

Advanced Business Decision Support Systems
Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imagineering Laboratory
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur
Lecture 26
Python Programming Language

Hello everyone, I welcome you all to the lecture series on Advanced Business Decision Support Systems. I am Dr. Prabal Pratap Singh from IIT Kanpur and I will be conducting the practical aspects of the Decision Support Systems by creating all the Decision Support Systems using the Python language. So, today we will start from the very basics of what Python programming is and more to that what is actually a programming language? Then, we will move to Python programming language.

Agenda

- Python Programming Language?? *- What is Python?*
- Programming Language??
- Elements of a Programming Language *components of a programming language*
- Classification of Programming Languages *on a system, what is the role of these compilers and interpreters*
- Compilers & Interpreters
- Python Language — Revisited

So, let us start with the agenda of this lecture. So, first we will see what is Python programming language, then after getting to know the very basics of this language, we will move towards what actually is a programming language.


So, that we can understand from where how many kinds of these programming languages are there and how python is different from those. Then, we learn about the elements of these programming languages like, what are the components of a Programming language. Then, we will try to see what are the different kinds of Programming languages by classifying them on various basis, mainly on the basis of the paradigm these programming languages follow. Then, we look at what compilers and interpreters do to run a code successfully on a particular system.

On a system what is the role of these compilers and interpreters? Finally, we will again look at the python language and we will try to see how these different things like Elements, Classifications of these programming languages. So, where does Python language stand in all these classifications and is it interpreted or is it using compiler behind the scenes. So, starting with what is Python programming language.

Python

Python is an interpreted, object-oriented high-level language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, makes it very attractive for rapid application development, as well as for use as a scripting language to connect existing components together.

[1]



Advanced Institute of Technology Kanpur
Dr. Prabal Pratap Singh
Dr. Anurag Singh
Dr. Prabal Pratap Singh

Dr. Prabal Pratap Singh Programming Languages 3 / 12

So, Python is an Object-Oriented high level language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding makes it very attractive for rapid application development as well as for use as a scripting to connect existing components together.

Now, after reading this statement you might be wondering, what are all these words like, what is interpreted? What is object oriented? what is high level and what is dynamic semantics? So, we will try to unwind all these terminologies in the next slides. So, that we can understand what is first a Programming language and then we will try to understand what the whole statement tells us about Python. So, this is a statement given by the developers and maintainers of this language. So, let us move forward.

Programming Language?

- * A system of notation for writing computer programs → set of instructions to a computer to perform a desired action
- * Can be textual or graphical
 - most known languages are textual in nature - C, Cobol, Python, etc.
 - Graphical languages manipulate the program flow using visual elements
- * Major components of a language -
 1. Syntax - Rules that govern the construction of a program that the code is grammatically correct.
 2. Semantic - (Meaning) - Graphical languages also check the meaning of the source code during compilation or execution.
- * Programming languages are a subset of Computer language.

Dr. Prabal Pratap Singh Programming Languages 4/12

So, let us start with the very basic concept, what is a Programming language? So, first of all, it is a system of notation for writing computer programs.

Now, what is a computer program? So, what is this? This is nothing but set of instructions to a computer to perform a desired action. So, this is the most basic definition of a Programming language that, it is a system of notations for writing various computer programs. So, the next thing about a computer programming language is that it can be textual or it can be graphical. Now, textual means that the whole source code is using the text form like, the Natural language or the Machine Code language. So, it is using only those things whereas, in a Graphical language, we can use entities and links between them.

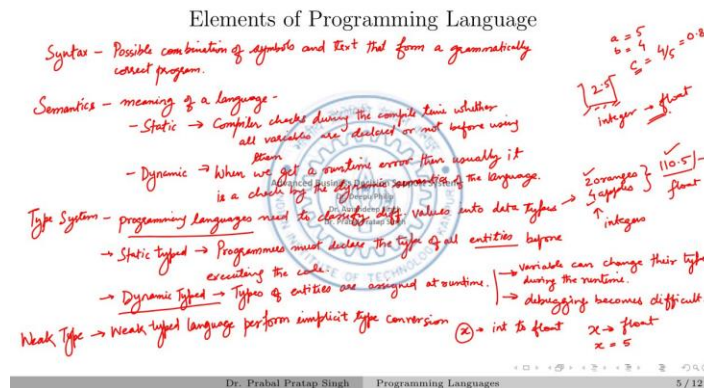
So, this is like the most known languages or textual in nature like C, Kobold, Python, etcetera. However, Graphical languages manipulate the program flow using visual elements. The next major thing is that the major components of a language are two, first is Syntax, second is Semantics. So, you may ask, what is the Syntax? So, these are nothing but the rules that govern two compilers or interpreters that the code is grammatically correct. However, semantics shows the meaning of code.

