Introduction to Programming Language (ITP101)

Module Roadmap

Mulualem Teku

July 29, 2019


Welcome

Module Specifics

- **Name**: Introduction to Programming Language
- **Code**: `ITP101`
- **Prerequisites**: None

Activity	**Hrs/Week**	**Credit Hrs**
Lecture	4	60
Lab	3	45
Independent study	1	15
Total		120

ITP101: General Objective

This module is designed to help students with no prior exposure to computer science or programming to think computationally and write useful programs. The module focuses on the fundamentals of programming to provide a foundation for the study of more advanced programming modules.

Learning Outcomes

At the end of the module, you will be able to:

1. Define the concepts of decomposition, pattern matching, abstraction, and algorithmic problem-solving approaches.
2. Develop solution strategy for solving a given computational problem using algorithm/pseudocode/flowchart.
3. Use core Python objects to implement your solution.
4. Effectively use Python packages to solve a given problem.
5. Apply debugging/exception handling techniques to address errors and remove/handle programming errors.

Learning Outcomes

At the end of the module, you will be able to:

1. Define the concepts of decomposition, pattern matching, abstraction, and algorithmic problem-solving approaches.
2. Develop solution strategy for solving a given computational problem using algorithm/pseudocode/flowchart.
3. Use core Python objects to implement your solution.
4. Effectively use Python packages to solve a given problem.
5. Apply debugging/exception handling techniques to write error-free/handle programming errors.

Learning Outcomes

At the end of the module, you will be able to:

1. Define the concepts of decomposition, pattern matching, abstraction, and algorithmic problem-solving approaches.
2. Develop solution strategy for solving a given computational problem using algorithm/pseudocode/flowchart.
3. Use core Python objects to implement your solution.
4. Effectively use Python packages to solve a given problem.
5. Apply debugging/exception handling techniques to discover and remove/handle programming errors.

Learning Outcomes

At the end of the module, you will be able to:

1. Define the concepts of decomposition, pattern matching, abstraction, and algorithmic problem-solving approaches.
2. Develop solution strategy for solving a given computational problem using algorithm/pseudocode/flowchart.
3. Use core Python objects to implement your solution.
4. Effectively use Python packages to solve a given problem.
5. Apply debugging/exception handling techniques to discover and remove/handle programming errors.

Learning Outcomes

At the end of the module, you will be able to:

1. Define the concepts of decomposition, pattern matching, abstraction, and algorithmic problem-solving approaches.
2. Develop solution strategy for solving a given computational problem using algorithm/pseudocode/flowchart.
3. Use core Python objects to implement your solution.
4. Effectively use Python packages to solve a given problem.
5. Apply debugging/exception handling techniques to discover and remove/handle programming errors.

Syllabus (I)

I) **Basics of Computational Thinking**

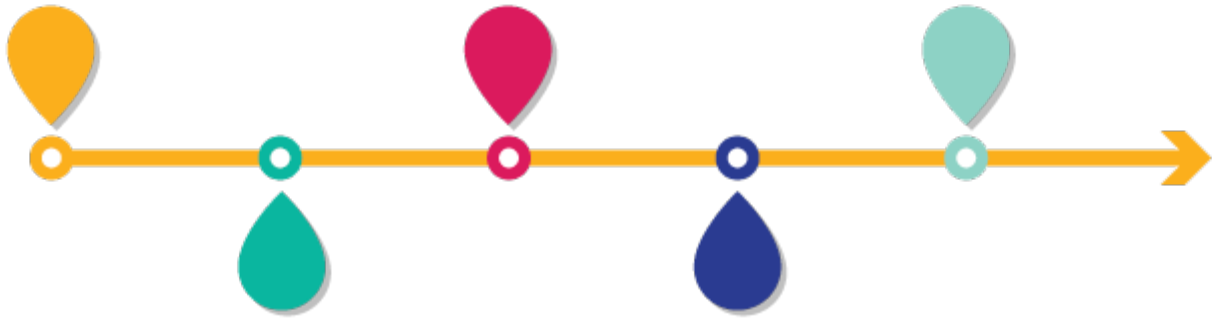
- Computational thinking techniques
- Algorithms, Flowcharts, Pseudocodes
- Process and building blocks of problem solving
- Programming paradigms

Syllabus (II)

II) Intro to Python Programming

- Identifiers, data types, operators, statements, I/O...
- Control flow (branching and looping)
- Core Python objects: Functions, Lists, Tuples, Strings and Dictionaries
- Debugging and Exception handling
- Python packages
- File processing

Timeline



See the timeline (semester plan) I have posted on VLE.

Assessment Components

Mid-semester Examination	*20%*
{Assignments/Class tests/Practical exams/labs*} [a]	*40%*
Semester End Examination	*40%*

[a] The hands-on component is the heart of the course

- Total Continuous Assessment (CA) 60%
- Semester End Examination (SE) 40%

Resources

1. Dierbach C, *Introduction to computer science using Python: A computational problem-solving focus.*
2. Downey A B, *Think Python: How to think like a computer scientist.*
3. Guttag J V, *Introduction to computation and programming using Python: With application to understanding data (2nd ed)*
4. Lutz M, *Learning Python.*
5. Rao R N, *Core Python programming.*

Other topic-specific, online and offline resources TBA as deemed necessary.

