PROGRAMMING IN PYTHON I

Basics of Programming

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

Institute for
Machine Learning

Alice trying to play croquet with a Flamingo.

The wonderland of programming

Don’t get demotivated if some of your colleagues seem to play croquet while you are still struggling with your flamingo. You are not alone and it will get better. :)

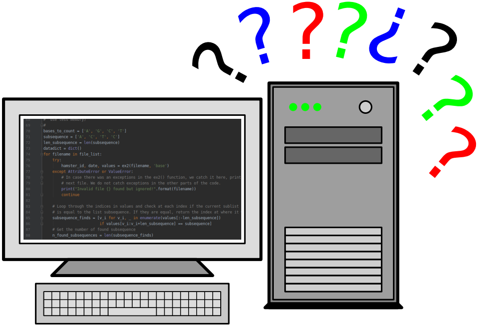
[Image: Illustration (1865) by J. Tenniel, of the novel by L. Carroll, Alice’s Adventures in Wonderland. Source: `https://en.wikipedia.org/wiki/Alice's_Adventures_in_Wonderland`]

BEFORE WE START…
GENERAL INFORMATION ON PROGRAMMING

Before we start...

- A computer is a machine. If you want to use it, it helps to know how it works.

→ These slides shall give you a rough (unprecise) understanding about programming and computers



General information on programming (1)

- Question: What is programming about?

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!
    1. Structure your thoughts on how to solve a problem

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!
    1. Structure your thoughts on how to solve a problem
    2. Formulate them as program code

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!
    1. Structure your thoughts on how to solve a problem
    2. Formulate them as program code
    3. Let the machine execute your code

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!
    1. Structure your thoughts on how to solve a problem
    2. Formulate them as program code
    3. Let the machine execute your code
    4. Look what went wrong and go back to previous steps

General information on programming (1)

- Question: What is programming about?
- Answer: Letting a machine do your work for you!
  1. Structure your thoughts on how to solve a problem
  2. Formulate them as program code
  3. Let the machine execute your code
  4. Look what went wrong and go back to previous steps

  $\rightarrow$ If the machine does it, you don't have to do it! :)

General information on programming (2)

- (In my opinion) 2 main sources of problems:

General information on programming (2)

- (In my opinion) 2 main sources of problems:

1. Syntax errors, wrong usage of code tools, insufficient knowledge about programming language
   - Syntax can be learned passively during lectures

General information on programming (2)

- (In my opinion) 2 main sources of problems:

1. Syntax errors, wrong usage of code tools, insufficient knowledge about programming language
   - Syntax can be learned passively during lectures
2. Semantic errors, mistakes in thinking through/formulating the solution
   - Generally wrong solutions (mistakes in thinking or understanding of task)
   - Missing consideration of some use-cases

General information on programming (2)

- (In my opinion) 2 main sources of problems:

1. Syntax errors, wrong usage of code tools, insufficient knowledge about programming language
   - Syntax can be learned passively during lectures
2. Semantic errors, mistakes in thinking through/formulating the solution
   - Generally wrong solutions (mistakes in thinking or understanding of task)
   - Missing consideration of some use-cases
   - The machine will do what you tell it to do

General information on programming (2)

- (In my opinion) 2 main sources of problems:

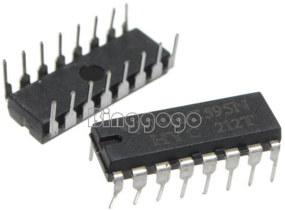
1. Syntax errors, wrong usage of code tools, insufficient knowledge about programming language
   - Syntax can be learned passively during lectures
2. Semantic errors, mistakes in thinking through/formulating the solution
   - Generally wrong solutions (mistakes in thinking or understanding of task)
   - Missing consideration of some use-cases
   - The machine will do what you tell it to do

- You will be directly confronted with your own errors
- Errors in code are also referred to as bugs

BITS AND BYTES

Bits and Bytes (1)

- Data on computer (usually) stored in bits
- Bit: element with 2 states (True/False, 0/1, . . .)
  - E.g. transistors, pneumatic elements, magnetic stripes, . . .
- Registers (small storages) usually able to hold 8 bits (or multitudes of 8 bits)
  - 8 bits are referred to as byte
  - Often described hexadecimal numbers (i.e. 2 bytes)



Bits and Bytes (2)

- We use bits to store all kinds of data
  - Values, text, programs, images, audio, ...
- We encode our data in bit patterns and later decode it to retrieve the meaning of the data
- Example:
  1. Right-Click on a (small) image, audio, PDF, or MS/Libre/Open Office file
  2. Open it with a text editor (`notepad`, `texteditor`, `gedit`)
  3. Enjoy the bit pattern of the file interpreted as pure text ;)

     Note: PDF or MS/Libre/Open Office files are not only text (contain formatting information etc.)

