FizzBuzz

From Rosetta Code

Write a program that prints the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz". [1] (http://weblog.raganwald.com/2007/01/dont-overthink-fizzbuzz.html)

FizzBuzz was presented as the lowest level of comprehension required to illustrate adequacy. [2] (http://www.codinghorror.com/blog/archives/000804.html)

FizzBuzz
You are encouraged to solve this task according to the task description, using any language you may know.

Contents

360 Assembly

```
FIZZBUZZ CSECT                       A SECTION OF CODE STARTS HERE, LABEL IT FIZZBUZZ
**********HOUSE KEEPING AREA**********************
         USING *,12                  FOR THIS PROGRAM WE ARE GOING TO USE REGISTER 12
         STM   14,12,12(13)          SAVE REGISTERS 14,15, AND 0-12 IN CALLER'S SAVE AREA
         LR    12,15                 PUT OUR ENTRY ADDRESS(IN R15) INTO OUR BASE REGISTER
         LA    15,SAVE               POINT R15 AT THE *OUR* SAVE AREA (DEFINED AT THE END)
         ST    15,8(13)              SET FORWARD CHAIN
         ST    13,4(15)              SET BACKWARD CHAIN
         LR    13,15                 SET R13 TO THE ADDRESS OF OUT NEW SAVE AREA
**********MAIN*PROGRAM*****************************
         LA    10,LOOP               PUT THE LOOP START ADDRESS IN R10
         LA    8,100                 PUT THE NUMBER OF ITERATIONS IN R8
         LA    5,=F'1'               INITIALIZE BINARY COUNTER TO ONE
LOOP     EQU   *                     LABEL THE LOOP START
         A     5,=F'1'               ADD TO BINARY LOOP COUNTER
         AP    NUM,=PL1'1'           ADD TO PACKED LOOP COUNTER
         B     CHK15                 CHECK IF COUNTER IS % 12
LCHK15   EQU   *                     IF NOT, COME BACK
         B     CHK3                  CHECK IF COUNTER IS % 3
LCHK3    EQU   *                     IF NOT, COME BACK
         B     CHK5                  CHECK IF COUNTER IS % 4
LCHK5    EQU   *                     IF NOT, COME BACK
         MVC   EOUT,EMSK             PREPARE TO PKD->EBCDIC
         EDMK  EOUT,NUM              PKD->EBCDIC
ENLOOP   EQU   *                     IF A TEST WAS POSITIVE RETURN HERE
         WTO   MF=(E,WTOSTART)       PRINT RESULT OF LOOP
         BCTR  8,10                  START OVER
**********HOUSE KEEPING AREA**********************
         L     13,4(13)              RESTORE ADDRESS TO CALLER'S SAVE AREA
         LM    14,12,12(13)          RESTORE REGISTERS AS ON ENTRY
         XR    15,15                 XOR R15 SO IT IS ALL 0 (R15 CREATES THE PROGRAM RETURN CODE
         BR    14                    RETURN WHERE YOU CAME FROM
**********SUBROUTINE AREA**************************
*////////CHK3//////////////////////////////////*
CHK3     EQU   *                     LABEL ENTRY POINT
         LR    6,5                   LOAD R6 WITH R5(THE BINARY LOOP INDEX)
         A     6,=F'1'               ADD ONE TO R6
         SRDA  6,32                  SHIFT RD VAL 32 BITS RIGHT(TO R7)
         D     6,=F'3'               DIVIDE BY 3
         C     6,=F'0'               IS REMAINDER 0?
         BE    DIV3                  IF SO GOTO DIV3 ROUTINE
         B     LCHK3                 IF NOT GO BACK TO LOOP
*////////CHK15/////////////////////////////////*
CHK15    EQU   *                     LABEL ENTRY POINT
         LR    6,5                   LOAD R6 WITH R5(THE BINARY LOOP INDEX)
         A     6,=F'1'               ADD ONE TO R6
         SRDA  6,32                  SHIFT RD VAL 32 BITS RIGHT(TO R7)
         D     6,=F'15'              DIVIDE BY 15
         C     6,=F'0'               IS REMAINDER 0?
```

```
         BE    DIV15                IF SO GOTO DIV15 ROUTINE
         B     LCHK15               IF NOT GO BACK TO LOOP
*////////CHK5//////////////////////////////////*
CHK5     EQU   *                    LABEL ENTRY POINT
         LR    6,5                  LOAD R6 WITH R5(THE BINARY LOOP INDEX)
         A     6,=F'1'              ADD ONE TO R6
         SRDA  6,32                 SHIFT RD VAL 32 BITS RIGHT(TO R7)
         D     6,=F'5'              DIVIDE BY 5
         C     6,=F'0'              IS REMAINDER 0?
         BE    DIV5                 IF SO GOTO DIV5 ROUTINE
         B     LCHK5                IF NOT GO BACK TO LOOP
*////////////////////////////////////////////*
DIV3     EQU   *                    LABEL ENRTY POINT
         MVC   EOUT,FIZZ            SAY FIZZ
         B     ENLOOP               RETURN TO LOOP
*////////////////////////////////////////////*
DIV5     EQU   *                    LABEL ENTRY POINT
         MVC   EOUT,BUZZ            SAY BUZZ
         B     ENLOOP               RETURN TO LOOP
*////////////////////////////////////////////*
DIV15    EQU   *                    LABEL ENTRY POINT
         MVC   EOUT,FIZZBUZ         SAY FIZZBUZZ
         B     ENLOOP               RETURN TO LOOP
**********VARIABLE STORAGE**********************
FIZZBUZ  DC    CL10'FIZZBUZZ!'      CREATE A STRING IN MEMORY, LABEL THE ADDRESS FIZZBUZ
FIZZ     DC    CL10'FIZZ!'          CREATE A STRING IN MEMORY, LABEL THE ADDRESS FIZZ
BUZZ     DC    CL10'BUZZ!'          CREATE A STRING IN MEMORY, LABEL THE ADDRESS BUZZ
NUM      DC    PL3'0'               CREATE A DECIMAL IN MEMORY, MAKE IT ZERO, LABEL IT NUM
TEMP     DS    D                    RESERVE A DOUBLE WORD (8 BYTES) IN MEMORY, LABEL IT TEMP
EMSK     DC    X'402020202020'      CREATE A HEX ARRAY IN MEMORY, LABEL IT EMSK
WTOSTART DC    Y(WTOEND-*,0)        LABEL THIS WTOSTART, DEFINE A CONSTANT ADDRESS EQUAL TO
*                                   "WTOEND" MINUS HERE(*)
EOUT     DS    CL10                 RESERVE SPACE FOR 10 CHARACTERS, LABEL THIS EOUT
WTOEND   EQU   *                    THE MEMORY ADDRESS LOCATED HERE IS LABELED WTOEND
**********HOUSE KEEPING AREA**********************
SAVE     DS    18F
         END   HELLO
```

6502 Assembly

The modulus operation is rather expensive on the 6502, so a simple counter solution was chosen.

```
        .lf  fzbz6502.lst
        .cr  6502
        .tf  fzbz6502.obj,ap1
;------------------------------------------------------
; FizzBuzz for the 6502 by barrym95838 2013.04.04
; Thanks to sbprojects.com for a very nice assembler!
; The target for this assembly is an Apple II with
;   mixed-case output capabilities and Applesoft
;   BASIC in ROM (or language card)
; Tested and verified on AppleWin 1.20.0.0
;------------------------------------------------------
; Constant Section
;
FizzCt   =   3          ;Fizz Counter (must be < 255)
BuzzCt   =   5          ;Buzz Counter (must be < 255)
Lower    =   1          ;Loop start value (must be 1)
Upper    =   100        ;Loop end value (must be < 255)
CharOut  =   $fded      ;Specific to the Apple II
IntOut   =   $ed24      ;Specific to ROM Applesoft
;======================================================
        .or  $0f00
;------------------------------------------------------
; The main program
;
main    ldx  #Lower     ;init LoopCt
        lda  #FizzCt
```

```
        sta  Fizz       ;init FizzCt
        lda  #BuzzCt
        sta  Buzz       ;init BuzzCt
next    ldy  #0         ;reset string pointer (y)
        dec  Fizz       ;LoopCt mod FizzCt == 0?
        bne  noFizz     ;  yes:
        lda  #FizzCt
        sta  Fizz       ;    restore FizzCt
        ldy  #sFizz-str ;    point y to "Fizz"
        jsr  puts       ;    output "Fizz"
noFizz  dec  Buzz       ;LoopCt mod BuzzCt == 0?
        bne  noBuzz     ;  yes:
        lda  #BuzzCt
        sta  Buzz       ;    restore BuzzCt
        ldy  #sBuzz-str ;    point y to "Buzz"
        jsr  puts       ;    output "Buzz"
noBuzz  dey             ;any output yet this cycle?
        bpl  noInt      ;  no:
        txa             ;    save LoopCt
        pha
        lda  #0         ;    set up regs for IntOut
        jsr  IntOut     ;    output itoa(LoopCt)
        pla
        tax             ;    restore LoopCt
noInt   ldy  #sNL-str
        jsr  puts       ;output "\n"
        inx             ;increment LoopCt
        cpx  #Upper+1   ;LoopCt >= Upper+1?
        bcc  next       ;  no:  loop back
        rts             ;  yes:  end main
;-------------------------------------------------------
; Output zero-terminated string @ (str+y)
;   (Entry point is puts, not outch)
;
outch   jsr  CharOut    ;output string char
        iny             ;advance string ptr
puts    lda  str,y      ;get a string char
        bne  outch      ;output and loop if non-zero
        rts             ;return
;-------------------------------------------------------
; String literals (in '+128' ascii, Apple II style)
;
str:    ;               string base offset
sFizz   .az     -"Fizz"
sBuzz   .az     -"Buzz"
sNL     .az     -#13
;-------------------------------------------------------
; Variable Section
;
Fizz    .da     #0
Buzz    .da     #0
;-------------------------------------------------------
        .en
```

ACL2

```
(defun fizzbuzz-r (i)
   (declare (xargs :measure (nfix (- 100 i))))
   (prog2$
    (cond ((= (mod i 15) 0) (cw "FizzBuzz~%"))
          ((= (mod i 5) 0) (cw "Buzz~%"))
          ((= (mod i 3) 0) (cw "Fizz~%"))
          (t (cw "~x0~%" i)))
    (if (zp (- 100 i))
        nil
        (fizzbuzz-r (1+ i)))))

(defun fizzbuzz () (fizzbuzz-r 1))
```

ActionScript

The ActionScript solution works just like the JavaScript solution (they share the ECMAScript specification). The difference is that ActionScript has the *trace* command to write out to a console.

```
for (var i:int = 1; i <= 100; i++) {
  if (i % 15 == 0)
    trace('FizzBuzz');
  else if (i % 5 == 0)
    trace('Buzz');
  else if (i % 3 == 0)
    trace('Fizz');
  else
    trace(i);
}
```

AutoIt

```
For $i = 1 To 100
        If Mod($i, 15) = 0 Then
                MsgBox(0, "FizzBuzz", "FizzBuzz")
        ElseIf Mod($i, 5) = 0 Then
                MsgBox(0, "FizzBuzz", "Buzz")
        ElseIf Mod($i, 3) = 0 Then
                MsgBox(0, "FizzBuzz", "Fizz")
        Else
                MsgBox(0, "FizzBuzz", $i)
        EndIf
Next
```

8086 Assembly

Assembly programs that output a number on the screen are programmable in two ways: calculating the number in binary to convert it next in ASCII for output, or keeping the number in Binary Coded Decimal (BCD) notation to speed up the output to the screen, because no binary to decimal conversion needs to be applied. The first approach is the most useful because the binary number is immediately recognizable to the computer, but, in a problem where the calculations are very few and simple and the final result is mainly text on the screen, using binary numbers would speed up calculations, but will greatly slow down the output.

The BCD used is based on the ASCII text encoding: zero is the hexadecimal byte 30, and nine is the hexadecimal byte 39. The BCD number is kept in the DX register, the most significant digit in DH and the less significant digit in DL. See the comments for further explaining of the program's structure, wich is meant for speed and compactness rather than modularity: there are no subroutines reusable in another program without being edited.

This program is 102 bytes big when assembled. The program is written to be run in an IBM PC because the 8086 processor alone does not provide circuitry for any kind of direct screen output. At least, I should point out that this program is a little bugged: the biggest number representable with the BCD system chosen is 99, but the last number displayed is 100, wich would be written as :0 because the program does provide overflow detecting only for the units, not for tens (39 hex + 1 is 3A, that is the colon symbol in ASCII). However, this bug is hidden by the fact that the number 100 is a multiple of five, so the number is never displayed, because it is replaced by the string "buzz".

```
                    ; Init the registers
mov dx,03030h       ; For easier printing, the number is
                    ;kept in Binary Coded Decimal, in
----

                    ;the DX register.
mov ah,0Eh          ; 0Eh is the IBM PC interrupt 10h
                    ;function that does write text on
                    ;the screen in teletype mode.
mov bl,100d         ; BL is the counter (100 numbers).
xor cx,cx           ; CX is a counter that will be used
                    ;for screen printing.
xor bh,bh           ; BH is the counter for counting
                    ;multiples of three.

writeloop:          ; Increment the BCD number in DX.
inc dl              ; Increment the low digit
cmp dl,3Ah          ; If it does not overflow nine,
jnz writeloop1      ;continue with the program,
mov dl,30h          ;otherwise reset it to zero and
inc dh              ;increment the high digit
writeloop1:
inc bh              ; Increment the BH counter.
cmp bh,03h          ; If it reached three, we did
                    ;increment the number three times
                    ;from the last time the number was
                    ;a multiple of three, so the number
                    ;is now a multiple of three now,
jz writefizz        ;then we need to write "fizz" on the
                    ;screen.
cmp dl,30h          ; The number isn't a multiple of
jz writebuzz        ;three, so we check if it's a
cmp dl,35h          ;multiple of five. If it is, we
jz writebuzz        ;need to write "buzz". The program
                    ;checks if the last digit is zero or
                    ;five.
mov al,dh           ; If we're here, there's no need to
int 10h             ;write neither "fizz" nor "buzz", so
mov al,dl           ;the program writes the BCD number
int 10h             ;in DX
writespace:
mov al,020h         ;and a white space.
int 10h
dec bl              ; Loop if we didn't process 100
jnz writeloop       ;numbers.

programend:         ; When we did reach 100 numbers,
cli                 ;the program flow falls here, where
hlt                 ;interrupts are cleared and the
jmp programend      ;program is stopped.

writefizz:          ; There's need to write "fizz":
mov si,offset fizz  ; SI points to the "fizz" string,
call write          ;that is written on the screen.
xor bh,bh           ; BH, the counter for computing the
                    ;multiples of three, is cleared.
cmp dl,30h          ; We did write "fizz", but, if the
jz writebuzz        ;number is a multiple of five, we
cmp dl,35h          ;could need to write "buzz" also:
jnz writespace      ;check if the number is multiple of
                    ;five. If not, write a space and
                    ;return to the main loop.
writebuzz:          ; (The above code falls here if
                    ;the last digit is five, otherwise
                    ;it jumps)
mov si,offset buzz  ;SI points to the "buzz" string,
call write          ;that is written on the screen.
jmp writespace      ; Write a space to return to the main
                    ;loop.

write:              ; Write subroutine:
mov cl,04h          ; Set CX to the lenght of the string:
                    ;both strings are 4 bytes long.
```

```
write1:
mov al,[si]          ; Load the character to write in AL.
inc si               ; Increment the counter SI.
int 10h              ; Call interrupt 10h, function 0Eh to
                     ;write the character and advance the
                     ;text cursor (teletype mode)
loop write1          ; Decrement CX: if CX is not zero, do
ret                  ;loop, otherwise return from
                     ;subroutine.

fizz:                ;The "fizz" string.
db "fizz"

buzz:                ;The "buzz" string.
db "buzz"
```

Ada

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Fizzbuzz is
begin
   for I in 1..100 loop
      if I mod 15 = 0 then
         Put_Line("FizzBuzz");
      elsif I mod 5 = 0 then
         Put_Line("Buzz");
      elsif I mod 3 = 0 then
         Put_Line("Fizz");
      else
         Put_Line(Integer'Image(I));
      end if;
   end loop;
end Fizzbuzz;
```

ALGOL 68

```
main:(
  FOR i TO 100 DO
    printf(($gl$,
      IF i %* 15 = 0 THEN
        "FizzBuzz"
      ELIF i %* 3 = 0 THEN
        "Fizz"
      ELIF i %* 5 = 0 THEN
        "Buzz"
      ELSE
        i
      FI
    ))
  OD
)
```

or simply:

```
FOR i TO 100 DO print(((i%*15=0|"FizzBuzz"|:i%*3=0|"Fizz"|:i%*5=0|"Buzz"|i),new line)) OD
```

APL

```
⍞IO←0
(L,'Fizz' 'Buzz' 'FizzBuzz')[¯1+(L×W=0)+W←(100×~0=W)+W←⊃+/1 2×0=3 5|⊂L←1+⍳100]
```

AppleScript

```
property outputText: ""
repeat with i from 1 to 100
  if i mod 15 = 0 then
    set outputText to outputText & "FizzBuzz"
  else if i mod 3 = 0 then
    set outputText to outputText & "Fizz"
  else if i mod 5 = 0 then
    set outputText to outputText & "Buzz"
  else
    set outputText to outputText & i
  end if
  set outputText to outputText & linefeed
end repeat
outputText
```

Arbre

```
fizzbuzz():
  for x in [1..100]
    if x%5==0 and x%3==0
      return "FizzBuzz"
    else
      if x%3==0
        return "Fizz"
      else
        if x%5==0
          return "Buzz"
        else
           return x

main():
  fizzbuzz() -> io
```

AutoHotkey

Search autohotkey.com: [3] (http://www.google.com/search?q=site:www.autohotkey.com+FizzBuzz)

```
Loop, 100
{
  If (Mod(A_Index, 15) = 0)
    output .= "FizzBuzz`n"
  Else If (Mod(A_Index, 3) = 0)
    output .= "Fizz`n"
  Else If (Mod(A_Index, 5) = 0)
    output .= "Buzz`n"
  Else
    output .= A_Index "`n"
}
FileDelete, output.txt
FileAppend, %output%, output.txt
Run, cmd /k type output.txt
```

A short example with cascading ternary operators and graphical output. Press Esc to close the window.

```
Gui, Add, Edit, r20
Gui,Show
Loop, 100
  Send, % (!Mod(A_Index, 15) ? "FizzBuzz" : !Mod(A_Index, 3) ? "Fizz" : !Mod(A_Index, 5) ? "Buzz
Return
Esc::
ExitApp
```

AWK

```
BEGIN {
   for (NUM=1; NUM<=100; NUM++)
       if (NUM % 15 == 0)
           {print "FizzBuzz"}
       else if (NUM % 3 == 0)
           {print "Fizz"}
       else if (NUM % 5 == 0)
           {print "Buzz"}
       else
           {print NUM}
}
```

```
echo {1..100} | awk '
BEGIN{RS=" "}
$1 % 15 == 0 {print "FizzBuzz"}
$1 % 5 == 0 {print "Buzz"}
$1 % 3 == 0 {print "Fizz"}
{print}
'
```

```
seq 100 | awk '$0=NR%15?NR%5?NR%3?$0:"Fizz":"Buzz":"FizzBuzz"'
```

Babel

```
main:
    { { iter 1 + dup

        15 %
            { "FizzBuzz" <<
                zap }
            { dup
            3 %
                { "Fizz" <<
                    zap }
                { dup
                5 %
                    { "Buzz" <<
                        zap}
                    { %d << }
                if }
            if }
        if

        "\n" << }

    100 times }
```

bash

Any bash hacker would do this as a one liner at the shell, so...

```
for n in {1..100}; do ([ $((n%15)) -eq 0 ] && echo 'FizzBuzz') || ([ $((n%5)) -eq 0 ] && echo 'B
```

For the sake of readability...

```
for n in {1..100}; do
  ([ $((n%15)) -eq 0 ] && echo 'FizzBuzz') ||
  ([ $((n%5)) -eq 0 ] && echo 'Buzz') ||
  ([ $((n%3)) -eq 0 ] && echo 'Fizz') ||
  echo $n;
done
```

Here's a very concise approach, with only 75 characters total. Unfortunately it relies on aspects of Bash which are rarely used.

```
for i in {1..100};do((i%3))&&x=||x=Fizz;((i%5))||x+=Buzz;echo ${x:-$i};done
```

Here's the concise approach again, this time separated into multiple lines.

```
# FizzBuzz in Bash.  A concise version, but with verbose comments.
for i in {1..100} # Use i to loop from "1" to "100", inclusive.
do  ((i % 3)) &&  # If i is not divisible by 3...
        x= ||     # ...blank out x (yes, "x= " does that).  Otherwise,...
        x=Fizz    # ...set (not append) x to the string "Fizz".
    ((i % 5)) ||  # If i is not divisible by 5, skip (there's no "&&")...
        x+=Buzz   # ...Otherwise, append (not set) the string "Buzz" to x.
   echo ${x:-$i}  # Print x unless it is blanked out.  Otherwise, print i.
done
```

It's a bit silly to optimize such a small & fast program, but for the sake of algorithm analysis it's worth noting that the concise approach is reasonably efficient in several ways. Each divisibility test appears in the code exactly once, only two variables are created, and the approach avoids setting variables unnecessarily. As far as I can tell, the divisibility tests only fire the minimum number of times required for the general case (e.g. where the 100/3/5 constants can be changed), unless you introduce more variables and test types. Corrections invited. I avoided analyzing the non-general case where 100/3/5 never change, because one "optimal" solution is to simply print the pre-computed answer,

BASIC

Works with: QuickBasic version 4.5

If/else ladder approach

```
FOR A = 1 TO 100
   IF A MOD 15 = 0 THEN
      PRINT "FizzBuzz"
   ELSE IF A MOD 3 = 0 THEN
      PRINT "Fizz"
   ELSE IF A MOD 5 = 0 THEN
      PRINT "Buzz"
   ELSE
      PRINT A
```

```
   END IF
NEXT A
```

Concatenation approach

```
FOR A = 1 TO 100
   OUT$ = ""

   IF A MOD 3 = 0 THEN
      OUT$ = "Fizz"
   END IF

   IF A MOD 5 = 0 THEN
      OUT$ = OUT$ + "Buzz"
   END IF

   IF OUT$ = "" THEN
      OUT$ = STR$(A)
   END IF

   PRINT OUT$
NEXT A
```

See also: RapidQ

Applesoft BASIC

```
10  DEF  FN M(N) = ((A / N) -  INT (A / N)) * N
20  FOR A = 1 TO 100
30  LET O$ = ""
40  IF  FN M(3) = 0 THEN O$ = "FIZZ"
50  IF  FN M(5) = 0 THEN O$ = O$ + "BUZZ"
60  IF O$ = "" THEN O$ =  STR$ (A)
70  PRINT O$
80  NEXT A
```

Batch File

FOR /L version:

```
@echo off
for /L %%i in (1,1,100) do call :tester %%i
goto :eof

:tester
  set /a test = %1 %% 15
  if %test% NEQ 0 goto :NotFizzBuzz
  echo FizzBuzz
  goto :eof

:NotFizzBuzz
  set /a test = %1 %% 5
  if %test% NEQ 0 goto :NotBuzz
  echo Buzz
  goto :eof

:NotBuzz
  set /a test = %1 %% 3
  if %test% NEQ 0 goto :NotFizz
  echo Fizz
  goto :eof
```

```
:NotFizz
  echo %1
```

Loop version:

```
@echo off
set n=1

:loop
  call :tester %n%
  set /a n += 1
  if %n% LSS 101 goto loop
  goto :eof

:tester
  set /a test = %1 %% 15
  if %test% NEQ 0 goto :NotFizzBuzz
  echo FizzBuzz
  goto :eof

:NotFizzBuzz
  set /a test = %1 %% 5
  if %test% NEQ 0 goto :NotBuzz
  echo Buzz
  goto :eof

:NotBuzz
  set /a test = %1 %% 3
  if %test% NEQ 0 goto :NotFizz
  echo Fizz
  goto :eof

:NotFizz
  echo %1
```

FOR /L with a block instead of very-high-overhead subroutine call:

```
@echo off & setlocal enabledelayedexpansion
for /l %%i in (1,1,100) do (
  set /a m5=%%i %% 5
  set /a m3=%%i %% 3
  set s=
  if !m5! equ 0 set s=!s!Fizz
  if !m3! equ 0 set s=!s!Buzz
  if "!s!"=="" set s=%%i
  echo !s!
)
```

BBC BASIC

```
      FOR number% = 1 TO 100
        CASE TRUE OF
          WHEN number% MOD 15 = 0: PRINT "FizzBuzz"
          WHEN number% MOD 3 = 0:  PRINT "Fizz"
          WHEN number% MOD 5 = 0:  PRINT "Buzz"
          OTHERWISE: PRINT ; number%
        ENDCASE
      NEXT number%
```

bc

This solution never uses `else`, because bc has no `else` keyword (but some implementations add `else` as an extension).

```
for (i = 1; i <= 100; i++) {
        w = 0
        if (i % 3 == 0) { "Fizz"; w = 1; }
        if (i % 5 == 0) { "Buzz"; w = 1; }
        if (w == 0) i
        if (w == 1) "
"
}
quit
```

Befunge

(befunge 93)

```
55*4*v     _    v
v    <>:1-:^
     |:<$       <     ,*48 <
     @>0"zzif">:#,_$       v
>:3%!|     >0"zzub">:#,_$^
      >:5%!|
v "buzz"0<>:.             ^
          |!%5:            <
>:#,_    $>               ^
```

Boo

```
def fizzbuzz(size):
    for i in range(1, size):
        if i%15 == 0:
            print 'FizzBuzz'
        elif i%5 == 0:
            print 'Buzz'
        elif i%3 == 0:
            print 'Fizz'
        else:
            print i

fizzbuzz(101)
```

Bracmat

```
0:?i&whl'(1+!i:<101:?i&out$(mod$(!i.3):0&(mod$(!i.5):0&FizzBuzz|Fizz)|mod$(!i.5):0&Buzz|!i))
```

Same code, pretty printed:

```
  0:?i
&   whl
  ' ( 1+!i:<101:?i
    &   out
      $ (   mod$(!i.3):0
          & ( mod$(!i.5):0&FizzBuzz
            | Fizz
            )
```

```
        | mod$(!i.5):0&Buzz
        | !i
        )
    )
```

Brat

```
1.to 100 { n |
  true? n % 15 == 0
    { p "FizzBuzz" }
    { true? n % 3 == 0
      { p "Fizz" }
      { true? n % 5 == 0
        { p "Buzz" }
        { p n }
      }
    }
  }
```

Brainf***

```
FizzBuzz

Memory:
  Zero
  Zero
  Counter 1
  Counter 2

  Zero
  ASCIIDigit 3
  ASCIIDigit 2
  ASCIIDigit 1

  Zero
  Digit 3
  Digit 2
  Digit 1

  CopyPlace
  Mod 3
  Mod 5
  PrintNumber

  TmpFlag

Counters for the loop
++++++++++[>++++++++++[>+>+<<-]<-]

Number representation in ASCII
>>>>
++++++++ ++++++++ ++++++++ ++++++++ ++++++++ ++++++++ [>+>+>+<<<-]
<<<<

>>
[
    Do hundret times:

    Decrement counter
    ->->

    Increment Number
    > >>+>
        > >>+>
        <<<<
    <<<<
```

```
Check for Overflow
++++++++++
>>> >>>>
>++++++++++<
[-<<< <<<<->>>> >>> >-<]
++++++++++
<<< <<<<

Restore the digit
[->>>> >>>-<<< <<<<]
>>>> [-]+ >>>>[<<<< - >>>>[-]]<<<< <<<<

If there is an overflow
>>>>[
    <<<<

    >>>----------> >>>----------<+<< <<+<<

    Check for Overflow
    ++++++++++
    >> >>>>
    >>++++++++++<<
    [-<< <<<<->>>> >> >>-<<]
    ++++++++++
    << <<<<

    Restore the digit
    [->>>> >>-<< <<<<]
    >>>> [-]+ >>>>[<<<< - >>>>[-]]<<<< <<<<

    If there (again) is an overflow
    >>>>[
        <<<<
        >>---------->> >>----------<+< <<<+<

        >>>>
        [-]
    ]<<<<

    >>>>
    [-]
]<<<<

>>>> >>>>

Set if to print the number
>>>[-]+<<<

Handle the Mod 3 counter
[-]+++

>>>>[-]+<<<<
>+[-<->]+++<
[->->>>[-]<<<<]
>>>>[
    <[-]>

    [-]
    Print "Fizz"
    ++++++++ ++++++++ ++++++++ ++++++++
    ++++++++ ++++++++ ++++++++ ++++++++
    ++++++.

    ++++++++ ++++++++ ++++++++ ++++++++
    +++.

    ++++++++ ++++++++ +..

    [-]
    <<<--->>>
]<<<<

Handle the Mod 5 counter
```

```
    [-]+++++

    >>>>[-]+<<<<
    >>+[-<<->>]++++++<<
    [->>->>[-]<<<<]
    >>>>[
        <[-]>

        [-]
        Print "Buzz"
        ++++++++ ++++++++ ++++++++ ++++++++
        ++++++++ ++++++++ ++++++++ ++++++++
        ++.

        ++++++++ ++++++++ ++++++++ ++++++++
        ++++++++ ++++++++ +++.

        +++++..

        [-]
        <<----->>
    ]<<<<

    Check if to print the number (Leading zeros)
    >>>[
        <<< <<<< <<<<
        >.>.>.<<<
        >>> >>>> >>>>
        [-]
    ]<<<

    <<<< <<<<

    Print New Line
    <<<<[-]++++ ++++ ++++ +.----.[-]>>
]
<<
```

C

```c
#include<stdio.h>

int main (void)
{
    int i;
    for (i = 1; i <= 100; i++)
    {
        if (!(i % 15))
            printf ("FizzBuzz");
        else if (!(i % 3))
            printf ("Fizz");
        else if (!(i % 5))
            printf ("Buzz");
        else
            printf ("%d", i);

        printf("\n");
    }
    return 0;
}
```

Implicit int main and return 0 (C99+):

```c
#include <stdio.h>

main() {
  int i = 1;
```

```
  while(i <= 100) {
    if(i % 15 == 0)
      puts("FizzBuzz");
    else if(i % 3 == 0)
      puts("Fizz");
    else if(i % 5 == 0)
      puts("Buzz");
    else
      printf("%d\n", i);
    i++;
  }
}
```

obfuscated:

```
#include <stdio.h>
#define F(x,y) printf("%s",i%x?"":#y"zz")
int main(int i){for(--i;i++^100;puts(""))F(3,Fi)|F(5,Bu)||printf("%i",i);return 0;}
```

This actually works (the array init part, saves 6 bytes of static data, whee):

```
#include<stdio.h>

int main ()
{
        int i;
        const char *s[] = { "%d\n", "Fizz\n", s[3] + 4, "FizzBuzz\n" };
        for (i = 1; i <= 100; i++)
                printf(s[!(i % 3) + 2 * !(i % 5)], i);

        return 0;
}
```

C++

```
#include <iostream>

using namespace std;
int main () {

       int i;
       for (i = 0; i <= 100; i++) {
               if ((i % 15) == 0)
                       cout << "FizzBuzz" << endl;
               else if ((i % 3) == 0)
                       cout << "Fizz" << endl;
               else if ((i % 5) == 0)
                       cout << "Buzz" << endl;
               else
                       cout << i << endl;
       }
       return 0;
}
```

Alternate version not using modulo 15:

```
#include <iostream>
using namespace std;

int main()
{
  for (int i = 0; i <= 100; ++i)
```

```
  {
    bool fizz = (i % 3) == 0;
    bool buzz = (i % 5) == 0;
    if (fizz)
      cout << "Fizz";
    if (buzz)
      cout << "Buzz";
    if (!fizz && !buzz)
      cout << i;
    cout << endl;
  }
  return 0;
}
```

A version using std::transform:

Works with: C++11

```
#include <iostream>
#include <algorithm>
#include <vector>

int main()
{
  std::vector<int> range(100);
  std::iota(range.begin(), range.end(), 1);

  std::vector<std::string> values;
  values.resize(range.size());

  auto fizzbuzz = [](int i) -> std::string {
    if ((i%15) == 0) return "FizzBuzz";
    if ((i%5) == 0)  return "Buzz";
    if ((i%3) == 0)  return "Fizz";
    return std::to_string(i);
  };

  std::transform(range.begin(), range.end(), values.begin(), fizzbuzz);

  for (auto& str: values) std::cout << str << std::endl;

  return 0;
}
```

Version computing FizzBuzz at compile time with metaprogramming:

```
#include <iostream>

template <int n, int m3, int m5>
struct fizzbuzz : fizzbuzz<n-1, (n-1)%3, (n-1)%5>
{
  fizzbuzz()
  { std::cout << n << std::endl; }
};

template <int n>
struct fizzbuzz<n, 0, 0> : fizzbuzz<n-1, (n-1)%3, (n-1)%5>
{
  fizzbuzz()
  { std::cout << "FizzBuzz" << std::endl; }
};

template <int n, int p>
struct fizzbuzz<n, 0, p> : fizzbuzz<n-1, (n-1)%3, (n-1)%5>
{
  fizzbuzz()
  { std::cout << "Fizz" << std::endl; }
```

```
};

template <int n, int p>
struct fizzbuzz<n, p, 0> : fizzbuzz<n-1, (n-1)%3, (n-1)%5>
{
  fizzbuzz()
  { std::cout << "Buzz" << std::endl; }
};

template <>
struct fizzbuzz<0,0,0>
{
  fizzbuzz()
  { std::cout << 0 << std::endl; }
};

template <int n>
struct fb_run
{
  fizzbuzz<n, n%3, n%5> fb;
};

int main()
{
  fb_run<100> fb;
  return 0;
}
```

Hardcore templates (compile with -ftemplate-depth-9000 -std=c++0x):

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <boost/mpl/string.hpp>
#include <boost/mpl/fold.hpp>
#include <boost/mpl/size_t.hpp>

using namespace std;
using namespace boost;

////////////////////////////////////////////////////////////////////////////////
// exponentiation calculations
template <int accum, int base, int exp> struct POWER_CORE : POWER_CORE<accum * base, base, exp -

template <int accum, int base>
struct POWER_CORE<accum, base, 0>
{
    enum : int { val = accum };
};

template <int base, int exp> struct POWER : POWER_CORE<1, base, exp>{};

////////////////////////////////////////////////////////////////////////////////
// # of digit calculations
template <int depth, unsigned int i> struct NUM_DIGITS_CORE : NUM_DIGITS_CORE<depth + 1, i / 10>

template <int depth>
struct NUM_DIGITS_CORE<depth, 0>
{
    enum : int { val = depth};
};

template <int i> struct NUM_DIGITS : NUM_DIGITS_CORE<0, i>{};

template <>
struct NUM_DIGITS<0>
{
    enum : int { val = 1 };
};

////////////////////////////////////////////////////////////////////////////////
```

```
// Convert digit to character (1 -> '1')
template <int i>
struct DIGIT_TO_CHAR
{
    enum : char{ val = i + 48 };
};

////////////////////////////////////////////////////////////////////////////
// Find the digit at a given offset into a number of the form 0000000017
template <unsigned int i, int place> // place -> [0 .. 10]
struct DIGIT_AT
{
    enum : char{ val = (i / POWER<10, place>::val) % 10 };
};

struct NULL_CHAR
{
    enum : char{ val = '\0' };
};

////////////////////////////////////////////////////////////////////////////
// Convert the digit at a given offset into a number of the form '0000000017' to a character
template <unsigned int i, int place> // place -> [0 .. 9]
    struct ALT_CHAR : DIGIT_TO_CHAR< DIGIT_AT<i, place>::val >{};

////////////////////////////////////////////////////////////////////////////
// Convert the digit at a given offset into a number of the form '17' to a character

// Template description, with specialization to generate null characters for out of range offset
template <unsigned int i, int offset, int numDigits, bool inRange>
    struct OFFSET_CHAR_CORE_CHECKED{};
template <unsigned int i, int offset, int numDigits>
    struct OFFSET_CHAR_CORE_CHECKED<i, offset, numDigits, false> : NULL_CHAR{};
template <unsigned int i, int offset, int numDigits>
    struct OFFSET_CHAR_CORE_CHECKED<i, offset, numDigits, true>  : ALT_CHAR<i, (numDigits - offs

// Perform the range check and pass it on
template <unsigned int i, int offset, int numDigits>
    struct OFFSET_CHAR_CORE : OFFSET_CHAR_CORE_CHECKED<i, offset, numDigits, offset < numDigits>

// Calc the number of digits and pass it on
template <unsigned int i, int offset>
    struct OFFSET_CHAR : OFFSET_CHAR_CORE<i, offset, NUM_DIGITS<i>::val>{};

////////////////////////////////////////////////////////////////////////////
// Integer to char* template. Works on unsigned ints.
template <unsigned int i>
struct IntToStr
{
    const static char str[];
    typedef typename mpl::string<
    OFFSET_CHAR<i, 0>::val,
    OFFSET_CHAR<i, 1>::val,
    OFFSET_CHAR<i, 2>::val,
    OFFSET_CHAR<i, 3>::val,
    OFFSET_CHAR<i, 4>::val,
    OFFSET_CHAR<i, 5>::val,
    /*OFFSET_CHAR<i, 6>::val,
    OFFSET_CHAR<i, 7>::val,
    OFFSET_CHAR<i, 8>::val,
    OFFSET_CHAR<i, 9>::val,*/
    NULL_CHAR::val>::type type;
};

template <unsigned int i>
const char IntToStr<i>::str[] =
{
    OFFSET_CHAR<i, 0>::val,
    OFFSET_CHAR<i, 1>::val,
    OFFSET_CHAR<i, 2>::val,
    OFFSET_CHAR<i, 3>::val,
    OFFSET_CHAR<i, 4>::val,
    OFFSET_CHAR<i, 5>::val,
    OFFSET_CHAR<i, 6>::val,
```

```
    OFFSET_CHAR<i, 7>::val,
    OFFSET_CHAR<i, 8>::val,
    OFFSET_CHAR<i, 9>::val,
    NULL_CHAR::val
};

template <bool condition, class Then, class Else>
struct IF
{
    typedef Then RET;
};

template <class Then, class Else>
struct IF<false, Then, Else>
{
    typedef Else RET;
};


template < typename Str1, typename Str2 >
struct concat : mpl::insert_range<Str1, typename mpl::end<Str1>::type, Str2> {};
template <typename Str1, typename Str2, typename Str3 >
struct concat3 : mpl::insert_range<Str1, typename mpl::end<Str1>::type, typename concat<Str2, St

typedef typename mpl::string<'f','i','z','z'>::type fizz;
typedef typename mpl::string<'b','u','z','z'>::type buzz;
typedef typename mpl::string<'\r', '\n'>::type mpendl;
typedef typename concat<fizz, buzz>::type fizzbuzz;

// discovered boost mpl limitation on some length

template <int N>
struct FizzBuzz
{
    typedef typename concat3<typename FizzBuzz<N - 1>::type, typename IF<N % 15 == 0, typename f
};

template <>
struct FizzBuzz<1>
{
    typedef mpl::string<'1','\r','\n'>::type type;
};

int main(int argc, char** argv)
{
    const int n = 7;
    std::cout << mpl::c_str<FizzBuzz<n>::type>::value << std::endl;
        return 0;
}
```

Note: it takes up lots of memory and takes several seconds to compile. To enable compilation for 7 < n <= 25, please, modify include/boost/mpl/limits/string.hpp BOOST_MPL_LIMIT_STRING_SIZE to 128 instead of 32).

C#

```
using System;
using System.Text;

namespace FizzBuzz
{
    class Program
    {
        static void Main()
        {
            var output = new StringBuilder();
            for (var i = 1; i <= 100; i++)
```

```
            {
                output.AppendFormat("{0}{1}{2}",
                    (i%3 == 0) ? "Fizz" : string.Empty,
                    (i%5 == 0) ? "Buzz" : string.Empty,
                    Environment.NewLine);
            }
            Console.WriteLine(output);
        }
    }
}
```

```
using System;
using System.Linq;

namespace FizzBuzz
{
    class Program
    {
        static void Main(string[] args)
        {
            Enumerable.Range(1, 100)
                .Select(a => String.Format("{0}{1}", a % 3 == 0 ? "Fizz" : string.Empty, a % 5 =
                .Select((b, i) => String.IsNullOrEmpty(b) ? (i + 1).ToString() : b)
                .ToList()
                .ForEach(Console.WriteLine);
        }
    }
}
```

```
using System;
using System.Globalization;
using System.Linq;

namespace FizzBuzz
{
    class Program
    {
        static void Main()
        {
            Enumerable.Range(1, 100)
                .GroupBy(e => e % 15 == 0 ? "FizzBuzz" : e % 5 == 0 ? "Buzz" : e % 3 == 0 ? "Fiz
                .SelectMany(item => item.Select(x => new {
                    Value = x,
                    Display = String.IsNullOrEmpty(item.Key) ? x.ToString(CultureInfo.InvariantC
                }))
                .OrderBy(x => x.Value)
                .Select(x => x.Display)
                .ToList()
                .ForEach(Console.WriteLine);
        }
    }
}
```

```
using System;
using System.Globalization;

namespace Rosettacode
{
    class Program
    {
        static void Main()
        {
            for (var number = 0; number < 100; number++)
            {
                if ((number % 3) == 0 & (number % 5) == 0)
                {
                    //For numbers which are multiples of both three and five print "FizzBuzz".
```

```
                    Console.WriteLine("FizzBuzz");
                    continue;
                }

                if ((number % 3) == 0) Console.WriteLine("Fizz");
                if ((number % 5) == 0) Console.WriteLine("Buzz");
                if ((number % 3) != 0 && (number % 5) != 0) Console.WriteLine(number.ToString(Cu

                if (number % 5 == 0)
                {
                    Console.WriteLine(Environment.NewLine);
                }
            }
        }
    }
}
```

TDD using delegates.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace FizzBuzz
{
    [TestClass]
    public class FizzBuzzTest
    {
        private FizzBuzz fizzBuzzer;

        [TestInitialize]
        public void Initialize()
        {
            fizzBuzzer = new FizzBuzz();
        }

        [TestMethod]
        public void Give4WillReturn4()
        {
            Assert.AreEqual("4", fizzBuzzer.FizzBuzzer(4));
        }

        [TestMethod]
        public void Give9WillReturnFizz()
        {
            Assert.AreEqual("Fizz", fizzBuzzer.FizzBuzzer(9));
        }

        [TestMethod]
        public void Give25WillReturnBuzz()
        {
            Assert.AreEqual("Buzz", fizzBuzzer.FizzBuzzer(25));
        }

        [TestMethod]
        public void Give30WillReturnFizzBuzz()
        {
            Assert.AreEqual("FizzBuzz", fizzBuzzer.FizzBuzzer(30));
        }

        [TestMethod]
        public void First15()
        {
            ICollection expected = new ArrayList
                {"1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz",

            var actual = Enumerable.Range(1, 15).Select(x => fizzBuzzer.FizzBuzzer(x)).ToList();
```

```
            CollectionAssert.AreEqual(expected, actual);
        }

        [TestMethod]
        public void From1To100_ToShowHowToGet100()
        {
            const int expected = 100;
            var actual = Enumerable.Range(1, 100).Select(x => fizzBuzzer.FizzBuzzer(x)).ToList()

            Assert.AreEqual(expected, actual.Count);
        }
    }

    public class FizzBuzz
    {
        private delegate string Xzzer(int value);
        private readonly IList<Xzzer> _functions = new List<Xzzer>();

        public FizzBuzz()
        {
            _functions.Add(x => x % 3 == 0 ? "Fizz" : "");
            _functions.Add(x => x % 5 == 0 ? "Buzz" : "");
        }

        public string FizzBuzzer(int value)
        {
            var result = _functions.Aggregate(String.Empty, (current, function) => current + fun
            return String.IsNullOrEmpty(result) ? value.ToString(CultureInfo.InvariantCulture) :
        }
    }
}
```

Cduce

```
(* FizzBuzz in CDuce *)

let format (n : Int) : Latin1 =
    if (n mod 3 = 0) || (n mod 5 = 0) then "FizzBuzz"
    else if (n mod 5 = 0) then "Buzz"
    else if (n mod 3 = 0) then "Fizz"
    else string_of (n);;

let fizz (n : Int, size : Int) : _ =
    print (format (n) @ "\n");
    if (n = size) then
        n = 0 (* do nothing *)
    else
        fizz(n + 1, size);;

let fizbuzz (size : Int) : _ = fizz (1, size);;

let _ = fizbuzz(100);;
```

Chef

This was clearly a challenge in a language without a modulus operator, a proper if statement except for checking if a variable is not exactly 0, and no way to define text except 1 character at a time on a stack.

```
Irish Soda Bread with Club Soda.

This is FizzBuzz

Ingredients.
```

1 l buttermilk

Method.
Take buttermilk from refrigerator.
Shake the buttermilk.
 Put buttermilk into the 6th mixing bowl.
 Serve with club soda.
 Pour contents of the 1st mixing bowl into the 1st baking dish.
 Clean the 1st mixing bowl.
Watch the buttermilk until shaked.

Serves 1.

Club Soda.

Gets whether to print fizz buzz fizzbuzz or number.

Ingredients.
70 g flour
105 g salt
122 ml milk
66 g sugar
117 ml vegetable oil
3 cups fizzle
5 cups seltzer
1 cup bmilk
1 cup ice
1 cup baking soda
1 g oregano
32 ml vinegar
1 g thyme
1 g sage

Method.
Put milk into the 1st mixing bowl.
Put salt into the 1st mixing bowl.
Put flour into the 1st mixing bowl.
Put vinegar into the 1st mixing bowl.
Put milk into the 1st mixing bowl.
Stir the 1st mixing bowl for 5 minutes.
Liquify contents of the 1st mixing bowl.
Put fizzle into the 3rd mixing bowl.
Combine seltzer into the 3rd mixing bowl.
Fold bmilk into the 6th mixing bowl.
Put bmilk into the 6th mixing bowl.
Put seltzer into the 6th mixing bowl.
Put bmilk into the 6th mixing bowl.
Serve with moist cake.
Fold bmilk into the 1st mixing bowl.
Fold sage into the 6th mixing bowl.
Fold sage into the 6th mixing bowl.
Put bmilk into the 4th mixing bowl.
Remove seltzer from the 4th mixing bowl.
Fold oregano into the 4th mixing bowl.
Smell the oregano.
 Fold bmilk into the 6th mixing bowl.
 Put bmilk into the 6th mixing bowl.
 Put fizzle into the 6th mixing bowl.
 Put bmilk into the 6th mixing bowl.
 Serve with moist cake.
 Fold bmilk into the 1st mixing bowl.
 Fold sage into the 6th mixing bowl.
 Fold sage into the 6th mixing bowl.
 Put bmilk into the 4th mixing bowl.
 Remove fizzle from the 4th mixing bowl.
 Fold oregano into the 4th mixing bowl.
 Crush the oregano.
  Clean the 1st mixing bowl.
  Fold bmilk into the 6th mixing bowl.
  Put bmilk into the 1st mixing bowl.
  Refrigerate.
 Grind until crushed.
 Refrigerate.
Shuffle until smelled.

```
Clean the 1st mixing bowl.
Put milk into the 1st mixing bowl.
Put vegetable oil into the 1st mixing bowl.
Put sugar into the 1st mixing bowl.
Put vinegar into the 1st mixing bowl.
Put milk into the 1st mixing bowl.
Stir the 1st mixing bowl for 5 minutes.
Liquify contents of the 1st mixing bowl.
Fold baking soda into the 3rd mixing bowl.
Fold bmilk into the 6th mixing bowl.
Put bmilk into the 6th mixing bowl.
Put baking soda into the 6th mixing bowl.
Put bmilk into the 6th mixing bowl.
Serve with moist cake.
Fold bmilk into the 1st mixing bowl.
Fold sage into the 6th mixing bowl.
Fold sage into the 6th mixing bowl.
Put bmilk into the 4th mixing bowl.
Remove baking soda from the 4th mixing bowl.
Fold oregano into the 4th mixing bowl.
Separate the oregano.
 Refrigerate.
Part until separated.
Put fizzle into the 6th mixing bowl.
Serve with club soda.
Stir the 1st mixing bowl for 1 minute.
Stir the 1st mixing bowl for 7 minutes.
Stir the 1st mixing bowl for 1 minute.
Stir the 1st mixing bowl for 7 minutes.
Stir the 1st mixing bowl for 5 minutes.
Fold the oregano into the 1st mixing bowl.
Fold the oregano into the 1st mixing bowl.
Fold the oregano into the 1st mixing bowl.
Fold the oregano into the 1st mixing bowl.
Fold the oregano into the 1st mixing bowl.
Refrigerate.

Moist cake.

Mods a number

Ingredients.
1 cup chocolate
70 g wheat flour
1 cup white chocolate chips
1 cup baking powder
105 g honey
5 cups syrup
1 g vanilla
1 g rosemary

Method.
Fold chocolate into the 6th mixing bowl.
Fold syrup into the 6th mixing bowl.
Clean the 1st mixing bowl.
Put chocolate into the 5th mixing bowl.
Fold wheat flour into the 5th mixing bowl.
Put white chocolate chips into the 5th mixing bowl.
Remove white chocolate chips from the 5th mixing bowl.
Fold baking powder into the 5th mixing bowl.
Put baking powder into the 5th mixing bowl.
Fold honey into the 5th mixing bowl.
Sift the wheat flour.
 Put honey into the 5th mixing bowl.
 Add white chocolate chips into the 5th mixing bowl.
 Fold honey into the 5th mixing bowl.
 Put honey into the 5th mixing bowl.
 Remove syrup from the 5th mixing bowl.
 Fold vanilla into the 5th mixing bowl.
 Sprinkle the vanilla.
  Put white chocolate chips into the 5th mixing bowl.
  Remove white chocolate chips from the 5th mixing bowl.
  Fold rosemary into the 5th mixing bowl.
  Set aside.
```

```
 Move until sprinkled.
 Recite the rosemary.
  Put white chocolate chips into the 5th mixing bowl.
  Remove white chocolate chips from the 5th mixing bowl.
  Fold honey into the 5th mixing bowl.
  Put baking powder into the 5th mixing bowl.
  Add white chocolate chips into the 5th mixing bowl.
  Fold baking powder into the 5th mixing bowl.
  Set aside.
 Repeat until recited.
 Put white chocolate chips into the 5th mixing bowl.
 Fold rosemary into the 5th mixing bowl.
Shuffle the wheat flour until sifted.
Put the baking powder into the 5th mixing bowl.
Combine syrup into the 5th mixing bowl.
Fold honey into the 5th mixing bowl.
Put chocolate into the 5th mixing bowl.
Remove honey from the 5th mixing bowl.
Fold chocolate into the 5th mixing bowl.
Put white chocolate chips into the 5th mixing bowl.
Fold rosemary into the 5th mixing bowl.
Siphon chocolate.
 Put white chocolate chips into the 5th mixing bowl.
 Remove white chocolate chips from the 5th mixing bowl.
 Fold rosemary into the 5th mixing bowl.
 Set aside.
Gulp until siphoned.
Quote the rosemary.
 Put syrup into the 5th mixing bowl.
 Fold chocolate into the 5th mixing bowl.
 Set aside.
Repeat until quoted.
Put chocolate into the 1st mixing bowl.
Refrigerate.
```

Clay

```
main() {
    for(i in range(1,100)) {
        if(i % 3 == 0 and i % 5 == 0) println("fizzbuzz");
        else if(i % 3 == 0) println("fizz");
        else if(i % 5 == 0) println("buzz");
        else print(i);
    }
}
```

Clipper

```
Procedure Main()
   Local n
   Local cFB
   For n := 1 to 100
      cFB := ""
      AEval( {{3,"Fizz"},{5,"Buzz"}}, {|x| cFB += Iif((n % x[1])==0, x[2], "")})
      ?? Iif(cFB == "", LTrim(Str(n)), cFB) + Iif(n == 100, ".", ", ")
   Next
Return
```

The advantage of this approach is that it is trivial to add another factor:

```
AEval( {{3,"Fizz"},{5,"Buzz"},{9,"Jazz"}}, {|x| cFB += Iif((n % x[1])==0, x[2], "")})
```

CLIPS

```
(deffacts count
  (count-to 100)
)

(defrule print-numbers
  (count-to ?max)
  =>
  (loop-for-count (?num ?max) do
    (if
      (= (mod ?num 3) 0)
      then
      (printout t "Fizz")
    )
    (if
      (= (mod ?num 5) 0)
      then
      (printout t "Buzz")
    )
    (if
      (and (> (mod ?num 3) 0) (> (mod ?num 5) 0))
      then
      (printout t ?num)
    )
    (priint depth, unsigned int i> struct NUM_DIGITS_CORE : NUM_DIGITS_COREntout t crlf)
  )
)
```

Clojure

```
(map (fn [x] (cond (zero? (mod x 15)) "FizzBuzz"
                   (zero? (mod x 5)) "Buzz"
                   (zero? (mod x 3)) "Fizz"
                     :else x))
     (range 1 101))
```

```
(map #(let [s (str (if (zero? (mod % 3)) "Fizz") (if (zero? (mod % 5)) "Buzz"))] (if (empty? s)
```

```
(def fizzbuzz (map
  #(cond (zero? (mod % 15)) "FizzBuzz"
         (zero? (mod % 5)) "Buzz"
         (zero? (mod % 3)) "Fizz"
               :else %)
  (iterate inc 1)))
```

```
(defn fizz-buzz
  ([] (fizz-buzz (range 1 101)))
  ([lst]
     (letfn [(fizz? [n] (zero? (mod n 3)))
             (buzz? [n] (zero? (mod n 5)))]
       (let [f     "Fizz"
             b     "Buzz"
             items (map (fn [n]
                          (cond (and (fizz? n) (buzz? n)) (str f b)
                                (fizz? n) f
                                (buzz? n) b
                                :else n))
                        lst)] items))))
```

```clojure
(map (fn [n]
       (if-let [fb (seq (concat (when (zero? (mod n 3)) "Fizz")
                                (when (zero? (mod n 5)) "Buzz")))]
           (apply str fb)
           n))
     (range 1 101))
```

```clojure
(take 100 (map #(let [s (str %2 %3) ] (if (seq s) s (inc %)) )
            (range)
            (cycle [ "" "" "Fizz" ])
            (cycle [ "" "" "" "" "Buzz" ])))
```

CMake

```cmake
foreach(i RANGE 1 100)
  math(EXPR off3 "${i} % 3")
  math(EXPR off5 "${i} % 5")
  if(NOT off3 AND NOT off5)
    message(FizzBuzz)
  elseif(NOT off3)
    message(Fizz)
  elseif(NOT off5)
    message(Buzz)
  else()
    message(${i})
  endif()
endforeach(i)
```

COBOL

Canonical version

Works with: OpenCOBOL

```cobol
      * FIZZBUZZ.COB
      * cobc -x -g FIZZBUZZ.COB
      *
       IDENTIFICATION        DIVISION.
       PROGRAM-ID.           fizzbuzz.
       DATA                  DIVISION.
       WORKING-STORAGE       SECTION.
       01 CNT      PIC 9(03) VALUE 1.
       01 REM      PIC 9(03) VALUE 0.
       01 QUOTIENT PIC 9(03) VALUE 0.
       PROCEDURE             DIVISION.
      *
       PERFORM UNTIL CNT > 100
         DIVIDE 15 INTO CNT GIVING QUOTIENT REMAINDER REM
         IF REM = 0
           THEN
             DISPLAY "FizzBuzz " WITH NO ADVANCING
           ELSE
             DIVIDE 3 INTO CNT GIVING QUOTIENT REMAINDER REM
             IF REM = 0
               THEN
                 DISPLAY "Fizz " WITH NO ADVANCING
               ELSE
                 DIVIDE 5 INTO CNT GIVING QUOTIENT REMAINDER REM
                 IF REM = 0
                   THEN
                     DISPLAY "Buzz " WITH NO ADVANCING
                   ELSE
```

```
                    DISPLAY CNT " " WITH NO ADVANCING
                END-IF
            END-IF
          END-IF
          ADD 1 TO CNT
        END-PERFORM
        DISPLAY ""
        STOP RUN.
```

Simpler version

I know this doesn't have the full-bodied, piquant flavor expected from COBOL, but it is a little shorter.

Works with: OpenCOBOL

```
Identification division.
Program-id. fizz-buzz.

Data division.
Working-storage section.

01 num pic 999.

Procedure division.
    Perform varying num from 1 by 1 until num > 100
        if function mod (num, 15) = 0 then display "fizzbuzz"
        else if function mod (num, 3) = 0 then display "fizz"
        else if function mod (num, 5) = 0 then display "buzz"
        else display num
    end-perform.
    Stop run.
```

Coco

```
for i from 1 to 100
    console.log do
       if      i % 15 == 0 then 'FizzBuzz'
       else if i % 3 == 0 then 'Fizz'
       else if i % 5 == 0 then 'Buzz'
       else i
```

```
for i from 1 to 100
    console.log(['Fizz' unless i % 3] + ['Buzz' unless i % 5] or String(i))
```

CoffeeScript

```
for i in [1..100]
  if i % 15 is 0
    console.log "FizzBuzz"
  else if i % 3 is 0
    console.log "Fizz"
  else if i % 5 is 0
    console.log "Buzz"
  else
    console.log i
```

```
for i in [1..100]
  console.log(['Fizz' if i % 3 is 0] + ['Buzz' if i % 5 is 0] or i)
```

Common Lisp

Solution 1:

```
(defun fizzbuzz ()
  (loop for x from 1 to 100 do
    (princ (cond ((zerop (mod x 15)) "FizzBuzz")
                 ((zerop (mod x 3))  "Fizz")
                 ((zerop (mod x 5))  "Buzz")
                 (t                  x)))
    (terpri)))
```

Solution 2:

```
(defun fizzbuzz ()
  (loop for x from 1 to 100 do
    (format t "~&~{~A~}"
      (or (append (when (zerop (mod x 3)) '("Fizz"))
                  (when (zerop (mod x 5)) '("Buzz")))
          (list x)))))
```

Solution 3:

```
(defun fizzbuzz ()
  (loop for n from 1 to 100
     do (format t "~&~[~[FizzBuzz~:;Fizz~]~*~:;~[Buzz~*~:;~D~]~]~%"
                (mod n 3) (mod n 5) n)))
```

Solution 4:

```
(loop as n from 1 to 100
      as fizz = (zerop (mod n 3))
      as buzz = (zerop (mod n 5))
      as numb = (not (or fizz buzz))
      do
  (format t
   "~&~:[~;Fizz~]~:[~;Buzz~]~:[~;~D~]~%"
   fizz buzz numb n))
```

Solution 5:

```
(format t "~{~:[~&~;~:*~:(~a~)~]~}"
  (loop as n from 1 to 100
        as f = (zerop (mod n 3))
        as b = (zerop (mod n 5))
        collect nil
        if f collect 'fizz
        if b collect 'buzz
        if (not (or f b)) collect n))
```

Solution 6:

```
(format t "~{~{~:[~;Fizz~]~:[~;Buzz~]~:[~*~;~d~]~}~%~}"
  (loop as n from 1 to 100
        as f = (zerop (mod n 3))
```

```
        as b = (zerop (mod n 5))
        collect (list f b (not (or f b)) n)))
```

First 16 lines of output:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
```

Cubescript

```
alias fizzbuzz [
        loop i 100 [
                push i (+ $i 1) [
                        cond (! (mod $i 15)) [
                                echo FizzBuzz
                        ] (! (mod $i 3)) [
                                echo Fizz
                        ] (! (mod $i 5)) [
                                echo Buzz
                        ] [
                                echo $i
                        ]
                ]
        ]
]
```

Chapel

```
proc fizzbuzz(n) {
        for i in 1..n do
                if i % 15 == 0 then
                        writeln("FizzBuzz");
                else if i % 5 == 0 then
                        writeln("Buzz");
                else if i % 3 == 0 then
                        writeln("Fizz");
                else
                        writeln(i);
}

fizzbuzz(100);
```

D

```
import std.stdio: writeln;
```

```
// with if-else
void fizzBuzz(int n) {
    foreach (i; 1 .. n+1)
        if (!(i % 15))
            writeln("FizzBuzz");
        else if (!(i % 3))
            writeln("Fizz");
        else if (!(i % 5))
            writeln("Buzz");
        else
            writeln(i);
}

// with switch case
void fizzBuzzSwitch(int n) {
    foreach (i; 1 .. n+1)
        switch(i % 15) {
            case 0:
                writeln("FizzBuzz"); break;
            case 3, 6, 9, 12:
                writeln("Fizz"); break;
            case 5, 10:
                writeln("Buzz"); break;
            default:
                writeln(i);
        }
}

void main() {
    fizzBuzz(100);
    writeln();
    fizzBuzzSwitch(100);
}
```

Dart

```
main() {
  for(int i=1;i<=100;i++)
    print((i%3==0?"Fizz":"")+(i%5==0?"Buzz":"")+(i%3!=0&&i%5!=0?i:""));
}
```

dc

Translation of: bc

```
[[Fizz]P 1 sw]sF
[[Buzz]P 1 sw]sB
[li p sz]sN
[[
]P]sW
[
 0 sw          [w = 0]sz
 li 3 % 0 =F   [Fizz if 0 == i % 3]sz
 li 5 % 0 =B   [Buzz if 0 == i % 5]sz
 lw 0 =N       [print Number if 0 == w]sz
 lw 1 =W       [print neWline if 1 == w]sz
 li 1 + si     [i += 1]sz
 li 100 !<L    [continue Loop if 100 >= i]sz
]sL
1 si           [i = 1]sz
0 0 =L         [enter Loop]sz
```

Delphi

```
program FizzBuzz;

{$APPTYPE CONSOLE}

uses SysUtils;

var
  i: Integer;
begin
  for i := 1 to 100 do
  begin
    if i mod 15 = 0 then
      Writeln('FizzBuzz')
    else if i mod 3 = 0 then
      Writeln('Fizz')
    else if i mod 5 = 0 then
      Writeln('Buzz')
    else
      Writeln(i);
  end;
end.
```

Déjà Vu

```
for i range 1 100:
        if = 0 % i 15:
                "FizzBuzz"
        elseif = 0 % i 3:
                "Fizz"
        elseif = 0 % i 5:
                "Buzz"
        else:
                i
        !print
```

DWScript

```
var i : Integer;

for i := 1 to 100 do begin
   if i mod 15 = 0 then
      PrintLn('FizzBuzz')
   else if i mod 3 = 0 then
      PrintLn('Fizz')
   else if i mod 5 = 0 then
      PrintLn('Buzz')
   else PrintLn(i);
end;
```

E

```
for i in 1..100 {
   println(switch ([i % 3, i % 5]) {
     match [==0, ==0] { "FizzBuzz" }
     match [==0, _  ] { "Fizz" }
     match [_,   ==0] { "Buzz" }
     match _          { i }
```

```
  })
}
```

ECL

```
DataRec := RECORD
    STRING  s;
END;

DataRec MakeDataRec(UNSIGNED c) := TRANSFORM
    SELF.s := MAP
        (
            c % 15 = 0  =>  'FizzBuzz',
            c % 3 = 0   =>  'Fizz',
            c % 5 = 0   =>  'Buzz',
            (STRING)c
        );
END;

d := DATASET(100,MakeDataRec(COUNTER));

OUTPUT(d);
```

Eero

```
#import <Foundation/Foundation.h>

int main()
  autoreleasepool

    for int i in 1 .. 100
      s := ''
      if i % 3 == 0
        s << 'Fizz'
      if i % 5 == 0
        s << 'Buzz'
      Log( '(%d) %@', i, s )

  return 0
```

Ela

```
open list

prt x | x % 15 == 0 = "FizzBuzz"
      | x % 3 == 0  = "Fizz"
      | x % 5 == 0  = "Buzz"
      | else        = x

[1..100] |> map prt
```

Elixir

```
Enum.each 1..100, fn x ->
  IO.puts(case { rem(x, 5) == 0, rem(x,3) == 0 } do
    { true, true } ->
      "FizzBuzz"
    { true, false } ->
```

```
      "Fizz"
    { false, true } ->
      "Buzz"
    { false, false } ->
      x
  end)
end
```

