Fundamental Ideas of Computer Science:

Basic fundamental ideas of computer science are

- Algorithms
- Information Processing

Algorithms:

The sequence of steps that describes the computational processes is called an **algorithm**.

It provides a set of instructions that tells us how to do something, such as bake bread, or put together a piece of furniture.

An algorithm describes a process that ends with a solution to a problem.

An algorithm has the following features:

- An algorithm consists of a finite number of instructions
- Each individual instruction in an algorithm is well defined. This means that the action described by the instruction can be performed effectively. Example an algorithmic step that says “compute the difference between two digits” would be well defined. On the other hand, a step that says “divide a number by 0” is not well defined, because no computing agent could carry it out
- An algorithm describes a process that eventually halts after arriving at a solution to a problem. For example, the process of subtraction halts after the computing agent writes down the difference between the two digits in the leftmost column of digits.
- An algorithm solves a general class of problems.

Information Processing:

- In the modern world of computers, information is also commonly referred to as **data**.
- Like mathematical calculations, information processing can be

described with algorithms.

- In carrying out the instructions of any algorithm, a computing agent manipulates information. The computing agent starts with some given information (known as **input**), transforms this information according to well-defined rules, and produces new information, known as **output**.
- It is important to recognize that the algorithms that describe information processing can also be represented as information.

The Structure of a Modern Computer System:

A modern computer system consists of **hardware** and **software**. Hardware consists of the physical devices required to execute algorithms. Software is the set of these algorithms, represented as **programs**, in particular **programming languages.**

Computer Hardware

The basic hardware components of a computer are **memory**, a **central processing unit(CPU)**, and a set of **input/output devices**, as shown in the below fig.

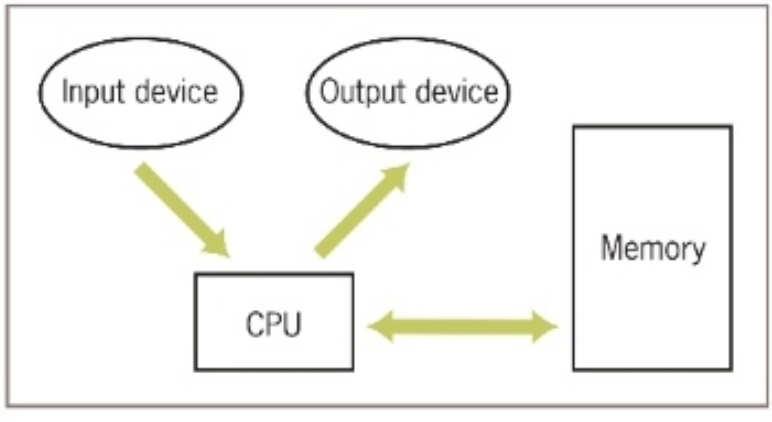


Fig: Hardware components of a modern computer system

Human users primarily interact with the input and output devices.

The input devices include a keyboard, a mouse, a trackpad, a microphone, and a touchscreen. Common output devices include a

monitor and speakers.

The purpose of most input devices is to convert information that human beings deal with, such as text, images, and sounds, into information for computational processing.

The purpose of most output devices is to convert the results of this processing back to human-usable form.

Computer memory is set up to represent and store information in electronic form. Specifically, information is stored as patterns of **binary digits** (1s and 0s)

To understand how this works, consider a basic device such as a light switch, which can only be in one of two states, on or off. Now suppose there is a bank of switches that control 16 small lights in a row. By turning the switches off or on, we can represent any pattern of 16 binary digits (1s and 0s).

Let us consider a tiny model of computer memory. The memory has 8 cells, each of which can store 16 **bits** of binary information. A diagram of this model, in which the memory cells are filled with binary digits, is shown in below fig. This memory is also sometimes called **primary** or **internal** or **random access memory (raM)**.

Cell 7	1	1	0	1	1	1	1	0	1	1	1	1	1	1	0	1
Cell 6	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1
Cell 5	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1
Cell 4	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1
Cell 3	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1
Cell 2	0	0	1	1	1	1	0	1	1	1	0	1	1	1	0	1
Cell 1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1
Cell 0	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1	0

Fig: A model of computer memory

The information stored in memory can represent any type of data, such as numbers, text,
images, or sound, or the instructions of a program.

The information that is stored in a memory is processed by using

CPU(central processing unit). It is also called as **processor**, consists of electronic switches arranged to perform simple logical, arithmetic, and control operations.

