Computational Methods I

lecturer: Falk Hassler (falk.hassler@uwr.edu.pl)
office: 448
lectures: Fri. 08:15 - 10:00
labs: Fri. 10:15 - 12:00 } 120

exercises, notebooks & handwritten notes at
https://www.fhassler.de/teaching/ws_24/comp_meth_1

- online ~ 1 week before tutorial
- everyone has to submit them

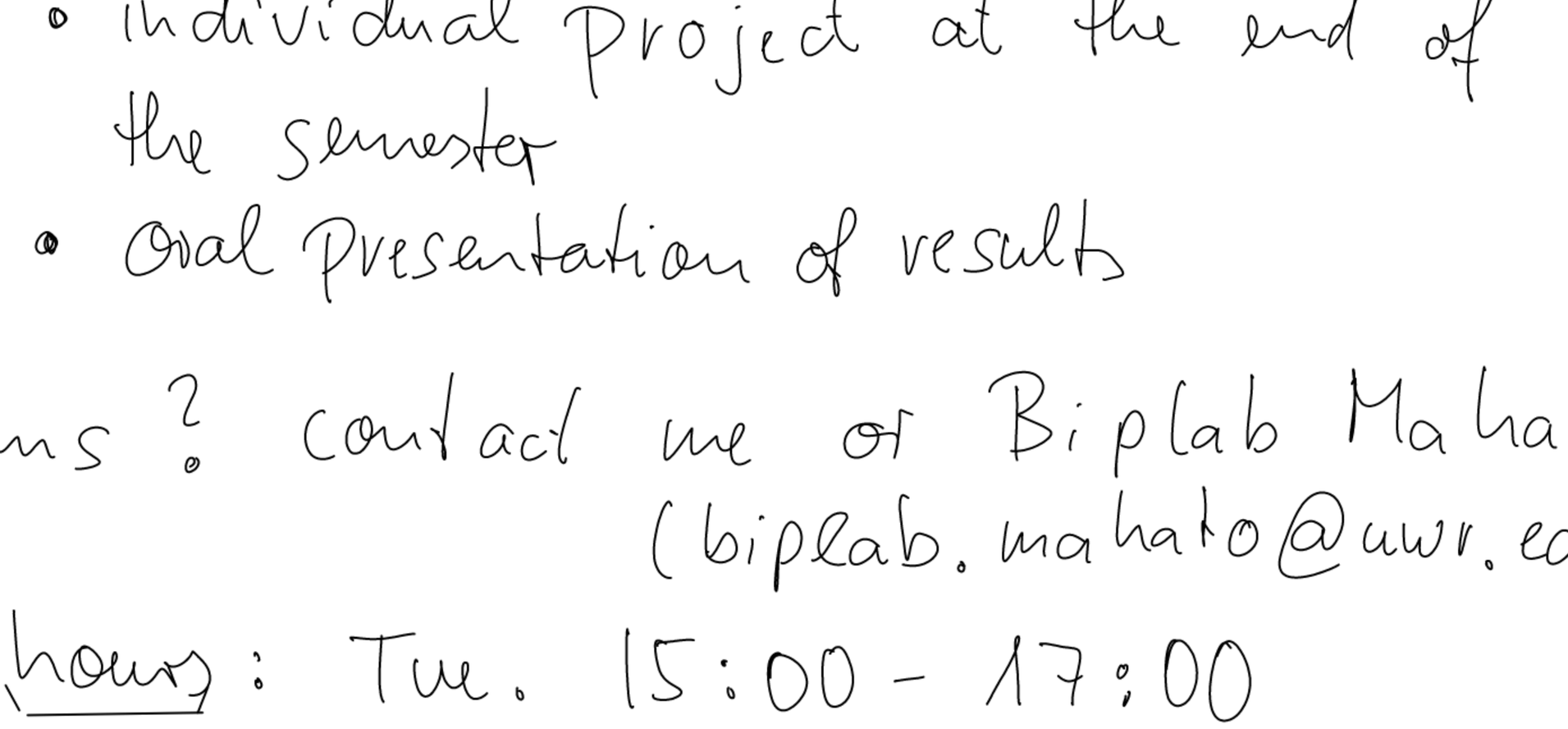
WILL BE GRADED > 50%
to qualify for the exam

exam
- individual project at the end of the semester
- oral presentation of results

Problems? contact me or Biplab Mahato (biplab.mahato@uwr.edu.pl)

office hours: Tue. 15:00 - 17:00

Motivation



mathematical model for observables ← System → measure observables in experiment

- ordinary partial diff. equation (ODE) } classical
- Partial diff. eq. (PDE) } quantum
- eigenvalue problem
-

Solve analytically (on paper)
1.) symbolic calculation, i.e. Mathematica
2.) numeric analysis
3.) Simulation

