INEN 202 Introduction to Computer Programming

- **Introductory programming course based on the C language**
- **Instructor: Dr. Y. Alex Xue, Associate Professor of NSE**
  - Office: NFE 4315
  - **Office Hour:** **MW 1:30-2:30 PM**
  - E-mail: **yxue@albany.edu**

- **Grading**
  - Two in-semester exams (16 points each).
  - One final take-home programming project (28 points)
  - Eight programming projects (5 points each)
  - **Check assignment due date on the course website**
  - Discussion encouraged but work independently please!

- **Required Text**
  - K.N. King, ***C Programming: A Modern Introduction, 2nd Edition*** (W.W. Norton & Company, 2008)
    - Ch. 21-Ch. 27 contain reference materials on C standard library.
- **Further materials on C programming can be found at the course website**

Weekly Course Topics

- **Part I**
  - Formatted Input/Output
  - Operators and Expressions
  - Selection/Looping Statements

- **Part II**
  - Basic Data Types
  - 1-D Array
  - Preprocessor Macros
  - Functions and Program Organization

- **Part III**
  - Pointers
  - Pointers and Arrays
  - Strings, Complex Numbers…

Week 1 Introducing C

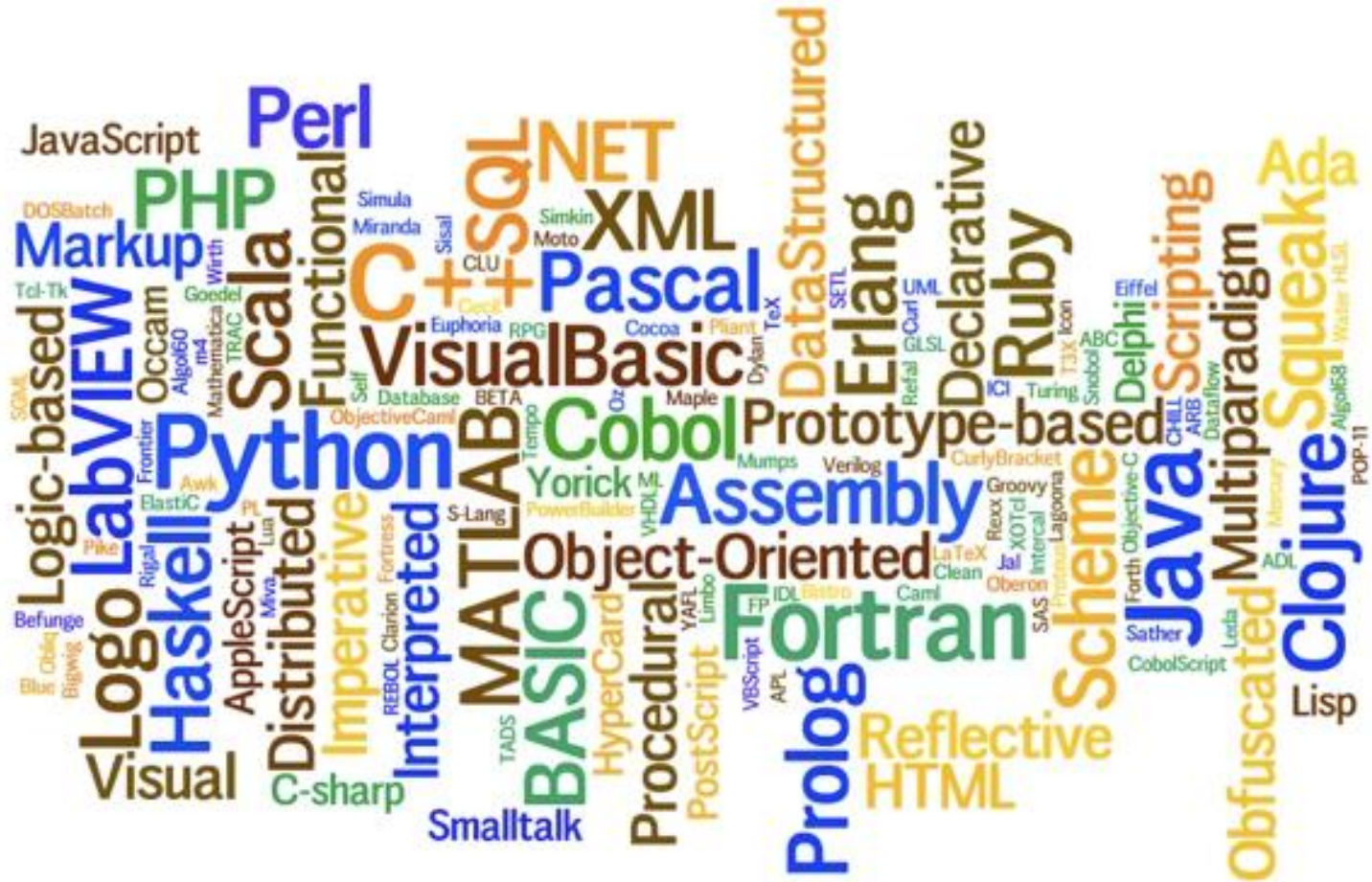
- **Introducing Computer Programming**
  - Programming and Programming Language
  - History of C
  - Strength and Weakness of C

