LANGUAGE DESIGN CRITERIA

- Since the inception, a lot of programming languages has been developed so far.

  However, not all of those are successful, and the criteria to judge a programming language if it is a success or a failure are not fixed.

  For instance, often one might consider human readability and mechanisms for abstractions and complexity control are the key requirements.

  ↳ This criteria may be insufficient as there are other <u>practical matters.</u>

  matters not directly connected to language definition
  - → Availability
  - → Price
  - → Quality of translators
  - → Politics
  - → Geography
  - → Timing
  - → Markets

- C programming

  Success of C programming had a strong relation with the success of UNIX operating system.

ALGOL 60 ... continues

Overview:

- Block structure: Allowed programmers to localize part of programs
  ↳ Local scoping
- Two different means of passing parameters to subprograms
  - → Pass by value
  - → Pass by name

COBOL Success

- Generally, ignored by computer science community.
- Continues as a significant language because of its use in industry.

  Also, because of the large number of old applications that are maintained.

Ada Language Success

- Because, this language is required in many US Defense department project.

Java and Python success

- Free distribution and growth of internet
- Programming environments

However,

Smalltalk language never came into widespread use.

↳ Though most successful object-oriented languages borrowed a large number of features from it.

Reasons for success ?

Some argue that

- Language designed by an individual, or small group of individuals, has a better chance of becoming a successful language.

  examples: Pascal, C, C++, LISP etc.

However,

there are other languages that are designed by committees. and are successful.

- COBOL
- Algol
- Ada

A common goal:

- Language design should have an overall goal.
  - Designers must keep the goal in mind during the overall design process.
- This is useful for both particular purpose language and general purpose language.

For particular purpose language —

such as, Database languages
Graphics languages
Real-time languages

particular abstractions for the target application area must be built during the language design.

For general purpose language —

such as

FORTRAN

↳ Designers focused on efficient execution

COBOL

↳ Designers focused on the readability

↳ English-like nontechnical

ALGOL 60

↳ Designers provided a block-structured language for describing algorithms.

C++

Designers focused on users' need for greater abstraction while preserving efficiency and copatibility with C.

Efficiency

Efficiency of a programming language can be evaluated from different angles —

* Efficiency may be termed from execution time point of view.

Java

```
int i = 10;
String a = "Hello world"
```

Types are defined

```
// Now, we can do something
   with i and s
```

Consider Python version

```
i = 10;
a = "Hellon world"
```

Here, types of i and a are not defined.

So, as the type specification is not done, it forces runtime system to check the type of a variable's value before executing any operations on it.

In contrast, in Java code segment, as the types are known at the compile time, the compiler can guarantee that only integer and string operations will be performed on (i) and (a)

Another view of efficiency : Programmer efficiency

From the users' perspective, alternative requirement for efficiency would be the expressiveness of the programming language.

- How easy is it to express complex process and structures?
- How easily can the design in programmer's head be mapped to actual program code?

Efficiency has direct relation with the abstraction mechanism's of the language.

Conciseness of the syntax is also a part of programming efficiency.

Languages that require a complex syntax are often less efficient.

For instance, Python is an ideal language for programmer efficiency

- Syntax is extremely concise

Comparison of conciseness of syntax

In C

```
if (x > 0) {
    num_solutions = 2;
    r1 = sqrt(x);
    r2 = -r1; }
else if (x == 0) {
    num_solutions = 1;
    r1 = 0.0 }
else
    num_solutions = 0;
```

Here, C uses
- statement terminators (;) → semicolon
- block delimeters → { }

Python

```
if x > 0.0 :
    num_solutions = 2
    r1 = sqrt(x)
    r2 = -r1
elif x = 0.0 :
    num_solutions = 1
    r1 = 0.0
else
    num_solutions = 0
```

Whereas, Python uses only
- indentation and
- the colon

- About 90% of the time of a software engineer is spent on debugging and Maintenance

  10% of time is in coding.

- Maintability

  - is a measure of how easily one can modify a code
  - Higher maintability means less time for making any necessary change.

