SEE1002
Introduction to Computing for Energy and Environment

Part 1: Introduction to Computing

Outline

1. What is computing?
2. Basic terms

Purpose of this lecture

Explain some basic terms and concepts that any programmer or computer-literate person should know.

Course Outline

Part 1: Introduction to computing

Part 2: Elements of Python programming

Section 1: Fundamentals

Section 2: Elementary data structures

Section 3: Branching or decision making

Section 4: Loops

Section 5: Functions

Part 3: Basic Python programming

Section 1: Modules

Section 2: Structure of a Python program

Section 3: Good programming practices

Part 4: Python for science and engineering

Section 1: File input and output

Section 2: Other topics

1. What is computing?

What do computers do?

- At heart computers only do two things:

  1. Perform calculations or operations

  2. Remember results of these calculations

- This doesn't sound very interesting or impressive…but computers can do these things amazingly well!

What are typical operations?

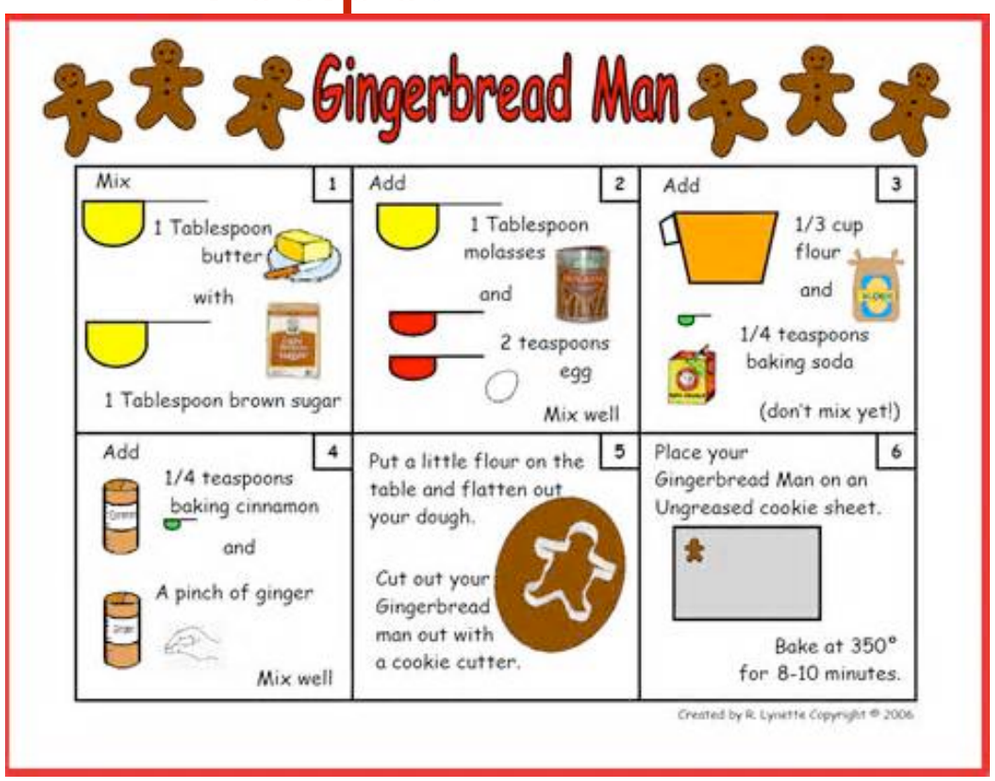
- The preceding list is overly simplified.
- The list of possible operations is longer but it's still surprisingly compact:

  1. Do arithmetical operations (e.g. addition and subtraction)
  2. Do operations on text (e.g. capitalization)
  3. Compare data
  4. Input data
  5. Output data

- These basic operations usually involve small pieces of data (e.g. a word or a pair of numbers).
- *We'll see examples of these operations in the next section*!

What is computing?

- Computing refers to the tasks that can be performed by computers.
- More precisely it refers to goal-oriented tasks that involve a list of instructions or algorithms.
- An algorithm can be thought of as a general recipe.



What is a computer program?

- A program refers to the collection of instructions that can be used to carry out a set of tasks.
- It is more general than an algorithm. Real programs contain algorithms and data.
- With appropriate data, a program can be used to carry out different tasks.
  - *Example:* baking instructions for 1, 6 or 12 cookies.
- One can think of a program as a collection of general recipes and ingredient amounts.