So, this is the compilers or interpreters also check the meaning of these four components. Source code during compile time or sometime, you may be wondering that, what is a Compiler or Interpreter? So, we will look about these two tools in the other slides, but for now, you can think of it like, as a tool that checks whether your code is grammatically correct and it has a valid meaning for the computers to perform your actions.

So, the last thing about programming languages is that, Programming languages are a subset of Computer languages. So, what does this Statement mean? It says that there are different kinds of Computer languages, Programming language is one of the kind and the other kind is a Markup language. So, a programming language tries to define some actions to a computer, whereas, Markup languages are usually used in the domain for

creating documents and maintaining all these documents like, a very popular Markup language is a LaTeX.

So, we use this kind of language to create various kind of documents, whereas Programming languages like Python, JavaScript, C, these are those kind of languages that try to define some actions for this processor to do.



So, next we will try to see, what are the basic Elements of a Programming language. So, first is the Syntax, as we have looked at earlier that, it contains the possible combination of symbols and text that form a grammatically correct program. Semantics is the next basic element. So, it shows the meaning of a language.

So, there can be two kinds of Semantics like, Static semantics. So, the tool like Compiler or Interpreter checks during the compile time, whether all variables are declared or not before using them. So, what do we mean by this statement is like, when we try to create a Source code, we tend to initiate different kind of variables to store different kind of data type.

Now, if we for some programming languages, we need to first declare the variable, then we need to initialize the variable. So, if we do not initialize a variable and directly try to compile it, then we may get an error by the compiler because the compiler does not know that a kind of variable exist in the system.

So, this is the Static semantics. Dynamic semantics also happens like, when we are running the code and if we are getting when we get a runtime error, then usually it is a check by the Dynamic semantics of the code. The next element is the type system. So, as I just said, that Programming languages need to classify different values into data types. For example, let us say, we have a bucket of fruits and the fruits cannot be a decimal value.

So, let us say, we have two oranges, four apples, but the price of these 2 plus 4, 6 fruits could be a decimal value. So, let us say, it is 110.5. So, these two values two and four are of different kind and this value is of different kind. So, a Programming language needs to first define what kind of variable the programmer is storing in a particular variable.

So, this for the programming languages, this is known as a float value and these are integers. So, there are different kind of data types. So, different kind of programming languages try to define their type systems differently. So, some kinds of languages are based on static type. What are these? In these kind of languages, programmers must declare the type of all entities.

Entities here is kind of variable. So, we are using a very general term here like entities before executing the code. The other kind of the type system followed by programming languages are dynamic type. So, in these languages, types of entities are assigned at run time. So, what does this mean is variables can change their type that is data type during the run time.

It means, let us say, here is a variable and we can think about a variable as a 'container'. So, let us say we are storing number two here. Now, as the code progresses, the programmer tries to overwrite this value with 2.5. So, what happened is initially, it was an integer, but after few progression, this container is holding a float value.

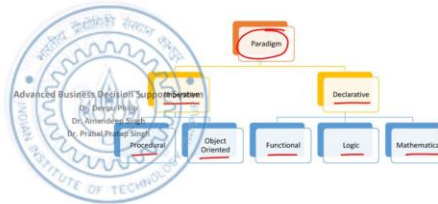
So, this can happen in a dynamic type language and thus, what happens is the debugging means, finding the error becomes very difficult. The last element is weakly typed or strong typed languages. So, let me start with an example here. So, let us say, you are storing an integer in variable a as 5 and another variable as 4. Now, these two a and b are integers.

However, if we divide and declare a c variable and initialize it with 4 by 5, the computation of 4 by 5, which is 0.8. So, this c variable holds a floating point value. So, what happened here is, weak typed languages perform implicit type conversions. So, what can happen is during the run time, the integer and a variable x can change from a integer to float without being explicitly told to it.

However, in strong typed languages, this does not happen until-unless your x variable is capable of storing a float value, it will only store the float values. Otherwise, it cannot store another type of value. So, if we have a variable like x and we declared it as if let us say, x has been declared as float. So, if we try to store value of 5 in it, then the strongly typed language will throw an error that it will the type does not match. So, these are few of the elements of Programming languages.

Classification of Programming Language

* Paradigm defines how a programmer builds the codebase.



So, let us move to Classification of Programming languages. Now, there are different kinds of classifications that are possible, but we are trying to focus on the classification based on the paradigm this. So, what does paradigm defines? Paradigm defines, how a programmer builds the code base. So, you may think that a programmer if a programmer is using a textual language, then he will or he or she will be using a natural language to build the code base, then how it is different from other things. So, the hierarchy looks like this.

The first differentiation happens like on the basis of paradigm, the programming languages can be classified into Imperative or Declarative. And, these Imperative can be divided further into two Procedural and Object-Oriented languages, whereas, Declarative can be classified as a Functional, Logical programming languages or Mathematical programming languages. So, next we will try to see what in detail, what are these different kinds of paradigms in programming languages?

Programming Paradigm

- Imperative → Program consists of commands for the computers to perform
 - Procedural → contains a step-by-step instructions for the computer to follow.
 - ↳ C, C++, etc.
 - Object-Oriented → encapsulate both data and method in an object.
 - Objects are instances of a class → blueprint of a particular system that programmer is imitating on computer.
- Declarative → Program contains the logic of computation without describing its control flow.
 - * Functional → Program contains functions that provide the logic. Functions are the first class citizens → functions can call other functions or can be returned by a function.
 - * Logic → Utilize formal logic statements to compute the output. Require predicates.
 - PROLOG.
 - * Mathematical → search available alternatives to select the best element based on given constraints → MATLAB.

So, starting with Imperative, these kind of programs consists of command for the computers to perform. So, the first kind of imperative paradigm is procedural. So, it contains step-by-step instructions for the computer to follow.

So, example of Procedural languages are like C, COBOL, etcetera. The next kind is Object-Oriented. So, in these kind of languages, what happens is, we encapsulate both data and method. Method is nothing but instructions or the processing or what we intend to do on data. So, we encapsulate both data and method in object.

So, here objects are nothing but instances of a Class. What is a Class? Class is a blueprint of a particular system that programmer is creating on computer. So, the next paradigm is declarative. Now, this is an interesting paradigm because here the program contains the logic of computation without describing its control flow. So, what happens here is like in Procedural languages, we have seen that everything happens by step-by-step instructions to the computer.

But during the Declarative programming, we only define the logic of computation, which means that we are only telling the computer to do the intended operation. We are not telling the computer how to do it. We are just telling what we need to accomplish. So, this is the major difference between Imperative and Declarative paradigm. So, the first kind of Declarative Paradigm programming languages is a functional programming language.

So, in these kind of languages, program contains functions that provide the logic and functions are the first class citizens. So, what does this statement means is, that function can call other functions or can be returned by a function. The other kind of Declarative programming language is logical. So, here we utilize formal logic statements to compute the output. So, if you are understanding about the predicate and formal logic, you can connect to this kind of Declarative Logical programming language.

Here, we always require predicate knowledge. An example of this kind of programming language is Prolog. This is an old language. The third and the last declarative programming language is Mathematical. In these languages, we try to search available alternatives, to select the best element based on given constraints and an example of this kind of language is MATLAB.

So, Dr. Amandeep has discussed the linear programming aspects in this course. And, to solve those kinds of problems, we can use the tools available by these mathematical languages.

Additional Classification

Machine Language → Low-level languages that gets interpreted directly by the hardware.

Assembly Language → Thin wrapper on machine language.

System Language → Designed for writing low-level tasks, memory management, etc.

Scripting Language → High-level language. Powerful and can perform different kinds of tasks.

General Purpose → Builds softwares for wide variety of application domains.

Domain specific language → Useful for specific purposes only.

Visual Language → Non-textual that utilizes entities and links to perform operations.

Dr. Prabal Pratap Singh Programming Languages 8/12

So, let us see some more classifications which are very common in the programming world like, there are some kinds of Machine languages. So, what is a Machine language? These are Low Level languages that gets interpreted by the hardware. The other language is Assembly language which is nothing but thin wrappers on Machine language.

The next is a System language. These languages were designed for writing low level tasks like, memory management, etcetera. Other kind of language is Scripting language. So, these are a kind of High Level language and they are usually powerful and can perform different kinds of programming or task. The other language is a General Purpose language. These builds software for wide variety of application domains.

Other is the opposite of general purpose language is a domain specific language which as the name suggests, used for specific purposes only. The last is a Visual language. These are as discussed non textual that utilizes entities and links to perform operations. These are different kinds of additional classifications. Now, here we have seen few new terms like Low Level languages and High Level languages.

