Introduction to Programming Using Python (MTA Exam 98-381)

- Fundamental Of Computer Programming
- Introduction to Python.

ISO 9001:2015 CERTIFIED IT COMPANY

MICROSPECTRA SOFTWARE TECHNOLOGIES PVT LTD

ISO 9001:2015 CERTIFIED IT COMPANY

Welcome to **Microspectra** formally called as **Microspectra Software Technologies Pvt. Ltd**.

MSTPL Corporate Identification Number is (CIN) U72200MH2015PTC267768 and its registration number is 267768. Microspectra Software Technologies is a Software development company Private Company incorporated on 26 August 2015.Well known for Software Solutions to Customized Software Applications such as ERP, CRM, Web Development, and Mobile APP Development. Microspectra is providing successful solutions to the clients and we are proud to say that our solutions are successfully installed to our direct or indirect clients. We have started to IT HUB in Vidarbha Region since year 2015. We are technically well versed with Industry Requirements and do proper analysis for the successful and reliable solutions for Clients. Microspectra aim is “Delivered quality solutions and customer satisfaction". We have happy and satisfied clients spread all over the world.

Introduction to Programming Using Python (Exam98-381)

Exam Type: - Multiple Type Questions (MCQ) No of questions:-45
Marks: - 100 Duration:-1hr.

Syllabus

Perform Operation Using Data Type and Operator.(20-25%)

Evaluate an expression to identify the data type Python will assign to each variable

- Identify str, int, float, and bool data types

Perform data and data type operations

- Convert from one data type to another type; construct data structures; perform indexing and slicing operations

Determine the sequence of execution based on operator precedence

- Assignment; Comparison; Logical; Arithmetic; Identity (is); Containment (in)

Select the appropriate operator to achieve the intended result

- Assignment; Comparison; Logical; Arithmetic; Identity (is); Containment (in)

Control Flow With Decision Loops(25-30%)

Construct and analyze code segments that use branching statements

- if; elif; else; nested and compound conditional expressions

Construct and analyze code segments that perform iteration

- while; for; break; continue; pass; nested loops and loops that include compound conditional expressions

Perform Input Output Operation(20-25%)

Construct and analyze code segments that perform file input and output operations

- Open; close; read; write; append; check existence; delete; with statement

Construct and analyze code segments that perform console input and output operations

- Read input from console; print formatted text; use of command line arguments

Document and structure code(15-20%)

Document code segments using comments and documentation strings

- Use indentation, white space, comments, and documentation strings; generate documentation by using pydoc

Construct and analyze code segments that include function definitions

- Call signatures; default values; return; def; pass

Perform Trouble Shooting and error handling(5-10%)

Analyze, detect, and fix code segments that have errors

- Syntax errors; logic errors; runtime errors

Analyze and construct code segments that handle exceptions

- Try; except; else; finally; raise

Perform Operations Using Modules and Tools(1-5%)

Perform basic operations using built-in modules

- Math; datetime; io; sys; os; os.path; random

Solve complex computing problems by using built-in modules

- Math; datetime; random

INDEX

1) Fundamental Of Computer Programming.

"Well begun is half done…. "

To those who are new in field of computer programming lots of question arises which we would briefly outline these issue. Following are some question through which we tried cover all beginners questioner

Q. How Does a Computer Program Work?

- This course aims to show you what the Python language is and what it is used for. Let's start from the absolute basics.
- A program makes a computer usable. Without a program, a computer, even the most powerful one, is nothing more than an object. Similarly, without a player, a piano is nothing more than a wooden box.
- Computers are able to perform very complex tasks, but this ability is not innate. A computer's nature is quite different.
- It can execute only extremely simple operations, e.g., a computer cannot evaluate the value of a complicated mathematical function by itself, although this isn't beyond the realms of possibility in the near future.
- Contemporary computers can only evaluate the results of very fundamental operations, like adding or dividing, but they can do it very fast, and can repeat these actions virtually any number of times.
- Imagine that you want to know the average speed you've reached during a long journey. You know the distance, you know the time, and you need the speed.
- Naturally, the computer will be able to compute this, but the computer is not aware of such things as distance, speed or time. Therefore, it is necessary to instruct the computer to:
    1. Accept a number representing the distance;
    2. Accept a number representing the travel time;
    3. Divide the former value by the latter and store the result in the memory;

    Display the result (representing the average speed) in a readable format.
- These four simple actions form a **Program**. Of course, these examples are not formalized, and they are very far from what the computer can understand, but they are good enough to be translated into a language the computer can accept.
- **Language** is the keyword.

