APL

El matemático Kenneth E. Iverson desarrolló, a partir de 1956, en el marco del programa de maestría de la Universidad de Harvard denominado *Procesamiento Automático de Datos* donde era profesor, una notación muy concisa para describir y analizar diversos temas dictados en ese programa. En 1960, Iverson dejó Harvard, pasó a trabajar en IBM y luego publicó *A Programming Language* (1962) y *Automatic Data Processing* (1963, en coautoría con Fred Brooks), dos libros en los que se utilizó la notación desarrollada en Harvard (por ejemplo, para describir la computadora IBM 7090, en el Capítulo 2 de *A Programming Language*). En 1964, Adin Falkoff junto con Iverson y Ed Sussenguth publicaron *A formal description of System/360*, un artículo en el que utilizaban esa misma notación para describir el hardware de la IBM System/360. Más tarde, la notación de Iverson fue implementada como un lenguaje de programación, teniendo el mayor impacto, a partir de 1966, la versión de tiempo compartido para la IBM System/360, denominada APL\360, cuyo manual fue publicado en 1968. El nombre del lenguaje APL es un acrónimo de *A Programming Language*. Desde entonces ha habido varios intentos de estandarización, destacándose las normas ISO 8485:1989 e ISO/IEC 13751:2001. Los dialectos de APL actuales incluyen sus propias extensiones para soportar la Programación Orientada a Objetos y la Programación Funcional, corren en múltiples plataformas y ofrecen, además, interoperabilidad con otros lenguajes y entornos, como .NET, JVM, R y Python.

1. GRAMÁTICA

Una versión básica de APL tiene (como mínimo) seis elementos gramaticales:

I) Sustantivos (*nouns*): Un sustantivo es un arreglo de rango 0 (un *escalar o ítem*, es decir, un número o un carácter alfabético literal), de rango 1 (un *vector* o *lista*) o de rango mayor o igual a 2 (una *matriz* o *tabla*).

Por ejemplo: `5`, `2.7`, `¯3`, `'R'`, `'r'`, `'$'` son escalares.

`'HOLA'` es un vector de caracteres (una cadena o *string*).

```
1 3 5
4 7 6
```
es una matriz de números.

II) Verbos (*verbs*): Un verbo es una *función* que actúa sobre uno o dos sustantivos y produce un nuevo sustantivo. A los sustantivos sobre los que actúan los verbos se los denomina *argumentos*. Un verbo puede ser una función monádica (actúa sobre un único argumento ubicado a su derecha) o diádica (actúa sobre dos argumentos: uno ubicado a su izquierda y otro a su derecha).

Por ejemplo: `-3` resulta `¯3`, porque la función monádica `-` significa *cambiar el signo*.

`2-3` resulta `¯1`, porque la función diádica `-` significa *restar*.

III) Adverbios (*adverbs*): Un adverbio es un *operador* (función de orden superior, por ejemplo `⌿` *reducción de columnas* o `\` *escaneo de filas*) que actúa sobre un verbo y produce un verbo derivado.

Por ejemplo: `+/ 1 3 5 7` resulta `16` porque el adverbio `/` (*reducción de filas*) actúa sobre el verbo `+` (*sumar*), produciendo el verbo derivado `+/` (*sumar las filas*)

`+⍀ 1 3 5 7` resulta `1 4 9 16` porque el adverbio `⍀` (*escaneo de columnas*) actúa sobre el verbo + (*sumar*), produciendo `+⍀` (*escanear las columnas sumando*)

`1 2 3 ∘.× 4 5 6 7` resulta
```
 4  5  6  7
 8 10 12 14
12 15 18 21
```
porque el adverbio `∘.` (*producto externo*) actúa sobre el verbo `×` (*multiplicar*), produciendo el verbo derivado `∘.×` (*multiplicar todos con todos*), que es la función *producto tensorial*.

IV) Conjunciones (*conjunctions*): Una conjunción es un *operador* (función de orden superior) que actúa sobre dos verbos y produce un nuevo verbo derivado.
Por ejemplo: `1 2 3 +.× 4 5 6` resulta `32` porque la conjunción `.` (*producto interno*) actúa sobre los verbos `+` (*sumar*) y `×` (*multiplicar*), produciendo el verbo derivado `+.×` (*sumar los productos*), que es la función *producto escalar*.

