```
'Hello world!'
printNl. for n in{1..100}; do ((( n % 15 == 0
)) && echo 'FizzBuzz') || ((( n % 5 == 0 )) && echo 'Buzz') ||
((( n % 3 == 0 )) && echo 'Fizz') || echo $n; done #include<stdio.h> int
main(){printf("\nHello world!");return 0;} import
Data.List (nub, permutations)      check f =
length . nub . zipWith f [0..] generate n = filter
(\x -> check (+) x == n && check (-) x == n)
$ permutations var s = "Hello, world!"; var
byteCount = s.length * 2  [0..n-1] function
powerset{T}(x::Vector{T} ) result = Vector{T}     [ [] ] for elem
in x, j in eachindex(result)push!(result, [result     [j] ; elem] ) end result end
error 'Goodbye, World!' lists:foldl(fun(X,Y) ->     X*Y end, 1, lists:seq(1,N)). sub factors
(Int $n) { squish sort ($_, $n div $_ if $n %%     $_ for 1 .. sqrt $n)}  fun fizzBuzz() { println
((1..100).map{i->mapOf(0 to i,i%3 to "Fizz",i%     5 to "Buzz",i%15 to     "FizzBuzz")[0]})}  from cmath import exp, pi def fft
(x): N = len(x) if N <= 1: return x even = fft(x     [0::2]) odd =  fft(x[1::2]) T= [exp     (-2j*pi*k/N)*odd[k] for k in range(N//2)] return [even
[k] + T[k] for k in range(N//2)] + [even[k] - T[k]     for k in range(N//2) ]   echo "Hello,     world!" | awk '{ print length($0) } Write-Host 'Hello
world!'object FizzBuzz extends App {1 to 100     foreach { n => println((n % 3, n %     5) match {case (0, 0) => "FizzBuzz" case (0, _) =>
"Fizz" case (_, 0) => "Buzz"case _ => n})}} def     solve(n, i, a, b, c): if i < n:   for j in     range(n): if j not in a and i+j not in b and i-j not in c:
for solution in solve(n, i+1, a+[j], b+[i+j], c+[i-     j]): yield solution else: yield a cat     ("Goodbye, World!", file=stderr())    template <int N>
struct Factorial { enum { value = N * Factorial<N     - 1>::value };}; template <> struct     Factorial<0> {enum { value = 1 };}; def fft(vec)
return vec if vec.size <= 1 evens_odds =     vec.partition.with_index{|_,i| i.even?}     evens, odds = evens_odds.map{|even_odd| fft
(even_odd)*2} evens.zip(odds).map.with_index     do |(even, odd),i| even + odd * Math::E **     Complex(0, -2 * Math::PI * i / vec.size) end end
(eprintf "Goodbye, World!\n") function( factorial     var n)  set( product 1)  foreach( i RANGE 2     ${n}) math(EXPR product "${product} * ${i}")
endforeach(i) set(${var} ${product}     PARENT_SCOPE)endfunction(factorial)     signature STACK = sig type 'a stack exception
EmptyStack val empty : 'a stack val isEmpty : 'a     stack -> bool val push : ('a * 'a stack) -> 'a
stack val pop  : 'a stack -> 'a stack val top  : 'a     stack -> 'a val popTop : 'a stack -> 'a stack
* 'a val map : ('a -> 'b) -> 'a stack -> 'b stack val     app : ('a -> unit) -> 'a stack -> unit end
using System; class Program{static void Main(string     [] args){var cube = new Func<double,
double>(x => Math.Pow(x, 3.0));var croot = new     Func<double, double>(x => Math.Pow(x, 1 /
3.0)); var functionTuples = new[] {(forward:     Math.Sin, backward: Math.Asin), (forward:
Math.Cos, backward: Math.Acos),(forward:     cube,backward: croot)};foreach (var ft in
functionTuples){Console.WriteLine(ft.backward     (ft.forward(0.5)));}}} :- write('Hello world!'), nl.
