

# Python

## Programming

A/L ICT – Lesson 09



What can Python do?

- ✔ Python can be used on a server to create web applications.
- ✔ Python can be used alongside software to create workflows.
- ✔ Python can connect to database systems. It can also read and modify files.
- ✔ Python can be used to handle big data and perform complex mathematics.
- ✔ Python can be used for rapid prototyping, or for production-ready software development.

## Competency 9 : Develops algorithms to solve problems and uses python programming language to encode algorithms

### 9.1 Uses problem-solving process

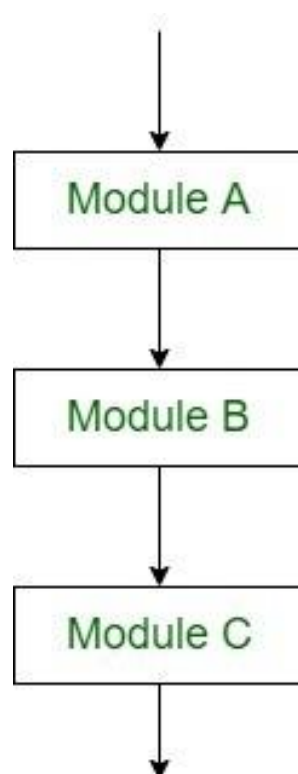
**An algorithm** is a step-by-step procedure for solving a problem. The need for this is to present a way to solve the problem with a plan.

**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:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

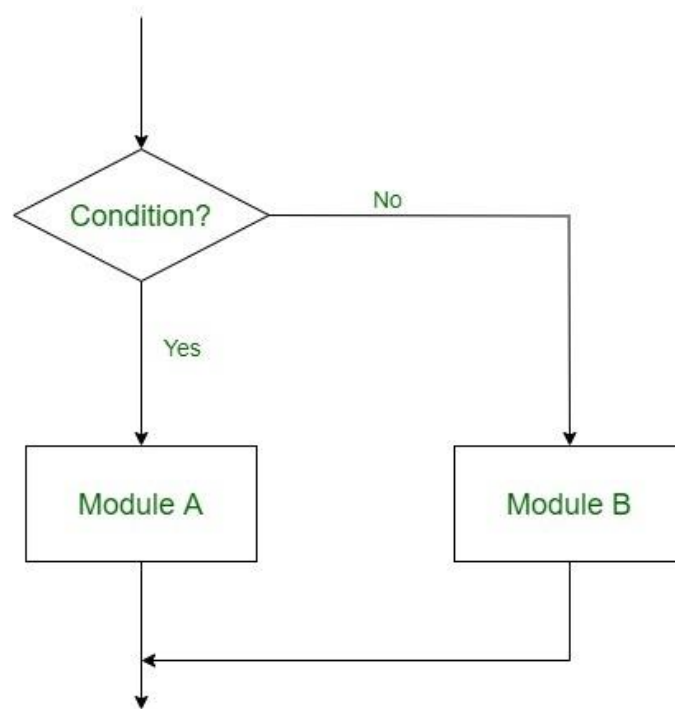
#### Sequential Logic (Sequential Flow)

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. The sequences may be given, by means of numbered steps explicitly. Also, implicitly follows the order in which modules are written. Most of the processing, even some complex problems, will generally follow this elementary flow pattern.



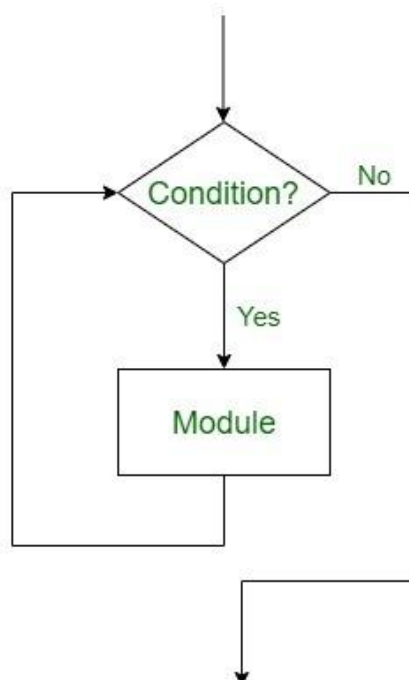
## Selection Logic (Conditional Flow)

Selection Logic simply involves a number of conditions or parameters which decides one out of several written modules. In this way, the flow of the program depends on the set of conditions that are written. This can be more understood by the following flow charts:



## Iteration Logic (Repetitive Flow)

The Iteration logic employs a loop which involves a repeat statement followed by a module known as the body of a loop. In this, there requires a statement that initializes the condition controlling the loop, and there must also be a statement inside the module that will change this condition leading to the end of the loop.



## 9.2 Explores the top down and stepwise refinement methodologies in solving problems

### Modularization

Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

### Stepwise refinement





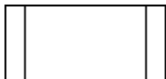

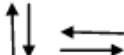
Stepwise refinement is the idea that software is developed by moving through the levels of abstraction, beginning at higher levels and, incrementally refining the software through each level of abstraction, providing more detail at each increment.

### Top down design

Top down program design is an approach to program design that starts with the general concept and repeatedly breaks it down into its component parts. In other words, it starts with the abstract and continually subdivides it until it reaches the specific.

## 9.3 Uses algorithmic approach to solve problems

**A flowchart** is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

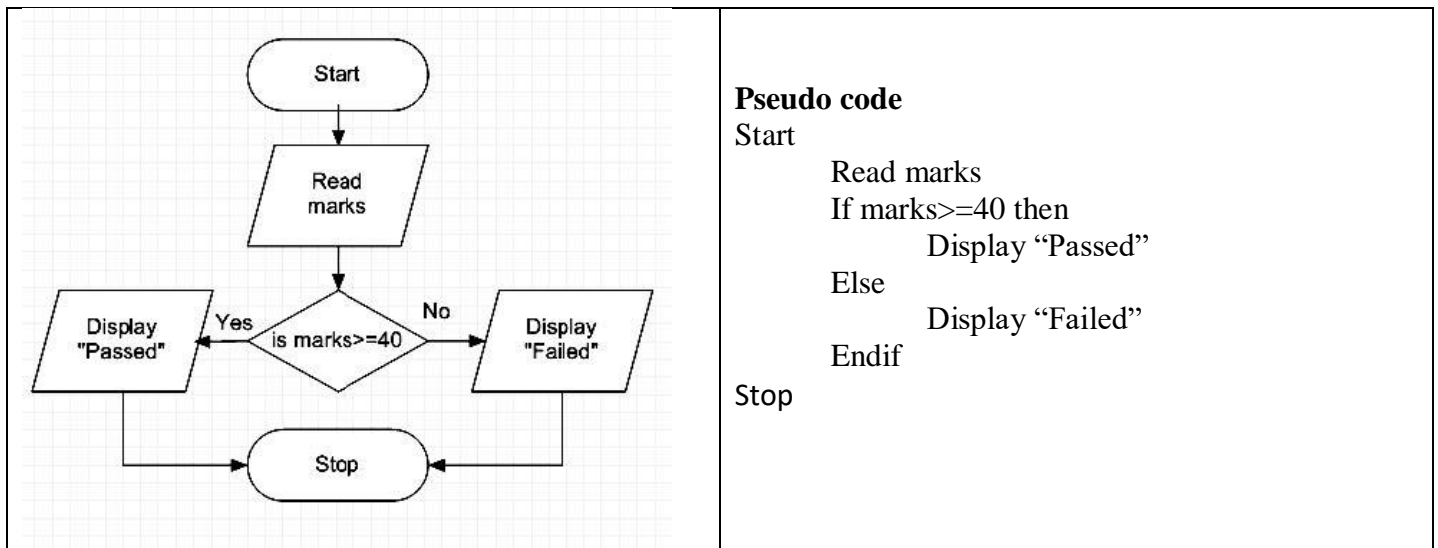
Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

**Pseudo code:** When an algorithm is presented in simple English terms it is called a pseudo code. Pseudo codes are independent of a computer language. Pseudocode is not an actual programming language. So it cannot be compiled into an executable program. It uses short terms or simple English language syntaxes to write code for programs before it is actually converted into a specific programming language. It is used for creating an outline or a rough draft of a program. Pseudocode summarizes a program's flow, but excludes underlying details. System designers write pseudocode to ensure that programmers understand a software project's requirements and align code accordingly.

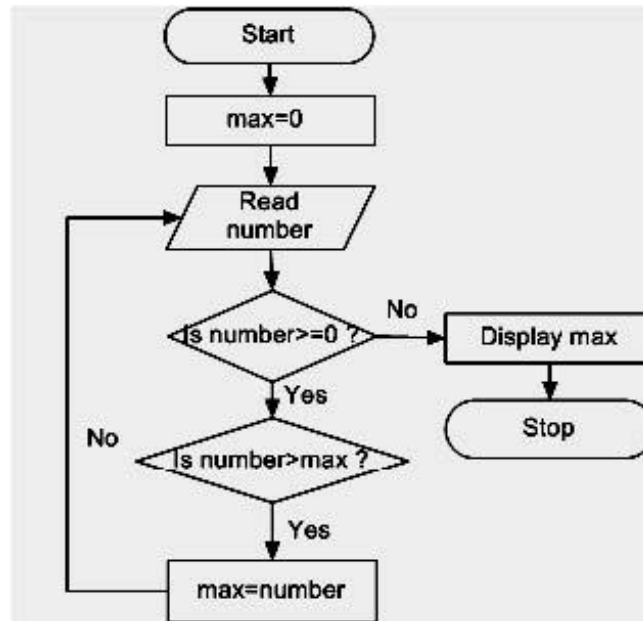
Let us see simple English terms used in an pseudo code.

BEGIN - To indicate a beginning	
END - To indicate an end	
INPUT , READ , GET - To indicate an input	
OUTPUT, DISPLAY , SHOW - To show an output	
PROCESS, CALCULATE - To indicate a process	
IF ... THEN .. .ELSE ... ENDIF - Used to indicate a selection	
FOR – DO	} Used to indicate a repetition
WHILE – ENDWHILE	
REPEAT - UNTIL	

Examples :- 1. Display "passed" if marks are greater than or equal 40, otherwise display "failed"



2. Develop a computer based solution to receive a set of positive numbers or 0 from user and display the maximum value. Input of negative value will stop the input of numbers.



**pseudo code:**

```

start
    max:=0
    read number
    while number>=0
        if number>max then
            max=number
        end if
        read number
    end while
    display max
stop
  
```

**python code:**

```

max:=0
number=int(input("Input number"))
while (number>=0):
    if (number>max):
        max=number
    number=int(input("Input number"))
print('max=',max)
  
```

**Hand traces** Once the pseudocode is written, a manual check is done before converting it into a programming language. This is done by working with an appropriate example situation and line by line. In our example, we could enter or input any age value and see what happens in the process. This is done in order to minimize possible errors when executing the program.