Solve numerically

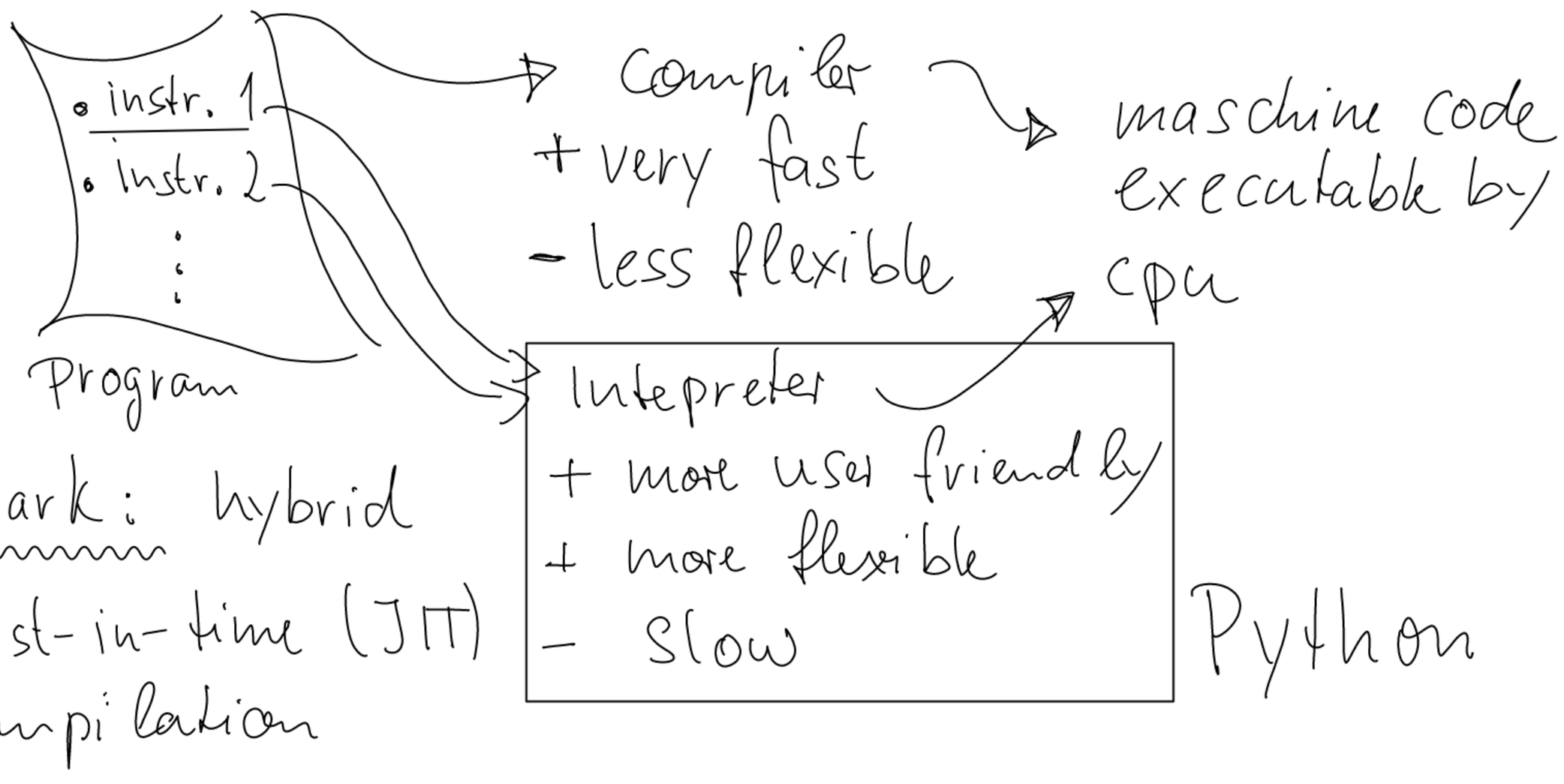
+ 4.) visualize the results

with COMPUTER

In this course 2.), 3.) and 4.) with Python

1. Introduction to Python

1.1. Interpreter vs. Compiler



Remark: hybrid = just-in-time (JIT) compilation

1.2. Comments

Ignored by the computer. Why?

↳ Document your code! For yourself & for others.

single line comment #, i.e. X=1 # set x to one

multi line comment """ long comment """

