15-110 Principles of Computing – S19

Lecture 1:
Introduction and Logistics

Teacher:
Gianni A. Di Caro

The 110 team

Teachers:

Prof. Giselle Reis
(logic, proof theory)

Prof. Gianni A. Di Caro
(robotics, AI, MAS)

Teaching Assistants:

- Abdullah Shaar
- Ward Ayan
- Keivin Isufaj
- Arambha Niraula

Road map

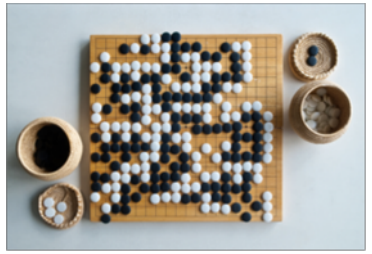
- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

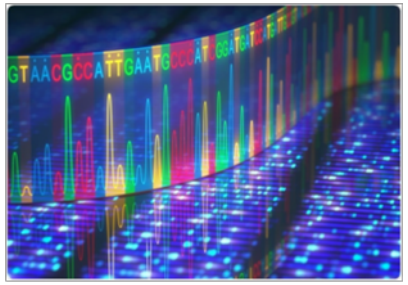
Road map

- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

Introduction to the *science (art)* of computational problem solving

Daily life, professional activities, leisure, business, … present a variety of ***problems*** that ask for finding a **solution**





- Different constraints (e.g., time, budget)
- Different representations / models
- Different requirements for the solution (e.g., provably the best, just one)
- Different available knowledge (e.g., chess, financial investment, path finding)
- Different degree of (un)certainty (e.g., poker, industrial manipulator)
- Different dimensionality (e.g., DNA sequencing, root finding, flight control)
- …

Often (usually) these problems are too complex to be **effectively** and **satisfactorily** solved by humans (alone)…

Introduction to the *science (art)* of computational problem solving

… ***computers*** can effectively help (us) solving the difficult problems!

- Perform **calculations**, at impressively high rates
  - $+ \; - \; * \; / \; > \; < \; = \; \neq$
  - Fastest computer in the world (IBM AC922): ≈ 10^{15} operations per sec on real numbers: 1,000,000,000,000,000 ops/sec
  - Mega = 10^6, Giga = 10^9, Tera = 10^{12}, Peta = 10^{15}, Exa = 10^{18}
- **Store** and **retrieve** data and results of calculations
  - 160TB internal memory, HP: 160,000,000,000,000 bytes
  - Single storage units > 100 TB: 100,000,000,000,000 bytes

… how do we connect *problem + humans + computers* for effective problem-solving?

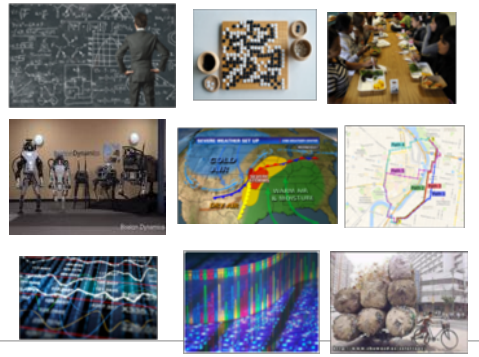
Introduction to the *science (art)* of computational problem solving

Computer programming

Computer Hardware

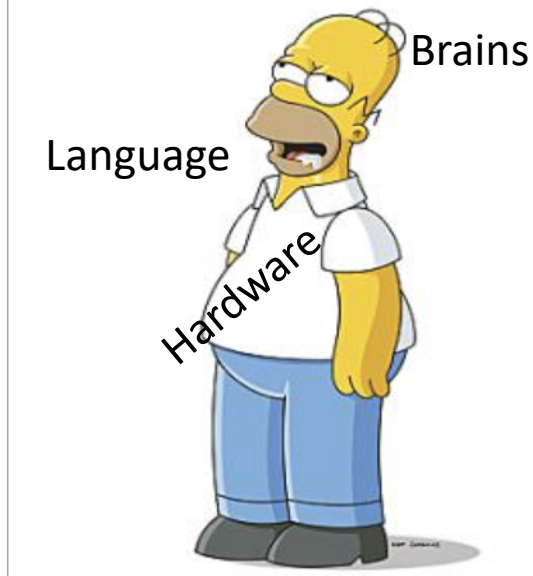
Abstraction
Model formulation

Algorithm

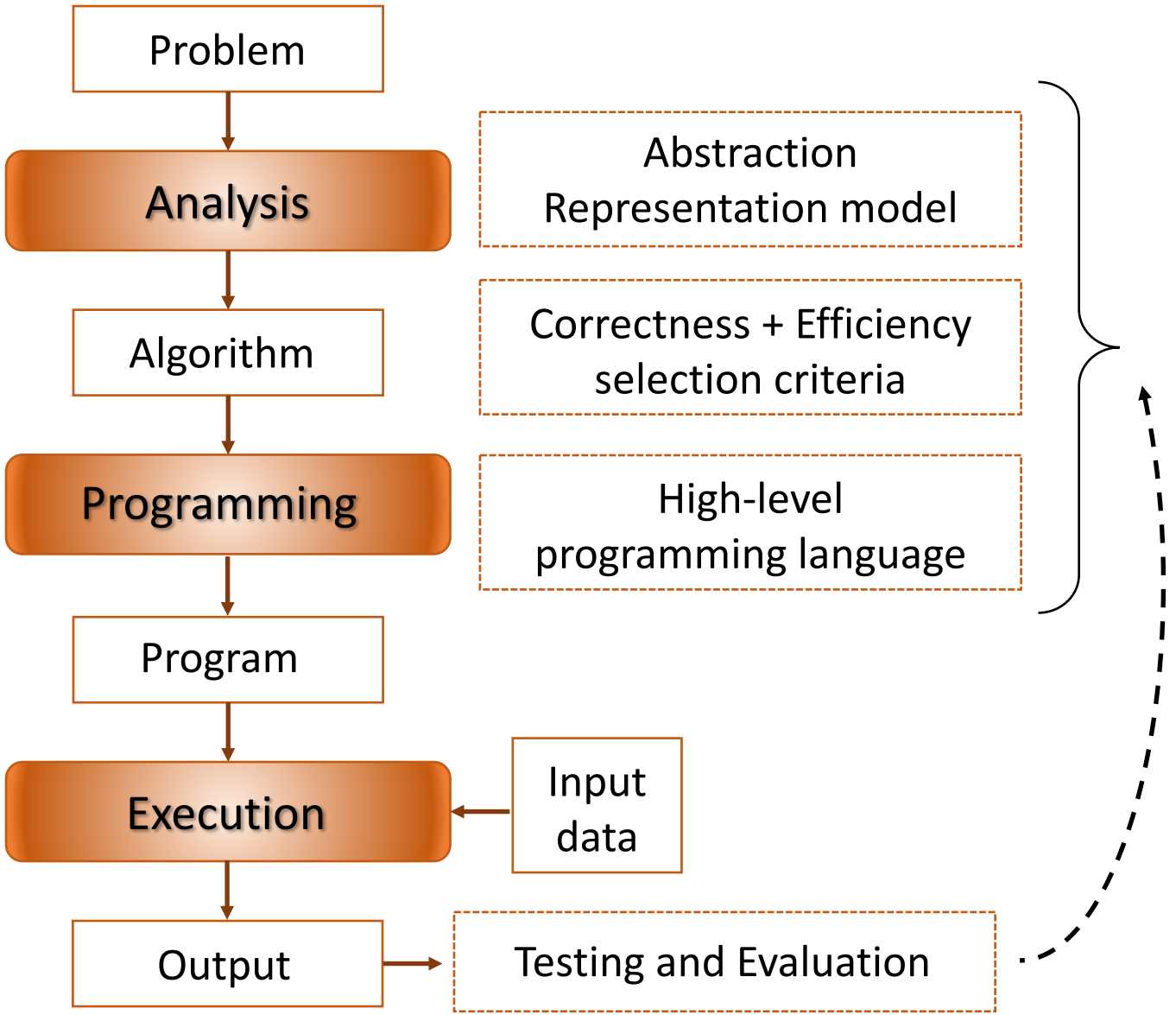


Correctness
Efficiency

- ✓ **Algorithm:** What to do (*step by step*) to solve the problem based on the defined problem abstraction / model → Problem-solving recipe
- ✓ **Programming (coding):** Use a high-level (computer) language to precisely describe (code) the algorithm → The code is used to tell the computer to do the things prescribed by the algorithm [what to do and how]
- ✓ **Computer hardware**: The high-level program is translated into a *machine-level* language which is then executed by the specific computer hardware