Policies

- Check out the college's various policies/rules and regulations (e.g. attendance, examination, etc).

Policies

- Check out the college's various policies/rules and regulations (e.g. attendance, examination, etc).
- Deadlines
- Plagiarism (A or B?)

A	B
• Sharing solutions/code verbatim: copying, retyping, submitting copy of a file. • Coaching: helping your friend to write code/solution line-by-line.	• Helping others use tools, clarify or help understand a concept. • Helping others debug their code.

- Automated plagiarism detection tools will be used. Be careful!

Policies

- Check out the college's various policies/rules and regulations (e.g. attendance, examination, etc).
- Deadlines
- Plagiarism (A or B?)

A

- Sharing solutions/code verbatim: copying, retyping, submitting copy of a file.
- Coaching: helping your friend to write code/solution line-by-line.

B

- Helping others use tools, clarify or help understand a concept.
- Helping others debug their code.

- Automated plagiarism detection tools will be used. Be careful!

Policies

- Check out the college's various policies/rules and regulations (e.g. attendance, examination, etc).
- Deadlines
- Plagiarism (A or B?)

A

- Sharing solutions/code verbatim: copying, retyping, submitting copy of a file.
- Coaching: helping your friend to write code/solution line-by-line.

B

- Helping others use tools, clarify or help understand a concept.
- Helping others debug their code.

- Automated plagiarism detection tools will be used. Be careful!

Help!

- Approach/mail me or LA (if any). Don't shy away.
- Assignment/topic-specific queries, forum on VLE.
- Online and offline resources.
- Practice, practice and practice!

FAQs

(1) I have done little to no programming before. Am I in the right place?

(2) How is the course load (assignments, lab activities, practical exams,...)?

(3) How useful is this course? Where do I apply it? Job prospects?

(4) Do I have to attend every lecture/lab session? What if I don't?

(5) I want to do really well in ITP101/be good at programming. Any tips? Your freshman experience?

(6) What if I have suggestions/constructive feedbacks/complaints/... regarding the module?

(7) Any other question?

Computing Basics: a Refresher

Brainstorm

1. Data vs Information? Information Technology?

Information Technology

"The application of computers and telecom equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise." Wikipedia

2. Computer?

3. Program?

Brainstorm

1. Data vs Information? Information Technology?

Information Technology

"The application of computers and telecom equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise." Wikipedia

2. Computer?

3. Program?

Brainstorm

1. Data vs Information? Information Technology?

Information Technology

"The application of computers and telecom equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise." Wikipedia

2. Computer?

3. Program?

Brainstorm

1. Data vs Information? Information Technology?

Information Technology

"The application of computers and telecom equipment to store, retrieve, transmit and manipulate data, often in the context of a business or other enterprise." Wikipedia

2. Computer?

3. Program?

Computer

- "One who computes". Referred to human computers before the advent of electronic computers.
- An electronic device for storing and processing data (typically in binary form) according to instructions given in a program.
- A machine that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming. (Wikipedia)

Introduction: Computation

- Computers were born for this. But why compute/program?
- "Necessity is ..."
- How many likes for, "To make life easier" ?

Introduction: Computation

- Computers were born for this. But why compute/program?
- "Necessity is ..."
- How many likes for, "To make life easier" ?

Introduction: Computation

- Computers were born for this. But why compute/program?
- “Necessity is ...”
- How many likes for, “To make life easier” ?

Introduction: Computation

- Computers were born for this. But why compute/program?
- "Necessity is ..."
- How many likes for, "To make life easier" ?

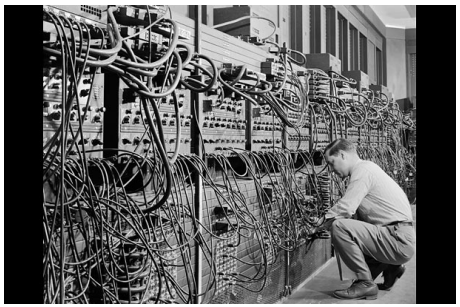
"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind."

E. Dijkstra, 1972 Turing Award Lecture

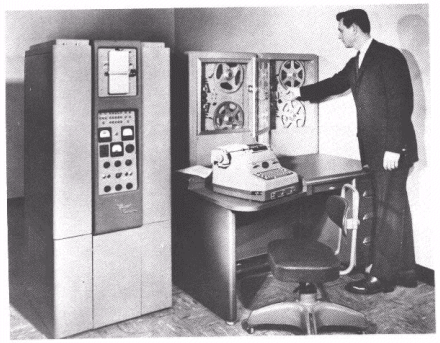
"Computers themselves, and software yet to be developed, will revolutionize the way we learn."

