Website:

http://nisl.cs.montana.edu/~pdonnelly/CSCI305/

Reading:

Chapter 1
A good programming language is a conceptual universe for thinking about programming.

A. Perlis
Programming Domains

The following user communities claim major developments in programming languages:

- Artificial Intelligence
- Computer Science Education
- Science and Engineering
- Information Systems
- Systems and Networks
- World Wide Web
function gcd(a, b)
    while b \neq 0
        t := b
        b := a \text{ mod } t
        a := t
    return a
Machine Code

What was wrong with using machine code?
Machine Code

What was wrong with using machine code?

- Poor readability
- Poor modifiability
- Expression coding was tedious
- Machine deficiencies—no indexing or floating point
### Plankalkül

**Designed by:** Konrad Zuse  
**Appeared in:** 1943 / 1972

**Features:**  
advanced data structures:  
- floating point, arrays, records  
never implemented

**Domains:**  
designed for engineering purposes

**Meaning:**  
“Plan Calculus”
Shortcode

Designed by: John Mauchly   Appeared in: 1949

Features:
- designed for BINAC computers
- statements represented mathematic expressions
- allowed for branching and calls to functions

- 50 times slower than machine code

Contributions: first higher-level language
ShortCode Example

Expressions were coded, left to right.

Example of operations:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>06 ABS</td>
</tr>
<tr>
<td>01</td>
<td>07 +</td>
</tr>
<tr>
<td>02</td>
<td>08 pause</td>
</tr>
<tr>
<td>03 =</td>
<td>09 (</td>
</tr>
<tr>
<td>04 /</td>
<td>etc ...</td>
</tr>
</tbody>
</table>

Example:

\[ a = \left( b + c \right) / b \times c \]

\[ X3 = \left( X1 + Y1 \right) / X1 \times Y1 \]

\[ X3 \ 03 \ 09 \ X1 \ 07 \ Y1 \ 02 \ 04 \ X1 \ Y1 \]
<table>
<thead>
<tr>
<th>Designed by:</th>
<th>John Backus</th>
<th>Appeared in:</th>
<th>1953</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td></td>
<td>Paradigm:</td>
<td>structured</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>assembly language, machine code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features:</td>
<td>pseudo ops for arithmetic and math functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>conditional and unconditional branching</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>auto-increment registers for array access</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slow!</td>
<td>interpreter took 310 words – 30% of the memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contributions:</td>
<td>first higher-level language for an IBM computer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Fortran

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>John Backus</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td></td>
</tr>
<tr>
<td>Appeared in:</td>
<td>1954-57</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>imperative</td>
</tr>
<tr>
<td>Extension:</td>
<td>.f</td>
</tr>
</tbody>
</table>

**Influenced by:** Speedcoding

**Features:**
- names could have up to six characters
- formatted I/O
- user-defined subprograms
- three-way selection statement

**Contributions:**
- code was very fast, quickly became widely used

**Domains:**
- scientific and engineering applications

**Acronym:** IBM Mathematical Formula Translating System
subroutine gcd_iter(value, u, v)
    integer, intent(out) :: value
    integer, intent(inout) :: u, v
    integer :: t

    do while( v /= 0 )
        t = u
        u = v
        v = mod(t, v)
    enddo
    value = abs(u)
end subroutine gcd_iter
<table>
<thead>
<tr>
<th>Designed by:</th>
<th>John McCarthy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT</td>
<td>Appeared in: 1958</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>functional</td>
</tr>
<tr>
<td>Extension:</td>
<td>.lisp</td>
</tr>
</tbody>
</table>

**Features:**
- processes data in lists
- symbolic computation
- only two data types: atoms and lists
- syntax is based on lambda calculus
- control via recursion and conditional expressions

