Anaconda Installation:-

```
1. https://www.anaconda.com/products/individual
2. https://repo.anaconda.com/archive/
```

What is Programming Language and Operating System and why do we need them?

A **Programming Language** is a set of predefined words that are combined into a program according to predefined rules(syntax). Programming languages are used to develop the application which helps in turn to get our task done.

An **Operating System** is a software that allows a user to run other applications on a computing device. The main job of operating system is to manage computer's software and hardware resources like I/O device, Network device, storage device etc. and to run the application on our system and resource allocation required for the same.

A brief introduction to computer memory

```
There are mainly two types of memory in computer:-
    1. Volatile Memory - Primary Memory - Temporary Storage - RAM, Cache, Register
   2. Non-volatile memory - Secondary Memory - Permanent Storage - HDD, SDD, Floppy, CD, Pen Drive

-> Every program runs into RAM.
-> Secondary memory are used for permanent storage.
```

Types of Programming Languages

There are mainly two types of programming language.
1. **Low Level Programming Language**
   I. Machine Level Language or Binary Code
   II. Assembly Language
2. **High Level Programming Language**
   I. Based on purpose
      i. General Purpose PL
      ii. Special Purpose PL
   II. Based on converters used
      i. Compiled -> C/C++/Java/Python
      ii. Interpreted -> Perl/Java Script/Python

Sl No	Low Level Programming Language	High Level Programming Language
1	More closer towards system, less closer towards user	More closer towards user, less closer towards system
2	Need to know the underlying architecture of machine on which you are coding	No need to know the underlying architecture of machine on which you are coding
3	Bit complex to learn and write the code, not manageable when program grows and not ideal to develop large application	Easy to learn and write the code, easily manageable when program grows and ideal to develop large application
4	As a programmer you need to manage the memory and handle the security issues in your program	Compiler or VM is responsible to manage the memory and handle the security issues your program
5	Code is not portable.	Code is portable.
6	Not expressive	Expressive
7	e.g. Machine code, Assembly code	e.g. C, C++, Java, Python etc
8	Sample code	Sample code

What is Source code, Byte code and Machine code in Python?

Source Code:- The code, written to develop the application. In python, (.-)file is source code.

Byte code:- A fixed set of instructions that represents all operations like arithmetic, comparison, memory related operations etc. The code generated after the compilation of source code. (.pyc) file is byte code. Byte code is system and platform independent. It is also called p-code, portable code or object code.

PVM:- Stands for **Python Virtual Machine** and is a software which comes with Python. PVM comes with Interpreter.

Machine code:- The code generated after interpretation of byte code. It is system and platform dependent.

Type of Errors in Python:-

Error Handling is one of the most important feature of Python. It was main deciding factor in development of Python.
There are mainly two type of errors in Python:-
   I. Syntax Error caught at compilation stage
   II. Run-time Error caught at run time or interpretation stage.

Syntax & Syntactical Errors:- Every Programming Language comes with a set of rules which defines that how the codes will be written and interpreted or compiled. These set of rules are called **Syntax.** Not following the rules may lead to errors which are called **Syntax errors. In Python,** at compilation stage we check for the syntax errors. If there are no syntax errors in your program then it will be compiled successfully and byte code will be generated. If there are some syntax error in your program then compilation will not happen and byte code will not be generated. **Syntax Error happens when Python can't understand what you are saying.**

Run Time Errors:- Run time error happens when Python can understand what you are saying but runs into problem while running your instructions. To find the list of all run time error you may execute the below code:-

>>> print(dir("__builtins__"))

We can handle runtime errors using exception handling concept in Python unlike syntax errors which has to be rectified before compilation. Run time errors may cause partial implementation of your code.

[Reference:-]
1. https://cscircles.cemc.uwaterloo.ca/1e-errors/

Compiled Languages vs Interpreted Languages

I. **Compiled languages** needs **compiler** to **translate** the code whereas **Interpreted languages** needs **interpreter** to **translate** the code from one form to another.

