Topic 6: Programming Languages

ICT170: Foundations of Computer Systems

Overview

- Overview
- Chronology
- A Selection of Languages
- Python Labs
- Summary

Objectives

In order to achieve the unit learning objectives, on successful completion of this topic, you should be able to:

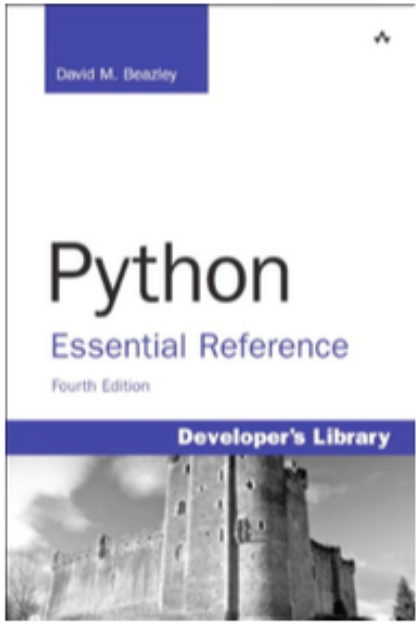
- Understand different styles of programming languages
- History of programming languages
- Roles and building blocks of a programming language

Reading

Title: **The C programming language**
Authors: Kernighan, Brian W. (Brian Wilson) and Ritchie, Dennis
Published:1988
Edition: 2nd ed.
Pages: 272
ISBN: 9780131103702
Language: English
Binding: Paperback
Reading: Chapter 1: A tutorial Introduction

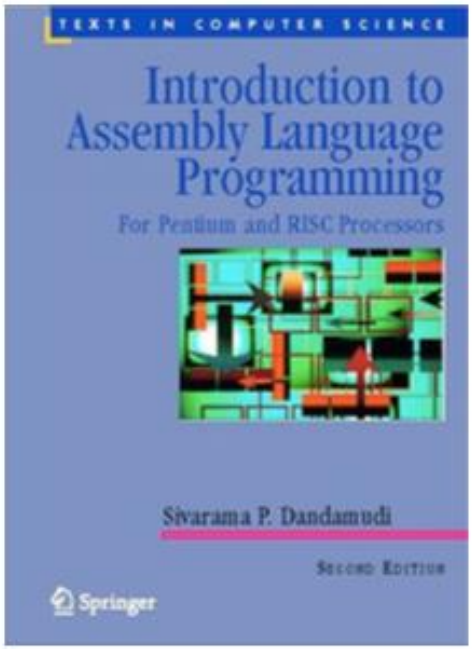
Reading

Title: **Python essential reference**
Authors: Beazley, David M
Published: 2001
Edition: 2nd ed
Pages: 476
ISBN: 0735710910
Language: English
Reading: Chapter 1. A Tutorial Introduction



Reading

TItle: **Introduction to assembly language programming: for Pentium and RISC processors**
Authors: Sivarama P. Dandamudi
Publisher: New York : Springer, c2005
Edition: 2nd ed
Language: English
Readings: Chapter 4. Overview of assembly language



Resources:

- The recorded lectures available on LMS.
- The lecture slides available on LMS.

Murdoch
UNIVERSITY

Overview: Programming Languages

What is a Programming Language?

Options:

- A formal language for describing computation?
- A “user interface” to a computer?
- Syntax + semantics?
- Compiler, or interpreter, or translator?
- A tool to support a programming paradigm?

> *A programming language is a notational system for describing computation in a machine-readable and human-readable form.*
>
> — **Programming Languages**: Principles and Practice by Kenneth C. **Louden**

What is a Programming Language?

Another view:

> *A programming language is a tool for developing executable models for a class of problem domains.*

Why Are There So Many Programming Languages

- Why do some people speak English? Some French?
- Programming languages have evolved over time as better ways have been developed to design them.
  - First programming languages were developed in the 1950s
  - Since then thousands of languages have been developed
- Different programming languages are designed for different types of programs.

Levels of Programming Languages

High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
00010010010001010010 01
00111011001010110100 1.
..
```

Generations of Programming Languages

First Generation Languages

Machine

0000 0001 0110 1110

0100 0000 0001 0010

Second Generation Languages

Assembly

LOAD x

ADD R1 R2

Third Generation Languages

High-level imperative/object oriented

```
public Token scan ( ) {
while (currentchar == ' '
|| currentchar == '\n')
{….} }
```

Fortran, Pascal, Ada, C, C++, Java, C#

Fourth Generation Languages

Database

```
select fname, lname
from employee
where department='Sales'
```

SQL

Fifth Generation Languages

Lisp, SML, Haskel, Prolog

Functional