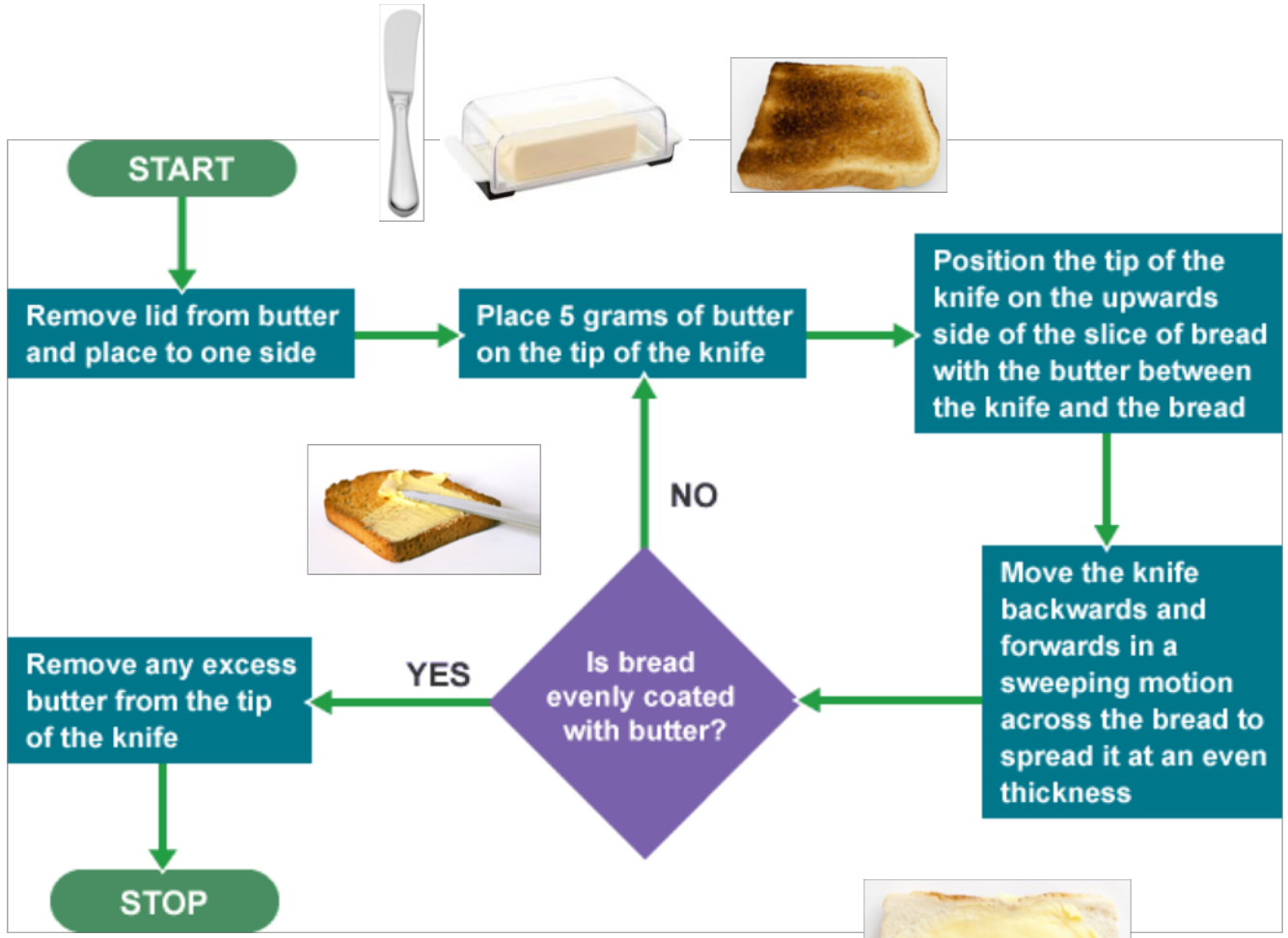
Introduction to the *science (art)* of computational problem solving



- **Algorithm:**
  - ✓ A finite list of instructions that describe a **computation**
  - ✓ when the instructions are executed on a provided set of inputs, the computation proceeds step by step through a set of well-defined states (configurations)
  - ✓ eventually, it ends, with some outputs being produced
- **Program:**
  - ✓ Algorithm encoding using a language that the computer understands
  - ✓ > *700 programming languages!*
  - ✓ Primitive constructs, syntax, static semantics, semantics

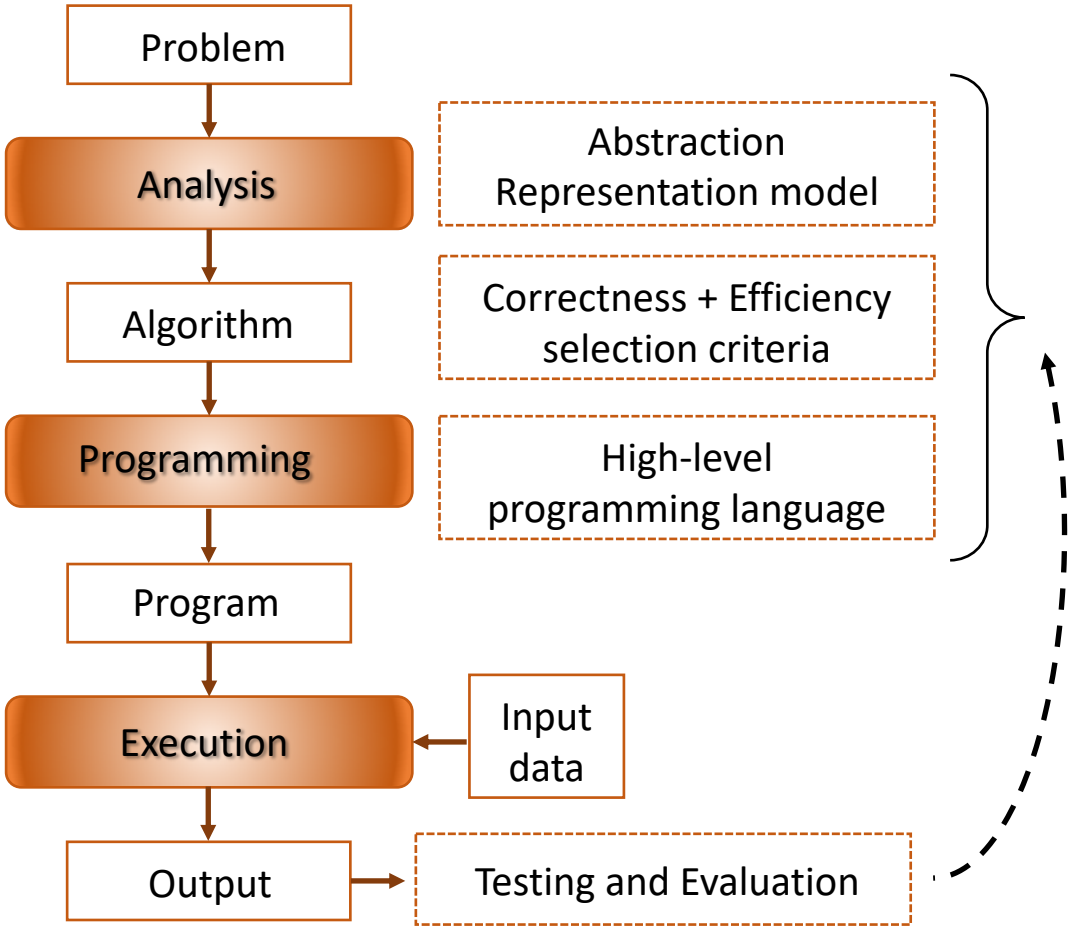
Algorithms

- ✓ A finite list of instructions that describe a **computation**
- ✓ when the instructions are executed on a provided set of inputs, the computation proceeds step by step through a set of well-defined states (configurations)
- ✓ eventually, it ends, with some outputs being produced



- Many ways to **describe** the same algorithm (including natural language)
- To be implemented, it needs to be described in an **unambiguous** form
- Can be applied to a **class of problems**

The 15-110 journey …



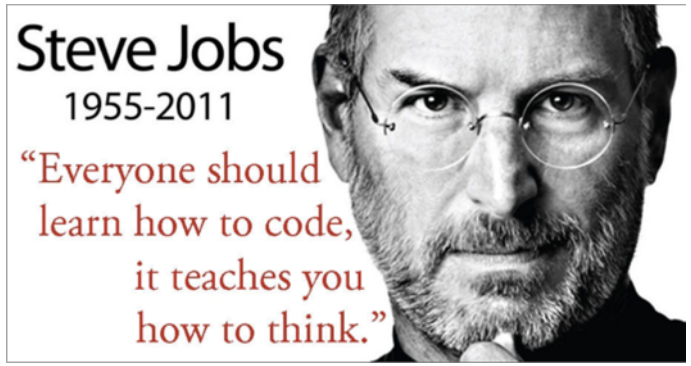
- ✓ **Problem analysis**: abstraction and representation
- ✓ Concepts central to design and understand **computation** (i.e., *algorithms*)
- ✓ Fundamental concepts of **programming**:
  - *Knowledge representation* (data types & structures)
  - *Flow control* (conditionals, iteration, recursion)
  - *Data organization* (lists, matrices, dictionaries)
  - *Decomposition, reusability, information hiding* (modules, functions, objects)
- ✓ ***Python* programming language**
- ✓ Computational **problem-solving techniques**
- ✓ **Correctness** and **efficiency** of algorithms (testing, error finding, complexity, optimization)
- ✓ **Data visualization** and **statistical analysis**
- ✓ **Randomness** and **simulation** (*Monte Carlo, GA, CA)*

Road map

- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

Motivations for following 15-110 …

- Understanding computation and learning computer programming helps shaping your mind, develops your critical thinking: it's like *math* but more interactive and more fun!



- ✓ Rational thinking for (general) problem analysis and solving
- ✓ Develop abstraction and formalization capabilities
- ✓ Devising an efficient solution to a difficult problem is always rewarding
- ✓ Finding errors in the code is as thrilling as as finding the assassin! ☺

- Why understanding computation and learning computer programming can be rewarding?

- Financial analysts: **$84,300**
- Market research analysts: **$63,230**
- Sales managers: **$121,060**
- Human resource specialists: **$60,350**
- Accountants and auditors: **$69,350**
- Management analysts: **$82,450**
- Personal financial advisors: **$90,640**

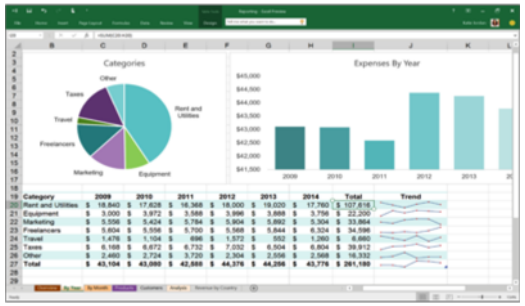
- **Software Engineer:** Average Salary $76,194
- **Senior Software Engineer:** Average Salary $89,599
- **Software Developer:** Average Salary $62,803
- **Senior Software Developer or Programmer:** Average Salary $100,303
- **Web Developer:** Average Salary $62,500
- **Programmer Analyst:** Average Salary $55,250

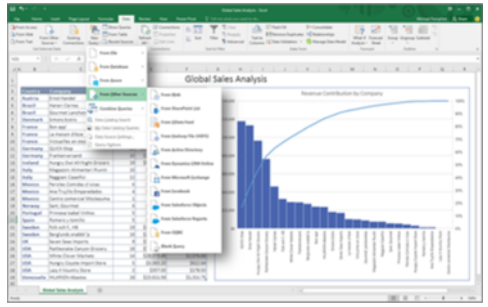
Salary range for a **Biologist**

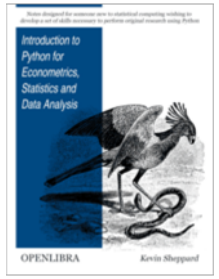
$44,946 to $56,017

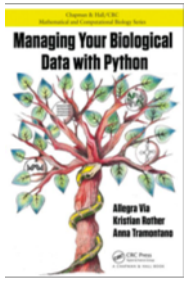
Motivations for following 15-110 …

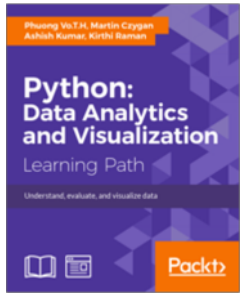
- Why understanding computation and learning computer programming can result in a professional advantage also for non-CS professionals?