II. **Compiler & Interpreter** both are **translator** and are **computer programs**.

III. **Compiler translates** the whole code from one form to another form **all at once** while **interpreter** is **instruction by instruction execution**.

IV. **In Python**, **Compiler** is used to translate **Source code to byte code** whereas **Interpreter** is used to translate the **Byte code to Machine code.**

V. All **Syntactical errors** are caught at **compilation** stage in Python whereas all **run time errors** are caught at **interpretation stage**.

NOTE:- Originally, Python is Interpreted Language. But over the time, Python is made compiled and Interpreted both as python has established itself as full fledged programming language and is acceptable across wide area of application.

[References:-]
1. https://en.wikipedia.org/wiki/Interpreted_language
2. https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/

Why do we need translator?

Computer's hardware are electronic devices and they can not understand human language, rather they communicate with each other with the help of **signals**. **Signals** are nothing but **binary code** and is completely low level language. To **translate** High level languages in human readable format to machine readable format(binary code), we need **translators**.

Why do we need Byte code?

Byte code are also called object code or p-code or portable code which may be be directly shipped to your client. As well as it is platform independent so it can run on any platform. Which makes service provider's and developer's life easy as they do not have to write the code for the same application to be used on different platform.

Why Python is compiled and interpreted both? Can't we make it either compiled or interpreted?

Traditional programming languages like C or C++ were compiled languages. If you execute any source code in C or C++, it first creates an executable file which is platform dependent and then you run this file to get the output on that particular platform.

Source Code -> Compiler -> Executable Code(.exe file) -> Output

You can see here that the intermediate code which are generated is not platform independent. But remember that the Source code written in these languages are portable.

So, to counter this problem, languages like **Java** and **Python** were developed which are first compiled to **object code or Byte code or p-code(portable code)**. These object codes are completely platform independent. Now when you move this object code to other platform then you need to translate it to machine code. For translation at this stage, you may either use compiler or interpreter, any translator which again translates the byte code to machine code. For Instance, **Java, PyPI, JPython** uses **JIT compiler**, where **CPython** uses **interpreter**.

Source Code -> Compiler -> Byte Code -> Interpreter or JIT compiler(PVM/JVM) -> Machine Code -> Output

So basically, we may say that to bridge the gap between portable programming languages and platform independent programming languages we came up with this idea to have two **translators**. First time it is used to generate shippable code and second time it is used to convert the byte code to machine code.

[References:-]
1. https://en.wikipedia.org/wiki/Python_(programming_language)
2. https://www.geeksforgeeks.org/history-of-python/
3. http://effbot.org/pyfaq/why-was-python-created-in-the-first-place.htm
4. https://www.artima.com/intv/python.html

What is Python?

Python is General purpose, High level programming language.

What are some good features of Python?

1. Simple & easy to learn:- Python is one of the simplest language ever. Syntaxes are simple, easy to remember and quite expressive. When it comes to learning, it has been found that the learning curve for python is quite steeper compared to other programming languages.

Sample Program in Java to add two numbers:-

```
public class add{
    public static void main(string[]args){
        int a = 10;
        int b = 20;
        int c = a+b;
        System.out.println(c);
    }
}
```

