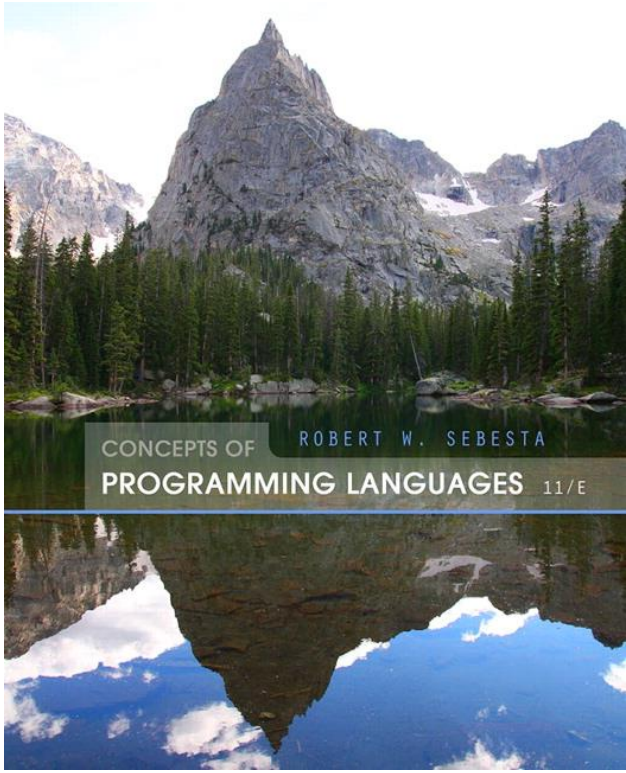
BBM 301 – Programming Languages

Fall 2020, Lecture 1

General Information



- **Webpage:** http://web.cs.hacettepe.edu.tr/~bbm301
- **Textbook:**
  - Robert W. Sebesta,
    "Concepts of Programming Languages",
    Pearson, 11th Edition (also 10th is OK)
- **Time and Place:**
  - Tuesdays 13:30 – 16:30
- **Communication:**
  - We will use Piazza
    https://piazza.com/hacettepe.edu.tr/fall2020/bbm301

Topics

- Introduction to Programming Languages
- Names, Bindings, Scope
- Syntax and Semantics
- Functional Languages
- Data Types
- Expressions, Assignments
- Control Statements
- Subprograms
- Implementing Subprograms
- Logic Languages

Grading Policy (Tentative)

- 20% Quizzes
- 20% Homeworks
- 20% Midterm exam
- 40% Final exam

- Attendance is mandatory.
- There will be a quiz every week, and you have to attend at least 70% of the quizzes
- At least 25/100 is required from final exam to pass the course

Lecture 1:
Introduction to Programming Languages

What is a (Programming) Language?

A language is a vocabulary and set of grammatical rules for communication between people.

A programming language is a vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. (Definition from webopedia)

Why Study Programming Languages?

- One or two languages is not enough for a computer scientist.
- You should know
  - the general concepts beneath the requirements
  - choices in designing programming languages.

Q: How many Programming Languages do you know?

How many programming languages are out there?

700 +

Source: Wikipedia (excluding dialects of BASIC)

https://en.wikipedia.org/wiki/List_of_programming_languages

New Languages will Keep Coming

Be prepared to program in new languages

Languages undergo constant change

- FORTRAN 1953
- ALGOL 60 1960
- C 1973
- C++ 1985
- Java 1995

Evolution steps: 12 years per widely adopted language

- are we overdue for the next big one?

... or is the language already here?

- *Hint:* are we going through a major shift in what computation programs need to express?
- your answer here:

JavaScript

Python

- programs are distributed
- more interactive
- "installed" as part of web page

11

Language as a thought shaper

We will cover less traditional languages, too.

The reason:

