

ITI1120 E

Professor: Dr. Sadiq Abbas

- Course Web page: **Blackboard Learn**
- Office: DMS 5140
 - Office hours: 14:00 – 16:00
 - TAs: posted on the BB

Labs and Lectures

We are at the maximum capacity. Thus

- Go to your section's lectures
- Go to your
 - Read all the announcements carefully

Course webpage and online material

Blackboard Learn:

- Supplemental course material (code from lectures, videos, some notes, labs ..)
- All communication via Blackboard Learn
- Discussion forum (use it and subscribe to all topics)
- Assignments and Assignment Submission
- Grades

To access Blackboard Learn go to

<https://uottawa.blackboard.com/>

Textbook

Main textbook (required):

“Practical Programming (2nd ed): An Introduction to Computer Science Using Python 3”

by Paul Gries, Jennifer Campbell, Jason Montojo

The eBook is available as an eBook for \$25.00.

<http://pragprog.com/book/gwpy2/practical-programming>

If you want a paper copy of the book order it on amazon.ca since the shipping and customs costs on pragprog.com are excessive.

https://www.amazon.ca/gp/cart/view.html/ref=gno_cart

Other recommended books

Free book: *"Think Python, How to Think Like a Computer Scientist"* by Allen Downey

Here are three Python 3 versions of the book:

- a pdf eBook version
<http://faculty.stedwards.edu/mikek/python/thinkpython.pdf>
- a "web" version here
<http://openbookproject.net/thinkcs/python/english3e/>
- an interactive version (some material here is still in Python 2 instead of Python 3)
<http://interactivepython.org/runestone/static/thinkcspy/toc.html>

Not free: *"Introduction to Computing Using Python: An Application Development Focus, 2nd Edition"*
by Ljubomir Perkovic

The eBook is available here for \$62.50

<http://ca.wiley.com/WileyCDA/countryConfirm?originalUrl=%2Fmarketbasket.cgi%3Fisbn%3D1118891058>

On line (interactive) references

On-line course developed by UofT for their equivalent class:

Learn to Program: The Fundamentals

<https://www.coursera.org/course/programming1>

Learn to Program: Crafting Quality Code

<https://www.coursera.org/course/programming2>

create an
account coursera

Labs

- Scheduled labs will be in groups of about 40+ students, with help available from a Teaching Assistant.
- You will be in a room with computers available for each student.
- All labs will involve programming in Python
- You should have been assigned to one of the available lab sections, and you must attend that session each week.
 - Your lab section assignment is available via BB – Assignment folder
- The **STE 0110** general lab is available at other times.

Labs

- Labs will start on Monday, 12th of Sept
- Labs are mandatory
- The first lab (11 Labs):
- each lab worth 0.8% percent (10 labs, 1 they can skip)
 - Assignment 0
 - How to use the Blackboard Learn to submit assignments
 - Bring your own laptop if you had problems installing python. TA may be able to help
 - Mark on labs 8%

Assignments

- There will be up to 5 programming assignments during the term.
- (5 assignments): 25% (each worth 5%)
- All assignments will involve programming in Python; and maybe some will involve written work.
- Learn how to submit assignments.
- As test, submit Assignment 0 (in 1st lab).

Quizzes

- There will be two quizzes during the class time
- - Quizzes: 7%
- Each 3.5%
- Date and time of the quizzes will be announced at least 1 week earlier

Midterm Examination

- Duration: 1h20m
- Date: Tuesday 1st. November class time – class room
- Put this in you calendar today
- written exam, closed-book
- Weight 20%

Final Examination

- To be scheduled by the University Registrar
 - The exam date and time will be scheduled during the exam period
 - Check the university web site later for the date, time, and location.
- Written exam, closed-book.
- Weight 40%

Determination of Final Grade

- Labs (11 labs): 8%
- Assignments (5 assignments): 25%
(each worth 5%)
- Quizzes (2): 7%
- Midterm: 20%
- Final examination: 40%

IMPORTANT NOTE

- One has to get at least 50% on all the tests together
- (quizzes + midterm + final), to pass the course.
- Obtaining less will:
- result in failing the course.

Things to do right away

1. Check your lab schedule
2. Familiarize yourself with Blackboard Learn System
3. Get the textbook
4. If you want to work on your own computer, install **Python 3.4** as per instructions given in my email.
5. Choose the right version according to your laptop system 32 bit or 64 bit
6. Find out how to start **IDLE**, write your first python program. run it, save it, close it, open it again (in lab, at home ...)