The processor can locate data in a computer's primary memory very quickly. However, these data exist only as long as electric power comes into the computer. If the power fails or is turned off, the data in primary memory are lost.

The permanent type of memory is called **external** or **secondary memory**, and it comes in several forms. **Magnetic storage media**, such as tapes and hard disks, allow bit patterns to be stored as patterns on a magnetic field. **Semiconductor storage media**, such as flash memory sticks and **optical storage media**, such as CDs and DVDs.

Computer Software:

Any programs that are stored in memory so that they can be executed later are called software.

A program stored in computer memory must be represented in binary digits, which is also known as **machine code**.

A loader takes a set of machine language instructions as input and loads them into the appropriate memory locations.

System Software(Operating System):

A loader exists, we can load and execute other programs that make the development, execution, and management of programs easier. This type of software is called **system software**. The most important example of system software is a computer's **operating system**.
Ex: Linux, Apple's macOS, and Microsoft's Windows

An operating system is responsible for managing and scheduling several concurrently running programs.

It also manages the computer's memory, including the external storage, and manages communications between the CPU, the input/output devices, and other computers on a network.

An important part of any operating system is its **file system**, which allows human users to organize their data and programs in permanent storage.

Another important function of an operating system is to provide **user interfaces**—using this users interact with the computer's software.

A **terminal-based interface** accepts inputs from a keyboard and displays text output on a monitor screen.

A **graphical user interface (GUI)** organizes the monitor screen around the metaphor of a desktop, with windows containing icons for folders, files, and applications. This type of user interface also allows the user to manipulate images with a pointing device such as a mouse.

A **touchscreen interface** supports more direct manipulation of these visual elements with gestures such as pinches and swipes of the user's fingers.

Application Software:

Another major type of software is called **applications software**, or simply **apps**.

An application is a program that is designed for a specific task, such as editing a document or
displaying a Web page.

Applications include Web browsers, word processors, spreadsheets, database managers, graphic design packages, music production systems, and games, among millions of others.

Coding Process:

Computer hardware can execute only instructions that are written in binary form—that is, in machine language. Writing a machine language program, however, would be an extremely tedious, error-prone task.

To ease the process of writing computer programs, computer scientists have developed **high-level programming languages** for expressing algorithms.

These languages resemble English and allow the author to express algorithms in a form that other people can understand.

A programmer starts writing high-level language statements in a **text editor.**

The programmer then runs another program called a **translator** to

convert the high-level
program code into executable code.

The translator checks for **syntax errors** before it completes the translation process. If it detects any of these errors, the translator alerts the programmer via error messages.

If the translation process succeeds without a syntax error, the program can be executed by the **run-time system**.

The run-time system might execute the program directly on the hardware or run yet another program called an **interpreter** or **virtual machine** to execute the program.

The below figure shows the steps and software used in the coding process.

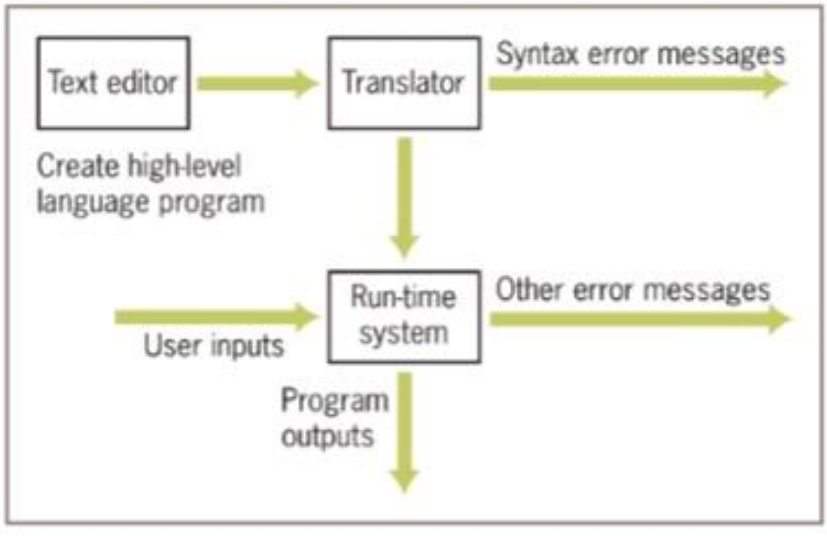


Fig: Software used in the coding process

History of Python:

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.

- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
    - ABC language.
    - Modula-3

Why the Name Python?

- There is a fact behind choosing the name Python. **Guido van Rossum** was reading the script of a popular BBC comedy series "**Monty Python's Flying Circus**". It was late on-air 1970s.
- Van Rossum wanted to select a name which unique, sort, and little-bit mysterious. So he decided to select naming Python after the **"Monty Python's Flying Circus"** for their newly created programming language.
- The comedy series was creative and well random. It talks about everything. Thus it is slow and unpredictable, which made it very interesting.
- Python is also versatile and widely used in every technical field, such as Machine Learning, Artificial Intelligence, Web Development, Mobile Application, Desktop Application, Scientific Calculation, etc.

Python Features

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type **print("Hello World")**. It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source

Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

10) Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

How to Install Python (Environment Set-up)

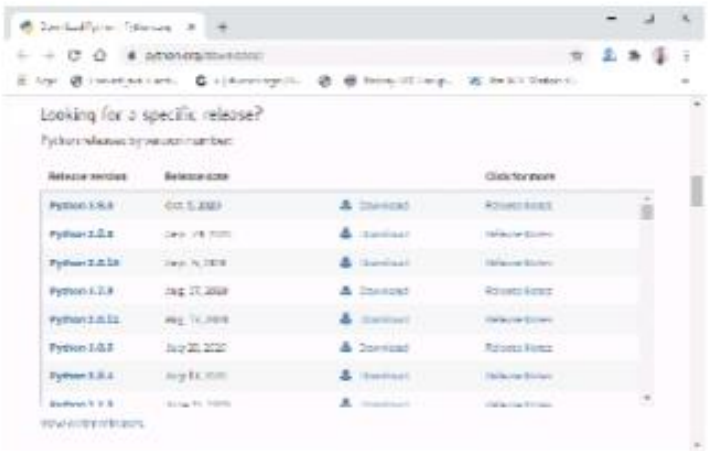
In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer. In this tutorial, we will discuss the installation of Python on various operating systems.

Installation on Windows

Visit the link *https://www.python.org/downloads/* to download the latest release of Python. In this process, we will install Python 3.8.6 on our Windows operating system. When we click on the above link, it will bring us the following page.

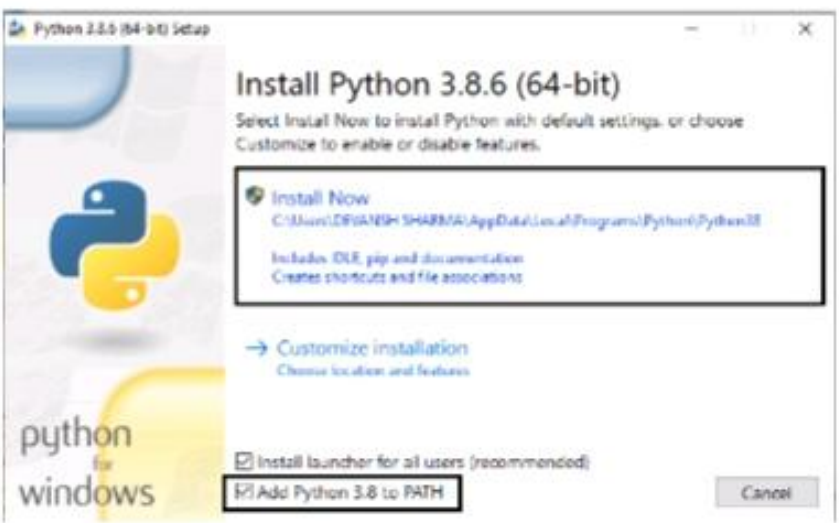
Step - 1: Select the Python's version to download.

Click on the download button.



