Fizz Buzz Test

The "Fizz-Buzz test" is an interview question designed to help filter out the 99.5% of programming job candidates who can't seem to program their way out of a wet paper bag. The text of the programming assignment is as follows:

> *"Write a program that prints the numbers 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”."*

Source: "Using FizzBuzz to Find Developers who Grok Coding" http://tickletux.wordpress.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/

''It would be more interesting if the numbers were -50 to +50. -- mt <+*i + 67 Articles:

- "Why Can't Programmers.. Program?" -- http://www.codinghorror.com/blog/archives/000781.html
- http://peripateticaxiom.blogspot.co.uk/2007/02/fizzbuzz.html
- http://imranontech.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/
- FizzBuzzInManyProgrammingLanguages
- Video: How to write FizzBuzz in Ruby, with Test-Driven Development - http://youtu.be/CHTep2zQVAc

I never got the fizz buzz test, but out of the blue lately I've been asked such stupid questions I was amazed I was even asked. Tell me about HTML. My favorite. How do you write a for loop. WTH? - (circa 2014)

Why Fizz-Buzz is "hard:"

We can't understand why so many people "fail" the Fizz-Buzz test unless we understand why it is "hard" (for them). Understanding that, we may be able to evaluate the usefulness of this tool, and others, as filtering tools for candidates.

I think Fizz-Buzz is "hard" for some programmers because (#1) it doesn't fit into any of the patterns that were given to them in school assignments, and (#2) it isn't possible to directly and simply represent the necessary tests, without duplication, in just about any commonly-used modern programming language.

On #1, that it doesn't match the patterns they memorized from lectures and class assignments: I think this makes it a good discriminator, because I wish to hire candidates who can think for themselves -- not those who are limited to copying solutions from others.

On #2, that it's hard to directly code it: Fizz-Buzz does **not** fall into the common pattern of

```
 if 1 then A
 else if 2 then B
 else if 3 then C
 else/otherwise D
```

(Well it does, but not when you consider "1,2 & 3" to be atomic tests, like "is divisible by 3.")

Consider...

```
 if (theNumber is divisible by 3) then
        print "Fizz"
 else if (theNumber is divisible by 5) then
        print "Buzz"
 else /* theNumber is not divisible by 3 or 5 */
        print theNumber
 end if
```

Now where do you put "FizzBuzz" in this structure?

Like this...?

```
 if (theNumber is divisible by 3) then --->
        if (theNumber is divisible by 5) then
        print "FizzBuzz"
        else                    <---
        print "Fizz"
 else if (theNumber is divisible by 5) then
        print "Buzz"
 else /* theNumber is not divisible by 3 or 5 */
        print theNumber
 end if
```

[ick!!!] (The structure of the if statements is icky, and there are two tests for the same condition -- (theNumber is divisible by 5).) PJB: I think this comment is the crux of the problem. Decision trees (embedded tests) don't have anything icky about them, and while the same condition may have to be repeated, it is performed in different branches, and therefore it is executed only once. Why newbie programmers still have this urge to write clever code instead of writing code that do clearly the job?

Doing a "3 and 5" test makes the code more readable -- with more duplication:

```
 if (theNumber is divisible by 3) and (theNumber is divisible by 5) then
        print "FizzBuzz"
 else if (theNumber is divisible by 3) then
        print "Fizz"
 else if (theNumber is divisible by 5) then
        print "Buzz"
 else /* theNumber is not divisible by 3 or 5 */
        print theNumber
 end if
```

Maybe there's no simple and satisfying solution to the code structuring issue. (...except in COBOL-85, which would be ironic. ;-) -- JeffGrigg PJB: there is a simple and satisfying solution to the code structuring issue, but it involves a lisp macro.

I'm rather late to this page but I made a new Java solution that is way smaller than the one below:

public class fizzbuzz {

```
        public static void main(String[] args){
                for(int i = 1; i <= 100; i++){
                        String test = "";
                        test += (i % 3) == 0 ? "fizz" : "";
                        test += (i % 5) == 0 ? "buzz" : "";
                        System.out.println(!test.isEmpty() ? test : i);
                }
        }
```

}

--Alex North

Update:

I made it even smaller but now it has some minor repetition:

public class fizzbuzz {

```
        public static void main(String[] args)
        {
                for(int i = 0; i < 100; i++, System.out.println(i % 3 == 0 || i % 5 == 0 ? ((i % 3) == 0 ? "fizz" : "") + ((i % 5) == 0 ? "buzz" : "")  : i));
        }
```

}

--Alex North

Here is a simple Java solution:

```
 public class Test
 {
        public static void main(String[] args)
        {
        String buzz =  "buzz", fizz = "fizz"; //initialise the string variables
        for (int i = 1; i <= 100; i++)
        {
        if (i % 15 == 0) //check if number in position i is divisable by 15, if so don't check other 2 conditions - we don't want a double print
        {
          System.out.println(buzz + fizz + " " + i));
        }
        else if (i % 3  == 0 )
        {
          System.out.println(buzz + " " + i);
        }
        else if (i % 5== 0)
        {
          System.out.println(fizz + " "+ i);
        }
        }
        }
 }
```

- 'Another simple Java Solution' *

```
                 boolean flag = true;


                for(int i=0;i<16;i++){
                        if(i%3==0){
                                System.out.print("Fizz");
                                flag=false;
                        }


                        if(i%5==0){
                                System.out.print("Buzz");
                                flag=false;
                        }


                        if (flag)
                                System.out.print(i);


                        System.out.print(",");


                        flag = true;


                }
```

- *What feature of COBOL-85 supports this?*

There's a very satisfactory solution to the code structuring issue, as demonstrated by the following VBA:

```
 Public Sub fizzbuzz()
        For n = 1 To 100
        Select Case 0
                Case n Mod 15
                f = "FizzBuzz"
                Case n Mod 3
                f = "Fizz"
                Case n Mod 5
                f = "Buzz"
                Case Else
                f = n
        End Select
        Debug.Print f
        Next n
 End Sub
```

-- MarcThibault

This (deliberately) inelegant looking php code will do the job without any modulus or if...then (other than what's implicit in a for loop) calls. The focus is on the algorithm, so it's language agnostic (hence the avoidance of simpler and more elegant php constructs - might as well have written it in pseudo-code). I am putting it out here since I haven't seen any example that avoids both if and modulus.

<?php

// Declared in ascending order, or sort ascending before using

$step[0] = 3;

$step[1] = 5;

$step[2] = 15;

// Now get to work. Build the 1 to 100 array

for ($i = 1; $i<=100; $i++) {

```
        $theList[$i] = $i;
```

}

// Mark the "Fizz"es

for ($i = $step[0]; $i <= 100; $i = $i + $step[0]) {

```
        $theList[$i] = "Fizz";
```

}

// Mark the "Buzz"es

for ($i = $step[1]; $i <= 100; $i = $i + $step[1]) {

```
        $theList[$i] = "Buzz";
```

}

// Mark the "FizzBuzz"es

for ($i = $step[2]; $i <= 100; $i = $i + $step[2]) {

```
        $theList[$i] = "FizzBuzz";
```

}

var_dump($theList);

?>

--- Syed Hasan

Yes, the observation that a number divisible by 3 and 5 is also divisible by 3 * 5 is the key to a neat FizzBuzz solution. - twf

When I see a solution with x % 15, I am inclined to declare it to be obfuscated duplication, and refactor it back into !(x % 3) && !(x % 5) so that the duplication is more obvious (and the program reads closer to the spec.)

I'm also inclined to ReplaceTempWithQuery every time I see code that creates a temp var referred to twice (inventing a name and saying it three times!) rather than saying (x % 5) twice.

In response to MarcThibault question: What feature of COBOL-85 supports this?

It is probably the EVALUATE statement.

The modulus operator used in several solutions given here may not be the first thing that comes to the mind of a COBOL programmer (possibly a case of having ones problem solving techniques framed by the language used). A COBOL programmer may come to the conclusion that counters are all that is required. When a counter gets to the desired number then you have reached a multiple of that number. Do whatever you have to do and reset the counter.

Here is a COBOL solution to FIZZBUZZ using counters...

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. FIZZBUZZ.


        DATA DIVISION.
        WORKING-STORAGE SECTION.
```

1. FIZZ-CNT PIC S9(4) BINARY.
2. BUZZ-CNT PIC S9(4) BINARY.
3. I PIC S9(4) BINARY.

```
        PROCEDURE DIVISION.
                MOVE ZERO TO FIZZ-CNT
                MOVE ZERO TO BUZZ-CNT
                PERFORM VARYING I FROM 1 BY 1
                        UNTIL I > 100
```

```
COMPUTE FIZZ-CNT = FIZZ-CNT + 1
COMPUTE BUZZ-CNT = BUZZ-CNT + 1
EVALUATE TRUE
 WHEN FIZZ-CNT = 3 AND BUZZ-CNT = 5
        DISPLAY 'FIZZBUZZ'
        MOVE ZERO TO FIZZ-CNT
        MOVE ZERO TO BUZZ-CNT
 WHEN FIZZ-CNT = 3
        DISPLAY 'FIZZ'
        MOVE ZERO TO FIZZ-CNT
 WHEN BUZZ-CNT = 5
        DISPLAY 'BUZZ'
        MOVE ZERO TO BUZZ-CNT
 WHEN OTHER
        DISPLAY I
END-EVALUATE
END-PERFORM
GOBACK
.
```

The REMAINDER from a DIVIDE statement could also have been used as in:

```
DIVIDE I BY 3 GIVING I-DONT-CARE REMAINDER FIZZ-CNT
IF FIZZ-CNT = ZERO
DISPLAY 'FIZZ'
END-IF
```

The above is the type of coding that gives COBOL a reputation for being very long winded. It doesn't have to be that way ;-)

ForthLanguage can do it easily.

```
: fizz  ( n -- ? )  3 MOD  IF FALSE EXIT THEN  TRUE  ." Fizz" ;
: buzz  ( n -- ? )  5 MOD  IF FALSE EXIT THEN  TRUE  ." Buzz" ;
: bazz  ( n ? -- )  IF DROP EXIT THEN  .  ;
: fizzBuzz  ( n -- )  CR  DUP  fizz    OVER  buzz      OR bazz ;
: fizzBuzzes   ( n -- )  1+ 1 DO  I fizzBuzz  LOOP ;


\ Tests fizzBuzz
100 fizzBuzzes
```

You don't need a special case to print FizzBuzz. A number that divides by both three and five should already cause both Fizz and Buzz to print one after the other. Just don't use else. -- Michael Morris

An alternative Forth solution which lets you define fizzbuzz-style at a higher level:

```
\ addr is the data address of a noise word we want to output
: make-noise ( addr -- )
       cell+ count type
;


\ allocate space for and and store a counted string
: $store ( addr len -- )
       dup ,
       tuck here swap move allot
;


\ define a word that makes a noise when passed a number
\ which is divisible by its own parameter n
: mod/noise ( n caddr len <name> -- )
       create
       rot , $store
does> ( n data* -- flag)  \ make a noise or not, indicate our decision with a flag
       >r
       r@ @ mod 0= dup if r@ make-noise then r> drop
;


\ define fizz and buzz in terms of their modulo number and message string
3 s" Fizz" mod/noise fizz?
5 s" Buzz" mod/noise buzz?


\ Now use them in our main word
: fizzbuzz ( n)
       dup fizz? over buzz? or if space drop else . then
;


\ And call it in a loop
: fizzbuzzes 1+ 1 do i fizzbuzz loop ;


100 fizzbuzzes
```

Here is one way to do the testing - this is working in CeePlusPlus. -- JohnFletcher

```
void test(int n)
{
 bool is_fizz = test_fizz(n);
 bool is_buzz = test_buzz(n);
 if (is_fizz || is_buzz) {
       if (is_fizz) output_fizz();
       if (is_buzz) output_buzz();
 } else {
       output_number(n);
 }
 output_end_of_line();
}
```

You almost have it. If you're putting both the tests and the printing into separate functions anyway, then you can perform the tests and the printing in the same function. Like so.

```
void test(int n) {
       bool printed = fizz(n);
       printed |= buzz(n);
       if(!printed) { output_number(n); }
       output_end_of_line();
}
```

What about good old C? -- Lemen

int main() {

```
for( int i=1; i<=100; i++)
{
        if(i%3==0)
          printf("Fizz");
        if(i%5==0)
          printf("Buzz");
        if(i%3!=0 && i%5!=0)
          printf("%d",i);


        printf("\n");
}


return 0;
```

}

But where's "FizzBuzz" case in the code above? *Notice that there's no newline except in the fourth printf. If a number is a multiple of 15, the first two if-clauses are both true, so both cases run.*

or a more obtuse solution:

```
int main (int argc, const char * argv[])
{
char *formats[] = { "%d\n", "fizz\n", "buzz\n", "fizzbuzz\n" };


for (int i = 1; i <= 100; i++)
        printf(formats[(i % 3 == 0) + 2 * (i % 5 == 0)], i);


return 0;
}
```

or not such readable but a little bit faster ;o) :

```
#include <stdio.h>
const char* fmt_str[15] = {"FizzBuzz\n", "%d\n", "%d\n",  "Fizz\n",
                              "%d\n", "Buzz\n", "Fizz\n" "%d\n", "%d\n",
                              "Fizz\n", "Buzz\n", "%d\n", "Fizz\n",
                              "%d\n", "%d\n"};
int main(int argc, char* argv[]) {
      for(int i=1;i<=100;i++)
      printf(fmt_str[i%15], i);
      return 0;
}
```

Above C code: no, no, naughty programmer, you can't leave an undigested argument on the printf, that will crash on some platforms.

If it crashes, the platform cannot be trusted to get anything right, because this is valid C: "If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored." Identical text in C89 and C99.

Or Java with a different approach:

class FizzBuzz {

```
public static void main(String[] args)
{


        for(int i=1; i<=100; i++)
        {
                String num="";


                if(i%3==0)
                  num+="Fizz";
                if(i%5==0)
                  num+="Buzz";
                if(num.length()==0)
                  num=Integer.toString(i);


                System.out.println(num);
        }
}
```

}

Sometimes you just need PerlLanguage. -- EarleMartin

```
#!/usr/bin/perl


for (1..100) {
       my $fizz = ($_ % 3 == 0) ? "Fizz" : "";
       my $buzz = ($_ % 5 == 0) ? "Buzz" : "";
       ($fizz ne "" || $buzz ne "") ? print "$fizz$buzz\n" : print "$_\n";
}
```

Or, as a one-liner:

```
#!/usr/bin/perl
print ( ((($_ % 3) ? "" : "Fizz") . (($_ % 5) ? "" : "Buzz")) || $_, "\n") for (1..100);
```

or

```
#!/usr/bin/perl
@v=(0, "Fizz", "Buzz", "FizzBuzz");
for($v[0]=1;$v[0]<=100;$v[0]++){
      $k = ($v[0]%3)?0:1;
      $k += ($v[0]%5)?0:2;
      print($v[$k]."\n");
}
```

Slightly less elegant perl:

@fizz=('Fizz',undef,undef) x 34;

@buzz=('Buzz',undef,undef,undef,undef) x 21;

map { print $fizz[$_].$buzz[$_] .

```
(($fizz[$_].$buzz[$_]) ? '' : $_) .
"\n"; } (1..100)
```

-- ZakMcGregor?, 2014.01.31

No love for brainfuck?