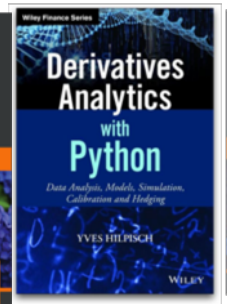


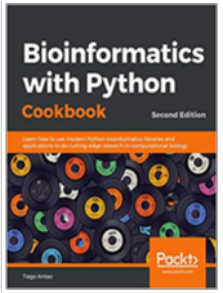


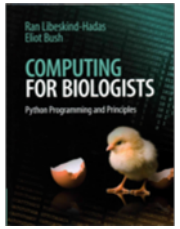


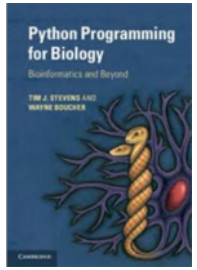


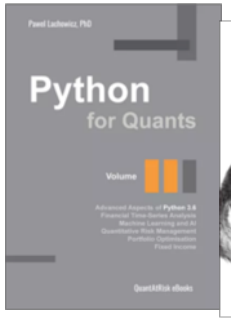


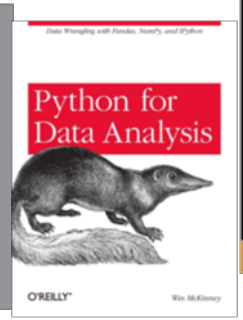


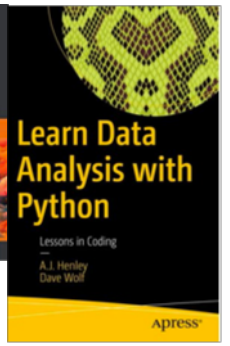










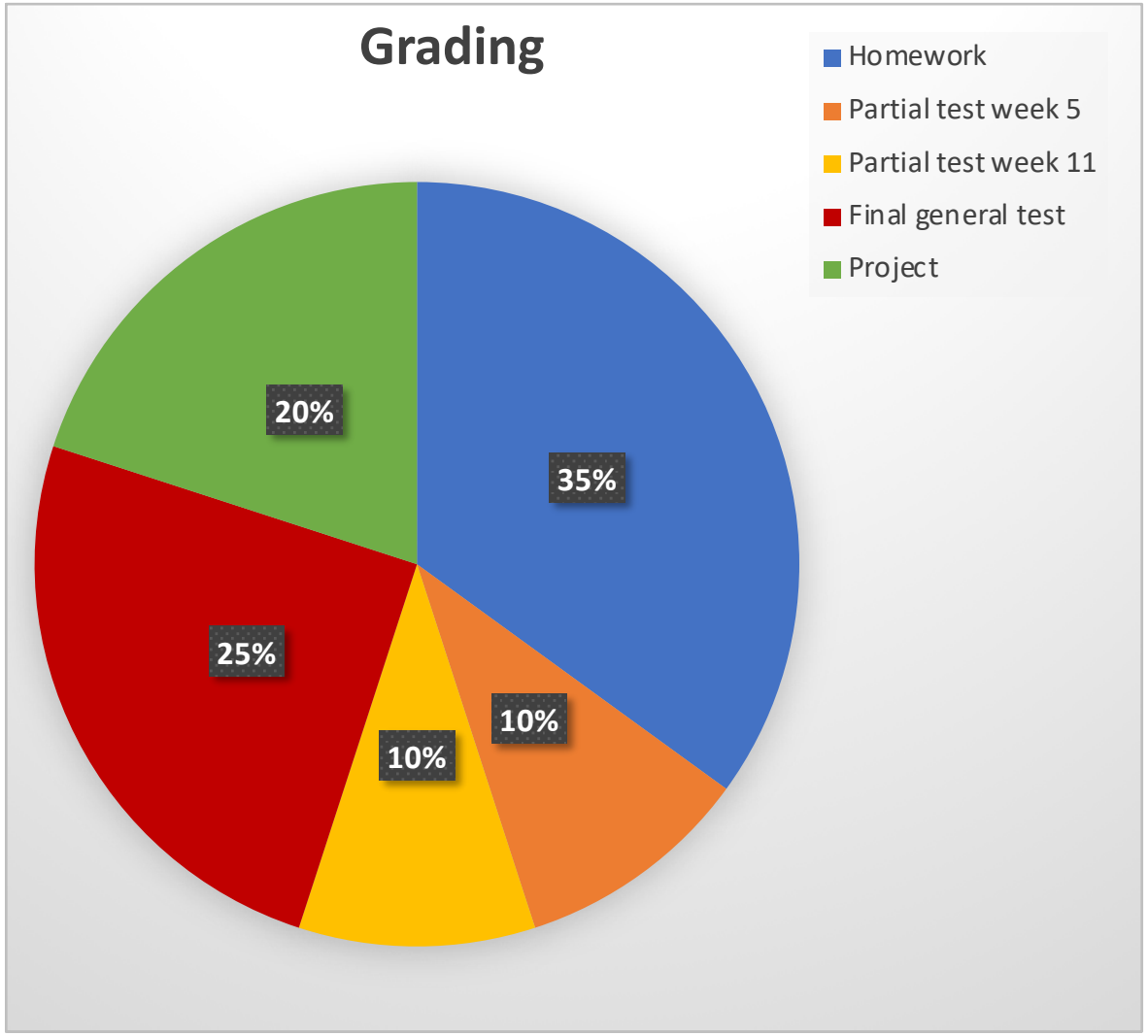


Road map

- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

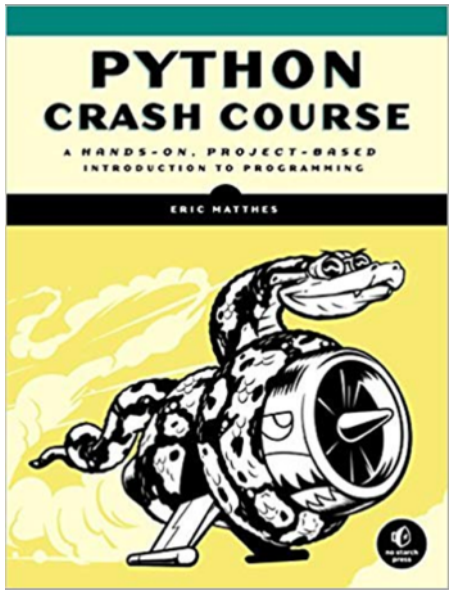
Logistics: Classes and Grading

- **Theory Classes**: Sundays, Tuesdays (1:30pm – 2:50pm), room 2052
- **Lab, hands-on**: Thursdays (1:30pm – 2:50pm), room 2062
- **Homework:** weekly, programming questions (autograded) + theory quizzes
- **Partial assessment tests:** two written tests during the course (at week 5 and week 11)
- **General assessment test:** one general written test at the end of the course
- **Project:** programming project issued at week 10 to be completed by the end of the course (deliver: code + report + 3m video presentation)

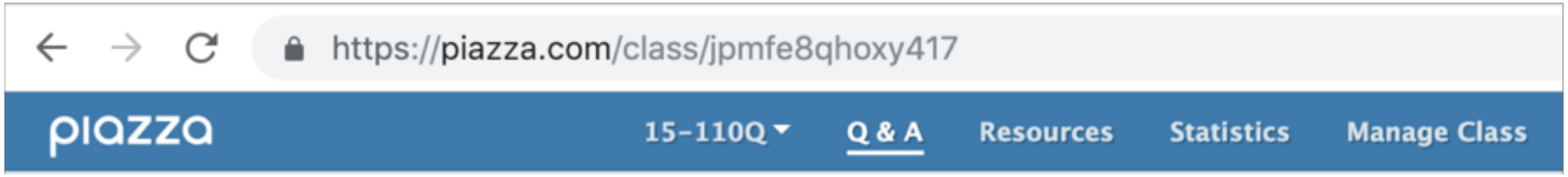


Logistics: Resources

- **Main sources for material:**
  - Lectures + lecture slides
  - Python reference book
  - Course website: https://web2.qatar.cmu.edu/cs/15110/



- **Piazza** for: Questions, announcements, discussions



- **Autolab** for submitting homework and project, and getting grades

Logistics: In-class rules

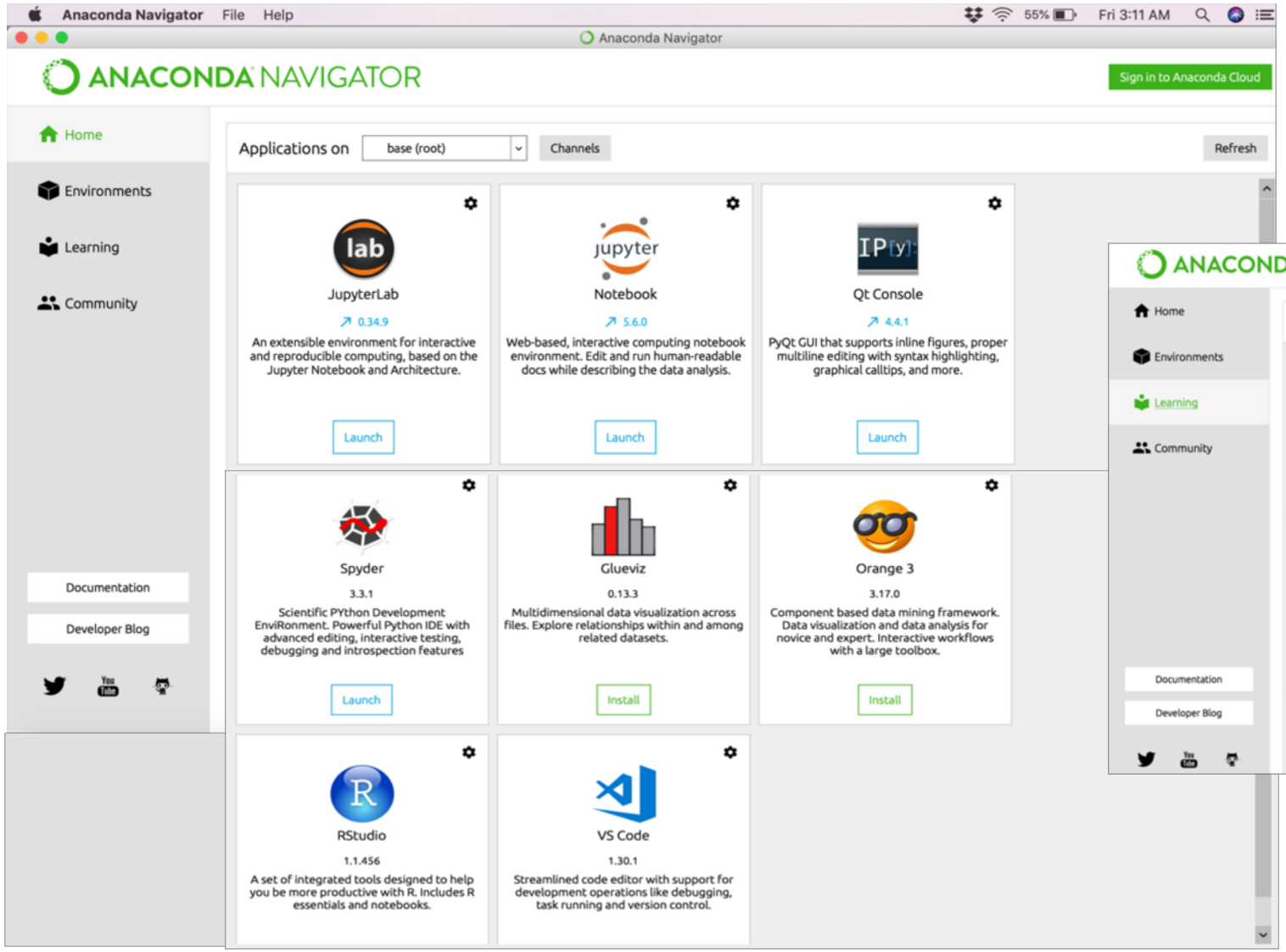
Be *alive* and **participate**!