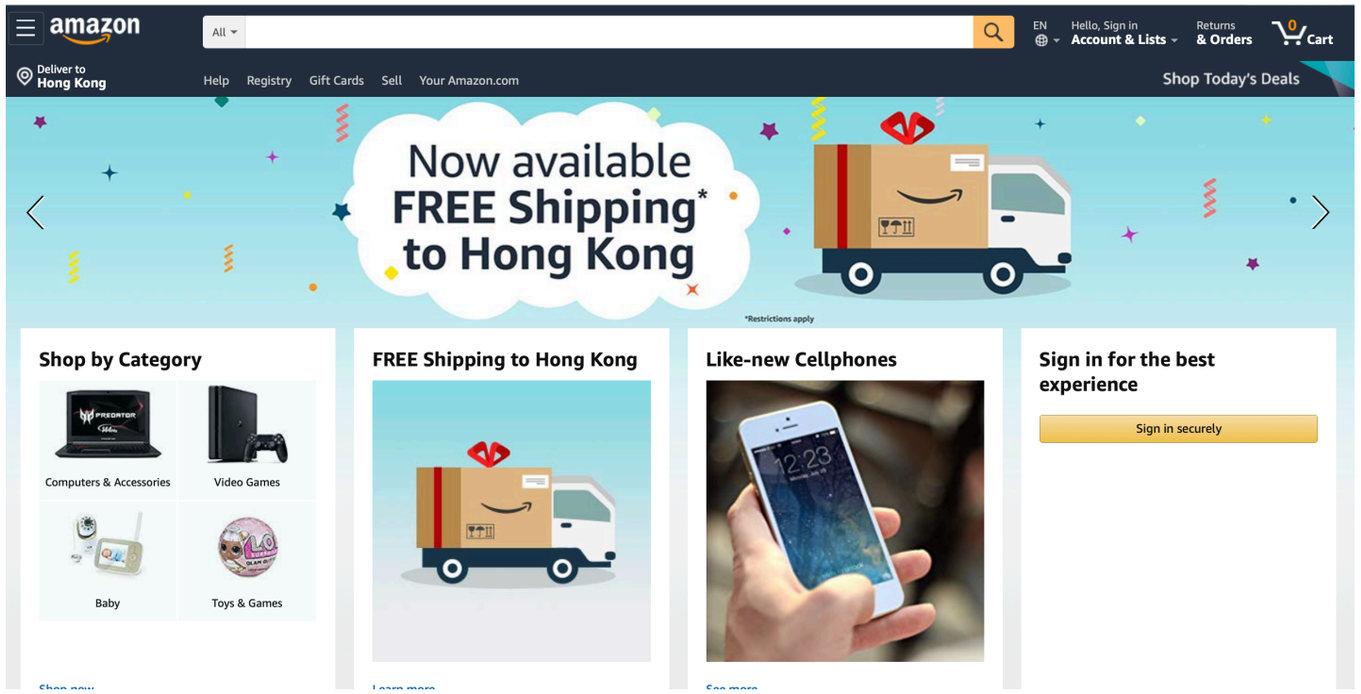
Lesson for the future

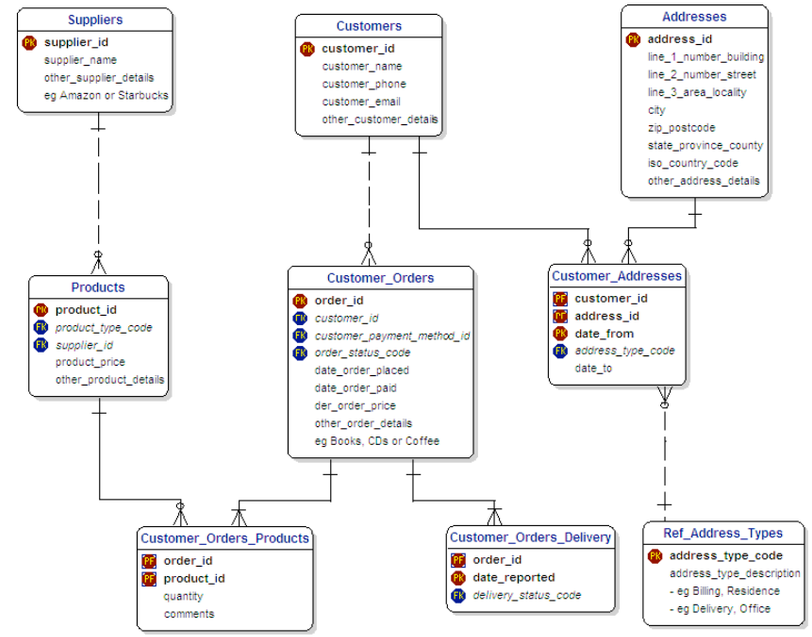
- Computers are incredibly literal-minded. Assuming the computer isn't malfunctioning, it does no more or no less than what we tell it to do.
  - *Advantage*. We don't need to worry about the computer developing an independent streak or disobeying our instructions.
  - *Disadvantage*. We need to be very clear in our instructions.
- Classic saying:

> GIGO = "Garbage in, garbage out"

Examples of computing

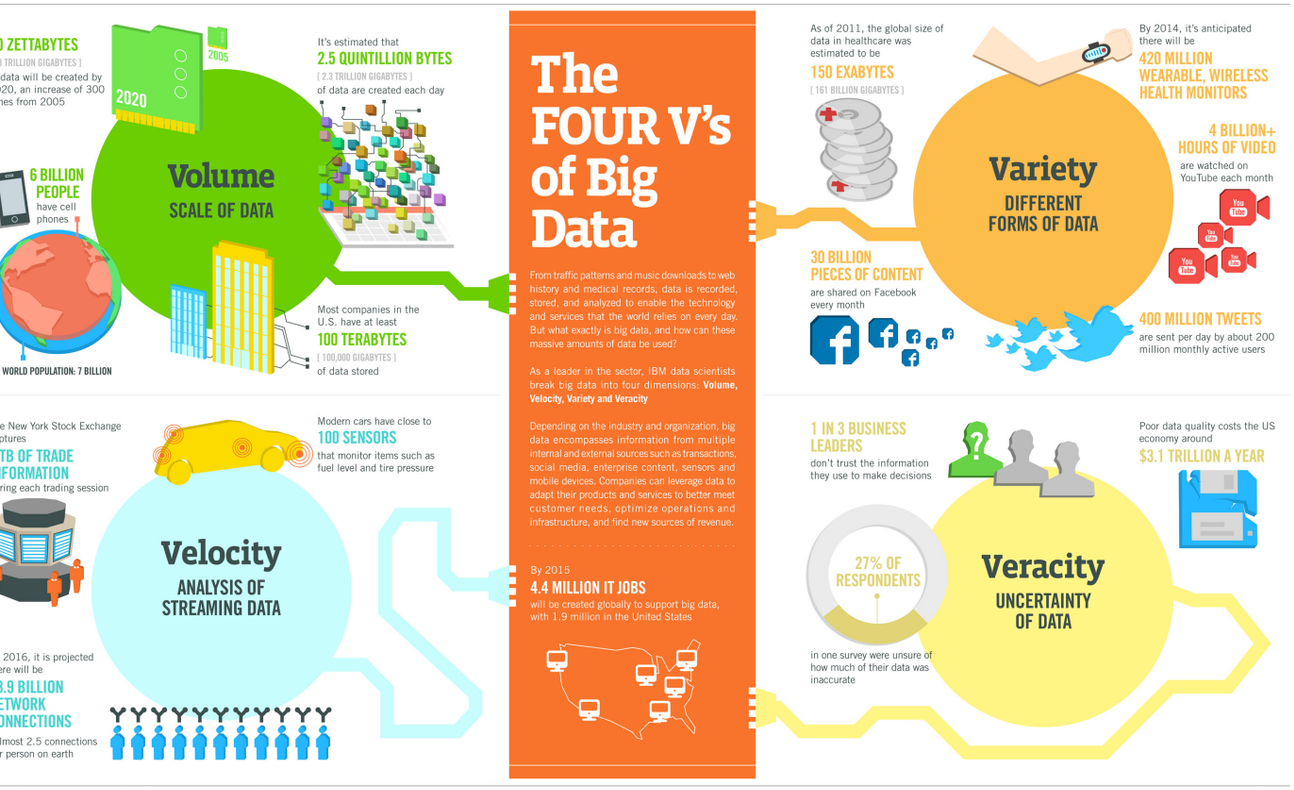
1) Storing and classifying data

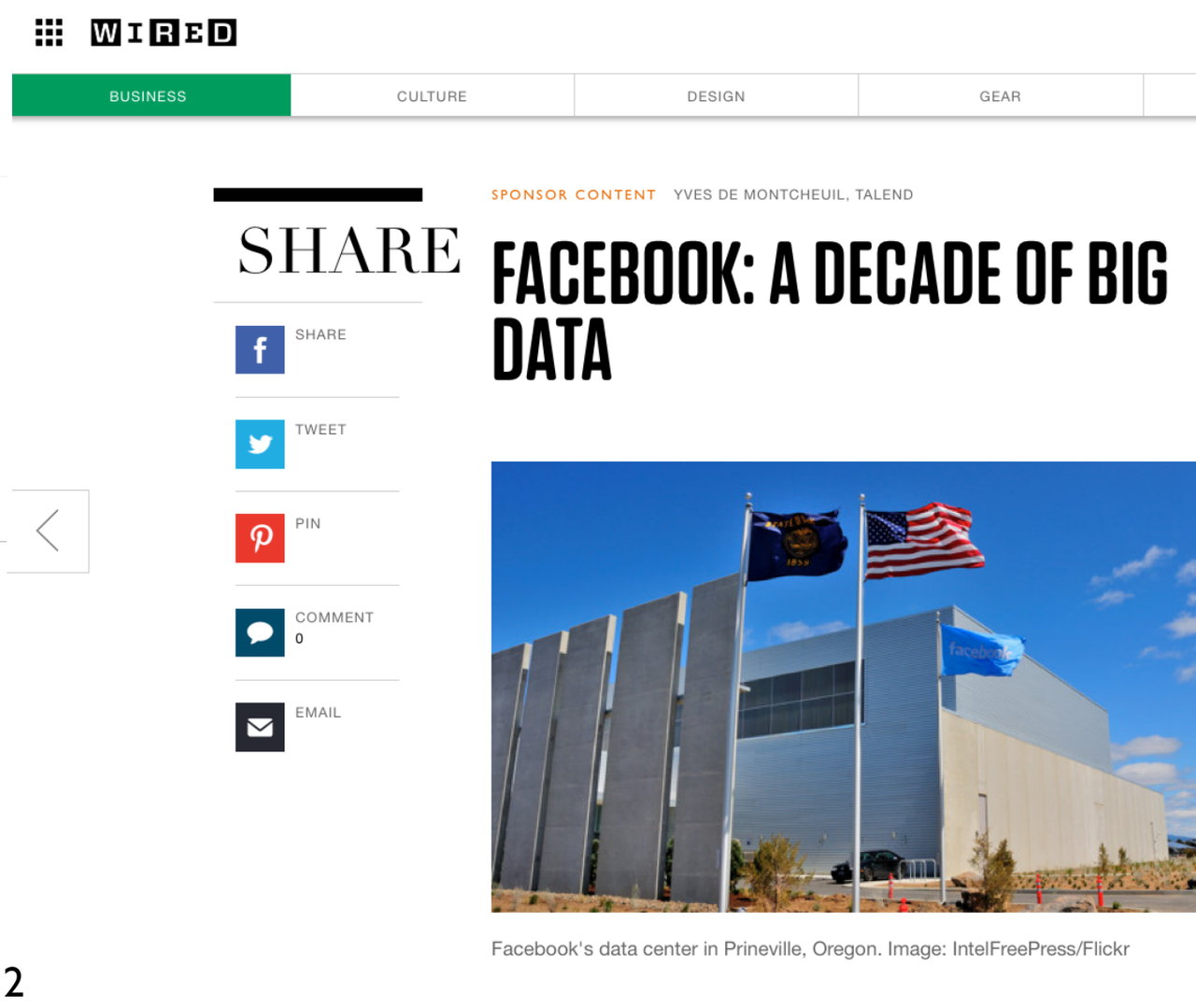




Examples of computing

2) Data analysis

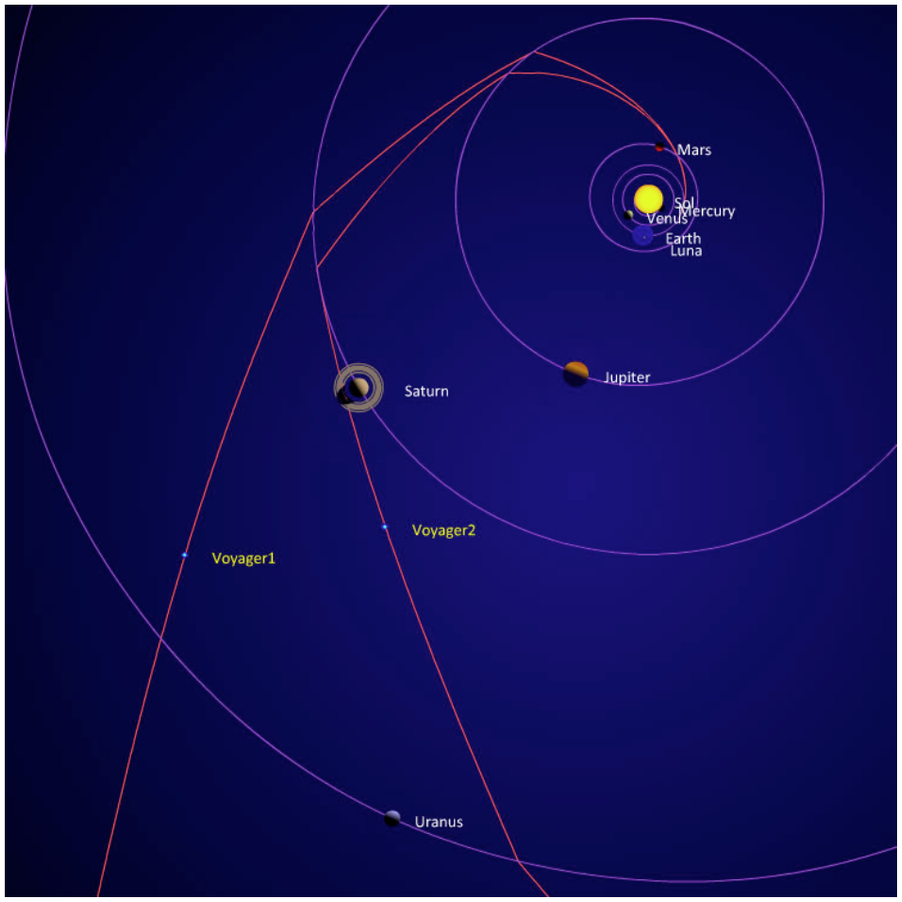


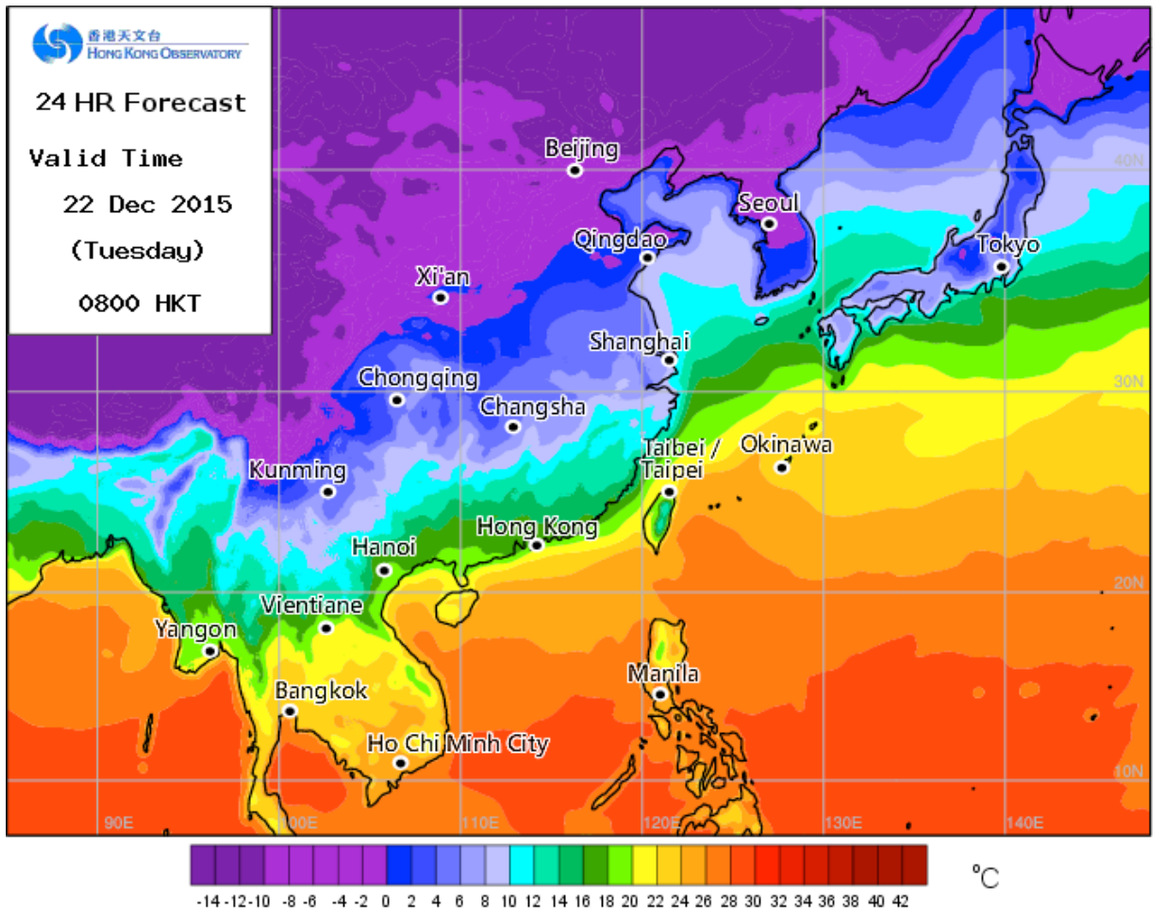


Facebook's data center in Prineville, Oregon. Image: IntelFreePress/Flickr

Examples of computing

3) Solving equations and simulating a system





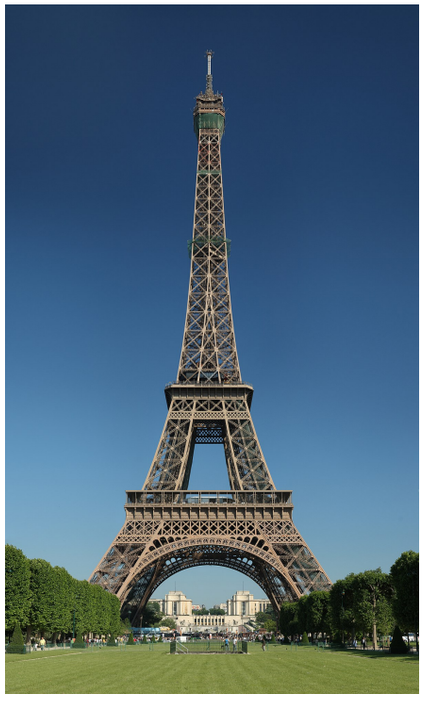
In many areas of science, computer programs can be used as a 'virtual laboratory'.