## **9.4 Compares and Contrasts different programming paradigms**

### **Evolution of programming languages**

Programming languages have been developed over the year in a phased manner. Each phase of developed has made the programming language more user-friendly, easier to use and more powerful. Each phase of improved made in the development of the programming languages can be referred to as a generation. The programming language in terms of their performance reliability and robustness can be grouped into five different generations,

#### **1. First Generation Language (Machine language)**

The first generation programming language is also called low-level programming language because they were used to program the computer system at a very low level of abstraction. i.e. at the machine level. The machine language also referred to as the native language of the computer system is the first generation programming language. In the machine language, a programmer only deals with a binary number.

Advantages of first generation language

- They are translation free and can be directly executed by the computers.
- The programs written in these languages are executed very speedily and efficiently by the CPU of the computer system.
- The programs written in these languages utilize the memory in an efficient manner because it is possible to keep track of each bit of data.

#### **2. Second Generation language (Assembly Language)**

The second generation programming language also belongs to the category of low-level-programming language. The second generation language comprises assembly languages that use the concept of mnemonics for the writing program. In the assembly language, symbolic names are used to represent the opcode and the operand part of the instruction.

Advantages of second generation language

- It is easy to develop understand and modify the program developed in these languages are compared to those developed in the first generation programming language.
- The programs written in these languages are less prone to errors and therefore can be maintained with a great ease.

### **3. Third Generation languages (High-Level Languages)**

The third generation programming languages were designed to overcome the various limitations of the first and second generation programming languages. The languages of the third and later generation are considered as a high-level language because they enable the programmer to concentrate only on the logic of the programs without considering the internal architecture of the computer system.

Advantages of third generation programming language

- It is easy to develop, learn and understand the program.
- As the program written in these languages are less prone to errors they are easy to maintain.
- The program written in these languages can be developed in very less time as compared to the first and second generation language.

Examples: FORTRAN, ALGOL, COBOL, C++, C

### **4. Fourth generation language (Very High-level Languages)**

The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer. The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.

Advantages of fourth generation languages

- These programming languages allow the efficient use of data by implementing the various database.
- They require less time, cost and effort to develop different types of software applications.
- The program developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

Examples: SOL, CSS, coldfusion

### **5. Fifth generation language (Artificial Intelligence Language)**

The programming languages of this generation mainly focus on constraint programming. The major fields in which the fifth generation programming language are employed are Artificial Intelligence and Artificial Neural Networks



## Advantages of fifth generation languages

- These languages can be used to query the database in a fast and efficient manner.
- In this generation of language, the user can communicate with the computer system in a simple and an easy manner.

Examples: mercury, prolog, OPS5

## Programming paradigms

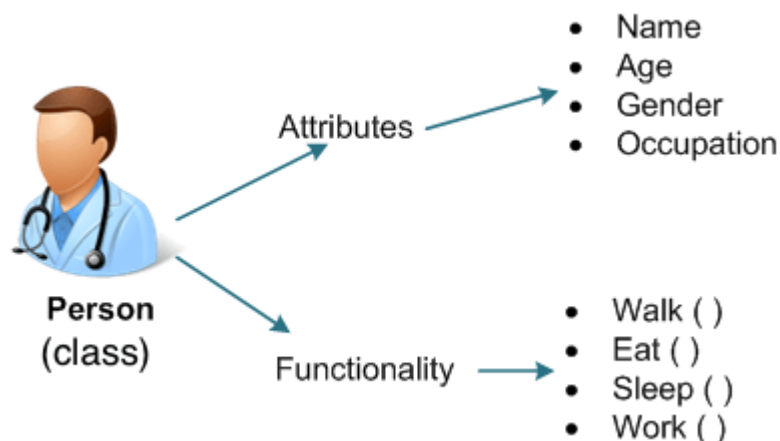
Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

**Declarative programming** is a computer programming paradigm that the developer defines what the program should accomplish rather than explicitly defining how it should go about doing so. This approach lends itself naturally to the programmatic definition of formal logic systems, and has the benefit of simplifying the programming of some parallel processing applications.

**Imperative programming** is a paradigm of computer programming where the program describes steps that change the state of the computer. Unlike declarative programming, which describes "what" a program should accomplish, imperative programming explicitly tells the computer "how" to accomplish it. Programs written this way often compile to binary executables that run more efficiently since all CPU instructions are themselves imperative statements.

**Object Oriented programming (OOP)** is a programming paradigm that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects. There are many object-oriented programming languages including JavaScript, C++, Java, and Python.

A class is an abstract blueprint used to create more specific, concrete objects. Classes often represent broad categories, like Car or Dog that share attributes. These classes define what attributes an instance of this type will have, like color, but not the value of those attributes for a specific object. Classes can also contain functions, called methods available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific type of object.



## 9.5 Explores the need of program translation and the type of program translators

### Source Program

Source program is a program written by a programmer by using HLL (High Level language), which is easily readable by humans. Source programs usually contain variable names which are meaningful and useful comments to make it more readable. A source program can not be directly executed on a machine. To run it, the source program is compiled using a compiler (a program, which changes source programs to executable code). Otherwise, using an interpreter a source program can be executed on the fly. Visual basic is a compiled language program, whereas Java is an interpreted language. When software applications are distributed in a characteristic way they will not include source files. However, if the application is open source, the source is distributed and the user sees and modifies the source code too.

### Object Program

Object program is typically a file which is machine executable, and it is the result of compiling a source file using a compiler. Other than the machine instructions, they can include debugging information, symbols, stack information, rearrangement and profiling information. As they contain instructions in machine code, they are not easily readable by humans. Sometimes object programs are referred as an intermediate object between executable and source files. Linkers are the tools that are used to connect a set of objects to executable format. As mentioned here, Visual basic and java produces object files known as bytecode and .exe respectively. .exe files can be directly executed on windows, while the interpreter is required for the bytecode execution. The executable or object files can be converted back to its original source files through decompilation. For example, .Class files of Java can be decompiled using tools in decompiler.java files.



## Purpose of Translator

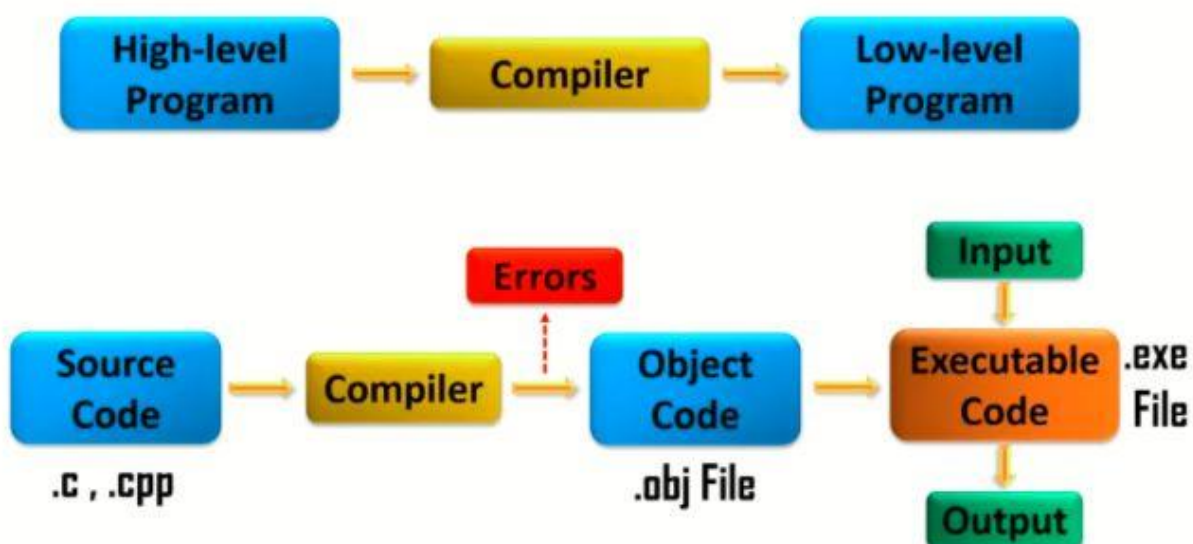
It translates a high-level language program into a machine language program that the central processing unit (CPU) can understand. It also detects errors in the program.

## Different Types of Translators

### 1. Compiler

A compiler is a translator used to convert high-level programming language to low-level programming language. It converts the whole program in one session and reports errors detected after the conversion. The compiler takes time to do its work as it translates high-level code to lower-level code all at once and then saves it to memory.

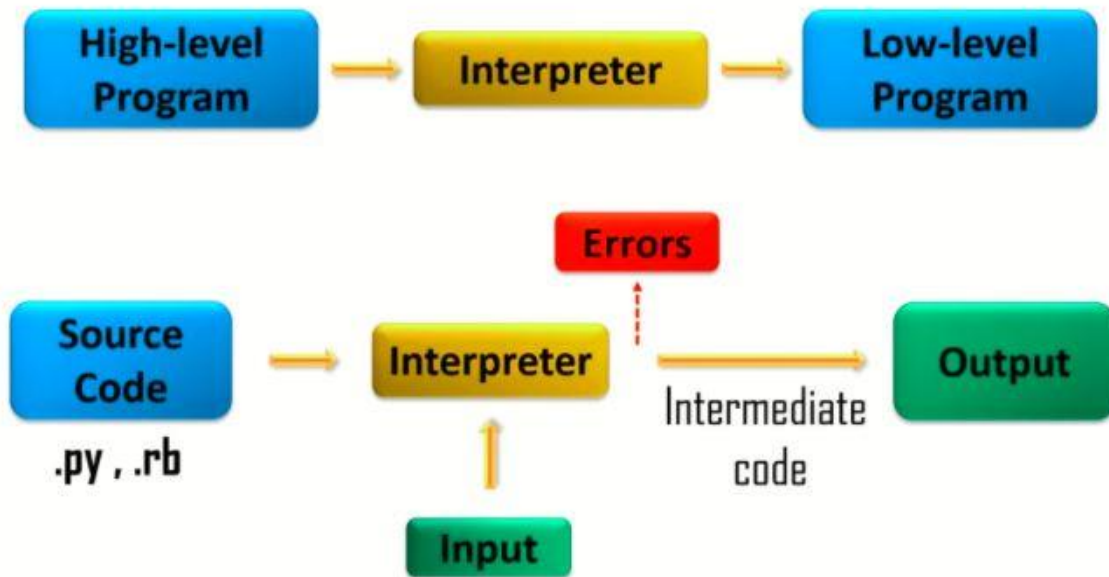
A compiler is processor-dependent and platform-dependent. But it has been addressed by a special compiler, a cross-compiler and a source-to-source compiler. Before choosing a compiler, the user has to identify first the Instruction Set Architecture (ISA), the operating system (OS), and the programming language that will be used to ensure that it will be compatible.