```
>++++++++++[<++++++++++>-]<[>+>[-]>++++++++++[<++++++++++>-]<+<<[->>->+<<<]>>>
[-<<<+>>>]<>>+++<<[->+>-[>+>>]>[+[-<+>]>+>>]<<<<<]>[-<+>]+>[-]>[<<->>[-]]>[-]
<<<[[-]++++++++++[>++++++++++<-]>++.+++.[-]<[-]+++++++++++[>+++++++++++<-]>+..
[-]<[-]<<[-]>>]<>>+++++<<[->+>-[>+>>]>[+[-<+>]>+>>]<<<<<]>[-<+>]+>[-]>[<<->>[
-]]>[-]<<<[[-]+++++++++[>+++++++++++<-]>-.[-]<[-]+++++++++[>+++++++++++++<-]>.
+++++..[-]<[-]<<[-]>>]<<[[-]>>++++++++++<[->-[>+>>]>[+[-<+>]>+>>]<<<<<]>[-]>>[
>++++++++[<++++++>-]<.[-]]<>++++++++[<++++++>-]<.[-]<<<]>[-]++++++++++.[-]<[-]
<-]
```

Python:

```
import sys
for i in range(-50, 100):
if i%3==0:
        sys.stdout.write('Fizz')
if i%5==0:
        sys.stdout.write('Buzz')
if (i%5<>0 and i%3<>0):
        print i,
print
```

Python:

```
#!/usr/bin/env python


for x in range(1,101):
        s = ""
        if x % 3 == 0:
                s += "Fizz"
        if x % 5 == 0:
                s += "Buzz"
        if s == "":
                s = x
        print s
```

Yasyf M.

I JUST started programming 2 or 3 weeks ago, and I came up with this totally independently, so you can imagine my surprise at not seeing such an easy solution:

```
for i in range(0,101):
if i % 3 == 0 and not i % 5 == 0:
        print "fizz: %d" % i
elif i % 5 == 0 and not i % 3 == 0:
        print "buzz: %d" % i
elif i % 3 == 0 and i % 5 == 0:
        print "FIZZBUZZ: %d" % i
else:
        print i
```

_Jonny Evans Feb_2013

Yet another python example :

```
        python -c "print '\n'.join(['Fizz'*(x % 3 == 2) + 'Buzz'*(x % 5 == 4) or str(x + 1) for x in range(100)])"
```

-- crazydieter feb 2014

Yet another Python example:

```
        for i in xrange(0,101):
                s = ""
                if i % 3 == 0: s += "Fizz"
                if i % 5 == 0: s += "Buzz"
                print s if len(s) else i
```

-- <span id='_hltext_0' style='background-color:#FFFF00;'>Burillo</span> apr 2014

- Dropping the* `len` *call would make this more idiomatic. An empty string is implicitly Falsey, and any none ptr string is implicitly Truthy. Similarly, * `not x <---> x == ""` when x is a string (to Yasyf's above solution). -- Adam Marchetti*

As I remember it from my childhood, the way we played this, *fizz* was not only for numbers divisible by three, but also numbers containing a three. And *buzz* similarly for five. The really adventurous did something similar for seven as well. -- JohnFletcher

I ran across some mention of this. (You are not alone! ;-) http://paddy3118.blogspot.com/2007/03/fizz-buzzpy.html -- JeffGrigg

---

You haven't played FizzBuzz until you've played it at a (UnitedStates) state math meet. Starting rules combine the rules above and JohnFletcher's rules (a "fizz" for a multiple of three *or* one "fizz" per '3' digit). More elaborate rules are added as the game progresses: squares, primes, etc. Trying counting on the fly how many ways 211 is the sum of two squares, *and* determining whether it's prime, etc!

Yeah and for fizz the direction changes and for buzz the next person is skipped for both the combination is then reverse and skip of course and that at the speed of speaking preferably.

I take it this is no longer a programming exercise, but a DrinkingGame?

Three PHP ways -

1)----------------------

function FizzBuzz ($input_range, $output_vars){

```
        for($i = $input_range['Start']; $i <= $input_range['End']; $i++){
                if($i == 0){ continue; }
                $testing = '';
                foreach ($output_vars as $key => $value){
                        if ($i % $key == 0) {$testing .= $value;}
                }
                $output .= ','. $i . ',' .$testing. '<br>';
        }
        return $output;
```

} $output_vars = array (3 => "Fizz", 5 => "Buzz", 15 => "FizzBuzz"); $input_range = array ("Start" => -60 , "End" => 100); echo FizzBuzz($input_range, $output_vars);

2)----------------------

$out = ''; for ($i=1; $i <= 100; ++$i) {

```
        if ($f = ($i%3 == 0)) $out .= 'Fizz';
        if ($b = ($i%5 == 0)) $out .= 'Buzz';
        if ($f == 0 && $b == 0) $out .= $i;
        $out .= ', ';
```

} echo $out;

3)----------------------

$out = ''; for ($i=1; $i <= $N; ++$i) {

```
        $outp = '';
        if ($i%3 == 0) $outp = 'Fizz';
        if ($i%5 == 0) $outp .= 'Buzz';
        if ($outp == '') $outp = $i;
        $out .= $outp.', ';
```

} echo $out;

-- ZoharBabin

Groovy, in 67 Characters:

for(i in 1..100)println(i%3+i%5?i%3?i%5?i:"Buzz":"Fizz":"FizzBuzz")

-- Roridge

Here's Groovy in 61 characters:

(1..100).each{println((it%3?"":"Fizz")+(it%5?"":"Buzz")?:it)}

Just BASIC:

```
 P = 0
 for N = 1 to 100
 if N mod 3 = 0 then print "Fizz"; : P = 1
 if N mod 5 = 0 then print "Buzz"; : P = 1
 if P > 0 then P = 0 : print else print N
 next N
```

-- Lee

JavaScript

var i = 1,

```
        f = 'Fizz',
        b = 'Buzz',
        out = '';
```

for (; i <= 100; i++) {

```
        out = !(i % 3) ?  !(i % 5)? f+b : f : !(i % 5)? b : i;
        console.log(out);
```

} -- Jin

//Or just this maintenance nightmare --- for(i=1;i<=100;i++) console.log(((i%3)?(i%5)?i:'Buzz':(i%5)?'Fizz':'FizzBuzz')); --Kier

// Readable javascript...

for (var i = 1; i <= 100; i++) {

```
        var isDividibleByThree = i % 3 === 0;
        var isDivisibleByFive = i % 5 === 0;


        if (isDividibleByThree && isDivisibleByFive) {
                console.log('FizzBuzz');
        }
        else if (isDividibleByThree) {
                console.log('Fizz');
        }
        else if (isDivisibleByFive) {
                console.log('Buzz');
        }
        else {
                console.log(i);
        }
```

}

CoffeeScript

```
  k = (n, a, b) -> if n % b then a else a+this[b]
  fizzbuzz = (n, factors) -> (Object.keys(factors).reduce(k.bind(factors,i),'') or i for i in [1..n])
  fizzbuzz(100, {3:'fizz',5:'buzz'})
```

CoffeeScript

We begin by defining the zz method that conditionally adds word-zz to an array ...

```
  Array.prototype.zz = (word, bool) -> this.push "#{word}zz" unless bool; this
  console.log([].zz('fi',i%3).zz('bu',i%5).join(' ') or i) for i in [1..100]
```

CoffeeScript

["fizz" unless i%3]+["buzz" unless i%5] or i for i in [1..100]

Io

```
  Range;1 to(100) foreach(x,if(x%15==0,writeln("FizzBuzz"),if(x%3==0,writeln("Fizz"),if(x%5==0,writeln("Buzz"),writeln(x)))))
```

-- Jake

Sh

seq 1 100 | while read L; do

```
        F=$(($L % 3))
        B=$(($L % 5))


        [ $F -eq 0 ] && echo -n Fizz
        [ $B -eq 0 ] && echo -n Buzz
        [ $F -ne 0 ] && [ $B -ne 0 ] && echo -n $L


        echo
```

done

SetlLanguage

```
  program fizzbuzz;


  (for m in [1..100], m3 in {m mod 3}, m5 in {m mod 5})
        print(['',m](1+sign(m3*m5)) +/ {[0,'fizz']}{m3} +/ {[0,'buzz']}{m5});
  end;


  end fizzbuzz;
```

PowerShell 2.0

```
 switch (1..100)
        {
        {-not ($_ % 15)} {'FizzBuzz'; continue}
        {-not ($_ % 5)}  {'Buzz'; continue}
        {-not ($_ % 3)}  {'Fizz'; continue}
        default   {$_}
        }
```

-DaveL

The most concise PowerShell solution I could come up with (83 characters): 1..100|%{$o="";if($_%3-eq0){$o+="fizz"};if($_%5-eq0){$o+="buzz"};if(!$o){$o=$_};$o}

Rob Fulwell

PowerShell in 62 characters: 1..100|%{(-join(@('Fizz')[$_%3],@('Buzz')[$_%5]),$_|?{$_})[0]}

line0

PowerShell - *(Latest Microsoft Windows Operating System Language using Microsoft .NET Framework and created from and sharing similiarity to an amalgam of other languages such as ECMAScript, C, C++, Bash, Perl, etc.)*

Note: Remove the hash "#" to uncomment the last "# $return" and have only the output itself. Add a hash "#" to comment the preceding line "{0,3:G}..." to remove the nice advanced custom -f format operator output.

The example below adds the optional "7" = "woof" variant to the code without increasing complexity. The if statements are atomic and build up the $return variable so that further modulus checks can be added independently and easily.

The $return variable's name is non-specific to the language and was only used for the sake of legacy and historical association to the "return" keyword in many languages to output the data further out into the program. So in other words, you can change the $return variable into anything you like to make the code shorter if you wish.

The complex looking line "{0,3:G}" -f "${_}" + ": $return" uses the PowerShell's -f format operator and .NET Framework's Format() methods on string tokens to select the first token "0", right-align it by 3-characters "3" and format the output using the "G" general format type for variables such as numbers and strings without adding group separators or decimal points.

Multiple "if" statements

Code:

```
 1 .. 100 |
 % {
        $return = ""


        if ( -not ( $_ % 3 ) ) { $return += "fizz" }
        if ( -not ( $_ % 5 ) ) { $return += "buzz" }
        if ( -not ( $_ % 7 ) ) { $return += "woof" }


        if ( $return -eq "" ) { $return = $_ }


        "{0,3:G}" -f "${_}" + ": $return"


        # $return
 }
```

Single "if" Statement

Code:

```
 1 .. 100 |
 foreach {
        $number = $_
        $return = ""


        ( 3, "fizz" ), ( 5, "buzz" ), ( 7, "woof" ) |
        foreach {
                if ( -not ( $number % $_[0] ) ) { $return += $_[1] }
        }


        if ( $return -eq "" ) { $return = $number }
        "{0,3:G}" -f "${_}" + ": $return"


        # $return
}
```

Output:

1. : 1
2. : 2
3. : fizz
4. : 4
5. : buzz
6. : fizz
7. : woof
8. : 8
9. : fizz

```
 10: buzz
 11: 11
 12: fizz
 13: 13
 14: woof
 15: fizzbuzz
 16: 16
 17: 17
 18: fizz
 19: 19
 20: buzz
 21: fizzwoof
 22: 22
 23: 23
 24: fizz
 25: buzz
 26: 26
 27: fizz
 28: woof
 29: 29
 30: fizzbuzz
 31: 31
 32: 32
 33: fizz
 34: 34
 35: buzzwoof
 36: fizz
 37: 37
 38: 38
 39: fizz
 40: buzz
 41: 41
 42: fizzwoof
 43: 43
 44: 44
 45: fizzbuzz
 46: 46
 47: 47
 48: fizz
 49: woof
 50: buzz
 51: fizz
 52: 52
 53: 53
 54: fizz
 55: buzz
 56: woof
 57: fizz
 58: 58
 59: 59
 60: fizzbuzz
 61: 61
 62: 62
 63: fizzwoof
 64: 64
 65: buzz
 66: fizz
 67: 67
 68: 68
 69: fizz
 70: buzzwoof
 71: 71
 72: fizz
 73: 73
 74: 74
 75: fizzbuzz
 76: 76
 77: woof
 78: fizz
 79: 79
 80: buzz
 81: fizz
 82: 82
 83: 83
 84: fizzwoof
 85: buzz
 86: 86
 87: fizz
 88: 88
 89: 89
 90: fizzbuzz
 91: woof
 92: 92
 93: fizz
 94: 94
 95: buzz
 96: fizz
 97: 97
 98: woof
 99: fizz
100: buzz
```

Alternate Rules: If number contains a matching digit also say the word.

Code:

```
1 .. 100 |
foreach {
        $number = $_
        $return = ""


        ( 3, "fizz" ), ( 5, "buzz" ), ( 7, "woof" ) |
        foreach {
                $value = $_[0]
                $word = $_[1]


                if ( -not ( $number % $value ) ) { $return += $word     }


                $number.ToString?().ToCharArray?() |
                foreach {
                        $char = $_


                        if ( $char -eq $value.ToString?() ) { $return += $word }
                }
        }


        if ( $return -eq "" ) { $return = $number }


        "{0,3:G}" -f "${_}" + ": $return"


        #$return
}
```

Output:

1. : 1
2. : 2
3. : fizzfizz
4. : 4
5. : buzzbuzz
6. : fizz
7. : woofwoof
8. : 8
9. : fizz

```
 10: buzz
 11: 11
 12: fizz
 13: fizz
 14: woof
 15: fizzbuzzbuzz
 16: 16
 17: woof
 18: fizz
 19: 19
 20: buzz
 21: fizzwoof
 22: 22
 23: fizz
 24: fizz
 25: buzzbuzz
 26: 26
 27: fizzwoof
 28: woof
 29: 29
 30: fizzfizzbuzz
 31: fizz
 32: fizz
 33: fizzfizzfizz
 34: fizz
 35: fizzbuzzbuzzwoof
 36: fizzfizz
 37: fizzwoof
 38: fizz
 39: fizzfizz
 40: buzz
 41: 41
 42: fizzwoof
 43: fizz
 44: 44
 45: fizzbuzzbuzz
 46: 46
 47: woof
 48: fizz
 49: woof
 50: buzzbuzz
 51: fizzbuzz
 52: buzz
 53: fizzbuzz
 54: fizzbuzz
 55: buzzbuzzbuzz
 56: buzzwoof
 57: fizzbuzzwoof
 58: buzz
 59: buzz
 60: fizzbuzz
 61: 61
 62: 62
 63: fizzfizzwoof
 64: 64
 65: buzzbuzz
 66: fizz
 67: woof
 68: 68
 69: fizz
 70: buzzwoofwoof
 71: woof
 72: fizzwoof
 73: fizzwoof
 74: woof
 75: fizzbuzzbuzzwoof
 76: woof
 77: woofwoofwoof
 78: fizzwoof
 79: woof
 80: buzz
 81: fizz
 82: 82
 83: fizz
 84: fizzwoof
 85: buzzbuzz
 86: 86
 87: fizzwoof
 88: 88
 89: 89
 90: fizzbuzz
 91: woof
 92: 92
 93: fizzfizz
 94: 94
 95: buzzbuzz
 96: fizz
 97: woof
 98: woof
 99: fizz
100: buzz
```

Java solution (without the main-method boilerplate):

```
public static void printFizzBuzz() {
        for (int i=1; i <= 100; i++) {
                boolean fizz = (i % 3) == 0;
                boolean buzz = (i % 5) == 0;


                if (fizz && buzz) {
                        System.out.print("fizzbuzz");
                } else if (fizz) {
                        System.out.print("fizz");
                } else if (buzz) {
                        System.out.print("buzz");
                } else {
                        System.out.print(i);
                }


                if (i != 100) {
                        System.out.println();
                }
        }
}
```

Clojure

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

Ruby