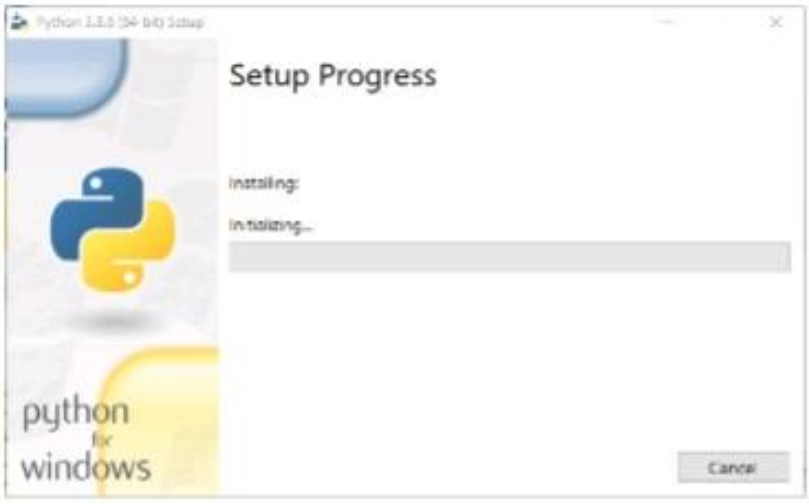
Step - 2: Click on the Install Now

Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.

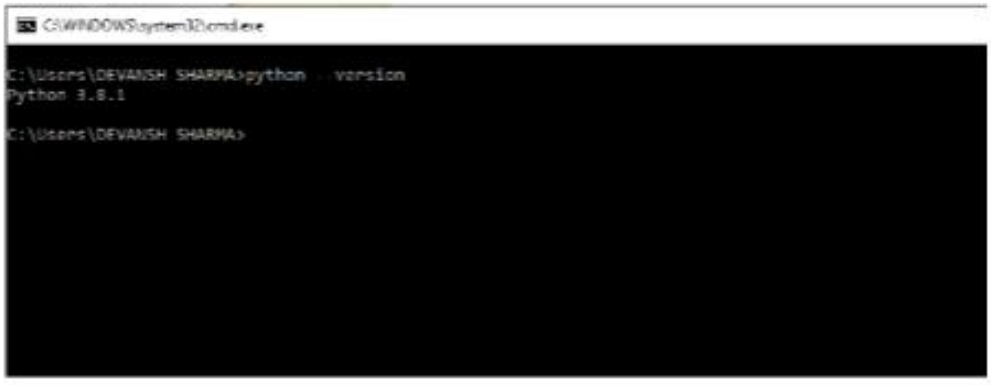


We can also click on the customize installation to choose desired location and features. Other important thing is install launcher for the all user must be checked.

Step - 3 Installation in Process



Now, try to run python on the command prompt. Type the command python -version in case of python3.



We are ready to work with the Python.

Python Interactive Shell:

Python is an interpreted language, and you can run simple Python expressions and statements in an interactive programming environment called the **shell**.

The easiest way to open a Python shell is to launch the IDLE (Integrated DeveLopment Environment). This is an integrated program development environment that comes with the Python installation.

The below image shows python interactive shell in windows operating system.

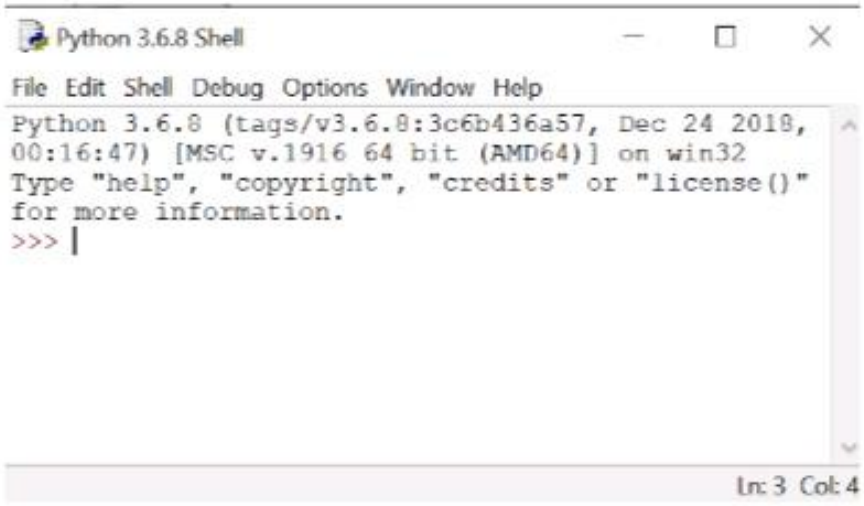


Fig: Python Shell

A shell window contains an opening message followed by the special symbol >>>, called
a shell prompt. The cursor at the shell prompt waits for you to enter a Python command.

We can get immediate help by entering **help** at the shell prompt or selecting **Help** from the window's drop-down menu.

The next few lines show the evaluation of several expressions and statements.

```
>>> 3 + 4                          # Simple arithmetic
7
>>> 3                              # The value of 3 is
3
>>> "Python is really cool!"       # Use a string for text
'Python is really cool!'
>>> name = "Ken Lambert"           # Give a variable a value
>>> name                           # The value of name is
'Ken Lambert'
>>> "Hi there, " + name            # Create some new text
'Hi there, Ken Lambert'
>>> print('Hi there')              # Output some text
Hi there
>>> print("Hi there,", name)       # Output two values
Hi there, Ken Lambert
```

Input, Processing, and Output:

Output Function:

The programmer can force the output of a value by using the **print** function. The simplest form for using this function looks like the following:

```
>>>print(<expression>)
```

This example shows you the basic **syntax** (or grammatical rule) for using the **print** function.

When running the **print** function, Python first evaluates the expression and then displays its value.

```
>>> print ("Hi there")
Hi there
```

In this example, the text **"Hi there"** is the text that we want Python to display.

We can also write a **print** function that includes two or more expressions separated by
commas.

The syntax for a **print** statement with two or more expressions looks like the following:

```
print(<expression>,..., <expression>)
```

print function always ends its output with a **newline**. It displays the values of the expressions, and then it moves the cursor to the next line on the console window.

To begin the next output on the same line as the previous one, you can place the expression **end = ""**, which says "end the line with an empty string instead of a newline".

```
print(<expression>, end = "")
```

Input function:

If your programs to ask the user for input. You can do this by using the **input** function.

This function causes the program to stop and wait for the user to enter a value from the keyboard. When the user presses the return or enter key, the function accepts the input value and makes it available to the program.

The following example receives an input string from the user and saves it for further processing.

```
>>> name = input("Enter your name: ")
Enter your name: Ken Lambert
>>> name
'Ken Lambert'
>>> print(name)
Ken Lambert
>>>
```

The **input** function does the following:
1. Displays a prompt for the input. In this example, the prompt is **"Enter your name: "**.
2. Receives a string of keystrokes, called characters, entered at the keyboard and
returns the string to the shell

The string returned by the function in our example is saved by assigning it to the variable
name. The form of an assignment statement with the **input** function is the following:

<variable identifier> = input(<a string prompt>)

A **variable identifier**, or **variable** for short, is just a name for a value. When a variable receives its value in an input statement, the variable then refers to this value.

The **input** function always builds a string from the user's keystrokes and

returns it to the program. After inputting strings that represent numbers, the programmer must convert them from strings to the appropriate numeric types.

In Python, there are two **type conversion functions** for this purpose, called **int** (for integers) and **float** (for floating point numbers).

The below code illustrates the use of type conversion functions.

```
>>> first = int(input("Enter the first number: "))
Enter the first number: 23
>>> second = int(input("Enter the second number: "))
Enter the second number: 44
>>> print("The sum is", first + second)
The sum is 67
```

Editing, Saving, and Running a Script:

It is more convenient to compose, edit, and save longer, more complex programs in files. We can run these program files or scripts either within IDLE or from the operating system's command prompt.

To compose and execute programs you need to perform the following steps:

- Select the option **New Window** from the **File** menu of the shell window.
- In the new window, enter Python expressions or statements on separate lines, in the order in which you want Python to execute them.
- At any point, you may save the file by selecting File/Save. If you do this, you should use a **. py** extension. For example, your first program file might be named **myprogram.py**
- To run this file of code as a Python script, select **Run Module** from the **Run** menu or press the F5 key.

The below sample code is used to find the area of a triangle.

```
area.py - E:\nag_python\area.py (3.6.8)
File Edit Format Run Options Window Help
base = int(input("Enter a base value"))
height = int(input("Enter a height value"))

area = 0.5 * base * height

print("The area of a triangle is",area)
```

When the script is run from the IDLE window, it produces the interaction with the user in
the shell window shown below.

```
Python 3.6.8 Shell
File Edit Shell Debug Options Window Help
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 201
(AMD64)] on win32
Type "help", "copyright", "credits" or "license(
>>>
---------------------- RESTART: E:\nag_python\a
Enter a base value4
Enter a height value6
The area of a triangle is 12.0
>>>
```

Python Program Execution Process:

Whether you are running Python code as a script or interactively in a shell, the Python interpreter performs the following steps.