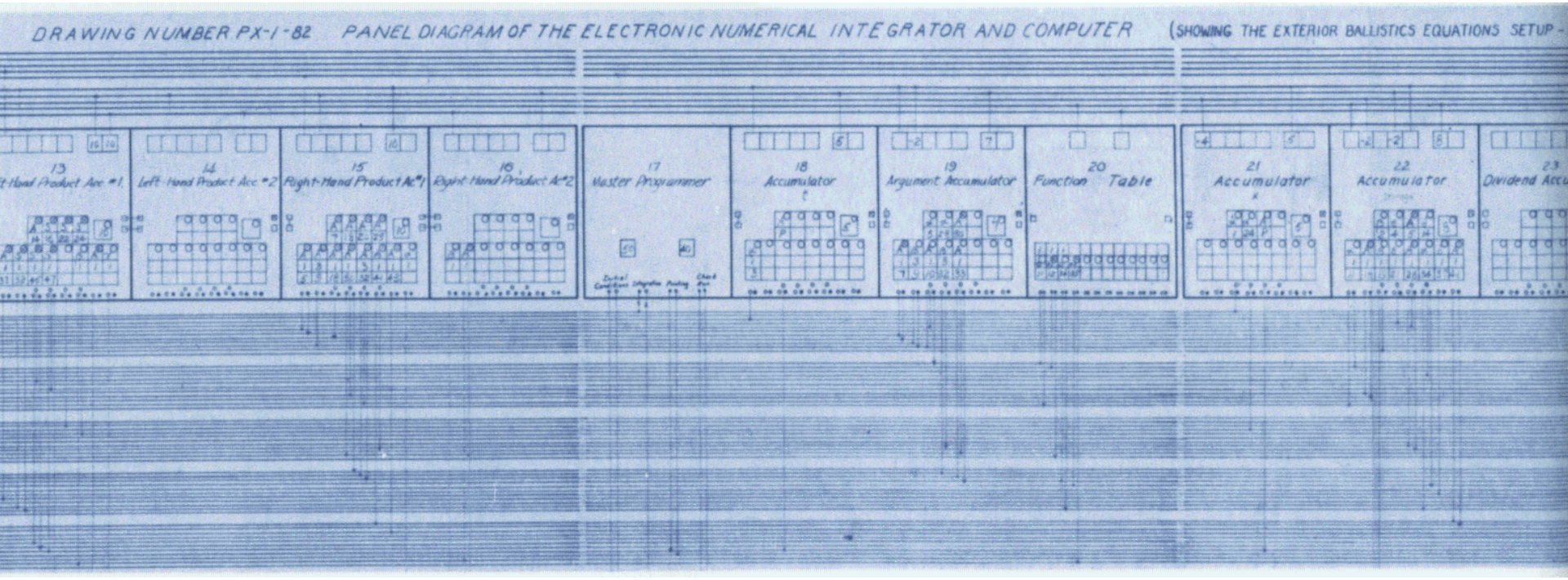
> *A language that doesn't affect the way you think about programming, is not worth knowing.*
>
> *an Alan Perlis epigram <http://www.cs.yale.edu/quotes.html>*

One of thought-shaper languages is Prolog.

You will both program in it and implement it.

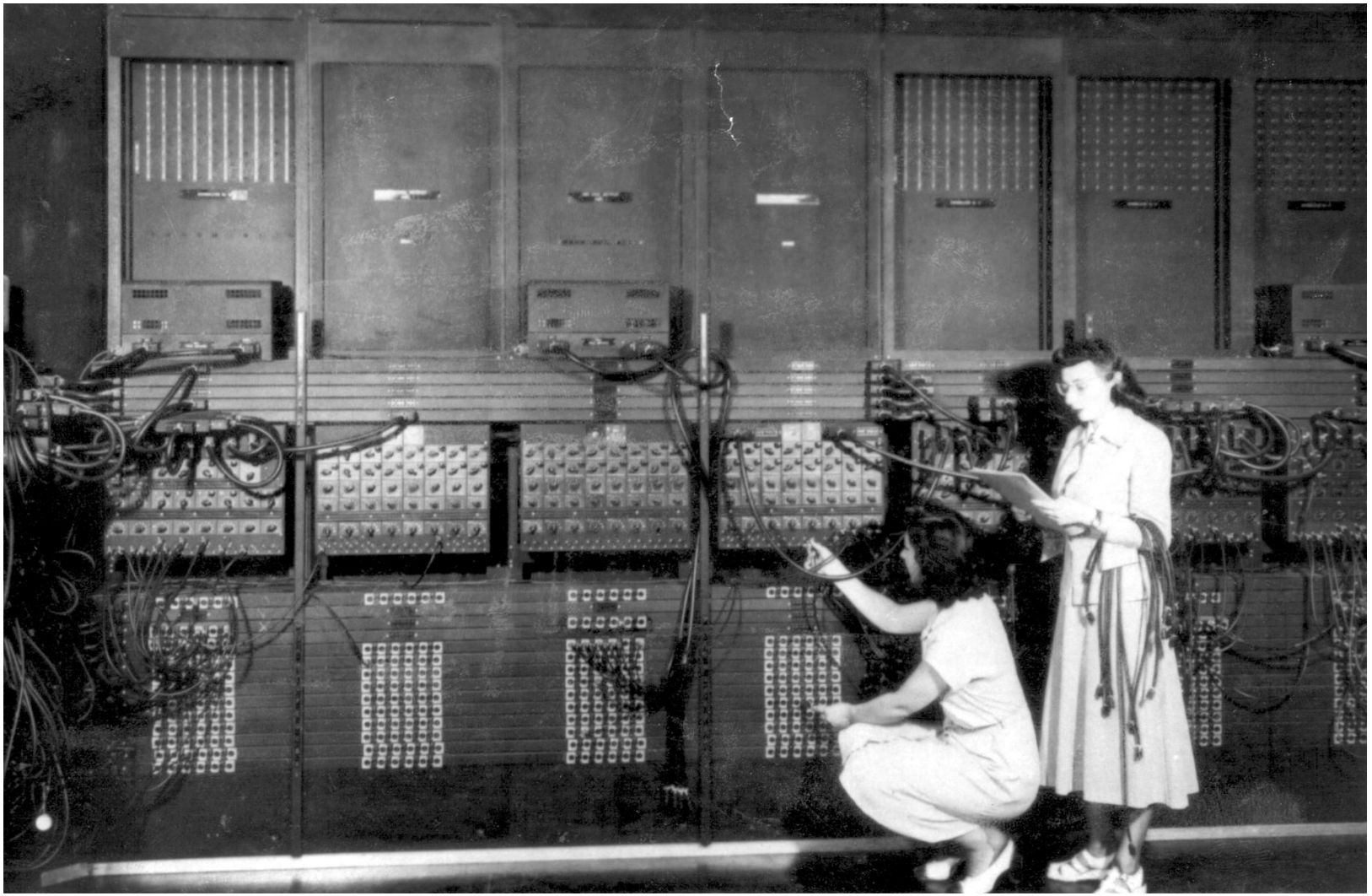
ENIAC (1946, University of Philadelphia)

