Unit I: Introduction to Basic Computer Programs

Unit I: Introduction to Basic Computer Programs
1.1 Basic Computer Programs; Computational Problems and Algorithms
1.2 Language Compilation Process
1.3 Binary Representation
1.4 Namespace, identifiers, variables, constants, arithmetic operators
1.5 Logical & Conditional Expression

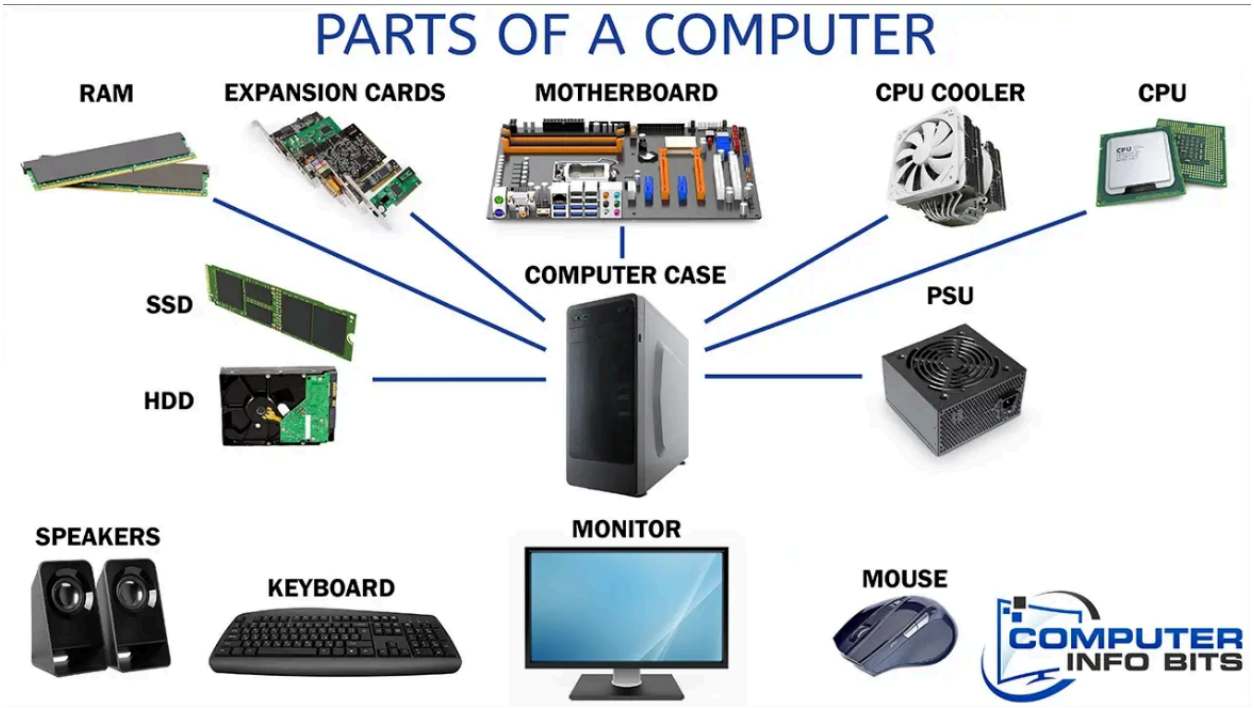
1.1 Basic Computer Programs; Computational Problems and Algorithms

- **Computer basics**

  What is a computer?

  At the end of a day, a computer is a piece of hardware that is connected to a power supply. How come so many things are possible? Like watching movies, playing video games, browsing the internet, and editing documents.

  The home desktop computer or your laptop can be divided into:



  PSU: Power Supply Unit

  It receives the power from the wall plug and distributes it to the various components.

  CPU: The Central Processing Unit is the component that will perform your instructions and computations.

RAM: For the CPU to work, it needs to store the data that it is working with somewhere. It is stored in the RAM (Random Access Memory)

Input Devices: These are devices that provide input from the user to the computer
Mouse
Keyboard
Output devices: These are the devices that the computer shows the output to interact with the user.
Speakers - Output in the form of audio.
Monitor - Output in the form of pictures.
Storage (SSD, HDD) - output in the form of a saved/stored file

What makes it capable of performing a wide range of tasks is its ability to execute software instructions.

- Software: Computers run software, which are sets of instructions that tell the hardware what to do.

- Operating System: manages the computer's resources and provides a platform for running applications.

- Applications: These are the programs that users interact with to perform specific tasks like watching movies, playing games, browsing the internet, editing documents, etc.

- Hardware: The physical components of the computer, such as the CPU (Central Processing Unit), GPU (Graphics Processing Unit), memory (RAM), storage (hard drive, SSD), input/output devices (keyboard, mouse, monitor, etc.), provide the necessary resources for executing software instructions and interacting with users and other devices.

- Connectivity: Computers can also communicate with other computers and devices over networks, such as the internet.

In a computer world, everything is represented using **binary digits (bits): 0 and 1.**
So how can we humans that work with numbers and texts and images communicate / work with computers that can understand just 0s and 1s?

Abstraction to the rescue:
We need computers that only work with thousands of 0s and 1s to understand how we humans work - the numbers, texts, images, and videos. We first find a way to represent our human numbers with 0s and 1s. Luckily computers have millions of switches / transistors that we can work with. Think of each switch as light bulbs. Turning on and off.

1 means the bulb has turned on. 0 means the bulb has turned off.

For humans we have 10 symbols: 0,1,2,3,4,5,6,7,8,9
After our symbols have been exhausted, we start again with an extra digit:
10, 11, 12, 13, 14, 15, 16, 17 … 99

When the extra two digits are exhausted, we again add another digit: 100, 101, 102, … 999
Similarly, with computers, when you exhaust a digit you add another one just like we do but it is with 0 and 1

Example:
We can say
0 in a computer is 0 for a human
1 in a computer is 1 for a human.
Now, the symbols for computers have been exhausted. Remember? It only works with 0s and 1s. When the symbols are exhausted, we add another one!

10 in a computer is 2 for a human
11 in a computer is 3 for a human

This is exactly how a computer can know when a human is typing 7 or 8 in his calculator program even if all a computer can understand is 0s and 1s. It has many switches (0s and 1s) to work with. And it is fast because electricity is fast.

But how can you represent text in a computer? Let's say you want to store ‘Hello’ in a computer. How can you do it?
This is how the people in the US represented texts in computers.
They call it ASCII.

A is 65	a is 97
B is 66	b is 98
.	.
.	.
.	.
Z is 90	z is 122

So When you are typing Hello.
Under the hood:

Text → ASCII → Binary (Computer World)
Hello → 72 101 108 108 111 → 01001000 01100101 01101100 01101100 01101111

In this way, we can forget about the 0s and 1s that the computers work at. We can work at a higher level of abstraction of just typing the text even if all a computer sees are 0s and 1s. This is the power of abstraction.

Computer Programs:
Despite all the wonderful and amazing things that computers can do, computers are not very smart. In fact, they are pretty dumb. Computers can only follow the instructions given by a human. A computer program is a sequence of step-by-step instructions given to computers to manipulate data in order to perform a certain task.
Programming is the art and science of writing a computer program.

If you take a calculator app, it is a program. It will take your number and symbols and will display the output on your screen.

How computer executes a program
In order to learn how to write a program, it is important to have an overview of how a computer executes a program.

The two important components of a computer are the CPU, or central processing unit and the memory, which is a generalized term for where we store the data to be processed or manipulated by the CPU, as well as the instructions to do so. A memory location is addressable using a memory address.

The instructions to the CPU come in the form of machine code, a sequence of bits (1s and 0s) that is interpreted and then followed by the CPU to do certain things. This machine code could instruct the CPU, for instance, to compare if a particular number at a certain memory location is more than 0, to add one number to another, or to execute another instruction at another memory location. The data stored in the memory is also stored as a sequence of 1s and 0s.

Writing Programs
You write a program with a programming language.
Some of the programming languages are: C, Python, Go, Java, Kotlin.

The different programming languages were created to solve different needs but at the end of the day, you are writing instructions for the computer that will at the end be run in the CPU.

Python
Python was built to be easier to use than other programming languages. It's usually much easier to read Python code and much faster to write code in Python than in other languages.
For instance, here's some simple code written in C, another commonly used programming language:

```
#include <stdio.h>


int main(void)
{
    printf("Hello, world\n");
}
```

All the program does is show the text Hello, world on the screen. That was a lot of work to output one phrase! Here's the same program, written in Python:

```
print("Hello, world")
```

That's pretty simple, right? The Python code is faster to write and easier to read. We find that it looks friendlier and more approachable, too! At the same time, Python has all the functionality of other languages and more. You might be surprised how many professional products are built on Python code: Instagram, YouTube, Reddit, Spotify, to name just a few.
To print text to the screen in Python, you use the print() function. A function is a bit of code that typically takes some input, called an argument, does something with that input, and produces some output, called the return value.