**Domains:** Artificial intelligence

**Acronym:** LIS\textit{t} Processing language
Representation of \((A \ B \ C \ D)\) and \((A \ (B \ C) \ D \ (E \ (F \ G)) )\)
LISP \textit{gcd} Function

\begin{verbatim}
(defun gcd2 (a b)
  (do () ((zerop b) (abs a))
    (shiftf a b (mod a b)))))
\end{verbatim}
Cobol

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Grace Hopper, et. al</th>
<th>Appeared in:</th>
<th>1959</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm:</td>
<td>imperative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension:</td>
<td>.cbl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Influenced by:</td>
<td>FLOW-MATIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features:</td>
<td>data and code were completely separate, English names for arithmetic operators, long names, be easy to use records, nested selection statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contributions:</td>
<td>first macro facility in a high-level language first language required by DoD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domains:</td>
<td>widely used business applications language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acronym:</td>
<td>COmmon Business-Oriented Language</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IDENTIFICATION DIVISION.
PROGRAM-ID. GCD.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 A           PIC 9(10)   VALUE ZEROES.
01 B           PIC 9(10)   VALUE ZEROES.
01 TEMP        PIC 9(10)   VALUE ZEROES.

.
.
.

COBOL gcd Function (1 / 2)
PROCEDURE DIVISION.

Begin.

DISPLAY "Enter first number, max 10 digits."
ACCEPT A
DISPLAY "Enter second number, max 10 digits."
ACCEPT B

IF A < B
  MOVE B TO TEMP
  MOVE A TO B
  MOVE TEMP TO B
END-IF

PERFORM UNTIL B = 0
  MOVE A TO TEMP
  MOVE B TO A
  DIVIDE TEMP BY B GIVING TEMP REMAINDER B
END-PERFORM

DISPLAY "The gcd is " A
STOP RUN.
# ALGOL 60

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>John Backus et al.</th>
<th>Appeared in:</th>
<th>1960</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm:</td>
<td>imperative</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Influenced by:** ALGOL 58

**Features:**
- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Semicolon as a statement separator
- Subprogram recursion
- Stack-dynamic arrays

**Contributions:**
- Subsequent imperative languages are based on it
- Standard way to publish algorithms for over 20 years
ALGOL \texttt{gcd} Function

\textbf{PROC} \texttt{gcd} = (\texttt{INT} a, b) \texttt{INT}: ( \\
    \textbf{IF} a = 0 \textbf{THEN} \\
    \quad b \\
    \textbf{ELIF} b = 0 \textbf{THEN} \\
    \quad a \\
    \textbf{ELIF} a > b \textbf{THEN} \\
    \quad \texttt{gcd}(b, a \ \texttt{MOD} \ b) \\
    \textbf{ELSE} \\
    \quad \texttt{gcd}(a, b \ \texttt{MOD} \ a) \\
    \textbf{FI} \\
); \\
\textbf{test}: ( \\
    \texttt{INT} a = 33, b = 77; \\
    \texttt{printf}((\texttt{x"The gcd of"g} \ "and" \ g \ g \ "is" \ \texttt{gl$\$,a,b,\texttt{gcd}(a,b))}); \\
    \texttt{INT} c = 49865, d = 69811; \\
    \texttt{printf}((\texttt{x"The gcd of"g} \ "and" \ g \ g \ "is" \ \texttt{gl$\$,c,d,\texttt{gcd}(c,d))})
)
<table>
<thead>
<tr>
<th><strong>Designed by:</strong></th>
<th>IBM, SHARE</th>
<th><strong>Appeared in:</strong></th>
<th>1964</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paradigm:</strong></td>
<td>imperative</td>
<td><strong>Influenced by:</strong></td>
<td>ALGOL, COBOL, Fortran</td>
</tr>
<tr>
<td><strong>Features:</strong></td>
<td>designed in five months</td>
<td><strong>Domains:</strong></td>
<td>scientific, engineering, business</td>
</tr>
<tr>
<td></td>
<td>floating point, English-like syntax</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Contributions:</strong></td>
<td>first unit-level concurrency, first exception handling</td>
<td></td>
<td>first array cross sections</td>
</tr>
</tbody>
</table>
PL/I gcd Function