V) Ligamientos (*copulas*): Un ligamiento de APL, denotado mediante el símbolo ←, permite asignar un elemento gramatical, por ejemplo, un sustantivo, un verbo, etc. a un nombre o identificador.
Por ejemplo: `prodesc ← +.×`
`1 2 3 prodesc 4 5 6` resulta `32` (ahora `prodesc` equivale a `+.×`)

`prom ← {(+/ω)÷ρω}`
`prom 3 3 4` resulta `3.33333` (el verbo `prom` es ahora una función monádica)

`veces ← {αρω}`
`5 veces 'D'` resulta *DDDDD* (el verbo `veces` es ahora una función diádica)

VI) Puntuación (*punctuation*): La única puntuación de APL son los paréntesis, utilizados para forzar un orden de evaluación distinto del predeterminado.

2. PARTICULARIDADES

I) Tipos de los identificadores: No se declaran. Durante una sesión, un mismo identificador puede utilizarse varias veces, para referirse a un escalar, un vector, una matriz, etc. Por ejemplo:

```
   V ← 5
   V+2
7
   V ← 5 6 7
   V+2
7 8 9
```

II) Orden de evaluación: La asociatividad es de derecha a izquierda y todos los verbos tienen igual prioridad. Por ejemplo: `2×3-1` resulta `4` porque resuelve primero `3-1=2` y luego `2×2`
`(2×3)-1` resulta `5` porque resuelve primero `2×3=6` y luego `6-1`

III) Resultado de un ligamiento: Es el valor de su parte derecha. Por ejemplo:

```
   R ← ι D ← 5     Además de asignársele al identificador D el valor 5, el ligamiento vale 5
   D
5                  Por ello, el verbo ι se aplica sobre el sustantivo 5
   R
1 2 3 4 5          El sustantivo resultante (el vector 1 2 3 4 5) se asigna al identificador R
```

IV) Selección de ítems de un arreglo: Para la selección de ítems de un arreglo, se utilizan los corchetes `[` y `]`, dentro de los cuales, separados por el símbolo `;` se colocan los arreglos de posiciones correspondientes a cada dimensión del arreglo (si el arreglo es un vector, habrá un único arreglo de posiciones). Por ejemplo: `'ABC'[3 2 1]` resulta `'CBA'`. Si el arreglo de posiciones de alguna dimensión se omite, se seleccionan todas las posiciones de esa dimensión.

V) Tipos de los argumentos de las funciones diádicas: Los dos argumentos de las funciones diádicas pueden ser de tipos iguales o distintos. Si son dos escalares, el resultado es otro escalar. Si son un escalar y un arreglo, el resultado es un arreglo con los resultados de aplicar la función diádica entre el escalar y cada uno de los ítems del arreglo. Si son dos arreglos de igual dimensión, el resultado es un arreglo con los resultados de aplicar la función diádica entre los elementos homólogos de ambos arreglos. Si son dos arreglos de distintas dimensiones, el resultado puede ser indefinido o no, dependiendo de la función.