Computational problems and Algorithms

A computational problem is a problem that can be solved step-by-step with a computer. These problems usually have a well-defined input, constraints, and conditions that the output must satisfy. Here are some types of computational problems:

A decision problem is one where the answer is yes or no. For instance, "given a number n, is n even?" is a decision problem. Some decision problems take more steps to solve than others. For instance, "given a number n, is n prime?" takes more steps than just checking the parity of a number.

A search problem is one where the solution consists of one or more values that satisfies a given condition. For instance, we may want to compute a path from one geographical location to another on a map.

A counting problem is one where the answer is the number of solutions to a search problem.

An optimization problem is one where the solution is the "best" possible solution, where the "best" can be defined in a different way. For instance, we may want to compute the fastest route from one location to another.

Questions such as "what is the meaning of life?" "do I look good in this outfit?" are not computational problems, because they do not have well-defined input, constraints, and conditions that the output must satisfy.

Example: Finding the maximum
Let's start with a simple problem. Given a finite list L of k integers (k > 0), find the integer with the maximum value from the list.

First, let's consider if this is a computational problem. The input is very well defined. We know what an integer is. We are told we have at least one, and we have a finite number of them2.

Second, let's consider the output. What conditions must the output satisfy? First, it has to be equal or larger than every other integer on the list. Second, it must be an integer in the list. This is well defined by the problem statement, so we can say that it is a computational problem.

Here is an example. Suppose the input consists of:

4 1 -4 0 9 9 3 5 8

The output should be 9.

Now, you should pause reading and think about how you would solve this step-by-step.

Algorithm
One way to solve this problem is to check through the integers in the list, one-by-one, and keep track of the maximum value so far. When you reach the end of the list, your "maximum value so far" will also be the maximum for the whole list.

Let's look at an example:

Integers Scanned	Maximum So Far
4	4
4 1	4
4 1 -4	4
4 1 -4 0	4
4 1 -4 0 9	9
4 1 -4 0 9 9	9
4 1 -4 0 9 9 3	9
4 1 -4 0 9 9 3 5	9
4 1 -4 0 9 9 3 5 8	9

An algorithm is any well-defined computational procedure that takes
some value, or set of values, as input and produces some value, or set of values, as output in a ûnite amount of time. An algorithm is thus a sequence of computational steps that transform the input into the output.

An algorithm is like a recipe from a cookbook,

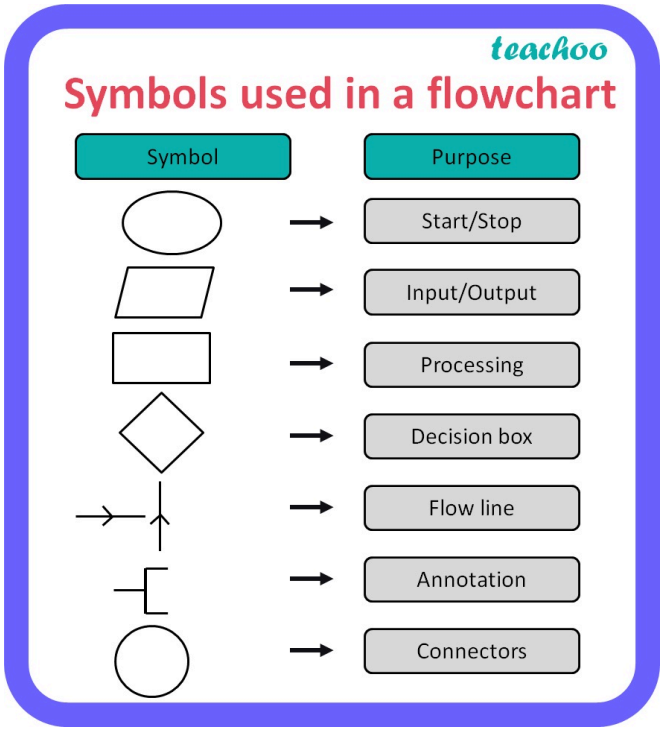
- For example, for making a cup of tea:

```
Organise everything together;
Plug in kettle;
Put teabag in cup;
Put water into kettle;
Wait for kettle to boil;
Add water to cup;
Remove teabag with spoon/fork;
Add milk and/or sugar;
Serve;
```

