**Table 1.1** Language evaluation criteria and the characteristics that affect them

| | CRITERIA | | |
|---|---|---|---|
| Characteristic | READABILITY | WRITABILITY | RELIABILITY |
| Simplicity | • | • | • |
| Orthogonality | • | • | • |
| Data types | • | • | • |
| Syntax design | • | • | • |
| Support for abstraction | | • | • |
| Expressivity | | • | • |
| Type checking | | | • |
| Exception handling | | | • |
| Restricted aliasing | | | • |

Judgement criteria used:

| BAD | MODERATE | GOOD |
|---|---|---|

For readability:

| | Simplicity | Orthogonality | Data types | Syntax design |
|---|---|---|---|---|
| C | -designed to be simple<br>-but compared to latest languages it is not so simple anymore | -returns structures but not arrays<br>-returns arrays if inside structure | -multiple types of data types just for integer type<br>-no type for string, string | -meaning of some constructs are context dependent such as static |
| Java | -code to execute a simple program is quite complex<br>-hard for people with no coding experience | -common examples show high orthogonality, such as applying “public” and “static” together causes no problems | -no support for complex numbers | |
| Python | -Small number of basic components<br>-Mimics english | -basic building blocks can be combined in minimal ways to build the language’s data structures | -everything is an object | -indented code |
| Javascript | | | -no support for complex numbers | -high feature multiplicity |

| | | | | |
|---|---|---|---|---|
| C++ | -case sensitive<br>-allows operator overloading<br>- low uniformity | | -support for many different types and lengths of integers | -high feature multiplicity<br>-fewer reserved words<br>-meaning of some constructs are context dependent such as static |
| R | -mimics english<br>-huge library support as it is open source | | -everything is an object | |

Judging from the information we have collected so far, Python seems to be the most readable language to most users. This is sensible considering the easier syntax and progamming practices, and also in its rapidly increasing popularity.

For writability:

| | Support for abstraction | Expressivity |
|---|---|---|
| C | -Poor data encapsulation<br>-Poor information hiding | -Coercion with demotion (narrowing).<br>-Expression side effects<br>-Intermixing of arithmetic, logical and relational operators in Boolean expressions<br>-Unconventional operator usage and overloading<br>-Scope overriding |
| Java | -high level abstraction | -closer to adjective-noun syntax of spoken language |
| Python | -allows high level abstraction which means more flexibility and robustness | -very expressive as it requires much fewer lines of code |
| Javascript | -no built in support | -highly expressive |
| C++ | -high level abstraction similar to java | -highly expressive |
| R | -very high level abstraction possible | -highly expressive due to being closer to english language and library support |

In terms of writability, Java, Python, C++ and R are strong contenders so far. Python and R are very close to english language, which makes them more writable to new users. Whereas experienced coders can easily write in Java and C++ as most of their constructs are self-explanatory and easily understandable by people who know about programing practices.

For Reliability:

| | Type checking | Exception handling | Restricted aliasing |
|---|---|---|---|
| C | -static type checking | -does not provide support | |
| Java | -static type checking | -strong support using catch and throws | |
| Python | -dynamic type checking | -moderate support using try except | |
| Javascript | -dynamic type checking | -moderate support using try catch throw | |
| C++ | -static type checking | -strong support similar to java | |
| R | -dynamic type checking | -moderate support | |

For reliability we still need to collect more information. Overall more research is needed to fill in the tables.

Experiments done so far:
Bubble sorting of 1000 numbers from a text file. Results:

We are comparing three metrics:

1. How readable/writable the code is when the algorithm is implemented from scratch?
2. How readable/writable the code is when a library function is used (if available)
3. Time needed to run the algorithms from scratch.

C:

```c
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}


int main(void)
{
    int arr[1000];
    int i = 0;
    FILE * fp;

    if (fp = fopen("numbers.txt", "r")) {
        while (fscanf(fp, "%d", &arr[i]) != EOF) {
            ++i;
        }
        fclose(fp);
    }

    int n = sizeof(arr)/sizeof(arr[0]);
     bubbleSort(arr, n);
     printf("Sorted array: \n");
     printArray(arr, n);


    return 0;
}
```

The library function for sorting in C is qsort:

```c
#include <stdio.h>
#include <stdlib.h>

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}

int main () {
   int n;

   printf("Before sorting the list is: \n");
   for( n = 0 ; n < 5; n++ ) {
      printf("%d ", values[n]);
   }

   qsort(values, 5, sizeof(int), cmpfunc);

   printf("\nAfter sorting the list is: \n");
   for( n = 0 ; n < 5; n++ ) {
      printf("%d ", values[n]);
   }

   return(0);
}
```

In the above snapshots we can see that C does not have good readability or writability. The use of pointers in both cases make the code hard to understand, and difficult to write as well. Even the library function involves writing a big program.

Java:

```java
package org.bitan.bubble;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class bubbleJAVA {

    static void bubbleSort(int[] arr) {
        int n = arr.length;
        int temp = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (arr[j - 1] > arr[j]) {
                    //swap elements
                    temp = arr[j - 1];
                    arr[j - 1] = arr[j];
                    arr[j] = temp;
                }

            }
        }
    }

    public static void main(String[] args) {
        int[] tall = new int[1000];
        try {
            Scanner scanner = new Scanner(new File("C:\Users\ZahinAhmed\Desktop\CSE425_Benchmarking\numbers.txt"));

            int i = 0;
            while (scanner.hasNextInt()) {
                tall[i++] = scanner.nextInt();
            }
        } catch (FileNotFoundException filenotfoundexception) {
            System.out.println("File not found.");
        }
        bubbleSort(tall);
        for(int i=0;i<tall.length;i++)    //length is the property of the array
            System.out.println(tall[i]);
    }
}
```

The library function used to sort arrays in java is Arrays.sort():

```java
import java.util.Arrays;

public class GFG {
    public static void main(String[] args)
    {
        int[] tall = new int[1000];
        try {
            Scanner scanner = new Scanner(new File("C:\Users\ZahinAhmed\Desktop\CSE425_Benchmarking\numbers.txt"));

            int i = 0;
            while (scanner.hasNextInt()) {
                tall[i++] = scanner.nextInt();
            }
        } catch (FileNotFoundException filenotfoundexception) {
            System.out.println("File not found.");
        }

        Arrays.sort(tall);

        System.out.printf("Modified arr[] : %s",
                          Arrays.toString(arr));
    }
}
```

As we can see, java is definitely more readable compared to C. However, constructs such as scanner and System.out.println make it harder for new programmers to learn. However, using the library function definitely increases the readability and writability even more.

Python:

```python
def bubbleSort(arr):
    n = len(arr)

    for i in range(n):

        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]

arr = list()
filename = 'numbers.txt'
with open(filename) as fin:
    for line in fin:
        arr.append(line)
print(arr)

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%s" %arr[i]),
```

In python, the built in function used for sorting is sort()

```python
arr = list()
filename = 'numbers.txt'
with open(filename) as fin:
    for line in fin:
        arr.append(line)
print(arr)

arr.sort()

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%s" %arr[i]),
```

Comparing with the previous code snippets, it is very easy to judge that python is far more readable and closer to the english language. Constructs such as “open(filename)” and “in range” make it very intuitive. While, the built in function makes the code much simpler, implementing the algorithm from scratch is not that difficult either.

C++:

```cpp
#include <fstream>
#include<iostream>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}


void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)


    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}


void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {

   int arr[1000];
   ifstream is("numbers.txt");
   int cnt= 0;
   int x;

   while (cnt < arr[1000] && is >> x)

   arr[cnt++] = x;

   int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    cout<<"Sorted array: \n";
    printArray(arr, n);

   is.close();
}
```

In C++ the built in function for sorting is sort():

```cpp
#include <fstream>
#include<iostream>
using namespace std;

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {

   int arr[1000];
   ifstream is("numbers.txt");
   int cnt= 0;
   int x;

   while (cnt < arr[1000] && is >> x)

   arr[cnt++] = x;

   int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr + n);
    cout<<"Sorted array: \n";
    printArray(arr, n);

   is.close();
}
```

C++ is definitely more readable than C, however file manipulation and I/O is complicated. As we can see the syntax for these are not intuitive, and they have reliability issues as well as the file needs to be closed everytime. However, using the library function greatly simplifies the program.

