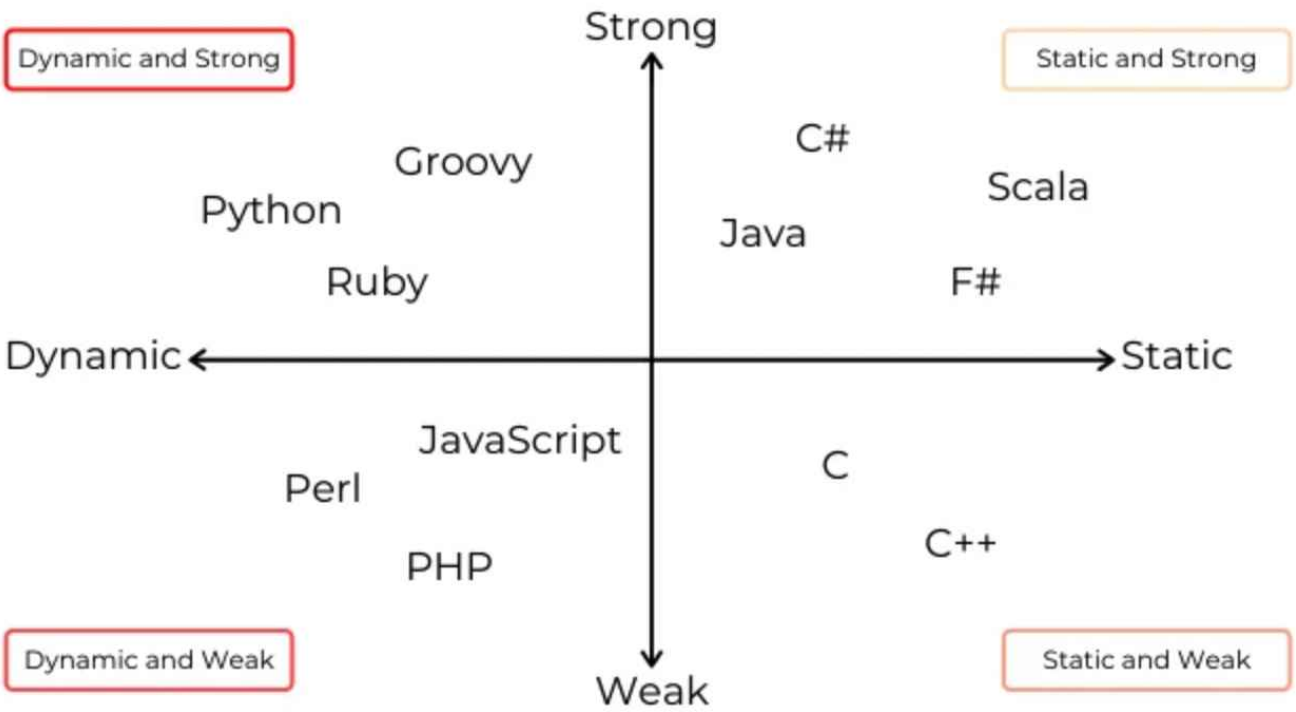
21-Strong-typed & Statically-typed

Introduction

All variables must have a type. It can either be a **primitive type**, such as *int, float, boolean,* OR an **object reference type**, such as *String, Array*, or other custom objects.

When we study Strong-typed & Statically-typed, it usually talks about the characteristics



- Java is a **strong-typed** & **statically-typed** language.

Strong-typed

Enforcing strict type checking and **disallowing operations between incompatible types without explicit conversion**. When you declare a variable with a specific type, the programming language ensures that the variable is used and **operated upon ONLY with values of the declared type**. Type declaration helps prevent type-related errors and promotes type safety.

```
1 int number = 10;
2 double result = (double) number; // Explicit type casting
```

Statically-typed

Statically-typed refers to a type of programming language where variable types are **checked at compile-time**. In statically-typed languages, variable types are explicitly declared, and the compiler **verifies** the compatibility of types and catches type-related errors **before the program is executed**.

```
1 int age = 25; // Declaring an integer variable
2 String name = "John"; // Declaring a string variable
3 double salary = 2500.50; // Declaring a double variable
4 String name = 13; // compile error
```

i.e. Lots of arguments on Strong/Weak type definition on the Internet. Check it out.

For the definition here, **we consider how WEAK the language is how it can automatically convert the types according to its own set of rules when you perform certain operations using data of different types and without any compile time error.**

- In `Java`, "5" + 5 will result in compilation error (**Integer** b = "5" + 5)

```
1 int result = "5" + 5; // compile-error
```

- In `Javascript` is a typical example of weak-typed language, "5" + 5 = "55", because it becomes string concatenation.

```
1 var result = "5" + 5;
2 console.log(result); // Output: "55"
```

- **In C, it does not have String, we take char '5' as example.** '5' + 5 may result as 58, refer to the following example.

```c
#include <stdio.h>

int add_numbers(int a, int b) {
    int result = a + b;
    return result;
}

int main() {
    char c = '5'; // The character '5' has an ASCII value of 53
    int n = 5;
    int result = add_numbers(c, n);
    printf("%d\n", result); // Output: 58
    return 0;
}
```

Another example in C

```c
char[] string = "abc";  // "abc" is stored as {'a', 'b', 'c', '\0'}
int res = string + 5;   // Adding 5 bytes to the memory address of 'string'
```

- The memory address of the character array string points to the first element 'a'.
- When you add 5 to the pointer, it advances by 5 bytes.