



Computers Technology and Programming with Python

Prepared by:

DR. WESSAM FIKRY
DR. HEBA GAMAL

2022

Table of Contents

Chapter 1: Introduction.....	4
1.1 What makes a computer a computer!.....	4
1.2 Computers then and Now.....	5
1.3 Classes of Computers.....	7
1.3.1 Personal Mobile Device (PMD)	7
1.3.2 Desktop Computing	7
1.3.3 Servers.....	8
1.3.4 Clusters/Warehouse-Scale Computers	8
1.3.5 Embedded Computers	8
Chapter 2: Binary, Data, and Circuits	10
2.1 Number Systems	11
2.1.1 Decimal number system	11
2.1.2 Binary number system	12
2.1.3 Converting decimal to binary.....	14
2.1.4 Converting binary to decimal.....	16
2.1.5 Hexadecimal number system.....	17
2.1.6 Converting decimal to hexadecimal.....	18
2.1.7 Converting hexadecimal to decimal.....	18
2.1.8 Converting binary to hexadecimal	20
2.1.9 Converting hexadecimal to binary	21
2.1.10 Binary Addition	23
2.2 Data representation.....	24
2.2.1 Numbers.....	25
2.2.2 Text.....	25
2.2.3 Images, videos and sound.....	26
2.2.4 Converting analog data to binary	27
2.3 Digital circuits.....	28

2.3.1	Logic gates.....	29
2.3.2	A simple circuit.....	32
Chapter 3: Computer Components		35
3.1	Input and Output devices	35
3.2	Central Processing Unit (CPU).....	37
3.3	Computer memory.....	38
3.4	Hardware and software	39
3.4.1	Computer programs.....	40
3.4.2	The operating system.....	41
Chapter 4: Problem Solving.....		43
4.1	Steps Involved in Problem Solving	44
4.1.1	Problem Solving Stages.....	44
4.2	Computer Algorithms.....	45
4.2.1	What Is an Algorithm?	45
4.2.2	Algorithm characteristics	45
4.2.3	Algorithms and Computers: A Perfect Match	46
4.2.4	Algorithms for Simple Problems	46
4.3	What Is Computer Software	48
4.3.1	Syntax and Semantics	49
4.3.2	Program Translation.....	49
4.3.3	Program Debugging: Syntax Errors vs. Semantic Errors	51
Chapter 5: Python Programming Language		52
5.1	Install Python	53
5.2	Write a Python Program.....	54
5.2.1	The Interactive Window.....	54
5.2.2	The Editor Window	55
5.2.3	Your first python program	55
5.3	Variables and Data types	56
5.3.1	Data types	56
5.3.2	Variables.....	57

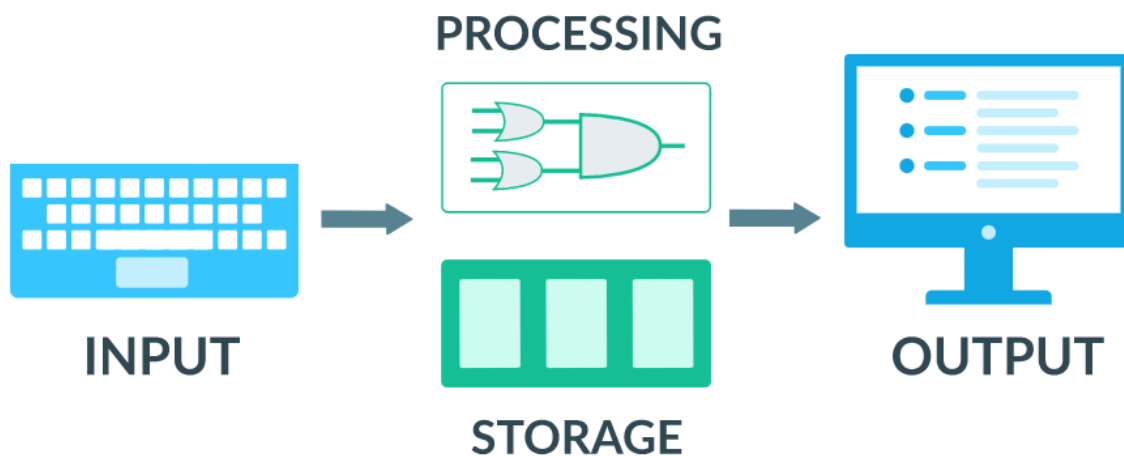
5.4	Operators and expresions.....	58
	Comparison Operators.....	60
5.5	Input and output.....	61
5.6	Comments.....	62
5.7	The import statement.....	64
5.8	Functions.....	65
5.9	Conditions (The if statement)	66
5.10	Loops (The for statement)	68
Chapter 6: Python Turtle		71
6.1	Turtle graphics	71
6.2	Moving your turtle	72
6.3	Examples using turtle.....	73
6.4	Function in python turtle	79
6.5	Drawing round and square shape	83
Chapter 7: The internet.....		88
7.1	How it all started.....	89
7.2	What is the internet.....	90
7.3	How does the internet works	90
7.3.1	Wires ,cables and wifi	90
7.3.2	IP address and DNS	91
7.3.3	Packets and Routing.....	92
7.3.4	HTTP and HTML.....	93
7.3.5	The ingredients of the Internet.....	94
References		95

1.1 What makes a computer a computer!

Humans have always searched for means to help them do the hard work. They started inventing machines to do the work for them to make their life easier. Then came the thought! could we create a machine to help us think? A machine designed to manipulate information.

When the pioneers started thinking about how to build computers, they realized they will need to perform **four** main tasks.

- Take input
- Store data
- Process it
- Output the result

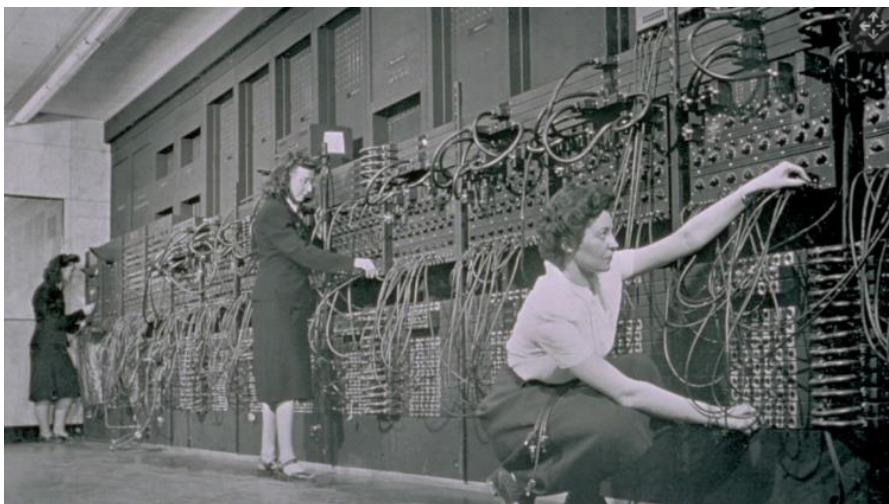


1.2 Computers then and Now

The idea of calculating with a machine date back to 500 BC when the Babylonians, the ancestors of the present-day Iraqis, invented the abacus, the first mechanical calculator. The abacus, which uses strings of beads to perform calculations.



The start of World War II produced a large need for computer capacity, especially for the military. In 1942, John P. Eckert, John W. Mauchly, and their associates at the Moore school of Electrical Engineering of University of Pennsylvania decided to build a high - speed electronic computer to do the job. This machine became known as **ENIAC** (Electrical Numerical Integrator and Calculator)The ENIAC was a huge machine, containing over 17,000 vacuum tubes and over 500 miles of wires. The ENIAC was programmed by rewiring its circuits—a process that took many workers several days to accomplish.



Early in the 50's an important engineering discovery changed the image of the electronic computer field, from one of fast but unreliable hardware to an image of relatively high reliability and even more capability. This discovery was the **Transistor** - Circuit Element.

This technical discovery quickly found its way into new models of digital computers. These machines were very expensive to purchase or even to rent and were particularly expensive to operate because of the cost of expanding programming. Such computers were mostly found in large computer centers operated by industry, government, and private laboratories - staffed with many programmers and support personnel.

Still, most people had no direct contact with either type of computer, and the machines were popularly viewed as impersonal giant brains that threatened to eliminate jobs through automation. The idea that anyone would have his or her own desktop computer was regarded as far-fetched.

Now, Computers are around us everywhere. Laptops, smartphones, inside cars, servers, game consoles, in routers, washing machines, ... etc. less than \$500 will purchase a mobile computer that has more performance, more main memory, and more disk storage than a computer bought in 1985 for \$1 million.



This rapid improvement has come both from advances in the technology used to build computers and from innovations in computer design. That smartphone in your pocket has more computing power than those refrigerator-sized mainframes used about 30 years ago.

1.3 Classes of Computers

The changes in computer use have led to five different computing markets, each characterized by different applications, requirements, and computing technologies. We can identify five mainstream of computing classes.

1.3.1 Personal Mobile Device (PMD)

Personal mobile device (PMD) is the term we apply to a collection of wireless devices with multimedia user interfaces such as cell phones, tablet computers, and so on.

1.3.2 Desktop Computing

The first, and probably still the largest market in computers is desktop computing. The combination of performance and price of a system is what matters most to customers in this market.

1.3.3 Servers

As the shift to desktop computing occurred in the 1980s, the role of servers grew to provide larger-scale and more reliable file and computing services. Such servers have become the backbone of large-scale enterprise computing, replacing the traditional mainframe. Servers are designed for applications that require a reliable and efficient system.



1.3.4 Clusters/Warehouse-Scale Computers



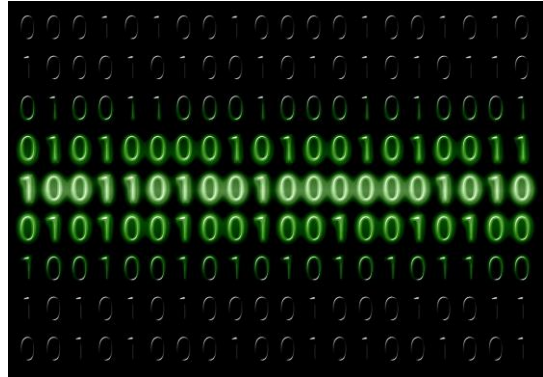
The growth of Software as a Service (SaaS) for applications like search, social networking, video sharing, multiplayer games, online shopping, and so on has led to the growth of a class of computers called clusters. Clusters are collections of desktop computers or servers connected by local area networks to act as a single larger computer. Each node runs its own operating system, and nodes communicate using a networking protocol. The largest of the clusters are called warehouse-scale computers (WSCs), in that they are designed so that tens of thousands of servers can act as one.

1.3.5 Embedded Computers

Embedded computers are found in everyday machines; microwaves, washing machines, most printers, most networking switches, and all cars contain simple

embedded microprocessors. The processors in a PMD are often considered embedded computers, but we are keeping them as a separate category because PMDs are platforms that can run externally developed software and they share many of the characteristics of desktop computers. Other embedded devices are more limited in hardware and software sophistication.

Computers work on ones and zeros and even though nobody today actually works directly with those ones and zeros they are the basis of how computers work on the inside.



Inside the computer are electrical wires and circuits that carry all the information in a computer. How do you store or represent information using that electricity. If you have a wire with electricity running through it the signal would either, be **ON** or **OFF**. This is an important start, with one wire we can represent a yes or a no, true or false, **one** or **zero**. This one piece of information is called a bit and it the smallest form of information the computer can store. To understand how to use these ones and zeros we need to understand the binary number system.



2.1 Number Systems

As humans, we typically represent numbers in the decimal system. Counting to ten is as simple as 1,2,3,4,5,6,7,8,9,10. Computers represent all information in bits, so to represent numbers with just 0s and 1s computers use the binary number system.

2.1.1 Decimal number system

Let's start by revisiting the decimal number system. When you learned how to count, you might have learned that the right-most digit is the "ones' place", the next is the "tens' place", the next is the "hundreds' place", etc.

Another way to say it is that the digit in the right-most position is multiplied by 1, the digit one place to its left is multiplied by 10, and the digit two places to its left is multiplied by 100. This is called positional notation.

Let's visualize the number 573

5	7	3
<i>hundreds' place</i>	<i>tens' place</i>	<i>ones' place</i>
× 100	× 10	× 1
500	70	3

$$573 = (5 \times 100) + (7 \times 10) + (3 \times 1)$$

We can also think of those places in terms of the powers of ten. The ones' place represents multiplying by 10^0 , the tens' place represents multiplying by 10^1 , and the hundreds' place represents multiplying by 10^2 . Each place we add, we're

multiplying the digit in that place by the next power of 10. So, the number 573 will be represented as follows.

$$573 = (5 \times 10^2) + (7 \times 10^1) + (3 \times 10^0)$$

2.1.2 Binary number system

The binary system works the same way as decimal. The only difference is that instead of multiplying the digit by a power of 10, we multiply it by a power of 2. This is called the base of the number system.

Now let's try to count to ten using binary numbers. We start by 0,1 ... 2 ?!, well two is not a number that exist in the binary number system which only has the two digits 0, and 1. The solution would be positional notation where we move to the next place so 2 in decimal will be represented as 10 in binary which we could write as 10_2 to distinguish it from decimal. It could be represented as:

$$(1 \times 2^1) + (0 \times 2^0), \text{ or } (1 \times 2) + (0 \times 1).$$

So, the place values for the decimal numbers are 1, 10, 100, ... and so on. While in binary the place values will be the powers of two: 1, 2, 8, 16, 32, ... and so on.