```
fact n = if n==0 then 1
else n*(fact n-1)
```

Logic

```
uncle(X,Y) :- parent(Z,Y), brother(X,Z).
```

Beyond Fifth Generation Languages

Some talk about

- Agent Oriented Programming
- Aspect Oriented Programming
- Intentional Programming
- Natural language programming

Maybe you will invent the next big language

How do Programming Languages Differ?

Common Constructs:

- basic data types (numbers, etc.); variables; expressions; statements; keywords; control constructs; procedures; comments; errors ...

Uncommon Constructs:

- type declarations; special types (strings, arrays, matrices, ...); sequential execution; concurrency constructs; packages/modules; objects; general functions; generics; modifiable state; ...

Programming Paradigms

A programming language is a problem-solving tool.

Imperative style: Fortran, Pascal, C	program = algorithms + data *good for decomposition*
Functional style: Lisp, Scheme, Haskell, SML, F#	program = functions and functions *good for reasoning*
Logic programming style: Prolog	program = facts + rules *good for searching*
Object-oriented style: Simula, SmallTalk, C++, Java, C#	program = objects + messages *good for modeling(!)*

Other styles and paradigms: blackboard, pipes and filters, constraints, lists, ...

What determines a “good” language

Formerly: Run-time performance

- (Computers were more expensive than programmers)

Now: Life cycle (human) cost is more important

- Ease of designing, coding
- Debugging
- Maintenance
- Reusability

FADS

Criteria in a good language design

- Readability
  - understand and comprehend a computation easily and accurately
- Write-ability
  - express a computation clearly, correctly, concisely, and quickly
- Reliability
  - assures a program will not behave in unexpected or disastrous ways
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
  - Lack of orthogonality leads to exceptions to rules

Criteria in a good language design

- Uniformity
  - similar features should look similar and behave similar
- Maintainability
  - errors can be found and corrected and new features added easily
