

9JABAZ

Want more books?

Visit 9jabaz.ng and download for free!!



OBÁFÈMÌ AWÓLÓWỌ UNIVERSITY
ILÉ-IFÈ, NIGERIA.

FACULTY OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INTRODUCTORY LECTURE NOTES

CSC 201: Computer Programming I

3 Units

Harmattan Semester, 2023-2024 Session

THIS DOCUMENT IS NOT FOR SALE !

ODÈJÒBÍ, Oḍètúnjí A.

Room 109, Computer Buildings,

oodejobi@oauife.edu.ng

November, 2024

Contents

1 Introduction to Computer programming	1
1.1 The mental activity of <i>İşirò</i> (Computing)	1
1.1.1 TASK 1	2
1.2 Agency of computing	2
1.2.1 Human language	4
1.2.2 Distinctions between Human and Programming languages	5
1.2.3 Language and computing process	6
1.2.4 The tools of language	8
1.2.5 Examples of Polarity and Logic expressed through structure	8
1.3 What is "A computer"?	9
1.4 History of Modern Computers	10
1.4.1 TASK 2	14
1.5 The Computer System	14
1.6 Computer hardware configuration	15
1.6.1 Input-Output devices	16
1.6.2 Central Processing Unit	16
1.6.3 Memory devices	17
1.6.4 Memory metric	18
1.7 Computer Software	20
1.7.1 System software	20
1.7.2 Application software	21
1.8 What is Computer Programming?	22
1.8.1 Operand: Data	23
1.8.2 Operation: Operator	23
1.9 Foundation of Computer Programming Language	24
1.9.1 Familiarly with Programming Language	25
1.9.2 Python Instruction format and structure	25
1.9.3 Python Identifiers	26
1.9.4 Python Reserved words	26
1.9.5 Python Variable and Constant identifiers	26
1.9.6 Python Statements	27
1.9.7 Familiarly with Programming Process	28
1.10 Algorithm	28
1.11 Computer program development process	29
1.11.1 TASK 3	31
1.12 Case example: FindAverage	32
1.13 TASK 4	35
1.14 TASK 5	36
1.15 Appendix A	38

List of Tables

1.1	Evolution of Computer technology	11
1.2	Computer memory metric	19
1.3	Python Reserved or Key words	27
1.4	Python Operators	27
1.5	Python code for the Flowchart in Figure 1.14	34
1.6	Python code for the Flowchart in Figure 1.15	35
1.7	Python code for flexible Average computation	35
1.8	Another Python code for flexible Average computation	36

List of Figures

1.1	Human communicating with Self (using L_0) and with Others using habitual language (L_H)	3
1.2	Human instructing machine using Programming language (L_p)	4
1.3	Human communicating through machine using Instrument of Language (L_I)	4
1.4	Information communicated with human language	5
1.5	Tools of human language	8
1.6	Definition of Computer	10
1.7	Ayo an example manual computing tool used in game playing	10
1.8	Plate of IBM 370 machine at the University of Ife Computer Centre	12
1.9	Evolution of Mobile computing and communication devices	13
1.10	Date stamp history of Computer programming language	14
1.11	Structure of a Computer	15
1.12	Fundamentals of Data and Instruction	23
1.13	Flowchart Symbols	30
1.14	Design for the Average Program	33
1.15	Design for the Average Program	34
1.16	Python Programming language operators	38
1.17	Precedence of operator in Python programming language	38

CSC201: LECTURE NOTE 1

Introduction to Computer programming

The Lectures in this aspect of the course is concerned with foundation and fundamental principles of **Computer Programming**. The abstraction and definitions of the terms (words) used in computer programming will be discussed. This is with the view to exposing students to language and process of modern computing. Familiarity with computing terms and how the terms are used to construct valid expression is essential to the effective understanding of this course.

We will also discuss the history of modern computers as well as the evolution of the languages used to program them.

The process of computer program development will also be explained and demonstrated with examples using the **Python** programming language.

1.1 The mental activity of *Īsirò* (Computing)

Èrò (Mental activity) is innate to individual human. *Èrò* (Mental activity) is conducted with **Human language**. Instances of state in human mental activity find expression through an **Instrument of language**. Humans inform each other by sharing state of their individual mental activity through an **Habitual Instrument of Human Language**.

The mental activity of computing is reduced into a process to facilitate its language expression. **A process comprises differentiable instances of state**. The *Àlàfo* (Interval) between a pair of state is called *ìsí* (a transition). Therefore, a process can be viewed as comprising instances of *state* and *transition*.

Process = state + transition

Modern humans have created special languages, such as Mathematics, for giving expression to their mental activity. Languages have also being created for instructing machines to carry out a formally prescribed process. The languages for instructing computing machines are called **Computer programming language**. A computer programming language is used to instruct the machine on:

- (i.) What action to perform (**Operation** or **Function**); (This corresponds to transition)
- (ii.) The identity of the items on which the action in (i.) above will be performed (**Data** or **Operand**) (This corresponds to state)

Therefore, a computing instruction comprises **Operation** and **Operand**.

Instruction = Operation and Operand

A **computing process** is executed by a **computing agency**.

1.1.1 TASK 1

Get and install Python environment into your MOBILE PHONE.
If you are using Android based Phone Install Python Android (Pydroid 3) from <i>Google Playstore</i> into your mobile phone or other mobile device.
If you are using Apple based Phone Install Python Coding IDE from <i>Apple Store</i> into your mobile phone or other mobile device.

1.2 Agency of computing

Ìsìrò (Computing) is a process (*Ìlànòṅ*). The *Yorùbá* word *Ìsìrò* (Computing) translates into English as (*Ì-ṣí-èrò*): “The act of giving expression to instances of state in human mental activity”. It also transcribes as: *Ìṣí+Èrò* (“The transition in mental activity”). *Ìlànòṅ* (A process) is the language rendering of the state and transition ascribed to human mental activity. Therefore, a process comprises Two (2) aspects: (i) State and (ii) Transition. The instances of state and transition in a process are expressed through an instrument of language. Therefore, a **process is the aspect of human mental activity prescribed and expressed using a language**.

Humans are biological agencies. **The computer is a machine**. A machine is a material agency. The computing machine is a tool for material rendering of an efficiently and precisely expressed process. The Computer programmer is a human or group of humans that instruct the computing machine on how to carry out a process. There are Two (2) fundamental differences between **biological** (e.g. humans) and **material** (e.g. computer machine) agencies:

- (i.) Faculty of language and
- (ii.) Sensory organs.

“Sensory organs” and “Faculty of language” are inherent in biological agencies. There is neither sensory organ nor faculty of language in a material agency. As depicted in Figure 1.1 an individual human communicates with himself/herself through his/her

own faculty of language (*Èdè-orí*). However, an individual can share his/her own mental state with other humans using an habitual instrument of language (*Ìpèdè*). Such language includes: (i) *Yorùbá*, (ii) *Hausa*, (iii) *Igbo*, (iv) *Tiv*, (v) English and so forth. Humans also use specialised language such as Mathematics to communicate amongst each other.

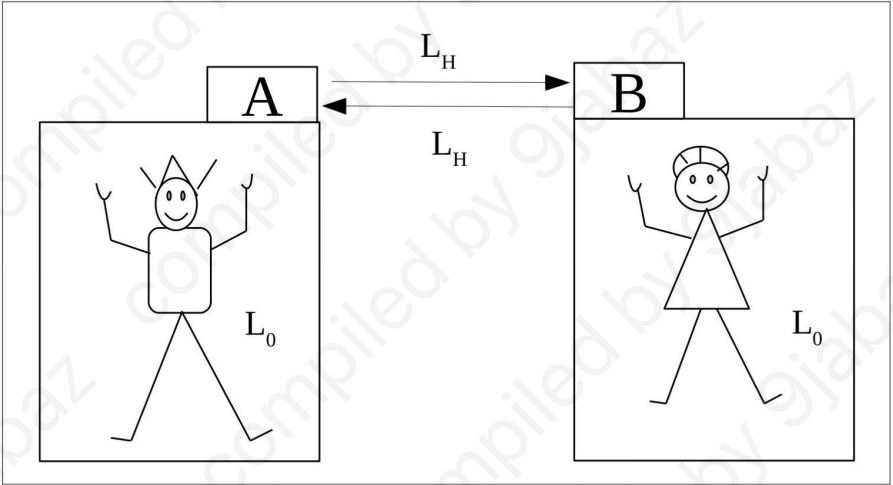


Figure 1.1: Human communicating with **Self** (using L_0) and with **Others** using habitual language (L_H)

A material agency (machine) is created by human as a tool for achieving a prescribed task or process. The purpose of material agency (machines) is determined by its human designer and/or users. Human also create specialised instrument of language to instruct a machine. As depicted in Figure 1.2, the programming language L_p is used by humans to instruct computing machines. The language L_I is used when human communicate with each other through a computing machine.

It is also the case that humans can communicate with each other by using a computing machine as depicted in Figure 1.3. This is called computer mediated communication.

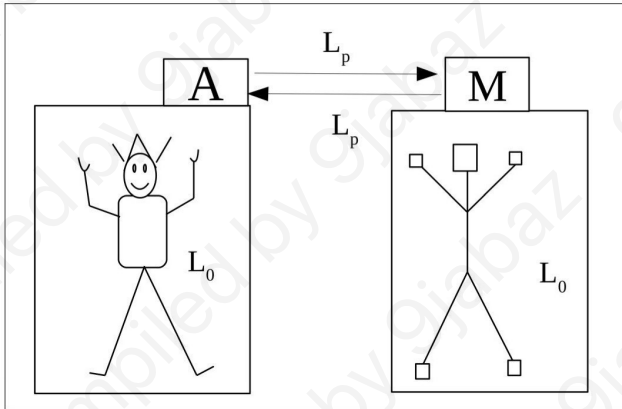


Figure 1.2: Human instructing machine using Programming language (L_p)

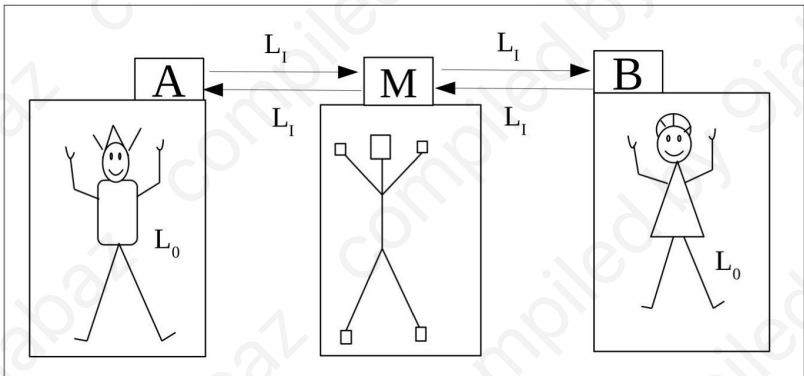


Figure 1.3: Human communicating through machine using Instrument of Language (L_i)

1.2.1 Human language

Humans use their habitual instruments of language to render, formulate and give material expression to instances of mental state.

An instrument of language is used to “give expression to” and/or “elicit *Àşamò* (Information) from”:

- (a.) *Òrò* (Speech)
- (b.) *Ìşẹ* (Action)
- (c.) *Àrokò* (Message)

Àşamò (A piece of information) can connote:

- (i.) An explanation.
- (ii.) A piece of advice,
- (iii.) A warning and
- (iv.) An instruction,

Àşamò (information) will be Àgan (ineffable) in the absence of an instrument of language. The information in a message is accessible to the habitual users of the language in which it is expressed.

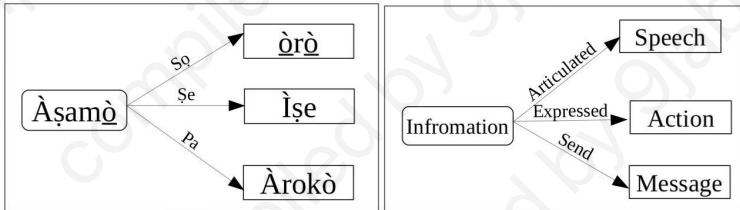


Figure 1.4: Information communicated with human language

For example, different messages will be elicited from the expression $Y = A + B$ by :

- (i.) A Mathematician,
- (ii.) A Physics,
- (iii.) A Genetics
- (iv.) In Computing .

1.2.2 Distinctions between Human and Programming languages

The operation of a Computing machine is to the extent of the instructions written by its human programmer. The instruction to achieve a problem-solving process is expressed using a “Computer programming language”. “Computer programming language” differs from human written language in the following aspects:

- (i.) “Computer programming languages are prescribed and specifically designed to communicate with machines. The expression of Computer instructions may NOT conform to the grammar of Witten human language, including that of Mathematical. For example, (a) $i = i + 1$, (b) $i ++$, and (c) $i = +$ are valid computer instruction. But they do NOT conform to the grammar of human written language.

- (ii.) Computer machines do NOT learn “A programming language” before they execute its instructions. Human must learn a language before they can use it to communicate or effect action.
- (iii.) Computer machines do NOT process the symbols in the instruction of “Programming languages”; they process a material rendering of the instruction. A computer manipulates prescribed signals rendered to the extent that technology permits. Such technology include Mechanical, Electrical (through Diode Valve and Transistors), Electro-magnetic, Electronic, Micro-electronic, and Nano-electronics. Symbols are directly manipulated by humans when processing an expression.
- (iv.) The capacity to determine whether there is a mistake in an instruction or program *TRANSCENDS* a Computer machine. However, a computer can be programmed to report errors in the invalid expression of an instruction.

In this course, therefore, you should NEVER confuse a material agency (machine), no matter its sophistication, with biological agency (human).

1.2.3 Language and computing process

Fundamentally, computing is human mental activity. The process ascribed to aspects of the human mental activity of computing is expressible through a **Language of symbols**. The process expressed through a language of symbols can further be reduced into a Computer Programming Language. The instructions in a programming language are used to influence the state and transition of computer hardware components. A computer program is a sequence of instructions that describe a process from its “Beginning to its “End.

“Programming Language are created by mimicking the *Written human language*. For example, the OPERAND (OR DATA) in computer instruction mimics the NOUN in written human language. Also, the OPERATOR in computer instruction mimics the VERB in written human language. Other aspects of the written human language mimicked in “Computer programming language” are discussed in Section 1.9. However, there are fundamental distinctions between the “Written human language” and “Computer programming language”.

However, you should NOT confuse the written human language for computer programming language. For example, only prescribed strings of symbols can be used to label the identity ascribed to the state (NOUN) and transition (VERB) in computer instruction. In written human language, a VERB is word ascribed to the motion (action) of an instance or agency. The metaphor of VERB in computer programming language is the **operator** (e.g. Addition, Subtraction, Multiplication) used to manipulate data. The manipulation of data results in the transition of computer states. Whereas **Verb** and **Noun** are **part-of-speech** in the grammar of written language, **Operator** and **Operand** are strings of symbols in the **syntax** of a computer programming language.

For example, give the following formulation:

Item 1 HUMAN EXPRESSION = ACTOR + ACTION

Item 2 WRITTEN HUMAN LANGUAGE SENTENCE = NOUN + VERB

Item 3 COMPUTER LANGUAGE INSTRUCTION = OPERAND + OPERATOR

Item 1 above is accessible through human sensory experience alone. **An Action and its Actor are inextricably intertwined and inseparable.** Special training is not required for an habitual users of a language to elicit the message in its expression.

Item 2 is located in a prescribed instrument of language. In that prescription, an Action is identified as a Verb and its Actor is identified as a Noun. **A Noun and its Verb are separated and individually expressed in a sentence.** Special training is required to elicit the message in a written human language expression.

Item 3 is comprises strings of symbols ascribed to the Manipulation (Operation) of Operand (Data) in a computer register or memory. The instruction in the message is targeted at a material agency (machine) NOT humans.

In this course, therefore, you should NEVER confuse Computer Programming Language (E.g. Python, Java, Fortran), no matter its sophistication, for Human Written Language (E.g. Yorùbá, Hausa, Igbo, English).

Two or more humans can communicate with each other to the extent of their familiarity with an habitual human instrument of language. Familiarity with human language is acquired through training and interaction with the habitual users of the language.

Familiarity with programming language is acquired through a careful study of:

- (i.) Its Alphabet (Admissible set of symbol)
- (ii.) Construction of valid terms using (i.)
- (iii.) Construction of valid expression using (ii.)
- (iv.) Construction and combination of sequence of instruction using (ii.) and (iii)
- (v.) Prescribed standard, convention and process for using (iv) into to write the instructions for a program
- (vi.) Constant practises with computer programming problem-solving activities.

The more programs you write in a Programming Language, the more your familiarity and competency with its use in computing problem-solving activities.

1.2.4 The tools of language

There are there (3) tools in human language. These are:

- (a.) *Ìhun* (Structure): It is used to to hold the terms that are ascribe and/or assign to the identities of instances of state and transition. Structure is the medium through which terms (terminal) in an instruction is expressed. A structure and the terms are used to bring intangible (mental) instance within the ambit of human sensory organs (E.g. eyes, finger, hear, etc). Indeed, the terms of an expression are in the Structure of its instrument of language.
- (b.) *Ìtòka* (Polarity): It is used to locate and/or arrange the terms into an expression. The individual terms in an expression can be pointed at by virtue of the language tool of Polarity. Indeed, the location occupied by each term is by virtue of the polarity ascribed to it in the structure of its expression.
- (c.) *Àròtò* (Logic): It is used to formulate the recursion ascribed to a temporal (mental) instance of state. Logic can neither be seen nor pointed at in the structure of an expression. Indeed, logic is everywhere present and nowhere located in an expression.



