

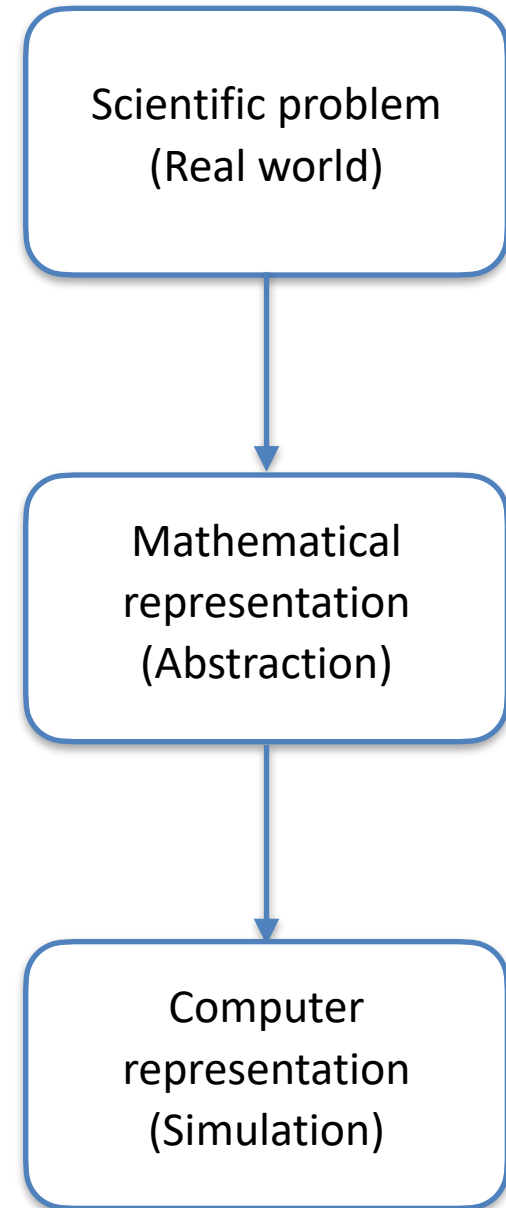
# Introduction to Computer Modelling

Computer Models

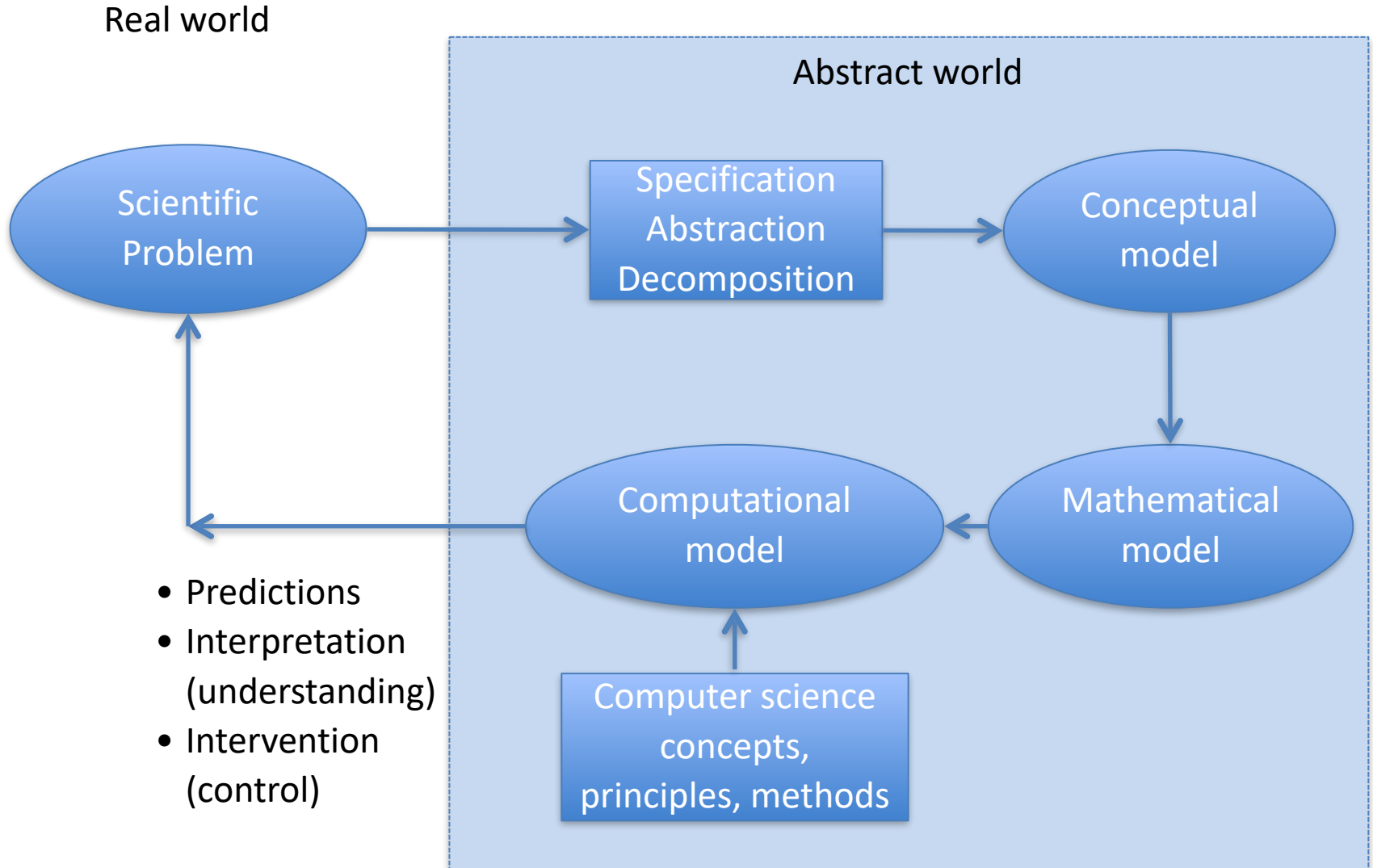
## Why modeling?

- Save money and time
- Get a deeper understanding of a problem
- Make predictions for outcomes of experiments
- Simulate experiments that are impossible to conduct in the real world
- ...

## Different levels of abstraction



## Levels of abstraction



# Computer problem solving workflow

- i) Problem definition (Purpose? What question shall be answered?)
- ii) Model specification (objects/components, interactions/relations, attributes/parameters, variables, constraints)
- iii) Mathematical Model (equations, algorithms)
- iv) Implementation (coding, debugging)
- v) **Verification** (Is the model behaving according to the mathematical and conceptual model?)
- vi) Validation (Does the model describe reality?)

## Verification

- Are constraints implemented and checked correctly?
- Does the output data reflect the properties of the model? (e.g. linear, quadratic etc.)
- Does the program reproduce special cases where analytical solutions are known correctly (within numerical precision)?
- If the model includes properties like conservation laws, symmetries, upper/lower bounds etc., are these fulfilled (within numerical precision)?
- ...

# Computer problem solving workflow

- i) Problem definition (Purpose? What question shall be answered?)
- ii) Model specification (objects/components, interactions/relations, attributes/parameters, variables, constraints)
- iii) Mathematical Model (equations, algorithms)
- iv) Implementation (coding, debugging)
- v) Verification (Is the model behaving according to the mathematical and conceptual model?)
- vi) **Validation** (Does the model describe reality?)

## Validation

- Compare with measurements or observations.
- Compare with code that is known to produce correct results.
- Are all necessary components/objects included?
- Is the model capable of describing real world with enough details and accuracy?  
Does it include the correct relations/interactions?  
Should additional or other properties (attributes/parameters) be included?

If not, start again at ii)

# Computer problem solving workflow

- i) Problem definition (Purpose? What question shall be answered?)
- ii) Model specification (objects/components, interactions/relations, attributes/parameters, variables, constraints)
- iii) Mathematical Model (equations, algorithms)
- iv) Implementation (coding, debugging)
- v) **Verification** (Is the model behaving according to the mathematical and conceptual model?)
- vi) Validation (Does the model describe reality?)