Design an algorithm to add two numbers and display the result.

step 1 − START
step 2 − declare three integers a, b & c
step 3 − define values of a & b
step 4 − add values of a & b
step 5 − store output of step 4 to c
step 6 − print c
step 7 − STOP

Flowchart

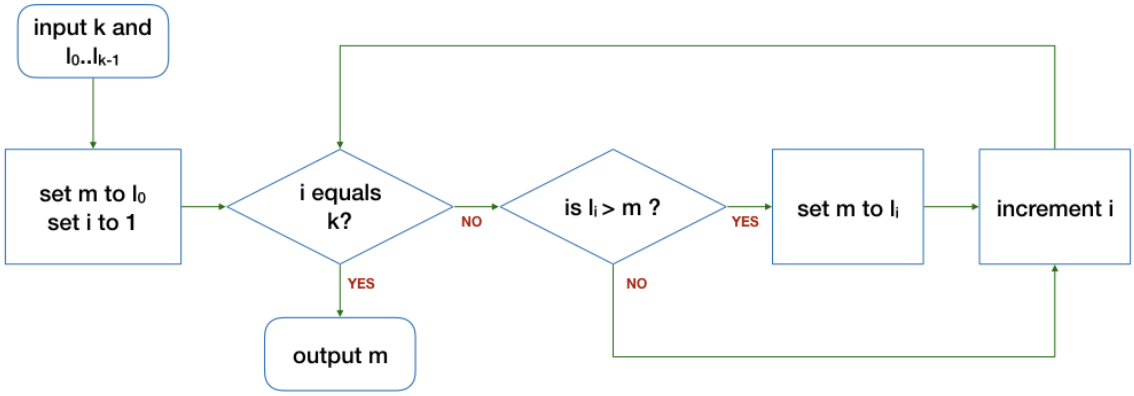


There are different ways one can describe an algorithm. The easiest way I find is to use a diagram called a flowchart. The flowchart for the algorithm above looks like this. A

diamond box represents a "question" that can be true or false (yes or no), and we trace through the "flow" step by step, following the corresponding path depending on the answer.

The input consists of:

4 1 -4 0 9 9 3 5 8



Please spend some time to trace through the walkthrough above. The snapshot of the values of the *i, l_i, k*, and *m*, at the point after "is *i* equals *k*" is shown in the table below.

All

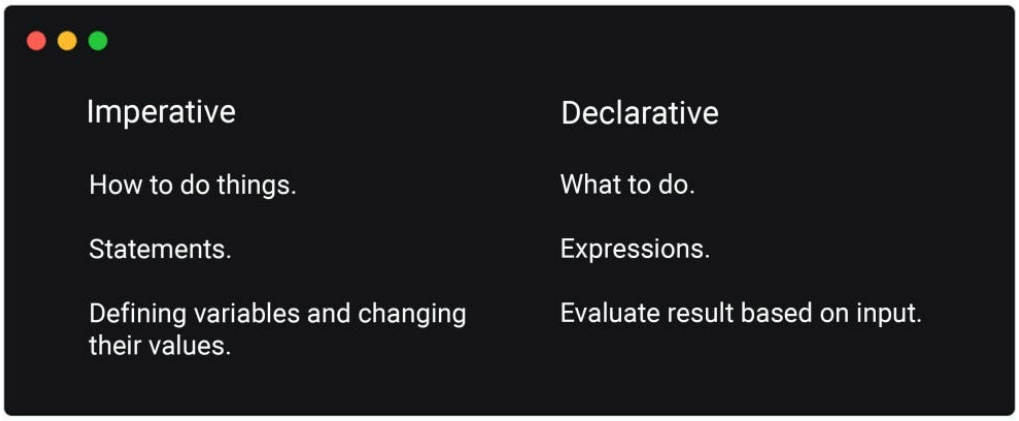
Integers scanned	**i**	**l_i**	**k**	**Maximum so far (m)**
4	0	4	9	4
4 1	1	1	9	4
4 1 -4	2	-4	9	4
4 1 -4 0	3	0	9	4
4 1 -4 0 9	4	9	9	4
4 1 -4 0 9 9	5	9	9	9
4 1 -4 0 9 9 3	6	3	9	9
4 1 -4 0 9 9 3 5	7	5	9	9
4 1 -4 0 9 9 3 5 8	8	8	9	9
4 1 -4 0 9 9 3 5 8	9		9	9

knowledge can be thought of as either **declarative** or i**mperative**.

