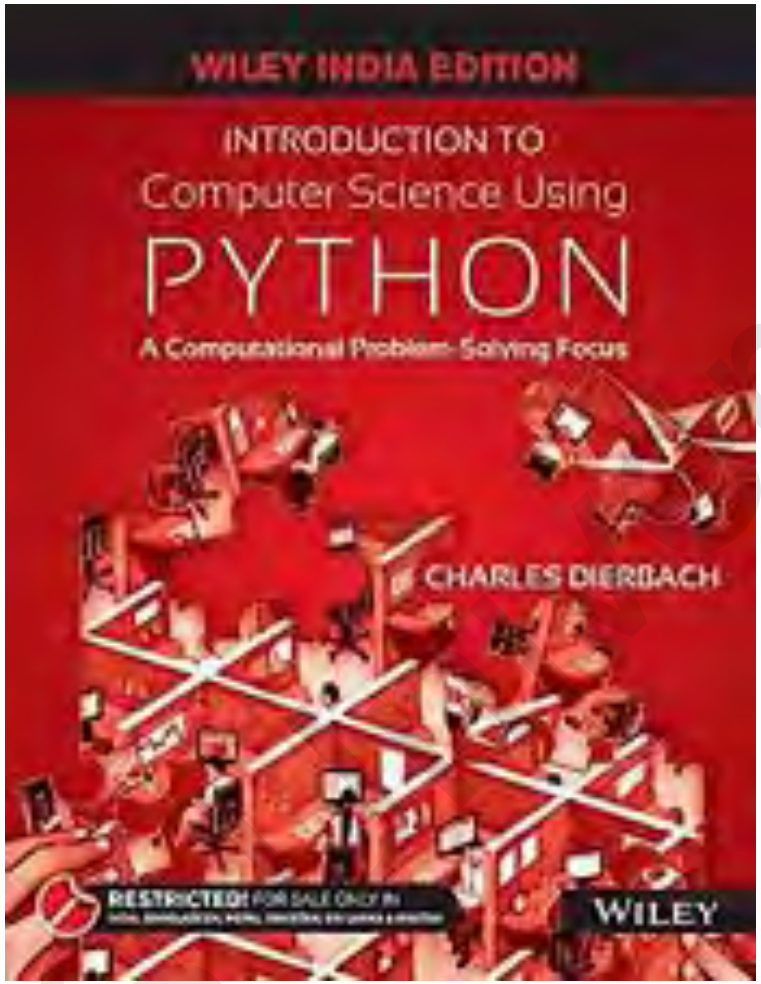
UE18CS101:
INTRODUCTION TO COMPUTING USING

Department of Computer Science and Engineering
PES UNIVERSITY

PES UNIVERSITY

TEXTBOOK



“Introduction to Computer Science Using Python: A Computational Problem-Solving Focus”, Charles Dierbach, John Wiley, 2012.

https://www.python.org/

PES UNIVERSITY

Course Contents

1. Introduction: Computation Problem Solving-Limits of Computational Problem Solving - Computer Algorithm - Computer Hardware - Digital Computer - Operating System- Limits of IC technology - Computer Software - Syntax, semantics and program translation.
2. Process of Computation Problem Solving: Introduction to Python Programming Language. Output function - variables, types and id operators and expressions. Control structures. Lists, Dictionaries, Sets, Tuples and Strings.
3. Functions: Definition, call, positional and keyword parameter. Default parameters, variable number of arguments. Modules - import mechanisms. Functional programming - map, filter, reduce, max, min. lambda function - list comprehension.
4. Object Oriented Programming: classes and objects - inheritance – polymorphism. Error handling & Exceptions - try, except and raise - exception propagation
5. File Processing: reading and writing files

Lecture 1

- Introduction- Computation Problem Solving - Limits of Computational Problem Solving
- Algorithms

Computer Software

What is computer software?

Computer software is a set of program instructions, including related data and documentation, that can be executed by computer. This can be in the form of instructions on paper, or in digital form.

While **system software** is intrinsic to a computer system, **application software** fulfills users' needs, such as a photo-editing program.

