### **Topics: PVM, identifier, keywords**

### **Python Virtual Machine (PVM)**

The Python Virtual Machine (PVM) is the runtime engine of the Python programming language. It is responsible for executing Python bytecode generated by the Python interpreter. Here's a breakdown of its key components and how it works:

### **Components of Python Virtual Machine (PVM):**

* **Interpreter:** The Python interpreter reads Python source code, parses it into abstract syntax trees (ASTs), and compiles it into bytecode. The bytecode is a low-level representation of the source code that can be executed by the PVM.
* **Bytecode**: Bytecode is a platform-independent representation of the Python source code. It consists of a sequence of instructions that the PVM can execute. Bytecode files have a **.pyc** extension and are generated by the Python interpreter when a Python script is imported or executed.
* **Execution Engine:** The execution engine of the PVM interprets and executes the bytecode instructions. It traverses the bytecode instructions sequentially, performing the necessary operations as directed by the bytecode.

### **How Python Virtual Machine Works:**

* **Loading Bytecode:** When a Python script is executed, the Python interpreter loads the bytecode generated from the script into memory.
* **Interpreting Bytecode:** The execution engine of the PVM interprets each bytecode instruction one by one. It fetches the next instruction, decodes it, and executes the corresponding operation.
* **Executing Operations:** The PVM executes various operations specified by the bytecode instructions. These operations can include arithmetic calculations, variable assignments, function calls, and control flow statements (such as loops and conditionals).
* Managing Memory and Resources: The PVM manages memory allocation and deallocation for objects created during program execution. It also handles system resources, such as file handles and network connections.
* **Garbage Collection:** The PVM includes a garbage collector that automatically deallocates memory for objects that are no longer referenced by the program. This helps prevent memory leaks and ensures efficient memory usage.
* **Platform Independence:** The PVM provides platform independence, allowing Python code to run on different operating systems without modification. It abstracts away the underlying hardware and operating system details, providing a consistent execution environment for Python programs.

-===================================================

# **Python Keywords and Identifiers**

## **Python Keywords**

Keywords are predefined, reserved words used in Python programming that have special meanings to the compiler.

We cannot use a keyword as a variable name, function name, or any other identifier. They are used to define the syntax and structure of the Python language.

All the keywords except True, False and None are in lowercase and they must be written as they are. The list of all the keywords is given below.

|  |  | Python Keywords List |  |  |
| --- | --- | --- | --- | --- |
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

Looking at all the keywords at once and trying to figure out what they mean might be overwhelming.

If you want to have an overview, here is the complete list of all the keywords with examples.

## **Python Identifiers**

Identifiers are the name given to variables, classes, methods(functions), etc. For example,

language = 'Python'

Here, language is a variable (an identifier) that holds the value 'Python'.

We cannot use keywords as variable names as they are reserved names that are built-in to Python. For example,

continue = 'Python'

The above code is wrong because we have used continue as a variable name.

To learn more about variables, visit Python Variables.

## **Rules for Naming an Identifier**

* Identifiers cannot be a keyword.
* Identifiers are case-sensitive.
* It can have a sequence of letters and digits. However, it must begin with a letter or \_. The first letter of an identifier cannot be a digit.
* It's a convention to start an identifier with a letter rather \_.
* Whitespaces are not allowed.
* We cannot use special symbols like !, @, #, $, and so on.

### **Some Valid and Invalid Identifiers in Python**

| Valid Identifiers | Invalid Identifiers |
| --- | --- |
| score | @core |
| return\_value | return |
| highest\_score | highest score |
| name1 | 1name |
| convert\_to\_string | convert to\_string |

## **Things to Remember**

Python is a case-sensitive language. This means, Variable and variable are not the same.

Always give the identifiers a name that makes sense. While c = 10 is a valid name, writing count = 10 would make more sense, and it would be easier to figure out what it represents when you look at your code after a long gap.

Multiple words can be separated using an underscore, like this\_is\_a\_long\_variable.