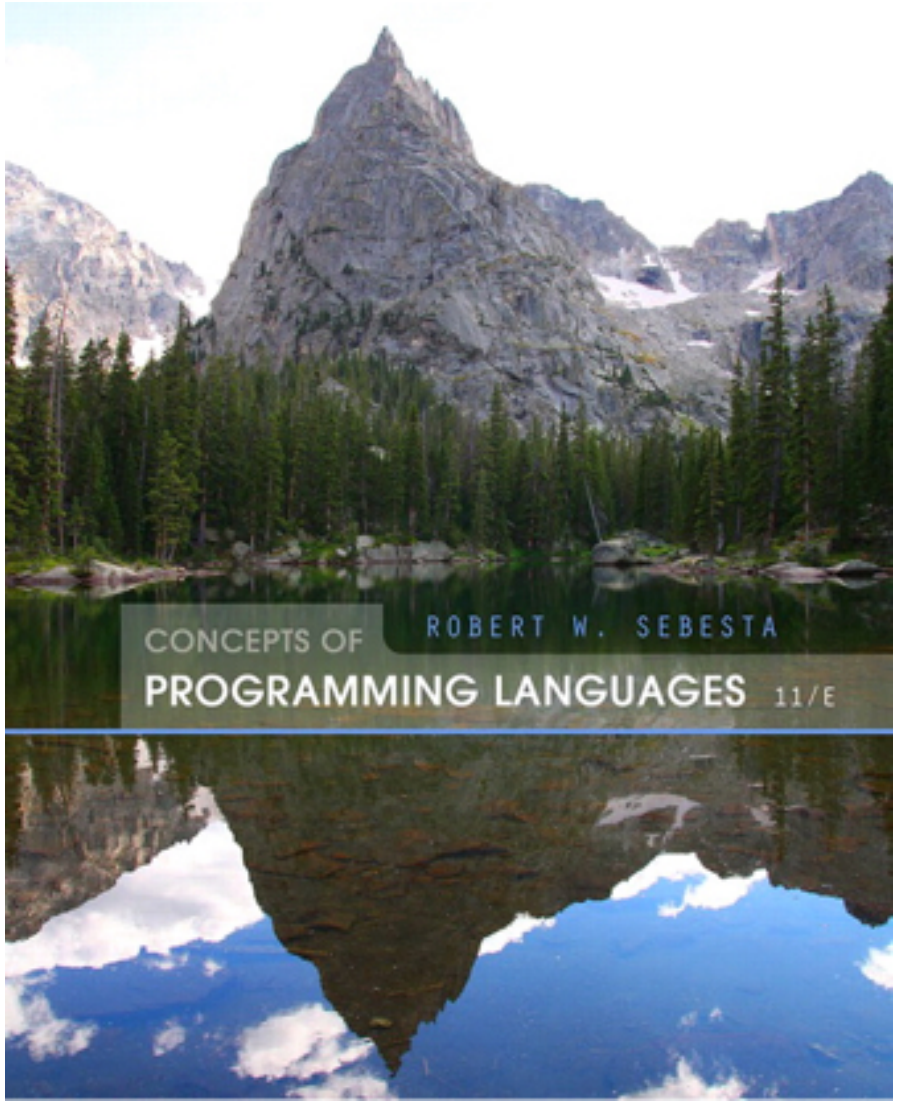
Programming Languages, Introduction

CSC419; Odelia Schwartz



leftoversalad.com

Know your programming languages:

Python

R

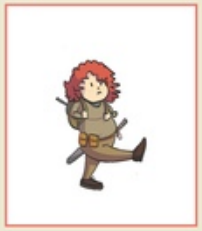
Java

JavaScript

PHP

Haskell

Perl



C

C++

C#

Ruby

Assembly

Erlang

Elixir

Go

Rust

MATLAB

Fortran

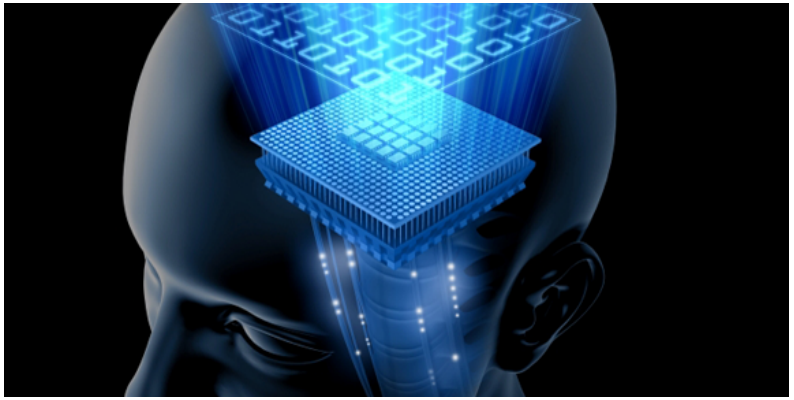
Lisp (dialects)