public static ArrayList<String> getpowerset(int a[],int     n,ArrayList<String> ps) {if(n<0) { return null; } if
(n==0) {if(ps==null) ps=new ArrayList<String>();     ps.add(" ");return ps;}ps=getpowerset(a, n-1, ps);
ArrayList<String> tmp=new ArrayList<String>(); for     (String s:ps) {if(s.equals(" ")) tmp.add(""+a[n-1]); else
s=[[io.write('s=[','[',s,']',']';',s)]];io.write('s=     [','[',s,']',']';',s) tmp.add(s+a[n-1]);} ps.addAll(tmp);
return ps;} fn main() {println!("Hello world!");}     FUNCTION factorial (n AS Integer) AS Integer IF n < 2 THEN
factorial = 1 ELSE factorial = n * factorial(n-1) END IF     END FUNCTION package main import "fmt" func main()
{ a := "package main\n\nimport \"fmt\"\n\nfunc main     () {\n\ta := %q\n\tfmt.Printf(a, a)\n}\n" fmt.Printf(a, a)}
function rk4(f::Function, x0::Float64, y0::Float64,     x1::Float64, n) vx = Vector{Float64}(n + 1) vy = Vector
{Float64}(n + 1) vx[1] = x = x0 vy[1] = y = y0 h =     (x1 - x0) / n for i in 1:n k1 = h * f(x, y) k2 = h * f(x + 0.5h,
y + 0.5k1) k3 = h * f(x + 0.5h, y + 0.5k2) k4 = h * f     (x + h, y + k3) vx[i + 1] = x = x0 + i * h vy[i + 1] = y = y
+ (k1 + 2k2 + 2k3 + k4) / 6 end return vx, vy end     <?php for ($i = 1; $i <= 100; $i++) {if (!($i % 15)) echo
"FizzBuzz\n"; else if (!($i % 3)) echo "Fizz\n"; else if     (!($i % 5)) echo "Buzz\n"; else echo "$i\n";}?> (display
"Hello world!")(newline) (q.tex \output {\message     {\output \the \output \end }\batchmode }\end  SELECT
'Hello world!' text FROM dual let longest xs ys = if
List.length xs > List.length ys then xs else ys let rec
lcs a b = match a, b with [], _ | _, [] -> [] | x::xs,     y::ys
-> if x = y then x :: lcs xs ys else longest (lcs a ys)     (  lcs   xs
b);(printf "str has ~a characters" (string-length str))     fun qs (_, _,
[]) = raise Fail "empty" | qs (k, cmp, x :: xs) = let     val (ys, zs) =
List.partition (fn y => cmp (y, x) = LESS) xs val l =     length ys  in if k <
l then qs (k, cmp, ys) else if k > l then qs (k-l-1, cmp,     zs)  else  x  end    fn
main() {let x = "fn main() {\n    let x = "; let y =     "print! (\"{}{:?};\n
let y = {:?};\n    {}\", x, x, y, y)\n}\n"; print!("{}     {:?}; let y = {:?}; {}", x,
x, y, y)} defmodule LCS do def lcs(a, b) do lcs     (to_charlist(a), to_charlist
(b), []) |> to_string end defp lcs([h|at], [h|bt], res), do:     lcs(at, bt, [h|res]) defp lcs([_|
at]=a, [_|bt]=b, res) do Enum.max_by([lcs(a, bt, res), lcs     (at, b, res)], &length/1) end defp
lcs(_, _, res), do: res |> Enum.reverse end     100   format     (5X,A,"!") print 100,"Hello world!" foi=: [:
I. 0 = (|~ i.@>:) 1 1 100 {/c false def dup     3 mod 0 eq { (Fizz) print /c true def } if dup 5 mod 0 eq { (Buzz) print /c
true def } if c {pop}{(   ) cvs print}     ifelse (\n) print} for Y <- function(f) {(function(x) { (x)(x) })( function
(y) { f( (function(a) {y(y)})(a) ) } )}     function gcd_recursive(u, v: longint): longint; begin if u mod v <> 0
then gcd_recursive :=     gcd_recursive(v, u mod v) else gcd_recursive := v; end; sub ms
{my @x = @_; return @x if @x     < 2; my $m = int @x / 2; my @a = ms(@x[0 .. $m - 1]);
my @b = ms(@x[$m ..     $#x]); for (@x) {$_ = !@a ? shift @b : !@b ? shift
@a : $a[0] <= $b[0] ?     shift @a : shift @b;}@x;} program
Project1; {$APPTYPE     CONSOLE} begin WriteLn
(ErrOutput,
'Goodbye,
World!');
end. println
("Hello
world!") LenB
(string|
```