- Generality
  - avoid special cases in the availability or use of constructs and by combining closely related constructs into a single more general one
- Extensibility
  - provide some general mechanism for the user to add new constructs to a language
- Standardability
  - allow programs to be transported from one computer to another without significant change in language structure
- Implementability
  - ensure a translator or interpreter can be written

Backus-Naur Form

Usually CFG are written in BNF notation.

A production rule in BNF notation is written as:

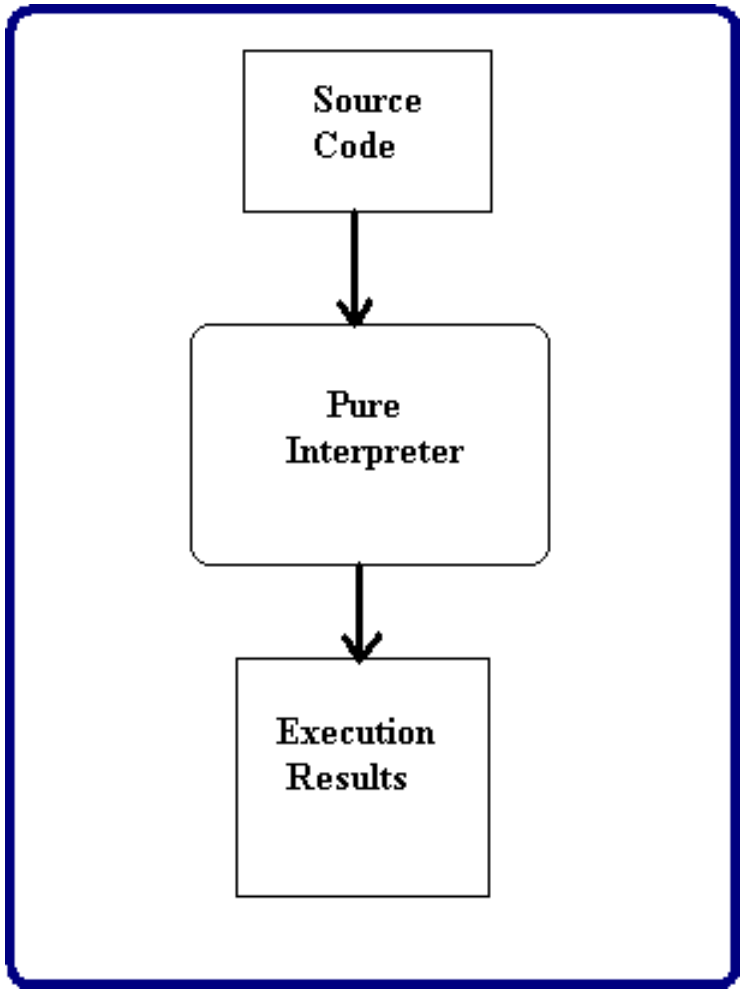
$N ::= \alpha$ where N is a non terminal
and α a sequence of terminals and non-terminals

$N ::= \alpha \mid \beta \mid \ldots$ is an abbreviation for several rules with N as left-hand side.

Murdoch
UNIVERSITY

Inside a programming language

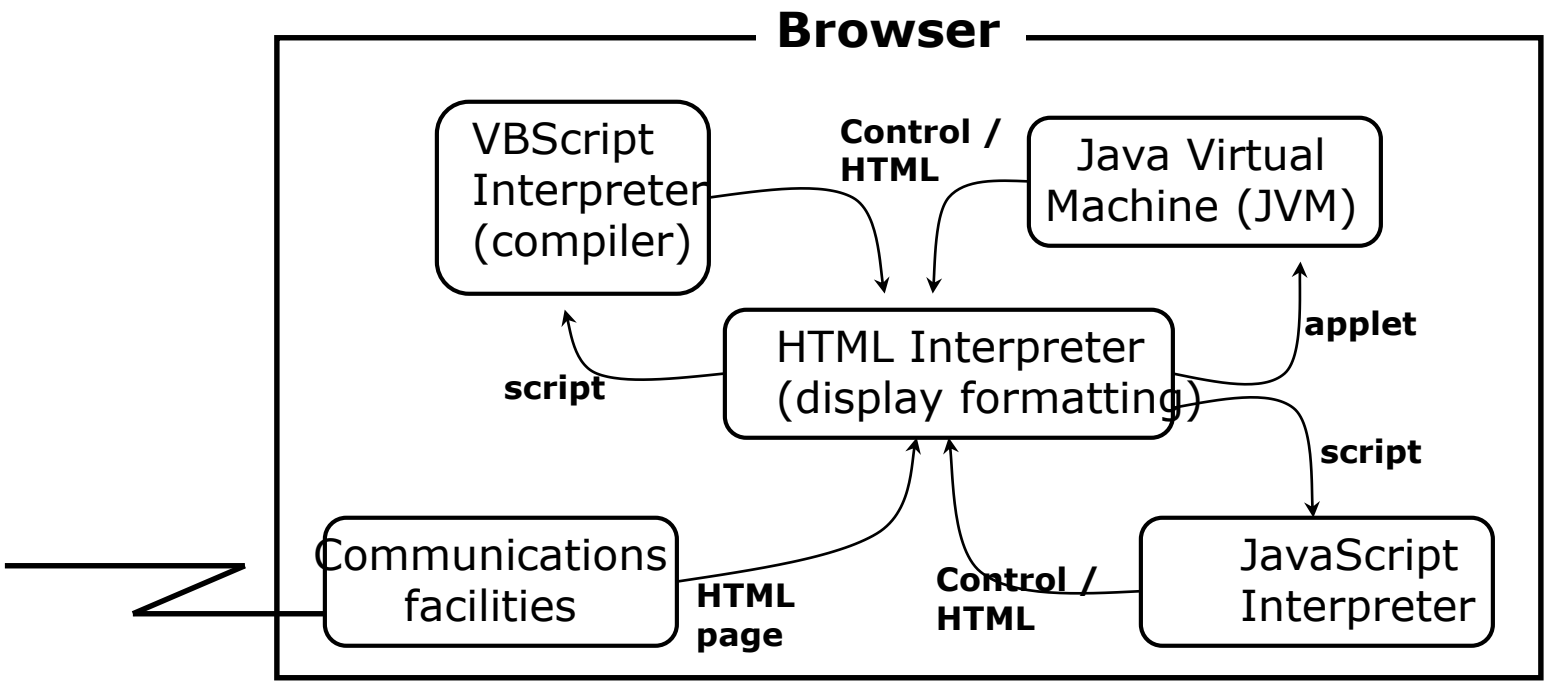
Interpreter



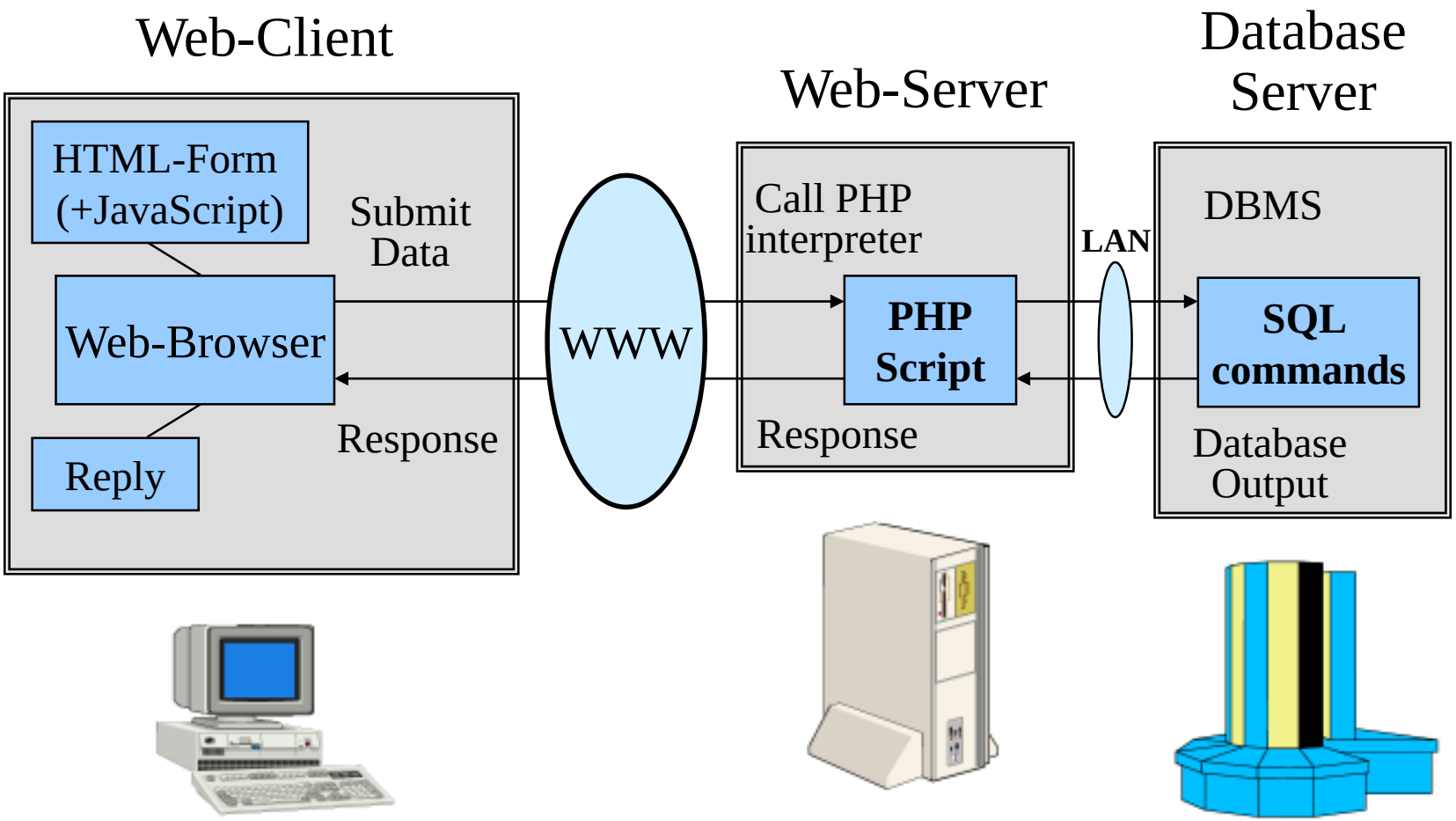
We use lots of interpreters every day!