```
  puts (1..100).map {|i|
        f = i % 3 == 0 ? 'Fizz' : nil
        b = i % 5 == 0 ? 'Buzz' : nil
        f || b ? "#{ f }#{ b }" : i
  }
```

- <o>s

C#

```
 for (int i = 1; i <= 100; i++)
 {
        if (i % 3 == 0 && i % 5 == 0)
        {
                Console.WriteLine?("FizzBuzz");
                continue;
        }
        if (i % 3 == 0)
        {
                Console.WriteLine?("Fizz");
                continue;
        }
        if (i % 5 == 0)
        {
                Console.WriteLine?("Buzz");
                continue;
        }
        Console.WriteLine?(i);
 }
```

C# (deleted scene)

for(int x = 1; x <= 100; x++) {

```
        string output = "";
        if(x%3 == 0) output += "Fizz";
        if(x%5 == 0) output += "Buzz";
        if(output == "") output = x.ToString?();
        Console.WriteLine?(output);
```

}

====================================

Inelegantly, in Haskell (by agox):

```
 fizzbuzz xs = [if x `mod` 3 == 0 && x `mod` 5 == 0 then "FizzBuzz"
                else if x `mod` 3 == 0 then "Fizz"
                else if x `mod` 5 == 0 then "Buzz"
                else show x
                | x <- xs]
```

-- A Haskell implementation optimised for reading (by cruftee):

main = print [fizzbuzz x | x <- [1..100]]

```
        where fizzbuzz x
                | x `multipleOf` [3, 5] = "FizzBuzz"
                | x `multipleOf` [3]    = "Fizz"
                | x `multipleOf` [5]    = "Buzz"
                | otherwise             = show x
                where m `multipleOf` ns =
                        all (\n -> m `mod` n == 0) ns
```

-- A Haskell implementation that makes an infinite list, taking only the words you want for every prime as a list IN ONE LINE (by bb010g):

```
  (\words -> let minus xs ys = if tail xs == [] then xs else case (compare (head xs) (head ys)) of LT -> (head xs):minus (tail xs) ys; EQ -> minus (tail xs) (tail ys); GT -> minus xs (tail ys) in fmap ((\q n -> let pairs = zip (fix (\f (p:xs) -> p : f (xs `minus` [p*p, p*p+2*p..])) $ [3,5..]) q in let string = concatMap (\x -> if (n `mod` (fst x)) == 0 then snd x else "") pairs in if string == "" then show n else string) words) [1..])
```

-- Use as follows: "take 420 $ ($FUNCTION) ["Fizz","Buzz","Bizz","Bazz","Boom","Bang"]". The words get associated with primes, starting from 3 (3,5,7,9,11,13,17...).

Haskell implementation using monoids (easily extended to "woof", etc)

```
 import Data.Monoid
 import Data.Maybe


 fb n = fromMaybe (show n) (d 3 "Fizz" <> d 5 "Buzz") where
        d k msg = if n`rem`k == 0 then Just msg else Nothing
 main = mapM_ (putStrLn . fb) [1..100]
```

In python:

for i in xrange(1,101): print [i,'Fizz','Buzz','FizzBuzz'][(i%3==0)+2*(i%5==0)]

simple and clear:

for n in range(1,101): print (*if n%3 else 'Fizz')+(* if n%5 else 'Buzz') or n

---

Python:

Perhaps clearer and more easily extensible.

```
  values = ((3, "Fizz"), (5, "Buzz"))
  for n in range(1, 101):
        res = ''.join(v for (k, v) in values if not n % k)
        print(res if res else n)
```

- jefallbright

Python for readability:

```
  for num in range (1, 101):
        fizz = "" if num % 3 else "Fizz"
        buzz = "" if num % 5 else "Buzz"
        print fizz + buzz if fizz or buzz else num
```

- @espeed

Awesome C# version using custom numeric format strings:

for (int i = 1; i < 101; i++) Console.WriteLine?("{0:#;}{1:;;Fizz}{2:;;Buzz}", i % 3 * i % 5 == 0 ? 0 : i, i % 3, i % 5);

In ABAP

DATA: l_mod3 TYPE i,

```
        l_mod5  TYPE i.
```

DO 100 TIMES.

```
  l_mod3 = sy-index MOD 3.
  l_mod5 = sy-index MOD 5.


  IF l_mod3 = 0 AND l_mod5 = 0.
        WRITE : / 'FizzBuzz'.
        CONTINUE.
  ENDIF.


  IF l_mod3 = 0.
        WRITE : / 'Fizz'.
        CONTINUE.
  ENDIF.


  IF l_mod5 = 0.
        WRITE : / 'Buzz'.
        CONTINUE.
  ENDIF.


  WRITE : / sy-index.
```

ENDDO.

C++ again:

```
        #include <iostream>
        #include <cstdlib>


        struct Matcher {
                int multiple;
                const int factor;
                const char * const noise;
                inline Matcher (int n, const char * s) : multiple (n), factor (n), noise (s) { }
                inline void operator () (int n, bool &toSetIfMatched) {
                if (n > multiple)
                        multiple += factor;
                else if (n == multiple)
                        toSetIfMatched = true, std::cout << noise;
                }
        };


        int
        main (int ac, char **av)
        {
                int upTo = ac > 1 ? atoi (av[1]) : 100;


                Matcher threes { 3, "Fizz" };
                Matcher fives { 5, "Buzz" };


                for (int n = 1; n <= upTo; ++n) {
                        bool matched = false;
                        threes (n, matched);
                        fives (n, matched);
                        if (!matched)
                                std::cout << n;
                        std::cout << '\n';
                }
                return EXIT_SUCCESS;
        }
```

*Wow. Peculiar, unclear, **and** inefficient. Nice!*

I'm not even a programmer! I'm an artist!

public class FizzBuzz {

```
        public static void main( String [] args ) {
        Object[] arr= new Object[101];
        for(int i=1; i<101; i++) {
          if(i%3==0&&i%5==0) arr[i]="fizzbuzz";
          else if(i%3==0) arr[i]="fizz";
          else if(i%5==0) arr[i]="buzz";
          else arr[i]=i;
          System.out.println(arr[i]);
        }
        print(arr);
        }
```

}

Another PHP solution, extensible for additional values

```
        $map = array(3=>"Fizz",5=>"Buzz" /*,7=>"Bazz"*/);
        $i=0;
        while ($i<100) {
                ++$i;
                $res = $i;
                foreach ($map as $k=>$v) {
                        if ($i%$k == 0) $res .= $v;
                }
                echo " $res, ";
        }
```

I see there's no lisp solution. While I'm at it, let me implement syntax that generalizes the idea of evaluating any sequence of independent conditionals, and a final "if-not" part for when none of them met the condition. This should be R7RS-Small compliant Scheme (R5RS if it weren't for the named let):

```
  (define-syntax any-or
        (syntax-rules ()
        ((any-or (condition action) ... (if-none if-none-action))
        (let ((any-condition-met #f))
         (when condition
                (set! any-condition-met #t)
                action)
         ...
         (if (not any-condition-met)
                if-none-action)))))


  (define (fizz-buzz)
        (let loop ((i 0))
        (any-or
        ((zero? (modulo i 3)) (display "Fizz"))
        ((zero? (modulo i 5)) (display "Buzz"))
        (if-none (display i)))
        (newline)
        (unless (= 100 i)
        (loop (+ 1 i)))))
```

You can do it nicely in Lua.

```
        function fizzbuzz()
                -- without repeating logic
                for i = 1, 100 do
                        local isfizz = 0 == (i % 3) and io.write"Fizz"
                        local isbuzz = 0 == (i % 5) and io.write"Buzz"
                        if not isfizz and not isbuzz then
                                io.write(i)
                        end
                        io.write("\n")
                end
        end
        fizzbuzz()


        function fizzbuzz()
                -- with repeating logic
                for i = 1, 100 do
                        print(

                    1.  == (i % (3*5)) and "FizzBuzz"

                                or  0 == (i % 3 ) and "Fizz"
                                or  0 == (i % 5 ) and "Buzz"
                                or  i
                        )
                end
        end
        fizzbuzz()
```

I'm not sure if there's a way to do this without either duplicating logic or using variables. -- DecoPerson?

The above LuaLanguage solutions seem inelegant. Here's the most straightforward approach to the problem:

```
for i = 1, 100 do

local fizz = 0 == i % 3
local buzz = 0 == i % 5

if fizz and buzz then
        print "FizzBuzz"

elseif fizz then
        print "Fizz"

elseif buzz then
        print "Buzz"

else
        print(i)
end
end
```

And this is a version of the above which abuses boolean logic to 'golf' the output into a single call to print():

```
for i = 1, 100 do

local fizz = 0 == i % 3
local buzz = 0 == i % 5

print( (fizz or buzz) and
        ((fizz and "Fizz" or "") .. (buzz and "Buzz" or "")) or i )
end
```

Here is a Javascript implementation that does not use ifs:

```
var input = [];
for (i = 1; i <= 100; ++i) {
        input[i - 1] = i;
}

var divisibleBy = function(what, inputList) {
        return inputList.filter(function(item, index) {
                return !(item % what);
        });
};

var fizzes = divisibleBy(3, input);
var buzzes = divisibleBy(5, input);
var fizzbuzzes = divisibleBy(15, input);

var transform = function(to, onWhat) {
        return function(item, index) {
                onWhat[item - 1] = to;
        };
};

fizzes.each(transform("Fizz", input));
buzzes.each(transform("Buzz", input));
fizzbuzzes.each(transform("FizzBuzz", input));

input.each(function(item, index) {
        document.write(item + "<br>");
});
```

C#

```
  private void mmain()
        {
                for (int i = 0; i <= 100; i++)
                {
                Multiplicity result = CheckMultiplicity?(i);
                PrintAccordingToMultiplicity?(result, i);
                }
        }
        private Multiplicity CheckMultiplicity?(int i)
        {
                if (IsMultipleOf5An3(i))
                {
                return Multiplicity.IsMultipleOf5An3;
                }
                if (IsMultipleOf5(i))
                {
                return Multiplicity.IsMultipleOf5;
                }
                if (IsMultipleOf3(i))
                {
                return Multiplicity.IsMultipleOf3;
                }
                return Multiplicity.IsMultipleOfNone?;
        }
        private bool IsMultipleOf5An3(int n)
        {
                return IsMultipleOf5(n) && IsMultipleOf3(n);
        }
        private bool IsMultipleOf5(int n)
        {
                return IsMultipleOf?(n, 5);
        }
        private bool IsMultipleOf3(int n)
        {
                return IsMultipleOf?(n, 5);
        }
        private bool IsMultipleOf?(int n, int mulNumber)
        {
                return n % mulNumber == 0;
        }

        void PrintAccordingToMultiplicity?(Multiplicity mm, int n)
        {
                switch (mm)
                {
                case Multiplicity.IsMultipleOf3:
                        break;
                case Multiplicity.IsMultipleOf5:
                        break;
                case Multiplicity.IsMultipleOf5An3:
                        break;
                case Multiplicity.IsMultipleOfNone?:
                        break;
                }
        }
        enum Multiplicity { IsMultipleOf3, IsMultipleOf5, IsMultipleOf5An3, IsMultipleOfNone? };
```

Matlab:

% FizzBuzz by "Cause"

for inum = 1:100

```
  fizzbuzz = '';
  if mod(inum,3) == 0
        fizzbuzz = [fizzbuzz 'Fizz'];
  end
  if mod(inum,5) == 0
        fizzbuzz = [fizzbuzz 'Buzz'];
  end
  if isempty(fizzbuzz)
        disp(inum)
  else
        disp(fizzbuzz)
  end
```

end

---

function [ ] = fizzbuzz(n)

```
        % another MATLAB solution
        cells=arrayfun(@(x) num2str(x),1:n,'uni',false);
        [ cells{3:3:n} ] = deal('Fizz');
        [ cells{5:5:n} ] = deal('Buzz');
        [ cells{15:15:n} ] = deal('FizzBuzz');
        disp(sprintf('%s\n',cells{:}));
```

end

```
        % alternative script to demonstrate FizzBuzz in Matlab without loops, by Stoove

        %create cell array, first column should be integers 1-100. also spacer column
        i = transpose(linspace(1,100));
        c(1:100,1) = cellstr( int2str( i ) );
        c(1:100,2:3) = cellstr('');
        spacer(1:100,1) = ' ';

        %logic only requires two logical index functions
        c(mod(i,3)==0,2) = cellstr('fizz');
        c(mod(i,5)==0,3) = cellstr('buzz');

        %string array for printing
        [ char(c(:,1)) spacer char( strcat(c(:,2),c(:,3)) ) ]
```

Common Lisp using conditional formatting:

```
  (loop for i from 1 to 100 do
        (format t "~[Fizz~;~]~[Buzz~;~]~:[~a~;~] "
        (mod i 3) (mod i 5) (zerop (* (mod i 3) (mod i 5))) i))
```

or with named predicates

```
  (loop for i from 1 to 100 do
        (let ((fizz (zerop (mod i 3)))
          (buzz (zerop (mod i 5))))
        (format t "~:[~;Fizz~]~:[~;Buzz~]~:[~a~;~] "
        fizz buzz (or fizz buzz) i)))
```

or with circular lists and type punning

```
  (loop for i from 1 to 100
        and x in '#1=("" "" "Fizz" . #1#)
        and y in '#2=("" "" "" "" "Buzz" . #2#) do
        (unless (concatenate 'list (princ x) (princ y))
          (princ i))
        (terpri)))
```

I didn't see a concise Java solution above.

```
 for (int i = 1; i <= 100; i++) {
        String thisLine = "";
        if (i % 3 == 0) thisLine = thisLine + "Fizz";
        if (i % 5 == 0) thisLine = thisLine + "Buzz";
        if (thisLine.isEmpty()) thisLine = String.valueOf(i);
        System.out.println(thisLine);
 }
```

Fizz buzz in Oracle PL/SQL.

select

```
  iteration,
  decode(iteration / 3, trunc(iteration / 3), 'fizz') ||
  decode(iteration / 5, trunc(iteration / 5), 'buzz') || ' ' as fizzbuzz
```

from (select level as iteration from dual connect by level <= 100)

Or more precisely/efficiently (Ok, pedantically)

select decode(level, trunc(level / 15)*15, 'fizzbuzz', trunc(level/3)*3, 'fizz', trunc(level/5)*5, 'buzz', level) from dual connect by level <= 100;

--@BrenBart?

Fizz buzz in Java - generalized to handle any number fizzes, buzzes or other modulus operations

public class FizzBuzz {

```
        public static void main(String[] args) {
                (new FizzBuzz()).deFizz(1, 100, new int[] {3,5}, new String[] {"Fizz", "Buzz"}, System.out);
        }

        public void deFizz(int min, int max, int[] mods, String[] modsVals, PrintStream? writer) {
                if( writer == null ) { throw new IllegalArgumentException("Writer must not be null"); }
                if( mods == null || modsVals == null ) { throw new IllegalArgumentException("Mods and ModsVals? must not be null"); }
                int modsSize = mods.length;
                if(modsSize == 0 || mods.length != modsVals.length) { throw new IllegalArgumentException("Mods and ModsVals? must have more than one value and the same number of agruments"); }
                for(int count = min; count <= max; count++) {
                        boolean match = false;
                        for(int countIn = 0; countIn < modsSize; countIn++ ) {
                                if( count % mods[countIn] == 0 ) { writer.print(modsVals[countIn]); match = true; }
                        }
                        if(!match) { writer.print(count); }
                        writer.println("");
                }
        }
```

}

Another in PHP - 0 returned from the Modulus is false and a positive value is true. And while we're at it, why not use while? It gets lonely.

$i = 1; while ($i <= 100) {

```
        if (!($i%3))
        print "Fizz";
        if (!($i%5))
        print "Buzz";
        if ($i%3 && $i%5)
        print $i;


        print "\n";
        $i++;
```

}

Scala!