Brainfuck

Special thanks: @ScudaCorp, @wycats,G. Kiczales, S. Sever

Short introduction

- Odelia Schwartz
- Research interests
  - Computational neuroscience
  - Machine learning



  - Office: Ungar Building, Room No 310D
  - Email: odelia@cs.miami.edu (preferred)
  - Office Hours: email for appointment or when door open

More introduction...

- Teaching Assistant: Jack McKeown
  - Email: jam771 at miami.edu
  - Office hours: TBA

Short introduction

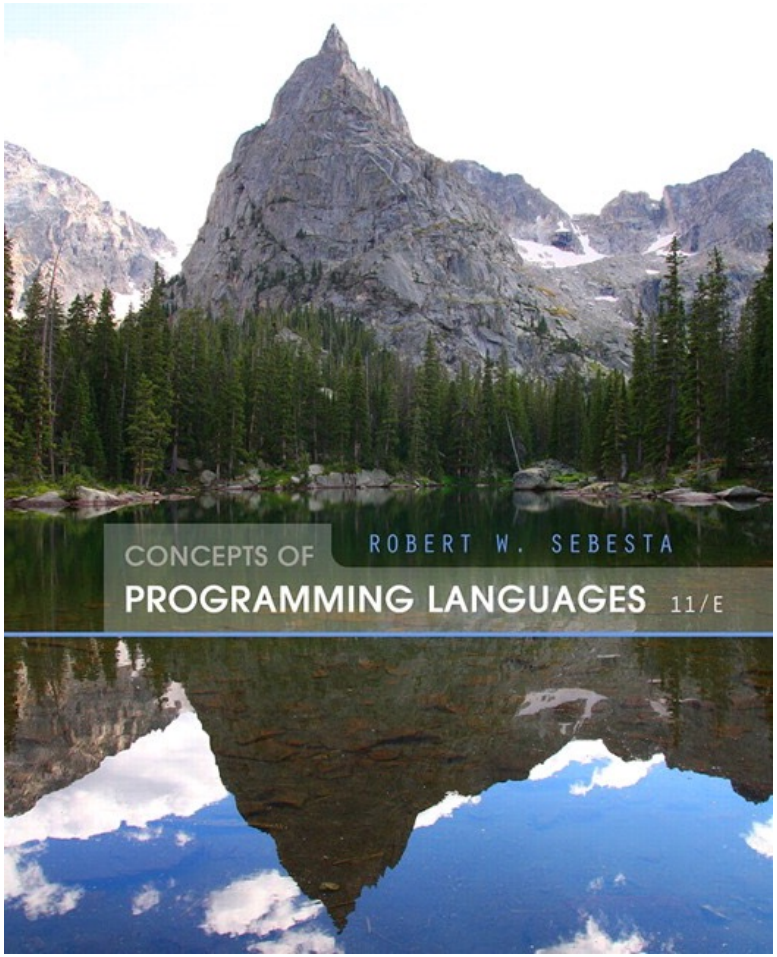
- Student introductions…

  what major?
  what languages?

- Hope to get out of course?

More introduction...

- Recommended Text Book
  - Concepts of Programming Languages 8-11/E (most recent version 12), Robert W. Sebesta, University of Colorado, Colorado Springs, Pearson.



More introduction...

- Recommended Text Book
  - Concepts of Programming Languages 8-11/E (most recent version 12), Robert W. Sebesta, University of Colorado, Colorado Springs, Pearson.

- Course Content
  - Chapters 1 to 3 (introductory chapters)
  - chapters 5-7 (aspects of imperative languages),
  - chapter 15 (functional programming languages),
  - chapter 16 (logic programming languages).
  - Course material will be uploaded after the lecture as .pdf files.
  - Check http://www.cs.miami.edu/home/odelia/teaching/csc419_spring19/index.html regularly. Content may change slightly during semester.