Erlang

```
fizzbuzz() ->
    F = fun(N) when N rem 15 == 0 -> "FizzBuzz";
           (N) when N rem 3 == 0  -> "Fizz";
           (N) when N rem 5 == 0  -> "Buzz";
           (N) -> integer_to_list(N)
        end,
    [F(N)++"\n" || N <- lists:seq(1,100)].
```

Euphoria

Works with: Euphoria version 4.0.0

This is based on the VBScript example.

```
include std/utils.e

function fb( atom n )
        sequence fb
        if remainder( n, 15 ) = 0 then
                fb = "FizzBuzz"
        elsif remainder( n, 5 ) = 0 then
                fb = "Fizz"
        elsif remainder( n, 3 ) = 0 then
                fb = "Buzz"
        else
                fb = sprintf( "%d", n )
        end if
        return fb
end function

function fb2( atom n )
        return iif( remainder(n, 15) = 0, "FizzBuzz",
                iif( remainder( n, 5 ) = 0, "Fizz",
                iif( remainder( n, 3) = 0, "Buzz", sprintf( "%d", n ) ) ) )
end function

for i = 1 to 30 do
        printf( 1, "%s ", { fb( i ) } )
end for

puts( 1, "\n" )

for i = 1 to 30 do
        printf( 1, "%s ", { fb2( i ) } )
end for

puts( 1, "\n" )
```

Factor

```
USING: math kernel io math.ranges ;
IN: fizzbuzz
: fizz ( n -- str ) 3 divisor? "Fizz" "" ? ;
: buzz ( n -- str ) 5 divisor? "Buzz" "" ? ;
: fizzbuzz ( n -- str ) dup [ fizz ] [ buzz ] bi append [ number>string ] [ nip ] if-empty ;
: main ( -- ) 100 [1,b] [ fizzbuzz print ] each ;
MAIN: main
```

F#

```
let fizzbuzz n =
    match n%3 = 0, n%5 = 0 with
    | true, false -> "fizz"
    | false, true -> "buzz"
    | true, true  -> "fizzbuzz"
    | _ -> string n

let printFizzbuzz() =
    [1..100] |> List.iter (fun n -> printfn "%s" (fizzbuzz n))
```

```
[1..100]
|> List.map (fun x ->
            match x with
            | _ when x % 15 = 0 ->"fizzbuzz"
            | _ when x % 5 = 0 -> "buzz"
            | _ when x % 3 = 0 -> "fizz"
            | _ ->  x.ToString())
|> List.iter (fun x -> printfn "%s" x)
```

Another example using (unnecessary) partial active pattern :D

```
let (|MultipleOf|_|) divisors number =
    if Seq.exists ((%) number >> (<>) 0) divisors
    then None
    else Some ()

let fizzbuzz = function
| MultipleOf [3; 5] -> "fizzbuzz"
| MultipleOf [3]    -> "fizz"
| MultipleOf [5]    -> "buzz"
| n                 -> string n

{ 1 .. 100 }
|> Seq.iter (fizzbuzz >> printfn "%s")
```

Falcon

```
for i in [1:101]
    switch i % 15
    case 0        : > "FizzBuzz"
    case 5,10     : > "Buzz"
    case 3,6,9,12 : > "Fizz"
    default       : > i
    end
end
```

FALSE

```
```

```
[\$@$@\/*=]d:
[1\$3d;!["Fizz"\%0\]?$5d;!["Buzz"\%0\]?\[$.]?"
"]f:
0[$100\>][1+f;!]#%
```

Fantom

```
class FizzBuzz
{
  public static Void main ()
  {
    for (Int i:=1; i <= 100; ++i)
    {
      if (i % 15 == 0)
        echo ("FizzBuzz")
      else if (i % 3 == 0)
        echo ("Fizz")
      else if (i % 5 == 0)
        echo ("Buzz")
      else
        echo (i)
    }
  }
}
```

FBSL

No 'MOD 15' needed.

```
#APPTYPE CONSOLE

DIM numbers AS STRING
DIM imod5 AS INTEGER
DIM imod3 AS INTEGER

FOR DIM i = 1 TO 100
    numbers = ""
    imod3 = i MOD 3
    imod5 = i MOD 5
    IF NOT imod3 THEN numbers = "Fizz"
    IF NOT imod5 THEN numbers = numbers & "Buzz"
    IF imod3 AND imod5 THEN numbers = i
    PRINT numbers, " ";
NEXT

PAUSE
```

Output:

```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz Fiz
z 22 23 Fizz Buzz 26 Fizz 28 29 FizzBuzz 31 32 Fizz 34 Buzz Fizz 37 38 Fizz Buzz
 41 Fizz 43 44 FizzBuzz 46 47 Fizz 49 Buzz Fizz 52 53 Fizz Buzz 56 Fizz 58 59 Fi
zzBuzz 61 62 Fizz 64 Buzz Fizz 67 68 Fizz Buzz 71 Fizz 73 74 FizzBuzz 76 77 Fizz
 79 Buzz Fizz 82 83 Fizz Buzz 86 Fizz 88 89 FizzBuzz 91 92 Fizz 94 Buzz Fizz 97
98 Fizz Buzz
Press any key to continue...
```

Forth

table-driven

```
: fizz ( n -- ) drop ." Fizz" ;
: buzz ( n -- ) drop ." Buzz" ;
: fb   ( n -- ) drop ." FizzBuzz" ;
: vector create does> ( n -- )
  over 15 mod cells + @ execute ;
vector .fizzbuzz
  ' fb   , ' . ,    ' . ,
  ' fizz , ' . ,    ' buzz ,
  ' fizz , ' . ,    ' . ,
  ' fizz , ' buzz , ' . ,
  ' fizz , ' . ,    ' . ,
```

or the classic approach

```
: .fizzbuzz ( n -- )
  0 pad c!
  dup 3 mod 0= if s" Fizz" pad  place then
  dup 5 mod 0= if s" Buzz" pad +place then
  pad c@ if drop pad count type else . then ;

: zz ( n -- )
  1+ 1 do i .fizzbuzz cr loop ;
100 zz
```

the well factored approach

SYNONYM is a Forth200x word.

```
SYNONYM NOT INVERT \ Bitwise boolean not

: Fizz?  ( n -- ? )  3 MOD 0=  DUP IF ." Fizz" THEN ;
: Buzz?  ( n -- ? )  5 MOD 0=  DUP IF ." Buzz" THEN ;
: ?print  ( n ? -- )  IF . THEN ;
: FizzBuzz  ( -- )
   101 1 DO CR  I  DUP Fizz? OVER Buzz? OR  NOT ?print  LOOP ;

FizzBuzz
```

Fortran

In ANSI FORTRAN 77 or later use structured IF-THEN-ELSE (example uses some ISO Fortran 90 features):

```
program fizzbuzz_if
   integer :: i

   do i = 1, 100
      if     (mod(i,15) == 0) then; print *, 'FizzBuzz'
      else if (mod(i,3) == 0) then; print *, 'Fizz'
      else if (mod(i,5) == 0) then; print *, 'Buzz'
      else;                         print *, i
      end if
   end do
end program fizzbuzz_if
```

In ISO Fortran 90 or later use SELECT-CASE statement:

```
program fizzbuzz_select
    integer :: i

    do i = 1, 100
       select case (mod(i,15))
          case 0;        print *, 'FizzBuzz'
          case 3,6,9,12; print *, 'Fizz'
          case 5,10;     print *, 'Buzz'
          case default;  print *, i
       end select
    end do
 end program fizzbuzz_select
```

Frink

```
for i = 1 to 100
{
   flag = false
   if i mod 3 == 0
   {
      flag = true
      print["Fizz"]
   }

   if i mod 5 == 0
   {
      flag = true
      print["Buzz"]
   }

   if flag == false
      print[i]

   println[]
}
```

GAP

```
FizzBuzz := function()
        local i;
        for i in [1 .. 100] do
                if RemInt(i, 15) = 0 then
                        Print("FizzBuzz\n");
                elif RemInt(i, 3) = 0 then
                        Print("Fizz\n");
                elif RemInt(i, 5) = 0 then
                        Print("Buzz\n");
                else
                        Print(i, "\n");
                fi;
        od;
end;
```

Go

```
package main

import "fmt"
```

```
func main() {
    for i := 1; i <= 100; i++ {
        switch {
        case i%15==0:
            fmt.Println("FizzBuzz")
        case i%3==0:
            fmt.Println("Fizz")
        case i%5==0:
            fmt.Println("Buzz")
        default:
            fmt.Println(i)
        }
    }
}
```

Gosu

```
for (i in 1..100) {

    if (i % 3 == 0 && i % 5 == 0) {
        print("FizzBuzz")
        continue
    }

    if (i % 3 == 0) {
        print("Fizz")
        continue
    }

    if (i % 5 == 0) {
        print("Buzz")
        continue
    }

    // default
    print(i)

}
```

One liner version (I added new lines to better readability but when you omit them it's one liner):

```
// note that compiler reports error (I don't know why) but still it's working
for (i in 1..100) {
    print(i % 5 == 0 ? i % 3 == 0 ? "FizzBuzz" : "Buzz" : i % 3 == 0 ? "Fizz" : i)
}
```

Groovy

```
for (i in 1..100) {
  println "${i%3?'':'Fizz'}${i%5?'':'Buzz'}" ?: i
}
```

Haskell

Variant directly implementing the specification:

```
main = mapM_ (putStrLn . fizzbuzz) [1..100]
```

```
fizzbuzz x
    | x `mod` 15 == 0 = "FizzBuzz"
    | x `mod`  3 == 0 = "Fizz"
    | x `mod`  5 == 0 = "Buzz"
    | otherwise = show x
```

```
main = putStr $ concat $ map fizzbuzz [1..100]

fizzbuzz n =
    "\n" ++ if null (fizz++buzz) then show n else fizz++buzz
    where fizz = if mod n 3 == 0 then "Fizz" else ""
          buzz = if mod n 5 == 0 then "Buzz" else ""
```

Does not perform the mod 15 step, extesible to arbitrary addtional tests, ex: [bar| n `mod` 7 == 0].

```
main = mapM_ (putStrLn . fizzbuzz) [1..100]

fizzbuzz n =
    show n <|> [fizz| n `mod` 3 == 0] ++
               [buzz| n `mod` 5 == 0]

-- A simple default choice operator.
-- Defaults if both fizz and buzz fail, concats if any succeed.
infixr 0 <|>
d <|> [] = d
_ <|> x = concat x

fizz = "Fizz"
buzz = "Buzz"
```

Alternate implementation using lazy infinite lists and avoiding use of "mod":

```
main = mapM_ putStrLn $ take 100 $ zipWith show_number_or_fizzbuzz [1..] fizz_buzz_list

show_number_or_fizzbuzz x y = if null y then show x else y

fizz_buzz_list = zipWith (++) (cycle ["","","Fizz"]) (cycle ["","","","","Buzz"])
```

Using heavy artillery (needs the mtl package):

```
import Control.Monad.State
import Control.Monad.Trans
import Control.Monad.Writer

main = putStr $ execWriter $ mapM_ (flip execStateT True . fizzbuzz) [1..100]

fizzbuzz :: Int -> StateT Bool (Writer String) ()
fizzbuzz x = do
 when (x `mod` 3 == 0) $ tell "Fizz" >> put False
 when (x `mod` 5 == 0) $ tell "Buzz" >> put False
 get >>= (flip when $ tell $ show x)
 tell "\n"
```

Using guards plus where.

```
fizzBuzz :: (Integral a) => a -> String
fizzBuzz i
  | fizz && buzz = "FizzBuzz"
```

```
  | fizz         = "Fizz"
  | buzz         = "Buzz"
  | otherwise    = show i
  where fizz = i `mod` 3 == 0
        buzz = i `mod` 5 == 0

main = mapM_ (putStrLn . fizzBuzz) [1..100]
```

HicEst

```
DO i = 1, 100
  IF(     MOD(i, 15) == 0 ) THEN
    WRITE() "FizzBuzz"
  ELSEIF( MOD(i, 5) == 0 ) THEN
    WRITE() "Buzz"
  ELSEIF( MOD(i, 3) == 0 ) THEN
    WRITE() "Fizz"
  ELSE
    WRITE() i
  ENDIF
ENDDO
```

Alternatively:

```
CHARACTER string*8

DO i = 1, 100
  string = " "
  IF( MOD(i, 3) == 0 ) string = "Fizz"
  IF( MOD(i, 5) == 0 ) string = TRIM(string) // "Buzz"
  IF( string == " ") WRITE(Text=string) i
  WRITE() string
ENDDO
```

Icon and Unicon

```
# straight-forward modulo tester
procedure main()
    every i := 1 to 100 do
        if i % 15 = 0 then
            write("FizzBuzz")
        else if i % 5 = 0 then
            write("Buzz")
        else if i % 3 = 0 then
            write("Fizz")
        else
            write(i)
end
```

```
# idiomatic modulo tester, 1st alternative
procedure main()
    every i := 1 to 100 do
        write((i % 15 = 0 & "FizzBuzz") | (i % 5 = 0 & "Buzz") | (i % 3 = 0 & "Fizz") | i)
end
```

```
# idiomatic modulo tester, 2nd alternative
procedure main()
    every i := 1 to 100 do
        write(case 0 of {
                  i % 15 : "FizzBuzz"
```

```
                i % 5  : "Buzz"
                i % 3  : "Fizz"
                default: i
        })
end
```

```
# straight-forward buffer builder
procedure main()
    every i := 1 to 100 do {
        s := ""
        if i % 3 = 0 then
            s ||:= "Fizz"
        if i % 5 = 0 then
            s ||:= "Buzz"
        if s == "" then
            s := i
        write(s)
    }
end
```

```
# idiomatic buffer builder, 1st alternative
procedure main()
    every i := 1 to 100 do
        write("" ~=== (if i % 3 = 0 then "Fizz" else "") || (if i % 5 == 0 then "Buzz" else "") |
end
```

```
# idiomatic buffer builder, 2nd alternative
procedure main()
    every i := 1 to 100 do {
        s   := if i%3 = 0 then "Fizz" else ""
        s ||:= if i%5 = 0 then "Buzz"
        write(("" ~= s) | i)
    }
end
```

Inform 6

```
[ Main i;
  for(i = 1: i <= 100: i++)
  {
    if(i % 3 == 0) print "Fizz";
    if(i % 5 == 0) print "Buzz";
    if(i % 3 ~= 0 && i % 5 ~= 0) print i;

    print "^";
  }
];
```

Inform 7

```
Home is a room.

When play begins:
        repeat with N running from 1 to 100:
                let printed be false;
                if the remainder after dividing N by 3 is 0:
                        say "Fizz";
                        now printed is true;
                if the remainder after dividing N by 5 is 0:
                        say "Buzz";
```

```
                    now printed is true;
                if printed is false, say N;
                say ".";
        end the story.
```

Io

Here's one way to do it:

```
for(a,1,100,
   if(a % 15 == 0) then(
      "FizzBuzz" println
   ) elseif(a % 3 == 0) then(
      "Fizz" println
   ) elseif(a % 5 == 0) then(
      "Buzz" println
   ) else (
      a println
   )
)
```

And here's a port of the Ruby version, which I personally prefer:

```
a := 0; b := 0
for(n, 1, 100,
    if(a = (n % 3) == 0, "Fizz" print);
    if(b = (n % 5) == 0, "Buzz" print);
    if(a not and b not, n print);
    "\n" print
)
```

And here is another more idiomatic version:

```
for (n, 1, 100,
    fb := list (
        if (n % 3 == 0, "Fizz"),
        if (n % 5 == 0, "Buzz")) select (isTrue)

    if (fb isEmpty, n, fb join) println
)
```

Ioke

```
(1..100) each(x,
  cond(
    (x % 15) zero?, "FizzBuzz" println,
    (x % 3) zero?, "Fizz" println,
    (x % 5) zero?, "Buzz" println
  )
)
```

Iptscrae

Written in Iptscrae, the scripting language for The Palace chat software.

```
; FizzBuzz in Iptscrae
1 a =
{
   "" b =
   { "fizz" b &= } a 3 % 0 == IF
   { "buzz" b &= } a 5 % 0 == IF
   { a ITOA LOGMSG } { b LOGMSG } b STRLEN 0 == IFELSE
   a ++
}
{ a 100 <= } WHILE
```

J

Solution _1: Using agenda (@.) as a switch:

```
   test =: +/@(1 2 * 0 = 3 5&|~)
   (":@]`('Fizz'"_)`('Buzz'"_)`('FizzBuzz'"_) @. test"0)  >:i.100
```

Solution 0

```
> }. (<'FizzBuzz') (I.0=15|n)} (<'Buzz') (I.0=5|n)} (<'Fizz') (I.0=3|n)} ":&.> n=: i.101
```

Solution 1

```
Fizz=: 'Fizz' #~ 0 = 3&|
Buzz=: 'Buzz' #~ 0 = 5&|
FizzBuzz=: ": [^:('' -: ]) Fizz,Buzz

FizzBuzz"0 >: i.100
```

Solution 2 (has taste of table-driven template programming)

```
CRT0=: 2 : ' (, 0 = +./)@(0 = m | ]) ;@# n , <@": '
NB. Rather (, 0 = +./) than (, +:/) because designed for
NB. 3 5 7 CRT0 (;:'Chinese Remainder Period') "0 >: i. */3 5 7
FizzBuzz=: 3 5 CRT0 (;:'Fizz Buzz')

FizzBuzz"0 >: i.100
```

Solution 3 (depends on an obsolete feature of @ in f`g`h@p)

```
'`f   b   fb'  =: ('Fizz'"_) ` ('Buzz'"_) ` (f , b)
'`cm3 cm5 cm15'=: (3&|)      ` (5&|)      ` (15&|)  (0&=@)
FizzBuzz=: ": ` f @. cm3 ` b @. cm5 ` fb @. cm15  NB. also:
FizzBuzz=: ": ` f @. cm3 ` b @. cm5 ` (f,b) @. (cm3 *. cm5)

FizzBuzz"0 >: i.100
```

Java

If/else ladder

```java
public class FizzBuzz{
	public static void main(String[] args){
		for(int i= 1; i <= 100; i++){
			if(i % 15 == 0){
				System.out.println("FizzBuzz");
			}else if(i % 3 == 0){
				System.out.println("Fizz");
			}else if(i % 5 == 0){
				System.out.println("Buzz");
			}else{
				System.out.println(i);
			}
		}
	}
}
```

Concatenation

```java
public class FizzBuzz{
	public static void main(String[] args){
		for(int i= 1; i <= 100; i++){
			String output = "";
			if(i % 3 == 0) output += "Fizz";
			if(i % 5 == 0) output += "Buzz";
			if(output.equals("")) output += i;
			System.out.println(output);
		}
	}
}
```

Ternary operator

```java
public class FizzBuzz{
	public static void main(String[] args){
		for(int i= 1; i <= 100; i++){
			System.out.println(i % 15 != 0 ? i % 5 != 0 ? i % 3 != 0 ?
			i : "Fizz" : "Buzz" : "FizzBuzz");
		}
	}
}
```

Recursive

```java
public String fizzBuzz(int n){
  String s = "";
  if (n == 0)
    return s;
  if((n % 5) == 0)
    s = "Buzz" + s;
  if((n % 3) == 0)
    s = "Fizz" + s;
  if (s.equals(""))
    s = n + "";
  return fizzBuzz(n-1) +  s;
}
```

Alternative Recursive

```java
public String fizzBuzz(int n){
```

```
  return (n>0) ? fizzBuzz(n-1) +
    (n % 15 != 0? n % 5 != 0? n % 3 != 0? (n+"") :"Fizz" : "Buzz" : "FizzBuzz")
                : "";
}
```

Using an array

```
class FizzBuzz {
  public static void main( String [] args ) {
    for( int i = 1 ; i <= 100 ; i++ ) {
      System.out.println( new String[]{ i+"", "Fizz", "Buzz", "FizzBuzz" }[ ( i%3==0?1:0 ) + ( i
    }
  }
}
```

JavaScript

Works with: NJS version 0.2.5

```
for (var i = 1; i <= 100; i++) {
  if (i % 15 == 0) {
    console.log("FizzBuzz");
  } else if (i % 3 == 0) {
    console.log("Fizz");
  } else if (i % 5 == 0) {
    console.log("Buzz");
  } else {
    console.log(i);
  }
}

Array.apply(null, { length: 100 }).map(function(n, i) {
  ++i;
  return !(i % 15) ?
    "FizzBuzz" :
    !(i % 3) ?
      "Fizz" :
      !(i % 5) ?
        "Buzz" : i;
}).join("\r\n");
```

Alternative version (one-liner)

```
for (var i=1; i<=100; i++) console.log( (i % 3 === 0 ? 'Fizz' : '') + (i % 5 === 0 ? 'Buzz' : ''
```

Bodyless for loop

```
for(var i=1; i<=100; console.log((i%3?'':'Fizz')+(i%5?'':'Buzz')||i), i++);
```

A little shorter

```
for(i=0;i<100;console.log(++i%15?i%5?i%3?i:f='Fizz':b='Buzz':f+b));
```

Compiled from CoffeeScript One-Liner

```
(function() {
  var i;

  for (i = 1; i <= 100; i++) {
    console.log([i % 3 === 0 ? 'Fizz' : void 0] + [i % 5 === 0 ? 'Buzz' : void 0] || i);
  }

}).call(this);
```

Zombie Version

Zombie -> decomposed.

Tries to avoid 'wall of code' by decomposition, creation of a solution-based language, generalization.