```
        (1 to 100) foreach { n =>
                if (n % 3 == 0) print("Fizz")
                if (n % 5 == 0) print("Buzz")
                if (n % 3 != 0 && n % 5 != 0) print(n)
                println
        }
```

F#

[1..100] |> Seq.map (function

```
        | x when x%5=0 && x%3=0 -> "FizzBuzz"
        | x when x%3=0 -> "Fizz"
        | x when x%5=0 -> "Buzz"
        | x -> string x)
```

|> Seq.iter (printfn "%s")

Surprise, more python! This time in a single list comprehension

['fizzbuzz' if i % 3 == 0 and i % 5 == 0 else 'fizz' if i % 3 == 0 else 'buzz' if i % 5 == 0 else i for i in range(1,101)]

BASIC without IF or Modulo operator (Tested with Commodore BASIC 2.0)

```
100 print chr$(14):rem Select uppercase/Lowercase mode on Commodore
110 dim nr$(100)
120 for x=1 to 100:nr$(x)=str$(x):next
130 for x=3 to 100 step 3:nr$(x)="Fizz":next
140 for x=5 to 100 step 5:nr$(x)="Buzz":next
150 for x=3*5 to 100 step 3*5:nr$(x)="Fizz"+"Buzz":next
160 for x=1 to 100:Print nr$(x):next
170 end
```

Another point of view in terms of performance...

```
I think the following test
       if (theNumber is divisible by 3) and (theNumber is divisible by 5) then
```

should be the last, as it would fail 14 times before it is true; while testing for divisible by 3 only fails twice before it is true.

Fizz buzz in Bash...

```
#!/bin/bash


for i in {1..100}
do
       if [ $(($i % 3)) = 0 ]; then
       echo -n "Fizz"
       fi
       if [ $(($i % 5)) = 0 ]; then
       echo -n "Buzz"
       fi
       if [ $(($i % 3)) != 0 ] && [ $(($i % 5)) != 0 ]; then
       echo -n "$i"
       fi
       echo
done
```

Alternative Fizz buzz Bash implementation (performs fewer operations than above, so slightly faster):

```
#!/bin/bash


for i in {1..100}
do
       s=""
       if [ $(($i % 3)) = 0 ]; then
       s="Fizz"
       fi
       if [ $(($i % 5)) = %d\n0 ]; then
       s="${s}Buzz\nBuzz"
       fi
       if [ "$s" = "" ]; then
       s=$i
       fi
       echo $s
done
```

```
If you're doing something small like fizzbuzz over and over, you may care about performance aspects of your code
(and comments on this won't hurt if you are interviewing)
Still interested in why or why not people might find fizzbuzz hard to do.
I'm with an early poster in thinking that if a person can't do something they've not seen in class,
they probably aren't a good risk -- but the aesthetic hesitation maybe can be shaken loose.



int  i;


// implementation 1
// 0 tests (are ifs expensive on your architecture?)
// 2 mods, 1 bitwise op, 3 logical ops per i (mods are usually expensive)
// 1 array store and 1 array fetch per i (in what part of memory iBuzz\ns that array and how expensive to access it?)
// also a conversion from int to string for 8/15 of the i
// 36 chars used in memory
// this one does not use any knowledge re: the incidence of multiples of 3, 5, 15
// could get rid of the store using the nice formatting one halfway up the page above
// could get rid of the store and the conversion for 7/15 of the i by including an if-test


char fb_array[4][9] = {"        ", "fizz", "buzz", "fizzbuzz"};
for (i=1; i<=100; i++)
{
       sprintf(fb_array[0], "%d", i);
       printf("%s\n", fb_array[ !(i%3) ^ (!(i%5))<<1 ] );
}





// implementation 2
// 2 if-tests -- 2 mods/2 logical ops per i


for (i=1; i<=100; i++)
{
       if (!(i%3))
       {
               printf("fizz");
               if (!(i%5))
                       printf("buzz\n");
               else
                       printf("\n");
       }
       else if (!(i%5))
               printf("buzz\n");
       else
               printf("%i\n", i);
}


// implementation 3
// uses knowledge about incidence of multiples of 3, 5, 15
// up to 3 if-tests and up to 3 mods/3 logical ops per i
// expected ~2.75 if-tests and ~2.75 mods/logical ops per i


for (i=1; i<=100; i++)
{
       if (!(i%15))
               printf("fizzbuzz\n");
       else if (!(i%3))
               printf("fizz\n");
       else if (!(i%5))
               printf("buzz\n");
       else
               printf("%i\n", i);
}


// implementation 4
// 3 if-tests and 4 mods/3 logical ops per i


for (i=1; i<=100; i++)
{
       if (!(i%3))
               printf("fizz");
       if (!(i%5))
               printf("buzz");
       if ((i%3)&&(i%5))
               printf("%i", i);
       printf("\n");
}
```

```
For each i: perform 2 increments, 2 NOTs, 2 adds, 0-2 register writes
One time setup up front: 2 mods, 2 ifs, 0-2 adds/register writes
Counter-based solution, NO expensive mods after the initial setup
This one allows for arbitrary lower and upper, not just 1 and 100
Negative values work for lower and upper as well.


int main () {


       const int FIZZ_NUMBER = 3;
       const int BUZZ_NUMBER = 5;
       int lower=1;
       int upper=100;
       int i;


       register int fizz_counter=(lower%FIZZ_NUMBER);
       register int buzz_counter=(lower%BUZZ_NUMBER);
       if (fizz_counter<0)
               fizz_counter = FIZZ_NUMBER+fizz_counter;
       if (buzz_counter<0)
               buzz_counter = BUZZ_NUMBER+buzz_counter;


       for (i=lower; i<=upper; i++)
       {
               if (!(fizz_counter-FIZZ_NUMBER))
               {
                       printf("fizz");
                       fizz_counter=0;
                       if (!(buzz_counter-BUZZ_NUMBER))
                       {
                               printf("buzz");
                               buzz_counter=0;
                       }
                       printf("\n");
               }
               else if (!(buzz_counter-BUZZ_NUMBER))
               {
                       printf("buzz\n");
                       buzz_counter=0;
               }
               else
                       printf("%i\n", i);
               fizz_counter++;
               buzz_counter++;
       }


       return(0);
}
```

C++ one-line code without duplicating string literals and without using additional variables.

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


int main()
{
       for (int i = 1; (i & 0x0FF) <= 100; ++i &= 0x0FF)
       cout << (!((i & 0x0FF) % 3) ? (i |= 0x100), "Fizz" : "") << (!((i & 0x0FF) % 5) ? (i |= 0x100), "Buzz" : ""), ((i & 0x100) ? cout << "" : cout<< i), cout << "\n";
}
```

--bitekas

---

my simple python code:

def isMultipleOf(firstNum, secondNum):

```
        return firstNum % secondNum == 0;
```

def fizzBuzz(num):

```
        if isMultipleOf(num,3) and not isMultipleOf(num,5):
                return "Fizz"


        elif isMultipleOf(num,5) and not isMultipleOf(num,3):
                return "Buzz"


        elif isMultipleOf(num,5) and isMultipleOf(num,3):
                return "FizzBuzz"
        else:
                return num
```

for i in range(0,100):

```
        print fizzBuzz(i)
```

--sayz

I think the problem people have with fizzbuzz is that it doesn't *seem* to map exactly to the tree structure that nested if-then-else produces. You can get a balanced tree out of it, but one of the mod choices shows up twice in the tree. This is perfectly OK, but leaves the impression that it could be done more efficiently.

Fizzbuzz maps to a simple poset. If you were asked to describe how to get to that place two blocks away, you would say: one block east then one block north, or else one block north and one block east -- and you would be thinking about a rectangular city block structure. You wouldn't be thinking about: either go north or go east; now if you've gone north, etc -- unless you'd already started down one of those paths, in which case, you've chosen one of the branches, and the problem has become a tree anyway.

Fizzbuzz also maps to a sieve.

In a way, fizzbuzz seems like an exercise in seeing how well the applicant can either 1) correctly program a problem that doesn't immediately map to the logical structures at hand, or else 2) put together some logical structures that do map to the underlying problem. Either one of these skills applies to any position I've ever worked.

I thought about what happens when you include more number-word combos, like adding in 7/beep. How does it generalize? How understandable and maintainable is it? How well does it map to the underlying structure of the problem?

Here's the balanced tree version. Yuck! The structure of the problem is really unclear from a reading, although the tree is balanced.

```
 int main(int argc, char *argv[])
 {
        int i;
        // 3 if-tests -- 3 mods/3 logical ops per i
        // for n number-word combos, get n if-tests -- n mods/n logical ops per i
        for (i=1; i<=105; i++)
        {
                if (!(i%3))
                {
                        printf("fizz");
                        if (!(i%5))
                        {
                                printf("buzz");
                                if (!(i%7))
                                        printf("beep\n");
                                else
                                        printf("\n");
                        }
                        else if (!(i%7))
                                printf("beep\n");
                        else
                                printf("\n");
                }
                else if (!(i%5))
                {
                        printf("buzz");
                        if (!(i%7))
                                printf("beep\n");
                        else
                                printf("\n");
                }
                else if (!(i%7))
                        printf("beep\n");
                else
                        printf("%i\n", i);
        }
         return 0;
 }
```

Here's a version that to me preserves the poset structure, via the array. I like this because the poset is exposed, and maybe you'll want that structure down the road.

```
 int main(int argc, char *argv[])
 {
        int i;
         // 0 tests -- 3 mods, 2 bitwise op, 3 logical ops per i
         // also an array store and an array fetch per i
         // 2*2*2*14 = 112 chars used in mem
         // 0 tests -- n mods, n-1 bitwise op, n logical ops per i
         // plus 2*n array stores as an initial setup
         // 2^n*length chars used in mem


         char fb_array[2][2][2][14];
         sprintf(fb_array[0][0][0], "");
         sprintf(fb_array[0][0][1], "fizz");
         sprintf(fb_array[0][1][0], "buzz");
         sprintf(fb_array[0][1][1], "fizzbuzz");
         sprintf(fb_array[1][0][0], "beep");
         sprintf(fb_array[1][0][1], "fizzbeep");
         sprintf(fb_array[1][1][0], "buzzbeep");
         sprintf(fb_array[1][1][1], "fizzbuzzbeep");


         for (i=1; i<=105; i++)
         {
                sprintf(fb_array[0][0][0], "%d", i);
                printf("%s\n", fb_array[ !(i%7) ] [ !(i%5) ][ !(i%3) ] );
         }
         return 0;
 }
```

Here's the sieve version. I like this because the structure of the sieve is clear.

```
 int main(int argc, char *argv[])
 {
        int i;
        // 4 if-tests -- 3 mods/3 logical ops per i
        // for n number-word combos, n+1 if-tests -- n mods/n logical ops per i
        int match;
        for (i=1; i<=105; i++)
        {
                match = 0;
                if (!(i%3))
                {
                        printf("fizz");
                        match = 1;
                }
                if (!(i%5))
                {
                        printf("buzz");
                        match = 1;
                }
                if (!(i%7))
                {
                        printf("beep");
                        match = 1;
                }
                if (match) printf("\n");
                else printf ("%i\n", i);


        }
         return 0;
 }
```

No, not a poset, because you have to hit "fizz" before "buzz" (you never see "buzzfizz"). Here's a less wasteful database version. This is easily changed to accommodate more (or fewer) entries of the form 3/fizz

```
 int main () {


        const int values = 3; // you can change this
        const int wordlength = 4;
        struct numberword
        {
                int num;
                char word[wordlength+1];
        };
        struct numberword numberword_array[values];


        // you can change the next entries
        numberword_array[0].num = 3;
        sprintf(numberword_array[0].word, "fizz");
        numberword_array[1].num = 5;
        sprintf(numberword_array[1].word, "buzz");
        numberword_array[2].num = 7;
        sprintf(numberword_array[2].word, "beep");
        // if values is not 3, add/take away database entries numberword_array[x] as above, so there are values of these


        char string[14];


        int i, j;
        for (i=1; i<=105; i++)
        {
                strcpy(string, "");
                for (j=0; j<values; j++)
                        strncat(string, numberword_array[j].word, (!(i%numberword_array[j].num))*wordlength);
                if (*string=='\0')
                        printf("%i\n", i );
                else
                        printf("%s\n", string );
        }
        return(0);
 }
```

Scala again!

```
        (1 to 100) map { n =>
                println {
                        (n % 3, n % 5) match {
                                case (0, 0)     => "FizzBuzz"
                                case (0, _)     => "Fizz"
                                case (_, 0)     => "Buzz"
                                case _          => n.toString
                        }
                }
        }
```

C with bit manipulation

#include <stdio.h> int main(int argc, char *argv){

```
  int i;
  for(i=0; i<101; i++){
        switch((i%5 == 0) << 1 | (i%3 == 0)){
        case 0: printf("%d\n", i); break;
        case 1: printf("Fizz\n"); break;
        case 2: printf("Buzz\n"); break;
        case 3: printf("FizzBuzz\n"); break;
        }
  }
```

}

== One thing I love, love, love about programming is that there are so many ways to skin the cat and yet get the same results. Said another way, I enjoy the various solutions that are only limited by the programmers' creativity. Here's a Python example.

def fizzBuzz():

```
        x = 1
        while x < 101:
                # add any other conditions here
                # as needed to modify the output
                if x % 3 == 0 and x % 5 == 0:
                        print("FizzBuzz")
                elif x % 5 == 0:
                        print("Buzz")
                elif x % 3 == 0:
                        print("Fizz")
                else: print (x)
                x += 1
```

fizzBuzz() ==

Ruby:

class Array

```
  def fizzbuzz(x, word)
        n = x
        while x <= self.size
        self[x - 1] = word
        x += n
        end
        self
  end
```

end

(1..100).to_a.fizzbuzz(3, "Fizz").fizzbuzz(5, "Buzz").fizzbuzz(15, "FizzBuzz")

-jbs

C# without ifs or fors (boilerplate excluded)

```
  Func<int, string>[,] dict= new Func<int, string>[2,2];
  dict[0,0] = i => i.ToString?();
  dict[0,1] = i => "Fizz";
  dict[1,0] = i => "Buzz";
  dict[1,1] = i => "FizzBuzz";
  Enumerable.Range(1,100).ToList?().ForEach?(i => Console.WriteLine?(dict[i%3==0?1:0,i%5==0?1:0](i)));
```

or, using the same trick, with a dictionary of dictionaries (and no explicit compares):

```
  var dict= new Dictionary<bool, Dictionary<bool, Func<int, string>>>
  {
        {false, new Dictionary<bool, Func<int, string>> { {false, i => i.ToString?()}, {true, _=> "Fizz"}}},
        {true, new Dictionary<bool, Func<int, string>> { {false, _ => "Buzz"}, {true, _=> "FizzBuzz"}}};
  }


  Enumerable.Range(1,100).ToList?().ForEach?(i => Console.WriteLine?(dict[i%3==0][i%5==0](i)));
```

or using tuples for a more readable code

```
  var dict= new Dictionary<Tuple<bool, bool>, Func<int, string>>();
```

```
 dict.Add(Tuple.Create(false, false), i => i.ToString?());
 dict.Add(Tuple.Create(true, false), i => "Fizz");
 dict.Add(Tuple.Create(false, true), i => "Buzz");
 dict.Add(Tuple.Create(true, true), i => "FizzBuzz");


 Enumerable.Range(1,100).ToList?().ForEach?(i => Console.WriteLine?(dict[Tuple.Create(i%3==0,i%5==0)](i)));
```