Course Policies

- Missed/late assignments: mark of zero.
 - One assignment may be exempted only if a the University of Ottawa Health Services certificate is given to the professor within one week of the due date.
 - In that case, the weight of missed assignment will be transferred to the final.
- Absence from midterm / final examination: see Engineering faculty regulations.
 - Illness requires The University of Ottawa Health Services certificate
 - Travel, employment, misreading the timetable are not valid excuses

Plagiarism

To reduce the need to worry about plagiarism, follow this rule of thumb:

Never look at any other person/group's assignment code, or have their code in your possession, in any portion or form whatsoever. Also never share your assignment code with other students.

Read the following three pages in detail

Academic Integrity

- What is academic fraud?
 - Misrepresenting someone else's work as your own:
 - Failure to cite sources, including the internet and discussions.
 - Use of the words of someone else without quotation marks or other highlighting.
 - Falsified lab data or citations.
 - Violation of examination regulations.
 - Tampering with academic evaluations.
 - Helping another student do any of the above.

Policy on plagiarism

Plagiarism is a kind of fraud: passing off someone else's work or ideas as your own in order to get a higher mark. Plagiarism is treated very seriously. The assignments you hand in must be your own and must not contain anyone else's ideas. Refer to the University of Ottawa's Policy on Academic Fraud for a more detailed description of plagiarism and sanctions. <http://web5.uottawa.ca/mcs-smc/academicintegrity/regulation.php>

Guidelines to help avoid plagiarism

You may discuss assignments with friends and classmates, but only up to a point: you may discuss and compare general approaches and also how to get around particular difficulties, but **you should not leave such a discussion with any written material**. You should not look at another student's solution to an assignment on paper or on the computer screen, even in draft form. The actual coding of your programs, analysis of results, writing of reports and answering assignment questions must be done individually.

Downloading code or any other material from the Internet, and submitting it as your own work without credit is also plagiarism. If you do talk with anyone about an assignment, please state this in your assignment and state the extent of your discussion. If you use another resource (such as textbooks, internet resources, etc) when solving your assignment, include the proper reference.

Note that it is also a serious offense to help someone commit plagiarism. Do not lend your assignment answers, printouts, reports or diskettes, and do not let others copy or read them. To protect yourself against people copying your work without your knowledge, retain all of your old printouts and draft notes until the assignments have been graded and returned to you. If you suspect that someone has stolen a printout or diskette, contact your instructor immediately.

Helping Each Other

Although you must not solve your assignments with the help of others, there are still many ways in which students can help each other. For instance, you can go over difficult lecture or lab material, work through exercises, or help each other understand an assignment handout. This sort of course collaboration can be done in study groups or through the discussion group.

The Blackboard Learn discussion forum can be used to discuss techniques and tools used in assignments, but the discussions should never mention or present any potential or partial solutions to assignment questions. You can consider creating a study group.

If in doubt about whether a question you are asking or answering is against these guidelines, ask your TA (teaching assistant) the question instead.

Detecting Plagiarism

Measures taken to detect plagiarism

1. TAs have been instructed to report any suspicious of plagiarism they find, when they mark assignments, to the professor.
2. Programs that you submit may be screened using plagiarism detection software that is very effective at detecting similarities. The professor will take appropriate measures, once plagiarism is detected on part or on the whole of an assignment. Note that copying or lending is considered to be equally serious offenses.

Reference

This document is a translation of the the French document prepared by Daniel Amyot and is based in part on "Policy on Plagiarism" by Dr. Amy Felty (Septembre 2002).

Faculty Eng. Regulations

- **Mandatory withdrawal**, if **second failure** in a mandatory course
- Academic standing is evaluated after students have attempted 24 course credits
 - **Mandatory withdrawal**, if **CGPA** is below 3.5
 - **Probation**, if $3.5 \leq \text{CGPA} < 4.5$
- Note that the **passing grade** for first year courses is **D (2)**, whereas 3.5 is between D+ and C, and 4.5 is between C and **C+** (65-69%)
- <http://www.uottawa.ca/about/policies-and-regulations>

Faculty regulations (2016)