Figure 1.5: Tools of human language

Polarity and Logic manifests through the terms in a structure. In the absences of structure, polarity and logic cannot be expressed. Whereas polarity is used to arrange terms into a structure, logic is used to ascribe mental state to the expression in the structure. Mental state is situated in its human agency.

1.2.5 Examples of Polarity and Logic expressed through structure

If $Y = 3$ and $X = 4$

In Polarity operation

(i.) $Y + Y = 33$

(ii.) $X + X = 44$

(iii.) $Y + X = 34$

(iv.) $X + Y = 43$

(v.) $Y^2 - 4 = ?$

Note that NO New Symbols is introduced in the Polarity operation. Every term in a polarity expression can be pointed at in its structure. Operations are realised through “Concatenation” (*Āsnp̄*) or “Extraction” (*Īfāy*) only. Note that NO New Symbol is produce outside of the ones in the input in Polarity operation. Only the symbols in the input appear in the output. The polarity of every terms is within the ambit of sensory experience. Logic is not required for processing polarity expression.

In Logical operation

If $Y = 3$ and $X = 4$

(i.) $Y + Y = 6$

(ii.) $X + X = 8$

(iii.) $Y + X = 7$

(iv.) $X + Y = 7$

(v.) $Y^2 - 4 = 5$

Note that the symbols that are NOT in the input appear in the output of logical operation. Therefore, the logical operation is NOT in the terms. The operation is in the logic formulated into the terms. The logical content of a term cannot be pointed at in its structure. The logical contents of terms are intangible instances in human mental state.

1.3 What is “A computer”?

Computer is a **device** with the capacity to **accept data** and **instructions**, **process** the data using the instruction and **produce output** in the format specified in the instruction.

Based on the above definition, the computer can be viewed as depicted in Figure 1.6.

A computing machine is, therefore, expected to have component for:

1. Accepting data and instruction.
2. *Processing* data following the instruction.



Figure 1.6: Definition of Computer

3. Produce the *output* of the processed data according to the instruction.

A material agency (machine) or tool that has the capacity to carry out the tasks listed above is a computer. Indeed modern computers have the capacity listed above, but they are unlike other machines. What sets the modern computer apart from all the machines before it is **Language**. Humans have created a language with which to give instruction to modern computers. The language is created by imitating written human languages. The language is called **Computer Programming Language**.

1.4 History of Modern Computers

The history computer evolves with technology. In the **Manual era**, material objects are physically manipulated by humans to assist or support *ìşirò* (a computing) process. Fingers, Stones, pebbles, animal bones, pieces of wood and bamboo, etc. are used to support memory and organisation during a computing process. Specialised tools such as the Abacus, Slide-rule, *Ọpón-Ayò* (Ayo game board See Figure 1.7), *Ìlẹ̀kẹ̀ Òhà* (Counting beads), *Okùn Àdìkà* (Counting ropes) and *Apásá* (Loom for cloth weaving) were developed and used during the manual era. The character of the manual era is that, humans physically and directly manipulate the materials used in the computing process. The main problem with the manual era is that its computation process is labour intensive and therefore, prone to error on the grounds of human mistakes. nonetheless, creativity is most effectively expressed through manual process.

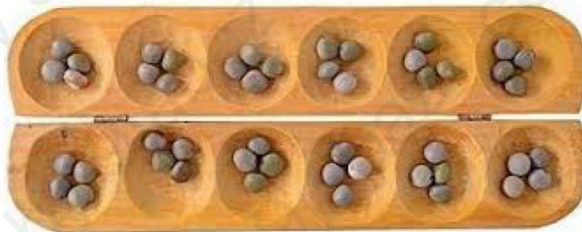


Figure 1.7: Ayo an example manual computing tool used in game playing

Table 1.1: Evolution of Computer technology

Ser. No.	Technology Era	Strength	Limitation
1.	Manual tool (Stone, Pebbles, Stick)	Effective for expressing creativity.	Labour intensive and prone to errors and mistakes
2.	Mechanical (Metal, skin and wood)	Reduced the amount of manual labour	Frequent faults and machine breakdown
3.	Vacuum and Cathode-ray Tube	Machine able to work automatically. Fast operation. Machine level programming through valves and switches is possible.	Frequent fault and very high maintenance cost.
4.	Diode devices	Occupy less space. Low level programming through symbols is possible.	High Maintenance cost
5.	Transistor	Less energy use. Reduced size.	Difficult High level programming
6.	Integrated circuit	Effective high level Programming.	Low interconnectivity
7.	Micro electronics	High interconnectivity and cheaper	Low versatility and communication speed
8.	Nano electronic	Seamless interaction and versatile applications	Frequent evolution

The manual era is followed by the mechanical era. During this era, metals such as iron, copper and silver were refined and used to fabricate devices that automatically give expression to the state and transition in a computing process. This is the beginning of the **Machine era**. A machine (*ἔργον*) is a material agency that is capable of working “by itself” (that is autonomously). Example of machines in the **Mechanical Era** include, (i) Pascal calculator, (ii) Leibniz calculator and (iii) Charles Babbage’s Difference Engine. The main problem with the Mechanical era is that its computation process is unreliable as machines broke down frequently. This makes the output of computations prone to error. The Mechanical era is followed by the **Vacuum and Cathode-ray Tube era**. The **Vacuum and Cathode-ray Tube** technology makes it possible to store and manipulate the electrical signals used to represent instances in the state of a computing process. This technology culminated in faster and more accurate material realisation of the computing process. Example of machines in this era include Harvard Mark I and IBM SSEC machines. This **Vacuum and Cathode-ray Tube era** is followed by the **Diode device era**. The major problem of the **Vacuum and Cathode-ray Tube** is the heat generated during computing process. Computer memory devices are realised using mercury delay. The tube and mercury technology consumes substantial energy as well. This problem was reduced by the **Diode device era**. The **Diode** technology makes it possible to create cooler and more accurate computing machine. Example of machine in the **Diode device era** include ENIAC and EDVAC.

The **Diode device era** is followed by **Transistor Era**. During the **Transistor Era**, circuits that implement basic logical operation such as AND, OR and NOT were fabricated as a single ship. This way, the computing machine during this era consumes less electricity. Magnetic cores and index register were incorporated into computing machines. Example of machines in this era include: Honeywell 800, UNIVAC, IBM 7000 Series.

Transistor Era is followed by the **Integrated Circuit Era**. During this era, electrical resistors, capacitors and transistors, are integrated as a silicon ship. The silicon ships are used to manufacture computing machine components. The microprocessor emerges during this era. The microprocessor is a device capable of automatically carrying out arithmetic and logical operation. This makes it possible to manufacture Calculator-on-a-chip devices. Example of machines in this era include Honeywell 6000 and the IBM 360/370 series. The disused chassis of IBM 370 machine used at the University of Ife Computer Centre is depicted in Figure 1.8.



Figure 1.8: Plate of IBM 370 machine at the University of Ife Computer Centre

From about 1975 onwards, Large Scale Integrated Circuit have emerged with more advancement in technology. The number of electronic components and part that can be integrated into a single ship has increased tremendously. The modern **Nano-technology era** makes it possible to put millions of circuit into a material ship that is about the size of human index-finger nail.

As a summary, you should note the following in respect of computer technology:

1. The hardware of computer machines evolved with technology;

2. The performance of computing machines improve with advancement in technology;
3. The cost of computing machines reduce with advancement in technology;
4. The size of computing machines reduce with advancement in technology;
5. The reliability of computing machines increase with advancement in technology;
6. The function and versatility of computing machines increase with advancement in technology;



Figure 1.9: Evolution of Mobile computing and communication devices

Figure 1.9 (Top) depicts the evolution of Laptop computing machines within a 25 year period. It also depicts (Bottom) the evolution of mobile hand-held devices within a 40 year period. Figure 1.10 is a date-stamp listing of the evolution of Computer programming languages between the year 1950 and 2010. Despite the evolution of computer technology, however, **the logic of computing process remains unchanged**. This implies that **the logic of addition, multiplication, division, searching, sorting, and so forth, remain unchanged**. This is so even when better technique have been advanced to achieve computing operation. **Therefore, a consistent definition of computing machines should NOT be grounded in technology.**

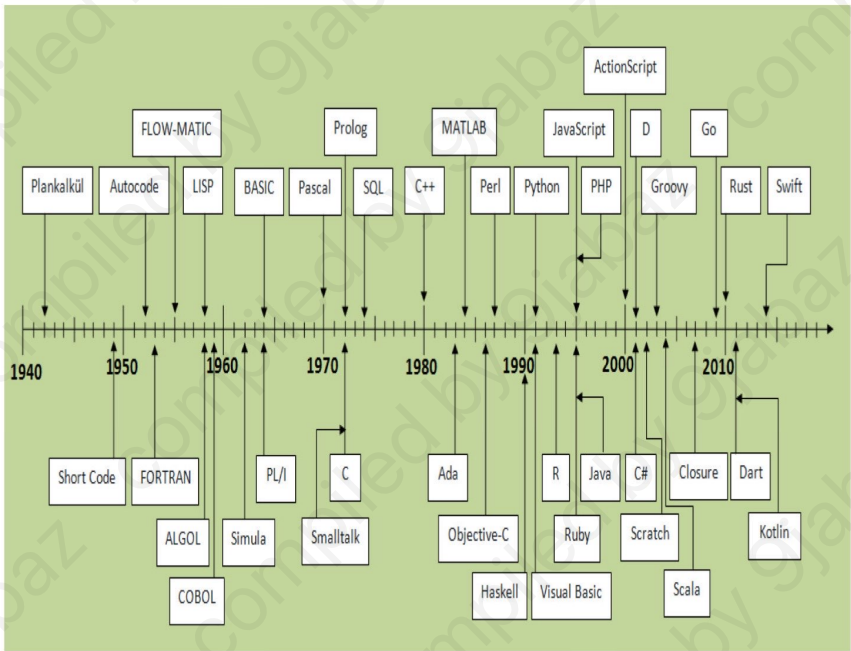


Figure 1.10: Date stamp history of Computer programming language

1.4.1 TASK 2

You are expected to read more about the history of computer in standard textbooks and on the internet. You may be required to submit an assignment on this.

1.5 The Computer System

The computer system is composed of Tangible (physical) and Intangible (non-physical) aspects. The physical aspect are all the thing that you can physically touch, feel and move. The Physical aspect of a computer are collectively called the **Computer Hardware**. Examples of computer hardware include: the mouse, monitor or screen, keyboard, input-output devices such as printer and scanner as well as memory devices such as your Flash Disk.

The Non-physical aspect are the intangible state and activity of the computer. You cannot see or touch this aspect but you can know that they are in operation through the activities of the physical aspect. The intangible aspects of the computer are collectively called **Computer Software**. Examples of computer software are: **Operating System**

Software such as Microsoft Windows, UNIX, Android; **Application Program Software** such as Microsoft Excel, Power-point as well as the Apps running on your mobile phones.

If we use human as a metaphor or analogy, the physical body, i.e. head, legs, hands, eyes, etc. are the hardware. Human mental activity are the software. Though you cannot see human mental activity, but you could see the physical activities through his/her action. However, human physical activities are the manifestation of human mental activity. Hence, human mental and physical aspects work in harmony in human activities. Similarly, the hardware and software must work in harmony before the computer can function properly.

The next section introduces you to the hardware components of the computer. The components discussed here are necessary for understanding how the computer program works. There are other hardware components of the computer which you do not really need to know much about before you can write a program. For example, the computer motherboard and power supply unit.

1.6 Computer hardware configuration

The structure of computer hardware is depicted in Figure 1.11. The structure comprises parts that are connected and configured to work together during the computer operation. In the figure, the *dotted arrow head line* is control signal and the *full arrow lines* are the data lines.

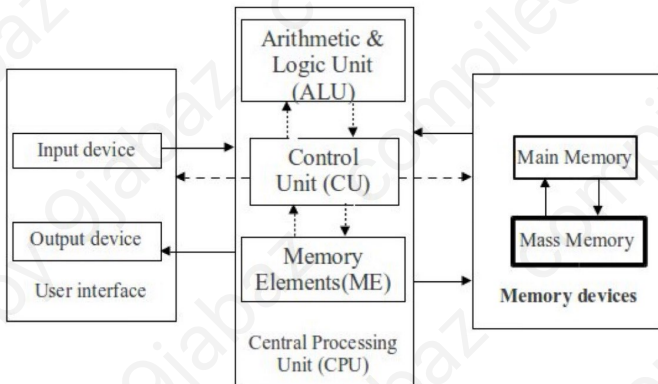


Figure 1.11: Structure of a Computer

Hardware are physical devices constituting the parts that facilitates a computer program execution.

1.6.1 Input-Output devices

The input component is also called the input device. Examples include Keyboard, mouse, touch screen, pen and so on. The ability to use the **Keyboard** and **Mouse** is central to computer programming.

1.6.2 Central Processing Unit

The Processing component comprises a number of other components that facilitate the execution of a programme. The Computer Process Unit comprises three (3) other subcomponents:

Memory Element (ME)

This is the place where the computer stores results of ongoing operation. The memory element are inside the processor. They are also called **register**. Each of the register inside a process serves a specified purpose. An example is the **Accumulator Register**, which stores the results of an ongoing arithmetic operation. Another example is the processor **Status Register**, which stores the status of an ongoing operation, such as when “an overflow” or “negative” result is generated. The Memory element can be likened to a scratch pad that you use while solving mathematics problem: a place where rough work is written. The content of the Memory Element or register is lost immediately power is switched off from the computer. This is why the Memory Element of register is called a **Volatile Memory**.

Control Unit (CU)

The Control Unit determines the operation of all the other hardware component of the computer. However, the activity of the Control Unit is determined by the **Computer Software**. The Control Unit (CU) will, for example, determine when the input device must read data, when the output device must write output and when data and instruction are transmitted between devices. The CU reads program instructions from the computer memory and interprets (decodes) the instructions. It then uses the interpreted instruction to determine what is to be done. To achieve the tasks specified in the instructions, the CU generates a series of control signals to the other parts of the computer hardware. When everything seems to be working well but nothing is working, the control unit is probably faulty.

Control units in advanced computers may change the order of some instructions so as to improve performance.

Arithmetic and logic unit (ALU)

The ALU carries out arithmetic and logical operation. Its arithmetic operations include: Addition (+) and Multiplication (\times). Its logical operations are those that evaluate to True or False. Examples includes Greater Than (>), Less than (<). Most computer instructions are realised through these simple operations. It turns out that by performing these simple operations, a computer can be programmed to achieved complex tasks. This is done by breaking the complex tasks into simple steps that are realised through the operation of the Arithmetic and Logical Unit. Therefore, modern computers can be programmed to perform a well-defined task; although it will take more time to do so if its ALU does not directly support the operations in the task. An ALU may also compare numbers and return boolean (true or false) value depending on whether one is equal to, greater than or less than the other (“is 64 greater than 65?”). Logic operations also include boolean logic: AND, OR, XOR and NOT. These can be useful both for creating complicated conditional statements processing boolean logic. Modern computers (e.g. Superscalar machines) contain multiple ALUs so that they can process several instructions at the same time. Graphics processors and computers with SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data) features often provide ALUs that can perform arithmetic on vectors and matrices.

Memory elements (ME), Arithmetic and Logical Unit (ALU) and Control Unit (CU) are together called the **Central Processing Unit (CPU)** of the computer.

$$\text{CPU} = \text{ME} + \text{ALU} + \text{CU}$$

The computer’s capacity to carry out small to large scale processes, its CPU must be supported by **Memory Devices**.

1.6.3 Memory devices

Memory Devices are the hardware components of a computer in which data and instruction are stored.

There are two categories of memory devices: (i) Main memory and (ii) Mass memory. These are discussed in the following subsection.

Main memory devices

The Main memory is the memory device in which the computer stores the **data and instruction** in the currently running program. The computer store temporary result of operation in this memory. When a program is executing, its data and instructions are store in the Main memory. The main memory is a Random Access Memory (RAM). This implies that the computer can access data in the memory at any location of the memory without delay and in any order. Main memory is usually smaller in size than

the Mass memory. It is also more expensive than the mass memory. A major weakness of the Main memory is that when electric power is lost, its contents will be lost as well. This is why it is called **Volatile** memory. The Main memory is also called **Primary memory** or **Core memory**. The data and instruction that we wish to retrieve latter are stored in the Mass memory. This is discussed as follows.

Mass memory devices

The Mass memory is the memory device in which the computer stores data and instruction that can be retrieved latter. When a program is NOT executing, its data and instructions are store in the Mass memory. The mass memory is a Sequential Access Memory (SAM). This implies that the computer can access data in the memory in the sequence or order they are store in the memory. Mass memory is usually much larger in size than the Main memory. The Mass memory is also **cheaper** than the Main memory. A major strength of the Mass memory is that when electric power is lost, its contents are retained. This is why it is called **Non-Volatile memory**. A major weakness of Mass memory is that it is slower than the Main memory. The Mass memory is also called **Secondary memory** or **Peripheral memory**. There are other memory devices, such as the Read only Memory (ROM) and Programmable Read only Memory (PROM). In these memory devices, specialised data and instructions are stored. For example, the data and instruction that the computer requires for its intrinsic operations are kept in these specialised memory devices. Examples of such specialise data and instruction are those that the computer system use during “start-up” and “Booting”. The computer timing data is also stored in Specialised memory devices. This class of memory are usually of interest to “Embedded” or “Real-time” system programming.

