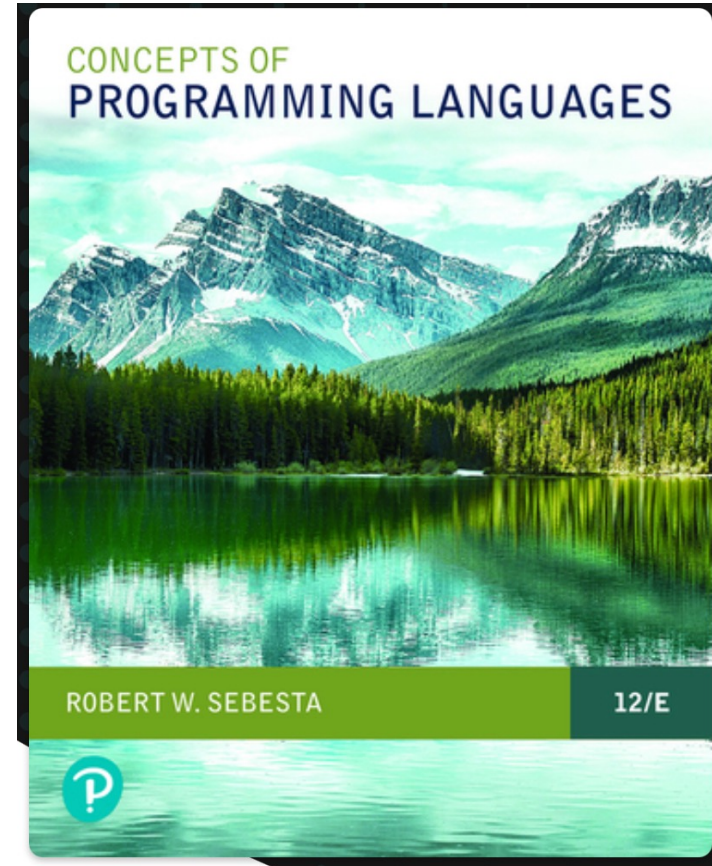


**BBM 301 –**  
**Programming Languages**  
*Lecture 1*

# General Information

- **Webpage:**  
<http://web.cs.hacettepe.edu.tr/~bbm301>
- **Textbook:**
  - Robert W. Sebesta,  
“Concepts of Programming Languages”,  
Pearson, 12<sup>th</sup> Edition (also 11<sup>th</sup> is OK)
- **Time and Place:**
  - Thursdays 09:40 – 12:30
- **Communication:**
  - We will use Piazza  
<https://piazza.com/hacettepe.edu.tr/fall2023/bbm301>



# Topics

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

# Grading Policy (Tentative)

- 30% Midterm exam 1
- 30% Midterm exam 2
- 40% Final exam

# **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](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

# 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.*

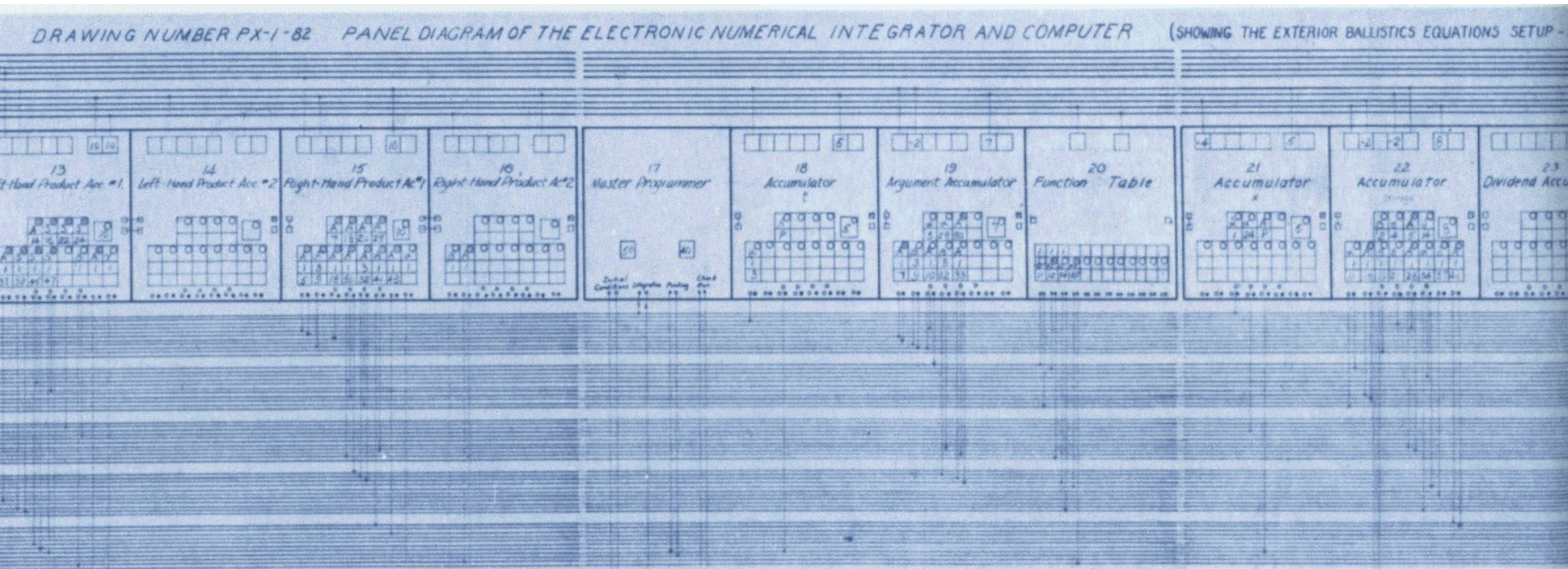
*an Alan Perlis epigram <<http://www.cs.yale.edu/quotes.html>>*

One of thought-shaper languages is Prolog.

You will both program in it and implement it.

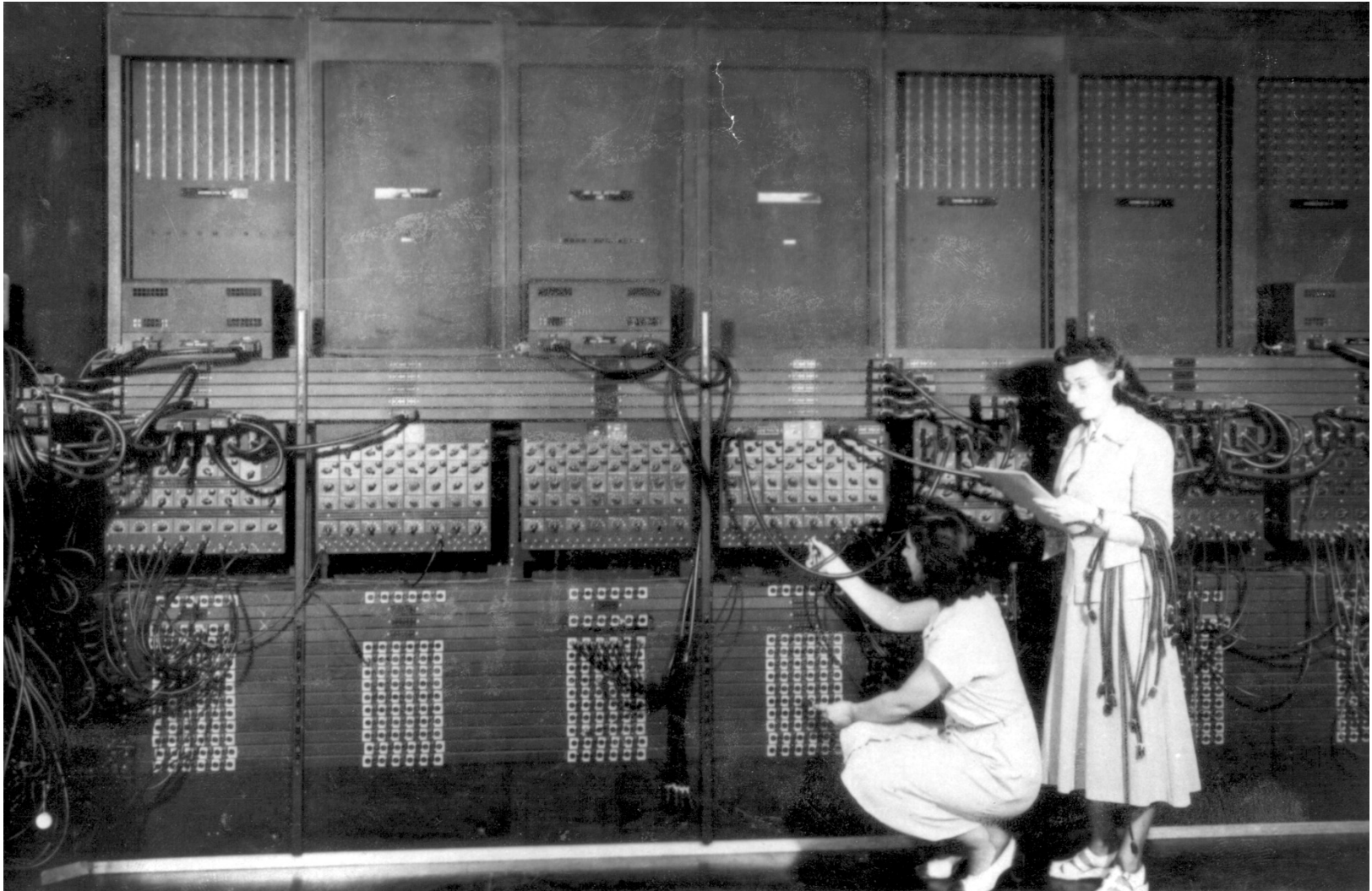
# ENIAC (1946, University of Philadelphia)