  Concept of regularity makes program readable and maintable.

- RE

  - Refers to how well the features of a language are integrated

    so that there are no unusual restrictions, interactions or behavior.

    In the way language features behave should not surprise us.

Orthogonality

✓ A language that is truly orthogonal, language constructs don't behave differently in different contexts.

↓

So, restrictions that are context dependent are NON-ORTHOGONAL

Examples: lack of orthogonality

* In C and C++, values of all data types, except array types, can be returned from a fnc.

* In C, Local variables must be defined at the beginning of a block.

  In C++ variable definitions can occur in anywhere inside a block, but before the use of course.

* C passes all parameters by value, except arrays, which are passed by the reference.

↓

In Ada, Python, and in most other functional languages, this lack of orthogonality is removed.

Uniformity

- It refers to the consistency of appearance and behavior of language constructs.

- When similar things in a language behave in a similar way, the language is uniform.

  ↳ inversely,

  Different things should look different

Examples of lack of uniformity

In C++, a semicolon is necessary after a class definition, but

it is forbidden for fn^c def^n.

Class :

```
class   classname
{
        };
int fn^c ( )
{
        }   no semicolon.
```

- This non-uniformity is forced to allow C++ to be compatible with C.

Example of a Class in C++

Generic C++ class could be defined as follows:

```
class   Test_CSE425
  {  Access specifier:
          private/public/protected
     Data members;
     Member fnc ()
        {
            }
};   C++ class ends with
```

For instance,

```
class   Test_CSE425
 {  public:
     string   section;
     void   printsection ( )
       {  cout << " This is CSE425 SEC: << section;
       }
 };
```

C++ program to demonstrate class and accessing the data members.

```
#include <bits/stdc++.h>
using namespace std;
class TEST_CSE425
{
  public: //Access specifier
  string SECTION; // Data
  void printsection()
    {
      cout << "This is CSE425 SECTION:" << SECTION;
    }
};

int main()
  {
    TEST_CSE425 objsample; // Declaring an object of class TEST_CSE425
    objsample.SECTION = "ONE";
    objsample.printsection()
    return 0;
  }
```

Output: This is CSE425 SECTION: ONE

In addition,

Absence of explicit data types in variable declarations allows more conciseness

Support for recursion and dynamic data structures provide extra layer of abstraction.

However, explicit focus on programmer efficiency may compromise the other language principles such as —

- Efficiency
- Reliability

⊞ Reliability as an efficiency issue

Unreliable program can incur many extra costs—

- modifications needed for the error isolation
- extra testing time
- extra time needed to correct the erroneous behavior.

Programmer's efficiency also depends on

- easeness of error finding and correction.
- easeness to add new features.

Regularity is subdivided into three concepts

- Generality
- Orthogonality
- Uniformity

Generality

- Avoids special cases in the availability or use of constructs.
- Combining closely related constructs into a more general one.

Examples of lack of generality

- Nested fn^c definitions is not possible in C
- "==" is generally used to compare equality. However,

  Two structures or two arrays cannot be compared directly using ("==").

  Instead, the comparison must be done elementwise.
- Pascal has no-variable length arrays; array lacks generality.

lack of uniformity

* In C++, the operators & (bitwise and), && (logical and) yield different results.
  ↓ But they look confusingly similar.

SIMPLICITY

Albert Einstein says:

* Everything should be made as simple as possible, but not simpler.

* If you can't explain it to a six year old, you don't understand it yourself.

but Too much simplicity can fire back.

* BASIC is a very simple language, but lacks fundamental constructs such as blocks.