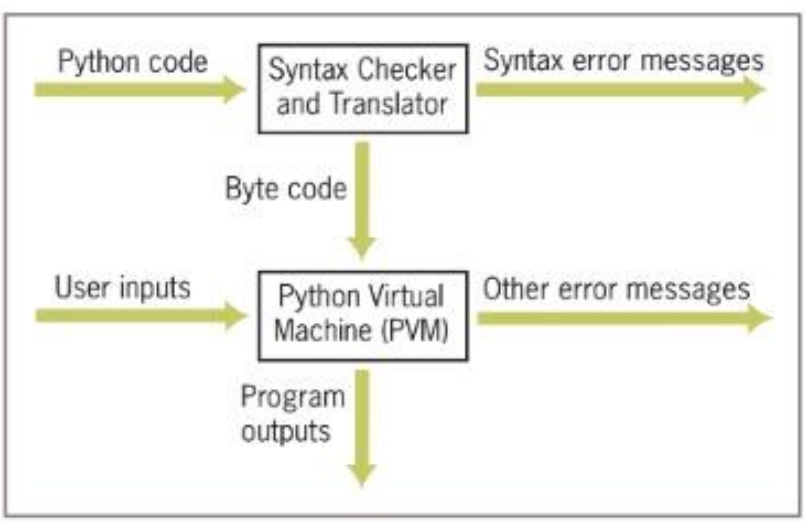


Fig: Steps in interpreting a Python program

- The interpreter reads a Python expression or statement, also called the **source code**, and verifies that it is well formed. Interpreter rejects any sentence that was not followed the grammar rules, or syntax, of the language. As soon as the interpreter encounters such an
error, it halts translation with an error message
- If a Python expression is well formed, the interpreter then translates it to an equivalent form in a low-level language called **byte code**. When the interpreter runs a script, it completely translates it to byte code.
- This byte code is next sent to another software component, called the **python virtual machine (PVM)**, where it is executed. If another error occurs during this step, execution also halts with an error message.

Data types:

A **data type** consists of a set of values and a set of operations that can be performed on those values. A **literal** is the way a value of a data type looks to a programmer.

When the Python interpreter evaluates a literal, the value it returns is simply that literal.

Type of Data	Python Type Name	Example Literals
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', "66"`

Fig: Literals for some Python data types

The first two data types listed in the above table, **int** and **float**, are called **numeric data types**, because they represent numbers.

String Literals

A string literal is a sequence of characters enclosed in single or double quotation marks.

```
>>> 'Hello there!'
'Hello there!'
>>> "Hello there!"
'Hello there!'
>>> ''
''
>>> ""
''
```

The last two string literals (" and "") represent the **empty string**. Although it contains no characters, the empty string is a string nonetheless. Note that the empty string is different from a string that contains a single blank space character, " ".

Double-quoted strings are handy for composing strings that contain single quotation marks or apostrophes. Here is a self-justifying example:

```
>>> "I'm using a single quote in this string!"
"I'm using a single quote in this string!"
>>> print("I'm using a single quote in this string!")
I'm using a single quote in this string!
```

To output a paragraph of text that contains several lines, you could use a separate **print** function call for each line. However, it is more convenient to enclose the entire string literal, line breaks and all, within three consecutive quotation marks (either single or double) for printing.

```
>>> print("""This very long sentence extends
all the way to the next line.""")
This very long sentence extends
all the way to the next line.
```

Escape Sequences:

Escape sequences are the way Python expresses special characters, such as the tab, the newline, and the backspace (delete key), as literals. The below table lists some escape sequences in Python.

Escape Sequence	Meaning
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

Table: Escape Sequences

Variables and the Assignment Statement:

A **variable** associates a name with a value, making it easy to remember and use the value later in a program.

You need to follow few rules when choosing names for your variables. Those are:

- some names, such as **if, def**, and **import**, (Keywords)are reserved for other purposes and thus cannot be used for variable names.
- A variable name must begin with either a letter or an underscore (_), and can contain any number of letters, digits, or other underscores.
- Python variable names are case sensitive; thus, the variable **WEIGHT** is a different name from the variable **weight**.
- The variable names that consist of more than one word, it's common to begin each word in the variable name (except for the first one) with an uppercase letter. This makes the variable name easier to read. For example, the name **interestRate** is slightly easier to read than the name **interestrate**.
- Programmers use all uppercase letters for the names of variables that contain values that the program never changes. Such variables are known as **symbolic constants**. Examples of symbolic constants in the tax calculator case study are **TAX_RATE** and **STANDARD_DEDUCTION.**

Variables receive their initial values and can be reset to new values with

an **assignment statement**.

