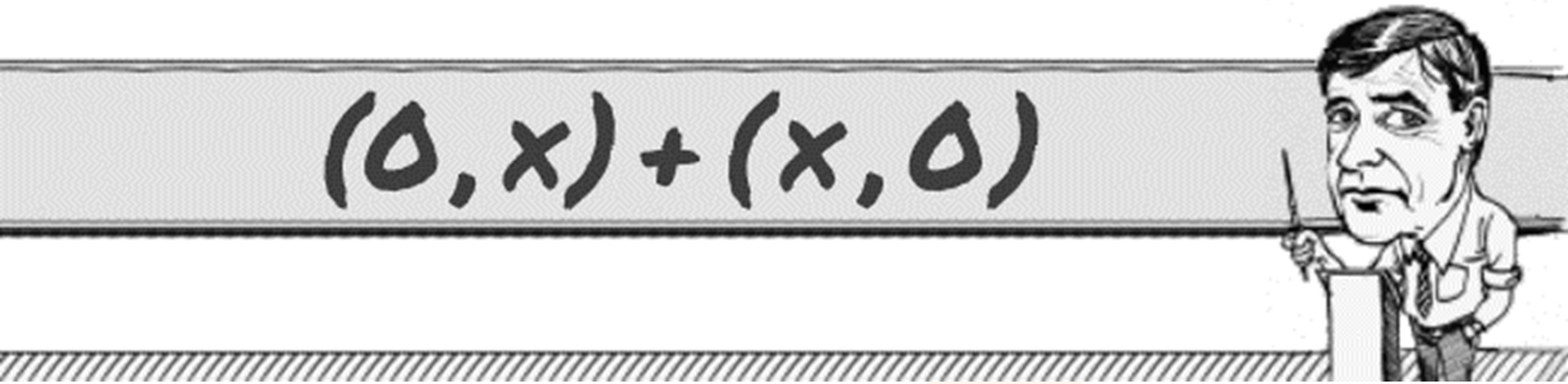
Ken Iverson, 1920–2020



Some Thoughts about the Future of Iverson Notation

Morten Kromberg mkrom@dyalog.com

Iverson@100 – Dec 17th, 2020

```
sin←1 ○ angle
```

Will the notation survive?

Yes!

Iverson did not invent APL...

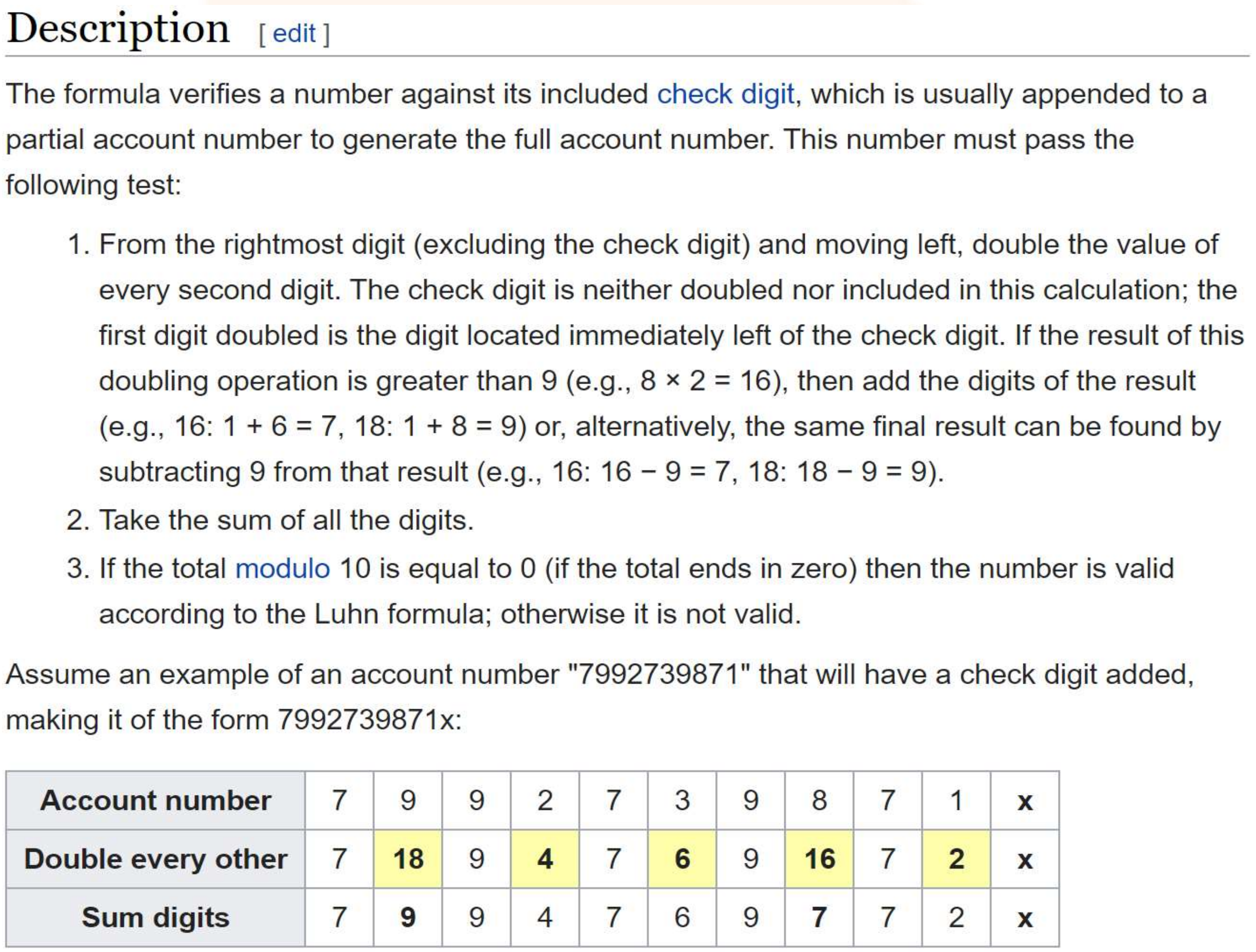
He ***discovered*** it!