Grading & general issues

- Grading
  - will be based on 100 points:

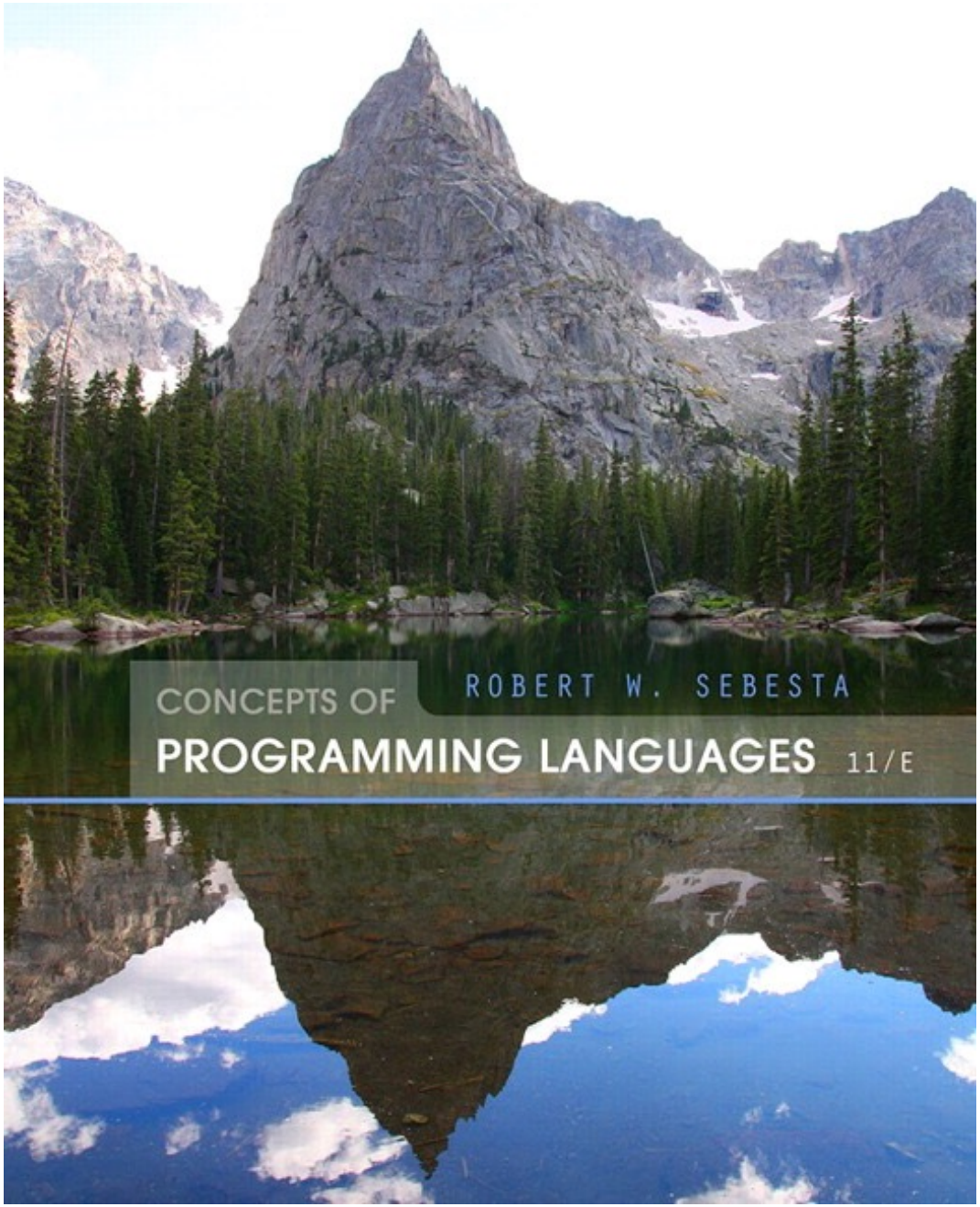
Item	Points
Homework	70
Final	30

- Scoring of Homework Assignments
  - The score of each homework will be mentioned in it.
  - The total score of all homework assignments will be scaled down to 70 points at the end of the semester for the purpose of final grading. For example, if all homework assignments collectively carry 100 points and a student gets 90 out of 100, he/she gets 90*70/100 or 63 out of 70 in Homework Assignment component for final grading.

- Participation, active engagement, and discussion in course encouraged … each of us may be expert in other languages and learn from one another

Chapter 1

Preliminaries



ISBN 0-321-49362-1

Chapter 1 Topics

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories
- Language Design Trade-Offs
- Implementation Methods
- Programming Environments

Reasons for studying concepts of PL?

Reasons for studying concepts of PL

- Increased ability to express ideas
- Improved background for choosing appropriate languages (when you open your startup… when solving particular problems)
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Better use of languages that are already known
- Overall advancement of computing

Programming Domains

- What languages used in?
- Scientific applications
- Business applications
- Artificial intelligence
- Systems programming
- Web Software

Programming Domains

- What languages used in?
- Scientific applications

Programming Domains

- What languages used in?
- Scientific applications
  - Large numbers of floating point computations; use of arrays
  - Fortran
  - More recently though not stressed in book: Matlab, Python !