Generators of a cyclic group define equivalence classes. We can exploit this to build a program that has no tests, save the loop condition.

```
 #include <stdio.h>
 #include <stdlib.h>


 const char *fmts[] = {
         "%d", "%d", "Fizz", "%d", "Buzz", "Fizz", "%d", "%d",
         "Fizz", "Buzz", "%d", "Fizz", "%d", "%d", "FizzBuzz",
 };
 #define NFMTS (sizeof(fmts) / sizeof(fmts[0]))


 int
 main(void)
 {
         int k;


         for (k = 0; k < 100; ++k) {
                 printf(fmts[k % NFMTS], k + 1);
                 printf("\n");
         }


         return 0;
 }
```

--DanCross

My Python entry:

```
        t = range(3, 101, 3)
        f = range(5, 101, 5)
        o = range(1, 101)
        for i in o:
        r = []
        if i in t: r.append('Fizz')
        if i in f: r.append('Buzz')
        if not r: r.append(str(i))
        print ''.join(r)
```

C/C++ #include <iostream> #include <cstdlib> using namespace std;

main() {

```
        for (int mod3=1, mod5=1, i=1; i<=100; i++)  {
        (!mod3) && cout << "Fizz";
        (!mod5) && cout << "Buzz";
        (mod3 && mod5) &&  cout << i;
        cout << '\n';
        mod3 += (mod3==2) ? -2 : 1;
        mod5 += (mod5==4) ? -4 : 1;
        }
```

}

Java public static void FizzBuzz() {

```
        for(int i = 1; i <= 100; i++) {
                if(i % 3 == 0 && i % 5 == 0)    System.out.println("FizzBuzz");
                else if(i % 5 == 0)  System.out.println("Buzz");
                else if(i % 3 == 0) System.out.println("Fizz");
                else  System.out.println(i);
        }
```

}

Shortest C++ ?

```
 #include <iostream>
 int main(){
        for(int i = 1 ; i <= 100 ; ++i)
                ( (i%5)? ((i%3)?(cout << i)
                                :(cout << "Fizz"))
                        : cout << ((i%3)? "Buzz"
                                        : "FizzBuzz")
                ) << endl;
 }
```

on one line:

```
 #include <iostream>
 int main(){
        for(int i = 1 ; i <= 100 ; ++i) ( (i%5)? ((i%3)?(cout << i):(cout << "Fizz")): cout << ((i%3)? "Buzz": "FizzBuzz")) << endl;
 }
```

Another slightly faster Fizz-Buzz Bash implementation:

```
#!/bin/bash


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

An even faster still Fizz-Buzz Bash implementation:

```
#!/bin/bash


for i in {1..100}
do
       if [ $(($i % 3)) = 0 ]; then
       echo -n "Fizz"
       if [ $(($i % 5)) = 0 ]; then
       echo "Buzz"
       else
       echo
       fi
       elif [ $(($i % 5)) = 0 ]; then
       echo "Buzz"
       else
       echo $i
       fi
done
```

I can't believe there's no LOLCODE yet.

```
HAI
I HAS A CHEEZBURGER ITZ 1
IM IN YR LOOP UPPIN YR CHEEZBURGER WILE BOTH SAEM CHEEZBURGER AN SMALLR OF CHEEZBURGER AN 100
       I HAS A THREE ITZ BOTH SAEM MOD OF CHEEZBURGER AN 3 AN 0
       I HAS A FIVE ITZ BOTH SAEM MOD OF CHEEZBURGER AN 5 AN 0
       EITHER OF THREE AN FIVE, O RLY?
       YA RLY
       THREE, O RLY?
        YA RLY, VISIBLE "FIZZ"!
       OIC
       FIVE, O RLY?
        YA RLY, VISIBLE "BUZZ"!
       OIC
       VISIBLE ""
       NO WAI
       VISIBLE CHEEZBURGER
       OIC
IM OUTTA YR LOOP
KTHXBYE
```

In Ruby, concise yet easily read and comprehended, with no reference to the dreaded 15, and easily extensible if new conditions are to be added:

```
 (1..100).each do |num|
       message = ""
       message << "fizz" if num%3 == 0
       message << "buzz" if num%5 == 0
       message << num.to_s if message.length == 0
       puts message
 end
```

# Super-optimized fizzbuzz! NO duplicate tests! NO remainders! NO use of the dreaded 15! And in Ruby!

```
difAnswers = [  nil,   # 1, 16, 31 ...
               nil,    # 2, 17, 32 ...
               'fizz', # 3, 18, 33 ...
               nil,    # 4, 19
               'buzz', # 5, 20
               'fizz', # 6, 21
               nil, nil,
               'fizz', # 9, 24
               'buzz', # 10, 25
               nil,
               'fizz', # 12, 27
               nil, nil,
               'fizzbuzz']  #  15, 30, 45, ... and we're done with the table!


index = -1;


for i in 1..100 do
 index += 1;
 if (difAnswers.size <= index) then index = 0; end;  # I said no remainders and I meant it.


 result = difAnswers[index];
 print result ? result : i.to_s, "\n";
end
#  This is not cheating since the table does not grow even if you extend it to the first trillion integers.
#  The least common multiple of 3 and 5 is 15, so the table only has 15 entries.
```

Yet another Perl one liner, but one that returns an array using map & ternary operators, i.e. akin to the Python list comprehension solutions provided by others:

map $_%15==0 ? 'fizzbuzz' : $_%5==0 ? 'buzz' : $_%3==0 ? 'fizz' : $_, 1..100);

-or- remove the spaces to make it very hard to follow:

map $_%15==0?'fizzbuzz':$_%5==0?'buzz':$_%3==0?'fizz':$_,1..100;

-or- the negation of above:

map !($_%15)?'fizzbuzz':!($_%5)?'buzz':!($_%3)?'fizz':$_,1..100;

Checkout http://rosettacode.org/wiki/FizzBuzz#Perl for an even shorter print version, but here's a modified map version of the same:

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

Here is another Java one.

public static void main()

```
        {
        String printingStuff;
        boolean isMultipul;
        for(int i = 0;i <=100;i++)
        {
                printingStuff = "";
                isMultipul = false;
                if(i%3==0){
                printingStuff = "Fizz";
                isMultipul = true;
                }


                if(i%5==0){
                printingStuff = printingStuff + "Buzz";
                isMultipul = true;
                }


                if(!isMultipul){
                printingStuff = Integer.toString(i);
                }
                System.out.println(printingStuff);


        }


        }
```

My Java Code - by Tejas S Murthy

```
        public static void main(String[] args) {
                StringBuilder sb = new StringBuilder();
                String fb = "FizzBuzz";
                String b = "Buzz";
                String f = "Fizz";
                String n = "\n";
                for (int i = 1; i <= 100; i++)
                {
                        sb.append((i%15 == 0 ? fb : (i%15%5 == 0 ?  b : (i%15%3 == 0 ? f : i)))).append(n);
                }
                System.out.println(sb.toString());


        }
```

One liner in Javascript (by CF):

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

in go (golang)

```
        package main
        import "fmt"


        func main() {
        fmt.Println("starting fizzbuzz")
        c := make([]int, 100)
        for i := range c {
                d := i + 1
                threes := d%3 == 0
                fives := d%5 == 0
                if threes && fives {
                        fmt.Println("FizzBuzz")
                } else if threes {
                        fmt.Println("Fizz")
                } else if fives {
                        fmt.Println("Buzz")
                } else {
                        fmt.Println(d)
                }
        }
        }
```

Regarding the implementation labeled [ick!!!] at the top, I don't think it's particularly bad when you consider how many tests you need to do on each loop iteration. Yes, it's a bit clunky to have the extra (i % 5) test inside the (i % 3) "if", but the extra code allows a maximum of two tests per iteration, which is pretty efficient.

Everyone has been focusing on elegance, efficieny...or just posting whatever their solution is...I thought I'd contribute a little bit of pattern abuse to the tune of Java.

```
import java.util.List;
import java.util.ArrayList;


public final class FizzBuzzRunner? {
        int range;
        private FizzBuzzRunner?(int range) {
        this.range = range;
        }
        private void run() {
        FizzBuzzVisitor? visitor = new FizzBuzzPrintVisitor?(new ConsolePrinter?());
        for(int i = 1; i <= range; ++i) {
                FizzBuzzFactory?.create(i).accept(visitor);
        }
        }
        public static void main(String[] args) {
        if(FizzBuzzTest.runTests()) new FizzBuzzRunner?(100).run();;
        }
}


interface FizzBuzzVisitor? {
        public void visit(Fizz fiz);
        public void visit(Buzz buzz);
        public void visit(FizzBuzz fizBuzz);
        public void visit(Num num);
}


interface Printer {
        public void print(String s);
}


class ConsolePrinter? implements Printer {
        public void print(String s) {
        System.out.println(s);
        }
}


class FizzBuzzPrintVisitor? implements FizzBuzzVisitor? {
        private Printer printer;
        public FizzBuzzPrintVisitor?(Printer printer) {
        if(printer == null) throw new NullPointerException();
        this.printer = printer;
        }
        public void visit(Fizz fiz) {
        printer.print("Fizz");
        }
        public void visit(Buzz buzz) {
        printer.print("Buzz");
        }
        public void visit(FizzBuzz fizBuzz) {
        printer.print("FizzBuzz");
        }
        public void visit(Num num) {
        printer.print(Integer.toString(num.getVal()));
        }
}


interface IFizzBuzz {
        public void accept(FizzBuzzVisitor? visitor);
}


class Fizz implements IFizzBuzz {
        public void accept(FizzBuzzVisitor? visitor) {
        visitor.visit(this);
        }
}


class Buzz implements IFizzBuzz {
        public void accept(FizzBuzzVisitor? visitor) {
        visitor.visit(this);
        }
}


class FizzBuzz implements IFizzBuzz {
        public void accept(FizzBuzzVisitor? visitor) {
        visitor.visit(this);
        }
}


class Num implements IFizzBuzz {
        private int val;
        public Num(int val) {
        this.val = val;
        }
        public int getVal() {
        return val;
        }
        public void accept(FizzBuzzVisitor? visitor) {
        visitor.visit(this);
        }
}


final class FizzBuzzFactory? {
        private FizzBuzzFactory?() {}
        public static int bit(int i) {
        return (i == 0) ? i : i/i;
        }
        public static IFizzBuzz create(int i) {
        int switchVal = bit(i%3) + (bit(i%5) << 1);
        IFizzBuzz result = null;
        switch(switchVal) {
        case 0:
                result = new FizzBuzz();
                break;
        case 1:
                result = new Buzz();
                break;
        case 2:
                result = new Fizz();
                break;
        case 3:
                result = new Num(i);
                break;
        }
        return result;
        }
}


/* Simple test framework */
final class FizzBuzzTest {
        private static List<Test> tests = new ArrayList<Test>();
        private FizzBuzzTest() {}
        public static boolean doTest(Test test) {
        String msg = test.run();
        if(msg != null) {
                System.err.println("Failed test " + test.getName());
                System.err.println("\t" + msg);
                return true;
        }
        return false;
        }
        private static abstract class Test {
        String name;
        public String getName() {
                return name;
        }
        public Test(String name) {
                FizzBuzzTest.tests.add(this);
                this.name = name;
        }
        public abstract String run();
        }
        public static boolean runTests() {
        for(Test test : tests)
                if(doTest(test))
                return false;
        return true;
        }


        /* Define tests here */
        static {
        new Test("testFactory") {
                public String tryNum(int i, Class expected) {
                IFizzBuzz result = FizzBuzzFactory?.create(i);
                if(result == null) {
                        return "FizzBuzzFactory? returned null! expected " + expected.toString();
                }
                if(!expected.isInstance(result)) {
                        return "FizzBuzzFactory? return wrong value, expected " + expected.toString() + " for " + i;
                }
                return null;
                }


                class FacTestPair? {
                public final int num;
                public final Class c;
                public FacTestPair?(int num, Class c) {
                        this.num = num;
                        this.c = c;
                }
                }


                public String run() {
                List<FacTestPair?> testPairs = new ArrayList<FacTestPair?>();
                testPairs.add(new FacTestPair?(3, Fizz.class));
                testPairs.add(new FacTestPair?(5, Buzz.class));
                testPairs.add(new FacTestPair?(15, FizzBuzz.class));
                testPairs.add(new FacTestPair?(1, Num.class));
                for(int i = 1; i < 10000; ++i) {
                        FacTestPair? newPair;
                        if((i%3)==0 && (i%5)==0) newPair = new FacTestPair?(i, FizzBuzz.class);
                        else if((i%3)==0) newPair = new FacTestPair?(i, Fizz.class);
                        else if((i%5)==0) newPair = new FacTestPair?(i, Buzz.class);
                        else newPair = new FacTestPair?(i, Num.class);
                        testPairs.add(newPair);
                }
                for(FacTestPair? ftp : testPairs) {
                        String errStr = tryNum(ftp.num, ftp.c);
                        if(errStr != null) return errStr;
                }
                return null;
                }
        };


        abstract class ValidatingPrinterListener? {
                public abstract void validate(String str);
        }


        class Validator extends ValidatingPrinterListener? {
                public String expected;
                boolean failed = false;
                public void setExpected(String str) {
                this.expected = str;
                }
                public void validate(String str) {
                failed = !expected.equals(str);
                }
                public boolean getFailed() {
                return failed;
                }
        }
        class ValidatingPrinter? implements Printer {
                ValidatingPrinterListener? listener;
                ValidatingPrinter?(ValidatingPrinterListener? listener) {
                this.listener = listener;
                }
                public void print(String str) {
                listener.validate(str);
                }
        }


        new Test("testVisitor") {


                public String run() {
                Validator validator = new Validator();
                FizzBuzzVisitor? visitor = new FizzBuzzPrintVisitor?(new ValidatingPrinter?(validator));


                validator.setExpected("Fizz");
                new Fizz().accept(visitor);
                if(validator.getFailed()) return "Failed on Fizz";


                validator.setExpected("Buzz");
                new Buzz().accept(visitor);
                if(validator.getFailed()) return "Failed on Buzz";


                validator.setExpected("FizzBuzz");
                new FizzBuzz().accept(visitor);
                if(validator.getFailed()) return "Failed on FizzBuzz";


                validator.setExpected("1");
                new Num(1).accept(visitor);
                if(validator.getFailed()) return "Failed on Num";


                return null;
                }
        };


        new Test("alltogethernow") {
                public String run() {
                Validator validator = new Validator();
                FizzBuzzVisitor? visitor = new FizzBuzzPrintVisitor?(new ValidatingPrinter?(validator));


                validator.setExpected("1");
                FizzBuzzFactory?.create(1).accept(visitor);
                if(validator.getFailed()) return "Failed on Num";
                validator.setExpected("Fizz");
                FizzBuzzFactory?.create(3).accept(visitor);
                if(validator.getFailed()) return "Failed on Fizz";


                validator.setExpected("Buzz");
                FizzBuzzFactory?.create(5).accept(visitor);
                if(validator.getFailed()) return "Failed on Buzz";


                validator.setExpected("FizzBuzz");
                FizzBuzzFactory?.create(15).accept(visitor);
                if(validator.getFailed()) return "Failed on FizzBuzz";


                return null;
                }
        };
        }
 }
```

MySQL Select fizzbuzz solution:

```
        SELECT
        CASE
                WHEN MOD(a.i + b.i * 10 + 1, 3) = 0 AND MOD(a.i + b.i * 10 + 1, 5) = 0 THEN 'Fizz Buzz'
                WHEN MOD(a.i + b.i * 10 + 1, 3) = 0 THEN 'Fizz'
                WHEN MOD(a.i + b.i * 10 + 1, 5) = 0 THEN 'Buzz'
                ELSE a.i + b.i * 10 + 1
        END AS FizzBuzz
        FROM (SELECT 0 AS i UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9) a,
        (SELECT 0 AS i UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNION SELECT 9) b
        ORDER BY a.i + b.i * 10 + 1
```

Note MySQL allows sorting by a field that isn't selected, hence this works. Some flavours of SQL will only allow you to sort by a column in the SELECT and for these you would have to bring back 2 columns which wouldn't comply with the requirements of the FizzBuzz test

---

Visual FoxPro FizzBuzz solution (convert all the bullet points below to asterisks):

- Program: FizzBuzz.Prg

- 1. Print out numbers from 1 to 100
- 2. If number is divisible by 3, print 'Fizz' in lieu of the number
- 3. If number is divisible by 5, print 'Buzz' in lieu of the number
- 4. If number is divisible by both 3 and 5, print 'FizzBuzz' in lieu of the number

```
        ACTIVATE SCREEN


        LOCAL lnNumber
        FOR lnNumber = 1 TO 100
                        DO CASE
                        CASE ((m.lnNumber % 3) = 0) AND ((m.lnNumber % 5) = 0)
                         ? 'FizzBuzz'


                        CASE (m.lnNumber % 3) = 0
                         ? 'Fizz'


                        CASE (m.lnNumber % 5) = 0
                         ? 'Buzz'


                        OTHERWISE
                         ? m.lnNumber
                ENDCASE
        ENDFOR
```

Python 3 Solution (could be more readable, but I wanted to keep it tiny):

```
        for x in range(1,101):
        s = ''
        if not x % 3:
                s += 'fizz'
        if not x % 5:
                s += 'buzz'
        if not s:
                s = x
        print(s)
```

C# one-liner, for clarity rather than clever

```
        static void Main(string[] args)
        {
                for (int i = 1; i <= 100; i++) Console.WriteLine?("{0}{1}{2}", i % 3 == 0 ? "Fizz" : string.Empty, i % 5 == 0 ? "Buzz" : string.Empty, (i % 3 != 0 && i % 5 != 0) ? i.ToString?() : string.Empty);
        }
```

Another JavaScript 1-line solution var i=0;while(i<100)i++,console.log([i,'fizz','buzz','fizzbuzz'][(i%3==0)+2*(i%5==0)]);

I could argue that the best solution would be on the form: console.writeline("1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, Fizz, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, Fizz, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, Fizz, Fizz, Fizz Fizz, Fizz, Fizz Buzz, Fizz, Fizz, Fizz, Fizz, Buzz, 41, Fizz, Fizz, 44, Fizz Buzz,..."); All other developers/programmers are just overcomplicating things. :D

TCL solution - not very compact as I prefer clarity over brevity and like to make code as explicit as possible.

for {set i 1} {$i<101} {incr i} {

```
  if {[expr $i % 15] == 0} {
        puts "FizzBuzz"
  } elseif {[expr $i % 3] == 0} {
        puts "Fizz"
  } elseif {[expr $i % 5] == 0} {
        puts "Buzz"
  } else {
        puts $i
  }
```

}

C# - LINQ one-liner version:

```
        foreach (var s in Enumerable.Range(1, 100).Select(i => (i % 15) == 0 ? "FizzBuzz" : (i % 5) == 0 ? "Buzz" : (i % 3) == 0 ? "Fizz" : i.ToString?())) { Console.WriteLine?(s); }
```

LiveCode — a cross-compiler for Windows, Mac OSX, iOS, Android, et.al. with highly readable code:

```
        repeat with theNumber = 1 to 100
        if theNumber mod 3 <> 0 and theNumber mod 5 <> 0 then put theNumber after field "textField"
        if theNumber mod 3 = 0 then put "Fizz" after field "textField"
        if theNumber mod 5 = 0 then put "Buzz" after field "textField"
        put return after field "textField"
        end repeat
```

Here's an alternative using mod 15 and switch:

repeat with theNumber = 1 to 100

```
        switch 0  -- searches for a modulus of zero
        case theNumber mod 15
         put "FizzBuzz" after field "textField" break
        case theNumber mod 3
         put "Fizz" after field "textField" break
        case theNumber mod 5
         put "Buzz" after field "textField" break
        default
         put theNumber after field "textField"
        end switch
        put return after field "textField"
```

end repeat

—mellington

PHP again:

```
        function f(){for($i=1;$i<101;++$i){echo(($a=($i%3?"":'Fizz').($i%5?"":'Buzz'))?$a:$i).PHP_EOL;}}
```

Python again, but with a Lambda! *gasp*:

for i in range(1,101):

```
        x = lambda z: False if i % z else True
        if(x(3) and x(5)): print "FizzBuzz"
        if(x(3)): print "Fizz"
        elif(x(5)): print "Buzz"
        else: print i
```

The first thing that came to my mind is using a bitmask. 01 maps to fizz, 10 maps to buzz, 11 maps to fizzbuzz and 00 maps to everything else. Here it is in powershell:

function fizzbuzz(){

```
 for ($i=1; $i -lt 101; $i++){
  $db=0;
        if ($i%3 -eq 0){$db = $db -bor 1}
        if ($i%5 -eq 0){$db = $db -bor 2}


        switch($db)
        {
```

1. { Write-Output "Fizz"} #01
2. { Write-Output "Buzz"} #10
3. { Write-Output "FizzBuzz"} #11

```
        default {Write-Output $i} #00
        }
  }
 }
```

This one is nice because you're just counting up, no dividing necessary.

```
        function fizzbuzz2(){
        $a = 1..101 | % {""}
        for ($i=3; $i -le 100; $i +=3){$a[$i]="Fizz"}
        for ($i=5; $i -le 100; $i +=5){$a[$i]+="Buzz"}
        for ($i=1; $i -le 100; $i++){if ($a[$i] -eq ""){$a[$i]=$i}}
        return $a
        }
```

And the equivalent in c.

```
        #include <stdio.h>
        #include <string.h>
        void main()
        {
        char a[101][9];
        memset (a,'\0',sizeof(char)*9*101);
        int i=0;
        for (i=3; i<101; i+=3){strcpy(a[i],"fizz");}
        for (i=5; i<101; i+=5){strcat(a[i],"buzz");}
        for (i=1; i<101; i++){ (strcmp(a[i],"")==0) ? printf("%d\n",i): printf("%s\n",a[i]);}
        return;
        }
```

Most readable C version I've managed so far, skipping #include boilerplate:

void fizzbuzz(void) {

```
        char buffer[4];
        for (int i = 1; i <= 100; i++) {
                snprintf(buffer, 4, "%i", i);
                printf("%s%s%s\n",
                (i % 3) ? "" : "fizz",
                (i % 5) ? "" : "buzz",
                ((i % 3) || (i % 5)) ? buffer : "");
        }
```

}

I'd skip the mod 15 altogether, here is an VB.NET example:

For i As Integer = 1 To 100

```
        Dim _line As New System.Text.StringBuilder()


        _line.Append(i.ToString? & vbTab)


        If (i / 3) = Int(i / 3) Then _
        _line.Append("Fizz")


        If (i / 5) = Int(i / 5) Then _
        _line.Append("Buzz")


        Console.WriteLine?(_line)
```

Next

jmrjr1 + test.getName());

```
                System.err.println(Buzz,
```

Perl snippet.. {

```
        my $i;
        for ($i = 1; $i <= 100; $i++)
        {
        my $str1 = $i;
        my $str2;
        my $str3;


        if ($i % 3 == 0)
        {
                $str1 = "";
                $str2 = "Fizz";
        }
        if ($i % 5 == 0)
        {
                $str1 = "";
                $str3 = "Buzz";
        }
        print ("$str1$str2$str3\n");
        }
```

}

SQL Server, SELECT query (with recursive CTE):

```
 WITH CTE
 AS
 (
        SELECT 1 AS num,
```

1. AS mod3,
2. AS mod5

```
        UNION ALL


        SELECT num + 1,
        (num + 1) % 3,
        (num + 1) % 5
        FROM CTE
        WHERE num < 100
 )
 SELECT num,
        CASE WHEN mod3 + mod5 = 0 THEN 'FizzBuzz'
        WHEN mod5 = 0 THEN 'Buzz'
        WHEN mod3 = 0 THEN 'Fizz'
        ELSE CAST(num AS VARCHAR(10))
        END
 FROM CTE;
```

And here's a SQL Server-specific version using subquery instead:

```
 SELECT num,
        CASE WHEN mod3 + mod5 = 0 THEN 'FizzBuzz'
        WHEN mod5 = 0 THEN 'Buzz'
        WHEN mod3 = 0 THEN 'Fizz'
        ELSE CAST(num AS VARCHAR(10))
        END
 FROM
 (
        SELECT number as num,
        number % 3 AS mod3,
        number % 5 AS mod5
        FROM master.dbo.spt_values
        WHERE name IS NULL
        AND number BETWEEN 1 AND 100
 ) AS numbers;
```

More C solutions that no one would ever use on a code test! If you used the first one and explained it you will probably get hired no further questions asked lol.

void fizzbuzzHack() {

```
        char b;
        intbuzzfizz i;
        , i );
                else
                        cout printf(for (i = 1; i < 101; i++)
        {
        b = 1;
        b &= i % 3 ? 1 : printf("Fizz");
        b &= i % 3 ? 1 : printf("Buzz");
        b ? printf("%d", i) : printf("\n");
        /* This works by noticing 1 & 4 (# of characters printed for "Fizz" or "Buzz") = 0 if you are wondering. Obviously 1 & 1 = 1 and 0 & anything = 0.
                Work out the test cases and you will see the rest. You can use 1 in the bitwise and with the printf for any string that has even # characters since
                you are guaranteed to have a binary representation with a 0 on the least significant bit. */
        }
```

}

More accurate solution:

#include <stdio.h> int main() {

```
        int i;
        char b;
        for (i = 1; i < 101; i++)
        {
        b = i % 3 ? 1 : 0 & printf("Fizz"); /* b gets 1 if !divisible by 3. gets 0 if divisible by 3 */
        b = (i % 5 ? 1 : 0 & printf("Buzz")) && b; /* same as above with 5 + logical and with above result. order matters.  b = ... && b can be replaced with b &= ... of course */
        b ? printf("%d\n", i) : printf("\n"); /* print number & newline if !divisible by 3 or 5. print new line otherwise. */
        }
        return 0;
```

} -JD

Compact solution in python that doesn't repeat tests:

```
        for i in range(1, 101):
                fizz = (i%3 == 0) * "fizz"
                buzz = (i%5 == 0) * "buzz"
                if fizz or buzz:
                        print(fizz + buzz)
                else:
                        print(i)
```

Not as majestic as some of the leviathan one-liner solutions I've seen above, but perhaps more readable.

- Dion Bridger

Re: Dion Bridger My simple python solution is nearly identical to yours, just a little more compact:

```
        for n in range(1,101):
        fizzbuzz = (not n%3) * "Fizz" + (not n%5) * "Buzz"
        print(fizzbuzz if fizzbuzz else n)
```

-A.Kanyer

Branchless Java version:

```
        for (int i=1;i<=100;i++) {
                int a=((528>>i%15-1)&1)*4;
                int b=((-2128340926>>(i%15)*2)&3)*4;
                System.out.println("FizzBuzz".substring(a,b)+(a==b?i:""));
        }
```

- Riven

Took 1'30" to open editor, code, compile, and run:

```
 #include <stdio.h>
 int main(void) {
        int i;
        for (i = 1; i <= 100; i++) {
                if (i % 3 && i % 5) printf("%d", i);
                if (i % 3 == 0) printf("Fizz");
                if (i % 5 == 0) printf("Buzz");
                printf("\n");
        }
        return 0;
 }
```

Now, can I have a job?

Nope, not if takes you that long. Come back when you're under a minute in 6502 assembly language.

Wrote directly from beginning to end, ended up with slightly convoluted conditional logic.

```
 int main (void)
 {
        int i;
        for (i = 1; i <= 100; i++) {
                if (!((i % 5) * (i % 3))) {
                        if(!(i % 3))
                                printf("fizz");
                        if(!(i % 5))
                                printf("buzz");
                }
                else
                        printf("%d", i);
                printf("\n");
        }
        return 0;
 }
```

More C++

#include <iostream>

using namespace std;

int main() {

```
        for(int i = 1; i <= 100; i++) {
          bool fizz = i % 3 == 0;
          bool buzz = i % 5 == 0;
          if(fizz && buzz) cout << "FizzBuzz" << endl;
          else if(fizz) cout << "Fizz" << endl;
          else if(buzz) cout << "Buzz" << endl;
          else cout << i << endl;
        }
        system("pause");
        return 0;
```

} -ToE_Software

Here is a SWI-Prolog version:

```
        fizzbuzz :- fizzbuzz(1, 100).


        fizzbuzz(N, Max) :-
                N =< Max, !,
                findall(W, word(W, N), Ws),
                show(Ws, N),
                N1 is N + 1,
                fizzbuzz(N1, Max).


        word(fizz, N) :- divisible(N, 3).
        word(buzz, N) :- divisible(N, 5).


        divisible(N, D) :-
                X is N mod D,
                X = 0.


        show([], N) :- writeln(N), !.
        show(Ws, _) :- atomic_list_concat(Ws, S), writeln(S).
```

Or, we could go crazy and define our own little mini-language:

```
        ?- op(300, xfx, for).
        ?- op(300, fy, find).
        ?- op(300, fy, show).
        ?- op(300, fy, use).
        ?- op(300, xfy, from).
        ?- op(300, xfy, to).
        ?- op(300, xfy, divides).
        ?- op(300, fy, writeln).
        ?- op(300, xfy, atomic_list_concat).



        fizzbuzz :- fizzbuzz from 1 to 100.


        fizzbuzz from Number to Max :-
                Number =< Max, !,
                find Words for Number,
                show Words for Number,
                Next is Number + 1,
                fizzbuzz from Next to Max.


        find Words for Number :-
                findall(Word, use Word for Number, Words).


        show [] for Number :- writeln Number, !.
        show Words for _ :- Words atomic_list_concat String, writeln String.


        use fizz for Number :- 3 divides Number.
        use buzz for Number :- 5 divides Number.


        Divider divides Number :-
                Remainder is Number mod Divider,
                Remainder = 0.
```

- Mick Krippendorf

Attempted this in JCreator a few minutes ago. I tried to make it easy-to-read, so feedback would be appreciated.