## Real world example

Real world example: you go on a vacation in the US and want to go on a hike the next day. Therefore you watch the weather forecast, but you do not know what it means to you because temperatures are in Fahrenheit. Should you bring a jacket?

- i) Europeans don't know Fahrenheit. Device a method to transform °F to °C.
- ii) Input F → output C, linear behavior, scalings, shifts, constraint: temperature  $\geq \text{abs}(0)$
- iii)  $C = 5(F-32)/9$
- iv) `def f2c(F): return 5.0*(F-32.0)/9.0 # no constraint checking!`
- v) Run and check if output is as expected following ii) and iii)
- vi) Check with real world (or other simulations)

# Task 1

Install the latest Python version on your laptop using anaconda

`https://docs.anaconda.com/anaconda/install/`

(Choose Python 3.9)

Work out the example: “Problem 1” (download the file and open in jupyter notebook)

In command prompt or terminal type: `jupyter notebook`

(Windows users may be also go through the start menu)

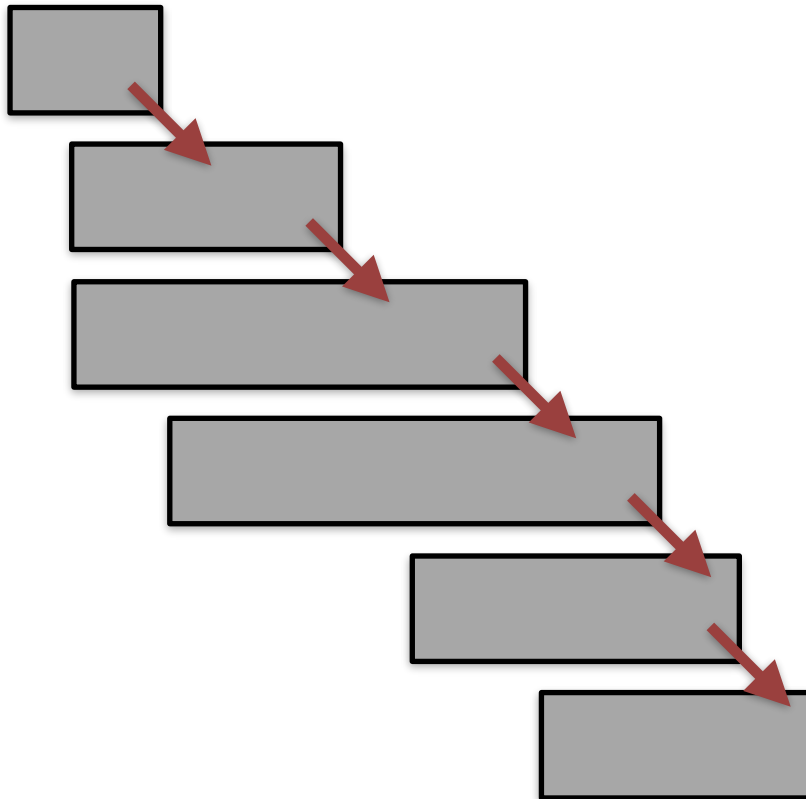
- use the circumference and area of a circle as an example
- prepare your own notebook to carry out the calculation
- provide detail about the above-mentioned steps i) to vi) as comments or markups in the notebook
- upload the notebook to Moodle
- before uploading check if the notebook runs without error:

`Kernel → Restart & run all`

# Problem decomposition

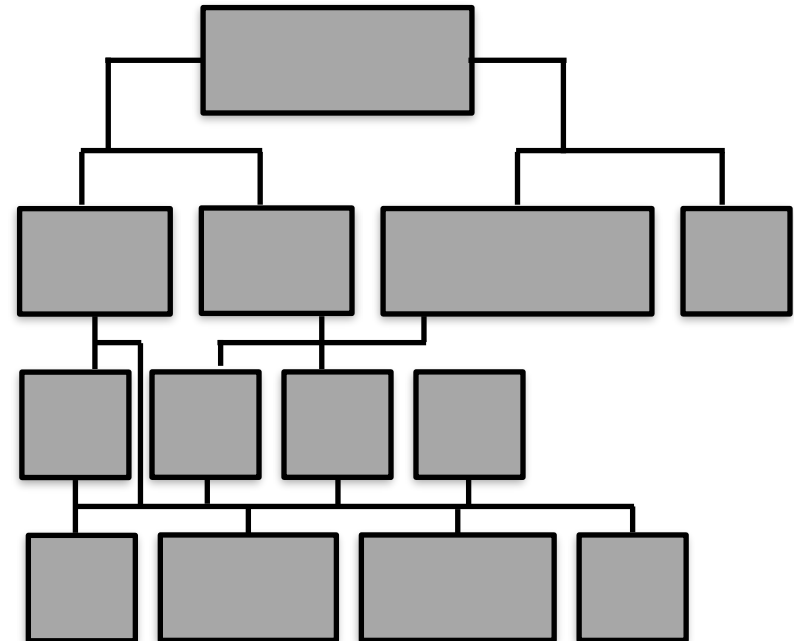
## top-down

(“waterfall”)



## modular

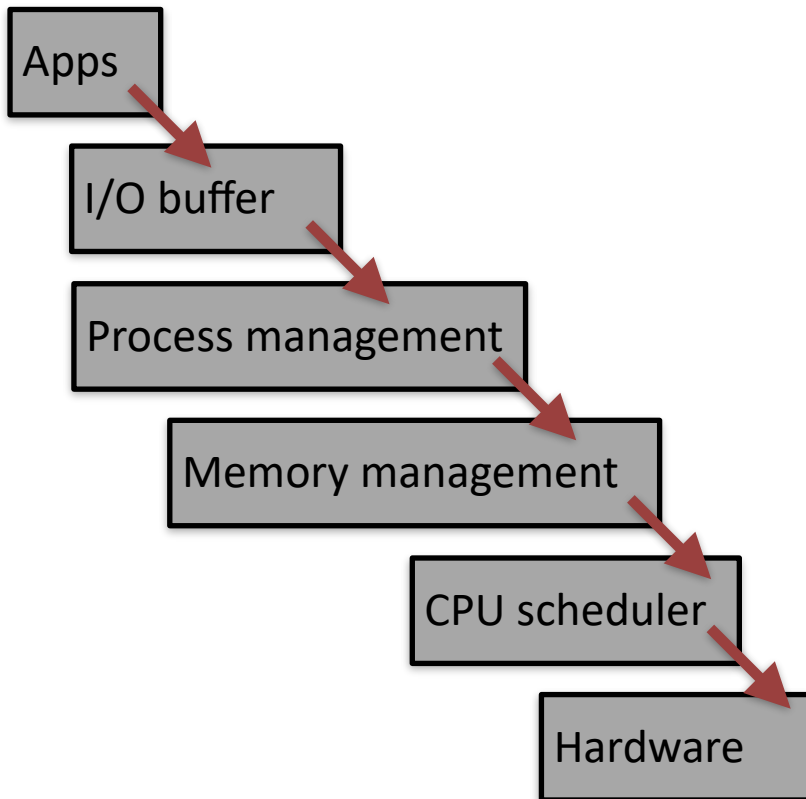
(„divide and conquer”, „Lego bricks“ )