Programming Domains

- What languages used in?
- Business applications

Programming Domains

- What languages used in?
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL

Programming Domains

- What languages used in?
- Artificial intelligence

Programming Domains

- What languages used in?
- Artificial intelligence
  - Symbols rather than numbers manipulated; use of linked lists
  - LISP
  - Not stressed in book: Machine learning - Python

Programming Domains

- What languages used in?
- Systems programming

Programming Domains

- What languages used in?
- Systems programming
  - Need efficiency because of continuous use (eg, reading from and writing to file; starting and stopping programs)
  - C

Programming Domains

- What languages used in?
- Web Software

Programming Domains

- What languages used in?
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., Javascript, PHP), general-purpose (e.g., Java)

Programming Domains

- Scientific applications
  - Large numbers of floating point computations; use of arrays
  - Fortran (more recently though not stressed in book: Matlab, Python)
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL
- Artificial intelligence
  - Symbols rather than numbers manipulated; use of linked lists
  - LISP
- Systems programming
  - Need efficiency because of continuous use (eg, reading from and writing to file)
  - C
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., Javascript, PHP), general-purpose (e.g., Java)

Language Categories?

Language Categories

- Imperative
- Functional
- Logical

Example languages?

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, JavaScript, Ruby, Visual BASIC .NET, C++, Python, …

Language Categories

- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples?

Language Categories

- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme, Haskell (e.g., recently in Facebook fighting spam)

Language Categories (2)

- Logic
  - Rule-based (rules are specified in no particular order)
  - Example?

Language Categories (2)

- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog (e.g., recently in IBM Watson)

Are there new languages?

Are there new languages?

- Yes…
  - Swift 2014 Apple
  - Google go 2009
  - Rust 2015 Mozilla
  - Pyro deep learning probabilistic language 2017 - eg, Uber AI

Language Examples

- Rosettacode:

  https://www.rosettacode.org/wiki/99_Bottles_of_Beer

  Also…

  http://www.99-bottles-of-beer.net

  https://www.rosettacode.org/wiki/A%2BB

- Look at some examples in different languages…

Language Examples

- C

https://www.rosettacode.org/wiki/99_Bottles_of_Beer#The_simple_solution

```c
int main(void)
{
  unsigned int bottles = 99;
  do
  {
    printf("%u bottles of beer on the wall\n", bottles);
    printf("%u bottles of beer\n", bottles);
    printf("Take one down, pass it around\n");
    printf("%u bottles of beer on the wall\n\n", --bottles);
  } while(bottles > 0);
  return EXIT_SUCCESS;
}
```

Language Examples

- Python

https://www.rosettacode.org/wiki/99_Bottles_of_Beer/Python#Normal_Code

```
def sing(b, end):
    print(b or 'No more','bottle'+('s' if b-1 else ''), end)

for i in range(99, 0, -1):
    sing(i, 'of beer on the wall,')
    sing(i, 'of beer,')
    print('Take one down, pass it around,')
    sing(i-1, 'of beer on the wall.\n')
```

Language Examples

- Java

  https://www.rosettacode.org/wiki/99_Bottles_of_Beer#Java

```
public class Beer
{
 static String bottles(final int n)
 {
  return MessageFormat.format("{0,choice,0#No more bottles|1#One bottle|2#{0} bottles} of beer", n);
 }
 public static void main(final String[] args)
 {
  String byob = bottles(99);
  for (int x = 99; x > 0;)
  {
   System.out.println(byob + " on the wall");
   System.out.println(byob);
   System.out.println("Take one down, pass it around");
   byob = bottles(--x);
   System.out.println(byob + " on the wall\n");
  }
 }
}
```

Language Examples

- Another Java

https://www.rosettacode.org/wiki/99_Bottles_of_Beer#Java

```
public class Beer extends JFrame implements ActionListener{
        private int x;
        private JButton take;
        private JTextArea text;
        public static void main(String[] args){
                new Beer();//build and show the GUI
        }

        public Beer(){
                x= 99;
                take= new JButton("Take one down, pass it around");
                text= new JTextArea(4,30);//size the area to 4 lines, 30 chars each
                text.setText(x + " bottles of beer on the wall\n" + x + " bottles of beer");
                text.setEditable(false);//so they can't change the text after it's displayed
                take.addActionListener(this);//listen to the button
                setLayout(new BorderLayout());//handle placement of components
                add(text, BorderLayout.CENTER);//put the text area in the largest section
                add(take, BorderLayout.SOUTH);//put the button underneath it
                pack();//auto-size the window
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//exit on "X" (I hate System.exit...
                setVisible(true);//show it
        }

        public void actionPerformed(ActionEvent arg0){
```