Remark: In Linux @ first line of file

#! /usr/bin/env python specifies the interpreter

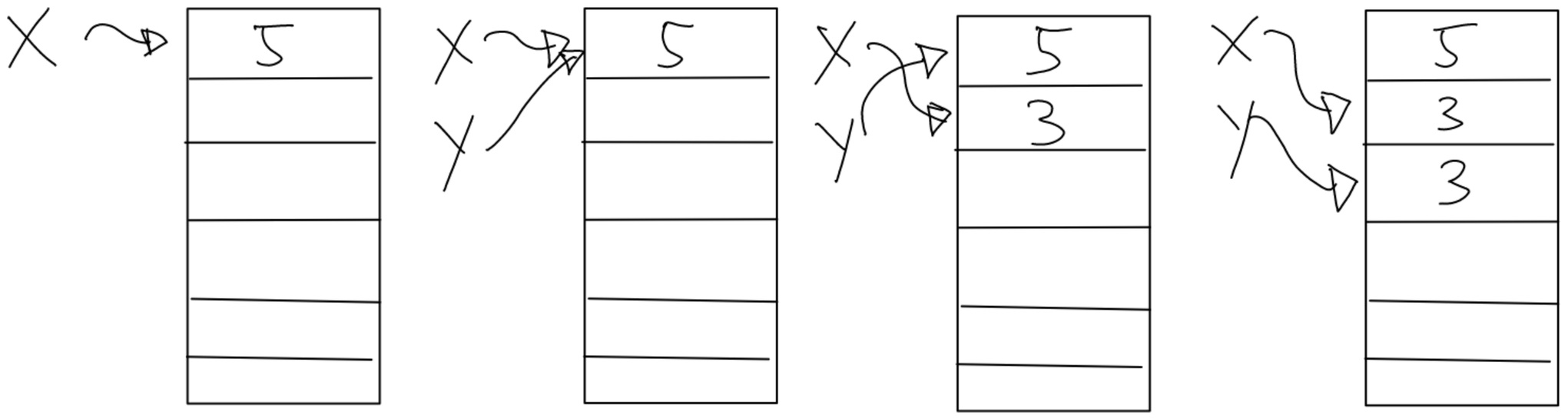
hash-bang

1.3. Variables

store data in the computers memory

i.e. X = 5 ← execution

allows for X = X + 1



X = 5 Y = X X = 3 Y = 3

also possible X = Y = Z = 1 or X, Y = Y, X

tuple

↳ there is a difference between copying pointers and the content of the underlying memory

Variable names: - Can contain letters, numbers, and _

- have to start with a letter
- are case sensitive

Variable types:

(1.) Numeric types

history lesson

Integer: 32-bit $\rightsquigarrow$ max = $2^{31}-1$

↯ int (1/2) $\Rightarrow$ 0

Long Integer: only limited by memory in computer

now only int for all up to sys.maxsize

Float: Rational numbers, more details later

1.0 / 2.0 = 1.0 / 2 = 1/2 = 0.5

upconverted to 2.0 →

Complex: $j = \sqrt{-1}$, $X = 0.5 + 1.2j$

(2.) Sequence Types

String: strings of characters delimited by

" ... " or ' ... '

escape sequences: '\n' = new line

'\t' = tab

'\#' = #

'\\' = \

Trick: multi line string: S = """ hello

World """

List: l = ["hello", "world"]

index

l[1] ⇒ "world"

Remarks: - first element has index 0

- negative index counts from the end
  i.e. l[-1] => last element
  l[-2] => element before the last element
- diff. types of elements possible
- <u>mutable</u>

Tuple: like List, but <u>immutable</u>

---

t = ("hello", "world")

<u>Dictionary</u>: is a key → value map
atomic type ↗ ↖ what ever you like

d = { "hello" : "world" }

d["hello"] => "world"

<u>Application</u>: Matrices

---

m = [[1,2,3], [4,5,6], [7,8,9]]

m[1][0] => 4

↳ 2·m or m1 + m2 will not give what you naively expect.

<u>1.4. Mathematical operations</u>

on <u>numerical types</u> +, -, *, /, ( ) work as expected, we also have

// integer division, 3//2 => 1
% remainder of div., 3%2 => 1
** exponent 2**8 => 256

priority: (), **, *, /, %, //, +, -
and from left to right

sequence types:

[1,2] * 2 => [1,2,1,2]

[1,2] + [3,4] => [1,2,3,4]

1.5. Control Structures

decide which instructions are executed next

Conditionals

can be either true or false examples

false are: the word **False**, 0 or 0.0,
empty sequences like "", (), [], {}

true are: the word **True**, number that aren't 0,
non-empty sequences

we also have comparisons

- < . less than
- > . greater —11—
- <= . less than or equal
- >= . greater —11—
- == . equal
- != . not equal

↯ don't confuse with assignment =

and boolean operators and, or, not

example: 1==2 or 1!=2 => True

for lists there is also 1 in [1,2] => True
while 1 in [3,4] => False

if ... elif ... else

```
if cond1:                                   executed when
    statement 1
    statement 2                             } Block 1   cond1 = True
elif cond2:                                 (not cond1) and cond2
    Block 2
    :                                       } optional
else:                                       otherwise
    Block n
```

↳ Blocks are identified by the same indentation; Python is white space sensitive

loops

```
while Condition:                executes Block as long as the condition is True
    Block
```

example:

```
x = 1
while x <= 9:
    x = x + 1

x => 10
```

for item in sequence: executes Block for each item in sequence
    Block

example: for i in range(10):
             Print(i)

1.6. Functions

function(arg1, arg2, ...) => return value
         optional

built-in functions, i.e. print("hello")
                     or range(1,2,10)

Trick: More about them with help function

define your own function:

```
def add(a, b=1):
    return a+b
```
↑ optional argument

with add(1,2) => 3
add(1) => 2
add(1, b=3) => 4