The simplest form of an assignment statement is the following:

<variable name> = <expression>

The notation **<variable name>** stands for any Python variable name, such as **totalIncome** or **taxRate**.

The notation **<expression>** stands for any Python expression, such as **" " * 10 + "Python"**.

The Python interpreter first evaluates the expression on the right side of the assignment symbol and then binds the variable name on the left side to this value. When this happens to the variable name for the first time, it is called **defining** or **initializing** the variable.

The '**=**' symbol means assignment, not equality. After you initialize a variable, subsequent uses of the variable name in expressions are known as **variable references**.

When the interpreter encounters a variable reference in any expression, it looks up the associated value. If a name is not yet bound to a value when it is referenced, Python signals an error.

```
>>> firstName = "Ken"
>>> secondName = "Lambert"
>>> fullName = firstName + " " + secondName
>>> fullName
'Ken Lambert'
```

The first two statements initialize the variables **firstName** and **secondName** to string values. The next statement references these variables, concatenates the values referenced by the variables to build a new string, and assigns the result to the variable **fullName** ("concatenate" means "glue together"). The last line of code is a simple reference to the variable **fullName**, which returns its value.

Numeric Data types and Character Sets:

The brief overview of numeric datatypes and their cousins, **character sets**.

Integers:

The **integers** include 0, the positive whole numbers, and the negative whole numbers.

Integer literals in a Python program are written without commas, and a leading negative sign indicates a negative value.

Although the range of integers is infinite, a real computer's memory places a limit on the magnitude of the largest positive and negative integers.

The most common implementation of the **int** data type in many programming languages consists of the integers from -2,147,483,648 (2^{31}) to 2,147,483,647 (2^{31} -1).

However, the magnitude of a Python integer is much larger and is limited only by the memory of your computer.

Floating-Point Numbers:

A real number in mathematics, such as the value of p (3.1416...), consists of a whole number, a decimal point, and a fractional part. Real numbers have **infinite precision**, which means that the digits in the fractional part can continue forever.

Because a computer's memory is not infinitely large, a computer's memory limits not only the range but also the precision that can be represented for real numbers.

Python uses **floating-point** numbers to represent real numbers. Values of the most common implementation of Python's **float** type range from approximately -10^{308} to 10^{308} and have 16 digits of precision.

A floating-point number can be written using either ordinary **decimal notation** or **scientific notation**. Scientific notation is often useful for mentioning very large numbers.

Decimal Notation	Scientific Notation	Meaning
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e–1	3.78×10^{-1}
0.00378	3.78e–3	3.78×10^{-3}

Table: Decimal and scientific notations for floating-point numbers

Character Sets:

Some programming languages use different data types for strings and individual characters. In
Python, character literals look just like string literals and are of the string type.

All data and instructions in a program are translated to binary numbers before being run on a real computer.

To support this translation, the characters in a string each map to an integer value. This mapping is defined in character sets, among them the **aSCII set** and the **unicode set**. (The term ASCII stands for American Standard Code for Information Interchange.)

The below table shows the mapping of character values to the first 128 ASCII codes. The digits in the left column represent the leftmost digits of an ASCII code, and the digits in the top row are the rightmost digits.

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	I	j	k	l	m
11	n	o	P	q	r	S	t	u	v	w
12	X	y	z	{	|	}	~	DEL		

Table: The original ASCII character set

Python's **ord** and **chr** functions convert characters to their numeric ASCII codes and back again, respectively.

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
'A'
>>> chr(66)
'B'
```

Expressions:

Expressions provide an easy way to perform operations on data values to produce other data values.

When entered at the Python shell prompt, an expression's operands are evaluated, and its operator is then applied to these values to compute the value of the expression. In this section, we examine arithmetic expressions in more detail.

Arithmetic Expressions:

An **arithmetic expression** consists of operands and operators combined in a manner that is already familiar to you from learning algebra.

Below table shows several arithmetic operators and gives examples of how you might use them in Python code.

Operator	Meaning	Syntax
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

Fig: Arithmetic operators

The **precedence rules** you learned in algebra apply during the evaluation

of arithmetic expressions in Python:

- Exponentiation has the highest precedence and is evaluated first.
- Unary negation is evaluated next, before multiplication, division, and remainder.
- Multiplication, both types of division, and remainder are evaluated before addition and subtraction.
- Addition and subtraction are evaluated before assignment.
- With two exceptions, operations of equal precedence are left associative, so they are evaluated from left to right. Exponentiation and assignment operations are right associative, so consecutive instances of these are evaluated from right to left.
- You can use parentheses to change the order of evaluation

Expression	Evaluation	Value
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`(3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

Mixed-Mode Arithmetic and Type Conversions:

Performing calculations involving both integers and floating-point numbers is called **mixed-mode arithmetic**. For instance, if a circle has radius 3, you compute the area as follows:

```
>>> 3.14 * 3 ** 2
28.26
```

How does Python perform this type of calculation? In a binary operation on operands of different numeric types, the less general type (**int**) is temporarily and automatically converted to the more general type (**float**) before the operation is performed. Thus, in the example expression, the value 9 is converted to 9.0 before the multiplication.

You must use a **type conversion function** when working with the input of numbers. A type conversion function is a function with the same name as the data type to which it converts.
Because the **input** function returns a string as its value, you must use the function **int** or **float** to convert the string to a number before performing arithmetic

```
>>> radius = input("Enter the radius: ")
Enter the radius: 3.2
>>> radius
'3.2'
>>> float(radius)
3.2
>>> float(radius) ** 2 * 3.14
32.153600000000004
```

Conversion Function	Example Use	Value Returned
int(<a number or a string>)	int(3.77)	3
	int("33")	33
float(<a number or a string>)	float(22)	22.0
str(<any value>)	str(99)	'99'

Table: Python Type Conversion Functions

Another use of type conversion occurs in the construction of strings from numbers and other strings. For instance, assume that the variable **profit** refers to a floating-point number that represents an amount of money in dollars and cents. Suppose that, to build a string that represents this value for output, we need to concatenate the **$** symbol to the value of **profit**.

```
>>> profit = 1000.55
>>> print('$' + str(profit))
```

$1000.55

Python is a **strongly typed programming language**. The interpreter checks data types of all operands before operators are applied to those operands. If the type of an operand is not appropriate, the interpreter halts execution with an error message.

Program Comments and Docstrings:

A comment is a piece of program text that the computer ignores but that provides useful documentation to programmers.

The author of a program can include his or her name and a brief statement about the program's purpose at the beginning of the program file. This type of comment, called a **docstring**, is a multi-line string of the form.

```
"""
Program: circle.py
Author: Ken Lambert
Last date modified: 10/10/17

The purpose of this program is to compute the area of a
circle. The input is an integer or floating-point number
representing the radius of the circle. The output is a
floating-point number labeled as the area of the circle.
"""
```

In addition to docstrings, **end-of-line comments** can document a program. These comments begin with the # symbol and extend to the end of a line. An end-of-line comment might explain the purpose of a variable or the strategy used by a piece of code.

```
>>> RATE = 0.85 # Conversion rate for Canadian to US dollars
```

It's a good idea to do the following:

1. Begin a program with a statement of its purpose and other information that would help orient a programmer called on to modify the program at some future date.
2. Accompany a variable definition with a comment that explains the variable's purpose.
3. Precede major segments of code with brief comments that explain their purpose.
4. Include comments to explain the workings of complex or tricky sections of code.

Detecting and Correcting Syntax errors:

Programmers inevitably make typographical errors when editing programs, and the Python interpreter will nearly always detect them. Such errors are called syntax errors. The term *syntax* refers to the rules for forming sentences in a language.

When Python encounters a syntax error in a program, it halts execution with an error message.

```
>>> length = int(input("Enter the length: "))
Enter the length: 44
>>> print(lenth)
Traceback (most recent call last):
File "<pyshell#l>", line 1, in <module>
NameError: name 'lenth' is not defined
```

From the above statements, the statement'print(lenth)' attempts to print the value of the variable **lenth**. Python responds that this name is not defined.

The next statement attempts to print the value of the correctly spelled variable. However, Python still generates an error message.

```
>>>    print(length)
SyntaxError: unexpected indent
```

In this error message, Python explains that this line of code is unexpectedly indented. In fact, there is an extra space before the word **print**. Indentation is significant in Python code. Each line of code entered at a shell prompt or in a script must begin in the leftmost column, with no leading spaces.

The Python language is much simpler than other programming languages. Consequently, there are fewer types of syntax errors to encounter and correct, and a lot less syntax for you to learn!

In our final example, the programmer attempts to add two numbers, but forgets to include the second one:

```
>>> 3 +
SyntaxError: invalid syntax
```