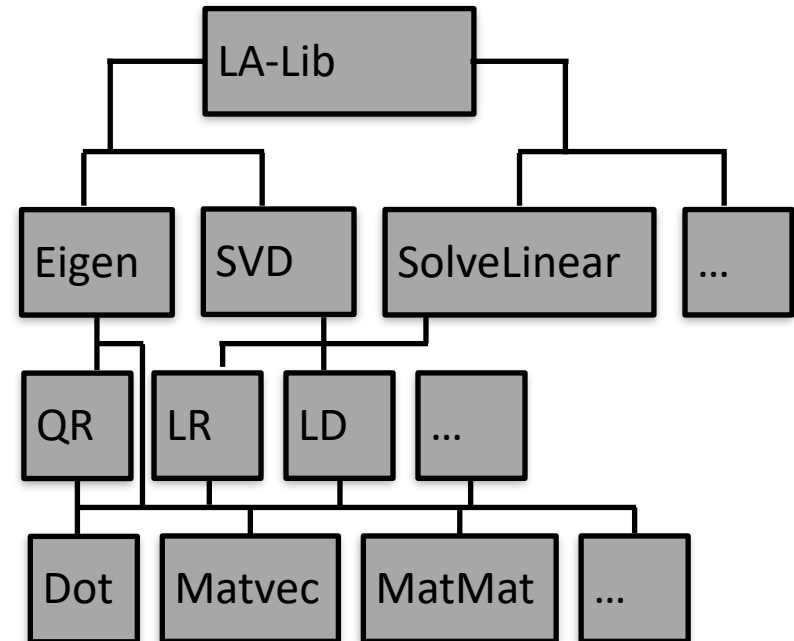


# Problem decomposition

**top-down: operating system**  
(“waterfall”)



**modular: linear algebra library**  
(„divide and conquer”, „Lego bricks“ )



# Programming languages

## High-level programming languages:

- Fortran, C, C++, Java, Python
- hardware-independent, readable
- source program or source code



- compiler compiles the program into machine code (Fortran, C/C++)
- interpreter reads and executes program (Python)
- compiler compiles the program into bytecode, which is then interpreted (Java)



- often, compiled program also needs to be linked to libraries
- compiled program is platform-dependent

## Low-level programming languages:

- machine language, assembly language
- hardware-dependent, low readability



- program directly runs on processor

# Python programming language

- Founded in 1989 by Guido van Rossum; being a big fan of Monty Python's flying circus, he picked the name "Python"
- Goals of Python: easy and intuitive language, open source, understandable code, suitable for daily tasks
- Comprehensive standard library
- Large collection of (scientific and numerical) third party modules
- (many people think Python is slow, but this is because they don't use scientific modules)











# TIOBE Index for April 2022

## April Headline: MATLAB about to drop out of the top 20

Good old MATLAB is about to drop out of the top 20 for the first time in more than 10 years. The MATLAB programming language is mainly used in the numerical analysis domain. It is often combined with Simulink models, which are from the same MathWorks company. Although MATLAB has a biannual release cycle, the language doesn't evolve that much. And since MATLAB licenses are rather expensive, alternatives are catching up quickly now. Its main competitors are Python (currently number 1) and Julia (moving from position 32 to position 26 this month). --Paul Jansen CEO TIOBE Software

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

| Apr 2022 | Apr 2021 | Change | Programming Language                                                                |                   | Ratings | Change |
|----------|----------|--------|-------------------------------------------------------------------------------------|-------------------|---------|--------|
| 1        | 3        | ▲      |    | Python            | 13.92%  | +2.88% |
| 2        | 1        | ▼      |    | C                 | 12.71%  | -1.61% |
| 3        | 2        | ▼      |    | Java              | 10.82%  | -0.41% |
| 4        | 4        |        |   | C++               | 8.28%   | +1.14% |
| 5        | 5        |        |  | C#                | 6.82%   | +1.91% |
| 6        | 6        |        |  | Visual Basic      | 5.40%   | +0.85% |
| 7        | 7        |        |  | JavaScript        | 2.41%   | -0.03% |
| 8        | 8        |        |  | Assembly language | 2.35%   | +0.03% |

# Different types of problems and models

continuous  
("analytical")

Classical motion of an object -  
e.g. a car driving down the road;  
a swinging pendulum

The temperature (field) of an  
object - e.g. a pizza in the oven;  
the heat generated through a  
chemical reaction

$$\int f(x)dx = \frac{x^3}{3}$$

discrete  
("numerical")

Motion of a particle through "jumps" -  
e.g. Brownian motion of a particle on an  
ordered surface

discrete states, e.g. spin  
states ( $\alpha, \beta$ )

$f(x) = x^2$   
Integral?

$$\int_b^a f(x)dx = \lim_{n \rightarrow \infty} \frac{b-a}{n} \sum_{k=1}^n f\left(a + k \frac{b-a}{n}\right)$$

# Number representation

floating-point arithmetic: approximation of real number through finite digits

$\pi = 3.141592653589793238462643383279502884197169399375105820974...$

in double precision (occupying 64 bits): correct up to 16 digits

$\pi = 3.141592653589793115997963468544185161590576171875...$

in single precision (occupying 32 bits): correct up to 8 digits

$\pi = 3.1415927410125732421875...$

overflow: occurs when value is too large to be represented with available digits; for example, for unsigned 64 bit integers

(In modern Python integers have infinite range)

$$2^{64} - 1 = 18,446,744,073,709,551,615$$

for floats in Python

```
print(sys.float_info.max)
```

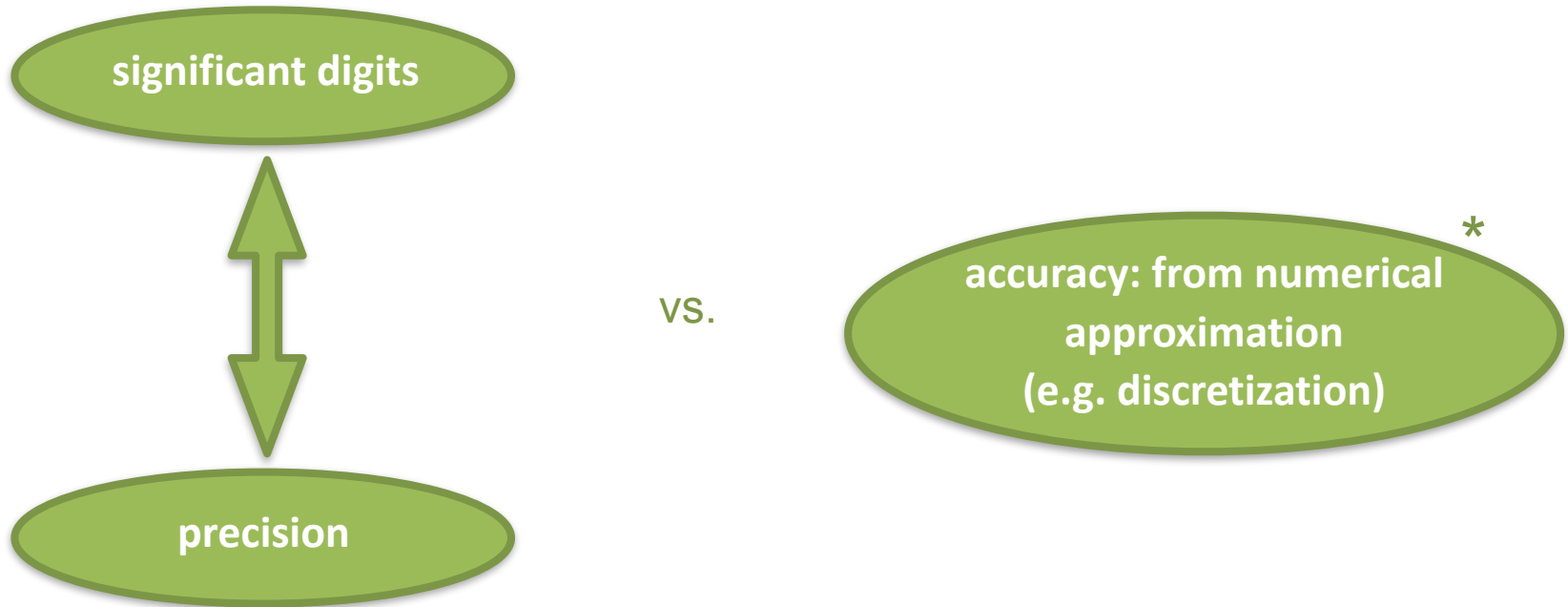
```
1.7976931348623157e+308
```

underflow: value is too small; for floats in python

```
: print(sys.float_info.min)
```

```
2.2250738585072014e-308
```

## Two sources of error



\* in the following, we will focus on **numerical approaches**  
**but:** precision is important

### types of variables:

numbers (integers, floats, complex)  
strings

lists (tuples)  
dictionaries

## Task 2

- work out the example: “Problem 2”
- upload the notebook to Moodle