### **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.

Some Amazing Python Programs - The Power of Python | Python Tutorial - Day #2

Modules and Pip | Python Tutorial - Day #3

Our First Python Program | Python Tutorial - Day #4

Comments, Escape Sequences & Print Statement | Python Tutorial - Day #5

Variables and Data Types | Python Tutorial - Day #6

Exercise 1: Calculator using Python | Python Tutorial - Day #7

Exercise 1: Calculator using Python (Solution) | Python Tutorial - Day #8

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

# **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) which 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.

# **Python Comments**

Comments are hints that we add to our code to make it easier to understand.

When executing code, Python's interpreter ignores comments.

For example, we have a program to print a text entered by the user.

name = input("Enter your name:")

print(name)

To make this program more readable, we can add comments like:

# Program to take the user's name

name = input('Enter your name')

print(name)

Here, the line starting with # is a comment. The Python compiler ignores everything after the # symbol.

Now, let's understand the different types of comments in Python.

## **Single-line Comment**

We use the **hash(#)** symbol to write a single-line comment. For example,

# declare a variable

name = 'John'

# print name

print(name) # John

In the above example, we have used three single-line comments:

* # declare a variable
* # print name
* # John

We can also use single-line comments alongside the code:

print(name) # John

**Note:** Remember the keyboard shortcut to apply comments. In most text editors, it's **Ctrl + /** if you are on Windows & **Cmd + /** if you are on a Mac.

## **Multiline Comments**

Python doesn't have dedicated multi-line comment syntax like some other programming languages like C++ and Java.

However, we can achieve the same effect by using the hash (#) symbol at the beginning of each line.

Let's look at an example.

# print(1)

# print(2)

# print(3)

We can also use multiline strings as comments like:

'''This program takes an input from the user

and prints it'''

name = input('Enter your name: ')

print(name)

**Output**

Enter your name: John

John

We can see that these unassigned multiline strings are ignored.

## **Prevent Executing Code Using Comments**

Comments are valuable when debugging code.

If we encounter an error while running a program, instead of removing code segments, we can comment them out to prevent execution.

For example,

number1 = 10

number2 = 15

sum = number1 + number2

print('The sum is:', sum)

print('The product is:', product)

Here, the code throws an error because we have not defined a product variable.

We can comment out the code that's causing the error.

For example,

number1 = 10

number2 = 15

sum = number1 + number2

print('The sum is:', sum)

# print('The product is:', product)

**Output**

The sum is 25

Now, the code runs without any errors.

Here, we have resolved the error by commenting out the code related to the product.

If we need to calculate the product in the near future, we can uncomment it.

## **Why Use Comments?**

We should use comments for the following reasons:

* Comments make our code readable for future reference.
* Comments are used for debugging purposes.
* We can use comments for code collaboration as it helps peer developers to understand our code.

**Exercise or task**

**Next Class discuss the Solution**