Steve Jobs

Computers: The Evolution

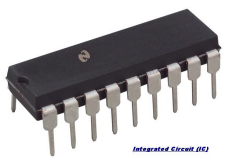


1G



2G

A paradigm shift in size, cost, speed, reliability, complexity, etc



3G, 4G

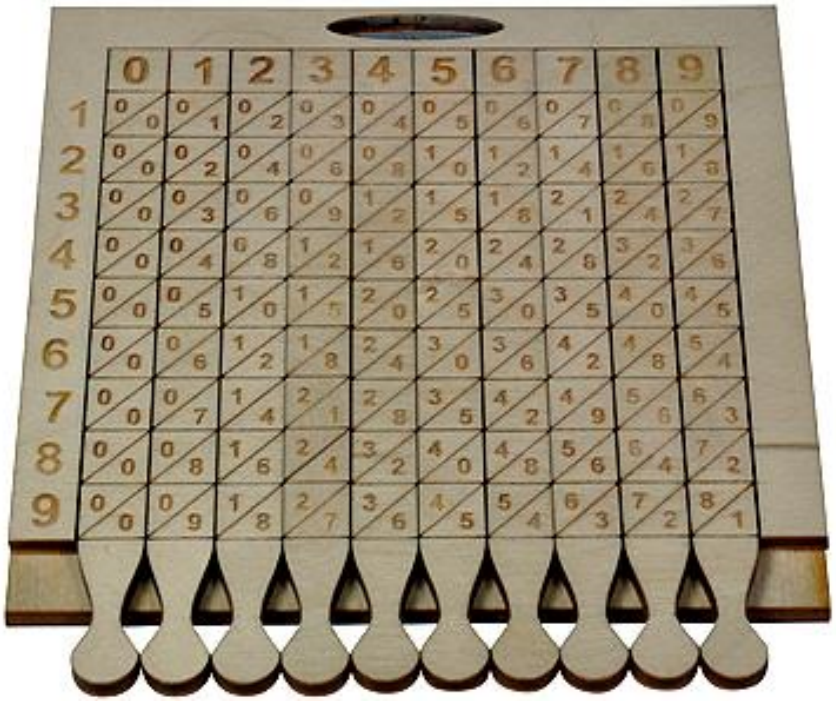
5G+

Computers: Quick History

Computers: Quick History

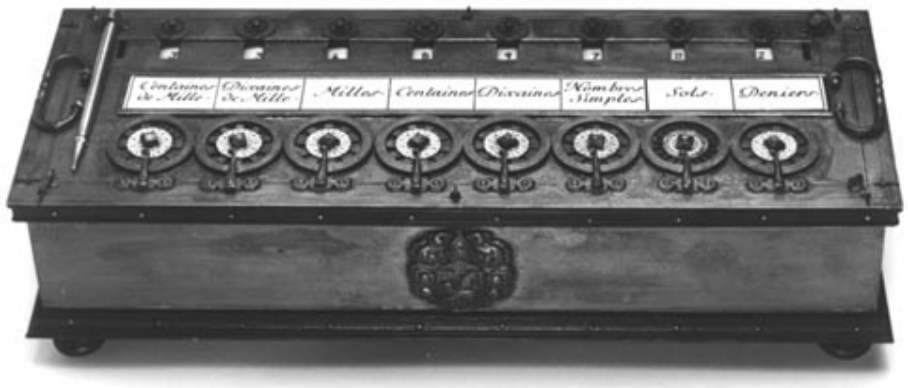
Abacus (a.k.a. counting frame)

- Around 5000 years ago.
- Precursor to calculators. Capable of doing counting, basic arithmetic.
- Sliding beads arranged on a rack of two parts: Upper *&* lower part. Upper - 2 beads per wire, lower - 5 beads per wire.
- Each bead in the upper rack has a value of 5 and 1 in the lower.



