QTM 350 - Data Science Computing

Lecture 02: Computational Literacy

Danilo Freire

danilo.freire@emory.edu

Department of Quantitative Theory and Methods
Emory University

04 September, 2024

Recap and lecture overview

Course information

danilo.freire@emory.edu

https://danilofreire.github.io/qtm350

https://github.com/danilofreire/qtm350

- Instructor: Danilo Freire
- Lectures: Mondays and Wednesdays, 2:30-3:45pm
- Office hours: At any time, just send me an email in advance
- Please remember to check the course repository regularly for updates and announcements 😉

Course information

Questions about the course organisation?

Software installation

- As we discussed in the first lecture, you will need to install the following software:
- A terminal
  - → You can install WSL if you are using Windows. Or you can use VS Code's built-in terminal
  - → Mac users already have a terminal. I suggest you install iTerm2, Homebrew and Oh My Zsh for a better experience
  - → Linux users are good to go
- Git
- GitHub
- Anaconda (or regular Python)
- Quarto
- PostgreSQL
- Docker
- VS Code (optional, but strongly recommended)

Tip

- Please check the installation tutorials for more information
- If you have any questions, please let me know!

Learning objectives

By the end of this lecture, you will be able to:

1. Learn how computers work from the ground up, starting with binary code
2. Get familiar with other key computer encodings like hexadecimal, ASCII, and Unicode
3. Learn about the pioneers of computing and the development of Assembly language
4. Understand the difference between low-level and high-level programming languages and when to use each

Let’s get started!  

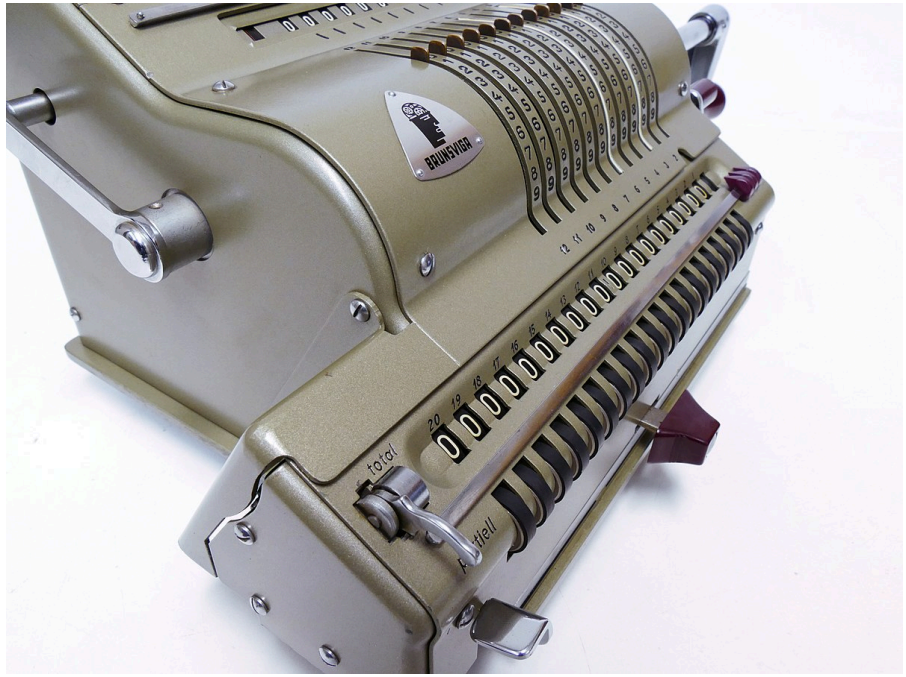
Brief history of computing

The first computers

- Historically, a computer was a person who makes calculations, especially with a calculating machine
- To do calculations we use numbers. How to represent them?
  - → Fingers
  - → Pebbles
  - → Strings (Inca Khipu)
  - → Abacus