Consider another number, By looking at the place values of the binary system the decimal number 6 could be the sum of 2 and 4, so in binary it could be represented as follows:

0	1	1	0
8	4	2	1

$$(1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0), \text{ or}$$

$$(1 \times 4) + (1 \times 2) + (0 \times 1)$$

Knowing that let's count to ten again, we will need four places to do so

Decimal number	2^3 8	2^2 4	2^1 2	2^0 1	Binary number
0	0	0	0	0	0000
1	0	0	0	1	0001
2	0	0	1	0	0010
3	0	0	1	1	0011
4	0	1	0	0	0100
5	0	1	0	1	0101
6	0	1	1	0	0110
7	0	1	1	1	0111
8	1	0	0	0	1000
9	1	0	0	1	1001
10	1	0	1	0	1010

2.1.3 Converting decimal to binary

We already have done that when counting to 10 and this technique could work with small numbers. Here we will represent a different technique to convert from decimal to binary.

For numbers smaller than 255 we can do the following to convert the number to binary:

- Write the place for each of the bits. If the number is less than 16, write 4 places. Otherwise, for numbers up to 255, write 8 places.
- Now start at the left-most number and ask yourself "Is the number greater than or equal to this place value?" If you answer yes, then write a 1 in that place and subtract that amount from the number. If you answer no, then write a 0 and move to the next place.
- Keep going from left to right, keeping track of how much remainder you still need to represent. When you're done, you'll have converted the number to binary.

Here's what that looks like for the decimal number 6:

- 6 is less than 8, so we'll write a 0 first..

0			
<hr/>			
8	4	2	1

- 6 is bigger than 4, so we'll write a 1 next...

0	1		
8	4	2	1

- $6 - 4 = 2$, so we still need to represent 2

0	1	1	
8	4	2	1

- $2 - 2 = 0$, so there's nothing left to represent

0	1	1	0
8	4	2	1

A general form for converting decimal to any system is to repeatedly divide the number by the base of the desired system. In the binary system the base is 2.

For example to convert the number 57 to binary we repeatedly divide the number by 2 and observe the remainder

Operation	Reminder	
$57/2 = 28$	1	← least significant bit
$28/2 = 14$	0	
$14/2 = 7$	0	
$7/2 = 3$	1	
$3/2 = 1$	1	
$1/2 = 0$	1	← Most significant bit

$$57_{10} = 111001_2$$

Example1 : Convert number 91 to the binary system

Solution:

dividing by 2

91		1	
45		1	
22		0	
11		1	
5		1	
2		0	
1		1	← Most significant bit

$$91_{10} = 1011011_2$$

2.1.4 Converting binary to decimal

To convert any number from any system to decimal what we do is multiplying each number by the base to the power of the position of the digit.

Let's reverse the previous example and convert the number 111001_2 to decimal. Starting with the least significant digit and multiplying each with powers of 2 raised to the position of the digit in the number

$$\begin{aligned}
 111001_2 &= (1 \times 2^0) + (0 \times 2^1) + (0 \times 2^2) + (1 \times 2^3) + (1 \times 2^4) + (1 \times 2^5) \\
 &= (1 \times 1) + (0 \times 2) + (0 \times 4) + (1 \times 8) + (1 \times 16) + (1 \times 32) = 57
 \end{aligned}$$

Example2: Convert the binary number 10101101 to decimal

Solution:

$$\begin{aligned}
 10101101_2 &= (1 \times 2^0) + (0 \times 2^1) + (1 \times 2^2) + (1 \times 2^3) + (0 \times 2^4) \\
 &\quad + (1 \times 2^5) + (0 \times 2^6) + (1 \times 2^7) \\
 &= (1 \times 1) + (0 \times 2) + (1 \times 4) + (1 \times 8) + (0 \times 16) \\
 &\quad + (1 \times 32) + (0 \times 64) + (1 \times 128) = 173 \\
 10101101_2 &= 173_{10}
 \end{aligned}$$

2.1.5 Hexadecimal number system

Binary numbers are a great way for computers to represent numbers. They're not as great for humans though—they're so very long, and it takes a while to count up all those 1s and 0s. When computer scientists deal with numbers, they often use either the decimal system or the hexadecimal system.

In the decimal system, each digit represents a power of 10. We call 10 the base of the decimal system. In the hexadecimal system, each digit represents a power of 16.

Now that you know how number systems work you can figure out that the hexadecimal system with 16 as its base will have 16 digit to represent it, from 0 to 15. Well, we only have ten digits from 0 to 9. To represent all the required digits the system uses letters A-F to represent numbers 10-15. So, the digits in the hexadecimal system looks as follows:

1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F ...

2.1.6 Converting decimal to hexadecimal

To represent a decimal number in hexadecimal we use the same way of repetitive division. In this case the number is divided by 16. Let's convert the number 57 into hexadecimal

Operation	Reminder
$57/16 = 3$	9
$3/16 = 0$	3

$$57_{10} = 39_H$$

Example3: Convert number 127_{10} to hexadecimal

Solution:

$$\begin{array}{r|l} 127 & 15 \rightarrow F \\ 7 & 7 \end{array}$$

$$127_{10} = 7F_h$$

2.1.7 Converting hexadecimal to decimal

To convert the number from hexadecimal to decimal each digit is multiplied by 16 raised to the position of the digit as follows

$$\begin{aligned} 39_H &= (9 \times 16^0) + (3 \times 16^1) \\ &= (9 \times 1) + (3 \times 16) = 57_{10} \end{aligned}$$

Example4: Convert number $81C_{16}$ to decimal

Solution:

$$81C = (12 \times 16^0) + (1 \times 16^1) + (8 \times 16^2) = 2076$$

$$81C_{16} = 2076_{10}$$

At this point, you might be wondering what it is that computer scientists like so much about the hexadecimal system. Why use a system where we have to use letters to represent numbers.

Early computers used 4-bit architectures, which meant that they always processed bits in groups of 4. How many values can 4 bits represent? The lowest value is 0 (all 0s, 0000) and the highest value is 15 (all 1s, 1111), so 4 bits can represent 16 unique values.

Each group of 4 bits in binary is a single digit in the hexadecimal system. That makes it really easy to convert binary numbers to hexadecimal numbers, and it makes it a natural fit for computers to use as well.

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

2.1.8 Converting binary to hexadecimal

We'll start with a short binary number:

0010

That's 4 bits long, which means it maps to a single hexadecimal digit. There's a 0 in every place except the twos' place, so it equals the decimal number 2. The decimal and hexadecimal systems both represent the numbers 0-9 the same way, so 0010 is simply 2 in hexadecimal.

Now, let's try a longer binary number

1001101001101100

One approach would be to convert the binary number into a decimal number and then convert that to hexadecimal. That approach would work, but it seems like a lot of work for this long of a number.

The easier approach is to convert each group of 4 bits, one at a time.

1001	1010	0110	1100
9	A	6	C

Example5: Convert 11010110101001010 to hexadecimal

Solution: We start by the right most number and we could add zeros to the number from the left side to make have sets of four bits

1	1010	1101	0100	1010
0001	1010	1101	0100	1010
1	A	D	4	A

2.1.9 Converting hexadecimal to binary

This too is done simply by converting each digit in hexadecimal to the equivalent 4 bits in binary.

For example, lets convert the number 4C29 to binary

4	C	2	9
0100	1100	0010	1001

Example 6: Convert the number 1B21D to binary

Solution:

1	B	2	1	D
0001	1011	0010	0001	1101

$$1B21D_h = 0001\ 1011\ 0010\ 0001\ 1101_2$$

Binary numbers can be intimidatingly long, but hexadecimal are the exact opposite. Hexadecimal numbers describe a number in fewer numbers than both binary. And the simplicity in the conversion between the two systems makes the hexadecimal a good choice to represent values that is used by computer scientists in different fields.

As one example, web developers use hexadecimal numbers to represent colors. We describe colors as a combination of three components: red, green, and blue. Each of those components can vary from 0 to 255. A color like blue can be written as `rgb(0, 0, 255)` or the more concise hexadecimal version, `#0000FF`.

2.1.10 Binary Addition

Mathematical operations can be performed on binary numbers just like decimal numbers. As an example of these operations, we will introduce binary addition.

The rules of binary addition are given in the following table

<i>Augend</i>	<i>Addend</i>	<i>Sum</i>	<i>Carry</i>	<i>Result</i>
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

The procedure of adding two binary numbers is same as that of two decimal numbers. Addition is carried out from the least significant bit (LSB) and it proceeds to higher significant bits, adding the carry resulting from the addition of two previous bits each time.

Example 6: Add the binary numbers

(a) 1010 and 1101

(b) 0110 and 1111

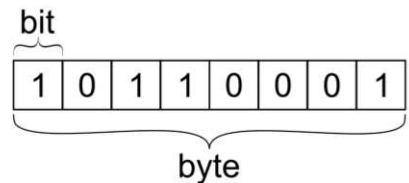
Solution:

$$\begin{array}{r}
 \text{(a)} \quad \quad \quad 1 \ 0 \ 1 \ 0 \\
 + \quad \quad \quad 1 \ 1 \ 0 \ 1 \\
 \hline
 1 \ 0 \ 1 \ 1 \ 1 \\
 \uparrow \\
 \text{Carry}
 \end{array}$$

		(1)	(1)		← Carry
(b)	0	1	1	0	
	1	1	1	1	
	1	0	1	0	1

2.2 Data representation

All data in the computer are stored in the form of ones and zeros. The smallest unit of data is called a bit which represents a single digit with one or zero. Because bits are so small, you rarely work with information one bit at a time. Bits are usually assembled into a group of 8 to form a byte. A kilobyte (KB) is 1024 bytes, not one thousand bytes as might be expected, because computers use binary math, instead of a decimal system. Computer storage and memory is often measured in megabytes (MB) and gigabytes (GB). Similarly, 1 MB is 1024 KB, and 1 GB is 1024 MB.



In summary:

- **Nibble** - 4 bits (half a byte)
- **Byte** - 8 bits
- **Kilobyte (KB)** - 1024 bytes
- **Megabyte (MB)** - 1024 kilobytes
- **Gigabyte (GB)** - 1024 megabytes
- **Terabyte (TB)** - 1024 gigabytes

2.2.1 Numbers

Guess what, by learning about the binary number system we now learned how the computer can represent numbers. In the computer, numbers are represented as a set of ones and zeros as we learned in the previous section. But what about other forms of data like text, images, sound, and ... video games !

2.2.2 Text

As we learned before the computer only deals with ones and zeros so there has to be away where we represent letters as ones and zeros. Some humans, many years ago, collectively decided on a standard mapping of numbers to letters. The letter “A”, for example, is the number 65, and “B” is 66, and so on. In binary, the letter “A” is the pattern 01000001. By using context, like the file format, different programs can interpret and display the same bits as numbers or text.

A common standard mapping called **ASCII** (American Standard Code for Information Interchange) is used to map lower- and upper-case letters as well as punctuation to numbers. ASCII was one of the first standardized encodings. It was invented back in the 1960s when telegraphy was the primary form of long-distance communication but is still in use today on modern computing systems.

When we receive a text message, we might be getting patterns of bits that have the decimal values 72, 73, and 33. Those bits would map to the letters HI!. And the sequences of bits we receive would look like 01001000, 01001001, and 00100001, with 8 bits for each character.

However, ASCII didn't provide enough space for different character sets from different languages. In 1987, a group of computer engineers attempted to solve this problem and came up with Unicode. **Unicode** uses more bits than ASCII to be able to represent all these characters. The Unicode character set started with 7,129 named characters in 1991 and grown to 137,929 named characters in 2019.

2.2.3 Images, videos and sound

With bits, we can map numbers to colors as well. There are many different systems to represent colors, but a common one is RGB, which represents colors by indicating the amount of red, green, and blue within each color. Each number might be 8 bits, with 256 possible values, so with three bytes, or 24 bits, we can represent millions of colors. You can try it here, <https://www.colorspire.com/rgb-color-wheel/>.

A digital image is actually a set of small dots called pixels. So, by using three bytes to represent the color for each pixel, we can create images.



Videos are sequences of many images, changing multiple times a second to give us the appearance of motion, as a flipbook.

https://www.youtube.com/watch?v=sz78_07Xg-U

Music can be represented with bits, too. MIDI is one such format which represents music with numbers for each of the notes and their duration and volume. So, all of these ideas are just zeroes and ones, interpreted and used by software we've written to interpret them in the ways that we want.

There are other formats, some of which use compression (mathematical ways to represent some data with fewer bits), or some which might be containers that store multiple types of data together.

And since there are many companies and groups developing software, we have lots of different file formats in existence, each with their own ways of representing data. But there are also organizations that work on some consensus, like the one responsible for maintaining the Unicode standard.

https://en.wikipedia.org/wiki/Unicode_Consortium

2.2.4 Converting analog data to binary

The real world is analog, a continuous stream of varying data. Just look around you, beyond your computer or phone. There's an infinite amount of visual information. If you zoom into one part of your visual field, you can notice more and more details.

Now sing a little song to yourself. That's an infinite stream of audio information. Your voice is constantly changing in big and little ways, microsecond by microsecond.

Analog data is infinitely detailed. Computers can only store digital data, finite data in a binary representation. That's why for every type of media there are ways to convert those continuous signals to the discrete form a computer can store and manipulate.

2.3 Digital circuits

Every input or output of a computer is a type of information that are represented by ones and zeros. To process the information that comes in as input and to produce the desired output a computer needs to modify and combine those signals. To do this a computer uses millions of tiny electronic components that come together to form circuits.