1.6.4 Memory metric

Computer memory devices are of various sizes. The different sizes of computer memory are reckoned using a metric based on the number of cells that makes up a unit of data. The Bit (**Binary Digit**) is the smallest data element in modern digital computing machines. The Bit is stored in a **Cell**. Therefore, a cell holds a **Binary digit (Bit)** during computer operation. Each cell holds either Zero $\boxed{0}$ or One $\boxed{1}$ **at an instance in the computer operation**. Four (4) cells, each of them holding a bit, e.g. $\boxed{0}\boxed{1}\boxed{0}\boxed{1}$, makes a **Nibble**. A **unit of data** in modern computer is a **Byte**. A computer **Byte** comprises Two (2) Nibbles or Eight (8) Bits, e.g. $\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}$. The number of unit data comprising the entire storage space in a memory device is reckoned in Bytes as depicted and described in Table 1.2. A Byte can hold one character, E.g. the letter ‘R’, the digit ‘5’, etc. Memory metric is, therefore, a rough estimate of the number of characters that can be stored into a memory device. If the number of character in **your notebook** is up to **Ten thousand** (10, 000), then you will need a memory of about Ten Kilobyte (10KB) to store it. This is estimated as 10×2^{10} Bytes, where 2^{10} Bytes

corresponds to about One thousand (1,000) characters.

Table 1.2: Computer memory metric

Ser. No.	Name	Description
1.	Bit	A status of the computer cell. It can be Zero (0) or One(1); $\boxed{0}$ and $\boxed{1}$
2.	Nibble	Four bits. $\boxed{0}\boxed{1}\boxed{0}\boxed{1}$
3.	Byte	Eight(8) Bits or Two (2) Nibble $\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}$
4.	Word	The number of bytes that a machine can process at one instance of its operation. It is the number of lines in the data bus of the computer.
5.	Kilobyte (KB)	A Kilobyte is computed as 2^{10} (1024) Bytes.
6.	Megabyte (MB)	One Megabyte is computed as 2^{20} (1,048,576) Bytes.
7.	Gigabyte (GB)	A Gigabyte is a computed as 2^{30} (1,073,741,824) Bytes.
8.	Terabyte (TB)	A Terabyte is computed as 2^{40} (1,099,511,627,776) Bytes.
9.	Petabyte (PB)	A Petabyte is computer as 2^{50} (1,125,899,906,842,624) Bytes.
10.	Exabyte (EB)	An Exabyte is computed 2^{60} (1,152,921,504,606,846,976) Bytes.

1.7 Computer Software

The intangible aspect of the computer system are called “Computer Software”. Computer Software often comprises several “Computer Programs”. If a program, or set of programs, fails, the computer hardware may not function properly. The human who writes program for controlling a computer is called a **Computer Programmer**. The humans or group of humans involved in the Specification, Design, Programming (Coding) and Evaluation (Testing) of software are also called **Software Developer**. Every programming for problem-solving involves familiarity with every aspects of software development. The job of a computer programmer (software developer) is to **design and implement a set of programs that will make the computer to do what a user wants**. Usually the user is a human being. The user can also be another computer. For the programmer to achieve this task he/she must first understand the problem that the user wants to use the computer to solve. The programmer must also understand the capability of the computer as well as how to instruct the computer to perform the specified task. In this course you will **learn how to instruct a computer using the Python programming language**.

A **software** comprises a set of programs that work together during the execution of a process.

Computer software are of Two (2) categories:

- (i) System software.
- (ii) Application software

These are explained in the following Subsection.

1.7.1 System software

System software is specifically developed by computer manufactures to make the computer ready for use by other programs and users.

System software is a set of programs that are used to create the environment for other software to run efficiently on the hardware of a computing machine.

System software performs the following tasks:

1. Create the environment for other programs to run. [**Run** is the terminology used for “a process executing on a computing machine”]
2. Manage the computer resources (disks, memory, printers, etc.) for effective use. E.g, Data Zipping (Compress data into smaller memory size)

3. Provide data about the status of the hardware and related computer component [E.g. time].
4. Provide prompts (message) that can help the user in computer usage.
5. Provide interfaces that suit users fancy and appeal;
6. Respire or reclaim unused computer resources such as memory. System Software that reclaims memory are call “**Garbage Collector**”.

A very important **System program** is the *Operating system*. Examples of operating systems include:

1. UNIX
2. LINUX
3. Ubuntu
4. Solaris
5. Mac OS
6. Windows XP
7. Android OS

Other examples of system programs includes: Editor, Linker, Loader. System programs are usually written by **System programmers**. System programs are commissioned by specialised Computer vendors, organisation or Business interests. Aspects of system programs are usually written using **Low Level Language** called **Assembly Language**.

1.7.2 Application software

An **Application Package** (Apps) is a collection or suit of software intended for solving problems in a particular user environment. Each of the set of programs in an Application software is for achieving specific user task. For example, a set of program can be written for assisting with all the work done in the office. The work may include document typing and processing also called word processing. Keeping a “table of inventory” also called spreadsheet. Other program may be written for “drawing of graphics” and/or “Architectural Design”. Some others may also be written for music compilation and processing. An example of **Application Package** is Microsoft Office. This package comprises (i) MS Word (for word processing) (ii) Excel (For spreadsheet) (iii) MS FrontPage (For graphic design) (iv) MS Publisher (For publishing works), (v) Ms PowerPoint (For making presentation slides), etc. Another examples is **OpenOffice**. It comprises similar applications like Microsoft Office. Other examples of Application packages include:

1. In Mathematics, Science and Engineering [Octave, Matlab, Mathematical and Maple]
2. In Internet browsing [Google Chrome, Safari, Firefox]
3. In Graphics design and Architecture [AutoCard, Adobe Photoshop, CorelDraw]
4. Online meeting [Skype, Hangouts, Google Meet, Zoom, and Whatsapp]
5. Banking [Oracle, TEMENOS, CorePlus, Bankware]
6. There are many Application programs (Apps) on modern mobile devices.

As you are already aware, there are Application programs (Apps) on your Mobile Phone for doing all sort of tasks from banking, to playing games, searching for location, and so forth. These Apps cannot run except there is a **System program** e.g Android, already running on the hardware of your Computer or Phone. Application programs are written by **Application programmers**. Substantial parts of **Application programmers** are written using **High Level Programming languages** such as Python, Java, C++, Pascal, FORTRAN.

1.8 What is Computer Programming?

Computers will do what humans instruct. The instructions for computers are written in a Programming Language. *Computer will NOT do what they are NOT programmed to do.* To ask a computer to perform a task, you have to give it Two (2) things:

1. The **data** on which the computer will perform the task.
2. The **set of instruction** on how to manipulate the data.

After applying this instruction to the data the computer produces an output. This means that all what you want the computer to do must be reduced into **data and instructions**. While following the instruction, the computer will be manipulating the data. If we adequately describe our instruction and accurately specify our data, then the output that the computer generates will be what we want.

The art of composing the sequence of instructions for controlling computer system operation **is called programming**.

The outcome of a computer programming activity is a *program*.

A program is a *sequence of instructions* which when executed by a computer, the output corresponding to an input data will be produced.

An instruction is a unit of message or command that influences the operation of a computer. An instruction is composed of Two (2) parts:

1. Operand (Data)
2. Operation (Operator)

INSTRUCTION = OPERAND + OPERATION

INSTRUCTION = DATA + OPERATOR

An analysis of the instruction:

ADD Two (2) data items identified with the names X and Y and Store the output in Z

is as depicted in Figure 1.12.

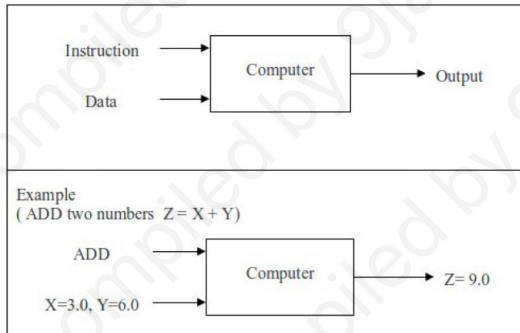


Figure 1.12: Fundamentals of Data and Instruction

1.8.1 Operand: Data

Names are used to identify the entity or item or element that the computer will manipulate during a process. Individual item or entity is a **datum**. Instances of datum are called Data (The plural of DATUM is DATA).

A name or identifier that can contain only one value when a program is running is called **Constant name**. An example of Constant name definition is $Pi = 3.147$. A name or identifier that can contain different datum when a program is running is called **Variable name**. An example of Variable name definition is $Radius = 7.5$. A constant or variable name can be occupied by different datum or data include: Numerical (Number), Literal (Text) or Logical (True/False).

Several data item can also be identified with One name. This is done in the definition of: (i) List, (ii) Array, (iii) Dictionary and (iv) Tuple

1.8.2 Operation: Operator

Operators prescribes the manipulation that will be performed on operands (Data, datum variable). This include **Arithmetic operation** on constants such as Addition and Multiplication; **Algebraic operation** on variables such as Subtraction and Division. It also

includes; **Logical operation** Negation and Inversion; **Input-Output** operation such as *input* and *print*.

1.9 Foundation of Computer Programming Language

The process of programming can be viewed as the process involved when two (2) human individuals communicate through **Writing Letter**. The two (2) individual are communicating using the written form of their habitual instrument of language. Their habitual instrument of language is **Human Language**. In this case, however, the Programmer is communicating with a machine using a computer **Programming Language**. Indeed the *Program writing* process is grounded in the metaphor of *Letter writing*.

These are the attributes of the **Letter Writing** process

1. The message in a letter is written in a Human Language { e.g. Yorùbá, Igbo, Hausa, English, etc. }.
2. The symbols in the message are drawn from the alphabet of the Human Language.
3. There is a format of presentation to which the text (narrative) in the letter conforms {e.g. formal and informal letter }.
4. There is an opening (beginning) and a closing (ending) statement.
5. There are sequence of words, phrases, sentences and paragraphs between the Opening and Closing statement. Each of the words, phrases, sentences and paragraphs expresses aspects of the message in the letter. Each word, phrase, sentence and/or paragraph is composed using the symbols drawn from the alphabet.
6. The message in the letter conveys information from the Writer (Sender) to the Reader(Receiver).

Note that a letter is written by humans and the instruction or message in it will be read by another human being.

In the same manner a computer program is written by human being (a Programmer) but the primary target or audience of the instructions or messages a machine (the computer). A program:

1. A computer instruction is written in a Programming Language { e.g. Python, FORTAN, JAVA, C++, VisualBasic, etc. }.
2. The symbols for composing an instruction are drawn from the alphabet of the computer language.
3. There is format to which admissible instruction must conform {e.g. this is called the Syntax or Format of instruction }.

4. A program has an opening and closing statement
5. There are sequence of instruction, functions and sub-programs between the Opening and Closing statements.
6. The program conveys the writer (programmer's) instructions to the computer (receiver).

The following subsections, and indeed, other aspects of this course will explicate the above analogy.

To compose an effective computer program, it is important to be familiar with the **Programming Language** and **Programming Process**. Familiarly with **Programming Language** includes:

1.9.1 Familiarly with Programming Language

Familiarly with a computer **Programming Language** involves:

1. Identifying the assignment and use of symbols in alphabet of the programming language. This include special or reserved symbols and strings.
2. Understand how to compose the string of symbols for naming data or operand.
3. Understand how to use *data* and *operators* to compose valid instructions.
4. Understand the admissible use of instruction individually and collectively.

1.9.2 Python Instruction format and structure

During this course, the format and structure of the instruction for following Python statements and expressions will be explained with examples. You will also be introduced to other formats and structures that are used in the Python Object Oriented Programming Language. In addition, you will learn the techniques for using the format and instruction in programming. Some of the instruction and statement formats and structures that would be consider include:

1. Data formatting.
2. Input and output statements.
3. Decision, Control and Conditional Statements.
4. Loops and Looping.
5. Array data structures.
6. Routings and Functions: Modular Programming (Breaking a complex programming problem into Function of smaller problems).

7. File processing.
8. Object Oriented Programming (Definition and use of **Class of Object**)

NOTE THAT

- (a.) Python is CASE sensitive. Therefore *Ade* and *ade* are NOT Equal.
- (b.) Python is space sensitive. Space must be used as prescribed by the syntax of an instruction for example in the: *(i) if (ii) for* and *(iii) while* instruction.

1.9.3 Python Identifiers

The variables in a Python program can be likened to the nominal part-of-speech in human written language. The nominal part-of-speech is a NOUN. The “Noun” is a class of strings, each of which is used to identify an instance or agency. Variable can hold one constant at an instance during a computer operation. Therefore, the content of a variable can change during the execution of a process. The constants in a Python program can be likened to the words in human written language. The “word” is a string of symbols used to identify a datum, data, or agency (i.e. Function or Object). Constants do NOT change during the execution of a computer program. An example of a constant identification is $Pi = 22/7$ or $Pi = 3.147$. When the value to be stored changes then it is given a variable name, for example *Sum* or *Average*. Variable names correspond to the identity of a memory location in a computer system. In the case of a constant, the exact value of the constant, e.g. “Ade”, “1234”, and “Lagos” is stated. In the expression `NAME = “Ade”`, the string NAME is a variable while the string “Ade” is “an instance of constant” that can occupy its content. Also in the expression `Y = 10`, the identifier Y is a variable while the number 10 is a constant. The number 10 can also be called a constant string of Two (2) digits, that is 1 and 0.

1.9.4 Python Reserved words

There are certain names that have special function in the Python programming language. These names are called **Reserved words** or **Key words**. **Reserved words** CANNOT be used to name a variable or constant. Thirty-Three (33) Python Reserved words are listed in Table 1.3. Selected Python operators are listed in Table 1.4 and details of their uses and precedence is provided in the Appendix A in Section 1.15.

1.9.5 Python Variable and Constant identifiers

Name is used to ascribe identity to an instance. The name of a constant or variable is composed through that concatenation of letters and digits. The composition is similar to how words are composed from written language alphabet. A string corresponding to a valid Python name is constructed using the following set of rules:

Table 1.3: Python **Reserved** or **Key** words

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

Table 1.4: Python **Operators**

+	*	-	%	**	//	<<
Add	Multiply	Subtract	Remainder	Power	Floor Division	Shift left
>>	&		<=	>	>=	<>
Shift right	Bitwise ADD	Bitwise OR	Less or Equal	Greater	Greater or equal	Not Equal
!=	/					==
Logical not Equal	Divide					Equal logical

1. A variable or constant name can only be composed using the Python **set of symbols**. This includes **upper case letters** {A, B, C, ..Z}, **lower case letters** {a, b, c, ..z} and **digits** {0,1,2,3,4,5,6,7,8,9}.
2. Variable or constant names are case-sensitive. This implies that Number, number and NUMBER are three (3) different names.
3. A variable or constant name CANNOT be any of the Python keywords.
4. A name must contain not more than Thirty-one (31) characters. This implies that the *length* of a name should not be more than 31 characters.
5. A number should not start a name. For example, the variable *5name* is wrong but *Name5* is correct.
6. A special character (except the underscore, `_`), should not be used in forming a name. For example, the variable *Fn*ame* is wrong but the name *Fn_ame* is correct.

1.9.6 Python Statements

A Python statement is an instruction, which when executed, will influence the computer operation. A statement is similar to a sentence in human languages. Each of the statement in a program is also called a **code**. Various Python statement and how they are used are discussed in your Laboratory and Tutorial Manual. Nonetheless, note that Python program state is SPACING SENSITIVE. This implies that SPACE can influence how a sequence of instructions are Executed.

1.9.7 Familiarly with Programming Process

Familiarly with **Programming Process** includes:

- (i) How to design the **algorithm** of a programming process
- (ii) How to reduce the algorithm into program code or instruction.
- (iii) How to create the digital version (the computer text) for the instruction composed in (ii)
- (iv) How to compile the digital version in (iii) into computer native language.
- (v) How to identify and remove errors and mistakes from your code.
- (vi) How to document your code for future use and improvement.

Familiarly with **Programming Process** is briefly discussed in the following sections.

1.10 Algorithm

The origin of the term *Algorithm* was traced to a 9th century Arabic scholar called *Abu Ja'far Muhammed Ibn Musa Al-Khowarizm*. An algorithm is a sequence of formal instruction describing the solution to a computing problem. An algorithm describes the solution one problem only. This implies that when an algorithm is written for a problem it cannot be used to solve a different problem.

The following are the features of an algorithm:

1. **[Terminal:]** There is exactly one ENTRY (START) and one EXIT (END) points in the algorithm. These are called the terminals of the algorithm.
2. **[Finite process:]** The steps in the algorithm must be finite. The number of steps from when the input is applied and an output is generated must be totally countable. A step corresponds to an instruction.
3. **[Effective:]** The algorithm must produce the total output corresponding to the input data.
4. **[Soundness:]** The algorithm must produce the same output for an input no matter the number of times the input is applied.
5. **[Practical:]** An algorithm must be executable on a computer system.
6. **[Halt:]** It is desirable that the program implemented using an algorithm releases all computer resources (memory, register, data bus, processor) immediately after its termination.

The instruction in an algorithm and how they are structured follows a standard set of rules. An algorithm intended for a program is design before it is reduced into codes (set of instruction) and implemented on a computer. Algorithm designs are primarily intended for use among programmers during the programming process. An algorithm is, therefore, an important aspect of computer programming just as the Plan is important in Building a house.

There are two (2) popular tools for designing an algorithm. These are: (i) *Pseudo-code* and (ii) *Flowchart*. A pseudo-code uses instructions written out in long-hand as well as mathematical statements to express an algorithm. This is done in a fashion similar to human written language discussed above. Pseudo-code are usually not very explicit as their interpretation often require an understanding of the rule for writing the pseudo-code expressions.

In the flowchart approach to algorithm design, however, **Geometric images** are used to represent instructions. Familiarity with only a few geometric images or symbols is required for flowchart algorithm design. We will be using the flowchart more often in this course. Some important flowchart symbols are shown in Figure 1.13. We will discuss them further during our lecture.

1.11 Computer program development process

To develop a program for the solution a problem using the Python programming Language, you need to follow some steps.

Step 1: Understand the problem: It is good to always start your programming by giving the problem it solves a NAME. The first step in computer problem solving is to understand the problem. To do this, you need to take a simple case example and solve the problem manually. This will allow you to better understand the mental activity involved in problem-solving process. The outcome of this process is a **PROGRAM SPECIFICATION DOCUMENT** containing a description of the solution process.

Step 2: Identify Input and Output: It this step the **Input data** are listed with their data types and sample values. Also, the expected outputs are listed and described. The goal of this step is to assign identifiers to each variable and constant in the Input and Output of the problem-solving process. The outcome of this step is a **DATA DICTIONARY** of the program.

Step 3: Formulate the process: The individual manipulation operations required for transforming the input data to its corresponding output data are listed. This will include: what calculations are required and how they will be invoked. You may need to consider a number of alternative solution methods at this stage before settling for the most appropriate for the problem at hand. The outcome of this




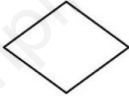


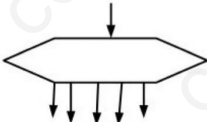


Ser. No.	Geometric Object	Name	Function	Uses
1.		Oval	Terminal	START STOP
2.		Parallelogram	Interface	READ WRITE
3.		Rectangle	Process	STATEMENTS
4.		Rhombus	Query	IF THEN
5.		Double Edge Rectangular	Routine	SUBROUTINE FUNCTION SUB-PROGRAM
6.		Three partition polygon	Repetition	LOOP
7.		Multi-exit Polygon	Multiple option Selection	CASE
8.		Arrow line	Direction	CONTROL FLOW
9.		Circle	Connection	INSTRUCTION LABEL

Figure 1.13: Flowchart Symbols

process is a **PROCESS DICTIONARY**. This comprises a list of the methods, functions and routines that will be execute on the input to produce the final output.

Step 4: Solution Algorithm Design: Formulate an algorithm to realise the Steps 2 and 3. This is the program design. To achieve this, you need to use design tool such as Flowchart symbols and/or Pseudo-code to construct the sequence of steps in the problem-solving into a unified process. It may be the case that the solution will be implemented by breaking a bigger problem into smaller ones. Each of the smaller problems is then implemented as a module. The outcome of this step is the **SOLUTION DESIGN**. The design is the basis for all other future activity that

concerns problem-solution process.

Step 5: Implement the Design: In this stage of the problem-solving, you need to convert the algorithm designed in Step 4 into a computer program. Here you will reduced the Flowchart or Pseudo-code into Python instructions. The tasks here includes:

- (i) Creating the source file. Here you type the program (from paper) into a computer Text Editor.
- (ii) Compiling the program. The program text will be applied to Python Compiler software which will check it for Syntax error. If there are syntax errors (or warning in your program) you will need to correct it in your source file. If no error is found, the system will generate the object or executable file.
- (iii) Repeat steps (i) and (ii) until your program is running as desired.

The outcome of step is a **RUNNING PROGRAM**.

Step 6: Program Testing and Documentation Run the program with the Input data you used in Step 1. This is to confirm that the output you computer manually is also what the computer generates. Thereafter, select a number of example input data that you already familiar with their outputs. If the output produce by your program is not correct, you need to go back to the design [Step 4] of the solution. If the incorrect result persists, then you move to Step 3, and so forth. The outcome of this step is a **PROGRAM TESTING AND DOCUMENTATION**. The document produced in this step will contains a description of how you carried out steps 1 to Step 5 as well as how the outcome of the testing of your program. The document can also contain your views and comments on the extent of the application of the program.

1.11.1 TASK 3

Type and run the following three (3) codes into the Python environment that you installed on your mobile phone. Explain your observations after running the programs. **(HINT: See Subsection 1.2.5).**

CODE 1

```
y = input("Enter a digit")
x = input("Enter another digit")
print (y+x)
print (x+y)
```

CODE 2

```
y = input("Enter a digit")
x = input("Enter another digit")
y = int(y)
x = int(x)
print (y+x)
print (x+y)
```

CODE 3

```
n = input("Please enter a number")
y = n+n
x = int(n) + int(n)
print("The polarity sum is ", y)
print("The logical sum is ", x)
```

1.12 Case example: FindAverage

We will demonstrate the software development process discussed in Section 1.11 by **developing a program to find the average of Five (5) numbers.**

Understand the problem: The name of the program is **FindAverage**. The task of finding the average of Five (5) numbers involves: (i) Finding the **Sum** by adding the numbers together and (ii) Dividing the **Sum** by 5. We will assume that the numbers are of type Real. Type Real are numbers with fractional part. For example, 5 is an Integer because it has no fractional part, but 5.0 is a Real number because it has a fractional part; even when the fractional part is zero.

The command *float* is used to convert an input into a real number in Python. Therefore, an input data is converted to real number by applying the *float* command to it.

The command *int* is used to convert an input into a integer number in Python. Therefore, an input data is converted to integer number by applying the *int* command to it.

If we assume that the numbers we wish to find the Average are:

- 5.0
- 8.0
- 7.0
- 9.0
- 24.0

We will

1. Sum the numbers, i.e. $5.0 + 8.0 + 7.0 + 9.0 + 24.0 = 53.0$

2. Divide the Sum by 5, i.e. $\frac{53.0}{5.0} = 10.6$

Identify the input and output: The input to the problem are the five numbers: let us label them using the variables:

Number1, Number2, Number3, Number4, Number5

The output from the program is the average: let us label it with the variable name: *Average*.

Formulate the process: The process require for computing the output from the input are as follows:

Sum = Number1 + Number2 + Number3 + Number4 + Number5
and

$$\text{Average} = \frac{\text{Sum}}{5.0}$$

Design the solution algorithm: The solution algorithm is represented by the flowchart in Figure 1.14.

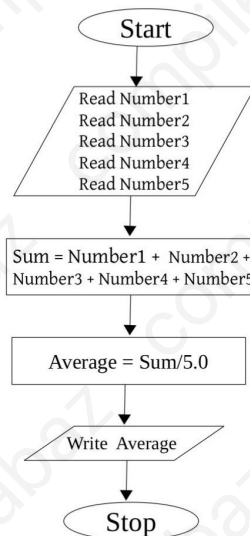


Figure 1.14: Design for the Average Program

Implement the solution: To implement the design, create the source file by typing the following program code into the Python editor.

Run the program as instructed in the CSC201 Laboratory and tutorial manual.

Table 1.5: Python code for the Flowchart in Figure 1.14

```
Number1 = float(input("Enter Number1: "))
Number2 = float(input("Enter Number2: "))
Number3 = float(input("Enter Number3: "))
Number4 = float(input("Enter Number4: "))
Number5 = float(input("Enter Number5: "))
Sum = Number1 + Number2 + Number3 + Number4 + Number5
Average = Sum/5
print("Average of the numbers is :", Avegrage)
```

Test the Program: Run the program using the data used in Step 1 and see your output. Now run the program again with at least Three (3) other set of input of your choice.

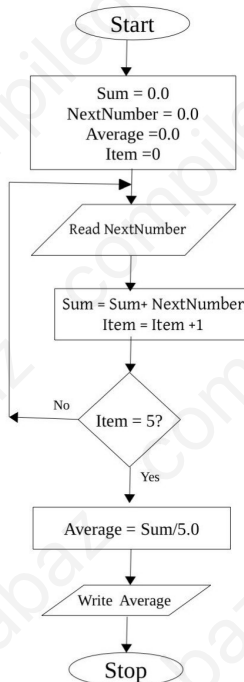


Figure 1.15: Design for the Average Program

The program in Table 1.5 will work only when the number of input is Five (5). What this implies is that even if you want to find the Average of Three (3) numbers,

Table 1.6: Python code for the Flowchart in Figure 1.15

```
item = 5
Sum = 0.0
for n in range(item):
    NextNumber = float(input("Enter the next number : "))
    Sum = Sum + NextNumber
Average = Sum / item
print("Average of ", item, " numbers is :", Average)
```

the program will not work. Also, the instruction to read every number is in the code. Most often, programs are designed to be more flexible. The flowchart in Figure 1.15 is the design for a program that can read a specified number of input data and output their average. The Python code for the flowchart design in Figure 1.15 is in Table 1.6.

The Python program codes in Table 1.5 and Table 1.6 will work only when the number of input is Five (5). What this implies is that even if you want to find the Average of Three (3) numbers, the program will NOT work. Most often, programs are designed to be more flexible. To improve the flexibility of the program, it will be designed to be able to handle various number of input data.

1.13 TASK 4

The Python code to find the average for any number of input data is in Table 1.7. You are to carry our the Program development Steps above in respect of the code.

Table 1.7: Python code for flexible Average computation

```
item = 0
Sum = 0.0
item = input("Enter number of items to average: ")
item = int(items)
while n < item :
    NextNumber = float(input("Enter next number : "))
    Sum = Sum + NextNumber
    n = n + 1
Average = Sum / item
print("Average of ", item, " numbers is :", Average)
```

Table 1.8: Another Python code for flexible Average computation

```
item = 0
Sum = 0.0
item = input("Enter number of items to average: ")
item = int(item)
for n in range(item):
    NextNumber = float(input("Enter next number : "))
    Sum = Sum + NextNumber
Average = Sum / item
print("Average of ", item, " numbers is :", Average)
```

1.14 TASK 5

Using the Python environment installed on **ON YOUR MOBILE PHONE**, carry out the following tasks:

1. Enter and run the following code. Explain your observation about the code.

```
i = input("Enter the number of times for looping")
i = int(i)
for k in range(0,i)
    print("Loop", k)
```

2. Enter and run the following code and explain what it is doing. Replace "Odetunji" with your own name, rerun the program and explain its output. Replace the line `y = "Odetunji"` with `y = input("What is your name")` rerun the program and explain its output.

```
k = 0
y = "Odetunji"
for i in range(len(y))
    x = y[i]
    k = k + 1
    print(x)
print("The number of letters in your name is ", k)
```

3. Modify the code in 2. above such that it can read any name and print the number of letters in it.
4. Enter and run the following code on your phone. Draw a flowchart to explain what the code is doing.

```
Num1 = input (" Please enter first number ")
Num2 = input (" Please enter second number ")
Num1 = int(Num1)
Num2 = int(Num2)
if Num1 > Num2:
    print ("First number is greater than the second")
elif Num2 > Num1:
    print ("Second number is greater than the first")
else:
    print ("The numbers you entered are equal")
```

5. Modify the code in 3. above such that it can read any name and print the number of vowel letters in it. For example your program should output 5 for the name "Odetunji". **Hint:** You need to use the Python *if* statement here.
6. Write an encryption program which reads a string comprising lower-case letter in the English language alphabet. Your program will then encode the string by replacing each letter with its corresponding alphabetic position. The following replacement will be used in the encryption $a = 1, b = 2, c = 3, d = 4, e = 5, \dots, z = 26$. For example your program should output 145 for the string "ade". **Hint:** You need to use the Python *if* statement here. You may also consider **Python list data structure**, that is Letters = [a, b, c d, ..]

1.15 Appendix A

Operator	Operator Name	Operator Function	Example and Explanation
+	Addition	Add two/more values	$x = 4; y = 5;$ $\Rightarrow x + y \Rightarrow 9$
-	Subtraction	Subtract two/more values	$x = 10; y = 4;$ $\Rightarrow x - y \Rightarrow 6$
*	Multiplication	Multiply two/more values	$x = 4; y = 5;$ $\Rightarrow x * y \Rightarrow 20$
/	Division	Divide two values, and produce the n	$x = 10; y = 4;$ $\Rightarrow x / y \Rightarrow 2.25$
%	Modulus	Gives the remainder after one value is divided with other	$x = 14; y = 4;$ $\Rightarrow x \% y \Rightarrow 2$
**	Exponential	Gives the exponential values	$x = 3; y = 4;$ $\Rightarrow x ** y \Rightarrow 81$
//	Floor Division	Gives the integer value on dividing one value with other	$x = 14; y = 4;$ $\Rightarrow x // y \Rightarrow 3$

Figure 1.16: Python Programming language operators

Python Operator Precedence


Precedence	Operator Sign	Operator Name	
Highest	**	Exponentiation	
	+x, -x, ~x	Unary positive, unary negative, bitwise negation	
	*, /, //, %	Multiplication, division, floor, division, modulus	
	+, -	Addition, subtraction	
	<<, >>	Left-shift, right-shift	
	&	Bitwise AND	
	^	Bitwise XOR	
		Bitwise OR	
	==, !=, <, <=, >, >=, is, is not	Comparison, Identity	
	not	Boolean NOT	
	and	Boolean AND	
	or	Boolean OR	
	Lowest		

Figure 1.17: Precedence of operator in Python programming language

CSC 201

ALGORITHMS AND FLOWCHARTS

INTRODUCTION

- The term algorithm originally referred to any computation performed via a set of rules applied to numbers written in decimal form.
- The word is derived from the phonetic pronunciation of the last name of Abu Ja'far Mohammed ibn Musa al-Khowarizmi, who was an Arabic mathematician who invented a set of rules for performing the four basic arithmetic operations (addition, subtraction, multiplication and division) on decimal numbers.

INTRODUCTION CONTD

- An algorithm is a representation of a solution to a problem. If a problem can be defined as a difference between a desired situation and the current situation in which one is, then a problem solution is a procedure, or method, for transforming the current situation to the desired one.
- We solve many such trivial problems every day without even thinking about it, for example making breakfast, travelling to the workplace etc. But the solution to such problems requires little intellectual effort and is relatively unimportant.

INTRODUCTION CONTD

- However, the solution of a more interesting problem of more importance usually involves stating the problem in an understandable form and communicating the solution to others.
- In the case where a computer is part of the means of solving the problem, a procedure, explicitly stating the steps leading to the solution, must be transmitted to the computer.
- This concept of problem solution and communication makes the study of algorithms important to computer science

INTRODUCTION CONTD

- Throughout history, man has thought of ever more elegant ways of reducing the amount of labour needed to do things.
- A computer has immense potential for saving time/energy, as most (computational) tasks that are repetitive or can be generalised can be done by a computer.
- For a computer to perform a desired task, a method for carrying out some sequence of events, resulting in accomplishing the task, must somehow be described to the computer.
- The algorithm can be described on many levels because the algorithm is just the procedure of steps to take and get the result. The language used to describe an algorithm to other people will be quite different from that which is used by the computer, however the actual algorithm will in essence be the same.

PROCEDURE

- A **procedure** is a finite sequence of well-defined instructions, each of which can be mechanically carried out in a finite amount of time.
- The procedure must break up the problem solution into parts that the recipient party can understand and execute.
- In the case of a computer, the problem solution is usually in the form of a program that encompasses the algorithm and explains to the computer a clearly defined procedure for achieving the solution.
- The procedure must consist of smaller steps each of which the computers understand. There may be no ambiguities in the translation of the procedure into the necessary action to be taken.
- A program is then just a specific realisation of an algorithm, which may be executed on a physical device

- A computer is essentially a physical device designed to carry out a collection of primitive actions.
- A procedure is a sequence of instructions written in terms of which evoke a proper operation.
- To make effective use of an algorithm on a computer one must not only find and understand a solution to the problem but also convey the algorithm to the computer, giving the correct sequence of understood commands that represent the same algorithm.

Algorithm

- An algorithm is a set of procedures for solving a problem. it is a finite sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
- An **algorithm** is procedure consisting of a finite set of unambiguous rules (instructions) which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems for any allowable set of input quantities (if there are inputs).
- In other word, an **algorithm** is a step-by-step procedure to solve a given problem
- Since an algorithm is just the solution steps for a problem, it can be represented by ordinary English expressions.

Characteristics of Algorithms

- An algorithm must have a beginning and an end
- The non-ambiguity requirement for each step of an algorithm cannot be compromised.
- The range of inputs for which an algorithm works has to be specified carefully
- The same algorithm can be represented in several different ways
- Several algorithms for solving the same problem may exist
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds
- It must terminate at a reasonable period of time.

- An algorithm is a step-by-step description of a procedure. Writing an algorithm allows you to think about the logic of a program without worrying about the syntax of the programming language.
- You can think of an algorithm as being like a recipe:
 - Put 1 tablespoon of oil in a frying pan.
 - i. Heat the oil on the stove at low temperature.
 - ii. Break three eggs into a bowl.
 - iii. Beat the eggs with a whisk.
 - iv. Add cheese.
 - v. Add seasonings.
 - vi. Add the egg mixture to the frying pan.
 - vii. Cook on both sides.

ALGORITHM DESIGN

- An algorithm can be written by using pseudocode and flowcharts
 - While writing algorithms we will use following symbol for different operations:
 - '+' for Addition
 - '-' for Subtraction
 - '*' for Multiplication
 - '/' for Division and
 - ' ← ' for assignment. For example $A \leftarrow X * 3$ means A will have a value of $X * 3$.

Writing Algorithms Using Pseudocode

- pseudocode is like code but different. In fact, pseudocode is a description of the program flow, written in the language you speak. When you write pseudocode, you usually keep the sentences short and to the point.
- **Pseudocode** is one of the tools that can be used to write a preliminary plan that can be developed into a computer program. **Pseudocode** is a generic way of describing an algorithm without use of any specific programming language syntax. It is, as the name suggests, *pseudo* code —it cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail.
- Example: Problem: Design an algorithm to find the average of two numbers.

Solution:

1. Start
2. Get the first number
3. Get the second number
4. Add the two numbers together
5. Show the result
6. Stop

- Problem 1: Find the area of a Circle of radius r.

Inputs to the algorithm:

Radius r of the Circle.

Expected output:

Area of the Circle

- **Algorithm:**

Step1: Read\input the Radius r of the Circle

Step2: Area = $\text{PI} * r * r$ // calculation of area

Step3: Print Area

Problem2: Write an algorithm to read two numbers and find their sum.

- Inputs to the algorithm:
First num1.
Second num2.
- Expected output:
Sum of the two numbers.
- **Algorithm:**
Step1: Start
Step2: Read\input the first num1.
Step3: Read\input the second num2.
Step4: Sum =num1+num2 // calculation of sum
Step5: Print Sum
Step6: End

Problem 3: Convert temperature Fahrenheit to Celsius










- Inputs to the algorithm:
Temperature in Fahrenheit
- Expected output:
Temperature in Celsius
- **Algorithm:**
Step1: Start
Step 2: Read Temperature in Fahrenheit F
Step 3: $C = 5/9 * (F - 32)$
Step 4: Print Temperature in Celsius: C
Step5: End

Writing Algorithms Using Flowcharts

- A flowchart is a graphical representation of an algorithm.
- You can draw a flowchart on paper or using a flowcharting tool such as Microsoft Visio. There are even flowcharting objects available in Microsoft Office.
- When you draw a flowchart, you should use industry-standard shapes to represent each step in the process. You usually draw the flow from top to bottom or from left to right. Arrows connect the shapes to define the flow.

FLOWCHART SYMBOLS

- There are 6 basic symbols commonly used in flowcharting of assembly language programs:
 - Terminal
 - Process
 - input/output
 - Decision
 - Connector and
 - Predefined Process.
- This is not a complete list of all the possible flowcharting symbols, it is the ones used most often in the structure of programming language

Symbol	Name	Meaning
	Flow line	Used to connect symbols and indicate the flow of logic.
	Terminal	Used to represent the beginning (start) or the end (end) of a task.
	Input/Output	Used for input and output operations, such as reading and printing. The data to be read or printed are described inside.
	Processing	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	Decision	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flow line, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no".
	Off page	Used to indicate that the flowchart continues to a second page.
	Connector	Used to join different flow lines
	Predefined	Used to represent a group of statements that perform one processing task.
	Annotation	Used to provide additional information about another flowchart symbol.

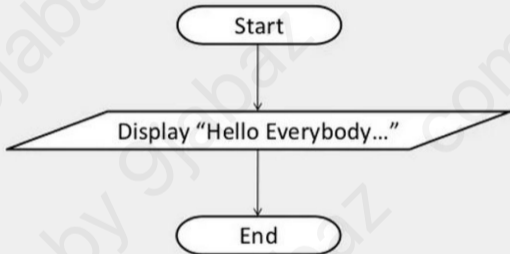
General Rules for flowcharting (1)

1. All boxes of the flowchart are connected with Arrows. (Not lines)
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points.
3. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
4. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
5. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.
6. Connectors are used to connect breaks in the flowchart. Examples are:
 - From one page to another page.
 - From the bottom of the page to the top of the same page.
 - An upward flow of more than 3 symbols

General Rules for flowcharting (2)

7. Subroutines and Interrupt programs have their own and independent flowcharts.
8. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.
9. All flowcharts end with a terminal or a contentious loop

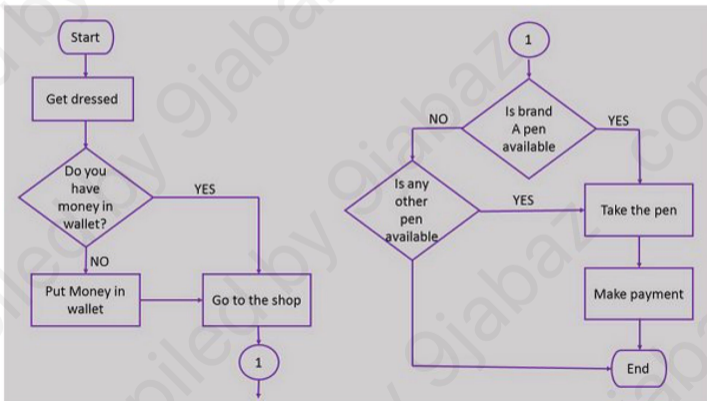
Flowchart :



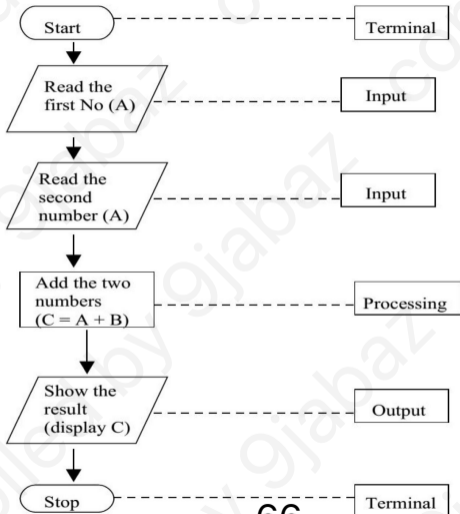
Dry Run :

Hello Everybody...
64

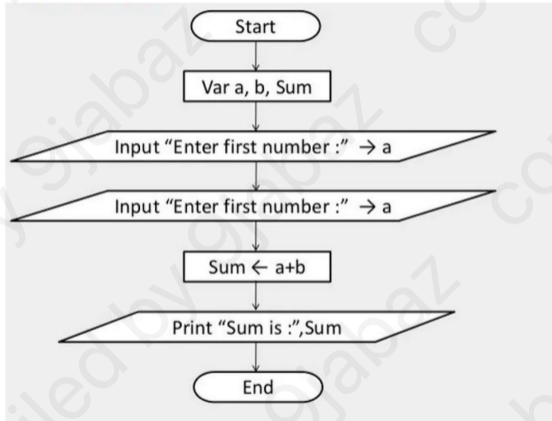
flowchart for going to the market to purchase a pen.



EXAMPLE



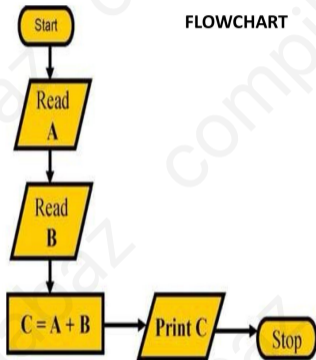
Flowchart to add two(2) numbers and display the result



Example 2: Write an algorithm to find the sum of two numbers.

• **Pseudocode:**

- Step 1 – Start
- Step 2 – Input A
- Step 3 – Input B
- Step 4 – Calculate $C = A + B$
- Step 5 – Output C
- Step 6 – Stop



Example 3: Write an algorithm to find the difference and the division of two numbers and display the results.

• **Pseudocode:**

Step 1: Start

Step 2: Input N1

Step 3: Input N2

Step 4: $D = N1 - N2$

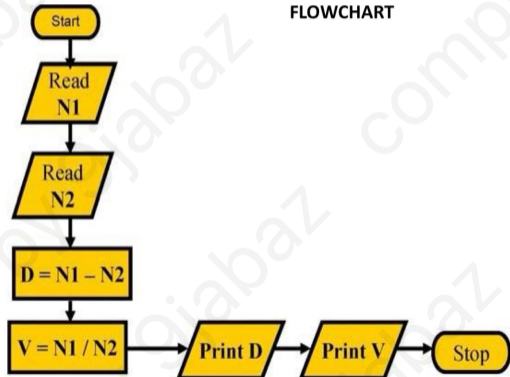
Step 5: $V = N1 / N2$

Step 6: Output D

Step 7: Output V

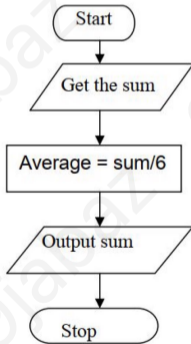
Step 8: Stop

FLOWCHART

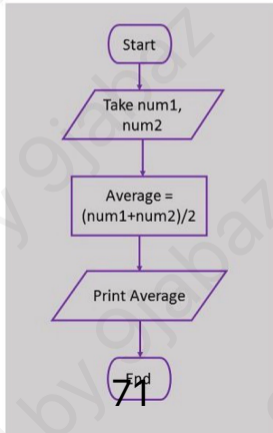


design an algorithm for finding the average of six numbers, and the sum of the numbers

- *Start*
- *Get the sum*
- *Average = sum / 6*
- *Output the average*
- *Stop*



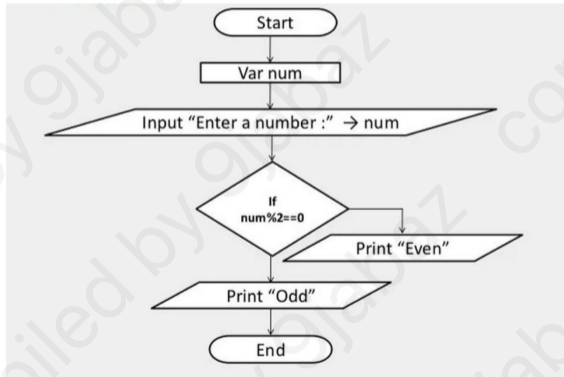
flowchart to calculate the average of two numbers.



input three numbers from the keyboard, ADD and output the result.

- *variables: sum, number1, number2, number3 of type integer*
Accept number1, number2, number3
Sum = number1 + number2 + number3
Print sum
End program

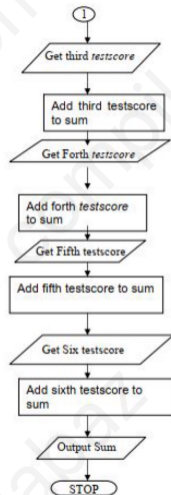
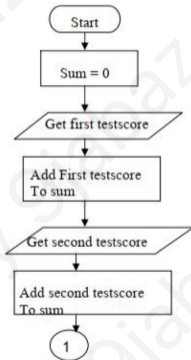
Input numbers from the user and check whether it is even or odd



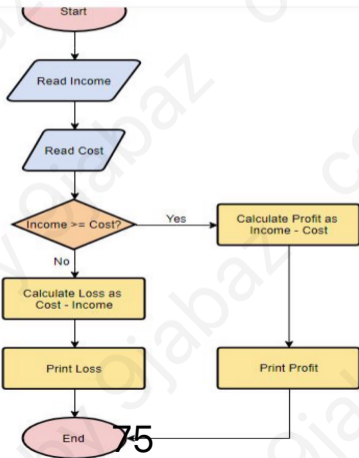
Example 4. Design an algorithm and the corresponding flowchart for adding the test scores as given below:
26, 49, 98, 87, 62, 75

• a) Algorithm

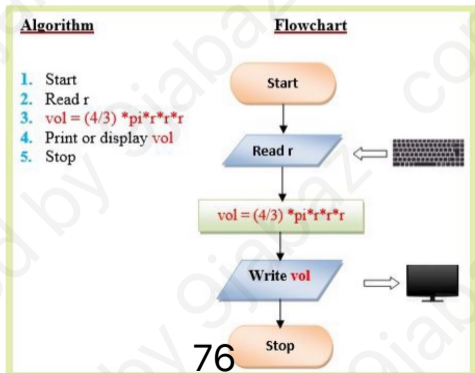
1. Start
2. Sum = 0
3. Get the first test score
4. Add first test score to sum
5. Get the second test score
6. Add to sum
7. Get the third test score
8. Add to sum
9. Get the fourth test score
10. Add to sum
11. Get the fifth test score
12. Add to sum
13. Get the sixth test score
14. Add to sum
15. Output the sum
16. Stop



Calculate Profit and Loss



compute the volume of a sphere. Use the formula:
 $V = (4/3) * \pi * r^3$ where pi is equal to 3.1416
approximately.



converts the input Celsius degree into its equivalent Fahrenheit degree. Use the formula:
 $F = (9/5) * C + 32$.

Algorithm

1. Start
2. Initialize $F=0, C=0$
3. Read C
4. $F_h = (1.8 * C) + 32$
5. Print or display F_h
6. Stop

Flowchart



converts the input dollar to its peso exchange rate equivalent. Assume that the present exchange rate is 51.50 pesos against the dollar. Then display the peso equivalent

Algorithm

1. Start
2. Read dollar
3. $\text{peso} = \text{dollar} * 51.50$
4. Print or display **peso**
5. Stop

Flowchart

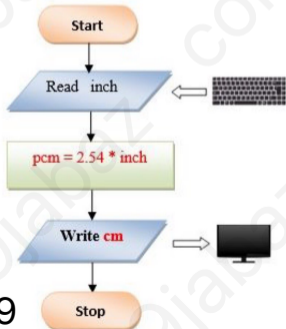


converts an input inch(es) into its equivalent centimeters. Take note that one inch is equivalent to 2.54cms.

Algorithm

1. Start
2. Read inch
3. $cm = 2.54 * inch$
4. Print or display cm
5. Stop

Flowchart

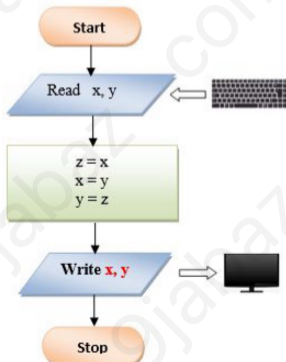


exchanges the value of two variables: x and y . The output must be: the value of variable y will become the value of variable x , and vice versa.

Algorithm

1. Start
2. Read x, y
3. Declare third variable, z
 $z = x$
 $x = y$
 $y = z$
4. Print or display x, y
5. Stop

Flowchart

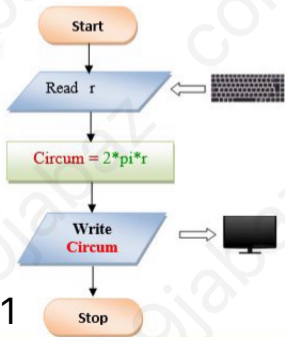


to find the circumference of a circle. Use the formula: $C=2\pi r$, where π is approximately equivalent 3.1416.

Algorithm

1. Start
2. Read r
3. Calculate circumference by the equation:
 $Circum = 2 * \pi * r$
4. Print $Circum$
5. Stop

Flowchart

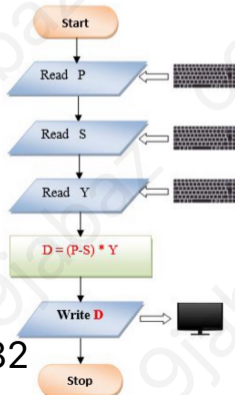


takes as input the purchase price of an item (P), its expected number of years of service (Y) and its expected salvage value (S). Then outputs the yearly depreciation for the item (D). Use the formula: $D = (P - S) / Y$.

Algorithm

1. Start
2. Read P
3. Read S
4. Read Y
5. $D = (P - S) / Y$
6. Print or display D
7. Stop

Flowchart

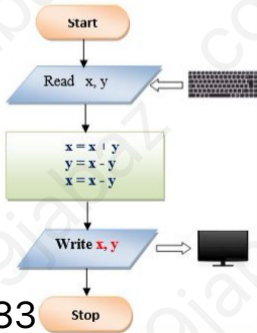


Swapping of 2 variables without using temporary (or 3rd variable).

Algorithm

1. Start
2. Read x and y
3. $x = x + y$
 $y = x - y$
 $x = x - y$
4. Print or display x, y
5. Stop

Flowchart

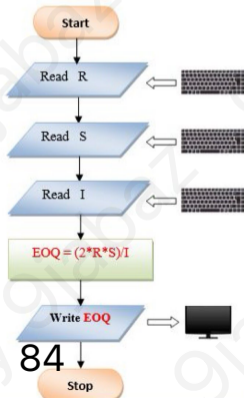


Determine the most economical quantity to be stocked for each product that a manufacturing company has in its inventory: This quantity, called economic order quantity (EOQ) is calculated as follows: $EOQ = \sqrt{2RS/I}$ where: R= total yearly production requirement S=set up cost per order I=inventory carrying cost per unit.

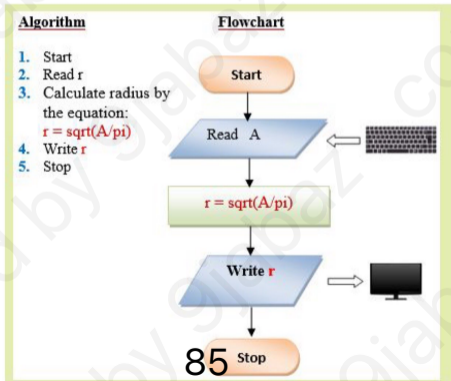
Algorithm

1. Start
2. Read R
3. Read S
4. Read I
5. $EOQ = \sqrt{2RS/I}$
6. Print EOQ
7. Stop

Flowchart



Write a program to compute the radius of a circle.
Derive your formula from the given equation:
 $A = \pi r^2$, then display the output.



Find Perimeter Of Circle using Radius($\pi = 3,14$)

Solve Quadratic Equation $ax^2 + bx + c = 0$

CSC 201

CONTROL STRUCTURES OR LOGICAL STRUCTURES

CONTROL STRUCTURES OR LOGICAL STRUCTURES

- Control Structures are just a way to specify flow of control in programs. Any algorithm or program can be more clear and understood if they use self-contained modules called as logic or control structures. It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions. There are three basic types of logic, or flow of control, known as:
- The key to better algorithm design and thus to programming lies in limiting the controlstructure to only three constructs.
- **The sequence/sequential structure**
- **Decision Structure or Selection Structure/conditional structure**
- **Repetition or Iteration/looping Structure**

1. The sequence structure

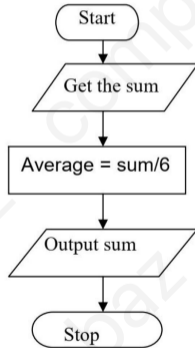
- The first type of control structures is called the sequence structure. This structure is the most elementary structure. The sequence structure is a case where the steps in an algorithm are constructed in such a way that, no condition step is required. The sequence structure is the logical equivalent of a straight line.
- Sequential logic as the name suggests follows a serial or sequential flow in which the flow depends on the series of instructions given to the computer. Unless new instructions are given, the modules are executed in the obvious sequence.

Sequence

For example, suppose you are required to design an algorithm for finding the average of six numbers, and the sum of the numbers is given. The pseudocode will be as follows:

Start
Get the sum
Average = sum / 6
Output the average
Stop

The corresponding flowchart will appear as follows:

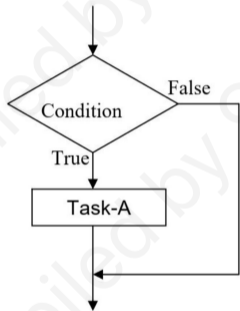


2. Decision Structure or Selection Structure

- The decision structure or mostly commonly known as a selection structure, is case where in the algorithm, one has to make a choice of two alternatives by making decision depending on a given condition.
- Selection Logic simply involves a number of conditions or parameters which decides one out of several written modules.
- Selection structures are also called **case** selection structures when there are two or more alternatives to choose from.
- The selection requires the following
 - Choose alternative actions as a result of testing a logical condition
 - Produce code to test a sequence of logical tests
- The decision or selection structure can be implemented using
 - **IF statement**
 - **Else statement**
 - **CASE statement**

Using if statement

From this structure, we have the following



If condition is true then

Do Task-A

else

Do task B

In this case, if condition is false, nothing happens. Otherwise Task-A is executed.

Making Choices....

- There are many occasions where a program is required to take alternative actions.
- For example, there are occasions where we need to take action according to the user choice.
- All computer languages provide a means of selection. Usually it is in the form of If statement and our pseudo-code is no exception to this.
- We will use the if statement together with logical operators to test for true or false as shown below.

```
If a = b
```

```
  print "a = b"
```

- The action is only taken when the test is true.

The logical operators used in our pseudo-code are

= is equal to

> is greater than

< is less than

>= is greater than or equal

<= is less than or equal

<> is not equal to

Compound Logical Operators

- There are many occasions when we need to extend the conditions that are to be tested. Often there are conditions to be linked.
- In everyday language we say things like **If I had the time and the money I would go on holiday**. The **and** means that both conditions must be **true** before we take an action.
- We might also say **I am happy to go to the theatre or the cinema**. The logical link this time is **or** .
- Conditions in if statements are linked in the same way. Conditions linked with **and** only result in an action when all conditions are true. For example, if $a > b$ and $a > c$ then display "a is the largest".
- Conditions linked with an or lead to an action when either or both are true.

Selection Example:

- The following shows how the selection control structure is used in a program where a user chooses the options for multiplying the numbers or adding them or subtracting.
- *Use variables: choice, of the type character
ans, number1, number2, of type integer
display "choose one of the following"
display "m for multiply"
display "a for add"
display "s for subtract"
accept choice
display "input two numbers you want to use"
accept number1, number2

if choice = m then ans = number1 * number2
if choice = a then ans = number1 + number2
if choice = s then ans = number1 - number2
display ans*

Example 2

- The program is to input an examination mark and test it for the award of a grade. The mark is a whole number between 1 and 100. Grades are awarded according to the following criteria:

>= 80 Distinction

>= 60 Merit

>= 40 Pass

< 40 fail

- The pseudo-code is

Use variables: mark of type integer

If mark >= 80 display "distinction"

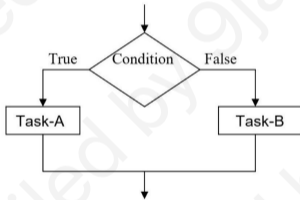
If mark >= 60 and mark < 80 display "merit"

If mark >= 40 and mark < 60 display "pass"

If mark < 40 display "fail"

Using else statement

- This structure can be illustrated in a flowchart as follows:



In pseudocode form we get
If condition is true
Then do task A
else
Do Task-B

In this example, the condition is evaluated, if the condition is true Task-A is evaluated and if it is false, then Task-B is executed.

Using else statement

- An if statement on its own is often not the best way of solving problems. A more elegant set of conditions can be created by adding an else statement to the if statement. The else statement is used to deal with situations as shown in the following examples.
- The else statement provides a neat way of dealing with alternative condition

Example : A person is paid at top for category 1 work otherwise pay is at normal rate.

- *If the work is category 1
pay-rate is top
Else
pay-rate is normal*

using case statement

- Repeating the if ... else statements a number of times can be somewhat confusing. An alternative method provided in a number of languages is to use a selector determined by the alternative conditions that are needed. In our pseudo-code, this will be called a **case statement**.

Example: The following program segment outputs a message to the monitor screen describing the insurance available according to a category input by the user.

Using else statement

```
Use variables: category of type character
Display "input category"
Accept category
If category = U
Display "insurance is not available"
Else
If category = A then
Display "insurance is double"
Else
If category = B then
Display "insurance is normal"
Else
If category = M then
Display "insurance is medically dependent"
Else
Display "entry invalid"
```

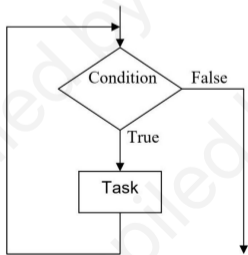
Using case statement

```
Use variables: category of type character
Display "input category"
Accept category
DO case of category
CASE category = U
Display "insurance not available"
CASE category = A
Display "insurance is double"
CASE category = B
Display "insurance is normal"
CASE category = M
Display "insurance is medically dependent"
OTHERWISE
Display "entry is invalid"
ENDCASE
```

Note: **101** instead of using the word otherwise, one can use else.

Repetition or Iteration Structure

- A third structure causes the certain steps to be repeated. Any program instruction that repeats some statement or sequence of statements a number of times is called an iteration or a loop. The commands used to create iterations or loops
- The commands used to create iterations or loops are all based on logical tests.



The Repetition structure can be implemented using:

- Repeat Until Loop
- The While Loop
- The For Loop

The Repeat Until loop.

-

The syntax is

REPEAT

A statement or block of statements

UNTIL *a true condition*

- **Example :** A program segment repeatedly asks for entry of a number in the range 1 to 100 until a valid number is entered.

REPEAT

DISPLAY "Enter a number between 1 and 100"

ACCEPT number

UNTIL number < 1 OR number > 100

Example 2. A survey has been carried out to discover the most popular sport. The results will be typed into the computer for analysis. Write a program to accomplish this.

- REPEAT
 DISPLAY "Type in the letter chosen or Q to finish"
 DISPLAY "A: Athletics"
 DISPLAY "S: Swimming"
 DISPLAY "F: Football"
 DISPLAY "B: Badminton"
 DISPLAY "Enter data"
 ACCEPT letter
 If letter = 'A' then
 Athletics = athletics + 1
 If letter = 'S' then
 Swimming = Swimming + 1
 If letter = 'F' then
 Football = Football + 1
 If letter = 'B' then
 Badminton = Badminton + 1
 UNTIL letter = 'Q'
 DISPLAY "Athletics scored", athletics, "votes"
 DISPLAY "Swimming scored", swimming, "votes"
 DISPLAY "Football scored", football, "votes"
 DISPLAY "Badminton scored", Badminton, "votes"

The WHILE loop

- The second type of iteration we will look at is the while iteration. This type of conditional loop tests for terminating condition at the beginning of the loop. In this case no action is performed at all if the first test causes the terminating condition to evaluate as false.

The syntax is:

- **WHILE** (a condition is true)
A statement or block of statements
ENDWHILE

- **Example 11:** A program segment to print out each character typed at a keyboard until the character 'q' is entered.

- *WHILE letter <> 'q'*
ACCEPT letter
DISPLAY "The character you typed is", letter
ENDWHILE

Example 2: Write an algorithm that will output the square root of any number input until the number input is zero.

- In some cases, a variable has to be initialised before execution of the loop as shown in the following example.
- *Use variable: number of type real*
DISPLAY "Type in a number or zero to stop"
ACCEPT number
WHILE number <> 0
*Square = number * number*
DISPLAY "The square of the number is", square
DISPLAY "Type in a number or zero to stop"
ACCEPT number
ENDWHILE

The FOR Loop

- The third type of iteration, which we shall use when the number of iterations is known in advance, is a **for loop**.
- This, in its simplest form, uses an initialisation of the variable as a starting point, a stop condition depending on the value of the variable. The variable is incremented on each iteration until it reaches the required value.
- The pseudo-code syntax will be:
FOR (*starting state, stopping condition, increment*)
Statements
ENDFOR

- **Example 1.**

```
FOR (n = 1, n <= 4, n + 1)  
  DISPLAY "loop", n  
ENDFOR
```

- The fragment of code will produce the output

```
Loop 1  
Loop 2  
Loop 3  
Loop 4
```

- In the example, n is usually referred as the loop variable, or counting variable, or index of the loop. The loop variable can be used in any statement of the loop. The variable should not be assigned a new value within the loop, which may change the behaviour of the loop.

Example 2: Write an algorithm to calculate the sum and average of a series of numbers.

•

The pseudo-code solution is:

Use variables: n, count of the type integer

Sum, number, average of the type real

DISPLAY "How many numbers do you want to input"

ACCEPT count

SUM = 0

FOR (n = 1, n <= count, n + 1)

DISPLAY "Input the number from your list"

ACCEPT number

SUM = sum + number

ENDFOR

Average = sum / count

DISPLAY "The sum of the numbers is ", sum



Introduction to Programming with Python

Introduction to Python

- Python is a high-level programming language
- Open source and community driven
- “Batteries Included”
 - a standard distribution includes many modules
- Dynamic typed
- Source can be compiled or run just-in-time
- Similar to perl, tcl, ruby

Features of Python

Python is a high-level, interpreted, interactive and object oriented-scripting language. Python was designed to be highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** This means that you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is Beginner's Language:** Python is a great language for the beginner programmers and supports the development of a wide range of applications, from simple text processing to WWW browsers to games.

- Python's feature highlights include:
- **Easy-to-learn:** Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language in a relatively short period of time.
- **Easy-to-read:** Python code is much more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's success is that its source code is fairly easy-to-maintain.
- **A broad standard library:** One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above mentioned features, Python has a big list of good features, few are listed below:

- Support for functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- Very high-level dynamic data types and supports dynamic type checking.
- Supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

By the way, the language is named after the BBC show “Monty Python’s Flying Circus” and has nothing to do with nasty reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

Python 3 is available for Windows, Mac OS and most of the flavors of Linux operating system. Even though Python 2 is available for many other OSs, Python 3 support either has not been made available for them or has been dropped.

Assignment 1:

- DOWNLOAD PYTHON INTERPRETER
- Installing Python Programming Language

Getting Python on Windows platform:

Binaries of latest version of Python 3 are available on Python Official Website.

For versions 3.0 to 3.4.x, Windows XP is acceptable. In order to install Python 3.5 to 3.8- minimum OS requirements are Windows 7 with SP1. Python 3.9+ cannot be used on Windows 7 or earlier.

Python 3.11 is the newest major release of the Python programming language:

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python. You can download Python documentation from the site. The documentation is available in HTML, PDF and PostScript formats.

Python Official Website: <http://www.python.org/> Python Official Website: <http://www.python.org/>

- You can create and run python codes on your mobile devices. There are so many python 3 interpreter and IDE available on Google and IOS play store. For example, Pydroid 3 on Android phone.
- -Understand the following:

- Local Environment Setup**
- Running Python**

First Python Program

•Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt-

```
$ python
Python 3.3.2 (default, Dec 10 2013, 11:35:01)
[GCC 4.6.3] on Linux
Type "help", "copyright", "credits", or "license" for more information.
>>>
On Windows:
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Type the following text at the Python prompt and press Enter-

```
>>> print ("Hello, Python!")
```

Program Elements

- Identifiers:
- Variables:
- Reserved words

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (`_`) followed by zero or more letters, underscores and digits (0 to 9).
- Must begin with letter or underscore, followed by any number of letters, digits, underscores

Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _.

Valid examples: Names like **myClass**, **var_1** and **print_this_to_screen**,

- An identifier cannot start with a digit.
1variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.

```
>>> global = 1
```

```
File "<interactive input>", line 1
```

```
  global = 1
```

```
    ^
```

```
SyntaxError: invalid syntax
```

Rules for writing identifiers

- We cannot use special symbols like !, @, #, \$, % etc. in our identifier.

```
>>> a@ = 0
```

```
File "<interactive input>", line 1
```

```
a@ = 0
```

```
^
```

SyntaxError: invalid syntax

- Identifier can be of any length.

Case-sensitivity

- Python is a case-sensitive language. This means, **Variable** and **variable** are not the same.
- Thus, **Manpower** and **manpower** are two different identifiers in Python.

Variables

- Variables:
 - Do not need to be declared
 - A variable is created when a value is assigned to it:
Examples: `num = 3`
 - Can't be used in an expression unless it has a value
 - Error message: *Name Error* – means no value is associated with this name

Variables

- In many programming languages, variables are statically typed. That means a variable is initially declared to have a specific data type, and any value assigned to it during its lifetime must always have that type.

```
>>> var = 23.5
>>> print(var)
23.5
```

- Variables in Python are not subject to this restriction. In Python, a variable may be assigned a value of one type and then later re-assigned a value of a different type:

```
>>> var = "Now I'm a string"
>>> print(var)
Now I'm a string
```

- Variable names don't have static types – values (or objects) do
A variable name has the type of the value it currently references

Variables contain references to data values

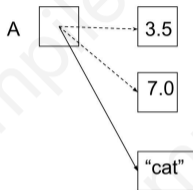
Variables actually contain references to values (similar to pointers).

This makes it possible to assign different object types to the same variable



`A = A * 2`

`A = "cat"`



- Python handles memory management automatically. It will create new objects and store them in memory; it will also execute garbage collection algorithms to reclaim any inaccessible memory locations.
- Python does not implement reference semantics for simple variables; if `A = 10` and `B = A`, `A = A + 1` does not change the value of `B`

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constants or variables or any other identifier names. All the Python keywords contain **lowercase letters only**.

Reserved words

and	exec	Not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Lines and Indentation

- Python does not use braces ({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example-

```
if True:
    print ("True")
else:
    print ("False")
```

However, the following block generates an error-

```
if True:
    print ("Answer")
    print ("True")
else:
    print "(Answer)"
    print ("False")
```

Thus, in Python all the continuous lines indented with the same number of spaces would form a block.

Multi-Line Statements

- Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (`\`) to denote that the line should continue. For example

```
total = item_one + \  
        item_two + \  
        item_three
```

- The statements contained within the `[]`, `{}`, or `()` brackets do not need to use the line continuation character. For example

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```

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."""
```

Comments in Python

- A hash sign (#) that is not inside a string literal is the beginning of a comment. All characters after the #, up to the end of the physical line, are part of the comment and the
- Python interpreter ignores them.

```
# First comment  
print ("Hello, Python!") # second comment
```

This produces the following result-

```
Hello, Python!
```


- You can type a comment on the same line after a statement or expression

```
name = "Madisetti" # This is again comment
```

- Python does not have multiple-line commenting feature. You have to comment each line individually as follows-

```
# This is a comment.  
# This is a comment, too.  
# This is a comment, too.  
# I said that already.
```

Using Blank Lines

- A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.
- In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Interesting Features

- White space does indicate meaning
 - Instead of curly brackets or begin-end pairs, whitespace is used for block delimiters.
 - Indent statements in a block, un-indent at end of block.
- Statements are terminated by <Enter>
- No variable declarations
- Dynamic typing

Python Basics

DR. B.O. AKINYEMI

Interactive Input statements

- The input function is a simple way for your program to get information from people using your program. The basic structure is:

variable = input("string")

variable name = input(message to user)

- Here is an example:

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

- *input()* reads input from the keyboard as a string;
- The input function's job is to ask the user to type something in and to capture what the user types. The part in quotes is the prompt that the user sees. It is called a *string* and it will appear to the program's user exactly as it appears in the code itself.

Input statements- numeric data

- To get numeric data use a cast :

```
>>> number = int(input("enter an integer: "))
```

```
enter an integer: 87
```

```
>>> number
```

```
87
```

- If types don't match (e.g., if you type 4.5 and try to cast it as an integer) you will get an error:

- Multiple inputs:

```
>>> x, y = int(input("enter an integer: ")),
```

```
float(input("enter a float: "))
```

```
enter an integer: 3
```

```
enter a float: 4.5
```

```
>>> print("x is", x, " y is ", y)
```

```
x is 3 y is 4.5
```

Input statements- eval() function

- Instead of the cast you can use the `eval()` function and Python choose the correct types:

```
>>> x, y = eval(input("Enter two numbers: "))
```

```
Enter two numbers: 3.7, 98
```

```
>>> x, y  
(3.7, 98)
```

Output Statements

- Statement syntax: (EBNF notation)
print (*{expression,}*)
where the final comma in the list is optional
- Examples:
 - >>>print() # prints a blank line
 - >>>print(3, y, z + 10) # go to new line
 - >>>print('result: ', z, end=" ")# no newline
- Output items are automatically separated by a single space.
 - >>>name = input('Enter your name: ')
 - >>>print('Hello, ', name)

Exercise 1

```
temp = eval(input('Enter a temperature in Celsius: '))  
print('In Fahrenheit, that is', 9/5*temp+32)
```

Let's examine how the program does what it does. The first line asks the user to enter a temperature. The `input` function's job is to ask the user to type something in and to capture what the user types. The part in quotes is the prompt that the user sees. It is called a *string* and it will appear to the program's user exactly as it appears in the code itself. The `eval` function is something we use here, but it won't be clear exactly why until later. So for now, just remember that we use it when we're getting numerical input.