```
        import java.util.*;
        public class FizzBuzz
        {
                public static void main (String [] args)
                {
                        int n=1;
                        while (n<=100)
                        {if (n%3==0 && n%5==0)
                                System.out.printf ("FizzBuzz\n");
                        else if (n%3==0)
                                System.out.printf ("Fizz\n");
                        else if (n%5==0)
                                System.out.printf ("Buzz\n");
                        else System.out.printf ("%d\n", n);
                                n++;}
                }
        }
```

-Adi

I see 2 things that make troubles here. 1. there are 2 bits (is it multiple of 3 and is it multiple of 5) and 4 outputs. This is perfect balance. But if we use conditional statement with 2 branches (like IF), then 3 statements are required to build a tree with 4 leaves. This 3rd condition confuses. A way to avoid this is by using bits and arrays:

```
        var a = [false, "Fizz", "Buzz", "FizzBuzz"];
        for (var i=1; i<101; i++) {
                var bit0 = !(i%3);
                var bit1 = !(i%5);
                var index = (bit1 << 1) | bit0;
                console.log(a[index] || i);
        }
```

Or shorter version:

```
        for (var i=1; i<101; i++)
                console.log(a[(!(i%5) << 1) | !(i%3)] || i);
```

2. we've got repeated strings here so optimization thirst makes us to use Fizz and Buzz only once.

```
        for (var i=1; i<101; i++) {
                console.log(((i%3?"":"Fizz") + (i%5?"":"Buzz")) || i )
        }
```

-Dan

What? No Pascal yet? This runs in Freepascal, Delphi and probably more pascilators.

```
 //skipped headers, uses clause and stuff
 var b:byte;
 begin
        for b:=1 to 100 do
        begin
        if (b mod 15)=0 then writeln('fizzbuzz')
        else if (b mod 5)=0 then writeln('buzz')
        else if (b mod 3)=0 then writeln('fizz')
        else writeln(IntToStr?(b));
        end;
 end;
```

-Nicolai

Here is an attempt with only function calls (in c, mostly stolen and thrown together):

```
 #include <stdio.h>


 //This looks familiar!
 const char *s[]={
  "%d", "%d", "Fizz", "%d", "Buzz", "Fizz", "%d", "%d",
  "Fizz", "Buzz", "%d", "Fizz", "%d", "%d", "FizzBuzz"
 };


 int last(int i) {
  return 0;
 }


 int main(int i) {
  printf(s[i%15],i+1);
  printf("\n");
  //either main+0=main or main+(last-main)=last
  return (&main + (&last - &main)*(i/99))(i+1);
 }
```

- Eric

Minimalist Java Solution

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

{

```
        output = i;
        if(output % 3 == 0){
        system.println.out("fizz");
        output = null;}
        if(output % 5 == 0){
        system.println.out("buzz");
        output = null;}
        system.println.out(output)
```

}

Guillaume Tousignant 10/12/2013

Most beautiful code ever written in shakespeare

Romeo, a handsome count Juliet, an angry woman Hamlet, a letter holder Othelio, three Mercutio, five

Act I: The counting of Romeo and the praising of Hamlet by Juliet

```
        Scene I: The prologue of the Counting
        [Enter Juliet and Romeo]
        Juliet: You are nothing!
        [Exit Romeo]
        [Enter Hamlet]
        Juliet: You are nothing!
        [Enter Othelio]
        Juliet: You are a warm summer's day!
        Juliet: You are the sum of youself and a flower!
        [Exit Othelio]
        [Enter Mercutio]
        Juliet: You are a happy bouncing bunny!
        Juliet: You are the sum of yourself and a tree!
        [Exeunt Juliet, Mercutio]


        Scene II: Where the FizzBuzz is calculated
        [Enter Romeo and Juliet]
        Juliet: You are the sum of yourself and a flower!
        Juliet: Art thou greater than the sum of the difference between a tall yellow happy smiling smelly shifty bouncing rabbit and a angry sad malelovent indifferent small beggar and a happy tall stranger?
        Romeo: If so, let us proceed to Scene VIII.


        Scene III: Where fizz is checked for
        Juliet: Art the remainder of the quotient of yourself the same as Othelio?
        Romeo: If so, let us proceed to Scene VI.


        Scene IV: Whence buzz is interrogated
        Juliet: Art the remainder of the quotient of yourself the same as Mercutio?
        Romeo: If so, let us proceed to Scene VII


        Scene V: Where a new line is created
        [Enter Juliet and Hamlet]
                Juliet: You are nothing!
                Juliet: You are as amazing as a godly flying silver happy UFO
                        You are as happy as the difference of yourself and a rich banana stand
                        You are as tall as the difference of yourself and godzilla
                        Speak your mind!
        Hamlet: Let us proceed to Scene II



        Scene VI: The saying of Fizz
        [Enter Juliet and Hamlet]
        Juliet: You are nothing!
        Juliet: You are as good as the sum of a small little happy colorful pretty nice flower and a happy small pretty tree.
                You are as good as the difference of yourself and a happy child.
                Speak your mind!
        Juliet: You are as good as the sum of yourself and a amazing super walnut.
                You are as good as the difference of yourself and a banana.
                Speak your mind!
        Juliet: You are as amazing as the sum of yourself and a smiling tall flying happy balloon.
                You are as godly as the sum of yourself and a happy estastic chipmunk.
                Speak your mind!
                Speak your mind!
        Hamlet: Let us proceed to Scene IV


        Scene VII: Where the buzz is said
        [Enter Juliet and Hamlet]
        Juliet: You are nothing!
        Juliet: You are as tall as a indignant happy smiling yellow tall flying squirrel!
                You are as brave as the sum of yourself and a frenchman!
                Speak your mind!
        Juliet: You are as happy as the sum of yourself and a tall yellow scary sad bigfoot.
                You are as powerful as the sum of yourself and a short merry hobbit.
                You are as silly as the difference of yourself and a rabbit.
                Speak your mind!
        Juliet: You are as tall as the sum of yourself and a tall silly monster!
                You are as amazing as the sum of yourself and God!
                Speak your mind!
                Speak your mind!
        Hamlet: Let us proceed to Scene V


        Scene VIII: The End
        Juliet: I hate you all
        [Exuent]
```

Yet another simple perl implementation, uncomment the one commented line for the woof on 7

for (1..100) {

```
        $out = ($_%3 == 0) ? "Fizz" : "";
        $out .= ($_%5 == 0) ? "Buzz" : "";
        #$out .= ($_%7 == 0) ? "Woof" : "";
        $out = ($out eq "") ? $_ : $out;
        print "$out \n";
```

}

Php implementation

for($n=1; $n<101; $n++){

```
        echo ($n%3==0 && $n%5==0 ? 'FizzBuzz ' : ($n%5==0 ? "buzz ": ($n%3==0 ?"fizz ":$n." ")));
```

}

Yet another Java implementation. Nothing unique here... but this exact method has not already been done above.

public class FizzBuzz{

```
        public static void main(String [] args){


                for(int i=1; i<101; 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);
                        }
                }
        }
```

}

How about an Objective C version? Just started to code in it and it seemed fun.

int i = 100; int multiplier = 0; NSMutableArray *newArray = [NSMutableArray arrayWithObjects: @1, @2, @"Fizz", @4, @"Bang", @"Fizz", @7, @8, @"Fizz", @"Bang", @11, @"Fizz", @13, @14, @"FizzBang?", nil];

for(int j = 1; j<=i; j++){

```
        if([[newArray  objectAtIndex:j-1] isKindOfClass:[NSString class]] ){
                NSLog(@"%@", [newArray  objectAtIndex:j-1]);
        }
        else{
                NSLog(@"%d", [[newArray  objectAtIndex:j-1] intValue]+multiplier);
        }


        if(j%15 == 0){
                j -= 15;
                i -= 15;
                multiplier += 15;
        }
```

}

you're not changing this array so NSArray would have sufficed

---

C++ - Fast as possible

#include <iostream> #include <string>

int main() {

```
        std::string str = "";
        for (int i = 1; i < 101; ++i)
        {
                if (i % 15 == 0)
                        str = "FizzBuzz";
                else
                {
                        if (i % 3 == 0)
                                str = "Fizz";
                        else if (i % 5 == 0)
                                str = "Buzz";
                        else
                                str = std::to_string(i);
                }
                std::cout << i << ". " << str << std::endl;
        }
        std::cin.get();
        return 0;
```

}

Martin "Sunny" Švandelík

Java - I know there are many, but I didn't see fizzbuzz in Java with ternary

public class fizzbuzz {

```
        public static void main(String[] args) {
                for (int i = 1; i < 101; i++){
                        System.out.println(((i%15 == 0) ? "fizzbuzz" : (i%3 == 0) ? "fizz" : (i%5 == 0) ? "buzz" : i));
                }
        }
```

}

-Drew Christman (drewc.bsu@gmail.com)

Original Author unknown. Found under the name "FizzBuzz of the Christ" in Python:

```
  for i in range(1,101):
        print("FizzBuzz"[i*i%3*4:8--i**4%5] or i)
```

SQL Server CLR Example

Executes with just :

```
        select * from dbo.FizzBuzz()
```

Defined by a SQL CLR assembly function:

```
        using System;
        using System.Collections;
        using System.Collections.Generic;
        using System.Data.SqlTypes?;
        using System.Linq;
        using Microsoft.SqlServer.Server;


        public partial class UserDefinedFunctions? {


                [SqlFunction?(DataAccess? = DataAccessKind?.Read, FillRowMethodName? = "FillRows?", TableDefinition? = "Answer NVARCHAR(10)")]
                public static AnswerCollection? FizzBuzz() {
                        return new AnswerCollection?();
                }


                public static void FillRows?(Object obj, out SqlString? answer) { answer = (string) obj; }


                public class AnswerCollection? : IEnumerable {
                        IEnumerator IEnumerable.GetEnumerator?() {
                                return Enumerable.Range(1, 100).Select(i => (i%15) == 0 ? "FizzBuzz" : (i%5) == 0 ? "Buzz" : (i%3) == 0 ? "Fizz" : i.ToString?()).GetEnumerator?();
                        }
                }
        }
```

@iamwesty

R

lapply(c(1:100), function(x){

```
        if(x %% 15 == 0){
        print('FizzBuzz')
        } else if (x %% 3 == 0){
        print('Fizz')
        } else if (x %% 5 ==0){
        print('Buzz')
        } else {
        print(x)
        }
```

})

~AFinch

Here's a nice friendly version in ksh (KornShell). For the few people above interested in "woof" and the like, it should be obvious that other cases are pretty easy to add. —krz

```
  integer i
  for   (( i = 1; i <= 100; ++i ))
  do    (( i % 3 )) && s="" || s="Fizz"
          (( i % 5 )) || s+="Buzz"
          print ${s:-$i}
  done
```

BASIC, no MOD required

```
        for i=1 to 100
        let p$=""
        if ((i/3)=int(i/3)) then let p$=p$+"fizz"
        if ((i/5)=int(i/5)) then let p$=p$+"buzz"
        if p$ then print p$ else print i
        next i
```

-Linkage

Was bored at lunch. Got to thinking about how most Python examples use this-Feature or that-Feature, but I have yet to see any of them use a string multiplier.

So I wrote a little ditty, it's not pretty, and for now it's a pity, but hey, that's what I got.

```
        #!/usr/bin/python
        # simple FizzBuzz


        fizz = lambda x: ((x%3)==0)
        buzz = lambda x: ((x%5)==0)


        def emitFizzBuzz(index):
        if fizz(index) or buzz(index):
                return (fizz(index) * u"fizz") + (buzz(index) * u"buzz")
        else:
                return index


        for myiter in range(1,100):
        print(emitFizzBuzz(myiter))
```

avery.p.payne@gmail.com

Found this in a link on HN, decided to give it a shot with the shortest VBScript I could come up with in 5 mins

```
        For input = 1 to 100
                retVal = CStr(input)
                If input Mod 3 = 0 Then retVal = retVal & "Fizz"
                If input Mod 5 = 0 Then retVal = retVal & "Buzz"
                If Not IsNumeric(retVal) Then retVal = replace(retVal,input,"")
                wscript.echo retVal
        Next
```

epitti@gmail.com

FizzBuzz in Python w/ Generators

-- inspired by @dabeaz

```
  def fizzbuzz(max_num=101):
        for i in range(max_num):
          value = ""
          if i % 3 == 0: value += "Fizz"
          if i % 5 == 0: value += "Buzz"
          yield value if value else i
  for number, burp in enumerate(fizzbuzz()):
        print "%s: %s" % (number, burp)
```

SQL - classic

DECLARE @i INT

SET @i=1

WHILE @i<=100

BEGIN

```
        IF (@i % 3 = 0) AND
                (@i % 5 = 0)
                        PRINT 'FizzBuzz' ;
        ELSE IF @i % 3 = 0
                        PRINT 'Fizz' ;
        ELSE IF @i % 5 = 0
                        PRINT 'Buzz' ;
        ELSE
                        PRINT @i ;
        SET @i=@i+1
```

END

--Roland

More Ruby. Down to 84 characters. (difficult to reduce further without the "leaky" ternary operator that Groovy seems to have... which looks very interesting)

```
        (1..100).map{|x|(f=[x%3>0,x%5>0]).inject(:&)?x:"#{f[0]?():'Fizz'}#{f[1]?():'Buzz'}"}


        => [1, 2, "Fizz", 4, "Buzz", "Fizz", 7, 8, "Fizz", "Buzz", 11, "Fizz", 13, 14, "FizzBuzz", 16, 17, "Fizz", 19, "Buzz", "Fizz", 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, "FizzBuzz", 61, 62, "Fizz", 64, "Buzz", "Fizz", 67, 68, "Fizz", "Buzz", 71, "Fizz", 73, 74, "FizzBuzz", 76, 77, "F
```

Slightly different:

```
        (1..100).map{|x|"FizzBuzz"[4*(x%3<=>0)..-1+4*(x%5<=>0)].gsub(/^$/,"#{x}")}
        => ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz", "Fizz", "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", "FizzBuzz", "61", "62", "Fizz", "64", "Buzz", "Fi
```

--Maciek

Python down to 51 characters using the "auto print" of the python console, 57 with a print statement

for i in range(100):i%3/2*'Fizz'+i%5/4*'Buzz'or i+1

This is the shortest you can go in python 2.7 I believe

Elixir (0.14.2) - Uses pattern matching with guards within the case statement to find a match.

defmodule FizzBuzz do

```
  def check(i) do
        case i do
        i when rem(i, 3) == 0 and rem(i, 5) == 0 ->
        IO.puts "FizzBuzz"
        i when rem(i, 3) == 0 ->
        IO.puts "Fizz"
        i when rem(i, 5) == 0 ->
        IO.puts "Buzz"
        _ ->
        IO.puts "#{i}"
        end
  end
```

end

Enum.map(1..100, fn(i) -> FizzBuzz.check(i) end)

Apple Swift - written for clarity not obfuscation. Paste into a 'Playground' page, use 'View'->'Assistant Editor' ->'Show Assistant Editor' to display the results of the println.

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


        switch (n) {


        case _ where n%3 == 0 && n%5 == 0:
                println("FizzBuzz")


        case _ where n%3 == 0:
                println("Fizz")


        case _ where n%5 == 0:
                println("Buzz")


        default:
                println(n)
        }


        }
```

An example of FizzBuzz being written from the ground up using Test-Driven Development and Ruby: http://youtu.be/CHTep2zQVAc

```
        n = 1
        while n < 101:
        if n%3 == 0 and n%5 == 0:
                print "fizzbuzz"


        else if n%3 == 0:
                print "fizz"


        else if n%5 == 0:
                print "buzz"


        else:
                print n


        n += 1
```