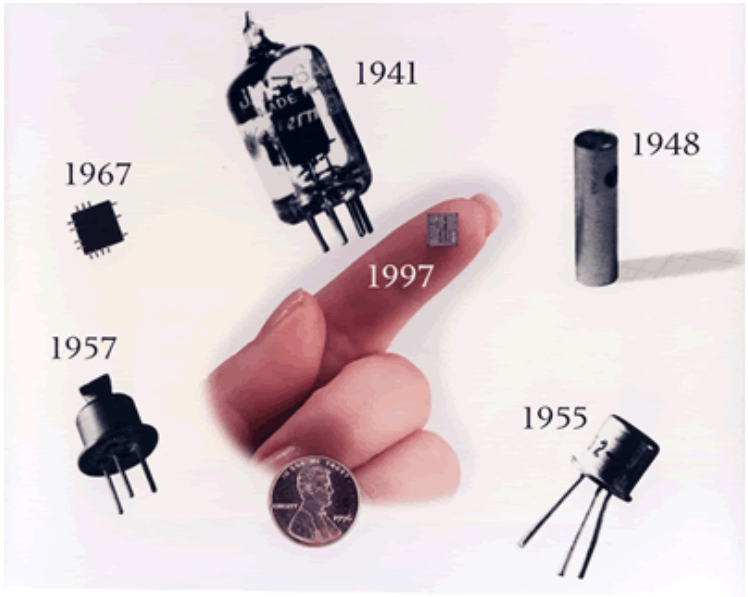
Video: How to use an abacus

Four-species mechanical calculators



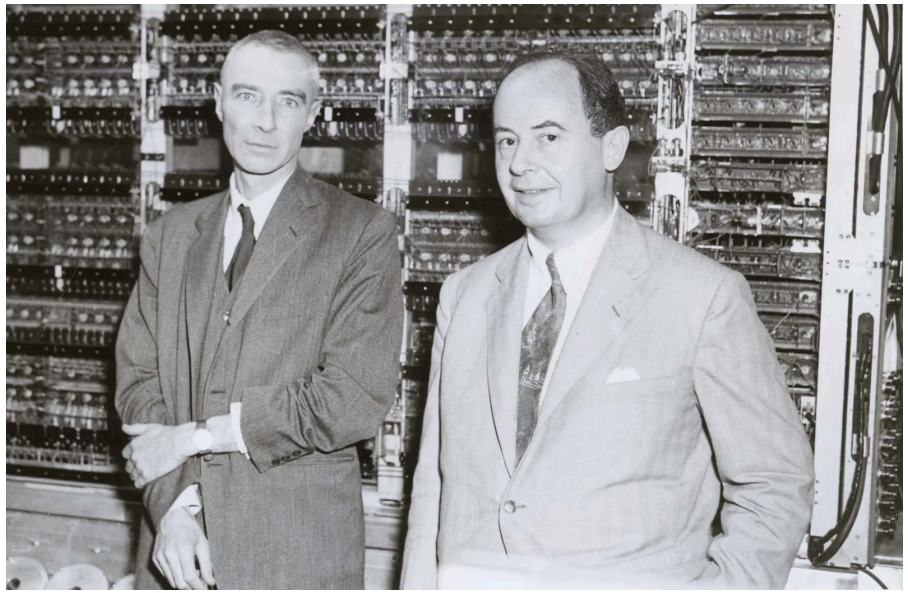
- The first mechanical calculator capable of performing all four basic arithmetic operations (addition, subtraction, multiplication, and division)
- Invented by Gottfried Wilhelm Leibniz in 1694
- If you took a statistics course before the late 1970s, you likely used this type of mechanical calculator for your computations

Silicon-based computers



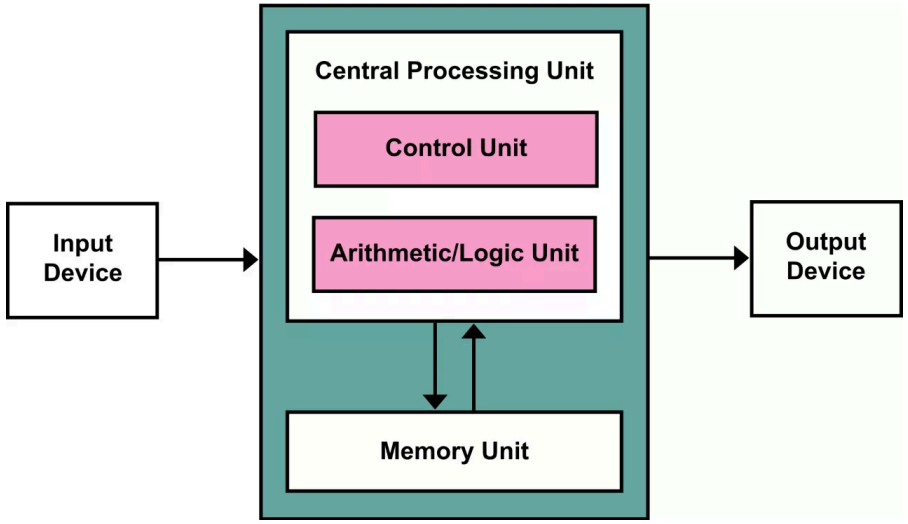
The 1970s marked the transition from mechanical to electronic:

Von Neumann Architecture



- The Von Neumann Architecture stores both program instructions and data together in a slow storage medium, such as a hard disk, and transfers them to faster RAM for execution by the CPU
- This is the basis for almost all modern computers
- When proposed in 1945, this architecture was revolutionary, as programs were previously seen as part of the machine, distinct from the data they operated on

Von Neumann Architecture



- Advantages:
  - → Efficient memory use, with less need for separate areas
  - → Flexibility in data storage and manipulation
  - → Simplicity in design and operation
- Disadvantages:
  - → Von Neumann bottleneck: Limits computing performance due to sequential processing of instructions and data through a single bus
  - → The CPU often waits for data due to its faster processing speed compared to memory transfer rates
  - → Harvard architecture is an alternative that separates data and instruction memory

Data representation

Computers run on 0s and 1s

- Computers represent everything by using 0s and 1s
- Transistors act as switches, with 1 for high voltage level and 0 for low voltage level
- Computers use binary because transistors are easy to fabricate in silicon and can be densely packed on a chip

Converting coins to dollars

- We can convert between number systems by translating a value from one system to the other
- For example, the coins on the left represent the same value as $0.87
- Using pictures is clunky. Let's make a new representation system for coins

Converting coins to dollars

- To represent coins, we will make a number with four digits
- The first represents quarters, the second dimes, the third nickels, and the fourth pennies
  - → c3102 =
  - → 3 x $0.25 + 1 x $0.10 + 0 x $0.05 + 2 x $0.01 =
  - → $0.87

	Q	D	N	P
c	3	1	0	2

Converting dollars to coins

Quick questions!

- Think-Pair-Share: do the following conversions
- What is c1112 in dollars?
- What is $0.61 in coin representation?

Solutions:

Number systems – binary

Quick question!

- Think-Pair-Share: what is the binary representation of the decimal number 3?

Your turn!

Practice Exercise 01:

1. What binary number represents 5?
2. What binary number represents 7?
3. What binary number represents 9?
4. What binary number represents 11?

Convert binary to decimal

To convert a binary number to decimal, just add each power of 2 that is represented by a 1.

- For example, 00011000 = 16 + 8 = 24

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	0

- Another example: 10010001 = 128 + 16 + 1 = 145

128	64	32	16	8	4	2	1
1	0	0	1	0	0	0	1

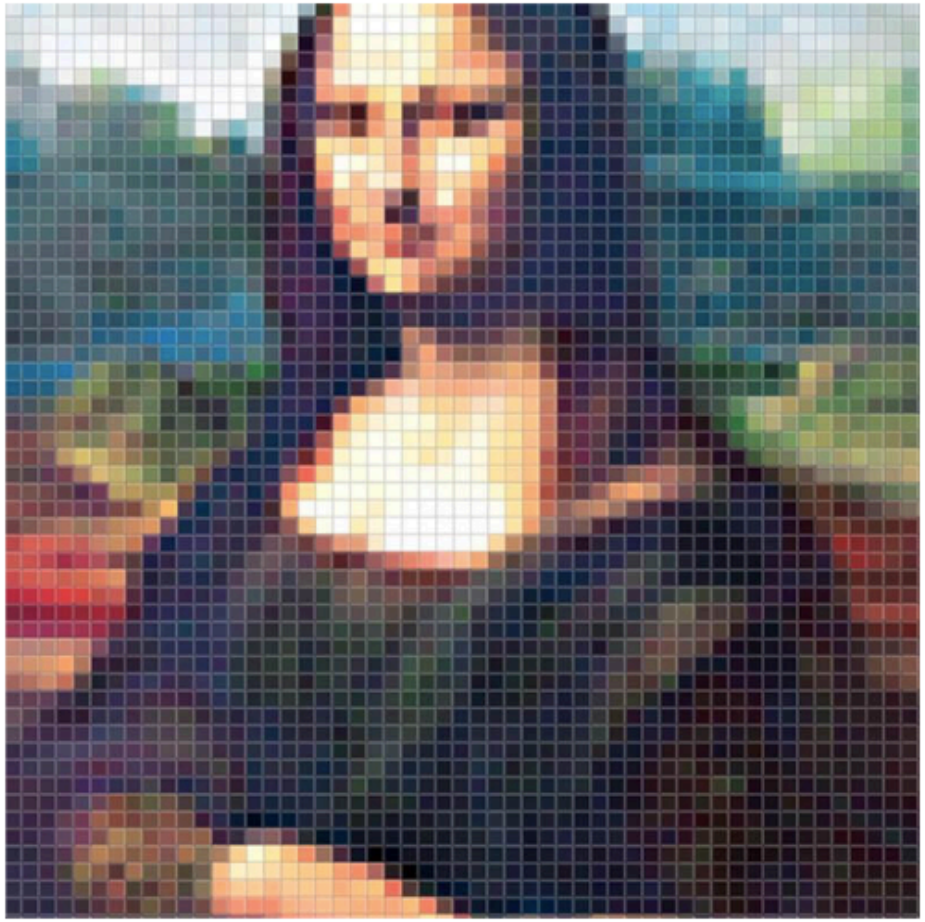
So far, so good? 😀

Binary and abstraction

Binary and abstraction

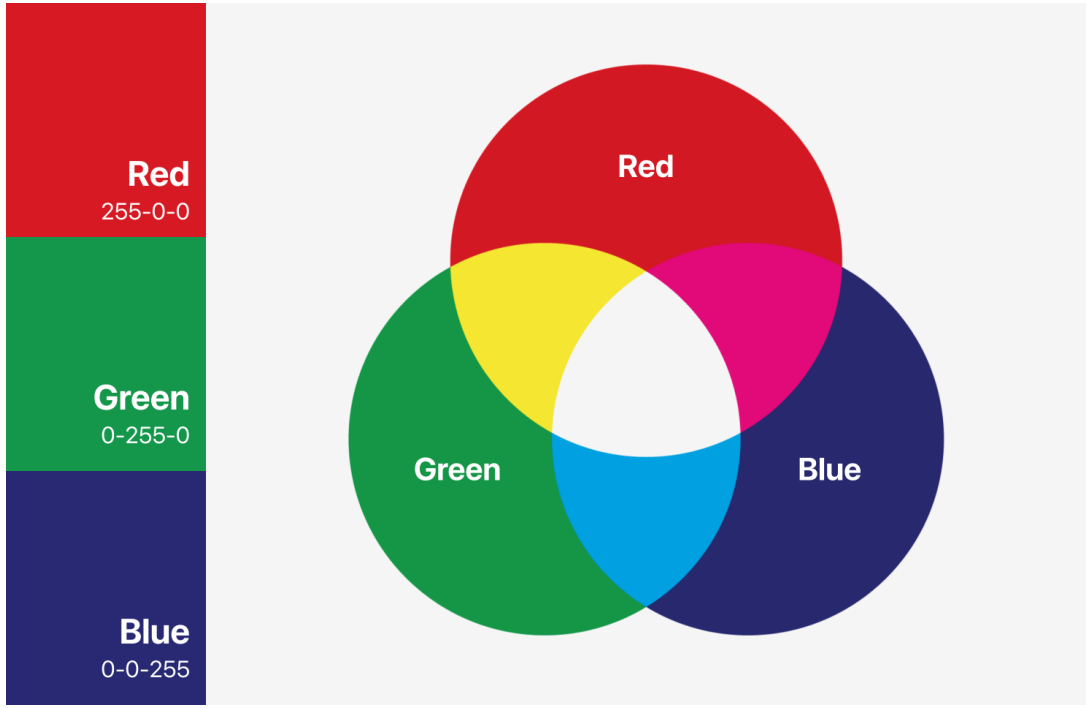
- Now that we can represent numbers using binary, we can represent everything computers store using binary
- We just need to use abstraction to interpret bits or numbers in particular ways
- Let us consider colours, images, and text

Images as collections of colours



Images as collections of colours

RGB colour model



Number systems – Hexadecimal

What is hexadecimal?

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Binary to hex conversion

- Convert binary to hex by grouping into blocks of four bits.
- Example: Binary `1001 1110 0000 1010` converts to Hex `9E0A`.

Hexadecimal and HTML

Hex and RGB

Represent text as individual characters

Characters and glyphs

Represent text as individual characters

Lookup tables

ASCII is nothing but a simple lookup table

Yes, really!

For basic characters, we can use the encoding system called ASCII. This maps the numbers 0 to 255 to characters. Therefore, one character is represented by one byte

Check it out here: ASCII Table

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciichars.com

ASCII is nothing but a simple lookup table

Translation

“Hello World” =

```
01001000 01100101
01101100 01101100
01101111 00100000
01010111 01101111
01110010 01101100
01100100
```

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciichars.com

Your turn!

Practice Exercise 03

- Translate the following binary into ASCII text:

```
01011001 01100001
01111001
```

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciichars.com

ASCII Limitations

- ASCII only includes unaccented characters.
- Languages requiring accented characters cannot be represented.
- Even English needs characters like ‘é’ for words such as ‘café’.

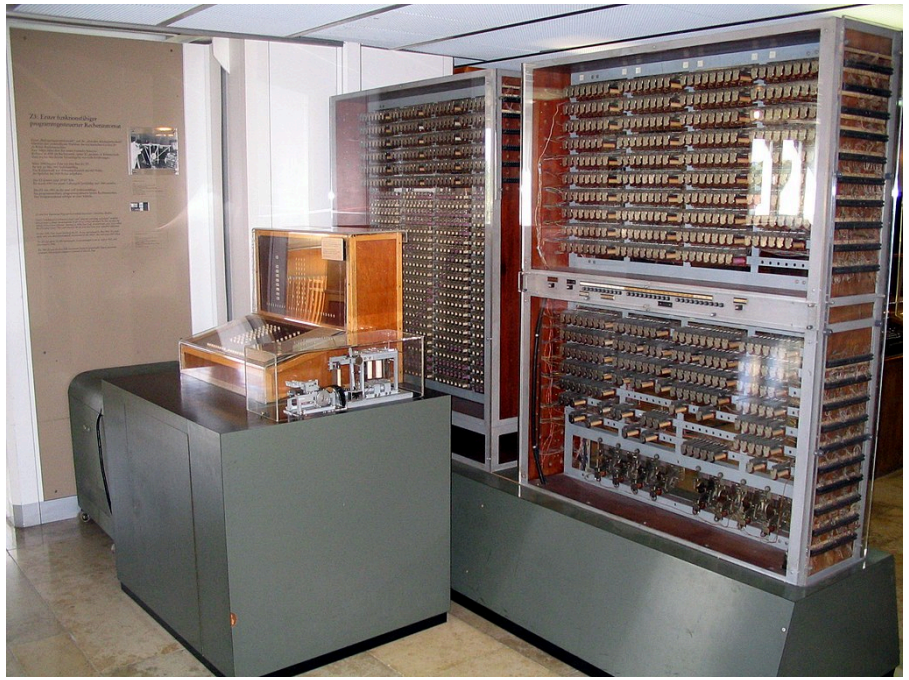
- To address this, Unicode was developed
- Unicode is a superset of ASCII that includes characters from all languages, as well as symbols and emojis
- The Unicode system represents every character that can be typed into a computer. It uses up to 5 bytes, which can represent up to 1 trillion characters!
- UTF-8 stands for Transformation Format 8-bit
- Find all the Unicode characters here: https://symbl.cc/en/unicode-table/
  - → “Danilo” in Unicode: `\u0044\u0061\u006e\u0069\u006c\u006f`
  - → “QTM 350” in Unicode: `\u0051\u0054\u004d\u0020\u0033\u0035\u0030`
- Decoder: https://symbl.cc/en/tools/decoder/