In 1947, engineers invented the transistor, a tiny physical device that acts like a digital switch in computers. The **transistor** turns on when enough electricity flows through and stays off otherwise. The circuits in a computer's processor are made up of billions of transistors, it is so small, we would need a high-powered microscope to see them. The smallest form of a circuit build from those transistors are called logic gates and they are the main building blocks for digital circuits.

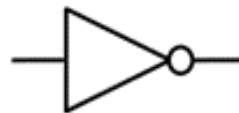
2.3.1 Logic gates

Computers use logic gates to transform the 1s and 0s from input wires. A logic gate accepts inputs and then outputs a result based on their state.

2.3.1.1 NOT Gate

The simplest gate is the NOT gate, also known as an inverter. It accepts a single input and outputs the opposite value. We represent all the possible values for the circuit using a truth table.

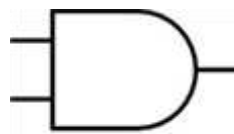
Input	Output
0	1
1	0



2.3.1.2 AND Gate

All other logic gates operate on multiple inputs. The AND gate accepts two inputs, and if both of those wires are "on" (representing 1), it outputs 1.

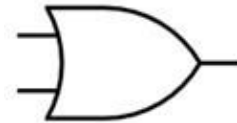
Input A	Input B	Output
0	0	0
0	1	0
1	0	0
1	1	1



2.3.1.3 OR Gate

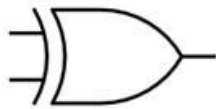
The OR logic gate accepts two inputs, and as long as either of those inputs is a 1, it outputs a 1

Input A	Input B	Output
0	0	0
0	1	1
1	0	1
1	1	1



Other gates include XOR, XNOR, NOR, and NAND

XOR Gate



Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

XNOR Gate



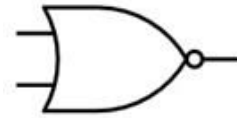
Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

NAND Gate



Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

NOR Gate



Input A	Input B	Output
0	0	
0	1	
1	0	
1	1	

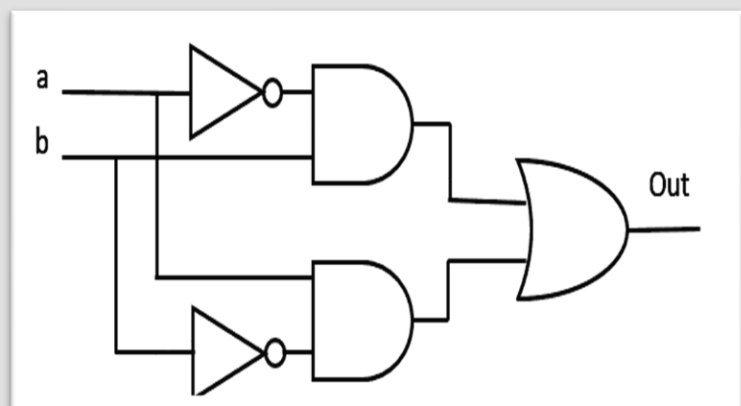
Here's a nice tool for simulating digital circuits, use it to fill the truth tables for the previous 4 gates: <https://logic.ly/demo/>

Activity 1: try by yourself, then verify using simulating tool

Fill in the truth table, given the following Logic Circuit made from Logic AND, OR, and NOT gates. What does the logic circuit do?

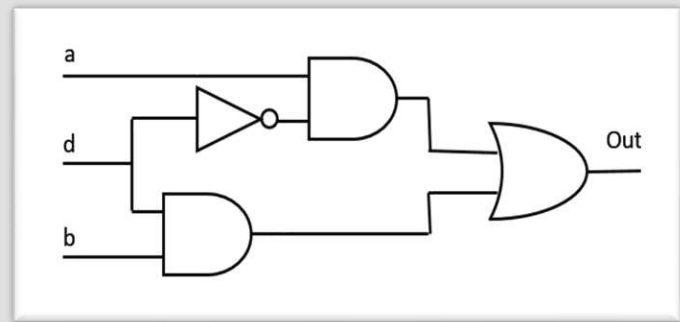
(a)

a	B	Output
0	0	
0	1	
1	0	
1	1	



(b)

a	b	d	Output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



Computer hardware manufacturers prefer to use **NAND** gates due to their universality and ease of fabrication, so your computer likely has millions of NAND gates inside its circuitry.

2.3.2 A simple circuit

Now let's use our knowledge of logic gates to design a circuit that actually do something. We will design a circuit for an adder. Our adder is a logic circuit that sums two binary numbers.

First, we need to know all the possible inputs and outputs for our circuit. We will do that in the form of a truth table. Our circuit will have two inputs which are the two binary bits we need to add. Let's call them X and Y.

The four possible inputs to the circuit will be:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

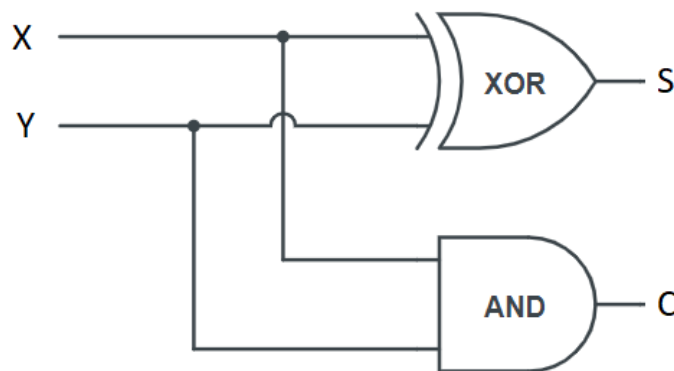
$$1 + 0 = 1$$

$$1 + 1 = 0, \quad \text{and a carry of } 1$$

So, we will also need two outputs one for the sum and another for the carry.

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

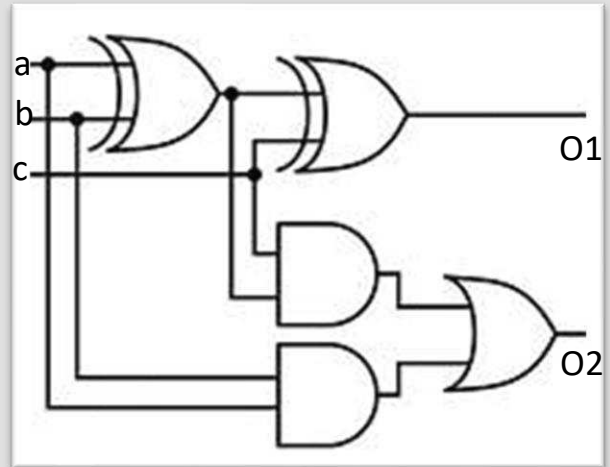
Looking at the truth table for the circuit we can figure out what logic gate we will need to build the adder. We will need an AND gate to output the carry and an XOR gate to output the sum.



Congratulations you just designed your first digital circuit.

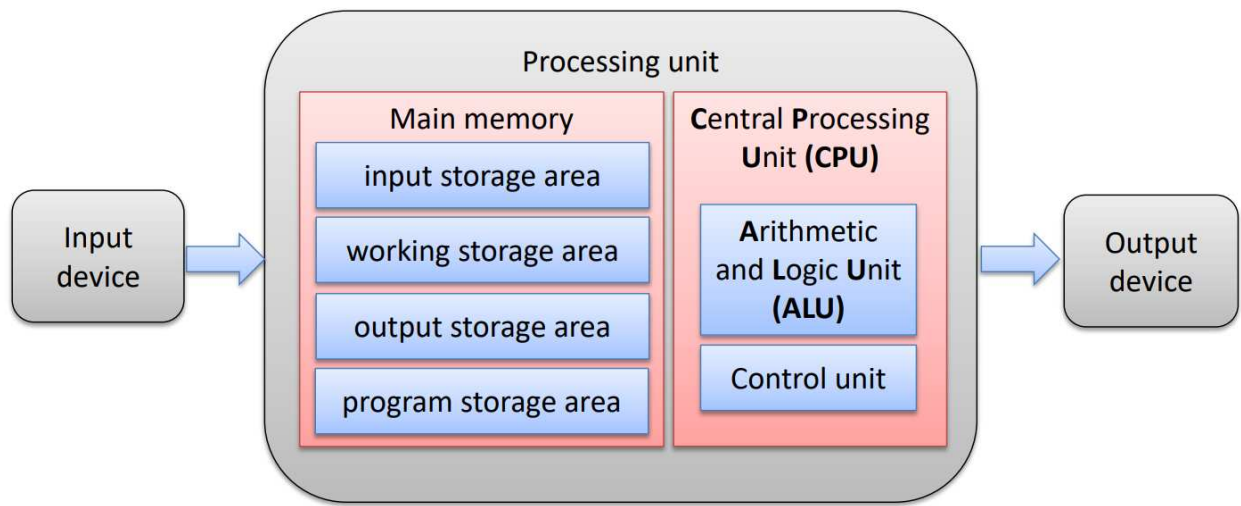
Activity 3: Fill the truth table for the following circuit.

a	b	c	O1	O2
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



After filling the table, what does this circuit do?

All computers are made up of a processor (CPU), memory, and input/output devices. Each computer receives input from a variety of devices, processes that data with the CPU and memory, and sends results to some form of output.



3.1 Input and Output devices

The most common input devices are the keyboard, mouse, and touch screen. There are hundreds of other input devices, like microphones to capture sound waves, scanners to capture image data, and virtual reality devices to capture our body movements. Computers also receive input from their environment using "sensors", like motion sensors that detect changes in movement.



KEYBOARD



MOUSE



JOYSTICK



SCANNER



WEB CAMERA



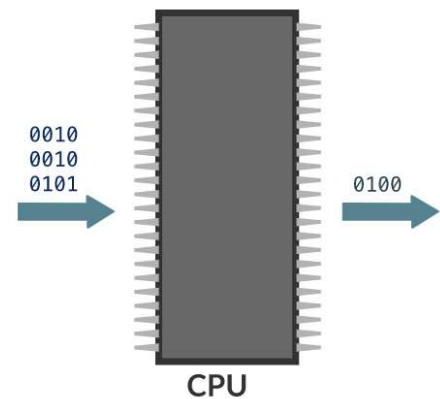
MICROPHONE

Once the CPU is done processing the data, it often needs to output a result. A standard output device is the computer monitor, which displays text, images, and user interface elements by lighting up thousands of pixels with different colors. There are many other ways a computer could output data. As long as the output device can interpret a stream of 1s and 0s, it can turn that data into anything - headphones output sound, printers' output ink on paper, and projectors output light.



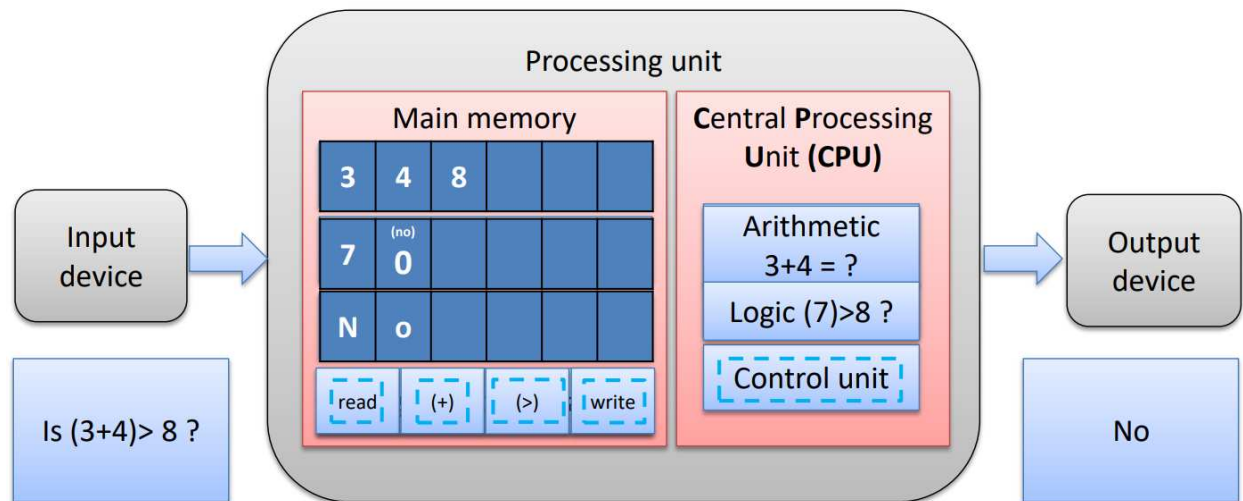
3.2 Central Processing Unit (CPU)

The CPU is the brain of a computer, containing all the circuitry needed to process input, store data, and output results. The CPU is constantly following instructions of computer programs that tell it which data to process and how to process it. Without a CPU, we could not run programs on a computer.



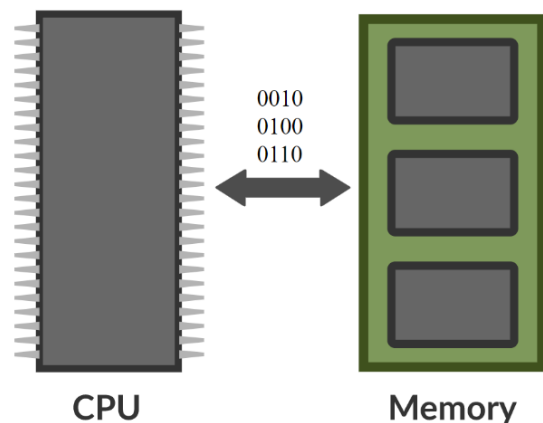
For example, a simple calculator program might instruct the CPU to take two numbers, **3** and **4**, add them, and send back the result. The CPU can process those instructions easily, thanks to a Control Unit (CU) that knows how to interpret program instructions and an Arithmetic Logic Unit (ALU) that knows how to add

numbers. With the control unit and ALU combined, the CPU can process much more complex programs than a simple calculator.