### 2. Interpreter

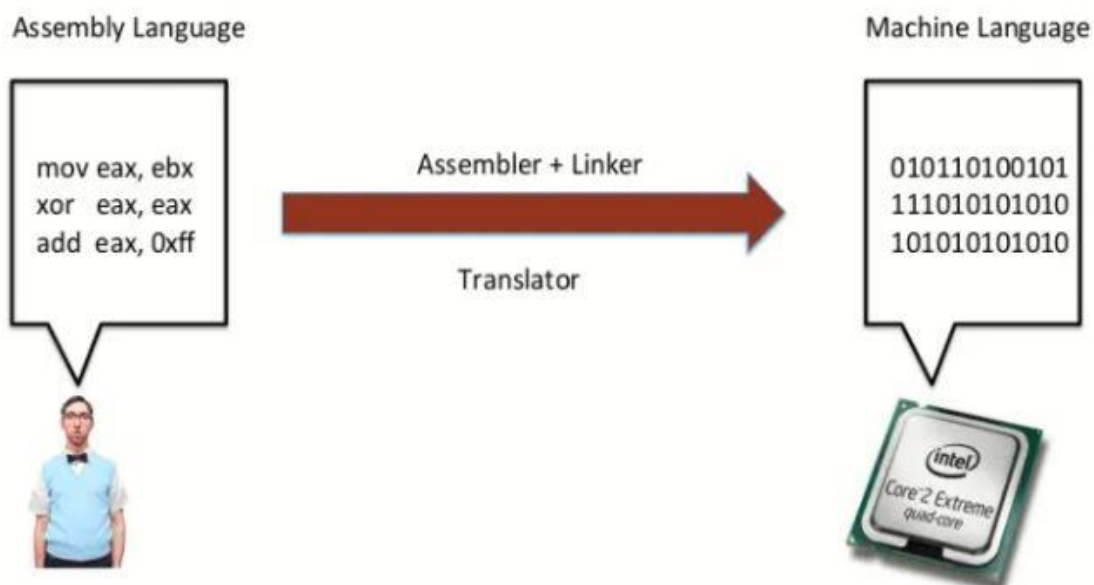
Just like a compiler, is a translator used to convert high-level programming language to low-level programming language. It converts the program one at a time and reports errors detected at once while doing the conversion. With this, it is easier to detect errors than in a compiler. An interpreter is faster than a compiler as it immediately executes the code upon reading the code.

It is often used as a debugging tool for software development as it can execute a single line of code at a time. An interpreter is also more portable than a compiler as it is not processor-dependent, you can work between hardware architectures.



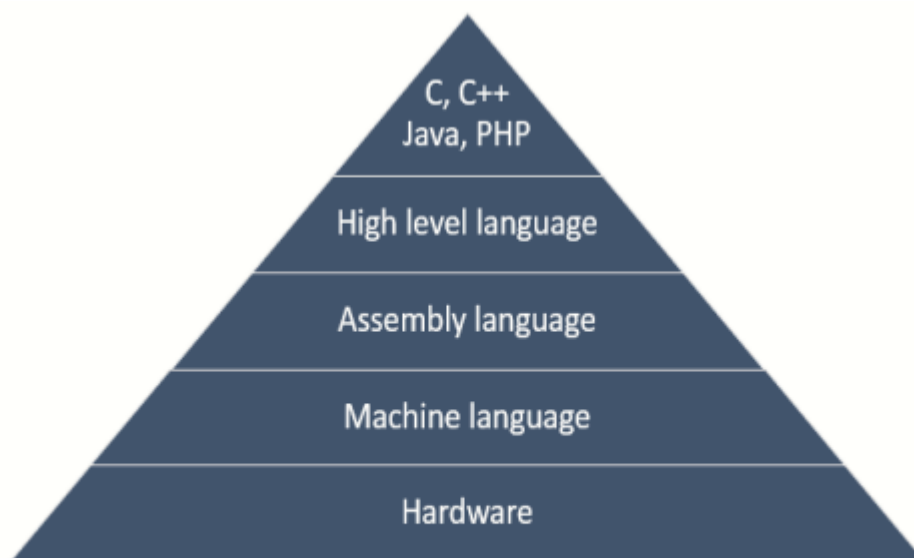
### 3. Assembler

An assembler is a translator used to translate assembly language to machine language. It is like a compiler for the assembly language but interactive like an interpreter. Assembly language is difficult to understand as it is a low-level programming language. An assembler translates a low-level language, an assembly language to an even lower-level language, which is the machine code. The machine code can be directly.



**Hybrid approach** When there are two or more ways of doing something, there are relative advantages and disadvantages, and to balance them, hybrid approaches are taken. An example is a hybridelectric car where there is a battery for town running, and an engine to run outside towns. Similarly, Interpreting and Compilation can be combined to improve the efficiency.

**Linker** Other than interpretation and compilation, there is another process known as Linking. Linking is connecting user commands with standard library functions. For example, the program Input and Output commands are linked to the main program during the linking process. For example, when a user writes printf the linker links the program with the printing subprogram stored in the Linker. The following figure depicts the hierarchical position of Hardware, Machine language, Assembly language and High level languages.



**Fig 9.3 Hardware and computer Languages**

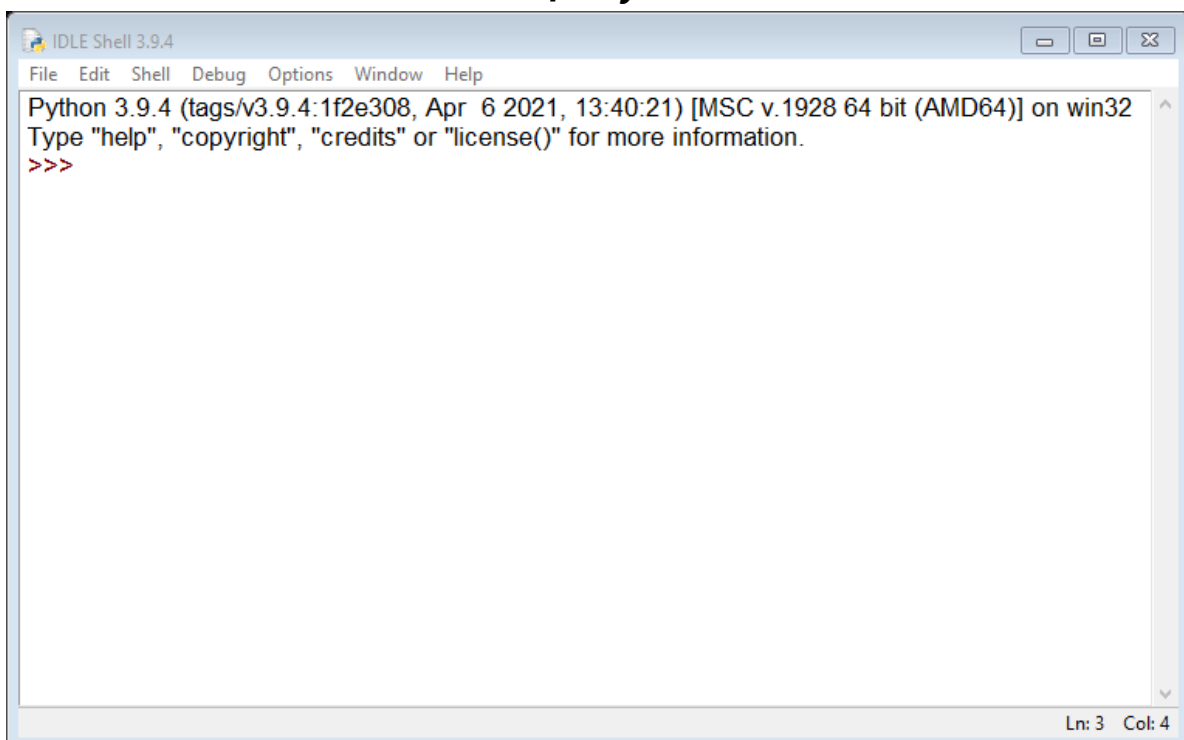
## 9.6 Explores integrated development environment (IDE) to identify their basic features

**An IDE or integrated development environment** is a software application that combines, in one place, all the tools needed for a software development project. On a more basic level, IDEs provide interfaces for users to write code, organize text groups, and automate programming redundancies. But instead of a bare-bones code editor, IDEs combine the functionality of multiple programming processes into one.

**Here are some standard features of an IDE:**

- **Text editor:** Virtually every IDE will have a text editor designed to write and manipulate source code. Some tools may have visual components to drag and drop front-end components, but most have a simple interface highlighting language-specific syntax.
- **Debugger:** Debugging tools assist users in identifying and remedying errors within source code.
- **Compiler:** Compilers are components that translate programming language into a form machines can process, such as binary code.
- **Code completion:** Code complete features assist programmers by intelligently identifying and inserting common code components. These features save developers time writing code and reduce the likelihood of typos and bugs.

### *IDE for Python*



# Python Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

## What can Python do?

- 🐍 Python can be used on a server to create web applications.
- 🐍 Python can be used alongside software to create workflows.
- 🐍 Python can connect to database systems. It can also read and modify files.
- 🐍 Python can be used to handle big data and perform complex mathematics.
- 🐍 Python can be used for rapid prototyping, or for production-ready software development.

## Comments

- Comments starts with a #, and Python will ignore them.

```
#This is a comment  
  
print("Hello, World!")
```

## Variables

- Variables are containers for storing data values.

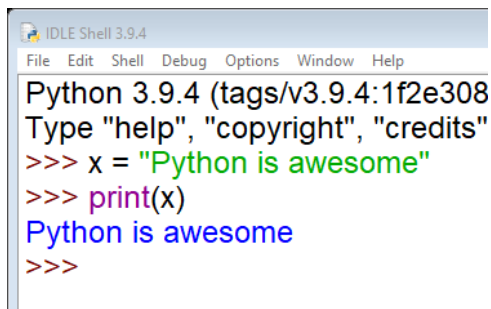
## Rules for Python variables:

1. A variable name must start with a letter or the underscore character
2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
4. Variable names are case-sensitive (age, Age and AGE are three different variables)
5. Python keywords can't be used.

```
x = 5 #integer type  
  
y = "John" #String type  
  
x = 'John' # strings can be declared either by using single or double quotes  
  
x, y, z = "Orange", "Banana", "Cherry" # assign values to multiple variables in one line  
  
x = y = z = "Orange" # assign the same value to multiple variables in one line  
  
Languages="php","python","java" # assign multiple values to single variable
```

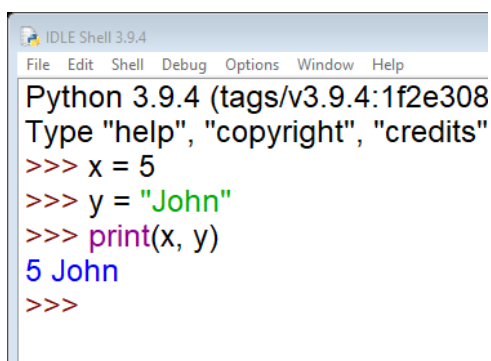
## Output Variables

The Python **print()** function is often used to output variables.



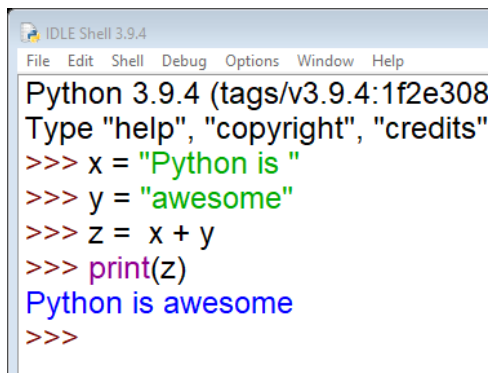
```
Python 3.9.4 (tags/v3.9.4:1f2e308)
Type "help", "copyright", "credits"
>>> x = "Python is awesome"
>>> print(x)
Python is awesome
>>>
```

If you want to output two variables of different data types with the **print()** function, separate them with a comma.