3. PRINCIPALES VERBOS (FUNCIONES PRIMITIVAS)

<table>
<tr><th>Verbo</th><th colspan="2">Función monádica</th><th colspan="2">Función diádica</th></tr>
<tr><td>+</td><td>Conjugate</td><td>Niega la parte imaginaria</td><td>Plus</td><td>2+5 → 7</td></tr>
<tr><td>−</td><td>Negative</td><td>-3 → ¯3</td><td>Minus</td><td>2-5 → ¯3</td></tr>
<tr><td>×</td><td>Signum</td><td>×¯3 → ¯1</td><td>Times</td><td>2×5 → 10</td></tr>
<tr><td>÷</td><td>Reciprocal</td><td>÷4 → 0.25</td><td>Divide</td><td>16÷5 → 3.2</td></tr>
<tr><td>|</td><td>Magnitude</td><td>|¯7 → 7</td><td>Residue</td><td>3|8 → 2</td></tr>
<tr><td>*</td><td>Exponential</td><td>*1 → 2.71828</td><td>Power</td><td>10*2 → 100</td></tr>
<tr><td>⍟</td><td>Natural logarithm</td><td>⍟2.71828 → 0.99999</td><td>Logarithm</td><td>2⍟1024 → 10</td></tr>
<tr><td>⌊</td><td>Floor</td><td>⌊6.3 → 6</td><td>Minimum</td><td>6⌊6.3 → 6</td></tr>
<tr><td>⌈</td><td>Ceiling</td><td>⌈6.3 → 7</td><td>Maximum</td><td>7⌈6.3 → 7</td></tr>
<tr><td>○</td><td>Pi times</td><td>○1 → 3.14159</td><td>Geometric (Sin, Cos…)</td><td>Varía según el 1º argumento</td></tr>
<tr><td>!</td><td>Factorial</td><td>!5 → 120</td><td>Binomial</td><td>m!n → (!n)÷(!m)×!(n-m)</td></tr>
<tr><td>?</td><td>Roll</td><td>→ Número natural al azar</td><td>Deal</td><td>→ Números naturales al azar</td></tr>
<tr><td>~</td><td>Not</td><td>~0 → 1</td><td>Without</td><td>3 1 4 5 ~ 5 1 → 3 4</td></tr>
<tr><td>∧</td><td></td><td></td><td>AND</td><td>1 1 0 ∧ 1 0 1 → 1 0 0</td></tr>
<tr><td>∨</td><td></td><td></td><td>OR</td><td>1 1 0 ∨ 1 0 1 → 1 1 1</td></tr>
<tr><td>⍲</td><td></td><td></td><td>NAND</td><td>1 1 0 ⍲ 1 0 1 → 0 1 1</td></tr>
<tr><td>⍱</td><td></td><td></td><td>NOR</td><td>1 1 0 ⍱ 1 0 1 → 0 0 0</td></tr>
<tr><td>=</td><td></td><td></td><td>Equal to</td><td>2=8 → 0</td></tr>
<tr><td>≠</td><td></td><td></td><td>Unequal</td><td>2≠8 → 1</td></tr>
<tr><td><</td><td></td><td></td><td>Less than</td><td>2<8 → 1</td></tr>
<tr><td>≤</td><td></td><td></td><td>Less than or equal to</td><td>2≤8 → 1</td></tr>
<tr><td>></td><td></td><td></td><td>Greater than</td><td>2>8 → 0</td></tr>
<tr><td>≥</td><td></td><td></td><td>Greater than or equal to</td><td>2≥8 → 0</td></tr>
<tr><td>,</td><td>Ravel</td><td>→ Argumento vectorizado</td><td>Catenate / Laminate</td><td>→ Argumentos concatenados</td></tr>
<tr><td>⍴</td><td>Shape</td><td>→ Vector con dimensiones</td><td>Reshape</td><td>5 ⍴ 1 2 → 1 2 1 2 1</td></tr>
<tr><td>⍳</td><td>Index generator</td><td>⍳5 → 1 2 3 4 5</td><td>Index of</td><td>8 5 1 3 ⍳ 5 → 2</td></tr>
<tr><td>↑</td><td></td><td></td><td>Take</td><td>4↑'AUTOBUS' → 'AUTO'</td></tr>
<tr><td>↓</td><td></td><td></td><td>Drop</td><td>4↓'AUTOBUS' → 'BUS'</td></tr>
<tr><td>⊤</td><td></td><td></td><td>Encode</td><td>2 2 2 2 ⊤ 13 → 1 1 0 1</td></tr>
<tr><td>⊥</td><td></td><td></td><td>Decode</td><td>2 ⊥ 1 1 0 1 → 13</td></tr>
<tr><td>∊</td><td></td><td></td><td>Member of</td><td>'ABC' ∊ 'CASO' → 1 0 1</td></tr>
<tr><td>⍋</td><td>Grade up</td><td>⍋'DABC' → 2 3 4 1</td><td></td><td></td></tr>
<tr><td>⍒</td><td>Grade down</td><td>⍒'DABC' → 1 4 3 2</td><td></td><td rowspan="4">Sea M la matriz:
M ← 2 4 ⍴ ⍳ 8
1 2 3 4
5 6 7 8</td></tr>
<tr><td>⍉</td><td>Transpose</td><td>→ Matriz transpuesta</td><td></td></tr>
<tr><td>⌽</td><td>Reverse</td><td>→ Reflejada por el eje vertical</td><td></td></tr>
<tr><td>⊖</td><td>Reverse first</td><td>→ Reflejada por el eje horizontal</td><td></td></tr>
<tr><td>/</td><td></td><td></td><td>Compress last →</td><td>0 1 1 0 / M → 2 3
6 7</td></tr>
<tr><td>⌿</td><td></td><td>0 1 ⌿ M → 5 6 7 8</td><td>← Compress first</td><td></td></tr>
<tr><td>\</td><td></td><td rowspan="2">1 0 1 ⍀ M → 1 2 3 4
0 0 0 0
5 6 7 8</td><td>Expand last →</td><td rowspan="2">0 1 1 0 1 0 1 \ M
→ 0 1 2 0 3 0 4
0 5 6 0 7 0 8</td></tr>
<tr><td>⍀</td><td></td><td>← Expand first</td></tr>
</table>