ENIAC program for external ballistic equations:





# Programming the ENIAC



# ENIAC (1946, Univ. of Philadelphia)

programming done by

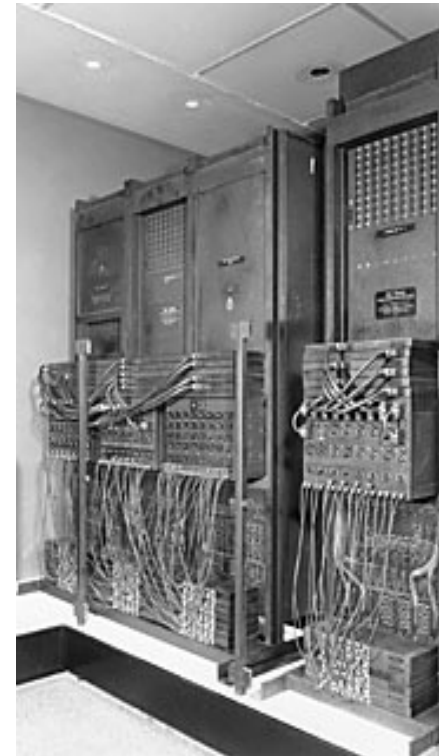
- rewiring the interconnections
- to set up desired formulas, etc

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 (2022)

Sep 2022	Sep 2021	Change	Programming Language		Ratings	Change
1	2	▲		Python	15.74%	+4.07%
2	1	▼		C	13.96%	+2.13%
3	3			Java	11.72%	+0.60%
4	4			C++	9.76%	+2.63%
5	5			C#	4.88%	-0.89%
6	6			Visual Basic	4.39%	-0.22%
7	7			JavaScript	2.82%	+0.27%
8	8			Assembly language	2.49%	+0.07%
9	10	▲		SQL	2.01%	+0.21%
10	9	▼		PHP	1.68%	-0.17%
11	24	▲▲		Objective-C	1.49%	+0.86%
12	14	▲		Go	1.16%	+0.03%
13	20	▲▲		Delphi/Object Pascal	1.09%	+0.32%
14	16	▲		MATLAB	1.06%	+0.04%
15	17	▲		Fortran	1.03%	+0.02%