GCD: procedure (a, b) returns
    (fixed binary (31)) recursive;
    declare (a, b) fixed binary (31);

    if b = 0 then return (a);

    return (GCD (b, mod(a, b))) ;

end GCD;
# BASIC

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>John Kemeny</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Thomas Kurtz</td>
</tr>
<tr>
<td>Appeared in:</td>
<td>1964</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>procedural</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>ALGOL 60, FORTRAN II</td>
</tr>
<tr>
<td>Features:</td>
<td>easy to learn and use</td>
</tr>
<tr>
<td>Notes:</td>
<td>first widely used language with time sharing</td>
</tr>
<tr>
<td></td>
<td>current popular dialect: Visual BASIC</td>
</tr>
<tr>
<td>Acronym:</td>
<td>Beginner’s All-purpose Symbolic Instruction Code.</td>
</tr>
</tbody>
</table>

Patrick Donnelly (Montana State University)  Concepts of Programming Languages  Spring 2014  25 / 75
BASIC \texttt{gcd} Function

FUNCTION \texttt{gcd}(a\%, \ b\%)
    IF a > b THEN
        factor = a
    ELSE
        factor = b
    END IF
    FOR l = factor TO 1 STEP -1
        IF a \mod l = 0 AND b \mod l = 0 THEN
            gcd = l
        END IF
    NEXT l
    gcd = 1
END FUNCTION
<table>
<thead>
<tr>
<th><strong>APL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Designed by:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Appeared in:</strong></td>
</tr>
<tr>
<td><strong>Paradigm:</strong></td>
</tr>
<tr>
<td><strong>Extension:</strong></td>
</tr>
<tr>
<td><strong>Influenced by:</strong></td>
</tr>
<tr>
<td><strong>Features:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Domains:</strong></td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Acronym:</strong></td>
</tr>
</tbody>
</table>
APL $\text{gcd}$ Function

\[
\lfloor(\div \equiv 0 \cdot |x) / a \rfloor / x \leftarrow 49865 69811
\]

9973
SNOBOL

Designed by: David J. Farber  Ralph E. Griswold  Bell Laboratories
Appeared in: 1964  Paradigm: multi-paradigm

Influenced by:

Features: powerful operators for string pattern matching

Domains: text processing tasks

Acronym: StriNg Oriented and symBOlic Language
SNOBOL \texttt{gcd} Function

\begin{verbatim}
define('gcd(i,j)') :(gcd_end)
gcd ?eq(i,0) :s(freturn)
    ?eq(j,0) :s(freturn)

loop gcd = remdr(i,j)
    gcd = ?eq(gcd,0) j :s(return)
    i = j
    j = gcd :(loop)
gcd_end

output = gcd(1071,1029)
end
\end{verbatim}
### Simula

**Designed by:** Ole-Johan Dahl  
**Designed by:** Kristen Nygaard  
**Appeared in:** 1967  
**Paradigm:** object-oriented

**Influenced by:** ALGOL 60

**Features:** classes, objects, and inheritance

**Domains:** designed for system simulation

**Contributions:** coroutines - a kind of subprogram  
first object-oriented programming language  
influenced C++
Simula gcd Function

Begin

Integer Procedure GCD(M, N); Integer M, N;
Begin

While M<>N do
    If M<N then N := N - M else M := M - N;
    GCD := M
End of GCD;

Integer A, B;
OutText("Enter an integer number: ");
OutImage; A := InInt;
OutText("Enter an integer number: ");
OutImage; B := InInt;
OutText("Greatest Common Divisor of your numbers is ");
OutInt(GCD(A,B), 4); OutImage;
End of Program;
Pascal

Designed by: Niklaus Wirth

Appeared in: 1971

Paradigm: imperative

Extension: .pas

Influenced by: ALGOL

Features: small, simple

Domains: Education

Contributions: From mid-1970s until the late 1990s, it was the most widely used language for teaching programming
Pascal \texttt{gcd} Function

\begin{verbatim}
function \texttt{gcd\_iterative}(u, v: longint): longint;
  var
    t: longint;
  begin
    while v <> 0 do
      begin
        t := u;
        u := v;
        v := t mod v;
      end;
    gcd\_iterative := abs(u);
  end;
\end{verbatim}
<table>
<thead>
<tr>
<th><strong>Designed by:</strong></th>
<th>Dennis Richie</th>
<th><strong>Appeared in:</strong></th>
<th>1972</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bell Labs</strong></td>
<td></td>
<td><strong>Paradigm:</strong></td>
<td>imperative</td>
</tr>
<tr>
<td><strong>Extension:</strong></td>
<td></td>
<td><strong>.c, .h</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Influenced by:</strong></td>
<td>ALGOL, Assembly, PL/I, FORTRAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Features:</strong></td>
<td>powerful set of operators</td>
<td>poor type checking</td>
<td></td>
</tr>
<tr>
<td><strong>Domains:</strong></td>
<td>designed as a systems language</td>
<td>used in many application areas</td>
<td></td>
</tr>
<tr>
<td><strong>Contributions:</strong></td>
<td>syntax influence is pervasive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C \texttt{gcd} Function

```c
int
gcd_iter(int u, int v) {
    int t;
    while (v) {
        t = u;
        u = v;
        v = t \% v;
    }
    return u < 0 ? -u : u; /* abs(u) */
}
```
## Prolog

**Designed by:** Alain Colmerauer  
**Appeared in:** 1972  
**Paradigm:** logic  
**Extension:** .pl

**Influenced by:** PLANNER

**Features:** based on formal logic  
non-procedural

**Domains:** natural language processing,  
but few application areas

**Contributions:** comparatively inefficient
Prolog \texttt{gcd} Function

\begin{verbatim}
gcd(X, 0, X) :- !.
gcd(0, X, X) :- !.
gcd(X, Y, D) :- X > Y, !, Z is X mod Y, gcd(Y, Z, D).
gcd(X, Y, D) :- Z is Y mod X, gcd(X, Z, D).
\end{verbatim}
**Smalltalk**

**Designed by:** Alan Kay  
Adele Goldberg  
Xerox PARC

**Appeared in:** 1972

**Paradigm:** object-oriented

**Extension:** .st

**Influenced by:** Lisp, Simula

**Features:**
- graphical user interface design
- data abstraction
- inheritance
- dynamic binding

**Domains:**

**Contributions:** first full implementation of an object-oriented language
Smalltalk $gcd$ Function

|gcd_iter|

[ v > 0 ]
whileTrue: [ |t|
  t := u copy.
  u := v copy.
  v := t rem: v
].
  u abs
].

(gcd_iter value: 40902 value: 24140)
printNl.
<table>
<thead>
<tr>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Designed by:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Influenced by:</strong></td>
</tr>
<tr>
<td><strong>Features:</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Domains:</strong></td>
</tr>
</tbody>
</table>
Scheme \texttt{gcd} Function

\begin{verbatim}
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (modulo a b)))))
\end{verbatim}
### AWK

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Alfred Aho, et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeared in:</td>
<td>1977</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>scripting</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>C SNOBOL</td>
</tr>
</tbody>
</table>

**Features:**
- extensively uses strings, hashes and reg ex’s
- designed to support one-liner programs
- standard feature of Unix

- inspired Larry Wall to write Perl

**Domains:**
- data extraction and reporting tool

**Name:**
- from its authors Aho, Weinberger, and Kernighan
$\text{AWK gcd Function}$

```
$ awk 'func gcd(p,q)
    {return (q?gcd(q, (p\%q)):p)}
    {print gcd($1,$2)}'
12 16
4
```
# Matlab

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Cleve Moler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U.of New Mexico</td>
</tr>
<tr>
<td>Appeared in:</td>
<td>1978</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>multi-paradigm</td>
</tr>
<tr>
<td>Extension:</td>
<td>.m, .mat</td>
</tr>
</tbody>
</table>