- **Mandatory withdrawal**, if **second failure** in a mandatory course
- Academic standing is evaluated after students have attempted 24 course credits
 - **Mandatory withdrawal**, if **CGPA** is below 3
 - **Probation**, if $3 \leq \text{CGPA} < \underline{5}$
- **5** corresponds to **C+** (65-69%)

General Suggestions

- Based on problems identified by the the Committee on Academic Standing
 - Stay engaged: attend classes, tutorials, etc.
 - Understand academic regulations
 - Do not take too many courses,
 - Consider dropping a course in which you have low average
 - Take university seriously
 - Do not work too many hours (in part-time jobs)
 - Not making connect between mathematics and science to engineering and computer science applications

Suggestions for ITI1120

- Do not fall behind
- Program as much as possible (assignments, labs ...)
- After class, run the code we did in class. See if you understand why it does what it does. Play with it.
- Consider bringing the laptop to lectures and type as I do (if you are fast typer and can multitask)
- When programming, do not be afraid to make mistakes (make sure each small part of program is correct before moving to next part)
- Make use of office hours

Mentoring Centre



- Designed to **compliment** tutorials and office hours.
- Designed to teach **study skills** in the context certain courses.
- Helps students **meet other students** in the courses.
- **Workshops** are going to be provided on:
 - Time Management
 - Stress and Anxiety

Workshop: Introduction to Programming Using the Turtle Graphics Python Library

When: Saturday Sept 10, 10am-1pm in STE B0138

Intended audience: students from ITI120 and IT1520 who are absolute beginner programmers.

Covers: Gentle introduction to programming concepts such as if statements, for loops and a bit of graphics.

The workshop will be bilingual.

Bring your own laptop !!!

This course is about **computational thinking**, not about learning a programming language.

Python is a programming language that is easy to learn and very powerful.

- Used in many universities for introductory courses.
- Main language used for web programming at Google
- Widely used in scientific computation, for instance at NASA, by mathematicians and physicists
- Available on embedded platforms, for instance Nokia mobile phones
- Large portions of games (such as Civilization IV) are written in Python

Once you learnt programming in one language, it is relatively easy to learn another language, such as C++ or Java.



Python programming language was developed by at the end of 80s by Dutch Programmer **Guido van Rossum**

In the Python community, he is known as a "Benevolent Dictator For Life", since he continues to oversee the Python development process.

Python is named after British sketch comedy from '70s **Monty Python's Flying Circus**

Python, is **free and open-source** software and has a community-based development model.

Python 2 vs. Python 3

There are currently two major versions of Python in use: Python 2 and Python 3.

Python 2, the latest version 2.7 released in 2000

Python 3, a new version designed to rectify certain fundamental design flaws in the language. Unfortunately it is not backwards compatible

We will use Python 3.

More here for those interested:

<https://wiki.python.org/moin/Python2orPython3>

Why are you here?

“Every scientist and engineer must know some programming. It is part of basic education, like calculus, linear algebra, introductory physics and chemistry, or English”

Alan Perlis 1961

And more a modern version:

<https://www.youtube.com/watch?v=nKlu9yen5nc>

■ Why are you here?

Computer science is not computer programming. We teach programming to teach **computational thinking**:

- Solving problems (with a computer).
- Thinking on multiple levels of abstraction.
- Decompose a problem into smaller problems.
- A way of human thinking (**not** “thinking like a computer”)
- Thinking about recipes (**algorithms**).
- 30 years ago the solution to a problem in science or engineering was usually a formula. Today it is usually an algorithm (DNA, proteins, chemical reactions, factory planning, logistics).

What is a program?

A program is **a sequence of instructions** that solves some problem (or achieves some effect).

For instance: Call your friend on the phone and give her instructions to find your favorite café. Or explain how to bake a cake.

Instructions are operations that the computer can already perform.

But we can define **new instructions** and raise the **level of abstraction!**

A program implements an **algorithm** (a recipe for solving a problem).

Abstraction of Montreal metro system



What is debugging?

A **bug** is a mistake in a program. Debugging means to find the mistake and to fix it.

Computer programs are very complex systems. Debugging is similar to an experimental science: You experiment, form hypotheses, and verify them by modifying your program.

Kinds of errors:

- **Syntax error.** Python cannot understand your program, and refuses to execute it.
- **Runtime error.** When executing your program (at runtime), your program suddenly terminates with an error message.
- **Semantic (logical) error.** Your program runs without error messages, but does not do what it is supposed to do.