Q. Can we Compare Natural Vs Programming Language?

- A language is a means (and a tool) for expressing and recording thoughts. There are many languages all around us. Some of them require neither speaking nor writing, such as body language; it's possible to express your deepest feelings very precisely without saying a word.
- Another language you use each day is your mother tongue, which you use to manifest your will and to think about reality. Computers have their own language, too, called **machine** language, which is very rudimentary.
- A computer, even the most technically sophisticated, is devoid of even a trace of intelligence. You could say that it is like a well-trained dog - it responds only to a predetermined set of known commands. The commands it recognizes are very simple. We can imagine that the computer responds to orders like "take that number, divide by another and save the result".
- A complete set of known commands is called **an instruction list**, sometimes abbreviated to **IL**. Different types of computers may vary depending on the size of their ILs, and the instructions could be completely different in different models.
- Note: machine languages are developed by humans.
- No Computer Currently Capable of Creating a new language but this scenario may change soon.

Q. What Makes a Language?

- We can say that each language (machine or natural, it doesn't matter) consists of the following elements:

AN ALPHABET

- A set of symbols used to build words of a certain language (e.g., the Latin alphabet for English, the Cyrillic alphabet for Russian, Kanji for Japanese, and so on)

LEXIS

- (aka a dictionary) a set of words the language offers its users (e.g., the word "computer" comes from the English language dictionary, while "cmoptrue" doesn't; the word "chat" is present both in English and French dictionaries, but their meanings are different)

SYNTAX

- a set of rules (formal or informal, written or felt intuitively) used to determine if a certain string of words forms a valid sentence (e.g., "I am a python" is a syntactically correct phrase, while "I a python am" isn't)

SEMANTICS

- a set of rules determining if a certain phrase makes sense (e.g., "I ate a doughnut" makes sense, but "A doughnut ate me" doesn't)

- The **IL** is, in fact, the **alphabet of a machine language**. This is the simplest and most primary set of symbols we can use to give commands to a computer. It's the computer's mother tongue.

- Unfortunately, this tongue is a far cry from a human mother tongue. We all (both computers and humans) need something else, a common language for computers and humans, or a bridge between the two different worlds.

- We need a language in which humans can write their programs and a language that computers may use to execute the programs, one that is far more complex than machine language and yet far simpler than natural language.

- Such languages are often called high-level programming languages. They are at least somewhat similar to natural ones in that they use symbols, words and conventions readable to humans. These languages enable humans to express commands to computers that are much more complex than those offered by ILs.

- A program written in **a high-level programming language** is called a **source code** (in contrast to the machine code executed by computers). Similarly, the file containing the source code is called **the source file**.

Q. What is Compilation Vs Interpretation?

- Computer programming is the act of composing the selected programming language's elements in the order that will cause the desired effect. The effect could be different in every specific case - it's up to the programmer's imagination, knowledge and experience.

- Of course, such a composition has to be correct in many senses:

- **alphabetically** - a program needs to be written in a recognizable script, such as Roman, Cyrillic, etc.
- **lexically** - each programming language has its dictionary and you need to master it; thankfully, it's much simpler and smaller than the dictionary of any natural language;
- **syntactically** - each language has its rules and they must be obeyed;
- **semantically** - the program has to make sense.
- Unfortunately, a programmer can also make mistakes with each of the above four senses. Each of them can cause the program to become completely useless.

- Let's assume that you've successfully written a program. How do we persuade the computer to execute it? You have to render your program into machine language. Luckily, the translation can be done by a computer itself, making the whole process fast and efficient.

- There are two different ways of transforming a program from a high-level programming language into machine language:

COMPILATION

- the source program is translated once (however, this act must be repeated each time you modify the source code) by getting a file (e.g., an .exe file if the code is intended to be run under MS Windows) containing the machine code; now you can distribute the file worldwide; the program that performs this translation is called a compiler or translator

INTERPRETATION

- you (or any user of the code) can translate the source program each time it has to be run; the program performing this kind of transformation is called an interpreter, as it interprets the code every time it is intended to be executed; it also means that you cannot just distribute the source code as-is, because the end-user also needs the interpreter to execute it.

- Due to some very fundamental reasons, a particular high-level programming language is designed to fall into one of these two categories.