Influenced by:

- matrix manipulations
- plotting of functions and data

Domains: applied mathematics, image processing
function [gcdValue] =
greatestcommondivisor(integer1, integer2)
gcdValue = \texttt{gcd}(integer1, integer2);
Ada

**Designed by:** Jean Ichbiah  
**MIL-STD-1815**  
** Appeared in:** 1980  
**Paradigm:** multi-paradigm  
**Extension:**

**Influenced by:** ALGOL, C++, Pascal

**Features:**
- generic program units
- packages - support for data abstraction
- elaborate exception handling

**Domains:** DoD

**Contributions**
- flexible libraries
- concurrency - through the tasking model
Ada gcd Function

with Ada.Text_Io; use Ada.Text_Io;
procedure Gcd_Test is
    function Gcd (A, B : Integer) return Integer is
        M : Integer := A;
        N : Integer := B;
        T : Integer;
    begin
        while N /= 0 loop
            T := M;
            M := N;
            N := T mod N;
        end loop;
        return M;
    end Gcd;
begin
    Put_Line("GCD of 100,5 is"&Integer'Image(Gcd(100, 5)));
    Put_Line("GCD of 5,100 is"&Integer'Image(Gcd(5, 100)));
    Put_Line("GCD of 7,23 is"&Integer'Image(Gcd(7, 23)));
end Gcd_Test;
| **C++** |
|------------------|------------------|
| **Designed by:**  | Bjarne Stroustrup  |
| **Appeared in:**  | 1983             |
| **Paradigm:**     | multi-paradigm   |
| **Extension:**    | .h, .cpp         |
| **Influenced by:**| Ada, ALGOL, C, ML |
| **Features:**     | large and complex language supports both procedural and OO programming efficient compiler to native code |
| **Domains:**      | systems software, application software, embedded software |
| **Contributions:**| rapidly grew in popularity |
```cpp
int gcd_iter(int u, int v) {
    int t;
    while (v) {
        t = u;
        u = v;
        v = t % v;
    }
    return u < 0 ? -u : u; /* abs(u) */
}
```
Objective C

Designed by: Brad Cox
Tom Love
Apple

Appeared in: 1983
Paradigm: object-oriented
Extension: .h,.m

Influenced by: C, Smalltalk

Features:
C plus support for OOP based on Smalltalk
uses Smalltalk’s method calling syntax
support for reflective features
superset of C

Domains:
used by Apple for systems programs
# Perl

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Larry Wall</th>
<th>Appeared in:</th>
<th>1987</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm:</td>
<td>multi-paradigm</td>
<td>Extension:</td>
<td>.pl</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>AWK, C++, Lisp, Pascal, Smalltalk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features:</td>
<td>3 distinctive namespaces, denoted in var’s name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Regular expression engine</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Variables are statically typed but implicitly declared</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domains:</td>
<td>CGI, graphics programming, system administration, network programming, finance, bioinformatics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backronym:</td>
<td>Practical Extraction and Reporting Language</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sub gcd_iter($$) {
    my ($u, $v) = @_;  
    while ($v) {
        ($u, $v) = ($v, $u % $v);
    }
    return abs($u);
}
<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Simon Jones et. al</th>
<th>Appeared in:</th>
<th>1990</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm:</td>
<td>functional</td>
<td>Extension:</td>
<td>.hs</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>Lisp, ML Scheme</td>
<td>Features:</td>
<td>primary control construct is the function non-strict semantics and strong static typing</td>
</tr>
</tbody>
</table>
Haskell \textit{gcd} Function