La lista anterior no es exhaustiva. Más detalles y ejemplos en: **http://tryapl.org**

4. EJEMPLO INTEGRADOR

```
      primos ← {(2 = +⌿ 0 = V ∘.| V) / V ← ⍳ ⍵}
      8 21 ⍴ primos 1000
  2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73
 79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181
191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307
311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433
439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571
577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701
709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853
857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

Realicemos el seguimiento de `primos 20` para ver, paso a paso, cómo trabaja la función `primos`.

```
      primos 20
2 3 5 7 11 13 17 19
```

1) Como `primos` es una función monádica, el valor del argumento lo toma `⍵`, que pasa a valer `20`

2) Se ejecuta `⍳ 20`

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

3) El vector anterior se le asigna a `V`, y el resultado del ligamiento es el propio vector, que más adelante será el argumento de la derecha del verbo `/` (*compress last*).

4) El verbo derivado `∘.|` (*los restos de dividir todos con todos*) se utiliza con los argumentos izquierdo y derecho valiendo `V`, lo que produce la siguiente matriz:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Los restos de dividir por 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	Los restos de dividir por 2
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2	Los restos de dividir por 3
1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0	Los restos de dividir por 4
1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0	Los restos de dividir por 5
1 2 3 4 5 0 1 2 3 4 5 0 1 2 3 4 5 0 1 2	Los restos de dividir por 6
1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6	Los restos de dividir por 7
1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4	Los restos de dividir por 8
1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2	Los restos de dividir por 9
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0	Los restos de dividir por 10
1 2 3 4 5 6 7 8 9 10 0 1 2 3 4 5 6 7 8 9	Los restos de dividir por 11
1 2 3 4 5 6 7 8 9 10 11 0 1 2 3 4 5 6 7 8	Los restos de dividir por 12
1 2 3 4 5 6 7 8 9 10 11 12 0 1 2 3 4 5 6 7	Los restos de dividir por 13
1 2 3 4 5 6 7 8 9 10 11 12 13 0 1 2 3 4 5 6	Los restos de dividir por 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 0 1 2 3 4 5	Los restos de dividir por 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4	Los restos de dividir por 16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 1 2 3	Los restos de dividir por 17
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 0 1 2	Los restos de dividir por 18
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 0 1	Los restos de dividir por 19
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0	Los restos de dividir por 20

5) Cada elemento de la matriz anterior se compara con el escalar `0` usando el verbo `=` (*equal to*):

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

6) El verbo derivado `+⌿` (*sumar las columnas*) se utiliza con la matriz anterior como argumento:

```
1 2 2 3 2 4 2 4 3 4 2 6 2 4 4 5 2 6 2 6
```

7) Cada elemento del vector anterior se compara con el escalar `2` usando el verbo `=` (*equal to*):