"Simplicity, clarity, generality" -- Pike & Kernighan

Runs under SpiderMonkey.

```
var divs = [15, 3, 5];
var says = ['FizzBuzz', 'Fizz', 'Buzz'];

function fizzBuzz(first, last) {
    for (var n = first; n <= last; n++) {
        print(getFizzBuzz(n));
    }
}

function getFizzBuzz(n) {
    var sayWhat = n;
    for (var d = 0; d < divs.length; d++) {
        if (isMultOf(n, divs[d])) {
            sayWhat = says[d];
            break;
        }
    }
    return sayWhat;
}

function isMultOf(n, d) {
    return n % d == 0;
}

fizzBuzz(1, 100);
```

Notice: This is satire. (Isn't it?)

Joy

The following program first defines a function "one", which handles the Fizz / Buzz logic, and then loops from 1 to 100 mapping the function onto each number, and printing ("put") the output.

```
DEFINE one == [[[dup 15 rem 0 =] "FizzBuzz"] [[dup 3 rem 0 =] "Fizz"] [[dup 5 rem 0 =] "Buzz"] [
1 [100 <=] [dup one put succ] while.
```

Julia

```
for i = 1:100
    if i % 15 == 0
        println("FizzBuzz")
    elseif i % 3 == 0
        println("Fizz")
    elseif i % 5 == 0
        println("Buzz")
    else
        println(i)
    end
end
```

Another solution

```
println( [ i%15 == 0? "FizzBuzz" :
           i%5  == 0? "Buzz" :
           i%3  == 0? "Fizz" :
           i for i = 1:100 ] )
```

Yet another solution

```
fb(i::Int) = "Fizz" ^ (i%3==0) *
             "Buzz" ^ (i%5==0) *
             string(i) ^ (i%3!=0 && i%5!=0)
[println(fb(i)) for i = 1:100];
```

K

```
`0:\:{:[0=#a:{,/$(:[0=x!3;"Fizz"];:[0=x!5;"Buzz"])}@x;$x;a]}'1_!101
```

Kamailio Script

To run it, send a SIP message to the server and FizzBuzz will appear in the logs.

This will only work up to 100 because Kamailio terminates all while loops after 100 iterations.

```
# FizzBuzz
log_stderror=yes
loadmodule "pv"
loadmodule "xlog"

route {
    $var(i) = 1;
    while ($var(i) <= 1000) {
        if ($var(i) mod 15 == 0) {
            xlog("FizzBuzz\n");
        } else if ($var(i) mod 5 == 0) {
            xlog("Buzz\n");
        } else if ($var(i) mod 3 == 0) {
            xlog("Fizz\n");
        } else {
            xlog("$var(i)\n");
        }
        $var(i) = $var(i) + 1;
    }
```

```
}
```

Kaya

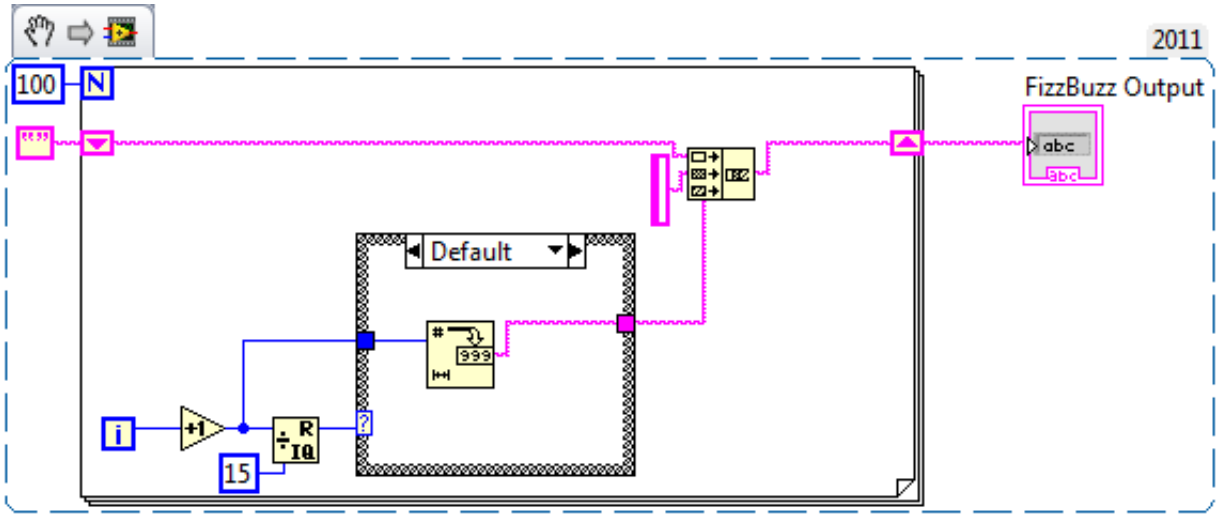
```
// fizzbuzz in Kaya
program fizzbuzz;

Void fizzbuzz(Int size) {
    for i in [1..size] {
        if (i % 15 == 0) {
            putStrLn("FizzBuzz");
        } else if (i % 5 == 0) {
            putStrLn("Buzz");
        } else if (i % 3 == 0) {
            putStrLn("Fizz");
        } else {
            putStrLn( string(i) );
        }
    }
}

Void main() {
    fizzbuzz(100);
}
```

LabVIEW

This image is a VI Snippet (http://zone.ni.com/devzone/cda/tut/p/id/9330) , an executable image of LabVIEW code. The LabVIEW version is shown on the top-right hand corner. You can download it, then drag-and-drop it onto the LabVIEW block diagram from a file browser, and it will appear as runnable, editable code.



Lasso

```
with i in generateSeries(1, 100)
select ((#i % 3 == 0 ? 'Fizz' | '') + (#i % 5 == 0 ? 'Buzz' | '') || #i)
```

LaTeX

Library: ifthen
Library: intcalc

This version uses the ifthen and intcalc packages. There sure are more native solutions including solutions in plain TeX, but for me this is a readable and comprehensible one.

```
\documentclass{minimal}
\usepackage{ifthen}
\usepackage{intcalc}

\newcounter{mycount}
\newboolean{fizzOrBuzz}

\newcommand\fizzBuzz[1]{%
\setcounter{mycount}{1}\whiledo{\value{mycount}<#1}
    {
        \setboolean{fizzOrBuzz}{false}
        \ifthenelse{\equal{\intcalcMod{\themycount}{3}}{0}}{\setboolean{fizzOrBuzz}{true}Fizz}{}
        \ifthenelse{\equal{\intcalcMod{\themycount}{5}}{0}}{\setboolean{fizzOrBuzz}{true}Buzz}{}
        \ifthenelse{\boolean{fizzOrBuzz}}{}{\themycount}
        \stepcounter{mycount}
        \\
    }
}

\begin{document}
    \fizzBuzz{101}
\end{document}
```

Liberty BASIC

```
for i = 1 to 100
    select case
        case i mod 15 = 0
            print "FizzBuzz"
        case i mod 3 = 0
            print "Fizz"
        case i mod 5 = 0
            print "Buzz"
        case else
            print i
    end select
next i
```

LiveScript

See: http://livescript.net/blog/fizzbuzzbazz.html

```
[1 to 100]map ->[k+\zz for k,v of{Fi:3,Bu:5}|it%v<1]*''||it
```

Logo

```
to fizzbuzz :n
  output cond [ [[equal? 0 modulo :n 15] "FizzBuzz]
                [[equal? 0 modulo :n  5] "Buzz]
                [[equal? 0 modulo :n  3] "Fizz]
                [else :n] ]
```

```
end

repeat 100 [print fizzbuzz #]
```

"cond" was undefined in Joshua Bell's online interpreter. So here is a version that works there. It also works in UCB logo by using # instead of "repcount". This version also factors away modulo 15:

```
to fizzbuzz :n
 make "c "
  if equal? 0 modulo :n 5 [make "c "Buzz]
  if equal? 0 modulo :n 3 [make "c word "Fizz :c]
 output ifelse equal? :c " [:n] [:c]
end

repeat 100 [print fizzbuzz repcount]
```

Lhogho can use the above code, except that 'modulo' must be replaced with 'remainder'.

LOLCODE

```
HAI 1.3

IM IN YR fizz UPPIN YR i TIL BOTH SAEM i AN 100
    I HAS A i ITZ SUM OF i AN 1 BTW, ALL LUPZ START AT 0 :/
    I HAS A mod3 ITZ NOT MOD OF i AN 3
    I HAS A mod5 ITZ NOT MOD OF i AN 5

    mod3, O RLY?, YA RLY, VISIBLE "Fizz"!, OIC
    mod5, O RLY?, YA RLY, VISIBLE "Buzz"!, OIC

    NOT EITHER OF mod3 AN mod5, O RLY?
        YA RLY, VISIBLE i!
    OIC

    VISIBLE ""
IM OUTTA YR fizz

KTHXBYE
```

LSE

```
1* FIZZBUZZ en L.S.E.
10 CHAINE FB
20 FAIRE 45 POUR I_1 JUSQUA 100
30 FB_SI &MOD(I,3)=0 ALORS SI &MOD(I,5)=0 ALORS 'FIZZBUZZ' SINON 'FIZZ' SINON SI &MOD(I,5)=0 ALO
40 AFFICHER[U,/] SI FB='' ALORS I SINON FB
45*FIN BOUCLE
50 TERMINER
100 PROCEDURE &MOD(A,B) LOCAL A,B
110 RESULTAT A-B*ENT(A/B)
```

Lua

If/else Ladder

```
for i = 1, 100 do
```

```
        if i % 15 == 0 then
                print("FizzBuzz")
        elseif i % 3 == 0 then
                print("Fizz")
        elseif i % 5 == 0 then
                print("Buzz")
        else
                print(i)
        end
end
```

Concatenation

```
for i = 1, 100 do
        output = ""
        if i % 3 == 0 then
                output = output.."Fizz"
        end
        if i % 5 == 0 then
                output = output.."Buzz"
        end
        if(output == "") then
                output = output..i
        end
        print(output)
end
```

Quasi bit field

```
word = {"Fizz", "Buzz", "FizzBuzz"}

for i = 1, 100 do
        print(word[(i % 3 == 0 and 1 or 0) + (i % 5 == 0 and 2 or 0)] or i)
end
```

M4

```
define(`for',
   `ifelse($#,0,``$0'',
   `ifelse(eval($2<=$3),1,
   `pushdef(`$1',$2)$5`'popdef(`$1')$0(`$1',eval($2+$4),$3,$4,`$5')')')')

for(`x',1,100,1,
   `ifelse(eval(x%15==0),1,FizzBuzz,
   `ifelse(eval(x%3==0),1,Fizz,
   `ifelse(eval(x%5==0),1,Buzz,x)')')
')
```

make

Works with: BSD make
Library: jot

```
MOD3 = 0
MOD5 = 0
ALL != jot 100

all: say-100
```

```
.for NUMBER in $(ALL)

MOD3 != expr \( $(MOD3) + 1 \) % 3; true
MOD5 != expr \( $(MOD5) + 1 \) % 5; true

. if "$(NUMBER)" > 1
PRED != expr $(NUMBER) - 1
say-$(NUMBER): say-$(PRED)
. else
say-$(NUMBER):
. endif
. if "$(MOD3)$(MOD5)" == "00"
        @echo FizzBuzz
. elif "$(MOD3)" == "0"
        @echo Fizz
. elif "$(MOD5)" == "0"
        @echo Buzz
. else
        @echo $(NUMBER)
. endif

.endfor
```

Mathematica

```
Do[Print[Which[Mod[i, 15] == 0, "FizzBuzz", Mod[i, 5] == 0, "Buzz", Mod[i, 3] == 0, "Fizz", True
```

Using rules,

```
fizz[i_] := Mod[i, 3] == 0
buzz[i_] := Mod[i, 5] == 0
Range[100] /. {i_ /; fizz[i]&&buzz[i] :> "FizzBuzz", \
               i_?fizz :> "Fizz", i_?buzz :> "Buzz"}
```

Using rules, but different approach:

```
SetAttributes[f,Listable]
f[n_ /; Mod[n, 15] == 0] := "FizzBuzz";
f[n_ /; Mod[n, 3] == 0] := "Fizz";
f[n_ /; Mod[n, 5] == 0] := "Buzz";
f[n_] := n;

f[Range[100]]
```

An extendible version using Table

```
Table[If[# === "", i, #]&@StringJoin[
   Table[If[Divisible[i, First@nw], Last@nw, ""],
         {nw, {{3, "Fizz"}, {5, "Buzz"}}}]],
     {i, 1, 100}]
```

Another one-liner using Map (the /@ operator shorthand of it) and a pure function with a very readable Which

```
 Which[ Mod[#,15] == 0, "FizzBuzz", Mod[#, 3] == 0, "Fizz", Mod[#,5]==0, "Buzz",  True, #]& /@ R
```

MATLAB

There are more sophisticated solutions to this task, but in the spirit of "lowest level of comprehension required to illustrate adequacy" this is what one might expect from a novice programmer (with a little variation in how the strings are stored and displayed).

```
function fizzBuzz()
    for i = (1:100)
        if mod(i,15) == 0
           fprintf('FizzBuzz ')
        elseif mod(i,3) == 0
           fprintf('Fizz ')
        elseif mod(i,5) == 0
           fprintf('Buzz ')
        else
           fprintf('%i ',i))
        end
    end
    fprintf('\n');
end
```

Here's a more extendible version that uses disp() to print the output:

```
function out = fizzbuzzS()
        nums = [3, 5];
        words = {'fizz', 'buzz'};
        for (n=1:100)
                tempstr = '';
                for (i = 1:2)
                        if mod(n,nums(i))==0
                                tempstr = [tempstr,  words{i}];
                        end
                end
                if length(tempstr) == 0
                        disp(n);
                else
                        disp(tempstr);
                end
        end
end
```

Maxima

```
for n thru 100 do
   if mod(n, 15) = 0 then disp("FizzBuzz")
   elseif mod(n, 3) = 0 then disp("Fizz")
   elseif mod(n,5) = 0 then disp("Buzz")
   else disp(n);
```

MAXScript

```
for i in 1 to 100 do
(
    case of
    (
        (mod i 15 == 0): (print "FizzBuzz")
        (mod i 5 == 0):  (print "Buzz")
```

```
        (mod i 3 == 0):  (print "Fizz")
        default:         (print i)
    )
)
```

MEL

```
for($i=1; $i<=100; $i++)
{
    if($i % 15 == 0)
        print "FizzBuzz\n";
    else if ($i % 3 == 0)
        print "Fizz\n";
    else if ($i % 5 == 0)
        print "Buzz\n";
    else
        print ($i + "\n");
}
```

Mercury

```
:- module fizzbuzz.
:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.
:- implementation.
:- import_module int, string, bool.

:- func fizz(int) = bool.
:- func buzz(int) = bool.
fizz(N) = ( if N mod 3 = 0 then yes else no ).
buzz(N) = ( if N mod 5 = 0 then yes else no ).

:- pred fizzbuzz(int::in, bool::in, bool::in, string::out) is det.
%           3?   5?
fizzbuzz(_, yes, yes, "FizzBuzz").
fizzbuzz(_, yes, no,  "Fizz").
fizzbuzz(_, no,  yes, "Buzz").
fizzbuzz(N, no,  no,  S) :- S = from_int(N).

main(!IO) :- main(1, 100, !IO).

:- pred main(int::in, int::in, io::di, io::uo) is det.
main(N, To, !IO) :-
        io.write_string(S, !IO), io.nl(!IO),
        fizzbuzz(N, fizz(N), buzz(N), S),
        ( if N < To then main(N + 1, To, !IO) else !:IO = !.IO ).
```

Metafont

```
for i := 1 upto 100:
message if i mod 15 = 0: "FizzBuzz" &
elseif i mod 3 = 0: "Fizz" &
elseif i mod 5 = 0: "Buzz" &
else: decimal i & fi "";
endfor
end
```

Mirah

```
1.upto(100) do |n|
    print "Fizz" if a = ((n % 3) == 0)
    print "Buzz" if b = ((n % 5) == 0)
    print n unless (a || b)
    print "\n"
end

# a little more straight forward
1.upto(100) do |n|
    if (n % 15) == 0
        puts "FizzBuzz"
    elsif (n % 5) == 0
        puts "Buzz"
    elsif (n % 3) == 0
        puts "Fizz"
    else
        puts n
    end
end
```

MMIX

```
t   IS $255
Ja  IS $127

        LOC Data_Segment
data    GREG   @

fizz    IS @-Data_Segment
        BYTE "Fizz",0,0,0,0

buzz    IS @-Data_Segment
        BYTE "Buzz",0,0,0,0

nl      IS @-Data_Segment
        BYTE #a,0,0,0,0,0,0,0

buffer IS @-Data_Segment



        LOC #1000
        GREG @

% "usual" print integer subroutine
printnum LOC @
        OR   $1,$0,0
        SETL $2,buffer+64
        ADDU $2,$2,data
        XOR  $3,$3,$3
        STBU $3,$2,1
loop    DIV  $1,$1,10
        GET  $3,rR
        ADDU $3,$3,'0'
        STBU $3,$2,0
        SUBU $2,$2,1
        PBNZ $1,loop
        ADDU t,$2,1
        TRAP 0,Fputs,StdOut
        GO   Ja,Ja,0

Main    SETL $0,1           % i = 1
1H      SETL $2,0           % fizz not taken
        CMP  $1,$0,100      % i <= 100
        BP   $1,4F          % if no, go to end
        DIV  $1,$0,3
        GET  $1,rR          % $1 = mod(i,3)
        CSZ  $2,$1,1        % $2 = Fizz taken?
        BNZ  $1,2F          % $1 != 0? yes, then skip
```

```
       ADDU t,data,fizz
       TRAP 0,Fputs,StdOut % print "Fizz"
2H     DIV  $1,$0,5
       GET  $1,rR          % $1 = mod(i,5)
       BNZ  $1,3F          % $1 != 0? yes, then skip
       ADDU t,data,buzz
       TRAP 0,Fputs,StdOut % print "Buzz"
       JMP  5F             % skip print i
3H     BP   $2,5F          % skip if Fizz was taken
       GO   Ja,printnum    % print i
5H     ADDU t,data,nl
       TRAP 0,Fputs,StdOut % print newline
       ADDU $0,$0,1
       JMP  1B             % repeat for next i
4H     XOR  t,t,t
       TRAP 0,Halt,0       % exit(0)
```

Modula-3

```
MODULE Fizzbuzz EXPORTS Main;

IMPORT IO;

BEGIN
   FOR i := 1 TO 100 DO
      IF i MOD 15 = 0 THEN
         IO.Put("FizzBuzz\n");
      ELSIF i MOD 5 = 0 THEN
         IO.Put("Buzz\n");
      ELSIF i MOD 3 = 0 THEN
         IO.Put("Fizz\n");
      ELSE
         IO.PutInt(i);
         IO.Put("\n");
      END;
   END;
END Fizzbuzz.
```

MUMPS

```
FIZZBUZZ
 NEW I
 FOR I=1:1:100 WRITE !,$SELECT(('(I#3)&'(I#5)):"FizzBuzz",'(I#5):"Buzz",'(I#3):"Fizz",1:I)
 KILL I
 QUIT
```

Nemerle

The naive approach:

```
using System;
using System.Console;

module FizzBuzz
{
    FizzBuzz(x : int) : string
    {
        |x when x % 15 == 0 => "FizzBuzz"
        |x when x %  5 == 0 => "Buzz"
        |x when x %  3 == 0 => "Fizz"
        |_                  => $"$x"
```

```
    }

    Main() : void
    {
        foreach (i in [1 .. 100])
            WriteLine($"$(FizzBuzz(i))")
    }
}
```

A much slicker approach is posted here (http://www.dreamincode.net/forums/blog/217/entry-3539-fizzbuzz-in-nemerle/)

NetRexx

```
loop j=1 for 100
  select
    when j//15==0 then say 'FizzBuzz'
    when j//5==0  then say 'Buzz'
    when j//3==0  then say 'Fizz'
    otherwise say j.right(4)
  end
end
```

NewtonScript

```
for i := 1 to 100 do
begin
        if i mod 15 = 0 then
                print("FizzBuzz")
        else if i mod 3 = 0 then
                print("Fizz")
        else if i mod 5 = 0 then
                print("Buzz")
        else
                print(i);
        print("\n")
end
```

Nickle

```
/* Fizzbuzz in nickle */

void function fizzbuzz(size) {
    for (int i = 1; i < size; i++) {
        if (i % 15 == 0) { printf("Fizzbuzz\n"); }
        else if (i % 5 == 0) { printf("Buzz\n"); }
        else if (i % 3 == 0) { printf("Fizz\n"); }
        else { printf("%i\n", i); }
    }
}

fizzbuzz(1000);
```

Nimrod

Translation of: Python

```
```

```
for i in 1..100:
  if i mod 15 == 0:
    echo("FizzBuzz")
  elif i mod 3 == 0:
    echo("Fizz")
  elif i mod 5 == 0:
    echo("Buzz")
  else:
    echo(i)
```

Without Modulus

```
var messages = @["", "Fizz", "Buzz", "FizzBuzz"]
var acc = 810092048
for i in 1..100:
  var c = acc and 3
  echo(if c == 0: $i else: messages[c])
  acc = acc shr 2 or c shl 28
```

Oberon-2

```
MODULE FizzBuzz;

   IMPORT Out;

   VAR i: INTEGER;

BEGIN
   FOR i := 1 TO 100 DO
      IF i MOD 15 = 0 THEN
         Out.String("FizzBuzz");
         Out.Ln;
      ELSIF i MOD 5 = 0 THEN
         Out.String("Buzz");
         Out.Ln;
      ELSIF i MOD 3 = 0 THEN
         Out.String("Fizz");
         Out.Ln;
      ELSE
         Out.Int(i,0);
         Out.Ln;
      END;
   END;
END FizzBuzz.
```