What does these words mean is like, if we are using a system and the more a computer understands Machine language, the Binary language of 0 and 1. So, this is Machine language. Now, the closer Programming language is to this kind of Binary language. It is usually known as a Low Level language and we humans usually use Natural language. So, any programming language that is closer to this kind of language are High Level language.

Now, the computer does not understand High Level languages, the Natural languages which we try to write or speak. So, we need to convert these languages into Low Level languages. So, what happens is here compilers and interpreters perform these conversions.

Compilers & Interpreters

- * Role of compiler & interpreter is to convert high-level languages into binary codes.
- * Compiler convert the whole program into an executable whereas interpreters perform this conversion instruction-by-instruction.
- * Executable file generated by compiler does not require language compiler during the runtime, however, interpreters are needed during every execution of the program.
- * Compiler can catch some programming errors during compilation, whereas, interpreters can only catch an error during the execution of the program.
- * Compiled code runs extremely fast as compared to interpreted codes. Just-in-Time compilation JIT compiler.
- * An interpreted language always require the interpreter to be present in the system's RAM.
- * Compiler generate platform dependent executable file, however, interpreters can execute the same code on any machine's architecture.

So, let us see what are compilers and interpreters in detail. So, the role of compilers and interpreters is to convert languages into binary codes.

Compilers convert the whole program at once into an executable file whereas, interpreters perform this conversion instruction by instruction. The next thing about compilers and interpreters is the executable file that gets generated does not require language compiler during the run time. So, what it means is, when the compiler is doing this conversion, once it is done by a compiler, a file gets generated.

This file is, when we run this file, the compiler is not needed during the execution time. However, when we are doing this conversion using interpreters, they need to be available in the RAM of the system.

So, however, interpreters are needed during every execution of the program. So, another characteristic is that, compilers can catch some programming errors during compilation of the fourth code whereas, interpreters can only catch an error during the run time of the code.

So, during compilations, if we can get some errors when we are trying to code the required stuff, if we are getting some errors by the compilers, then we can debug the code at that time only. But during the interpreters, it will catch these kind of errors, when we try to run the code and it will only report the first error, will not report all the errors. So, because the execution is step by step, as we have already seen here, that the conversion happens instruction by instructions in interpreters.

So, interpreters will only report the errors as and when it finds the first error. The errors could be many more in the code, but the compilers can show you all the errors which can be identified during the compile time. So, the next characteristic of compilers and interpreters is compiled code runs extremely fast as compared to interpreted codes.

Why? Because the compilers try to create an executable file and this executable file will only have to run the code and it since all the syntax and semantic checks has already been done by the compiler. So, this executable file is a binary code file and it will only

run the code whereas, an interpreter will always checks the Syntax and Semantics at each run.

So, it takes time and therefore, this adds the overhead while running the code. However, these days, there is a new technology which is known as just in time compilation. So, these JIT compilers can reduce the time between a compiled code and interpreted code. An Interpreted language always require the interpreter to be present in the system's RAM.

So, this we have already discussed. Last thing is, compilers generate platform dependent executable file. However, inter interpreters can execute the same code on any machines architecture. So, why this is happening is because the compilers are trying to compile the code and then creating an executable file on a particular system. So, the particular system, the file generated by the compilers is only dependent on that system because the binary code has been generated for that system. So, the same executable file may not be workable on different kind of system.

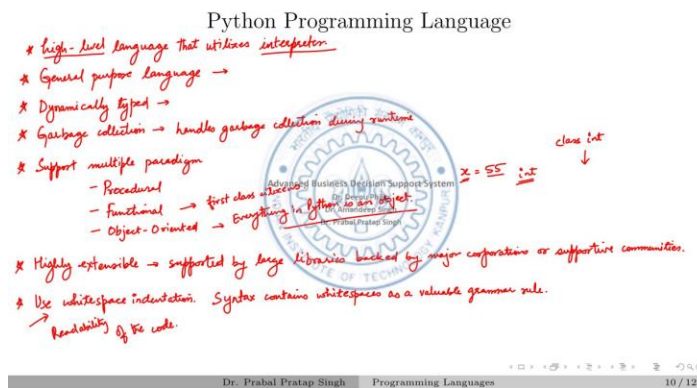
However, when we are using interpreters, what happens is, interpreters are always available in RAM otherwise, the code will not run and if the interpreter is available in RAM, then the code will get executed on any machine be it Linux, Mac or Windows anything.