R:

```
start_time<-Sys.time()

bubblesort <- function(arr){

  n = length(arr)
  v = arr

  for(j in 1:(n-1))
  {
    for(i in 1:(n-j))
    {
      if(v[i+1]<v[i])
      {
        t = v[i+1]
        v[i+1] = v[i]
        v[i] = t
      }
    }
  }
  arr = v
}

y<-scan(file = "numbers.txt", what = numeric(), sep = "\n")

sortedarray<-bubblesort(y)

print("Sorted array is:")

sortedarray

end_time<-Sys.time()

end_time-start_time
```

The Library function used in base R is sort()

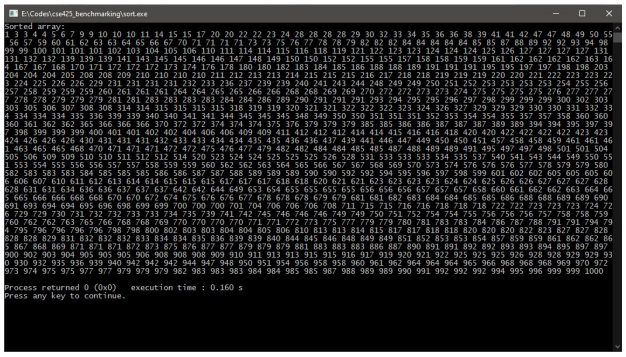
```
start_time<-Sys.time()

y<-scan(file = "numbers.txt", what = numeric(), sep = "\n")

sortedarray<-sort(y)

print("Sorted array is:")

sortedarray

end_time<-Sys.time()

end_time-start_time
```

R is even more intuitive than python. It's I/O operations and file manipulation is very easy, which makes it very likable for most users. The code is much shorter than most languages, which makes it very readable and writable.

Comparison based on runtime:

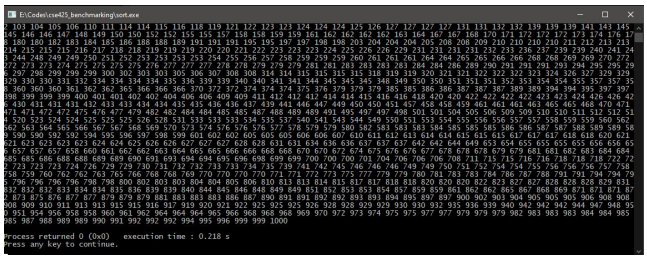
**C**

Terminal Output:0.160 Seconds




**C++**

Terminal Output: 0.218s




**Python**

Terminal Output: 0.183 seconds

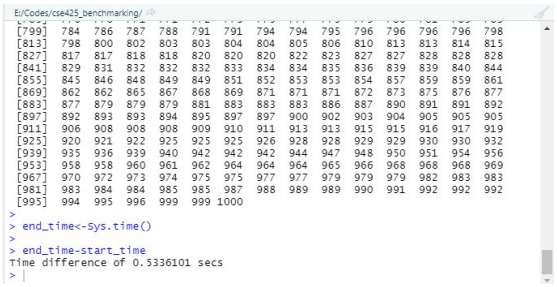

**Java**

Terminal Output: 544ms

```
989
990
991
992
992
992
994
995
996
999
999
1000

BUILD SUCCESSFUL in 544ms
2 actionable tasks: 2 executed
```

**R**

Terminal Output: 0.5446101 secs




| Language | Time(secs) | Readability rank (1 being best, 6 being worst) | Writability rank (1 being best, 6 being worst | Analysis |
|---|---|---|---|---|
| C | 0.160 | 6 | 6 | Sacrifices readability, writability for speed |
| Java | 0.544 | 3 | 3 | Sacrifices speed for readability, writability |
| Python | 0.183 | 2 | 1 | Efficient while being readable and writable |
| C++ | 0.218 | 4 | 4 | Sacrifices readability, writability for speed |
| R | 0.5446101 | 1 | 2 | Sacrifices speed for readability, writability |
| Javascript | Couldn't carry out experiment due to time limitation | | | |

Speed is hardware dependent, hence more research is needed for a proper analysis.