```
0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0
```

8) Utilizando el vector anterior como argumento de la izquierda, se aplica el verbo `/` (*compress last*) que había quedado pendiente en el punto 3, y se obtiene el siguiente vector que es el resultado de *primos* `20`:

```
2 3 5 7 11 13 17 19
```

Ejercicios

1. Resolver:

   a) 9 - 27
   b) 144 * 1 ÷ 2
   c) 144 * .5
   d) - (2 * 4) × (15 - 2 + 5) ÷ 0.5 + 1.5
   e) 2 * 2 * 3
   f) 1 ÷ 2 ÷ 3 ÷ 4 ÷ 5 ÷ 6
   g) 3 × 12 1 5 4 3
   h) ÷ 1 2 4 5 8
   i) 3 4 5 7 ⌊ 2 4 6 1
   j) 7 3 4 12 ⌈ 6
   k) ⌈ 3.1 ¯2.3 4 3.2 4.6 5.1
   l) 2 ! 6
   m) ! 4 3 5 6 7
   n) 2 3 ⍟ 8 9
   o) 3 | 1 5 4 7 9
   p) 3 | ¯5 ¯7 3 2 ¯4 6
   q) (14 ≥ ¯50) ∧ (15 ≤ 25)
   r) ~ 17 ≥ 3
   s) (17 = 15) ⍲ 3 > 15

2. Evaluar sucesivamente:

   a) *AREA* ← (*PI* ← 3.14159) × (*RADIO* ← 2 3 4 1) * 2
   b) *AREA*
   c) *LONG* ← *DOS* × *PI* × *RADIO*
   d) *DOS* ← 2
   e) *LONG*
   f) *RADIO* × × *RADIO* - 3

3. Evaluar:

   a) 2 × ⍳ 5
   b) ¯1 + 2 × ⍳ 6
   c) 2 ? 10
   d) ? 6 6 6 6 6
   e) (⍳ 5 × 2)
   f) ⍴ ⍳ 6
   g) ⍳ 0
   h) ⍴ ⍳ 0

4. Evaluar en secuencia:

   a) `A ← (1 + ι 3) , 3 + ι 3`
   b) `A [1 4]`
   c) `A [A]`
   d) `A [A , A]`
   e) `A [⌊A ÷ 2]`

5. Evaluar en secuencia:

   a) `B ← 'SIC TRANSIT' , 'GLORIA MUNDI'`
   b) `ρ B`
   c) `B [2 × ι 3]`
   d) `B [1 + (ρ B) - ι ρ B]`

6. Evaluar en secuencia:

   a) `A ← 2 3 4 4 3 5 6`
   b) `B ← 6 5 3 4 13`
   c) `ρ A , ρ B`
   d) `(ρ A , ρ B)`
   e) `ρ A , (ρ B)`
   f) `(ρ A) , (ρ B)`

7. Evaluar en secuencia:

   a) `4 5 ρ V ← 2 1 3 2 4 5 6 6 2 1`
   b) `T ← 3 3 4 ρ V`
   c) `, T`
   d) `ρ T`
   e) `ρ , T`

8. Evaluar:

   a) `(V ← 1 4 4 2 3) ι 1 2`
   b) `V ι 1 2 8 11`

9. Evaluar:

   a) `ρ 3`
   b) `ρ 4 6`
   c) `ρ (3 2) ρ 4`
   d) `ρ (2 5 2) ρ 1 2 3`
   e) `ρ ρ 3`
   f) `ρ ρ 4 6`
   g) `ρ ρ (3 2) ρ , 4`
   h) `ρ ρ (2 5 2) ρ 1 2 3`

10. Evaluar sucesivamente:

a) `X ← 1 + ⍳ 6`
b) `+ / X`
c) `× / X`
d) `- / X`
e) `÷ / X`
f) `⌊ / X`
g) `⌈ / X`
h) `≠ / X`
i) `> / X`
j) `< / X`
k) `= / 1 0 1 6 8`
l) `⍱ / 0 1 1 1 0 0`
m) `! / 2 4`

11. Escribir una única expresión para obtener:

a) El valor de la sumatoria $\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+...+\frac{1}{500^2}$

b) El valor de la sumatoria $\frac{1}{1^2}-\frac{1}{3^2}+\frac{1}{5^2}-...-\frac{1}{999^2}$

c) El promedio aritmético de todos los números del vector *A*.

d) El valor numérico del polinomio $1 + 2x + 3x^2 + 4x^3$

e) La media cuadrática del vector *V*. La fórmula es la siguiente: $V_{rms}=\sqrt{\frac{1}{n}\sum_{i=1}^{n}V_i^2}$

f) La cantidad de elementos que componen un arreglo *A* de cualquier dimensión.

g) El promedio entre el primer número positivo de un vector *V* y el último número negativo del mismo.

h) El máximo de *n* números al azar, menores o iguales a *n*, con repetición, perteneciendo *n* a los naturales.

12. Dado un vector *B* booleano (compuesto por ceros y unos), que denota un número en base 2, escribir una expresión que devuelva el mismo número expresado en base 10.