Python Programming Language

- * High-level language that utilizes interpreter
- * General purpose language →
- * Dynamically typed →
- * Garbage collection → handles garbage collection during runtime
- * Support multiple paradigm
 - Procedural
 - Functional → first class functions
 - Object-Oriented → Everything in Python is an object.
- * Highly extensible → supported by large libraries backed by major corporations or supportive communities.
- * Use whitespace indentation. Syntax contains whitespace as a valuable grammar rule.
→ readability of the code.

class int
↓

= SS int



So, after learning all these terminologies, let us get back to what is Python programming language. So, first of all, it is a High Level language that utilizes interpreter. So, now, you understand that it is a high level language that means, we can code in natural language and there is an interpreter who will convert it into a machine language.

The next is it is a General Purpose language. So, we have already seen that these kind of languages can be used in different kind of domains and there are variety of applications

that can be performed with Python like, you may have seen that we can do image recognitions or we can also perform UI development. The next characteristic is it is dynamically typed. We have also seen it in earlier slides that, what is dynamically typed that, the programmer does not need to initialize and tell the interpreter, what kind of variable we are defining.

So, a particular variable can have different kind of data types during the program execution. So, this removes the burden from the programmer, but it has an overhead for the interpreter time to execute the complete code.

The next characteristic is that it has garbage collection. So, what happens is, let us say, we define different kind of variables and during the program execution, we do not need those many variables or the in between computations. So, the interpreter will itself determine whether, it can collect all those variables and remove from the memory. So, this way, the memory can be freed up. So, this also adds the overhead to the programming interpreter.

So, this feature handles garbage collection during runtime. Also, Python can support multiple paradigms. So, we have seen there are imperative and declarative paradigms. Now, this is a very special property of python that, it can support Procedural programming, and also support Functional programming and Object-Oriented as well. Also, functions are here, the first class citizens in python and everything in python is an object. So, what does this statement means is that, let us say, you define a variable `x = 55`.

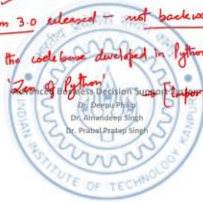
So, during the runtime, the interpreter will tell that, `x` holds the value 55 and it is a kind of integer value. So, the developers of the language has already defined a blueprint of this integer. That means, a class with name `int` has already been defined in the backend and the `x` variable becomes an object of this `int` class. So, either it is a variable or anything other else, the Python programming follows the Object-Oriented methodology.

So, everything becomes an object in python. It is also highly extensible supported by large libraries backed by major corporations or supportive communities. The last but not the least, it uses white space indentation. So, what happens is there were different kind of languages before Python and after Python, which try to develop the grammar of a language which we know as Syntax.

So, the Syntax of this python language syntax contains white space as a valuable grammar rule. So, when we look at the different statements in the python or the blocks of python, we can differentiate between different blocks by using indentation. So, usually, four blank spaces or a tab character show an indented block. So, this feature adds to the readability of the code.

Python Programming Language

1991 - Python 0.9.0 was released.
2000 - Python 2.0 released - End of life in 2020
2008 - Python 3.0 released - not backwards compatible
Philosophy of the codebase developed in Python gets its guidelines from the Zen of Python [import this] → Simple is better than complex.



Next, so let us deep dive into the history of the python programming language. So, in So, in 1991, the first version of python that is 0.9.0 was released. After that, in 2000, Python 2.0 was released. Recently, in 2020, this version has seen its end of life in 2020. In 2008, python 3.0 was released and this version was a very controversial release because it was not backwards compatible.

So, what does this statement means is that, there were some features in python 2.0 which programmers have used in their source code. Now, if they try to run using python 3.0, they may face different kinds of errors because the underlying code of the python interpreter was changed.

So, this problem was known as Not Backwards Compatible. The older versions cannot be run with this python version and the philosophy of the code base developed in Python, gets its guidelines from the 'Zen of Python'. So, when we will write the code, we will try to first see by writing this import this.

So, by writing this statement, the interpreter will give few lines of output which includes various kinds of philosophical statements about the python code. So, for example, simple is better than complex is a very famous statement. So, these kind of statements guides programmer to create a code base so that, anybody can understand it simply.

So, this is the reference for this whole lecture and why we are learning this python language is that, you have already seen that, professor Philip and Dr. Amandeep has discussed few models that need different kinds of computations. Now, these computations by hand gets very hectic. So, at that time, we need computers and their computing powers so that, we can handle large amount of data and we can get a good decision using those data. So, Python holds a very high value to create these kind of decisions in this Decision Support Systems and we will be going to develop few

different kinds of Decision Support Systems using Python and its capabilities. So, thank you and we will meet in the next lecture.