Questions? 🤔

Programming languages and the genesis of programming 🌟 🔡 🐍

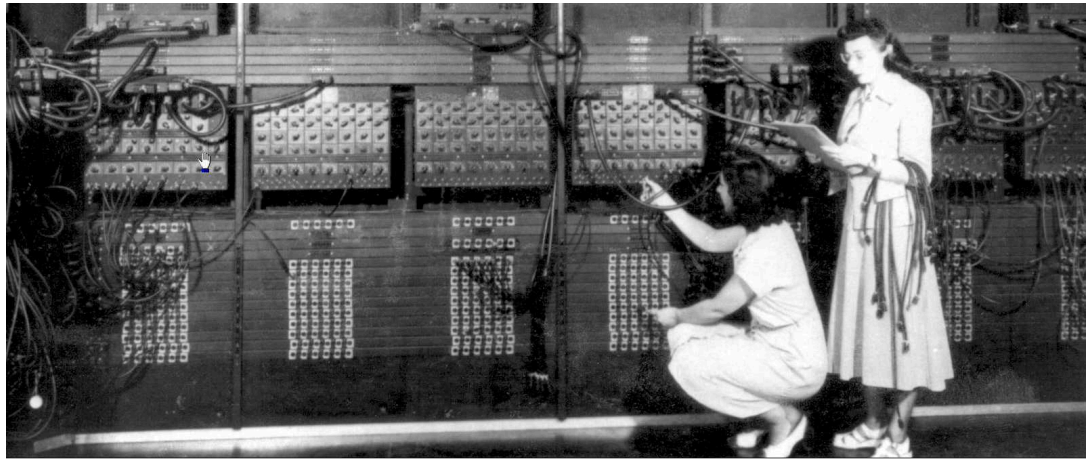
The genesis of programming

Zuse's computers



- Konrad Zuse was a German engineer and computer pioneer
- He created the first programmable computer, the Z3, in 1941
- The Z3 was the first computer to use binary arithmetic and read binary instructions from punch tape
- Example: Z4 had 512 bytes of memory
- Zuse also created the first high-level programming language, Plankalkül

What is Assembly language?



- Assembly language is a low-level programming language that allows writing machine code in human-readable text
- Each instruction corresponds to a single machine code instruction
- The first assemblers were human!
- Programmers wrote assembly code, which secretaries transcribed to binary for machine processing

Some curious facts about Assembly!

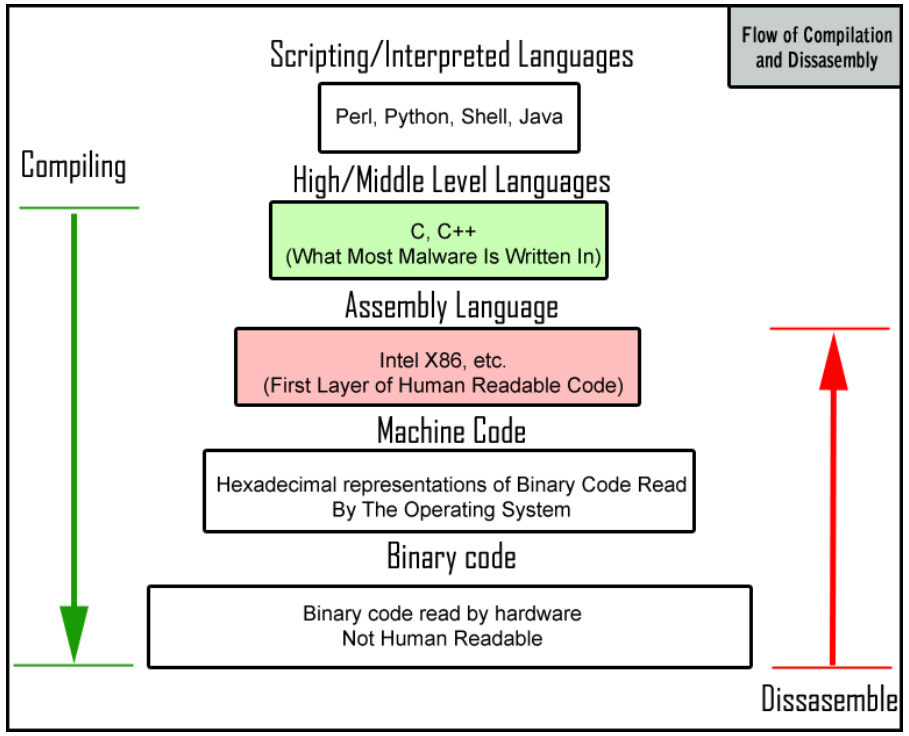
Margaret Hamilton and the Apollo 11 code

- The Apollo 11 mission to the moon was programmed in assembly language
- The code is available here: https://github.com/chrislgarry/Apollo-11 (good luck reading it! 😅)
- One of the files is the `BURN_BABY_BURN--MASTER_IGNITION_ROUTINE.agc` 🔥🚀
- But if Assembly is so fast and efficient, why don't we use it all the time?