Examples of computing

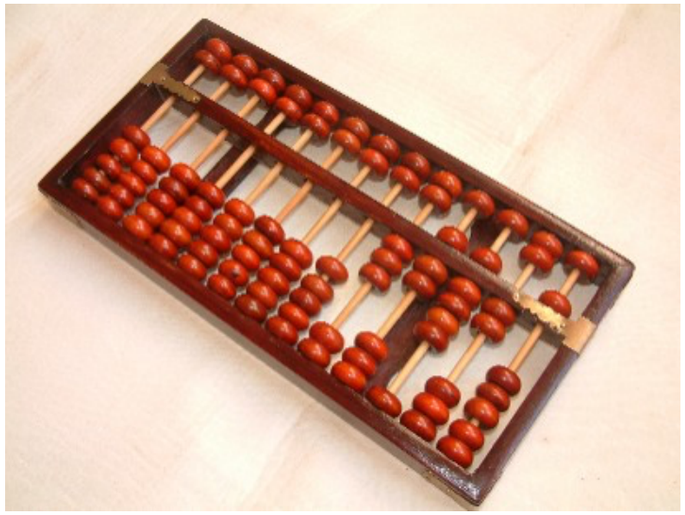
4) Visualization and design

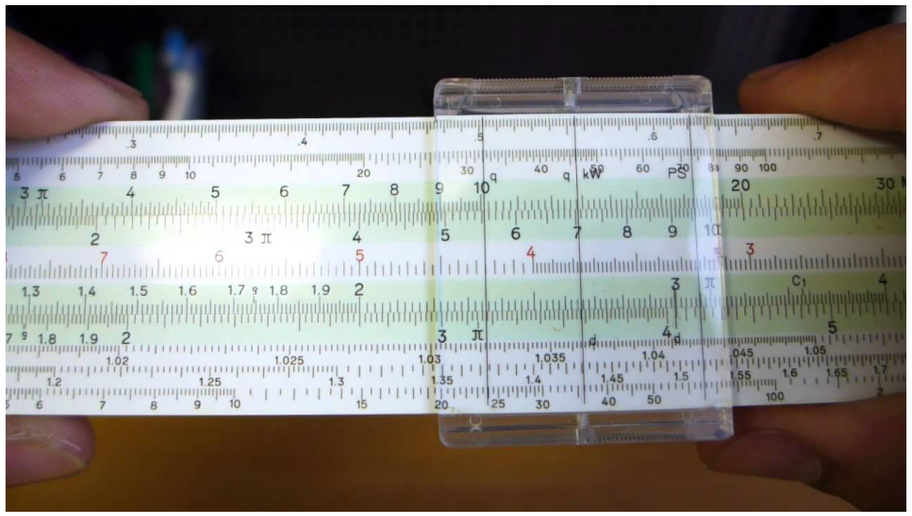
Relevance to engineering

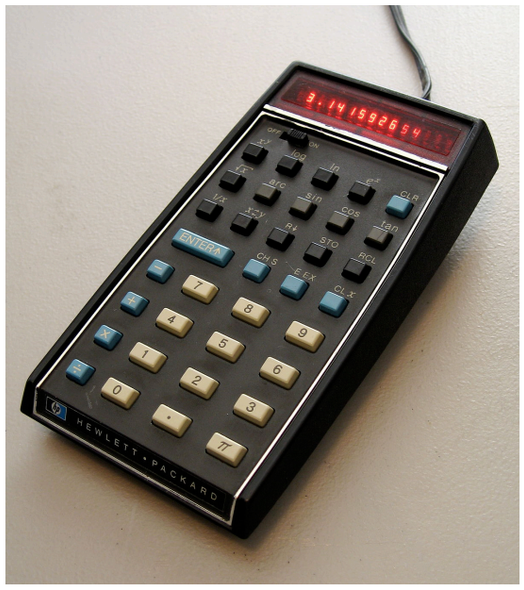
- There are very few real-world engineering problems that can be solved entirely using pencil and paper.
- *Example*: Designing a building. To a large extent, structural engineering is nothing more than an application of Newton's Second Law ($F=Ma = \sum F_i=0$). In the past this was done using a calculator or a slide rule, which was slow and tedious. The use of computers has revolutionized structural engineering.

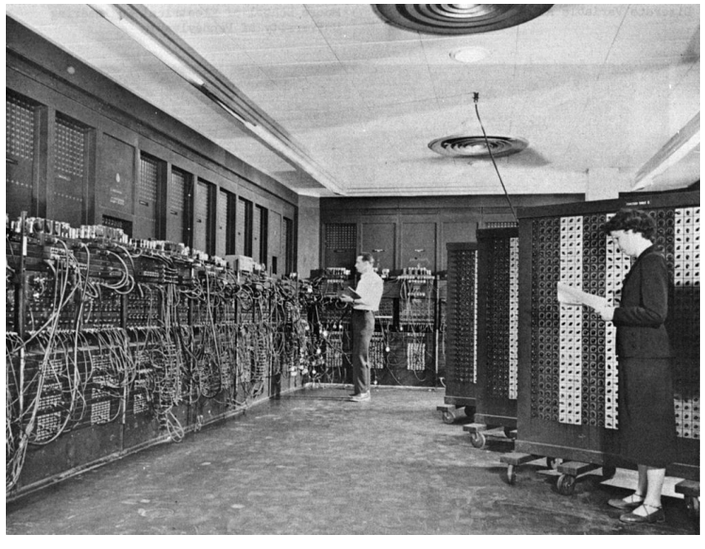


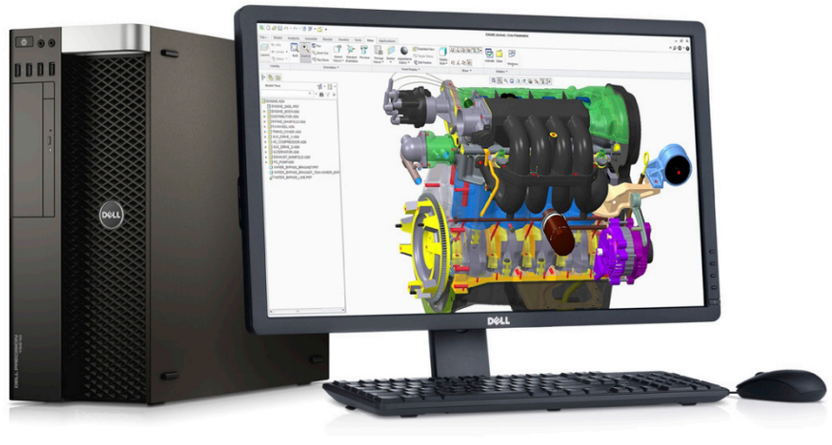
Evolution of engineering computation











2. Basic terms

Overview

This is course is concerned mostly with programming in Python. Nonetheless it's useful for you to be familiar with some basic terms.

- We'll do this by considering hardware and software separately.

A. Hardware

- Hardware refers to the physical components of a computer.
- In programming courses and books, one usually takes the computer as given. In this course we're not going to worry about how a computer works. This is a topic for computer engineering.
- Even so, a programmer should know a little about what goes on inside a computer. Hence we'll briefly cover how data are represented or stored.