Fundamental Hardware Components

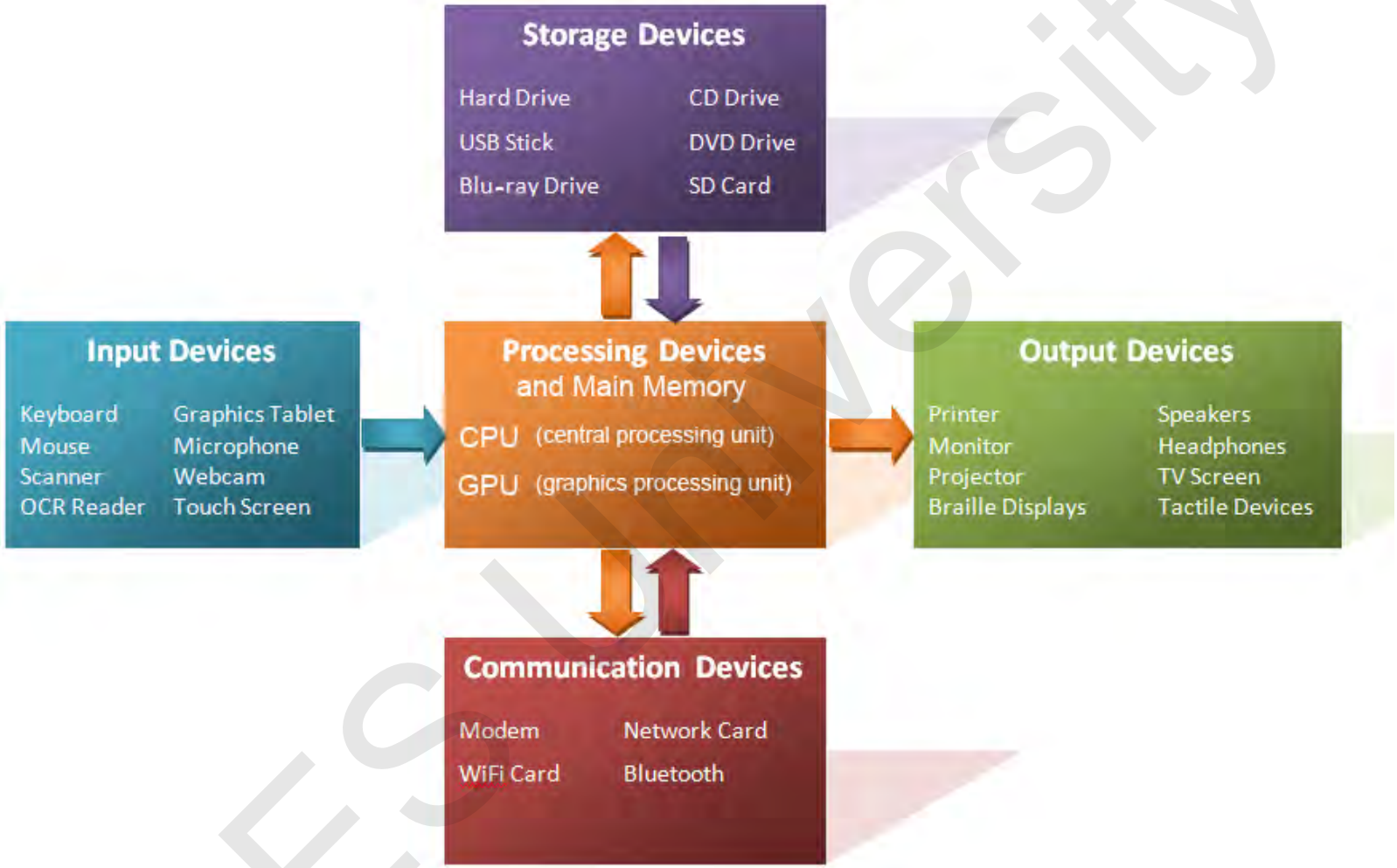
Central processing unit (CPU) – the "brain" of a computer system. Interprets and executes instructions.

Main memory – is where currently executing programs reside.
It is *volatile*, the contents are lost when the power is turned off.

Secondary memory – provides long-term storage of programs and data.
N*onvolatile*, the contents are retained when power is turned off. Can be magnetic (hard drive), optical (CD or DVD), or flash memory (USB drive).

Input/output devices – mouse, keyboard, monitor, printer, etc.

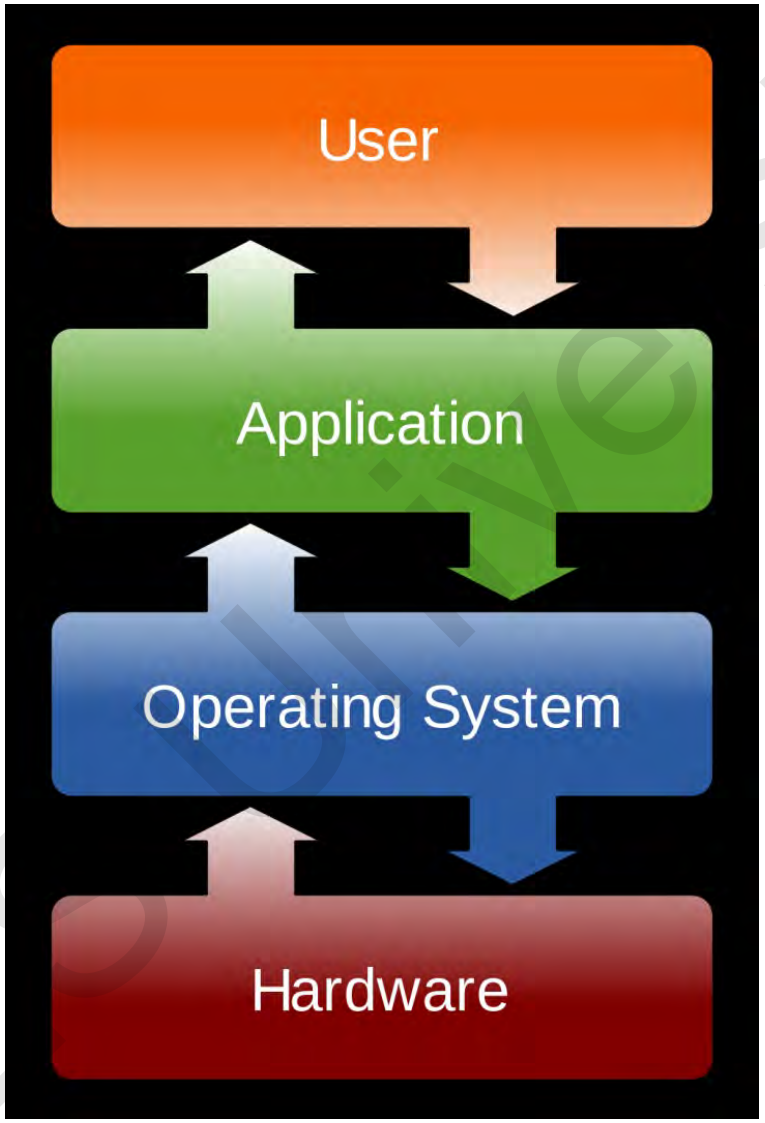
Buses – **transfer data between components within a computer system**. System bus (between CPU and main memory).



Operating Systems

An **operating system** is software that manages and interacts with the hardware resources of a computer.

Because an operating system is intrinsic to the operation of a computer, it is referred to as **system software**.



Operating systems also provide a particular **user interface**.

It is the operating system installed on a given computer that determines the **"look and feel"** of the user interface and how the user interacts with the system, and not the particular model computer.

Motivation

Computing technology has changed—and is continuing to change—the world. Essentially every aspect of life has been impacted by computing. Computing related fields in almost all areas of study are emerging.

Various Computational-Related Fields		
Computational Biology	Computational Medicine	Computational Journalism
Computational Chemistry	Computational Pharmacology	Digital Humanities
Computational Physics	Computational Economics	Computational Creativity
Computational Mathematics	Computational Textiles	Computational Music
Computational Materials Science	Computational Architecture	Computational Photography
Computer-Aided Design	Computational Social Science	Computational Advertising
Computer-Aided Manufacturing	Computational Psychology	Computational Intelligence

What is Computer Science?

Computer science is fundamentally about computational problem solving.

Programming and computers are only tools in the field of computing. The field has tremendous breadth and diversity. Areas of study include:

- Programming Language Design
- Systems Programming
- Computer Architecture
- Human–Computer Interaction
- Robotics
- Artificial Intelligence
- Software Engineering
- Database Management / Data Mining
- Computer Networks
- Computer Graphics
- Computer Simulation
- Information Security

Problems

- Computable
- Non-computable

Problems

- Deterministic

a **deterministic** *algorithm* is an *algorithm* which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states.

- Non-deterministic

a **nondeterministic** algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a **deterministic** algorithm. There are several ways an algorithm may behave differently from run to run.