3.3 Computer memory

When input devices send binary data to a CPU, it immediately stores that data in memory to make it easier to process. Since the CPU is constantly using data from memory, they're connected via a memory bus, a high-speed communication transfer system, typically made from wires, conductors, or optical fibers.



This type of memory, also called **main memory** or RAM (Random Access Memory), is only used for temporary storage of data. When you restart a computer, it typically

wipes the memory entirely. Memory wouldn't be a good place to store data for later, like files and programs.

Computers store long-term data in a different type of memory: **external memory** or **secondary storage**, like a hard drive or USB drive. Secondary storage devices can be implemented using a variety of electronic technologies, like flash memory, optical discs, and magnetic disks.

3.4 Hardware and software

When you look inside a computer you see circuits, chips, wires, plugs and so on. This is the hardware. But what you don't see is the software, software is all the computer programs or code running on your machine, video games, webpages, text editors, or the piece of code that turns the screen brightness up when you hit a button for that.

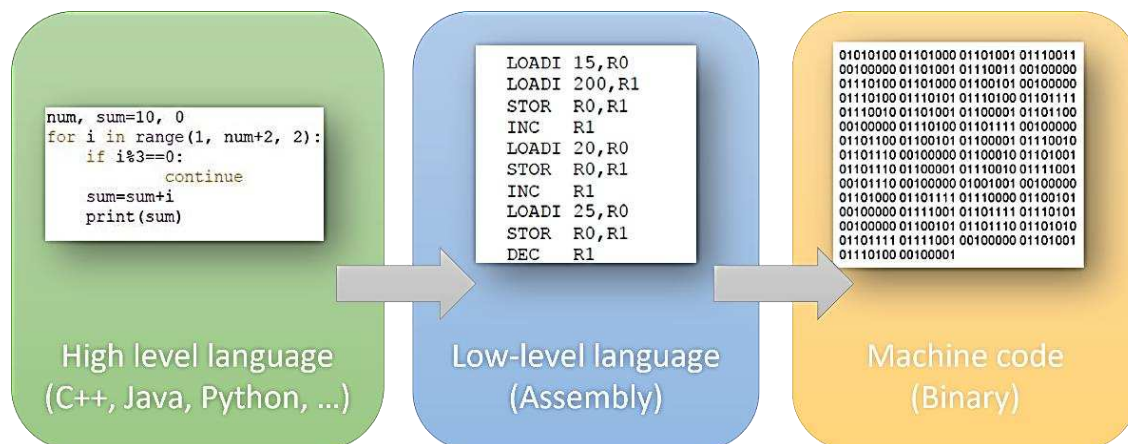
How does the software and hardware interact with one another? To know that we will go back to the CPU or the central processing unit. We mentioned earlier that this is the brain of the computer where all the operations in the computer take place. Inside the CPU are different circuits for different tasks.

The CPU has a set of simple commands that it receives to learn what circuit to use to do a specific task. For example, an add command tells the CPU to use its adder circuit to calculate a new number. And a store command tells the CPU to use a different circuit to save the result of the add command into memory.

All those commands are in binary code so that are interpreted by the CPU to perform a task. The memory hold these commands and the CPU fetches them and executes them in a sequence. This sequence of binary commands is in fact a very simple computer program or software.

3.4.1 Computer programs

Binary code is the most basic form of software, and it controls all the hardware of a computer. Today nobody writes software in binary because it would take forever. Nowadays programmers use different high-level programming languages that looks like written English.



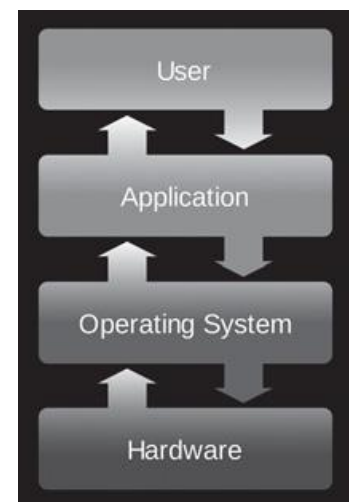
Computer programs are sets of instructions. Each instruction is translated into machine code - simple binary codes that activate the CPU. Programmers write computer code in **high-level languages**, and this code is converted by a translator into **low-level language** which is then translated into a **machine code** that the processor can execute.

When you are typing on your keyboard in a text editor, while listening to music and browsing the internet your computer is running multiple programs not just one. To do that you need another piece of software that could manage all the programs in your computer. Hence comes the need for the **operating system (OS)**.

3.4.2 The operating system

The operating system of a computer is the main program that manages how software gets to use the hardware of the computer. It is a program with special abilities that let it control the other software on the computer.

One task that the **OS** do is the installation of a new programs by loading them into the computer's memory. It decides when a program is run by the CPU, and whether



that program can access the computer's input and output devices. Managing all the



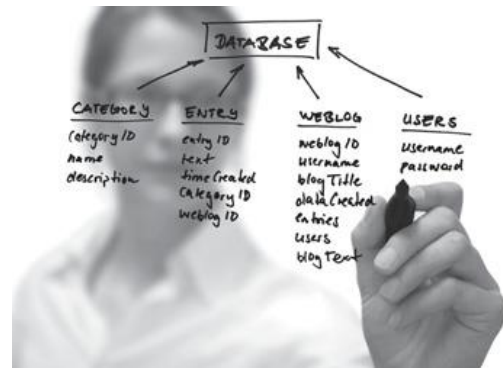
devices attached to your computer is another task performed by the operating system. All this and more other tasks make the operating system indispensable for any modern computer.



Computers are useful in all sorts of ways but the only reason they are useful is you!

When you learn to code you get to define the problem you want to solve and write the software that turns those ideas into reality that gives you the power to build things that matter to you your community and the world

A **computer** is a very powerful and versatile machine capable of performing a multitude of different tasks. It performs many tasks exactly in the same manner as it is told to do. This places responsibility on the user to instruct the computer in a correct and precise manner, so that the machine is able to perform the required job in a proper way. A wrong or ambiguous instruction may sometimes prove disastrous.



So, what Is Computer Science? Computer science is fundamentally about **computational problem solving** —that is, solving problems by the use of computation.

In order to solve a problem computationally, two things are needed: clear understanding of the problem to be solved, and an **algorithm** or of series of sequential steps, to solve it. that solves the problem by use of the representation. Once the problem is well-defined and a method of solving it is developed, then instructing the computer to solve the problem becomes relatively easier task.

4.1 Steps Involved in Problem Solving

A computer cannot solve a problem on its own. One has to provide step by step solutions of the problem to the computer. In fact, the task of problem solving is not that of the computer. It is the programmer who has to write down the solution to the problem in terms of simple operations which the computer can understand and execute.

4.1.1 Problem Solving Stages

4.1.1.1 *First: Problem Definition*

Problem definition implies the identification of required outputs, available inputs and, arithmetic and logical operations to be executed.

4.1.1.2 *Second: Performing step-by-step instructions (Algorithm) to solve a Problem*

An algorithm is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

4.1.1.3 *Third: Program design*

Having drawn a “Flowchart”, to solve the problem, using a computer; we have to translate this flowchart into one of the programming languages.

4.1.1.4 *Fourth: Program Testing*

During writing a program we unintentionally make some mistakes,

e.g. writing a minus sign (-) instead of (+). We cannot detect errors unless we begin entering data to the program with previously known results; to compare the results of

the current program to those of the well-known results; therefore, we check the errors and debug them.

4.1.1.5 Fifth: Program Documentation

All steps taken for solving the problem that include given Input, output, plan for solving the problem, drawn flowchart, programming language used for coding and, instructions, date of last modification of the program and, people who contribute to the program development process, to have the program documented to go back for feedback and correction.

The documentation is beneficial when more than one person participates in writing or modifying the program.

4.2 Computer Algorithms

4.2.1 What Is an Algorithm?

The word “**algorithm**” is derived from the ninth- century Arab mathematician, **Al-Khwarizmi**. An **algorithm** is a finite number of clearly described, unambiguous “doable” steps that can be systematically followed to produce a desired result for given input in a finite amount of time.



4.2.2 Algorithm characteristics

- **Input:** An algorithm may or may not require input
- **Output:** Each algorithm is expected to produce at least one result

- **Definiteness:** Each instruction must be clear and unambiguous.
- **Finiteness:** If the instructions of an algorithm are executed, the algorithm should terminate after finite number of steps.

4.2.3 Algorithms and Computers: A Perfect Match

the study of algorithms does not depend on the existence of computers. The algorithm for performing long division is such an example. However, most algorithms are not as simple or practical to apply manually. Most require the use of computers either because they would require too much time for a person to apply or involve so much detail as to make human error likely.

Because computers can execute instructions very quickly and reliably without error, algorithms and computers are a perfect match.

4.2.4 Algorithms for Simple Problems

Now let's look at some examples for algorithms that we can use to solve some simple problems.

Example 1: Write an algorithm to find the average of three numbers

Step 1: Read the numbers a, b, c

Step 2: Compute the sum of a, b and c

Step 3: Divide the sum by 3

Step 4: Store the result in variable d

Step 5: Print the value of d

Step 6: End of the program.

Example 2: Write an algorithm to calculate the simple interest using the formula.

Simple interest = $P * N * R / 100$.

Where P is principal Amount, N is the number of years and R is the rate of interest.

Step 1: Read the three input quantities' P, N and R.

Step 2: Calculate simple interest as

$$\text{Simple interest} = P * N * R / 100$$

Step 3: Print simple interest.

Step 4: Stop.

Example 3: Write an algorithm to find the area of the triangle. Let b, c be the sides of the triangle ABC and A the included angle between the given sides.

Step 1: Input the given elements of the triangle

Namely side's b, c and angle between the sides A.

Step 2: Area = $(1/2) * b * c * \sin A$

Step 3: Output the Area

Step 4: Stop.

Example 4: Write an algorithm to find the largest of three numbers X, Y, Z.

Step 1: Read the numbers X, Y, Z.

Step 2: if (X > Y)

Big = X

else BIG = Y

Step 3: if (BIG < Z)

Step 4: Big = Z

Step 5: Print the largest number (Big)

Step 6: Stop.

Example 5: Write an algorithm to calculate the perimeter and area of rectangle. Given its length and width.

Step 1: Read length of the rectangle.

Step 2: Read width of the rectangle.

Step 3: Calculate perimeter of the rectangle using the formula

$$\text{Perimeter} = 2 * (\text{length} + \text{width})$$

Step 4: Calculate area of the rectangle using the formula

$$\text{Area} = \text{length} * \text{width}.$$

Step 5: Print Perimeter.

Step 6: Print Area.

Step 7: Stop.

4.3 What Is Computer Software

Computer software is a set of program instructions, including related data and documentation, that can be executed by computer.

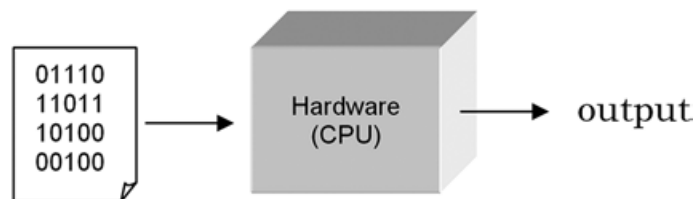
4.3.1 Syntax and Semantics

The **syntax** of a language is a set of characters and the acceptable sequences of those characters.

The **semantics** of a language is the meaning associated with each syntactically correct sequence of characters.

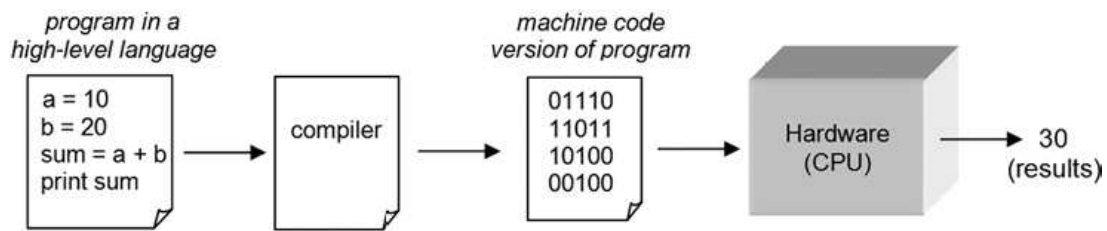
4.3.2 Program Translation

A central processing unit (CPU) is designed to interpret and execute a specific set of instructions represented in binary form (i.e., 1s and 0s) called machine code . Only programs in machine code can be executed by a CPU.

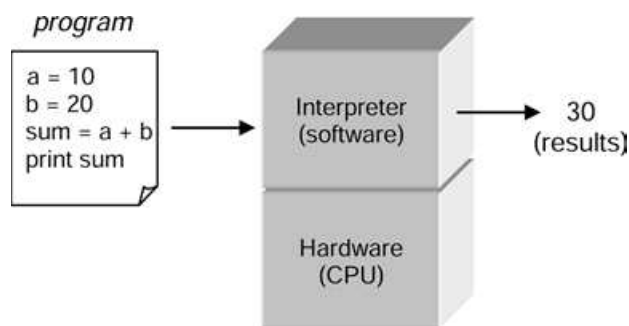


Therefore, most programs are written in a “high-level” programming language such as Python. Since the instructions of such programs are not in machine code that a CPU can execute, a translator program must be used.