1. Representation of data

Digital devices like computers store information as binary numbers, i.e. a sequence of 0s and 1s. Binary is equivalent to base 2.

Base 2 number	Base 10 number
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10

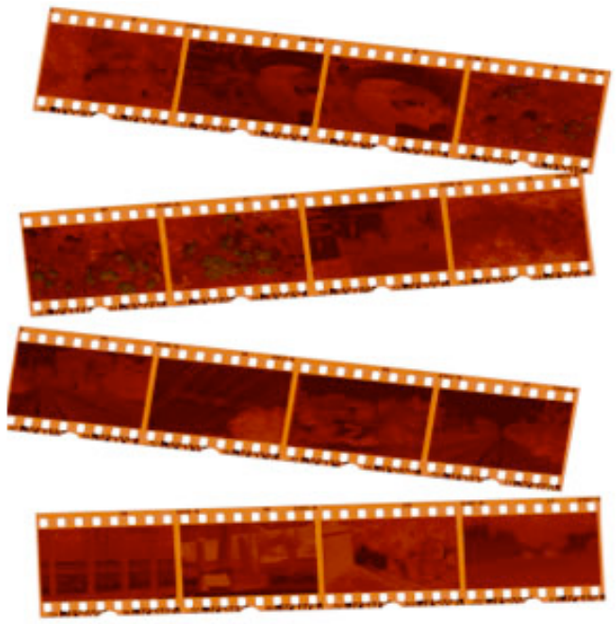
- A bit refers to a single binary digit, e.g. 0 or 1.
- Arbitrarily large numbers can be formed by combining bits. 2 is a 2-bit number (i.e. 10).

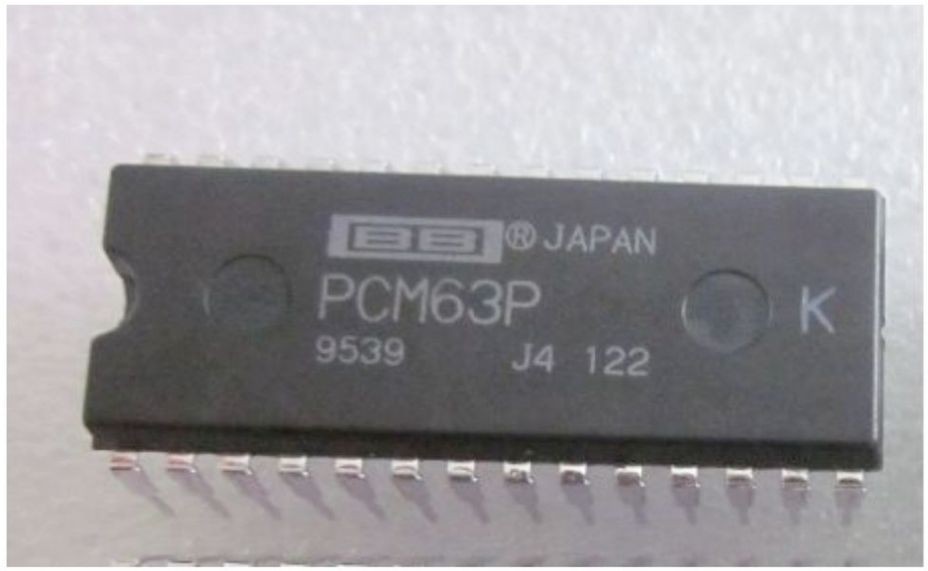
Digital versus analogue

Nowadays almost everything is digital. But previously analogue formats were more common.

In analogue formats, a direct representation of the original signal is stored without recourse to binary numbers.

For digital data to be useful to humans, some kind of digital-to-analogue converter is required.





Storage

As most of you know, some common abbreviations are used to quantify data:

- byte = B = 8 bits *(2^3 bits)*
- *kilo*byte = KB = "1 thousand = 10^3 bytes" *or* **1024 bytes *(2^{10} bytes)***
- *mega*byte =MB = "1 million = 10^6 bytes" *or* **1048576 bytes *(2^{20} bytes)***
- *giga*byte = GB = "1 billion = 10^9 bytes" *or* ***2^{30} bytes***
- *tera*byte = TB = "1 trillion = 10^{12} bytes" *or* ***2^{40} bytes***

Complication: there are two definitions!

More precise abbreviations

There's often some ambiguity as to whether storage (RAM, hard disks) is reported with respect to *powers of 10* or *powers of 2*.

Example from Wikipedia:

- A manufacturer advertises its product as a *500 GB (=500x10^9 bytes)* hard disk. However, the OS reports it as 465 GB (=465x1024^3 *bytes*)

To avoid ambiguity, one sometimes uses KiB, GiB, TiB to emphasize that the numbers are expressed with respect to powers of 2.

Representation of other characters

Binary data can be used to represent numbers or characters.

One way of doing this is through encodings or translation tables

- ASCII (128 characters) - basic set
- UTF-8 *or* unicode (>1M characters) - expanded set (standard for most webpages)

Computer files that are directly readable by humans are usually stored in ASCII or text format. But many files are for computers only (e.g. MP3 or JPG). They are stored in binary format.

Your Python programs will be stored in text format.

An ASCII table

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

Note: hexadecimal is another way of representing numbers. It is base 16 (i.e. 0-9, A-F)

2. Processing

- Computers perform calculations or process data.
- Most of these calculations are performed by central processing unit or CPU.
- This isn't completely true because calculations are also performed by specialized chips, e.g. the graphical processing unit or GPU.