We need to give a name to the value that the user enters so that the program can remember it and use it in the second line. The name we use is `temp` and we use the equals sign to assign the user's value to `temp`.

The second line uses the `print` function to print out the conversion. The part in quotes is another string and will appear to your program's user exactly as it appears in quotes here. The second

argument to the `print` function is the calculation. Python will do the calculation and print out the numerical result.

Assignment Statements

- It calculates the value of the expression to the right of the equal sign and assigns that value to the variable named on the left of the equal sign.
- Syntax: *variable = expression*
- Example
D= 2 * A*B, Test= 4 >= 8
- A variable's type is determined by the type of the value assigned to it.
- Multiple assignment $\rightarrow var\{, var\} = expr\{, expr\}$

```
>>> x, y = 4, 7
```

```
>>> x
```

```
4
```

```
>>> y
```

```
7
```

```
>>> x, y = y, x
```

```
>>> x
```

```
7
```

```
>>> y
```

```
4
```

```
>>> 140
```

Expression, Operators and Operands

Expression can be any valid combination of constants, variables, parentheses and arithmetic and logical operators. Tables 1.1 and 1.2 respectively described arithmetic and relational operators.

The operators **indicate what action or operation to perform**. The operands indicate what items to apply the action to.

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

=	Equal to (
/=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Logical operators are words (*and, or, not*) *not* symbols

Precedence Rule

A set of formation rules is used to establish the interpretation of an arithmetic expression that contains two or more operators. This rule determines the order in which the operands are to be combined unless the order is changed by the use of parentheses. The precedence of the arithmetic operators is as follows:

<u>Operator</u>	<u>Precedence</u>
**	<u>Highest</u>
* and /	<u>Intermediate</u>
+ and -	<u>Lowest</u>

Expressions

- An expression calculates a value
- Arithmetic operators: $+$, $-$, $*$, $/$, $**$ (exponentiation)
- Add, subtract, multiply, divide work just as they do in other C-style languages (BEDMAS)
- OR---A common mnemonic to remember this rule is *PEMDAS*, or *Please Excuse My Dear Aunt Sally*.
- Spaces in an expression are not significant
- Parentheses override normal precedence

Expressions

- Mixed type (integer and float) expressions are converted to floats:

```
>>> 4 * 2.0 / 6
1.3333333333333333
```

- Mixed type (real and imaginary) conversions:

```
>>> x = 5 + 13j
>>> y = 3.2
>>> z = x + y
>>> z
(8.2 + 13J)
```

- Explicit casts are also supported:

```
>>> y = 4.999      >>> x = 8
>>> int(y)         >>> float(x)
4                  8.0
```

1. if $x=2.0$, $a=2.0$ and $b=4.0$. What is the value of y if:

$$y = a*x + b**2/x ?$$

- i. 8.0
- ii. **12.0**
- iii. 16.0
- iv. 32.0
- v. 64.0

2. If $x=1.5$, $i=3$, and $j=2$. What is the value of y if:

$$y = 2*x + i/j$$

- i. 3.0
- ii. **3.5**
- iii. 4.0
- iv. 4.5
- v. 5.0

3. If $x=4.5$, $y=3.0$, and $w=1.5$, What is the value of z if: $z = x+w/y+1$

- i. 1.5
- ii. 3.0**
- iii. 4.5
- iv. 6.0
- v. not determined due to a Syntax error

4. if $x=1.0$, $y=2.0$ and $w=3.0$. What is the value of z if:
 $z=2.0(x(y+3.0)+w)$

- i. 12.0
- ii. 16.0
- iii. 18.0
- iv. 22.0**
- v. not determined due to a syntax error

5. If $y = c/d + a*x**2-5$

Which operation is performed first by the computer?

- i. /
- ii. +
- iii. ***
- iv. **

6. Which of the following set of integers are incorrect

i) 1,234 ii) -345 iii) 0 iv) 12.0 v) --3

- a) None of the above
- b) All of the above
- c) i,ii,v
- d) i,iv,v**

Exercise

A program that computes the average of two numbers that the user enters:

```
num1 = eval(input('Enter the first number: '))
num2 = eval(input('Enter the second number: '))
print('The average of the numbers you entered is', (num1+num2)/2)
```

- For this program we need to get two numbers from the user. There are ways to do that in one line, but for now we'll keep things simple. We get the numbers one at a time and give each number its own name.
- The only other thing to note is the parentheses in the average calculation. This is because of the order of operations. All multiplications and divisions are performed before any additions and subtractions, so we have to use parentheses to get Python to do the addition first.

Points to Understand the Code

- **Case:** Case matters. To Python, print, Print, and PRINT are all different things. For now, stick with lowercase as most Python statements are in lowercase.
- **Spaces:** Spaces matter at the beginning of lines, but not elsewhere. (Indentation matters to meaning the code). For example, the code below will not work.

```
temp = eval(input('Enter a temperature in Celsius: '))
```

```
print('In Fahrenheit, that is', 9/5*temp+32)
```

- Block structure indicated by indentation
- The first assignment to a variable creates it
 - Dynamic typing: no declarations, names don't have types, objects do
- Assignment uses = and comparison uses ==
- For numbers + - * / % are as expected.
 - Use of + for string concatenation.
 - Use of % for string formatting (like printf in C)

PYTHON DATA TYPES

DR. B.O. AKINYEMI

Python data types

- Python data types are divided in two categories, mutable data types and immutable data types.
- **Immutable Data types in Python – cannot be changed**
 1. Numeric
 2. String
 3. Tuple
- **Mutable Data types in Python**
 1. List
 2. Dictionary
 3. Set

Basic Data Types

- Numeric types: integer, floats, complex
- A literal with a decimal point is a float; otherwise an integer
- Complex numbers use “j” or “J” to designate the imaginary part: $x = 5 + 2j$
- `type()` returns the type of any data value:

`int()` `float()`

Data Types

```
>>> type(15)
<class 'int'>
>>> type(3.)
<class 'float'>
>>> x = 34.8
>>> type(x)
<class 'float'>
```

```
>>> 1j * 1j
(-1+0j)
>>> s = 3 + 1j
>>> type(s)
<class 'complex'>
>>> x = "learning"
>>> type(x)
<class 'str'>
```

Numeric Data Type in Python

- Integer – In Python 3, there is no upper bound on the integer number which means we can have the value as large as our system memory allows.

```
# Integer number
```

```
num = 100
```

```
print(num)
```

```
print("Data Type of variable num is", type(num))
```

- Output:

```
100
```

```
Data Type of variable num is 154 class 'int'>
```


Data Types in Python

- Python Numbers
- Python List
- Python Tuple
- Python Strings
- Python Set
- Python Dictionary

Basic Datatypes

- **Integers (default for numbers)**

`z = 5 / 2` # Answer 2, integer division

- **Floats**

`x = 3.456`

- **Strings**

- Can use `"..."` or `'...'` to specify, `"foo" == 'foo'`
- Unmatched can occur within the string
`"John's"` or `'John said "foo!".'`
- Use triple double-quotes for multi-line strings or strings than contain both `'` and `"` inside of them:
`"""a'b'c"""`

- **Long** – Long data type is deprecated in Python 3 because there is no need for it, since the integer has no upper limit, there is no point in having a data type that allows larger upper limit than integers.
- **Float** – Values with decimal points are the float values, there is no need to specify the data type in Python. It is automatically inferred based on the value we are assigning to a variable. For example here fnum is a float data type.

```
# float number
```

```
fnum = 34.45
```

```
print(fnum)
```

```
print("Data Type of variable fnum is", type(fnum))
```

Complex Number

- Numbers with real and imaginary parts are known as complex numbers.

```
# complex number
```

```
cnum = 3 + 4j
```

```
print(cnum)
```

```
print("Data Type of variable cnum is", type(cnum))
```

- Output:

```
(3+4j)
```

```
Data Type of variable cnum is <class 'complex'>
```

Binary, Octal and Hexadecimal numbers

- In Python we can print decimal equivalent of binary, octal and hexadecimal numbers using the prefixes.
- 0b(zero + 'b') and 0B(zero + 'B') – Binary Number
- 0o(zero + 'o') and 0O(zero + 'O') – Octal Number
- 0x(zero + 'x') and 0X(zero + 'X') – Hexadecimal Number

```
# integer equivalent of binary number 101
```

```
num = 0b101
```

```
print(num)
```

```
# integer equivalent of Octal number 32
```

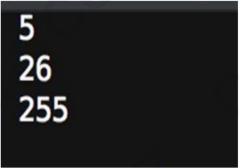
```
num2 = 0o32
```

```
print(num2)
```

```
# integer equivalent of Hexadecimal number FF
```

```
num3 = 0xFF
```

```
print(num3)
```



5
26
255

Python Data Type – String

- String is a sequence of characters in Python. The data type of String in Python is called “str”.
- Strings in Python are either enclosed with single quotes or double quotes.
- # Python program to print strings and type

```
s = "This is a String"  
s2 = 'This is also a String'
```

```
# displaying string s and its type  
print(s)  
print(type(s))
```

```
# displaying string s2 and its type  
print(s2)  
print(type(s2))
```

```
This is a String  
<class 'str'>  
This is also a String  
<class 'str'>
```

Python Data Type – Tuple

- Tuple is immutable data type in Python which means it cannot be changed. It is an ordered collection of elements enclosed in round brackets and separated by commas

```
# tuple of integers
```

```
t1 = (1, 2, 3, 4, 5)
```

```
# prints entire tuple
```

```
print(t1)
```

```
# tuple of strings
```

```
t2 = ("hi", "hello", "bye")
```

```
# loop through tuple elements
```

```
for s in t2:
```

```
    print (s)
```

```
# tuple of mixed type elements
```

```
t3 = (2, "Lucy", 45, "Steve")
```

```
print(t3[2])
```

```
(1, 2, 3, 4, 5)
```

```
hi
```

```
hello
```

```
bye
```

```
45
```

Python Data Type – List

- List is similar to tuple, it is also an ordered collection of elements, however list is a mutable data type which means it can be changed unlike tuple which is an immutable data type. A list is enclosed with square brackets and elements are separated by commas.

```
# list of integers
```

```
lis1 = (1, 2, 3, 4, 5)
```

```
# prints entire list
```

```
print(lis1)
```

```
# list of strings
```

```
lis2 = ("Apple", "Orange", "Banana")
```

```
# loop through tuple elements
```

```
for x in lis2:
```

```
    print (x)
```

```
# List of mixed type elements
```

```
lis3 = (20, "Chaitanya", 15, "BeginnersBook")
```

```
print("Element at index 3 is:",lis3[3])
```

```
(1, 2, 3, 4, 5)
```

```
Apple
```

```
Orange
```

```
Banana
```

```
Element at index 3 is: BeginnersBook
```


Python Data Type – Dictionary

- Dictionary is a collection of key and value pairs. A dictionary doesn't allow duplicate keys but the values can be duplicate. It is an ordered, indexed and mutable collection of elements.
- The keys in a dictionary doesn't necessarily to be a single data type,
- # Dictionary example

```
dict = {1:"Chaitanya","lastname":"Singh", "age":31}
# prints the value where key value is 1
print(dict[1])
# prints the value where key value is "lastname"
print(dict["lastname"])
# prints the value where key value is "age"
print(dict["age"])
```

```
Chaitanya
Singh
31
```

```
163
```

Python Data Type – Set

- A set is an unordered and unindexed collection of items. This means when we print the elements of a set they will appear in the random order and we cannot access the elements of set based on indexes because it is unindexed.
- Elements of set are separated by commas and enclosed in curly braces.
- Set Example

```
myset = {"hi", 2, "bye", "Hello World"}  
# loop through set  
for a in myset:  
    print(a)  
# checking whether 2 exists in myset  
print(2 in myset)  
# adding new element  
myset.add(99)  
print(myset)
```

```
Hello World  
2  
hi  
bye  
True  
{'Hello World', 2, 99, 'hi', 'bye'}
```

Python Programming Examples

DR. B.O. AKINYEMI

Python program to print Hello World

```
# This prints Hello World on the output screen  
print('Hello World')
```

- Output:

Hello World

Getting Input from User in Python

- input() function is used in Python to get user input.

- Get String input from user**

```
# Python Program - Get String Input from User
```

```
str = input("Enter any string: ")
```

```
print(str)
```

- Output:**

```
Enter any string: AKINYEMI
```

```
AKINYEMI
```

Get Integer Input from user

```
# Python Program - Get Integer Input from User  
num = int(input("Enter an Integer: "))  
print(num)
```

•Output:

Enter an Integer : 10

10

Get Float Input from user

```
# Python Program - Get Float Input from User  
num = float(input("Enter a float value: "))  
print(num)
```

- **Output:**

Enter a float value: 25.5

25.5

Python Program to Add Two Numbers

- **hardcoded values in the source code:**

```
# two float values
```

```
val1 = 100.99
```

```
val2 = 76.15
```

```
# Adding the two given numbers
```

```
sum = float(val1) + float(val2)
```

```
# Displaying the addition result
```

```
print("The sum of given numbers is: ", sum)
```

- **Output:**

The sum of given numbers is: 177.14 170

- **Adding the numbers entered by User**

```
# Getting the values of two numbers from user
```

```
val1 = float(input("Enter first number: "))
```

```
val2 = float(input("Enter second number: "))
```

```
# Adding the numbers
```

```
sum = val1 + val2
```

```
# Displaying the result
```

```
print("The sum of input numbers is: ", sum)
```

Output:

```
Enter first number: 10
```

```
Enter second number: 35.2
```

```
The sum of input numbers is: 45.2
```

Using built-in functions

- Note: The reason we used the float() function over the input() function is because the input() function receives the value as String, so to convert it into a float number we must use the float() function

- If integer numbers:

```
val1 = int (input("Enter first number: "))
```

The **int() function** converts the given string into an integer number

- If Binary

```
val1 = bin (input("Enter first number: "))
```

The **bin() function** converts the given string into a binary number

Program for adding two binary numbers

```
# decimal value 1
num1 = '00001'
# decimal value 17
num2 = '10001'
# sum - decimal value 18
# binary value 10010
sum = bin(int(num1,2) + int(num2,2))
print(sum)
```

- Conversion of the string(which is a binary value) into an integer number so we are passing the base value as 2 (binary numbers have base 2, decimals have base value 10).
- Once the strings are converted into an integer values then we are adding them and the result is converted back to binary number using the **bin()** function.

Python Program for simple interest

- Simple interest formula is given by:

$$\text{Simple Interest} = (P \times T \times R) / 100$$

Where,

P is the principle amount

T is the time and

R is the rate

```
# Python program to find simple interest
```

```
SI = (P * R * T) / 100
```

```
# Print the resultant value of SI
```

```
print("simple interest is", SI)
```

Python Program for Program to find area of a circle

- Area of a circle can simply be evaluated using following formula.

$$\text{Area} = \pi * r^2$$

where r is radius of circle

```
# Python program to find Area of a circle
```

```
PI = 3.142
```

```
Area= PI * (r*r)
```

```
print("Area is", Area));
```

Python Program for compound interest

- Formula to calculate compound interest annually is given by:

$$\text{Compound Interest} = P(1 + R/100)^T$$

Where,

P is principle amount

R is the rate and

T is the time span

```
# Python program to find compound interest for given values.
```

```
CI = principle * (pow((1 + rate / 100), time))
```

```
print("Compound interest is", CI)
```

Python in-Built Math Functions

- Evaluating some complex mathematical operations like trigonometric operations, logarithmic operations, etc.
- The **math** module is the standard module in Python, and it is always available to work with complex scientific calculations.
- If we want to use mathematical functions under the math module, first, you have to import the module using **import math** statement in your program. After that, you can call any method of that module.
- The math module does not support the complex data types. For that, you can use the **cmath module** for the complex counterpart.

Examples: PI

```
# calculating PI  
import math  
print('The value of PI:', math.pi)
```

Output: The value of PI: 3.141592653589

Python Power Function

```
pow(x,y,z)
```

The pow() method returns the value of x to power of y (x^y).

x A number, the base

y A number, the exponent

z Optional. A number, the modulus

```
# calculating power
```

```
import math
```

```
print(math.pow(2, 3))
```

Output:

8.0

Python Math Trigonometric Functions

- All the trigonometric functions are available in python math module, so you can easily calculate them using `sin()`, `cos()`, `tan()`, `acos()`, `asin()`, `atan()` etc functions.
- Also ,you can convert angles from degree to radian and radian to degree.

```
# Trig functions
import math
angleInDegree = 90
angleInRadian = math.radians(angleInDegree)
print('The given angle is :', angleInRadian)
print('sin(x) is :', math.sin(angleInRadian))
print('cos(x) is :', math.cos(angleInRadian))
print('tan(x) is :', math.tan(angleInRadian))
```

```
The given angle is : 1.5707963267948966
sin(x) is : 1.0
cos(x) is : 6.123233995736766e-17
tan(x) is : 1.633123935319537e+16
```

Class Exercises

1. Write a Python program which accepts the radius of a circle from the user and compute the area.

```
from math import pi
r = float(input ("Input the radius of the circle :"))
print ("The area of the circle with radius " + str(r) + " is: " + str(pi *
r**2))
```

2. Write a Python program that will accept the base and height of a triangle and compute the area.

```
b = int(input("Input the base : "))
```

```
h = int(input("Input the height : "))
```

```
area = b*h/2
```

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

3. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.

```
fname = input("Input your First Name : ")  
lname = input("Input your Last Name : ")  
print ("Hello " + lname + " " + fname)
```

ASSIGNMENT

1. Write a python program that calculates the factorial of a number
2. Write a python program that computes the real roots of a quadratic equation.