Low-level vs high-level languages

- Low-level languages are closer to machine code and are harder to read and write
- High-level languages abstract from hardware details and are portable across different systems

- **Compiled Languages:** Convert code to binary instructions before execution (e.g., `C++`, `Fortran`, `Go`).
- **Interpreted Languages:** Run inside a program that interprets and executes commands immediately (e.g., `R`, `Python`).



Low-level vs high-level languages

Code that is worth a thousand words

- “Hello, World!” in machine code:

```
R> 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21
```

- “Hello, World!” in Assembly (x86 Assembly for Linux)

```
R> section .data
+     message db 'Hello, World!', 10    ; 10 is the ASCII code
+
+ section .text
+     global _start
+
+ _start:
+     mov eax, 4          ; system call number for write
+     mov ebx, 1          ; file descriptor 1 is stdout
+     mov ecx, message    ; address of string to output
+     mov edx, 14         ; number of bytes
+     int 0x80            ; call kernel
+
+     mov eax, 1          ; system call number for exit
+     xor ebx, ebx        ; exit status 0
+     int 0x80            ; call kernel
```

- “Hello, World!” in Python:

```
R> print("Hello, World!")
```

Summary 

Summary

- **Computational Literacy**: Binary and hexadecimal numbers, characters (ASCII, Unicode), and distinction between high vs low-level programming languages
- **Early Computing**: Konrad Zuse's pioneering work with programmable digital computers and the use of binary arithmetic
- **Assembly Language**: The initial approach to programming using human-readable instructions for machine code
- **Calculators**: The evolution from Leibniz's four-species calculating machine to modern electronic computing
- **Silicon Microchip Computers**: The 1970s revolution with transistors, integrated circuits, and the emergence of Von Neumann architecture
- **Modern Programming Languages**: From low-level assembly languages to high-level languages like `Python`; distinction between compiled and interpreted languages

Next class

Thank you very much and see you next class! 😊 🙏

Solution - Practice Exercise 01

1. What binary number represents 5?

- In binary, the number 5 is represented as 101, which equates to $(1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$.

2. What binary number represents 7?

- In binary, the number 7 is represented as 111, which equates to $(1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$.

3. What binary number represents 9?

- In binary, the number 9 is represented as 1001, which equates to $(1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$.

4. What binary number represents 11?

- In binary, the number 11 is represented as 1011, which equates to $(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$.

Solution - Practice Exercise 02

1. Decimal 13 is `1101` in binary.
   - Break it down: $13 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$.
2. Binary 1101 is `D` in hexadecimal.

- Group the binary into blocks of four: `1101`.
- Convert each block to hex: `1101` (binary) = `D` (hex).
- Let's take a closer look at how to convert the binary number `1101` to hexadecimal:
- Start with the binary number: `1101`
- Convert it to decimal by summing the powers of 2:
  - → $1 \times 2^3 = 8$
  - → $1 \times 2^2 = 4$
  - → $0 \times 2^1 = 0$
  - → $1 \times 2^0 = 1$
- Add the decimal values: $8 + 4 + 0 + 1 = 13$
- The decimal number `13` corresponds to the hexadecimal number `D`.
- Therefore, binary `1101` is `D` in hexadecimal.

Solution - Practice Exercise 03

- Step 1: Identify the binary strings: `01011001 01100001 01111001`
- Step 2: Convert each binary string to its decimal equivalent
  - → 01011001 = 89
  - → 01100001 = 97
  - → 01111001 = 121
- Step 3: Map each decimal value to its corresponding ASCII character
  - → 89 = Y
  - → 97 = a
  - → 121 = y
- Step 4: Combine the ASCII characters to form the final text
  - → Result: Yay