\texttt{gcd} :: (Integral \textit{a}) \rightarrow \textit{a} \rightarrow \textit{a} \\
\texttt{gcd} 0 0 = \texttt{error} "Prelude.gcd: gcd 0 0 is undefined" \\
\texttt{gcd} \texttt{x y} = \texttt{gcd'} (\texttt{abs} \texttt{x}) (\texttt{abs} \texttt{y}) \texttt{where} \\
\hspace{1cm} \texttt{gcd'} \texttt{a 0} = \texttt{a} \\
\hspace{1cm} \texttt{gcd'} \texttt{a b} = \texttt{gcd'} \texttt{b (a \texttt{ rem} b)}
## Python

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Guido van Rossum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeared in:</td>
<td>1991</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>multi-paradigm</td>
</tr>
<tr>
<td>Extension:</td>
<td>.py</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>ALGOL, C, C++, Haskell, Java, Lisp, Perl</td>
</tr>
<tr>
<td>Features:</td>
<td>OO interpreted scripting language</td>
</tr>
<tr>
<td></td>
<td>type checked but dynamically typed</td>
</tr>
<tr>
<td></td>
<td>supports lists, tuples, and hashes</td>
</tr>
<tr>
<td>Domains:</td>
<td>CGI programming, form processing</td>
</tr>
</tbody>
</table>
def gcd_iter(u, v):
    while v:
        u, v = v, u % v
    return abs(u)
Lua

Designed by: R. Ierusalimschy
W. Celes
L.H. de Figueiredo

Appear in: 1993

Paradigm: multi-paradigm

Influenced by: C++, Modula, Scheme

Features: OO interpreted scripting language

Type checked but dynamically typed

Single data structure – table

Easily extendable

Domains: CGI programming, form processing

Means: “moon” in Portuguese
function gcd(a, b)
    if b ~= 0 then
        return gcd(b, a % b)
    else
        return math.abs(a)
    end
end

demo(a, b)
    print("GCD of "..a.." and "..b.." is "..gcd(a, b))
end
<table>
<thead>
<tr>
<th><strong>JavaScript</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed by:</td>
</tr>
<tr>
<td>Netscape</td>
</tr>
<tr>
<td>Paradigm: multi-paradigm</td>
</tr>
<tr>
<td>Influenced by: C, Java, Perl, Python, Scheme</td>
</tr>
<tr>
<td>Features: client-side HTML-embedded scripting language purely interpreted</td>
</tr>
<tr>
<td>Domains: dynamic HTML documents</td>
</tr>
</tbody>
</table>
function gcd(a, b) {
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    if (b > a) {var temp = a; a = b; b = temp;}
    while (true) {
        a %= b;
        if (a == 0) return b;
        b %= a;
        if (b == 0) return a;
    }
}
| **PHP** |
|---|---|---|---|
| **Designed by:** | Rasmus Lerdorf | **Appeared in:** | 1995 |
| **Paradigm:** | imperative, OO | **Extension:** | .php |
| **Influenced by:** | C, C++, Java, Perl |
| **Features:** | server-side scripting language purely interpreted |
| **Domains:** | form processing and database access |
| **Acronym:** | **PHP:** Hypertext Preprocessor |
function gcdIter($n, $m) {
    while(true) {
        if($n == $m) {
            return $m;
        } else {
            if($n > $m) {
                $n -= $m;
            } else {
                $m -= $n;
            }
        }
    }
}

PHP gcd Function
# Ruby

**Designed by:** Yukihiro Matsumoto  
**Appeared in:** 1995  
**Paradigm:** multi-paradigm  
**Extension:** .rb  

**Influenced by:** Ada, C++, Lisp, Perl, Python, Smalltalk

**Features:**  
- Pure object-oriented scripting language  
- Purely interpreted  
- Operators are implemented as methods
Ruby \texttt{gcd} Function

```ruby
def gcd(u, v)
    u, v = u.abs, v.abs
    while v > 0
        u, v = v, u \% v
    end
    u
end
```

### Java

**Designed by:** James Gosling  
**Appeared in:** 1995  
**Sun Microsystems**  
**Paradigm:** multi-paradigm  
**Extension:** .java, .class