Why is programming fun?

Programming is a creative process.

A single person can actually build a software system of the complexity of the space shuttle hardware. Nothing similar is true in any other discipline.

There is a large and active **open-source community**: people who write software in their free time for fun, and distribute it for free on the internet. For virtually any application there is code available that you can download and modify freely.

Formal and natural languages

- **Natural languages**: languages that people speak. They were **not designed** by people; they evolved naturally. (**ambiguous**: we use contextual clues and other info to deal with this. Full of metaphors and idioms)
- **Formal languages**: **designed** by people for specific applications. Have strict **syntax rules**.(tokens & struct.)

$3+3=6$ vs $3\$=$

H_2O vs $_2Zz$

(**not ambiguous** : each stmt has exactly one meaning regardless of context.)

- Programming languages are formal languages that have been designed to express computations.

What is a programming language?

High-level programming languages: Java, Python, C, C++, Perl

- **Enormous advantages:** much easier (less time, shorter, easier to read), more likely to be correct, portable.
- **Small disadvantage:** have to be translated before running (compiling or interpreting)
- Thus, almost all programs written in them

Low-level programming languages: machine and assembly

- **Big disadvantages:** only run on one kind of computer. Have to be rewritten to run on another
- Thus only used for a few special applications

Interpreting / Compiling

There are 2 ways to translate a program written in high level language (also called **source code**):

Interpreting: An **interpreter** is a program that reads a high-level program and does what it says.

Compiling: A **compiler** is a program that reads a high-level program and translates it all at once, before running any of the commands. It compiles a program first and then runs the compiled code later. The compiled code is called **executable**.

Representation of numbers in Python

Integers can be represented exactly. There is no limits to their size (other than the size of you memory).

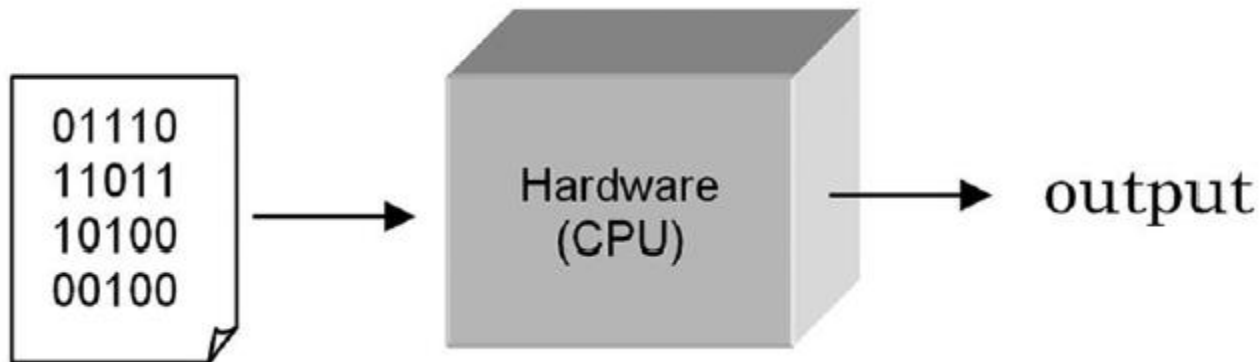
Floating points are approximation of real numbers. Python use. Python uses a double-precision standard format (IEEE 754) that can represent real numbers in a range of 10^{-308} to 10^{308} with 16 to 17 digits of precision.

Operations : -, **, *, /, %, //, +, -
<, <=, ==, !=, >, >=

== should be avoided in floats

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.

