```
(take 5 sense-codons)                          ; => (#{"ACC" "CCA" "CAC"}
                                               ;     #{"GCC" "CGC" "CCG"}
                                               ;     #{"CAA" "ACA" "AAC"}
                                               ;     #{"CTC" "CCT" "TCC"}
                                               ;     #{"AGC" "CAG" "GCA"})

(def sense (map first sense-codons))
(count sense)   ; => 20
(take 7 sense)  ; => ("ACC" "GCC" "CAA" "CTC" "AGC" "TAT" "GAG")
(def nonsense (remove (set sense) (set triples)))
(count nonsense)                               ; => 44

(def code (zipmap sense (keys amino-acids)))
(sort-by second code)                          ; => (["AGC" :A]
                                               ;     ["TAA" :C]
                                               ;     ["TGT" :D]
                                               ;     ["TCA" :E]
                                               ;     ["TAT" :F]
                                               ;     ["TAG" :G]
                                               ;     ["CTG" :H]
                                               ;     ["CAA" :I]
                                               ;     ["ACG" :K]
                                               ;     ["ACC" :L]
                                               ;     ["GCC" :M]
                                               ;     ["GAA" :N]
                                               ;     ["CTA" :P]
                                               ;     ["GAT" :Q]
                                               ;     ["CTC" :R]
                                               ;     ["TCG" :S]
                                               ;     ["CTT" :T]
                                               ;     ["GTG" :V]
                                               ;     ["GAG" :W]
                                               ;     ["GGC" :Y])

(map string (take 3 (partition 3 1 strand))) ; => ("TTA" "TAA" "AAT")

(def amino-keys (map (comp code string) (partition 3 1 strand)))
(take 13 amino-keys) ; => (nil :C nil nil nil :Y nil nil :M nil nil :Y :M)

(def aminos (map amino-acids (remove nil? amino-keys)))
(take 4 aminos) ;=> ("cysteine" "tyrosine" "methionine" "tyrosine")
```

This is fun programming but (as of 1961) bad biology.

Biology is not physics and certainly not engineering.
Life's processes are rarely efficient and often messy.
(Crick and Gamow were both degreed in physics.)

What is this program about? Can you follow it?

This programming language is as old as this paper.
It was designed to help machines help us think.

It is almost the simplest language that can work.
So the program is about the world more than itself.

A programming language can be a tool of discovery
and not just the means of commanding a computer.

You can develop and explore a program while it runs.
And see the value of each expression as it's evaluated.
They appear here literally as the computer prints them.
("Notebook systems" like Jupyter return to this idea.)

How do you think with your programming language?
Does it aid or hinder your understanding of the world?

(-: BTW: Did you notice any commas in this code? :-)

CODES WITHOUT COMMAS

BROAD
INSTITUTE

BROAD INSTITUTE

CODES WITHOUT COMMAS

By F. H. C. Crick, J. S. Griffith, and L. E. Orgel

MEDICAL RESEARCH COUNCIL UNIT, CAVENDISH LABORATORY, AND DEPARTMENT OF THEORETICAL CHEMISTRY, CAMBRIDGE, ENGLAND

Communicated by G. Gamow, February 11, 1957

This paper deals with a mathematical problem which arose in connection with protein synthesis. We present the solution here because it gives the "magic number" 20, so that our answer may perhaps be of biological significance. To make this clear, we sketch in the biochemical background first.

It is assumed in one of the more popular theories of protein synthesis that amino acids are ordered on a nucleic acid strand (see, for example, Dounce[1]) and that the order of the amino acids is determined by the order of the nucleotides of the nucleic acid. There are some twenty naturally occurring amino acids commonly found in proteins, but (usually) only four different nucleotides. The problem of how a sequence of four things (nucleotides) can determine a sequence of twenty things (amino acids) is known as the "coding" problem.

This problem is a formal one. In essence, it is not concerned with either the chemical steps or the details of the stereochemistry. It is not even essential to specify whether RNA or DNA is the nucleic acid being considered. Naturally, all these points are of the greatest interest, but they are only indirectly involved in the formal problem of coding.

The first definite proposal was made by Gamow.[2] His code, which was suggested by the structure of DNA, was of the "overlapping" type. The meaning of this is illustrated in Figure 1. Gamow's code was also "degenerate"—that is, several sets of three letters (picked in a special way) stood for a particular amino acid. However, all the 64 ($4 \times 4 \times 4$) possible sets of three letters stood for one amino acid or another, so that any sequence whatever of the four letters stood for a definite sequence of amino acids.