Python Lecture

Functions and Modules

CSC 201

WEEK 6 LECTURE

Modular Programming

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

Advantages of Modular Programming

- Less code has to be written.
- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily because a small team deals with only a small part of the entire code.
- Modular programming allows many programmers to collaborate on the same application.
- The code is stored across multiple files.
- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.
- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

Functions

- A function is a device that groups a set of statements so they can be run more than once in a program(a packaged procedure invoked by name).
- Functions are also the most basic program structure python provides for maximizing code reuse.
- Functions could also be called subroutines and procedures in other programming environments.

Why do we need functions?

- **Maximizing code reuse and minimizing redundancy:** because a function allows us to code an operation in a single place and use in many places. Hence reduce code redundancy in our programs and thereby reduce maintenance effort.
- **Procedural decomposition**-Functions provide a tool for splitting systems into pieces that have well defined roles i.e it allows to debug the parts one at a time and then assemble them into a working whole.
- Well designed functions are often useful for many programs, i.e once you write and debug one you can reuse it.
- Makes program easy to read and debug.
- Reduce size of code by eliminating repetitive code (i.e later if you make a change ,you only have to make it in one place.

Defining Functions in Python

Functions in python are defined using the **def statement**

Function blocks begin with the keyword `def` followed by the function name and parentheses `()`.

Any input parameters or arguments should be placed within these parentheses.

You can also define parameters inside these parentheses.

The code block within every function starts with a colon `:` and is indented.

The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

The def statement creates a function object and assigns it to a name

```
def functionname( (parameters)arg1, arg2, arg3,.....,argN):  
    Statements
```

Function bodies often contain a return statement:

```
def functionname(arg1,arg2,.....,argN):  
    .....  
    return [expression]
```

Note: Two types of functions are available in python: built in such as dir(), len() or abs() and user defined functions.

```
def fun().....# Create function object
fun ()        # Call object
fun.attr=value # Attach attributes
```

```
e.g def times(x,y):      # Create and assign function
    return x*y          # Body executed when called
```

```
times(2,4)             # Arguments in parentheses
returns 8
```

```
x=times(3.14,4)        # Save the result object
```

```
x=12.56
```

```
and
```

```
>>> times('Ni',4)
```

```
NiNiNiNi
```

Functions types

- always available for usage
- those contained in external modules
- programmer defined

We also have built in functions

- `>>>type(32)` `>>>Int (-2.3)`
- `-<type 'int'>` `-2`
- `>>>float (32)`
- `32.0`
- `>>>int('Hello')`
ValueError, Invalid literal for Int(): Hello
- `>>> int (3.99999)`
- `3`

The following function takes a string as input parameter and prints it on a standard screen

Function definition is here

```
def printme(str):
```

```
    “ This prints a passed string into this function”
```

```
    print (str)
```

```
    return
```

Now you can call printme function

```
printme(“ This is first call to the user defined function!”)
```

```
printme(“ Again second call to the same function”)
```

When the above is executed it produces the following result

```
This is first call to the user defined function!
```

```
Again second call to the same function
```


Local and Global scope

Variables that are defined inside a function body have a local scope and those defined outside the function body have a global scope

#Global Scope

```
x=99
```

```
# x and func assigned in module: global
```

```
def func(y):
```

```
# y and z assigned in function: local
```

#Local Scope

```
z=x+y
```

```
# x is global
```

```
return z
```

```
e.g func(1)
```

```
# func in module:
```

```
result=100
```

- Global `x` is global because it is assigned at the top level of the module file
- Local `y`, `z`: `y` and `z` are local to the function (and exist only while the function runs) because they are both assigned values in the function definition
- `z` by virtue of the statement
- `y` because arguments are always passed by assignment

Example2

```
total=0 # this is a global variable
# Function definition is here
def sum(arg1,arg2):
# Add both parameters and return them.
total=arg1+arg2; # Here total is local variable
print("Inside the function local total:",total)
return total
# Now you can now call sum function
sum(10,20)
print(" Outside the function global total:", total)
```

Result

Inside the function local total:30

Outside the function global total:0

Categories of Functions

Fruitful functions- functions that yield results e.g math functions

Other functions that do not return a value are called void functions.

The return keyword is used to return value no return returns

None

e.g `def print-twice(bruce):`

`print bruce`

`print bruce`

```
>>> print_twice('Spam')
```

```
Spam
```

```
Spam
```

```
>>> print_twice(17)
```

```
17
```

```
17
```

```
>>> n = [1, 2, 3, 4, 5]
```

```
>>> def stats(x):
```

```
...     mx = max(x)
```

```
...     mn = min(x)
```

```
...     ln = len(x)
```

```
...     sm = sum(x)
```

```
    return mx, mn, ln, sm
>>> mx, mn, ln, sm = stats(n)
>>> print stats(n)
(5, 1, 5, 15)
>>>
>>> print mx, mn, ln, sm
5 1 5 15
```

Function arguments
single arguments
multiple arguments

Pass by Reference vs Value

All parameters (arguments) in the python language are passed by reference. It means that if you change what a parameter refers to within a function, the change also reflects back in the calling function.

Passing by reference means the called functions' parameter will be the same as the callers' passed argument (not the **value**, but the identity - the variable itself). **Pass by value** means the called functions' parameter will be a copy of the callers' passed argument. ... Java only supports **pass by value**.

```
def changeme(mylist):
```

```
    “ This changes a passed list into this function”
```

```
    print(“ Values inside the function before change:”, mylist)
```

```
    mylist[2]=50
```

```
    print(“ Values inside the function after change:”, mylist)
```

```
    return
```

if you now call changeme function

```
mylist=[10,20,30]
```

```
changeme(mylist)
```

```
print(“Values outside the function”, mylist)
```

```
Values inside the function before change:[10,20,30]
```

```
Values inside the function after change:[10,20,50]
```

```
Values outside the function[10,20,50]
```


Definitions

A method: Is a piece of code that is called by name that is associated with an object. In most respects it is identical to a function except for two key differences

It is implicitly passed for the object for which it is called

It is able to operate on data that is contained within the class

Pass by Value- Means copying the value from the argument variable into the parameter variable i.e Parameters are the placeholders for arguments.

Pass by Reference -means creating a new reference to variable value that refers to the argument variable and putting that in the parameter variable.

Parameter-A parameter is the variable which is part of the method's signature(method declaration)

Argument-An argument is an expression used when calling the method

Python Modules

A module allows you to logically organize your python code.

Grouping related code into a module makes the code easier to understand and use.

A module is a file consisting of python code. A module can define function, classes and variables .A module can also include runnable code.

What are modules for?

Python modules are used to organize Python code. For example, database related code is placed inside a database module, security code in a security module etc.

Smaller Python scripts can have one module. But larger programs are split into several modules.

Modules are grouped together to form packages.

Modules names

A module name is the file name with the .py extension.

When we have a file called empty.py, empty is the modulename.

The `__name__` is a variable that holds the name of the module being referenced.

The current module, the module being executed (called also the main module) and it has a special name: `'__main__'`. With this name it can be referenced from the Python code.

Example of a simple module support.py

```
def print_fun(par):  
    print "Hello:", par  
    return
```

You can use any python source file as a module by executing an import statement in some other python source file

The syntax is

```
import module1[,module2[,.....moduleN]
```

For example # import module support

```
import support
```

```
#Now you can call defined function
```

```
support.print_func("zara")
```

When the above code is executed it produces the following

Results. Hello: zara

Python's **from statement** lets you import specific attributes from a module into the current namespace

The from.....import has the following syntax

```
from modname import name1[,name2[,.....nameN]]
```

e.g to import the function Fibonacci from the module fib

#fibonacci numbers module

```
def fib(n):    #return Fibonacci series up to n
```

```
result=[]
```

```
    a, b =0, 1
```

```
    while b<n:
```

```
        result.append(b)
```

```
        a,b= b, a+b
```

```
    return result
```

```
>> from fib import fib
```

```
>>fib(100)
```

```
[1,1,2,3,5,8,13,21,34,55,89]
```

Hence this statement does not import the entire module fib into the current namespace; it just introduces the item Fibonacci from the module fib into the global symbol table of the importing module.

The from import* statement

It is also possible to import the names from a module into the current namespace by using

```
from modname import*
```

This provides an easy way to import all the items from a module into the current namespace; however this statement should be used sparingly.

Frequently used modules

sys Information about Python itself (path, etc.)

os Operating system functions

os.path Portable pathname tools

shutil Utilities for copying files and directory trees

cmp Utilities for comparing files and directories

glob Finds files matching wildcard pattern

re Regular expression string matching

time Time and date handling

datetime Fast implementation of date and time handling

doctest, unittest Modules that facilitate unit test

pdb Debugger

hotshot Code profiling

pickle, cpickle, marshal, shelve Used to save objects and code to files

getopt, optparse Utilities to handle shell-level argument parsing

math, cmath Math functions (real and complex) faster for scalars

random Random generators (likewise)

gzip read and write gzipped files

struct Functions to pack and unpack binary data structures

StringIO, cStringIO String-like objects that can be read and written as files (e.g., in-memory files)

types Names for all the standard Python type

The main difference between a module and a package is that a package is a collection of modules AND it has an `__init__.py` file.

`myMath/`

`__init__.py`

`adv/`

`__init__.py`

`sqrt.py`

`add.py`

`subtract.py`

`multiply.py`

`divide.py`



FLOW CONTROL

Dr. F. O Asahiah

Department of Computer Science and Engineering
Obafemi Awolowo University, Ile-Ife.



Flow Control (Introduction)

- Programming:
 - i. accomplishing tasks
 - ii. with instructions to computer
 - iii. Communicated in a programming language
- Tasks often involves
 - a) Execution of sequence of instructions
 - b) Repetition of actions
 - c) Decisions on next steps of actions
 - d) Often a mixture of (a) - (c)



What is Flow Control?

- Fundamental component of all programming languages
 - Python
 - Java
 - Javascript
 - etc
- Enable us to dictate the flow of execution within a program.
 - **What statement** (or set of statements) is executed ?
 - **How many times** should a statement or set of statements be executed?
 - **What happens next** after the execution of current statement(s)?



Importance of Control Statements

- **Control statements:**
 - specific keywords
 - used within control structures
 - to modify the flow of execution
- **Importance**
 - **Flexibility:** dictate how program behaves based on certain conditions or occurrence of specific events.
 - **Decision Making:** execute different code blocks under varying circumstances
 - **Code Efficiency:** creating dynamic and responsive programs.

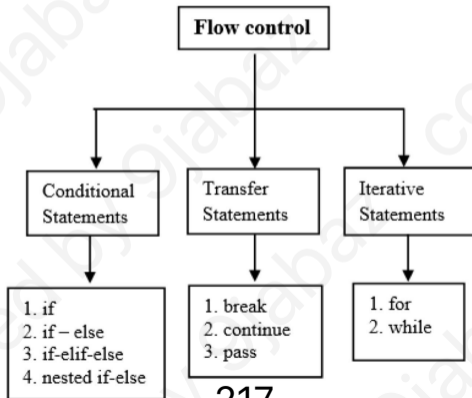


Flow Control Statements in Python

- Different programming languages
 - have different details and implementations
 - most of them accomplish all the flow control found in Python
- In Python, flow control is of **three types**:
 - a) Conditional statements
 - b) Iterative statements.
 - c) Transfer statements
 - d) A fourth type of control: Error handling; normal treated separately



Types of Flow Control in Python





Fourth type of Control

- Error Handling
 - Not normally treated under control structure
 - Affects also the execution of statements
 - Will be treated separately in this course



Type 1: Conditional Control

- Conditional Control or Conditional structure:
 - use of **conditional keywords** with variables in various arrangement
 - to facilitate **selection of block of codes** to executed based on
 - whether **certain condition(s)** is/are **true or false**
 - Also know as **selection control**
 - Opposite to **sequential control**
- The common conditional/selection constructs are:
 - **If**
 - **If-else**
 - **If-elif-else** (pronounced **if-else-if-else**)
 - **Nested if**



Conditional: If Statement

- This is the simplest form of conditional statement.
- It checks a condition and execute the associated block if the condition is true
- Otherwise, it skips that associated block of code
- **Syntax:**
 - *if* <condition>:
code block to execute if condition is true
 - where
 - **Keyword:** *if*
 - <condition>: that which can logically evaluate to **True** or **False**
 - **# code block to execute if condition is true:** what to execute if <condition> is **True**



If Statement flowchart

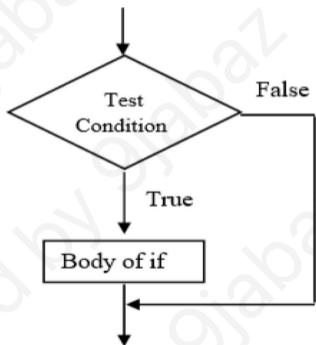


Fig. Flowchart of if statement



Examples for if conditional

- Example 1: Printing the square of a number if it is greater than 5

```
number = 6
```

```
if number > 5:
```

```
    # Calculate square
```

```
    print(number * number)
```

```
print('Next lines of code')
```



Examples for if conditional

- Example 2: Printing the admission status of jamb candidate based on age

```
cand_age = int(input("supply the age of intending candidate"))
if cand_age < 18:
    # Print candidate status
    print("person not qualified to for admission")
print('Processing of admission will progress')
```



Examples of Conditional if

- Example 3

```
letter = "t"
```

```
if letter in 'Python':
```

```
    print("Yes")
```

- Note that:

- A colon is always at the end of the condition
- The code block may be one line or several lines
- The code block is identified by the indentation.



Conditional: If-else Statement

- This is the second form of conditional statement.
 - checks the condition and executes block associated with the if construct if condition evaluate to True
 - Otherwise, it executes an alternate block of code associate with the else construct.
 - It can only execute only one of the blocks. Either the first or the second.
- **Syntax:**
 - *if* <condition>:
 # first code block to execute if condition is true
 - else*:
 # second code block to execute if condition is false
- where
 - **Keywords:** *if, else*
 - <condition>: that which can logically evaluate to *True* or *False*
 - **# first code block to execute if condition is true:** what to execute if <condition> is True
 - **#second code block to execute if condition is false:** what to execute if <condition> is False

If-else Statement flowchart

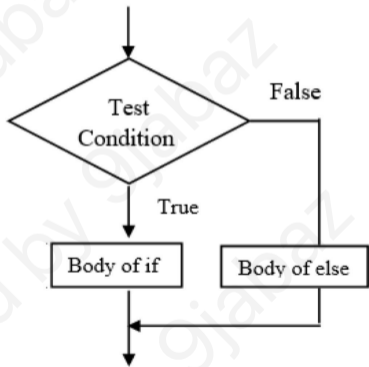


Fig. Flowchart of If-else



Example 1 for if-else

- Checking if password is correct

```
password = input('Enter password ')
```

```
if password == "PYnative@#29":
```

```
    print("Correct password") # first code block
```

```
else:
```

```
    print("Incorrect Password") # second code block
```



Example 2 for if-else

- Comparing hecking if password is correct

```
x = 10
```

```
if x > 15:
```

```
    print("x is greater than 15")
```

```
else:
```

```
    print("x is not greater than 15")
```



Conditional: If-elif-else Statement

- This is the third form of conditional statement.
 - checks the condition1 and executes block associated with the if construct if condition2 evaluate to True
 - Otherwise, check condition2 and executes block associated with the elif construct if condition2 evaluate to True
 - Otherwise, it executes an alternate block of code associate with the else construct.
 - It can only execute only one of the blocks. Either the first or the second.
- Note:
 - as many elif as needed is allowed between the starting if and the closing else.
 - Each elif has a condition that must be tested.
- **Syntax:**
 - `if <condition1>:`
first code block to execute if condition is true
 - `elif <condition2>:`
second code block to execute if condition2 is true
 - `.`
 - `.`
 - `.`
 - `elif <condition_m>:`
m code block to execute if condition_m is true
 - `else:`
m+1 code block to execute if all previous conditions are false



If-elif-else continued

- where
 - **Keywords:** *if, else*
 - **<condition>**: that which can logically evaluate to *True* or *False*
 - **# first code block to execute if condition is true**: what to execute if **<condition>** is *True*
 - **#second code block to execute if condition is false**: what to execute if **<condition>** is *False*



If-elif-else example

- Example to compare

```
a = 33
```

```
b = 33
```

```
if b > a:
```

```
    print("b is greater than a")
```

```
elif a == b:
```

```
    print("a and b are equal")
```

```
else:
```

```
    print("a is greater than b")
```



Nested If

- Nested if occurs when an if control is put inside another if control . The second if control becomes part of the code block for the first first if control structure.



Examples

```
temperature = 30
weather = "sunny"

if temperature > 25:
    if weather == "sunny":
        print("It's a perfect day for the beach!")
    elif weather == "cloudy":
        print("It's warm but might be a bit gloomy.")
    else:
        print("Stay hydrated, it's hot outside!")
else:
    if weather == "rainy":
        print("It's cool and rainy, a good day to stay indoors.")
    else:
        print("It's a bit chilly, dress warmly!")
```



Example 2: Nested If

```
score = 85

if score >= 90:
    grade = 'A'
    if score >= 95:
        comment = "Excellent!"
    else:
        comment = "Very Good!"
elif score >= 80:
    grade = 'B'
    if score >= 85:
        comment = "Good Job!"
    else:
        comment = "Well Done!"
else:
    grade = 'C'
    comment = "Keep Trying!"

print(f"Grade: {grade}, Comment: {comment}")
```


9JABAZ

Want more books?

Visit 9jabaz.ng and download for free!!