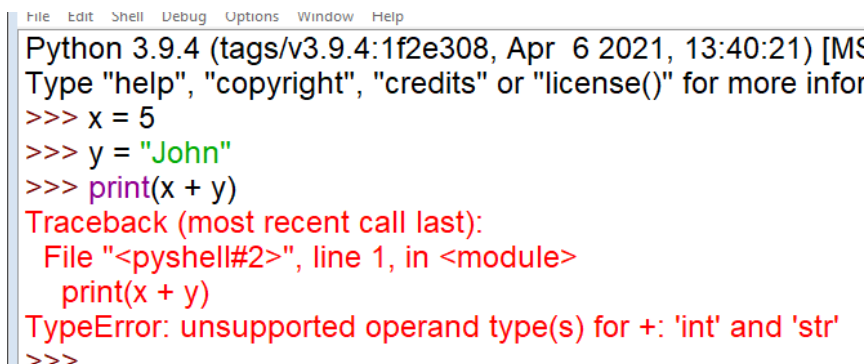
```
Python 3.9.4 (tags/v3.9.4:1f2e308)
Type "help", "copyright", "credits"
>>> x = 5
>>> y = "John"
>>> print(x, y)
5 John
>>>
```

You can also use the **+** operator to add variables into another variable.



```
Python 3.9.4 (tags/v3.9.4:1f2e308)
Type "help", "copyright", "credits"
>>> x = "Python is "
>>> y = "awesome"
>>> z = x + y
>>> print(z)
Python is awesome
>>>
```

In the **print()** function, when you try to combine a string and a number with the **+** operator, Python will give you an error.



```
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MS
Type "help", "copyright", "credits" or "license()" for more infor
>>> x = 5
>>> y = "John"
>>> print(x + y)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(x + y)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```



Use the string formatting with the help of %. One of the two ways to implement string formatting is using the % sign. The % sign, followed by a letter, works as a placeholder for the variable.

```
IDLE Shell 3.9.4
File Edit Shell Debug Options Window Help
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021,
Type "help", "copyright", "credits" or "license()"
>>> var1 = 123
>>> var2 = 'World'
>>> print("Hello to the %s %d " %(var2,var1))
Hello to the World 123
>>>
```

## Data Types

In programming, data type is an important concept. Variables can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories:

- Text Type: **str**
- Numeric Types: **int, float, complex**
- Sequence Types: **list, tuple, range**
- Mapping Type: **dict**
- Set Types: **set, frozenset**
- Boolean Type: **bool**
- Binary Types: **bytes, bytearray, memoryview**

You can get the data type of any object by using the **type()** function:

```
>>> x=6
>>> print(type(x))
<class 'int'>
>>> y="3.0"
>>> print(type(y))
<class 'str'>
>>> z=12.3
>>> print(type(z))
<class 'float'>
>>>
```

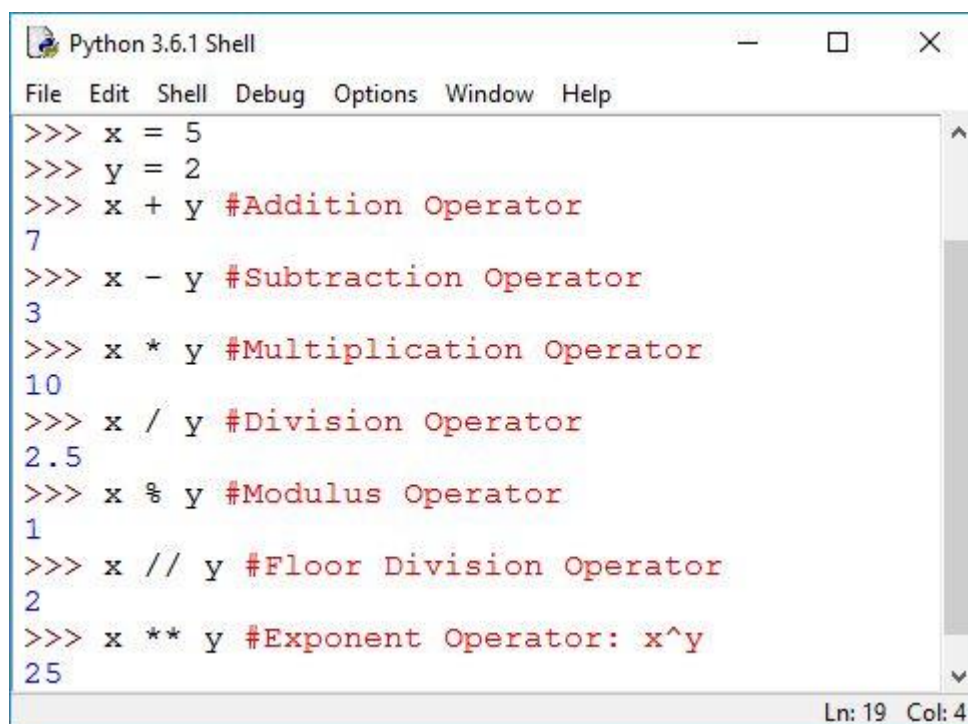
## Operators

**Operators are used to perform operations on variables and values.**

Python divides the operators in the following groups:

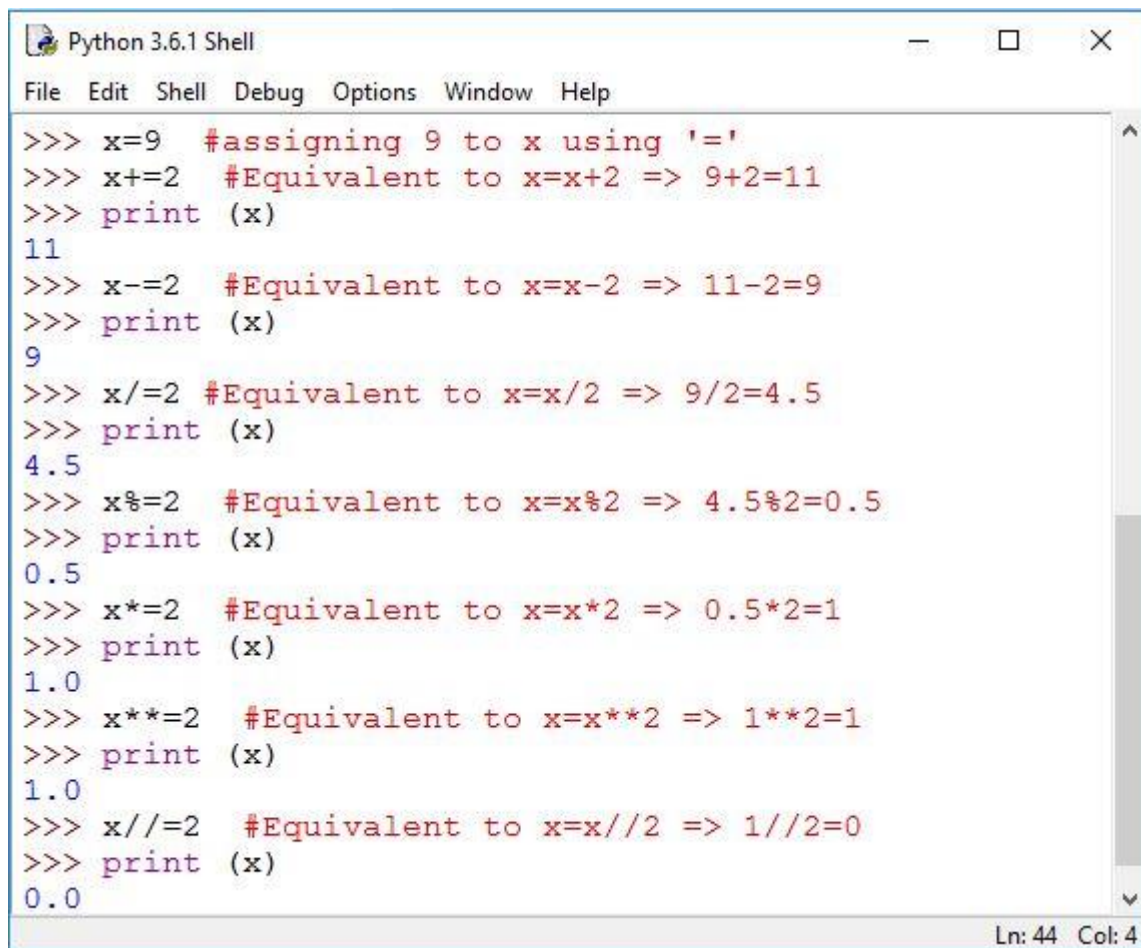
- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

**Arithmetic operators** are used with numeric values to perform common mathematical operations:

A screenshot of a Python 3.6.1 Shell window. The window has a title bar with the text 'Python 3.6.1 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains a series of Python code lines and their outputs. The code defines variables x and y, then performs various arithmetic operations: addition, subtraction, multiplication, division, modulus, floor division, and exponentiation. Each operation is followed by a comment in red text. The outputs are displayed on the lines immediately following the code. At the bottom right of the window, a status bar shows 'Ln: 19 Col: 4'.

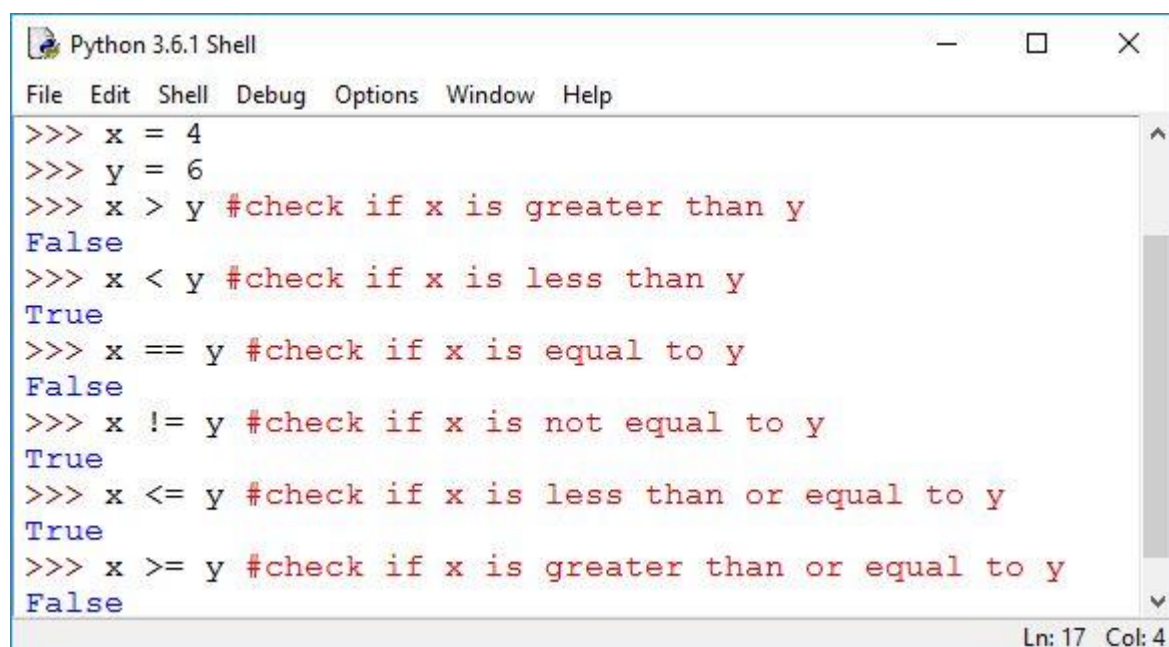
```
>>> x = 5
>>> y = 2
>>> x + y #Addition Operator
7
>>> x - y #Subtraction Operator
3
>>> x * y #Multiplication Operator
10
>>> x / y #Division Operator
2.5
>>> x % y #Modulus Operator
1
>>> x // y #Floor Division Operator
2
>>> x ** y #Exponent Operator: x^y
25
```

**Assignment operators** are used to assign values to variables:

A screenshot of a Python 3.6.1 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains a series of Python commands and their outputs. The commands demonstrate various assignment operators: '=', '+=', '-=', '/=', '%=', '\*=', '\*\*=', and '//='. Each command is followed by a comment explaining its equivalent assignment statement. The status bar at the bottom right shows 'Ln: 44 Col: 4'.

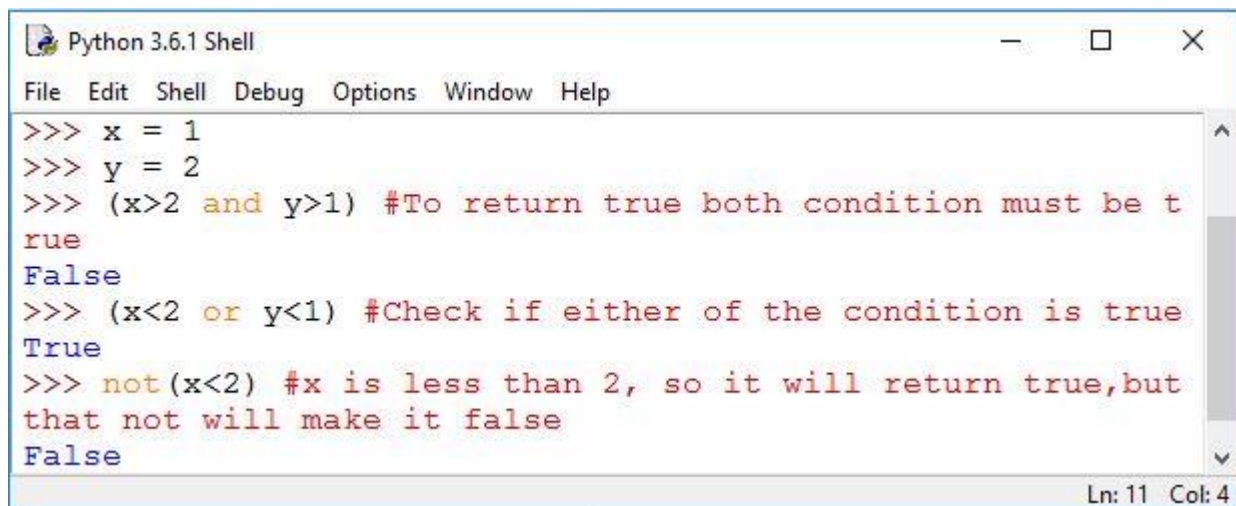
```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
>>> x=9 #assigning 9 to x using '='
>>> x+=2 #Equivalent to x=x+2 => 9+2=11
>>> print (x)
11
>>> x-=2 #Equivalent to x=x-2 => 11-2=9
>>> print (x)
9
>>> x/=2 #Equivalent to x=x/2 => 9/2=4.5
>>> print (x)
4.5
>>> x%=2 #Equivalent to x=x%2 => 4.5%2=0.5
>>> print (x)
0.5
>>> x*=2 #Equivalent to x=x*2 => 0.5*2=1
>>> print (x)
1.0
>>> x**=2 #Equivalent to x=x**2 => 1**2=1
>>> print (x)
1.0
>>> x//=2 #Equivalent to x=x//2 => 1//2=0
>>> print (x)
0.0
Ln: 44 Col: 4
```

**Comparison operators** are used to compare two values:

A screenshot of a Python 3.6.1 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains a series of Python commands and their outputs. The commands demonstrate various comparison operators: '>', '<', '==', '!=', '<=', and '>='. Each command is followed by a comment explaining its purpose. The status bar at the bottom right shows 'Ln: 17 Col: 4'.

