Thu Nov 18 11:06:16 CET 2010

In [4]: ls
        myfile.py yourprog.py

In [5]: mkdir mytestdir

In [6]: cd mytestdir
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

IPython (cont.)

Remark

Suppose that some Python variables have the same name as an OS command

- (`date=30`)

The OS command must be called with an exclamation mark (`!`) in front

- (`!date`)

IPython (cont.)

It is recommended to run all of your Python programs from inside `ipython`

- `ipython` can help examine the state of variables and locate bugs
- (When something goes wrong)

Remark

⇝ To execute a program in `ipython`, type `run` before program name

⇝ To run a program in a Terminal, `python` prior to program name

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

IPython (cont.)

Definition

Output from statements or expressions in `ipython` are preceded by `Out [X]`

- *`X` is the command number of the last `In [X]` prompt*

When programs are executed, as with the `run` command or when OS commands are run, the output is from the OS itself

- In this case, the output is not preceded by any `Out[X]` label

Definition

Output recovery

Outputs (`Out [X]`) from previous statements in `ipython` are available/usable

They are in variables of the form `_iX` (underscore `_`, `i`, and a number `X`)

`X` is `1` for the last statement, `2` for the second last statement, and so forth

- *Short forms are `_` (for `_i1`), `__` (for `_i2`), and `___` (for `_i3`)*

IPython (cont.)

Example

Remember that the output from input `In [1]` was `1.2342`

We can now refer to it by an underscore

We can also perform operations on it

- Say, we multiply it by `10`

```
In [2]: _*10
Out[2]: 12.341999999999999
```

Computing with formulas

FC
CK0030
2019.1

A formula
Programs and programming
Variables
Comments, text and numbers formatting
Another formula
Integer division
Objects in Python
Integer division
Arithmetic operators
Mathematical functions
Examples
Rounding errors
Interactive computing
The shell
Type conversion
IPython

IPython (cont.)

Definition

Command recovery

The command history can navigated by typing specific keystrokes

- *`Ctrl+p` or ⇑ to go backward, `Ctrl+n` or ⇓ to go forward*

Any command you previously hit can be edited and re-executed

- *Also commands from previous sessions (in the history)*

Definition

Command history

*The **command history** from previous `ipython` sessions is available*

- *This feature makes it easy to modify previous work*
- *Hit arrow-up to recall commands and edit them*

Definition

Tab completion

Pressing the `TAB` key completes incompletely typed variable names

- *It can save some typing*

IPython (cont.)

Remark

Notebooks

It allows to record/replay interactive sessions as a mix

- Text, mathematics, Python code, and graphics
- Alternative to interactive shells