**Influenced by:** Ada, C++, C#, Pascal, Smalltalk

**Features:**  
supports only OOP references, but not pointers  
support for applets  
supports concurrency  
Java Virtual Machine concept  
libraries for applets, GUIs, database access  
widely used for Web programming
public static long gcd(long a, long b) {
    long factor = Math.max(a, b);
    for (long loop = factor; loop > 1; loop--) {
        if (a % loop == 0 && b % loop == 0) {
            return loop;
        }
    }
    return 1;
}
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm:</td>
<td>functional, OO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Influenced by:</td>
<td>Standard ML</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features:</td>
<td>large standard library</td>
<td>robust object-oriented programming constructs</td>
<td>static type system</td>
</tr>
<tr>
<td>Name:</td>
<td>Objective Caml</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Caml gcd Function

let rec gcd a b =
    if a = 0 then b
  else if b = 0 then a
  else if a > b then gcd b (a mod b)
  else gcd a (b mod a)
### C#

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appeared in:</td>
<td>2000</td>
</tr>
<tr>
<td>Paradigm:</td>
<td>multi-paradigm</td>
</tr>
<tr>
<td>Extension:</td>
<td>.cs</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>C++, Java, Pascal</td>
</tr>
<tr>
<td>Features:</td>
<td>includes pointers, delegates, properties, enumeration types, limited kind of dynamic typing, anonymous types</td>
</tr>
<tr>
<td>Domains:</td>
<td>.NET</td>
</tr>
<tr>
<td>Contributions:</td>
<td>is evolving rapidly</td>
</tr>
</tbody>
</table>
private static int gcd(int a, int b)
{
    int t;
    // Ensure B > A
    if (a > b)
    {
        t = b;
        b = a;
        a = t;
    }
    // Find
    while (b != 0)
    {
        t = a % b;
        a = b;
        b = t;
    }
    return a;
}
Go

<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Robert Griesemer</th>
<th>Appeared in:</th>
<th>2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>et al.</td>
<td>Google</td>
<td>Paradigm:</td>
<td>imperative</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extension:</td>
<td>.go</td>
</tr>
<tr>
<td>Influenced by:</td>
<td>C, Modula, Pascal, Python</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Features:</th>
</tr>
</thead>
<tbody>
<tr>
<td>loosely based on C, but also quite different</td>
</tr>
<tr>
<td>does not support traditional OOP</td>
</tr>
<tr>
<td>goroutines, small lightweight threads</td>
</tr>
<tr>
<td>visibility according to capitalization</td>
</tr>
<tr>
<td>efficient, latency-free garbage collection</td>
</tr>
<tr>
<td>line-ending semicolons are optional</td>
</tr>
<tr>
<td>designed for exceptionally fast compiling times</td>
</tr>
</tbody>
</table>
package main

import "fmt"

func gcd(x, y int) int {
    for y != 0 {
        x, y = y, x % y
    }
    return x
}

func main() {
    fmt.Println(gcd(33, 77))
    fmt.Println(gcd(49865, 69811))
}
<table>
<thead>
<tr>
<th>Designed by:</th>
<th>Graydon Hoare et al.</th>
<th>Appeared in:</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Influenced by:</td>
<td>Mozilla Research</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension:</td>
<td>.rs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paradigm:</td>
<td>multi-paradigm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features:</td>
<td>designed for large client and server programs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>syntax similar to subset of C and C++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory safe (no null or dangling pointers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>type system supports ‘traits’, inspired by Haskell</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>type inference</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>supports concurrency</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>performance of safe code is slower than C++</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sponsored by Mozilla and Samsung</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>open community project</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Rust $gcd$ Function

```rust
fn gcd(mut m: int, mut n: int) -> int {
    while m != 0 {
        let temp = m;
        m = n % temp;
        n = temp;
    }
    n.abs()
}

fn gcd(m: int, n: int) -> int {
    if m == 0
    { n.abs() }
    else
    { gcd(n % m, m) }
}
```