The Limits of Computational Problem Solving

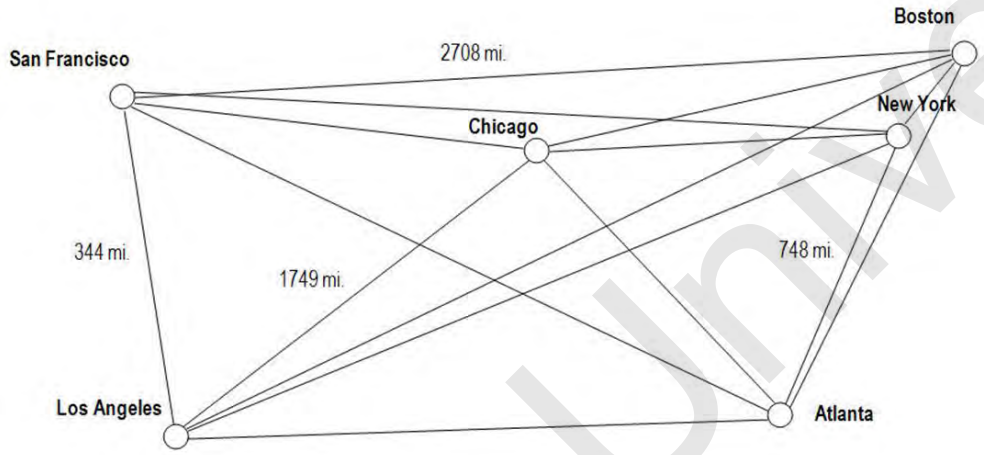
Once an algorithm for a given problem is developed or found, an important question is "**Can a solution to the problem be found in a reasonable amount of time**?"

"But aren't computers very fast, and getting faster all the time?"

Yes, but some problems require an amount of time to compute a solution that is astronomical compared to the capabilities of current computing devices.

A classic problem in computer science that demonstrates this is the **Traveling Salesman problem**.

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. That is, by using a **brute force approach**. **The computational issue, therefore, is for a given set of cities, how many possible routes are there?**

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, Loa 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?

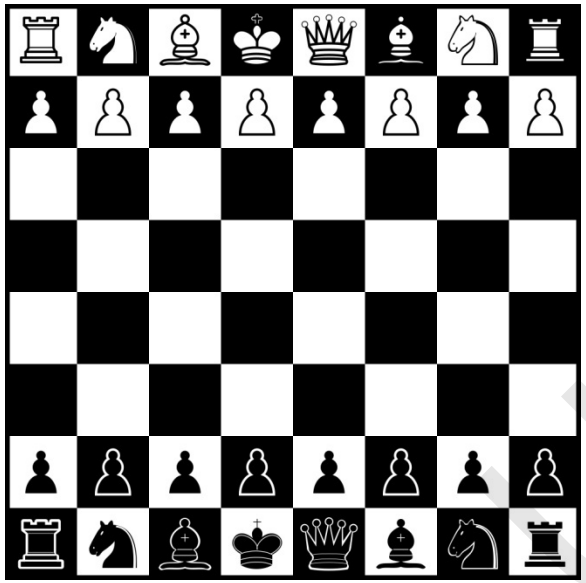
Below are the number of permutations (and this number of routes) there are for varies numbers of cities:

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.

A brute force approach can also be used for a computer playing a game of chess. A program can consider all the possible moves that can be made, each ending in a win, loss, or draw. It can then select the move that leads 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.

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

An algorithm for computing the greatest common denominator (GCD) of two given integers. It is one of the oldest numerical algorithms still in common use.

1. Assign M the value of the larger of the two values.

2. Divide M by N, call the remainder R.

3. If R is not 0, then assign M the value of N, assign N the value of R, and
   go to step 2.
4. The greatest common divisor is N.

Example Use

Finding the GCD of 18 and 20

1. Assign M the value of the larger of the two values, and N the smaller.
   M ← 20 N ← 18

Example Use

Finding the GCD of 18 and 20

1. Assign M the value of the larger of the two values, and N the smaller.
   M ← 20 N ← 18

2. Divide M by N, call the remainder R.

   M/N = 20 / 18 = 1, with R ← 2

Example Use

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

1. Assign M the value of the larger of the two values, and N the smaller.
   M ← 20 N ← 18

2. Divide M by N, call the remainder R.
   M/N = 20 / 18 = 1, with R ← 2
   M/N = 18 / 2 = 9, with R ← 0

3. If R is not 0, assign M the value of N, assign N the value of R, and go to step 2.
   R = 2. Therefore, M ← 18, N ← 2. Go to step 2.

Example Use

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

1. Assign M the value of the larger of the two values, and N the smaller.
   M ← 20 N ← 18

2. Divide M by N, call the remainder R.

   M/N = 20 / 18 = 1, with R ← 2

   M/N = 18 / 2 = 9, with R ← 0

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

   R = 2. Therefore, M ← 18, N ← 2. 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 M the value of the larger of the two values, and N the smaller.
   M ← 20 N ← 18

2. Divide M by N, call the remainder R.

   M/N = 20 / 18 = 1, with R ← 2

   M/N = 18 / 2 = 9, with R ← 0

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

   R = 2. Therefore, M ← 18, N ← 2. Go to step 2.

   R is 0. Therefore, proceed to step 4.

4. The greatest common divisor is N.

   GCD = N = 2

Animation of Euclid's Algorithm