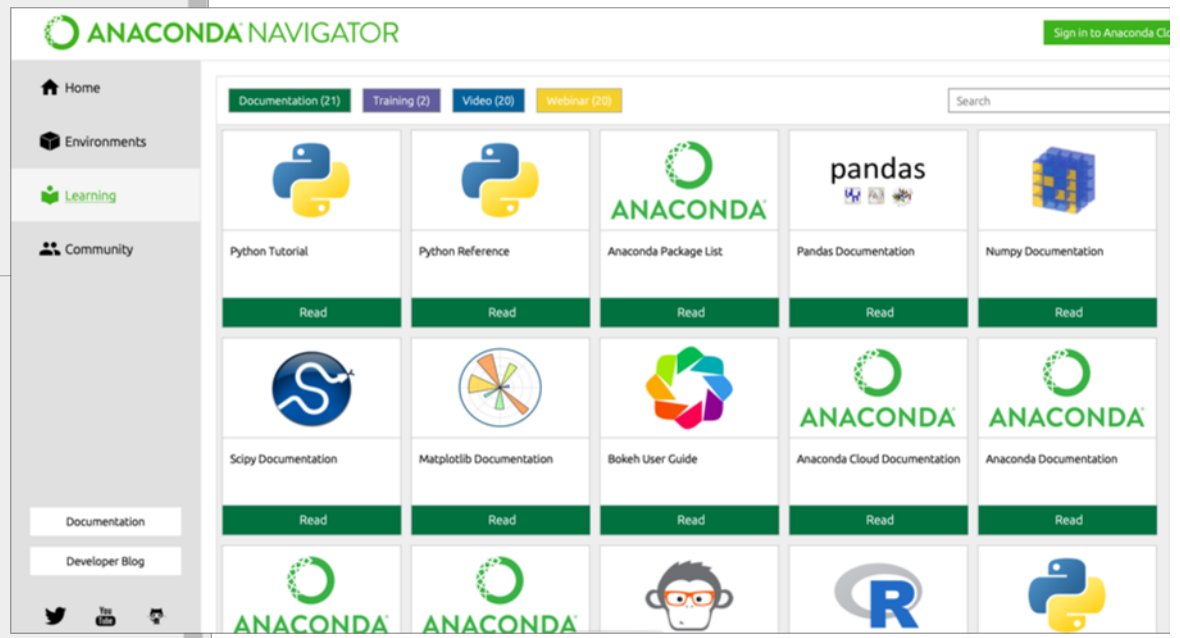
Ask **questions!**

Road map

- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

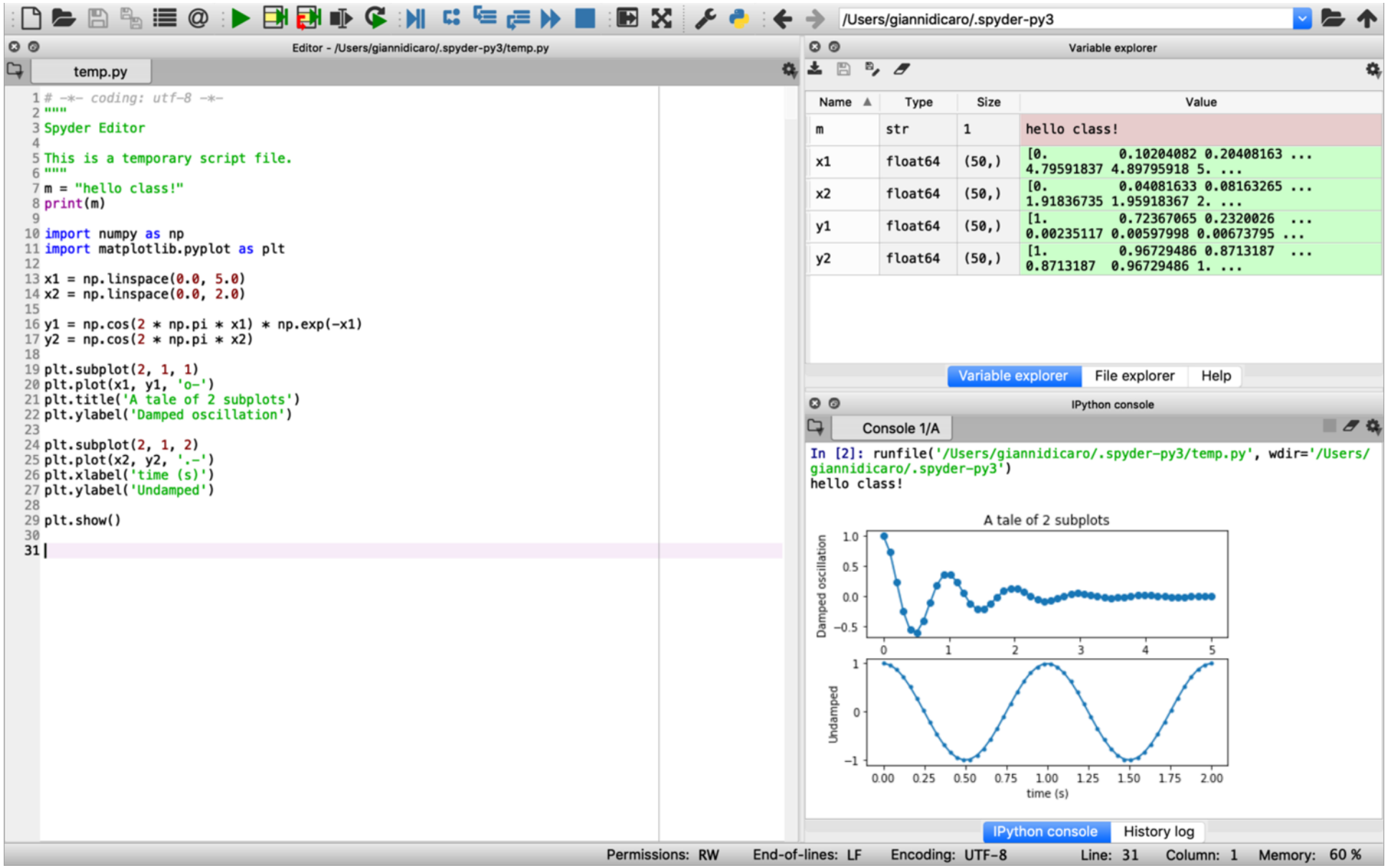
ANACONDA Navigator





https://www.anaconda.com/

Spyder: Python Integrated Development Environment (IDE)



Road map

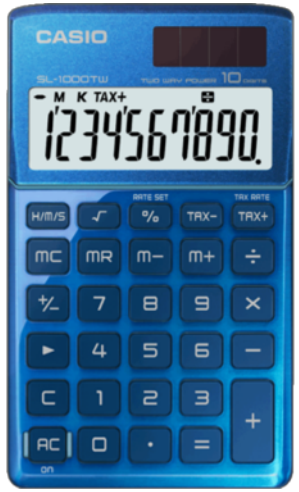
- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

What does a computer do?

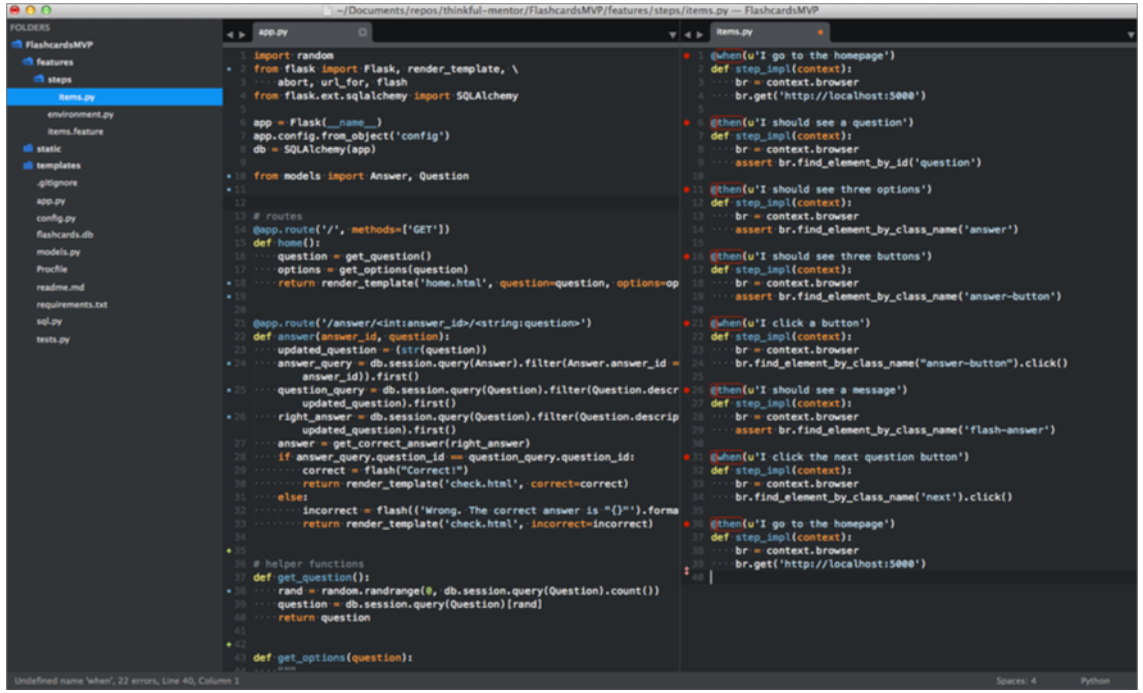
✓ Calculations (> billions per second) + Remember and access results (memory, TB of storage)

- What kind of calculations?