(Bernard Legrand)

Notation
vs
Programming Language

The most successful APL users did not have programming as their primary skill

The Luhn algorithm (according to WikiPedia)



Description [edit]

The formula verifies a number against its included check digit, which is usually appended to a partial account number to generate the full account number. This number must pass the following test:

1. From the rightmost digit (excluding the check digit) and moving left, double the value of every second digit. The check digit is neither doubled nor included in this calculation; the first digit doubled is the digit located immediately left of the check digit. If the result of this doubling operation is greater than 9 (e.g., 8 × 2 = 16), then add the digits of the result (e.g., 16: 1 + 6 = 7, 18: 1 + 8 = 9) or, alternatively, the same final result can be found by subtracting 9 from that result (e.g., 16: 16 − 9 = 7, 18: 18 − 9 = 9).
2. Take the sum of all the digits.
3. If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is valid according to the Luhn formula; otherwise it is not valid.

Assume an example of an account number "7992739871" that will have a check digit added, making it of the form 7992739871x:

Account number	7	9	9	2	7	3	9	8	7	1	**x**
Double every other	7	**18**	9	**4**	7	**6**	9	**16**	7	**2**	**x**
Sum digits	7	**9**	9	4	7	6	9	**7**	7	2	**x**

Array Oriented Luhn

CardNo

7	9	9	2	7	3	9	8	7	1	3

```
Body←(Count←¯1+≢CardNo)↑CardNo
```

```
Check←⊢/CardNo
```

Body

7	9	9	2	7	3	9	8	7	1

3	Check

```
Weights←Count⍴(2|Count)⌽1 2
```

Weights

1	2	1	2	1	2	1	2	1	2

```
Products←Body×Weights
```

Products

7	18	9	4	7	6	9	16	7	2

```
Digits←0 10⊤Products
```

Digits

0	1	0	0	0	0	0	1	0	0
7	8	9	4	7	6	9	6	7	2

SumDigits

67

```
SumDigits←+/,Digits
```

```
Check←10|-SumDigits
```

3	Check

Functional

+

Array Orientation

=

Mechanical Sympathy

Criticisms of APL

Criticisms of the APL **Language**

Weird Symbols

Infix Notation for ALL functions

Operators vs Functions

No Type Declarations

Dynamic Scope / Global by Default

Strengths of the APL **Language**

Wonderful Symbols
Infix Notation for ALL functions
Operators vs Functions
No Type Declarations
Dynamic Scope / Global by Default

Oh all right then,
we *have* added...

control structures
local-by-default lexical scope
and OOP (if you must)

but

:Repeat … :Until

and

:Implements Constructor

… are not part of ***the notation***

How do we ensure that the notation is still relevant on Iverson's 200th birthday...

(in other words, when Python and Javascript have been swept aside,
[web] platforms have matured,
GUI and Security API madness is behind us?)

NOT by making the language more like Python & JavaScript!

Criticisms of APL

Criticisms of APL **Eco-Systems**

Poor libraries & poor library support in the language
Insufficient training materials and samples
Closed, ageing community
"Corporate" rather than "Hacker" vibe

... fair enough, we will work on these

There is a steady supply
of people who want to learn
how to solve problems
on a computer...

I am a high school student.
This is the fourth year I am entering this competition.
I really enjoy it every year.

I am a big fan of the array-oriented approach I get to use with APL.
I feel like I spend a lot more time focusing on solving the problem
instead of having to focus on syntax like with other languages.

I am also familiar with C# and Python
but APL is definitely my favourite language.

Poetry

```
      rippleShuffle ← {⍵[⍋⍒(⍴⍵)⍴1 0]}
      rippleShuffle ⍳10
0 5 1 6 2 7 3 8 4 9
```

```
      nestDepth ← {+\-⌿'()' ∘.= ⍵}
      nestDepth 'a←(2×(3+4))÷10'
(formatted)   00111222210000
```

```
      nextPascal ← {(0,⍵)+(⍵,0)}
      nextPascal 1 3 3 1
1 4 6 4 1
```

```
      mean ← +⌿ ÷ ≢
      mean 1 2 3 4
2.5
```

```
      palindrome ← ▯≡⌽⌽≡▯
      palindrome 'ABBA'
1
```

```
      leapYear ← 0 ≠.= 4 100 400 ∘.| ⊢
      leapYear 2020
1
```

```
      {(⍳2×⍵ ⍵)∊↓⍉⍵+⌊⍵×1 2 ∘.○ ○+\200/0.01} 5
0 0 1 1 1 1 1 1 0 0
0 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 1 0
0 0 1 1 1 1 1 1 0 0
```

You have been using the same programming language for more than 30 years, and you are still smiling!?

(comment from young Indian programmer after an APL talk at FunctionalConf, Bangalore)

```
      {(⍳2×⍵ ⍵)∊↓⌽⍵+⌊⍵×1 2 ∘.○ ○+\200/0.01} 5
0 0 1 1 1 1 1 1 0 0
0 1 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 1 0
0 0 1 1 1 1 1 1 0 0
```

You have been using the same programming language for more than 30 years, and you are still smiling!?

(comment from young Indian programmer after an APL talk at FunctionalConf, Bangalore)

Ken Iverson, 1920–2020