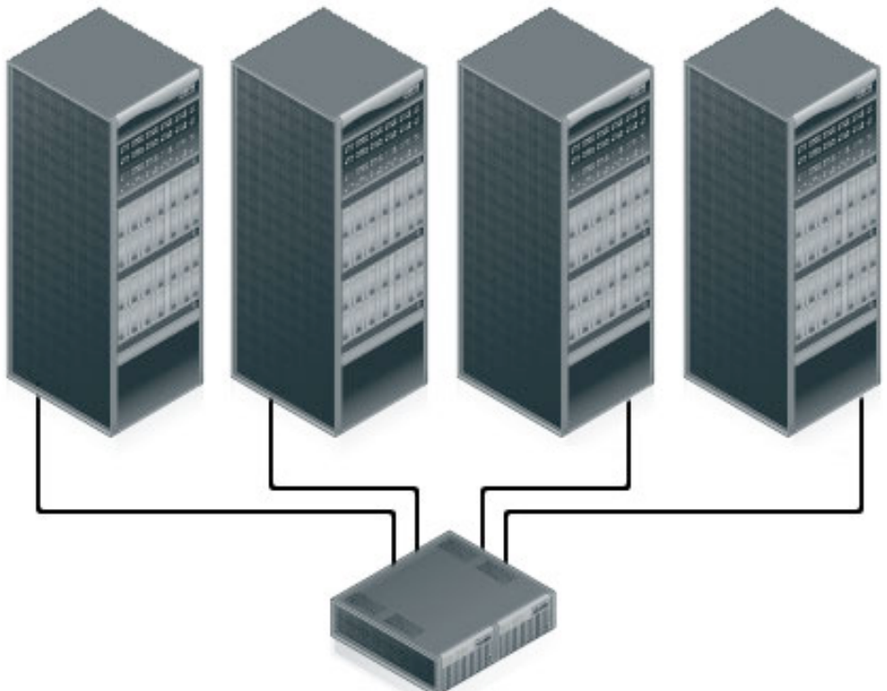
MOS TECHNOLOGY, INC.

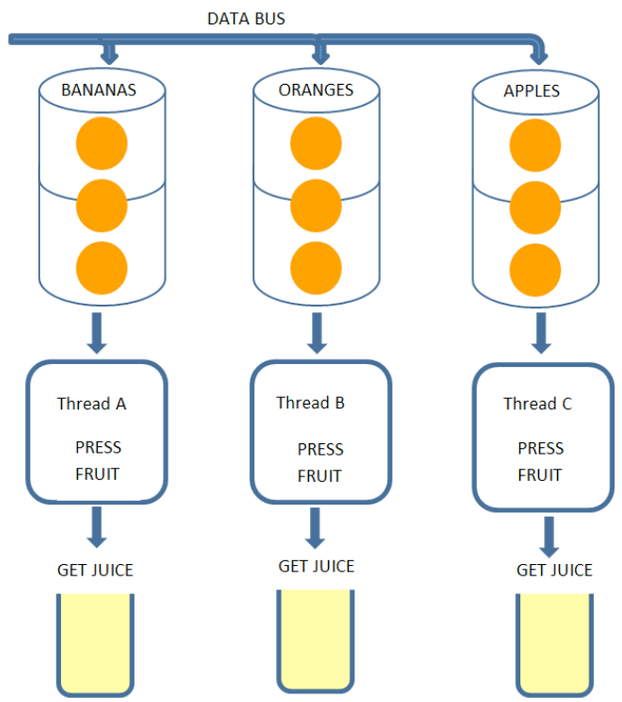
Serial computing

- The simplest programs are those that perform one task at a time, i.e., serially.
- The operation of such programs can be visualized as a pipeline or queue.
- This is the only kind of program that will be covered in this course.

Parallel computing

- One can go quite far with serial computing. But beyond a point one reaches the limits of a CPU or a computer.
- In parallel computing one runs a program on multiple CPUs or computers.
- Nowadays challenging computational problems in science and engineering are almost always done on some kind of parallel computer, e.g., a cluster or supercomputer.



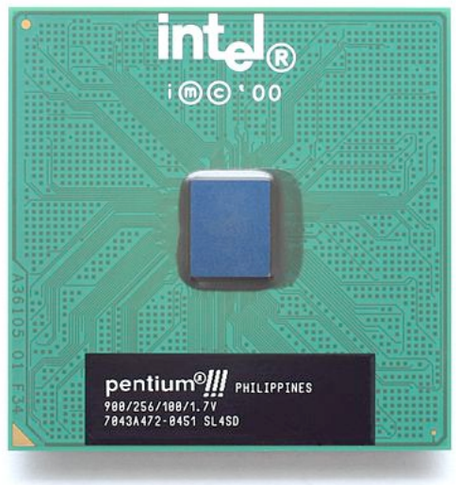


Multicore CPU

- Nowadays most CPUs have multiple cores. A core is the part of the CPU that carries out computations.
  - For computers: consumer CPUs typically have ~2-4 cores.
  - For phones: high-end models now have ~10 cores.
  - For GPUs: high-end video cards will have ~5000 cores.
- Using multiple cores is an example of parallel computing. This is much cheaper than buying additional CPUs or computers.
- Generally special programming is required to take advantage of additional cores. *We will not cover this!*

What is a 64-bit CPU?

- Roughly speaking, a 64-bit CPU accesses data in 64-bit chunks or words.
- This means that it can address up to 2^{64} bytes of memory (which is a huge amount!).
- By contrast a 32-bit CPU is usually limited to 2^{32} bytes (which is about 4 GiB).
- Older CPUs were limited to 16 or 8 bits.



B. Software

Since this is a programming course, our main interest is in software not hardware.

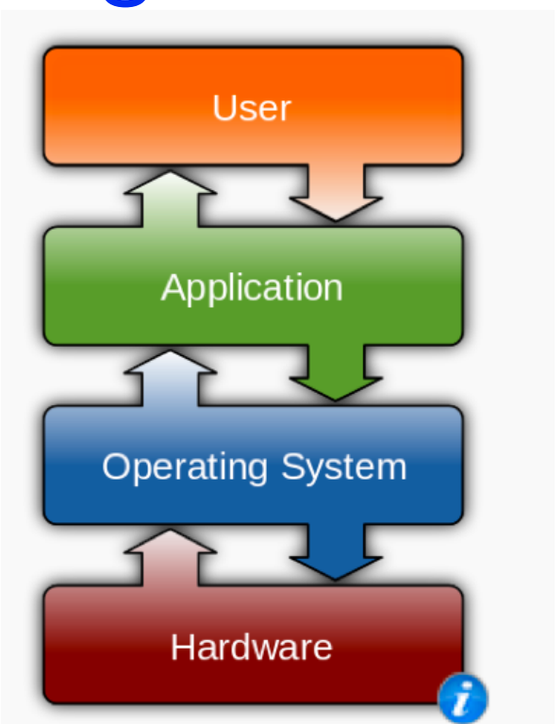
Software refers to the instructions or commands executed by a computer.