1. Built-in to some restricted language that the machine understands



2. Those that the **programmer defines**, based on a more flexible language



Types of knowledge to specify what to do

✓ In order to do what the programmer wants them to do, computers need precise **knowledge** **(**about the actions to execute, the task to perform, the environment the computer interact with)

- There are two main types of knowledge that can be used to program a computer:

1. **Declarative knowledge**
   - Involves knowing WHAT is the case
   - Collection of statements of **facts** (including **goals**) and **truth**
     - Square root y of a number x is such that $y \cdot y = x$
     - This pen is red; this is a mac notebook
     - (I need) a table for two
   - WHAT is true
   - Conscious, can be verbalized

2. **Imperative knowledge**
   - Involves knowing HOW to do something
   - It is a recipe, a sequence of steps / commands (*imperative*), possibly based on declarative knowledge
     - How to drive a car
     - How to make a strawberry cake
     - Based on declarative knowledge of the square root, an iterative procedure for finding $\sqrt{x}$
   - HOW to do things

Examples: Declarative vs. Imperative approach

Declarative approach

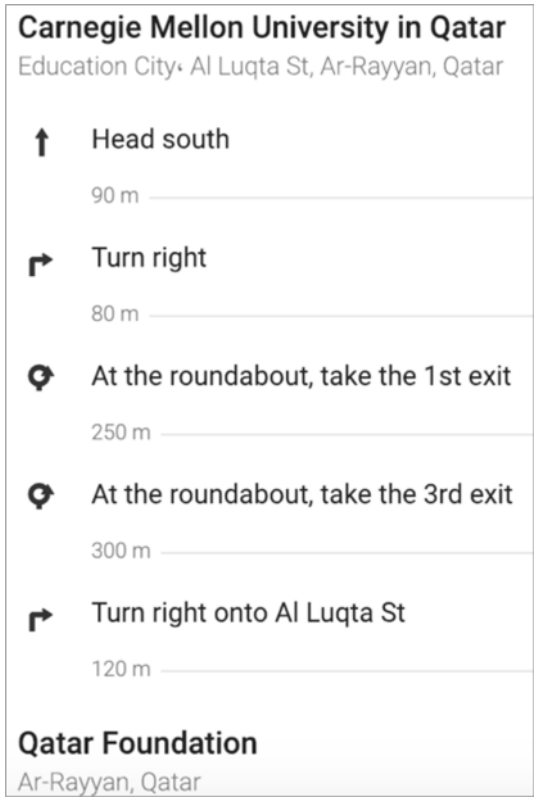
Imperative approach

At the front desk of the restaurant:

✓ Table for two, please

✓ I see that table for two located under the tree, on right side, is empty. My wife and I are going to walk over there over the free path and sit down.

I need to go from CMU-Q to QF:



Address is:
Qatar Foundation
Al-Rayyan, Qatar

Types of knowledge and programming paradigms: Declarative

Declarative knowledge, WHAT is true ➡ **Declarative programming languages**

- ✓ Allow to specify *what* it is **known** and *what* are the **goals,** *what* to compute
- ✓ No need to specify the implementation, or *how* the goals are achieved
- ✓ Express the **logic of a computation** without describing its *control flow*

- Examples: SQL, HTML, Prolog, Lisp, Haskell, …

15-150 (Prof. Giselle!)

- SQL: `SELECT * FROM Doctors WHERE Specialty='Dentist';`
- HTML: <img src="cake.jpg"> This combination of text and image looks bad on most browsers.
- Prolog:

Knowledge base

```
woman(mary).
man(joe).
happy(mary).
listens2Music(joe).
listens2Music(mary):- happy(mary).
playsAirGuitar(joe):- listens2Music(joe).
playsAirGuitar(mary):- listens2Music(mary).
```

Facts

Rules

Rule: `A :- B` A is implied by B: if B is true then also A is true

Query:

```
?- woman(mary).
?- woman(giselle).
?- playsAirGuitar(joe).
```

Types of knowledge and programming paradigms: Imperative

Imperative knowledge, HOW to do things ➡ **Imperative programming languages**

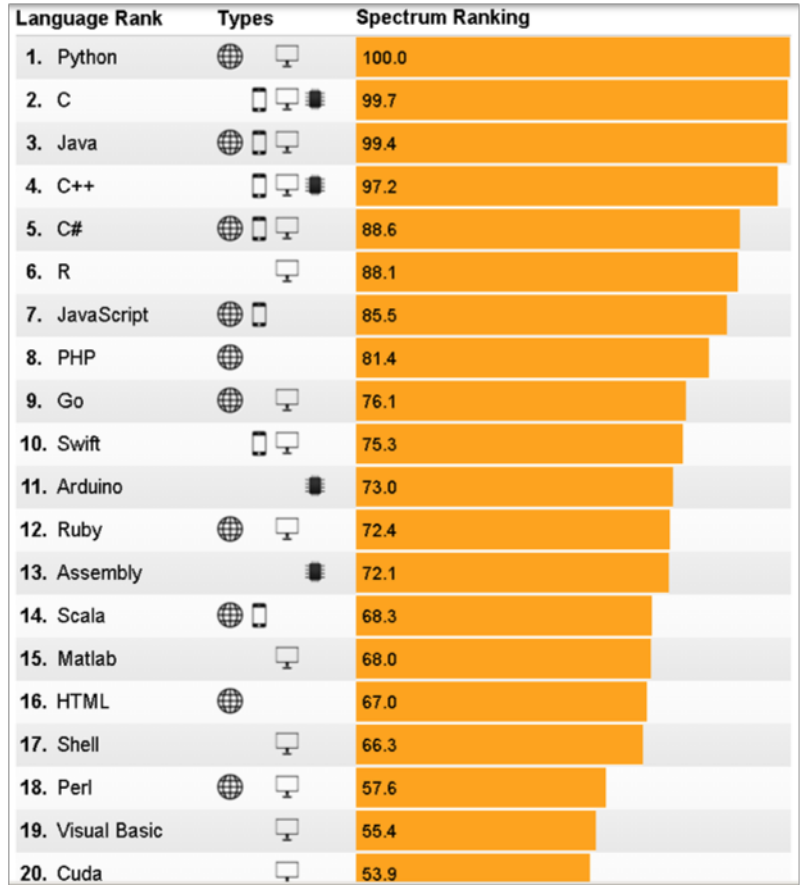
- ✓ Allow / need to tell the machine what we want to happen, *step by step*
- ✓ In *natural languages* the imperative mood expresses commands ↔ In *programming languages*, an imperative program consists of a list of **commands for the computer to perform**
- ✓ Imperative programming describes how step by step a program operates to achieve its goals
- ✓ We might use *declarative knowledge* to build the algorithm!
- ▪ Examples: C, C++, Python, C#, Java, Visual Basic, Fortran, …

We will focus on algorithms expressed using imperative knowledge and implemented using the imperative programming paradigm (as encoded in python language)

Types of knowledge and programming paradigms: Imperative

Good reasons to focus on the imperative approach (and on python):

- **Most used approach**, coming in a number of different flavors
- Quite *intuitive* as a first approach to computing
- The **hardware** architecture of almost all computers is designed after the imperative approach
- Nearly all computer hardware is designed to execute **machine code**, which is native to the computer and is written in the imperative style



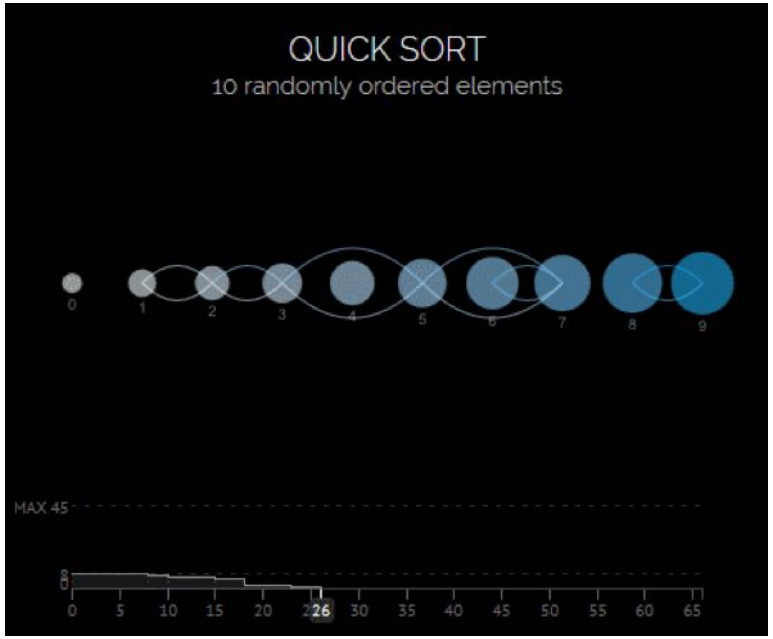
https://adtmag.com/articles/2017/07/24/ieee-spectrum-ranking.aspx

Types of knowledge and programming paradigms: Imperative

What is the general “recipe” of imperative knowledge for problem solving?

- ✓ sequence of (simple) **steps**
- ✓ **flow control**: specifies when each step is being (conditionally, if-then-else) executed or repeated (while, until, for)
- ✓ means of determining when to **stop**

Algorithm



Example: Imperative calculation of the square root

Class of problems to solve: given a real number x, what is the value of $\sqrt{x}$?

- *Declarative knowledge*: Square root y of a number x is such that $y \cdot y = x$ (from $y = \sqrt{x}$, squaring both sides)

1. Start with an arbitrary guess value, g
2. **If** $g \cdot g$ is close enough to x (with a given numeric approximation)

   Then `Stop`, and say that g is the `answer`
3. **Otherwise** create a new guess value by averaging g and x/g:

$$g = \frac{(g + x/g)}{2}$$

4. **Repeat** the steps 2 and 3 until $g \cdot g$ is close enough to x

Flow control: conditionals

Stop condition

Simple steps

Flow control: repeat

Heron of Alexandria, 60 A.D.
(Babylonians, much earlier)

Example: Imperative calculation of the square root

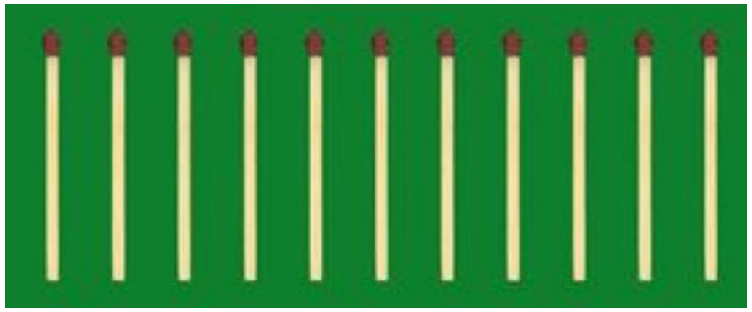
- $x = 25, \quad y = \sqrt{25}$?

1. Start with an arbitrary guess value, $g = 3$
2. $(3 \cdot 3 = 9) - 25 \approx 0$? No
3. New guess: $g = \frac{(3+25/3)}{2} = 5.66$
4. $(5.66 \cdot 5.66 = 32.04) - 25 \approx 0$? No
5. New guess: $g = \frac{(5.66+25/5.66)}{2} = 5.04$
6. $(5.04 \cdot 5.04 = 25.4) - 25 \approx 0$? Yes

   $\rightarrow \quad y = 5.04$

Guess-and-check algorithm

Another example: algorithm for one-pile NIM-11 game



- **Rules:**
  - 11 items (e.g., matches, cards) are on the table
  - Two players taking turn
  - At each turn a player removes k items from the table, $1 \leq k \leq 3$
  - The player taking the last item(s) loses the game

Algorithm (winning strategy for first player):

1. Remove two items at the first turn
2. If the other player removes k items, remove $4 - k$ items
3. Repeat 2 until there are no more items to remove

Road map

- General overview: what this course is about
- Utility of the course
- Logistics: classes, labs, homework, exams, grading, website, books, piazza, rules
- Software for python programming
- Computers: ways to tell a computer what to do
- Computers: general architecture and way of functioning

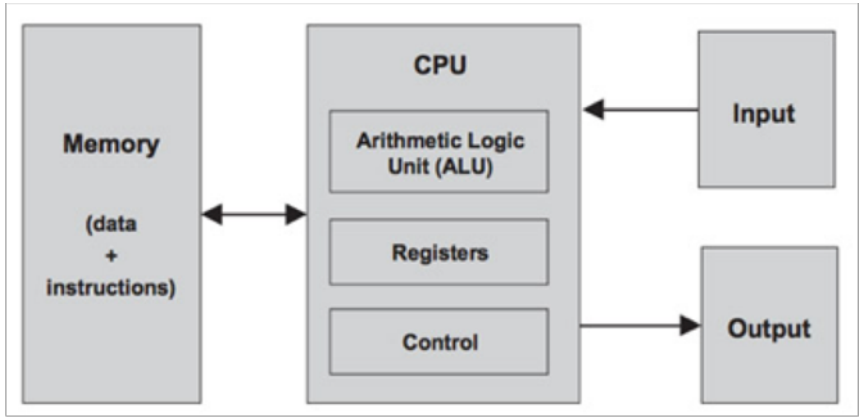
Let's get a closer look into the *machine*

How to capture an algorithm in a process operated by a *machine*?

Fixed program computer:
machines that perform one or a limited set of predefined actions

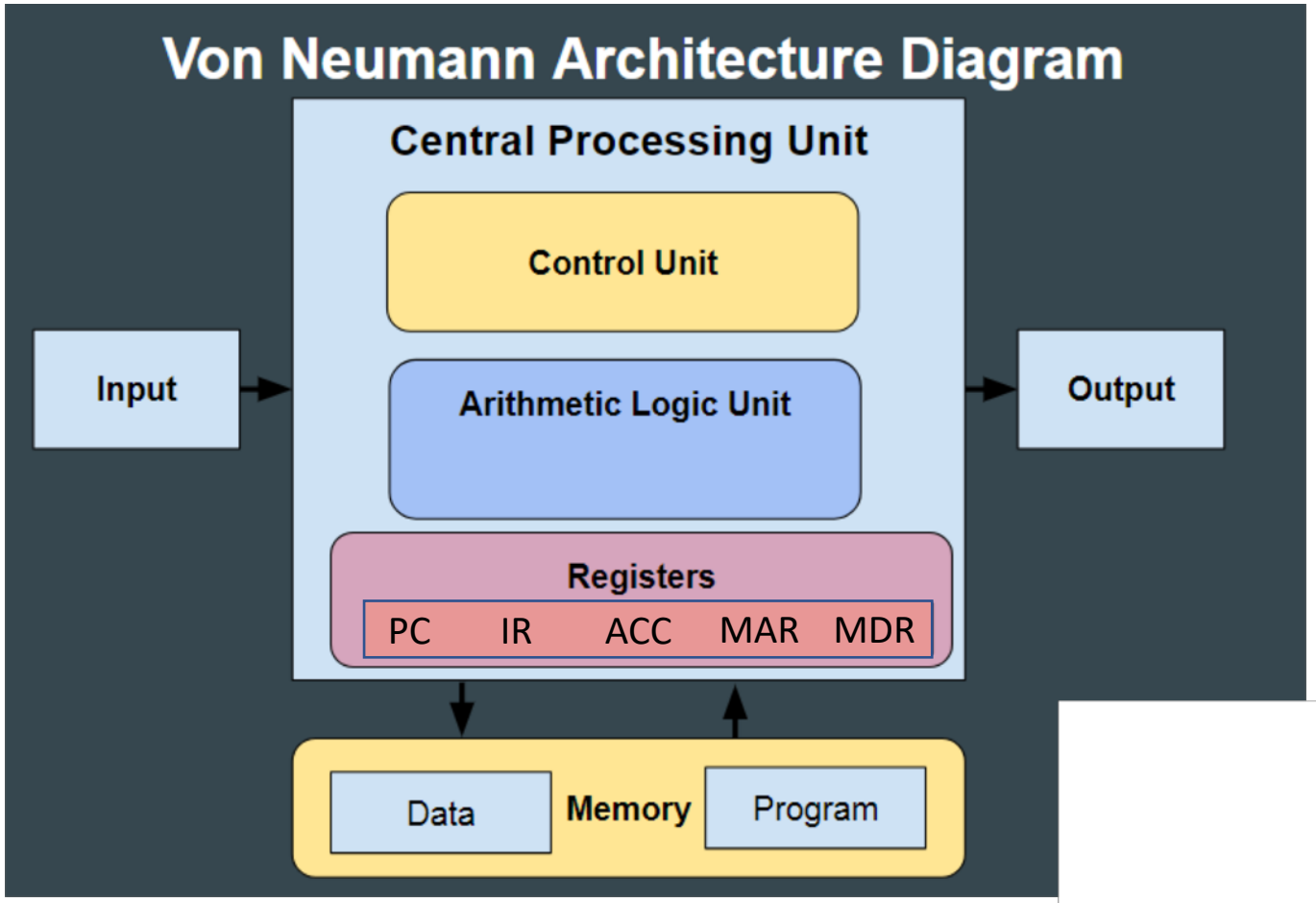
Stored program computer:
machine that stores and execute custom instructions ⟶ machines that can perform a large variety of tasks

What is the basic architecture of a (stored program) computer, that enables it to be a versatile, "programmable" machine?



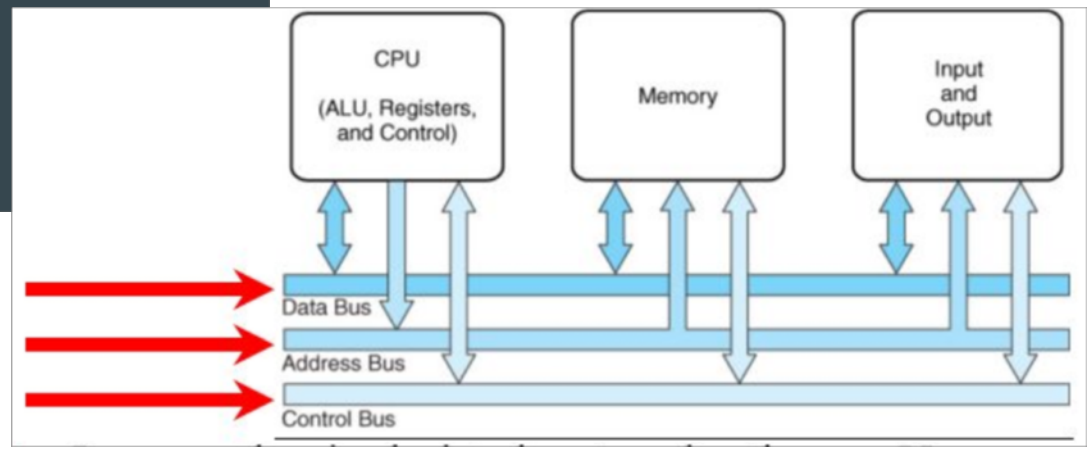
General computer architecture

Keyboard
Mouse
Joypad
Disk
….



Disk
External memory
Monitor
….