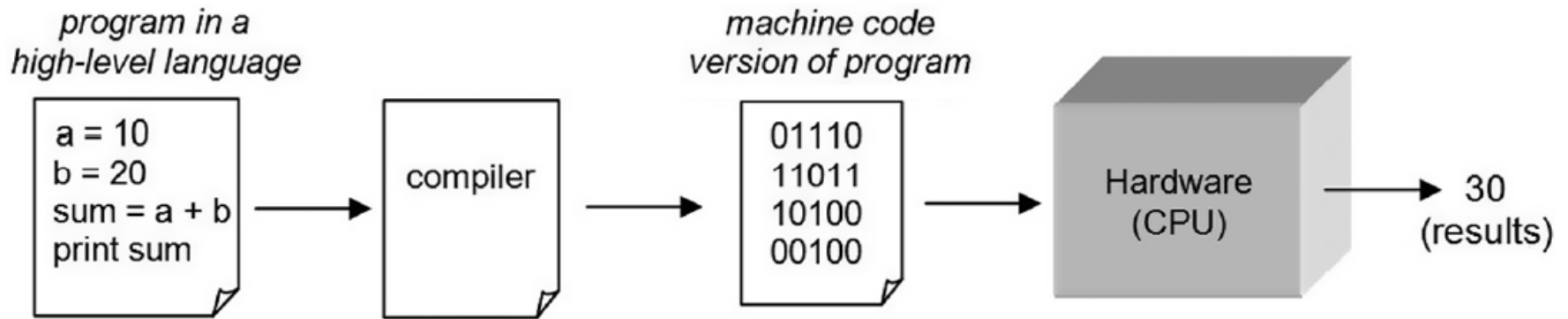


Writing programs at this “low level” is tedious and error-prone. Therefore, **most programs are written in a “high-level” programming language such as Python**. Since the instructions of such programs are not in machine code that a CPU can execute, a translator program must be used.

There are two fundamental types of translators:

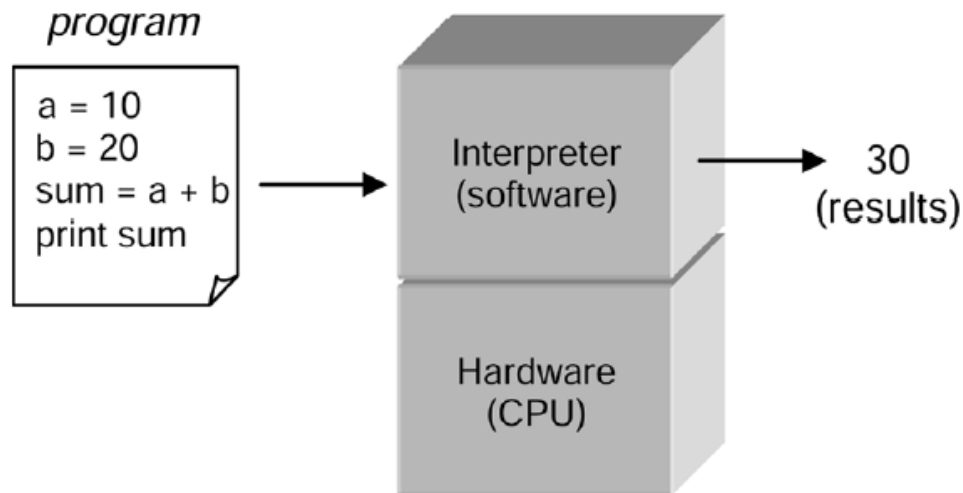
- **Compiler** software that **translates programs** into machine code to be executed by the CPU
- **Interpreter** software that **executes programs** in place of the CPU

Compiler



Compiled programs generally execute faster than interpreted programs.

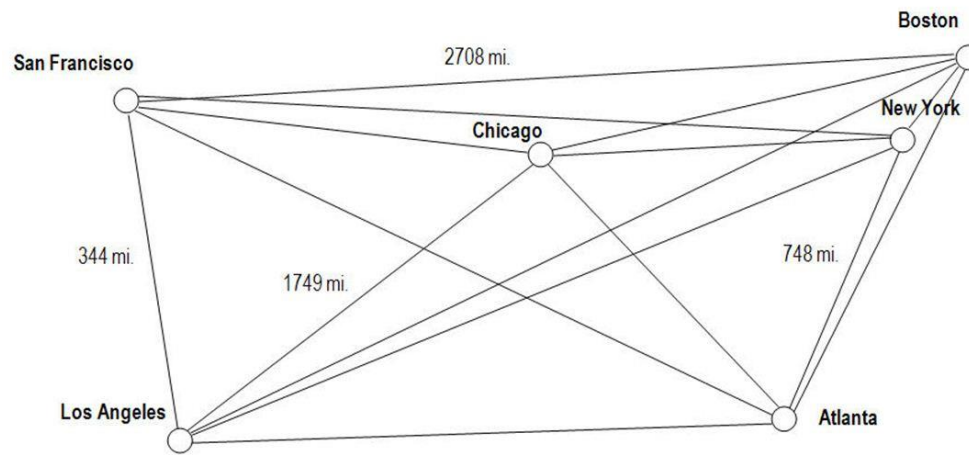
Interpreter



An interpreter can immediately execute instructions as they are entered. This is referred to as **interactive mode**. This is a very useful feature for program development. **Python**, as we shall see, is executed by an interpreter.