Several languages are used to add dynamics and animation to HTML.

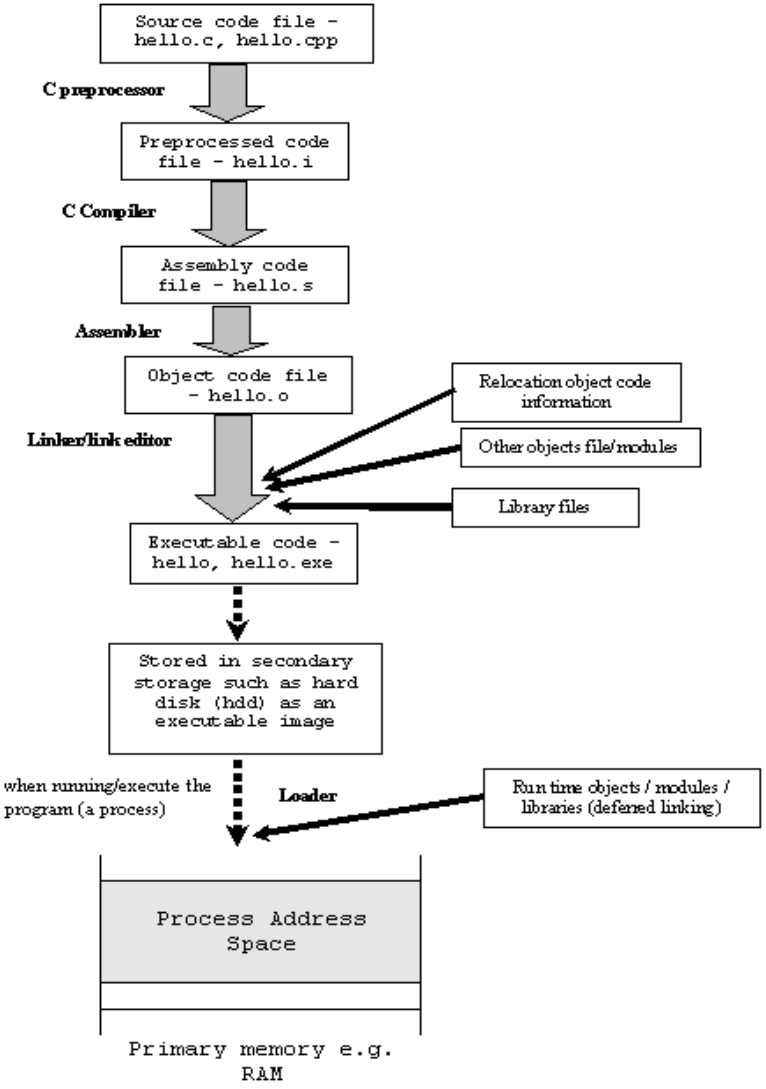
Many programming languages are executed (possibly simultaneously) in the browser!



And also across the web



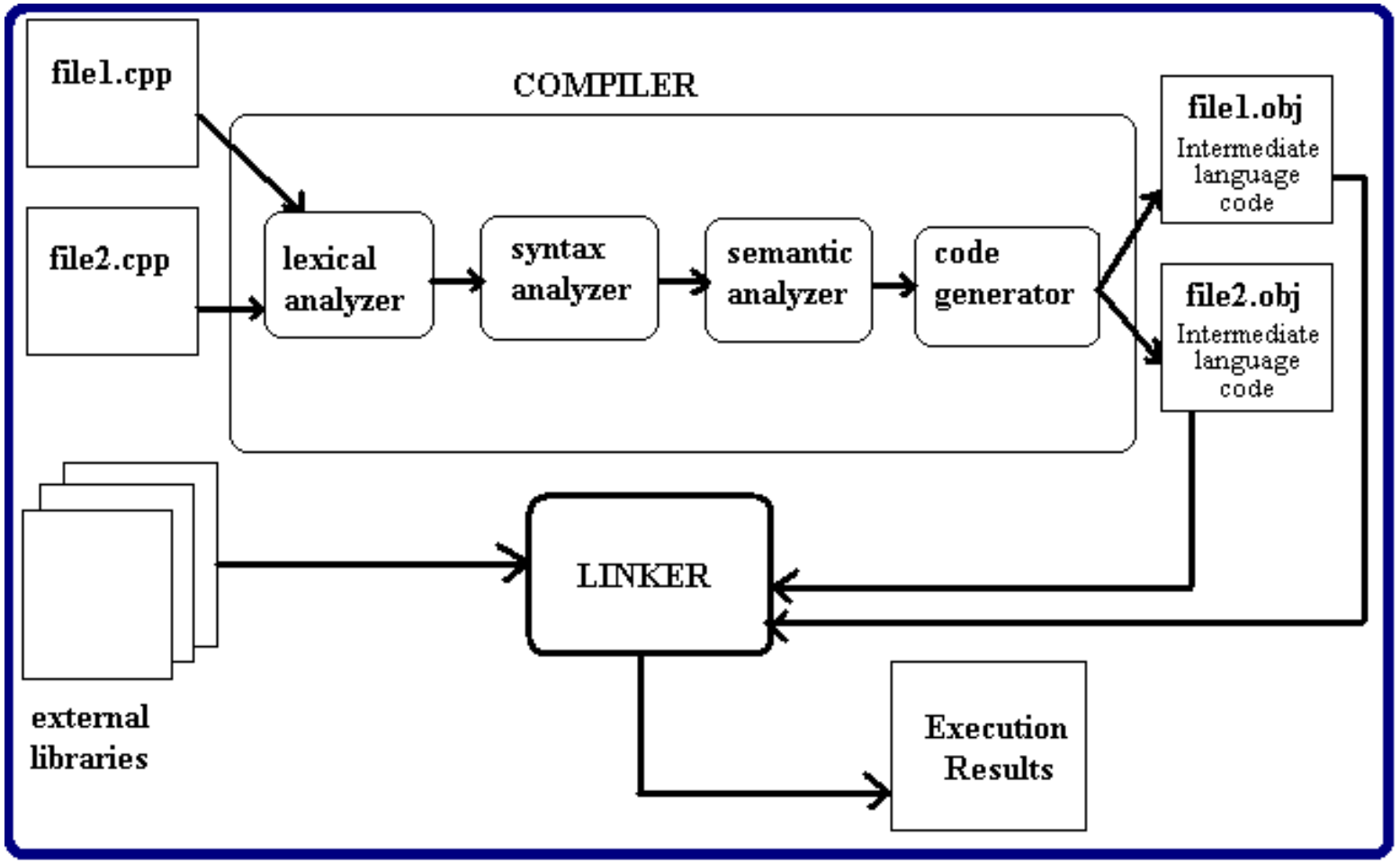
Source code to machine code



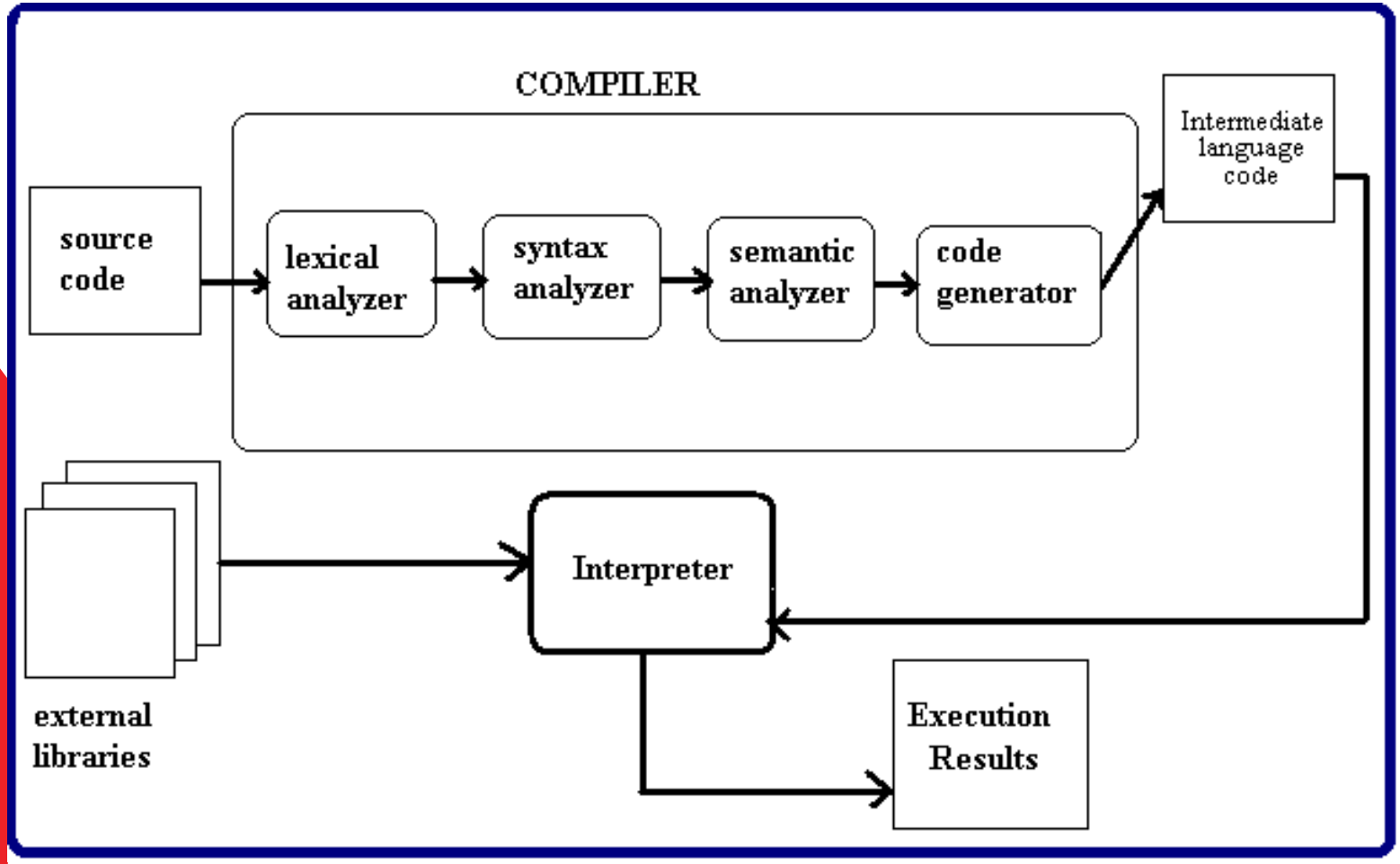
Compilation

- Compilation is at least a two-step process, in which the original program (source program) is input to the compiler, and a new program (target program) is output from the compiler. The compilation steps can be visualized as the following.

Compiler


file1.cpp
file2.cpp
COMPILER
lexical analyzer
syntax analyzer
semantic analyzer
code generator
file1.obj
Intermediate language code
file2.obj
Intermediate language code
external libraries
LINKER
Execution Results

Hybrid compiler / interpreter



Murdoch
UNIVERSITY

Chronology

Mur
UNIV

A Brief Chronology

Early 1950s		*"order codes" (primitive assemblers)*
1957	FORTRAN	*the first high-level programming language*
1958	ALGOL	*the first modern, imperative language*