- **Your First C Program**
  - Writing, Compiling and Linking
  - Cygwin/gcc/notepad++

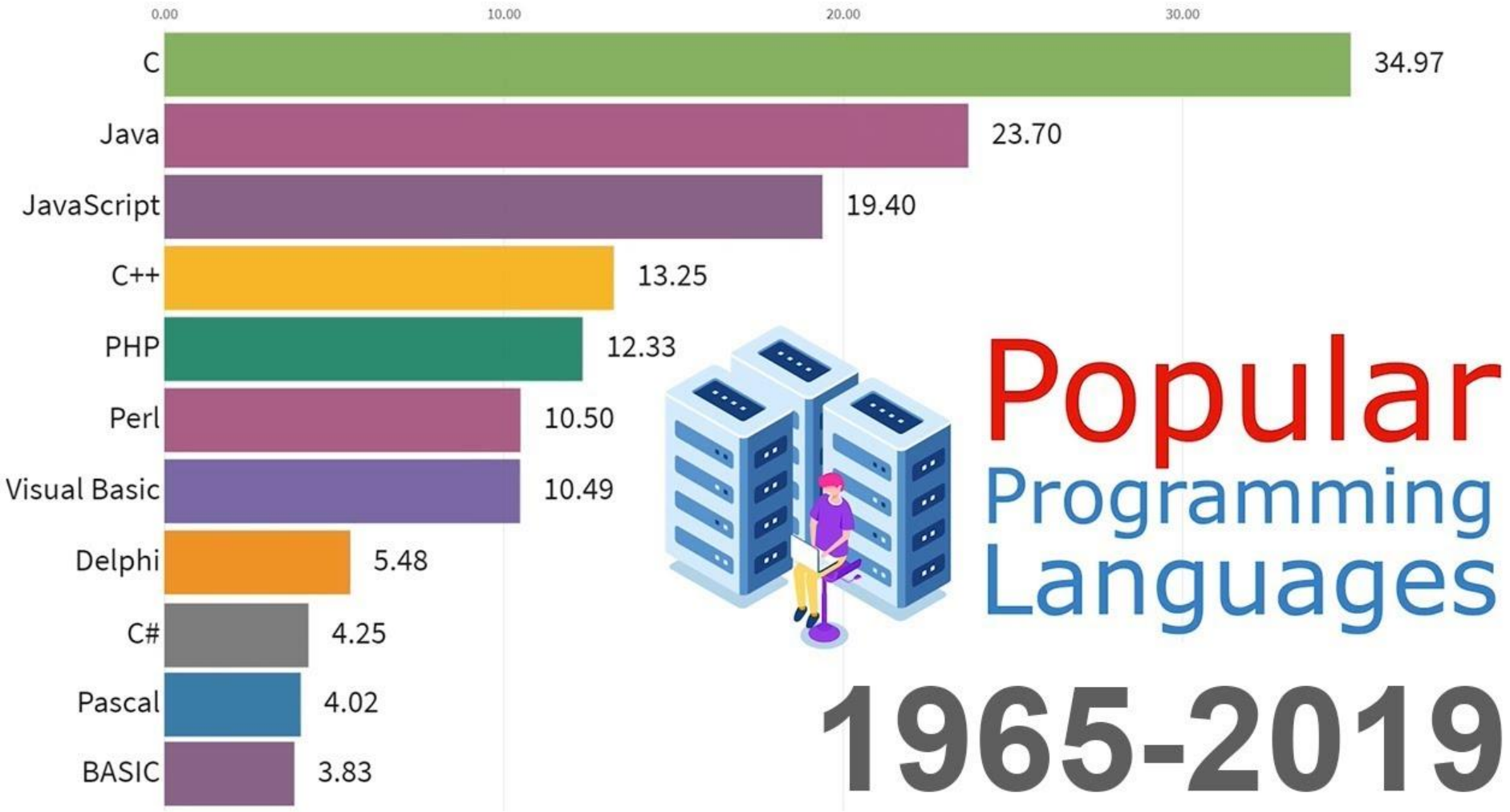
Computer Programming and Programming Language