Why do we care about running time?

The Traveling Salesman Problem



A salesman needs to visit a set of cities. He wants to find the shortest route of travel, starting and ending at any city for a given set of cities. What route should he take?

The algorithm for solving this problem is a simple one. Determine the lengths of all possible routes that can be taken, and find the shortest one – a **brute force approach**. The computational issue, therefore, is for a given set of cities, how many possible routes are there to consider?

If we consider a route to be a specific sequence of names of cities, then **how many permutations of that list are there?**

New York, Boston, Chicago, San Francisco, Los Angeles, Atlanta

New York, Boston, Chicago, San Francisco, Atlanta, Los Angeles

New York, Boston, Chicago, Los Angeles, San Francisco, Atlanta

etc.

Mathematically, **the number of permutations for n entities is $n!$** (n factorial).

How big a number is that for various number of cities?

Below are the number of permutations (and thus the number of routes) there are for various numbers of cities:

Ten Cities 10! 3,628, 800 (over three million)

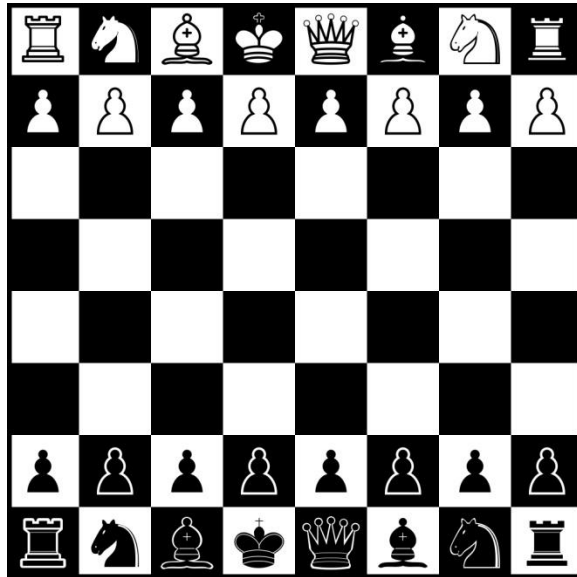
Twenty Cities 20! 2,432,902,008,176,640,000

Fifty Cities 50! over 10^{64}

If we assume that a computer could compute the routes of one million cities per second:

- for **twenty cities**, it would take **77,000 years**
- for **fifty cities**, it would take **longer than the age of the universe!**

The Game of Chess



When a computer plays chess against a human opponent, both have to “think ahead” to the possible outcomes of each move it may make.

Therefore, a brute force approach can also be used for a computer playing a game of chess of “looking ahead” at all the possible moves that can be made, each ending in a win, loss, or draw. It can then select the move each time leading to the most number of ways of winning.

(Chess masters, on the other hand, only think ahead a few moves, and “instinctively” know the value of each outcome.)

There are approximately 10^{120} possible chess games that can be played. This is related to the average number of look-ahead steps needed for deciding each move.