There are two fundamental types of translators. One, called a **compiler** , translates programs directly into machine code to be executed by the CPU.



The other type of translator is called an **interpreter** , which executes program instructions in place of (“running on top of”) the CPU, denoted in Figure Python, as we shall see, is executed by an interpreter.



Compiler	Interpreter
A compiler takes the entire program in one go.	An interpreter takes one line of code at a time
The compiler generates an intermediate machine code	The interpreter does not generate intermediate machine code
The compiler is best suited for the production environment	The interpreter is best suited for the software development environment
The compiler is used by programming languages such as: C, C++, Java, etc	The interpreter is used by programming languages such as: Python, PHP, Ruby, etc

4.3.3 Program Debugging: Syntax Errors vs. Semantic Errors

Program **debugging** is the process of finding and correcting errors (“bugs”) in a computer program. Programming errors are inevitable during program development. **Syntax errors** are caused by invalid syntax (for example, entering `prnt` instead of `print`). Since a translator cannot understand instructions containing syntax errors, translators terminate when encountering such errors indicating where in the program the problem occurred.

In contrast, **semantic errors** (generally called **logic errors**) are errors in program logic. Such errors cannot be automatically detected, since translators cannot understand the intent of a given computation. For example, if a program computed the average of three numbers as follows:

$$(\text{num1} + \text{num2} + \text{num3}) / 2.0$$

a translator would have no means of determining that the divisor should be 3 and not 2. Computers do not understand what a program is meant to do, they only follow the instructions given . It is up to the programmer to detect such errors. Program debugging is not a trivial task and constitutes much of the time of program development.



A program with a syntax error will not run. A program with a logic error will run but it will not perform as expected.

Syntax errors are caused by invalid syntax. **Semantic (logic) errors** are caused by errors in program logic.

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to programming.



Python has a simple syntax. Python programs are clear and easy to read. At the same time, Python provides powerful programming features, and is widely used. Companies and organizations that use Python include YouTube, Google, Yahoo, and NASA. Python is well supported and freely available at www.python.org.

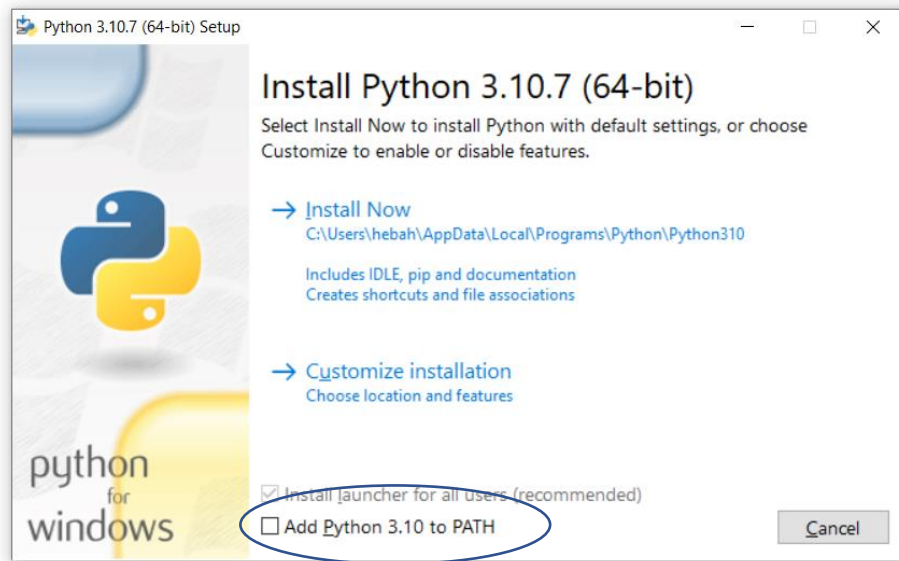
Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw away programs, or to test functions during bottom-up program development. Python enables programs to be written compactly and readably.

5.1 Install Python

Let's start by installing python,

Step 1: download the python IDE, the latest version of python could be downloaded from this link: <https://www.python.org/downloads/>

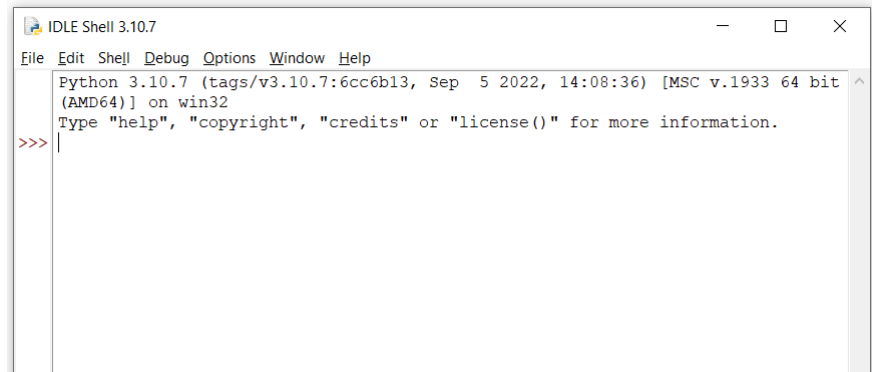
Step 2: double-click the file to run the installer. A dialog that looks like the following one will appear



Make sure you select the box that says Add Python 3.10 to PATH. If you install Python without selecting this box, then you can run the installer again and select it. Click Install Now to install Python 3. Wait for the installation to finish, then continue to open IDLE.

Step 3: From the start menu Open IDLE (Python 3.10)

The Python shell window looks like this



5.2 Write a Python Program

There are two main windows that you will work with in IDLE: the **interactive window**, which is the one that opens when you start IDLE, and the **editor window**.

5.2.1 The Interactive Window

The interactive window or the python shell opens automatically when you start IDLE. The `>>>` symbol in the last line is called the prompt. This is where you'll type in your code.

Go ahead and type `1 + 1` at the prompt and press Enter :

```
>>> 1 + 1
2
>>>
```

- 1- Python reads the code entered at the prompt.
- 2- Evaluates the code.
- 3- And prints the result and waits for more input.

5.2.2 The Editor Window

You'll write your Python files using IDLE's editor window. You can open the editor window by selecting **File -> New File** from the menu at the top of the interactive window.

The interactive window stays open when you open the editor window. It displays the output generated by code in the editor window, so you'll want to arrange the two windows so that you can see them both at the same time.

5.2.3 Your first python program

In the editor window, type in:

```
print("Hello, World")
```

Before you run your program, you need to save it. Select **File -> Save** from the menu and save the file as **hello_world.py**

To run your program, select **Run -> Run Module** from the menu in the editor window.

The function **print**, prints the string inside the quotes to the screen. And now you will see the words `Hello, World` written on the interactive screen or the python shell.

Activity : Try the following code and observe the output on the screen

```
print("  /|")
print(" / |")
print(" /  |")
print("/___|")
```

When you run a code in python what happens is that the interpreter goes through the lines of code sequentially and executes whatever is required in this line

5.3 Variables and Data types

5.3.1 Data types

When we code, we deal with different **values** like a letter or number. These values belong to different types: 2 is an integer, and 'Hello, World!' is a string, so-called because it contains a “string” of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

If you are not sure what type a value has, the interpreter can tell you. By using the function `type()`

```
>>> type("Hello, World!")  
<type 'str'>  
>>> type(17)  
<type 'int'>
```

Some examples of these data types are:

- Integer (int): to represent whole numbers, without fractions
- Floating point (float): represent numbers with a decimal point
- Strings (str): represent a set of characters
- Boolean (bool): represent a logic value, either **True** or **False**.

5.3.2 Variables

One of the most powerful features of a programming language is the ability to manipulate variables. A **variable** is a name that refers to a value.

An **assignment statement** creates new variables and gives them values:

```
>>> message = "Hi there, this is a message"
>>> n = 17
>>> pi = 3.1415926535897932
```

Here we just created three variables, with the names `message`, `n`, and `pi`. The interpreter will decide the type of each variable based on the value you assign to it.

There are just a couple of rules to follow when naming your variables.

- Variable names can contain letters, numbers, and the underscore.
- Variable names cannot contain spaces.
- Variable names cannot start with a number.
- Case matters, for instance **temp** and **Temp** are different.
- They cannot be one of python's keywords!

It helps make your program more understandable if you choose names that are descriptive, but not so long that they clutter up your program.

Keywords are names used by python that you can't use as a name for a variable. Words like `for`, `if`, `import`, `else`, `class`, You will learn more about those words as you start using python.

You can also declare a list of values under one name, and you can access each value in the list by specifying its index.

```
>>> colors = ["red", "blue", "green"]
>>> colors[0]
red
>>> colors[2]
green
>>> L = [10,20,30]
>>> L[1]
20
```

5.4 Operators and expressions

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called **operands**.

```
>>> a = 10
>>> b = 20
>>> a + b - 5
25
```

A sequence of operands and operators, like $a + b - 5$, is called an **expression**. Python supports many operators for combining data objects into expressions. Here we will talk about two types of those operators.

Assignment operator

The '=' operator in python doesn't mean two values are equal. It means to assign value of right side of expression to left side operand.

```
>>> a = 10
```

Arithmetic Operators

Operator	Example	Meaning	Result
+	a + b	Addition	Sum of a and b
-	-a	Negation	Value equal to a but opposite in sign
-	a - b	Subtraction	b subtracted from a
*	a * b	Multiplication	Product of a and b
/	a / b	Division	Quotient when a is divided by b. The result always has type float.
%	a % b	Modulo	Remainder when a is divided by b
**	a ** b	Exponentiation	a raised to the power of b

Here are some examples of these operators for you to try

```
>>> a = 4
>>> b = 3
>>> +a
4
>>> -b
-3
>>> a + b
7
>>> a - b
1
>>> a * b
12
>>> a / b
1.3333333333333333
>>> a % b
1
>>> a ** b
64
```

The precedence in operators is something to consider, in arithmetic operations the precedence is ** then *,/,% then +,- . And you can override this precedence by using parentheses ().

If you are trying the expression $\frac{a}{2b}$ and you wrote it like this

```
>>> a / 2 * b
```

You will get the result for $\frac{a}{2} * b$, however it should be written as follows:

```
>>> a / (2 * b)
```

Comparison Operators

Operator	Example	Meaning	Result
==	a == b	Equal to	True if the value of a is equal to the value of b False otherwise
!=	a != b	Not equal to	True if a is not equal to b False otherwise
<	a < b	Less than	True if a is less than b False otherwise
<=	a <= b	Less than or equal to	True if a is less than or equal to b False otherwise
>	a > b	Greater than	True if a is greater than b False otherwise
>=	a >= b	Greater than or equal to	True if a is greater than or equal to b False otherwise

Here are some examples of comparison operators in use

```
>>> a = 10
>>> b = 20
>>> a == b
False
>>> a != b
True
>>> a <= b
True
>>> a >= b
False
>>> a = 30
```

```
>>> b = 30
>>> a == b
True
>>> a <= b
True
>>> a >= b
True
```

5.5 Input and output

When writing our first program we already used `print()` function which is our main output function for now. We used `print` to output a string on the screen, now let's use it print a variable. By default, `print()` separates objects by a single space and appends a newline to the end of the output:

```
>>> first_name = "Winston"
>>> last_name = "Smith"
>>> age = 30

>>> print("Name:", first_name, last_name)
Name: Winston Smith
>>> print("Age:", age)
Age: 30
>>> print("Name:"+first_name+last_name)
Name: Winston Smith
>>> print("Age:", age)
```

Can you see the difference between printing a string as it is and printing a variable by using its name.

Input means that you will be reading a value entered by the user using the keyboard.

The `input()` function pauses program execution to allow the user to type in a line of input from the keyboard. Once the user presses the `Enter` key, all characters typed are read and returned as a [string](#):

```
>>> age = input()
50

>>> print("Age:", age)
Age: 50
```

For clarity you can add a message to be displayed with the input prompt

```
>>> age = input("Enter your Age:")
Enter your Age: 50

>>> print("Age:", age)
Age: 50
```

`input()` always returns a string. If you want a numeric type, then you need to [convert the string](#) to the appropriate type with functions like `int()`, `float()`

```
>>> number = input("Enter a number: ")
Enter a number: 50

>>> print(number + 100)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int

>>> number = int(input("Enter a number: "))
Enter a number: 50
>>> print(number + 100)
150
```

5.6 Comments

While writing your code you might want to leave some useful tips or clarifications for yourself or for whomever reading your code. Those are called comments, a comment is a line that is not considered by the interpreter. It is only there to

provide useful information for the code reader. To write a comment you only need to add the symbol # before the line.

```
# this is a comment
```

Now we can declare variables, put them in expressions using operators, take input from the user and produce output on the screen. Let's start writing some code in the editor.

Example 1: Write a Python program that prompts the user for two values and displays the result of the first number divided by the second

```
x = input("Enter the first number: ")
y = input("Enter the second number: ")

x = float(x)
y = float(y)

print("Result = ", x/y)
```

After saving the file and running the code you get the following output

```
Enter the first number: 5
Enter the second number: 2
Result = 2.5
```

Example 2: Write a Python program that takes the radius of a circle and outputs its area.

```
import math

radius = float(input("Enter the radius: "))

area = math.pi * radius**2

print("Area = ", area)
```

After saving the file and running the code you get the following output

```
Enter the radius: 5
Area = 78.53981633974483
```

Did you notice something new in the previous example. The word `import` is new. So, what does it do.

5.7 The import statement

In python there are something called modules, which is a set of useful code gathered in one file and can be called if needed.

To use a module in your code you need to use `import` to be able to use the functions inside the module. What we used earlier is the **math** module which has many useful math function that you could use in your code. After importing the module you can use it by using the module name followed by a dot (.)

```
import math

x = int(input("Enter number: "))
print("sin(num)= ", math.sin(x))
```

Another method to use `import` use is to specify the function you need. Then you don't need to use its name.

```
from math import sin

x = int(input("Enter number: "))
print("sin(num)= ", sin(x))
```

Or maybe import all the contents in the module

```
from math import *

x = int(input("Enter number: "))
print("sin(num)= ", sin(x))
```

As you write more python programs you will get to know different modules or maybe even write your own module.

Example 3: Write a Python program that generate a random integer between 0 and 10.

```
from random import randint
x = randint(0,10)    #this function generates a random integer between 0 and 10
print("Random number = ", x)
```

5.8 Functions

A function is a named piece of code that does a certain task. When you define a function, you specify the name and the sequence of statements. Later, you can “call” the function by name.

Let’s go back the example where we calculated the area of circle and write a function to do the job. We will name our function Area

```
import math
Area()

# the code inside the function is indented to define
# that it is part of this function

def Area()
    radius = float(input("Enter the radius: "))
    area = math.pi * radius**2
    print("Area = ", area)
```

Now all the code needed to calculate the area of a circle is enclosed under the name **Area**. And to use it all we need to do is to call it by its name and the code inside the function will be executed.

Now let's extend the program to calculate the circumference of the circle as well. We will add another function to calculate it. The other function will be called `Circ` and to execute it we will call it by its name.

```
import math

Area()
Circ()

def Area()

    radius = float(input("Enter the radius: "))
    area = math.pi * radius**2
    print("Area = ", area)

def Circ()

    radius = float(input("Enter the radius: "))
    Circ = 2* math.pi * radius
    print("Circumference = ", Circ)
```

5.9 Conditions (The if statement)

To write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly.

The **if** statement uses conditional operators and Boolean expressions to decide if a condition is met. For example, if we want to test a number to know if it has a certain value we will use it as following:

```
x = 30

if x == 10:      # x==10 -> False, then the condition is not met
    print("Condition is met")
```

In the previous example nothing will be printed. If we want other statement to be executed, we can do so by using **if-else** statement

```
x = 30

if x == 10:
    print("Condition is met")
else:      # if the condition is not met this statement will be executed
    print("Condition is not met")
```

Output:

```
Condition is not met
```

Sometimes there are more than two possibilities, and we need more than two branches. One way to express a computation like that is a chained conditional.

```
x = 30
y = 50

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x equals y")
```

Output:

```
Condition is not met
```


Example 4: Write a Python program that receives a number from the user and then print if the number is odd or even.

```
x = int(input("Enter number = "))  
  
#Deviding by 2 if the remainder is zero then the number is even  
  
if x%2 == 0:  
    print("The number is even")  
else:  
    print("The number is odd")
```

Output:

```
Enter number = 5  
The number is odd
```

5.10 Loops (The for statement)

One important task to perform in programing is repetition, when you need to repeat a statement for a number of times or until some condition is met. Or maybe to run your program until the used close it himself. The way to do so in programing is by using **loops**.

Here we will only talk about one type of loops, the **for** loop. A for loop is used for iterating over a sequence (that is either a list, a set, or a string). Lets start by using for with a list.

```
colors = ["red", "blue", "green"]  
  
for x in colors:  
    print(x)
```

output:

```
red
```

```
blue  
green
```

One of the most common uses of the for loop is using it as a counter to repeat a task for a fixed number of times. To do so we use the function range with it to define a range for the statement to be repeated.

```
for x in range(5):  
    print(x)
```

output:

```
0  
1  
2  
3  
4
```

```
for x in range(3,6):  
    print(x)
```

output:

```
3  
4  
5
```

Example 5: Write a python program to sum the values from 1 to 100.

```
sum = 0                # use the variable sum that start with 0 to add the values to  
  
for x in range(1,101):  
    sum = sum + x  
  
print("Sum = ", sum)
```

Example 5: Write a python program to sum the **odd** numbers from 1 to 100.

Solution1

```
sum = 0

for x in range(1,101):
    if x%2 != 0:                # check if the number is odd, add it to the sum
        sum = sum + x

print("Sum = ", sum)
```

Solution2

```
sum = 0

for x in range(1,101,2):       # the third number 2 means to count 2 each step so
    sum = sum + x              # the range function would produce 1,3,5,...

print("Sum = ", sum)
```

More examples using for loops in the next chapter using the turtle module in python.

Python turtle graphics is one of the cool ways to implement your knowledge in Python before you start writing complex programs to solve problems in life.



In 1967, Seymour Papert and Wally Feurzeig created an interpretive programming language called Logo. Papert added commands to Logo so that he could control a turtle robot, which drew shapes on paper, from his computer.

Turtle graphics is now part of Python. Using the Turtle involves instructing the turtle to move on the screen and draw lines to create the desired shape.

6.1 Turtle graphics

Turtle graphics is a method of programming “vector” graphics using a relative cursor upon a Cartesian plane

- Python has a built-in module that supports turtle graphics called the “turtle” module. Importing this module gives you access to all the turtle graphics functions you will need to draw vector graphics on the screen.
- In turtle graphics you control a cursor, also known as a “turtle”. It has the following properties

- A position in 2D space
- An orientation, or direction.
- A pen that can lay up, down, width, or color on the canvas

6.2 Moving your turtle

You can cause your turtle to move to any coordinate on the screen by using the turtle.

You can move your turtle forward or backward by value x in pixel, using the following command:

 **forward (x)**

And you can have your turtle turn with angle x in clockwise positive value or in anti-clockwise negative value, by using the following command:

 **left (x)**


You can tell the turtle to stop drawing when moving by using the command:


 **penup ()**

You can tell it to start drawing again when moving by using the command:

 **pendown ()**

There are many other commands when drawing with *turtle* graphics:

 **clear ()** Clear the screen, leave the turtle position and orientation unchanged.

 **reset ()** Clear screen, reset turtle to initial configuration.

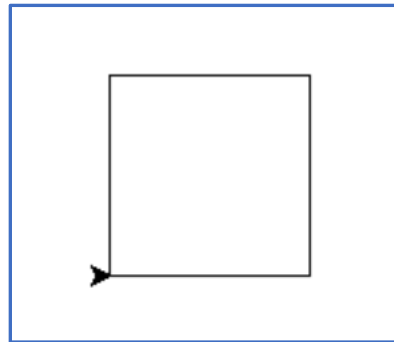
- ✚ **isvisible ()** Show the turtle.
- ✚ **hideturtle ()** Hide the turtle.
- ✚ **width (x)** Set the line thickness.
- ✚ **color ("x")** Set the line color.
- ✚ **for i in range(x):** "for" loop will repeat these function x times.

6.3 Examples using turtle

1- Draw a square using the python turtle in two different methods

- **First method (with series commands)**

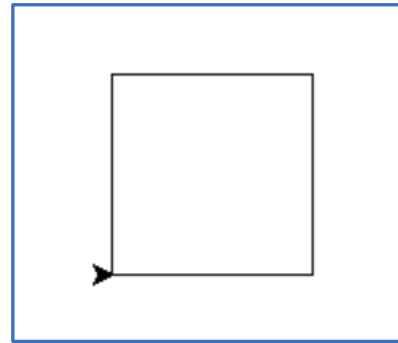
```
File Edit Format Run Options Window
from turtle import*
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
```



✚ التقدم للامام بمقدار 100 بكسل ثم الانعطاف بزاوية 90 درجة يمينا مع عقارب الساعة.
✚ تكرار ما سبق 4 مرات.

- **Second method (with "for" loop)**

```
File Edit Format Run Options Window
from turtle import*
for i in range(4):
    forward(100)
    left(90)
```



بما أن أضلاع المربع هي أربعة أضلاع استخدمنا أمر التكرار **for** وحددنا عدد مرات التكرار بـ 4 مرات.

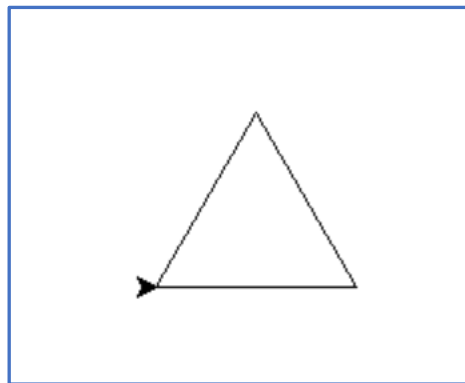
نختار أمر تقدم السلحفاة بمقدار 100 بكسل.

زاوية الانعطاف ستكون 90 درجة يمين (مع عقارب الساعة)

2- Draw a triangle using the python turtle

File Edit Format Run Options Window

```
from turtle import *
clear()
for i in range(3):
    forward(100)
    left(360/3)
```



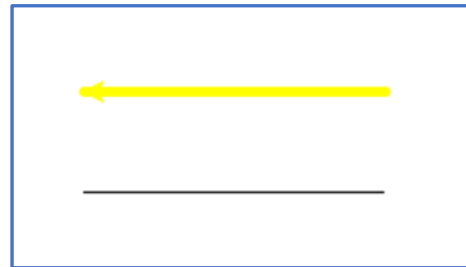
بما أن أضلاع المثلث هي ثلاثة أضلاع استخدمنا أمر التكرار **for** وحددنا عدد مرات التكرار بـ 3 مرات.

نختار أمر تقدم السلحفاة بمقدار 100 بكسل.

زاوية الانعطاف ستكون $120 = 360/3$ درجة يمين (مع عقارب الساعة)

3- Draw two parallel lines

```
from turtle import *
forward(150)
penup()
left(90)
forward(50)
left(90)
pendown()
color("yellow")
width(5)
forward(150)
```



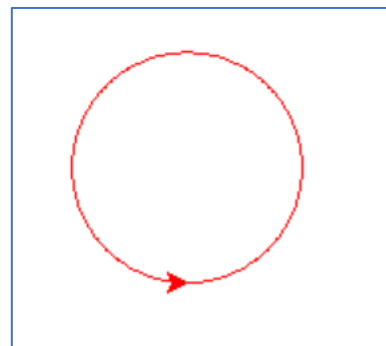
كل مستقيم يكون طوله 150 بكسل وزاوية الانعطاف تكون بمقدار 90 درجة وفق الاتجاه.

استخدمنا امر رفع القلم وإنزاله للتباعد بين المستقيمتين وكان التباعد بمقدار 50 بكسل.

تم تلوين كل مستقيم بلون مختلف وسماكة مختلفة لتمييزه عن المستقيم الآخر.

4- Draw a circle with python turtle

```
from turtle import *
color("red")
for i in range(360):
    forward(1)
    left(1)
```

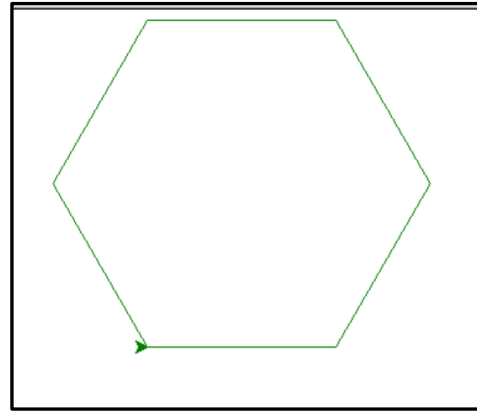


رسم الشكل الواضح أمامنا يعتمد على استخدام أمر يقوم بالتكرار 360 مرة لتأخذ شكل الدائرة بطول واحد بكسل وزاوية واحد درجة.

لا تظهر الزاوية ولا الانعطافات بسبب صغر الرسم ولكن عند التكبير في مقدار التحرك تظهر بشكل أوضح.

5- Drawing a Hexagonal shape

```
from turtle import*
color("green")
for i in range(6):
    forward(150)
    left(360/6)
```



✚ رسم السداسي يعتمد على الالتفاف بزاوية 60 درجة، والسداسي زواياه متساوية لذلك استخدمنا أمر التكرار.

✚ بما أن أضلاع السداسي هي ستة أضلاع استخدمنا أمر التكرار **for** وحددنا عدد مرات التكرار بـ 6 مرات.

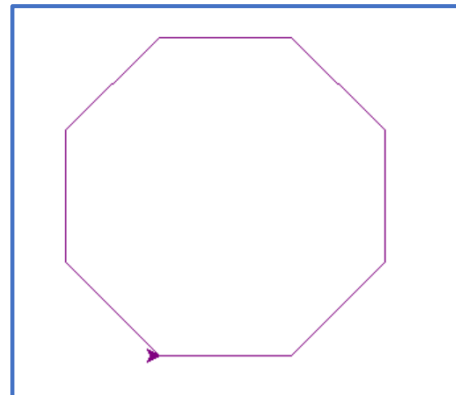
✚ تم تغيير لون خط السلحفاة إلى اللون الأصفر.

✚ حددنا أمر تقدم السلحفاة بمقدار 150 بكسل.

✚ زاوية الانعطاف ستكون $60 = 360/6$ درجة يمين (مع عقارب الساعة).

6- Drawing an octal form

```
from turtle import*
color("purple")
for i in range(8):
    forward(100)
    left(45)
    isvisible()
```



✚ رسم الثماني يعتمد على الالتفاف بزاوية 45 درجة، والثماني زواياه متساوية لذلك استخدمنا أمر التكرار.

✚ بما أن أضلاع الثماني هي ثمانية أضلاع استخدمنا أمر التكرار **for** وحددنا عدد مرات التكرار بـ 8 مرات.

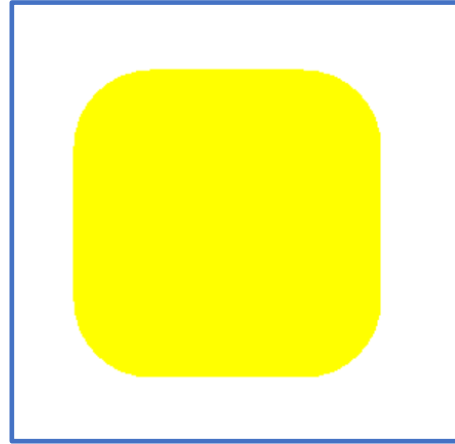
✚ تم تغيير لون خط السلحفاة إلى اللون البنفسجي **Purple**.

حددنا امر تقدم السلحفاة بمقدار 150 بكسل.

زاوية الانعطاف ستكون 45 درجة يمين (مع عقارب الساعة).

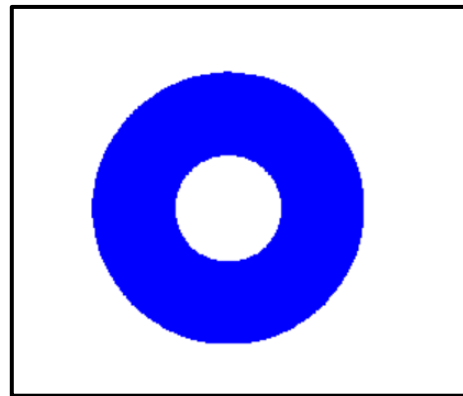
7- Drawing an square form in definite width (fill shape)

```
from turtle import *
color("yellow")
width(100)
for i in range(4):
    forward(100)
    left(90)
```

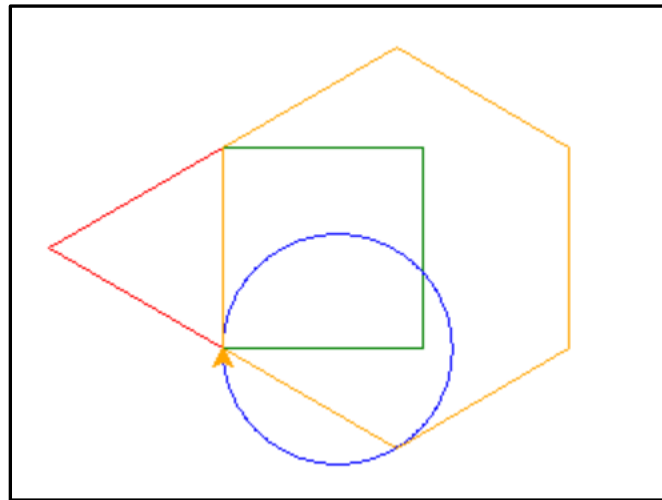


8- Drawing an circle form in definite width

```
from turtle import *
for i in range(360):
    color("blue")
    width(50)
    forward(1)
    left(1)
```



 **Exercise1:** Show how to draw the shape below using turtle python commands

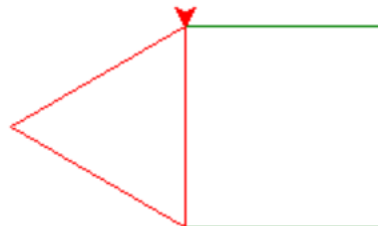


Solution:

1-

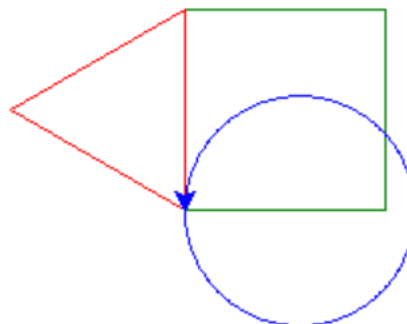
```
from turtle import *
right(90)
for i in range(4):
    color("green")
    forward(100)
    left(90)

for i in range(3):
    color("red")
    forward(100)
    left(-120)
```



2-

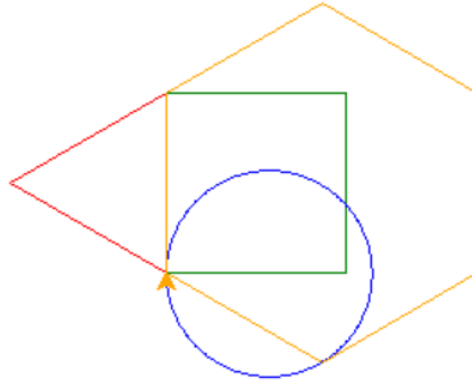
```
forward(100)
for i in range(360):
    color("blue")
    forward(1)
    left(1)
```



3-

```
left(180)

for i in range(6):
    color("orange")
    forward(100)
    right(60)
```



6.4 Function in python turtle

In Python, a **function** is a named sequence of statements that belong together. Their primary purpose is to help us organize programs into chunks that match how we think about the problem.

The syntax for a **function definition** is:

```
def name( parameters ):
    Statements
```

We can make up any names we want for the functions we create, except that we can't use a name that is a Python keyword, and the names must follow the rules for legal identifiers.

Defining a new function does not make the function run. To do that we need a function call only. Once we've defined a function, we can call it as often as we like, and its statements will be executed each time we call it.

Example 1: To draw any polygon in turtle python, using function, you may define it by command:

def polygon (n):

```
from turtle import*
def polygon(n):
    for i in range(n):
        forward(50)
        left(360.0/n)
```

```
width(10)
color("blue")
polygon(5)
```

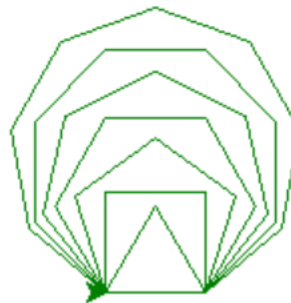


Example 2: draw polygons with different number of lines

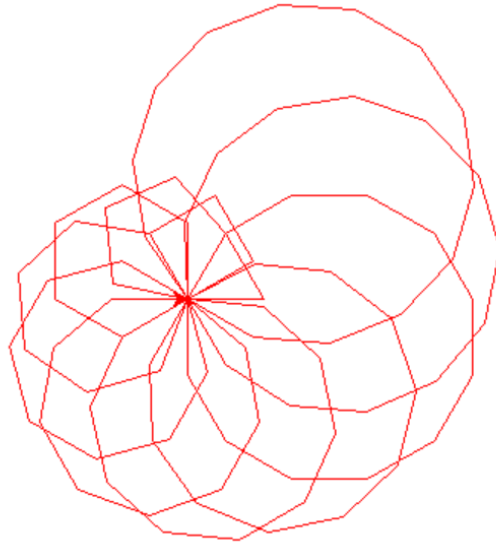
The number of polygons is the **difference** between the range numbers (**7 polygons**), starting with polygon in **three** ribs.

```
from turtle import*
def polygon(n):
    for i in range(n):
        forward(50)
        left(360.0/n)
```

```
color("green")
for i in range(3,10):
    polygon(i)
```



Example 3: Show how to draw the shape below using turtle python commands



Solution:

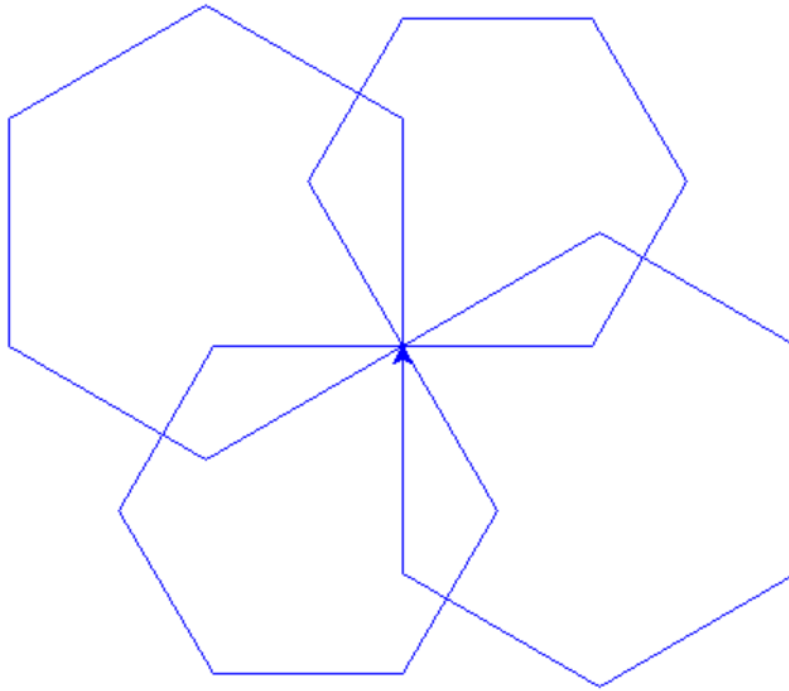
```
from turtle import*
def polygon(n):
    for i in range(n):
        forward(50)
        left(360.0/n)

color("red")
for i in range(3,15):
    polygon(i)
    left(30)
```

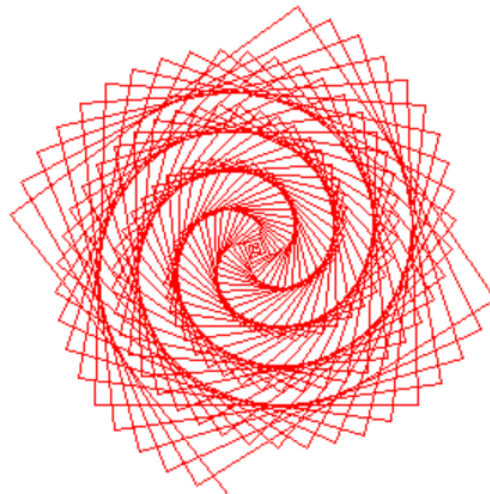
Example 4: Write a function that allows you to draw hexagons of any size you want, each time you call the function.

```
from turtle import*
def hexagon(size):
    for i in range(6):
        forward(size)
        left(60)

color("blue")
hexagon(100)
left(-90)
hexagon(120)
left(-90)
hexagon(100)
left(-90)
hexagon(120)
```



Example 5: Show how to draw the shape below (spiral) using turtle python commands:

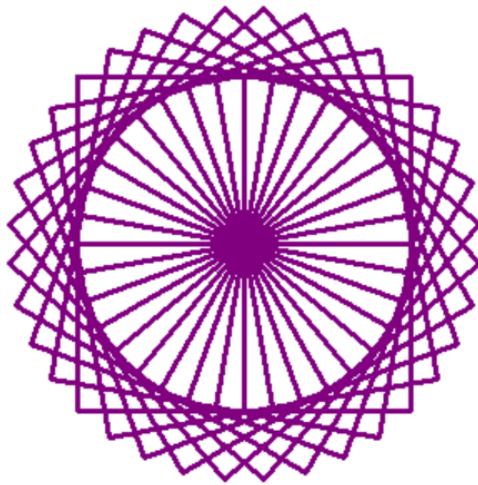


Solution:

```
from turtle import*  
color("red")  
for i in range(200):  
    forward(i)  
    left(92)  
hideturtle()
```

6.5 Drawing round and square shape

By using two repeating function loops. For every shape, use "for" loop. The first loop to draw the circle; when turn with angle (10), draw a square as shown in figure below:




```

from turtle import*
width(3)
color("purple")
for i in range(36):
    left(10)
    for i in range(4):
        forward(100)
        left(90)

hideturtle()

```

✚ أمر التكرار الأول يرسم دائرة وعند بداية التكرار تلتف بمقدار 10 درجات.

✚ بعد أول التفاف تدخل في حلقة تكرار جديدة لرسم المربع بمقدار 100 بكسل وزاوية قائمة 90 درجة.

✚ عند الانتهاء من رسم المربع يخرج أمر التكرار الثاني للدخول في تكرار جديد من الأوامر الرئيسية للتكرار، لتكرر نفسها الى ان تصل الى الانتهاء ورسم الشكل.

Example 6: Show how to draw a large circle with two circular shapes, one inside and one outside using turtle python commands

هذا التمرين عبارة عن دائرة تحوي على شكلين دائريين واحد بداخل الدائرة والآخر بالخارج، وما يميزه هو تباعد الدائرتين والقياسات، ولاحظ ان الكود البرمجي هو تجميع للأوامر البرمجية السابقة التي تم دراستها.

```

from turtle import*

screensize(1000, 1000)
for i in range(4):
    forward(30)
    left(10)

for i in range(36):
    forward(10)
    left(10)
    color("green")
    for i in range(3): } Triangle
        forward(30)
        left(120)

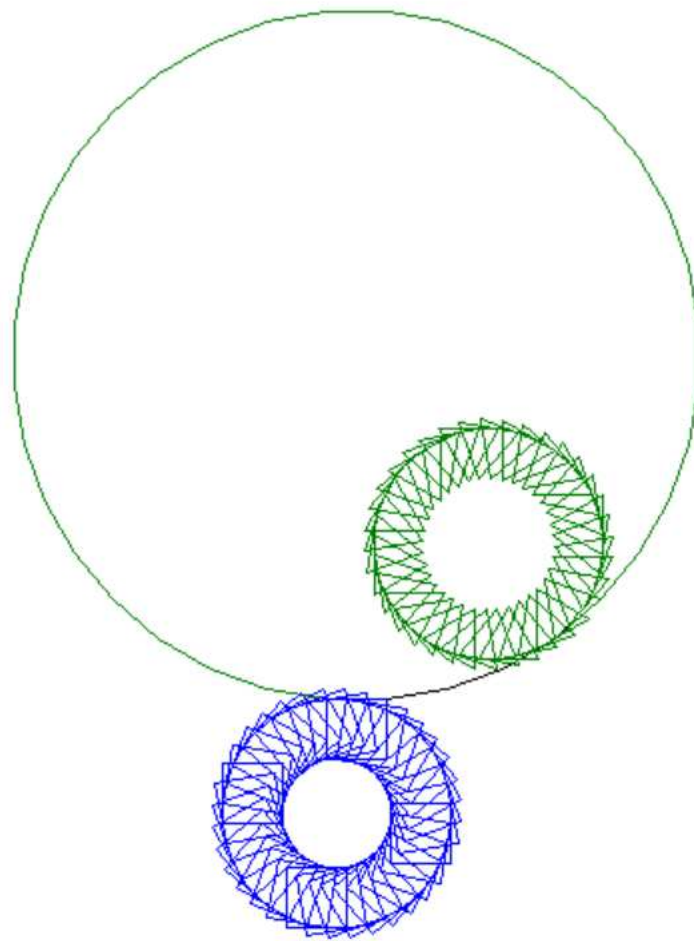
for i in range(23):
    forward(30)
    left(10)

for i in range(9):
    forward(30)
    left(10)

for i in range(36):
    forward(10)
    left(-10)
    color("blue")
    for i in range(4): } Square
        forward(30)
        left(-90)

hideturtle()

```



Exercise: Show the result for writing this code on python turtle

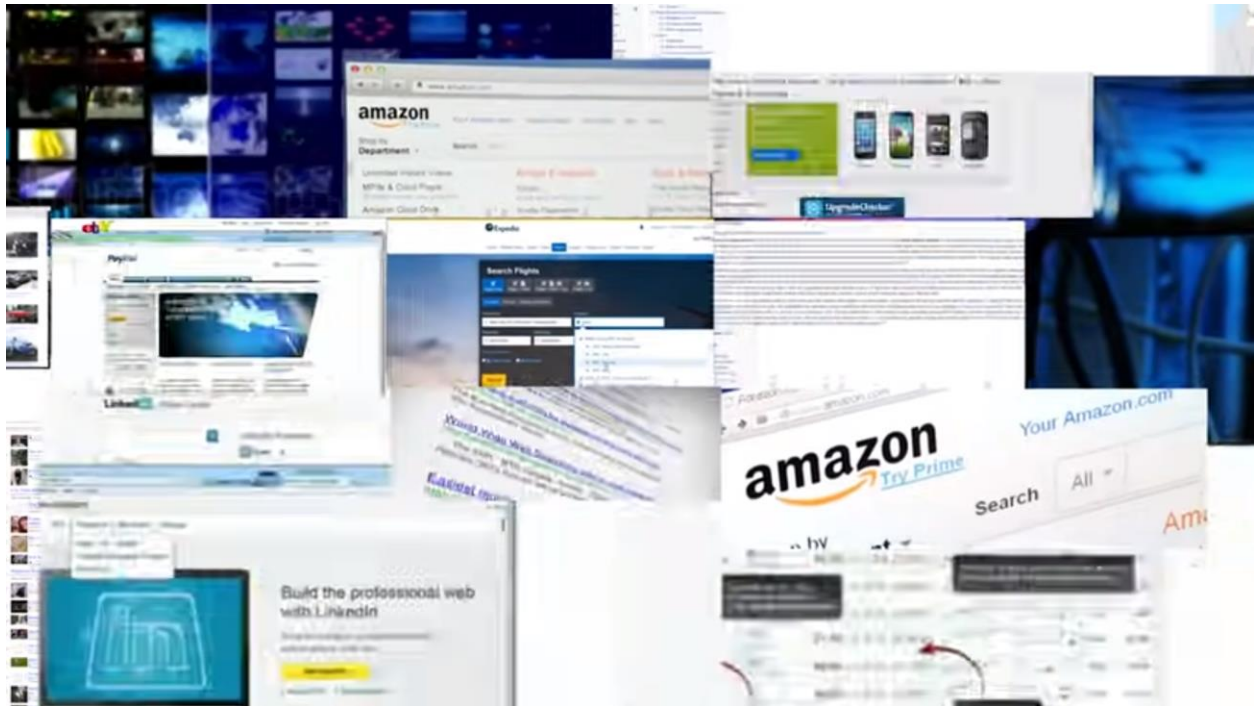
```
from turtle import*  
color("red")  
  
width(2)  
left(180)  
for i in range(3):  
    forward(150)  
    left(-120)  
  
forward(75)  
left(90)  
forward(30)  
left(90)  
forward(75)  
forward(-160)  
left(-50)  
forward(40)  
left(50)  
forward(110)  
left(50)  
forward(40)  
  
hideturtle()
```



CHAPTER 7

The internet

What is the internet? Is it waves moving in the air? Is it in the satellites up there? It is just there and we use it every day and sometimes nobody thinks exactly what is it or that one day somebody actually invented it.



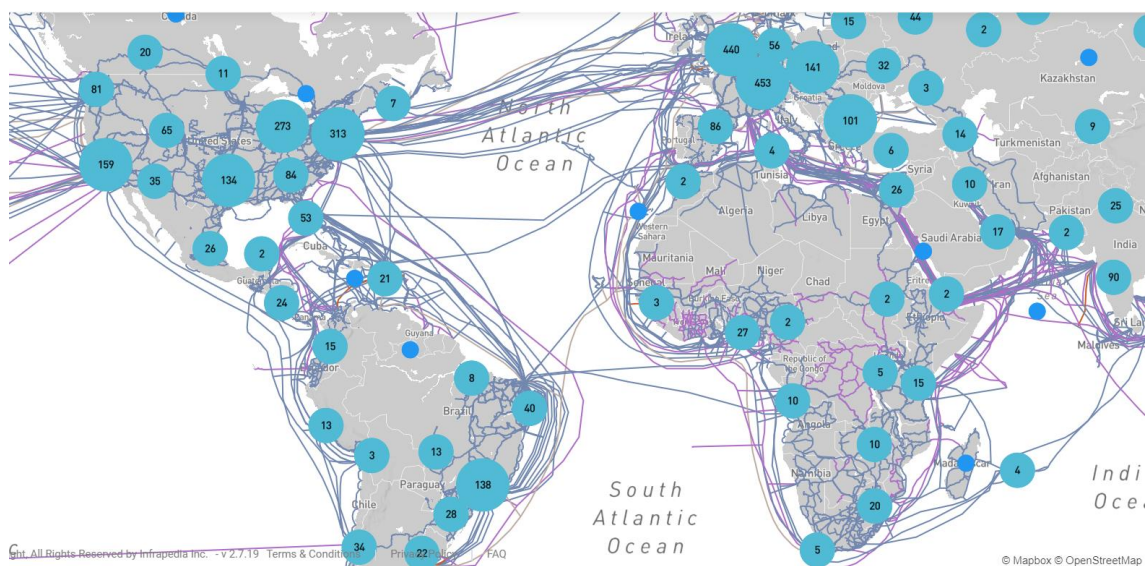
7.1 How it all started

The original “internet” was established in 1969, called [ARPANET](#), which connected computers between various institutions.



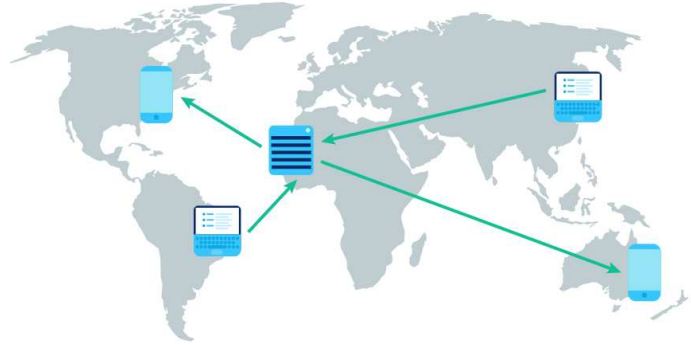
The ARPANET in December 1969

And now fast forward looking at this image which is visualization of the internet as it is today <https://www.infrapedia.com/app>, which shows how interconnected is the world today



7.2 What is the internet

The **Internet** is a global network of computing devices communicating with each other in some way, whether they're sending emails, downloading files, or sharing websites. In other words the internet is the network of all networks.

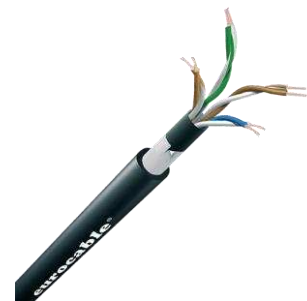


The internet is made up of a large number of independently operated networks. They are all motivated to ensure that there are end to end connectivity between all the nodes on the network. Any device on the network can communicate with any other device just as you can make phone calls to any other telephone in the world.

7.3 How does the internet works

7.3.1 Wires ,cables and wifi

How data is communicated between devices is by sending **binary** information. Earlier we learned that every form of data can be represented in the form of **ones** and **zeros**. But what is the actual physical signal that can be send over the wires. Nowadays bits are sent by one of three forms, electricity, light or radio waves.



Ones and zeros are sent as an electric signal that either is on or off. Electrical signals are sent through copper wires. However copper wires experience signal loss over long distances and they are not as fast as required to reach long distances around the world in short amount of time. So,



what moves faster than electricity, **light**.. Fiber optic cable are used to transfer internet for large distances in the form of a beam of light. This signal does not experience signal loss, this is how you can go hundreds of miles from one continent to another.

In 2008 a cable was near Alexandria which interrupted the internet for most of the middle east and India. So, the internet is actually a fragile physical system.

Wireless uses radio signals to send bits from one place to another. The receiving machine then transform the radio signal to a stream of bits. The radio signal makes the internet mobile but its doesn't travel so far so it's only for short range.

How do you think the physical methods to transfer bits may change in the future?

7.3.2 IP address and DNS

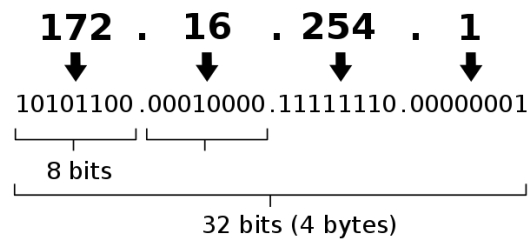
On its core, the internet works upon a set of protocols. A **protocol** is a set of rules and standards used for machines to be able to communicate.

All the different devices on the internet have a unique address. An address is just a number that is unique for each device on the internet. The addressing system for

computers on the internet is called **IP** or internet protocol. If a computer wants to send information to another computer, it must know its IP address. Traditional IP addresses is 32 bits long.

This protocol is called IPv4 and it can provide for up to 4 billion unique address. But as it turned out they are not enough. That why there will be a transition to a new protocol called IPv6, it's a 128 bit address.

IPv4 address in dotted-decimal notation



Most users don't deal directly with internet addresses. A system called **DNS** or **Domain Name System** associates addresses with names to ease the access for different locations on the web. A **DNS server** translates the name required to the its IP address.

7.3.3 Packets and Routing

Any data is broken down into pieces called **packets** and then its reassembled on the other side. Information travel in the form of packets and the packet choses the path based on the traffic. The way the information gets transferred from one device to another is not follow a fixed path.

Routers acts like traffic manages to keep the packets moving and make sure they arrive at their destination. The protocol used for file transfer is called TCP or Transmission Control Protocol. TCP acts as a guaranteed mail service, the protocol

sends acknowledgment for every packet received to make sure all the packets arrived.

Then internet is made up of many devices connected physically. These devices connect, communicate and collaborate with each other because of those agreed upon protocols for how the data is sent around the internet.

7.3.4 HTTP and HTML

When you want to access a website, you open your web browser type in the URL or the Uniform Resource Locator for the website you want to visit. In that moment your computer asks a server for a website. The way your computer and the server communicate is by using HTTP (hypertext transfer protocol).

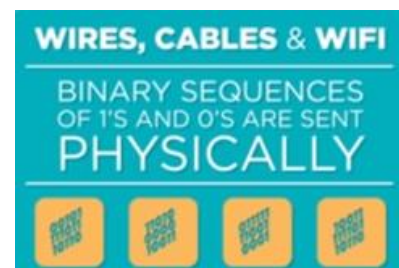
The server will send the Html code for the page you required to load. HTML stands for hypertext markup language, the language that tells how the webpage will look.

The internet is completely open, this makes it possible for hackers to spy on any personal information that you send over the web. When you see the little lock on top of your browser that's mean that the website uses **secure connection** to protect your information. When a website uses secure connection it provides what is called a **digital certificate** which ensures that you are dealing with right website. If the web site is not secure your browser will warn you.

7.3.5 The ingredients of the Internet

To summarize:

- HTTP and DNS manage the sending and receiving of HTML, media files, or anything on the web.
- What makes this possible under the hood are TCP/IP and router networks that break down and transport information in small packets.
- These packets themselves are up of binary, sequences of 1s and 0s that are physically sent through electric wires, fiber optic cables and wireless networks.



References

- Dierbach, Charles. *“Introduction to computer science using python: A computational problem-solving focus.”*
- Allen Downey *“Think Python: How to think like a computer scientist.”*
- Roth Jr, Charles H., Larry L. Kinney *“Fundamentals of Logic Design.”*
- “CS50’s Introduction to Computer Science” by Harvard university
- <https://www.khanacademy.org/>