Objeck

```
bundle Default {
  class Fizz {
    function : Main(args : String[]) ~ Nil {
      for(i := 0; i <= 100; i += 1;) {
        if(i % 15 = 0) {
          "FizzBuzz"->PrintLine();
        }
        else if(i % 3 = 0) {
          "Fizz"->PrintLine();
        }
        else if(i % 5 = 0) {
          "Buzz"->PrintLine();
        }
        else {
          i->PrintLine();
```

```
        };
      };
    }
  }
}
```

Objective-C

```
// FizzBuzz in Objective-C
#import <stdio.h>

main() {
        for (int i=1; i<=100; i++) {
                if (i % 15 == 0) {
                    printf("FizzBuzz\n");
                } else if (i % 3 == 0) {
                    printf("Fizz\n");
                } else if (i % 5 == 0) {
                    printf("Buzz\n");
                } else {
                    printf("%i\n", i);
                }
        }
}
```

OCaml

```
let output x =
  match x mod 3 = 0, x mod 5 = 0 with
    true,  true  -> "FizzBuzz"
  | true,  false -> "Fizz"
  | false, true  -> "Buzz"
  | false, false -> string_of_int x

let _ =
  for i = 1 to 100 do print_endline (output i) done
```

Octave

```
for i = 1:100
  if ( mod(i,15) == 0 )
    disp("FizzBuzz");
  elseif ( mod(i, 3) == 0 )
    disp("Fizz")
  elseif ( mod(i, 5) == 0 )
    disp("Buzz")
  else
    disp(i)
  endif
endfor
```

OOC

```
fizz: func (n: Int) -> Bool {
  if(n % 3 == 0) {
    printf("Fizz")
    return true
  }
```

```
  return false
}

buzz: func (n: Int) -> Bool {
  if(n % 5 == 0) {
    printf("Buzz")
    return true
  }
  return false
}

main: func {
  for(n in 1..100) {
    fizz:= fizz(n)
    buzz:= buzz(n)
    fizz || buzz || printf("%d", n)
    println()
  }
}
```

Order

```
#include <order/interpreter.h>

// Get FB for one number
#define ORDER_PP_DEF_8fizzbuzz ORDER_PP_FN(            \
8fn(8N,                                                \
    8let((8F, 8fn(8N, 8G,                              \
                  8is_0(8remainder(8N, 8G)))),         \
         8cond((8ap(8F, 8N, 15), 8quote(fizzbuzz))     \
               (8ap(8F, 8N, 3), 8quote(fizz))          \
               (8ap(8F, 8N, 5), 8quote(buzz))          \
               (8else, 8N)))) )

// Print E followed by a comma (composable, 8print is not a function)
#define ORDER_PP_DEF_8print_el ORDER_PP_FN( \
8fn(8E, 8print(8E 8comma)) )

ORDER_PP(  // foreach instead of map, to print but return nothing
  8seq_for_each(8compose(8print_el, 8fizzbuzz), 8seq_iota(1, 100))
)
```

Oz

```
declare
  fun {FizzBuzz X}
     if     X mod 15 == 0 then 'FizzBuzz'
     elseif X mod  3 == 0 then 'Fizz'
     elseif X mod  5 == 0 then 'Buzz'
     else                      X
     end
  end
in
  for I in 1..100 do
     {Show {FizzBuzz I}}
  end
```

PARI/GP

```
{for(n=1,100,
  print(if(n%3,
    if(n%5,
```

```
      n
    ,
      "Buzz"
    )
  ,
    if(n%5,
      "Fizz"
    ,
      "FizzBuzz"
    )
  ))
)}
```

Pascal

```
program fizzbuzz(output);
var
  i: integer;
begin
  for i := 1 to 100 do
    if i mod 15 = 0 then
      writeln('FizzBuzz')
    else if i mod 3 = 0 then
      writeln('Fizz')
    else if i mod 5 = 0 then
      writeln('Buzz')
    else
      writeln(i)
end.
```

Perl

```
use strict;
use warnings;
use feature qw(say);

for my $i (1..100) {
    say $i % 15 == 0 ? "FizzBuzz"
      : $i %  3 == 0 ? "Fizz"
      : $i %  5 == 0 ? "Buzz"
      : $i;
}
```

More concisely:

```
print 'Fizz'x!($_ % 3) . 'Buzz'x!($_ % 5) || $_, "\n" for 1 .. 100;
```

For code-golfing:

```
print+(Fizz)[$_%3].(Buzz)[$_%5]||$_,$/for 1..1e2
```

For array of values:

```
map((Fizz)[$_%3].(Buzz)[$_%5]||$_,1..100);
```

Cheating:

```
use feature "say";

@a = ("FizzBuzz", 0, 0, "Fizz", 0, "Buzz", "Fizz", 0, 0, "Fizz", "Buzz", 0, "Fizz");

say $a[$_ % 15] || $_ for 1..100;
```

Perl 6

Works with: Rakudo Star version 2010-08

Most straightforwardly:

```
for 1 .. 100 {
    when $_ %% (3 & 5) { say 'FizzBuzz'; }
    when $_ %% 3       { say 'Fizz'; }
    when $_ %% 5       { say 'Buzz'; }
    default            { .say; }
}
```

Or abusing multi subs:

```
multi sub fizzbuzz(Int $ where * %% 15) { 'FizzBuzz' }
multi sub fizzbuzz(Int $ where * %%  5) { 'Buzz' }
multi sub fizzbuzz(Int $ where * %%  3) { 'Fizz' }
multi sub fizzbuzz(Int $number        ) { $number }
(1 .. 100)».&fizzbuzz.join("\n").say;
```

Most concisely:

```
say 'Fizz' x $_ %% 3 ~ 'Buzz' x $_ %% 5 || $_ for 1 .. 100;
```

And here's an implementation that never checks for divisibility:

```
.say for
  (('' xx 2, 'Fizz') xx * Z~
   ('' xx 4, 'Buzz') xx *) Z||
  1 .. 100;
```

PHL

Translation of: C

```
module fizzbuzz;

extern printf;

@Integer main [
        var i = 1;
        while (i <= 100) {
                if (i % 15 == 0)
                    printf("FizzBuzz");
```

```
                else if (i % 3 == 0)
                    printf("Fizz");
                else if (i % 5 == 0)
                    printf("Buzz");
                else
                    printf("%d", i);

                printf("\n");
                i = i::inc;
        }

        return 0;
]
```

PHP

if/else ladder approach

```php
<?php
for ($i = 1; $i <= 100; $i++)
{
    if (!($i % 15))
        echo "FizzBuzz\n";
    else if (!($i % 3))
        echo "Fizz\n";
    else if (!($i % 5))
        echo "Buzz\n";
    else
        echo "$i\n";
}
?>
```

concatenation approach

Uses PHP's concatenation operator (.=) to build the output string. The concatenation operator allows us to add data to the end of a string without overwriting the whole string. Since Buzz will always appear if our number is divisible by five, and Buzz is the second part of "FizzBuzz", we can simply append "Buzz" to our string.

In contrast to the if-else ladder, this method lets us skip the check to see if $i is divisible by both 3 and 5 (i.e. 15). However, we get the added complexity of needing to reset $str to an empty string (not necessary in some other languages), and we also need a separate if statement to check to see if our string is empty, so we know if $i was not divisible by 3 or 5.

```php
<?php
for ( $i = 1; $i <= 100; ++$i )
{
     $str = "";

     if (!($i % 3 ) )
          $str .= "Fizz";

     if (!($i % 5 ) )
          $str .= "Buzz";

     if ( empty( $str ) )
          $str = $i;

     echo $str . "\n";
}
?>
```

One Liner Approach

```
<?php
for($i = 1; $i <= 100 and print(($i % 15 ? $i % 5 ? $i % 3 ? $i : 'Fizz' : 'Buzz' : 'FizzBuzz')
?>
```

PicoLisp

We could simply use 'at (http://software-lab.de/doc/refA.html#at) ' here:

```
(for N 100
   (prinl
      (or (pack (at (0 . 3) "Fizz") (at (0 . 5) "Buzz")) N) ) )
```

Or do it the standard way:

```
(for N 100
   (prinl
      (cond
         ((=0 (% N 15)) "FizzBuzz")
         ((=0 (% N 3)) "Fizz")
         ((=0 (% N 5)) "Buzz")
         (T N) ) ) )
```

Pike

```
int main(){
   for(int i = 1; i <= 100; i++) {
      if(i % 15 == 0) {
         write("FizzBuzz\n");
      } else if(i % 3 == 0) {
         write("Fizz\n");
      } else if(i % 5 == 0) {
         write("Buzz\n");
      } else {
         write(i + "\n");
      }
   }
}
```

PIR

Works with: Parrot version tested with 2.4.0

```
.sub main :main
  .local int f
  .local int mf
  .local int skipnum
  f = 1
LOOP:
  if f > 100 goto DONE
  skipnum = 0
  mf = f % 3
```

```
  if mf == 0 goto FIZZ
FIZZRET:
  mf = f % 5
  if mf == 0 goto BUZZ
BUZZRET:
  if skipnum > 0 goto SKIPNUM
  print f
SKIPNUM:
  print "\n"
  inc f
  goto LOOP
  end
FIZZ:
  print "Fizz"
  inc skipnum
  goto FIZZRET
  end
BUZZ:
  print "Buzz"
  inc skipnum
  goto BUZZRET
  end
DONE:
  end
.end
```

PL/I

```
do i = 1 to 100;
   select;
      when (mod(i,15) = 0) put skip list ('FizzBuzz');
      when (mod(i,3)  = 0) put skip list ('Fizz');
      when (mod(i,5)  = 0) put skip list ('Buzz');
      otherwise put skip list (i);
   end;
end;
```

Pop11

```
lvars str;
for i from 1 to 100 do
  if i rem 15 = 0 then
    'FizzBuzz' -> str;
  elseif i rem 3 = 0 then
    'Fizz' -> str;
  elseif i rem 5 = 0 then
    'Buzz' -> str;
  else
    '' >< i -> str;
  endif;
  printf(str, '%s\n');
endfor;
```

PL/SQL

```
CREATE OR REPLACE PROCEDURE FIZZBUZZ AS
  i NUMBER;
BEGIN

  FOR i IN 1 .. 100 LOOP
    IF MOD(i, 15) = 0 THEN
      DBMS_OUTPUT.PUT_LINE('FizzBuzz');
```

```
    ELSIF MOD(i, 5) = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Buzz');
    ELSIF MOD(i, 3) = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Fizz');
    ELSE
      DBMS_OUTPUT.PUT_LINE(i);
    END IF;
  END LOOP;

END FIZZBUZZ;
```

PostScript

```
1 1 100 {
        /c false def
        dup 3 mod 0 eq { (Fizz) print /c true def } if
        dup 5 mod 0 eq { (Buzz) print /c true def } if
    c {pop}{(    ) cvs print} ifelse
    (\n) print
} for
```

or...

```
/fizzdict 100 dict def
fizzdict begin
/notmod{ (    ) cvs } def
/mod15 { dup 15 mod 0 eq { (FizzBuzz)def }{pop}ifelse} def
/mod3  { dup 3 mod 0 eq {(Fizz)def}{pop}ifelse} def
/mod5  { dup 5 mod 0 eq {(Buzz)def}{pop}ifelse} def
1 1 100 { mod3 } for
1 1 100 { mod5 } for
1 1 100 { mod15} for
1 1 100 { dup currentdict exch known { currentdict exch get}{notmod} ifelse print (\n) print} fo
end
```

Potion

```
1 to 100 (a):
  if (a % 15 == 0):
    'FizzBuzz'.
  elsif (a % 3 == 0):
    'Fizz'.
  elsif (a % 5 == 0):
    'Buzz'.
  else: a. string print
  "\n" print.
```

PowerShell

Straightforward, looping

```
for ($i = 1; $i -le 100; $i++) {
    if ($i % 15 -eq 0) {
        "FizzBuzz"
    } elseif ($i % 5 -eq 0) {
        "Buzz"
```

```
    } elseif ($i % 3 -eq 0) {
        "Fizz"
    } else {
        $i
    }
}
```

Pipeline, Switch

```
$txt=$null
1..100 | ForEach-Object {
    switch ($_) {
        { $_ % 3 -eq 0 }  { $txt+="Fizz" }
        { $_ % 5 -eq 0 }  { $txt+="Buzz" }
        $_                { if($txt) { $txt } else { $_ }; $txt=$null }
    }
}
```

Concatenation

Translation of: C#

```
1..100 | ForEach-Object {
    $s = ''
    if ($_ % 3 -eq 0) { $s += "Fizz" }
    if ($_ % 5 -eq 0) { $s += "Buzz" }
    if (-not $s) { $s = $_ }
    $s
}
```

Processing

Visualization & Console, Straightforward

Reserved variable "width" in Processing is 100 pixels by default, suitable for this FizzBuzz exercise. Accordingly, range is pixel index from 0 to 99.

```
for (int i = 0; i < width; i++) {
  if (i % 3 == 0 && i % 5 == 0) {
    stroke(255, 255, 0);
    println("FizzBuzz!");
  }
  else if (i % 5 == 0) {
    stroke(0, 255, 0);
    println("Buzz");
  }
  else if (i % 3 == 0) {
    stroke(255, 0, 0);
    println("Fizz");
  }
  else {
    stroke(0, 0, 255);
    println(i);
  }
  line(i, 0, i, height);
}
```

Visualization & Console, Ternary

```
for (int i = 0; i < width; i++) {
  stroke((i % 5 == 0 && i % 3 == 0) ? #FFFF00 : (i % 5 == 0) ? #00FF00 : (i % 3 == 0) ? #FF0000
  line(i, 0, i, height);
  println((i % 5 == 0 && i % 3 == 0) ? "FizzBuzz!" : (i % 5 == 0) ? "Buzz" : (i % 3 == 0) ? "Fiz
}
```

Prolog

Works with: SWI Prolog version 4.8.0

Maybe not the most conventional way to write this in Prolog. The fizzbuzz predicate uses a higher-order predicate and print_item uses the if-then-else construction.

```
fizzbuzz :-
        foreach(between(1, 100, X), print_item(X)).

print_item(X) :-
        (  0 is X mod 15
        -> print('FizzBuzz')
        ;  0 is X mod 3
        -> print('Fizz')
        ;  0 is X mod 5
        -> print('Buzz')
        ;  print(X)
        ),
        nl.
```

More conventional:

```
fizzbuzz(X) :- 0 is X mod 15, write('FizzBuzz').
fizzbuzz(X) :- 0 is X mod 3, write('Fizz').
fizzbuzz(X) :- 0 is X mod 5, write('Buzz').
fizzbuzz(X) :- write(X).

dofizzbuzz :- foreach(between(1, 100, X), (fizzbuzz(X),nl)).
```

Clearer:

```
%        N  /3?  /5?  V
fizzbuzz(_, yes, yes, 'FizzBuzz').
fizzbuzz(_, yes, no,  'Fizz').
fizzbuzz(_, no,  yes, 'Buzz').
fizzbuzz(N, no,  no,  N).

% Unifies V with 'yes' if D divides evenly into N, 'no' otherwise.
divisible_by(N, D, V) :-
  ( 0 is N mod D -> V = yes
  ;                 V = no).

% Print 'Fizz', 'Buzz', 'FizzBuzz' or N as appropriate.
fizz_buzz_or_n(N) :-
  divisible_by(N, 3, Fizz),
  divisible_by(N, 5, Buzz),
  fizzbuzz(N, Fizz, Buzz, FB),
  format("~p -> ~p~n", [N, FB]).

main :-
  foreach(between(1,100, N), fizz_buzz_or_n(N)).
```

Protium

Variable-length padded English dialect

```
<# DEFINE USERDEFINEDROUTINE LITERAL>__FizzBuzz|<# SUPPRESSAUTOMATICWHITESPACE>
<# TEST ISITMODULUSZERO PARAMETER LITERAL>1|3</#>
<# TEST ISITMODULUSZERO PARAMETER LITERAL>1|5</#>
<# ONLYFIRSTOFLASTTWO><# SAY LITERAL>Fizz</#></#>
<# ONLYSECONDOFLASTTWO><# SAY LITERAL>Buzz</#></#>
<# BOTH><# SAY LITERAL>FizzBuzz</#></#>
<# NEITHER><# SAY PARAMETER>1</#></#>
</#></#>
<# ITERATE FORITERATION LITERAL LITERAL>100|<# ACT USERDEFINEDROUTINE POSITION FORITERATION>__Fi
```

Fixed-length English dialect

```
<@ DEFUDRLIT>__FizzBuzz|<@ SAW>
<@ TSTMDOPARLIT>1|3</@>
<@ TSTMDOPARLIT>1|5</@>
<@ O12><@ SAYLIT>Fizz</@></@>
<@ O22><@ SAYLIT>Buzz</@></@>
<@ BTH><@ SAYLIT>FizzBuzz</@></@>
<@ NTH><@ SAYPAR>1</@></@>
</@></@>
<@ ITEFORLITLIT>100|<@ ACTUDRPOSFOR>__FizzBuzz|...</@> </@>
```

PureBasic

```
OpenConsole()
For x = 1 To 100
  If x%15 = 0
    PrintN("FizzBuzz")
  ElseIf x%3 = 0
    PrintN("Fizz")
  ElseIf x%5 = 0
    PrintN("Buzz")
  Else
    PrintN(Str(x))
  EndIf
Next
Input()
```

Python

```
for i in xrange(1, 101):
    if i % 15 == 0:
        print "FizzBuzz"
    elif i % 3 == 0:
        print "Fizz"
    elif i % 5 == 0:
        print "Buzz"
    else:
        print i
```

Little shorter :

```
for n in range(1,101):
    msg = ""
    if not (n%3):
        msg += "Fizz"
    if not (n%5):
        msg += "Buzz"
    print msg or str(n)
```

And a shorter, but less clear version, using a list comprehension and logical expressions:

```
for i in range(1, 101):
    words = [word for n, word in ((3, 'Fizz'), (5, 'Buzz')) if not i % n]
    print ''.join(words) or i
```

Purely functional (and somewhat obfuscated).

```
print ('\n'.join(''.join(''.join(['' if i%3 else 'Fizz',
                                  '' if i%5 else 'Buzz'])
                          or str(i))
                 for i in range(1,101)))
```

Short and without using 'if' conditional. Relies on casting int to bool and back again. Python 2.7.3.

```
for i in range(1, 101):
  print 'Fizz'*(not(i%3))+'Buzz'*(not(i%5)) or i
```

Without modulus

I came across this crazy version[4] (http://ja.doukaku.org/77/lang/ocaml/) without using the modulus operator.

```
messages = [None, "Fizz", "Buzz", "FizzBuzz"]
acc = 810092048
for i in xrange(1, 101):
    c = acc & 3
    print messages[c] if c else i
    acc = acc >> 2 | c << 28
```

Explanation

It relies on realizing that the occurrences of Fizz, Buzz, and FizzBuzz forms a repeating pattern of length 15. Arranging two bits 00 to mean print the number, 01: print Fizz, 10: print Buzz, and 11: print FizzBuzz, you can encode 30 binary bits as constant 810092048. You can decode the lowest two bits to decide what to print then rotate them to the top of the constant for successive lines of print.

```
>>> ' '.join(''.join(''.join(['' if i%3 else 'F',
                                      '' if i%5 else 'B'])
                          or str('00'))
                 for i in range(1,16))
'00 00 F 00 B F 00 00 F B 00 F 00 00 FB'
>>> _
'00 00 F 00 B F 00 00 F B 00 F 00 00 FB'
>>> _.replace('FB','11').replace('F','01').replace('B','10').split()[::-1]
['11', '00', '00', '01', '00', '10', '01', '00', '00', '01', '10', '00', '01', '00', '00']
>>> '0b' + ''.join(_)
```

```
'0b11000001001001000011000010000'
>>> eval(_)
810092048
>>>
```

Or, from @natw https://gist.github.com/4079502

```
import random

for i in range(0, 100):
    if not i % 15:
        random.seed(1178741599)
    print [i+1, "Fizz", "Buzz", "FizzBuzz"][random.randint(0,3)]
```

Given the random character of this task, this is probably the most appropriate implementation.

Lazily

You can also create a lazy, unbounded sequence by using generator expressions:

```
from itertools import cycle, izip, count, islice

fizzes = cycle([""] * 2 + ["Fizz"])
buzzes = cycle([""] * 4 + ["Buzz"])
both = (f + b for f, b in izip(fizzes, buzzes))

# if the string is "", yield the number
# otherwise yield the string
fizzbuzz = (word or n for word, n in izip(both, count(1)))

# print the first 100
for i in islice(fizzbuzz, 100):
    print i
```

R

```
x <- 1:100
xx <- as.character(x)
xx[x%%3==0] <- "Fizz"
xx[x%%5==0] <- "Buzz"
xx[x%%15==0] <- "FizzBuzz"
xx
```

Or, (ab)using the vector recycling rule:

```
x <- paste(rep("", 100), c("", "", "Fizz"), c("", "", "", "", "Buzz"), sep="")
cat(ifelse(x == "", 1:100, x), "\n")
```

Or, with a more straightforward use of ifelse:

```
x <- 1:100
ifelse(x %% 15 == 0, 'FizzBuzz',
       ifelse(x %% 5 == 0, 'Buzz',
              ifelse(x %% 3 == 0, 'Fizz', x)))
```

Racket

```
(for ([n (in-range 1 101)])
  (displayln
   (match (gcd n 15)
     [15 "fizzbuzz"]
     [3 "fizz"]
     [5 "buzz"]
     [_ n])))
```

RapidQ

The BASIC solutions work with RapidQ, too. However, here is a bit more esoteric solution using the IIF() function.

```
FOR i=1 TO 100
    t$ = IIF(i MOD 3 = 0, "Fizz", "") + IIF(i MOD 5 = 0, "Buzz", "")
    PRINT IIF(LEN(t$), t$, i)
NEXT i
```

Rascal

```
import IO;

public void fizzbuzz() {
   for(int n <- [1 .. 100]){
      fb = ((n % 3 == 0) ? "Fizz" : "") + ((n % 5 == 0) ? "Buzz" : "");
      println((fb == "") ?"<n>" : fb);
   }
}
```

Raven

```
100 each 1 + as n
  ''
  n 3 mod 0 = if 'Fizz' cat
  n 5 mod 0 = if 'Buzz' cat
  dup empty if drop n
  say
```

REALbasic

```
  For i As Integer = 1 To 100
    If i mod 3 = 0 And i mod 5 = 0 Then
      Print("FizzBuzz")
    ElseIf i mod 3 = 0 Then
      Print("Fizz")
    ElseIf i mod 5 = 0 Then
      Print("Buzz")
    Else
      Print(Str(i))
    End If
  Next
```

REBOL

Shortest implementation:

```
repeat i 100 [case/all [i // 3 = 0 [print"fizz"] i // 5 = 0 [print "buzz"] 1 [print i]]]
```

A long implementation that concatenates strings and includes a proper code header (title, date, etc.)

```
rebol [
        Title: "FizzBuzz"
        Author: oofoe
        Date: 2009-12-10
        URL: http://rosettacode.org/wiki/FizzBuzz
]

; Concatenative. Note use of 'case/all' construct to evaluate all
; conditions. I use 'copy' to allocate a new string each time through
; the loop -- otherwise 'x' would get very long...

repeat i 100 [
        x: copy ""
        case/all [
                0 = mod i 3 [append x "Fizz"]
                0 = mod i 5 [append x "Buzz"]
                "" = x      [x: mold i]
        ]
        print x
]
```

Here are two examples by Nick Antonaccio.

```
repeat i 100 [
    print switch/default 0 compose [
        (mod i 15) ["fizzbuzz"]
        (mod i 3)  ["fizz"]
        (mod i 5)  ["buzz"]
    ][i]
]

; And minimized version:

repeat i 100[j:""if 0 = mod i 3[j:"fizz"]if 0 = mod i 5[j: join j"buzz"]if j =""[j: i]print j]
```

The following is presented as a curiosity only, not as an example of good coding practice:

```
m: func [i d] [0 = mod i d]
spick: func [t x y][either any [not t  "" = t][y][x]]
zz: func [i] [rejoin [spick m i 3 "Fizz" ""  spick m i 5 "Buzz" ""]]
repeat i 100 [print spick z: zz i z i]
```

Retro

This is a port of some Forth code (http://weblog.raganwald.com/2007/01/dont-overthink-fizzbuzz.html) .

```
: fizz?    ( s-f ) 3 mod 0 = ;
: buzz?    ( s-f ) 5 mod 0 = ;
: num?     ( s-f ) dup fizz? swap buzz? or 0 = ;
: ?fizz    ( s-  ) fizz? [ "Fizz" puts ] ifTrue ;
```

```
: ?buzz    ( s-  ) buzz? [ "Buzz" puts ] ifTrue ;
: ?num     ( s-  ) num? &putn &drop if ;
: fizzbuzz ( s-  ) dup ?fizz dup ?buzz dup ?num space ;
: all      (  -  ) 100 [ 1+ fizzbuzz ] iter ;
```

It's cleaner to use quotes and combinators though:

```
needs math'
: <fizzbuzz>
  [ 15 ^math'divisor? ] [ drop "FizzBuzz" puts ] when
  [  3 ^math'divisor? ] [ drop "Fizz"     puts ] when
  [  5 ^math'divisor? ] [ drop "Buzz"     puts ] when putn ;
: fizzbuzz cr 100 [ 1+ <fizzbuzz> space ] iter ;
```

REXX

three IF-THEN

```
/*REXX program displays numbers  1 ──► 100  for the  FizzBuzz  problem. */

  do j=1  to 100;       z=j
  if j//3    ==0  then z='Fizz'
  if j//5    ==0  then z='Buzz'
  if j//(3*5)==0  then z='FizzBuzz'
  say right(z,8)
  end   /*j*/
                                      /*stick a fork in it, we're done.*/
```

Output:

```
       1
       2
    Fizz
       4
    Buzz
    Fizz
       7
       8
    Fizz
    Buzz
      11
    Fizz
      13
      14
FizzBuzz
      16
```

SELECT-WHEN

```
/*REXX program displays numbers  1 ──► 100  for the  FizzBuzz  problem. */

  do n=1  for 100
     select
     when n//15==0  then say 'FizzBuzz'
     when n//5 ==0  then say '    Buzz'
     when n//3 ==0  then say '    Fizz'
     otherwise           say right(n,8)
     end   /*select*/
  end      /*n*/
                                      /*stick a fork in it, we're done.*/
```

output is identical to version 1.

two IF-THEN

```
/*REXX program displays numbers  1 ──► 100  for the  FizzBuzz  problem. */

  do n=1  for 100;  _=
  if n//3 ==0  then _= 'Fizz'
  if n//5 ==0  then _=_'Buzz'
  say right(word(_ n,1),8)
  end   /*n*/
                                        /*stick a fork in it, we're done.*/
```

output is identical to version 1.

Ruby

```
1.upto(100) do |n|
  print "Fizz" if a = (n % 3).zero?
  print "Buzz" if b = (n % 5).zero?
  print n unless (a || b)
  print "\n"
end
```

A bit more straightforward:

```
1.upto(100) do |n|
  if (n % 15).zero?
    puts "FizzBuzz"
  elsif (n % 5).zero?
    puts "Buzz"
  elsif (n % 3).zero?
    puts "Fizz"
  else
    puts n
  end
end
```

Enumerable#Lazy and classes:

We can grab the first n fizz/buzz/fizzbuzz numbers in a list with a user defined function (filter_map), starting at the number we desire

i.e, grabbing the first 10 fizz numbers starting from 30, fizz = Fizz.new(30,10) #=> [30, 33, 36, 39, 42, 45, 48, 51, 54, 57]

```
class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do |holder, *values|
      result = yield *values
      holder << result if result
    end
  end
end
```

```ruby
class Fizz
  def initialize(head, tail)
    @list = (head..Float::INFINITY).lazy.filter_map{|i| i if i % 3 == 0}.first(tail)
  end

  def fizz?(num)
    search = @list
    search.include?(num)
  end

  def drop(num)
    list = @list
    list.delete(num)
  end

  def to_a
    @list.to_a
  end
end

class Buzz
  def initialize(head, tail)
    @list = (head..Float::INFINITY).lazy.filter_map{|i| i if i % 5 == 0}.first(tail)
  end

  def buzz?(num)
    search = @list
    search.include?(num)
  end

  def drop(num)
    list = @list
    list.delete(num)
  end

  def to_a
    @list.to_a
  end
end

class FizzBuzz
  def initialize(head, tail)
    @list = (head..Float::INFINITY).lazy.filter_map{|i| i if i % 15 == 0}.first(tail)
  end

  def fizzbuzz?(num)
    search = @list
    search.include?(num)
  end

  def to_a
    @list.to_a
  end

  def drop(num)
    list = @list
    list.delete(num)
  end
end
stopper = 100
@fizz = Fizz.new(1,100)
@buzz = Buzz.new(1,100)
@fizzbuzz = FizzBuzz.new(1,100)
def min(v, n)
  if v == 1
    puts "Fizz: #{n}"
    @fizz::drop(n)
  elsif v == 2
    puts "Buzz: #{n}"
    @buzz::drop(n)
  else
    puts "FizzBuzz: #{n}"
    @fizzbuzz::drop(n)
  end
```

```
end
(@fizz.to_a & @fizzbuzz.to_a).map{|d| @fizz::drop(d)}
(@buzz.to_a & @fizzbuzz.to_a).map{|d| @buzz::drop(d)}
while @fizz.to_a.min < stopper or @buzz.to_a.min < stopper or @fizzbuzz.to_a.min < stopper
  f, b, fb = @fizz.to_a.min, @buzz.to_a.min, @fizzbuzz.to_a.min
  min(1,f)  if f < fb and f < b
  min(2,b)  if b < f and b < fb
  min(0,fb) if fb < b and fb < f
end
```

An example using string interpolation:

```
(1..100).each do |n|
  v = "#{"Fizz" if n % 3 == 0}#{"Buzz" if n % 5 == 0}"
  puts v.empty? ? n : v
end
```

Interpolation inspired one-liner:

```
1.upto(100) { |n| puts "#{'Fizz' if n % 3 == 0}#{'Buzz' if n % 5 == 0}#{n if n % 3 != 0 && n % 5
```

An example using append:

```
1.upto 100 do |n|
  r = ''
  r << 'Fizz' if n % 3 == 0
  r << 'Buzz' if n % 5 == 0
  r << n.to_s if r.empty?
  puts r
end
```

Yet another solution:

```
1.upto(100) { |i| p "#{[:Fizz][i%3]}#{[:Buzz][i%5]}"[/.+/m] || i }
```

Monkeypatch example:

```
class Integer
  def fizzbuzz
    v = "#{"Fizz" if self % 3 == 0}#{"Buzz" if self % 5 == 0}"
    v.empty? ? self : v
  end
end

puts *(1..100).map(&:fizzbuzz)
```

Without mutable variables or inline printing.

```
fizzbuzz = ->(i) do
  (i%15).zero? and next "FizzBuzz"
  (i%3).zero?  and next "Fizz"
  (i%5).zero?  and next "Buzz"
  i
end
```

```
puts (1..100).map(&fizzbuzz).join("\n")
```

Jump anywhere#Ruby has a worse example of FizzBuzz, using a continuation!

Run BASIC

```
for i = 1 to 100
 print i;
 if (i mod 15) = 0 then print " FuzzBuzz";
 if (i mod  3) = 0 then print " Fuzz";
 if (i mod  5) = 0 then print " Buzz";
 print
next i
```

Rust

```
// rust 0.8

fn main() {
    let mut n:uint = 1;
    while n <= 100 {
        if n % 15 == 0 {
            println("FizzBuzz");
        }
        else if n % 3 == 0 {
            println("Fizz");
        }
        else if n % 5 == 0 {
            println("Buzz");
        }
        else {
            println(n.to_str());
        }
        n += 1;
    }
}
```

or

```
// rust 0.8

fn main() { for n in std::iter::range_inclusive(1,100) { fizzbuzz(n) }}

fn fizzbuzz(n:int) {
    let mut buf = ~"";
    if n % 3 == 0 { buf.push_str("Fizz") }
    if n % 5 == 0 { buf.push_str("Buzz") }
    if buf != ~"" { println!("{}", buf ) }
    else          { println!("{}", n   ) }
}
```

Using pattern matching on ints:

```
// rust 0.8

fn main() {
    for num in std::iter::range_inclusive(1, 100) {
        println(
            match (num % 3, num % 5) {
                (0, 0) => ~"FizzBuzz",
```

```
                (0, _) => ~"Fizz",
                (_, 0) => ~"Buzz",
                (_, _) => num.to_str()
            }
        );
    }
}
```

Using pattern matching on bools:

```
// rust 0.8

fn main() {
    for num in std::iter::range_inclusive(1, 100) {
        println(
            match (num % 3 == 0, num % 5 == 0) {
                (false, false) => num.to_str(),
                (true, false) => ~"Fizz",
                (false, true) => ~"Buzz",
                (true, true) => ~"FizzBuzz"
            }
        );
    }
}
```

```
// rust 0.8

fn main() {
    for num in range(1,101) {
        let  answer =
         if  num % 15 == 0 {
             ~"FizzBuzz"
         }
         else if num % 3 == 0 {
             ~"Fizz"
         }
         else if num % 5 == 0 {
             ~"Buzz"
         }
         else {
            num.to_str()
         };
         println(answer);
    }
}
```

Salmon

```
iterate (x; [1...100])
  ((x % 15 == 0) ? "FizzBuzz" :
   ((x % 3 == 0) ? "Fizz" :
    ((x % 5 == 0) ? "Buzz" : x)))!;
```

or

```
iterate (x; [1...100])
  {
    if (x % 15 == 0)
        "FizzBuzz"!
    else if (x % 3 == 0)
        "Fizz"!
    else if (x % 5 == 0)
        "Buzz"!
```

```
    else
        x!;
  };
```

Sather

```
class MAIN is
  main is
    loop i ::= 1.upto!(100);
      s:STR := "";
      if i % 3 = 0 then s := "Fizz"; end;
      if i % 5 = 0 then s := s + "Buzz"; end;
      if s.length > 0 then
        #OUT + s + "\n";
      else
        #OUT + i + "\n";
      end;
    end;
  end;
end;
```

Scala

Library: Scala

Idiomatic scala code

```
  for (x <- 1 to 100) println(
    (x % 3, x % 5) match {
      case (0, 0) => "FizzBuzz"
      case (0, _) => "Fizz"
      case (_, 0) => "Buzz"
      case _      => x
    })
```

Geeky over-generalized solution ☺

```
  def replaceMultiples(x: Int, rs: (Int, String)*) =
    rs map { case (n, s) => Either cond (x % n == 0, s, x) } reduceLeft ((a, b) =>
      a fold ((_ => b), (s => b fold ((_ => a), (t => Right(s + t))))))

  def fizzbuzz(n: Int) =
    replaceMultiples(n, 3 -> "Fizz", 5 -> "Buzz") fold ((_ toString), identity)

  1 to 100 map fizzbuzz foreach println
```

By a two-liners geek

```
  def f(a: Int, b: Int, c: String, d: String): String = if (a % b == 0) c else d
  for (i <- 1 to 100) println(f(i, 15, "FizzBuzz", f(i, 3, "Fizz", f(i, 5, "Buzz", i.toString))))
```

Scheme

```
(do ((i 1 (+ i 1)))
    ((> i 100))
    (display
      (cond ((= 0 (modulo i 15)) "FizzBuzz")
            ((= 0 (modulo i 3))  "Fizz")
            ((= 0 (modulo i 5))  "Buzz")
            (else                i)))
    (newline))
```

Sed

```
#n
# doesn't work if there's no input
# initialize counters (0 = empty) and value
s/.*/  0/
: loop
# increment counters, set carry
s/^\(a*\) \(b*\) \([0-9][0-9]*\)/\1a \2b \3@/
# propagate carry
: carry
s/ @/ 1/
s/9@/@0/
s/8@/9/
s/7@/8/
s/6@/7/
s/5@/6/
s/4@/5/
s/3@/4/
s/2@/3/
s/1@/2/
s/0@/1/
/@/b carry
# save state
h
# handle factors
s/aaa/Fizz/
s/bbbbb/Buzz/
# strip value if any factor
/z/s/[0-9]//g
# strip counters and spaces
s/[ab ]//g
# output
p
# restore state
g
# roll over counters
s/aaa//
s/bbbbb//
# loop until value = 100
/100/q
b loop
```

Using seq:

```
seq 1 100 | sed -r '3~3 s/[0-9]*/Fizz/; 5~5 s/[0-9]*$/Buzz/'
```

Seed7

```
$ include "seed7_05.s7i";
```

```
const proc: main is func
  local
    var integer: number is 0;
  begin
    for number range 1 to 100 do
      if number rem 15 = 0 then
        writeln("FizzBuzz");
      elsif number rem 5 = 0 then
        writeln("Buzz");
      elsif number rem 3 = 0 then
        writeln("Fizz");
      else
        writeln(number);
      end if;
    end for;
  end func;
```

Shen

```
(define fizzbuzz
  101 -> (nl)
  N -> (let divisible-by? (/. A B (integer? (/ A B)))
         (cases (divisible-by? N 15) (do (output "Fizzbuzz!~%")
                                         (fizzbuzz (+ N 1)))
                (divisible-by? N 3) (do (output "Fizz!~%")
                                        (fizzbuzz (+ N 1)))
                (divisible-by? N 5) (do (output "Buzz!~%")
                                        (fizzbuzz (+ N 1)))
                true (do (output (str N))
                         (nl)
                         (fizzbuzz (+ N 1))))))

(fizzbuzz 1)
```

Slate

```
n@(Integer traits) fizzbuzz
[
  output ::= ((n \\ 3) isZero ifTrue: ['Fizz'] ifFalse: ['']) ; ((n \\ 5) isZero ifTrue: ['Buzz'
  output isEmpty ifTrue: [n printString] ifFalse: [output]
].
1 to: 100 do: [| :i | inform: i fizzbuzz]
```

Smalltalk

Since only GNU Smalltalk supports file-based programming, we'll be using its syntax.

```
Integer extend [
    fizzbuzz [
        | fb |
        fb := '%<Fizz|>1%<Buzz|>2' % {
            self \\ 3 == 0.  self \\ 5 == 0 }.
        ^fb isEmpty ifTrue: [ self ] ifFalse: [ fb ]
    ]
]
1 to: 100 do: [ :i | i fizzbuzz displayNl ]
```

A Squeak/Pharo example using the Transcript window:

```
(1 to: 100) do:
        [:n |
                ((n \\ 3)*(n \\ 5)) isZero
                        ifFalse: [Transcript show: n].
                (n \\ 3) isZero
                        ifTrue: [Transcript show: 'Fizz'].
                (n \\ 5) isZero
                        ifTrue: [Transcript show: 'Buzz'].
                Transcript cr.]
```

The Squeak/Pharo examples below present possibilities using the powerful classes available. In this example, the dictionary can have as keys pairs of booleans and in the interaction the several boolean patterns select the string to be printed or if the pattern is not found the number itself is printed.

```
fizzbuzz := Dictionary with: #(true true)->'FizzBuzz'
                        with: #(true false)->'Fizz'
                        with: #(false true)->'Buzz'.

1 to: 100 do:
        [ :i | Transcript show:
                (fizzbuzz at: {i isDivisibleBy: 3. i isDivisibleBy: 5}
                         ifAbsent: [ i ]); cr]
```

Smalltalk does not have a case-select construct, but a similar effect can be attained using a collection and the #includes: method:

```
1 to: 100 do: [:n | |r|
        r := n rem: 15.
        Transcript show: (r isZero
           ifTrue:['fizzbuzz']
           ifFalse: [(#(3 6 9 12) includes: r)
                ifTrue:['fizz']
                ifFalse:[((#(5 10) includes: r))
                        ifTrue:['buzz']
                        ifFalse:[n]]]);
        cr].
```

If the construction of the whole collection is done beforehand, Smalltalk provides a straightforward way of doing because collections can be heterogeneous (may contain any object):

```
fbz := (1 to: 100) asOrderedCollection.
 3 to: 100 by:  3 do: [:i | fbz at: i put: 'Fizz'].
 5 to: 100 by:  5 do: [:i | fbz at: i put: 'Buzz'].
15 to: 100 by: 15 do: [:i | fbz at: i put: 'FizzBuzz'].
fbz do: [:i | Transcript show: i; cr].
```

The approach building a dynamic string can be done as well:

```
1 to: 100 do: [:i | |fb s|
        fb := {i isDivisibleBy: 3. i isDivisibleBy: 5. nil}.
        fb at: 3 put: (fb first | fb second) not.
        s := '<1?Fizz:><2?Buzz:><3?{1}:>' format: {i printString}.
        Transcript show: (s expandMacrosWithArguments: fb); cr].
```

SNOBOL4

Merely posting a solution by Daniel Lyons

```
        I = 1
LOOP    FIZZBUZZ = ""
        EQ(REMDR(I, 3), 0)              :F(TRY_5)
        FIZZBUZZ = FIZZBUZZ "FIZZ"
TRY_5   EQ(REMDR(I, 5), 0)              :F(DO_NUM)
        FIZZBUZZ = FIZZBUZZ "BUZZ"
DO_NUM  IDENT(FIZZBUZZ, "")             :F(SHOW)
        FIZZBUZZ = I
SHOW    OUTPUT = FIZZBUZZ
        I = I + 1
        LE(I, 100)                      :S(LOOP)
END
```

SNUSP

```
        /               'B' @=@@=@@++++#
       // /             'u' @@@@@=@+++++#
      // // /           'z' @=@@@@+@++++#
     // // // /         'i' @@@@@@=+++++#
    // // // // /       'F' @@@=@@+++++#
   // // // // // /       LF  ++++++++++#
  // // // // // // /   100 @@@=@@@=++++#
$@/>@/>@/>@/>@/>@/\    0
/                     /
!                           /======= Fizz <<<.<.<..>>>#
/                           |      \
\?!#->+ @\.>?!#->+ @\.>?!#->+@/.>\    |
/         !          !          !  /    |
\?!#->+ @\.>?!#->+@\ .>?!#->+@/.>\    |
/         !          \!===\!    !  /    |
\?!#->+ @\.>?!#->+ @\.>?!#->+@/.>\    |
/         !          !   |      !  /    |
\?!#->+@\ .>?!#->+ @\.>?!#->+@/.>\    |
/        \!==========!===\!    !  /    |
\?!#->+ @\.>?!#->+ @\.>?!#->+@/>>@\.>/
         !          |   |          |
         /==========/   \========!\=== Buzz <<<<<<<.>.>..>>>#
         |
         \!/=dup==?\>>@\<!/back?\<<<#
           \<<+>+>-/   |  \>+<- /
                       |
/======================/
|
|       /recurse\    #/?\ zero
\print=!\@\>?!\@/<@\.!\-/
          |   \=/  \=itoa=@@@+@+++++#
          !     /+ !/+ !/+ !/+   \    mod10
          /<+> -\!?-\!?-\!?-\!?-\!
          \?!\-?!\-?!\-?!\-?!\-?/\    div10
             #  +/! +/! +/! +/! +/
```

