

CSE 1001

Problem Solving and Programming

Course Objective

- **Develop** essential **skills** for a **logical thinking** to solve problems
- **Develop** essential **skills** in **programming** for solving problems using computers

Outcomes

On completion of the course, students will have the

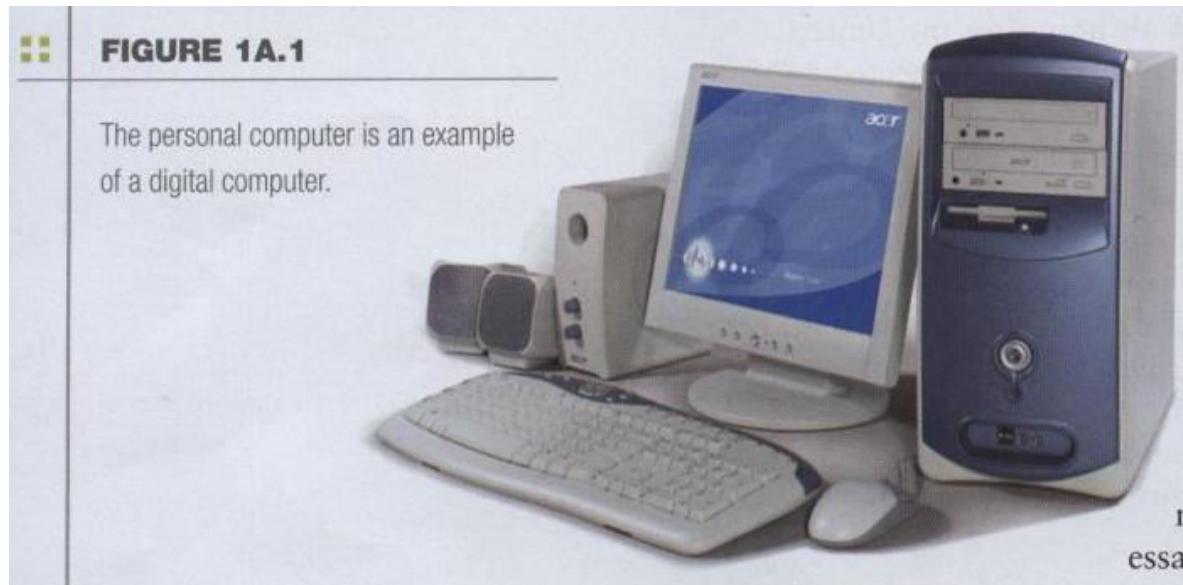
- ability to identify an appropriate approach to solve a problem
- ability to write a pseudo code for the identified strategy
- ability to translate the pseudocode into an executable program
- ability to validate the program for all the possible inputs

Points to Ponder

- Do not miss **any class, practice problem, and assessments**
- Be **ethical and professional** throughout the course
- **Unethical practices are punishable**

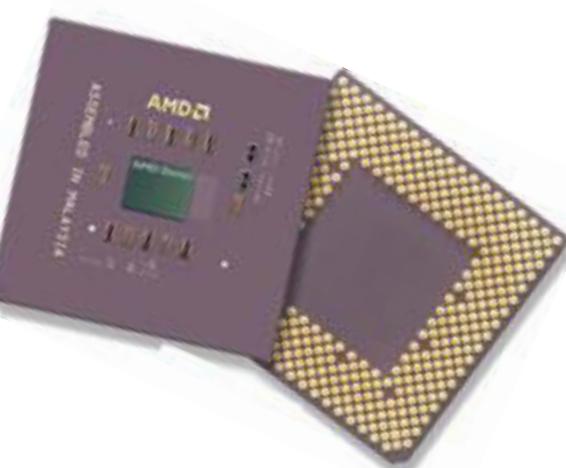
The Computer Defined

- Electronic device
- Converts data into information
- Modern computers are digital
 - Two digits combine to make data (0, 1)



A computer is:

- An electronic machine that can be programmed to accept data (*input*), and process it into useful information (*output*). Data is put in secondary storage (*storage*) for safekeeping or later use.
- The *processing* of input into output is directed by the software, but performed by the hardware.



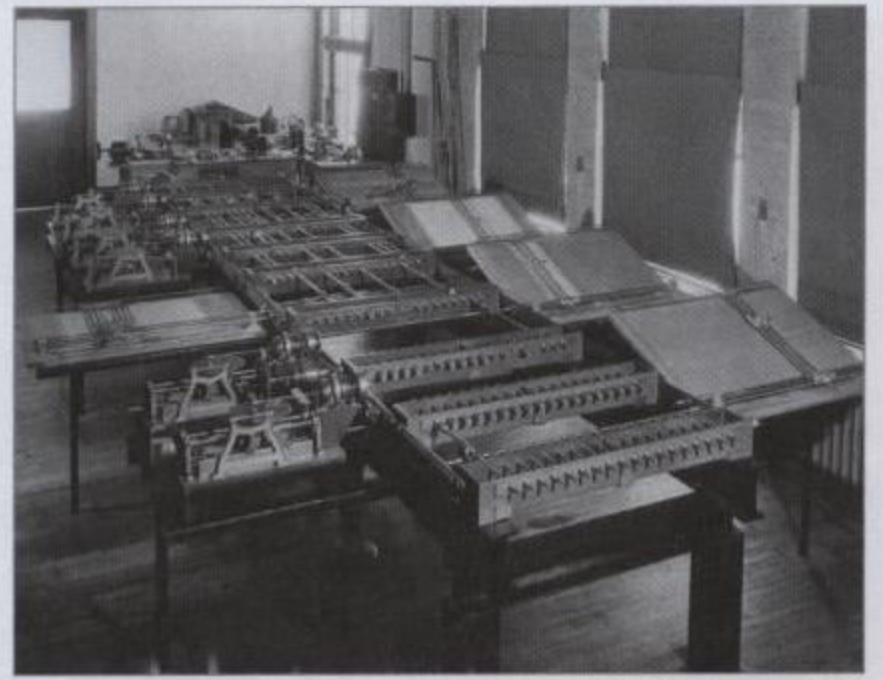
The Computer Defined

- Older computers were analog
 - A range of values made data



FIGURE 1A.2

This early analog computer, created by Vannevar Bush in the late 1920s, was called a “differential analyzer.” It used electric motors, gears, and other moving parts to solve equations.



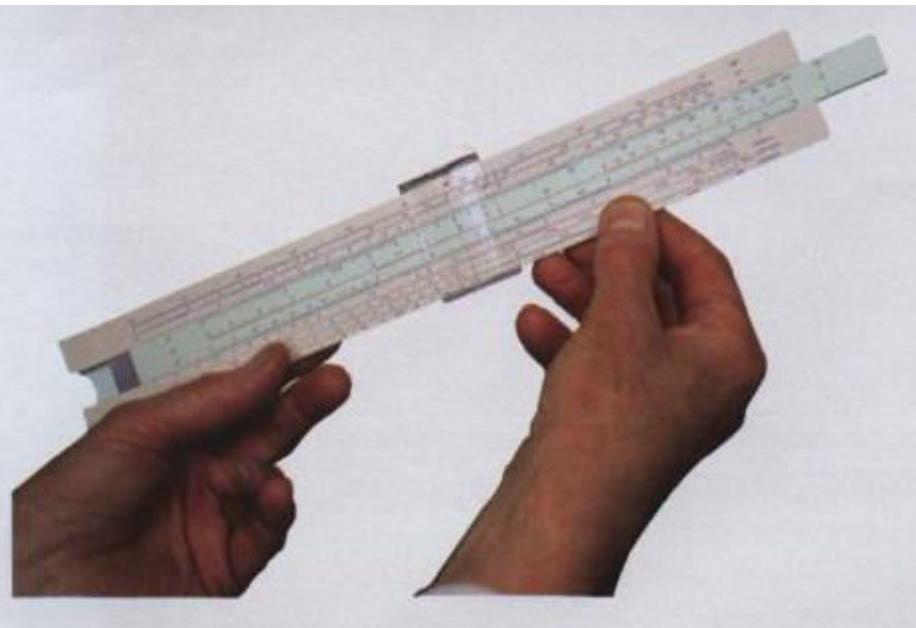
The Computer Defined

- Older computers were analog
 - A more manageable type -- the old-fashioned slide rule



FIGURE 1A.3

Although analog computers have largely been forgotten, many of today's computer scientists grew up using slide rules—a simple kind of analog computer.



Computers for Individual Use

- Desktop computers
 - Different design types

FIGURE 1A.7

This desktop PC follows the traditional design, with the monitor stacked on top of the system unit.

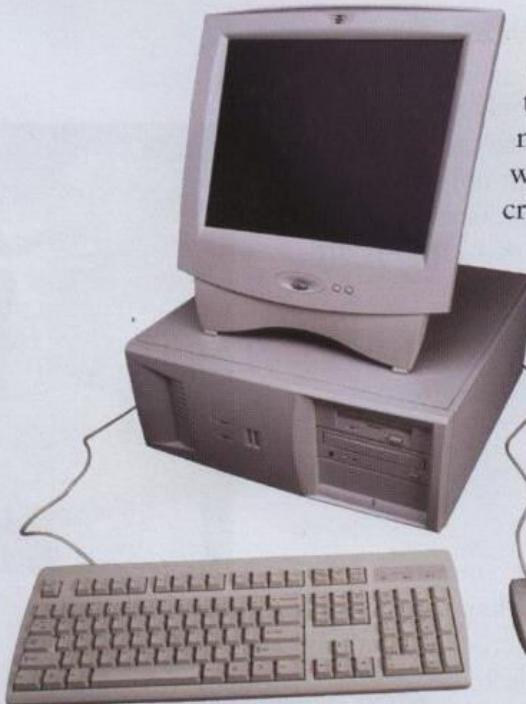


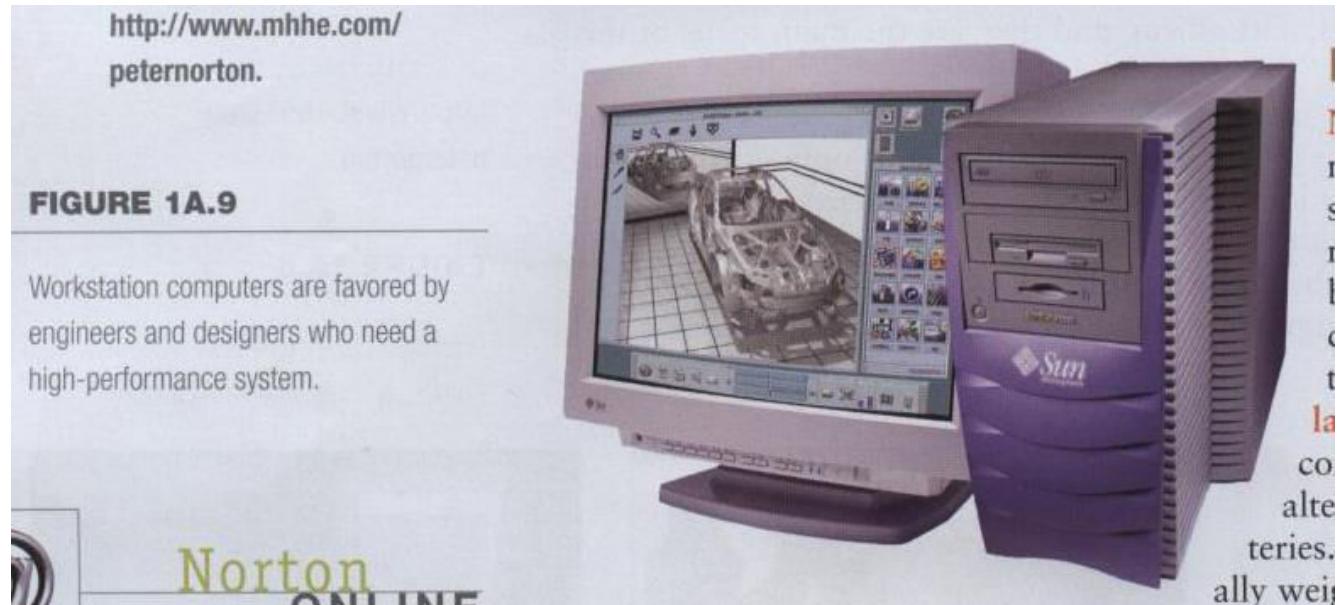
FIGURE 1A.8

This desktop PC has a "tower" design, with a system unit that sits upright and can be placed on either the desk or the floor.



Computers for Individual Use

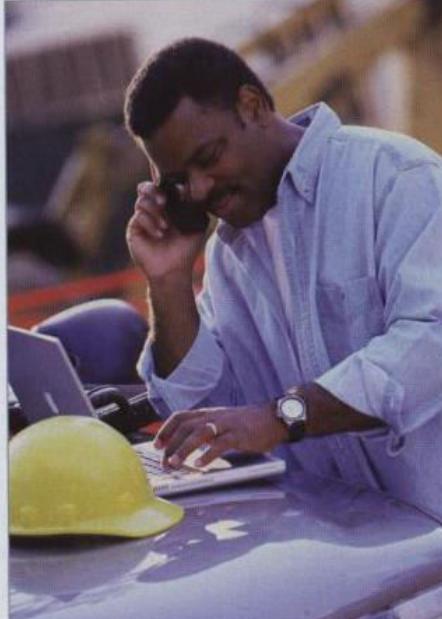
- Workstations
 - Specialized computers
 - Optimized for science or graphics
 - More powerful than a desktop



Computers for Individual Use

- Notebook computers
 - Small portable computers
 - Weighs between 3 and 8 pounds

FIGURE 1A.10



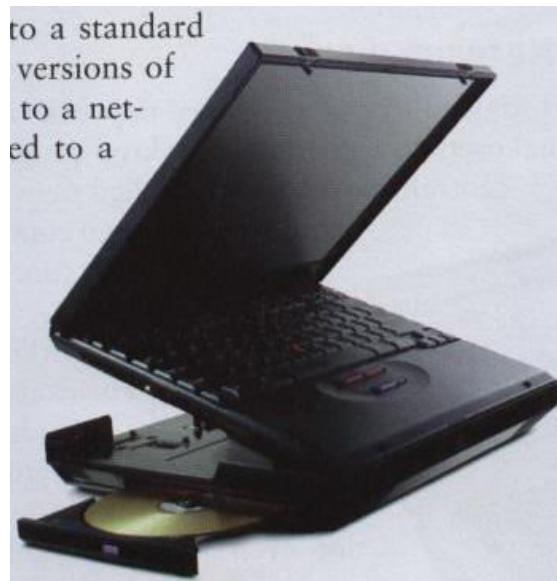
Notebook computers have the power and features of desktop PCs but are light and portable.

Norton ONLINE

For more information on tablet PCs, visit
[http://www.mhhe.com/
peternorton](http://www.mhhe.com/peternorton).

Computers for Individual Use

- Notebook computers
 - About 8 ½ by 11 inches
 - Typically as powerful as a desktop
 - Can include a docking station



to a standard
versions of
to a net-
ed to a

FIGURE 1A.11

A docking station can make a notebook computer feel like a desktop system, by adding a full-size monitor, keyboard, and other features.

Norton
ONLINE

For more information on
handheld PCs, visit



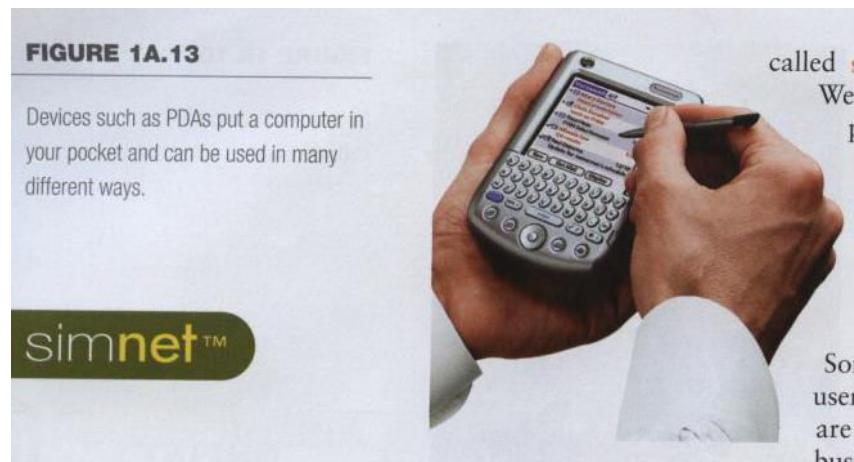
Computers for Individual Use

- Tablet computers
 - Newest development in portable computers
 - Input is through a pen
 - Run specialized versions of office products



Computers for Individual Use

- Handheld computers, palm computer
 - Very small computers
 - Personal Digital Assistants (PDA)
 - Note taking or contact management
 - Data can synchronize with a desktop



Computers for Individual Use

- Smart phones
 - Hybrid of cell phone and PDA
 - Web surfing, e-mail access

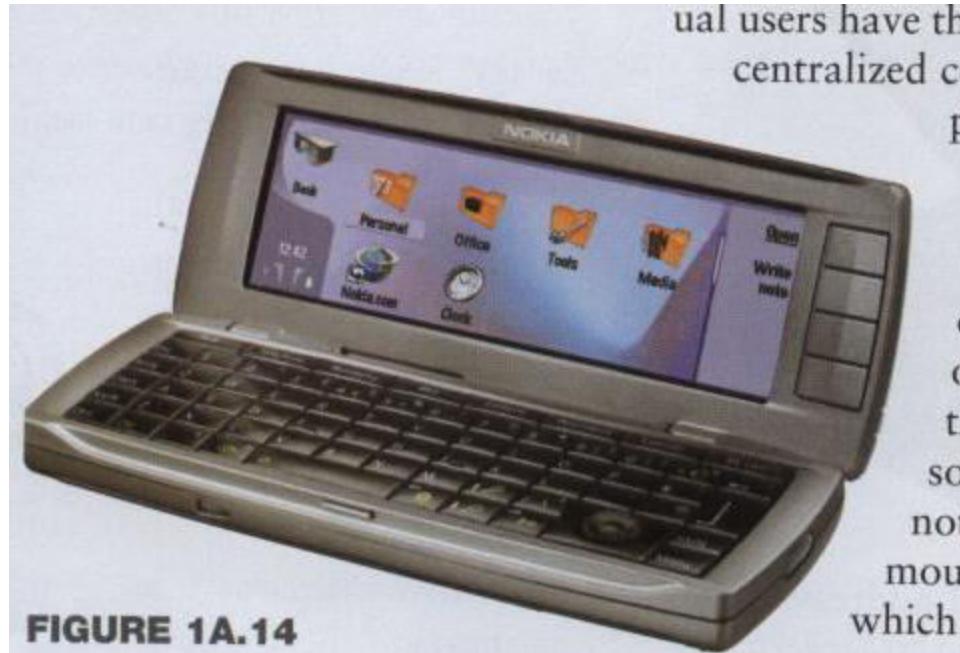


FIGURE 1A.14

New cellular phones, like the Nokia 9500 Communicator, double as tiny computers, offering many of the features of PDAs.



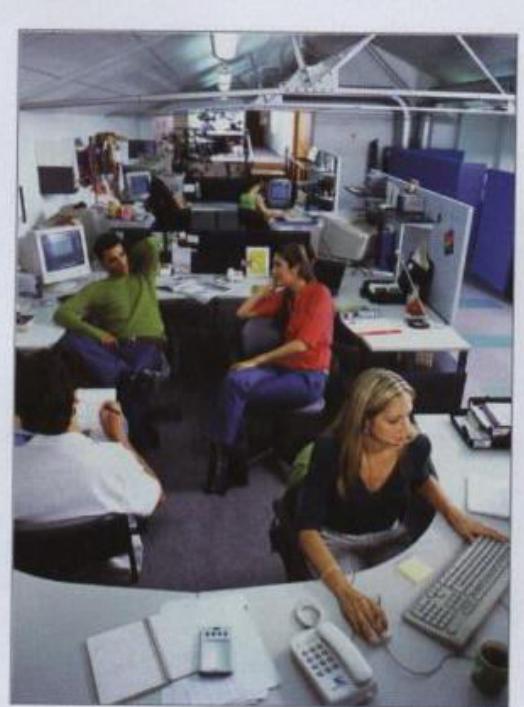
Computers for Organizations

- Network servers
 - Centralized computer
 - All other computers connect

New cellular phones, like the Nokia 9500 Communicator, double as tiny computers, offering many of the features of PDAs.

FIGURE 1A.15

In many companies, workers use their desktop systems to access a central, shared computer.



Computers for Organizations

- Network servers
 - Provides access to network resources
 - Multiple servers are called server farms
 - Often simply a powerful desktop

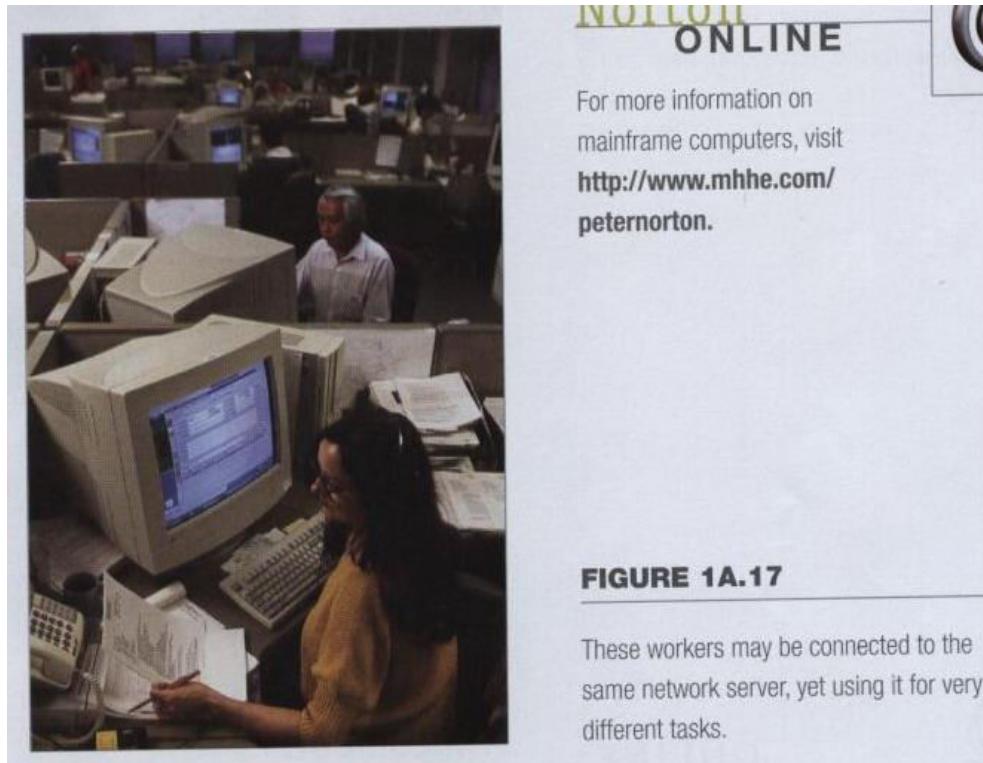


FIGURE 1A.16

Large corporate networks can use hundreds of servers.

Computers for Organizations

- Network servers
 - Flexibility to different kinds of tasks



Computers for Organizations

- Network servers
 - Users use the Internet as a means of connecting even if away from the offices.

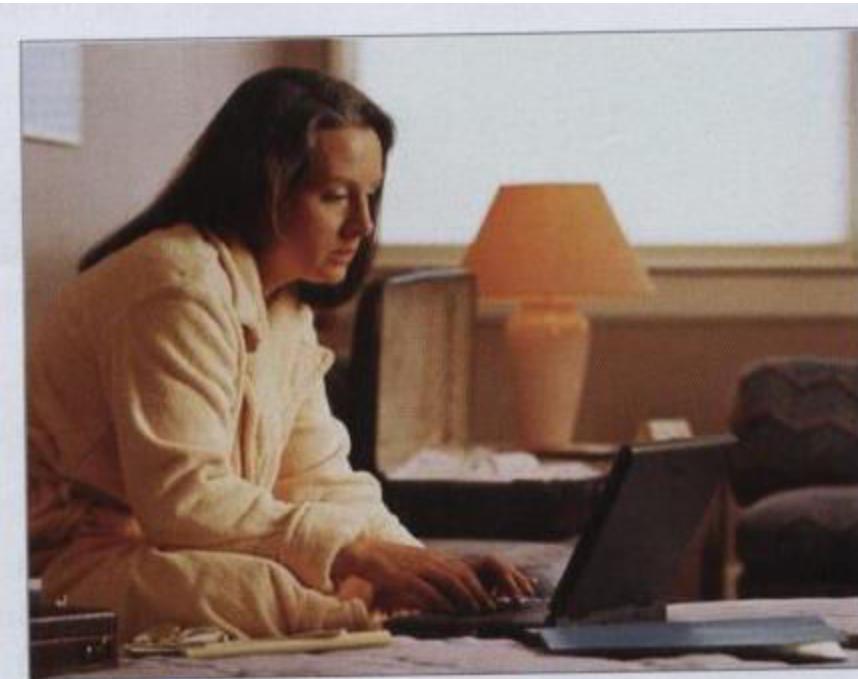


FIGURE 1A.18

Many users can access their organization's network no matter where they go.

Computers for Organizations

- Mainframes
 - Used in large organizations
 - Handle thousands of users
 - Users access through a terminal

FIGURE 1A.19

Hundreds, even thousands, of mainframe users may use terminals to work with the central computer.



Computers for Organizations

- Mainframes
 - Large and powerful systems



Computers for Organizations

- Minicomputers
 - Called midrange computers
 - Power between mainframe and desktop
 - Handle hundreds of users
 - Used in smaller organizations
 - Users access through a terminal

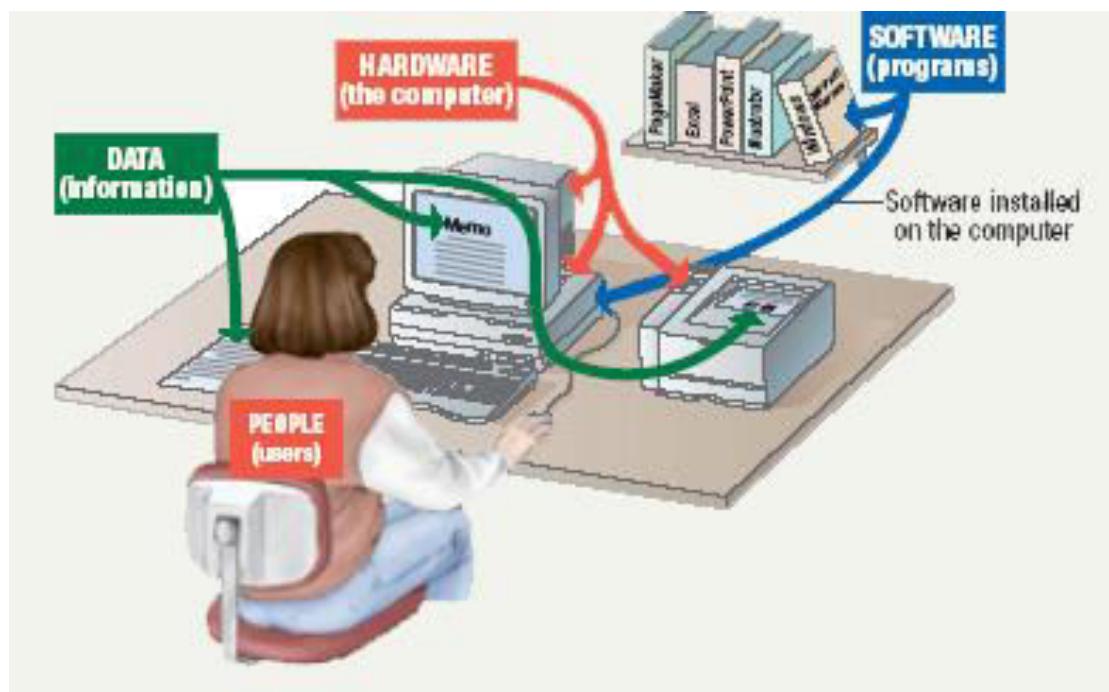
Computers for Organizations

- Supercomputers
 - The most powerful computers made
 - Handle large and complex calculations
 - Process trillions of operations per second
 - Found in research organizations



Parts of the Computer System

- Computer systems have four parts
 - Hardware
 - Software
 - Data
 - User



Parts of the Computer System

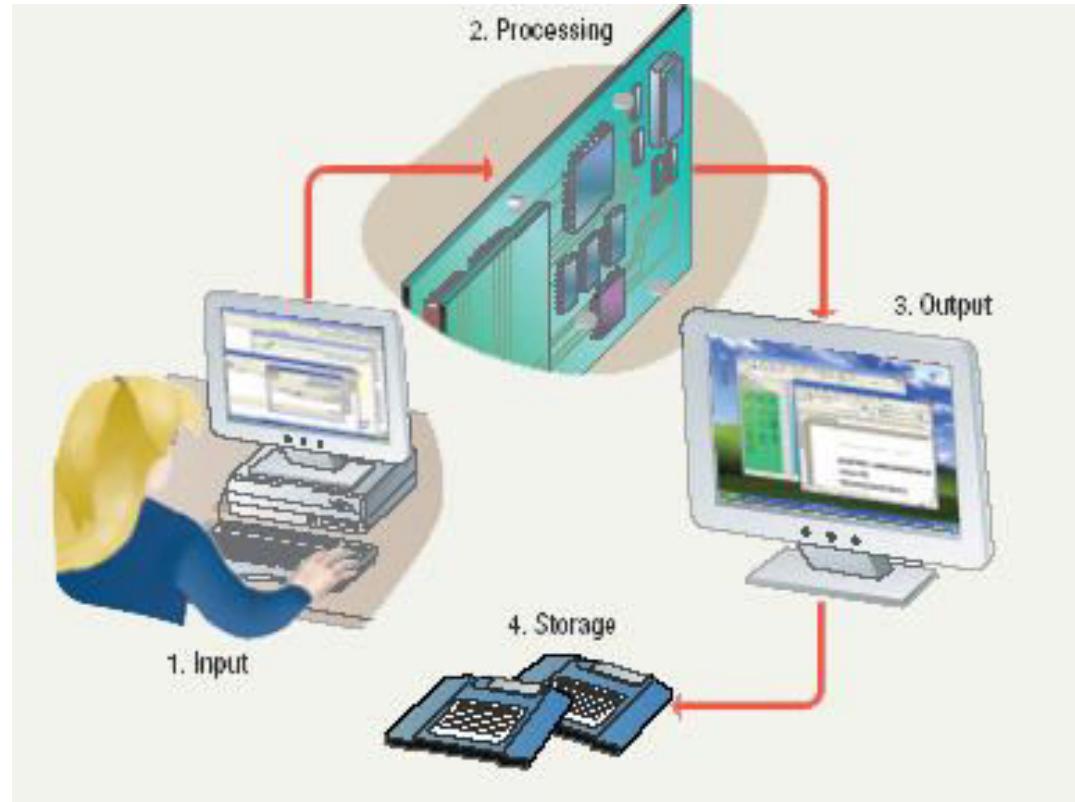
- Hardware
 - Mechanical devices in the computer
 - Anything that can be touched
- Software
 - Tell the computer what to do
 - Also called a program
 - Thousands of programs exist

Parts of the Computer System

- Data
 - Pieces of information
 - Computers organize and present data
- Users
 - People operating the computer
 - Most important part
 - Tell the computer what to do

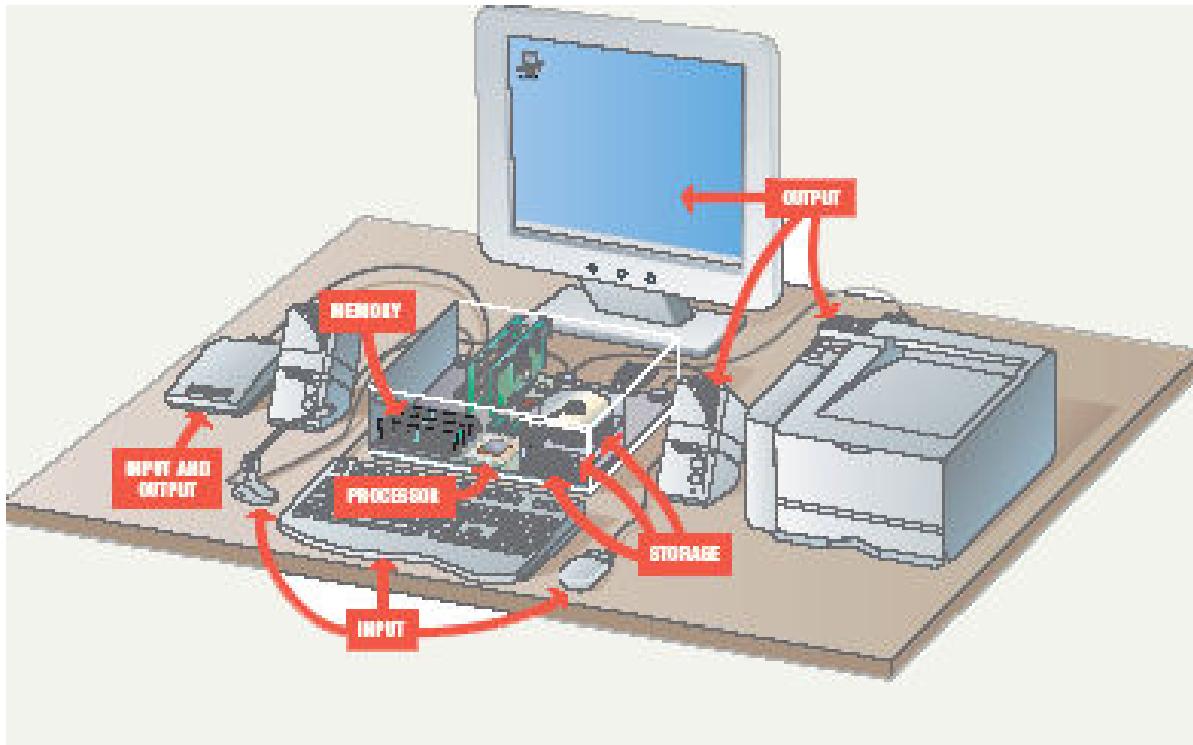
Information Processing Cycle

- Steps followed to process data
- Input
- Processing
- Output
- Storage



Essential Computer Hardware

- Computers use the same basic hardware



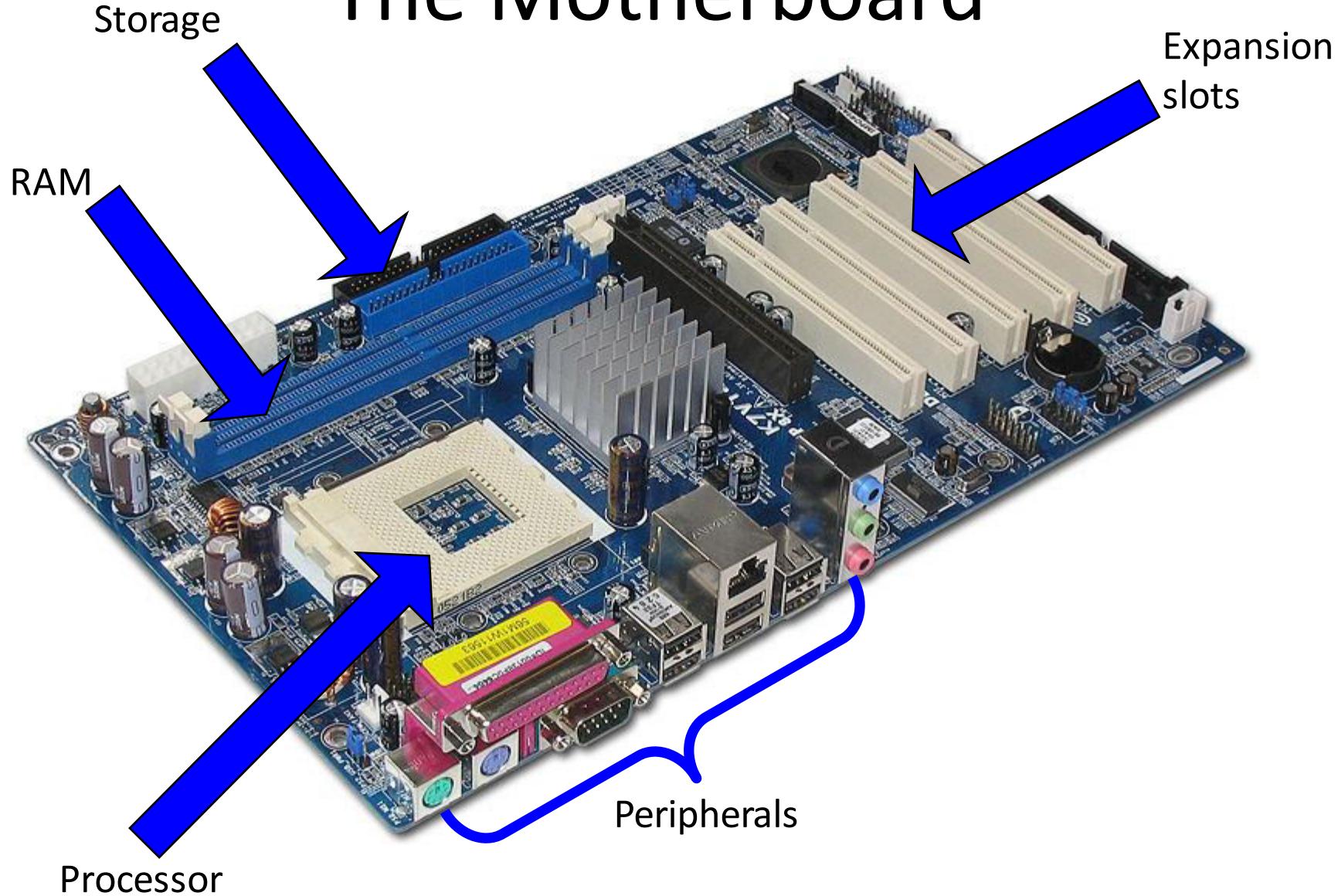
The System Unit

- ❖ The System Unit houses the central processing unit, memory modules, expansion slots, and electronic circuitry as well as expansion cards that are all attached to the motherboard; along with disk drives, a fan or fans to keep it cool, and the power supply.
- ❖ All other devices (monitor, keyboard, mouse, etc., are linked either directly or indirectly into the system unit.