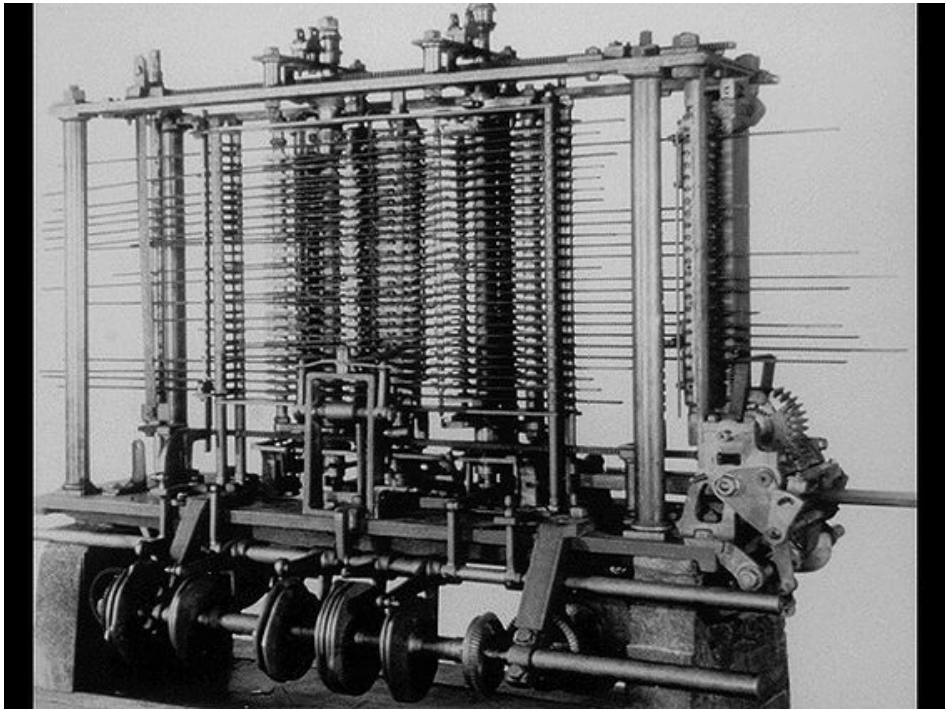
Napier's Bones (1614)

- By John Napier, a Scotish mathematician.
- A calculating device for products and quotients.
- Multiplication table engraved on the 10 rods.
- The rods made of ivory bones.
- Enabled multiplication and division by way of addition and subtraction.



Pascaline (1642)

- By Blaise Pascal, a French mathematician, philosopher, scientist. Invented to help his father, a tax supervisor.
- The first functional mechanical calculator (just + and -).
- The parts: wheels, gears, windows (for displaying numbers), dials (attached to wheels)
- Check out this video to see how it worked.



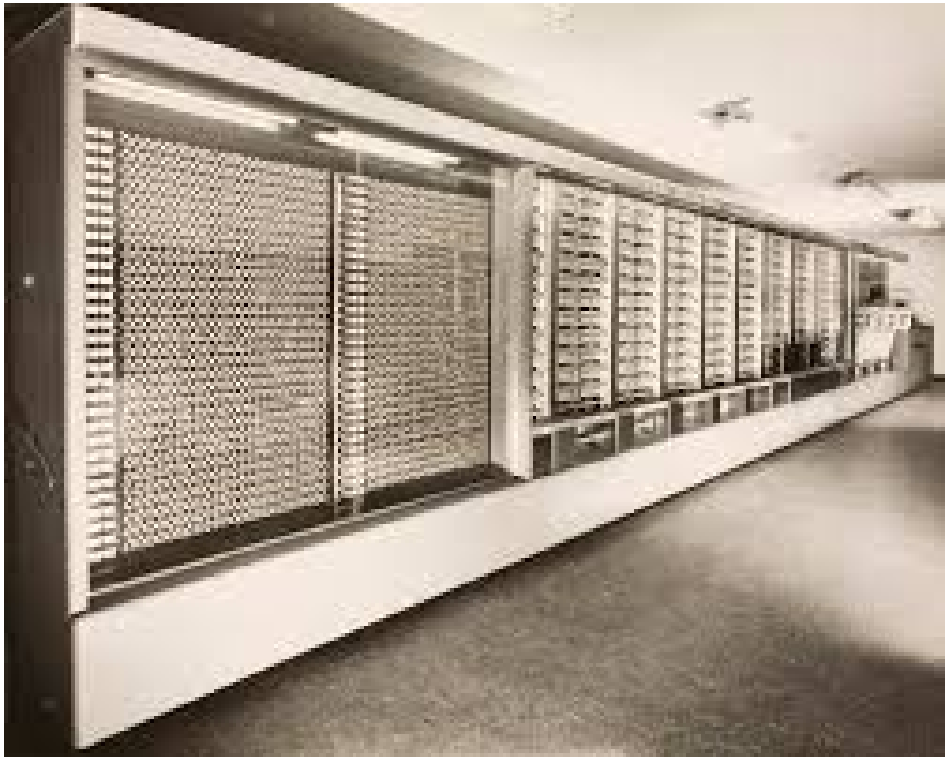
Difference Engine (1822)

- By Prof Charles Babbage, an English mathematician - **the father of computers**.
- A computing machine intended to solve differential equations.
- Quit working on it, moved to designing **Analytical Engine** (1833), the first programmable mechanical computer (had ALU, memory, control flow).

Ada Lovelace (1815-1852)

- An English mathematician, often deemed **the first programmer**, worked on Babbage's Analytical Engine.
- One of the pioneers of modern computers.
- Wrote *Notes* (=algorithm/program) for the Analytical Engine.
- The computer language **Ada** named after her.