SQL

Library: SQL

Oracle SQL

```
SELECT (CASE
    WHEN MOD(lvl,15)=0 THEN 'FizzBuzz'
    WHEN MOD(lvl,3)=0 THEN 'Fizz'
    WHEN MOD(lvl,5)=0 THEN 'Buzz'
    ELSE TO_CHAR(lvl)
    END) FizzBuzz
```

```
FROM (
    SELECT LEVEL lvl
    FROM dual
    CONNECT BY LEVEL <= 100)
```

Or using Oracle's DECODE and NVL:

```
SELECT nvl(decode(MOD(n,3),0,'Fizz')||decode(MOD(n,5),0,'Buzz'),n)
FROM (SELECT level n FROM dual CONNECT BY level<=100)
```

PostgreSQL specific

```
SELECT i, fizzbuzz
  FROM
    (SELECT i,
            CASE
              WHEN i % 15 = 0 THEN 'FizzBuzz'
              WHEN i %  5 = 0 THEN 'Buzz'
              WHEN i %  3 = 0 THEN 'Fizz'
              ELSE NULL
            END AS fizzbuzz
       FROM generate_series(1,100) AS i) AS fb
 WHERE fizzbuzz IS NOT NULL;
```

Using Generate_Series and tables only:

```
SELECT COALESCE(FIZZ || BUZZ, FIZZ, BUZZ, OUTPUT) AS FIZZBUZZ FROM
(SELECT GENERATE_SERIES AS FULL_SERIES, TO_CHAR(GENERATE_SERIES,'99') AS OUTPUT
FROM GENERATE_SERIES(1,100)) F LEFT JOIN
(SELECT TEXT 'Fizz' AS FIZZ, GENERATE_SERIES AS FIZZ_SERIES FROM GENERATE_SERIES(0,100,3)) FIZZ
FIZZ.FIZZ_SERIES = F.FULL_SERIES LEFT JOIN
(SELECT TEXT 'Buzz' AS BUZZ, GENERATE_SERIES AS BUZZ_SERIES FROM GENERATE_SERIES(0,100,5)) BUZZ
BUZZ.BUZZ_SERIES = F.FULL_SERIES;
```

Recursive Common Table Expressions (MSSQL 2005+)

```
WITH nums (n, fizzbuzz ) AS (
        SELECT 1, CONVERT(nvarchar, 1) UNION ALL
        SELECT
                (n + 1) AS n1,
                CASE
                        WHEN (n + 1) % 15 = 0 THEN 'FizzBuzz'
                        WHEN (n + 1) % 3  = 0 THEN 'Fizz'
                        WHEN (n + 1) % 5  = 0 THEN 'Buzz'
                        ELSE CONVERT(nvarchar, (n + 1))
                END
        FROM nums WHERE n < 100
)
SELECT n, fizzbuzz FROM nums
ORDER BY n ASC
OPTION ( MAXRECURSION 100 )
```

Generic SQL using a join

This should work in most RDBMSs, but you may need to change `MOD(i,divisor)` to `i % divisor`.

```
-- Load some numbers
CREATE TABLE numbers(i INTEGER);
INSERT INTO numbers VALUES(1);
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
INSERT INTO numbers SELECT i + (SELECT MAX(i) FROM numbers) FROM numbers;
-- Define the fizzes and buzzes
CREATE TABLE fizzbuzz (message VARCHAR(8), divisor INTEGER);
INSERT INTO fizzbuzz VALUES('fizz',      3);
INSERT INTO fizzbuzz VALUES('buzz',      5);
INSERT INTO fizzbuzz VALUES('fizzbuzz', 15);
-- Play fizzbuzz
SELECT COALESCE(MAX(message),CAST(i AS VARCHAR(99))) AS RESULT
FROM numbers LEFT OUTER JOIN fizzbuzz ON MOD(i,divisor) = 0
GROUP BY i
HAVING i <= 100
ORDER BY i;
-- Tidy up
DROP TABLE fizzbuzz;
DROP TABLE numbers;
```

Squirrel

```
function Fizzbuzz(n) {
    for (local i = 1; i <= n; i += 1) {
        if (i % 15 == 0)
            print ("FizzBuzz\n")
        else if (i % 5 == 0)
            print ("Buzz\n")
        else if (i % 3 == 0)
            print ("Fizz\n")
        else {
            print (i + "\n")
        }
    }
}
Fizzbuzz(100);
```

Standard ML

First using two helper functions, one for deciding what to output and another for performing recursion with an auxiliary argument j.

```
local
  fun fbstr i =
      case (i mod 3 = 0, i mod 5 = 0) of
          (true , true ) => "FizzBuzz"
        | (true , false) => "Fizz"
        | (false, true ) => "Buzz"
        | (false, false) => Int.toString i

  fun fizzbuzz' (n, j) =
      if n = j then () else (print (fbstr j ^ "\n"); fizzbuzz' (n, j+1))
in
  fun fizzbuzz n = fizzbuzz' (n, 1)
  val _ = fizzbuzz 100
end
```

Second using the standard-library combinator List.tabulate and a helper function, fb, that calculates and prints the output.

```
local
  fun fb i = let val fizz = i mod 3 = 0 andalso (print "Fizz"; true)
                 val buzz = i mod 5 = 0 andalso (print "Buzz"; true)
             in fizz orelse buzz orelse (print (Int.toString i); true) end
in
  fun fizzbuzz n = (List.tabulate (n, fn i => (fb (i+1); print "\n")); ())
  val _ = fizzbuzz 100
end
```

Tcl

```
proc fizzbuzz {n {m1 3} {m2 5}} {
    for {set i 1} {$i <= $n} {incr i} {
        set ans ""
        if {$i % $m1 == 0} {append ans Fizz}
        if {$i % $m2 == 0} {append ans Buzz}
        puts [expr {$ans eq "" ? $i : $ans}]
    }
}
fizzbuzz 100
```

The following example shows Tcl's substitution mechanism that allows to concatenate the results of two successive commands into a string:

```
while {[incr i] < 101} {
    set fb [if {$i % 3 == 0} {list Fizz}][if {$i % 5 == 0} {list Buzz}]
    if {$fb ne ""} {puts $fb} {puts $i}
}
```

This version uses list rotation, so avoiding an explicit mod operation:

```
set f [lrepeat 5 "Fizz" {$i} {$i}]
foreach i {5 10} {lset f $i "Buzz"};lset f 0 "FizzBuzz"
for {set i 1} {$i <= 100} {incr i} {
    puts [subst [lindex [set f [list {*}[lassign $f ff] $ff]] 0]]
}
```

TI-83 BASIC

```
PROGRAM:FIZZBUZZ
:For(I,1,100)
  :0→N
  :If fPart(I/5)=0
    :2→N
  :If fPart(I/3)=0
    :1+N→N
  :If N=0
    :Disp I
  :If N=1
    :Disp "FIZZ"
  :If N=2
    :Disp "BUZZ"
  :If N=3
    :Disp "FIZZBUZZ"
:End
```

Turing

```
setscreen("nocursor,noecho")

for i : 1 .. 100
    if i mod 15 = 0 then
        put "Fizzbuzz" ..
    elsif i mod 5 = 0 then
        put "Buzz" ..
    elsif i mod 3 = 0 then
        put "Fizz" ..
    else
        put i ..
    end if
end for
```

TUSCRIPT

```
$$ MODE TUSCRIPT
LOOP n=1,100
mod=MOD (n,15)
SELECT mod
CASE 0
PRINT n," FizzBuzz"
CASE 3,6,9,12
PRINT n," Fizz"
CASE 5,10
PRINT n," Buzz"
DEFAULT
PRINT n
ENDSELECT
ENDLOOP
```

TXR

```
$ txr -p "(mapcar (op if @1 @1 @2) (repeat '(nil nil fizz nil buzz fizz nil nil fizz buzz nil fi
```

UNIX Shell

This solution should work with any Bourne-compatible shell.

```
i=1
while expr $i '<=' 100 >/dev/null; do
        w=false
        expr $i % 3 = 0 >/dev/null && { printf Fizz; w=true; }
        expr $i % 5 = 0 >/dev/null && { printf Buzz; w=true; }
        if $w; then echo; else echo $i; fi
        i=`expr $i + 1`
done
```

Versions for specific shells

The other solutions work with fewer shells.

The next solution requires `$(( ))` arithmetic expansion, which is in every POSIX shell; but it also requires the `seq(1)` command which is not part of some systems. (If your system misses `seq(1)`, but it has BSD `jot(1)`, then change `` `seq 1 100` `` to `` `jot 100` ``.)

```
for n in `seq 1 100`; do
  if [ $((n % 15)) = 0 ]; then
    echo FizzBuzz
  elif [ $((n % 3)) = 0 ]; then
    echo Fizz
  elif [ $((n % 5)) = 0 ]; then
    echo Buzz
  else
    echo $n
  fi
done
```

The next solution requires the `(( ))` command from the Korn Shell.

Works with: pdksh version 5.2.14

```
NUM=1
until ((NUM == 101)) ; do
   if ((NUM % 15 == 0)) ; then
       echo FizzBuzz
   elif ((NUM % 3 == 0)) ; then
       echo Fizz
   elif ((NUM % 5 == 0)) ; then
       echo Buzz
   else
       echo "$NUM"
   fi
   ((NUM = NUM + 1))
done
```

A version using concatenation:

Works with: bash version 3

```
for ((n=1; n<=100; n++))
do
  fb=''
  [ $(( n % 3 )) -eq 0 ] && fb="${fb}Fizz"
  [ $(( n % 5 )) -eq 0 ] && fb="${fb}Buzz"
  [ -n "${fb}" ] && echo "${fb}" || echo "$n"
done
```

A version using some of the insane overkill of Bash 4:

Works with: bash version 4

```
command_not_found_handle () {
  local Fizz=3 Buzz=5
  [ $(( $2 % $1 )) -eq 0 ] && echo -n $1 && [ ${!1} -eq 3 ]
}

for i in {1..100}
do
  Fizz $i && ! Buzz $i || echo -n $i
  echo
```

```
done
```

Bash one-liner

```
for i in {1..100};do (((($i%15==0))&& echo FizzBuzz)||((($i%5==0))&& echo Buzz;)||((($i%3==0))&&
```

Ursala

```
#import std
#import nat

fizzbuzz = ^T(&&'Fizz'! not remainder\3,&&'Buzz'! not remainder\5)|| ~&h+ %nP

#show+

main = fizzbuzz*t iota 101
```

V

```
[fizzbuzz
    1 [>=] [
     [[15 % zero?] ['fizzbuzz' puts]
      [5 % zero?]  ['buzz' puts]
      [3 % zero?]  ['fizz' puts]
      [true] [dup puts]
    ] when succ
  ] while].
 |100 fizzbuzz
```

Second try

(a compiler for fizzbuzz)

define a command that will generate a sequence

```
[seq [] swap dup [zero? not] [rolldown [dup] dip cons rollup pred] while pop pop].
```

create a quote that will return a quote that returns a quote if its argument is an integer (A HOF)

```
[check [N X F : [[integer?] [[X % zero?] [N F cons] if] if]] view].
```

Create a quote that will make sure that the above quote is applied correctly if given (Number Function) as arguments.

```
[func [[N F] : [dup N F check i] ] view map].
```

And apply it

```
100 seq [
```

```
        [15 [pop 'fizzbuzz' puts]]
        [5  [pop 'buzz' puts]]
        [3  [pop 'fizz' puts]]
        [1  [puts]]] [func dup] step
        [i true] map pop
```

the first one is much better :)

Vala

```
int main() {
        for(int i = 1; i < 100; i++) {
                if(i % 3 == 0) stdout.printf("Fizz");
                if(i % 5 == 0) stdout.printf("Buzz");
                if(i % 3 != 0 && i % 5 != 0) stdout.printf("%d", i);
                stdout.printf("\n");
        }
        return 0;
}
```

VBScript

Works with: Windows Script Host version *

```
For i = 1 To 100
        If i Mod 15 = 0 Then
                WScript.Echo "FizzBuzz"
        ElseIf i Mod 5 = 0 Then
                WScript.Echo "Buzz"
        ElseIf i Mod 3 = 0 Then
                WScript.Echo "Fizz"
        Else
                WScript.Echo i
        End If
Next
```

An Alternative

Works with: Windows Script Host version *

```
With WScript.StdOut
        For i = 1 To 100
                If i Mod 3 = 0 Then .Write "Fizz"
                If i Mod 5 = 0 Then .Write "Buzz"
                If .Column = 1 Then .WriteLine i Else .WriteLine ""
        Next
End With
```

Visual Basic .NET

Platform: .NET

Works with: Visual Basic .NET version 9.0+

```
Sub Main()
```

```
    For i = 1 To 100
        If i Mod 15 = 0 Then
            Console.WriteLine("FizzBuzz")
        ElseIf i Mod 5 = 0 Then
            Console.WriteLine("Buzz")
        ElseIf i Mod 3 = 0 Then
            Console.WriteLine("Fizz")
        Else
            Console.WriteLine(i)
        End If
    Next

End Sub
```

Wart

```
for i 1 (i <= 100) ++i
  prn (if (divides i 15)
            "FizzBuzz"
          (divides i 3)
            "Fizz"
          (divides i 5)
            "Buzz"
          :else
            i)
```

Whitespace

This solution was generated from the following pseudo-Assembly.

```
push 1 ; Initialize a counter.

0:
    dup dup ; Get two copies for the mod checks.
    push 3 mod jz 1
    push 5 mod jz 2
    dup onum jump 4 ; If we're still here, just print the number.

1:  ; Print "Fizz", then maybe "Buzz".
    push F ochr
    push i ochr
    call 3 push 5 mod
        jz 2
        jump 4

2:  ; Print "Buzz".
    push B ochr
    push u ochr
    call 3 jump 4

3:  ; Print "zz"; called as a function for convenient return.
    push z dup ochr ochr ret

4:
    push 10 ochr    ; Print a newline.
    push 1 add dup ; Increment the counter.
    push 101 sub
        jn 0        ; Go again unless we're at 100.
        pop exit    ; Exit clean.
```

Wortel

```
@each &x!console.log x !*&x?{%%x 15 'FizzBuzz' %%x 5 'Buzz' %%x 3 'Fizz' x} @to 100
```

XPL0

```
code CrLf=9, IntOut=11, Text=12;
int     N;
[for N:= 1 to 100 do
       [if rem(N/3)=0 then Text(0,"Fizz");
        if rem(N/5)=0 then Text(0,"Buzz")
        else if rem(N/3)#0 then IntOut(0,N);
        CrLf(0);
       ];
]
```

Output:

```
1
2
Fizz
4
Buzz
Fizz
7
...
89
FizzBuzz
91
92
Fizz
94
Buzz
Fizz
97
98
Fizz
Buzz
```

XPath 2.0

```
for $n in 1 to 100 return
  concat('fizz'[not($n mod 3)], 'buzz'[not($n mod 5)], $n[$n mod 15 = (1,2,4,7,8,11,13,14)])
```

...or alternatively...

```
for $n in 1 to 100 return
  ($n, 'Fizz', 'Buzz', 'FizzBuzz')[number(($n mod 3) = 0) + number(($n mod 5) = 0)*2 + 1]
```

XSLT 1.0

Plain XSLT

Works with: xsltproc version libxslt 10126

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
        <xsl:output method="text" encoding="utf-8"/>

        <!-- Outputs a line for a single FizzBuzz iteration. -->
        <xsl:template name="fizzbuzz-single">
                <xsl:param name="n"/>

                <!-- $s will be "", "Fizz", "Buzz", or "FizzBuzz". -->
```

```
                <xsl:variable name="s">
                        <xsl:if test="$n mod 3 = 0">Fizz</xsl:if>
                        <xsl:if test="$n mod 5 = 0">Buzz</xsl:if>
                </xsl:variable>

                <!-- Output $s. If $s is blank, also output $n. -->
                <xsl:value-of select="$s"/>
                <xsl:if test="$s = ''">
                        <xsl:value-of select="$n"/>
                </xsl:if>

                <!-- End line. -->
                <xsl:value-of select="'&#10;'"/>
        </xsl:template>

        <!-- Calls fizzbuzz-single over each value in a range. -->
        <xsl:template name="fizzbuzz-range">
                <!-- Default parameters: From 1 through 100 -->
                <xsl:param name="startAt" select="1"/>
                <xsl:param name="endAt" select="$startAt + 99"/>

                <!-- Simulate a loop with tail recursion. -->

                <!-- Loop condition -->
                <xsl:if test="$startAt &lt;= $endAt">
                        <!-- Loop body -->
                        <xsl:call-template name="fizzbuzz-single">
                                <xsl:with-param name="n" select="$startAt"/>
                        </xsl:call-template>

                        <!-- Increment counter, repeat -->
                        <xsl:call-template name="fizzbuzz-range">
                                <xsl:with-param name="startAt" select="$startAt + 1"/>
                                <xsl:with-param name="endAt" select="$endAt"/>
                        </xsl:call-template>
                </xsl:if>
        </xsl:template>

        <!-- Main procedure -->
        <xsl:template match="/">
                <!-- Default parameters are used -->
                <xsl:call-template name="fizzbuzz-range"/>
        </xsl:template>
</xsl:stylesheet>
```

With EXSLT

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="xsl exsl">
<xsl:output method="text"/>

<xsl:template name="FizzBuzz" match="/">
  <xsl:param name="n" select="1" />
  <xsl:variable name="_">
    <_><xsl:value-of select="$n" /></_>
  </xsl:variable>
  <xsl:apply-templates select="exsl:node-set($_)/_" />
  <xsl:if test="$n < 100">
    <xsl:call-template name="FizzBuzz">
      <xsl:with-param name="n" select="$n + 1" />
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template match="_[. mod 3 = 0]">Fizz
</xsl:template>

<xsl:template match="_[. mod 5 = 0]">Buzz
```

```
</xsl:template>

<xsl:template match="_[. mod 15 = 0]" priority="1">FizzBuzz
</xsl:template>

<xsl:template match="_">
  <xsl:value-of select="concat(.,'&#x0A;')" />
</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:value-of separator="&#x0A;" select="
    for $n in 1 to 100 return
      concat('fizz'[not($n mod 3)], 'buzz'[not($n mod 5)], $n[$n mod 15 = (1,2,4,7,8,11,13,14)])
</xsl:template>

</xsl:stylesheet>
```

Yorick

Iterative solution

```
for(i = 1; i <= 100; i++) {
    if(i % 3 == 0)
        write, format="%s", "Fizz";
    if(i % 5 == 0)
        write, format="%s", "Buzz";
    if(i % 3 && i % 5)
        write, format="%d", i;
    write, "";
}
```

Vectorized solution

```
output = swrite(format="%d", indgen(100));
output(3::3) = "Fizz";
output(5::5) = "Buzz";
output(15::15) = "FizzBuzz";
write, format="%s\n", output;
```


Categories: Programming Tasks | Classic CS problems and programs | Iteration | Recursion | 360 Assembly | 6502 Assembly | ACL2 | ActionScript | AutoIt | 8086 Assembly | Ada | ALGOL 68 | APL | AppleScript | Arbre | AutoHotkey | AWK | Babel | Bash | BASIC | Applesoft BASIC | Batch File | BBC BASIC | Bc | Befunge | Boo | Bracmat | Brat | Brainf*** | C | C++ | C sharp | Cduce | Chef | Clay | Clipper | CLIPS | Clojure | CMake | COBOL | Coco | CoffeeScript | Common Lisp | Cubescript | Chapel | D | Dart | Dc | Delphi | Déjà Vu | DWScript | E | ECL | Eero | Ela | Elixir | Erlang | Euphoria | Factor | F Sharp | Falcon | FALSE | Fantom | FBSL | Forth | Fortran

| Frink | GAP | Go | Gosu | Groovy | Haskell | HicEst | Icon | Unicon | Inform 6 | Inform 7 | Io | Ioke | Iptscrae | J | Java | JavaScript | Joy | Julia | K | Kamailio Script | Kaya | LabVIEW | Lasso | LaTeX | Ifthen | Intcalc | Liberty BASIC | LiveScript | Logo | LOLCODE | LSE | Lua | M4 | Make | Jot | Mathematica | MATLAB | Maxima | MAXScript | MEL | Mercury | Metafont | Mirah | MMIX | Modula-3 | MUMPS | Nemerle | NetRexx | NewtonScript | Nickle | Nimrod | Oberon-2 | Objeck | Objective-C | OCaml | Octave | OOC | Order | Oz | PARI/GP | Pascal | Perl | Perl 6 | PHL | PHP | PicoLisp | Pike | PIR | PL/I | Pop11 | PL/SQL | PostScript | Potion | PowerShell | Processing | Prolog | Protium | PureBasic | Python | R | Racket | RapidQ | Rascal | Raven | REALbasic | REBOL | Retro | REXX | Ruby | Run BASIC | Rust | Salmon | Sather | Scala | Scala Implementations | Scheme | Sed | Seed7 | Shen | Slate | Smalltalk | SNOBOL4 | SNUSP | SQL | SQL Implementations | Squirrel | Standard ML | Tcl | TI-83 BASIC | Turing | TUSCRIPT | TXR | UNIX Shell | Ursala | V | Vala | VBScript | Visual Basic .NET | Wart | Whitespace | Wortel | XPL0 | XPath 2.0 | XSLT 1.0 | XSLT 2.0 | Yorick

---