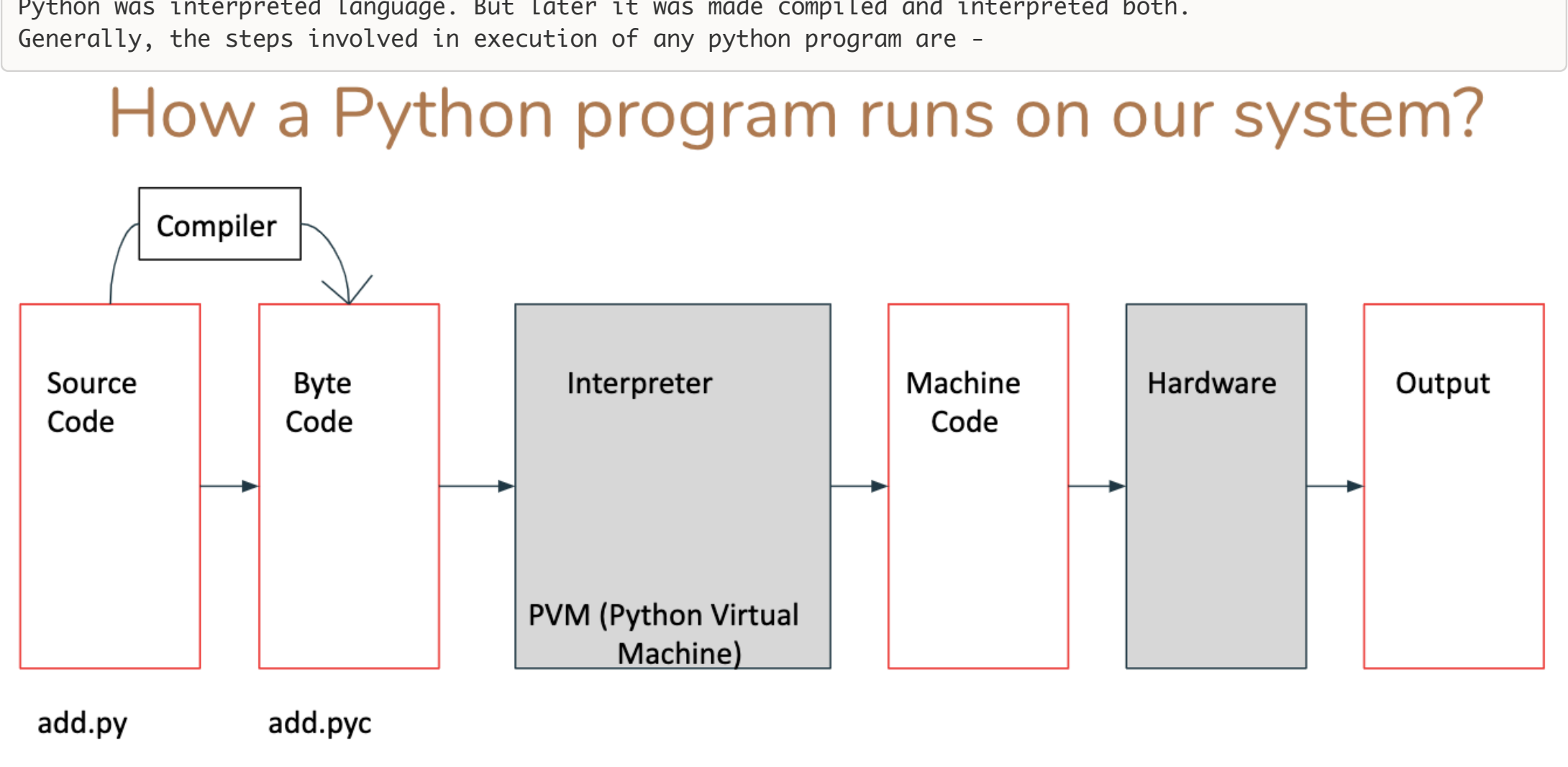
Sample Program in Python to add two numbers:-

```
a = 10
b = 20
c = a+b
print(c)
```

2. Most expressive language:- Being one of the most expressive language is easy to write the code in fewer lines, which leads to less cluttered program, faster execution and easy to debug and maintain.

3. Freeware and open source:- Python being freeware, you don't have to spend on licensing. And since it is open source so its original source code is freely available and can be redistributed and modifiable.

4. Compiled and Interpreted both:- Originally, Python was developed to bridge the gap between C and shell scripting and also include the feature of exception handling from ABC language. So we can say that, initially Python was interpreted language. But later it was made compiled and interpreted both.
Generally, the steps involved in execution of any python program are -



In the above diagram, compiler is responsible to compile the source code. In the compilation stage all the syntactical errors are checked whereas interpreter is responsible to interpret the byte code and check the run time error in this stage.
Compiler and interpreter both are translator. Compiler translates the source code to byte code(.pyc file) all at once whereas interpreter translates byte code to machine code one instruction at a time which interacts with hardware to give the final output.

