**Reliability**

The performance of a program depends considerably on the language it has been written on. Its affects bear the decision of whether the program will be reliable for further extension and modification in the future if required. Of course reliability is a more abstract term and not all languages need to have the same standard and it depends on the purpose of the program it’s written with.

A significant criterion to base our judgment has been mentioned in John D Gannon and J.J Horning’s paper “Language Design for Programming Reliability”[1] which is the ability of a programming language to decrease programming errors and the ability to detect them if any. This can be related to the readability and writability of the said programming language. A language that is more readable theoretically should have less fuss to deal with when trying to detect anomalies or errors.

For the purpose of our paper we made a comparative analysis of the six given languages over the above mentioned criteria to create a rough comparison of where they stand in terms of reliability. We are by no means commending one language over the other as each of them had different priority for reliability.

**Restricted Aliasing**

As we know it aliasing is when two pointers are directing to the same memory location. This is undesirable as we don’t want mismatched data, undefined results or overwrite any value. Thus a strict aliasing rule is often integrated within the language system to prevent pointers of different objects to never indicate to the same memory location.

In case of **C and C++** it is assumed pointers of different types will not alias each other and so memory access need not be conservative.[2] This has a significant advantage as you can be sure not to expect ‘undefined’ as a result when running the following code for example:

int example(int \*x, long \*y) {

  \*x = 0;

   \*y = 1;

  return \*x; }

Furthermore, C/C++ allows using restrict keyword to manually declare that two given pointers are not aliasing each other.

In **python,** the data type of variable isn’t explicitly defined and hence aliasing can happen when variables refer to the same location. That’s a problem for mutable objects such as lists, dictionaries. To solve this python allows explicitly declaring type or just making a copy of the object [3].

**Java** handles aliasing issues in the runtime. You may have two variables pointing to the same location but depending on the object that is being referenced in the runtime, java decides which of the two variables to use. This makes for easy aliasing handling comparatively.

In **java script**, function variables are passed by value and not passed by reference. This means when we pass a value to an argument, it is copied and sent to a function as the parameter, the variable address here is not referenced. Therefore, aliasing is not possible unless explicitly coded in to the program.

**R** is a functional programming language where objects or variables are passed by value and not passed by reference much like javascript. However to do so, one must explicitly code an environment slot for any object created which is of course a very cumbersome process.[4]

References

1. <https://ieeexplore.ieee.org/document/6312838>
2. <https://cellperformance.beyond3d.com/articles/2006/06/understanding-strict-aliasing.html>
3. <http://gestaltrevision.be/wiki/python/aliases>
4. https://www.stat.berkeley.edu/~paciorek/computingTips/Pointers\_passing\_reference\_.html