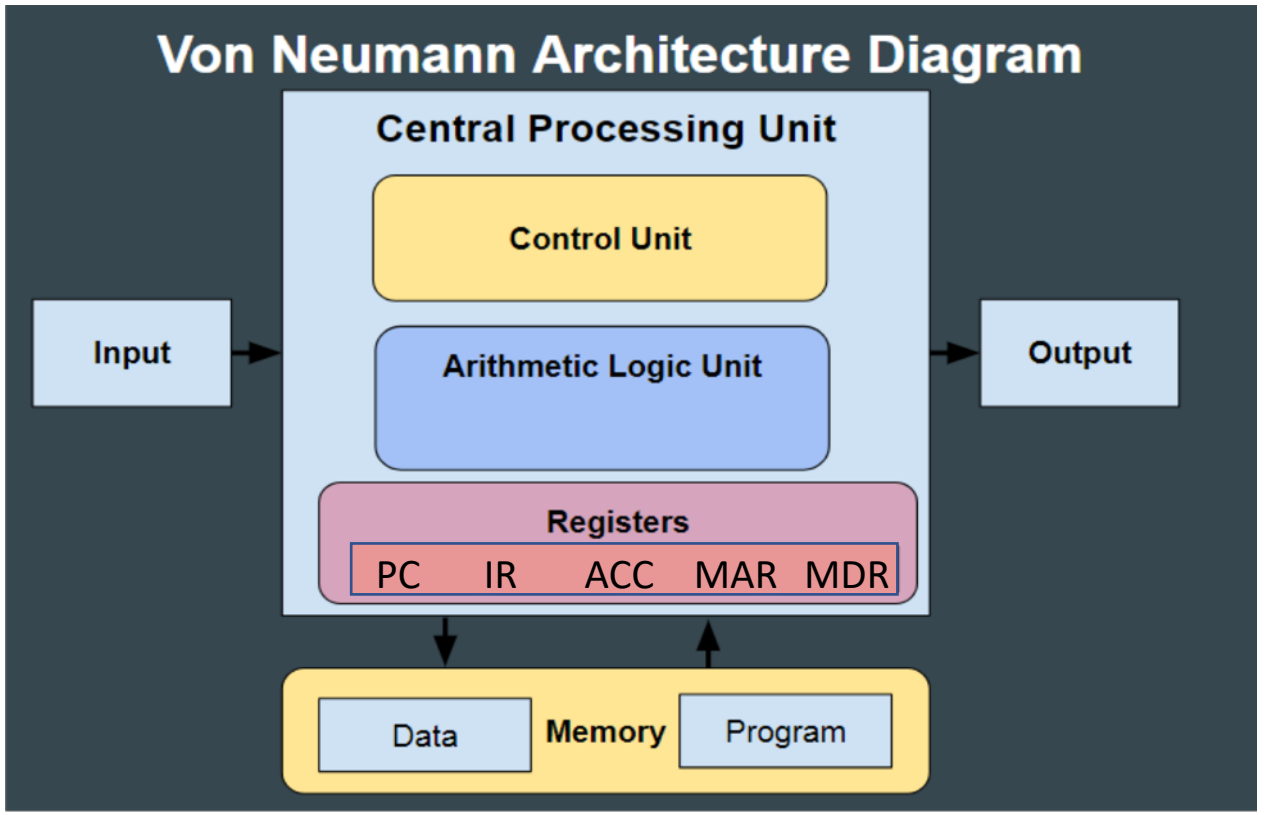
Internal memory
RAM: volatile
ROM: permanent

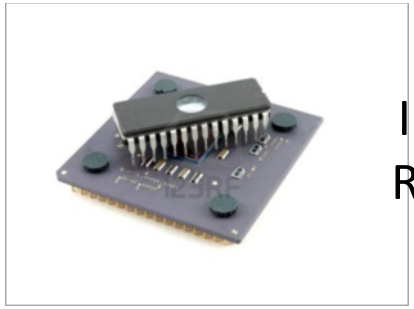


Steps of execution: loading the computer program

Boot process:
Operating System

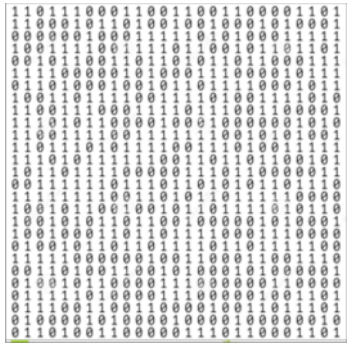
Storage memory:
OS programs



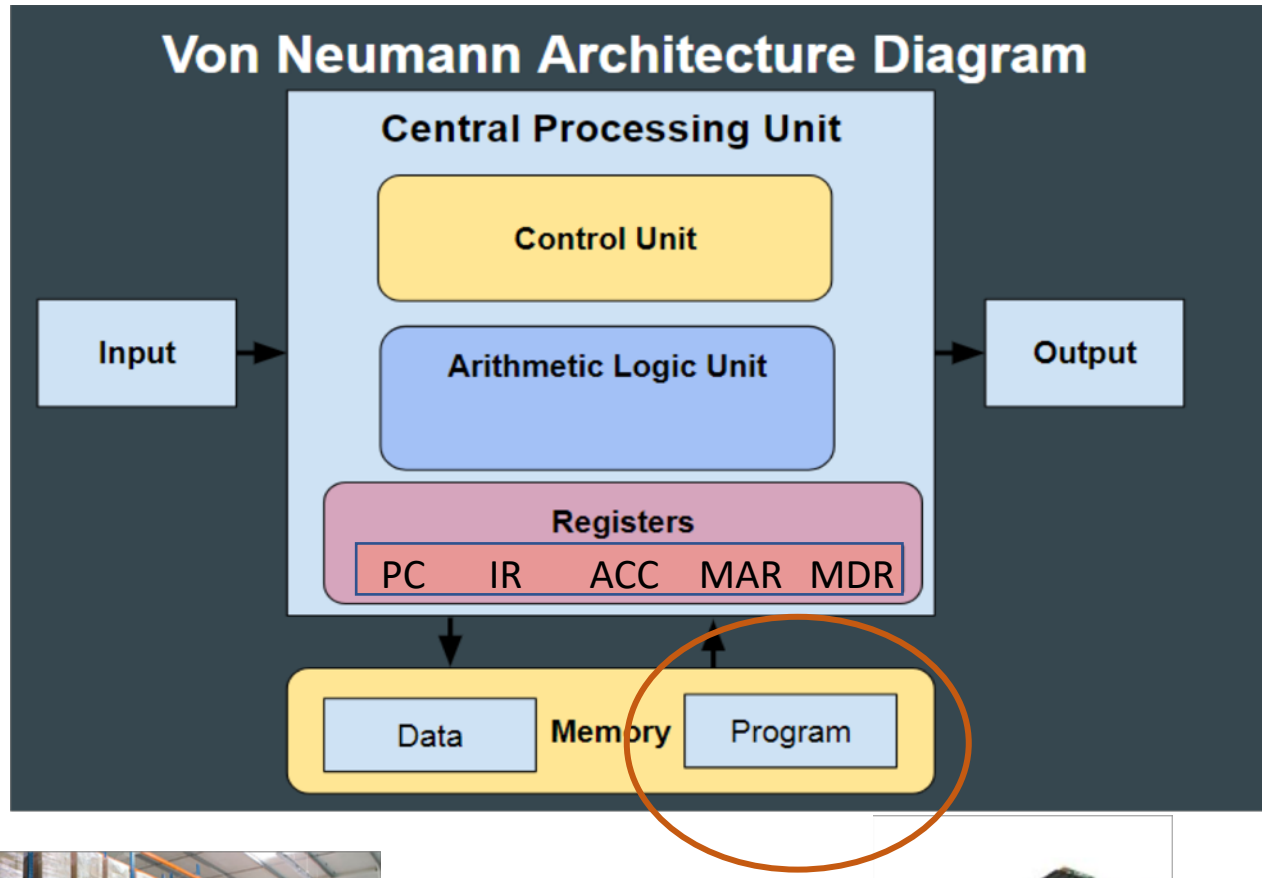


Internal memory
ROM: permanent

Steps of execution: loading the computer program



Storage memory:
User programs



Program encoding:

- Sequence of binary digits (01) that the machine can understand
- Sequence of instructions

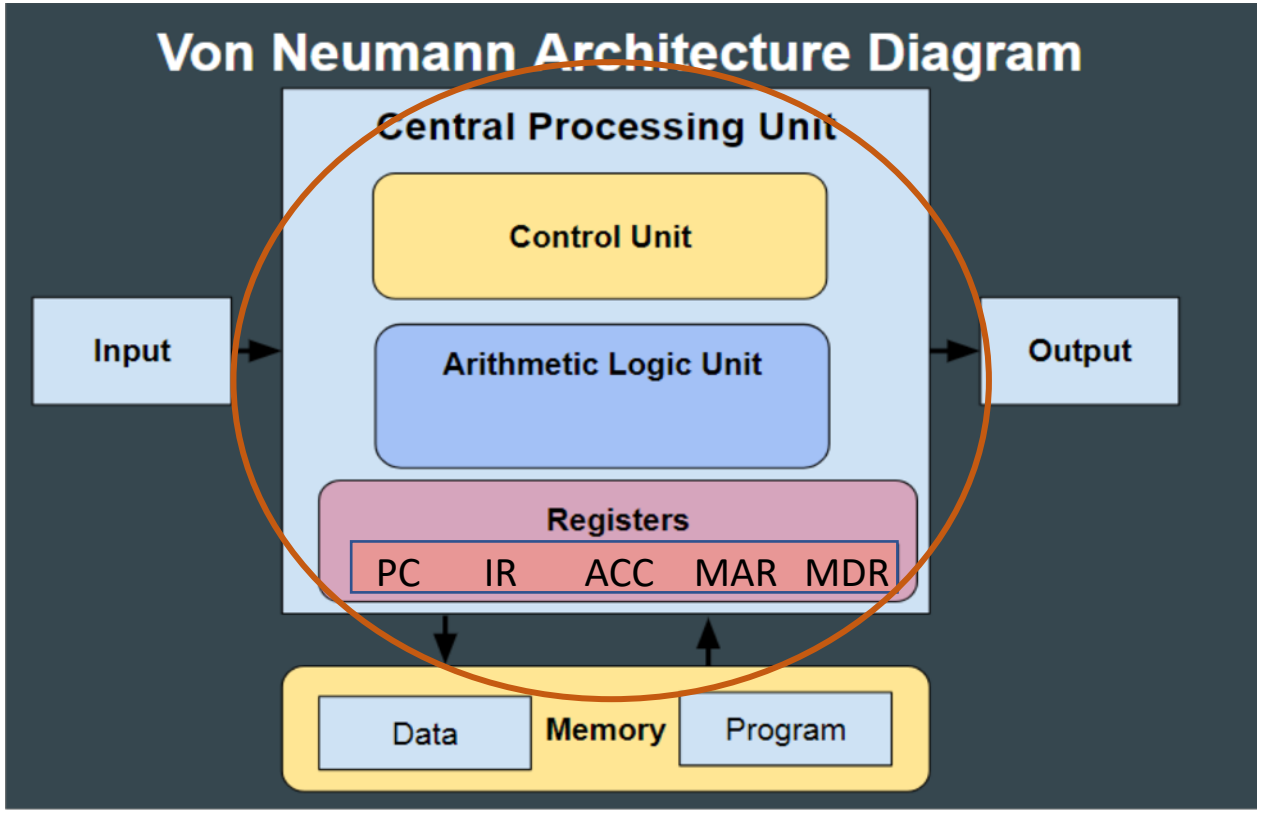
Internal memory
RAM: volatile

Steps of execution: Fetching instructions

CPU:
Clock-based system

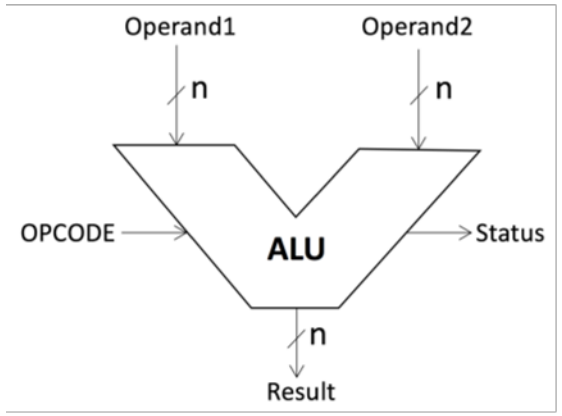
GHz(s)!