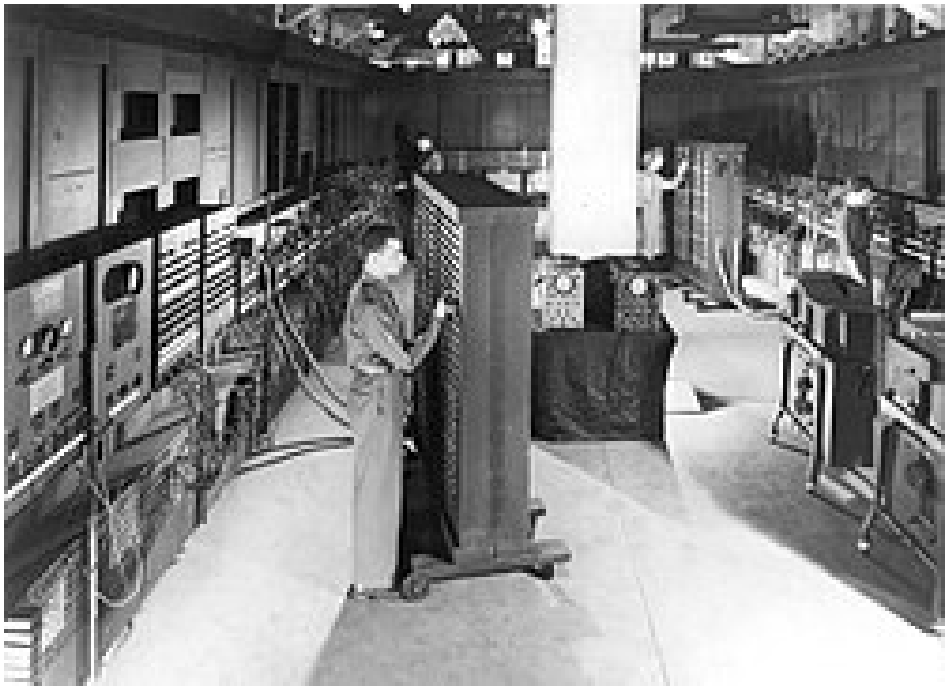
Fast Forward...



Mark-I Computer (1937-44)

- A.k.a. **A**utomatic **S**equence **C**ontrolled **C**alculator (ASCC)

- Developed by Howard Aiken of Harvard Univ., in collab with IBM.
- A general-purpose electromechanical computer (relays *&* electromagnetic parts replaced mechanical parts).
- Based on the principles of Babbage's Analytical Engine. Could perform +, -, x, / and more.
- Bulky and noisy (l=50 ft, h=8 ft, wt=5 tons)
- Successors: Mark II, Mark III and Mark IV.



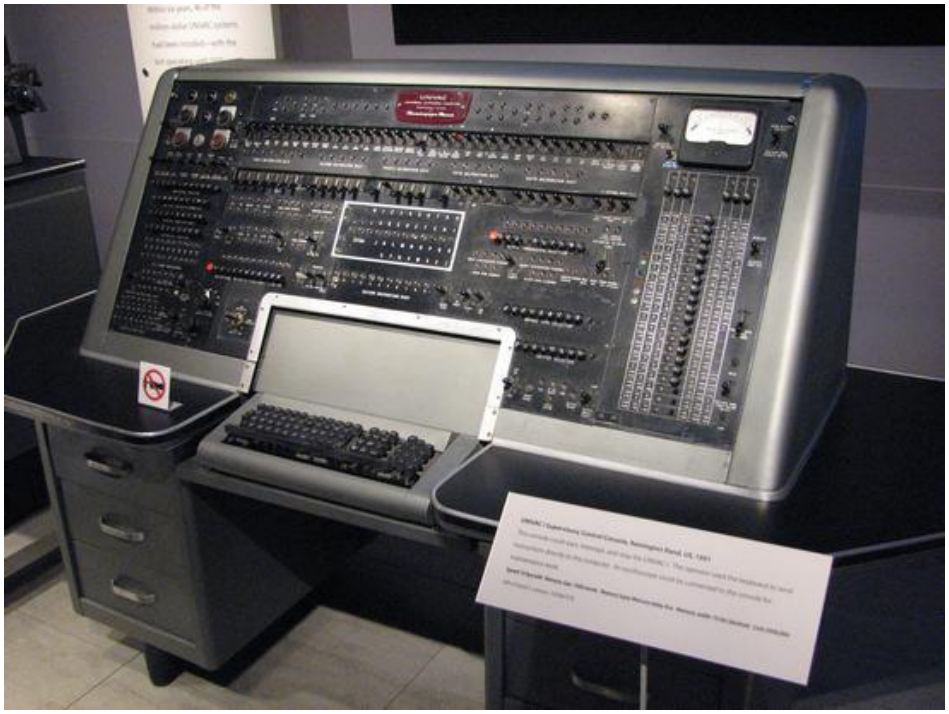
ENIAC (1943-46)

- = **E**lectronic **N**umerical **I**ntegrator **A**nd **C**omputer

- By a group of scientists such as Presper Eckert and John Maulchy - for US army.
- The first all-electronic programmable computer. Could solve many numerical problems.
- Its speed, 1000x faster than electro-mechanical computers.
- Features: 20,000 vacuum tubes, size=8x3x100 ft, wt=27+ tons, occupied 1800 ft^2, electricity consumed=150kW

EDVAC (1946-52)

- = **E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer
- By ENIAC's inventors + **John Von Neumann**.
- Unlike its predecessors, it was binary (ENIAC was decimal). capable of automatic +, -, x, / and auto checking.
- Used a revolutionary concept known as
  the stored program concept - i.e. program instructions stored in memory and executed.



UNIVAC (1951)

- = **UNIV**ersal **A**utomatic **C**omputer

- By Eckert-Mauchly Computer Corporation (EMCC) company (founded by Eckert and Mauchly of ENIAC).
- First intended for use by Bureau of the Census (US).
- The *first commercially available* electronic, stored-program computer.
- Marked the herald of the computer era.
- IBM and others soon joined the race. The rest is history.

Moore's Law

Moore's Law

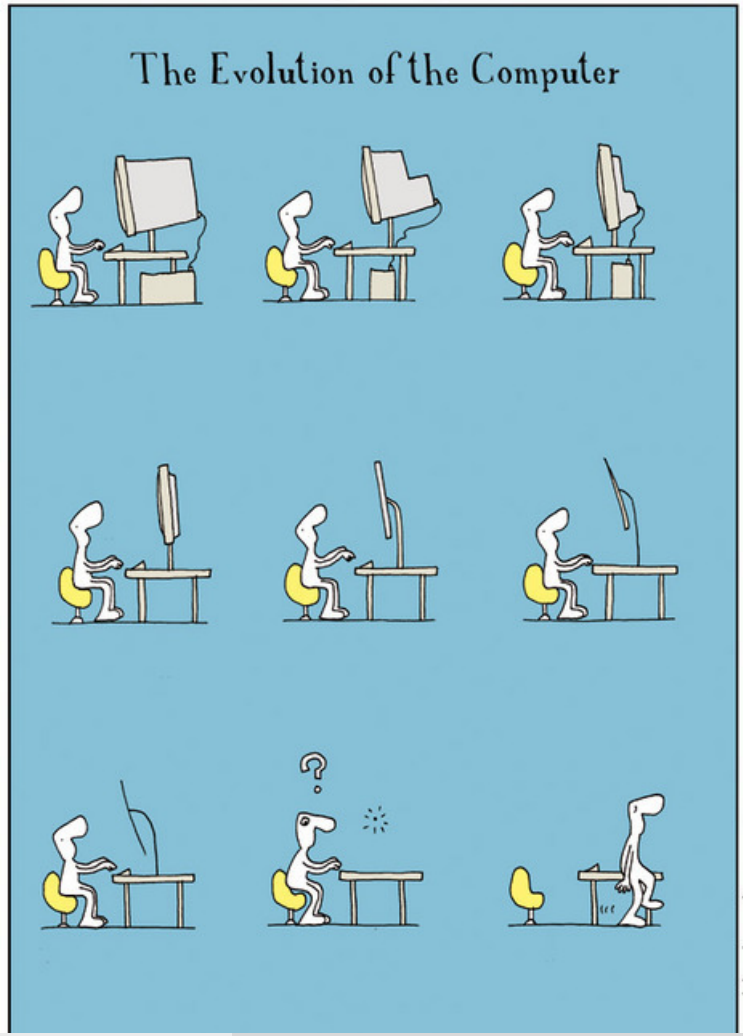
"The number of transistors incorporated in a chip will approximately double every 24 months."

Gordon Moore, Intel Co-Founder

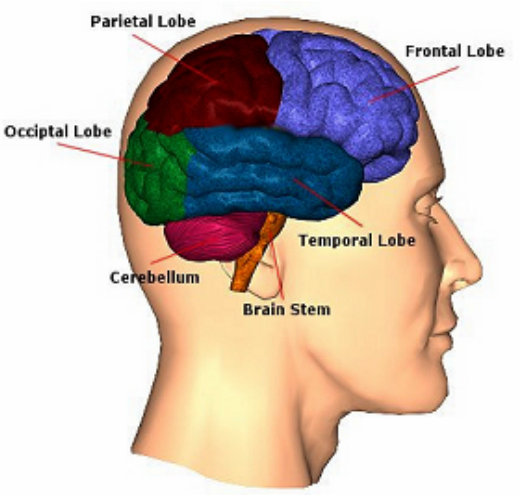
Moore's Law

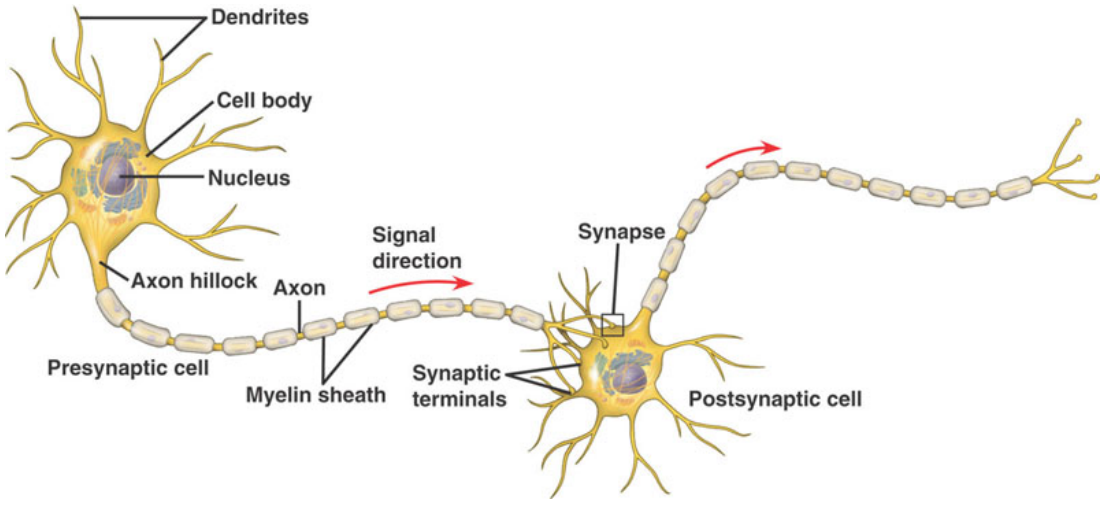
"The number of transistors incorporated in a chip will approximately double every 24 months."

Gordon Moore, Intel Co-Founder



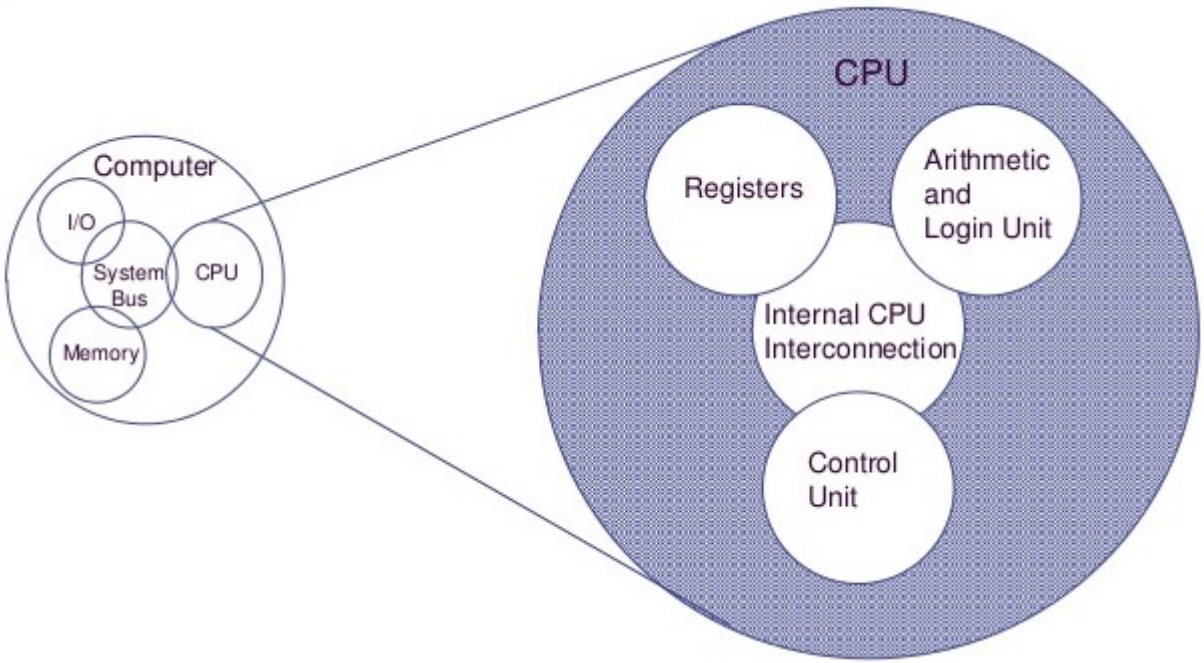
Human Computation





- Nerve cells/*neurons* are central in human computation.
- *Dendrites* receive signal from other neurons via *synapses*.
- Signal transmission is a *chemical* process resulting in high/low electrical potential inside the cell.

The Computer System: Hardware



1. *I/O Unit*
2. *Memory Unit*

   {primary, secondary}
3. *Processing Unit*

   {ALU, CU, registers}
4. *Interconnection Structures*

   {control, address & data buses}

Processing Unit/CPU

- Considered as **brain** of the computer system.
- Includes the Arithmetic Logic Unit(ALU) and Control Unit (CU).
- The ALU is the center of arithmetic and logical (comparison) operations. Data may move to and fro memory unit for the processing.
- The CU, heart of the processing unit, manages and coordinates the operations of all units. Reads instructions from memory, interprets and causes them to be executed.
- CPU = ALU + CU

Input Unit

- The 'sensory organ' part of the computer system.
- **Input unit** - reads data + instructions from outside world using input devices (examples?)
- Converts them to computer-acceptable form (using units known as Input Interfaces).
- Forwards the converted form for further processing by —?—.