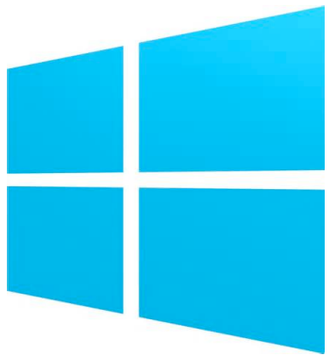
We don't need to know much about software in general, but it's helpful to review a few basic terms.

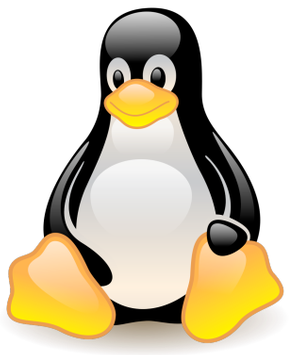
i. Operating system

Everybody can name an operating system (e.g. MacOS or Windows). But what is an operating system?

- Interface to hardware
- Provides a framework within which programs can run.







How do operating systems differ?

- From the perspective of a professional computer programmer, they present different libraries (*tools for specific tasks*) or application program interfaces or APIs (*interfaces to the libraries*).
- For most users, the choice is primarily one of personal taste. Everything we'll cover in this course is operating-system independent.

ii. Programming languages

- Applications are written in a programming language.
- Simply put, a programming language allows us to specify the tasks to be performed by a computer in a way that's understandable by humans, i.e., we don't need to write our program using just 0's and 1's.

Syntax

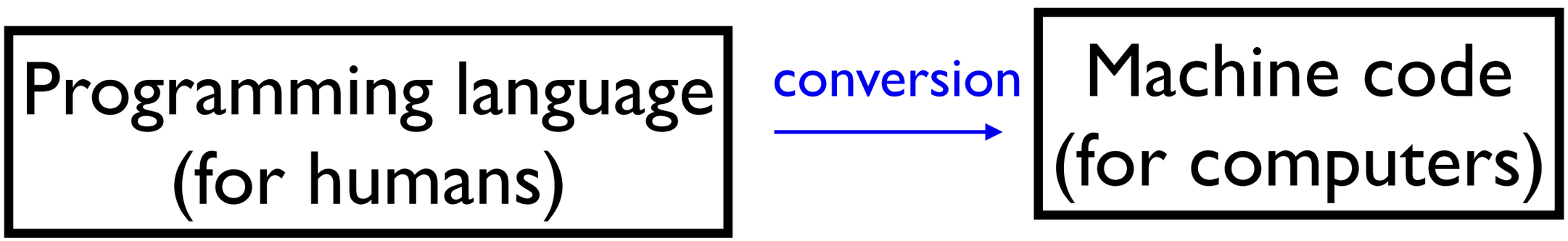
- Syntax refers to the rules or grammar of a programming language.
  - Programmers, regardless of experience, often encounter syntax errors. Thus the program crashes.
  - As we'll see, these are not the most difficult errors or bugs to fix. Here the program runs, but it doesn't do what we want. Fixing bugs requires careful thought because as far as the computer is concerned, everything is fine!

Examples

- Syntax errors are analogous to grammatical errors:
  - The dogs is furry.
- A bug is analogous to a grammatically correct but nonsensical sentence:
  - The students uploaded their assignments while Canvas was down for maintenance.

Languages for humans and computers

- Programming languages are for humans not computers!
- In order to run the program, it must be converted into a suitable format for the computer, which is referred to as machine code.



Generating executable code

- How do we convert our program?
  - A compiler converts the entire program before we run it. This yields a separate executable file. Most of the programs on your computer or phone were generated using a compiler.
  - An interpreter converts the program line by line, every time that it's run. Interpreted programs are slower. Python is (basically) an interpreter.

Advantages of an interpreter

1. Convenience

   - One file for the source code and the executable.

2. Ease of use

   - The syntax checking of compilers is much stricter.
   - This can be very frustrating! One can spend days just trying to get a code to compile…

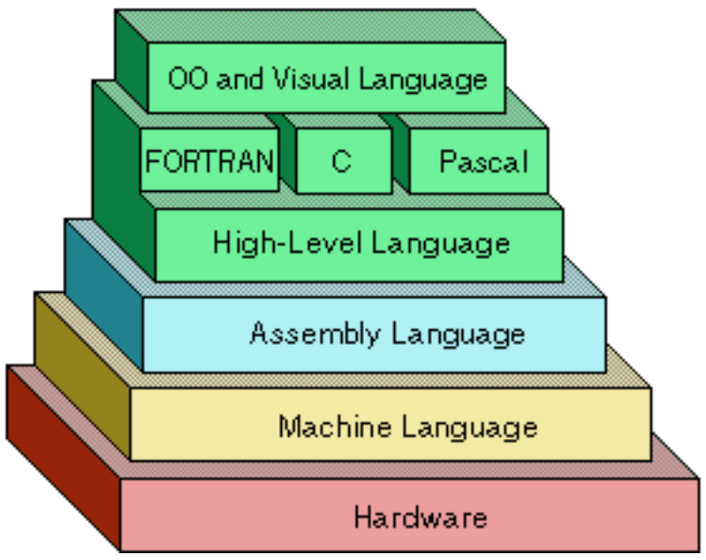
3. Interactive environment

   - You can try out the commands without running a program. In Python this can be done with the Python shell.

In most cases these advantages outweigh the (slight) performance penalty.

High-level versus low-level languages

- A low-level computer language resembles the actual **machine code** understood by the CPU.
- A high-level computer language skips many details, i.e., it allows for an abstracted representation.
- *Example*: 'bake a cookie' as opposed to all the steps in the recipe.



link

Developing code

- One can write a program with a text editor before compiling or running it. This is a very old-fashioned approach.
  - *Note*: Word is a word processor not a text editor!
- Alternatively one can use an integrated development environment (IDE).
- In the computer labs, you can use Anaconda. It is a Python IDE that includes an editor and the Python interpreter. You can run your program by clicking on a button.

Note: we'll say more about Python in the next section.

Summary

1. Computers are literal-minded.
2. Computing is unavoidable in modern engineering.
3. Computers store information as binary numbers or bits.
4. Syntax refers to the grammar of a computer language.