- There are very few languages that can be both compiled and interpreted. Usually, a programming language is projected with this factor in its constructors' minds - will it be compiled or interpreted?

Q. What Does Interpreter actually do?

- Let's assume once more that you have written a program. Now, it exists as a computer file: a computer program is actually a piece of text, so the source code is usually placed in text files. Note: it has to be pure text, without any decorations like different fonts, colors, embedded images or other media. Now you have to invoke the interpreter and let it read your source file.
- The interpreter reads the source code in a way that is common in Western culture: from top to bottom and from left to right. There are some exceptions - they'll be covered later in the course.
- First of all, the interpreter checks if all subsequent lines are correct (using the four aspects covered earlier).
- If the compiler finds an error, it finishes its work immediately. The only result in this case is an error message. The interpreter will inform you where the error is located and what caused it. However, these messages may be misleading, as the interpreter isn't able to follow your exact intentions, and may detect errors at some distance from their real causes.

- For example, if you try to use an entity of an unknown name, it will cause an error, but the error will be discovered in the place where it tries to use the entity, not where the new entity's name was introduced.
- In other words, the actual reason is usually located a little earlier in the code, e.g., in the place where you had to inform the interpreter that you were going to use the entity of the name.
- If the line looks good, the interpreter tries to execute it (note: each line is usually executed separately, so the trio "read-check-execute" can be repeated many times - more times than the actual number of lines in the source file, as some parts of the code may be executed more than once).
- It is also possible that a significant part of the code may be executed successfully before the interpreter finds an error. This is normal behavior in this execution model.
- You may ask now: which is better? The "compiling" model or the "interpreting" model? There is no obvious answer. If there had been, one of these models would have ceased to exist a long time ago. Both of them have their advantages and their disadvantages.

Compilation vs. interpretation - advantages and disadvantages

	Compilation	Interpretation
Advantages	the execution of the translated code is usually faster;	
	only the user has to have the compiler - the end-user may use the code without it;	you can run the code as soon as you complete it - there are no additional phases of translation;
	the translated code is stored using machine language - as it is very hard to understand it, your own inventions and programming tricks are likely to remain your secret.	the code is stored using programming language, not the machine one - this means that it can be run on computers using different machine languages; you don't compile your code separately for each different architecture.
Disadvantages	the compilation itself may be a very time-consuming process - you may not be able to run your code immediately after any amendment;	don't expect that interpretation will ramp your code to high speed - your code will share the computer's power with the interpreter, so it can't be really fast;
	you have to have as many compilers as hardware platforms you want your code to be run on.	both you and the end user have to have the interpreter to run your code.

Q. What does this all mean for you?

Python is **an interpreted language**. This means that it inherits all the described advantages and disadvantages. Of course, it adds some of its unique features to both sets. If you want to program in Python; you'll need the **Python interpreter**. You won't be able to run your code without it. Fortunately, **Python is free**. This is one of its most important advantages. Due to historical reasons, languages designed to be utilized in the interpretation manner are often called **scripting languages**, while the source programs encoded using them are called **scripts**.

2.Introduction to Python

"On Your Mark Get set go…."

Q. What is Python?

Python is a widely used

- **General Purpose:-** In computer software, a **general-purpose programming language** is a programming language designed to be used for writing software in the widest variety of application domains (a general-purpose language). A general-purpose programming language has this status because it does not include language constructs designed to be used within a specific application domain.
- **Interpreted**:- An **interpreted language** is a type of programming language for which most of its implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions.
- **object-oriented:-** **Object-oriented programming** (**OOP**) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as *attributes* or *properties*), and code, in the form of procedures (often known as *methods*).
- **high-level programming:-** A **high-level language** (HLL) is a **programming language** such as C, FORTRAN, or Pascal that enables a **programmer** to write programs that are more or less independent of a particular type of computer.
- **Dynamic semantics:-** is a perspective on natural language **semantics** that emphasizes the growth of information in time
- **scripting language:-** is a computer **language** with a series of commands within a file that is capable of being executed without being compiled.
- **Easy to learn** Programming Language.

Q. What Are Characteristics of Python?

Every programming Language is made of some Characteristics or which define them like people in programming world this characteristics are taken from other languages so lets check out for Characteristics of python and from where they are taken.

Following are some important Characteristics that every programming must have lets check them one by one.