Datatypes

- We can use a group of bits to encode a value
- There are different ways to encode values as bits (=datatypes)
- The more bits per value we use, the more unique values we can encode (typically multitudes of bytes)
- Our main datatypes will be

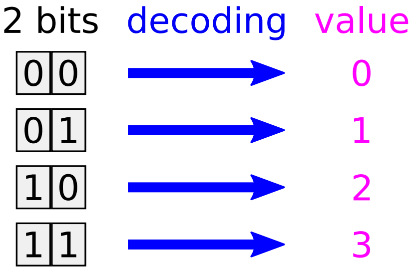
int	Integer – Integral numbers
float	Float – Floating point numbers
string	String – (String of) characters

Datatypes: Integer

- Integer datatype assigns one bit-pattern to one value

$\rightarrow$ Precise because no ambiguous bit-patterns

$\rightarrow$ Only integral numbers in certain range



Datatypes: Float

- Float datatype uses the formula

$$value = significand \times base^{exponent}$$

- $significand$ and $exponent$ are integers and $base$ is fixed

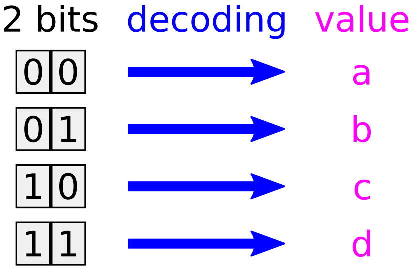
→ Not precise because values are approximated

→ Allows for floating point numbers in very large range

2 bits	decoding	value
0 0	→	0.0
0 1	→	0.0
1 0	→	1.0
1 1	→	16.0

Datatypes: String

- Character datatype assigns one bit-pattern (typically a byte) to one character/letter
- Such characters are concatenated, which gives datatype string (we will see more about this later)
- Different encoding formats: Unicode, UTF-8, ASCII, . . .

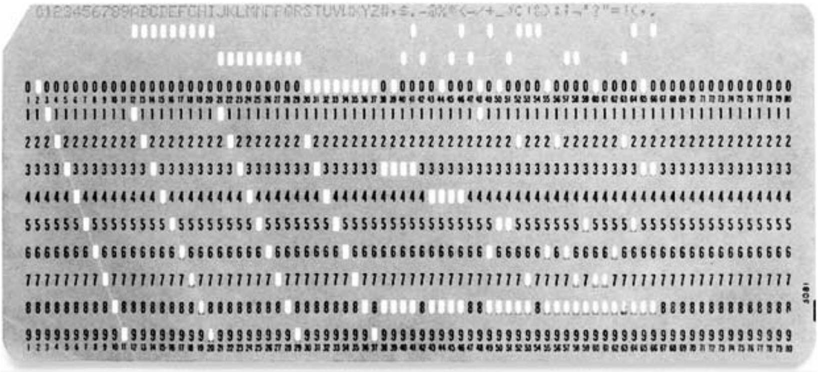


Machine Code

- Instructions to the machine (e.g. the controller) are also encoded in bits
  - Machine Code is often visualized as a sequence of hexadecimal numbers

bit pattern	decimal	hexadecimal	command
0000	0	0	MOVE
0001	1	1	ADD
0010	2	2	MULTIPLY
0011	3	3	...
0100	4	4	...
0101	5	5	...
0110	6	6	...
0111	7	7	...
1000	8	8	...
1001	9	9	...
1010	10	A	...
1011	11	B	...
1100	12	C	...
1101	13	D	...
1110	14	E	...
1111	15	F	...

Machine Code Example: Punch card



Program as IBM Punched Card (1928)

SIMPLIFIED EXECUTION OF A PROGRAM

Simplified execution of a program: Setup (1)

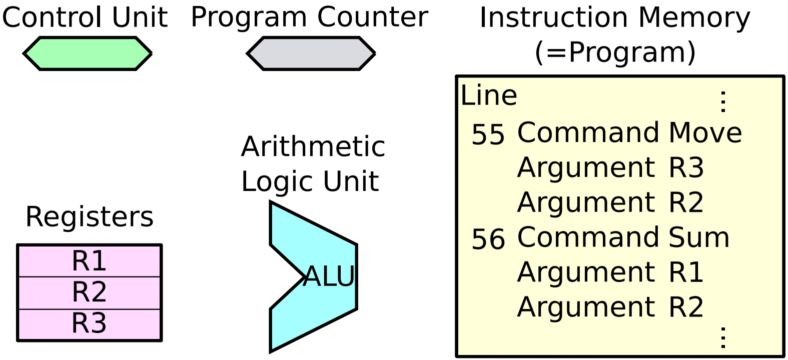
- This section will give you a rough idea about how a program is executed
- This section (Simplified execution of a program) is not relevant for the exam but will hopefully give you some idea about the basics

Simplified execution of a program: Setup (2)

- There are several hardware parts in a controller, we will see:

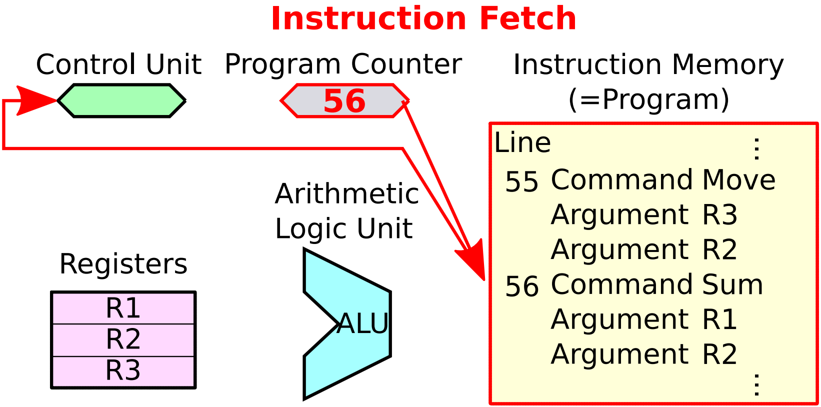
1. Instruction Memory (IM): The program in machine code (e.g. stamp-card or flash-storage)
2. Program Counter (PC): Holds a value that represents the line in the code (starts at 0)
3. Registers (R): (Temporary) storage to store bit patterns
4. Arithmetic Logic Unit (ALU): Circuit that performs arithmetic operations
   - These operations often use the same specific registers as input/output (=wired to the registers)
5. Control Unit (CU): Circuit that activates registers, ALU, and Program Counter based on current bit pattern in code

Simplified execution of a program (1)



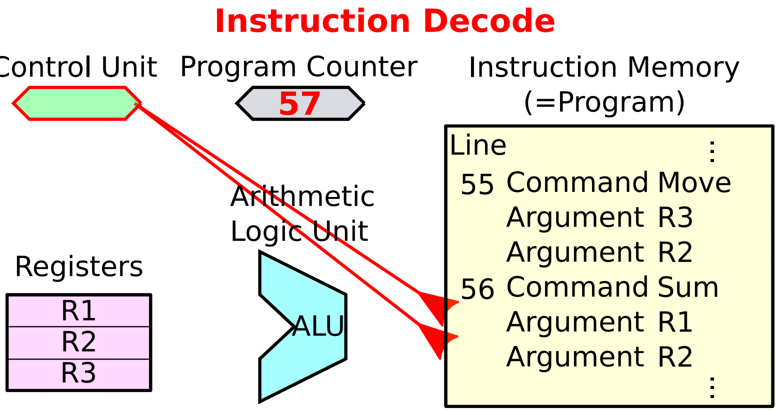
This is our processor and on the right side we see our program (IM)

Simplified execution of a program (2)



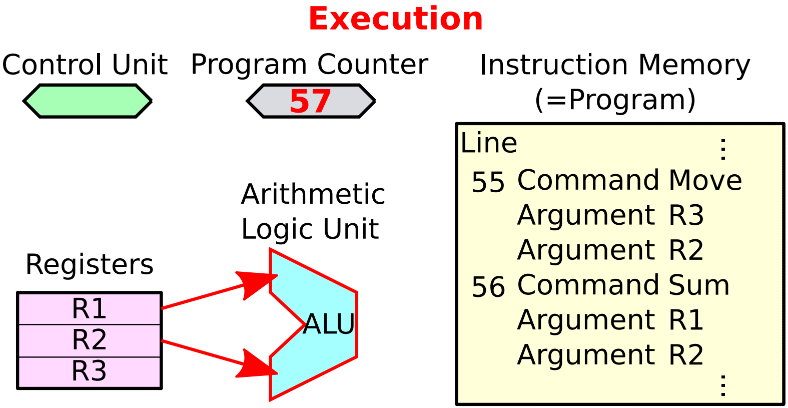
CU fetches bit pattern from IM at line number stored in PC (e.g. 56). CU increases PC by one (e.g. to 57).

Simplified execution of a program (3)



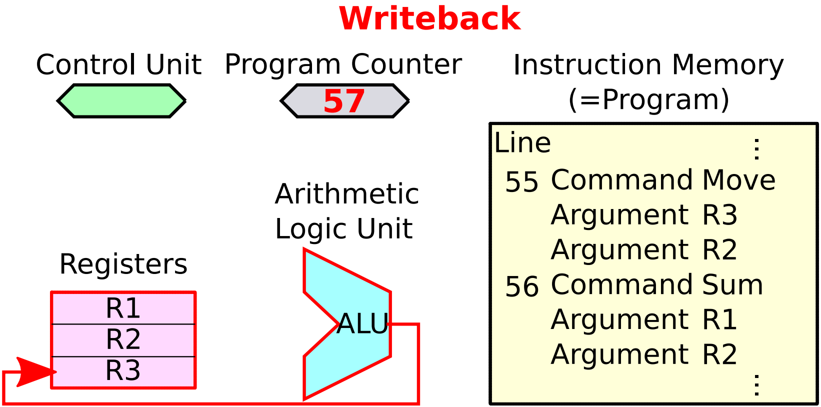
Bit pattern `Command Sum` triggers CU to fetch next 2 bit patterns in code (adress R1 and R2) and set ALU to summation.

Simplified execution of a program (4)



ALU performs arithmetic summation with values (bit patterns) from inputs R1 and R2.

Simplified execution of a program (5)



Result value (bit pattern) from ALU is stored in R3. In next step, CU will fetch the next bit pattern.

Simplified execution of a program (6)

- As you can see, summing up two values effectively can require more than one line of Machine Code
  1. Move values to registers
  2. Perform summation
  3. Move result from register to somewhere else
- This can get tedious and complex/difficult to read and is highly dependent on hardware

→ We would often like to get rid of (abstract from) these details/hardware

ABSTRACTION AND LANGUAGES

Abstraction and Languages (1)

Machine code	Instructions as bits
Assembly	Abstracts from Machine Code: More readable than bits; Still close to hardware; Potentially very fast (full control, special hardware functionalities accessible)
C, etc.	Abstracts from Assembly: Readable code; Abstracts registers and instructions; Fast because whole program is compiled and optimized at once (possibly for specific CPU architecture)

Abstraction and Languages (2)

C#, Java	Abstracts even further: Readable, convenient code but often no idea about actual instructions happening; Medium/fast and still compiled at once (to architecture independent intermediary format)
Python, R	Interpreted languages: Lines of code are executed one-by-one (i.e. *interpreted*) $\rightarrow$ in general no compilation of whole program but only individual lines; slow if not using specialized packages

HARDWARE

Hardware (1)

- CPU: Central processing unit (the actual main processor)
- RAM: Random-access memory (the working memory as volatile[1] storage)
- GPU: Graphics processing unit (computer graphics, image processing, deep learning, etc.)
- SSD: Solid-state drive (non-volatile[1] storage via integrated circuit)
- HDD: Hard disk drive (non-volatile[1] storage via rotating disks)

[1] Volatile memory needs constant power in order to retain data.

Hardware (2)

- To use a computer efficiently, you have to think about which parts to use for which task
- CPU (general computations) vs. GPU (dedicated to e.g. matrix operations)
- RAM (small, fast) vs. SSD/HDD (large, slow)

Hardware – approximate operation times[1]

System Event	Actual Latency	Scaled Latency
1 CPU cycle	0.4ns	1min
Level 1 cache	0.9ns	2.25min
DDR RAM	100ns	4.17h
SSD I/O	50–150µs	86–260d
Rotational disk	1–10ms	4.76–47.56yr

[1] **Intel Xeon processor E5 v4, 2.4GHz, latency times.**
Source: `https://www.prowesscorp.com/computer-latency-at-a-human-scale/`.

Hardware – approximate operation times[1]

System Event	Actual Latency	Scaled Latency
1 CPU cycle	0.4ns	1min
Level 1 cache	0.9ns	2.25min
DDR RAM	100ns	4.17h
SSD I/O	50–150µs	86–260d
Rotational disk	1–10ms	4.76–47.56yr
You sneezing	1s	951.29yr

[1] **Intel Xeon processor E5 v4, 2.4GHz, latency times.**
Source: `https://www.prowesscorp.com/computer-latency-at-a-human-scale/`.