Output Unit

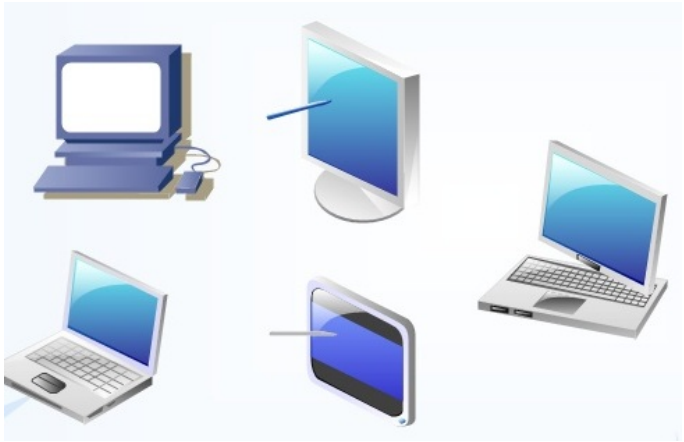
- Accepts post-processing results that are in machine-coded form.
- Converts them to user-understandable form (using units known as Output Interfaces)
- Supplies the new form to the user using output devices (examples?).

Memory Unit

Stores:

- Data + instructions needed for processing, possibly read from the —?— unit.
- Intermediate results of computation
- Final results of computation. These could then be sent to the —?— unit.

Computer Types: Size



Microcomputer (Micro)

Minicomputer (Mini)

Mainframe computer

Supercomputer

Computer Types: Mode of Operation

1. Analog computer
2. Digital computer
3. Hybrid computer

And the winner is...

- Organization
- Mode of processing
- Storage
- Speed
- Complexity
- Control mechanism
- Resilience

Software

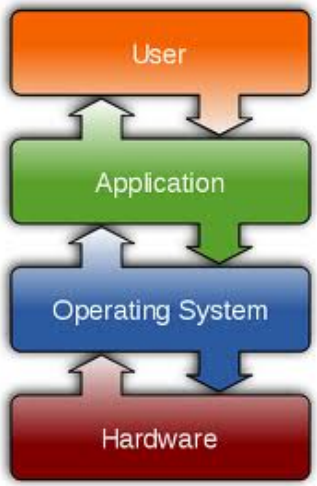
A set of programs, procedures, algorithms and its documentation.

- Written using *programming languages*
  - High-level languages e.g. `C`, `C++`, `Java`, `Python` ...
  - Assembly language - mnemonic-based e.g. `ADD`, `SUB`, `MOV` ...
  - Low-level language - native language of computer circuitry
- Each language has syntactic + semantic rules.
- Otherwise, syntax, logical and/or runtime errors.

Did you know? Some famous software bugs: Mark II's Moth (1945/7), Mariner I space probe (1962), Y2K bug (2000)

The Computer System: Software (2)

- Compiled vs Interpreted languages
- Application vs System software



Example

translators, `operating systems`, device drivers, etc

The Operating System

A set of software that *manages* computer hardware resources and *serves* other programs.

Examples

`Windows, Unix, BSD, Linux, OS X, iOS, Android, etc`

- Past → Expensive hardware, 'cheap' people!

  Goal: maximize hardware utilization.

- Now → Cheap hardware, expensive people!

  Goal: make it easy for people to use computer.

  Characteristics: Enormous and complex (millions of LOC)

- Future →

The Operating System: Types

- Proprietary vs Open-source
- Single-user vs Multi-user
- Single-tasking vs Multi-tasking
- Distributed
- Embedded
- Real-time operating systems (RTOS)
  - Multi-tasking OS for apps with fixed deadlines (industrial robots, embedded systems, engine controllers, etc).
  - e.g. `VxWorks`, `QNX`, `MontaVista Linux`

The Operating System: Functions

Generally, a coordination and control center.

- Abstraction layer (i.e. hides hardware details)
- Resource Manager
  - Processor manager
  - Memory manager
  - File manager
  - Device manager
- User interface
- Networking *&* Security (e.g. access control)

Computing Trends & Environments

1. Distributed Computing
   e.g. sensor networks, Internet
2. Mobile Computing
   e.g. ipod, ipad, iphone, etc
3. Ubiquitous/pervasive Computing/IoT
   e.g. smart homes, cars, buildings, etc
4. DNA/Biomolecular computing
5. Cloud Computing
6. Social Computing
   SNS and collaborations
7. Green Computing/ Green IT

   "Average Google search releases 7 gm of CO2 " (Google says 0.2 gm)

Next...

- Unit 1: Computational thinking and building blocks

Reading Task! (Take your own notes)

- Computational thinking
- Algorithms