Declarative knowledge is composed of statements of fact. Declarative knowledge don't give us to explore beyond the truth.If something is true or facts then you have to believe that is the way it is.e.g 2+2=4 is true but let us look at underneath the hood i.e how number like 2 and 4 existed.

Declarative languages allow the programmer to specify what they want, and not concern themselves with the implementation, or how it's done.It's a limited subset of programming and doesn't apply to most languages like Python or Java.

> Imperative knowledge is the knowledge based on"how to knowledge or information" .It is like a recipe or a cookbook .i.e All the steps are well and clearly defined.
>
> *Imperative code tells the computer how to do things and declarative code focuses on what you want from the computer.*



- syntax
  Syntax refers to the rules that define the structure of a language. Syntax in computer programming means the rules that control the structure of the symbols, punctuation, and words of a programming language.

  Without syntax, the meaning or semantics of a language is nearly impossible to understand.

  For example, a series of English words, such as - subject a need and does sentence a verb - has little meaning without syntax.

  Applying basic syntax results in the sentence - Does a sentence need a subject and verb?

Programming languages function on the same principles. If the syntax of a language is not followed, the code will not be understood by a compiler or interpreter.

Compilers convert programming languages into binary code that computers can understand. If the syntax is incorrect, the code will not compile.

Interpreters execute programming languages such as Python at runtime. The incorrect syntax will cause the code to fail. That's why it is crucial that a programmer pays close attention to a language's syntax. No programmer likes to get a syntax error.

- Python syntax

  Indentation

  Indentation refers to the spaces at the beginning of a code line.
  Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.

```
if 5 > 2:
  print("Five is greater than two!")
```

  Python will give you an error if you skip the indentation:

  Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

  The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

  You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

  Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
       print("Five is greater than two!")
```

1.2 Git, Version Control, Basic Linux Commands

- Git: A distributed Version Control System (VCS), which means it is a useful tool for easily tracking changes to your code, collaborating, and sharing. With Git you can track the changes you make to your project so you always have a record of what you've worked on and can easily revert back to an older version if need be. It also makes working with others easier—groups of people can work together on the same project and merge their changes into one final source!

- GitHub: A way to use the same power of Git all online with an easy-to-use interface. It's used across the software world and beyond to collaborate and maintain the history of projects.

  GitHub is home to some of the most advanced technologies in the world. Whether you're visualizing data or building a new game, there's a whole community and set of tools on GitHub that can get you to the next step.

- Understanding the GitHub flow
  The GitHub flow is a lightweight workflow that allows you to experiment and collaborate on your projects easily, without the risk of losing your previous work.

  Repositories:
  A repository is where your project work happens--think of it as your project folder. It contains all of your project's files and revision history. You can work within a repository alone or invite others to collaborate with you on those files.

  Cloning:
  When a repository is created with GitHub, it's stored remotely in the cloud. You can clone a repository to create a local copy on your computer and then use Git to sync the two. This makes it easier to fix issues, add or remove files, and push larger commits.

  You clone a repository with `git clone <url>`. For example, if you want to clone the Git linkable library called lib1, you can do so like this:

  ```
  $ git clone https://github.com/lib1/lib1
  ```

  Committing and pushing
  Committing and pushing are how you can add the changes you made on your local machine to the remote repository in GitHub.

  The `git add` and `git commit` commands compose the fundamental Git workflow. After editing your files in the working directory. When you're ready to save a copy of the current state of the project, you stage changes with `git add`

In addition to `git add` and `git commit`, a third command `git push` is essential for a complete Git workflow. `git push` is utilized to send the committed changes to remote repositories for collaboration. This enables other team members to access a set of saved changes.

In conjunction with these commands, you'll also need `git status` to view the state of the working directory and the staging area.

- **GitHub terms to know**

  Repositories:
  Repositories also contain READMEs. You can add a README file to your repository to tell other people why your project is useful, what they can do with your project, and how they can use it.

  Branches
  You can use branches on GitHub to isolate work that you do not want merged into your final project just yet. Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.

  Fork
  A fork is another way to copy a repository, but is usually used when you want to contribute to someone else's project. Forking a repository allows you to freely experiment with changes without affecting the original project and is very popular when contributing to open source software projects.

  Pull requests
  When working with branches, you can use a pull request to tell others about the changes you want to make and ask for their feedback. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add more changes if need be.

  Issues
  Issues are a way to track enhancements, tasks, or bugs for your work on GitHub. Issues are a great way to keep track of all the tasks you want to work on for your project and let others know what you plan to work on.

  Your user profile
  Your profile page tells people the story of your work through the repositories you're interested in, the contributions you've made, and the conversations you've had. You can also give the world a unique view into who you are with your profile README.

- **Basic Commands**

  A command line is an interface that accepts lines of text and processes them into instructions for your computer.

  Any graphical user interface (GUI) is just an abstraction of command-line programs. For example, when you close a window by clicking on the "X," there's a command running behind that action.

  `ls`- list the contents of the directory you want (the current directory by default), including files and other nested directories.

  cd- switches you to the directory you're trying to access. '`cd`': go to home folder, '`cd ..`' : move a level up, '`cd -`' : Return to the previous directory

  ```
  cd <directory name>
  ```

  `rm`- remove files and directories.

  ```
  rm <file name>
  ```

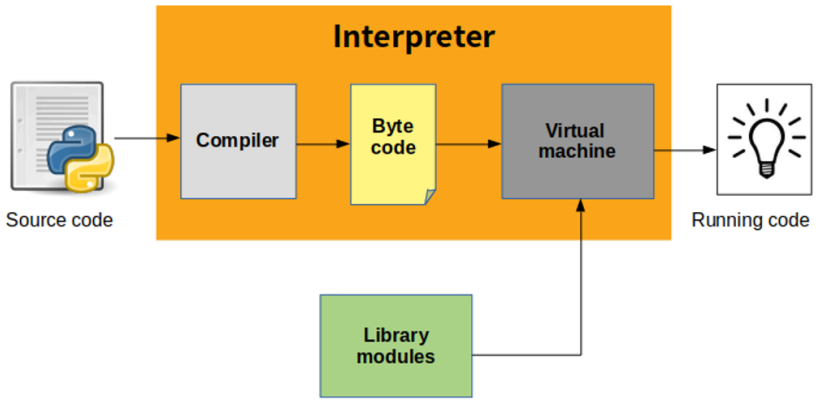
  `mkdir`- create folders in the shell

  ```
  mkdir <directory name>
  ```

  `sudo`- act as a superuser or root user while you're running a specific command. It's how Linux protects itself and prevents users from accidentally modifying the machine's filesystem or installing inappropriate packages.

1.3 Language Compilation Process

A compiler for a given source language and machine translates a source program into an equivalent program (called the object program) written in the machine's native language. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



Source code: The Python compiler reads a Python source code or instruction in the code editor. In this first stage, the execution of the code starts.

Step 2: After writing Python code it is then saved as a .py file in our system. In this, there are instructions written by a Python script for the system.

Byte code: source code is converted into a byte code. Python compiler also checks the syntax error in this step and generates a .pyc file.

VM: Byte code that is .pyc file is then sent to the PVM which is the Python interpreter. PVM converts the Python byte code into machine-executable code and in this interpreter reads and executes the given file line by line. If an error occurs during this interpretation then the conversion is halted with an error message

Step 5: Within the PVM the bytecode is converted into machine code that is the binary language consisting of 0's and 1's. This binary language is only understandable by the CPU of the system as it is highly optimized for the machine code.

Step 6: In the last step, the final execution occurs where the CPU executes the machine code and the final desired output will come as according to your program.

1.4 Namespace, Identifiers, Variables, Constants, Arithmetic Operators

- Namespace and Identifiers
  Namespace is a container for a set of identifiers. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.
  Identifiers are names used for variables, functions, etc.

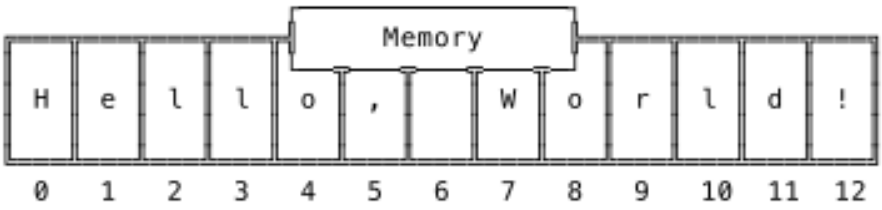
```
# global namespace
var_a = 75
def A_func():

    # local namespace
    var_b = 38
    def B_inner_func():

        # nested local
    # namespace
        var_c = 24
```

- Variables
  Your computer will be storing data in memory. The CPU will use that data whenever it needs it during execution. Remember that everything is 0s and 1s. You need to specify the data and access it in your programs.

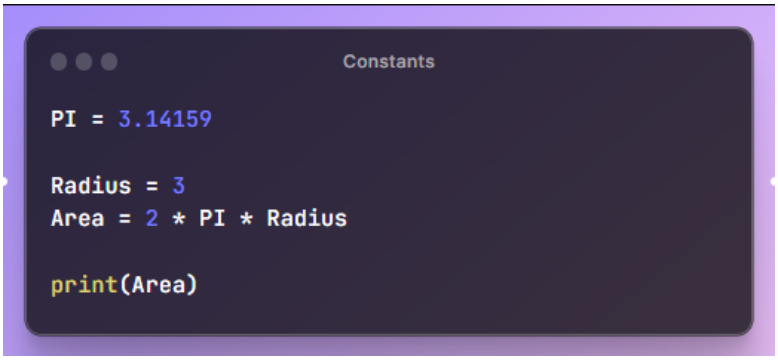
  To do it meaningfully, you give the data a name using the assignment operator '='

```
First Program

text = 'Hello World'
X = 10
Y =  15

# Read it as: Now, the new value of X is X + 5
X = X + 5 # X is being reassigned
print(X)
```

  A variable is a reference to the data which is stored in the memory so that you don't have to remember the lower level details of memory. When you are writing a program you don't want to think that the text 'Hello, World!' is stored in the memory location from 0 till 12. You just want to define a variable and not think about the lower-level memory details.



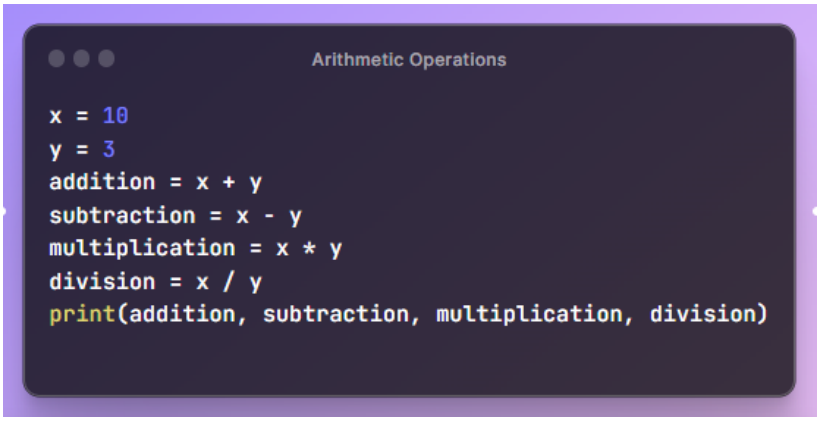
- Constants
  They can be thought of as variables but variables that cannot be changed once assigned. Say you are creating a calculator game and you don't want the value of PI to change. Not even by mistake in your code. That is why constants are needed.



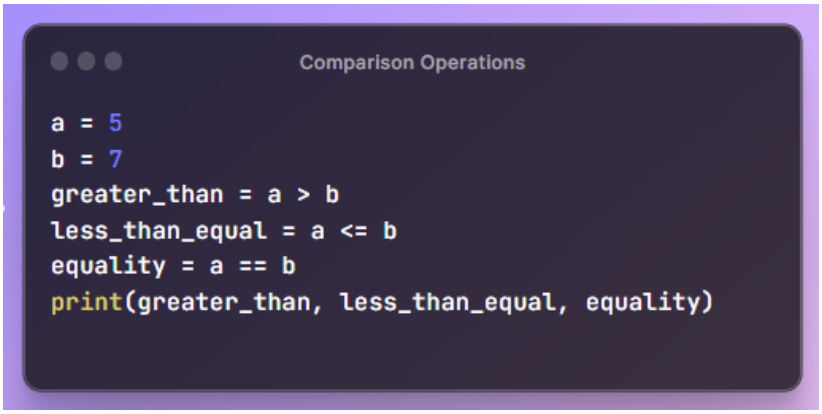
- Operators
  In your programs, you want the computer to be able to do operations on the data; basically, calculations.

  The assignment of a variable is one of the operations, it uses the '=' operator.
  You can perform other kinds of operations on your data like:
  Arithmetic
  Comparison
  Logical

- Arithmetic operations:
  When your data and variables are numerical, you can perform arithmetic operations on the data. These operations include addition, subtraction, multiplication, division among other operations.



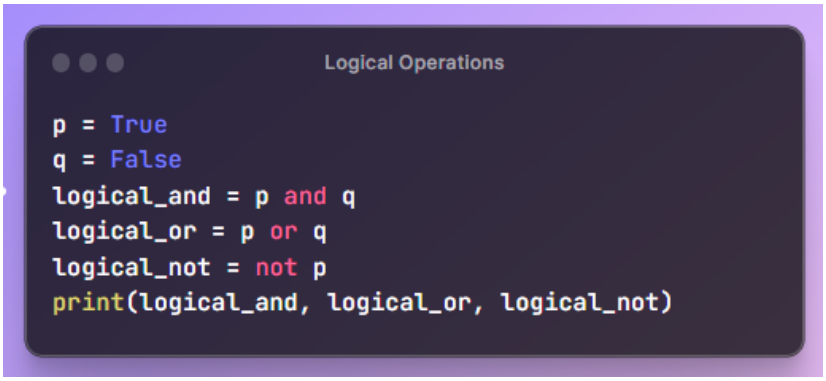
- Comparison operators:
  You can also perform comparison on your data checking to see if it is equal, greater than or less than using the comparison operators <, >, ==.



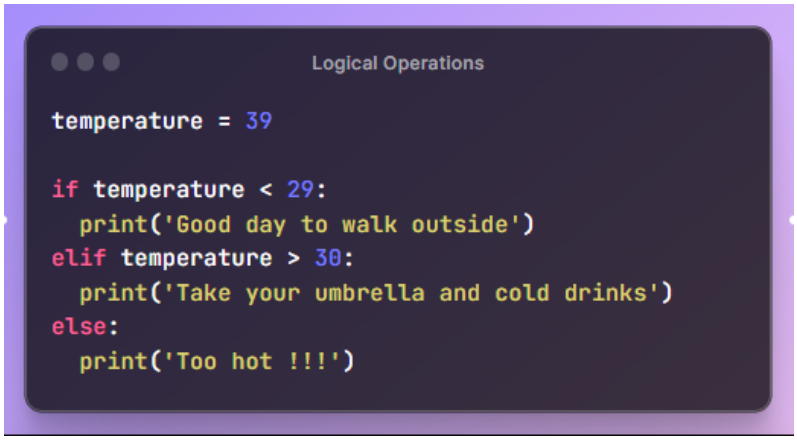
Note: Since the equal symbol '=' has been taken by the assignment operator, to compare different variables if they are equal, we use the double equal symbol: '=='

1.6 Logical & Conditional Expression

- Logical operators:
  With comparison operators, you can get if one value is greater than the other one and get the results as True or False. If you want to combine multiple comparisons, you can use the logical operators 'and', 'or', and 'not' operators to combine that suits your needs.



- Conditional Operators
  With comparison and logical operators, you can ask if a given condition is true or false. With this in hand, you can instruct the computer to perform a specific task if and only if a given condition is true or false. With this, you can create conditions and be able to ask the computer to perform certain actions if a condition is met.


Logical Operations
temperature = 39
if temperature < 29:
print('Good day to walk outside')
elif temperature > 30:
print('Take your umbrella and cold drinks')
else:
print('Too hot !!!')

Reference

Computational Problem & Algorithms - CS1010 Programming Methodology. (n.d.).

Atlassian. (n.d.). Basic Git Commands | Atlassian Git Tutorial.

Python, R. (2024, January 18). Conditional statements in Python.

Bader, D., Jablonski, J., & Heisler, F. (2021). Python Basics: A Practical Introduction to Python 3. Real Python (Realpython.Com).