Sources: Tom's Hardware site: <http://www.tomshardware.com>

The Motherboard



Essential Computer Hardware

- Processing devices
 - Brains of the computer
 - Carries out instructions from the program
 - Manipulate the data
 - Most computers have several processors
 - Central Processing Unit (CPU)
 - Secondary processors
 - Processors made of silicon and copper

Essential Computer Hardware

- Memory devices
 - Stores data or programs
 - Random Access Memory (RAM)
 - Volatile
 - Stores current data and programs
 - More RAM results in a faster system
 - Read Only Memory (ROM)
 - Permanent storage of programs
 - Holds the computer boot directions

Essential Computer Hardware

- Input and output devices
 - Allows the user to interact
 - Input devices accept data
 - Keyboard, mouse
 - Output devices deliver data
 - Monitor, printer, speaker
 - Some devices are input and output
 - Touch screens

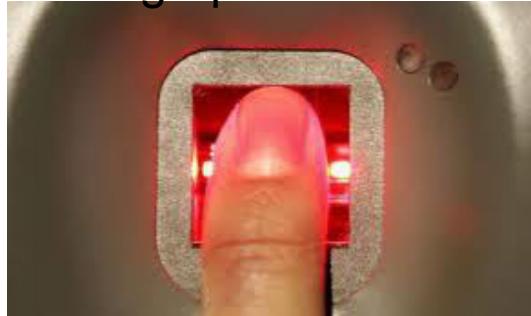
Input Devices

Any peripheral used to provide data and input signals to the computer



Input Devices

Fingerprint scanner



Joystick



Wearable input gloves



Digital Camera



3-D scanner



Scanner



Racing Wheel



Barcode Scanner



Tablet



Output Devices

A Place to present processed data

Monitor



Projector

Speakers



Plotter



Laser Printer

Storage Vs. Memory

Memory (e.g., RAM)

- The information stored is needed now
- Keep the information for a shorter period of time (usually volatile)
- Faster
- More expensive
- Low storage capacity (~1/4 of a DVD for 1 GB)



Storage (e.g., Hard disk)

- The information stored is not needed immediately
- The information is retained longer (non-volatile)
- Slower
- Cheaper
- Higher storage capacity (~50 DVD's for 200 GB)



Essential Computer Hardware

- Storage devices
 - Hold data and programs permanently
 - Different from RAM
 - Magnetic storage
 - Floppy and hard drive
 - Uses a magnet to access data
 - Optical storage
 - CD and DVD drives
 - Uses a laser to access data

Storage Devices

Tape drives



5 1/2" Floppy drive



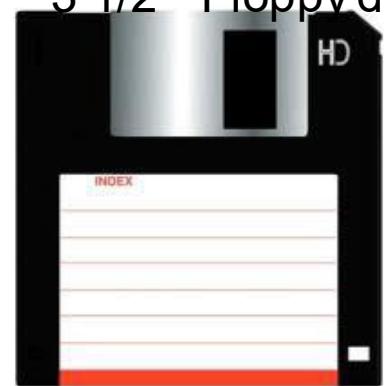
Flash memory card



USB flash drive



3 1/2 " Floppy drive



Storage Devices

Zip drives



CD/DVD/Bluera
y



Hard disk



Magenta Optic

Software Runs the Machine

- Tells the computer what to do
- Reason people purchase computers
- Two types
 - System software
 - Application software

Software Runs the Machine

- System software
 - Most important software
 - Operating system
 - Windows XP
 - Network operating system (OS)
 - Windows Server 2003
 - Utility
 - Symantec AntiVirus

Software Runs the Machine

- Application software
 - Accomplishes a specific task
 - Most common type of software
 - MS Word
 - Covers most common uses of computers

Computer Data

- Fact with no meaning on its own
- Stored using the binary number system
- Data can be organized into files

Computer Users

- Role depends on ability
 - Setup the system
 - Install software
 - Manage files
 - Maintain the system
- “Userless” computers
 - Run with no user input
 - Automated systems

Inputting Data In Other Ways

Devices for the Hand

- Pen based input
 - Tablet PCs, PDA
 - Pen used to write data
 - Pen used as a pointer
 - Handwriting recognition
 - On screen keyboard



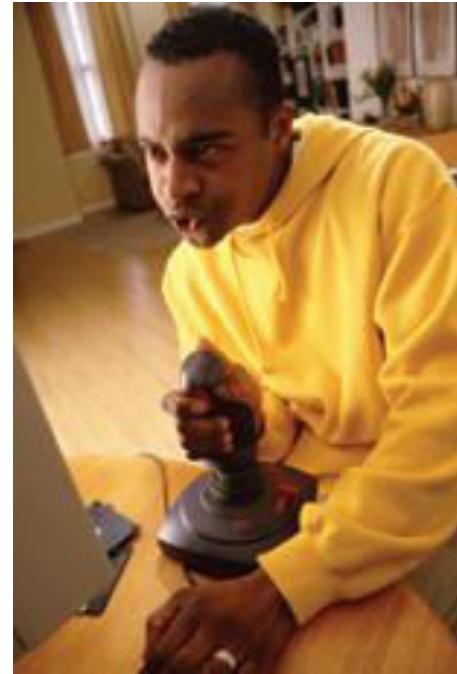
Devices for the hand

- Touch screens
 - Sensors determine where finger points
 - Sensors create an X,Y coordinate
 - Usually presents a menu to users



Devices for the hand

- Game controllers
 - Enhances gaming experience
 - Provide custom input to the game
 - Modern controllers offer feedback
 - Joystick
 - Game pad



Optical Input Devices

- Allows the computer to see input
- Bar code readers
 - Converts bar codes to numbers
 - UPC code
 - Computer finds number in a database
 - Works by reflecting light
 - Amount of reflected light indicates number

Optical Input Devices

- Image scanners
 - Converts printed media into electronic
 - Reflects light off of the image
 - Sensors read the intensity
 - Filters determine color depths

Optical input devices

- Optical character recognition (OCR)
 - Converts scanned text into editable text
 - Each letter is scanned
 - Letters are compared to known letters
 - Best match is entered into document
 - Rarely 100% accurate

Audiovisual Input Devices

- Microphones
 - Used to record speech
 - Speech recognition
 - “Understands” human speech
 - Allows dictation or control of computer
 - Matches spoken sound to known phonemes
 - Enters best match into document

Audiovisual Input Devices

- Musical Instrument Digital Interface
 - MIDI
 - Connects musical instruments to computer
 - Digital recording or playback of music
 - Musicians can produce professional results



Audiovisual Input Devices

- Digital cameras
 - Captures images electronically
 - No film is needed
 - Image is stored as a JPG file
 - Memory cards store the images
 - Used in a variety of professions



Transforming Data Into Information

How Computers Represent Data

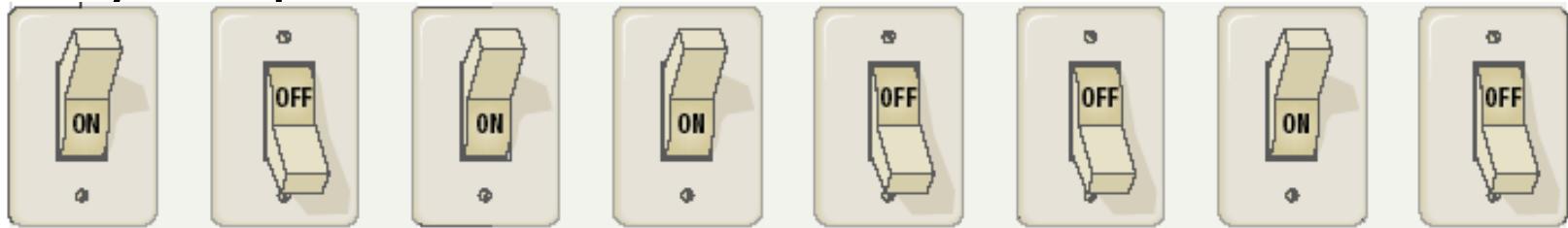
- Number systems
 - A manner of counting
 - Several different number systems exist
- Decimal number system
 - Used by humans to count
 - Contains ten distinct digits
 - Digits combine to make larger numbers

How Computers Represent Data

- Binary number system
 - Used by computers to count
 - Two distinct digits, 0 and 1
 - 0 and 1 combine to make numbers

How Computers Represent Data

- Bits and bytes
 - Binary numbers are made of bits
 - Bit represents a switch
 - A byte is 8 bits
 - Byte represents one character



How Computers Represent Data

- Text codes
 - Converts letters into binary
 - Standard codes necessary for data transfer
 - ASCII
 - American English symbols
 - Extended ASCII
 - Graphics and other symbols
 - Unicode
 - All languages on the planet

How Computers Process Data

- The CPU
 - Central Processing Unit
 - Brain of the computer
 - Control unit
 - Controls resources in computer
 - Instruction set
 - Arithmetic logic unit
 - Simple math operations
 - Registers

How Computers Process Data

- Machine cycles
 - Steps by CPU to process data
 - Instruction cycle
 - CPU gets the instruction
 - Execution cycle
 - CPU performs the instruction
 - Billions of cycles per second
 - Pipelining processes more data
 - Multitasking allows multiple instructions

How Computers Process Data

- Memory
 - Stores open programs and data
 - Small chips on the motherboard
 - More memory makes a computer faster



How Computers Process Data

- Nonvolatile memory
 - Holds data when power is off
 - Read Only Memory (ROM)
 - Basic Input Output System (BIOS)
 - Power On Self Test (POST)

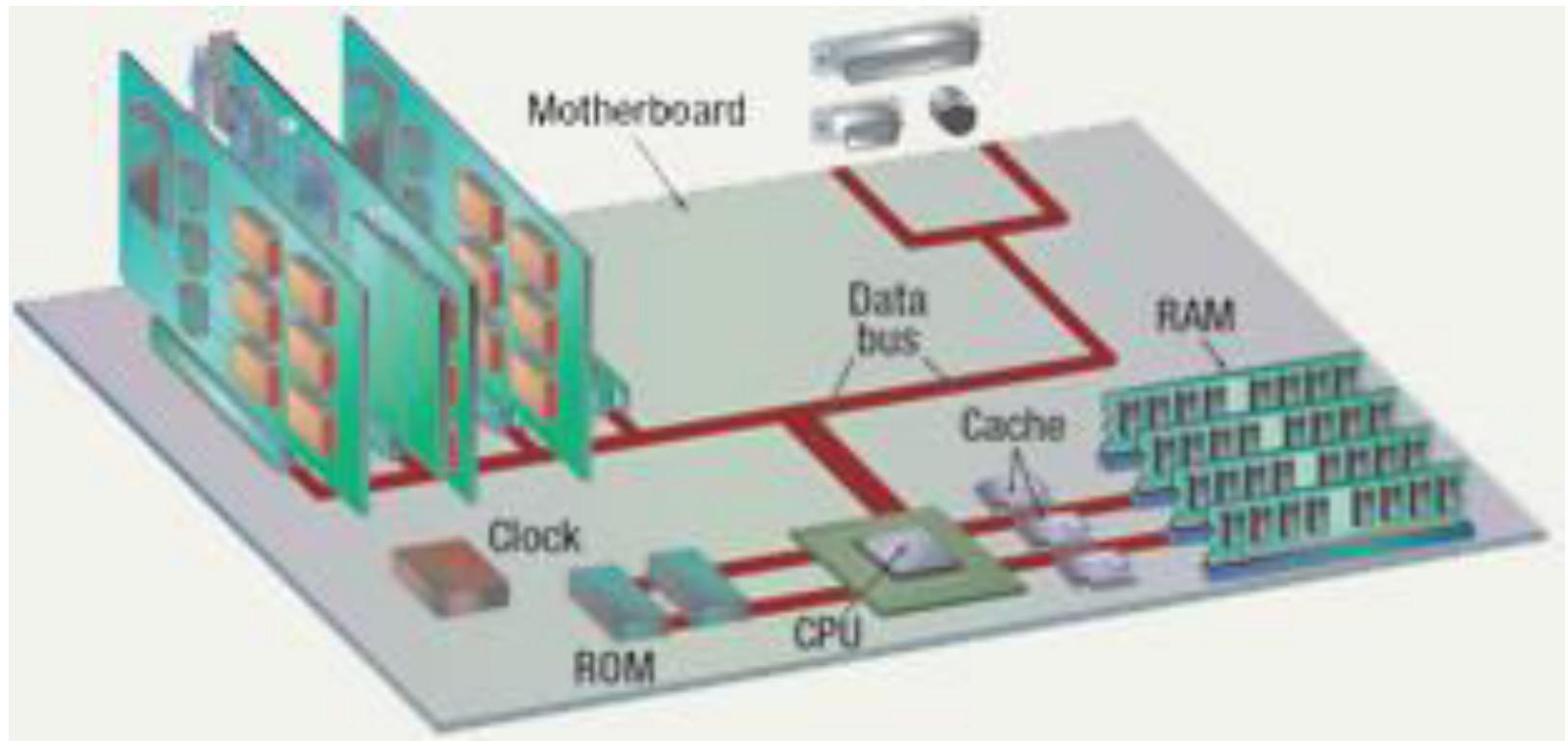
How Computers Process Data

- Flash memory
 - Data is stored using physical switches
 - Special form of nonvolatile memory
 - Camera cards, USB key chains

How Computers Process Data

- Volatile memory
 - Requires power to hold data
 - Random Access Memory (RAM)
 - Data in RAM has an address
 - CPU reads data using the address
 - CPU can read any address

Components affecting Speed



Affecting Processing Speed

- Registers
 - Number of bits processor can handle
 - Word size
 - Larger indicates more powerful computer
 - Increase by purchasing new CPU

Affecting Processing Speed

- Virtual RAM
 - Computer is out of actual RAM
 - File that emulates RAM
 - Computer swaps data to virtual RAM
 - Least recently used data is moved

Affecting Processing Speed

- The computer's internal clock
 - Quartz crystal
 - Every tick causes a cycle
 - Speeds measured in Hertz (Hz)
 - Modern machines use Giga Hertz (GHz)

Affecting Processing Speed

- The bus
 - Electronic pathway between components
 - Expansion bus connects to peripherals
 - System bus connects CPU and RAM
 - Bus width is measured in bits
 - Speed is tied to the clock

Affecting Processing Speed

- External bus standards
 - Industry Standard Architecture (ISA)
 - Local bus
 - Peripheral control interface
 - Accelerated graphics port
 - Universal serial bus
 - IEEE 1394 (FireWire)
 - PC Card

Affecting Processing Speed

- Peripheral control interface (PCI)
 - Connects modems and sound cards
 - Found in most modern computers

Affecting Processing Speed

- Accelerated Graphics Port (AGP)
 - Connects video card to motherboard
 - Extremely fast bus
 - Found in all modern computers

Affecting Processing Speed

- Universal Serial Bus (USB)
 - Connects external devices
 - Hot swappable
 - Allows up to 127 devices
 - Cameras, printers, and scanners

Affecting Processing Speed

- PC Card
 - Used on laptops
 - Hot swappable
 - Devices are the size of a credit card



Affecting Processing Speed

- Cache memory
 - Very fast memory
 - Holds common or recently used data
 - Speeds up computer processing
 - Most computers have several caches
 - L1 holds recently used data
 - L2 holds upcoming data
 - L3 holds possible upcoming data

Modern CPUs

A Look Inside The Processor

- Architecture
 - Determines
 - Location of CPU parts
 - Bit size
 - Number of registers
 - Pipelines
 - Main difference between CPUs

Microcomputer Processors

- Intel
 - Leading manufacturer of processors
 - Intel 4004 was worlds first microprocessor
 - IBM PC powered by Intel 8086
 - Current processors
 - Centrino
 - Itanium
 - Pentium IV
 - Xeon



Microcomputer Processors

- Advanced Micro Devices (AMD)
 - Main competitor to Intel
 - Originally produced budget products
 - Current products outperform Intel
 - Current processors
 - Sempron
 - Athlon FX 64
 - Athlon XP



Microcomputer Processors

- Freescale
 - A subsidiary of Motorola
 - Co-developed the Apple G4 PowerPC
 - Currently focuses on the Linux market

Microcomputer Processors

- IBM
 - Historically manufactured mainframes
 - Partnered with Apple to develop G5
 - First consumer 64 bit chip

Comparing Processors

- Speed of processor
- Size of cache
- Number of registers
- Bit size
- Speed of Front side bus

Advanced Processor Topics

- RISC processors
 - Reduced Instruction Set Computing
 - Smaller instruction sets
 - May process data faster
 - PowerPC and G5

Advanced Processor Topics

- Parallel Processing
 - Multiple processors in a system
 - Symmetric Multiple Processing
 - Number of processors is a power of 2
 - Massively Parallel Processing
 - Thousands of processors
 - Mainframes and super computers

Extending The Processors Power

- Standard computer ports
 - Keyboard and mouse ports
 - USB ports
 - Parallel
 - Network
 - Modem
 - Audio
 - Serial
 - Video

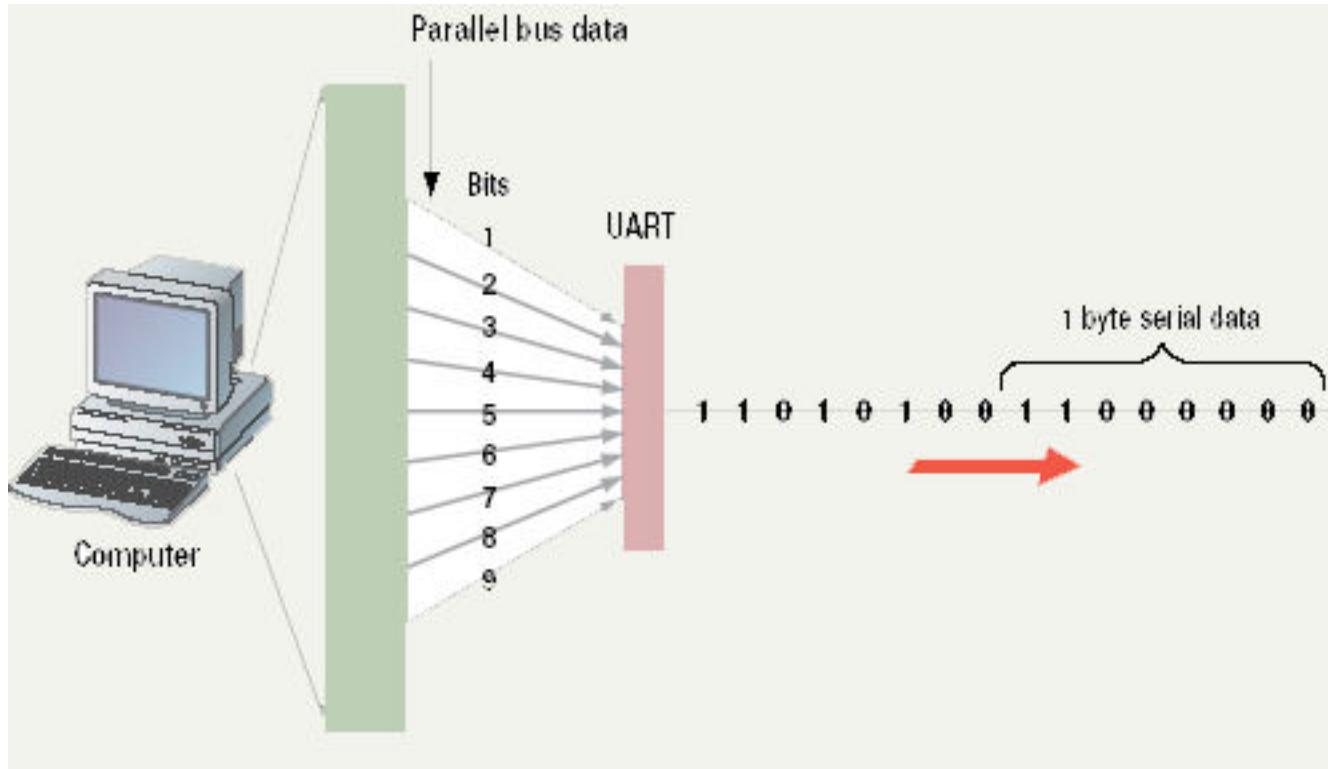
Standard Computer Ports



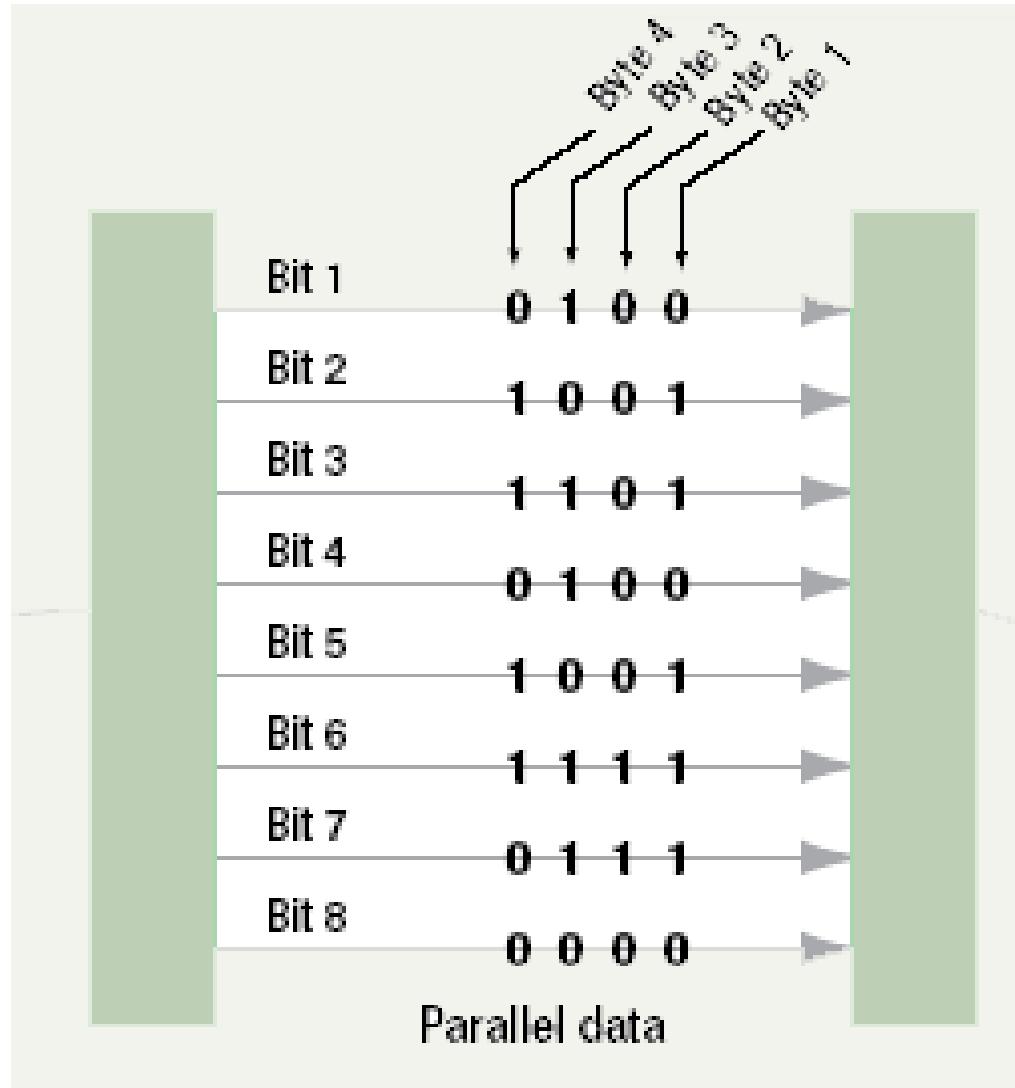
Extending The Processors Power

- Serial and parallel ports
 - Connect to printers or modems
 - Parallel ports move bits simultaneously
 - Made of 8 – 32 wires
 - Internal busses are parallel
 - Serial ports move one bit
 - Lower data flow than parallel
 - Requires control wires
 - UART converts from serial to parallel

Serial Communications



Parallel Communications



Extending The Processors Power

- SCSI
 - Small Computer System Interface
 - Supports dozens of devices
 - External devices daisy chain
 - Fast hard drives and CD-ROMs

Extending The Processors Power

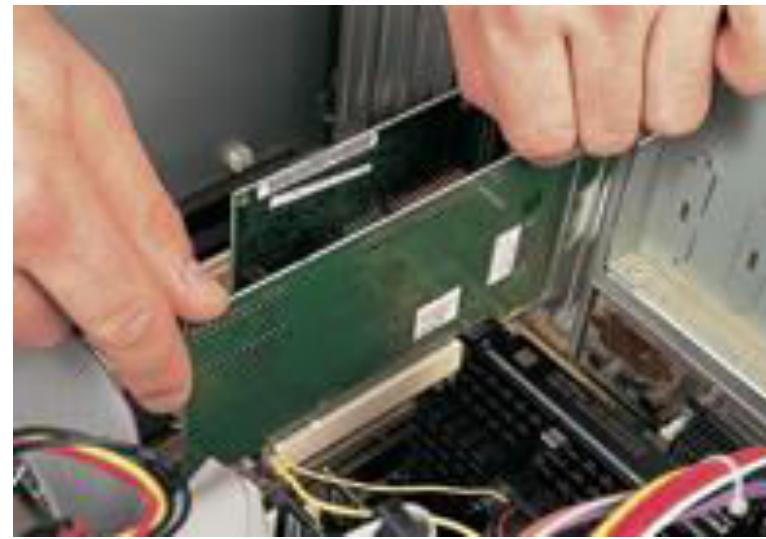
- USB
 - Universal Serial Bus
 - Most popular external bus
 - Supports up to 127 devices
 - Hot swappable

Extending the Processors Power

- FireWire
 - IEEE 1384
 - Cameras and video equipment
 - Hot swappable
 - Port is very expensive

Extending the Processors Power

- Expansion slots and boards
 - Allows users to configure the machine
 - Slots allow the addition of new devices
 - Devices are stored on cards
 - Computer must be off before inserting



Extending the Processors Power

- PC Cards
 - Expansion bus for laptops
 - Hot swappable
 - Small card size
 - Three types, I, II and III
 - Type II is most common

Extending the Processors Power

- Plug and play
 - New hardware detected automatically
 - Prompts to install drivers
 - Non-technical users can install devices

Operating System Basics

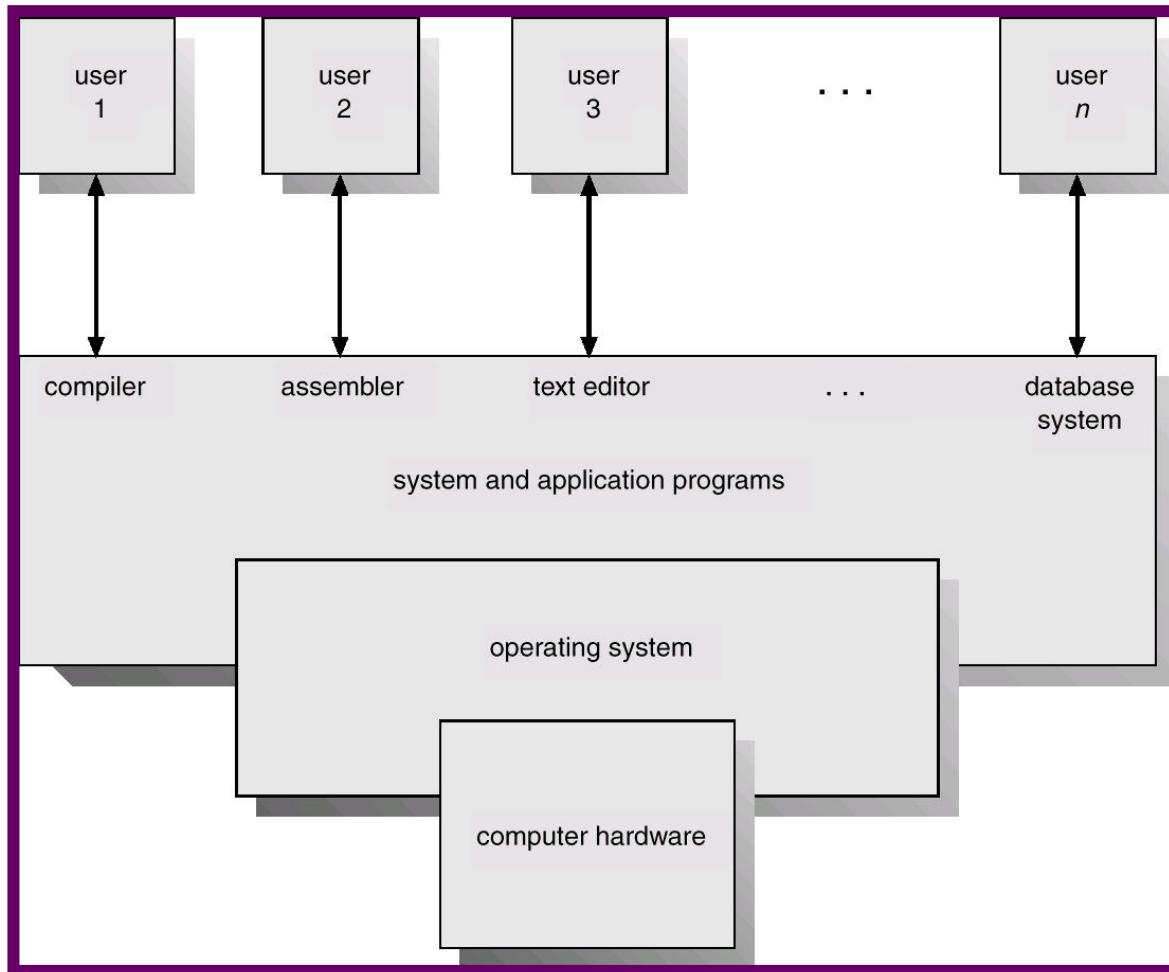
Operating System

- λ Like the brain the OS manages the computer
- λ A program that manages the computer hardware
- λ Provides services for application software
- λ Acts as an intermediary between a user and the computer hardware
- λ Without OS, no application program will run

Other Operating System Definitions

- ❑ Resource allocator – manages and allocates resources.
- ❑ Control program – controls the execution of user programs and operations of I/O devices .
- ❑ Kernel – the one program running at all times (all else being application programs).

Abstract View of a Comp. System



Silberschatz

Operating System ...

- „ Provides the means for proper use of the resources available
- „ Like a government, it performs no useful function by itself. It provides an environment within which other programs can do useful work

Functions of Operating Systems

- Provide a user interface
- Run programs
- Manage hardware devices
- Organized file storage

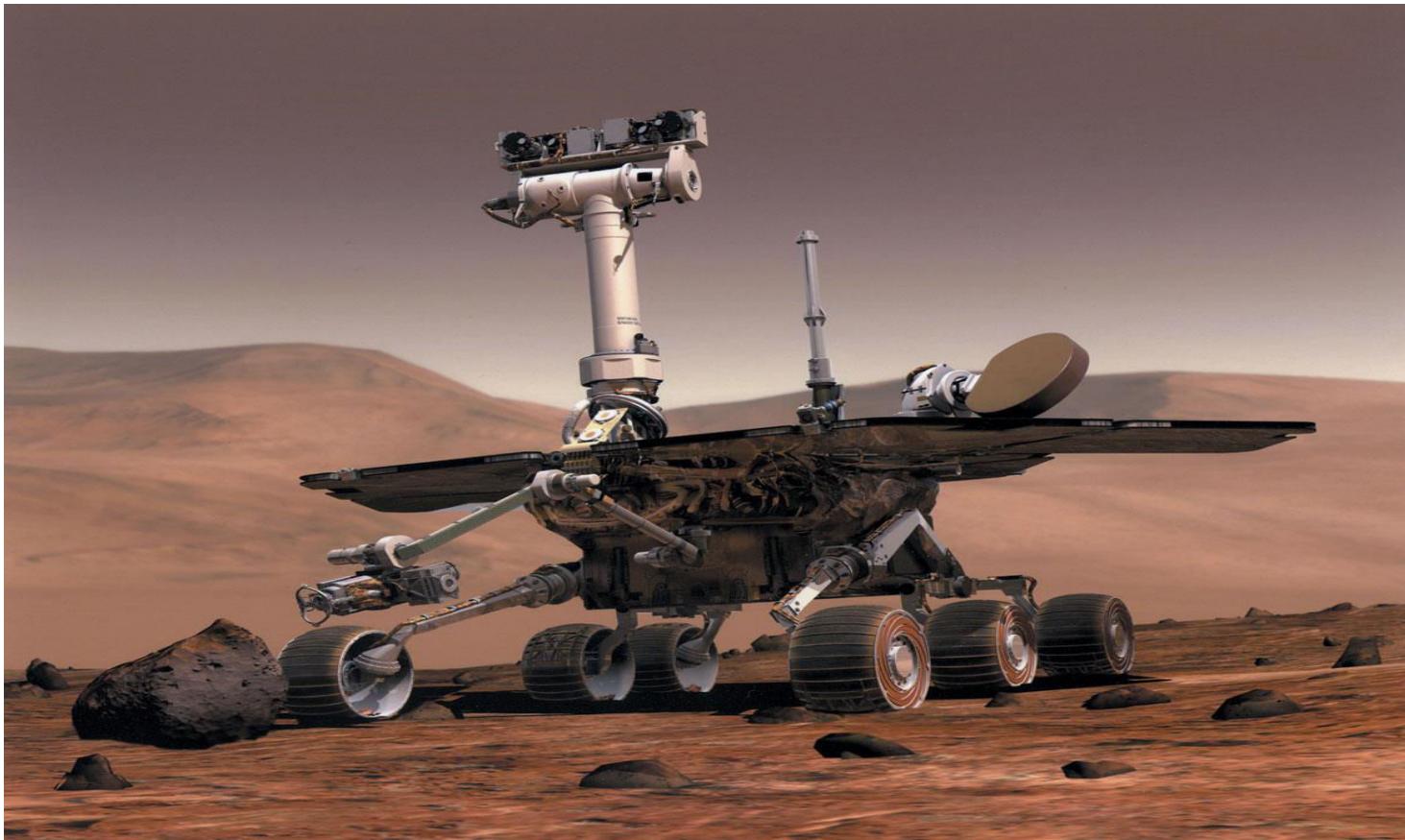
One OS for all Systems ?

- Different systems exist

Types of Operating Systems

- Real-time operating system
 - Very fast small OS
 - Built into a device
 - Respond quickly to user input
 - MP3 players, Medical devices

Real Time Systems



Types of Operating Systems

- Single user/Single tasking OS
 - One user works on the system
 - Performs one task at a time
 - MS-DOS and Palm OS
 - Take up little space on disk
 - Run on inexpensive computers

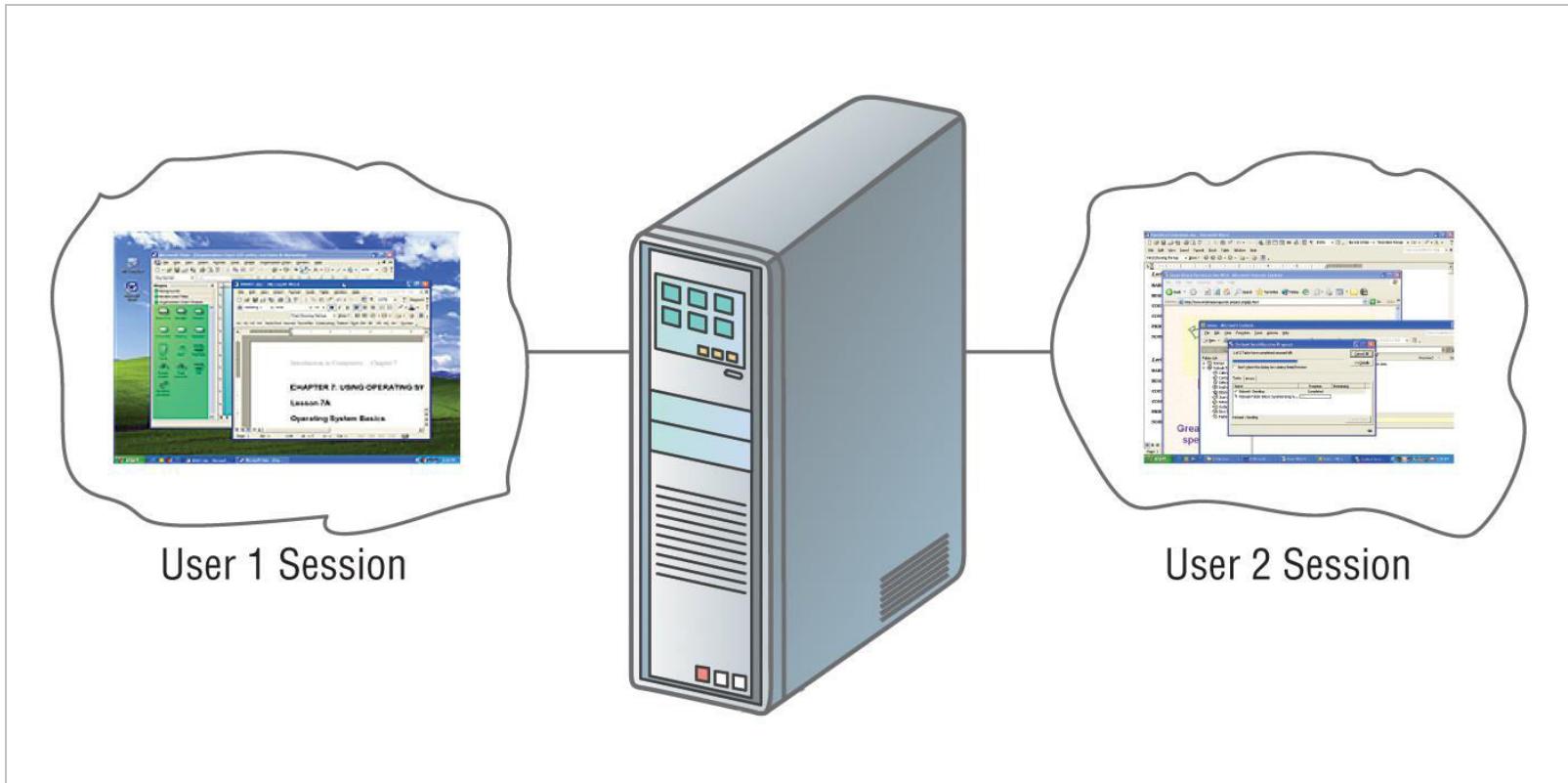
Types of Operating Systems

- Single user/Multitasking OS
 - User performs many tasks at once
 - Most common form of OS
 - Windows XP and OS X
 - Require expensive computers
 - Tend to be complex

Types of Operating Systems

- Multi user/Multitasking OS
 - Many users connect to one computer
 - Each user has a unique session
 - UNIX, Linux, and VMS
 - Maintenance can be easy
 - Requires a powerful computer

Multi user/Multi tasking OS



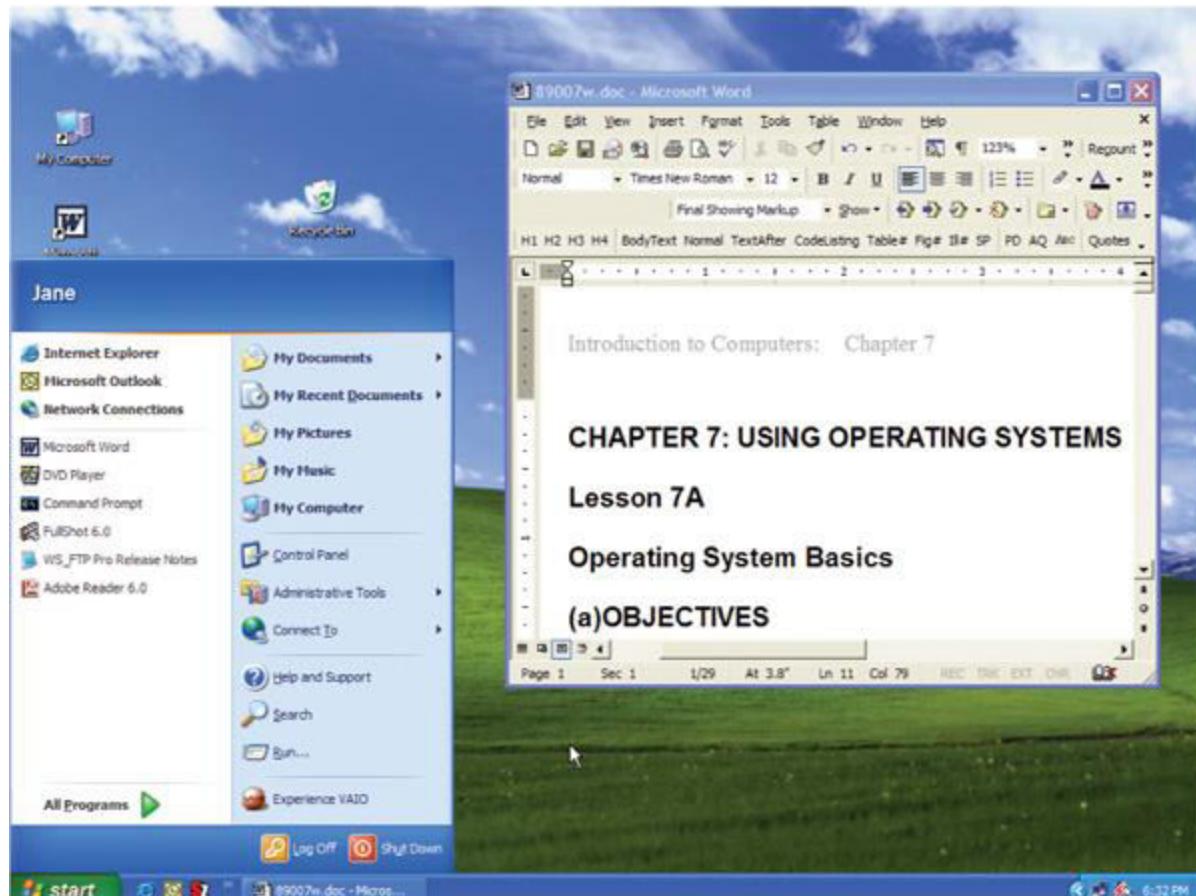
Providing a User Interface

- User interface
 - How a user interacts with a computer
 - Require different skill sets

Providing a User Interface

- Graphical user interface (GUI)
 - Most common interface
 - Windows, OS X, Gnome, KDE
 - Uses a mouse to control objects
 - Uses a desktop metaphor
 - Shortcuts open programs or documents
 - Open documents have additional objects
 - Task switching
 - Dialog boxes allow directed input

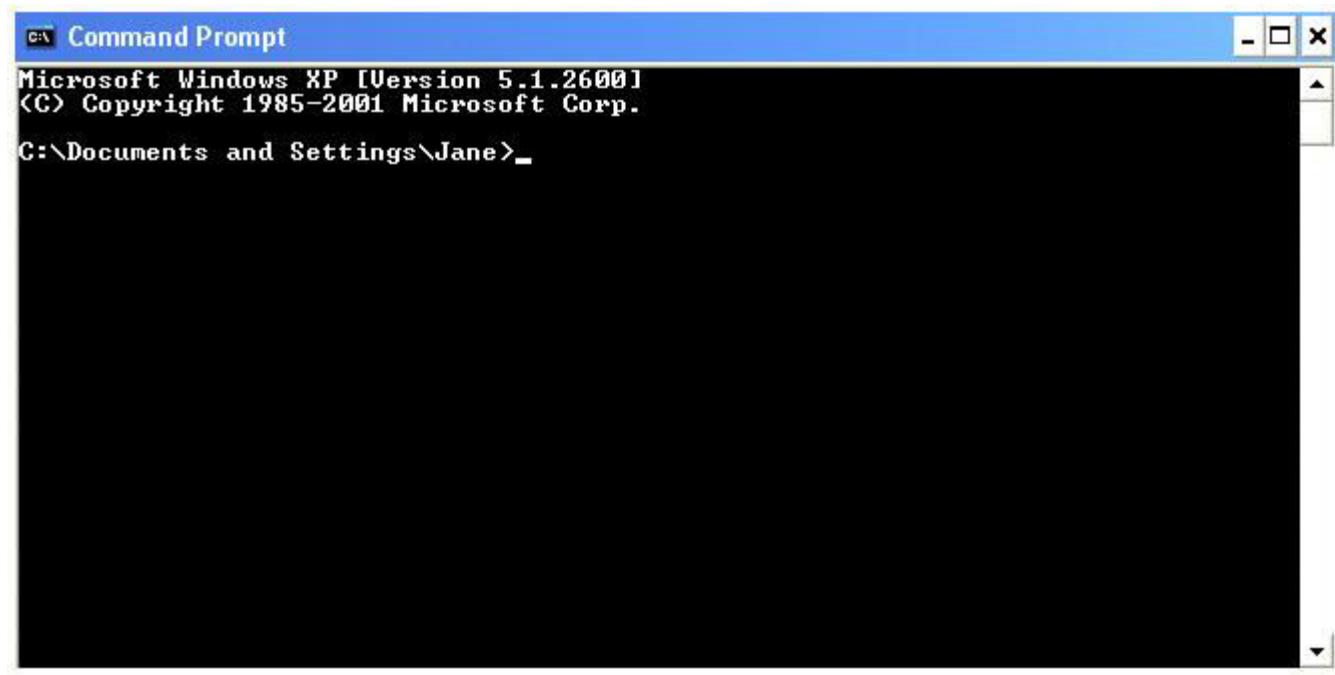
Graphical User Interface



Providing a User Interface

- Command line interfaces
 - Older interface
 - DOS, Linux, UNIX
 - User types commands at a prompt
 - User must remember all commands
 - Included in all GUIs

Command Line Interface



Running Programs

- Many different applications supported
- System call
 - Provides consistent access to OS features
- Share information between programs
 - Copy and paste
 - Object Linking and Embedding

Managing Hardware

- Programs need to access hardware
- Interrupts
 - CPU is stopped
 - Hardware device is accessed
- Device drivers control the hardware

Organizing Files and Folders

- Organized storage
- Long file names
- Folders can be created and nested
- All storage devices work consistently

Enhancing an OS

- Utilities
 - Provide services not included with OS
 - Goes beyond the four functions
 - Firewall, anti-virus and compression
 - Prices vary

Enhancing an OS

- Backup software
 - Archives files onto removable media
 - Most OS include a backup package
 - Many third party packages exist

Enhancing an OS

- Anti-virus software
 - Crucial utility
 - Finds, blocks and removes viruses
 - Must be updated regularly
 - McAfee and Norton Anti-Virus

Enhancing an OS

- Firewall
 - Crucial utility
 - Protects your computer from intruders
 - Makes computer invisible to hackers
 - Zone Labs is a home firewall
 - Cisco sells hardware firewalls

Enhancing an OS

- Intrusion detection
 - Often part of a firewall package
 - Announces attempts to breach security
 - Snort is a Linux based package

Enhancing an OS

- Screen savers
 - Crucial utility for command line systems
 - Prevents burn in
 - Merely fun for GUI systems
 - Screen saver decorates idle screens



Survey of PC and Network Operating Systems

PC Operating Systems

- Microsoft Windows is the most popular
 - Installed more than other OS combined
 - Installed on about 95% of computers
 - Apple and Linux represent the other 5%

PC Operating Systems

- DOS
 - Disk Operating System
 - Single user single-tasking OS
 - Command line interface
 - 16-bit OS
 - Powerful
 - Fast
 - Supports legacy applications

DOS Application

Employee Maintenance		Change	View	Load new Location	Options	Help	Quit	
View Screen								
Funburgers								
Employee Name		Mon	Tue	Wed	Thr	Fri	Sat	Sun
Cashier		Off	11:00a	11:00a	11:00a	11:00a	Off	Off
Jeff	Howister	Off	07:00p	07:00p	07:00p	07:00p	Off	Off
Beer Server		Off	Off	09:30a	Off	09:30a	Off	Off
Jody	Loveless	Off	Off	09:30p	Off	09:30p	Off	Off
Grill Attendant		10:00a	10:00a	Off	10:00a	10:00a	Off	Off
Eva	Perone	09:00p	09:00p	Off	09:00p	09:00p	Off	Off
Runner		09:00a	09:00a	09:00a	09:00a	Off	Off	09:00a
Todd	Jones	10:00p	10:00p	10:00p	10:00p	Off	Off	10:00p
Sweeper		Off	09:00a	09:00a	09:00a	09:00a	Off	Off
Andy	Kaufmann	Off	10:00p	10:00p	10:00p	10:00p	Off	Off
Cashier		11:00a	Off	Off	Off	Off	Off	11:00a
Mandy	Williams	07:00p	Off	Off	Off	Off	Off	07:00p
Beer Server		09:30a	09:30a	Off	Off	Off	Off	Off
Gloria	A Reimann	09:30p	09:30p	Off	Off	Off	Off	Off
Beer Server		Off	Off	Off	09:30a	Off	Off	Off
Mandy	Williams	Off	Off	Off	09:30p	Off	Off	Off

Enter new employees or Edit/Delete employees. Alt-F1 for Help Index

PC Operating Systems

- Windows NT
 - Designed for a powerful system
 - 32-bit OS
 - Very stable
 - Windows NT Workstation
 - Single user multi tasking OS
 - Windows NT Server
 - Multi user multi tasking OS
 - Network operating system

PC Operating Systems

- Windows 9x
 - 95, 98, and Millennium Edition (Me)
 - 32-bit OS
 - Supported 16-bit programs well
 - Very pretty not stable OS
 - Still found in large corporations
 - 95 introduced the Start button
 - 98 introduced active desktop
 - Me improved multimedia software

PC Operating Systems

- Windows 2000
 - Look of 9x with NT stability
 - Optimized for office and developers
 - Application software ran very well
 - Entertainment software ran very poorly

PC Operating Systems

- Windows XP
 - Microsoft's newest desktop product
 - Different look from 2000
 - Many different versions
 - Digital multimedia support was enhanced
 - Communications was enhanced
 - Mobile computing became a priority

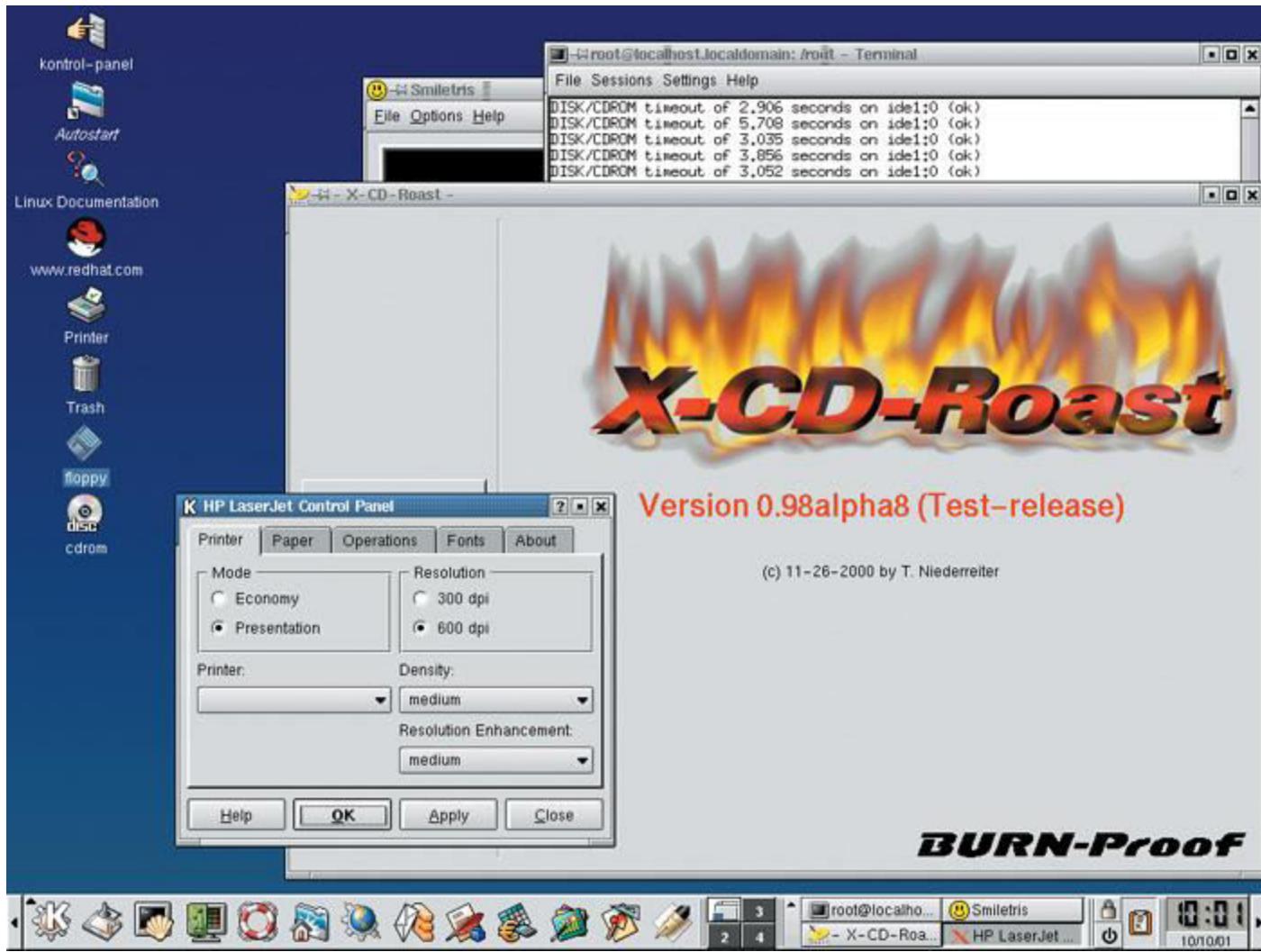
PC Operating Systems

- UNIX
 - Runs on all computer types
 - 32- or 64-bit
 - Very stable and fast
 - Command-line interface
 - Can cost thousands of dollars

PC Operating Systems

- Linux
 - Free or inexpensive version of UNIX
 - 32-bit OS
 - Very stable and fast
 - Most flavors are open source
 - X Windows GUI
 - Command line interface is available

Linux Desktop



PC Operating Systems

- Macintosh operating systems
 - OS X
 - Based on FreeBSD Linux
 - Very stable and easy to configure
 - Only runs on Mac hardware

OS X Desktop



NOS Features

- Network operating system
- Fast and stable
- Runs on servers
- Multi-user and multitasking OS
- 32- or 64-bit

NOS Features

- File and print sharing
 - Users access the same files
 - Hundreds of users use a printer
 - Different OS can interact

NOS Features

- Data integrity
 - Backups copies data onto tape
 - RAID copies data onto other drives
 - Redundant Array of Inexpensive Disks
 - Relies on two or more hard drives
 - May speed access to data

NOS Features

- Data security
 - Access to data can be restricted
 - Access to server resources is controlled
 - Audits can be kept

Networking Operating Systems

- NT Server
 - Same core as NT Workstation
 - Security added
 - Multi user capability added
 - RAID support included

Networking Operating Systems

- Windows 2000 Server
 - Same technology as 2000 workstation
 - Active Directory (AD)
 - Central database of server resources
 - Simplifies network management
 - Distributed programs supported
 - Server Standard Edition
 - Advanced Server Edition
 - Data Center Edition

Networking Operating Systems

- Windows 2003 Server
 - Designed to compete with UNIX
 - XP like interface
 - Better support for XP computers
 - MS .NET framework supported
 - Distributed programs are supported
 - Server Standard Edition
 - Data Center Edition

Networking Operating Systems

- UNIX for servers
 - Oldest NOS in widespread use
 - Stable secure and fast
 - Main OS for Internet and Web
 - Large organizations depend on UNIX

Networking Operating Systems

- Linux for servers
 - Popular in small businesses
 - Stable, fast and inexpensive
 - Linux's popularity is growing

Embedded Operating Systems

- Devices have EOS built in
- Cell phones, PDAs, medical equipment
- Stable and fast

Embedded Operating Systems

- Windows XP embedded
 - Based on Windows XP
 - Customized for each device

Embedded Operating Systems

- Windows CE
 - Not based on a desktop OS
 - Customized for each device
 - PDA and cell phones
 - Microsoft Automotive will run in cars

Embedded Operating Systems

- Palm OS
 - Standard on Palm PDA
 - First PDA OS for consumers
 - Can be found on cell phones



Embedded Operating Systems

- Pocket PC
 - Developed to compete with Palm
 - Not customizable
 - Interacts securely with business networks
 - Can control PCs through PC

Embedded Operating Systems

- Symbian
 - Found in smart cell phones
 - Games, Instant Messaging, Internet
 - Full color display

Programming Languages and the Programming Process

Programming Languages

- λ Communicate with the computer
- λ Understood by human and machine
- λ Languages
 - Lower level language (closer to machine)
 - Higher level language (closer to humans)

Programming Languages ...

- λ Machine Languages (1st Generation)
- λ Assembly Languages (2nd Generation)
- λ Procedural Languages (3rd Generation)
- λ Problem Oriented Languages (4th Generation)
- λ Natural Languages (5th Generation)

Machine Languages

- λ Machine knows two things, 1 & 0
- λ All instructions in terms of ones and zeros
- λ $5 \times 4 = 00110101\ 00101010\ 00110100$

a=5

b=10

c=a*b

```
01100001 00111101 00110101 00001101 00001010 01100010  
00111101 00110001 00110000 00001101 00001010 01100011  
00111101 01100001 00101010 01100010
```

Machine Language

Advantages

- Execution very fast and efficient

Disadvantages

- Not easy to encode programs
- No one standard machine language
- Machine dependent - will not easily run on another machine

Language Categories

- Second generation languages
 - Assembly languages
 - Statements that represent machine code
 - Code converted by an assembler
 - Still used to optimize video games

```
;CLEAR SCREEN USING BIOS
CLR: MOV AX,0600H      ;SCROLL SCREEN
      MOV BH,30          ;COLOUR
      MOV CX,0000          ;FROM
      MOV DX,184FH         ;TO 24,79
      INT 10H              ;CALL BIOS;

;INPUTTING OF A STRING
KEY: MOV AH,0AH          ;INPUT REQUEST
     LEA DX,BUFFER        ;POINT TO BUFFER WHERE STRING STORED
     INT 21H                ;CALL DOS
     RET                   ;RETURN FROM SUBROUTINE TO MAIN PROGRAM;

; DISPLAY STRING TO SCREEN
SCR: MOV AH,09             ;DISPLAY REQUEST
     LEA DX,STRING          ;POINT TO STRING
     INT 21H                ;CALL DOS
     RET                   ;RETURN FROM THIS SUBROUTINE;
```

Assembly Language

- λ Reduce program complexity
- λ Symbolic language using mnemonic codes
- λ Codes replacing ones and zeros
- λ Does not replace machine language
- λ Assembly code converted into machine lang.
- λ One line of assembly code converted to one/many lines of machine code

Assembly Language

λ Advantages

- More standardized & easier to use
- Operates efficiently (not as efficient as machine language code)
- Easier to debug

λ Disadvantages

- Very lengthy programs
- Complex
- Machine dependent

Start	Move N, R1
	Move N2,
	CLR R0
Loop	Add (R2), R0
	Inc R2
	Dec R1
	BGTZ
loop	
	Move R0,

High Level Languages

- ❑ All languages above assembly language
- ❑ Procedure oriented Languages (3rd Gen.)
 - General purpose language
 - Express a logic
 - Solve a variety of problems

e.g. Pascal, BASIC, FORTRAN, COBOL

Advantages	Disadvantages
❑ English like statements	Executes more slowly
❑ Easy to understand and modify	
❑ Machine independent	

Language Categories

- Third generation languages (3GL)
 - First higher level language
 - Supports structured and OOP
 - Code is reusable
 - Code is portable
 - Typically written in an IDE
 - C/C++ creates games and applications
 - Java creates web applets
 - ActiveX creates Web and Windows applets

High Level Languages ...

- λ Problem oriented languages (4th Gen.)
- λ Solve specific problems or develop specific applications
- λ PC Application software
- λ Query languages & report generators
- λ Application generators

Language Categories

- Fourth generation languages (4GL)
 - Easier to use than 3GL
 - Coded in a visual IDE
 - Tools reduce the amount of code
 - Object oriented programming
 - Microsoft .Net is a language
 - Dream Weaver is an 4GL IDE

High Level Languages ...

- λ Natural language (5th Gen.)
- λ Make man-machine interface humanlike
- λ Simulate learning process
- λ LISP and PROLOG

Programming Languages

- Used to generate source code
- Avoids using machine code
- Have strict rules of syntax
 - Symbols and punctuation have meaning
 - Spelling must be exact
- Code is converted into machine language

Interpreter

- λ Spanish PM talking to Japanese PM
- λ Dumb machine, knows only 1 & 0
- λ Interprets the high level language
- λ Interprets line by line
- λ Converts each line of code just before execution
- λ Executes program line immediately

Interpreter ...

- λ Converts to intermediate level language machine can understand and executes
- λ No object code is stored
- λ One line executed several times, them converted to machine language each time
- λ Delay
- λ Example BASIC, Perl, MATLAB, Python, etc..

UN Assembly ...



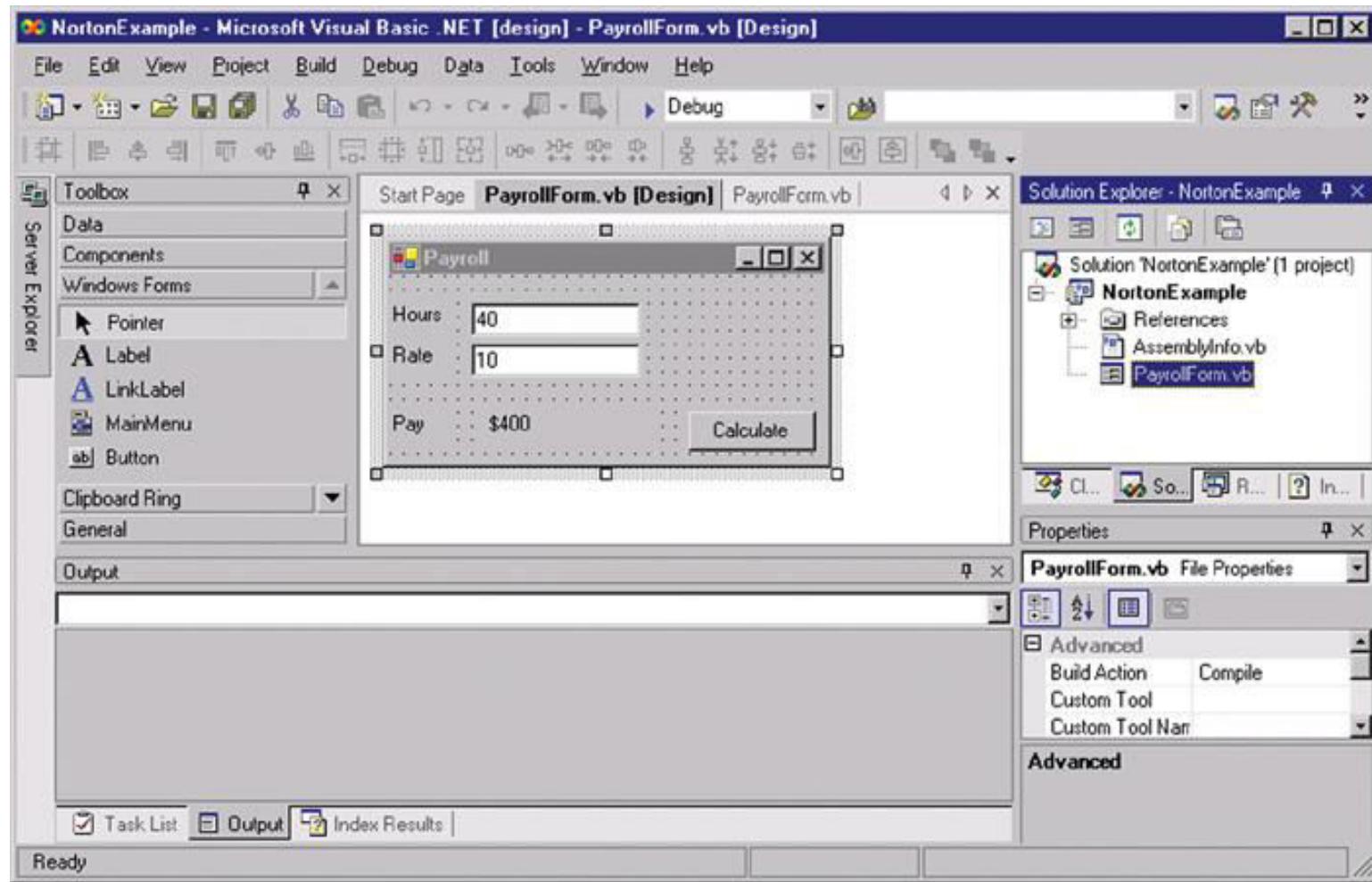
Compiler

- λ UN General Assembly
- λ Predefined text, pretranslated, audio and printed text available during live address in own language
- λ Complied text
- λ Compile program in machine language
- λ Compiler program's job

Compiler ...

- ❑ Translate source code into object code/machine language
- ❑ Look at entire source code, collect and reorganise instructions
- ❑ Fast
- ❑ Better than interpreted languages
- ❑ Executed faster and more efficiently once object code is obtained
- ❑ Many compilers exist for same language
- ❑ Example C, FORTRAN, COBOL

Microsoft.NET



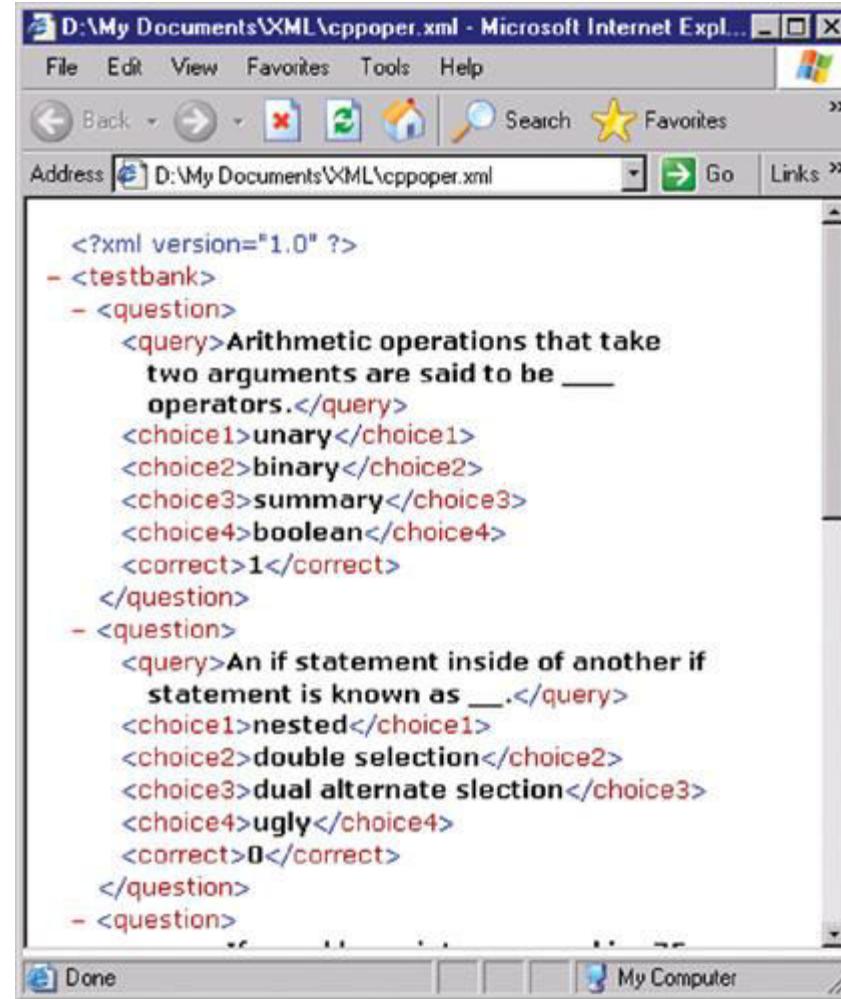
WWW Development Languages

- Markup languages
 - Describe how the text is formatted
- Hyper Text Markup Language (HTML)
 - Basis of all web pages
 - Defines web structure using tags
 - Easy to learn and use
 - Created with a text editor

WWW Development Languages

- Extensible Markup Language (XML)
 - Stores data in a readable format
 - Looks like HTML
 - Allows developers to create tags
 - Depends on HTML for formatting

XML



A screenshot of Microsoft Internet Explorer version 6.0 displaying an XML document. The window title is "D:\My Documents\XML\cppoper.xml - Microsoft Internet Expl...". The address bar shows the file path "D:\My Documents\XML\cppoper.xml". The XML code is displayed in the main content area:

```
<?xml version="1.0" ?>
- <testbank>
  - <question>
    <query>Arithmetic operations that take
      two arguments are said to be __
      operators.</query>
    <choice1>unary</choice1>
    <choice2>binary</choice2>
    <choice3>summary</choice3>
    <choice4>boolean</choice4>
    <correct>1</correct>
  </question>
  - <question>
    <query>An if statement inside of another if
      statement is known as __.</query>
    <choice1>nested</choice1>
    <choice2>double selection</choice2>
    <choice3>dual alternate slection</choice3>
    <choice4>ugly</choice4>
    <correct>0</correct>
  </question>
  - <question>
```

WWW Development Languages

- Extensible HTML (XHTML)
 - Newer version of HTML
 - Stricter rules
 - Based on XML rules

WWW Development Languages

- Extensible Style Sheet Language (XSL)
 - Format and displays XML documents
 - Rules that dictate formatting
 - Create a standard web page

WWW Development Languages

- Extensible HTML Mobile Profile
 - XHTML MP
 - Initially Wireless Markup Language (WML)
 - Creates pages viewable on a handheld

WWW Development Languages

- Cascading Style Sheets (CSS)
 - Format HTML, XHTML and XSL
 - Applies consistent formatting to all pages

WWW Development Languages

- Web authoring environments
 - Reduces tedium for creating pages
 - Tools that simplify web site creation
 - Macromedia Dream weaver
 - Simplifies large sites
 - CSS support is exceptional
 - Microsoft FrontPage simplifies large sites
 - Macromedia Flash creates web animations

WWW Development Languages

- Scripting languages
 - Create dynamic web pages
 - Change based on user input
 - HTML can create static pages
 - Page is generated as needed

WWW Development Languages

- JavaScript
 - Developed by Netscape
 - Works inside of HTML
 - Page verification and simple animation
 - Based on Java

WWW Development Languages

- Active Server Pages (ASP)
 - Developed by Microsoft
 - Based on Visual Basic
 - Good at connecting to Microsoft databases
 - Runs only on Microsoft servers

WWW Development Languages

- Perl
 - Old UNIX language
 - Found on all Windows and Linux servers
 - Excellent web scripting language

WWW Development Languages

- Hypertext Pre-Processor (PHP)
 - Especially good at connecting to MySQL
 - Very popular language
 - Runs on UNIX and Windows

CSE 1001

Problem Solving and Programming

Skills Required for a Software Engineer

Technical Skills

- Software Design
- Coding
- Testing

Problem Solving Skills

- logical and analytical thinking

Soft Skills

- Communication
- Team Work

Problem

- Problem is a puzzle that requires logical thought and /or mathematics to solve.
- A puzzle could be a set of questions on a scenario which consists of ***description of reality*** and **set of constraints about the scenario**.

Example Scenario: VIT Chennai campus has a library. The librarian issues books only to VIT employees. Careful observation suggests...

Description of reality : There is a library in VIT Chennai campus and there is a librarian in the library.

Problem

- **Constraint :** Librarian issues books only to VIT employees

Questions about the scenario:

- 1 How many books are there in the library?
- 2 How many books can be issued to an employee?
3. Does the librarian issue a book to himself? etc

Case study - Discussion

Consider a bigger scenario...



Have you ever observed this scenario?
Yes!!! What are the problems in the scenario?

Types of Problems

- All Problems do not have a straightforward solutions.
- Some problems, such as balancing a checkbook or baking a cake, can be solved with a series of actions.
- These solutions are called **algorithmic solutions**.
- There may be more than one solution for a problem
- Identify all possible ways to solve a problem and choose one among them

Types of Problems

- The solutions of other problems, such as how to buy the best stock or whether to expand the company, are not so straightforward.
- These solutions require reasoning built on knowledge and experience, and a process of trial and error.
- Solutions that cannot be reached through a direct set of steps are called **heuristic solution**

Problem Solving with Computers

- Computers are built to solve problems with algorithmic solutions, which are often difficult or very time consuming when input is large
- Solving a complicated calculus problem or alphabetizing 10,000 names is an easy task for the computer
- So the basis for solving any problem through computers is by developing an algorithm

- Field of computers that deals with heuristic types of problems is called Artificial Intelligence (AI)
- Artificial intelligence enables a computer to do things like human by building its own knowledge bank
- As a result, the computer's problem-solving abilities are similar to those of a human being.
- Artificial intelligence is an expanding computer field, especially with the increased use of Robotics.

Computational Problems

- Computation is the process of evolution from one **state to** another in accordance with some **rules**.

Types of Computational Problems

where the answer for every instance is either yes or no.

Decision Problem

Deciding whether a given number is prime

Searching an element from a given set of elements. Or arranging them in an order

Searching & Sorting Problem

Finding product name for given product ID and arranging products in alphabetical order of names

Counting no. of occurrences of a type of elements in a set of elements

Counting Problem

Counting how many different type of items are available in the store

Finding the best solution out of several feasible solutions

Optimization Problem

Finding best combination of products for promotional campaign

Problem Solving Life Cycle

What do you understand with point of sale problem?

1- Under
problem

What plan can you prepare for solving the problem?

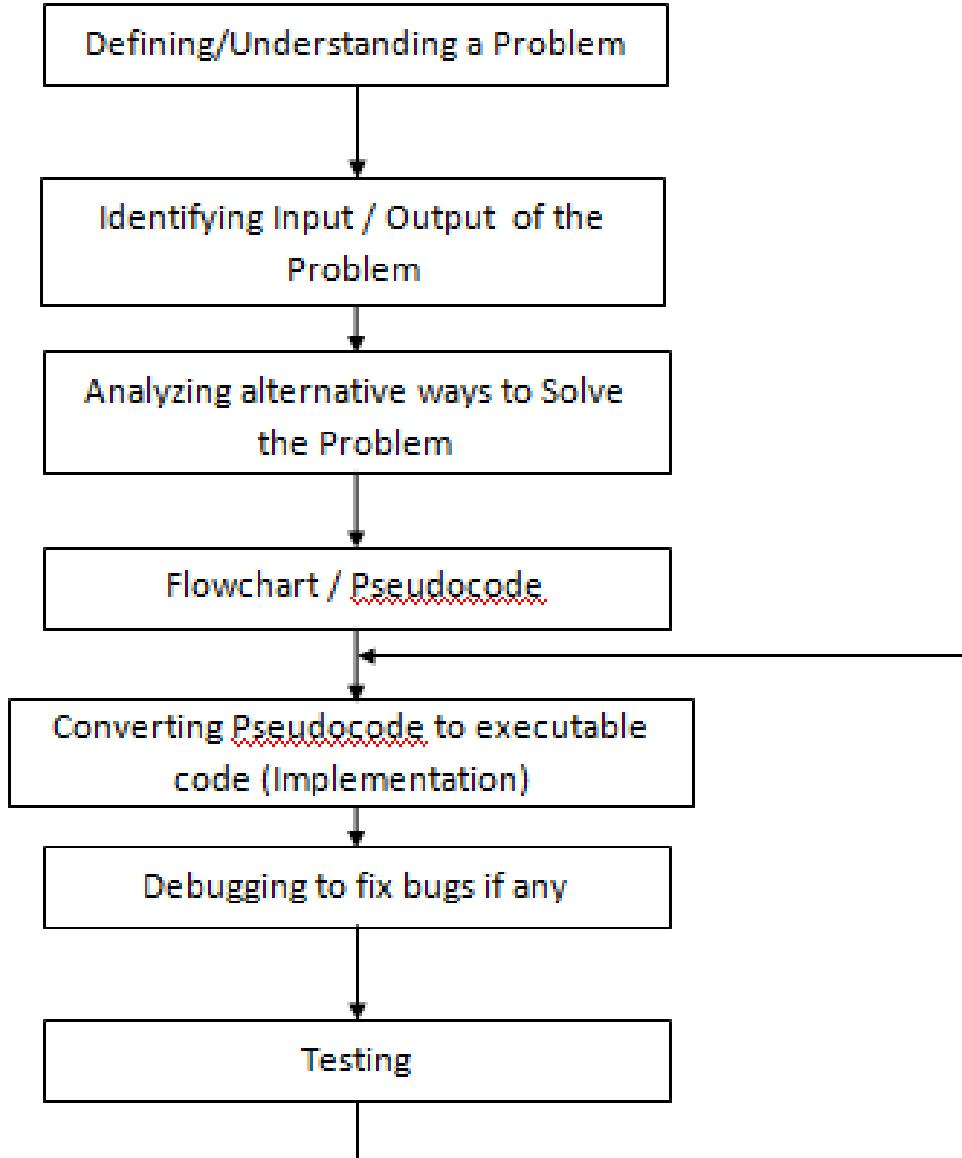
4. Look back i.e. verifying

2. Make a
plan of
solution

3. Carry
out the
plan

How can you ensure the success of the plan?

How will you carry out the plan?



Logic – Basis for solving any problem

- **Definition : A method of human thought that involves thinking in a linear, step by step manner about how a problem can be solved**
- Logic is a language for reasoning.
- It is a collection of rules we use when doing reasoning.
- Eg: John's mum has four children.
- The first child is called April.
- The second May.
- The third June.
- What is the name of the fourth child?



What Problem Can Be Solved By Computer

- Solving problem by computer undergo two phases:
 - Phase 1:
 - Organizing the problem or pre-programming phase.
 - Phase 2:
 - Programming phase.

PRE-PROGRAMMING PHASE

- **Analyzing The Problem**
 - Understand and analyze the problem to determine whether it can be solved by a computer.
 - Analyze the requirements of the problem.
 - Identify the following:
 - Data requirement.
 - Processing requirement or procedures that will be needed to solve the problem.
 - The output.

PRE-PROGRAMMING PHASE

- All these requirements can be presented in a Problem Analysis Chart (PAC)

Data	Processing	Output	Solution Alternatives
given in the problem or provided by the user	List of processing required or procedures.	Output requirement.	List of ideas for the solution of the problem.

PRE-PROGRAMMING PHASE

- **Payroll Problem**
 - Calculate the salary of an employee who works by hourly basis. The formula to be used is

Salary = Hour works * Pay rate

Data	Processing	Output	Solution Alternatives
Hours work, Pay rate	Salary = Hours work * payrate	Salary	<ol style="list-style-type: none">1. Define the hours worked and pay rate as constants.*2. Define the hours worked and pay rate as input values.

Miles to Km

Write a Problem Analysis Chart (PAC) to convert the distance in miles to kilometers where 1.609 kilometers per mile.

Data	Processing	Output	Solution Alternatives
Distance in miles	Kilometers = 1.609 x miles	Distance in kilometers	<ol style="list-style-type: none">1. Define the miles as constants.*2. Define the miles as input values.

Importance of Logic in problem solving

Determine whether a given number is prime or not?

Data	Processing	Output	Solution Alternatives
Number, N	Check if there is a factor for N	Print Prime or Not Prime	<ol style="list-style-type: none">1. Divide N by numbers from 2 to N and if for all the division operations, the remainder is non zero, the number is prime otherwise it is not prime2. Same as 1 but divide the N from 2 to $N/2$3. Same as Logic 1 but divide N from 2 to square root of N

Importance of Logic in problem solving

In a fun game, MXM grid is given with full of coins. The player has to give a number 'N' of his choice. If N is lesser than M then he is out of game and doesn't gain any points. Otherwise he has to place all coins in the MXM grid in the NXN grid and he gains points equal to the number of free cells in the N X N grid.

Data	Processing	Output	Solution Alternatives
Numbers M and N	If N is less than M Points = 0 Otherwise Compute Points as $N^2 - M^2$	Number of points gained	1. Compute $N^2 - M^2$ as NXN – MXM 2. Compute $(N + M) \times (N - M)$ (Number of multiplication is reduced)

Problem 2

- Write a Problem Analysis Chart (PAC) to find an area of a circle where $\text{area} = \pi * \text{radius} * \text{radius}$

Data	Processing	Output
radius	$\text{area} = 3.14 \times \text{radius} \times \text{radius}$	area

Problem 3

- Write a Problem Analysis Chart (PAC) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are

$$\text{Celsius} = 10 \times (\text{depth}) + 20$$

$$\text{Fahrenheit} = 1.8 \times (\text{Celsius}) + 32$$

Data	Processing	Output
depth	$\text{celsius} = 10 \times (\text{depth}) + 20$ $\text{fahrenheit} = 1.8 \times (\text{celsius}) + 32$	Display celsius, Display fahrenheit

Problem 4

Given the distance of a trip in miles, miles per gallon by the car that is used in the trip and the current price of one gallon of fuel (gas), write a program to determine the fuel required for the trip and the cost spent on the fuel.

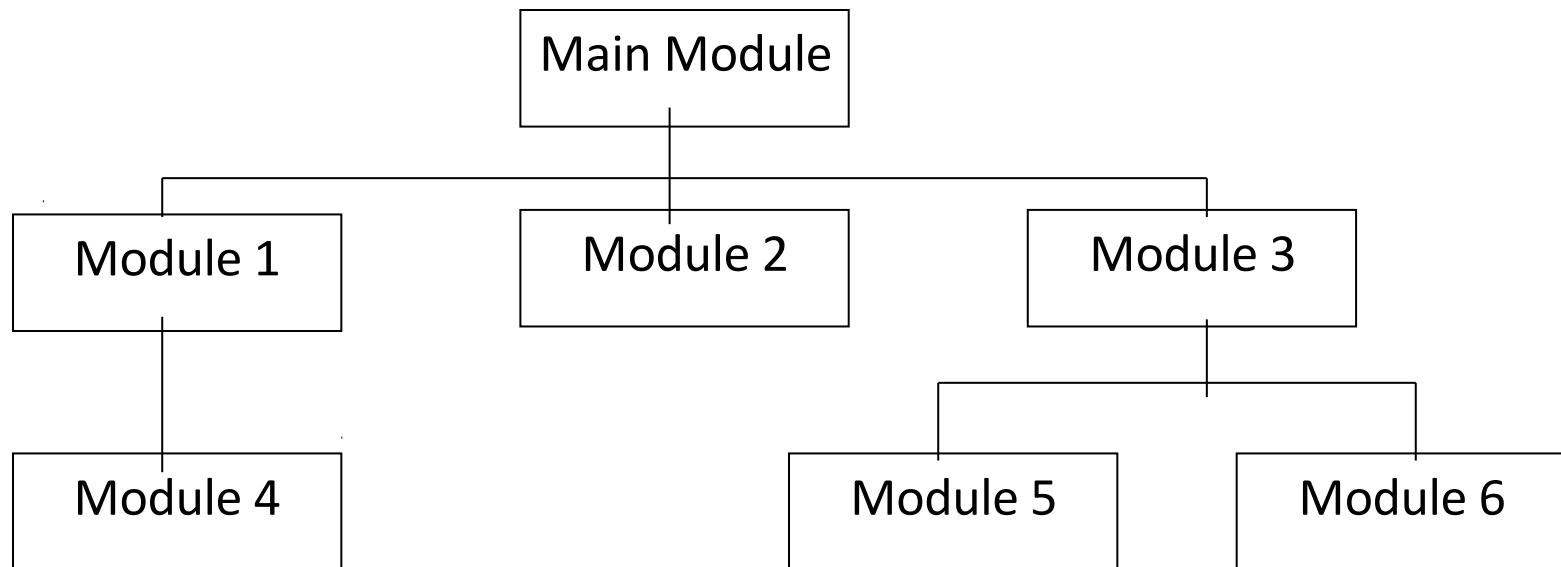
Input	Processing	Output
Distance in miles, miles per gallon, cost per gallon	<p>gas needed = distance / miles per gallon.</p> <p>estimated cost = cost per gallon x gas needed</p>	<p>Display gas needed</p> <p>Display estimated cost</p>

HIPO Chart

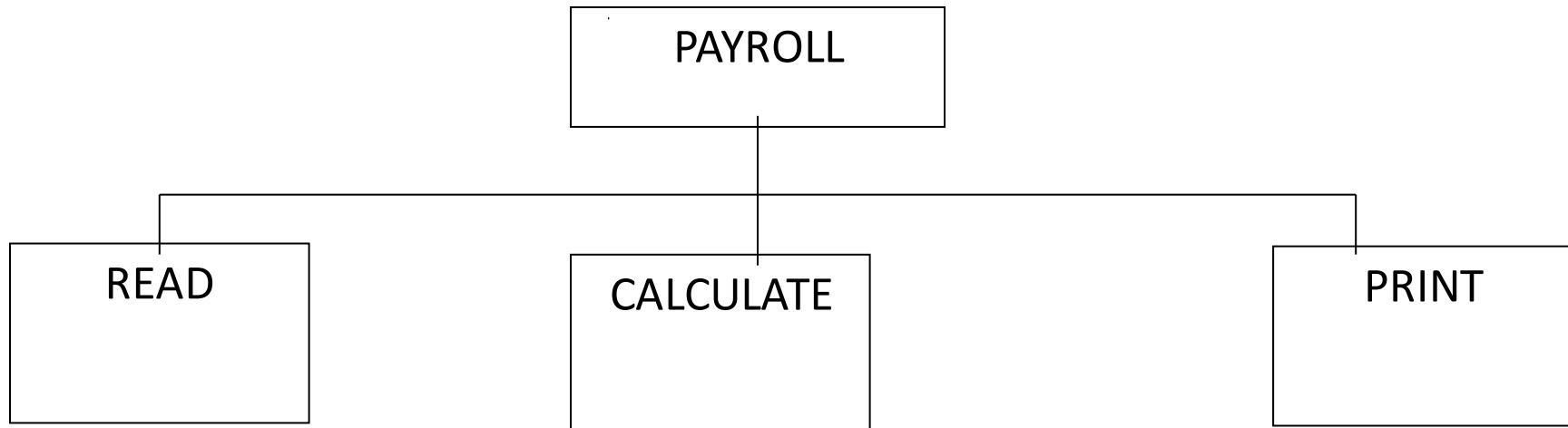
- **Developing the Hierarchy Input Process Output (HIPO) or Interactivity Chart**
 - When problem is normally big and complex.
 - Processing can be divided into subtasks called modules
 - Each module accomplishes one function
 - These modules are connected to each other to show the interaction of processing between the modules

HIPO Chart

- Programming which use this approach (problem is divided into subtasks) is called *Structured Programming*



HIPO Chart for Payroll Problem



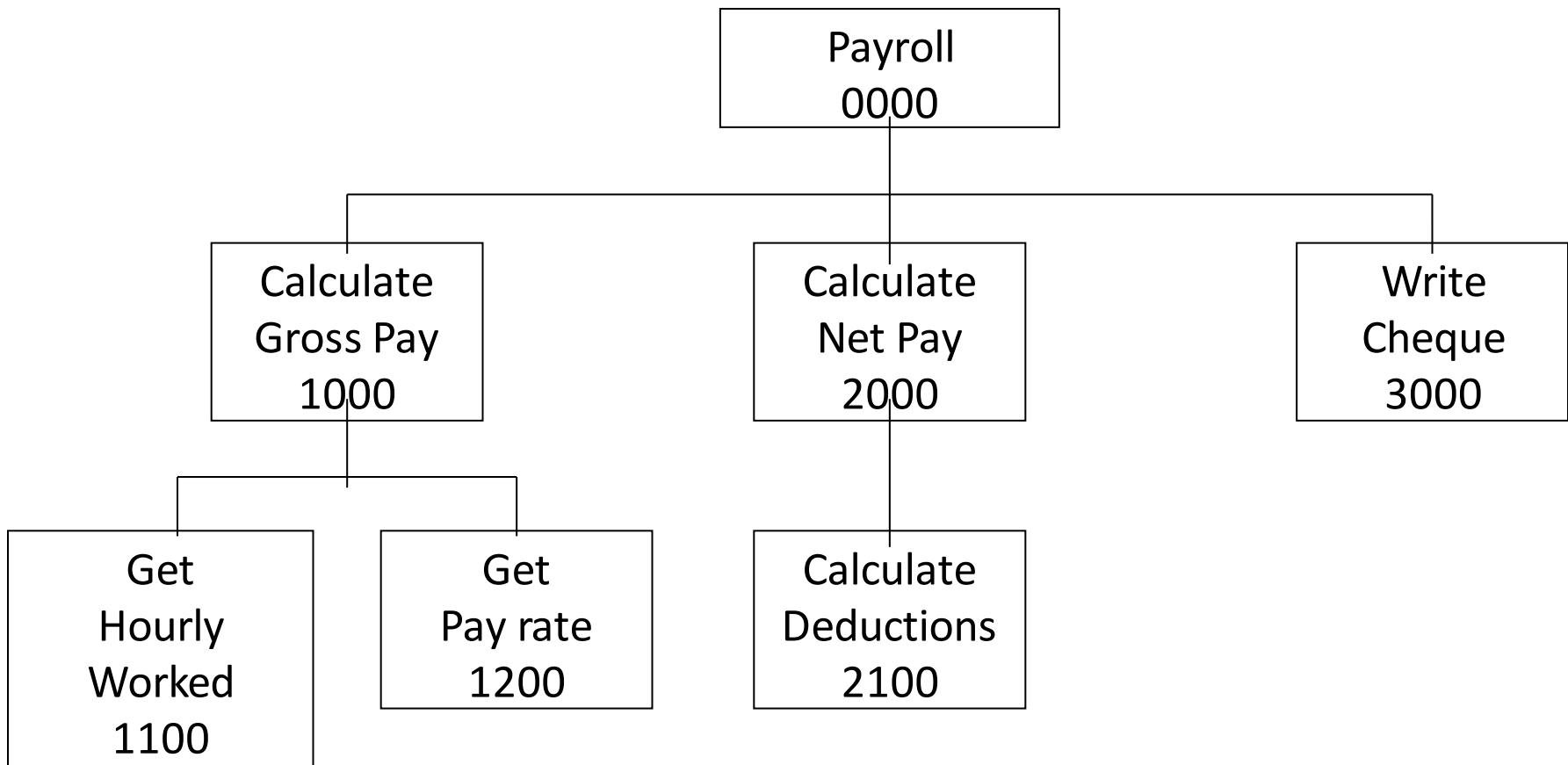
Extended Payroll Problem

- You are required to write a program to calculate both the gross pay and the net pay of every employee of your company. Use the following formulae for calculation:
 - Gross pay = number of hours worked * pay rate
 - Net pay = gross pay – deductions
- The program should also print the cheque that tells the total net pay.

PAC for Extended Payroll Problem

Input	Processing	Output
Number of hours worked, pay rate, deductions	Gross pay = number of hours * pay rate Net pay = Gross pay – deductions	Net pay and write net pay in cheque

HIPO Chart

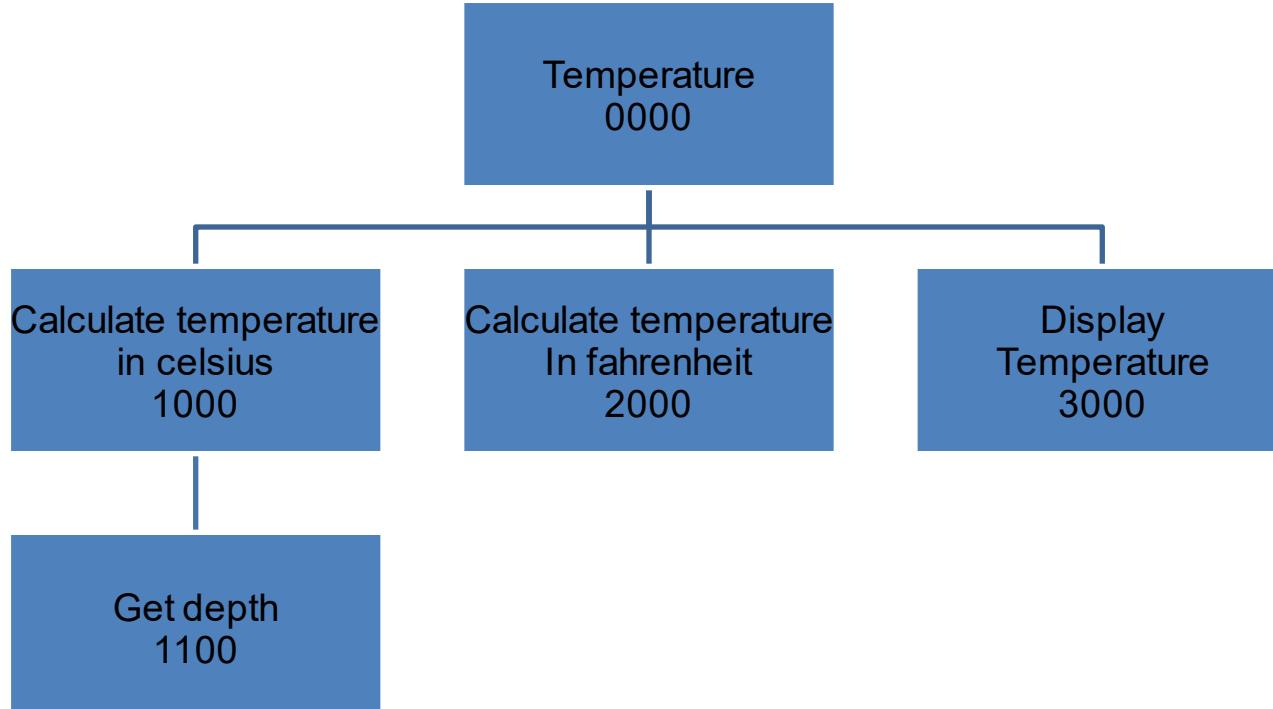


Temperature of Earth

- Write a Hierarchy Input Process Output (HIPO) to compute and display the temperature inside the earth in Celsius and Fahrenheit. The relevant formulas are

$$\text{Celsius} = 10 \times (\text{depth}) + 20$$

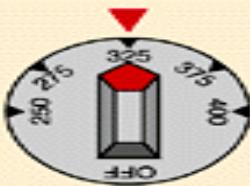
$$\text{Fahrenheit} = 1.8 \times (\text{Celsius}) + 32$$



Algorithm

- Step by step procedure to solve a problem
- In Computer Science following notations are used to represent algorithm
- Flowchart: This is a graphical representation of computation
- Pseudo code: They usually look like English statements but have additional qualities

Heat oven to 325°F



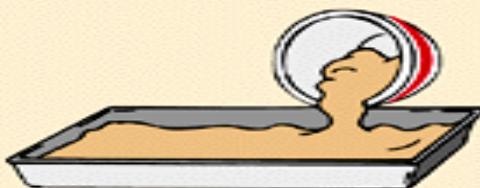
Gather the ingredients



**Mix ingredients thoroughly
in a bowl**



**Pour the mixture into a
baking pan**



**Bake in the oven
50 minutes**

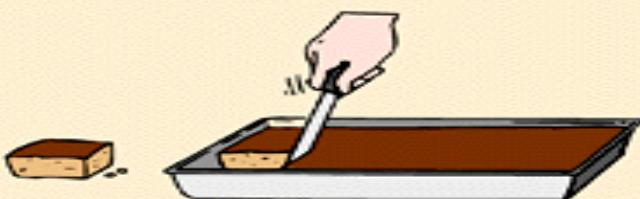


Repeat

Bake 5 minutes more

Until cake top springs back when touched in the center

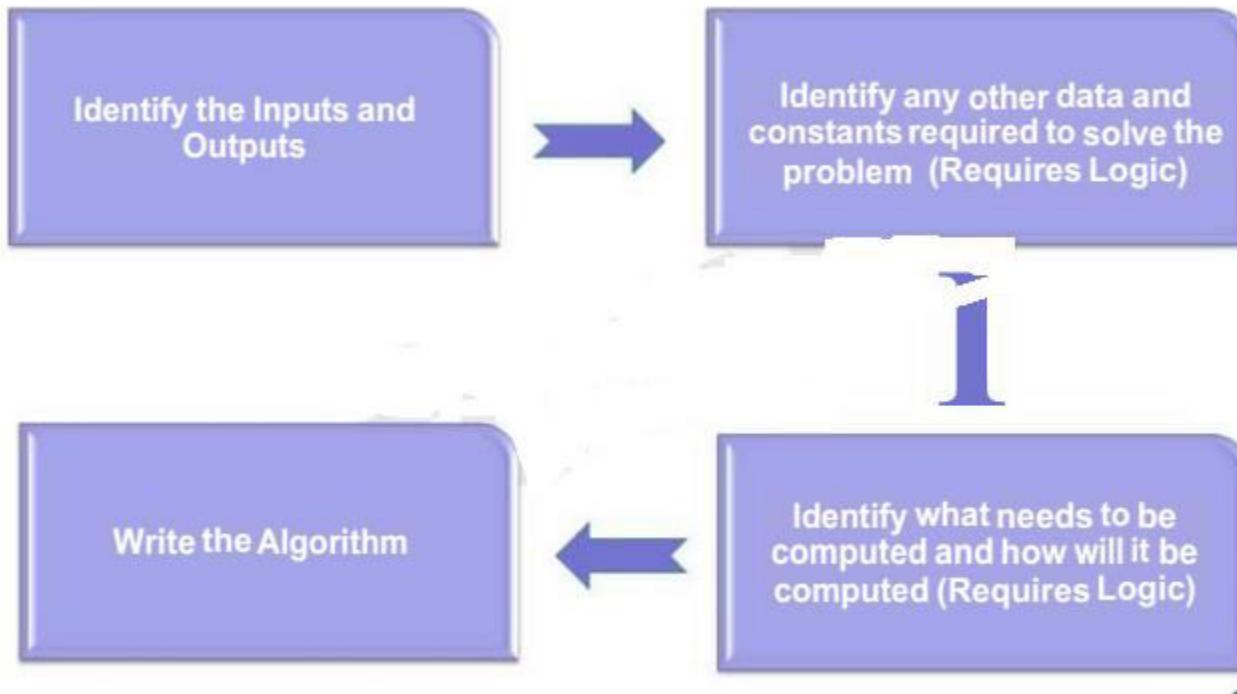
Cool on a rack before cutting



Algorithm

- Algorithms are not specific to any programming language
- An algorithm can be implemented in any programming language
- Use of Algorithms
 - Facilitates easy development of programs
 - Iterative refinement
 - Easy to convert it to a program
 - Review is easier

Steps to Develop an Algorithm



Algorithm for Real life Problem

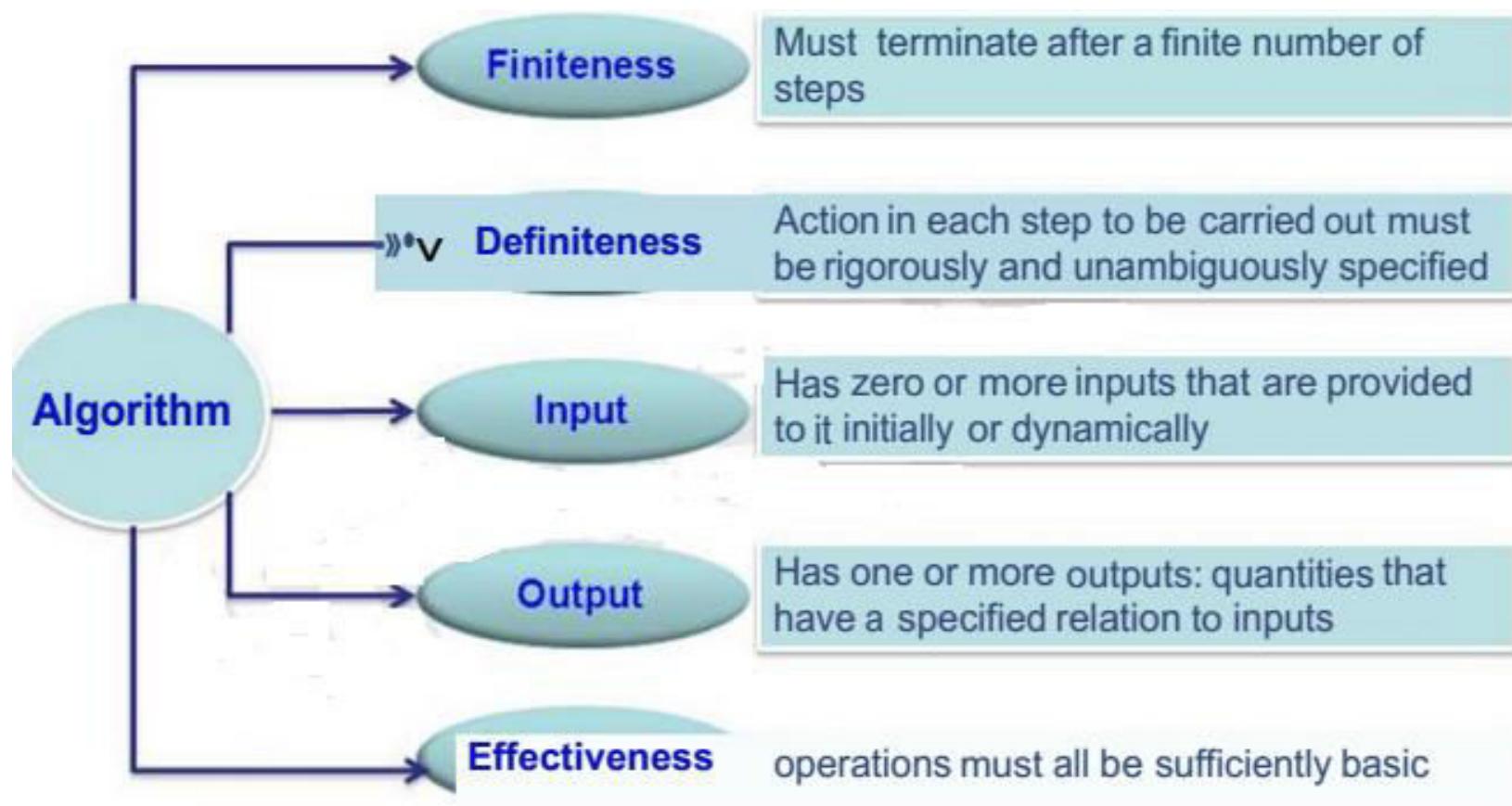
PROBLEM: Heat up a can of soup

ALGORITHM:

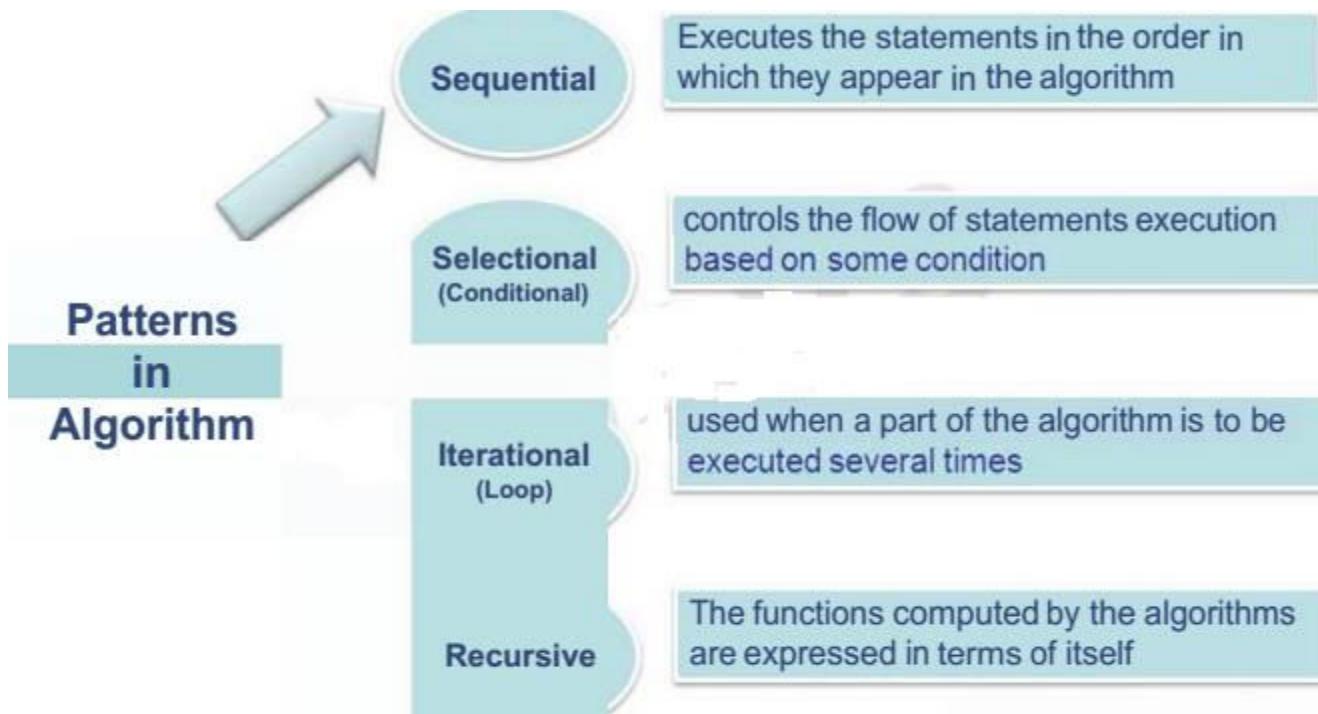
- 1 open can using can opener
- 2 pour contents of can into saucepan
- 3 place saucepan on ring of cooker
- 4 turn on correct cooker ring
- 5 stir soup until warm

may seem a bit of a silly example but it does show us that the order of the events is important since we cannot pour the contents of the can into the saucepan before we open the can.

Properties of an Algorithm



Different patterns in Algorithm



Sequential Algorithms

Algorithm for adding two numbers

Step 1: Read two numbers A and B

Step 2: Let $C = A + B$

Step 3: Display C

Area of a Circle

Step 1 : Read the RADIUS of a circle

Step 2 : Find the square of RADIUS and store it in SQUARE

Step 3 : Multiply SQUARE with 3.14 and store the result in AREA

Step 4: Print AREA

Average Marks

- Find the average marks scored by a student in 3 subjects:

Step 1 : Read Marks1, Marks2, Marks3

Step 2 : Sum = Marks1 + Marks2 + Marks3

Step 3 : Average = Sum / 3

Step 4 : Display Average

Selectional Algorithms

Algorithm for Conditional Problems

PROBLEM: To decide if a fire alarm should be sounded

ALGORITHM:

1 IF fire is detected condition

2 THEN sound fire alarm action

Another example is:-

PROBLEM: To decide whether or not to go to school

ALGORITHM:

1 IF it is a weekday AND it is not a holiday

2 THEN go to school

3 ELSE stay at home

Pass/ Fail and Average

- Write an algorithm to find the average marks of a student. Also check whether the student has passed or failed. For a student to be declared pass, average marks should not be less than 65.

Step 1 : Read Marks1, Marks2, Marks3

Step 2 : Total = Marks1 + Marks2 + Marks3

Step 3 : Average = Total / 3

Step 4 : Set Output = “Student Passed”

Step 5 : if Average < 65 then Set Output =
“Student Failed”

Step 6 : Display Output

Leap Year or Not

Step 1 : Read YEAR

Step 2 : IF ({YEAR%4=0 AND
YEAR%100!=0)OR (YEAR%400=0)})

Display "Year is a leap year"

ELSE

Display "Year is not a leap year"

ENDIF

Algorithm for Iterative Problems

This type of loop keeps on carrying out a command or commands UNTIL a given condition is satisfied, the condition is given with the UNTIL command, for example:-

PROBLEM: To wash a car

ALGORITHM:

1 REPEAT

2 wash with warm soapy water

3 UNTIL the whole car is clean

Iterational Algorithms – Repetitive Structures

- Find the average marks scored by ‘N’ number of students

Step 1 : Read Number Of Students

Step 2 : Let Counter = 1

Step 3 : Read Marks1, Marks2, Marks3

Step 4 : Total = Marks1 + Marks2 + Marks3

Step 5 : Average = Total / 3

Step 6 : Set Output = "Student Passed"

Step 7 : If (Average < 65) then Set Output = "Student Failed"

Step 8 : Display Output

Step 9 : Set Counter = Counter + 1

Step 10 : If (Counter <= NumberOfStudents) then goto step 3

Bigger Problems

- If you are asked to find a solution to a major problem, it can sometimes be very difficult to deal with the complete problem all at the same time.
- For example building a car is a major problem and no-one knows how to make every single part of a car.
- A number of different people are involved in building a car, each responsible for their own bit of the car's manufacture.
- The problem of making the car is thus broken down into smaller manageable tasks.
- Each task can then be further broken down until we are left with a number of step-by-step sets of instructions in a limited number of steps.
- The instructions for each step are exact and precise.

Top Down Design

- Top Down Design uses the same method to break a programming problem down into manageable steps.
- First of all we break the problem down into smaller steps and then produce a Top Down Design for each step.
- In this way sub-problems are produced which can be refined into manageable steps.

Top Down Design for Real Life Problem

PROBLEM: To repair a puncture on a bike wheel.

ALGORITHM:

1. remove the tyre
2. repair the puncture
3. replace the tyre

Step 1: Refinement:

1. Remove the tyre

1.1 turn bike upside down

1.2 lever off one side of the tyre

1.3 remove the tube from inside the tyre

Step 2: Refinement:

2. Repair the puncture Refinement:

2.1 find the position of the hole in the tube

2.2 clean the area around the hole

2.3 apply glue and patch

Step 3: Refinement:

3. Replace the tyre Refinement:

3.1 push tube back inside tyre

3.2 replace tyre back onto wheel

3.3 blow up tyre

3.4 turn bike correct way up

Still more Refinement:

Sometimes refinements may be required to some of the sub-problems, for example if we cannot find the hole in the tube, the following refinement can be made to 2.1:-

Still more Refinement:

Step 2.1: Refinement

2.1 Find the position of the hole in the tube

2.1.1 WHILE hole cannot be found

2.1.2 Dip tube in water

2.1.3 END WHILE

Flow Chart and Phases of Making an Executable Code

Drawing Flowcharts

- Flowchart is the graphic representations of the individual steps or actions to implement a particular module
- Flowchart can be likened to the blueprint of a building
- An architect draws a blueprint before beginning construction on a building, so the programmer draws a flowchart before writing a program
- Flowchart is independent of any programming language.

Flow Charts

A flow chart is an organized combination of shapes, lines and text that graphically illustrate a process or structure.

Symbols used



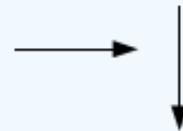
Start/Stop



Process



Input/Output (Data)



Flow Lines

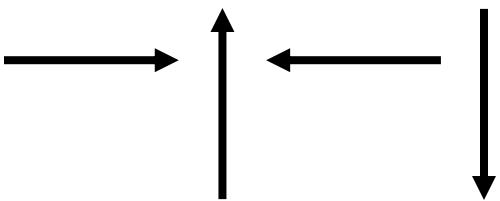
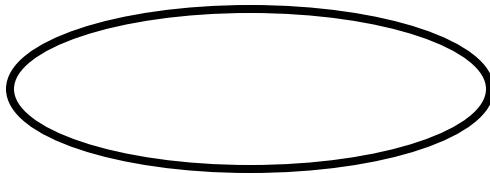


Decision symbol

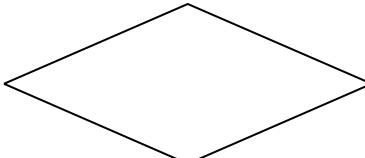
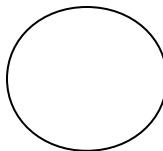


Connector

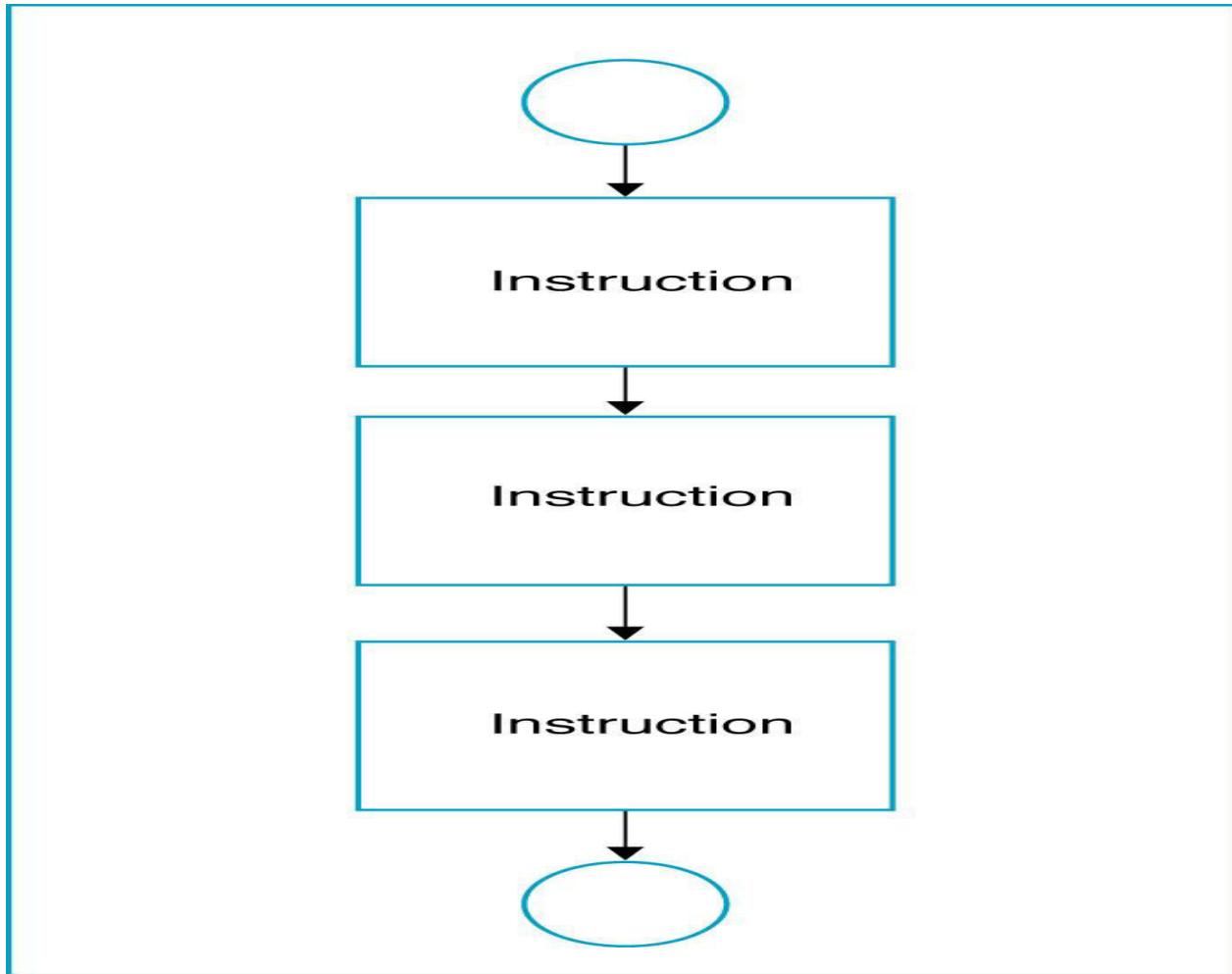
PRE-PROGRAMMING PHASE

Symbol	Function
	Show the direction of data flow or logical solution.
	Indicate the beginning and ending of a set of actions or instructions (logical flow) of a module or program.
	Indicate a process, such as calculations, opening and closing files.

PRE-PROGRAMMING PHASE

	Indicate input to the program and output from the program.
	Use for making decision. Either True or False based on certain condition.
	Use for doing a repetition or looping of certain steps.
	Connection of flowchart on the same page.
	Connection of flowchart from page to page.

Sequential Logic Structure



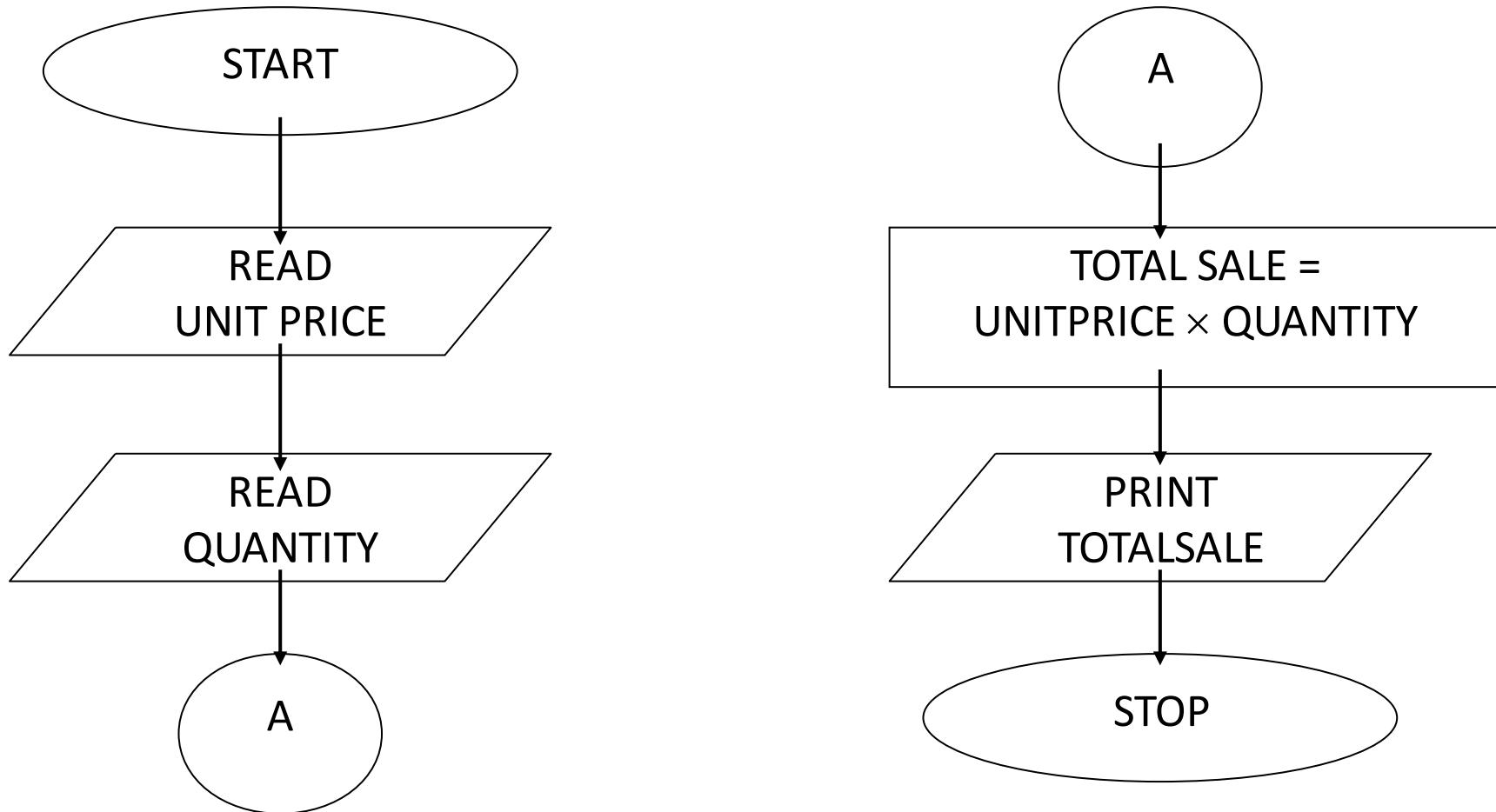
Sale Problem

Given the unit price of a product and the quantity of the product sold, draw a flowchart to calculate and print the total sale.

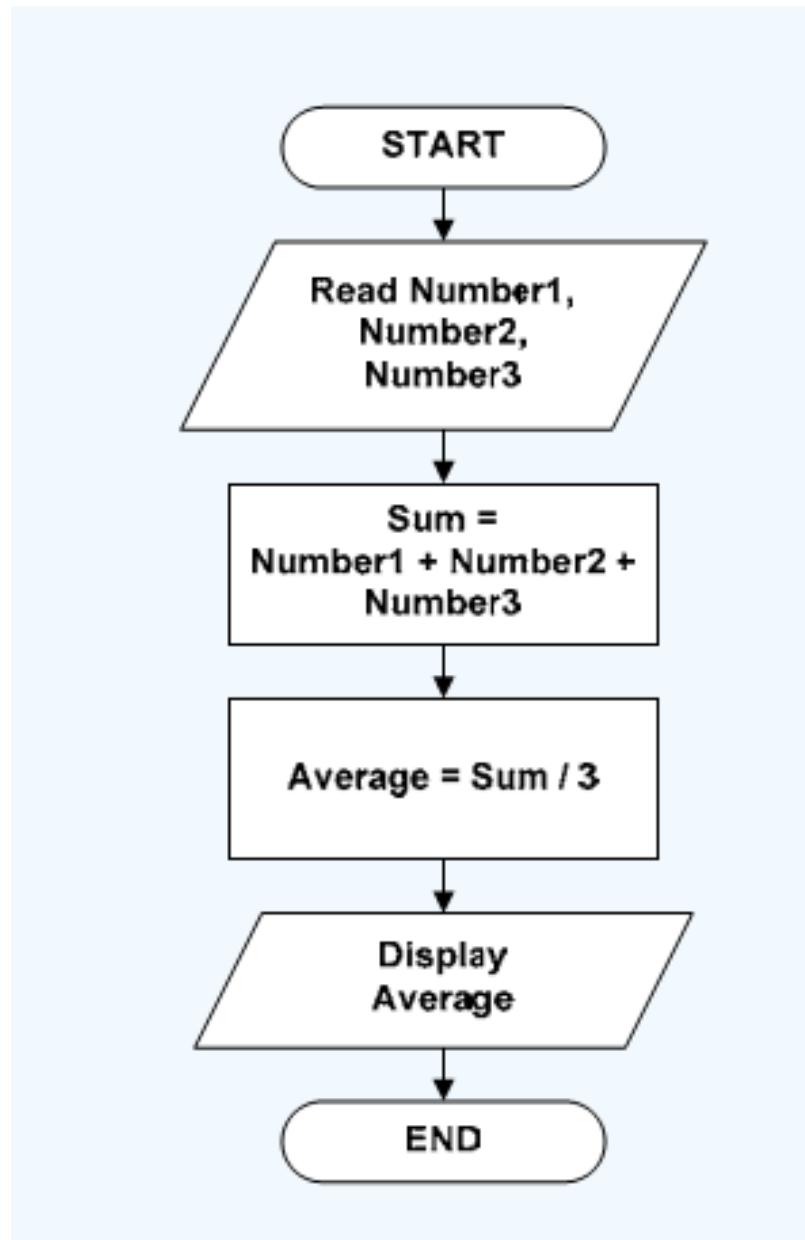
Solution: Stepwise Analysis of the Sale Problem

- Read the unit price and the quantity
- Calculate total sale = unit price and quantity
- Print total sale

PRE-PROGRAMMING PHASE



Find the average of three numbers

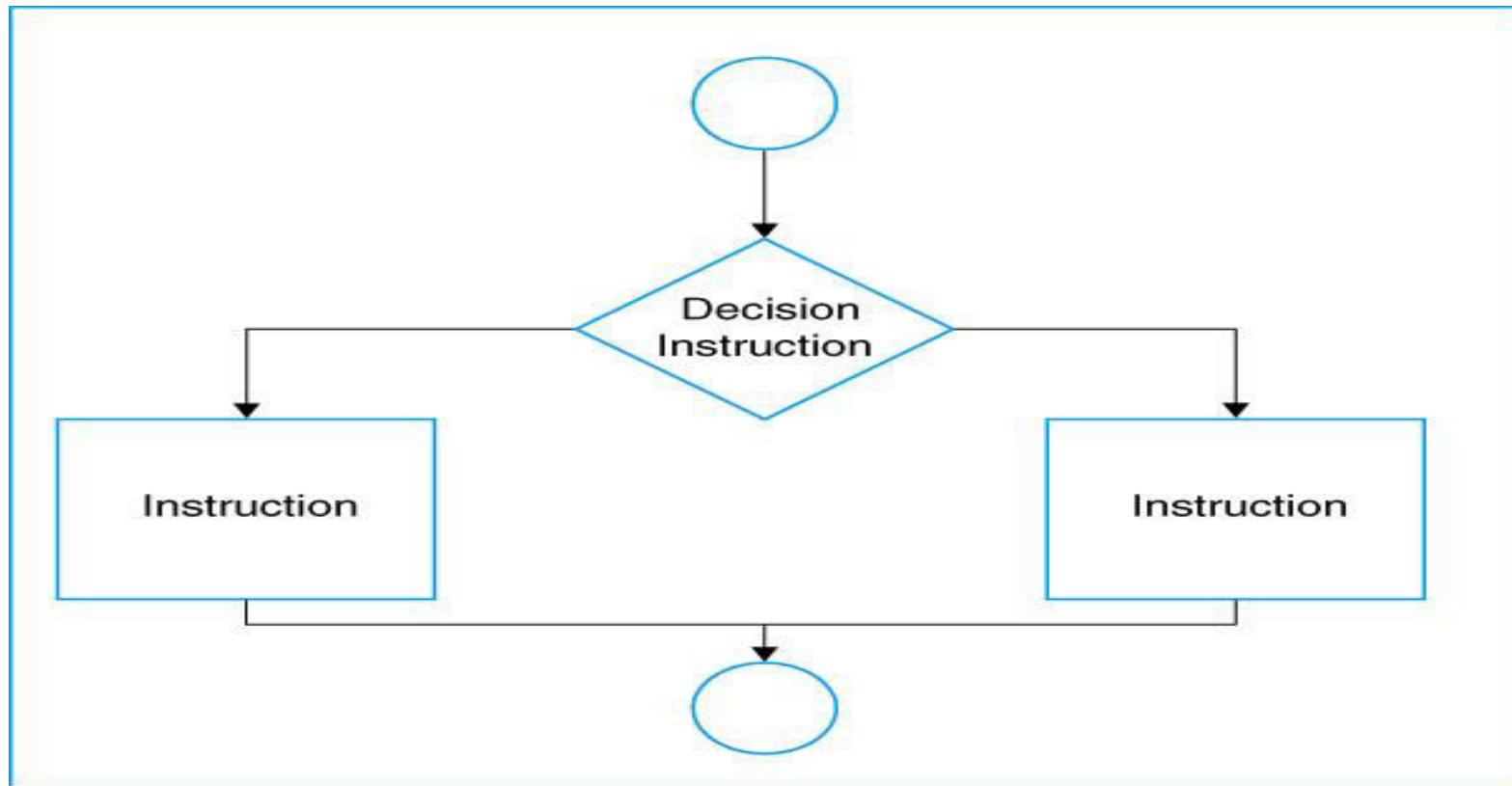


The Decision Logic Structure

- Implements using the IF/THEN/ELSE instruction.
- Tells the computer that IF a condition is true, THEN execute a set of instructions, or ELSE execute another set of instructions
- ELSE part is optional, as there is not always a set of instructions if the conditions are false.
- Algorithm:

```
IF <condition(s)> THEN  
    <TRUE instruction(s)>  
ELSE  
    <FALSE instruction(s)>
```

Decision Logic Structure



Examples of conditional expressions

- $A < B$ (A and B are the same data type – either numeric, character, or string)
- $X + 5 \geq Z$ (X and Z are numeric data)
- $E < 5$ or $F > 10$ (E and F are numeric data)

Conditional Pay Calculation

- Assume you are calculating pay at an hourly rate, and overtime pay(over 40 hours) at 1.5 times the hourly rate.
 - IF the hours are greater than 40, THEN the pay is calculated for overtime, or ELSE the pay is calculated in the usual way.

Example Decision Structure

Algorithm	Flowchart
<pre>IF HOURS > 40 THEN PAY = RATE * (40 + 1.5 * (HOURS - 40)) ELSE PAY = RATE * HOURS</pre>	<pre>graph TD A((A)) --> D{IF HOURS > 40} D -- F --> R1[PAY = RATE * HOURS] R1 --> B((B)) D -- T --> R2[PAY = RATE * (40 + 1.5 * (HOURS - 40))] R2 --> B</pre>

Note: For all flowcharts with decision blocks, T = TRUE and F = FALSE

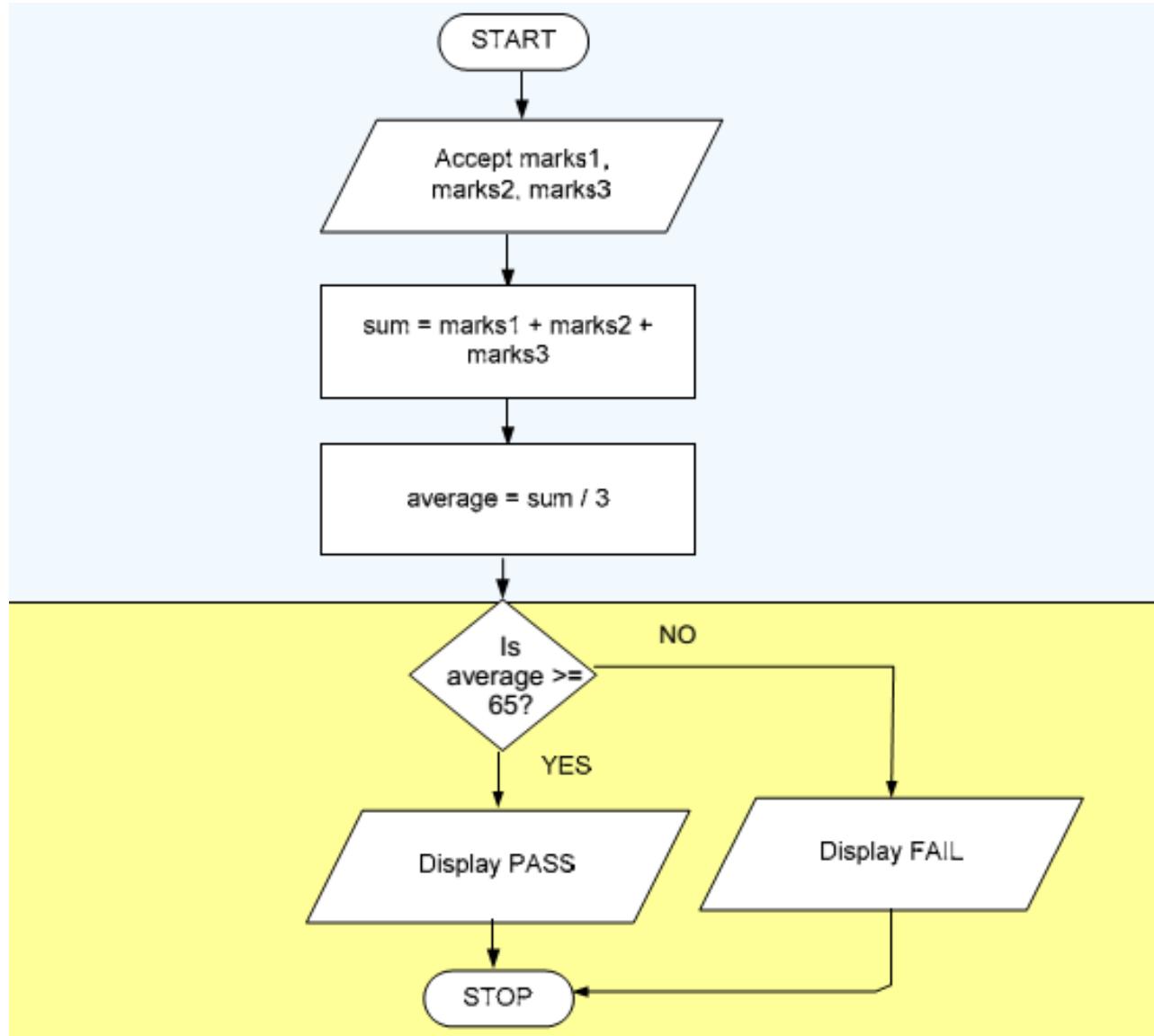
NESTED IF/THEN/ELSE INSTRUCTIONS

- Multiple decisions.
- Instructions are sets of instruction in which each level of a decision is embedded in a level before it.

NESTED IF/THEN/ELSE INSTRUCTIONS

Algorithm	Flowchart
<pre> IF PAYTYPE = "HOURLY" THEN IF HOURS > 40 THEN PAY = RATE * (40 + 1.5 * (HOURS - 40)) ELSE PAY = RATE * HOURS ELSE PAY = SALARY END IF END IF </pre>	<pre> graph TD A((A)) --> D1{IF PAYTYPE = "HOURLY"} D1 -- F --> R1[PAY = SALARY] R1 --> B((B)) D1 -- T --> D2{IF HOURS > 40} D2 -- F --> R2[PAY = RATE * HOURS] R2 --> B D2 -- T --> R3[PAY = RATE * (40 + 1.5 * (HOURS - 40))] R3 --> B </pre>

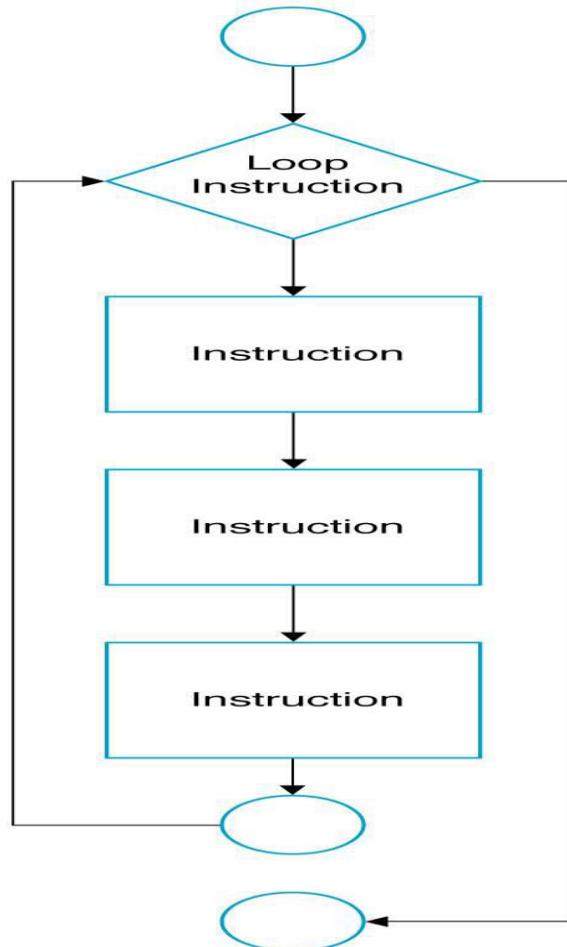
Flow Chart - Selectional



Iterational Structure

- Repeat structure
- To solve the problem that doing the same task over and over for different sets of data
- Types of loop:
 - WHILE loop
 - Do..WHILE loop
 - Automatic-Counter Loop

Loop Logic Structure



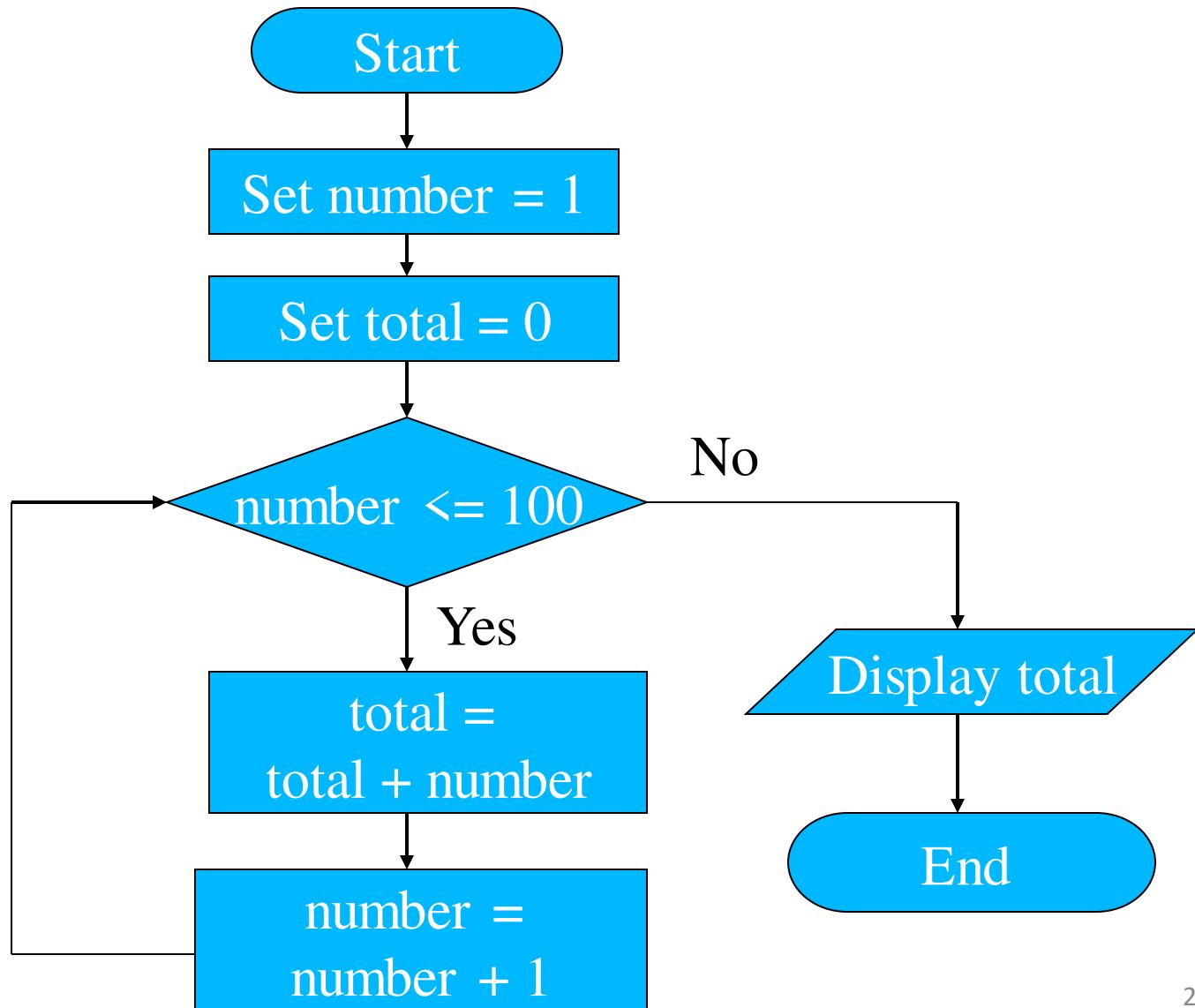
WHILE loop

Algorithm	Flowchart
<pre>100 IF <CONDITION(S)> THEN INSTRUCTION INSTRUCTION GOTO 100</pre>	<pre>graph TD A((A)) --> IF{IF <CONDITION(S)>} IF -- T --> IN1[INSTRUCTION] IN1 --> IN2[INSTRUCTION] IN2 -- GOTO --> IF IF -- F --> B((B))</pre>

WHILE loop

- Do the loop body if the condition is true.
- Example: Get the sum of 1, 2, 3, ..., 100.
 - Algorithm:
 - Set the number = 1
 - Set the total = 0
 - While (number <= 100)
 - total = total + number
 - number = number + 1
 - End While
 - Display total

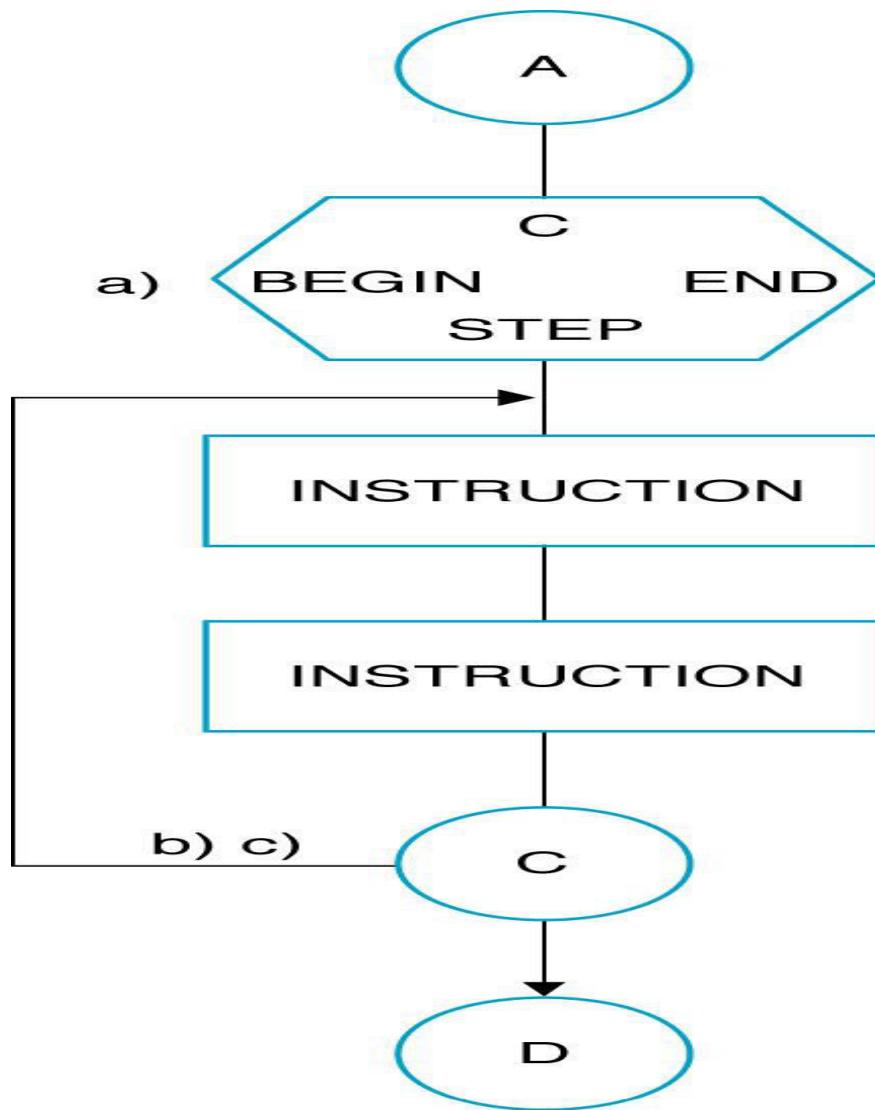
WHILE loop



Automatic Counter Loop

- Use variable as a counter that starts counting at a specified number and increments the variable each time the loop is processed.
- The beginning value, the ending value and the increment value may be constant.
- They should not be changed during the processing of the instruction in the loop.

Automatic-Counter Loop

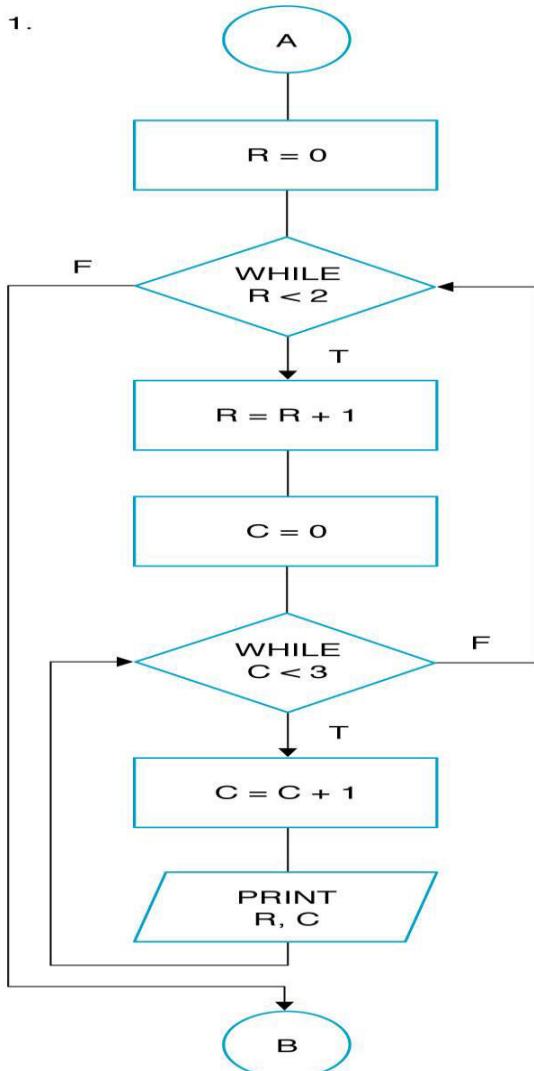


Automatic-Counter Loop

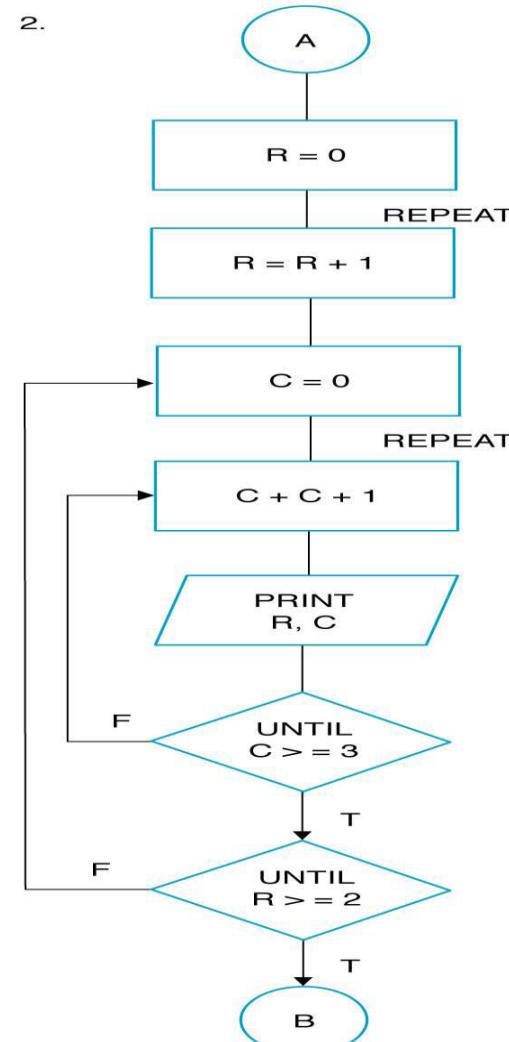
Algorithm	Flowchart
<p>AVERAGE AGE</p> <ol style="list-style-type: none">1. SUM = 02. COUNTER = 03. LOOP: J = 1 TO 12 SUM = SUM + AGE COUNTER = COUNTER + 1 LOOP-END: J4. AVERAGE = SUM/COUNTER5. PRINT COUNTER, AVERAGE6. END	<pre>graph TD; START([START]) --> SUM0[SUM = 0]; SUM0 --> COUNTER0[COUNTER = 0]; COUNTER0 --> Decision{J}; Decision -- 1 --> ENTER[ENTER AGE]; Decision -- 12 --> SUM0; ENTER --> SUM[SUM = SUM + AGE]; SUM --> COUNTER[COUNTER = COUNTER + 1]; COUNTER --> Decision; Decision -- J --> ENTER; COUNTER --> AVERAGE[AVERAGE = SUM / COUNTER]; AVERAGE --> PRINT[/PRINT COUNTER, AVERAGE/]; PRINT --> END([END]);</pre>

NESTED LOOP

1.

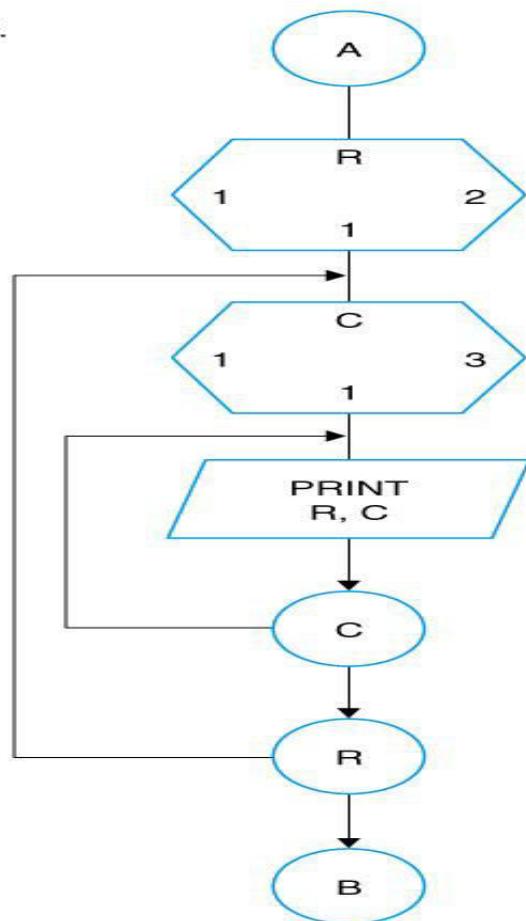


2.

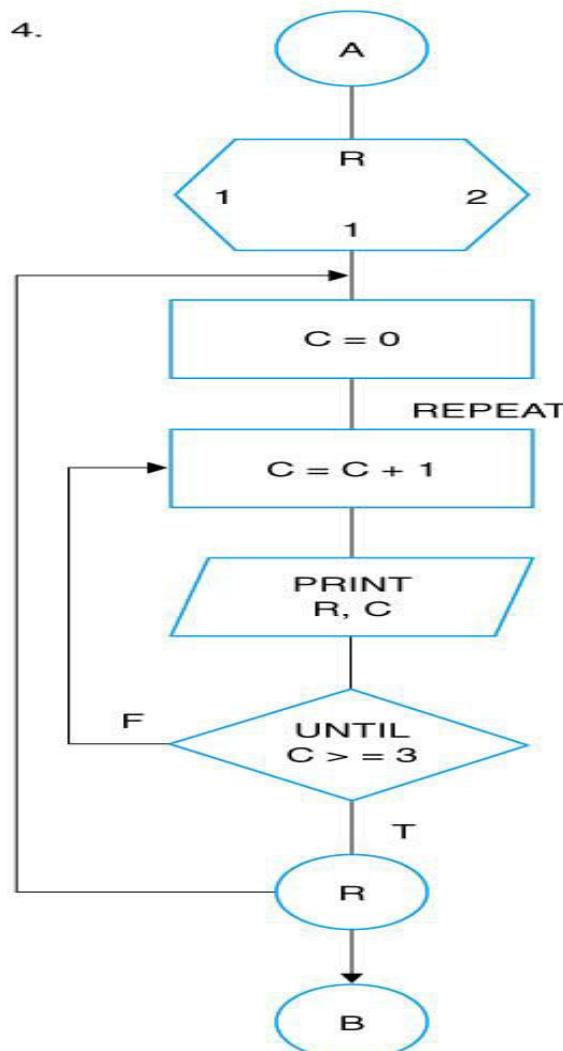


NESTED LOOP

3.



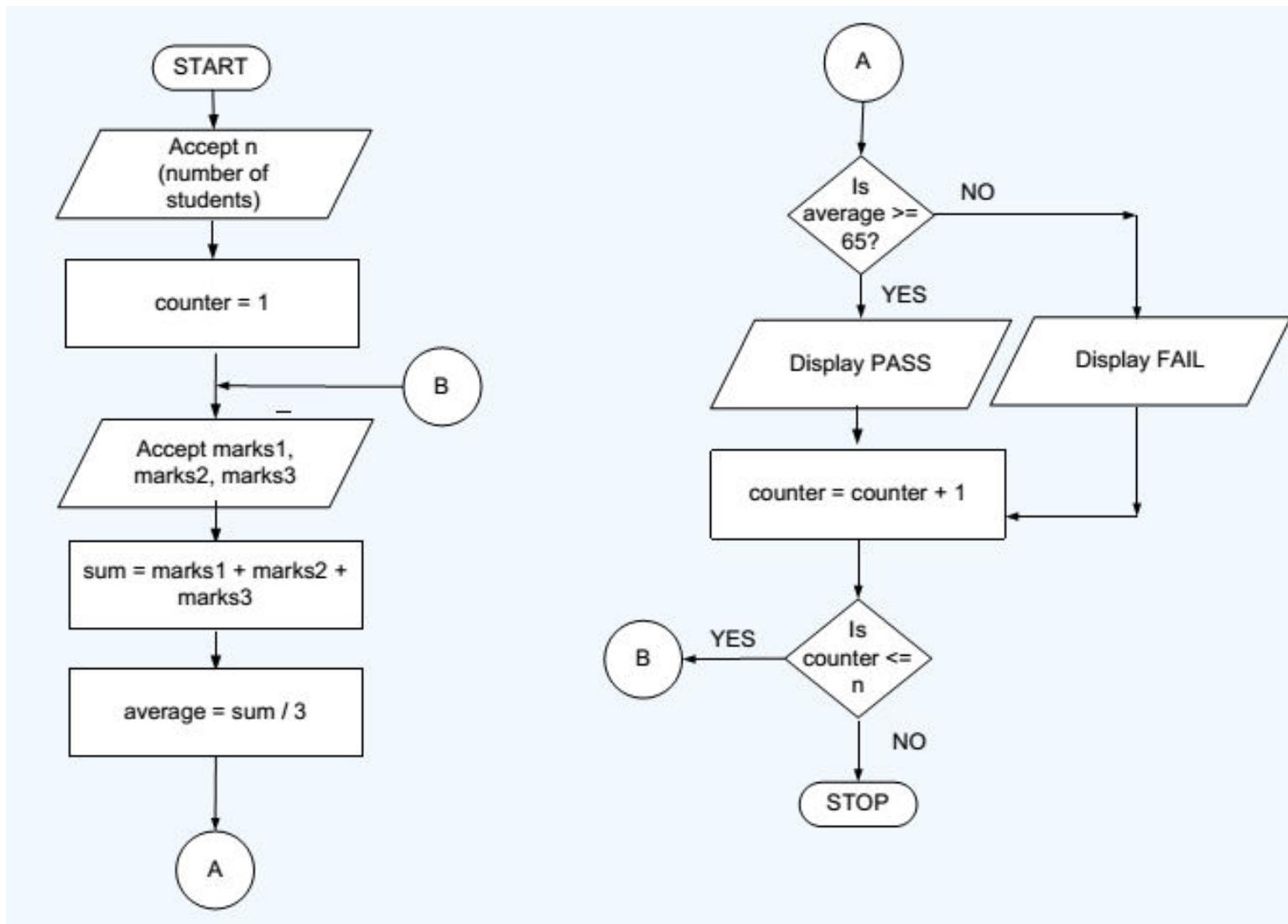
4.



Example (Iterational)

- Write a program to find the average of marks scored in three subjects for ‘N’ students. And then test whether he passed or failed. For a student to pass, average should not be less than 65.

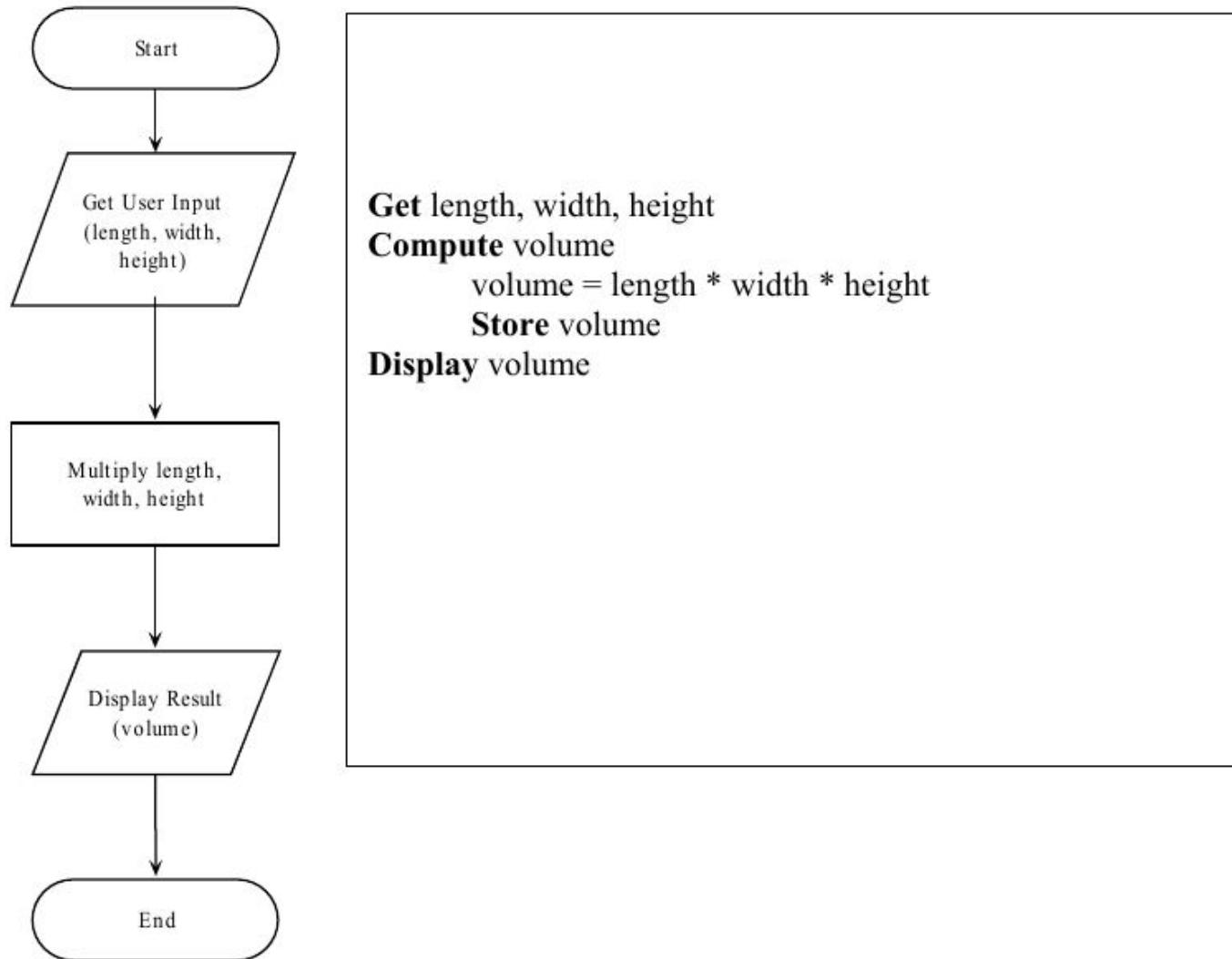
Flow Chart Iterational



Tool demo

- Yed tool shall be used for giving demo

Pseudocode – Partial English and Programming Language terms

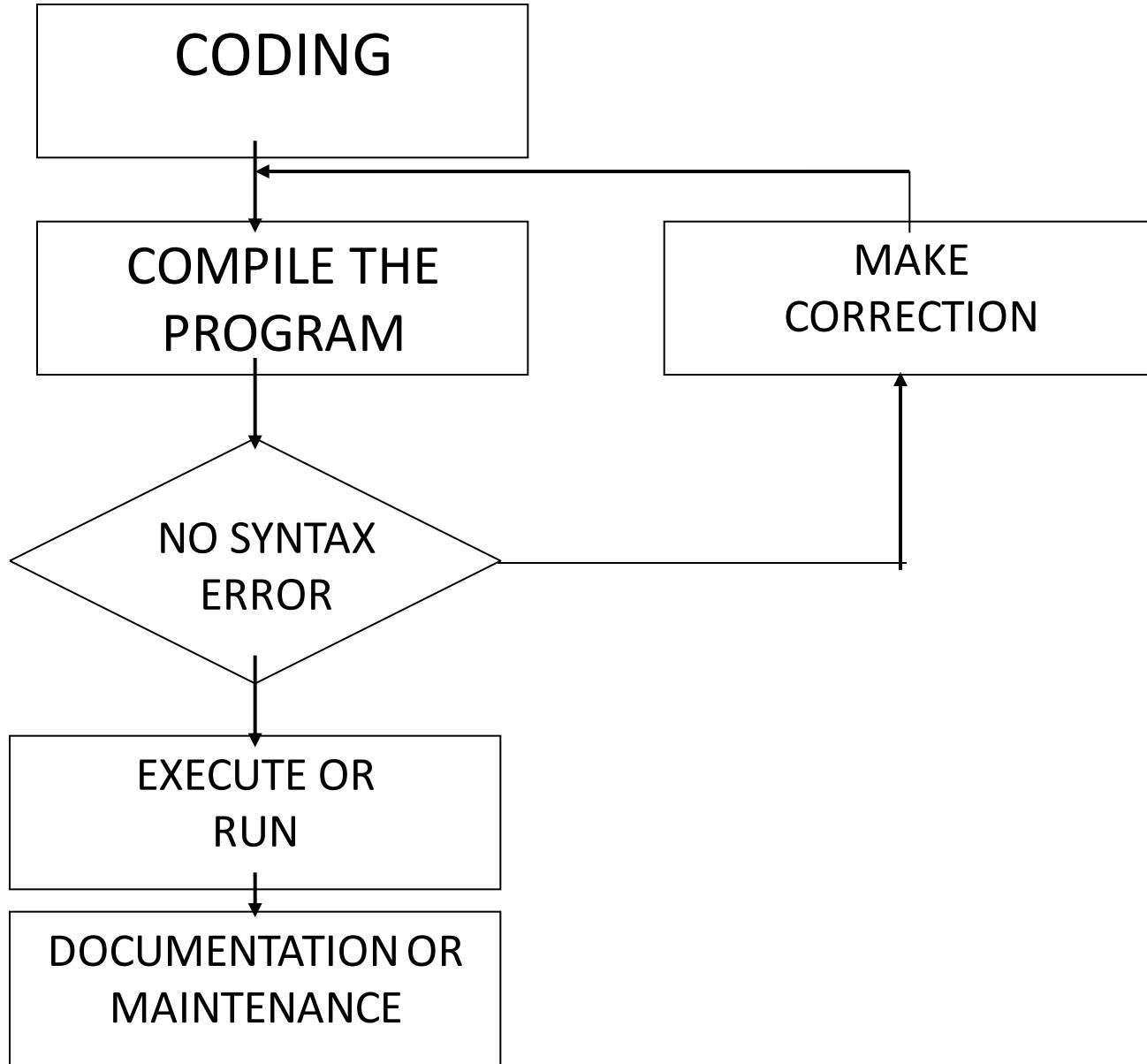


Programming Or Implementation Phase

- Transcribing the logical flow of solution steps in flowchart or algorithm to program code and run the program code on a computer using a programming language.
- Programming phase takes 5 stages:
 - Coding.
 - Compiling.
 - Debugging.
 - Run or Testing.
 - Documentation and maintenance.

Programming Or Implementation Phase

- Once the program is coded using one of the programming language, it will be compiled to ensure there is no syntax error.
- Syntax free program will then be executed to produce output and subsequently maintained and documented for later reference.



Coding

- Translation or conversion of each operation in the flowchart or algorithm (pseudocode) into a computer-understandable language.
- Coding should follow the format of the chosen programming language.

Compiling and Debugging

- Compiling - Translates a program written in a particular high-level programming language into a form that the computer can understand
- Compiler checks the program code so that any part of source code that does not follow the format or any other language requirements will be flagged as syntax error.
- This syntax error is also called bug, when error is found the programmer will debug or correct the error and then recompile the source code again
- Debugging process is continued until there is no more error in program

Testing

- The program code that contains no more error is called executable program. It is ready to be tested.
- When it is tested, the data is given and the result is verified so that it should produced output as intended.
- Though the program is error free, sometimes it does not produced the right result. In this case the program faces logic error.
- Incorrect sequence of instruction is an example that causes logic error.

Documentation and Maintenance

- When the program is thoroughly tested for a substantial period of time and it is consistently producing the right output, it can be documented.
- Documentation is important for future reference. Other programmer may take over the operation of the program and the best way to understand a program is by studying the documentation.
- Trying to understand the logic of the program by looking at the source code is not a good approach.
- Studying the documentation is necessary when the program is subjected to enhancement or modification.
- Documentation is also necessary for management use as well as audit purposes.

Best Practices

Develop efficient computer solution to problems:

1. Use Modules
2. Use four logic structures
 - a. Sequential structure
 - Executes instructions one after another in a sequence.
 - b. Decision structure
 - Branches to execute one of two possible sets of instructions.
 - c. Loop structure
 - Executes set of instruction many times.
 - d. Case structure
 - Executes one set of instructions out of several sets.
3. Eliminate rewriting of identical process by using modules.
4. Use techniques to improve readability including four logic structure, proper naming of variables, internal documentation and proper indentation.

Introduction to Python

Need of programming Languages

- Computers can execute tasks very rapidly and assist humans
- Programming languages – Helps for communication between human and machines.
- They can handle a greater amount of input data.
- But they cannot design a strategy to solve problems for you

Python – Why?

- Simple syntax
- Programs are clear and easy to read
- Has powerful programming features
- Companies and organizations that use Python include YouTube, Google, Yahoo, and NASA.
- Python is well supported and freely available at www.python.org.

Python – Why?

- Guido van Rossum – Creator
- Released in the early 1990s.
- Its name comes from a 1970s British comedy sketch television show called *Monty Python's Flying Circus*.

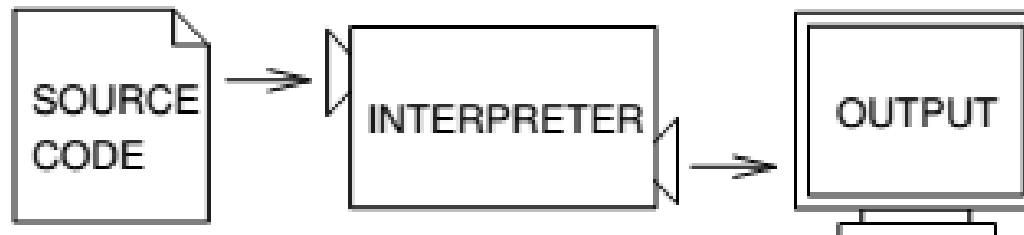


Python....

- **High-level language**; other high-level languages you might have heard of are C, C++, Perl, and Java.
- There are also **low-level languages**, sometimes referred to as “machine languages” or “assembly languages.”
- Computers can only run programs written in low-level languages.
- So programs written in a high-level language have to be processed before they can run.

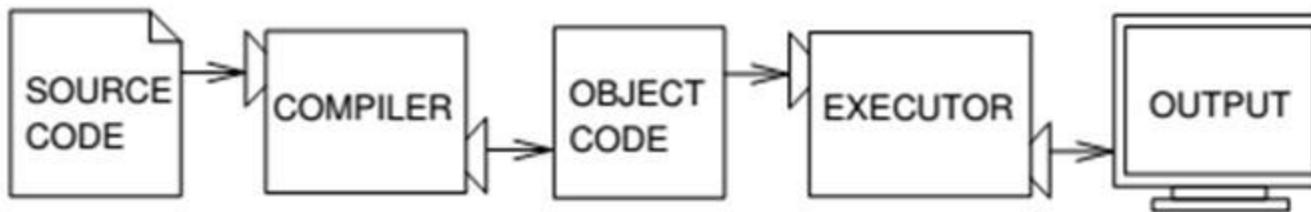
Python....

- Two kinds of program translator to convert from high-level languages into low-level languages:
 - **Interpreters**
 - **Compilers.**
- An interpreter processes the program by reading it line by line



Python....

- Compiler translates completely a high level program to low level it completely before the program starts running.
- High-level program is called **source code**
- Translated program is called the **object code** or the **executable**.



A compiler translates source code into object code, which is run by a hardware executor.

Python....

- Python is considered an interpreted language because Python programs are executed by an interpreter.
- There are two ways to use the interpreter:
interactive mode and script mode.

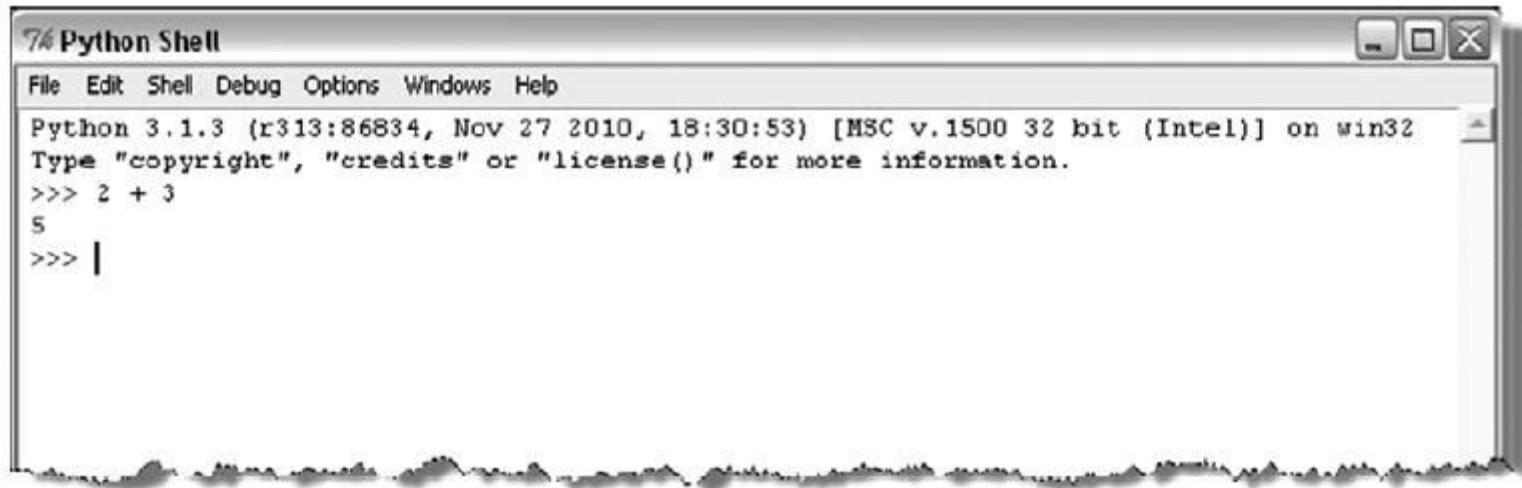
Python....Interactive mode

In interactive mode, you type Python programs and the interpreter displays the result:

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

The shell prompt, `>>>`, is the **prompt** the interpreter uses to indicate that it is ready. If you type `1 + 1`, the interpreter replies 2.

Python Shell



Python.... Script Mode

- Can also store code in a file and use the interpreter to execute the contents of the file, which is called a **script**.
- Python scripts have names that end with .py.
- Interactive mode is convenient for testing small pieces of code because you can type and execute them immediately.
- But for anything more than a few lines, should save your code as a script so you can modify and execute it in future.

Python....Script Mode ...

File name : first.py

```
print(4+3)
print(4-3)
print(4>3)
print("Hello World")
```

```
C:\Python34\python.exe C:/Users/sathisbsk/first.py
7
1
True
Hello World

Process finished with exit code 0
```

Problem

- Little Bob loves chocolate, and he goes to a store with Rs. N in his pocket. The price of each chocolate is Rs. C . The store offers a discount: for every M wrappers he gives to the store, he gets one chocolate for free. This offer is available only once. How many chocolates does Bob get to eat?

PAC For Chocolate Problem

Input	Processing	Output
Amount in hand, N Price of one chocolate, C Number of wrappers for a free chocolate, M	Number of Chocolates $P = \text{Quotient of } N / C$ Free chocolate F = Quotient of P/M	Total number of chocolates got by Bob

Pseudocode

- READ N and C
- COMPUTE num_of_chocolates as N/C
- CALCULATE returning_wrapper as number of chocolates/m
- TRUNCATE decimal part of returning_wrapper
- COMPUTE Chocolates_recieved as num_of_chocolates + returning_wrapper
- PRINT Chocolates_recieved

Knowledge Required

- Following knowledge is required in Python to write a code to solve the above problem
- Read input from user
- Data types in Python
- Perform arithmetic calculations
- Write output

What is an Identifier?

- An **identifier** is a sequence of one or more characters used to name a given program element.
- In Python, an identifier may contain letters and digits, but cannot begin with a digit.
- Special underscore character can also be used
- Example : line, salary, emp1, emp_salary

Rules for Identifier

- Python is *case sensitive*, thus, Line is different from line.
- Identifiers may contain letters and digits, but cannot begin with a digit.
- The underscore character, `_`, is also allowed to aid in the readability of long identifier names. It should not be used as the *first* character
- Spaces are not allowed as part of an identifier.

Identifier Naming

Valid Identifiers	Invalid Identifiers	Reason Invalid
totalSales	'totalSales'	quotes not allowed
totalsales	total sales	spaces not allowed
salesFor2010	2010Sales	cannot begin with a digit
sales_for_2010	_2010Sales	should not begin with an underscore

Keywords

- A **keyword** is an identifier that has pre-defined meaning in a programming language.
- Therefore, keywords cannot be used as “regular” identifiers. Doing so will result in a syntax error, as demonstrated in the attempted assignment to keyword and below,

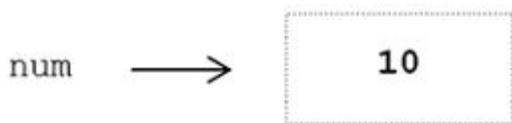
```
>>> and = 10  
SyntaxError: invalid syntax
```

Keywords in Python

and	as	assert	break	class	continue	def
del	elif	else	except	finally	for	from
global	if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try	while
with	yield	false	none	true		

Variables and Identifiers

- A **variable** is a name (identifier) that is associated with a value.
- A simple description of a variable is “a name that is assigned to a value,”



- Variables are assigned values by use of the **assignment operator**,
`num = 10`
`num = num + 1`

Comments

- Meant as documentation for anyone reading the code
- Single-line comments begin with the hash character ("#") and are terminated by the end of line.
- Python ignores all text that comes after # to end of line
- Comments spanning more than one line are achieved by inserting a multi-line string (with """" as the delimiter one each end) that is not used in assignment or otherwise evaluated, but sits in between other statements.
- *#This is also a comment in Python*
- """" This is an example of a multiline comment that spans multiple lines ... """

Literals

- A **literal** is a sequence of one or more characters that stands for itself.

Numeric literal

- A **numeric literal** is a literal containing only the digits 0–9, an optional sign character (+ or -), and a possible decimal point. (The letter e is also used in exponential notation).
- If a numeric literal contains a decimal point, then it denotes a **floating-point value**, or “**float**” (e.g., 10.24); otherwise, it denotes an **integer value** (e.g., 10).
- *Commas are never used in numeric literals*

Numeric Literals in Python

Numeric Literals						
integer values	floating-point values					incorrect
5	5.	5.0	5.125	0.0005	5000.125	5,000.125
2500	2500.	2500.0		2500.125		2,500 2,500.125
+2500	+2500.	+2500.0		+2500.125		+2,500 +2,500.125
-2500	-2500.	-2500.0		-2500.125		-2,500 -2,500.125

Since numeric literals without a provided sign character denote positive values, an explicit positive sign character is rarely used.

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> 1024
```

```
???
```

```
>>> 1,024
```

```
???
```

```
>>> -1024
```

```
???
```

```
>>> 0.1024
```

```
???
```

```
>>> .1024
```

```
???
```

```
>>> 1,024.46
```

```
???
```

String Literals

- **String literals**, or “strings,” represent a sequence of characters,
`'Hello' 'Smith, John' "Baltimore, Maryland
21210"`
- In Python, string literals may be surrounded by a matching pair of either single ('') or double ("") quotes.
- `>>> print('Welcome to Python!')`
`>>>Welcome to Python!`

String Literal Values

```
'A'  
'jsmith16@mycollege.edu'  
"Jennifer Smith's Friend"  
' '  
''
```

- a string consisting of a single character
- a string containing non-letter characters
- a string containing a single quote character
- a string containing a single blank character
- the empty string

Note

If this string were delimited with single quotes, the apostrophe (single quote) would be considered the matching closing quote of the opening quote, leaving the last final quote unmatched,

'Jennifer Smith's Friend' ... *matching quote?*

Thus, Python allows the use of more than one type of quote for such situations. (The convention used in the text will be to use single quotes for delimiting strings, and only use double quotes when needed.)

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

>>> print('Hello') ???	>>> print('Hello") ???	>>> print('Let's Go') ???
>>> print("Hello") ???	>>> print("Let's Go!") ???	>>> print("Let's go!") ???

Control Characters

- Special characters that are not displayed on the screen. Rather, they *control* the display of output
- Do not have a corresponding keyboard character
- Therefore, they are represented by a combination of characters called an *escape sequence*.
- The backslash (\) serves as the escape character in Python.
- For example, the escape sequence '\n', represents the *newline control character*, used to begin a new screen line

```
print('Hello\nJennifer Smith')
```

which is displayed as follows,

```
Hello  
Jennifer Smith
```

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> print('Hello World')
???
>>> print('Hello World\n')
???
>>> print('Hello World\n\n')
???
>>> print('\nHello World')
???
>>> print('Hello\nWorld')
???
>>> print('Hello\\nWorld')
???
>>> print('Hello\\n\\nWorld')
???
>>> print(1, '\n', 2, '\n', 3)
???
>>> print('\n', 1, '\n', 2, '\n', 3)
???
```



www.HouseConstructionTips.com

Data Types

- Python's data types are built in the core of the language
- They are easy to use and straightforward.
- Data types supported by Python
 - Boolean values
 - None
 - Numbers
 - Strings
 - Tuples
 - Lists
 - Sets

Boolean values

- Primitive datatype having one of two values: True or False
- some common values that are considered to be True or False

Boolean values

print bool(True)	True
print bool(False)	False
print bool("text")	True
print bool("")	False
print bool(' ')	True
print bool(0)	False
print bool()	False
print bool(3)	True
print bool(None)	False

None

- Special data type - None
- Basically, the data type means non existent, not known or empty
- Can be used to check for emptiness

Numbers

Types of numbers supported by Python:

- Integers
- floating point numbers
- complex numbers
- Fractional numbers

Integers

- Integers have no fractional part in the number
- Integer type automatically provides extra precision for large numbers like this when needed.
- `>>> a = 10`
- `>>> b = a`

Binary, Octal and Hex Literals

- 0b1, 0b10000, 0b11111111 # Binary literals:
base 2, digits 0-1
- 0o1, 0o20, 0o377
- # Octal literals: base 8, digits 0-7
- 0x01, 0x10, 0xFF # Hex literals: base 16,
digits 0-9/A-F
- (1, 16, 255)

Conversion between different bases

- Provides built-in functions that allow you to convert integers to other bases' digit strings
- `oct(64)`, `hex(64)`, `bin(64)`
- # Numbers=>digit strings ('0o100', '0x40', '0b1000000')
- These literals can produce arbitrarily long integers

Numbers can be very long

Floating Point Numbers

- Number with fractional part
- `>>> 3.1415 * 2`
- `>>>6.283`

Floating Point Numbers

Table 5-1. Numeric literals and constructors

Literal	Interpretation
1234, -24, 0, 9999999999999999	Integers (unlimited size)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	Floating-point numbers
0o177, 0x9ff, 0b101010	Octal, hex, and binary literals in 3.X
0177, 0o177, 0x9ff, 0b101010	Octal, octal, hex, and binary literals in 2.X
3+4j, 3.0+4.0j, 3j	Complex number literals
set('spam'), {1, 2, 3, 4}	Sets: 2.X and 3.X construction forms
Decimal('1.0'), Fraction(1, 3)	Decimal and fraction extension types
bool(X), True, False	Boolean type and constants

Arithmetic overflow

- a condition that occurs when a calculated result is too large in magnitude (size) to be represented,

```
>>>1.5e200 * 2.0e210
```

```
>>> inf
```

This results in the special value `inf` (“infinity”) rather than the arithmetically correct result `3.0e410`, indicating that arithmetic overflow has occurred.

Arithmetic underflow

- a condition that occurs when a calculated result is too small in magnitude to be represented,
`>>>1.0e-300 / 1.0e100`
`>>>0.0`
- This results in 0.0 rather than the arithmetically correct result 1.0e-400, indicating that arithmetic underflow has occurred.

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> 1.2e200 * 2.4e100
```

```
???
```

```
>>> 1.2e200 * 2.4e200
```

```
???
```

```
>>> 1.2e200 / 2.4e100
```

```
???
```

```
>>> 1.2e-200 / 2.4e200
```

```
???
```

Arithmetic overflow occurs when a calculated result is too large in magnitude to be represented.

Arithmetic underflow occurs when a calculated result is too small in magnitude to be represented.

Repeated Print

```
>>>print('a'*15)
```

prints 'a' fifteen times

```
>>>print('\n'*15)
```

prints new line character fifteen times

Complex Numbers

- A complex number consists of an ordered pair of real floating point numbers denoted by $a + bj$, where a is the real part and b is the imaginary part of the complex number.
- **complex(x)** to convert x to a complex number with real part x and imaginary part zero
- **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y .
- x and y are numeric expressions

Complex Numbers

```
>>> 2 + 1j * 3
```

```
(2+3j)
```

```
>>> (2 + 1j) * 3
```

```
(6+3j)
```

- A = 1+2j; B=3+2j
- # Multiple statements can be given in same line using semicolon
- C = A+B; print(C)

Complex Numbers

prints real part of the number

- `print(A.real)`

prints imaginary part of the number

- `print(A.imag)`

Can do operations with part of complex number

- `print(A.imag+3)`

Input and output function

Input function : input

```
Basic_pay = input('Enter the Basic Pay: ')
```

Output function : print

```
print('Hello world!')
```

```
print('Net Salary', salary)
```

By Default...

- Input function reads all values as strings, to convert them to integers and float, use the function int() and float()

Type conversion...

```
line = input('How many credits do you have?')  
num_credits = int(line)  
line = input('What is your grade point average?')  
gpa = float(line)
```

Here, the entered number of credits, say '24', is converted to the equivalent integer value, 24, before being assigned to variable num_credits. For input of the gpa, the entered value, say '3.2', is converted to the equivalent floating-point value, 3.2. Note that the program lines above could be combined as follows,

```
num_credits = int(input('How many credits do you have? '))  
gpa = float(input('What is your grade point average? '))
```

Assignment Statement

Statement	Type
spam = 'Spam'	Basic form
spam, ham = 'yum', 'YUM'	Tuple assignment (positional)
[spam, ham] = ['yum', 'YUM']	List assignment (positional)
a, b, c, d = 'spam'	Sequence assignment, generalized
a, *b = 'spam'	Extended sequence unpacking (Python 3.X)
spam = ham = 'lunch'	Multiple-target assignment
a += 42	Augmented assignment (equivalent to a = a + 42)

Range

```
>>>a,b,c = range(1,4)
```

```
>>>a
```

```
1
```

```
>>>b
```

```
2
```

```
>>> S = "spam" >>> S += "SPAM" # Implied concatenation
```

```
>>> S
```

```
'spamSPAM'
```

Assignment is more powerful in Python

```
>>> nudge = 1
```

```
>>> wink = 2
```

```
>>> nudge, wink = wink, nudge
```

Tuples: swaps values

Like T = nudge; nudge = wink; wink = T

```
>>> nudge, wink
```

```
(2, 1)
```

Operators and Expressions in Python

Basic Arithmetic operators in Python

Command	Name	Example	Output
+	Addition	$4 + 5$	9
-	Subtraction	$8 - 5$	3
*	Multiplication	$4 * 5$	20
/	True Division	$19 / 3$	6.3333
//	Integer Division	$19//3$	6
%	Remainder <u>(modulo)</u>	$19 \% 3$	1
**	Exponent	$2 ** 4$	16

Order of Operations

Remember that thing called [order of operations](#) that they taught in maths? Well, it applies in Python, too. Here it is, if you need reminding:

1. parentheses ()
2. exponents **
3. multiplication *, division \, and remainder %
4. addition + and subtraction -

Order of Operations

Operator	Operation	Precedence
()	parentheses	0
**	exponentiation	1
*	multiplication	2
/	division	2
//	int division	2
%	remainder	2
+	addition	3
-	subtraction	3

- The computer scans the expression from left to right,
- first clearing parentheses,
- second, evaluating exponentiations from left to right in the order they are encountered
- third, evaluating *, /, //, % from left to right in the order they are encountered,
- fourth, evaluating +, - from left to right in the order they are encountered

$$2^{**}3+2^*(2+3)$$

- 18

Example 1 – Order of operations

```
>>> 1 + 2 * 3
```

7

```
>>> (1 + 2) * 3
```

9

- In the first example, the computer calculates $2 * 3$ first, then adds 1 to it. This is because multiplication has the higher priority (at 3) and addition is below that (at a lowly 4).
- In the second example, the computer calculates $1 + 2$ first, then multiplies it by 3. This is because parentheses have the higher priority (at 1), and addition comes in later than that.

Example 2 – Order of operations

Also remember that the math is calculated from left to right, *unless* you put in parentheses. The innermost parentheses are calculated first.

Watch these examples:

```
>>> 4 - 40 - 3
```

-39

```
>>> 4 - (40 - 3)
```

-33

- In the first example, $4 - 40$ is calculated, then $- 3$ is done.
- In the second example, $40 - 3$ is calculated, then it is subtracted from 4.

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> num = input('Enter number: ')      >>> num = input('Enter name: ')
Enter number: 5                      Enter name: John
???                                ???
>>> num = int(input('Enter number: ')) >>> num = int(input('Enter name: '))
Enter number: 5                      Enter name: John
???                                ???
```

Quotation in Python

- Python accepts single ('), double ("") and triple (" " or """") quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines. For example, all the following are legal –
- word = 'word'
- sentence = "This is a sentence."
- paragraph = """This is a paragraph. It is made up of multiple lines and sentences."""

Built-in format Function

- Because floating-point values may contain an arbitrary number of decimal places, the built-in **format** function can be used to produce a numeric string version of the value containing a specific number of decimal places.

```
>>> 12/5
```

```
2.4
```

```
>>> format(12/5, '.2f')  
'2.40'
```

```
>>> 5/7
```

```
0.7142857142857143
```

```
>>> format(5/7, '.2f')  
'0.71'
```

- In these examples, *format specifier* '.2f' rounds the result to two decimal places of accuracy in the string produced.

- For very large (or very small) values 'e' can be used as a format specifier,

```
>>> format(2 ** 100, '.6e')  
'1.267651e+30'
```

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> format(11/12, '.2f')
```

```
???
```

```
>>> format(11/12, '.2e')
```

```
???
```

```
>>> format(11/12, '.3f')
```

```
???
```

```
>>> format(11/12, '.3e')
```

```
???
```

Python is a Dynamic Type language

- Same variable can be associated with values of different type during program execution, as indicated below.
- It's also very dynamic as it rarely uses what it knows to limit variable usage

var = 12	integer
var = 12.45	float
var = 'Hello'	string

LET'S TRY IT

From the Python Shell, enter the following and observe the results.

```
>>> num = 10
```

```
>>> num
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> num = 20
```

```
>>> num
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = num
```

```
>>> k
```

```
???
```

```
>>> id(k)
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = 30
```

```
>>> k
```

```
???
```

```
>>> num
```

```
???
```

```
>>> id(k)
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> k = k + 1
```

```
>>> k
```

```
???
```

```
>>> id(num)
```

```
???
```

```
>>> id(k)
```

```
???
```

Bitwise Operations

- This includes operators that treat integers as strings of binary bits, and can come in handy if your Python code must deal with things like network packets, serial ports, or packed binary data
- `>>> x = 1` # 1 decimal is 0001 in bits
`>>> x << 2` # Shift left 2 bits: 0100
- 4

- `>>> x | 2` # Bitwise OR (either bit=1): 0011
- 3
- `>>> x & 1` # Bitwise AND (both bits=1): 0001
1
- In the first expression, a binary 1 (in base 2, 0001) is shifted left two slots to create a binary 4 (0100).
- The last two operations perform a binary OR to combine bits ($0001|0010 = 0011$) and a binary AND to select common bits ($0001&0001 = 0001$).

- To print in binary format use bin function:
- `>>> X = 0b0001 # Binary literals`
- `>>> X << 2 # Shift left 4`
- `>>> bin(X << 2) # Binary digits string
'0b100'`
- `>>> bin(X | 0b010) # Bitwise OR: either
'0b11'`
- `>>> bin(X & 0b1) # Bitwise AND: both
'0b0'`

Logical Operators

Assume $a = 10$ and $b = 20$

Operator	Description	Example
and	If both the operands are true then condition becomes true.	$(a \text{ and } b)$ is true.
Or	If any of the two operands are non-zero then condition becomes true.	$(a \text{ or } b)$ is true.
not	Used to reverse the logical state of its operand.	$\text{Not}(a \text{ and } b)$ is false.

Python is a Strongly Typed language

- interpreter keeps track of all variable types
- Check type compatibility while expressions are evaluated
- `>>> 2+3 # right`
- `>>>"two"+1 # Wrong!!`

Table 11-2. Augmented assignment statements

$X += Y$	$X \&= Y$	$X -= Y$	$X = Y$
$X *= Y$	$X ^= Y$	$X /= Y$	$X >>= Y$
$X \% = Y$	$X <<= Y$	$X **= Y$	$X // = Y$

Python Program for Bob Problem

```
n = float(input("Enter amount in hand"))
c = float(input("Enter price of one chocolate"))
m = int(input("Enter num of wrapper for free chocolate"))
#compute number of chocolates bought
p = n//c
#compute number of free chocolates
f = p//m
print("Number of chocolates",int(p+f))
```

Problem -1

ABC company Ltd. is interested to computerize the pay calculation of their employee in the form of Basic Pay, Dearness Allowance (DA) and House Rent Allowance (HRA). DA and HRA are calculated as certain % of Basic pay(For example, DA is 80% of Basic Pay, and HRA is 30% of Basic pay). They have the deduction in the salary as PF which is 12% of Basic pay. Propose a computerized solution for the above said problem.

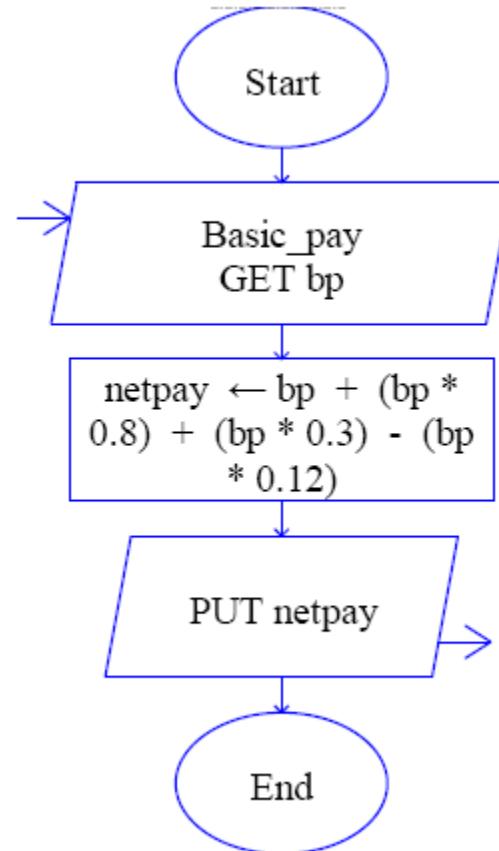
Input : Basic Pay

Process : Calculate Salary

(Basic Pay + (Basic Pay * 0.8) + (Basic Pay * 0.3) - (Basic Pay * 0.12)
-----allowances ----- --- deductions---

Output : Salary

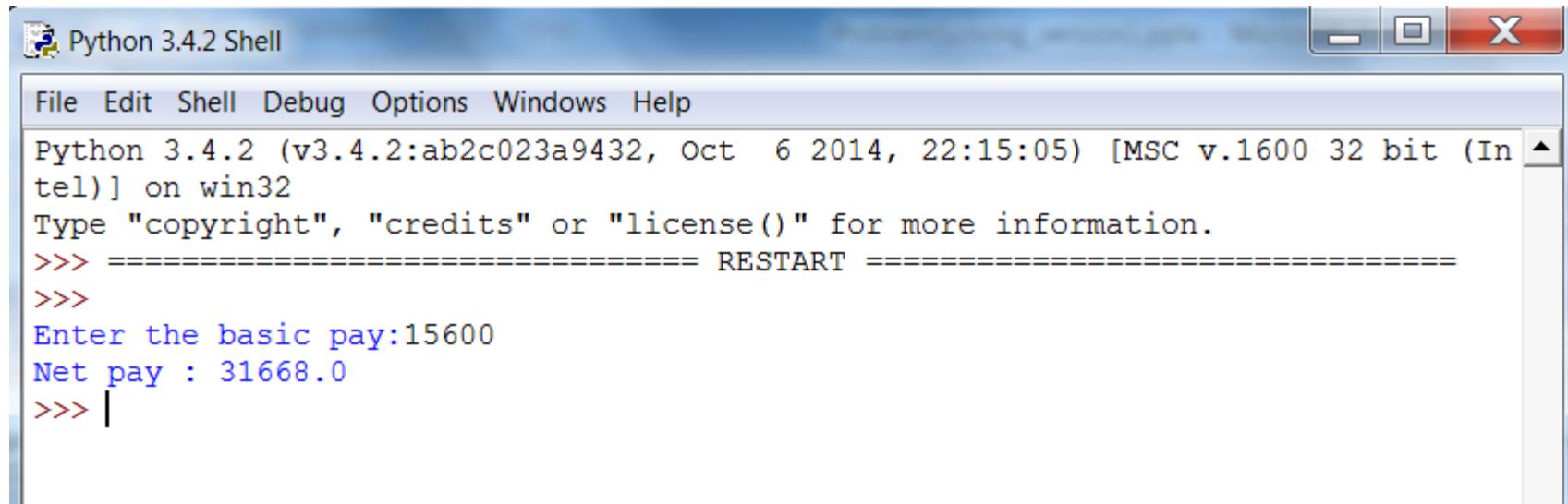
Flow chart



Python code

```
#Enter the basic pay  
bp=float (input('Enter the basic pay:'))  
  
# net pay calucluation  
netpay =bp + (bp*0.8) + (bp*0.3) - (bp*0.12)  
  
# display net salary  
print ('Net pay :',netpay)
```

Output



The screenshot shows a Windows application window titled "Python 3.4.2 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. The main window displays the Python 3.4.2 startup message, followed by a "RESTART" message, and then user input and output. The user has entered "Enter the basic pay:15600" and received the output "Net pay : 31668.0". The window has standard Windows-style borders and a title bar.

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter the basic pay:15600
Net pay : 31668.0
>>> |
```

Python features..... lambda operator

The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name.

```
>>> ftoc =lambda f: (f-32)*5.0/9
```

```
>>> ftoc(104)
```

Different patterns in Algorithm

Sequential

- Sequential structure executes the program in the order in which they appear in the program

Selectional (conditional-branching)

- Selection structure control the flow of statement execution based on some condition

Iterational (Loops)

- Iterational structures are used when part of the program is to be executed several times

Sequential Pattern

Example1: Find the average runs scored by a batsman in 4 matches

Algorithm:

Step 1: Start

Step 2: Input 4 scores say **runs1,runs2,runs3** and **runs4**

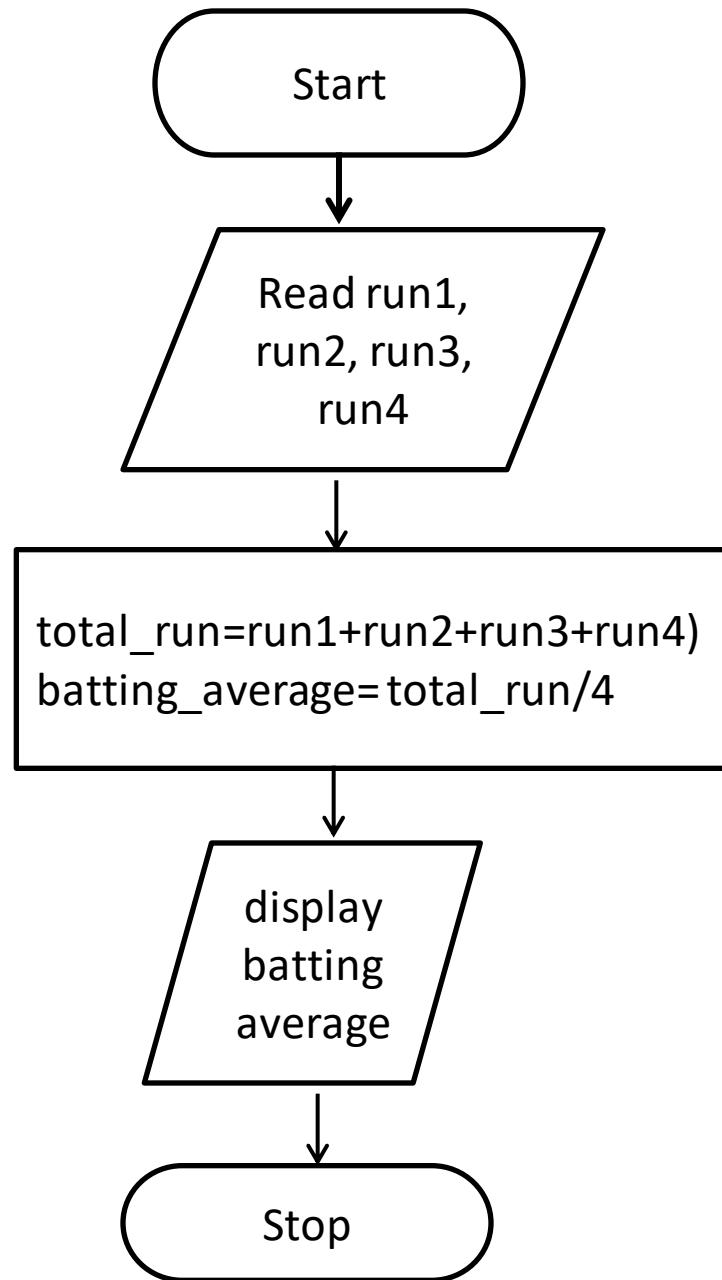
Step 3: Accumulate **runs1,runs2,run3, and runs4** and store it
in the variable called **total_runs**

Step 4: Divide **total_runs** by 4 and find the **average**

Step 5: Display the **average**

Step 6: Stop

Flowchart



Pseudo code:

Begin

read run1,run2,run3 and run4

compute total_run= run1+run2+run3+run4

compute batting_average= total_run/4

display batting_average

end

Batting Average

```
print("Enter four scores")
run1 = int(input())
run2 = int(input())
run3 = int(input())
run4| = int(input())
total_run=(run1+run2+run3+run4)
batting_average= total_run/4
print('batting_average is' ,batting_average)
```

Area of a circle

Step 1 : Start

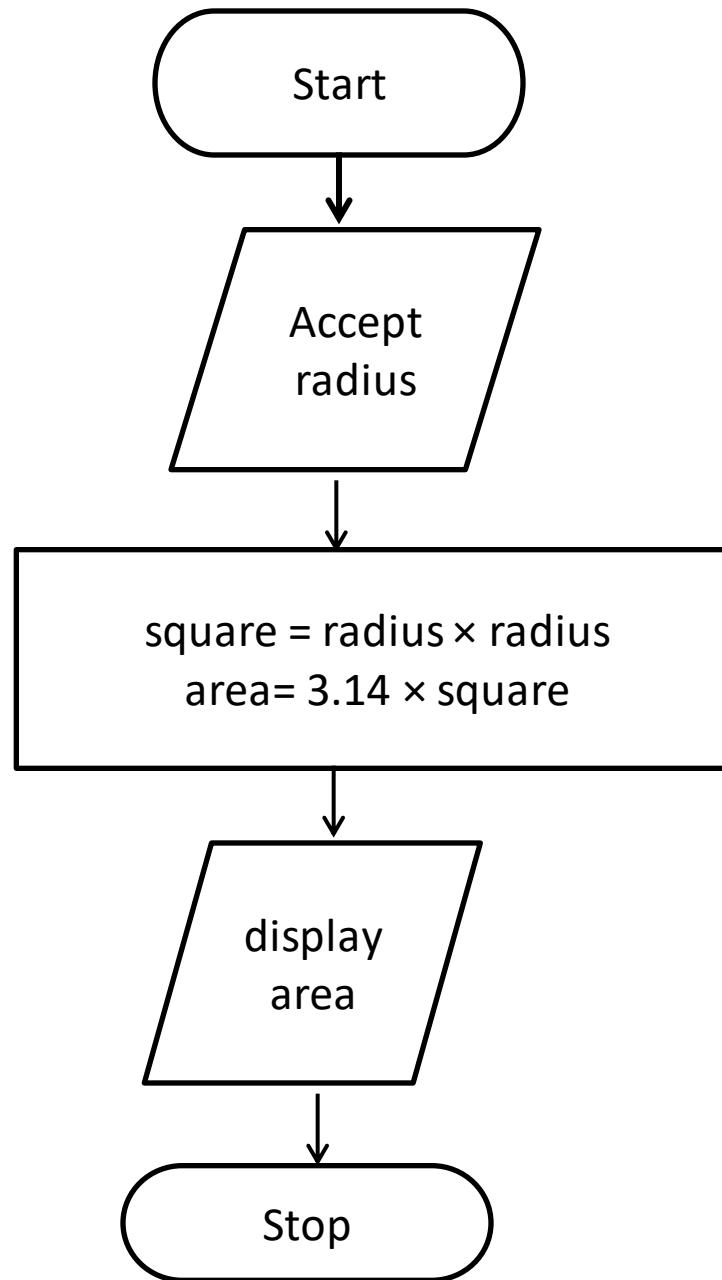
Step 2: Get the input for **RADIUS**

Step 3 : Find the square of **RADIUS** and store it in **SQUARE**

Step 4 : Multiply **SQUARE** with 3.14 and store the result in
AREA

Step 5: Stop

Flowchart



Pseudo code:

begin

accept radius

compute square = radius * radius

compute area = pi * square

display area

end

Area of a circle

```
import math  
print("Enter radius")  
radius=float(input())  
area = math.pi*radius*radius  
print("area of circle is ", area)
```

Exercise

An university is setting up a new lab at their premises. Design an algorithm and write Python code to determine the approximate cost to be spent for setting up the lab. Cost for setting the lab is sum of cost of computers, cost of furnitures and labour cost. Use the following formulae for solving the problem:

Cost of computer = cost of one computer * number of computers

Cost of furniture = Number of tables * cost of one table + number of chairs * cost of one chair

Labour cost = number of hours worked * wages per hour

Budget for Lab

Input	Processing	Output
cost of one computer, number of computers, number of tables, cost of one table, number of chairs, cost of one chair, number of hours worked, wages per hour	$\text{Budget} = \text{Cost of computers} + \text{cost of furniture} + \text{labour cost}$ $\text{Cost of computer} = \text{cost of one computer} * \text{number of computers}$ $\text{Cost of furniture} = \text{Number of tables} * \text{cost of one table} + \text{number of chairs} * \text{cost of one chair}$ $\text{Labour cost} = \text{number of hours worked} * \text{wages per hour}$	Budget for Lab

Python Program

```
print("Enter cost of one computer")
cost_Computer = float(input())
print("Enter num of computers")
num_Computer = int(input())
print("Enter cost of one table")
cost_Table = float(input())
print("Enter num of tables")
num_Tables = int(input())
print("Enter cost of one chair")
cost_Chair = float(input())
print("Enter num of chairs")
num_Chairs = int(input())
print("Enter wage for one hour")
wages_Per_Hr = float(input())
print("Enter num of hours")
num_Hrs = int(input())
```

Python Program

```
cost_of_Computers = cost_Computer* num_Computer
cost_of_Furnitures = num_Tables * cost_Table +\
                      cost_Chair*num_Chairs
wages = wages_Per_Hr * num_Hrs
budget = cost_of_Computers + cost_of_Furnitures + wages
#format for two decimal places
print ("Budget for Lab ",format(budget,'.2f'))
```

Browsing Problem

Given the number of hours and minutes browsed, write a program to calculate bill for Internet Browsing in a browsing center. The conditions are given below.

- (a) 1 Hour Rs.50
- (b) 1 minute Re. 1
- (c) Rs. 200 for five hours

Boundary condition: User can only browse for a maximum of 7 hours

Check boundary conditions

Browsing Program

Input	Processing	Output
Number of hours and minutes browsed	<p>Check number of hours browsed, if it is greater than 5 then add Rs 200 to amount for five hours and subtract 5 from hours</p> <p>Add Rs for each hour and Re 1 for each minute</p> <p>Basic process involved: Multiplication and addition</p>	Amount to be paid

Pseudocode

READ hours and minutes

SET amount = 0

IF hours >=5 then

 CALCULATE amount as amount + 200

 COMPUTE hours as hours – 5

END IF

COMPUTE amount as amount + hours * 50

COMPUTE amount as amount + minutes * 1

PRINT amount

Test Cases

Input

Hours = 6

Minutes = 21

Output

Amount = 271

Processing Involved

Amount = 200 for first five hours

50 for sixth hour

21 for each minute

Test Cases

Input

Hours = 8

Minutes = 21

Output

Invalid input

Processing Involved

Boundary conditions are violated

Already Know

- To read values from user
- Write arithmetic expressions in Python
- Print values in a formatted way

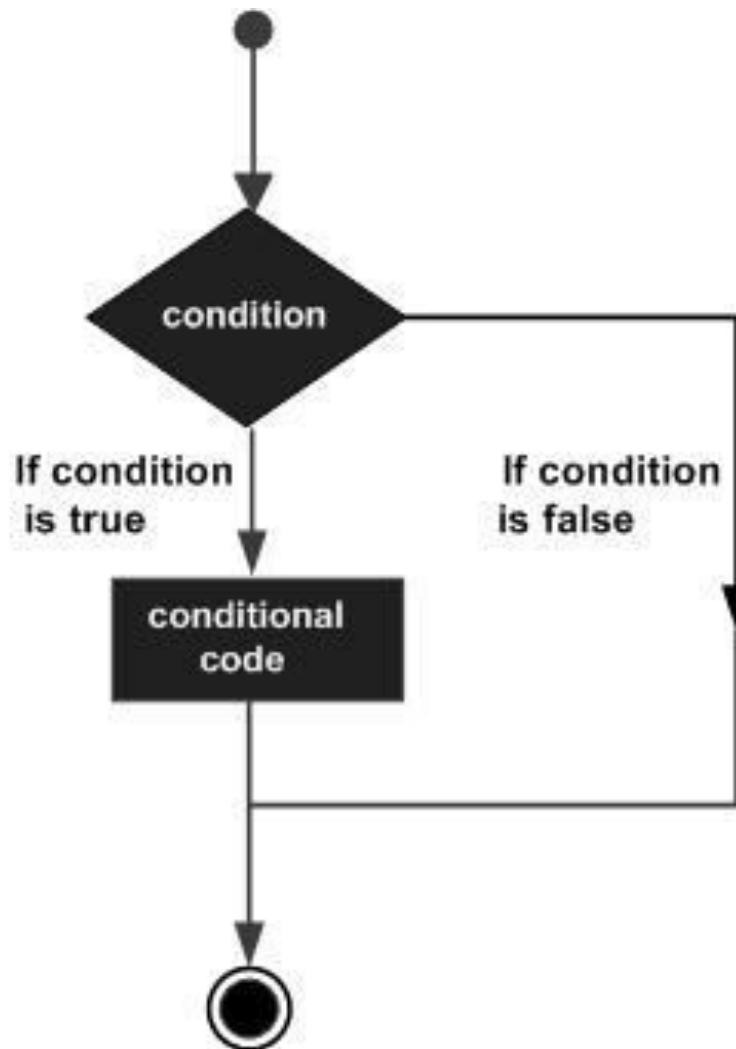
Yet to learn

- Check a condition

Selection pattern

- A **selection control statement** is a control statement providing selective execution of instructions.

Control flow of decision making



If Statement

- An **if statement** is a selection control statement based on the value of a given Boolean expression.

The if statement in Python is

If statement	Example use
If condition: statements else: statements	If grade >=70: print('pass') else: Print('fail')

If Statement

If statement	Example use
If condition: statements else: statements	If grade >=70: print('pass') else: Print('fail')

Indentation in Python

- One fairly unique aspect of Python is that the amount of indentation of each program line is significant.
- In Python indentation is used to associate and group statements

Valid indentation	Invalid indentation
(a) <pre>if condition: statement statement else: statement statement</pre>	(b) <pre>if condition: statement statement else: statement statement</pre>
(c) <pre>if condition: statement statement else: statement statement</pre>	(d) <pre>if condition: statement statement else: statement statement</pre>

Nested if Statements

- There are often times when selection among more than two sets of statements (suites) is needed.
- For such situations, if statements can be nested, resulting in **multi-way selection**.

Nested if statements	Example use
if condition: statements else: if condition: statements else: if condition: statements else: etc.	if grade >= 90: print('Grade of A') else: if grade >= 80: print('Grade of B') else: if grade >= 70: print('Grade of C') else: if grade >= 60: print('Grade of D') else: print('Grade of F')

Else if Ladder

```
if grade >= 90:  
    print('Grade of A')  
elif grade >= 80:  
    print('Grade of B')  
elif grade >= 70:  
    print('Grade of C')  
elif grade >= 60:  
    print('Grade of D')  
else:  
    print('Grade of F')
```

Multiple Conditions

- Multiple conditions can be checked in a ‘if’ statement using logical operators ‘and’ and ‘or’.
- Python code to print ‘excellent’ if mark1 and mark2 is greater than or equal to 90, print ‘good’ if mark1 or mark2 is greater than or equal to 90, print ‘need to improve’ if both mark1 and mark2 are lesser than 90

```
if mark1>=90 and mark2 >= 90:
```

```
    print('excellent')
```

```
if mark1>=90 or mark2 >= 90:
```

```
    print('good')
```

```
else:
```

```
    print('needs to improve')
```

Browsing Program

```
print("enter num of hours")
hour = int(input())
print("enter num of minutes")
min = int(input())
if(hour>7):
    print("Invalid input")
elif hour>=5:
    amount = 200
    hour = hour - 5
    amount = amount+hour*50+min
print(amount)
```

Eligibility for Scholarship

Government of India has decided to give scholarship for students who are first graduates in family and have scored average > 98 in math, physics and chemistry. Design an algorithm and write a Python program to check if a student is eligible for scholarship

Boundary Conditions: All marks should be > 0

Scholarship Program

Input	Processing	Output
Read first graduate, physcis, chemistry and maths marks	Compute total = phy mark + che mark + math mark Average = total/3 Check if the student is first graduate and average ≥ 98	Print either candidate qualified for Scholarship or candidate not qualified for Scholarship

Algorithm

Step 1 : Start

Step 2: Read first graduate, physcis,chemistry and maths
marks

Step 3: If anyone of the mark is less than 0 then print
'invalid input' and terminate execution

Step 3 : Accumulate all the marks and store it in **Total**

Step 4 : Divide **Total** by 3 and store it in **Average**

Step 5 : If student is first graduate **Average** score is
greater than or equal to 98 then print candidate
qualified for Scholarship

Else

Print candidate not qualified for scholarship

Stop 6: Stop

Test Cases

Input

First graduate = 1 Phy mark = 98, Che mark = 99,
math mark = 98

Output

candidate qualified for Scholarship

Processing Involved

Total = 295

Average = 98.33

Student is first graduate and average > 98

Test Cases

Input

First graduate = 0 Phy mark = 98, Che mark = 99,
math mark = 98

Output

candidate is not qualified for Scholarship

Processing Involved

Total = 295

Average = 98.33

Student is not first graduate but average > 98

Test Cases

Input

First graduate = 1 Phy mark = 98, Che mark = 99,
math mark = 90

Output

candidate is not qualified for Scholarship

Processing Involved

Total = 287

Average = 95.67

Student is first graduate but average < 98

```
print('Is first graduate(1 for yes and 0 for no')
first = int(input())
print('Enter Physics Marks')
phy_mark = float(input())
print('Enter Chemistry Marks')
che_mark=float(input())
print('Enter Math Marks')
mat_mark=float(input())
total_mark= phy_mark+che_mark+mat_mark
```

```
if(phy_mark <0 or che_mark <0 or mat_mark<0):
    print('Invalid input')
else:
    average = total_mark/3
    if first==1 and average >= 98 :
        print('candidate qualified for Scholarship')
    else:
        print('candidate not qualified for Scholarship')
```

Algorithm for Largest of Three numbers

Step1: Start

Step2: Read value of **a**, **b** and **c**

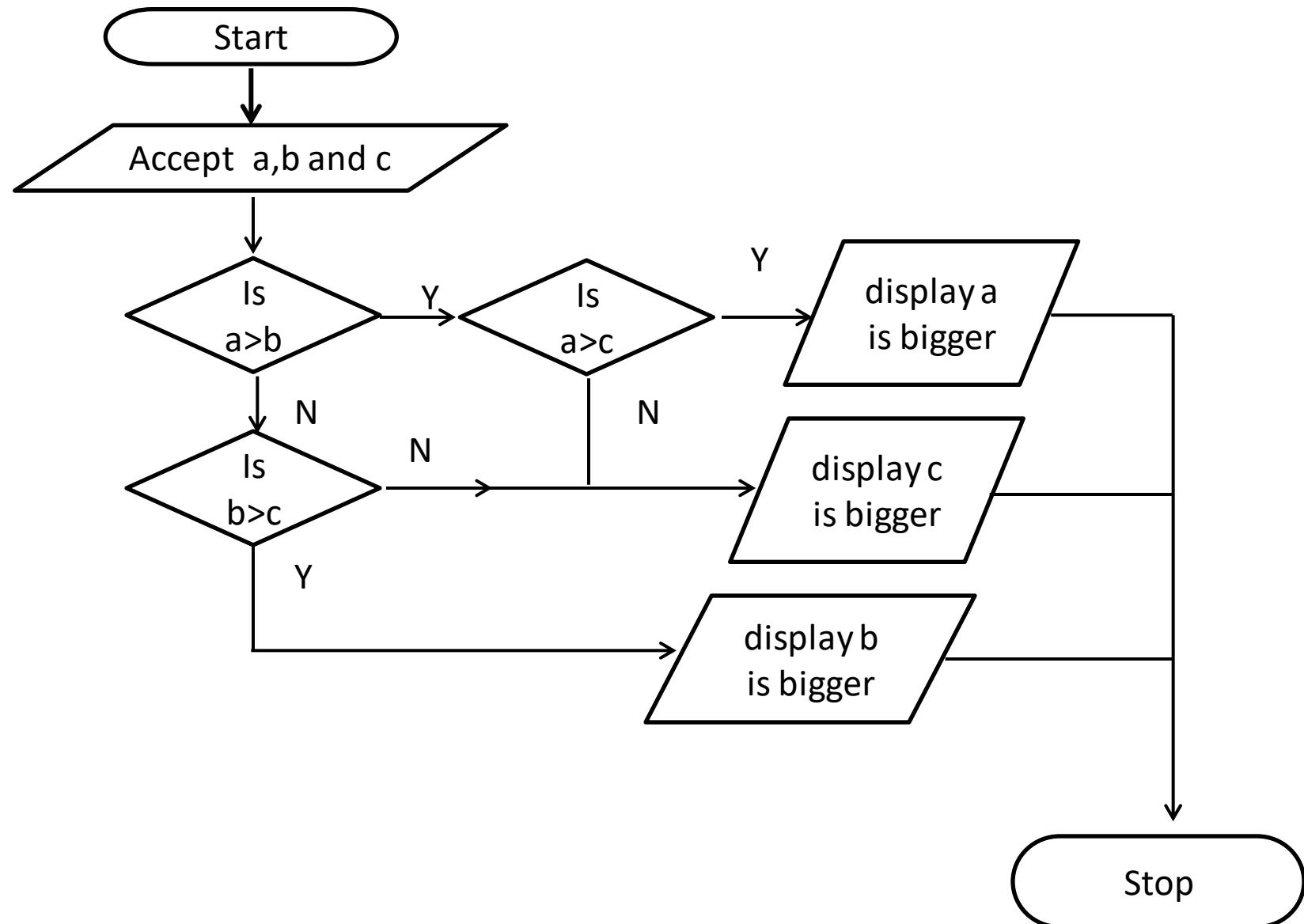
Step3: If **a** is greater than **b** then

 compare **a** with **c** and if **a** is bigger then say
 a is biggest else say **c** is biggest

 else Compare **b** with **c** , if **b** is greater than
 c say **b** is biggest else **c** is biggest

Step 5: Stop

Flowchart



Test Cases

Input

$a = 12, b = 13, c = 14$

Output

c is greatest

Processing Involved

B is greater than a but c is greater than b

Test Cases

Input

a = 13, b = 12, c = 14

Output

c is greatest

Processing Involved

a is greater than b but c is greater than a

Test Cases

Input

a = 13, b = 2, c = 4

Output

a is greatest

Processing Involved

a is greater than b and a is greater than c

Test Cases

Input

$a = 3, b = 12, c = 4$

Output

b is greatest

Processing Involved

b is greater than a and b is greater than c

```
a = int(input())
b = int(input())
c = int(input())
if a>b:
    if a>c:
        print ('a is greatest')
    else:
        print ('c is greatest')
else:
    if b>c:
        print ('b is greatest')
    else:
        print ('c is greatest')|
```

The if/else Ternary Expression

Consider the following statement, which sets A to either Y or Z, based on the truth value of X:

if X:

A = Y

else:

A = Z

new expression format that allows us to say the same thing in one expression:

- $A = Y \text{ if } X \text{ else } Z$

```
>>> A = 't' if 'spam' else 'f'
```

```
>>> A
```

```
't'
```

```
>>> A = 't' if " else 'f'
```

```
>>> A
```

```
'f'
```

Exercise Problem

1. Write a python code to check whether a given number of odd or even?
2. Write a python code to check whether a given year is leap year or not?
3. Write a python code in finding the roots of a quadratic equation?
4. Write a python program to segregate student based on their CGPA. The details are as follows:

<=9 CGPA <=10	- outstanding
<=8 CGPA <9	- excellent
<=7 CGPA <8	- good
<=6 CGPA <7	- average
<=5 CGPA <6	- better
CGPA<5	- poor

Class Average

- Given marks secured in CSE1001 by the students in a class, design an algorithm and write a Python code to determine the class average. Print only two decimal digits in average

Class Average

Input	Processing	Output
Number of students in class, mark scored by each student	Determine total of marks secured by students Find average of marks	Class average of marks

Average marks scored by ‘N’ number of Students

Step 1: Start

Step 2 : Read Number Of Students

Step 3 : Initialize counter as 0

Step 4 : Input mark

Step 5 : Add the mark with total

Step 6 : Increment the counter by 1

Step 7: repeat Step 4 to Step 6 until counter less than number of students

Step 7: Divide the total by number of students and store it in average

Step 8: Display the average

Step 9: Stop

Test Cases

Input

5

90 85 70 50 60

Output

71.00

Processing Involved

Already Know

- To read values from user
- To check if a condition is satisfied
- Print characters

Yet to learn

- Repeatedly execute a set of statements

Need of iterative control

Repeated execution of set of statements

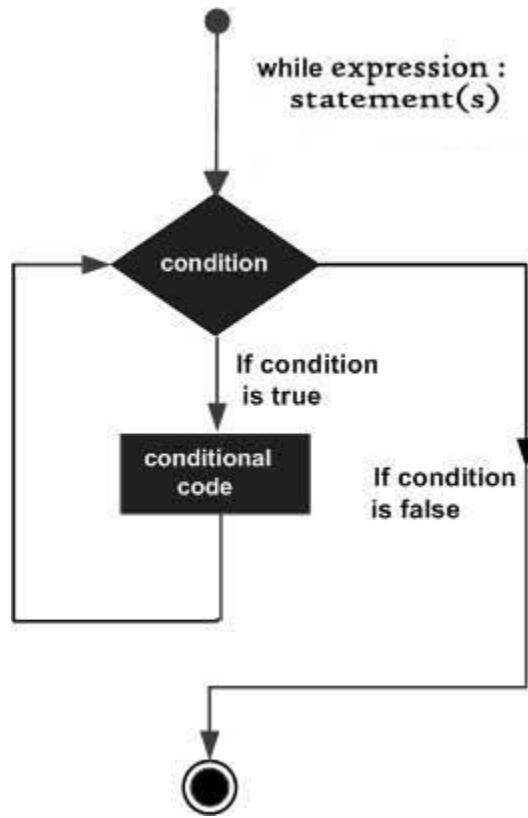
- An **iterative control statement** is a control statement providing repeated execution of a set of instructions
- Because of their repeated execution, iterative control structures are commonly referred to as “loops.”

While statement

- Repeatedly executes a set of statements based on a provided Boolean expression (condition).
- All iterative control needed in a program can be achieved by use of the while statement.

Syntax of While in Python

```
while test:                      # Loop test
    statements                   # Loop body
else:                           # Optional else
    statements
# Run if didn't exit loop with break
```



Example use

Sum of first 'n' numbers

$\text{sum} = 0$

$\text{current} = 1$

$n=3$

$\text{while current} \leq n:$

$\text{sum} = \text{sum} + \text{current}$

$\text{current} = \text{current} + 1$

Iteration	sum	current	$\text{current} \leq 3$	$\text{sum} = \text{sum} + \text{current}$	$\text{current} = \text{current} + 1$
1	0	1	True	$\text{sum} = 0 + 1 \ (1)$	$\text{current} = 1 + 1 \ (2)$
2	1	2	True	$\text{sum} = 1 + 2 \ (3)$	$\text{current} = 2 + 1 \ (3)$
3	3	3	True	$\text{sum} = 3 + 3 \ (6)$	$\text{current} = 3 + 1 \ (4)$
4	6	4	False	loop termination	

Print values from 0 to 9 in a line

```
a=0; b=10
```

```
while a < b:      # One way to code counter loops
    print(a, end=' ')
    a += 1          # Or, a = a + 1
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Include `end=' '` in print statement to suppress default move to new line

Break, continue, pass, and the Loop else

- break Jumps out of the closest enclosing loop
- continue Jumps to the top of the closest enclosing loop
- pass Does nothing at all: it's an empty statement placeholder
- Loop else block Runs
- if and only if the loop is exited normally (i.e., without hitting a break)

Break statement

- while True:

```
name = input('Enter name:')

if name == 'stop': break

age = input('Enter age: ')
print('Hello', name, '=>', int(age) ** 2)
```

Output:

Enter name:bob

Enter age: 40

Hello bob => 1600

Pass statement

- Infinite loop
 - while True: pass
- # Type Ctrl-C to stop me!

Print all even numbers less than 10 and greater than or equal to 0

```
x = 10
while x:
    x = x-1                      # Or, x -= 1
    if x % 2 != 0: continue        # Odd? -- skip print
    print(x, end=' ')
```

Class Average

```
File Edit Format Run Options Window Help
count =0
total = 0
n=int(input('enter how many mark you want to read: '))
while count < n:
    mark=int(input('enter mark :'))
    if mark<0:
        print ("mark should be greater than 0, terminates.
        break
    total = total + mark
    count = count + 1
else:
    average=total/n
    print("average mark is" , format(average,"0.2f"))
```

Pattern Generation

- Your teacher has given you the task to draw the structure of a staircase. Being an expert programmer, you decided to make a program for the same. You are given the height of the staircase. Given the height of the staircase, write a program to print a staircase as shown in the example. For example,
Staircase of height 6:

```
#  
##  
###  
####  
#####  
######
```

Boundary Conditions: height >0

Pattern Generation

Input	Processing	Output
Staircase height	Create steps one by one To create a step print character equal to length of step	Pattern

Pseudocode

READ staircase_height

if staircase_height > 0

x = 1

Repeat

y = 1

Repeat

 print #

 y = y + 1

Until y <= x

x = x + 1

Until x <= staircase_height

End if

Else

Print “Invalid input”

Test Cases

Input

3

Output

###

Processing Involved

Print step by step

Test Cases

Input

-1

Output

Invalid input

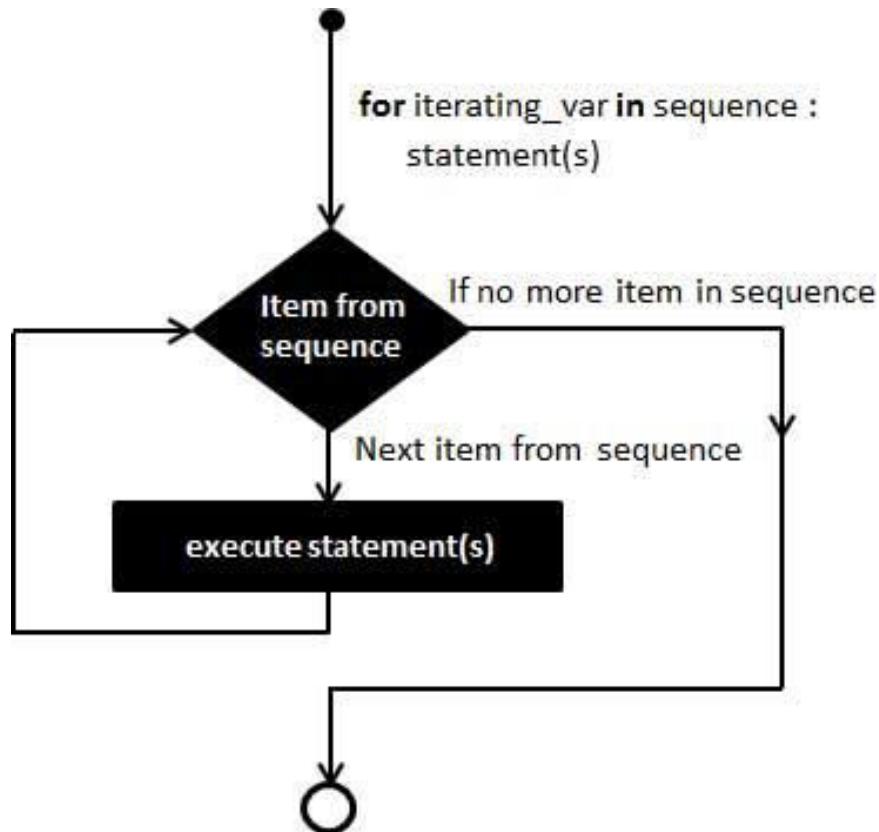
Processing Involved

Boundary condition check fails

For iteration

- In while loop, we cannot predict how many times the loop will repeat
- The number of iterations depends on the input or until the conditional expression remains true

Control flow of for statement



Syntax of for Statement

for target in object:

Assign object items to target

statements

if test: break

Exit loop now, skip else

if test: continue

Go to top of loop now

else: statements

If we didn't hit a 'break'

For and Strings

for iterating_var in sequence or range:
 statement(s)

Example:

```
for letter in 'Python':  
    print ('Current Letter :', letter)
```

For and Strings

When the above code is executed:

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : h

Current Letter : o

Current Letter : n

For and Range

```
for n in range(1, 6):  
    print(n)
```

When the above code is executed:

1
2
3
4
5

range function call

Syntax - `range(begin,end,step)`

where

Begin - first value in the range; if omitted, then default value is 0

end - one past the last value in the range; end value may not be omitted

Step - amount to increment or decrement; if this parameter is omitted, it defaults to 1 and counts up by ones

begin, end, and step must all be **integer values**;
floating-point values and other types are not allowed

Example for Range

range(10) → 0,1,2,3,4,5,6,7,8,9

range(1, 10) → 1,2,3,4,5,6,7,8,9

range(1, 10, 2) → 1,3,5,7,9

range(10, 0, -1) → 10,9,8,7,6,5,4,3,2,1

range(10, 0, -2) → 10,8,6,4,2

range(2, 11, 2) → 2,4,6,8,10

range(-5, 5) → -5,-4,-3,-2,-1,0,1,2,3,4

range(1, 2) → 1

range(1, 1) → (empty)

range(1, -1) → (empty)

range(1, -1, -1) → 1,0

range(0) → (empty)

Print Even Numbers Using Range

```
>>> for i in range(2,10,2):  
    print(i)
```

Output:

```
2  
4  
6  
8
```

```
print("Enter number of steps")
n = int(input())
for i in range(0, n):
    for j in range(0, i+1):
        print('#', end = ' ')
    print()
```

Exercise Problem

1. Write a program that read a group ‘g’ of five numbers and another number ‘n’ and print a number in ‘g’ if it is a factor for a given number n?
2. Write a program to find the factorial of a number n?
3. Write a menu driven program which get user choice to perform add/sub/mul/div with the obtained two input?
4. Write a program to display few odd multiples of a odd number n ?

Exercise Problem

5. The Head Librarian at a library wants you to make a program that calculates the fine for returning the book after the return date. You are given the actual and the expected return dates. Calculate the fine as follows:
- If the book is returned on or before the expected return date, no fine will be charged, in other words fine is 0.
 - If the book is returned in the same month as the expected return date, $\text{Fine} = 15 \text{ Rupees} \times \text{Number of late days}$
 - If the book is not returned in the same month but in the same year as the expected return date, $\text{Fine} = 500 \text{ Rupees} \times \text{Number of late months}$
 - If the book is not returned in the same year, the fine is fixed at 10000 Rupees.

Strings in Python

Check Validity of a PAN

In any of the country's official documents, the PAN number is listed as follows

<alphabet>< alphabet> < alphabet > < alphabet >
< alphabet > <digit><digit><digit><digit>< alphabet >

Your task is to figure out if the PAN number is valid or not. A valid PAN number will have all its letters in uppercase and digits in the same order as listed above.

PAC For Chocolate Problem

Input	Processing	Output
PAN number	Take each character and check if alphabets and digits are appropriately placed	Print Valid or Invalid

Pseudocode

READ PAN

If length of PAN is not ten then print “Invalid” and exit

FOR x=0 to5

if PAN[x] is not a character THEN

PRINT ‘invalid’

BREAK;

END IF

END FOR

FOR x=5 to 9

if PAN[x] is not a digit THEN

PRINT ‘invalid’

BREAK;

END IF

END FOR

IF PAN[9] is not a character THEN

PRINT ‘invalid’

END IF

PRINT ‘valid’

Test Case 1

abcde1234r

Valid

Test Case 2

abcde12345

Invalid

Test Case 3

abcd01234r

Invalid

Strings

- Immutable sequence of characters
- A string literal uses quotes
 - 'Hello' or "Hello" or ""Hello""
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string

String Operations

A few more string operations

Operation	Interpretation
<code>S = ''</code>	Empty string
<code>S = "spam's"</code>	Double quotes, same as single
<code>S = 's\np\ta\xoom'</code>	Escape sequences
<code>S = """...multiline..."""</code>	Triple-quoted block strings
<code>S = r'\temp\spam'</code>	Raw strings (no escapes)
<code>B = b'sp\xc4m'</code>	Byte strings in 2.6, 2.7, and 3.X (Chapter 4 , Chapter 37)
<code>U = u'sp\u00c4m'</code>	Unicode strings in 2.X and 3.3+ (Chapter 4 , Chapter 37)
<code>S1 + S2</code>	Concatenate, repeat
<code>S * 3</code>	
<code>S[i]</code>	Index, slice, length
<code>S[i:j]</code>	

String Operations

`len(s)`

<code>"a %s parrot" % kind</code>	String formatting expression
<code>"a {0} parrot".format(kind)</code>	String formatting method in 2.6, 2.7, and 3.X
<code>s.find('pa')</code>	String methods (see ahead for all 43): search,
<code>s.rstrip()</code>	remove whitespace,
<code>s.replace('pa', 'xx')</code>	replacement,
<code>s.split(',')</code>	split on delimiter,

String Operations

Operation	Interpretation
<code>S.isdigit()</code>	content test,
<code>S.lower()</code>	case conversion,
<code>S.endswith('spam')</code>	end test,
<code>'spam'.join(strlist)</code>	delimiter join,

Example Strings

- Single quotes: 'spa"m'
- Double quotes: "spa'm"
- Triple quotes: "... spam ...", """... spam ..."""
- Escape sequences: "s\tp\na\0m"
- Raw strings: r"C:\new\test.spm"

Escape Sequences

- Represent Special Characters

```
>>> s = 'a\nb\tc'
```

```
>>> print(s)
```

a

b c

```
>>> len(s)
```

5

Escape Sequences

TABLE 7-2. OBTAINING DISCRETE CHARACTER

Escape	Meaning
\newline	Ignored (continuation line)
\\\	Backslash (stores one \)
\'	Single quote (stores ')
\"	Double quote (stores ")
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\xhh	Character with hex value <i>hh</i> (exactly 2 digits)
\ooo	Character with octal value <i>ooo</i> (up to 3 digits)
\0	Null: binary 0 character (doesn't end string)

Length of a String

```
>>> s = 'bobby'
```

```
>>> len(s)
```

```
5
```

```
>>> print(s)
```

```
a b c
```

Backslash in Strings

- if Python does not recognize the character after a \ as being a valid escape code, it simply keeps the backslash in the resulting string:

```
>>> x = "C:\py\code"
```

- # Keeps \ literally (and displays it as \\)

```
>>> x
```

```
'C:\\py\\code'
```

```
>>> len(x)
```

Backslash in Strings

```
>>> x = "C:\py\code"
```

```
>>> x
```

```
'C:\\py\\\\code'
```

```
>>> len(x)
```

```
10
```

Check this

```
s = "C:\new\text.dat"
```

```
>>>s
```

```
print(s)
```

```
s1 = r"C:\new\text.dat"
```

```
>>>s1
```

```
print(s1)
```

```
s2 = "C:\\new\\\\text.dat"
```

```
print(s2)
```

```
>>>s2
```

Opening a File

- `myfile = open('C:\\new\\text.dat', 'w')` - Error
- `myfile = open(r'C:\\new\\text.dat', 'w')`
- Alternatively two backslashes may be used
- `myfile = open('C:\\\\new\\\\text.dat', 'w')`
- `>>> path = r'C:\\new\\text.dat'`
- `>>> print(path)` # User-friendly format

`C:\\new\\text.dat`

□ `>>> len(path)`

Basic Operations

```
>>> 'Ni!' * 4
```

```
'Ni!Ni!Ni!Ni!'
```

```
>>> print('-' * 80) # 80 dashes, the easy way
```

```
>>> myjob = "hacker"
```

```
>>> for c in myjob:
```

```
    print(c, end=' ')
```

```
h a c k e r
```

Basic Operations

```
>>> for c in 'myjob':
```

```
    print(c, end=' ')
```

```
h a c k e r
```

Using 'in' Operator in Strings

❑ >>> "k" in myjob # Found

❑ True

❑ >>> "z" in myjob # Not found

❑ False

❑ >>> 'spam' in 'abcspamdef'

❑ # Substring search, no position returned

❑ True

Counting

- Count the number of ‘a’

Counting

- Count the number of 'a'

- Example

- *word = 'Btechallbranches'*

- *count = 0*

- *for letter in word :*

- *if letter == 'a' :*

- *count = count + 1*

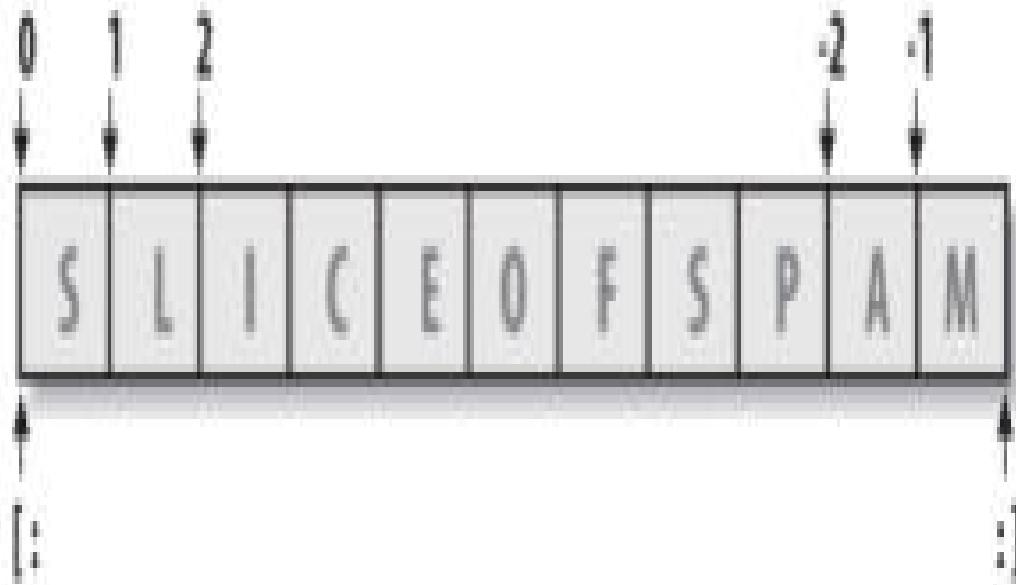
- *print(count)*

Indexing and Slicing

- >>> S = 'spam'
- Last character in the string has index -1 and the one before it has index -2 and so on

`[start,end]`

Indexes refer to places the knife "cuts."



Defaults are beginning of sequence and end of sequence.

Indexing

- Take one letter from a word at a time
- Use square bracket and give the index of the letter to be extracted
- Indexing can be done either from front or from end
- >>> S[0], S[-2]
- ('s', 'a')

Slicing

- Take a part of a word
- Square bracket with two arguments with a colon
- First value indicates the starting position of the slice and second value indicates the stop position of the slice
- Character at the stop position is not included in the slice
- `>>> S[1:3]`
- `'pa'`

Slicing

- If the second number is beyond the end of the string, it stops at the end
- If we leave off the first or last number of the slice, it is assumed to be beginning or end of the string respectively
 - `s = 'spam'`
 - `>>>s[:3]`
 - ‘spa’
 - `>>>s[1:]`
 - ‘pam’

Properties of Slicing

- $S[1:3]$ - fetches items at offsets 1 up to but not including 3.
- $S[1:]$ - fetches items at offset 1 through the end
- $S[:3]$ - fetches items at offset 0 up to but not including 3
- $S[:-1]$ - fetches items at offset 0 up to but not including last item
- $S[:]$ - fetches items at offsets 0 through the end—making a top-level copy of S

Extended slicing

- $X[I:J:K]$ - means “extract all the items in X , from offset I through $J-1$, by K .”
- Third limit, K , defaults to +1
- If you specify an explicit value it is used to skip items
- Extraction is reversed when negative value is given for $K-1$
- Each time $K-1$ items are skipped

```
ation.  
=> str1 = '0123456789'  
=> str1[:]  
'0123456789'|  
=> str1[::2]  
'02468'  
=> str1[::3]  
'0369'  
=> str1[1::2]  
'13579'  
=> str1[1:6:2]  
'135'  
=>
```



Extended slicing Examples

- >>> S = 'abcdefghijklmnoP'
- >>> S[1:10:2] # Skipping items
□ 'bdfhj'
- >>> S[::-2]
□ 'acegikmo'
- >>> S = 'hello'
- >>> S[::-1] # Reversing items
□ 'olleh'

Extended slicing Examples

- >>> S = 'hello'

- >>> S[::-1] # Reversing items

- 'olleh'

String Conversion Tools

- ❑ >>> "42" + 1

- ❑ TypeError: Can't convert 'int' object to str implicitly

- ❑ >>> int("42"), str(42) # Convert from/to string

- ❑ (42, '42')

- ❑ int("42") + 1

- ❑ 43

- ❑ >>> "42" + str(1)

- ❑ '421'

Character code Conversions

- `ord()` - Convert a single character to its underlying integer code (e.g., its ASCII byte value)—this value is used to represent the corresponding character in memory.

- `>>> ord('s')`

- 115

Character code Conversions

- `chr()` – Does inverse of `ord`

- `>>> chr(115)`

- `'s'`

Character code Conversions - Example

- >>> S = '5'

- >>> S = chr(ord(S) + 1)

- >>> S

- '6'

- >>> S = chr(ord(S) + 1)

- >>> S

- '7'

Character code Conversions - Example

- >>> ord('5') - ord('0')

- 5

- >>> int('1101', 2) # Convert binary to integer

- 13

- >>> bin(13) # Convert integer to binary

- '0b1101'

Concatenation

- >>> S1 = 'Welcome'

- >>> S2 = 'Python'

- >>> S3 = S1 + S2

- >>> S3

- 'WelcomePython'

Changing Strings

- String - “immutable sequence”
- Immutable - you cannot change a string in place
 - >>> S = 'spam'
 - >>> S[0] = 'x' # Raises an error!
- TypeError: 'str' object does not support item assignment

Changing Strings

- >>> S = S + 'SPAM!'
- # To change a string, make a new one
- >>> S
- 'spamSPAM!'
- >>> S = S[:4] + 'Burger' + S[-1]
- >>> S
- 'spamBurger!'

Replace

```
□>>> S = 'splot'  
□>>> S = S.replace('pl', 'pamal')  
□>>> S  
□'spamalot'
```

Formatting Strings

- >>> 'That is %d %s bird!' % (1, 'dead')
- That is 1 dead bird!
- >>> 'That is {0} {1} bird!'.format(1, 'dead')
- 'That is 1 dead bird!'

String Library

- Python has a number of string functions which are in the string library
- These functions do not modify the original string, instead they return a new string that has been altered

String Library

- >>> greet = 'Hello Arun'
- >>> zap = greet.lower()
- >>> print (zap)
- hello arun
- >>> print ('Hi There'.lower())
- hi there

Searching a String

- `find()` - function to search for a string within another
- `find()` - finds the first occurrence of the substring
- If the substring is not found, `find()` returns -1
- `>>> name = 'pradeepkumar'`
- `>>> pos = name.find('de')`
- `>>> print (pos)`
- 3

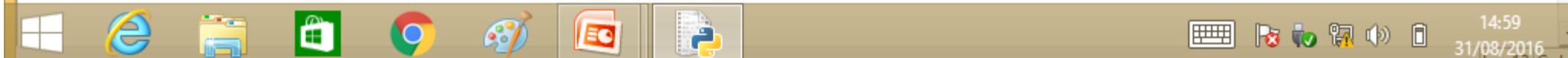
Searching a String

- >>> aa = "fruit".find('z')
- >>> print (aa)
- -1
- >>> name = 'pradeepkumar'
- >>> pos = name.find('de',5,8)
- >>>pos
- -1

Other Common String Methods in Action

```
□>>> line = "The knights who say Ni!"  
“  
□>>> line.rstrip()  
□'The knights who say Ni!'  
□>>> line.upper()  
□'THE KNIGHTS WHO SAY NI!'
```

```
>>> s.find('e',3,4)
-1
>>> s.find('t',3,6)
-1
>>> s='Hello\t'
>>> s
'Hello\t'
>>> s.rstrip()
'Hello'
>>> |
```



Other Common String Methods in Action

- >>> line.isalpha()
- False
- >>> line.endswith('Ni!')
- True
- >>> line.startswith('The')
- True

Other Common String Methods in Action

length and slicing operations can be used to mimic endswith:

>>> line = 'The knights who say Ni!\n'

>>> line.find('Ni') != -1

True

>>> 'Ni' in line

True

Other Common String Methods in Action

- >>> sub = 'Ni!\\n'
- >>> line.endswith(sub) # End test via
method call or slice
- True

```
pan = input("enter pan")
invalid = False
if len(pan) !=10:
    invalid = True
else:
    for i in range(0,5):
        if not pan[i].isalpha():
            invalid = True
            break
    for i in range(5,9):
        if not pan[i].isdigit():
            invalid = True
            break
    if not pan[9].isalpha():
        invalid = True
if invalid == True:
    print ("Invalid")
```

To check if all letters in a String are in Uppercase

`isupper()` function is used to check if all letters in a string are in upper case

Examples

```
>>> 'a'.isupper()
```

```
False
```

```
>>> 'A'.isupper()
```

```
True
```

```
>>> 'AB'.isupper()
```

```
True
```

```
>>> 'ABC'.isupper()
```

```
False
```

```
>>>
```

To check if all letters in a String are in Lowercase

`islower()` function is used to check if all letters in a string are in lower case

Examples

```
>>> 'a'.islower()
```

```
True
```

```
>>> 'aB'.islower()
```

```
False
```

To check if a sentence is in Title case

`istitle()` function is used to check if all letters in a string are in title case

Examples

```
>>> 'Apple Is A Tree'.istitle()
```

```
True
```

```
>>> 'Apple Is A tree'.istitle()
```

```
False
```

Looping Through Strings

Using a while statement and an iteration variable, and the len function, we can construct a loop to look at each of the letters in a string individually.

```
fruit = 'banana'  
index = 0  
while index < len(fruit) :  
    letter = fruit[index]  
    print (index, letter)  
    index = index + 1
```

For loop

A definite loop using a for statement
is much more elegant

The iteration variable is completely
taken care of by the for loop

```
fruit = 'banana'  
for letter in fruit :  
    print (letter)
```

Looping and Counting

This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character.

```
word = 'banana'  
count = 0  
for letter in word :  
    if letter == 'a' :  
        count = count + 1  
print (count)
```

String Comparison

```
word=input()
if word == 'banana':
    print ( 'All right, bananas')
```

Strings

String Data

Type

- A string is a sequence of characters
- A string literal uses quotes 'Hello' or "Hello"
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print bob
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: cannot
concatenate 'str' and 'int' objects
>>> x = int(str3) + 1
>>> print (x)
124
>>>
```

Reading and Converting

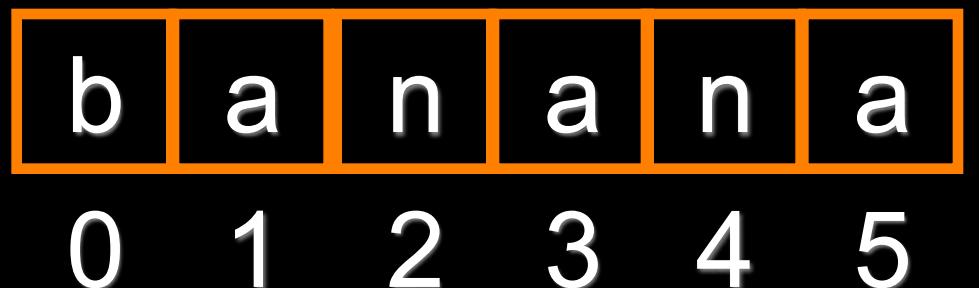
- Data entered from keyboard is by default String
- input numbers must be converted from strings

```
>>> msg = input('Enter:')  
Enter:Hello  
>>> print (msg)  
Hello  
>>> number = input('Enter:')  
Enter:100  
>>> x = number - 10  
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>TypeError: unsupported  
operand type(s) for -: 'str' and 'int'  
>>> x = int(number) - 10  
>>> print( x)  
90
```



Looking Inside Strings

- We can get at any single character in a string using an index specified in **square brackets**
- The **index** value must be an integer and starts at zero
- The **index** value can be an expression that is computed



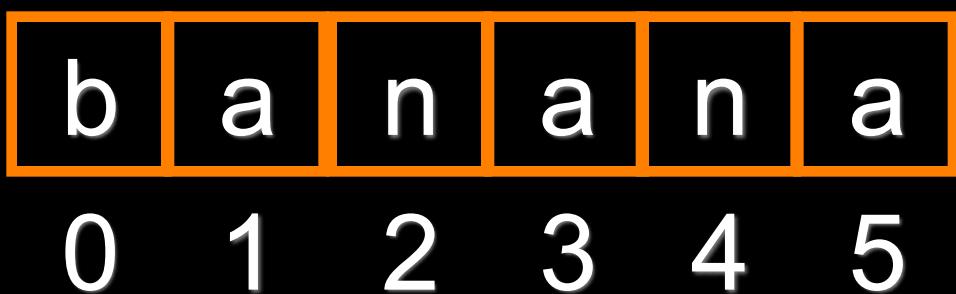
```
>>> fruit = 'banana'  
>>> letter = fruit[1]  
>>> print (letter)  
a  
>>> n = 3  
>>> w = fruit[n - 1]  
>>> print (w)  
n
```

A Character Too Far

- You will get a **python error** if you attempt to index beyond the end of a string.
- So be careful when constructing index values and slices

```
>>> zot = 'abc'  
>>> print zot[5]  
Traceback (most recent call last):  
File "<stdin>", line 1, in  
<module>IndexError: string index  
out of range  
>>>
```

Strings Have Length



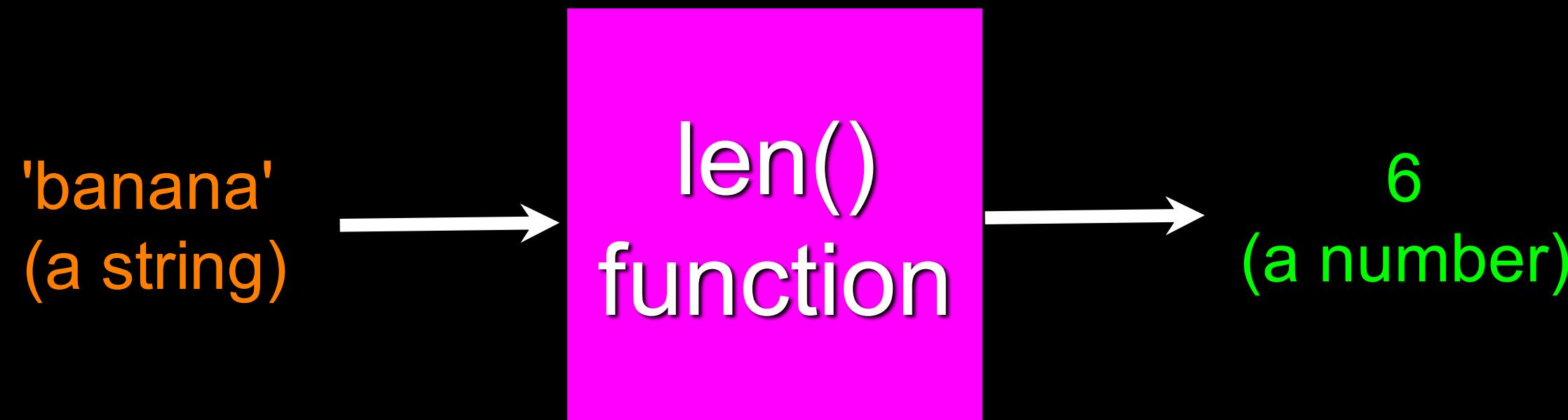
- There is a built-in function `len` that gives us the length of a string

```
>>> fruit = 'banana'  
>>> print (len(fruit))  
6
```

Len Function

```
>>> fruit = 'banana'  
>>> x = len(fruit)  
>>> print(x)  
6
```

A function is some stored code that we use. A function takes some **input** and produces an **output**.

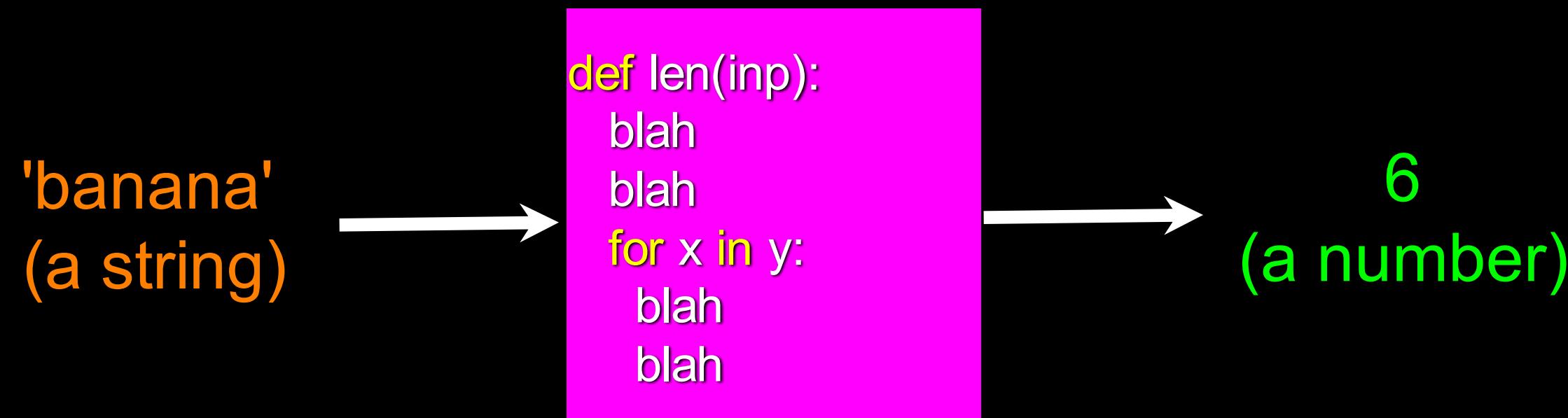


Guido wrote this code

Len Function

```
>>> fruit = 'banana'  
>>> x = len(fruit)  
>>> print x  
6
```

A function is some stored code that we use. A function takes some **input** and produces an **output**.



Looping Through Strings

- Using a `while` statement and an `iteration variable`, and the `len` function, we can construct a loop to look at each of the letters in a string individually

```
fruit = 'banana'          0 b
index = 0                  1 a
while index < len(fruit) : 2 n
    letter = fruit[index] 3 a
    print (index, letter)
    index = index + 1      4 n
                            5 a
```

Looping Through Strings

- A definite loop using a `for` statement is much more elegant
- The **iteration variable** is completely taken care of by the `for` loop

```
fruit = 'banana'  
for letter in fruit :  
    print (letter)
```

b
a
n
a
n
a

Looping Through Strings

- A definite loop using a `for` statement is much more elegant
- The `iteration variable` is completely taken care of by the `for` loop

```
fruit = 'banana'  
for letter in fruit :  
    print letter
```

```
index = 0  
while index < len(fruit) :  
    letter = fruit[index]  
    print (letter)  
    index = index + 1
```

b
a
n
a
n
a

Looping and Counting

- This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character.

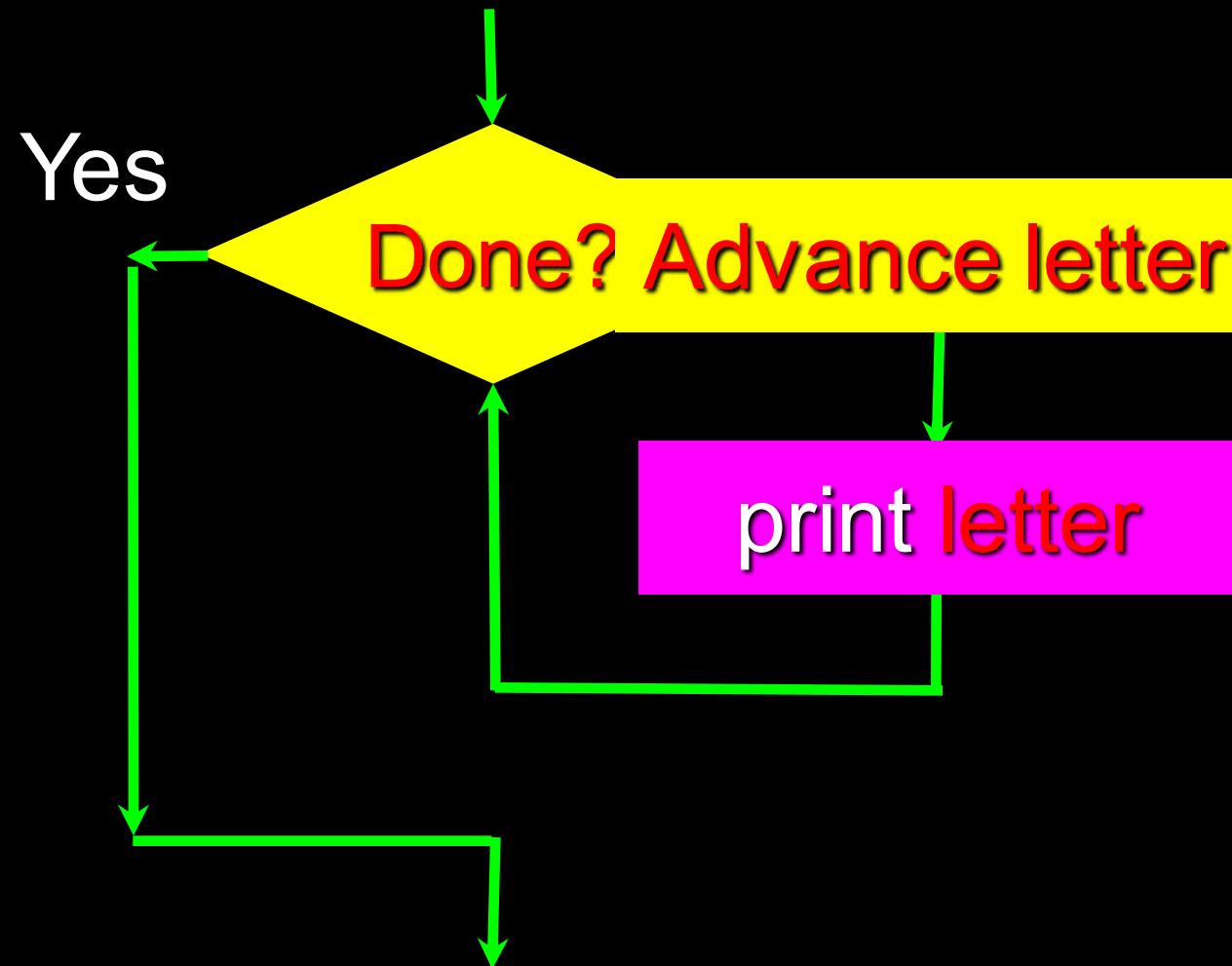
```
word = 'banana'  
count = 0  
for letter in word :  
    if letter == 'a' :  
        count = count + 1  
print (count)
```

Looking deeper into `in`

- The **iteration variable** “iterates” though the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the sequence
- The **iteration variable** moves through all of the values **in** the **sequence**

```
for letter in 'banana':  
    print(letter)
```

Six-character string



```
for letter in 'banana' :
    print (letter)
```

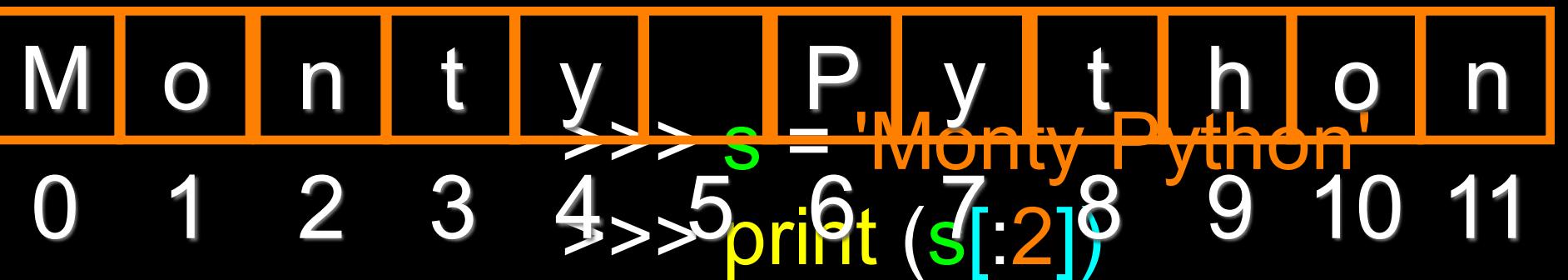
The **iteration variable** “iterates” through the **string** and the **block (body)** of code is executed once for each value **in** the **sequence**

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- We can also look at any continuous section of a string using a colon operator
- The second number is one beyond the end of the slice
 - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end

```
>>> s = 'Monty Python'
>>> print (s[0:4])
Mont
>>> print ( s[6:7])
P
>>> print ( s[6:20])
Python
```

Slicing Strings



- If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```

Mo
>>> print (s[8:])
Thon
>>> print (s[:])
Monty Python
>>> print( s[:-6])
Monty
>>> print (s[-4])
t
>>>print (s[-3:])
'hon'

```

String Concatenation

- When the `+` operator is applied to strings, it means "concatenation"

```
>>> a = 'Hello'  
>>> b = a + 'There'  
>>> print b  
HelloThere  
>>> c = a + ' ' + 'There'  
>>> print c  
Hello There  
>>>
```

Using `in` as an Operator

- The `in` keyword can also be used to check to see if one string is "in" another string
- The `in` expression is a logical expression and returns `True` or `False` and can be used in an `if` statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if 'a' in fruit :  
...     print 'Found it!'  
...  
Found it!  
>>>
```

String Comparison

```
if word == 'banana':  
    print ( 'All right, bananas.')  
  
if word < 'banana':  
    print ( 'Your word,' + word + ', comes before banana.')  
elif word > 'banana':  
    print ( 'Your word,' + word + ', comes after banana.')  
else:  
    print ('All right, bananas.')
```

String Library

- Python has a number of string **functions** which are in the **string library**
- These **functions** are already *built into* every string - we invoke them by appending the function to the string variable
- These **functions** do not modify the original string, instead they return a new string that has been altered

```
>>> greet = 'Hello Bob'  
>>> zap = greet.lower()  
>>> print (zap)  
hello bob  
>>> print (greet)  
Hello Bob  
>>> print ('Hi There'.lower())  
hi there  
>>>
```

```
>>> stuff = 'Hello world'  
>>> type(stuff)<type 'str'>  
>>> dir(stuff)  
['capitalize', 'center', 'count', 'decode', 'encode', 'endswith',  
'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit',  
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',  
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',  
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',  
'translate', 'upper', 'zfill']
```

`str.replace(old, new[, count])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.rfind(sub[, start[, end]])`

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within *s[start,end]*. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.rindex(sub[, start[, end]])`

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.rjust(width[, fillchar])`

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(s)`.

String Library

`str.capitalize()`

`str.center(width[, fillchar])`

`str.endswith(suffix[, start[, end]])`

`str.find(sub[, start[, end]])`

`str.lstrip([chars])`

`str.replace(old, new[, count])`

`str.lower()`

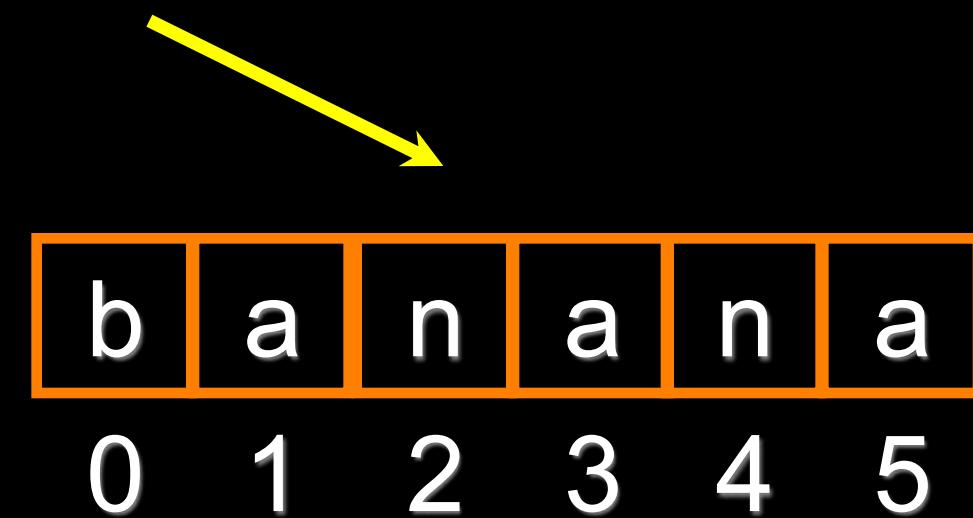
`str.rstrip([chars])`

`str.strip([chars])`

`str.upper()`

Searching a String

- We use the `find()` function to search for a substring within another string
- `find()` finds the first occurrence of the substring
- If the substring is not found, `find()` returns -1
- Remember that string position starts at zero



```
>>> fruit = 'banana'  
>>> pos = fruit.find('na')  
>>> print pos  
2  
>>> aa = fruit.find('z')  
>>> print aa  
-1
```

Making everything UPPER CASE

- You can make a copy of a string in **lower case** or **upper case**
- Often when we are searching for a string using **find()** - we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'  
>>> nnn = greet.upper()  
>>> print (nnn)  
HELLO BOB  
>>> www = greet.lower()  
>>> print (www)  
hello bob  
>>>
```

Search and Replace

- The `replace()` function is like a “search and replace” operation in a word processor
- It replaces **all occurrences** of the **search string** with the **replacement string**

```
>>> greet = 'Hello Bob'  
>>> nstr = greet.replace('Bob','Jane')  
>>> print (nstr)  
Hello Jane  
  
>>> nstr = greet.replace('o','X')  
>>> print (nstr)  
HellX BXb  
>>>
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- `lstrip()` and `rstrip()` to the left and right only
- `strip()` Removes both begin and ending whitespace

```
>>> greet = ' Hello Bob '
>>> greet.lstrip()
'Hello Bob '
>>> greet.rstrip()
' Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

Prefixes

```
>>> line = 'Please have a nice day'  
>>> line.startswith('Please')  
True  
>>> line.startswith('p')  
False
```

Summary

- String type
- Read/Convert
- Indexing strings []
- Slicing strings [2:4]
- Looping through strings with **for** and **while**
- Concatenating strings with +
- String operations

Regular Expressions

Problem

Write a Python code to check if the given mobile number is valid or not. The conditions to be satisfied for a mobile number are:

- a) Number of characters must be 10
- b) All characters must be digits and must not begin with a ‘0’

Validity of Mobile Number

Input	Processing	Output
A string representing a mobile number	Take character by character and check if it valid	Print valid or invalid

Test Case 1

- abc8967891
- Invalid
- Alphabets are not allowed

Test Case 2

- 440446845
- Invalid
- Only 9 digits

Test Case 3

- 0440446845
- Invalid
- Should not begin with a zero

Test Case 4

- 8440446845
- Valid
- All conditions statisfied

Python code to check validity of mobile number (Long Code)

```
import sys
number = input()
if len(number)!=10:
    print ('invalid')
    sys.exit(0)
if number[0]=='0':
    print ('invalid')
    sys.exit(0)
for chr in number:
    if chr.isalpha():
        print ('invalid')
        break
else:
    print('Valid')
```

- Manipulating text or data is a big thing
- If I were running an e-mail archiving company, and you, as one of my customers, requested all of the e-mail that you sent and received last February, for example, it would be nice if I could set a computer program to collate and forward that information to you, rather than having a human being read through your e-mail and process your request manually.

- Another example request might be to look for a subject line like “ILOVEYOU,” indicating a virus-infected message, and remove those e-mail messages from your personal archive.
- So this demands the question of how we can program machines with the ability to look for patterns in text.
- Regular expressions provide such an infrastructure for advanced text pattern matching, extraction, and/or search-and-replace functionality.
- Python supports regexes through the standard library `re` module

- regexes are strings containing text and special characters that describe a pattern with which to recognize multiple strings.
- Regexs without special characters

Regex Pattern	String(s) Matched
foo	foo
Python	Python
abc123	abc123

- These are simple expressions that match a single string
- Power of regular expressions comes in when special characters are used to define character sets, subgroup matching, and pattern repetition

Special Symbols and Characters

Notation	Description	Example Regex
Symbols		
<i>literal</i>	Match literal string value <i>literal</i>	foo
<i>re1 re2</i>	Match regular expressions <i>re1</i> or <i>re2</i>	foo bar
.	Match <i>any character</i> (except \n)	b.b
^	Match <i>start of string</i>	^Dear
\$	Match <i>end of string</i>	/bin/*sh\$
*	Match <i>0 or more</i> occurrences of preceding regex	[A-Za-z0-9]*
+	Match <i>1 or more</i> occurrences of preceding regex	[a-z]+\.\com
?	Match <i>0 or 1</i> occurrence(s) of preceding regex	goo?

Special Symbols and Characters

{ <i>N</i> }	Match <i>N</i> occurrences of preceding regex	[0-9]{3}
{ <i>M,N</i> }	Match from <i>M</i> to <i>N</i> occurrences of preceding regex	[0-9]{5,9}
[...]	Match any single character from <i>character class</i>	[aeiou]
[.. <i>x-y</i> ..]	Match any single character in the <i>range from x to y</i>	[0-9],[A-Za-z]

Special Symbols and Characters

Symbols

[^...]

Do not match any character from character class, including any ranges, if present

[^aeiou],
[^A-Za-z0-9_]

Matching Any Single Character (.)

- dot or period (.) symbol (letter, number, whitespace (not including “\n”), printable, non-printable, or a symbol) matches any single character except for \n
- To specify a dot character explicitly, you must escape its functionality with a backslash, as in “\.”

Regex Pattern	Strings Matched
f.o	Any character between "f" and "o"; for example, fao, f9o, f#o, etc.
..	Any pair of characters
.end	Any character before the string end

```
import re  
if re.match("f.o","fooo"):  
    print("Matched")  
else:  
    print("Not matched")
```

Output:

Prints matched

Since it searches only for the pattern ‘f.o’ in the string

```
import re  
if re.match("f.o$","fooo"):  
    print("Matched")  
else:  
    print("Not matched")
```

Check that the entire string starts with ‘f’, ends with ‘o’ and contain one letter in between

```
import re  
if re.match(..","fooo"):  
    print("Matched")  
else:  
    print("Not matched")
```

Matched

Two dots matches any pair of characters.

```
import re  
if re.match(..$","foo"):  
    print("Matched")  
else:  
    print("Not matched")
```

Not matched

Including a '\$' at the end will match only strings of length 2

```
import re  
if re.match(".end","bend"):  
    print("Matched")  
else:  
    print("Not matched")
```

Matched

The expression used in the example, matches any character for ‘.’

```
import re  
if re.match(".end","bends"):  
    print("Matched")  
else:  
    print("Not matched")
```

Prints Matched

```
import re  
if re.match(".end$","bends"):  
    print("Matched")  
else:  
    print("Not matched")
```

Prints Not matched - \$ check for end of string

Matching from the Beginning or End of Strings or Word Boundaries (^, \$)

^ - Match beginning of string

\$ - Match End of string

Regex Pattern	Strings Matched
^From	Any string that starts with From
/bin/tcsh\$	Any string that ends with /bin/tcsh
^Subject: hi\$	Any string consisting solely of the string Subject: hi

if you wanted to match any string that ended with a dollar sign, one possible regex solution would be the pattern `.*$`

Register number validity

Check whether the given register number of a VIT student is valid or not.

Example register number – 15bec1032

Register number is valid if it has two digits

Followed by three letters

Followed by four digits

Denoting Ranges (-) and Negation (^)

- brackets also support ranges of characters
- A hyphen between a pair of symbols enclosed in brackets is used to indicate a range of characters;
- For example A–Z, a–z, or 0–9 for uppercase letters, lowercase letters, and numeric digits, respectively

Regex Pattern	Strings Matched
z . [0-9]	"z" followed by any character then followed by a single digit
[r-u][env-y] [us]	"r," "s," "t," or "u" followed by "e," "n," "v," "w," "x," or "y" followed by "u" or "s"
[^aeiou]	A non-vowel character (Exercise: why do we say "non-vowels" rather than "consonants"?)
[^t\n]	Not a TAB or \n
"-a"]	In an ASCII system, all characters that fall between "" and "a," that is, between ordinals 34 and 97

Multiple Occurrence/Repetition Using Closure Operators (*, +, ?, {})

- special symbols *, +, and ?, all of which can be used to match single, multiple, or no occurrences of string patterns
- **Asterisk or star operator (*)** - match zero or more occurrences of the regex immediately to its left
- **Plus operator (+)** - Match one or more occurrences of a regex

- **Question mark operator (?)** - match exactly 0 or 1 occurrences of a regex.
- There are also brace operators ({}) with either a single value or a comma-separated pair of values. These indicate a match of exactly N occurrences (for {N}) or a range of occurrences; for example, {M, N} will match from M to N occurrences

Code to check the validity of register number

```
import re  
register= input()  
if re.match("^[1-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z][0-  
9][0-9][0-9][0-9]$",register):  
    print("Matched")  
else:  
    print("Not matched")
```

^ - denote begin (Meaning is different when we put this symbol inside the square bracket)

\$ - denote end

Regex Pattern**Strings Matched**

[dn]ot?

"d" or "n," followed by an "o" and, at most, one "t" after that; thus, do, no, dot, not.

0?[1-9]

Any numeric digit, possibly prepended with a "0." For example, the set of numeric representations of the months January to September, whether single or double-digits.

[0-9]{15,16}

Fifteen or sixteen digits (for example, credit card numbers).

Refined Code to check the validity of register number

{n} – indicate that the pattern before the braces should occur n times

```
import re  
register= input()
```

```
if re.match("^[1-9][0-9][a-zA-Z]{3}[0-  
9]{4}$",register):  
    print("Matched")  
else:  
    print("Not matched")
```

Check validity of Mobile Number (Shorter Code)

```
import re  
number = input()  
if re.match('^[0-9]{9}',number):  
    print('valid')  
else:  
    print('invalid')
```

Bug: Will also accept a843338320

Check validity of Mobile Number (Shorter Code)

```
import re  
number = input()  
if re.match('[1-9][0-9]{9}',number):  
    print('valid')  
else:  
    print('invalid')
```

Check validity of PAN card number with RE

```
import re  
pan=input()  
if len(pan)< 10 or len(pan)> 10 :  
    print ("PAN Number should be 10 characters")  
    exit
```

```
elif re.search("[^a-zA-Z0-9]",pan):  
    print ("No symbols allowed, only  
alphanumerics")  
    exit  
  
elif re.search("[0-9]",pan[0:5]):  
    print ("Invalid - 1")  
    exit
```

```
elif re.search("[A-Za-z]",pan[5:9]):  
    print ("Invalid - 2")  
    exit  
  
elif re.search("[0-9]",pan[-1]):  
    print ("Invalid - 3")  
    exit  
  
else:  
    print ("Your card "+ pan + " is valid")
```

Python read all input as string

In some cases it is necessary to check if the value entered is an integer

We can check it using regular expressions

Rules for an integer

optionally begin with a negative sign include ^ symbol

first digit must be a number other than zero

may be followed zero to any number of digits

string must end with it so add \$ symbol

```
import re  
register= input()  
  
#optionally begin with a negative sign include ^  
# symbol  
  
#first digit must be a number other than zero  
# may be followed zero to any number of digits  
# string must end with it so add $ symbol
```

```
if re.match("^\\-?[1-9][0-9]*$",register):
    #'\' is added in front of '-' to overcome its default
    meaning in REs
    print("Matched")
else:
    print("Not matched")
```

Rules for an integer or a floating point value
optionally begin with a negative sign include ^
symbol

first digit must be a number other than zero
may be followed zero to any number of digits
string must end with it so add \$ symbol

Optionally followed by a ‘:

Followed by zero or more digits

String ends here

```
import re  
register= input()  
if re.match("^\\-?[1-9][0-9]*\\.?[0-9]*$",register):  
    # '.' can occur zero or one time followed by a  
    # digit occurred zero to infinite number of times  
    print("Matched")  
else:  
    print("Not matched")
```

Problem

A farmer with a fox, a goose, and a sack of corn needs to cross a river. Now he is on the east side of the river and wants to go to west side. The farmer has a rowboat, but there is room for only the farmer and one of his three items. Unfortunately, both the fox and the goose are hungry. The fox cannot be left alone with the goose, or the fox will eat the goose. Likewise, the goose cannot be left alone with the sack of corn, or the goose will eat the corn. Given a sequence of moves find if all the three items fox, goose and corn are safe. The input sequence indicate the item carried by the farmer along with him in the boat. ‘F’ – Fox, ‘C’ – Corn, ‘G’ – Goose, N-Nothing. As he is now on the eastern side the first move is to west and direction alternates for each step.

Pseudocode

```
READ items_carried
SET east as G, C, F
SET west as empty
SET from_Dir = east and to_Dir = west
FOR each item in items_carried
    IF from_Dir == east THEN
        remove item from east and add to west
        IF east or west contains 'C' and 'G' or 'G' and 'F' THEN
            PRINT 'NOT SAFE'
            BREAK
    ELSE
        remove item from west and add to east
        IF east or west contains 'C' and 'G' or 'G' and 'F' THEN
            PRINT 'NOT SAFE'
            BREAK
    END IF
    IF from_Dir == east THEN
        SET from_Dir = west
        SET to_Dir = east
    ELSE
        SET from_Dir = east
        SET to_Dir = west
    END IF
END FOR
PRINT 'SAFE'
```

While going for a Shopping!!!???

Imagine you have the scores in “Python Programming” for 100 students. If you want to find the average score in Python...?

Simple Statistics

- Many programs deal with large collections of similar information.
 - Words in a document
 - Students in a course
 - Data from an experiment
 - Customers of a business
 - Graphics objects drawn on the screen
 - Cards in a deck

List

Examples

- Apple, Banana, Berry, Mango
- Football, Basketball, Throwball, Tennis, Hockey
- Sunrise, Sugar, Cheese, Butter, Pickle, Soap, Washing Powder, Oil....
- Agra, Delhi, Kashmir, Jaipur, Kolkata...

Introduction

- Contains **multiple values** that are **logically related**
- List is a type of **mutable sequence** in Python
- Each element of a list is assigned a number – **index / position**
- Can do indexing, slicing, adding, multiplying, and checking for membership
- Built-in functions for finding **length** of a sequence and for finding its largest and smallest elements

What is a List?

- Most flexible data type in Python
- Comma-separated items can be collected in square brackets
- Good thing is..
 - THE ITEMS IN THE LIST NEED NOT BE OF SAME TYPE

Creating a list

- Creating an EMPTY list

listname = []

Example:

L1 = []

MyList = []

Books = []

- Creating a list with items

listname = [item1, item2,]

Example:

Temp = [100, 99.8, 103, 102]

S = ['15BIT0001', 'Achu', 99.9]

L2 = [1, 2, 3, 4, 5, 6, 7]

Course = ['Python', 'C', 'C++',
'Java']

Accessing Values

- Using index or indices

```
>>>L1 = [1, 2, 3, 4, 5, 6]
```

```
>>>print (L1[3]) #indexing
```

```
>>>4
```

```
>>>print (L1[2:5]) #slicing
```

```
>>>[3, 4, 5]
```

Updating Elements

- Update an element in list using index

```
>>>L1 = [1, 2, 3, 4, 5, 6]
```

```
>>>L1[2] = 111
```

```
>>>L1
```

```
[1, 2, 111, 4, 5, 6]
```

Deleting Elements

- Delete an element in list using index

```
>>>L1 = [1, 2, 3, 4, 5, 6]
```

```
>>>del (L1[4])
```

```
>>>L1
```

```
[1, 2, 111, 4, 6]
```

Basic Operations in List

- `>>> len([1, 2, 3])` # Length
3
- `>>> [1, 2, 3] + [4, 5, 6]` # Concatenation
[1, 2, 3, 4, 5, 6]
- `>>> ['Ni!'] * 4` # Repetition
['Ni!', 'Ni!', 'Ni!', 'Ni!']

Basic Operations in List

- ```
>>> str([1, 2]) + "34" # Same as "[1, 2]" + "34"
```

```
'[1, 2]34'
```
- ```
>>> [1, 2] + list("34")
```



```
# Same as [1, 2] + ["3", "4"]
```



```
[1, 2, '3', '4']
```

List Iteration

- `>>> 3 in [1, 2, 3]` # Membership
True
- `>>> for x in [1, 2, 3]:
 print(x, end=' ')`
Iteration (2.X uses: `print x,`) ... 1 2 3

List Comprehensions

```
>>> res = [c * 4 for c in 'SPAM']
```

List comprehensions

```
>>> res
```

```
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

- expression is functionally equivalent to a for loop that builds up a list of results manually
- list comprehensions are simpler to code and likely faster to run today:

List Comprehensions

```
# List comprehension equivalent ...
```

```
>>> res = []
```

```
>>> for c in 'SPAM':  
    res.append(c * 4)
```

```
>>> res
```

```
['SSSS', 'PPPP', 'AAAA', 'MMMM']
```

Map

- map built-in function applies a function to **items** in a sequence and collects all the results in a new list
- `>>> list(map(abs, [-1, -2, 0, 1, 2]))`
- # Map a function across a sequence
`[1, 2, 0, 1, 2]`

Indexing, Slicing

```
>>> L = ['spam', 'Spam', 'SPAM!']
```

```
>>> L[2]      # Offsets start at zero
```

```
'SPAM!'
```

```
>>> L[-2]     # Negative: count from the right
```

```
'Spam'
```

```
>>> L[1:]     # Slicing fetches sections
```

```
['Spam', 'SPAM!']
```

Matrixes

- a basic 3×3 two-dimensional list-based array:
`>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
- With **one index**, you get an **entire row** (really, a nested sublist), and with two, you get an item within the row:

```
>>> matrix[1]
```

```
[4, 5, 6]
```

Matrixes

```
>>> matrix[1][1]
```

```
5
```

```
>>> matrix[2][0]
```

```
7
```

```
>>> matrix = [[1, 2, 3],  
             [4, 5, 6],  
             [7, 8, 9]]
```

```
>>> matrix[1][1]
```

```
5
```

Insertion, Deletion and Replacement

```
>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5]      # Replacement/insertion
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7]    # Insertion (replace nothing)
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = []        # Deletion (insert nothing)
>>> L
[1, 7, 4, 5, 3]
```

Insertion, Deletion and Replacement

```
>>> L = [1]
```

```
>>> L[:0] = [2, 3, 4]
```

Insert all at :0, an empty slice at front

```
>>> L
```

```
[2, 3, 4, 1]
```

```
>>> L[len(L):] = [5, 6, 7]
```

Insert all at len(L):, an empty slice at end

```
>>> L
```

```
[2, 3, 4, 1, 5, 6, 7]
```

List method calls

```
>>> L = ['eat', 'more', 'SPAM!']
```

```
>>> L.append('please')
```

Append method call: add item at end

```
>>> L
```

```
['eat', 'more', 'SPAM!', 'please']
```

```
>>> L.sort() # Sort list items ('S' < 'e')
```

```
>>> L
```

```
['SPAM!', 'eat', 'more', 'please']
```

More on Sorting Lists

```
>>> L = ['abc', 'ABD', 'aBe']
```

```
>>> L.sort() # Sort with mixed case
```

```
>>> L
```

```
['ABD', 'aBe', 'abc']
```

```
>>> L = ['abc', 'ABD', 'aBe']
```

```
>>> L.sort(key=str.lower) # Normalize to lowercase
```

```
>>> L
```

```
['abc', 'ABD', 'aBe']
```

More on Sorting Lists

```
>>> L.sort(key=str.lower, reverse=True)
```

Change sort order

```
>>> L ['aBe', 'ABD', 'abc']
```

Other common list methods

```
>>> L = [1, 2]
```

```
>>> L.extend([3, 4, 5])
```

Add many items at end (like in-place +)

```
>>> L [1, 2, 3, 4, 5]
```

```
>>> L.pop()
```

Delete and return last item

Other common list methods

```
>>> L
```

```
[1, 2, 3, 4]
```

```
>>> L.reverse() # In-place reversal method
```

```
>>> L
```

```
[4, 3, 2, 1]
```

```
>>> list(reversed(L)) # Reversal built-in with a result  
(iterator)
```

```
[1, 2, 3, 4]
```

Other common list methods

```
>>> L = ['spam', 'eggs', 'ham']
```

```
>>> L.index('eggs')      # Index of an object (search/find)
```

```
1
```

```
>>> L.insert(1, 'toast') # Insert at position
```

```
>>> L
```

```
['spam', 'toast', 'eggs', 'ham']
```

```
>>> L.remove('eggs')      # Delete by value
```

```
>>> L
```

```
['spam', 'toast', 'ham']
```

Other common list methods

```
>>> L.pop(1) # Delete by position 'toast'
```

```
>>> L
```

```
['spam', 'ham']
```

```
>>> L.count('spam') # Number of occurrences 1
```

```
1
```

Other common list methods

```
>>> L = ['spam', 'eggs', 'ham', 'toast']
```

```
>>> del L[0] # Delete one item
```

```
>>> L
```

```
['eggs', 'ham', 'toast']
```

```
>>> del L[1:] # Delete an entire section
```

```
>>> L # Same as L[1:] = []
```

```
['eggs']
```

Other common list methods

```
>>> L = ['Already', 'got', 'one']
```

```
>>> L[1:] = []
```

```
>>> L
```

```
['Already']
```

```
>>> L[0] = []
```

```
>>> L
```

```
[]()
```

Hence...

- A list is a sequence of items stored as a single object.
- Items in a list can be accessed by indexing, and sub lists can be accessed by slicing.
- Lists are mutable; individual items or entire slices can be replaced through assignment statements.
- Lists support a number of convenient and frequently used methods.
- Lists will grow and shrink as needed.

Strings and Lists

```
>>> S = 'spammy'  
>>> L = list(S)  
>>> L  
['s', 'p', 'a', 'm', 'm', 'y']  
>>> L[3] = 'x' # Works for lists, not strings  
>>> L[4] = 'x'  
>>> L  
['s', 'p', 'a', 'x', 'x', 'y']  
>>> S = " ".join(L) #uses " for joining elements of list  
>>> S  
'spaxxy'
```

Strings and Lists

```
>>> 'SPAM'.join(['eggs', 'sausage', 'ham', 'toast'])
```

```
'eggsSPAMsausageSPAMhamSPAMtoast'
```

uses 'SPAM' for joining elements of list

```
>>> line = 'aaa bbb ccc'
```

```
>>> cols = line.split()
```

```
>>> cols
```

```
['aaa', 'bbb', 'ccc']
```

Statistics using List

Find the mean, standard deviation and median
of a set of numbers

Table 8-1. Common list literals and operations

Operation	Interpretation
<code>L = []</code>	An empty list
<code>L = [123, 'abc', 1.23, {}]</code>	Four items: indexes 0..3
<code>L = ['Bob', 40.0, ['dev', 'mgr']]</code>	Nested sublists
<code>L = list('spam')</code>	List of an iterable's items, list of successive integers
<code>L = list(range(-4, 4))</code>	
<code>L[i]</code>	Index, index of index, slice, length
<code>L[i][j]</code>	
<code>L[i:j]</code>	
<code>len(L)</code>	
<code>L1 + L2</code>	Concatenate, repeat

Operation	Interpretation
<code>L * 3</code>	
<code>for x in L: print(x)</code>	Iteration, membership
<code>3 in L</code>	
<code>L.append(4)</code>	Methods: growing
<code>L.extend([5,6,7])</code>	
<code>L.insert(i, X)</code>	
<code>L.index(X)</code>	Methods: searching
<code>L.count(X)</code>	
<code>L.sort()</code>	Methods: sorting, reversing,
<code>L.reverse()</code>	copying (3.3+), clearing (3.3+)
<code>L.copy()</code>	
<code>L.clear()</code>	
<code>L.pop(i)</code>	Methods, statements: shrinking
<code>L.remove(X)</code>	
<code>del L[i]</code>	
<code>del L[i:j]</code>	
<code>L[i:j] = []</code>	
<code>L[i] = 3</code>	Index assignment, slice assignment
<code>L[i:j] = [4,5,6]</code>	
<code>L = [x**2 for x in range(5)]</code>	List comprehensions and maps (Chapter 4 , Chapter 14 , Chapter 20)
<code>list(map(ord, 'spam'))</code>	

Exercise 1

Given positions of coins of player1 and player2 in a 3X3 Tic Tac Toc board, write a program to determine if the board position is a solution and if so identify the winner of the game. In a Tic Tac Toc problem, if the coins in a row or column or along a diagonal is of the same player then he has won the game. Assume that player1 uses '1' as his coin and player2 uses '2' as his coin. '0' in the board represent empty cell.

Exercise 2

- In a supermarket there are two sections S1 and S2. The sales details of item₁ to item_n of section1 and item₁ to item_p of section2 are maintained in a sorted order. Write a program to merge the elements of the two sorted lists to form the consolidated list.

Exercise 3

- Watson gives Sherlock an list of N numbers. Then he asks him to determine if there exists an element in the list such that the sum of the elements on its left is equal to the sum of the elements on its right. If there are no elements to the left/right, then the sum is considered to be zero.

Exercise 4

- Sunny and Johnny together have M dollars they want to spend on ice cream. The parlor offers N flavors, and they want to choose two flavors so that they end up spending the whole amount.
- You are given the cost of these flavors. The cost of the i th flavor is denoted by c_i . You have to display the indices of the two flavors whose sum is M .

Exercise 5

- Given a list of integer values, find the fraction of count of positive numbers, negative numbers and zeroes to the total numbers. Print the value of the fractions correct to 3 decimal places.

Exercise 6

- Given N integers, count the number of pairs of integers whose difference is K .

Problem

Write a kids play program that prints the capital of a country given the name of the country.

PAC For Quiz Problem

Input	Processing	Output
A set of question/answer pairs and a question	Map each question to the corresponding answer. Find the answer for the given question	Answer for the question

Pseudocode

```
READ num_of_countries
FOR i=0 to num_of_countries
    READ name_of_country
    READ capital_of_country
    MAP name_of_country to capital_of_country
END FOR
READ country_asked
GET capital for country_asked
PRINT capital
```

Already we know

- To read values from user
- Print values
- We have to yet know to
 - Map a pair of values

Python provides Dictionaries to Map a pair of values

Introduction to Dictionaries

- Pair of items
- Each pair has key and value
- Keys should be unique
- Key and value are separated by :
- Each pair is separated by ,

Example:

```
dict = {'Alice' : 1234, 'Bob' : 1235}
```

Properties of Dictionaries

- unordered mutable collections;
- items are stored and fetched by key,
- Accessed by key, not offset position
- Unordered collections of arbitrary objects
- Variable-length, heterogeneous, and arbitrarily nestable

Creating a Dictionary

- Creating an EMPTY dictionary

```
dictname = {}
```

Example:

```
Dict1 = {}
```

```
MyDict = {}
```

```
Books = {}
```

- Creating a dictionary with items
dictname = {key1:val1, key2:val2,}

Example:

```
MyDict = { 1 : 'Chocolate', 2 : 'Icecream'}
```

```
MyCourse = {'MS' : 'Python', 'IT' : 'C',  
           'CSE' : 'C++', 'MCA' : 'Java'}
```

```
MyCircle = {'Hubby':9486028245,  
           'Mom':9486301601}
```

Accessing Values

- Using keys within square brackets

```
>>> print (MyDict[1])
```

‘Chocholate’

```
>>> print (MyCourse['CSE'])
```

‘C++’

Updating Elements

- update by adding a new item (key-value) pair
- modify an existing entry

```
>>>MyDict[1] = 'Pizza'
```

```
>>>MyCourse['MCA'] = 'UML'
```

Deleting Elements

- remove an element in a dictionary using the key

```
>>>del MyCourse['IT']
```

- remove all the elements

```
>>>MyCourse.clear()
```

- delete the dictionary

```
>>>del MyCourse
```

Basic Operations

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}  
  
>>> len(D) # Number of entries in dictionary 3  
  
>>> 'ham' in D # Key membership test  
True  
  
>>> list(D.keys()) # Create a new list of D's keys  
['eggs', 'spam', 'ham']
```

Basic Operations

```
>>> list(D.values()) [3, 2, 1]
```

```
>>> list(D.items()) [('eggs', 3), ('spam', 2), ('ham', 1)]
```

```
>>> D.get('spam') # A key that is there
```

2

```
>>> print(D.get('toast')) # A key that is missing
```

None

Update Method

```
>>> D
```

```
{'eggs': 3, 'spam': 2, 'ham': 1}
```

```
>>> D2 = {'toast':4, 'muffin':5}
```

```
>>> D.update(D2)
```

```
>>> D
```

```
{'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2, 'ham': 1}
```

#unordered

Pop Method

Delete and return value for a given key

```
>>> D = {'eggs': 3, 'muffin': 5, 'toast': 4, 'spam': 2,  
        'ham': 1}
```

```
>>> D.pop('muffin')
```

```
5
```

```
>>> D.pop('toast')
```

```
4
```

```
>>> D
```

```
{'eggs': 3, 'spam': 2, 'ham': 1}
```

List vs Dictionary

```
>>> L = []
```

```
>>> L[99] = 'spam'
```

```
Traceback (most recent call last): File "<stdin>", line  
1, in ? IndexError: list assignment index out of  
range
```

```
>>> D = {}
```

```
>>> D[99] = 'spam'
```

```
>>> D {99: 'spam'}
```

Nesting in dictionaries

```
>>> jobs = []  
>>> jobs.append('developer')  
>>> jobs.append('manager')
```

Nesting in dictionaries

```
rec = {}
```

```
>>> rec['name'] = 'Bob'
```

```
>>> rec['age'] = 40.5
```

```
>>> rec['job'] = jobs
```

Nesting in dictionaries

```
>>> rec
```

```
{'name': 'Bob', 'age': 40.5, 'job': ['developer',  
 'manager']}
```

Nesting in dictionaries

```
>>> rec['name']
```

```
'Bob'
```

```
>>> rec['job']
```

```
['developer', 'manager']
```

```
>>> rec['job'][1]
```

```
'manager'
```

Other Ways to Make Dictionaries

```
D = {'name': 'Bob', 'age': 40}
```

```
D = {} # Assign by keys dynamically
```

```
D['name'] = 'Bob'
```

```
D['age'] = 40
```

Creating a dictionary by assignment

```
dict(name='Bob', age=40)
```

Creating dictionary with tuples form

```
dict([('name', 'Bob'), ('age', 40)])
```

Comprehensions in Dictionaries

```
>>> D = {k: v for (k, v) in zip(['a', 'b', 'c'], [1, 2, 3])}
```

```
>>> D
```

```
{'b': 2, 'c': 3, 'a': 1}
```

```
>>> D = {x: x ** 2 for x in [1, 2, 3, 4]}
```

```
# Or: range(1, 5)
```

Comprehensions in Dictionaries

```
>>> D
```

```
{1: 1, 2: 4, 3: 9, 4: 16}
```

```
>>> D = {c: c * 4 for c in 'SPAM'}
```

```
>>> D
```

```
{'S': 'SSSS', 'P': 'PPPP', 'A': 'AAAA', 'M': 'MMMM'}
```

Comprehensions in Dictionaries

```
>>> D = {c.lower(): c + '!' for c in ['SPAM', 'EGGS',  
'HAM']}  
  
>>> D  
  
{'eggs': 'EGGS!', 'spam': 'SPAM!', 'ham': 'HAM!'}
```

Initializing Dictionaries

Initialize dict from keys

```
>>> D = dict.fromkeys(['a', 'b', 'c'], 0)
```

```
>>> D {'b': 0, 'c': 0, 'a': 0}
```

Same, but with a comprehension

```
>>> D = {k:0 for k in ['a', 'b', 'c']}
```

```
>>> D {'b': 0, 'c': 0, 'a': 0}
```

Initializing Dictionaries

Comprehension

```
>>> D = {k: None for k in 'spam'}
```

```
>>> D {'s': None, 'p': None, 'a': None, 'm': None}
```

Dictionary methods

- `<dict>.items()`
 - displays the items in the dictionary (pair of keys and values)
- `<dict>.keys()`
 - display the keys in the dictionary
- `<dict>.values()`
 - displays the values in the dictionary
- `<dict>.pop()`
 - removes the last item from the dictionary
- `<dict2> = <dict1>.copy()`
 - copies the items from dict1 to dict2
- `<dict>.clear()`
 - removes all the items from the dictionary

Other methods

- `str(dict)`
 - produces printable string representation of a dictionary
- `len(dict)`
 - returns the number of items in the dictionary

Dictionaries can replace elif ladder

```
print ({1:'one',2:'two',3:'three',4:'four',5:'five'}  
[choice])
```

if choice = 3 then the code prints three

Tuples

Problem

A hospital has received a set of lab reports. Totally five tests are conducted in the lab and the report is prepared. The report consist of name of the test and the value observed for that particular patient. Given the details of a test made for a patient, write an algorithm and the subsequent Python program to print if the test result is normal or not normal by referring the values in Table 1. Since the value is sensitive, provide a mechanism so that the values do not get altered.

Name of the Test	Minimum Value	Maximum Value
Test1	20	30
Test2	35.5	40
Test3	12	15
Test4	120	150
Test5	80	120

PAC For Lab Test Problem

Input	Processing	Output
Test name and a pair of values indicating the minimum and the maximum value for the five tests	Check if the observed value is in the range of permissible values for the test and print 'Normal' or 'Abnormal'	Print 'Normal' or 'Abnormal'
Name of the test and the value observed		

Pseudocode LabTest Problem

FOR i =0 to 5

 READ test_Name_i

 READ minimum_i

 READ maximum_i

 Map test_Name_i to minimum_i and maximum_i

 READ test_Name_Chk

 READ observed_Value

END_FOR

```
IF observed_Value > min of test_Name_Chk  
and observed_Value < max of test_Name_Chk  
THEN  
    PRINT 'Normal'  
ELSE  
    PRINT 'Abnormal'  
END IF
```

Store the values such that it is not getting modified

We Already Know

- To read values
- Map a value to another - Dictionary
- Print Values
- Form a pair of values – List – But the values can be changed
- Yet to learn about pairing values that cannot be modified

Tuples

Sequence of **immutable** Python objects

Tuples cannot be changed like lists and tuples use **parentheses**, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values.

Optionally you can put these comma-separated values between **parentheses** also.

For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 );
```

```
tup3 = "a", "b", "c", "d";
```

empty tuple –

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma –

```
a = (50) # an integer
```

```
tup1 = (50,); # tuple containing an integer
```

tuple indices start at 0

```
print ("tup1[0]: ", tup1[0]) # print physics
```

```
print ("tup2[1:5]: ", tup2[1:5]) # print (2,3,4,5)
```

Tuples in Action

```
>>> (1, 2) + (3, 4) # Concatenation  
(1, 2, 3, 4)
```

```
>>> (1, 2) * 4          # Repetition  
(1, 2, 1, 2, 1, 2, 1, 2)
```

```
>>> T = (1, 2, 3, 4)    # Indexing, slicing
```

```
>>> T[0], T[1:3]  
(1, (2, 3))
```

Sorted method in Tuples

```
>>> tmp = ['aa', 'bb', 'cc', 'dd']
```

```
>>> T = tuple(tmp) # Make a tuple from the list's items
```

```
>>> T
```

```
('aa', 'bb', 'cc', 'dd')
```

```
>>> sorted(T)
```

```
['aa', 'bb', 'cc', 'dd']
```

List comprehensions can also be used with tuples.

The following, for example, makes a list from a tuple, adding 20 to each item along the way:

```
>>> T = (1, 2, 3, 4, 5)
```

```
>>> L = [x + 20 for x in T]
```

Equivalent to:

```
>>>L = []
```

```
>>>for x in T:
```

```
    L.append(x+20)
```

```
>>> L
```

```
[21, 22, 23, 24, 25]
```

Index method can be used to find the position of particular value in the tuple.

```
>>> T = (1, 2, 3, 2, 4, 2)
```

```
>>> T.index(2)      # Offset of first appearance of 2
```

```
1
```

```
>>> T.index(2, 2)    # Offset of appearance after offset 2
```

```
3
```

```
>>> T.count(2)      # How many 2s are there?
```

```
3
```

Nested Tuples

```
>>> T = (1, [2, 3], 4)
```

```
>>> T[1] = 'spam' # fails: can't change
```

tuple itself `TypeError: object doesn't support item assignment`

```
>>> T[1][0] = 'spam'
```

Works: can change mutables inside

```
>>> T
```

```
(1, ['spam', 3], 4)
```

```
>>> bob = ('Bob', 40.5, ['dev', 'mgr'])
```

Tuple record

```
>>> bob
```

```
('Bob', 40.5, ['dev', 'mgr'])
```

```
>>> bob[0], bob[2]
```

Access by position

```
('Bob', ['dev', 'mgr'])
```

Prepares a Dictionary record from tuple

```
>>> bob = dict(name='Bob', age=40.5, jobs=['dev',  
'mgr'])
```

```
>>> bob  
{'jobs': ['dev', 'mgr'], 'name': 'Bob', 'age': 40.5}
```

```
>>> bob['name'], bob['jobs'] # Access by key  
('Bob', ['dev', 'mgr'])
```

Dictionary to Tuple

We can convert parts of dictionary to a tuple if needed:

```
>>> tuple(bob.values())          # Values to tuple  
(['dev', 'mgr'], 'Bob', 40.5)  
  
>>> list(bob.items())          # Items to list of tuples  
[('jobs', ['dev', 'mgr']), ('name', 'Bob'), ('age', 40.5)]
```

Using Tuples

Immutable which means you **cannot update or change the values of tuple elements**

```
tup1 = (12, 34.56);
```

```
tup2 = ('abc', 'xyz');
```

Following action is **not valid for tuples**

```
tup1[0] = 100;
```

You are able to take portions of existing tuples to
create new tuples as the following example
demonstrates

```
tup3 = tup1 + tup2;  
print (tup3)
```

Delete Tuple Elements

Removing individual tuple elements is **not possible**

But possible to **remove** an entire tuple

```
tup = ('physics', 'chemistry', 1997, 2000);
```

```
print (tup)
```

```
del tup;
```

```
print ("After deleting tup : ")
```

```
print (tup)
```

Error

Basic Tuples Operations

Python Expression	Results	Description
<code>len((1, 2, 3))</code>	3	Length
<code>(1, 2, 3) + (4, 5, 6)</code>	<code>(1, 2, 3, 4, 5, 6)</code>	Concatenation
<code>('Hi!',) * 4</code>	<code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code>	Repetition
<code>3 in (1, 2, 3)</code>	True	Membership
<code>for x in (1, 2, 3): print x,</code>	1 2 3	Iteration

Indexing, Slicing

If `L = ('spam', 'Spam', 'SPAM!')`

Python Expression	Results	Description
<code>L[2]</code>	'SPAM!'	Offsets start at zero
<code>L[-2]</code>	'Spam'	Negative: count from the right
<code>L[1:]</code>	['Spam', 'SPAM!']	Slicing fetches sections

Built-in Tuple Functions

```
tuple1, tuple2 = (123, 'xyz'), (456, 'abc')
```

```
len(tuple1)
```

```
2
```

When we have numerical tuple:

```
t1 = (1,2,3,7,4)
```

```
max(t1) #prints 7
```

```
min(t1) #prints 1
```

Converts a list into a tuple

`tuple(seq)`

`t2=tuple([2,4])`

```
>>> t2  
(2, 4)
```

Sets

Problem:

An University has published the results of the term end examination conducted in April. List of failures in physics, mathematics, chemistry and computer science is available. Write a program to find the number of failures in the examination. This includes the count of failures in one or more subjects

PAC For University Result Problem

Input	Processing	Output
<p>Read the register number of failures in Maths, Chemistry and Computer Science</p>	<p>Create a list of register numbers who have failed in one or more subjects</p> <p>Count the count of failures</p>	<p>Print Count</p>

Pseudocode

```
READ maths_failure, physics_failure, chemistry_failure and cs_failure
Let failure be empty
FOR each item in maths_failure
    ADD item to failure
FOR each item in physics_failure
    IF item is not in failure THEN
        ADD item to failure
    END IF
FOR each item in chemistry_failure
    IF item is not in failure THEN
        ADD item to failure
    END IF
FOR each item in cs_failure
    IF item is not in failure THEN
        ADD item to failure
    END IF
SET count = 0
FOR each item in failure
    count = count + 1
PRINT count
```

Sets

an unordered collection of unique and immutable objects that supports operations corresponding to mathematical set theory

Set is mutable

No duplicates

Sets are iterable, can grow and shrink on demand, and may contain a variety of object types

Does not support indexing

```
x = {1, 2, 3, 4}
```

```
y = {'apple','ball','cat'}
```

```
x1 = set('spam') # Prepare set from a string  
print (x1)
```

```
{'s', 'a', 'p', 'm'}
```

```
x1.add('alot') # Add an element to the set  
print (x1)
```

```
{'s', 'a', 'p', 'alot', 'm'}
```

Set Operations

Let $S1 = \{1, 2, 3, 4\}$

Union ()

$S2 = \{1, 5, 3, 6\} | S1$

`print(S2)` # prints $\{1, 2, 3, 4, 5, 6\}$

Intersection (&)

$S2 = S1 & \{1, 3\}$

`print(S2)` # prints $\{1, 3\}$

Difference (-)

```
S2 = S1 - {1, 3, 4}
```

```
print(S2) # prints {2}
```

Super set (>)

```
S2 = S1 > {1, 3}
```

```
print(S2) # prints True
```

Empty sets must be created with the set built-in, and print the same way

```
S2 = S1 - {1, 2, 3, 4}
```

```
print(S2) # prints set() – Empty set
```

Empty curly braces represent empty dictionary but
not set

In interactive mode – type({}) gives

```
<class 'dict'>
```

```
>>> {1, 2, 3} | {3, 4}
```

```
{1, 2, 3, 4}
```

```
>>> {1, 2, 3} | [3, 4]
```

```
TypeError: unsupported operand type(s) for |: 'set'  
and 'list'
```

```
>>> {1, 2, 3} | set([3, 4]) #Convert list to set and work
```

```
{1,2,3,4}
```

```
>>> {1, 2, 3}.union([3, 4]) #if you use union it will not  
give you the error
```

```
{1,2,3,4}
```

```
>>> {1, 2, 3}.union({3, 4})
```

```
{1,2,3,4}
```

Immutable constraints and frozen sets

Can only contain immutable (a.k.a. “hashable”) object types

lists and dictionaries cannot be embedded in sets, but tuples can if you need to store compound values.

Tuples compare by their full values when used in set operations:

```
>>> S  
{1.23}
```

```
>>> S.add([1, 2, 3])
```

TypeError: unhashable type: 'list'

```
>>> S.add({'a':1})  
TypeError: unhashable type: 'dict'
```

Works for tuples:

```
>>> S.add((1, 2, 3))  
>>> S  
{1.23, (1, 2, 3)}
```

```
>>> S | {(4, 5, 6), (1, 2, 3)}
```

```
{1.23, (4, 5, 6), (1, 2, 3)}
```

```
>>> (1, 2, 3) in S # Check for tuple as a whole
```

```
True
```

```
>>> (1, 4, 3) in S
```

```
False
```

clear()

All elements will **removed** from a set.

```
>>> cities = {"Stuttgart", "Konstanz", "Freiburg"}  
>>> cities.clear()  
>>> cities  
set() # empty  
>>>
```

Copy

Creates a **shallow copy**, which is returned.

```
>>> more_cities = {"Winterthur","Schaffhausen","St.  
Gallen"}  
>>> cities_backup = more_cities.copy()  
>>> more_cities.clear()  
>>> cities_backup # copied value is still available  
{'St. Gallen', 'Winterthur', 'Schaffhausen'}
```

Just in case, you might think, an **assignment** might be enough:

```
>>> more_cities = {"Winterthur","Schaffhausen","St.  
Gallen"}  
>>> cities_backup = more_cities #creates alias name  
>>> more_cities.clear()  
>>> cities_backup  
set()  
>>>
```

The assignment "cities_backup = more_cities" just creates a pointer, i.e. **another name**, to the same data structure.

difference_update()

removes all elements of another set from this set.

`x.difference_update()` is the same as "`x = x - y`"

```
>>> x = {"a", "b", "c", "d", "e"}
```

```
>>> y = {"b", "c"}
```

```
>>> x.difference_update(y)
```

```
>>> x
```

```
{'a', 'e', 'd'}
```

discard(el)

el will be removed from the set, if it is contained in the set and nothing will be done otherwise

```
>>> x = {"a","b","c","d","e"}
```

```
>>> x.discard("a")
```

```
>>> x
```

```
{'c', 'b', 'e', 'd'}
```

```
>>> x.discard("z")
```

```
>>> x
```

```
{'c', 'b', 'e', 'd'}
```

remove(el)

works like `discard()`, but if `el` is not a member of the set, a `KeyError` will be raised.

```
>>> x = {"a","b","c","d","e"}  
>>> x.remove("a")  
>>> x  
{'c', 'b', 'e', 'd'}  
  
>>> x.remove("z")  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    KeyError: 'z'
```

isdisjoint()

This method returns True if two sets have a null intersection

issubset()

x.issubset(y) returns True, if x is a subset of y

"<=" is an abbreviation for "Subset of" and ">=" for "superset of"

"<" is used to check if a set is a proper subset of a set

issuperset()

x.issuperset(y) returns True, if x is a superset of y.
">=" - abbreviation for "issuperset of"
">" - to check if a set is a proper superset of a set

```
>>> x = {"a","b","c","d","e"}
```

```
>>> y = {"c","d"}
```

```
>>> x.issuperset(y)
```

True

```
>>> x > y
```

True

```
>>> x >= y
```

True

```
>>> x >= x
```

True

```
>>> x > x
```

False

```
>>> x.issuperset(x)
```

True

```
>>> x = {"a","b","c","d","e"}
```

```
>>> y = {"c","d"}
```

```
>>> x.issubset(y)
```

False

```
>>> y.issubset(x)
```

True

```
>>> x < y
```

False

```
>>> y < x # y is a proper subset of x
```

True

```
>>> x < x # a set is not a proper subset of oneself.
```

False

```
>>> x <= x
```

True

pop()

pop() removes and returns an arbitrary set element.

The method raises a **KeyError** if the set is empty

```
>>> x = {"a","b","c","d","e"}
```

```
>>> x.pop()
```

```
'a'
```

```
>>> x.pop()
```

```
'c'
```

Sets themselves are mutable too, and so cannot be nested in other sets directly;

if you need to store a set inside another set, the `frozenset` built-in call works just like `set` but creates an immutable set that cannot change and thus can be embedded in other sets

To create frozenset:

```
cities = frozenset(["Frankfurt", "Basel", "Freiburg"])
```

```
cities.add("Strasbourg") #cannot modify
```

Traceback (most recent call last):

```
  File "<stdin>", line 1, in <module>
```

```
AttributeError: 'frozenset' object has no attribute 'add'
```

Set comprehensions

run a loop and collect the result of an expression on each iteration

result is a new set you create by running the code, with all the normal set behavior

```
>>> {x ** 2 for x in [1, 2, 3, 4]}
```

```
{16, 1, 4, 9}
```

```
>>> {x for x in 'spam'}
```

```
{'m', 's', 'p', 'a'}
```

```
>>> S = {c * 4 for c in 'spam'}
```

```
>>> print(S)  
{'pppp','aaaa','ssss', 'mmmm'}
```

```
>>>> S = {c * 4 for c in 'spamham'}  
{'pppp','aaaa','ssss', 'mmmm','hhh'}
```

```
>>>S | {'mmmm', 'xxxx'}  
{'pppp', 'xxxx', 'mmmm', 'aaaa', 'ssss'}
```

```
>>> S & {'mmmm', 'xxxx'}  
{'mmmm'}
```

```
math = set()
phy = set()
che = set()
cs = set()
m_N = int(input())
for i in range(0,m_N):
    val =input()
    math = math|{val}
m_P = int(input())
for i in range(0,m_P):
    val = input()
    phy = phy|{val}
m_C = int(input())
for i in range(0,m_C):
    val = input()
    che = che|{val}
```

```
m_CS = int(input())
for i in range(0,m_CS):
    val = input()
    cs = cs | {val}
failure = math|phy|che|cs
print(len(failure))
```