ENIAC program for external ballistic equations:



Slide from Ras Bodik

Programming the ENIAC



Slide from Ras Bodik

ENIAC (1946, Univ. of Philadelphia)

programming done by

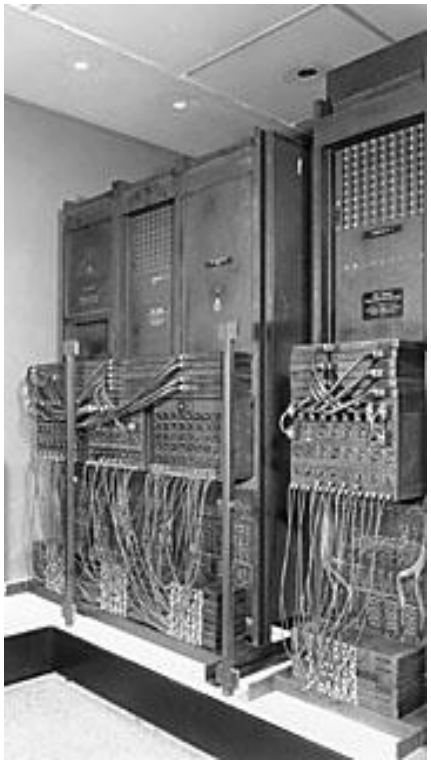
- rewiring the interconnections
- to set up desired formulas, etc

Problem (what's the tedious part?)

- programming = rewiring
- slow, error-prone

solution:

- store the program in memory!
- birth of *von Neuman* paradigm



Reasons for Studying Concepts of Programming Languages

Increased Ability to Express Ideas

- Natural languages:
  - The depth at which people think is influenced by the expressive power of the language.
  - The more appropriate constructs you have, the easier it is to communicate.
  - It is difficult for people to conceptualize structures that they cannot describe verbally.

Increased Ability to Express Ideas

- Programming Languages:
  - This is similar for PL's. The language in which you develop software puts limits on the kinds of data structures, control structures and abstractions that can be used.
  - Awareness of a wider variety of programming language features can reduce such limitations in software development.
  - Programmers increase the range of software development by learning new language constructs.
  - For example, if you learn associate arrays in Perl, you can simulate them in C.

Improved background for choosing appropriate languages

- Not every programming language can be suitable for all the software requirements.
- Many programmers learn one or two languages specific to the projects.
- Some of those languages may no longer be used.
- When they start a new project they continue to use those languages which are old and not suited to the current projects.

Improved background for choosing appropriate languages

- However another language may be more appropriate to the nature of the project.
  - **Lots of text processing -> Perl may be a good option.**
  - **Lots of matrix processing -> MATLAB can be used.**
- If you are familiar with the other languages, you can choose better languages.
- Studying the principles of PLs provides a way to judge languages:
  - **"The advantages of Perl for this problem are….","The advantages of Java are…."**

Increased ability to learn new languages

- Programming languages are still evolving
  - Many languages are very new, new languages can be added in time.
- If you know the programming language concepts you will learn other languages much easier.
  - For example, if you know concept of object oriented programming, it is easier to learn C++ after learning Java
- Just like natural languages,
  - The better you know the grammar of a language, the easier you find to learn a second language.
  - learning new languages actually causes you to learn things about the languages you already know

Languages in common use (2018)

Oct 2018	Oct 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.801%	+5.37%
2	2		C	15.376%	+7.00%
3	3		C++	7.593%	+2.59%
4	5	↑	Python	7.156%	+3.35%
5	8	↑	Visual Basic .NET	5.884%	+3.15%
6	4	↓	C#	3.485%	-0.37%
7	7		PHP	2.794%	+0.00%
8	6	↓	JavaScript	2.280%	-0.73%
9	-	⇈	SQL	2.038%	+2.04%
10	16	⇈	Swift	1.500%	-0.17%
11	13	↑	MATLAB	1.317%	-0.56%
12	20	⇈	Go	1.253%	-0.10%
13	9	⇊	Assembly language	1.245%	-1.13%
14	15	↑	R	1.214%	-0.47%
15	17	↑	Objective-C	1.202%	-0.31%
16	12	⇊	Perl	1.168%	-0.80%
17	11	⇊	Delphi/Object Pascal	1.154%	-1.03%

TIOBE programming community index

Languages in common use (2019)

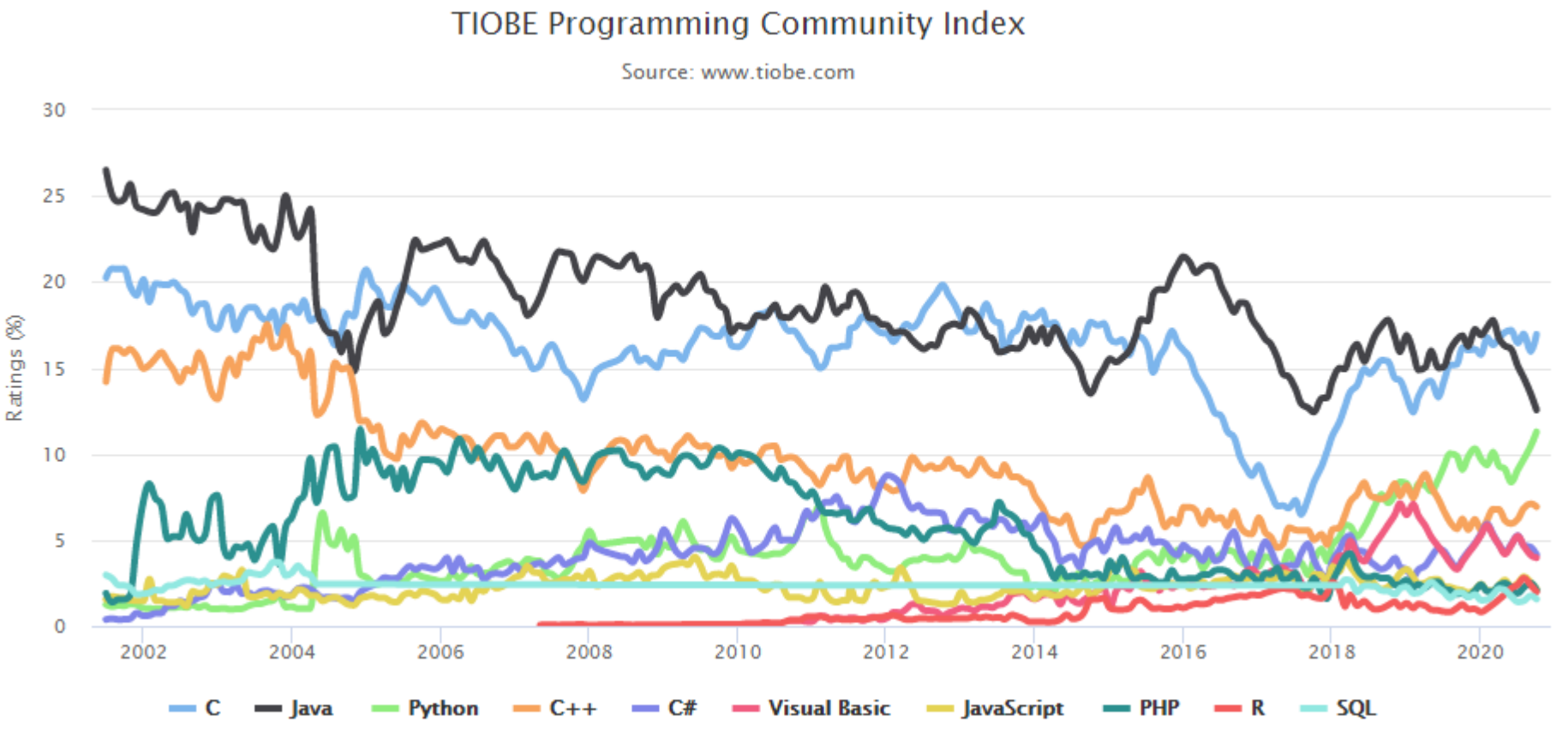
Oct 2019	Oct 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.884%	-0.92%
2	2		C	16.180%	+0.80%
3	4	↑	Python	9.089%	+1.93%
4	3	↓	C++	6.229%	-1.36%
5	6	↑	C#	3.860%	+0.37%
6	5	↓	Visual Basic .NET	3.745%	-2.14%
7	8	↑	JavaScript	2.076%	-0.20%
8	9	↑	SQL	1.935%	-0.10%
9	7	↓	PHP	1.909%	-0.89%
10	15	⇈	Objective-C	1.501%	+0.30%
11	28	⇈	Groovy	1.394%	+0.96%
12	10	↓	Swift	1.362%	-0.14%
13	18	⇈	Ruby	1.318%	+0.21%
14	13	↓	Assembly language	1.307%	+0.06%
15	14	↓	R	1.261%	+0.05%
16	20	⇈	Visual Basic	1.234%	+0.58%
17	12	⇊	Go	1.100%	-0.15%

TIOBE programming community index

Languages in common use (2020)

Oct 2020	Oct 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.95%	+0.77%
2	1	↓	Java	12.56%	-4.32%
3	3		Python	11.28%	+2.19%
4	4		C++	6.94%	+0.71%
5	5		C#	4.16%	+0.30%
6	6		Visual Basic	3.97%	+0.23%
7	7		JavaScript	2.14%	+0.06%
8	9	↑	PHP	2.09%	+0.18%
9	15	⇈	R	1.99%	+0.73%
10	8	↓	SQL	1.57%	-0.37%
11	19	⇈	Perl	1.43%	+0.40%
12	11	↓	Groovy	1.23%	-0.16%
13	13		Ruby	1.16%	-0.16%
14	17	↑	Go	1.16%	+0.06%
15	20	⇈	MATLAB	1.12%	+0.19%
16	12	⇊	Swift	1.09%	-0.28%
17	14	↓	Assembly language	1.08%	-0.23%
18	10	⇊	Objective-C	0.86%	-0.64%
19	16	↓	Classic Visual Basic	0.77%	-0.46%
20	22	↑	PL/SQL	0.77%	-0.06%

Languages in common use (today)



TIOBE programming community index

Change of indexes of languages

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	3	-	-
C	2	1	2	2	1	1	1
Python	3	7	6	6	22	20	-
C++	4	4	3	3	2	2	2
Visual Basic .NET	5	9	-	-	-	-	-
C#	6	5	5	7	17	-	-
JavaScript	7	8	8	9	13	-	-
PHP	8	6	4	5	-	-	-
SQL	9	-	-	89	-	-	-
Objective-C	10	3	26	36	-	-	-
Lisp	32	17	16	13	14	5	3
Pascal	219	15	14	88	7	3	21

An Interactive List

https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

Go, also known as Golang, is a statically typed, compiled programming language designed at Google by Robert Griesemer

Better understanding of significance of implementation

- The best programmers are the ones having at least understanding of **how things work under the hood**
  - Understand the implementation issues
- You can simply write a code and let the compiler do everything, but knowing implementation details helps you to **use a language more intelligently** and **write the code that is more efficient**
- Also, it allows you to visualize how a computer executes language constructs
  - Cost optimization; e.g. recursion is slow

Better use of languages that are already known

- Many programming languages are large and complex
  - It is uncommon to know all the features
- By studying the concepts of programming languages, programmers can learn about previously unknown parts of the languages easily.

Overall advancement of computing

- New ways of thinking about computing, new technology, hence need for new appropriate language concepts
- Not to repeat history
  - Although ALGOL 60 was a better language (with better block structure, recursion, etc) than FORTRAN, it did not become popular. If those who choose languages are better informed, better languages would be more popular.

Develop your own language

Are you kidding? No. Guess who developed:

- PHP
- Ruby
- JavaScript
- Perl

Done by smart hackers like you

- in a garage
- not in academic ivory tower

Our goal: learn good academic lessons

- so that your future languages avoid known mistakes

Slide from Ras Bodik

Ability to Design New Languages

- You may need to design a special purpose language to enter the commands for a software that you develop.
  - A language for an robotics interface

- Studying PL concepts will give you the ability to design a new language suitable and efficient for your software requirements.

Language Evaluation Criteria

The main criteria needed to evaluate various constructs and capabilities of programming languages

- Readability
- Writability
- Reliability
- Cost

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			•

Language Evaluation Criteria

The main criteria needed to evaluate various constructs and capabilities of programming languages

- **Readability**
- Writability
- Reliability
- Cost

Readability

Ease with which programs can be read and understood

- in the early times, efficiency and machine readability was important
- 1970s-Software life cycle: coding (small) + maintenance (large)

Readability is important for maintenance

- Characteristics that contribute to readability:
  - Overall simplicity
  - Orthogonality
  - Control statements
  - Data types and structures
  - Syntax Considerations

Overall Simplicity

- A manageable set of features and constructs
  - Large number of basic components - difficult to learn
- Minimal feature multiplicity
  - **Feature multiplicity:** having more than one way to accomplish an operation
    - e.g. In Java

```
count = count + 1
count += 1
count ++
++count
```

Overall Simplicity

- Minimal operator overloading
  - **Operator overloading:** a single operator symbol has more than one meaning
  - This can lead to reduced readability if users are allowed to create their own and do not do it sensibly

    Example:

    • using + for integer and floating point addition is acceptable and contributes to simplicity

    • but if a user defines + to mean the sum of all the elements of two single dimensional arrays is not, different from vector addition
- Simplest does not mean the best
  - Assembly languages: Lack the complex control statements, so program structure is less obvious

Orthogonality

- A relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language
- Every possible combination of primitives is legal and meaningful.
- Example:
  - Four primitive data types : integer, float, double and character
  - Two type operators : array and pointer
  - If the two type operators can be applied to themselves and the four primitive data types, a large number of data structures can be defined
  - However, if pointers were not allowed to point to arrays, many of those useful possibilities would be eliminated

Orthogonality

- **Example :** Adding two 32-bit integers residing in memory or registers, and replacing one of them with the sum

- IBM (Mainframe) Assembly language has two instructions:

```
A  Register1, MemoryCell1
AR Register1, Register2
```

More restricted
Less writable

Not orthogonal

meaning

Register1 ← contents(Register1) + contents(MemoryCell1)

Register1 ← contents(Register1) + contents(Register2)

- VAX Assembly language has one instruction:

```
ADDL operand1, operand2
```

orthogonal

meaning

operand2 ← contents(operand1) + contents(operand2)

Here, either operand can be a register or a memory cell.

Orthogonality

- Orthogonality is closely related to simplicity
- The more orthogonal the design of a language, the fewer exceptions the language rules require
- Too much orthogonality can cause problems as well:
- ALGOL68 is the most orthogonal language.
  - Every construct has a type
  - Most constructs produce values
  - This may result in extremely complex constructs,

Ex: A conditional can appear as the left side of an assignment statement, as long as it produces a location:

```
(if (A<B) then C else D) := 3
```

- This extreme form of orthogonality leads to unnecessary complexity
- *Functional languages offer a good combination of simplicity and orthogonality.*

Data Types and Structures

- Facilities for defining data types and data structures are helpful for readability
  - If there is no boolean type available then a flag may be defined as integer:

    `found = 1` (instead of `found = true`)

    May mean something is found as boolean or what is found is 1
  - An array of record type is more readable than a set of independent arrays

Syntax considerations

- **Identifier Forms:**
  - restricting identifier length is bad for readability.
    - FORTRAN77 identifiers can have at most 6 characters.
    - ANSI BASIC : an identifier is *either a single character* or *a single character followed by a single digit*.
  - Availability of word concatenating characters (e.g., _) is good for readability.
- **Special Words**: Readability is increased by special words(e.g., **`begin, end, for`**).
  - In PASCAL and C, **`end`** or **`}`** is used to end a compound statement. It is difficult tell what an end or **`}`** terminates.
  - However, ADA uses **`end if`** and **`end loop`** to terminate a selection and a loop, respectively.

Syntax Considerations

- **Forms and Meaning:** Forms should relate to their meanings. Semantics should directly follow from syntax.

  For example,

  `sin(x)` => should be the sine of x,

  not the sign of x or cosign of x.

  - **`grep`** is hard to understand for the people who are not familiar with using regular expressions

  `grep : g/reg_exp/p`

  => `/reg_exp/` : search for that reg_exp

  `g`: scope is whole file , make it global

  `p`: print

Language Evaluation Criteria

The main criteria needed to evaluate various constructs and capabilities of programming languages

- Readability
- **Writability**
- Reliability
- Cost

Writability

- Ease of creating programs
- Most of the characteristics that contribute to readability also contribute to writability
- Characteristics that contribute to writability
  - Simplicity and Orthogonality
  - Support for abstraction
  - Expressivity
- Writability of a language can also be application dependent

Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
  - A set of relatively convenient ways of specifying operations
  - Strength and number of operators and predefined functions

Simplicity and orthogonality

- Simplicity and orthogonality are also good for writability.
- When there are large number of constructs, programmers may not be familiar with all of them, and this may lead to either misuse or disuse of those items.
- A smaller number of primitive constructs (simplicity) and consistent set of rules for combining them (orthogonality) is good for writability
- However, too much orthogonality may lead to undetected errors, since almost all combinations are legal.

Support for abstraction

- **Abstraction:** ability to define and use complicated structures and operations in ways that allows ignoring the details.
- PLs can support two types of abstraction:
  - process
  - data
- Abstraction is the key concept in contemporary programming languages
- The degree of abstraction allowed by a programming language and the naturalness of its expressions are very important to its writability.

Process abstraction

- The simplest example of abstraction is subprograms (e.g., methods).
- You define a subprogram, then use it by ignoring how it actually works.
- Eliminates replication of the code
- Ignores the implementation details
  - e.g. sort algorithm

Data abstraction

- As an example of data abstraction, a tree can be represented more naturally using pointers in nodes.
- In FORTRAN77, where pointer types are not available, a tree can be represented using 3 parallel arrays, two of which contain the indexes of the offspring, and the last one containing the data.

Expressivity

- Having more convenient and shorter ways of specifying computations.
- For example, in C,

```
count++;
```

is more convenient and expressive than

```
count = count + 1;
```

`for` is more easier to write loops than `while`

Language Evaluation Criteria

The main criteria needed to evaluate various constructs and capabilities of programming languages

- Readability
- Writability
- **Reliability**
- Cost

Reliability

Reliable: it performs to its specifications under all conditions

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - The easier a program to write, the more likely it is correct.
  - Programs that are difficult to read are difficult both to write and modify.

Type Checking

- Testing for type errors in a given program either by the compiler or during program execution
- The compatibility between two variables or a variable and a constant that are somehow related (e.g., assignment, argument of an operation, formal and actual parameters of a method).
- **Run-time** (Execution-time) checking is expensive.
- **Compile-time** checking is more desirable.
- The earlier errors in programs are detected, the less expensive it is to make the required repairs

Type Checking

- Original C language requires no type checking neither in compilation nor execution time. That can cause many problems.
  - Current version required all parameters to be type-checked
- For example, the following program in original C compiles and runs!

```
foo  (float  a)  {
  printf  (“a:  %g  and  square(a):  %g\n”,  a,a*a);
}
main  ()  {
  char  z  =  ‘b’;
  foo(z);
}
```

- Output is : a: 98 and square(a): 9604

Language Evaluation Criteria

The main criteria needed to evaluate various constructs and capabilities of programming languages

- Readability
- Writability
- Reliability
- **Cost**

Cost

- Types of costs:
  1. **Cost of training the programmers:** Function of simplicity and orthogonality, experience of the programmers.
  2. **Cost of writing programs:** Function of the writability

  Note: These two costs can be reduced in a good programming environment

  3. **Cost of compiling programs:** cost of compiler, and time to compile
     - First generation Ada compilers were very costly

Cost

4. **Cost of executing programs:** If a language requires many run-time type checking, the programs written in that language will execute slowly.

– Trade-off between compilation cost and execution cost.

– *Optimization:* decreases the size or increases the execution speed.

- Without optimization, compilation cost can be reduced.
- Extra compilation effort can result in faster execution.
- More suitable in a production environment, where compiled programs are executed many times

Cost

5. **Cost of the implementation system.** If expensive or runs only on expensive hardware it will not be widely used.
6. **Cost of reliability** – important for critical systems such as a power plant or X-ray machine
7. **Cost of maintaining programs**. For corrections, modifications and additions.
   - Function of readability.
   - Usually, and unfortunately, maintenance is done by people other that the original authors of the program.
   - For large programs, the maintenance costs is about 2 to 4 times the development costs.

Other Criteria for Evaluation

- **Portability:** The ease with which programs can be moved from one implementation to another
- **Generality:** The applicability to a wide range of applications
- **Well-definedness:** The completeness and precision of the language's official definition

Language Design Trade-Offs

- Reliability vs. cost of execution
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs
- Readability vs. writability

  Example: APL provides many powerful operators for arrays(and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
  - Example: C++ pointers are powerful and very flexible but are unreliable

Programming Domains

- Scientific Applications (first digital computers –1940s)
  - Large floating-point arithmetic, execution efficiency, arrays and matrices, counting loops
  - **Examples**: FORTRAN, ALGOL 60, C
- Business Applications (1950s)
  - Producing elaborate reports, decimal numbers and character data
  - **Examples:** COBOL(1960s),Spreadsheets,Wordprocessors,Databases(SQL)
- Artificial Intelligence
  - Symbolic programming (names rather than numbers, linked lists rather than arrays)
  - **Examples:** LISP(1959),PROLOG(early1970s)

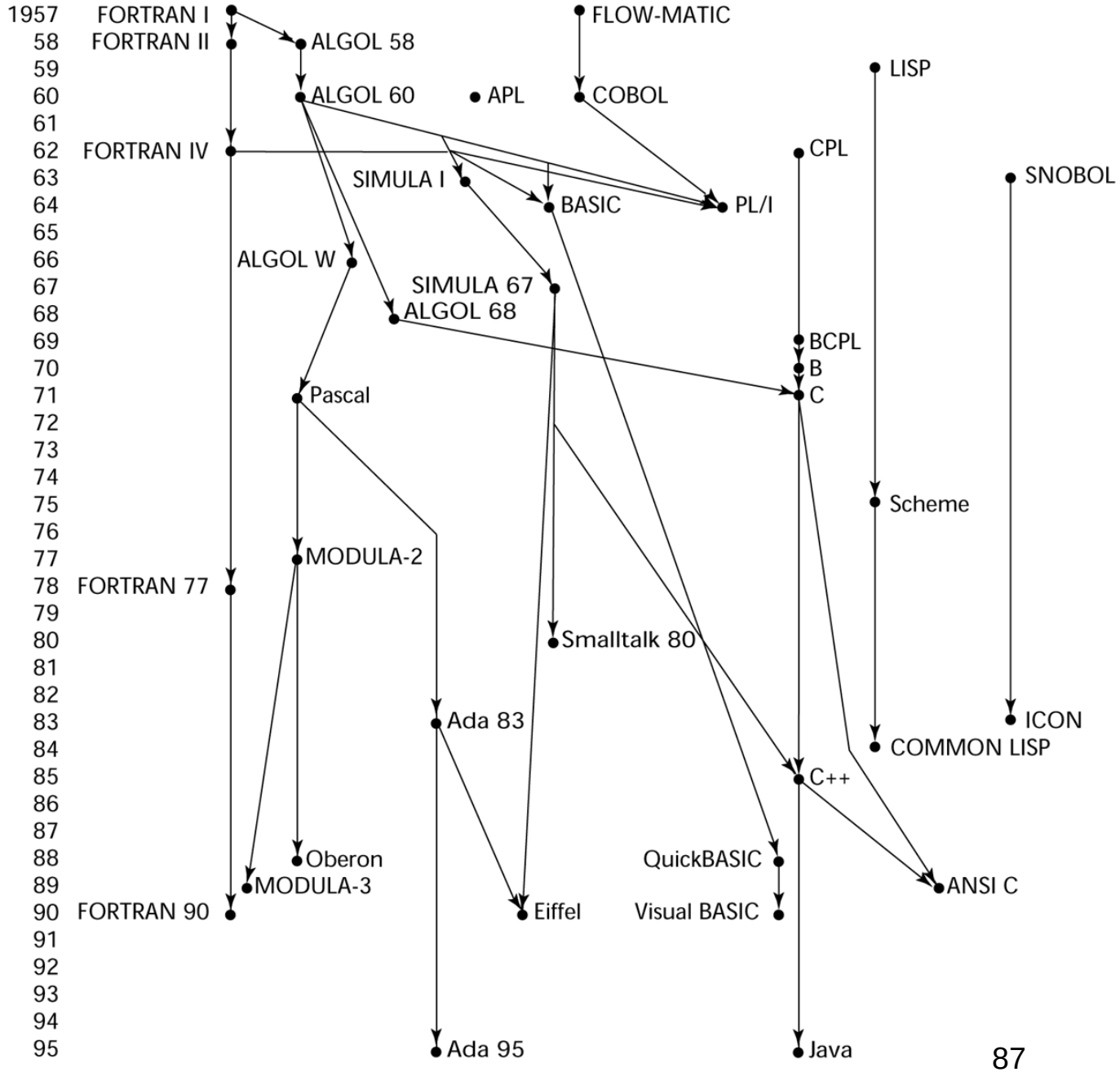
Programming Domains (cont'd.)

- Systems Programming
  - System software: Operating system and all of the programming support tools of a computer system
  - Efficient and fast execution, low-level features for peripheral device drivers
  - **Examples**: PLS/2(IBM Mainframe), BLISS (Digital), C (Unix)
- Scripting Languages
  - List of commands (Script) to be executed is put in a file.
  - **Examples:**sh, csh, tcsh, awk, gawk, tcl, perl, javascript
- Special-Purpose languages
  - **Examples**: RPG (Business Reports), SPICE (Simulation of Electronic Circuitry), SPSS (Statistics), Latex (Document preparation). HTML, XML (web prog.)

Language Categories

- **Imperative**
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include *scripting languages*
  - Include the *visual languages*
  - Examples: C, Java, Perl, Visual BASIC .NET, C++
- **Functional**
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme, ML, F#
- **Logic**
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- **Markup/programming hybrid**
  - Markup languages extended to support some programming
  - Examples: JSTL, XSLT

Genealogy of PLs



Living & Dead Languages

- Hundreds of programming languages popped up in the 1960s, most quickly disappeared
- Some dead:
  - JOVIAL, SNOBOL, Simula-67, RPG, ALGOL, PL/1, and many, many more
- Some still kicking:
  - LISP (1957)
  - BASIC (1964)
  - Pascal (1970)
  - Prolog (1972)
  - And of course, C (1973)

Slide by Thomas J. Cortina

“Hello World” in different languages

http://www.all-science-fair-projects.com/science_fair_projects_encyclopedia/Hello_world_program

Java

```
public class Hello {
  public static void main(String[] args) {
  System.out.println("Hello, world!");
  }
}
```

Assembly Language

```
bdos    equ 0005H   ; BDOS entry point
start:  mvi c,9     ; BDOS function: output string
lxi     d,msg$      ; address of msg
call    bdos ret    ; return to CCP
msg$:   db 'Hello, world!$'
end     start
```

FORTRAN

```
PROGRAM
HELLO
WRITE(*,10)
10 FORMAT('Hello, world!')
STOP
END
```

COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO-WORLD.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
DISPLAY "Hello, world!".
STOP RUN.
```

Ada

```
with Ada.Text_Io;
procedure Hello is
begin
  Ada.Text_Io.Put_Line ("Hello, world!");
end Hello;
```

C

```
#include <stdio.h>
int main()
{
  printf("Hello, world!\n");
  return 0;
}
```

C++

```
#include <iostream>
int main()
{
    std::cout << "Hello, world!\n";
}
```

C#

```
using System;
class HelloWorldApp
{
  public static void Main()
  {
     Console.WriteLine("Hello, world!");
  }
}
```

Scala

```
object HelloWorld extends App {
    println("Hello, World!")
    }
```

LISP

```
(format t "Hello world!~%")
```

PERL

```
print "Hello, world!\n";
```

Prolog

```
write('Hello world'),nl.
```

Python

```
print("Hello World!")
```

Swift

```
import Swift
print(“Hello World!”)
```

HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Hello, world!</title>
<meta http-equiv="Content-Type“ content="text/html;
charset=UTF-8">
</head>
<body>
<p>Hello, world!</p>
</body>
</html>
```

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost

An optional exercise

List three new languages, or major features added to established major languages, that have appeared in the last seven years. For each language, answer with one sentence these questions. (Use your own critical thinking; do not copy text from the Web.)

- *Why did the languages appear? Or, why have these features been added?* Often, a new language is motivated by *technical* problems faced by programmers. Sometimes the motivation for a new feature is *cultural*, related to, say, the educational background of programmers in a given language.

- *Who are the intended users of this language/feature?* Are these guru programmers, beginners, end-users (non-programmers)?

- *Show a code fragment that you find particularly cool.* The fragment should exploit the new features to produce highly readable and concise code.

Links that may help you start your exploration of the programming language landscape:

- http://lambda-the-ultimate.org/
- http://bit.ly/ddH47v
- http://www.google.com