- **Functional Programming From "C"**
- **Object Oriented Programming From "C++"**
- **Scripting Language Features From "Pearl"& "Shell"**
- **Modular Programming Features From "Modula-3"**
- **Syntax from "C" & "ABC" Language.**

Q. What are Features of Python?

Look out for each and every Language one uses in day today life Natural or Programming the basic reason people use it because of its Features so let's check out what features does Python provides.

1) Easy to Learn and Use
Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language
Python language is more expressive means that it is more understandable and readable.

3) Free and Open Source
Python language is freely available at official web address. The source-code is also available. Therefore it is open source.

4) 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.

5) Large Standard Library
Python has a large and broad library and provides rich set of module and functions for rapid application development.

6) Integrated
It can be easily integrated with languages like C, C++, JAVA etc.

7) Platform Independent
A term that describes a technology (usually a Programming Language or a Framework) that you can use to implement things on one machine and use them on another machine without (or with minimal) changes.

8) Portable
A **portable language** is a computer programming **language** capable of developing software for more than one computer system.

9) Both Procedure and Object Oriented
Gives best of two worlds very few languages are there that carry these Features.

Q. What are Limitations of Python?
With such a great Characteristics and Features there come some limitations.

1) Performance and Speed
Many studies have proved that Python is slower than other modern programming languages like Java and C++. So the developers have to frequently explore ways to enhance the Python application's speed.

2) Incompatibility of two versions
Beginners often find it pick and learn the right version of Python. Officially, Python 2.x is described as legacy, whereas Python 3.x is described as current and futuristic. But both versions of the programming language have been updated on a regular basis. Also, a large percentage of programmers still prefer Python 2 to Python 3. There are also a number of popular frameworks and libraries that support only Python 2.

3) Weak Mobile Computing
The steady decline in mobile web usage has made it essential for modern businesses to launch mobile apps. Often developers have to write mobile apps in a particular programming language according to the targeted mobile platform. For instance, they need to write iOS app in either Objective-C or Swift. Likewise, mobile apps for Android need to be written in Java. But the developers cannot use Python directly for developing mobile apps by targeting any popular mobile platforms. They have to use frameworks like Kivy to build cross-platform mobile apps using Python.

"Programs must be written for people to read, and only incidentally for machines to execute".

Q. What are Different Flavors of Python?

a. CPython
This is the most widely accepted implementation of Python. It is written in the language C, and is an interpreter.

b. Jython
Jython is a Python implementation written in Java. A Jython program can import any Java class. It compiles to Java bytecode.

c. IronPython
IronPython is implemented in C#. It can function as an extensibility layer to application frameworks written in a .NET language.

d. Brython

Brython stands for Browser Python. It is an implementation of Python that runs in the browser.

e. RubyPython
It acts as a bridge between the Python and Ruby interpreters. It marshals data between Python and Ruby virtual machines.

f. PyPy
It is interesting to know that PyPy is Python implemented in Python. This makes it faster and easier to experiment with. However, the standard implementation is CPython.

g. MicroPython
This is an implementation of Python meant to run on a microcontroller. It uses a MicroPython board that runs MicroPython on bare metal.

Q. What are different versions of Python?

In Python Community this question is very hot topic to discuss here we will discuss about the Core version of python for in depth analysis you can visit https://www.python.org/doc/versions/
This site is official site where you will get each and every detail of Python and there versions.

Version	Year Of Release
Python 1.0	Jan 1994
Python 2.0	Oct 2000
Python 3.0	Dec 2008
Python 3.8	Oct 2019

Q . Who Created Python? And why Name it as Python?

One of the amazing features of Python is the fact that it is actually one person's work. Usually, new programming languages are developed and published by large companies employing lots of professionals, and due to copyright rules, it is very hard to name any of the people involved in the project. Python is an exception.

There are not many languages whose authors are known by name. Python was created by Guido van Rossum, born in 1956 in Haarlem, the Netherlands. Of course, Guido van Rossum did not develop and evolve all the Python components himself.

The speed with which Python has spread around the world is a result of the continuous work of thousands (very often anonymous) programmers, testers, users (many of them aren't IT specialists) and enthusiasts, but it must be said that the very first idea (the seed from which Python sprouted) came to one head - Guido's.

And while you may know the python as a large snake, the name of the Python programming language comes from an **old BBC television comedy sketch** series called **Monty Python's Flying Circus.**

Since Monty Python is considered one of the two fundamental nutrients to a programmer (the other being pizza), Python's creator named the language in honor of the TV show.