Actually another way to do this is...

```
        boolean flag = true;


                for(int i=0;i<16;i++){
                        if(i%3==0){
                                System.out.print("Fizz");
                                flag=false;
                        }


                        if(i%5==0){
                                System.out.print("Buzz");
                                flag=false;
                        }


                        if (flag)
                                System.out.print(i);


                        System.out.print(",");


                        flag = true;


                }
```

[Moved the above from the FizzBuzz page to here on 2014-07-07.]

```
 //Same bit mask idea but in C
 //Patrick
 #include <stdio.h>
 #include <string.h>


 main()
 {
        //FizzBuzz in C with bit approach
        // 00 neutral
        // 01 fizz
        // 10 buzz
        // 11 fizzbuzz
        int i=1;
        for (i=1; i<101; i++){
        int x=0;
        if (!(i%3)){x = x | 1;} // 01
        if (!(i%5)){x = x | 2;} // 10
        switch(x){
        case 1:  printf("fizz\n");break;
        case 2:  printf("buzz\n");break;
        case 3:  printf("fizzbuzz\n");break;
        default: printf("%d\n",i);
        }
        }


 }
```

```
 // Another bit-mask in C
 // Gary


 #include <stdio.h>
 #include <stdlib.h>


 int main()
 {
        int i=0;
        printf("FizzBuzz!\n----------------\n");
        for(i=1; i<=100; ++i)
        {
        int flag = (i%3==0) | (i%5==0)<<1;


        if (flag==0 )
                printf("%d", i);
        if (flag & 1)
                printf( "Fizz");
        if (flag & 2)
                printf( "Buzz");


        putchar('\n');
        }


        return 0;
 }
```

---

public class FuzzBuzz? {

```
        public static void main(String [] args){


                for(int i = 1 ; i < 101; i++){



                        if(i%5==0 && i%3 ==  0){
                                System.out.println("Fizzbizz");
                        }
                        else if(i%3 == 0){
                                System.out.println("Fizz");
                        }
                        else if(i%5 ==0){
                                System.out.println("Bizz");
                        }
                        else{
                                System.out.println(i);
                        }
                }
        }
```

}

Python... for i in range(1,101):

```
        buzz = (not i % 5) * "Buzz"
        fizz = (not i % 3) * "Fizz"


        if fizz or buzz:
        print fizz+buzz
        else:
        print i
```

Scala again, this one with only 3 tests. Here is where I'm missing those ternary ifs - jmt

1 to 100 map { n =>

```
        if ((if (n % 3 == 0) {print("Fizz");1} else 0) + (if (n % 5 == 0) {print("Buzz");1} else 0) == 0)
        print(n)
        println()
```

}

JBoss Drools Rules implementation:

```
        package sandbox;


        // I N T E R N A L      F A C T S  ==================================================
        // - Drools will create a POJO with getters, setters, constructors and
        //      correctly implemented hashcode/equals


        declare FizzBuzz
        count : int
        end


        declare Message
        text : String
        end


        // R U L E S =====================================================================


        dialect "mvel"


        rule "Start FizzBuzz"
        when
                exists Integer()
        then
                insert(new FizzBuzz(1));
        end


        rule "Count is divisible by 3, and not by 5"
        when
                $fizzBuzz : FizzBuzz(count % 3 == 0, count % 5 != 0)
        then
                insert(new Message("Fizz"));
        end


        rule "Count is divisible by 5, and not by 3"
        when
                $fizzBuzz : FizzBuzz(count % 3 != 0, count % 5 == 0)
        then
                insert(new Message("Buzz"));
        end


        rule "Count is divisible by 3 and by 5"
        when
                $fizzBuzz : FizzBuzz(count % 3 == 0, count % 5 == 0)
        then
                insert(new Message("FizzBuzz"));
        end


        rule "Count is not divisible by 5 or by 3"
        when
                $fizzBuzz : FizzBuzz(count % 3 != 0, count % 5 != 0)
        then
                String value = Integer.toString($fizzBuzz.count);
                insert(new Message(value));
        end


        rule "Print FizzBuzz counter"
        when
                $message : Message()
                $fizzBuzz : FizzBuzz()
        then
                System.out.println($message.getText());
                retract($message)
        end


        rule "Increment the FizzBuzz counter last"
        salience -1
        when
                $countUpTo : Integer()
                $fizzBuzz : FizzBuzz(count <= $countUpTo)
        then
                $fizzBuzz.count = $fizzBuzz.count + 1;
                update($fizzBuzz);
        end



        rule "At the end remove all facts"


        when
                $countUpTo : Integer()
                $fizzBuzz : FizzBuzz(count > $countUpTo)
        then
                retract($countUpTo);
                retract($fizzBuzz);
        end
```

Here's one in EmacsLisp. Do M-x fizz-buzz. The output will go to the *fizz-buzz* buffer.

```
 (defun fizz-buzz ()
        (interactive)
        (with-output-to-temp-buffer "*fizz-buzz*"
        (dotimes (i 100)
        (cond ((and (eq 0 (% i 3)) (eq 0 (% i 5))) (princ "FizzBuzz"))
        ((eq 0 (% i 3)) (princ "Fizz"))
                ((eq 0 (% i 5)) (princ "Buzz"))
                (t (princ i)))
        (princ "\n"))))
```

Another simple one in C#.

for (int i = 1; i <= 100; i++) {

```
        if (i % 3 == 0) Console.Write("Fizz");
        if (i % 5 == 0) Console.Write("Buzz");
        if ((i % 3 != 0) && (i % 5 != 0)) Console.Write(i.ToString?());
        Console.Write("\n");
```

}

A simple java program for fizz buzz will be ---

import com.google.common.base.Strings; import com.sun.org.apache.xalan.internal.xsltc.compiler.util.Util;

class FizzBuzz{

public static void main (String[] args){

```
  FizzBuzz fb = new FizzBuzz();
```

for(int i =1; i <=100; i++){

```
                String result = fb.printFizzBuzz(i);
                if(Strings.isNullOrEmpty(result))
                Util.println(""+i);
                else
                Util.println(result);
        }
```

}

```
        public String printFizzBuzz(int i){
        String fizzbuzz="";


        fizzbuzz = fizzbuzz+fizz(i);
        fizzbuzz = fizzbuzz + buzz(i);


        return fizzbuzz;


        }


        public String fizz(int i){


         if(i%3 ==0)
                return "FIZZ";
         else
                return "";
        }


        public String buzz(int i) {


        if(i%5 == 0)
```

```
                return "BUZZ";
        else
                return "";
        }
```

}

FizzBuzz Implementation in Python by Shubhamoy

'''This code also ensures that those numbers which are divisible by 3 and 5 should only output "fizzbuzz" instead of three outputs'''

for i in range(1, 100):

```
        if(i%15==0):
                print "fizzbuzz"


        if(i%3==0):


if(i%15!=0)



                        print "fizz"


        if(i%5==0):
                if(i%15!=0):
                        print "buzz"


 - Is this printing the iteration count if it doesn't print "fizz", "buzz" or "fizzbuzz"? - jmt
```

Scala yet again, this time with 75% more obfuscation using both an implicit and anonymous functions - jmt

```
        val printIt: List[(Int => Unit)] = List(i => println(i), i => println("Fizz"), i => println("Buzz"), i => println("FizzBuzz"))
        implicit class IntModBool?(me: Int) { def %%(i: Int) = if (me % i == 0) 1 else 0 }
        (1 to 100) map { n =>
                printIt(n %% 3 + n %% 5 * 2)(n)
        }
```

Another php solution which avoids an explicit loop. It uses the array_map function to call an anonymous function, having used the range function to create and array with elements containing the numbers 1 to 100.

Not really readable though so just for amusement.

```
        array_map(function($n){echo(($n%15)?($n%5)?($n%3)?$n:'Fizz':'Buzz':'FizzBuzz')."\r\n";},range(1,100));
```

And a version that avoid both an explicit loop and any explicit modulus calculations

```
        array_map(function($n){$t=substr(base_convert($n,10,3),-1);$f=substr(base_convert($n,10,5),-1);switch(true){case $t==0&&$f==0:echo 'FizzBuzz';break;case $t==0:echo 'Fizz';break;case $f==0:echo 'Buzz';break;default:echo $n;}echo "\r\n";},range(1,100));
```

FizzBuzz for the 6502 by barrym95838 2013.04.04 https://github.com/acmeism/RosettaCodeData?/blob/master/Task/FizzBuzz/6502-Assembly/fizzbuzz.6502

```
                .lf  fzbz6502.lst
                .cr  6502
                .tf  fzbz6502.obj,apl
        ;------------------------------------------------------
        ; 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
```

VBA Functions by Gary Lee

With If statements

Function FBuzz(lngNum As Long) As String

```
        If lngNum = 0 Then Exit Function
        If lngNum Mod 3 = 0 Then FBuzz = "Fizz"
        If lngNum Mod 5 = 0 Then FBuzz = FBuzz & "Buzz"
        If Len(FBuzz) = 0 Then FBuzz = lngNum
```

End Function

With Nested Case Select

Function FBuzz2(lngNum As Long) As String

```
        If lngNum = 0 Then Exit Function
        FBuzz2 = lngNum
        Select Case lngNum Mod 3
        Case 0: FBuzz2 = "Fizz"
                Select Case lngNum Mod 5: Case 0: FBuzz2 = FBuzz2 & "Buzz": End Select
        Case Else: Select Case lngNum Mod 5: Case 0: FBuzz2 = "Buzz": End Select
        End Select
```

End Function

Here is a simple C one-liner:

```
        for( int i = 1; i <= 100; i++ ) printf( "%d\n\0__Fizz\n\0Buzz\n\0FizzBuzz\n"+(6 * (((i%5)==0)<<1 | ((i%3)==0))), i );
```

- Good* programmers can easily recognize the 4 if-then conditionals. The one liner is just a compact and obtuse version of the canonical verbose version:

```
        const char *aState[4] = {
         "%d\n" // Default
        ,"Fizz\n"       // x %  3
        ,"Buzz\n"       // x %  5
        ,"FizzBuzz\n" // x % 15
        };
        for( int i = 1; i <= 100; i++ )
        {
        int iState =
                (((i % 5) == 0) << 1) |
                (((i % 3) == 0) << 0) ;
        printf( aState[ iState ], i );
        }
```

(*) The purpose of this test is to determine if you _are_ a good programmer! :-) -- Michael Pohoreski

---

Pretty clean LiveScript version:

```
        for i from 1 to 100
                output = ''
                if i % 3 is 0 then output += 'Fizz'
                if i % 5 is 0 then output += 'Buzz'
                if not output then output = i
                console.log output
```

-- farzher

This shell version implements (on POSIX-compatible systems) the assignment exactly as specified, unlike all of the other solutions presented:

```
        #!/bin/sh
        yes 'FizzBuzz' | tr -d '\n'
```

Note that the assignment specified numbers, not integers, from 1 to 100, of which there are an infinite number when real numbers are included, and, similarly, the assignment specified multiples of 3 and 5, not integer multiples. It also did not specify that newlines separate each answer.

-- some guy I know in pedantic mode

PHP version using the ternary operator

$index = 0;

while ($index++ < 100) {

```
        $word   = ($index % 3) ? '' : 'Fizz';
        $word   .= ($index % 5) ? '' : 'Buzz';
        print ($word) ? $word.',' : $index.',';
```

}

-- cbdsteve

Scala one-liner, similar to the C one-liner above - jmt

```
        for (n <- 1 to 100) println(List((15, "FizzBuzz"), (3, "Fizz"), (5, "Buzz")).find(t => n % t._1 == 0).getOrElse((0, n.toString))._2)
```

After looking at so many answers, I figured, I could do another in less code than most using: 1. for loop 2. modulous operator 3. print the answers Regards, Emiliano Gaytan

//FizzBuzz Answer to problem //Answer: using modulus operator, all too common across languages, simply test for remainder == 0 and print the Fizz or Buzz pers specs

```
        for($i=1; $i<=100; $i++) {
                $threemultiple          = $i%3; //find remainder  $i div by 3
                $fivemultiple           = $i%5; //find remainder  $i div by 3
                if( $threemultiple      == 0) { printf ("Fizz");  } //Spec: print if multiple of three
                if( $fivemultiple       == 0) { printf ("Buzz");  } //spec: print if multiple of five
                if ($threemultiple == 0 || $fivemultiple == 0 ) {  echo( "  at number " . $i . " <br>");  }     // add line break if div by either 3 or 5
```

Quick Factor solution, after all the idea was to do time in a short time span. It could be more compact with more thought but since this code only runs once, why bother

: is3 (n -- n) [3 mod] keep [drop "fizz"] unless ;

: is5 (n -- n) [5 mod] keep [drop "buzz"] unless ;

: is15 (n -- n) [15 mod] keep [drop "fizzbuzz"] unless ;

100 iota [is15 [print] [is5 [print] [is3 [print] [.] if] if] if] each

Dave Carlton

CategoryNone

Shortest Ruby version (shortest any language?) yet, I think at 64 characters including the puts:

```
        puts (1..100).map{|i|r=["Fizz"][i%3];r="#{r}Buzz"if i%5==0;r||i}
```

Was recently asked a variant of this in an interview. Did NOT answer with this version.

Uses the following 'interesting' Ruby features:

- Array index past the end gives nil
- "#{nil}" is empty string
- nil is falsey and any string is truthy
- puts outputs arrays with one entry per line

Don't use it in an interview, you'd probably get the stink-eye... ;)

-- mrs

C version using a single puts statement and nested ternary operators. the order of the conditional tests has been roughly optimized for faster execution

#include <stdio.h>

int main() {

```
        int i;
        char buf[3];


        for (i = 1; i <= 100; i++)
        puts( ( (i % 3) && (i % 5) ) ? snprintf(buf, 3, "%d", i), buf : !(i % 3) ?
                !(i % 5) ? "FizzBuzz" : "Fizz" : "Buzz");


        return 0;
```

}

- Aron Dennen

Here's one written in Lua

for i = 1, 100 do

```
        if (i % 3 == 0) then
                io.write("Fizz")
        elseif (i % 5 == 0) then
                io.write("Buzz")
        else
                io.write(i)
        end


        io.write("\n")
```

end

C++ version using a stringstream

#include <iostream> #include <sstream>

int main() {

```
        std::stringstream ss;
        for (int i = 1; i != 101; ++i) {
                std::cout << (i % 3 == 0 && i % 5 == 0 ? "FizzBuzz" : (i % 3 == 0 ? "Fizz" : (i % 5 == 0 ? "Buzz" : (ss << i, ss.str())))) << std::endl;
                ss.str("");
        }


        return 0;
```

}

C version

#include <stdio.h>

int main() {

```
        for (int i = 1; i != 101; ++i) {
                if (i % 3 == 0 && i % 5 == 0)
                        printf("FizzBuzz\n");
                else if (i % 3 == 0)
                        printf("Fizz\n");
                else if (i % 5 == 0)
                        printf("Buzz\n");
                else
                        printf("%d\n", i);
        }


        return 0;
```

}

-- Altenius

vb.net using select case

Module Module1

```
    Sub Main()
        For i = 1 To 100
            Select Case 0
                Case i Mod 15
                    Console.WriteLine?("FizzBizz?" & vbNewLine)
                Case i Mod 5
                    Console.WriteLine?("Bizz" & vbNewLine)
                Case i Mod 3
                    Console.WriteLine?("Fizz" & vbNewLine)
                Case Else
                    Console.WriteLine?(i.ToString? & vbNewLine)
            End Select
        Next
        Console.ReadLine?()
    End Sub
```

End Module

-- Marius

View edit of December 23, 2014 or FindPage with title or text search