There are approximately,

10^{80} atoms in the observable universe

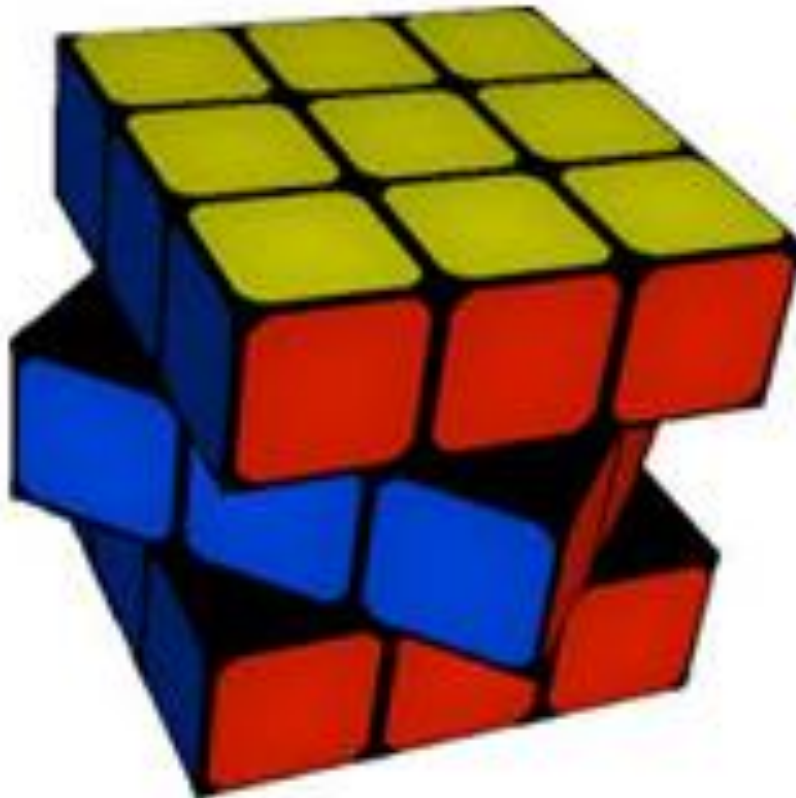
and an estimated

3×10^{90} grains of sand to fill the universe solid

Thus, there are *more possible chess games that can be played than grains of sand to fill the universe solid!*

Therefore, for problems such as this and the Traveling Salesman problem in which a brute-force approach is impractical to use, **clever and more efficient problem-solving methods must be discovered that find either an exact or an approximate solution** to the problem.

- **How many combinations you estimate in Rubik's Cube? GUESS A NUMBER !!!!!**



43,252,003,274,489,856,000

Rubik's Cube Combinations

Computer Algorithms



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** (that is, it eventually terminates).

The word “algorithm” is derived from the ninth-century Arab mathematician, **Al-Khwarizmi** who worked on “written processes to achieve some goal.”

Algorithms and Computers: A Perfect Match

Computer algorithms are central to computer science. They provide step-by-step methods of computation that a machine can carry out.

Having high-speed machines (computers) that can consistently follow a given set of instructions provides a reliable and effective means of realizing computation. However, **the computation that a given computer performs is only as good as the underlying algorithm used.**

Because computers can execute a large number of instructions very quickly and reliably without error, **algorithms and computers are a perfect match!**

Euclid's Algorithm

One of the Oldest Known Algorithms

Euclid's Algorithm is an algorithm for computing the greatest common denominator (GCD) of two given integers (A,B). It is one of the oldest numerical algorithms still in common use.

1. Assign A the value of the larger of the two values.
2. If $A = 0$ then $\text{GCD}(A,B)=B$, and we can stop.
3. If $B = 0$ then $\text{GCD}(A,B)=A$, and we can stop.
4. Divide A by B, call the remainder R.
5. assign A the value of B, assign B the value of R, and go to step 2.

Example Use

Finding the GCD of 18 and 20

1. **Assign A the value of the larger of the two values, and B the smaller.**

$A \leftarrow 20$ $B \leftarrow 18$

Example Use

Finding the GCD of 18 and 20

1. Assign A the value of the larger of the two values, and B the smaller.

$A \leftarrow 20$ $B \leftarrow 18$

2. **Divide A by B, call the remainder R.**

$A/B = 20 / 18 = 1$, with $R \leftarrow 2$

Example Use

Finding the GCD of 18 and 20 (**first time through**, **second time through**)

1. Assign A the value of the larger of the two values, and B the smaller.

$A \leftarrow 20$ $B \leftarrow 18$

2. Divide A by B, call the remainder R.

$A/B = 20 / 18 = 1$, $R \leftarrow 2$

3. If R is not 0, assign A the value of B, assign B the value of R, and go to step 2.

$A \leftarrow 18$, $B \leftarrow 2$.

Example Use


Finding the GCD of 18 and 20 (**first time through**, **second time through**)

1. Assign A the value of the larger of the two values, and B the smaller.

$A \leftarrow 20$ $B \leftarrow 18$

- 
2. **Divide A by B, call the remainder R.**

$A/B = 18 / 2 = 9, R \leftarrow 0$

- 
3. If R is not 0, assign A the value of B, assign B the value of R, and go to step 2.

$A \leftarrow 18, B \leftarrow 2.$

Example Use

Finding the GCD of 18 and 20 (**first time through**, **second time through**)

1. Assign A the value of the larger of the two values, and B the smaller.

$A \leftarrow 20$ $B \leftarrow 18$

2. Divide A by B, call the remainder R.

$A/B = 20 / 18 = 1$, with $R \leftarrow 2$

$A/B = 18 / 2 = 9$, with $R \leftarrow 0$

3. **If R is not 0, assign A the value of B, assign B the value of R, and go to step 2.**

R is 0. Therefore, proceed to step 4.

Example Use

Finding the GCD of 18 and 20 (**first time through**, **second time through**)

1. Assign A the value of the larger of the two values, and B the smaller.

$A \leftarrow 20$ $B \leftarrow 18$

2. Divide A by B, call the remainder R.

$A/B = 20 / 18 = 1$, with $R \leftarrow 2$

$A/B = 18 / 2 = 9$, with $R \leftarrow 0$

3. If R is not 0, assign A the value of B, assign B the value of R, and go to step 2.

$R = 2$. Therefore, $A \leftarrow 18$, $B \leftarrow 2$. Go to step 2.

R is 0. Therefore, proceed to step 4.

4. **The greatest common divisor is B.**

GCD is 2 (the value of B)

Animation of Euclid's Algorithm



1599

Information Theory

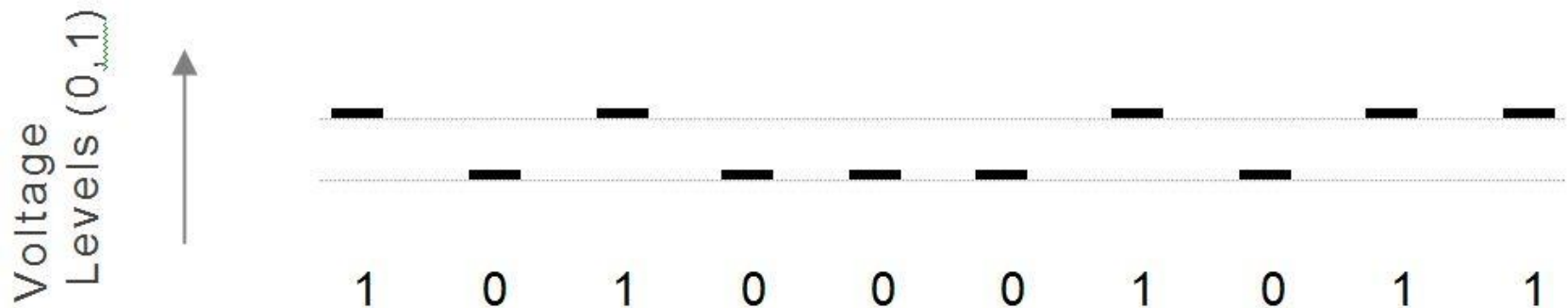


The **Fundamental Theorem of Information Science** of **Claude Shannon** (known as the “**Father of Information Theory**”) states that **all information can be represented by the use of only two symbols, e.g., 0 and 1.**

This is referred to as **binary representation.**

Binary Digitalization

In a binary representation, each digit can be one of only two possible values, similar to a light switch that can be either on or off. Computer hardware, therefore, is based on the use of simple electronic “on/off” switches called **transistors** that can be switched at essentially the speed of light.



The Binary Number System

For representing numbers, any base (radix) can be used. For example, in base 10, there are ten possible digits (0, 1, ..., 9), in which column values are a power of ten:

10,000,000	1,000,000	100,000	10,000	1,000	100	10	1
10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0
						9	9 = 99

For representing numbers in base 2, there are two possible digits (0, 1) in which column values are a power of two:

128	64	32	16	8	4	2	1								
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0								
<hr/>															
0	1	1	0	0	0	1	1								
0	+	64	+	32	+	0	+	0	+	0	+	2	+	1	= 99

Although values represented in base 2 are significantly longer than those in base 10, **binary representation is used in digital computing because of the resulting simplicity of hardware design**

Bits and Bytes

Each **binary** **digit** is referred to as a **bit**. A group of (usually) eight bits is called a **byte**. Converting a base ten number to base two involves the successive division of the number by 2.

Convert decimal number 13_{10} to a binary number:

$$13 / 2 = 6 \text{ Remainder } 1$$

$$6 / 2 = 3 \text{ Remainder } 0$$

$$3 / 2 = 1 \text{ Remainder } 1$$

$$1 / 2 = 0 \text{ Remainder } 1$$



1101_2

13_{10} is equal to 1101_2