```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
>>> x = 4
>>> y = 6
>>> x > y #check if x is greater than y
False
>>> x < y #check if x is less than y
True
>>> x == y #check if x is equal to y
False
>>> x != y #check if x is not equal to y
True
>>> x <= y #check if x is less than or equal to y
True
>>> x >= y #check if x is greater than or equal to y
False
Ln: 17 Col: 4
```

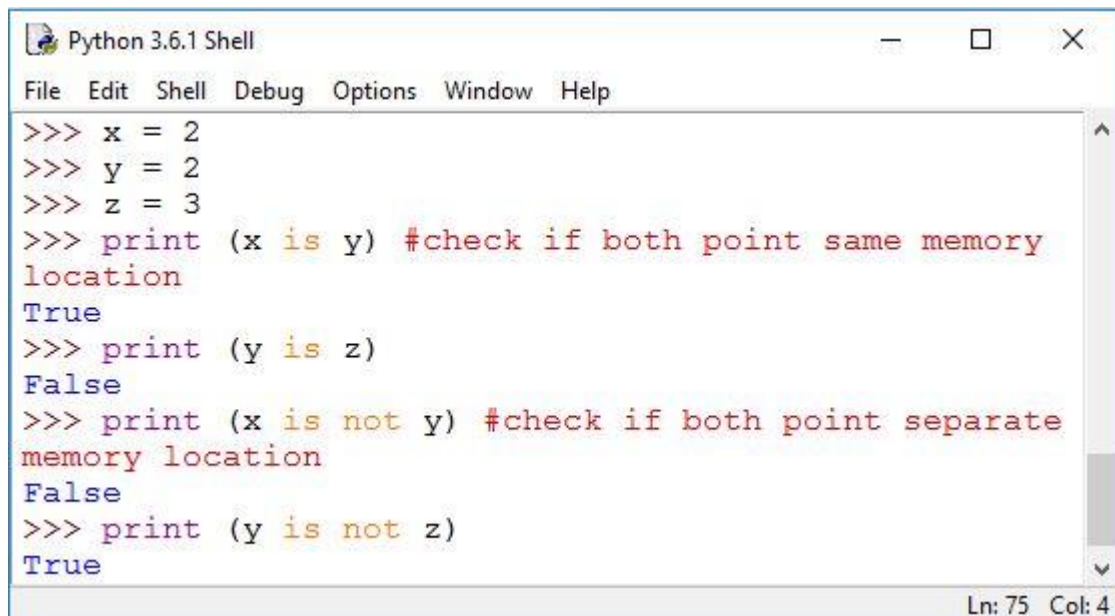
**Logical operators** are used to combine conditional statements:

A screenshot of a Python 3.6.1 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code:

```
>>> x = 1
>>> y = 2
>>> (x>2 and y>1) #To return true both condition must be t
rue
False
>>> (x<2 or y<1) #Check if either of the condition is true
True
>>> not(x<2) #x is less than 2, so it will return true,but
that not will make it false
False
```

The status bar at the bottom right shows 'Ln: 11 Col: 4'.

**Identity operators** are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

A screenshot of a Python 3.6.1 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following code:

```
>>> x = 2
>>> y = 2
>>> z = 3
>>> print (x is y) #check if both point same memory
location
True
>>> print (y is z)
False
>>> print (x is not y) #check if both point separate
memory location
False
>>> print (y is not z)
True
```

The status bar at the bottom right shows 'Ln: 75 Col: 4'.

**Membership operators** are used to test if a sequence is presented in an object:

```
>>> "p" in "python"
True
>>> 1 in [1,2,3,4,5]
True
>>> "x" not in "banana"
True
>>> "z" not in "apple"
True
```

**Bitwise operators** are used to compare (binary) numbers:

```
>>> a=10    a -> 1010
>>> b=7     b -> 0111
>>>
>>> print(a&b) 1010 & 0111 = 0010 = 2
2
>>> print(a|b) 1010 | 0111 = 1111 = 15
15
>>> print(a^b) 1010 ^ 0111 = 1101 = 13
13
>>> print(~a)  ~1010 = - (1011) = -11
-11
>>> print(a<<1) 1010 << 1 = 10100 = 20
20
>>> print(a>>1) 1010 >> 1 = 101 = 5
5
>>>
```

## Precedence of Python Operators

Here is the list of Python Operators in descending order, listing from higher precedence to lower precedence.

Operator	Description
()	Parentheses
**	Exponent (raise to the power)
+, -, ~	Unary plus, Unary minus and Bitwise NOT
*, /, %, //	Multiplication, Division, Modulus and Floor Division
+, -	Addition and Subtraction
>>, <<	Bitwise Right Shift and Bitwise Left Shift
&	Bitwise AND
^,	Bitwise XOR and OR
<=, <, >, >=	Comparison Operators
==, !=	Equality Operators
=, %=, /=, //=, -=, +=, *=, **=	Assignment Operators
is, is not	Identity Operators
in, not in	Membership Operators
not, or, and	Logical Operators

Applying the operator precedence, the expression  $3 + 4 * 4 > 5 * (4 + 3) - 1$  is evaluated as follows:

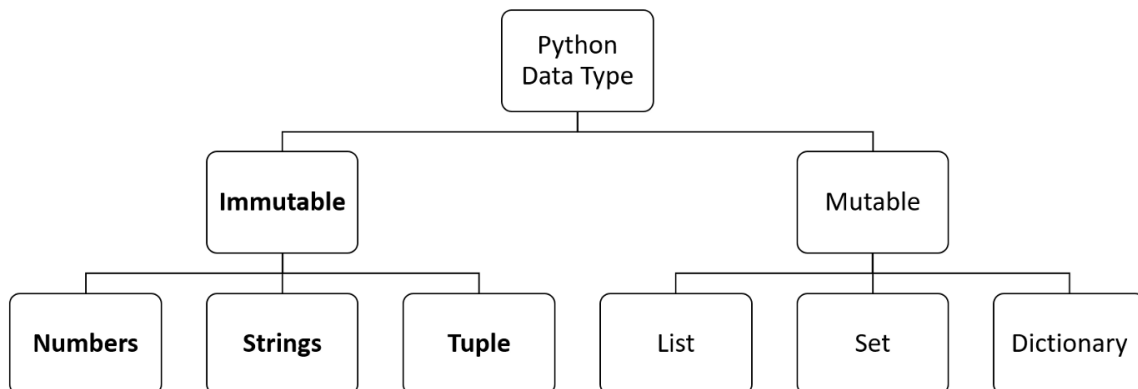
```
3 + 4 * 4 > 5 * (4 + 3) - 1
3 + 4 * 4 > 5 * 7 - 1
3 + 16 > 5 * 7 - 1
3 + 16 > 35 - 1
19 > 35 - 1
19 > 34
false
```

(1) inside parentheses first  
(2) multiplication  
(3) multiplication  
(4) addition  
(5) subtraction  
(6) greater than



## Mutable and immutable objects

Python mutability means an object's ability to change. An immutable object cannot change, but a mutable object can. For example, a list is mutable, but a tuple is not. In other words, you can freely change the elements of a list, but not the ones of a tuple.



## Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks. 'hello' is the same as "hello".

### Slicing Strings

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string.

#### Positive Indexing

**str = "HELLO"**

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'	str[:] = 'HELLO'
str[1] = 'E'	str[0:] = 'HELLO'
str[2] = 'L'	str[:5] = 'HELLO'
str[3] = 'L'	str[:3] = 'HEL'
str[4] = 'O'	str[0:2] = 'HE'
	str[1:4] = 'ELL'

#### Negative Indexing

**str = "HELLO"**

H	E	L	L	O
-5	-4	-3	-2	-1

str[-1] = 'O'	str[-3:-1] = 'LL'
str[-2] = 'L'	str[-4:-1] = 'ELL'
str[-3] = 'L'	str[-5:-3] = 'HE'
str[-4] = 'E'	str[-4:] = 'ELLO'
str[-5] = 'H'	str[::-1] = 'OLLEH'

## Modify Strings

Python has a set of built-in methods that you can use on strings.

Method	Example	Output
The upper() method returns the string in upper case:	a = "Hello, World!" print(a.upper())	HELLO, WORLD!
The lower() method returns the string in lower case:	a = "Hello, World!" print(a.lower())	hello, world!
The strip() method removes any whitespace from the beginning or the end:	a = " Hello, World! " print(a.strip()) # returns "Hello, World!"	Hello, World!
The replace() method replaces a string with another string:	a = "Hello, World!" print(a.replace("H", "J"))	Jello, World!
The split() method splits the string into substrings if it finds instances of the separator:	a = "Hello, World!" print(a.split(",")) # returns ['Hello', ' World!']	['Hello', ' World!']

## String Formatting

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

- %s - String (or any object with a string representation, like numbers)
- %d - Integers
- %f - Floating point numbers

Let's say you have a variable called "name" with your user name in it, and you would then like to print(out a greeting to that user.)

Code	Output
name = "John" print("Hello, %s!" % name)	"Hello, John!"
x = 'looked' print("Misha %s and %s around"%('walked',x))	"Misha walked and looked around."
print('Joe stood up and %s to the crowd.' % 'spoke')  print('There are %d dogs.' %4)	"Joe stood up and spoke to the crowd." "There are 4 dogs."



## Python Lists

```
list1 = ["abc", 34, True, 40, "male"]
```

- 💡 Lists are used to store multiple items in a single variable.
- 💡 Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
- 💡 Lists are created using square brackets.
- 💡 Lists are ordered.
- 💡 Lists are changeable/Mutable
- 💡 Lists can have items with the same value.

**To determine how many items a list has, use the len() function:**

```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist)) #Outputs 3
```

### Access List Items

List items are indexed and you can access them by referring to the index number:

	<b>-6</b>	<b>-5</b>	<b>-4</b>	<b>-3</b>	<b>-2</b>	<b>-1</b>
Thislist=	["Apple",	"Banana",	"Cherry",	"Mango",	"Berry",	"Orange"]
	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>

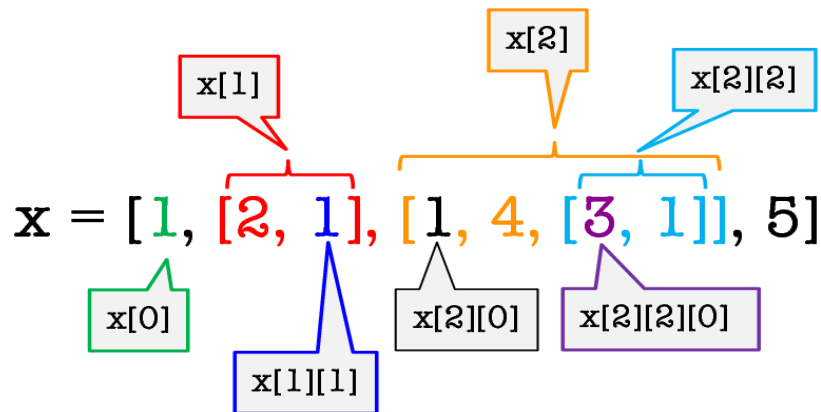
**Remember that the first item has index 0.**

Examples -

```
>>> Thislist=["Apple","Banana","Cherry","Mango","Berry","Orange"]  
>>> print(Thislist[::-1])  
['Orange', 'Berry', 'Mango', 'Cherry', 'Banana', 'Apple']  
>>> print(Thislist[::2])  
['Apple', 'Cherry', 'Berry']  
>>> print(Thislist[::])  
['Apple', 'Banana', 'Cherry', 'Mango', 'Berry', 'Orange']  
>>> print(Thislist[2:5])  
['Cherry', 'Mango', 'Berry']  
>>> print(Thislist[-5:-1:2])  
['Banana', 'Mango']
```

## Nested Lists

A list can contain any sort object, even another list (sublist), which in turn can contain sublists themselves, and so on. This is known as nested list.



```
>>> aList=["java","sql",["php","html","css"],"c++","nodeJs"]
>>> aList[0]
'java'
>>> aList[0][0]
'j'
>>> aList[2]
['php', 'html', 'css']
>>> aList[2][:2]
['php', 'css']
>>> aList[2][-1][0]
'c'
>>> aList[-1][: -1]
'sJedon'
>>> aList[-4:-1]
['sql', ['php', 'html', 'css'], 'c++']
>>> aList[:3]
['java', 'c++']
>>> aList[::-2]
['nodeJs', ['php', 'html', 'css'], 'java']
```

## Change/Add/Remove List Items

Code	Output
To change the value of a specific item, refer to the index number:	
thislist = ["apple", "banana", "cherry"] thislist[1] = "blackcurrant" print(thislist)	['apple', 'blackcurrant', 'cherry']
thislist = ["apple", "banana", "cherry"] thislist[1:2] = ["blackcurrant", "watermelon"] print(thislist)	['apple', 'blackcurrant', 'watermelon', 'cherry']
To add an item to the end of the list, use the append() method	
thislist = ["apple", "banana", "cherry"] thislist.append("orange") print(thislist)	['apple', 'banana', 'cherry', 'orange']
The insert() method inserts an item at the specified index	
thislist = ["apple", "banana", "cherry"] thislist.insert(1, "orange") print(thislist)	['apple', 'orange', 'banana', 'cherry']
The remove() method removes the specified item.	
thislist = ["apple", "banana", "cherry"] thislist.remove("banana") print(thislist)	['apple', 'cherry']
If you do not specify the index, the pop() method removes the last item.	
thislist = ["apple", "banana", "cherry"] thislist.pop() print(thislist)	['apple', 'banana']
The del keyword also removes the specified index:	
thislist = ["apple", "banana", "cherry"] del thislist[0] print(thislist)	['banana', 'cherry']
The sort() method sorts the list ascending by default.	
cars = ['Ford', 'BMW', 'Volvo'] cars.sort()	['BMW', 'Ford', 'Volvo']

## Get a list as input from user

```
# For list of integers
lst1 = []

# For list of strings/chars
lst2 = []

lst1 = [int(item) for item in input("Enter the list items : ").split()]

lst2 = [item for item in input("Enter the list items : ").split()]

print(lst1)
print(lst2)
```

## How it works?

```
Enter the list 1 items : 3 2 4 1 0 6 7
```

```
Enter the list 2 items : ABC DEF GHI
```

```
[3, 2, 4, 1, 0, 6, 7]
['ABC', 'DEF', 'GHI']
```

## Join Two Lists

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
print(list3)
```

Outputs - ['a', 'b', 'c', 1, 2, 3]

## Sort Descending

To sort descending, use the keyword argument `reverse = True`:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Outputs - ['pineapple', 'orange', 'mango', 'kiwi', 'banana']

## Python Tuples

```
mytuple = ("apple", "banana", "cherry")
```

- ✔ Tuples are used to store multiple items in a single variable.
- ✔ Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- ✔ Tuples are written with round brackets.
- ✔ Tuples are ordered
- ✔ Tuples are **unchangeable/Immutable**.
- ✔ Allow Duplicates

Operations	examples	description
<b>Creating a tuple</b>	<pre>&gt;&gt;&gt;a=(20,40,60,"apple","ball")</pre>	Creating the tuple with elements of different data types.
<b>Indexing</b>	<pre>&gt;&gt;&gt;print(a[0]) 20 &gt;&gt;&gt; a[2] 60</pre>	Accessing the item in the position 0 Accessing the item in the position 2
<b>Slicing</b>	<pre>&gt;&gt;&gt;print(a[1:3]) (40,60)</pre>	Displaying items from 1st till 2nd.
<b>Concatenation</b>	<pre>&gt;&gt;&gt; b=(2,4) &gt;&gt;&gt;print(a+b) &gt;&gt;&gt;(20,40,60,"apple","ball",2,4)</pre>	Adding tuple elements at the end of another tuple elements
<b>Repetition</b>	<pre>&gt;&gt;&gt;print(b*2) &gt;&gt;&gt;(2,4,2,4)</pre>	repeating the tuple in n no of times
<b>Membership</b>	<pre>&gt;&gt;&gt; a=(2,3,4,5,6,7,8,9,10) &gt;&gt;&gt; 5 in a True &gt;&gt;&gt; 100 in a False &gt;&gt;&gt; 2 not in a False</pre>	Returns True if element is present in tuple. Otherwise returns false.
<b>Comparison</b>	<pre>&gt;&gt;&gt; a=(2,3,4,5,6,7,8,9,10) &gt;&gt;&gt;b=(2,3,4) &gt;&gt;&gt; a==b False &gt;&gt;&gt; a!=b True</pre>	Returns True if all elements in both elements are same. Otherwise returns false

methods	example	description
a.index(tuple)	>>> a=(1,2,3,4,5) >>> a.index(5) 4	Returns the index of the first matched item.
a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple
min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

### Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Tuple = ( 0, 1, 2, 3, 4, 5 )

<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
----------	----------	----------	----------	----------	----------

Tuple[0] = 0                  Tuple[0:] = (0, 1, 2, 3, 4, 5)

Tuple[1] = 1                  Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2] = 2                  Tuple[2:4] = (2, 3)

Tuple[3] = 3                  Tuple[1:3] = (1, 2)

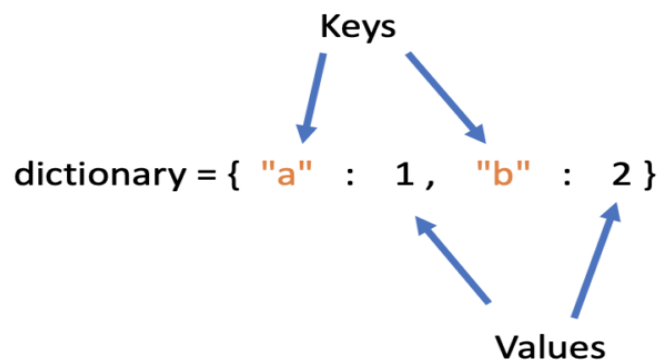
Tuple[4] = 4                  Tuple[:4] = (0, 1, 2, 3)