1960	LISP, COBOL	*Interactive programming; business programming*
1962	APL, SIMULA	*the birth of OOP (SIMULA)*
1964	BASIC, PL/I	
1966	ISWIM	*first modern functional language (a proposal)*
1970	Prolog	*logic programming is born*
1972	C	*the systems programming language*
1975	Pascal, Scheme	*two teaching languages*
1978	CSP	*Concurrency matures*
1978	FP	*Backus' proposal*
1983	Smalltalk-80, Ada	*OOP is reinvented*
1984	Standard ML	*FP becomes mainstream (?)*
1986	C++, Eiffel	*OOP is reinvented (again)*
1988	CLOS, Oberon, Mathematica	
1990	Haskell	*FP is reinvented*
1990s	Perl, Python, Ruby, JavaScript	*Scripting languages become mainstream*
1995	Java	*OOP is reinvented for the internet*
2000	C#	

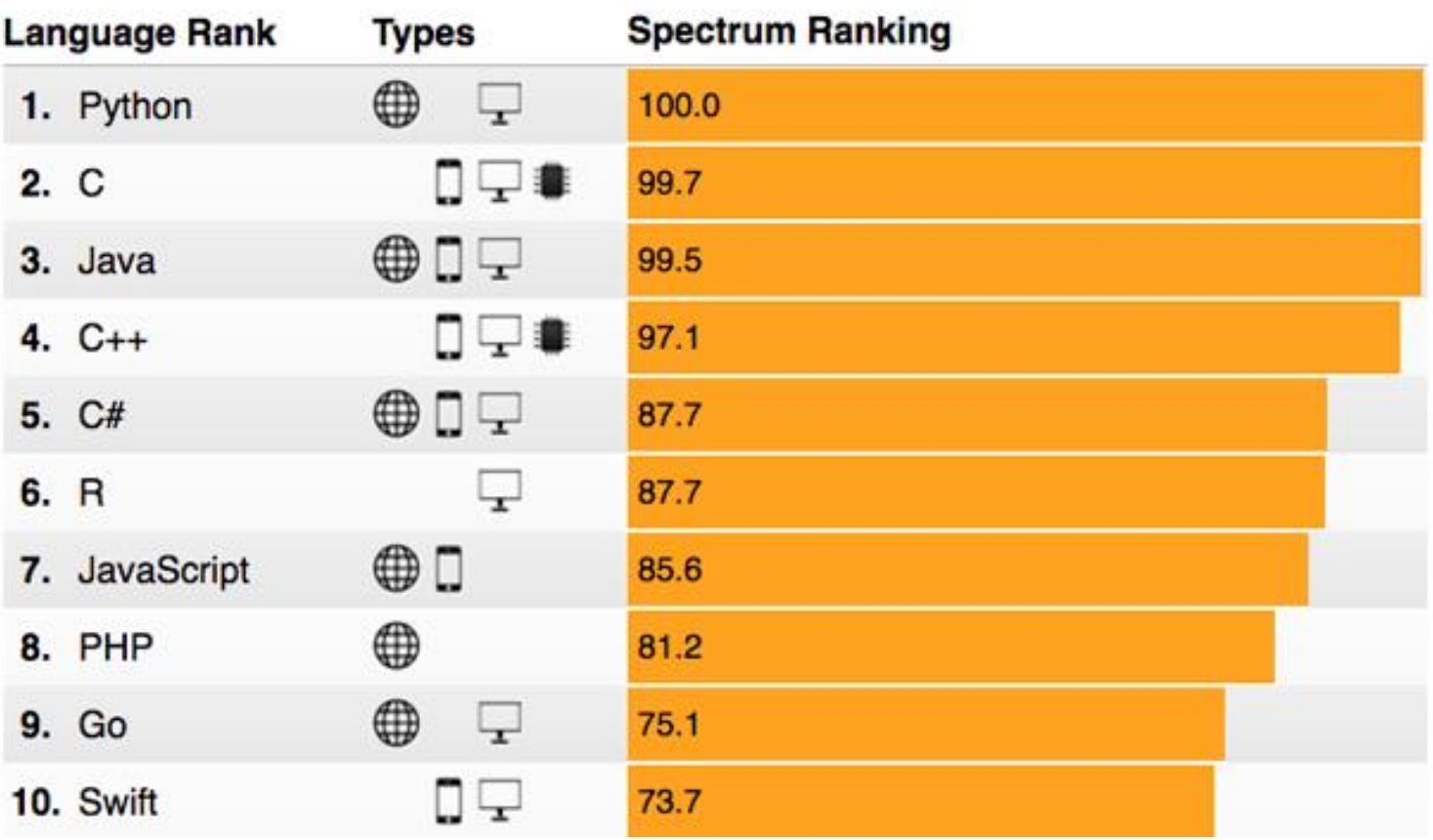
Murdoch
UNIVERSITY

A Selection of Languages

Contemporary programming languages

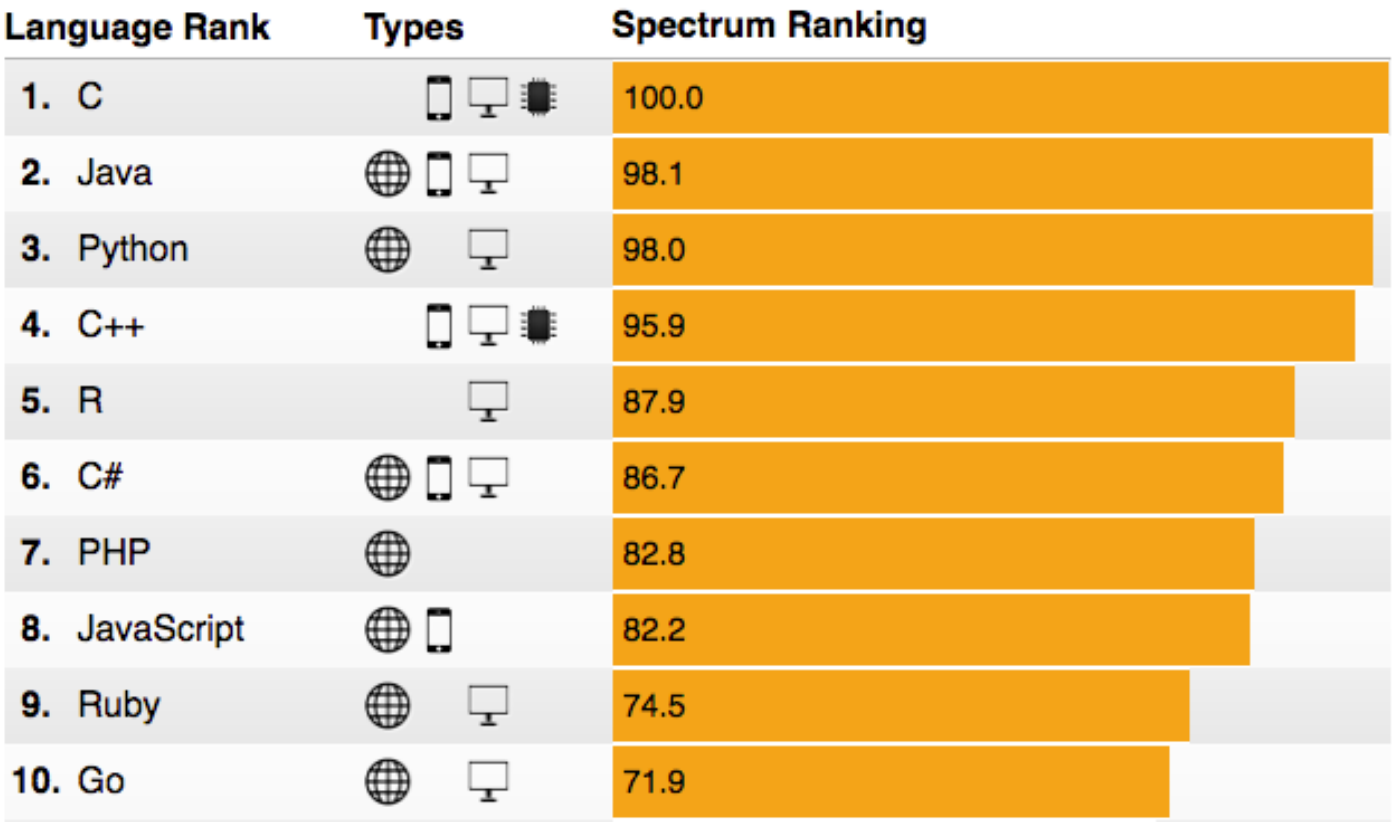
Source: IEEE Spectrum 20 July 2015

The 2017 top ten programming languages



Source: IEEE Spectrum 18 July 2017

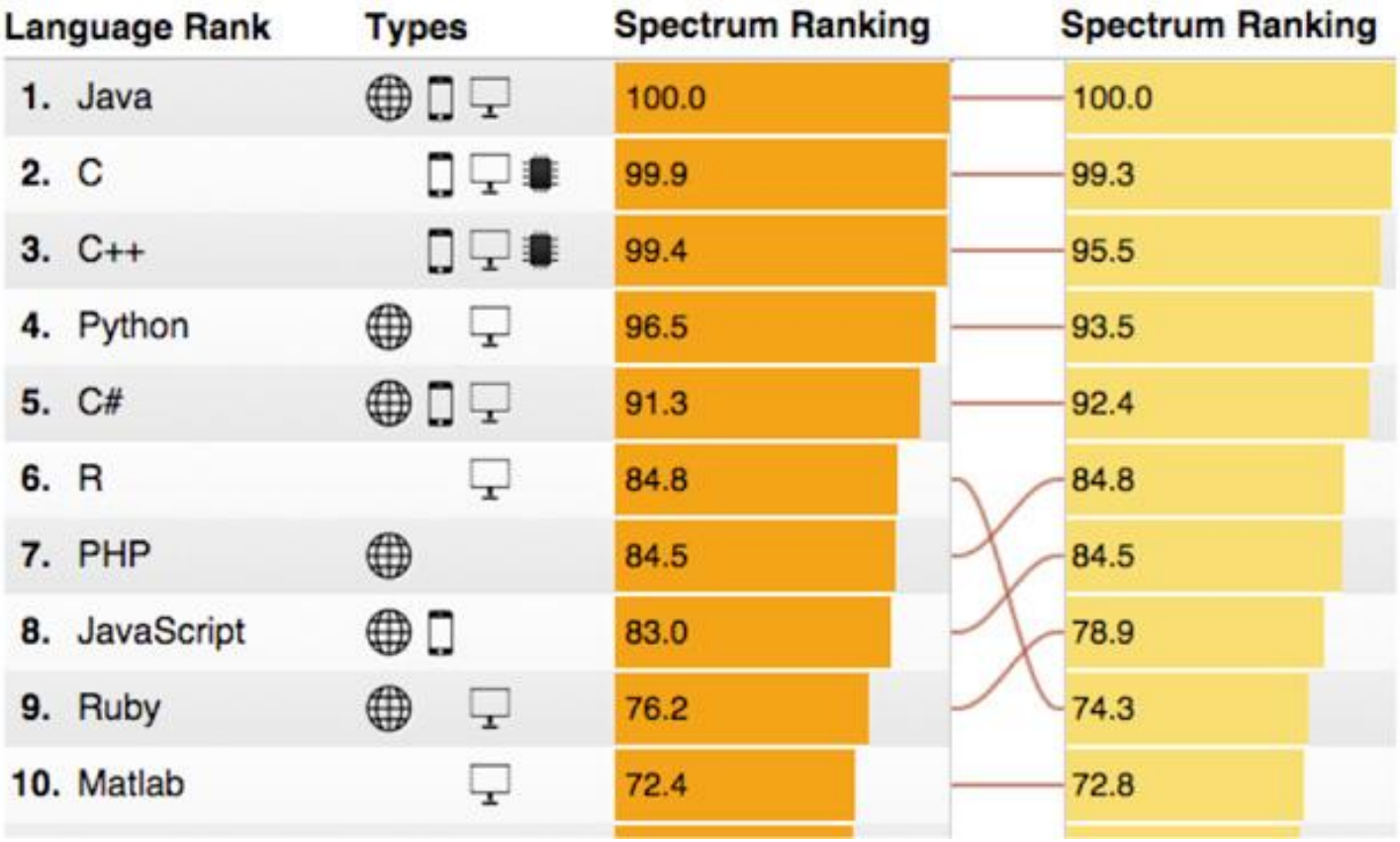
The 2016 top ten programming languages



Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		98.0
4. C++		95.9
5. R		87.9
6. C#		86.7
7. PHP		82.8
8. JavaScript		82.2
9. Ruby		74.5
10. Go		71.9

Source: IEEE Spectrum 26 July 2016

The 2015 top ten programming languages



Source: IEEE Spectrum 20 July 2015

Fortran

History

- *John Backus (1953) sought to write programs in conventional mathematical notation, and generate code comparable to good assembly programs.*
- No language design effort (made it up as they went along)
- Most effort spent on code generation and optimization
- FORTRAN I released April 1957; working by April 1958
- The current standard is FORTRAN 2003 (FORTRAN 2008 is work in progress)

Fortran …

Innovations

- Symbolic notation for subroutines and functions
- Assignments to variables of complex expressions
- DO loops
- Comments
- Input/output formats
- Machine-independence

Successes

- Easy to learn; high level
- Promoted by IBM; addressed large user base
- (scientific computing)

“Hello World” in FORTRAN

```
      PROGRAM HELLO
      DO 10, I=1,10
      PRINT *,'Hello World'
10    CONTINUE
      STOP
      END
```

All examples from the ACM "Hello World" project:

www2.latech.edu/~acm/HelloWorld.shtml

ALGOL 60

History

- *Committee of PL experts formed in 1955 to design universal, machine-independent, algorithmic language*
- First version (ALGOL 58) never implemented; criticisms led to ALGOL 60

Innovations

- BNF (Backus-Naur Form) introduced to define syntax (led to syntax-directed compilers)
- First block-structured language; variables with local scope
- Structured control statements
- Recursive procedures
- Variable size arrays

Successes

- Highly influenced design of other PLs but never displaced FORTRAN

“Hello World” in ALGOL

```
BEGIN
FILE F (KIND=REMOTE);
EBCDIC ARRAY E [0:11];
REPLACE E BY "HELLO WORLD!";
WHILE TRUE DO
   BEGIN
   WRITE (F, *, E);
   END;
END.
```

```
BEGIN
FILE F (KIND=REMOTE);
EBCDIC ARRAY E [0:11];
REPLACE E BY "HELLO WORLD!";
WRITE (F, *, E);
END.
```

```
BEGIN DISPLAY("HELLO WORLD!")
END.
```

```
BEGIN
FILE F (KIND=REMOTE);
WRITE (F, <"HELLO
WORLD!“>);
END.
```

COBOL

History

- *Designed by committee of US computer manufacturers*
- Targeted business applications
- Intended to be readable by managers (!)

Innovations

- Separate descriptions of environment, data, and processes

Successes

- Adopted as de facto standard by US DOD
- Stable standard for 25 years
- Still *the most widely used PL* for business applications (!)

“Hello World” in COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     HELLOWORLD.
DATE-WRITTEN.   02/05/96        21:04.
AUTHOR BRIAN COLLINS
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. RM-COBOL.
OBJECT-COMPUTER. RM-COBOL.
DATA DIVISION.
FILE SECTION.

PROCEDURE DIVISION.
     DISPLAY 'Hello World'.
STOP RUN.
```

http://www.tutorialspoint.com/cobol/

PL/1

History

- *Designed by committee of IBM and users (early 1960s)*
- Intended as (large) general-purpose language for broad classes of applications

Innovations

- Support for concurrency (but not synchronization)
- Exception-handling on conditions

Successes

- Achieved both run-time efficiency and flexibility (at expense of complexity)
- First “complete” general purpose language

Functional Languages

- treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- is a declarative programming paradigm, which means programming is done with expressions or declarations instead of statements

ISWIM (If you See What I Mean): Peter Landin (1966) — paper proposal

FP: John Backus (1978) — Turing award lecture

ML:

- initially designed as meta-language for theorem proving
- Hindley-Milner *type inference*
- "non-pure" functional language (with assignments/side effects)

Miranda, Haskell: "pure" functional languages with *"lazy evaluation"*

Prolog

History

- Originated at U. Marseilles (early 1970s), and compilers developed at Marseilles and Edinburgh (mid to late 1970s)

Innovations

- Theorem proving paradigm
- Programs as sets of clauses: facts, rules and questions
- Computation by “unification”

Successes

- Prototypical logic programming language
- Used in Japanese Fifth Generation Initiative

Object-Oriented Languages

History

- ***Simula*** was developed by Nygaard and Dahl (early 1960s) in Oslo as a language for simulation programming, by adding *classes and inheritance* to ALGOL 60

```
Begin
   while 1 = 1 do begin
      outtext ("Hello World!");
      outimage;
   end;
End;
```

- ***Smalltalk*** was developed by Xerox PARC (early 1970s) to drive graphic workstations

```
Transcript show:'Hello World';cr
```

Object-Oriented Languages

Innovations

- *Encapsulation* of data and operations (contrast ADTs)
- *Inheritance* to share behaviour and interfaces

Successes

- Smalltalk project pioneered OO user interfaces
- Large commercial impact since mid 1980s
- Countless new languages: C++, Objective C, Eiffel, Beta, Oberon, Self, Perl 5, Python, Java, Ada 95 ...

Interactive Languages

- Made possible by advent of time-sharing systems (early 1960s through mid 1970s).

BASIC

- Developed at Dartmouth College in mid 1960s
- Minimal; easy to learn
- Incorporated basic O/S commands (NEW, LIST, DELETE, RUN, SAVE)

```
10 print "Hello World!"
20 goto 10
```

Interactive Languages ...

APL

- Developed by Ken Iverson for concise description of numerical algorithms
- Large, non-standard alphabet (52 characters in addition to alphanumerics)
- Primitive objects are arrays (lists, tables or matrices)
- Operator-driven (power comes from composing array operators)
- No operator precedence (statements parsed right to left)

```
'HELLO WORLD'
```

Special-Purpose Languages

SNOBOL

- First successful string manipulation language
- Influenced design of text editors more than other PLs
- String operations: pattern-matching and substitution
- Arrays and associative arrays (tables)
- Variable-length strings

```
   OUTPUT = 'Hello World!'
END
```

...

Symbolic Languages ...

Lisp

- Performs computations on symbolic expressions
- Symbolic expressions are represented as *lists*
- Small set of constructor/selector operations to create and manipulate lists
- Recursive rather than iterative control
- *No distinction between data and programs*
- First PL to implement storage management by garbage collection
- Affinity with lambda calculus

```
(DEFUN HELLO-WORLD ()
       (PRINT (LIST 'HELLO 'WORLD)))
```

Scripting Languages

History

Countless “shell languages” and “command languages” for operating systems and configurable applications

- ***Unix shell*** (ca. 1971) developed as user shell and scripting tool

```
echo "Hello, World!"
```

- ***HyperTalk*** (1987) was developed at Apple to script HyperCard stacks

```
on OpenStack
    show message box
    put "Hello World!" into message box
end OpenStack
```

- ***TCL*** (1990) developed as embedding language and scripting language for X windows applications (via Tk)

```
puts "Hello World "
```

- ***Perl*** (~1990) became de facto web scripting language

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

Scripting Languages ...

Innovations

Pipes and filters (Unix shell)

Generalized embedding/command languages (TCL)

Successes

Unix Shell, awk, emacs, HyperTalk, AppleTalk, TCL, Python, Perl, VisualBasic ...


Murdoch
UNIVERSITY
Python
Mu
UNIV

Brief History of Python

- a high-level, general-purpose, interpreted, dynamic programming language.
  - with design philosophy emphasizes code readability
- Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles.

- Invented in the Netherlands, early 90s by Guido van Rossum
- Named after Monty Python
- Open sourced from the beginning
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning
- Increasingly popular – is awesome!

Brief History of Python

"Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered."

- Guido van Rossum

The Python Interpreter

Typical Python implementations offer both an interpreter and compiler

Interactive interface to Python:

On Unix…

% python

>>> 3+3

6

Python prompts with '>>>'.

To exit Python (not Idle):

- In Unix, type CONTROL-D
- In Windows, type CONTROL-Z + <Enter>
- Evaluate exit()

Murdoch
UNIVERSITY

Summary

Summary

- Overview
- Chronology
- A Selection of Languages

Murdoch

UNIVERSITY