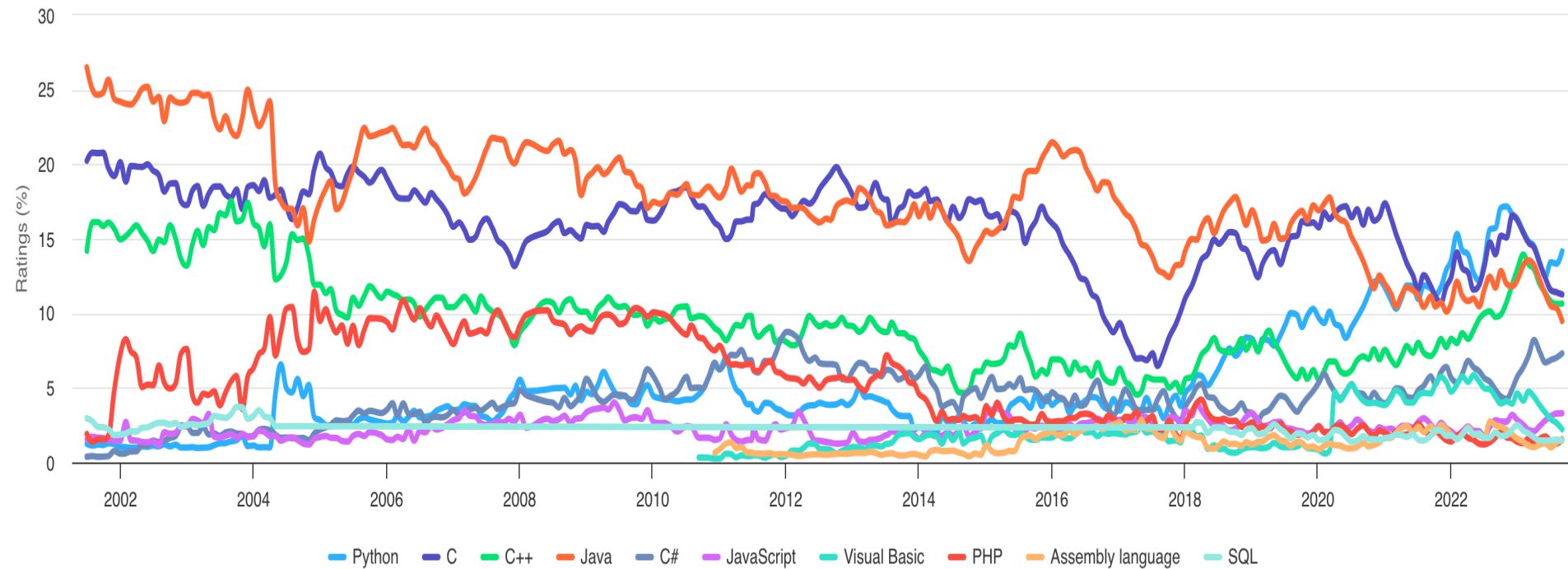
# Languages in common use (2023)

Sep 2023	Sep 2022	Change	Programming Language		Ratings	Change
1	1			Python	14.16%	-1.58%
2	2			C	11.27%	-2.70%
3	4	^		C++	10.65%	+0.90%
4	3	v		Java	9.49%	-2.23%
5	5			C#	7.31%	+2.42%
6	7	^		JavaScript	3.30%	+0.48%
7	6	v		Visual Basic	2.22%	-2.18%
8	10	^		PHP	1.55%	-0.13%
9	8	v		Assembly language	1.53%	-0.96%
10	9	v		SQL	1.44%	-0.57%
11	15	^^		Fortran	1.28%	+0.26%
12	12			Go	1.19%	+0.03%
13	14	^		MATLAB	1.19%	+0.13%
14	22	^^		Scratch	1.08%	+0.51%
15	13	v		Delphi/Object Pascal	1.02%	-0.07%

# Languages in common use (today)

TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)





# 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.
- 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

# Language Evaluation Criteria

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

- **Readability:** the ease with which programs can be read and understood
- **Writability:** the ease with which a language can be used to create programs
- **Reliability:** conformance to specifications (i.e., performs to its specifications)
- **Cost:** the ultimate total cost

# 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”**

## **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

$\text{Register1} \leftarrow \text{contents}(\text{Register1}) + \text{contents}(\text{MemoryCell1})$

$\text{Register1} \leftarrow \text{contents}(\text{Register1}) + \text{contents}(\text{Register2})$

- VAX Assembly language has one instruction:

`ADDL operand1, operand2`

**orthogonal**

meaning

$\text{operand2} \leftarrow \text{contents}(\text{operand1}) + \text{contents}(\text{operand2})$

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.

# 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 abstraction:** subprograms
  - **Data abstraction:** trees, linked lists, etc.
- 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.

# 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

**Table 1.1** Language evaluation criteria and the characteristics that affect them

Characteristic	CRITERIA		
	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**

# 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 than 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

# Home Exercise

## The Zen of Python

```
>>> import this  
The Zen of Python, by Tim Peters
```

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *\*right\** now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!



Daniel Greenfeld  
pydanny.com / @pydanny

How does Zen of Python relate to Language Evaluation criteria?

# 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



# Implementation Methods

- **Compilation**
  - Programs are translated into machine language; includes JIT systems
  - Use: Large commercial applications
- **Pure Interpretation**
  - Programs are interpreted by another program known as an interpreter
  - Use: Small programs or when efficiency is not an issue
- **Hybrid Implementation Systems**
  - A compromise between compilers and pure interpreters
  - Use: Small and medium systems when efficiency is not the first concern

**Watch the lecture recording on  
compilation and interpretation**

**From Piazza/resources**

# “Hello World” in different languages

<https://www.geeksforgeeks.org/hello-world-in-30-different-languages/>

# 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   ; 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:

- *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>