Tuple[5] = 5

## Python Dictionaries

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
```

- ✔ Dictionaries are used to store data values in key:value pairs.
- ✔ A dictionary is a collection which is ordered, changeable and do not allow duplicates.
- ✔ As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.
- ✔ Dictionaries are written with curly brackets, and have keys and values:
- ✔ **Duplicates Not Allowed.**



### Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

<pre>thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } print(thisdict["model"])</pre>	<p>Output</p> <p>"Mustang"</p>
--	--------------------------------

### Change Dictionary Items

You can change the value of a specific item by referring to its key name:

<pre>thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict["year"] = 2018 print(thisdict)</pre>	<p>Output</p> <pre>{ 'brand': 'Ford', 'model': 'Mustang', 'year': 2018 }</pre>
---	--

## Adding Items

<pre>thisdict = {     "brand": "Ford",     "model": "Mustang",     "year": 1964 } thisdict["color"] = "red" print(thisdict)</pre>	<pre>Output { 'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red' }</pre>
<pre>thisdict = {     "brand": "Ford",     "model": "Mustang",     "year": 1964 } thisdict.update({"color": "red"})</pre>	<pre>{ 'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red' }</pre> <p>Note - The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.</p>

## Removing Items

<pre>thisdict = {     "brand": "Ford",     "model": "Mustang",     "year": 1964 } thisdict.pop("model") print(thisdict)</pre>	<pre>Output { 'brand': 'Ford', 'year': 1964 }</pre>
<pre>thisdict = {     "brand": "Ford",     "model": "Mustang",     "year": 1964 } del thisdict["model"] print(thisdict)</pre>	<pre>Output { 'brand': 'Ford', 'year': 1964 }</pre>
<pre>car = {     "brand": "Ford",     "model": "Mustang",     "year": 1964 }  car.popitem()  print(car)</pre>	<pre>Output  { 'brand': 'Ford', 'model': 'Mustang' }</pre> <p>Note - The popitem() method removes the item that was last inserted into the dictionary.</p>



## Python If ... Else



The `if` statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the `else` statement. We can use the `else` statement with `if` statement to execute a block of code when the condition is false.

The **`elif` keyword** is python's way of saying "if the previous conditions were not true, then try this condition".

The **`else` keyword** catches anything which isn't caught by the preceding conditions.

<pre>i = 20 if (i &lt; 15):     print("i is smaller than 15")     print("i'm in if Block") else:     print("i is greater than 15")     print("i'm in else Block") print("i'm not in if and not in else Block")</pre>	<b>Output</b>  i is greater than 15  i'm in else Block  i'm not in if and not in else Block
<pre>i = 20 if (i == 10):     print("i is 10") elif (i == 15):     print("i is 15") elif (i == 20):     print("i is 20") else:     print("i is not present")</pre>	<b>Output</b>       i is 20

<pre>x = 41  if x &gt; 10:     print("Above ten,") if x &gt; 20:     print("and also above 20!") else:     print("but not above 20.")</pre>	<p><b>Output</b></p> <p>Above ten, and also above 20!</p>
<pre>n=int(input("Enter number:- ")) #inputs 45 if(n&gt;=50):     result=0     if(n&gt;=80):         result="a"     else:         if(n&gt;=60):             result="c1"         else:             result="c2" else:     if(n&gt;40):         result="s"     else:         result="f" print(result)</pre>	<p><b>Output</b></p> <p>f</p>
<pre>a = 200 b = 33 c = 500 if a &gt; b or a &gt; c:     print("At least one of the conditions is True")</pre>	<p><b>Output</b></p> <p>At least one of the conditions is True</p>
<pre>a = 200 b = 33 c = 500 if a &gt; b and c &gt; a:     print("Both conditions are True")</pre>	<p><b>Output</b></p> <p>Both conditions are True</p>
<pre>marks = 60 if not(marks&lt;50):     print("pass") else:     print("fail")</pre>	<p><b>Output</b></p> <p>Pass</p>

## While Loops

With the while loop we can execute a set of statements as long as a condition is true.

<pre>i = 1 while i &lt; 6:     print(i)     i += 1</pre>	1 2 3 4 5
<pre>i = 1 while i &lt; 6:     print(i)     if i == 3:         break     i += 1</pre>	1 2 3
<pre>i = 0 while i &lt; 6:     i += 1     if i == 3:         continue     print(i)</pre>	1 2 4 5 6
<pre>thislist = ["apple", "banana", "cherry"] i = 0 while i &lt; len(thislist):     print(thislist[i])     i = i + 1</pre>	"apple" "banana" "cherry"
<pre>aList=[2,3,5,6,-12,4,9,-1,8,13] i=total=0 while(i&lt;len(aList)):     if (aList[i]%2==0) and (aList[i]&gt;0):         total+=aList[i]     i+=1 print(total)</pre>	20
<pre>aList=[2,3,5,6,-12,4] i=total=0 while True:     if (aList[i]%2==1):         total+=aList[i]     if (aList[i]&lt;0):         break     i+=1 print(total)</pre>	8  <b>Note - With the break statement we can stop the loop even if the while condition is true.</b>
<pre>thistuple = ("apple", "banana", "cherry") i = 0 while i &lt; len(thistuple):     print(thistuple[i])     i = i + 1</pre>	"apple" "banana" "cherry"

## For Loops

- 🌿 A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- 🌿 This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- 🌿 With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Code	Output
<pre>for x in "banana":     print(x)</pre>	<pre>b a n a n a</pre>
<pre>fruits = ["apple", "banana", "cherry"] for x in fruits:     print(x)     if x == "banana":         break</pre>	<pre>apple banana</pre>
<pre>fruits = ["apple", "banana", "cherry"] for x in fruits:     if x == "banana":         break     print(x)</pre>	<pre>apple</pre> <p>Note - With the break statement we can stop the loop even if the while condition is true.</p>
<pre>fruits = ["apple", "banana", "cherry"] for x in fruits:     if x == "banana":         continue     print(x)</pre>	<pre>apple cherry</pre> <p>Note - With the continue statement we can stop the current iteration, and continue with the next.</p>
<pre>for x in range(6):     print(x)</pre>	<pre>0,1,2,3,4,5</pre>
<pre>for x in range(2, 6):     print(x)</pre>	<pre>2,3,4,5</pre>
<pre>for x in range(2, 30, 3):     print(x,end=",")</pre>	<pre>2,5,8,11,14,17,20,23,26,29</pre>
<pre>adj = ["red", "big", "tasty"] fruits = ["apple", "banana", "cherry"]  for x in adj:     for y in fruits:         print(x, y)</pre>	<pre>red apple red banana red cherry big apple big banana big cherry tasty apple tasty banana tasty cherry</pre>

<pre>thislist = ["apple", "banana", "cherry"] for i in range(len(thislist)):     print(thislist[i])</pre>	<pre>"apple" "banana" "cherry"</pre>
<pre>aList=[10,5,3,11,-2,1,8] bList=[] for i in range(len(aList)):     if(aList[i]&gt;5):         bList.append(aList[i]) print(sorted(bList))</pre>	<pre>[8,10,11]</pre> <p>Note - The append() method add an item to the end of the list.</p>
<pre>aList=[10,5,3,11,-2,1,8] bList=[4,8,6,4,5,9,1] count=0 for i in range(len(aList)):     if(aList[i] not in bList):         count+=1 print(count)</pre>	<pre>4</pre>
<pre>aList=[10,5,3,11,-2,1,8] total=0 for i in range(len(aList)):     if(aList[i]&gt;=10):         continue     total+=aList[i] print(total)</pre>	<pre>15</pre> <p>Note - With the continue statement we can stop the current iteration, and continue with the next.</p>
<pre>aList=[int(x) for x in input('Enter :-').split(",")] total=0 for i in range(len(aList)):     if not(aList[i]&gt;=10) and (aList[i]%2!=0):         continue     total+=aList[i] print(total) #inputs - 2,5,10,13,7,0,11</pre>	<pre>36</pre>

## The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

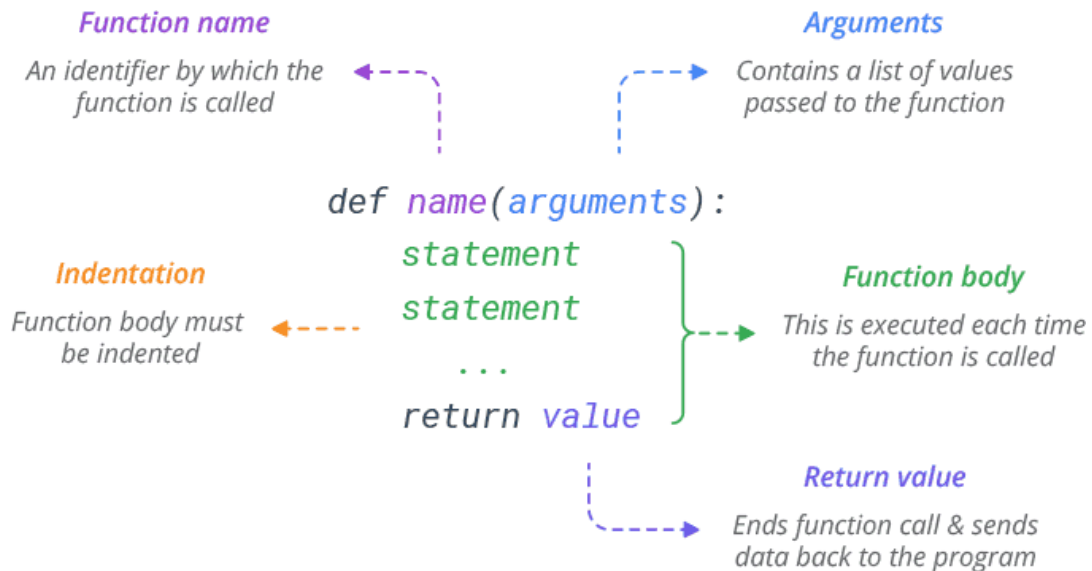
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number. However it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

**range(start, stop, step)**

**range(2, 10, 3) -> 2,5,8**

**range(5,0,-1) -> 5,4,3,2,1**

## Functions



- 💡 A function is a block of code which only runs when it is called.
- 💡 You can pass data, known as parameters, into a function.
- 💡 A function can return data as a result.

### Arguments

- 💡 Information can be passed into functions as arguments.
- 💡 Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- 💡 The following example has a function with one argument (`fname`). When the function is called, we pass along a first name, which is used inside the function to print the full name:

<pre>def my_function(fname):     print(fname + " Refsnes")  my_function("Emil") my_function("Tobias") my_function("Linus")</pre>	<b>Output</b>  Emil Refsnes Tobias Refsnes Linus Refsnes
--	--

### Default Parameter Value

If we call the function without argument, it uses the default value

<pre>def my_function(country = "Norway"):     print("I am from " + country)  my_function("Sweden") my_function("India") my_function() my_function("Brazil")</pre>	<b>Output</b>  I am from Sweden I am from India I am from Norway I am from Brazil
---	--

## Return Values

To let a function return a value, use the return statement:

<pre>def my_function(x):     return 5 * x  print(my_function(3)) print(my_function(5)) print(my_function(9))</pre>	<b>Output</b>  15 25 45
--	-------------------------------------

A variable is only available from inside the region it is created. This is called **scope**.

## Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

<pre>def myfunc():     x = 300 #Local     print(x)  myfunc()</pre>	<b>Output</b>  300
--	--------------------------

## Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

<pre>x = 300 #Global  def myfunc():     print(x)  myfunc()  print(x)</pre>	<b>Output</b>  300 300
<pre>x = 300 #Global  def myfunc():     x = 200 #Local     print(x)  myfunc()  print(x)</pre>	<b>Output</b>  200 300

## Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword. **The global keyword makes the variable global.**

<pre>x = 300 #Global  def myfunc():     global x     x = 200 #Global  myfunc()  print(x)</pre>	<b>Output</b>  200
--	--------------------------

## File Handling

- ✔ The key function for working with files in Python is the open() function.
- ✔ The open() function takes two parameters; *filename*, and *mode*.
- ✔ There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

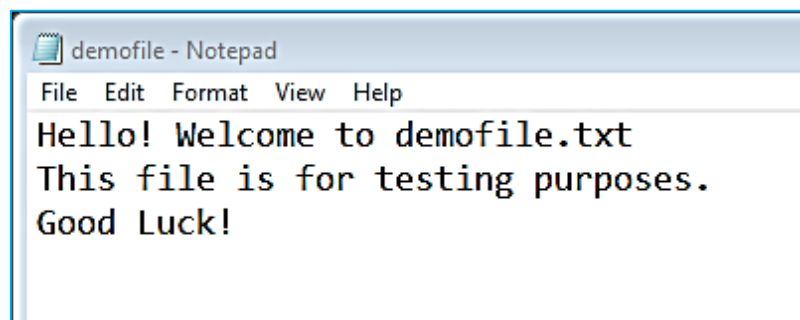
"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

## File Read

Assume we have the following demofile.txt file, located in the same folder as Python:





To open the file, use the built-in `open()` function. The `open()` function returns a file object, which has a `read()` method for reading the content of the file. To open a file for reading it is enough to specify the name of the file:

*Example :- `f = open("demofile.txt")` same as `f = open("demofile.txt", "r")`*

<p>It is a good practice to always close the file when you are done with it.</p> <pre>f = open("demofile.txt", "r") print(f.read()) f.close()</pre>	<p><b>Output</b></p> <p>Hello! Welcome to demofile.txt This file is for testing purposes. Good Luck!</p>
<p>By default the <code>read()</code> method returns the whole text, but you can also specify how many characters you want to return:</p> <pre>f = open("demofile.txt", "r") print(f.read(5)) f.close()</pre>	<p>Hello</p>
<p>You can return one line by using the <code>readline()</code> method:</p> <pre>f = open("demofile.txt", "r") print(f.readline()) f.close()</pre>	<p>Hello! Welcome to demofile.txt</p>
<p>By looping through the lines of the file, you can read the whole file, line by line:</p> <pre>f = open("demofile.txt", "r") for x in f:     print(x) f.close()</pre>	<p>Hello! Welcome to demofile.txt This file is for testing purposes. Good Luck!</p>
<p>By calling <code>readline()</code> two times, you can read the two first lines:</p> <pre>f = open("demofile.txt", "r") print(f.readline()) print(f.readline()) f.close()</pre>	<p>Hello! Welcome to demofile.txt This file is for testing purposes.</p>
<p>If the file is located in a different location, you will have to specify the file path, like this:</p> <pre>f = open("D:\\myfiles\\welcome.txt", "r") print(f.read()) f.close()</pre>	<p>Welcome to this text file! This file is located in a folder named "myfiles", on the D drive. Good Luck!</p>

## File Write

To write to an existing file, you must add a parameter to the open() function:

- ✔ "a" - Append - will append to the end of the file
- ✔ "w" - Write - will overwrite any existing content

<pre>f = open("demofile.txt", "a") f.write("Now the file has more content!") f.close()  #open and read the file after the appending: f = open("demofile2.txt", "r") print(f.read())</pre> 	<p><b>Output</b></p> <p>Hello! Welcome to demofile.txt This file is for testing purposes. Good Luck!Now the file has more content!</p> 
<pre>f = open("demofile.txt", "w") f.write("Woops! I have deleted the content!") f.close()  #open and read the file after the appending: f = open("demofile.txt", "r") print(f.read())</pre> 	<p><b>Output</b></p> <p>Woops! I have deleted the content!</p> 

## Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

- ✔ "x" - Create - will create a file, returns an error if the file exist
- ✔ "a" - Append - will create a file if the specified file does not exist
- ✔ "w" - Write - will create a file if the specified file does not exist

Create a file called "myfile.txt":- `f = open("myfile.txt", "x")`

Create a new file if it does not exist:- `f = open("myfile.txt", "w")`

## Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function. To delete an entire folder, use the `os.rmdir()` method:

📄 Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

📄 Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```



Python can be used in database applications. One of the most popular databases is MySQL. Python needs a [MySQL driver](https://dev.mysql.com/downloads/connector/python/) to access the MySQL database.

Creating a Database	Creating a Table
<pre>import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword" )  mycursor = mydb.cursor()  mycursor.execute("CREATE DATABASE mydatabase")</pre>	<pre>import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword",     database="mydatabase" )  mycursor = mydb.cursor()  mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")</pre>

<p><b>Insert</b></p> <pre> import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword",     database="mydatabase" )  mycursor = mydb.cursor()  sql = "INSERT INTO customers (name, address) VALUES (%s, %s)" val = ("John", "Highway 21") mycursor.execute(sql, val)  mydb.commit()  print(mycursor.rowcount, "record inserted.") </pre>	<p><b>Select</b></p> <pre> import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword",     database="mydatabase" )  mycursor = mydb.cursor()  mycursor.execute("SELECT name, address FROM customers")  myresult = mycursor.fetchall()  for x in myresult:     print(x) </pre>
<p><b>Delete</b></p> <pre> import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword",     database="mydatabase" )  mycursor = mydb.cursor()  sql = "DELETE FROM customers WHERE address = 'Mountain 21'"  mycursor.execute(sql)  mydb.commit()  print(mycursor.rowcount, "record(s) deleted") </pre>	<p><b>Update</b></p> <pre> import mysql.connector  mydb = mysql.connector.connect(     host="localhost",     user="yourusername",     password="yourpassword",     database="mydatabase" )  mycursor = mydb.cursor()  sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"  mycursor.execute(sql)  mydb.commit()  print(mycursor.rowcount, "record(s) affected") </pre>

## Searches and sorts data

Searching is the algorithmic process of finding a particular item in a collection of items. A search typically answers either True or False as to whether the item is present. In Python, there is a very easy way to ask whether an item is in a list of items. We use the in operator.