It is easy to see that codes of the overlapping type impose severe restrictions on the allowed amino acid sequences. Unfortunately, no such restrictions have been found, although considerable (unpublished) efforts have been made, by a number of workers, to find them. Part of this work has been reviewed by Gamow, Rich,


```
(ns commas
  "CODES WITHOUT COMMAS -- with apologies to F.H.C. Crick et al")

(comment "http://www.pnas.org/content/43/5/416" is the paper.
         The year is 1957. What do we know now?)

(def nucleotides ["Adenine" "Cytosine" "Guanine" "Thymine"])
nucleotides ; => ["Adenine" "Cytosine" "Guanine" "Thymine"]
(count nucleotides)                              ; => 4

(def essential   {:F "phenylalanine"
                  :H "histidine"
                  :I "isoleucine"
                  :K "lysine"
                  :L "leucine"
                  :M "methionine"
                  :T "threonine"
                  :V "valine"
                  :W "tryptophan"})

(def conditional {:C "cysteine"
                  :G "glycine"
                  :P "proline"
                  :Q "glutamine"
                  :R "arginine"
                  :Y "tyrosine"})

(def dispensable {:A "alanine"
                  :D "aspartic acid"
                  :E "glutamic acid"
                  :N "asparagine"
                  :S "serine"})

(def amino-acids (merge essential conditional dispensable))
(count amino-acids)                              ; => 20

"The problem of how a sequence of four things (nucleotides)
 can determine a sequence of twenty things (amino acids)
 is known as the 'coding' problem."

(comment Now ... given that. What can we find out?)
```

Read the => in this comment as "evaluates to".

This is fun programming but (as of 1961) bad biology.

Biology is not physics and certainly not engineering.
Life's processes are rarely efficient and often messy.
(Crick and Gamow were both degreed in physics.)

What is this program about? Can you follow it?

This programming language is as old as this paper.
It was designed to help machines help us think.

It is almost the simplest language that can work.
So the program is about the world more than itself.
A programming language can be a tool of discovery
and not just the means of commanding a computer.

You can develop and explore a program while it runs.
And see the value of each expression as it's evaluated.
They appear here literally as the computer prints them.
("Notebook systems" like Jupyter return to this idea.)

How do you think with your programming language?
Does it aid or hinder your understanding of the world?

(-: BTW: Did you notice any commas in this code? :-)

```
(take 5 sense-codons)                    ; => (#{"ACC" "CCA" "CAC"}
                                         ;     #{"GCC" "CGC" "CCG"}
                                         ;     #{"CAA" "ACA" "AAC"}
                                         ;     #{"CTC" "CCT" "TCC"}
                                         ;     #{"AGC" "CAG" "GCA"})

(def sense (map first sense-codons))
(count sense)    ; => 20
(take 7 sense)  ; => ("ACC" "GCC" "CAA" "CTC" "AGC" "TAT" "GAG")
(def nonsense (remove (set sense) (set triples)))
(count nonsense)                         ; => 44

(def code (zipmap sense (keys amino-acids)))
(sort-by second code)                    ; => (["AGC" :A]
                                         ;     ["TAA" :C]
                                         ;     ["TGT" :D]
                                         ;     ["TCA" :E]
                                         ;     ["TAT" :F]
                                         ;     ["TAG" :G]
                                         ;     ["CTG" :H]
                                         ;     ["CAA" :I]
                                         ;     ["ACG" :K]
                                         ;     ["ACC" :L]
                                         ;     ["GCC" :M]
                                         ;     ["GAA" :N]
                                         ;     ["CTA" :P]
                                         ;     ["GAT" :Q]
                                         ;     ["CTC" :R]
                                         ;     ["TCG" :S]
                                         ;     ["CTT" :T]
                                         ;     ["GTG" :V]
                                         ;     ["GAG" :W]
                                         ;     ["GGC" :Y])

(map string (take 3 (partition 3 1 strand))) ; => ("TTA" "TAA" "AAT")

(def amino-keys (map (comp code string) (partition 3 1 strand)))
(take 13 amino-keys) ; => (nil :C nil nil nil :Y nil nil :M nil nil :Y :M)

(def aminos (map amino-acids (remove nil? amino-keys)))
(take 4 aminos) ;=> ("cysteine" "tyrosine" "methionine" "tyrosine")
```