- **Computer programming** is a process that leads from an original formulation of a computing problem to executable programs.
  - It involves activities such as analysis, understanding, and generically solving such problems resulting in an **algorithm**, …, implementation (or coding) of the algorithm in a target programming language, …
  - The algorithm is often only represented in human-parsable form and reasoned about using logic. Source code is written in one or more **programming languages.**
  - The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solve a given problem.
  - A **programming language** is an artificial language designed to communicate instructions to a machine

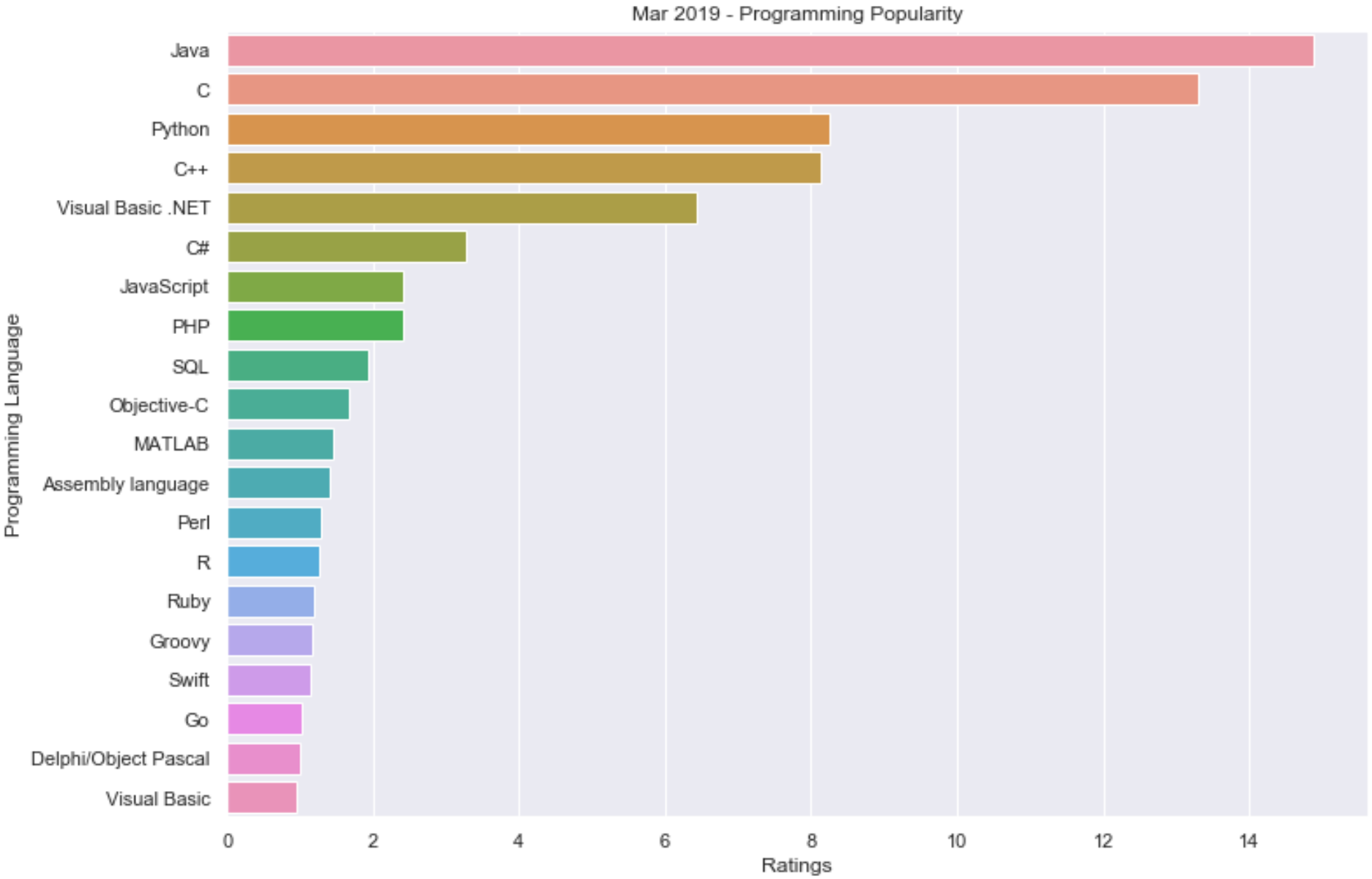
Programming Languages



Programming Languages Popularity



Programming Languages Popularity: 2019



Origins of C

- C is a by-product of UNIX, developed at Bell Laboratories by Ken Thompson, Dennis Ritchie, and others.
- Thompson designed a small language named B.
- B was based on BCPL, a systems programming language developed in the mid-1960s, which in turn was based on Algol.
- By 1971, Ritchie began to develop an extended version of B. He called his language NB (“New B”) at first.
- As the language developed further, he changed its name to C.
- The language was stable enough by 1973 that UNIX could be rewritten in C.