* C programming is designed to be simple
  - → efficient in generating target code
  - → Excellent for creating UNIX OS code
  - → " " " " OS ∧ device drivers

* However, C has some major flaws
  - ↳ Obscure operator syntax
  - ↳ Weak type checking

EXPRESSIVENESS

- It is easy to write understandable code
- the ease: Language can express complex processes and structures concisely.
- example:

Advances to expressiveness was the addition of RECURSION

Programming Example:

```
while (*a++ = *b++);
```

- ✓ copies a string to another
- ✓ Very expressive, very concise, but very readable
- a, b are pointers
- a is the destination
- content of a (*a) is being copied to b (*b)
- a and b are incremented ++
- Assignment (copy) returns the copied character to the while
- while continues until the character is zero. ↳ end of a string

```
while (*b != 0)
{
    *a = *b;
    a++;
    b++;
}
```

could be done using do { statement; } while (cond);

Extensibility

- General mechanisms should be there, with which a user can add features to a language.

- Example:
  Defining new data types
  Creating libraries
  Adding fn^c to a library

- Most common practice: Allow users to define
  - New data types
  - Operations needed for data types

Examples:

- Java and Python provides new releases regularly.
  ↳ Extend the built-in features

- Macro: specifies the syntax of a piece of code that expands to other.
  ↳ example: LISP → has general do loop
  ↕ Macro
  while (> b 0)
  :

- User-defined operators such as +++, as seen in fn^cal programming language.

Generality Vs. Orthogonality

Let's use fn^c concept to discuss the difference

Generality

Avoids special cases whenever possible

In C, pointers can be used treat fn^c as data

But nest fn^c definition is not possible

Whereas,

Python has a completely general way of treating fn^c
- → nested fn^c
- → fn^c as data object

Orthogonality

Language constructs do not behave differently in different contexts.

In C, values of all data types, except arrays, can be returned from a fn^c.

↳ Arrays are treated differently.

* Concept of orthogonality is generally used to mean that any language element can be used in all reasonable contexts with a meaning that is uniform in all those contexts.

↓ important

Orthogonality doesn't refer to missing functionality, but rather expected fn^cality.

Restrictability

Language design should make it possible for a programmer to be able to write program (for non-trivial task)

with minimal knowledge of the language

with minimal knowledge of the language constructs.

For instance,

In C programming,

knowing the syntax and semantics of if and goto statements to accomplish looping.

In a simplified way, we can say that language features should not be so complex for others to learn.

↳ may be, complex features are left out.

Another example is that

Operator overloading is omitted in Java language.

Operator overloading:

Depending upon the arguments we have for particular operations. we can have different types of computations going on in the program.

Operator overloading [expression and Assignment statements]

Generally, "+" sign is used to specify integer addition and floatpoint addition.

```
a = 10;
b = 5;
a+b = 10+5 = 15;
```

However, the same sign "+" may be used to perform other tasks.

↓

This multiple use of an operator is known as Operator overloading

For instance, in Java "+" is also used for string catenation.

```
String a = "Hello";
String b = "World";
```

a+b ⇒ "Hello World"

Problem/Dangers:

Consider the use of "&" (ampersand)

When "&" is used as a binary operator, it is bitwise logical AND

When "&" is used as an unary operator

expression value is an address of a variable

... continues

Unary operator (acts on single operand)

```
x = &y;
```

execution of the above statement causes address of y to be placed in x.

Multiple use of ampersand (&)

- Problem1: Same symbol for two completely unrelated tasks/operations.
  - ↓ Detrimental to
  - Readability
- Problem2: Consider that, a programmer misses the first operand for a bitwise AND

  ~~a~~ + b

  So,

  This may go undetected by the compiler.
  - ↓ Because
  - it may be interpreted as an address operator.

A few commonly overloaded parameters:

=

+, -, *, /

==, !=

etc.

there are more ...

SECURITY

Security is closed to the concept of reliability.

When restrictions are not imposed on the certain features, reliability of the programming language may be seriously affected.

In Pascal, pointers are specifically restricted, to reduce errors.

Whereas,
In C, it less restricted, results in more errors, and are more prone to errors

Java, pointers are eliminated at the cost of a more complicated runtime environment.

A few tasks/arrangements for security measures

Goal "minimize the number of errors that could not be made"

- Types
- Type checking
- Variable declarations

However, in Python considered that variable declarations and type-checking make it difficult to program complex operations