13. Dado un vector *V*, escribir una expresión que modifique el estado del mismo:

a) Eliminando todas las ocurrencias del menor de sus elementos.
b) Eliminando todas las ocurrencias del mayor de sus elementos.
c) Eliminando el *n*-ésimo elemento, siendo que *n* es una variable ya asignada con un número natural menor que la dimensión del vector *V*.

14. Escribir una expresión que, aplicada a una matriz A de dos dimensiones (plana):

    a) Dé como resultado,
        - si la matriz es cuadrada: `0`
        - si tiene más filas que columnas: `1`
        - si tiene más columnas que filas: `¯1`

    b) Elimine la primera fila y la última columna.

15. Escribir una expresión que genere una matriz cuadrada de orden N (natural), cuya diagonal principal esté formada por `0` (ceros), el triángulo inferior por `1` (unos) y el triángulo superior por `¯1` (menos uno).

16. Dada una matriz *M*, cuadrada de orden par, desarrollar una expresión que, aplicada a la misma, devuelva como resultado una matriz similar a la original pero con `0` (ceros) en las columnas pares.

17. Escribir en APL una expresión o una función que:

    a) Determine si un vector *V* es capicúa.

    b) Calcule, para un vector *V*, la productoria de los elementos menores que cierto número *N* dividido por la productoria de sus posiciones respectivas.

    c) Calcule la traza de una matriz *M* (la suma de los elementos de la diagonal principal).

    d) Produzca un desplazamiento (*shift*) de los elementos de un vector *V* hacia la derecha, en una cantidad *N* no negativa de posiciones, llenando con ceros a la izquierda.

    e) Verifique si un número *N* pertenece a un vector *V*.

    f) Determine el elemento mínimo entre los máximos por fila de una matriz *M*.

    g) Obtenga los números pares de un vector *V* que sean menores que el máximo.

    h) Obtenga los números que estén ubicados en las posiciones pares de un vector *V* y que sean menores que el máximo.

    i) Devuelva un vector con ceros intercalados entre los elementos del vector *V*.

    j) Devuelva los números impares del vector *V* que sean mayores que el primer elemento.

    k) Devuelva los elementos iguales ubicados en iguales posiciones de dos vectores *V* y *W*.

l) Elimine todos los múltiplos de 5 en un vector *V*.

m) Devuelva los 2 últimos múltiplos de 9 de un vector *V*, sabiendo que existen por lo menos 2 múltiplos de 9 en el mismo.

18. Sean los vectores *M*, *N* y *L*, que contienen conjuntos de ciudades, y las matrices *MN* y *NL*, que contienen las distancias entre cada ciudad de *M* y *N*, y entre cada ciudad de *N* y *L*, respectivamente. Escribir en APL una expresión que, a partir de estas dos matrices, genere una tercera matriz *ML* que contenga las mínimas distancias para ir de una ciudad de *M* a otra de *L*, pasando por alguna ciudad de *N*. En caso de que alguna de estas distancias supere el valor 10, se pondrá 10 en su lugar.