```
>>> "x" in "banana"
False
>>> "x" not in "banana"
True
>>> 1 in [1,2,3,4,5,6]
True
```

### Sequential Search

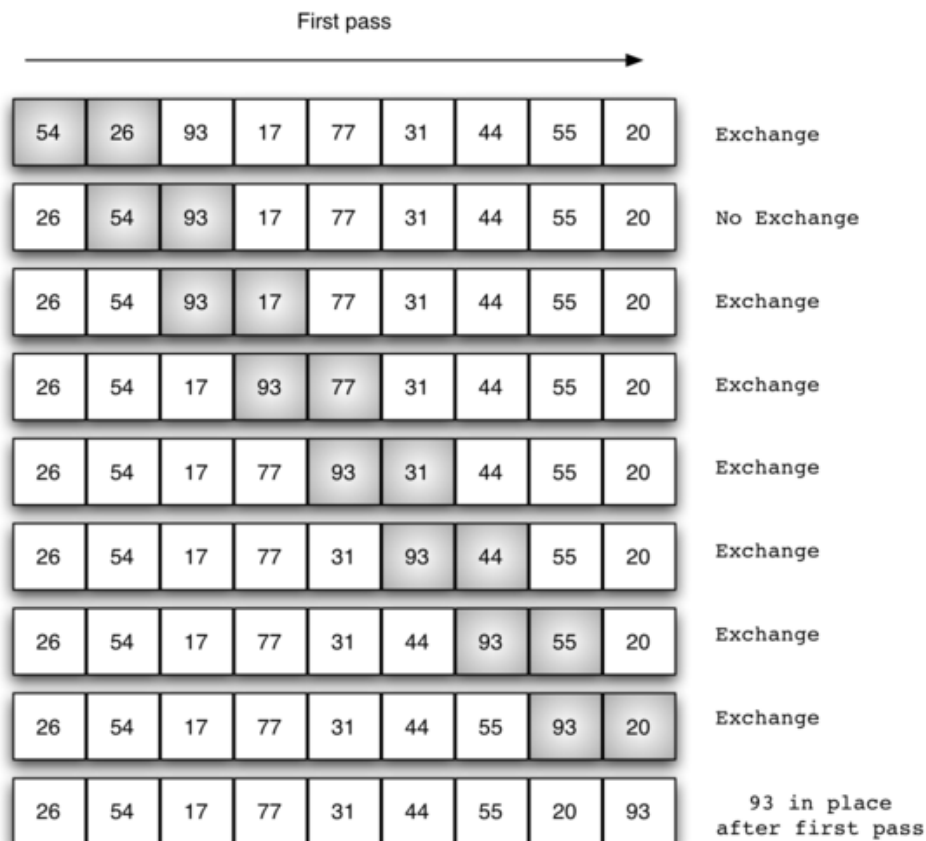
When data items are stored in a collection such as a list, they have a linear or sequential relationship. Each data item is stored in a position relative to the others. In Python lists, these relative positions are the index values of the individual items. Since these index values are ordered, it is possible to visit them in sequence. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data structure.

**Sorting Techniques:** Sorting refers to arranging data in a particular format. It can be ascending or descending order. Sorting algorithm specifies the way to arrange data in a particular order.

**Bubble sort:** The bubble sort makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item “bubbles” up to the location where it belongs.

### Sorting list in ascending order:

- ✔ compare 1st and 2nd elements
- ✔ if 1st is larger than the 2nd, swap
- ✔ compare 2nd and 3rd, and swap if necessary
- ✔ continue until compare the last two elements
- ✔ largest element is now at the last element in the array
- ✔ repeat starting from the beginning until no swaps are needed (i.e.the list is sorted)
- ✔ each time you travel the array, elements bubbling up the largest element to the end of the array



At the start of the second pass, the largest value is now in place. There are  $n-1$  items left to sort, meaning that there will be  $n-2$  pairs ( $n$  is a number of elements in the list). Since each pass places the next largest value in place, the total number of passes necessary will be  $n-1$ . After completing the  $n-1$  passes, the smallest item must be in the correct position with no further processing required.

## Bubble Sort as a Python Function

```
def bubble_sort(L):
    swapped = True # set flag to True to repeat sorting
    while swapped:
        swapped = False
        for i in range(len(L) - 1):
            if L[i] > L[i + 1]:
                # Swap the elements
                L[i], L[i + 1] = L[i + 1], L[i]
                # Set the flag to True so we'll loop again
                swapped = True
```