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



- Control Unit:
  - The (first) next instruction if fetched from memory and stored in the IR register
  - The PC register keeps track of the instruction to be executed next

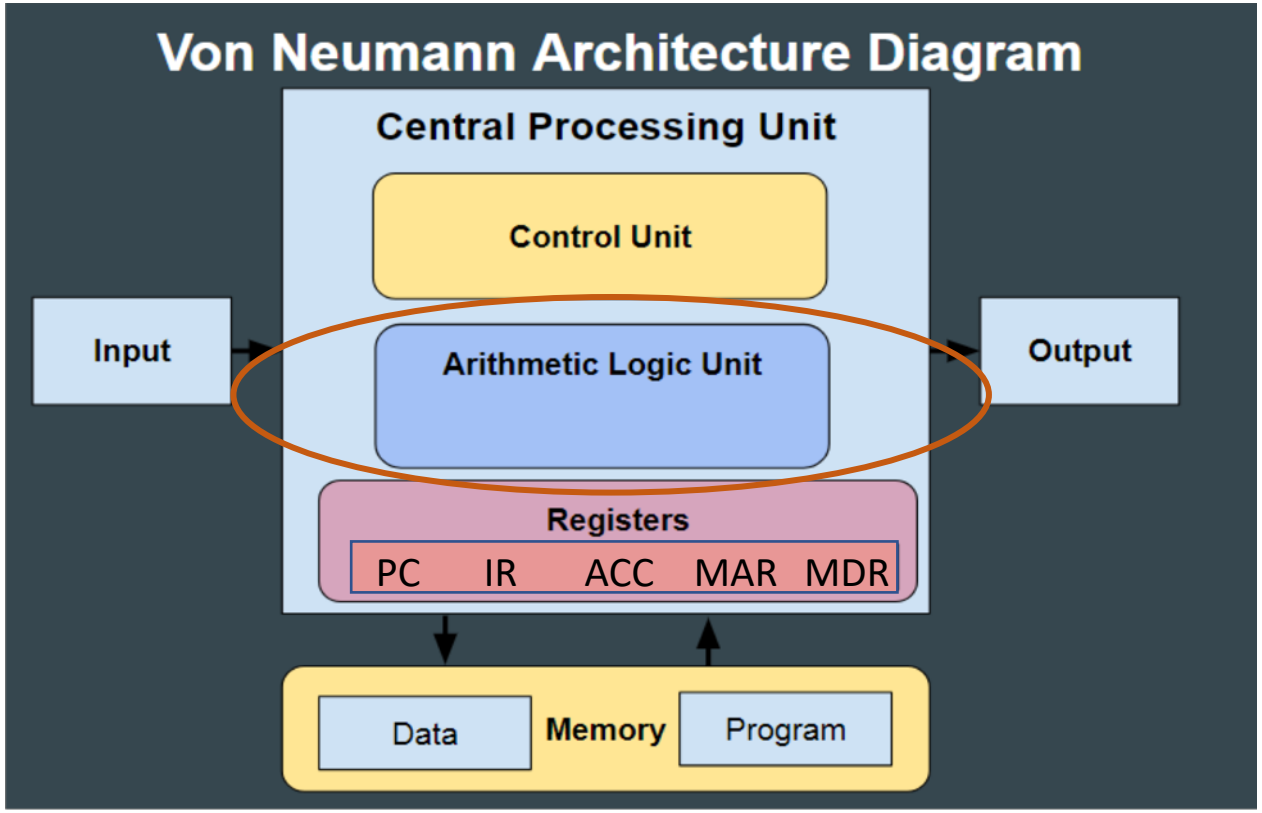
Steps of execution: Performing operations



ALU:
Where operations are performed

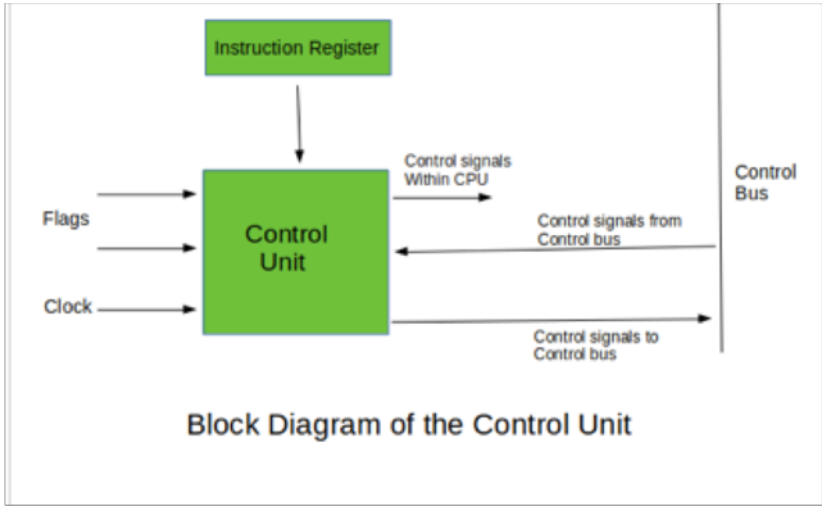
What operations?

- Addition
- Comparison (equal, different)
- Moving data
- … (optional)



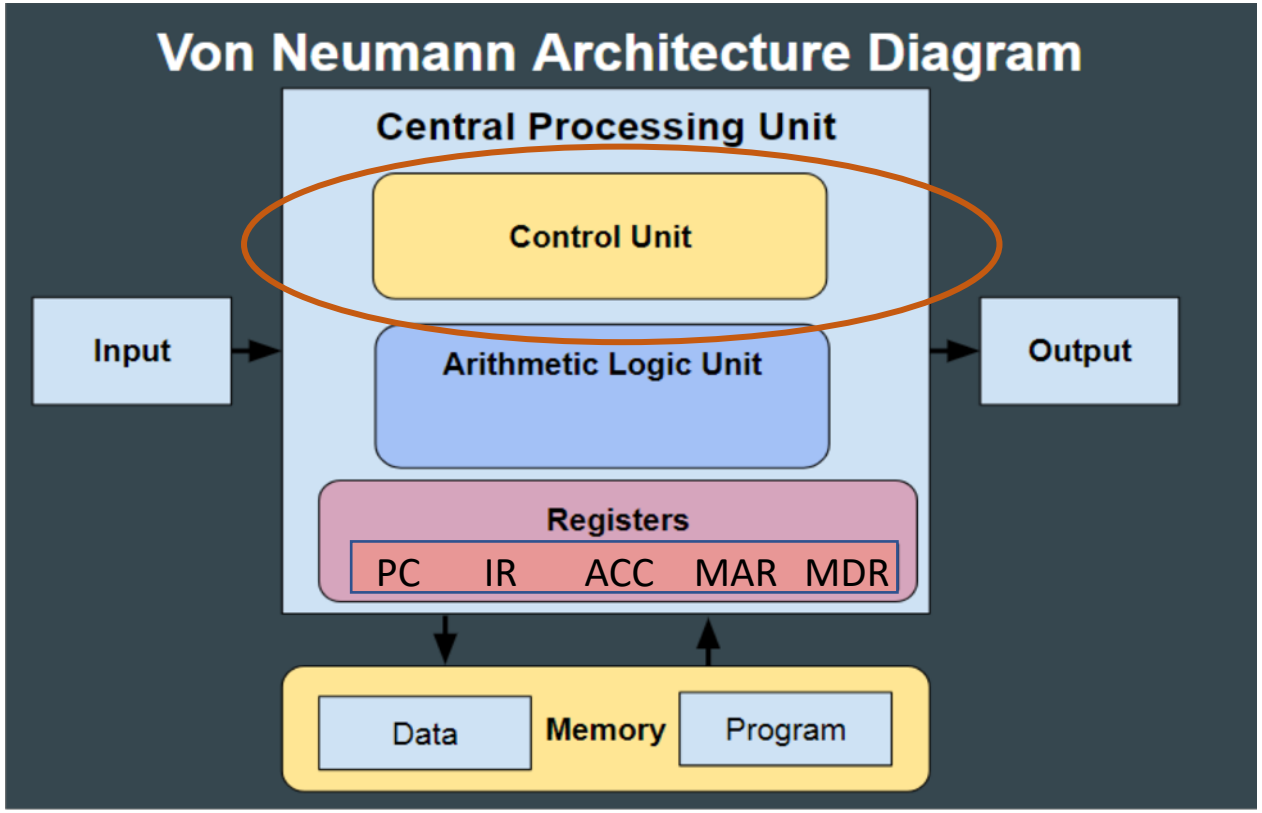
- ALU:
  - Gets the instructions from the CU
  - Gets data from the memory
  - Store data in the memory

Steps of execution: Repeating and branching



Block Diagram of the Control Unit

CU:
Keep iterating the process
Perform branching if necessary



- CU:
  - E.g., a test operation has returned False, then the next operation is not the next in the sequence
  - PC gets updated accordingly

Summary

- Overview of the topics: a journey into computation
  - Problem solving
  - Designing and understanding algorithms
  - Fundamentals programming constructs
  - Core techniques for computational problem-solving
  - Python as programming language
  - Tools (graphics, simulation, data manipulation)
  - Applications

- We have got some basic intuitions on computation and on computers
- Imperative vs. Declarative knowledge representation and programming langauges
- Basic functioning of a computer
- Hierarchy of languages (next time)

- We have set the basics for the course, next we will start with Python!