Language Examples

- Swift

  https://www.rosettacode.org/wiki/99_Bottles_of_Beer#Swift

```
for i in reverse(1...99) {

    println("\(i) bottles of beer on the wall, \(i) bottles of
beer.")
    let next = i == 1 ? "no" : i.description
    println("Take one down and pass it around, \(next) bottles of
beer on the wall.")
}
```

Language Examples

- APL code (1960s)

  https://www.rosettacode.org/wiki/99_Bottles_of_Beer#APL

```
bob ← { (⍕⍵), ' bottle', (1=⍵)↓'s of beer'}
bobw ← {(bob ⍵) , ' on the wall'}
beer ← { (bobw ⍵) , ', ', (bob ⍵) , ';
take one down and pass it around, ', bobw ⍵-1}↑beer¨ ⌽(1-⎕IO)+⍳99
```

Language Examples

- Scheme

https://www.rosettacode.org/wiki/99_Bottles_of_Beer#Scheme_2

```
(define (sing)
 (define (sing-to-x n)
  (if (> n -1)
    (begin
        (display n)
        (display "bottles of beer on the wall")
        (newline)
        (display "Take one down, pass it around")
        (newline)
        (sing-to-x (- n 1)))
    (display "would you wanna me to sing it again?")))
 (sing-to-x 99))
```

Language Examples

- Prolog

https://www.rosettacode.org/wiki/99_Bottles_of_Beer/Python#Normal_Code

```
bottles(0):-!.
bottles(X):-
    writef('%t bottles of beer on the wall \n',[X]),
    writef('%t bottles of beer\n',[X]),
    write('Take one down, pass it around\n'),
    succ(XN,X),
    writef('%t bottles of beer on the wall \n\n',[XN]),
    bottles(XN).

:- bottles(99).
```

Language Examples

- Assembly

https://www.rosettacode.org/wiki/99_Bottles_of_Beer/Assembly#360_Assembly

```
*         99 Bottles of Beer         04/09/2015
BOTTLES   CSECT
          USING  BOTTLES,R12
          LR     R12,R15
BEGIN     LA     R2,99               r2=99 number of bottles
          LR     R3,R2
LOOP      BCTR   R3,0                r3=r2-1
          CVD    R2,DW               binary to pack decimal
          MVC    ZN,EDMASKN          load mask
          ED     ZN,DW+6             pack decimal (PL2) to char (CL4)
          CH     R2,=H'1'            if r2<>1
          BNE    NOTONE1             then goto notone1
          MVI    PG1+13,C' '         1 bottle
          MVI    PG2+13,C' '         1 bottle
NOTONE1   MVC    PG1+4(2),ZN+2       insert bottles
          MVC    PG2+4(2),ZN+2       insert bottles
          CVD    R3,DW               binary to pack decimal
          MVC    ZN,EDMASKN          load mask

…
```

Language Evaluation Criteria

- What criteria are important?

Language Evaluation Criteria

- 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

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Minimal feature multiplicity
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures

Evaluation: Simplicity example

- Java: how many ways to increment an integer variable?

Evaluation: Simplicity example

- Example: Java; multiplicity of ways to increment:
  - count = count + 1
  - count+=1
  - count++
  - ++count

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Minimal feature multiplicity
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures

Evaluation: Examples of Orthogonality

- IBM: only certain combos allowed adding integers in registers and memory (not orthogonal)

  A Reg1, memory_cell
  AR Reg1, Reg2

  Reg1 <- contents(Reg1) + contents(memory_cell)
  Reg1 <- contents(Reg1) + contents(Reg2)

- VAX (orthogonal)

  ADDL operand_1, operand_2

  operand_1 <- contents(operand_1) + contents(operand_2)

Evaluation: Examples of Orthogonality

- The more orthogonal the design of a language, the fewer exceptions the language rule require. Also easier to learn.
- Too much orthogonality can also cause problems
  - ALGOL 68 is most orthogonal language
  - Unnecessary complexity due to extremely complex structures

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Minimal feature multiplicity
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures

Evaluation Criteria: Example of control

Lack of readability (goto; 1960s)…

```
Basic:
10 PRINT "Hello"
20 GOTO 50
30 PRINT "This text will not be printed"
40 END
50 PRINT "Goodbye"
```