Few important commands:-
1. python File_Name.py -> To run the file
2. python -m py_compile File_Name.py -> To compile the file
3. python -m dis File_Name.py -> To see the byte code.
4. python File_Name.cpython-38.pyc -> To run the compiled code.

5. Portable and Platform Independent:- Python is portable and platform independent both. Source code is portable whereas byte code(.pyc file) is platform independent. C is portable but not platform independent.

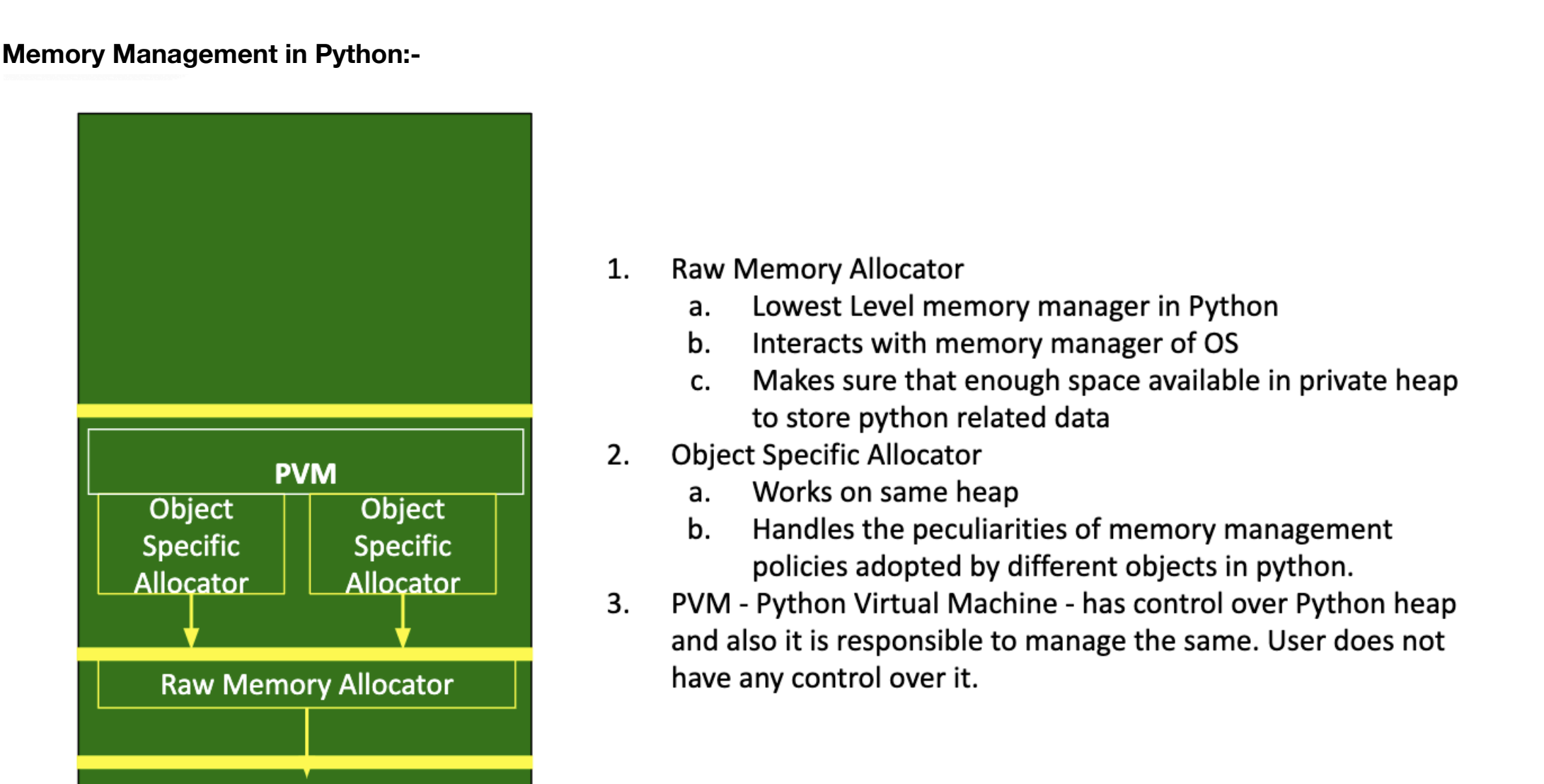
6. Dynamically typed programming language:- In python, everything is an object. Objects get stored in the heap area and it's memory address is referenced by the variable or name with which it is binded. Unlike other programming languages, In python, memory location does not work like a container rather it works on reference model. No need to specify the data type explicitly in python.

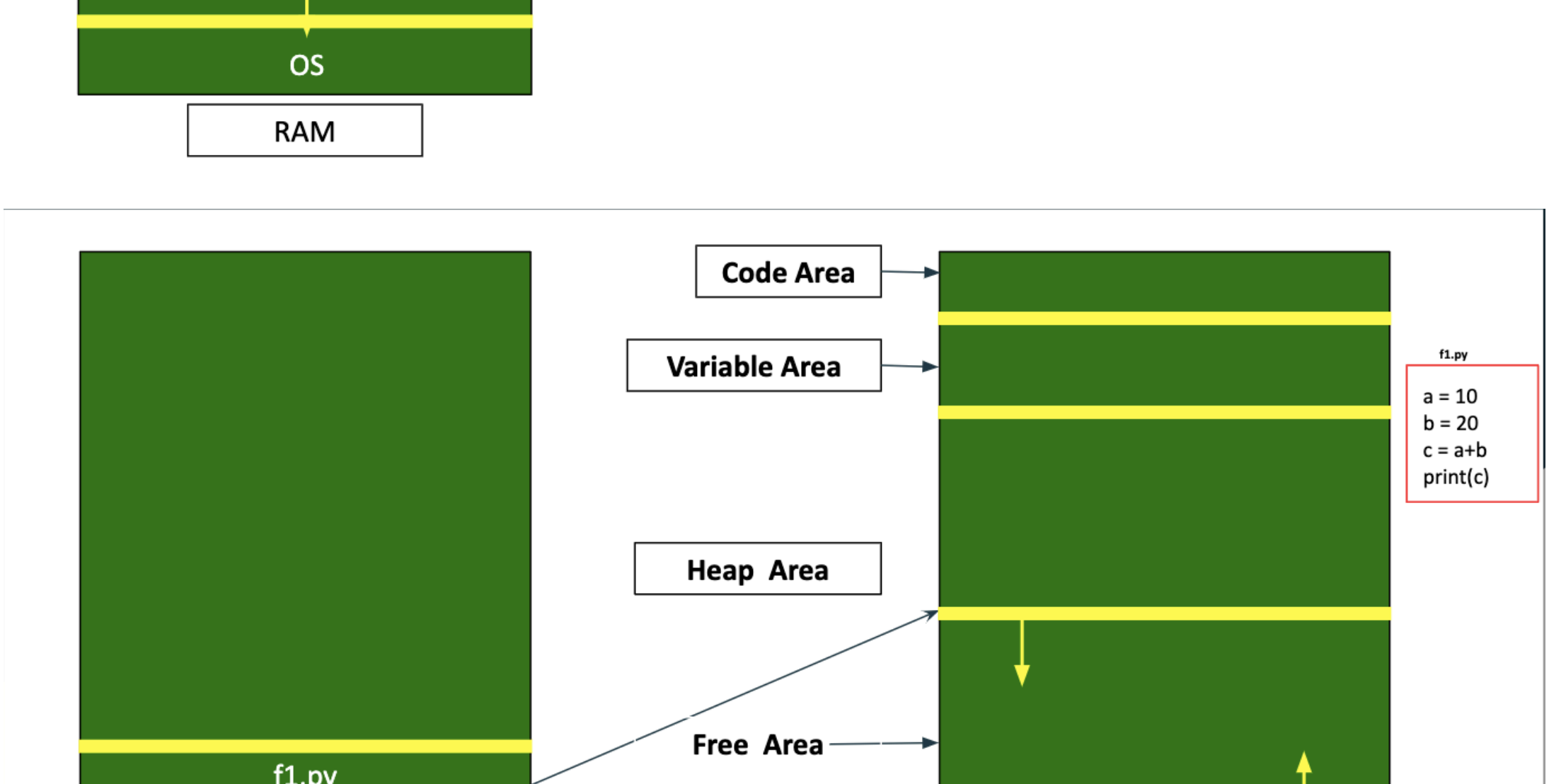
7. Extensible and Embedded both - Extensibility - Extending the code from other programming languages into Python code. **Embedded** - Embedding Python code to other programming languages code.

8. Batteries included:- Python comes with huge library support required for full usability.

9. Community Support:- Huge Python community support available for beginner, intermediate and advance level programmers.
10. CBSE 10th standard syllabus has Python now.
11. The growth and acceptance python has shown over the years, is exceptional. Click Here for more details.
[References:-]
1. https://en.wikipedia.org/wiki/Python_(programming_language)

Memory Management in Python:-





Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution(late binding) which binds methods and variable names during program execution.
As we know, whenever we run any application it gets loaded into RAM and some memory gets allocated by OS for that application. Fig-1 shows the same. Now, Fig-2 shows the different area created in the allocated memory for running the application.
Code Area:- Fixed in size and code gets loaded in this area.
Stack Area:- It may grow or shrink dynamically and is responsible for the order in which method will be resolved. Also in this local variable reference are created.
Heap Area:- This also may grow or shrink dynamically and is responsible to store any object in Python. Here object get stored in random order. Every object which gets stored here will have a specific ID. Garbage collector runs in heap area from time to time and destroys the object which are not having any reference.
Variable area:- In This section variables are stored and they refer their corresponding object in heap area.

Garbage Collection in Python

Python supports automatic garbage collection. It uses a combination of reference counting and cycle-detecting garbage collector for memory management an optimisation. gc module in python stands for garbage collector which runs from time to time and which ever object's reference count reaches to zero, that gets garbage collected automatically. As a user you are not allowed to interfere in memory management. It is task of the PVM to manage the memory. Yes, but if you want to check whether gc module is enabled or disabled or if you want to enable or disable it then you may use it like below:-

In [9]:
```
import gc
print(dir(gc), '\n')
print(gc.isenabled(),'\n')
gc.disable()
print(gc.isenabled(), '\n')
gc.enable()
print(gc.isenabled(), '\n')
```

```
['DEBUG_COLLECTABLE', 'DEBUG_LEAK', 'DEBUG_SAVEALL', 'DEBUG_STATS', 'DEBUG_UNCOLLECTABLE', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'callbacks', 'collect', 'disable', 'enable', 'freeze', 'garbage', 'get_count', 'get_debug', 'get_freeze_count', 'get_objects', 'get_referents', 'get_referrers', 'get_stats', 'get_threshold', 'is_t
racked', 'isenabled', 'set_debug', 'set_threshold', 'unfreeze']

True

False

True
```

Different Flavours of Python

There are different flavours or implementation of python are available. Out of which the most common are:-
1. **Cpython:- C** implementation of Python, developed to run **C** application on PVM, which was designed using **C**.
2. **Jython:- Java** implementation of Python, developed to run **Java** application on PVM, which was designed using **Java**.
3. **PyPI:- Python** implementation of Python. Implements JIT compiler, to improve the performance of python application.
4. **Ruby Python:- Ruby** implementation of Python and was developed to run the Ruby application.
5. **Stackless Python:-** Was developed to implement concurrency in Python.
6. **Anaconda Python:-** Was developed to handle and process the big data.