Standardization of C

- ***K&R C***
  - Described in Kernighan and Ritchie, ***The C Programming Language* (1978)**
  - De facto industry standard
- ***C89/C90***
  - ANSI standard X3.159-1989 (completed in 1988; formally approved in December 1989)
  - International standard ISO/IEC 9899:1990
- ***C99***
  - International standard ISO/IEC 9899:1999
  - Incorporates changes from Amendment 1 (1995)

C-Based Programming Languages

- *C++* (invented by Bjarne Stroustrup at Bell Lab, 1979-) includes all the features of C, but adds classes, templates and other features to support object-oriented and generic programming features.
- *Java* is based on C++ and therefore inherits many C features.
- *C#* is a more recent language derived from C++ and Java.
- *Python* is a high-level language that is implemented using C.
- *MATLAB*, originally based on FORTRAN, is implemented in C

Strengths of C

- Efficiency
- Portability
- Relatively small language
- Low-level constructs

Weaknesses of C

- Programs can be error-prone.
- Programs can be difficult to understand.
- Programs can be difficult to modify.
- **Think about Programming Languages vs. Natural Languages vs. Mathematics**
  - **Context**
  - **Grammar/Rule**
  - **Style**

Effective Uses of C

- Adopt a sensible set of coding conventions.
- Avoid “tricks” and overly complex code.
- Stick to the standard.
- Learn how to avoid pitfalls.
- …

"Hello, World!": Your First C Program

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C: that is the question.\n");
  return 0;
}
```

- This program might be stored in a file named `pun.c`.
- The file name doesn't matter, but the `.c` extension is often required.

Compiling and Linking

- **Before a program can be executed, three steps are usually necessary:**
  - ***Preprocessing.*** The ***preprocessor*** obeys commands that begin with # (known as ***directives***)
  - ***Compiling.*** A ***compiler*** translates then translates the program into machine instructions (***object code***).
  - ***Linking.*** A ***linker*** combines the object code produced by the compiler with any additional code needed to yield a complete executable program.
- **The preprocessor is usually integrated with the compiler.**

Compiling and Linking with gcc

- gcc is the GNU Project C compiler
- A command-line program
- gcc takes C source files as input

  (Save your pun.c file in the folder c:/cygwin/home/your_user_name)
- To compile and link the `pun.c` program under UNIX, enter the following command in a terminal or command-line window:

```
% gcc pun.c
```

- Outputs an executable by default: a.exe
- Linking is automatic when using gcc; no separate link command is necessary.

Compiling and Linking with gcc

- **To compile with a different executable output name simply type:**

  % gcc -o pun pun.c -std=c99 -Wall

  - All gcc options are prefixed with hyphen '-' !
  - '-o' option tells the compiler to name the executable pun
  - '-Wall' tells it to print out all relevant warnings (very useful!!!)

- **To execute the program in cygwin, simply type:**

  % ./pun.exe

Compiling and Linking

- **Before a program can be executed, three steps are usually necessary:**
  - ***Preprocessing.*** The ***preprocessor*** obeys commands that begin with # (known as ***directives***)
  - ***Compiling.*** A ***compiler*** translates then translates the program into machine instructions (***object code***).
  - ***Linking.*** A ***linker*** combines the object code produced by the compiler with any additional code needed to yield a complete executable program.
- **The preprocessor is usually integrated with the compiler.**
- **Preprocessing, Compiling and Linking are integrated when using gcc**
  - **$ gcc -o pun pun.c -O -Wall -std=c99**
  - **$ ./pun.exe**

General Form of a Simple C Program

- Simple C programs have the form

  directives

  ```
  int main(void)
  {
    statements
  }
  ```

- C uses `{` and `}` in much the same way that some other languages use words like `begin` and `end`.

- Even the simplest C programs rely on three key language features:
  - Directives
  - Functions
  - Statements

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C: that is the question.\n");
  return 0;
}
```