SOURCE CODE :

Week 1: Overview of Julia Platform, working with data types and variables.

Solution:

Overview of Julia programming

Scientific programming demands a high-performance, adaptable dynamic programming language, which is one of its facts. Unfortunately, the domain specialists have mostly switched to slower dynamic programming languages. Such dynamic programming languages may be used for a variety of beneficial purposes, and their use cannot be reduced. What, on the other hand, might we anticipate from contemporary language design and compiler techniques? The following are some of the expectations.

- The performance trade-off ought to disappear as a result.
- It needs to provide the subject matter specialists a solitary, productive place for prototyping.
- It ought to give subject-matter specialists access to a single, effective environment for installing performance-intensive applications.

This is what the Julia programming language delivers. It is a versatile general-purpose programming language that may be used to create any type of application. It works well for computing in science and mathematics.

The most popular programming languages, Matlab, R, and Python, are compared in depth below with Julia.

- Julia's syntax is comparable to that of MATLAB, however, it is a far more versatile language than MATLAB. Although OCTAVE (the open source alternative to MATLAB), which most Julia function names mimic, the calculations are very different. Julia is just as capable as MATLAB in the area of linear algebra, but it won't present its users with the same licence price problems. Additionally, Julia is a lot quicker than OCTAVE. The package that allows Julia to connect with MATLAB is called MATLAB.Jl.
- Julia converts Python-like code into machine code, giving programmers performance similar to that of the C programming language. When comparing Julia's performance to Python's, Julia outperforms Python by a factor of 10 to 30. The PyCall module enables Julia users to invoke Python functions.
- R is well-known for being one of the top programming languages in the statistical field, but Julia is just as useful in this field because to a speed boost of 10 to 1,000 times. Julia is ideal for doing both statistics and linear algebra, but MATLAB and R are not appropriate for both tasks. But when we contrast the type systems of R and Julia, the former has a considerably richer type system.

Julia's working environment

#= This is an example to demonstrate the multi-line comments=#
This is single line comments

Variables

Assigning Integer
julia> x=3
Output: 3

Assigning Float Value
julia> y=3.0
Output: 3.0
julia> z= -3
Output: -3

Assigning String
julia> y= "hello"
Output: "hello"

Using Operator as variable name
julia> + = "abc"
Output: "abc"

Using Unicode as variable name
julia> ∀ = 30

Output: 30

Data types

```
julia> for T in [Int8,Int16,Int32,Int64,Int128,UInt8,UInt16,UInt32,UInt64,UInt128]
          println("$(lpad(T,7)): [$(typemin(T)),$(typemax(T))]")
       end
```
Output: Int8: [-128,127]
Int16: [-32768,32767]
Int32: [-2147483648,2147483647]
Int64: [-9223372036854775808,9223372036854775807]
Int128: [-170141183460469231731687303715884105728,170141183460469231731687303715884105727]
UInt8: [0,255]
UInt16: [0,65535]
UInt32: [0,4294967295]
UInt64: [0,18446744073709551615]
UInt128: [0,340282366920938463463374607431768211455]
julia> typeof(0x1)
Output: UInt8
julia> typeof(0x1234567)
Output: UInt32
julia> typeof(5.)
Output: Float64