CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 24 C
Nov 17, 2022

TIOBE index of Programming Languages

November 2022

Rank	Language	Rating
1	Python	17.18%
2	**C**	15.08%
3	Java	11.98%
4	C++	10.75%
5	C#	4.25%
6	Visual Basic	4.11%
7	JavaScript	2.74%
8	Assembly Language	2.18%
20	**Rust**	0.75%
29	**Prolog**	0.44%
30	Lisp	0.38%

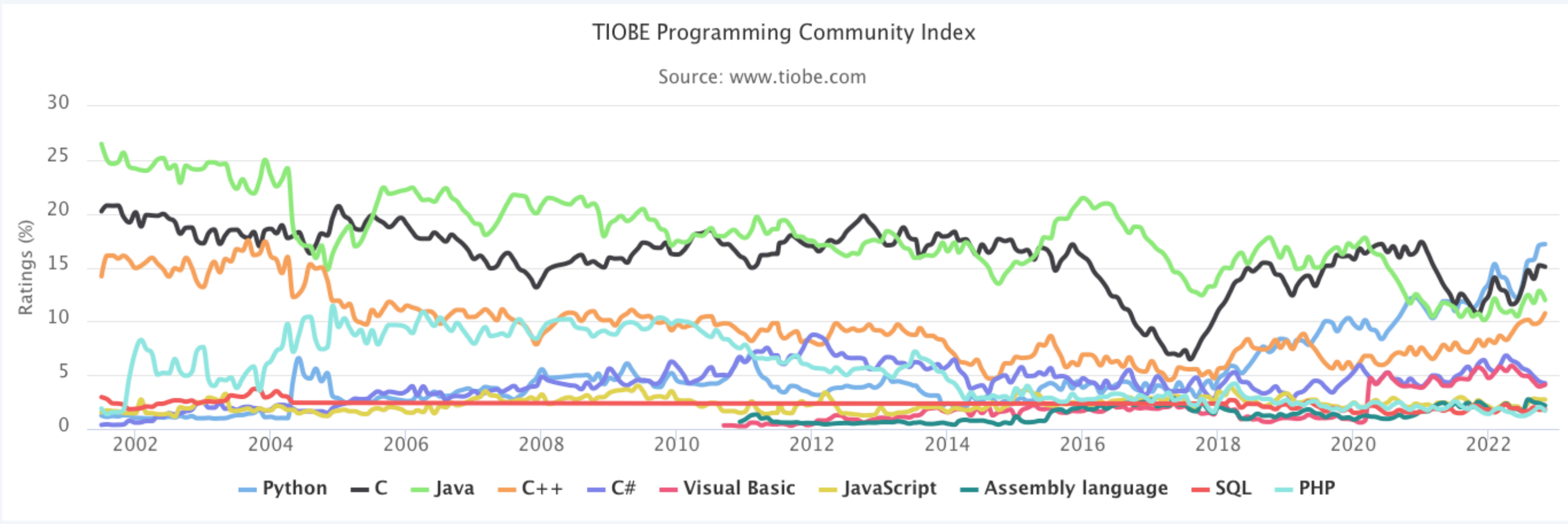
Clojure 50-100 ranking

Last week

NSA (National Security Agency) recommendation

Stop using C and C++

Tiobe Index History



PDP-7

History of C

1972

K&R released

Created by Dennis Richie to use in the development of Unix

1989-90 ANSI C, C89, C90

1995 C95 International character support

1999 C99

New data types, variable-length arrays, flexible array member

2011 C11

Generic macros, anonymous structures, improved Unicode support, atomic operations, multi-threading, and bounds-checked functions

2017 C17

Technical corrections & clarifications

2023 C23

C23

Decimal floating-point types (_Decimal32, _Decimal64, and _Decimal128)

Bit-precise integers (_BitInt(N))

Binary integer constants

u8 character constants

Type change of u8 string literals

Digit separator '

Empty initializer = {}

Unnamed parameters in function definitions

Identical cvr-qualifications for array types and their element types

Single-argument _Static_assert

static_assert becomes a keyword (may be a predefined macro for compatibility reasons)

thread_local becomes a keyword (may be a predefined macro for compatibility reasons)

Labels followed by declarations and }

nullptr constant and the associated nullptr_t type

true and false become keywords (may be predefined macros for compatibility reasons)

Reserved Words

C89

auto	default	float	long	sizeof	union
break	do	for	register	static	unsigned
case	double	goto	return	struct	void
char	else	if	short	switch	volatile
const	enum	int	signed	typedef	while
continue	extern				

C99

_Bool _Complex _Imaginary inline restrict

C11

_Alignas _Alignof _Atomic _Generic _Noreturn _Static_assert _Thread_local

Problems with C

the power of assembly language and the convenience of ... assembly language

— Dennis Ritchie

- Memory management with malloc & free is error prone
- Pointer arithmetic - memory corruption, vulnerabilities
- Poor type checking
- Generated code has few checks
  - Buffer overruns, stack overflows, memory exhaustion, race conditions
- Some library functions can lead to buffer overruns
- Limited standardization for low-level variants
  - Different structure packing conventions
  - Different byte ordering within larger integers including endianness

C is more than a Language

Other languages are implemented in C

- Python, Real, Ruby, PHP

Computationally-intensive libraries

- GNU Scientific Library,
- Mathematica
- MATLAB
- NumPy

Used as an intermediate language

- Eiffel, Sather, Some Lisps, Haskel

Popular in System Programming

Operating Systems & Embedded Systems

- Control over memory allocation
- Does not demand many system features
- Fast execution
- Easy to execute
- Maps well onto machine instructions
- Interacts with assembly
- Data structures use memory efficiently

Cost of Garbage Collection

If have 5X times the memory

Garbage collection is as fast as explicit memory management

If have 3X times the memory

Garbage collection is 17% slower than explicit memory management

If have 2X times the memory

Garbage collection is 70% slower than explicit memory management

When memory is scarce, paging can make garbage collection orders of magnitudes slower

Quantifying the Performance
of Garbage Collection vs. Explicit Memory Management
2005, Hertz & Berger

https://people.cs.umass.edu/~emery/pubs/gcvsmalloc.pdf

Hello World

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Base Types

int
float
double
char
_Bool

Modifiers
short
long
long long
unsigned
signed

int
usually 32 bits, but depends on machine and compiler

long int
Usually same size as int

short int
Usually 1/2 size of int but at least 16 bits

long long int
At least 64 bits

Base Types

```
#include <stdio.h>
int main (void)
{
    int         integerVar = 100;
    float     floatingVar = 331.79;
    double    doubleVar = 8.44e+11;
    char       charVar = 'W';
    _Bool     boolVar = 0;

    printf ("integerVar = %i\n", integerVar);
    printf ("floatingVar = %f\n", floatingVar);
    printf ("doubleVar = %e\n", doubleVar);
    printf ("doubleVar = %g\n", doubleVar);
    printf ("charVar = %c\n", charVar);

    printf ("boolVar = %i\n", boolVar);

    return 0;
}
```

```
integerVar = 100
floatingVar = 331.790009
doubleVar = 8.440000e+11
doubleVar = 8.44e+11
charVar = W
boolVar = 0
```

Type	Constant Examples	printf chars
char	'a', '\n'	%c
_Bool	0, 1	%i, %u
short int	—	%hi, %hx, %ho
unsigned short int	—	%hu, %hx, %ho
int	12, -97, 0xFFE0, 0177	%i, %x, %o
unsigned int	12u, 100U, 0XFFu	%u, %x, %o
long int	12L, -2001, 0xffffL	%li, %lx, %lo
unsigned long int	12UL, 100ul, 0xffeeUL	%lu, %lx, %lo
long long int	0xe5e5e5e5LL, 500ll	%lli, %llx, &llo
unsigned long long int	12ull, 0xffeeULL	%llu, %llx, %llo
float	12.34f, 3.1e-5f, 0x1.5p10,0x1P-1	%f, %e, %g, %a
double	12.34, 3.1e-5, 0x.1p3	%f, %e, %g, %a
long double	12.341, 3.1e-5l	%Lf, $Le, %Lg

Some C Fun

```
#include <stdio.h>
#include <stdbool.h>
int main (void)
{
    double    doubleVar = 8.44e+11;

    printf("%d %d %d\n", true && false, true || false, !false);
    printf("%d %d\n", true ^ true, true + true);
    printf ("doubleVar = %e\n", doubleVar);
    printf ("doubleVar = %i\n", doubleVar);
    printf ("doubleVar = %li\n", doubleVar);
    return 0;
}
```

Output

```
0 1 1
0 2
doubleVar = 8.440000e+11
doubleVar = 1056964608
doubleVar = 478523325264009625
```

A = 60 = 0011 1100

B = 13 = 0000 1101

Table 1

Operator	Description	Example
&	Binary AND	(A & B) = 12, i.e., 0000 1100
|	Binary OR	(A | B) = 61, i.e., 0011 1101
^	Binary XOR	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

=	Simple assignment operator.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Complex Numbers

```
#include <stdio.h>
#include <complex.h>
#include <tgmath.h>

int main(void)
{
    double complex z1 = I * I;     // imaginary unit squared
    printf("I * I = %.1f%+.1fi\n", creal(z1), cimag(z1));
    double complex z2 = pow(I, 2); // imaginary unit squared
    printf("pow(I, 2) = %.1f%+.1fi\n", creal(z2), cimag(z2));

    double PI = acos(-1);
    double complex z3 = exp(I * PI); // Euler's formula
    printf("exp(I*PI) = %.1f%+.1fi\n", creal(z3), cimag(z3));

    double complex z4 = 1+2*I, z5 = 1-2*I; // conjugates
    printf("(1+2i)*(1-2i) = %.1f%+.1fi\n", creal(z4*z5), cimag(z4*z5));
}
```

```
I * I = -1.0+0.0i
pow(I, 2) = -1.0+0.0i
exp(I*PI) = -1.0+0.0i
(1+2i)*(1-2i) = 5.0+0.0i
```

Coming Features

_BitInt(N) - Integer with N bits

_Decimal64 x = 0.10dd;

- Base 10 representation
- Exact value

Looping - for

```
#include <stdio.h>

int main (void)
{
    int  triangularNumber = 0, count = 0;

    for ( int n = 1;  n <= 200;  n = n + 1 )
        triangularNumber = triangularNumber + n;
        count += 1;

    // n is not defined here
    printf ("The 200th triangular number is %i\n", triangularNumber);
    printf ("count is %i\n", count);
    return 0;
}
```

```
The 200th triangular number is 20100
count is 1
```

```
#include <stdio.h>

int main (void)
{
    int  n, triangularNumber = 0;

    for ( n = 1;  n <= 200;  n = n + 1 )
        triangularNumber = triangularNumber + n;

    printf ("n is %i\n", n);
    return 0;
}
```

Output

```
n is 201
```

```
#include <stdio.h>

int main (void)
{
    int  n, triangularNumber = 0;

    for (int n = 1;  n <= 200;  n++ )
        triangularNumber = triangularNumber + n;

    printf ("n is %i\n", n);
    return 0;
}
```

Output

```
n is 50806880
```

```
for ( i = 0, j = 0;  i < 10;  ++i )
```

```
for  ( i = 0, j = 100;  i < 10;  ++i, j = j - 10 )
```

```
for  (  ;  j != 100;  ++j )
```

```
init_expression;
while ( loop_condition ) {
    program statement (or statements)
    loop_expression;
}


do
    program statement (or statements)
while ( loop_expression );
```

```
 if  ( expression )
      program statement
```

```
if  ( expression )
     program statement 1
else
     program statement 2
```

```
int main (void)
{
    if ( 11 ) printf(" if 11\n");
    if ( 12.34 ) printf(" if 12.34\n");
    if ( "cat" ) printf(" if cat\n");
    if ( "" ) printf(" if empty string\n");
    if ( 0 ) printf(" if 0\n");
    return 0;
}
```

Output

```
if 11
if 12.34
if cat
if empty string
```

0 = false

all other value are true

Switch

```
#include <stdio.h>
int main (void) {
    float  value1, value2;
    char   operator;
    printf ("Type in your expression.\n");
    scanf ("%f %c %f", &value1, &operator, &value2);

    switch (operator) {
        case '+':
            printf ("%.2f\n", value1 + value2);
            break;
        case '-':
            printf ("%.2f\n", value1 - value2);
            break;
        default:
            printf ("Unknown operator.\n");
            break;
    }
    return 0;
}
```

I/O

```
Type in your expression.
1 + 2
3.00
```

What Happens without the Default?

```
#include <stdio.h>
int main (void) {
    float  value1, value2;
    char   operator;
    printf ("Type in your expression.\n");
    scanf ("%f %c %f", &value1, &operator, &value2);

    switch (operator) {
        case '+':
            printf ("%.2f\n", value1 + value2);
            break;
        case '-':
            printf ("%.2f\n", value1 - value2);
            break;
    }
    return 0;
}
```

I/O

Type in your expression.

1 @ 2

Without the Break

```
int main (void) {
    float  value1, value2;
    char   operator;

    printf ("Type in your expression.\n");
    scanf ("%f %c %f", &value1, &operator, &value2);

    switch (operator) {
        case '+':
            printf ("%.2f\n", value1 + value2);
        case '-':
            printf ("%.2f\n", value1 - value2);
    }
    return 0;
}
```

I/O

Type in your expression.

1 + 2

3.00

-1.00

```
#include <stdio.h>
int main (void) {
    float  value1, value2;
    char   operator;

    printf ("Type in your expression.\n");
    scanf ("%f %c %f", &value1, &operator, &value2);

    switch (operator){
        case '+':
            printf ("%.2f\n", value1 + value2);
            break;
        case '-':
            printf ("%.2f\n", value1 - value2);
            break;
        case '*':
        case 'x':
            printf ("%.2f\n", value1 * value2);
            break;
    }
    return 0;
}
```

Conditional Operator

condition ? trueExpression : falseExpression

```
s = ( x < 0 ) ? -1 : x * x;

maxValue = ( a > b ) ? a : b;
```