Source: http://www.readybasic.com/referencemanual/commands/GOTO.html

Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Minimal feature multiplicity
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures

Evaluation Criteria: Readability (2)

- Data types and structures
  - Adequate predefined data types and structures (e.g., boolean with TRUE/FALSE)
  - The presence of adequate facilities for defining data structures
- Syntax considerations
  - Special words (eg, while, class, for) and methods of forming compound statements (all end or close brackets for any control statement, versus **end if** in Fortran95 and Ada, versus indent for Python)
  - Form and meaning: self-descriptive constructs, meaningful keywords (eg, grep in Unix)

Evaluation Criteria: 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 (e.g., OOP)
- Expressivity
  - A set of relatively convenient ways of specifying operations
  - Strength and number of operators and predefined functions

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors (e.g., original C 1978, int could be used in function that expected float but returned nonsense)
- Exception handling
  - Intercept run-time errors and take corrective measures (e.g., C ++,Java,C#, but not C)
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support “natural” ways of expressing an algorithm will have reduced reliability

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. C does not require index range checking.
- Readability vs. writability
  - Example: APL provides many powerful operators (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. Not included in Java.

Language Design Trade-Offs

- Readability: Example APL code:
  https://www.rosettacode.org/wiki/99_Bottles_of_Beer#APL

```
bob ← { (⍕⍵), ' bottle', (1=⍵)↓'s of beer'}
bobw ← {(bob ⍵) , ' on the wall'}
beer ← { (bobw ⍵) , ', ', (bob ⍵) , ';
take one down and pass it around, ', bobw ⍵-1}↑beer¨ ⍳(1-
⎕IO)+⍳99
```

Language Design Trade-Offs

- Readability: Compare to Python

https://www.rosettacode.org/wiki/99_Bottles_of_Beer/Python#Normal_Code

```
def sing(b, end):
    print(b or 'No more','bottle'+('s' if b-1 else ''), end)

for i in range(99, 0, -1):
    sing(i, 'of beer on the wall,')
    sing(i, 'of beer,')
    print('Take one down, pass it around,')
    sing(i-1, 'of beer on the wall.\n')
```

Language Evaluation Criteria

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

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity/orthogonality	•	•	•
Control structures	•	•	•
Data types and structures	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Evaluation Criteria: Cost

- Training programmers to use the language
- Writing programs (closeness to particular applications)
- Compiling programs (Ada initially high cost)
- Executing programs (run-time checks)
- Language implementation system: availability of free compilers (e.g., Java)
- Reliability: poor reliability leads to high costs
- Maintaining programs

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another (standardization: C++ committee in 1989, approved 1998!)
- 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

- Hoare 1973: “There are so many important but conflicting criteria, that their reconciliation and satisfaction is a major engineering task”

Influences on Language Design

- Computer Architecture
  - Languages are developed around the prevalent computer architecture, known as the von Neumann architecture
- Programming Methodologies
  - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages

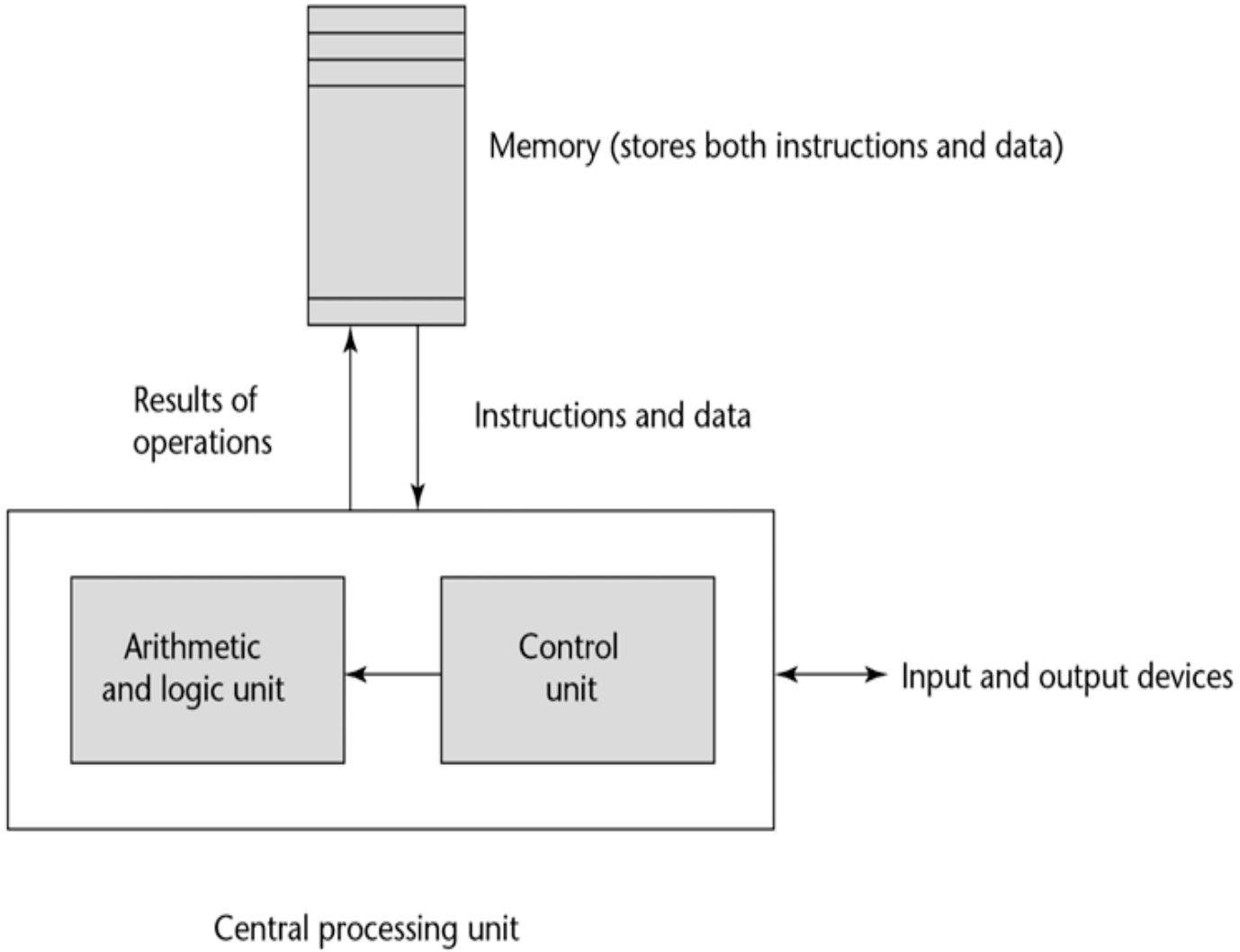
Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient

The von Neumann Architecture

Figure 1.1

The von Neumann computer architecture



The von Neumann Architecture

- Fetch-execute-cycle (on a von Neumann architecture computer)

```
initialize the program counter
repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

Von Neumann Bottleneck

Backus 1977 ACM Turing Award Lecture:

"… and a connecting tube that can transmit a single word between the CPU and the store (and send an address to the store). I propose to call this tube the Von Neumann bottleneck. The task of a program is to change the contents of the store in some major way; when one considers that this task must be accomplished entirely by pumping single words back and forth through the von Neumann bottleneck, the reason for its name becomes clear…"

Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed much faster than the speed of the connection; the connection speed thus results in a bottleneck
- Known as the von Neumann bottleneck; it is the primary limiting factor in the speed of computers

Programming Methodologies Influences

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming
  - Data abstraction + inheritance + polymorphism

Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
  - translate high-level programs to an intermediate language

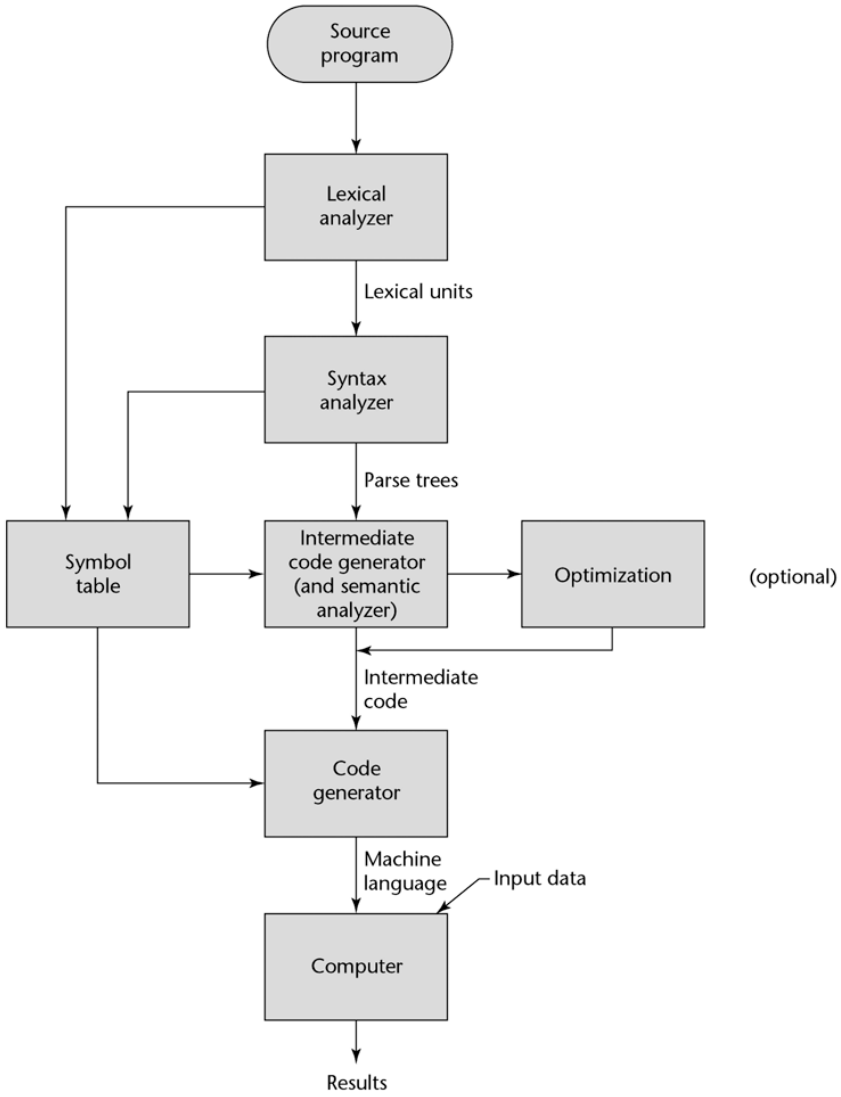
Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
  - lexical analysis: converts characters in the source program into lexical units
  - syntax analysis: transforms lexical units into parse trees which represent the syntactic structure of program
  - Semantics analysis: generate intermediate code
  - code generation: machine code is generated

The Compilation Process

Figure 1.3

The compilation process



Pure Interpretation

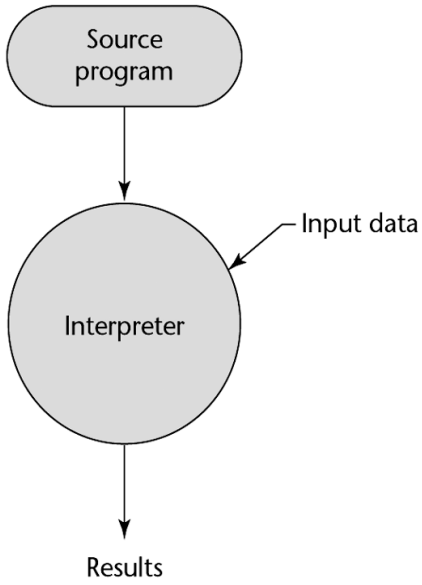


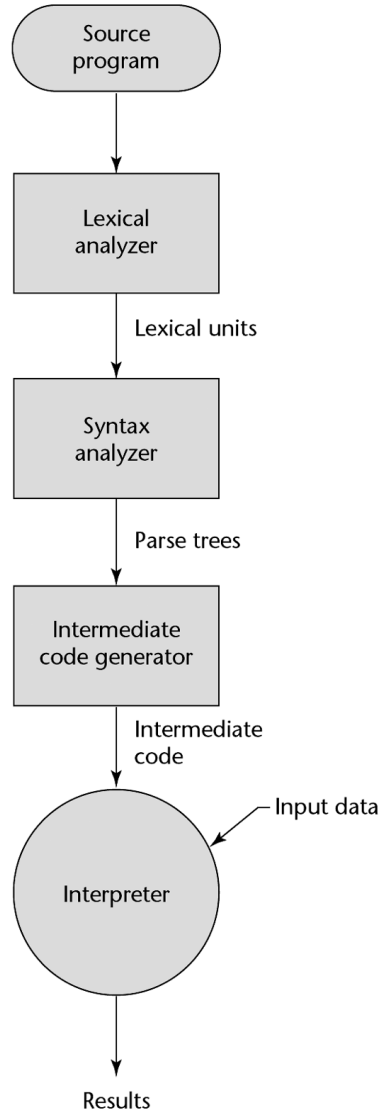
Figure 1.4
Pure interpretation

- No translation
- Easier implementation of programs (run-time errors can easily and immediately be displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Now rare for traditional high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript, PHP)

Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, byte code, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called Java Virtual Machine)

Figure 1.5
Hybrid implementation system



Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system

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
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation