APL (programming language)

APL (named after the book *A Programming Language*)[3] is a programming language developed in the 1960s by Kenneth E. Iverson. Its central datatype is the multidimensional array. It uses a large range of special graphic symbols[4] to represent most functions and operators, leading to very concise code. It has been an important influence on the development of concept modeling, spreadsheets, functional programming,[5] and computer math packages.[6] It has also inspired several other programming languages.[7][8]

APL

Paradigm	Array, functional, structured, modular
Designed by	Kenneth E. Iverson
Developer	Larry Breed, Dick Lathwell, Roger Moore and others
First appeared	November 27, 1966[1]
Stable release	ISO/IEC 13751:2001 / February 1, 2001
Typing discipline	Dynamic
Platform	Cross platform
License	Proprietary, open source
Website	aplwiki.com (https://aplwiki.com/)

Major implementations

APL\360 · APL\1130 · APL*Plus · Sharp APL · APL2 · Dyalog APL · NARS2000 · APLX · GNU APL

Influenced by

Mathematical notation

Influenced

A and A+ · C++[2] · FP · J · K · MATLAB · Nial · PPL · Python · q

History

Mathematical notation

A mathematical notation for manipulating arrays was developed by Kenneth E. Iverson, starting in 1957 at Harvard University. In 1960, he began work for IBM where he developed this notation with Adin Falkoff and published it in his book *A Programming Language* in 1962.[3] The preface states its premise:

> Applied mathematics is largely concerned with the design and analysis of explicit procedures for calculating the exact or approximate values of various functions. Such explicit procedures are called algorithms or *programs*. Because an effective notation for the description of programs exhibits considerable syntactic structure, it is called a *programming language*.

This notation was used inside IBM for short research reports on computer systems, such as the Burroughs B5000 and its stack mechanism when stack machines versus register machines were being evaluated by IBM for upcoming computers.

Iverson also used his notation in a draft of the chapter *A Programming Language*, written for a book he was writing with Fred Brooks, *Automatic Data Processing*, which would be published in 1963.[9][10]

In 1979, Iverson received the Turing Award for his work on APL.[11]

(kdb) · S · Snap! · Speakeasy · Wolfram Language

Development into a computer programming language

As early as 1962, the first attempt to use the notation to describe a complete computer system happened after Falkoff discussed with William C. Carter his work to standardize the instruction set for the machines that later became the IBM System/360 family.

In 1963, Herbert Hellerman, working at the IBM Systems Research Institute, implemented a part of the notation on an IBM 1620 computer, and it was used by students in a special high school course on calculating transcendental functions by series summation. Students tested their code in Hellerman's lab. This implementation of a part of the notation was called Personalized Array Translator (PAT).[12]

In 1963, Falkoff, Iverson, and Edward H. Sussenguth Jr., all working at IBM, used the notation for a formal description of the IBM System/360 series machine architecture and functionality, which resulted in a paper published in *IBM Systems Journal* in 1964. After this was published, the team turned their attention to an implementation of the notation on a computer system. One of the motivations for this focus of implementation was the interest of John L. Lawrence who had new duties with Science Research Associates, an educational company bought by IBM in 1964. Lawrence asked Iverson and his group to help use the language as a tool to develop and use computers in education.[13]

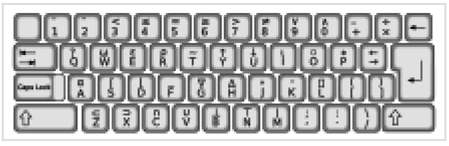
After Lawrence M. Breed and Philip S. Abrams of Stanford University joined the team at IBM Research, they continued their prior work on an implementation programmed in FORTRAN IV for a part of the notation which had been done for the IBM 7090 computer running on the IBSYS operating system. This work was finished in late 1965 and later named IVSYS (for Iverson system). The basis of this implementation was described in detail by Abrams in a Stanford University Technical Report, "An Interpreter for Iverson Notation" in 1966. The academic aspect of this was formally supervised by Niklaus Wirth.[14] Like Hellerman's PAT system earlier, this implementation did not include the APL character set but used special English reserved words for functions and operators. The system was later adapted for a time-sharing system and, by November 1966, it had been reprogrammed for the IBM System/360 Model 50 computer running in a time-sharing mode and was used internally at IBM.[15]

Hardware

IBM typeballs and typewheel containing APL Greek characters

A key development in the ability to use APL effectively, before the wide use of cathode ray tube (CRT) terminals, was the development of a special IBM Selectric typewriter interchangeable typing element with all the special APL characters on it. This was used on paper printing terminal workstations using the Selectric typewriter and typing element mechanism, such as the IBM 1050 and IBM 2741 terminal. Keycaps could be placed over the normal keys to show which APL characters would be entered and typed when that key was struck. For the first time, a programmer could type in and see proper APL characters as used in Iverson's notation and not be forced to use awkward English keyword representations of them. Falkoff and Iverson had the special APL Selectric typing elements, 987 and 988, designed in late 1964, although no APL computer system was available to use them.[16] Iverson cited Falkoff as the inspiration for the idea of using an IBM Selectric typing element for the APL character set.[17]

Many APL symbols, even with the APL characters on the Selectric typing element, still had to be typed in by over-striking two extant element characters. An example is the *grade up* character, which had to be made from a *delta* (shift-H) and a *Sheffer stroke* (shift-M). This was necessary because the APL character set was much larger than the 88 characters allowed on the typing element, even when letters were restricted to upper-case (capitals).


A programmer's view of the IBM 2741 keyboard layout with the APL typing element print head inserted

Commercial availability

The first APL interactive login and creation of an APL workspace was in 1966 by Larry Breed using an IBM 1050 terminal at the IBM Mohansic Labs near Thomas J. Watson Research Center, the home of APL, in Yorktown Heights, New York.[16]

IBM was chiefly responsible for introducing APL to the marketplace. The first publicly available version of APL was released in 1968 for the IBM 1130. IBM provided *APL\1130* for free but without liability or support.[18][19] It would run in as little as 8k 16-bit words of memory, and used a dedicated 1 megabyte hard disk.

APL gained its foothold on mainframe timesharing systems from the late 1960s through the early 1980s, in part because it would support multiple users on lower-specification systems that had no dynamic address translation hardware.[20] Additional improvements in performance for selected IBM System/370 mainframe systems included the *APL Assist Microcode* in which some support for APL execution was included in the processor's firmware, as distinct from being implemented entirely by higher-level software. Somewhat later, as suitably performing hardware was finally growing available in the mid- to late-1980s, many users migrated their applications to the personal computer environment.

Early IBM APL interpreters for IBM 360 and IBM 370 hardware implemented their own multi-user management instead of relying on the host services, thus they were their own timesharing systems. First introduced for use at IBM in 1966, the *APL\360*[21][22][23] system was a multi-user interpreter. The ability to programmatically communicate with the operating system for information and setting interpreter system variables was done through special privileged "I-beam" functions, using both monadic and dyadic operations.[24]

In 1973, IBM released *APL.SV*, which was a continuation of the same product, but which offered shared variables as a means to access facilities outside of the APL system, such as operating system files. In the mid-1970s, the IBM mainframe interpreter was even adapted for use on the IBM 5100 desktop computer, which had a small CRT and an APL keyboard, when most other small computers of the time only offered BASIC. In the 1980s, the *VSAPL* program product enjoyed wide use with Conversational Monitor System (CMS), Time Sharing Option (TSO), VSPC, MUSIC/SP, and CICS users.

In 1973–1974, Patrick E. Hagerty directed the implementation of the University of Maryland APL interpreter for the 1100 line of the Sperry UNIVAC 1100/2200 series mainframe computers.[25] In 1974, student Alan Stebbens was assigned the task of implementing an internal function.[26] Xerox APL was available from June 1975 for Xerox 560 and Sigma 6, 7, and 9 mainframes running CP-V and for Honeywell CP-6.[27]

In the 1960s and 1970s, several timesharing firms arose that sold APL services using modified versions of the IBM APL\360[23] interpreter. In North America, the better-known ones were IP Sharp Associates, Scientific Time Sharing Corporation (STSC), Time Sharing Resources (TSR), and The Computer Company (TCC). CompuServe also entered the market in 1978 with an APL Interpreter based on a modified version of Digital Equipment Corp and Carnegie Mellon's, which ran on DEC's KI and KL 36-bit machines. CompuServe's APL was available both to its commercial market and the consumer information service. With the advent first of less expensive mainframes such as the IBM 4300, and later the personal computer, by the mid-1980s, the timesharing industry was all but gone.

Sharp APL was available from IP Sharp Associates, first as a timesharing service in the 1960s, and later as a program product starting around 1979. *Sharp APL* was an advanced APL implementation with many language extensions, such as *packages* (the ability to put one or more objects into a single variable), a file system, nested arrays, and shared variables.

APL interpreters were available from other mainframe and mini-computer manufacturers also, notably Burroughs, Control Data Corporation (CDC), Data General, Digital Equipment Corporation (DEC), Harris, Hewlett-Packard (HP), Siemens, Xerox and others.

Garth Foster of Syracuse University sponsored regular meetings of the APL implementers' community at Syracuse's Minnowbrook Conference Center in Blue Mountain Lake, New York. In later years, Eugene McDonnell organized similar meetings at the Asilomar Conference Grounds near Monterey, California, and at Pajaro Dunes near Watsonville, California. The SIGAPL special interest group of the Association for Computing Machinery continues to support the APL community.[28]

Microcomputers

On microcomputers, which became available from the mid-1970s onwards, BASIC became the dominant programming language.[29] Nevertheless, some microcomputers provided APL instead – the first being the Intel 8008-based MCM/70 which was released in 1974[30][31] and which was primarily used in education.[32] Another machine of this time was the VideoBrain Family Computer, released in 1977, which was supplied with its dialect of APL called APL/S.[33]

The Commodore SuperPET, introduced in 1981, included an APL interpreter developed by the University of Waterloo.[34]

In 1976, Bill Gates claimed in his Open Letter to Hobbyists that Microsoft Corporation was implementing APL for the Intel 8080 and Motorola 6800 but had "very little incentive to make [it] available to hobbyists" because of software piracy.[35] It was never released.

APL2

Starting in the early 1980s, IBM APL development, under the leadership of Jim Brown, implemented a new version of the APL language that contained as its primary enhancement the concept of *nested arrays*, where an array can contain other arrays, and new language features which facilitated integrating nested arrays into program workflow. Ken Iverson, no longer in control of the development of the APL language, left IBM and joined I. P. Sharp Associates, where one of his major contributions was directing the evolution of Sharp APL to be more in accord with his vision.[36][37][38] APL2 was first released for CMS and TSO in 1984.[39] The APL2 Workstation edition (Windows, OS/2, AIX, Linux, and Solaris) followed later.[40][41]

As other vendors were busy developing APL interpreters for new hardware, notably Unix-based microcomputers, APL2 was almost always the standard chosen for new APL interpreter developments. Even today, most APL vendors or their users cite APL2 compatibility as a selling point for those products.[42][43] IBM cites its use for problem solving, system design, prototyping, engineering and scientific computations, expert systems,[44] for teaching mathematics and other subjects, visualization and database access.[45]

Modern implementations

Various implementations of APL by APLX, Dyalog, et al., include extensions for object-oriented programming, support for .NET, XML-array conversion primitives, graphing, operating system interfaces, and lambda calculus expressions. Freeware versions include GNU APL for Linux and NARS2000 for Windows (which runs on Linux under Wine). Both of these are fairly complete versions of APL2 with various language extensions.

Derivative languages

APL has formed the basis of, or influenced, the following languages:

- A and A+, an alternative APL, the latter with graphical extensions.
- FP, a functional programming language.
- Ivy, an interpreter for an APL-like language developed by Rob Pike, and which uses ASCII as input.[46]
- J, which was also designed by Iverson, and which uses ASCII with digraphs instead of special symbols.[7]
- K, a proprietary variant of APL developed by Arthur Whitney.[8]
- MATLAB, a numerical computation tool.[6]
- Nial, a high-level array programming language with a functional programming notation.
- Polymorphic Programming Language, an interactive, extensible language with a similar base language.
- S, a statistical programming language (usually now seen in the open-source version known as R).
- Snap*!*, a low-code block-based programming language, born as an extended reimplementation of Scratch
- Speakeasy, a numerical computing interactive environment.
- Wolfram Language, the programming language of Mathematica.[47]

Language characteristics

Character set

APL has been criticized and praised for its choice of a unique, non-standard character set. In the 1960s and 1970s, few terminal devices or even displays could reproduce the APL character set. The most popular ones employed the IBM Selectric print mechanism used with a special APL type element. One of the early APL line terminals (line-mode operation only, *not* full screen) was the Texas Instruments TI Model 745 (c. 1977) with the full APL character set[48] which featured half and full duplex telecommunications modes, for interacting with an APL time-sharing service or remote mainframe to run a remote computer job, called an RJE.

Over time, with the universal use of high-quality graphic displays, printing devices and Unicode support, the APL character font problem has largely been eliminated. However, entering APL characters requires the use of input method editors, keyboard mappings, virtual/on-screen APL symbol sets,[49][50] or easy-reference printed keyboard cards which can frustrate beginners accustomed to other programming languages.[51][52][53] With beginners who have no prior experience with other programming languages, a study involving high school students found that typing and using APL characters did not hinder the students in any measurable way.[54]

In defense of APL, it requires fewer characters to type, and keyboard mappings become memorized over time. Special APL keyboards are also made and in use today, as are freely downloadable fonts for operating systems such as Microsoft Windows.[55] The reported productivity gains assume that one spends enough time working in the language to make it worthwhile to memorize the symbols, their semantics, and keyboard mappings, not to mention a substantial number of idioms for common tasks.

Design

Unlike traditionally structured programming languages, APL code is typically structured as chains of monadic or dyadic functions, and operators[56] acting on arrays.[57] APL has many nonstandard *primitives* (functions and operators) that are indicated by a single symbol or a combination of a few symbols. All primitives are defined to have the same precedence, and always associate to the right. Thus, APL is *read* or best understood from right-to-left.

Early APL implementations (c. 1970 or so) had no programming loop-flow control structures, such as `do` or `while` loops, and `if-then-else` constructs. Instead, they used array operations, and use of structured programming constructs was often not necessary, since an operation could be performed on a full array in one statement. For example, the `iota` function (`ɩ`) can replace for-loop iteration: ɩN when applied to a scalar positive integer yields a one-dimensional array (vector), 1 2 3 ... N. More recent implementations of APL generally include comprehensive control structures, so that data structure and program control flow can be clearly and cleanly separated.

The APL environment is called a *workspace*. In a workspace the user can define programs and data, i.e., the data values exist also outside the programs, and the user can also manipulate the data without having to define a program.[58] In the examples below, the APL interpreter first types six spaces before awaiting the user's input. Its own output starts in column one.

<pre> n ← 4 5 6 7</pre>	Assigns vector of values, {4 5 6 7}, to variable n, an array create operation. An equivalent yet more concise APL expression would be `n ← 3 + ⍳4`. Multiple values are stored in array n, the operation performed *without formal loops or control flow language*.

```
      n
4 5 6 7
```
	Display the contents of n, currently an array or vector.

```
      n+4
8 9 10 11
```
	4 is now added to all elements of vector n, creating a 4-element vector {8 9 10 11}.
	As above, APL's interpreter displays the result because the expression's value was not assigned to a variable (with a ←).

```
      +/n
22
```
	APL displays the sum of components of the vector n, i.e., `22 (= 4 + 5 + 6 + 7)` using a very compact notation: read +/ as "plus, over..." and a slight change would be "multiply, over..."

```
      m ← +/3+⍳4
      m
22
```
	These operations can be combined into one statement, remembering that APL evaluates expressions right to left: first `⍳4` creates an array, `[1,2,3,4]`, then 3 is added to each component, which are summed together and the result stored in variable `m`, finally displayed.
	
	In normal mathematical notation, it is equivalent to: $m = \sum_{i=1}^{4}(i+3)$. Recall that mathematical expressions are not read or evaluated from right-to-left.

The user can save the workspace with all values, programs, and execution status.

APL uses a set of non-ASCII symbols, which are an extension of traditional arithmetic and algebraic notation. Having single character names for single instruction, multiple data (SIMD) vector functions is one way that APL enables compact formulation of algorithms for data transformation such as computing Conway's Game of Life in one line of code.[59] In nearly all versions of APL, it is theoretically possible to express any computable function in one expression, that is, in one line of code.

Because of the unusual character set, many programmers use special keyboards with APL keytops to write APL code.[60] Although there are various ways to write APL code using only ASCII characters,[61] in practice it is almost never done. (This may be thought to support Iverson's thesis about notation as a tool of thought.[62]) Most if not all modern implementations use standard keyboard layouts, with special mappings or input method editors to access non-ASCII characters. Historically, the APL font has been distinctive, with uppercase italic alphabetic characters and upright numerals and symbols. Most vendors continue to display the APL character set in a custom font.

Advocates of APL claim that the examples of so-called *write-only code* (badly written and almost incomprehensible code) are almost invariably examples of poor programming practice or novice mistakes, which can occur in any language. Advocates also claim that they are far more productive with APL than with more conventional computer languages, and that working software can be implemented in far less time and with far fewer programmers than using other technology.

They also may claim that because it is compact and terse, APL lends itself well to larger-scale software development and complexity, because the number of lines of code can be reduced greatly. Many APL advocates and practitioners also view standard programming languages such as COBOL and Java as being comparatively tedious. APL is often found where time-to-market is important, such as with trading systems.[63][64][65][66]

Terminology

APL makes a clear distinction between *functions* and *operators*.[56][67] Functions take arrays (variables or constants or expressions) as arguments, and return arrays as results. Operators (similar to higher-order functions) take functions or arrays as arguments, and derive related functions. For example, the *sum* function is derived by applying the *reduction* operator to the *addition* function. Applying the same reduction operator to the *maximum* function (which returns the larger of two numbers) derives a function which returns the largest of a group (vector) of numbers. In the J language, Iverson substituted the terms *verb* for *function* and *adverb* or *conjunction* for *operator*.

APL also identifies those features built into the language, and represented by a symbol, or a fixed combination of symbols, as *primitives*. Most primitives are either functions or operators. Coding APL is largely a process of writing non-primitive functions and (in some versions of APL) operators. However a few primitives are considered to be neither functions nor operators, most noticeably assignment.

Some words used in APL literature have meanings that differ from those in both mathematics and the generality of computer science.

Terminology of APL operators

Term	Description
function	operation or mapping that takes zero, one (right) or two (left & right) arguments which may be scalars, arrays, or more complicated structures, and may return a similarly complex result. A function may be: ▪ Primitive: built-in and represented by a single glyph;[68] ▪ Defined: as a named and ordered collection of program statements;[68] ▪ Derived: as a combination of an operator with its arguments.[68]
array	data valued object of zero or more orthogonal dimensions in row-major order in which each item is a primitive scalar datum or another array.[69]
niladic	not taking or requiring any arguments, nullary[70]
monadic	requiring only one argument; on the right for a function, on the left for an operator, unary[70]
dyadic	requiring both a left and a right argument, binary[70]
ambivalent or monadic	capable of use in a monadic or dyadic context, permitting its left argument to be elided[68]
operator	operation or mapping that takes one (left) or two (left & right) function or array valued arguments (operands) and derives a function. An operator may be: ▪ Primitive: built-in and represented by a single glyph;[68] ▪ Defined: as a named and ordered collection of program statements.[68]

Syntax

APL has explicit representations of functions, operators, and syntax, thus providing a basis for the clear and explicit statement of extended facilities in the language, and tools to experiment on them.[71]

Examples

Hello, world

This displays "Hello, world":

```
'Hello, world'
```

A design theme in APL is to define default actions in some cases that would produce syntax errors in most other programming languages.

The 'Hello, world' string constant above displays, because display is the default action on any expression for which no action is specified explicitly (e.g. assignment, function parameter).

Exponentiation

Another example of this theme is that exponentiation in APL is written as `2*3`, which indicates raising 2 to the power 3 (this would be written as `2^3` or `2**3` in some languages, or relegated to a function call such as `pow(2, 3);` in others). Many languages use `*` to signify multiplication, as in `2*3`, but APL chooses to use `2×3`. However, if no base is specified (as with the statement `*3` in APL, or `^3` in other languages), most programming languages one would see this as a syntax error. APL, however, assumes the missing base to be the natural logarithm constant e, and interprets `*3` as `2.71828*3`.

Simple statistics

Suppose that `X` is an array of numbers. Then `(+/X)÷ρX` gives its average. Reading *right-to-left*, `ρX` gives the number of elements in X, and since `÷` is a dyadic operator, the term to its left is required as well. It is surrounded by parentheses since otherwise X would be taken (so that the summation would be of `X÷ρX`—each element of X divided by the number of elements in X), and `+/X` gives the sum of the elements of X. Building on this, the following expression computes standard deviation:

```
((+/((X - (+/X)÷ρX)*2))÷ρX)*0.5
```

Naturally, one would define this expression as a function for repeated use rather than rewriting it each time. Further, since assignment is an operator, it can appear within an expression, so the following would place suitable values into T, AV and SD:

```
SD←((+/((X - AV←(T←+/X)÷ρX)*2))÷ρX)*0.5
```

Pick 6 lottery numbers

This following immediate-mode expression generates a typical set of *Pick 6* lottery numbers: six pseudo-random integers ranging from 1 to 40, *guaranteed non-repeating*, and displays them sorted in ascending order:

```
x[⍋x←6?40]
```

The above does a lot, concisely, although it may seem complex to a new APLer. It combines the following APL *functions* (also called *primitives*[72] and *glyphs*[73]):

- The first to be executed (APL executes from rightmost to leftmost) is dyadic function `?` (named `deal` when dyadic) that returns a vector consisting of a select number (left argument: 6 in this case) of random integers ranging from 1 to a specified maximum (right argument: 40 in this case), which, if said maximum ≥ vector length, is guaranteed to be non-repeating; thus, generate/create 6 random integers ranging from 1 to 40.[74]
- This vector is then *assigned* (←) to the variable `x`, because it is needed later.
- This vector is then *sorted* in ascending order by a monadic `⍋` function, which has as its right argument everything to the right of it up to the next unbalanced *close-bracket* or close-parenthesis. The result of `⍋` is the indices that will put its argument into ascending order.
- Then the output of `⍋` is used to index the variable `x`, which we saved earlier for this purpose, thereby selecting its items in *ascending* sequence.

Since there is no function to the left of the left-most x to tell APL what to do with the result, it simply outputs it to the display (on a single line, separated by spaces) without needing any explicit instruction to do that.

`?` also has a monadic equivalent called `roll`, which simply returns one random integer between 1 and its sole operand [to the right of it], inclusive. Thus, a role-playing game program might use the expression `?20` to roll a twenty-sided die.

Prime numbers

The following expression finds all prime numbers from 1 to R. In both time and space, the calculation complexity is $O(R^2)$ (in Big O notation).

```
(~R∊R∘.×R)/R←1↓⍳R
```

Executed from right to left, this means:

- *Iota* `⍳` creates a vector containing integers from `1` to `R` (if `R= 6` at the start of the program, `⍳R` is `1 2 3 4 5 6`)
- *Drop* first element of this vector (`↓` function), i.e., `1`. So `1↓⍳R` is `2 3 4 5 6`
- *Set* `R` to the new vector (`←`, *assignment* primitive), i.e., `2 3 4 5 6`
- The `/` *replicate* operator is dyadic (binary) and the interpreter first evaluates its left argument (fully in parentheses):
- Generate *outer product* of `R` multiplied by `R`, i.e., a matrix that is the *multiplication table* of `R` by `R` (`°.×` operator), i.e.,

4	6	8	10	12
6	9	12	15	18
8	12	16	20	24
10	15	20	25	30
12	18	24	30	36

- Build a vector the same length as `R` with `1` in each place where the corresponding number in `R` is in the outer product matrix (∈, *set inclusion* or *element of* or *Epsilon* operator), i.e., `0 0 1 0 1`
- Logically negate (*not*) values in the vector (change zeros to ones and ones to zeros) (`∼`, logical *not* or *Tilde* operator), i.e., `1 1 0 1 0`
- Select the items in `R` for which the corresponding element is `1` (`/` *replicate* operator), i.e., `2 3 5`

(Note, this assumes the APL origin is 1, i.e., indices start with 1. APL can be set to use 0 as the origin, so that `ι6` is `0 1 2 3 4 5`, which is convenient for some calculations.)

Sorting

The following expression sorts a word list stored in matrix X according to word length:

```
X[⍋X+.≠' ';]
```

Game of Life

The following function "life", written in Dyalog APL,[75][76] takes a boolean matrix and calculates the new generation according to Conway's Game of Life. It demonstrates the power of APL to implement a complex algorithm in very little code, but understanding it requires some advanced knowledge of APL (as the same program would in many languages).

```
life ← {⊃1 ω ∨.∧ 3 4 = +/ +⌿ ¯1 0 1 ∘.⊖ ¯1 0 1 ⌽¨ ⊂ω}
```

HTML tags removal

In the following example, also Dyalog, the first line assigns some HTML code to a variable `txt` and then uses an APL expression to remove all the HTML tags (explanation (http://aplwiki.com/AplIn20Minutes#Extract_content_from_Code)):

```
      txt←'<html><body><p>This is <em>emphasized</em> text.</p></body></html>'
      {ω /⍨ ~{ω∨≠\ω}ω∊'<>'} txt
This is emphasized text.
```

Naming

APL derives its name from the initials of Iverson's book *A Programming Language*,[3] even though the book describes Iverson's mathematical notation, rather than the implemented programming language described in this article. The name is used only for actual implementations, starting with APL\360.

Adin Falkoff coined the name in 1966 during the implementation of APL\360 at IBM:

> As I walked by the office the three students shared, I could hear sounds of an argument going on. I poked my head in the door, and Eric asked me, "Isn't it true that everyone knows the notation we're using is called APL?" I was sorry to have to disappoint him by confessing that I had never heard it called that. Where had he got the idea it was well known? And who had decided to call it that? In fact, why did it have to be called anything? Quite a while later I heard how it was named. When the implementation effort started in June of 1966, the documentation effort started, too. I suppose when they had to write about "it", Falkoff and Iverson realized that they would have to give "it" a name. There were probably many suggestions made at the time, but I have heard of only two. A group in SRA in Chicago which was developing instructional materials using the notation was in favor of the name "Mathlab". This did not catch on. Another suggestion was to call it "Iverson's Better Math" and then let people coin the appropriate acronym. This was deemed facetious.
>
> Then one day Adin Falkoff walked into Ken's office and wrote "A Programming Language" on the board, and underneath it the acronym "APL". Thus it was born. It was just a week or so after this that Eric Iverson asked me his question, at a time when the name hadn't yet found its way the thirteen miles up the Taconic Parkway from IBM Research to IBM Mohansic.
>
> —Eugene McDonnell, [77]

APL is occasionally re-interpreted as *Array Programming Language* or *Array Processing Language*,[78] thereby making *APL* into a backronym.

Logo

British APL Association (BAPLA) conference laptop bag.

There has always been cooperation between APL vendors, and joint conferences were held on a regular basis from 1969 until 2010.[79] At such conferences, APL merchandise was often handed out, featuring APL motifs or collection of vendor logos. Common were apples (as a pun on the similarity in pronunciation of *apple* and *APL*) and the code snippet `⍺*⎕` which are the symbols produced by the classic APL keyboard layout when holding the APL modifier key and typing "APL".

Despite all these community efforts, no universal vendor-agnostic logo for the programming language emerged. As popular programming languages increasingly have established recognisable logos, Fortran getting one in 2020,[80] British APL Association launched a campaign in the second half of 2021, to establish such a logo for APL, and after a community election and multiple rounds of feedback, a logo was chosen in May 2022.[81]

Use

APL is used for many purposes including financial and insurance applications,[82] artificial intelligence,[83][84] neural networks[85] and robotics.[86] It has been argued that APL is a calculation tool and not a programming language;[87] its symbolic nature and array capabilities have made it popular with domain experts and data scientists[88] who do not have or require the skills of a computer programmer.

APL is well suited to image manipulation and computer animation, where graphic transformations can be encoded as matrix multiplications. One of the first commercial computer graphics houses, Digital Effects, produced an APL graphics product named *Visions*, which was used to create television commercials and animation for the 1982 film *Tron*.[89] Latterly, the Stormwind (https://stormwind.fi/en/) boating simulator uses APL to implement its core logic, its interfacing to the rendering pipeline middleware and a major part of its physics engine.[90]

Today, APL remains in use in a wide range of commercial and scientific applications, for example investment management,[82] asset management,[91] health care,[92] and DNA profiling.[93][94]

Notable implementations

APL\360

The first implementation of APL using recognizable APL symbols was APL\360 which ran on the IBM System/360, and was completed in November 1966[1] though at that time remained in use only within IBM.[39] In 1973 its implementors, Larry Breed, Dick Lathwell and Roger Moore, were awarded the Grace Murray Hopper Award from the Association for Computing Machinery (ACM). It was given "for their work in the design and implementation of APL\360, setting new standards in simplicity, efficiency, reliability and response time for interactive systems."[95][96][97]

In 1975, the IBM 5100 microcomputer offered APL\360[98] as one of two built-in ROM-based interpreted languages for the computer, complete with a keyboard and display that supported all the special symbols used in the language.[99]

Significant developments to APL\360 included CMS/APL, which made use of the virtual storage capabilities of CMS and APLSV, which introduced shared variables, system variables and system functions. It was subsequently ported to the IBM System/370 and VSPC platforms until its final release in 1983, after which it was replaced by APL2.[39]

APL\1130

In 1968, APL\1130 became the first publicly available APL system,[100] created by IBM for the IBM 1130. It became the most popular IBM Type-III Library software that IBM released.[101]

APL*Plus and Sharp APL

APL*Plus and Sharp APL are versions of APL\360 with added business-oriented extensions such as data formatting and facilities to store APL arrays in external files. They were jointly developed by two companies, employing various members of the original IBM APL\360 development team.[102]

The two companies were I. P. Sharp Associates (IPSA), an APL\360 services company formed in 1964 by Ian Sharp, Roger Moore and others, and STSC, a time-sharing and consulting service company formed in 1969 by Lawrence Breed and others. Together the two developed APL*Plus and thereafter continued to work together but develop APL separately as APL*Plus and Sharp APL. STSC ported APL*Plus to many platforms with versions being made for the VAX 11,[103] PC and UNIX, whereas IPSA took a different approach to the arrival of the personal computer and made Sharp APL available on this platform using additional PC-XT/360 hardware. In 1993, Soliton Incorporated was formed to support Sharp APL and it developed Sharp APL into SAX (Sharp APL for Unix). As of 2018, APL*Plus continues as APL2000 APL+Win.

In 1985, Ian Sharp, and Dan Dyer of STSC, jointly received the Kenneth E. Iverson Award for Outstanding Contribution to APL.[104]

APL2

APL2 was a significant re-implementation of APL by IBM which was developed from 1971 and first released in 1984. It provides many additions to the language, of which the most notable is nested (non-rectangular) array support.[39] The entire APL2 Products and Services Team was awarded the Iverson Award in 2007.[104]

In 2021, IBM sold APL2 to Log-On Software, who develop and sell the product as *Log-On APL2*.[105]

APLGOL

In 1972, APLGOL was released as an experimental version of APL that added structured programming language constructs to the language framework. New statements were added for interstatement control, conditional statement execution, and statement structuring, as well as statements to clarify the intent of the algorithm.[106] It was implemented for Hewlett-Packard in 1977.[107]

Dyalog APL

Dyalog APL was first released by British company Dyalog Ltd.[108] in 1983[109] and, as of 2018, is available for AIX, Linux (including on the Raspberry Pi), macOS and Microsoft Windows platforms. It is based on APL2, with extensions to support object-oriented programming,[110] functional programming,[111] and tacit programming.[112] Licences are free for personal/non-commercial use.[113]

In 1995, two of the development team – John Scholes and Peter Donnelly – were awarded the Iverson Award for their work on the interpreter.[104] Gitte Christensen and Morten Kromberg were joint recipients of the Iverson Award in 2016.[114]

NARS2000

NARS2000 is an open-source APL interpreter written by Bob Smith, a prominent APL developer and implementor from STSC in the 1970s and 1980s. NARS2000 contains advanced features and new datatypes and runs natively on Microsoft Windows, and other platforms under Wine. It is named after a development tool from the 1980s, NARS (Nested Arrays Research System).[115]

APLX

APLX is a cross-platform dialect of APL, based on APL2 and with several extensions, which was first released by British company MicroAPL in 2002. Although no longer in development or on commercial sale it is now available free of charge from Dyalog.[116]

York APL

York APL[117] was developed at the York University, Ontario around 1968, running on IBM 360 mainframes. One notable difference between it and APL\360 was that it defined the "shape" (ρ) of a scalar as 1 whereas APL\360 defined it as the more mathematically correct 0 — this made it easier to write functions that acted the same with scalars and vectors.

GNU APL

GNU APL is a free implementation of Extended APL as specified in ISO/IEC 13751:2001 and is thus an implementation of APL2. It runs on Linux, macOS, several BSD dialects, and on Windows (either using Cygwin for full support of all its system functions or as a native 64-bit Windows binary with some of its system functions missing). GNU APL uses Unicode internally and can be scripted. It was written by Jürgen Sauermann.[118]

Richard Stallman, founder of the GNU Project, was an early adopter of APL, using it to write a text editor as a high school student in the summer of 1969.[119]

Interpretation and compilation of APL

APL is traditionally an interpreted language, having language characteristics such as weak variable typing not well suited to compilation.[120] However, with arrays as its core data structure[121] it provides opportunities for performance gains through parallelism,[122] parallel computing,[123][124] massively parallel applications,[125][126] and very-large-scale integration (VLSI),[127][128] and from the outset APL has been regarded as a high-performance language[129] – for example, it was noted for the speed with which it could perform complicated matrix operations "because it operates on arrays and performs operations like matrix inversion internally".[130]

Nevertheless, APL is rarely purely interpreted and compilation or partial compilation techniques that are, or have been, used include the following:

Idiom recognition

Most APL interpreters support idiom recognition[131] and evaluate common idioms as single operations.[132][133] For example, by evaluating the idiom `BV/⍳⍴A` as a single operation (where `BV` is a Boolean vector and `A` is an array), the creation of two intermediate arrays is avoided.[134]

Optimised bytecode

Weak typing in APL means that a name may reference an array (of any datatype), a function or an operator. In general, the interpreter cannot know in advance which form it will be and must therefore perform analysis, syntax checking etc. at run-time.[135] However, in certain circumstances, it is possible to deduce in advance what type a name is expected to reference and then generate bytecode which can be executed with reduced run-time overhead. This bytecode can also be optimised using compilation techniques such as constant folding or common subexpression elimination.[136] The interpreter will execute the bytecode when present and when any assumptions which have been made are met. Dyalog APL includes support for optimised bytecode.[136]

Compilation

Compilation of APL has been the subject of research and experiment since the language first became available; the first compiler is considered to be the Burroughs APL-700[137] which was released around 1971.[138] In order to be able to compile APL, language limitations have to be imposed.[137][139] APEX is a research APL compiler which was written by Robert Bernecky and is available under the GNU Public License.[140]

The STSC APL Compiler is a hybrid of a bytecode optimiser and a compiler – it enables compilation of functions to machine code provided that its sub-functions and globals are declared, but the interpreter is still used as a runtime library and to execute functions which do not meet the compilation requirements.[141]

Standards

APL has been standardized by the American National Standards Institute (ANSI) working group X3J10 and International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), ISO/IEC Joint Technical Committee 1 Subcommittee 22 Working Group 3. The Core APL language is specified in ISO 8485:1989, and the Extended APL language is specified in ISO/IEC 13751:2001.

References

1. "APL Quotations and Anecdotes" (http://www.jsoftware.com/papers/APLQA.htm#APL_birthday). *jsoftware.com*. jsoftware. Retrieved April 14, 2018.
2. "std::iota" (https://en.cppreference.com/w/cpp/algorithm/iota). *cppreference.com*.
3. Kenneth E. Iverson (1 December 1962). *A Programming Language* (https://www.softwarepreservation.org/projects/apl/Books/APROGRAMMING%20LANGUAGE). Wiley. ISBN 978-0-471-43014-8. OL 26792153M (https://openlibrary.org/books/OL26792153M). Wikidata Q105954505. Retrieved 2023-08-06.

4. McIntyre, Donald B. (1991). "Language as an Intellectual Tool: From Hieroglyphics to APL" (https://web.archive.org/web/20160304051735/http://domino.research.ibm.com/tchjr/journalindex.nsf/e90fc5d047e64ebf85256bc80066919c/9c834f5a16efa82085256bfa00685c72!OpenDocument). *IBM Systems Journal*. **30** (4): 554–581. doi:10.1147/sj.304.0554 (https://doi.org/10.1147%2Fsj.304.0554). Archived from the original (http://domino.research.ibm.com/tchjr/journalindex.nsf/e90fc5d047e64ebf85256bc80066919c/9c834f5a16efa82085256bfa00685c72!OpenDocument) on March 4, 2016. Retrieved January 9, 2015.
5. "ACM Award Citation – John Backus" (https://web.archive.org/web/20080212043802/https://awards.acm.org/citation.cfm?id=0703524&srt=all&aw=140&ao=AMTURING). Awards.acm.org. 1977. Archived from the original (http://awards.acm.org/citation.cfm?id=0703524&srt=all&aw=140&ao=AMTURING) on February 12, 2008. Retrieved February 3, 2010.
6. Moler, Cleve. "The Growth of MATLAB" (https://web.archive.org/web/20090411120119/http://www.mathworks.com/company/newsletters/news_notes/clevescorner/jan06.pdf) (PDF). Archived from the original (http://www.mathworks.com/company/newsletters/news_notes/clevescorner/jan06.pdf) (PDF) on April 11, 2009. Retrieved February 3, 2010.
7. "A Bibliography of APL and J" (http://www.jsoftware.com/jwiki/Essays/Bibliography). Jsoftware.com. Retrieved March 2, 2010.
8. "An Interview with Arthur Whitney" (https://web.archive.org/web/20090404064737/http://kx.com/Company/press-releases/arthur-interview.php). Kx Systems. January 4, 2004. Archived from the original (http://kx.com/Company/press-releases/arthur-interview.php) on April 4, 2009. Retrieved March 2, 2010.
9. Iverson, Kenneth E., "Automatic Data Processing: Chapter 6: A programming language" (http://www.softwarepreservation.org/projects/apl/Books/Iverson-AutomaticDataProcessing-bilevel.pdf/view) Archived (https://web.archive.org/web/20090604091847/http://www.softwarepreservation.org/projects/apl/Books/Iverson-AutomaticDataProcessing-bilevel.pdf/view) June 4, 2009, at the Wayback Machine, 1960, Draft copy for Brooks and Iverson 1963 book, *Automatic Data Processing*.
10. Brooks, Fred; Iverson, Kenneth, (1963), *Automatic Data Processing*, John Wiley & Sons Inc.
11. "Turing Award Citation 1979" (https://web.archive.org/web/20091223064709/http://awards.acm.org/citation.cfm?id=9147499&srt=all&aw=140&ao=AMTURING). Awards.acm.org. Archived from the original (http://awards.acm.org/citation.cfm?id=9147499&srt=all&aw=140&ao=AMTURING) on 2009-12-23. Retrieved February 3, 2010.
12. Hellerman, H. (July 1964). "Experimental Personalized Array Translator System" (https://doi.org/10.1145%2F364520.364573). *Communications of the ACM*. **7** (7): 433–438. doi:10.1145/364520.364573 (https://doi.org/10.1145%2F364520.364573). S2CID 2181070 (https://api.semanticscholar.org/CorpusID:2181070).
13. Falkoff, Adin D.; Iverson, Kenneth E., "The Evolution of APL" (http://www.jsoftware.com/papers/APLEvol.htm), ACM SIGPLAN Notices 13, 1978-08.
14. Abrams, Philip S., *An interpreter for "Iverson notation"* (http://infolab.stanford.edu/TR/CS-TR-66-47.html), Technical Report: CS-TR-66-47, Department of Computer Science, Stanford University, August 1966;
15. Haigh, Thomas (2005). "Biographies: Kenneth E. Iverson". *IEEE Annals of the History of Computing*. doi:10.1109/MAHC.2005.4 (https://doi.org/10.1109%2FMAHC.2005.4).
16. Breed, Larry, "The First APL Terminal Session" (http://portal.acm.org/citation.cfm?id=138094.140933), *APL Quote Quad*, Association for Computing Machinery, Volume 22, Number 1, September 1991, p. 2–4.
17. 19, 2009 Adin Falkoff (http://www.computerhistory.org/tdih/?setdate=December) – Computer History Museum. "Iverson credited him for choosing the name APL and the introduction of the IBM golf-ball typewriter with the replacement typehead, which provided the famous character set to represent programs."

18. Breed, Larry (August 2006). "How We Got to APL\1130" (https://web.archive.org/web/20080512031437/http://www.vector.org.uk/archive/v223/APL_1130.htm). *Vector (British APL Association)*. **22** (3). ISSN 0955-1433 (https://www.worldcat.org/issn/0955-1433). Archived from the original (http://www.vector.org.uk/archive/v223/APL_1130.htm) on 2008-05-12. Retrieved 2007-04-02.
19. APL\1130 Manual (http://bitsavers.org/pdf/ibm/1130/lang/1130-03.3.001_APL_1130_May69.pdf) Archived (https://web.archive.org/web/20110221034650/http://www.bitsavers.org/pdf/ibm/1130/lang/1130-03.3.001_APL_1130_May69.pdf) 2011-02-21 at the Wayback Machine, May 1969
20. "Remembering APL" (http://www.quadibloc.com/comp/aplint.htm). Quadibloc.com. Retrieved June 17, 2013.
21. Falkoff, Adin; Iverson, Kenneth E., "APL\360 Users Guide" (http://bitsavers.org/pdf/ibm/apl/APL_360_Users_Manual_Aug68.pdf) Archived (https://web.archive.org/web/20120229200744/http://bitsavers.org/pdf/ibm/apl/APL_360_Users_Manual_Aug68.pdf) 2012-02-29 at the Wayback Machine, IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY, August 1968.
22. "APL\360 Terminal System" (http://bitsavers.org/pdf/ibm/apl/APL_360_Terminal_System_Mar67.pdf) Archived (https://web.archive.org/web/20100711092528/http://bitsavers.org/pdf/ibm/apl/APL_360_Terminal_System_Mar67.pdf) 2010-07-11 at the Wayback Machine, IBM Research, Thomas J. Watson Research Center, March 1967.
23. Pakin, Sandra (1968). *APL\360 Reference Manual*. Science Research Associates, Inc. ISBN 978-0-574-16135-2.
24. Falkoff, Adin D.; Iverson, Kenneth E.,*The Design of APL* (http://www.research.ibm.com/journal/rd/174/ibmrd1704F.pdf), *IBM Journal of Research and Development*, Volume 17, Number 4, July 1973. "These environmental defined functions were based on the use of still another class of functions—called "I-beams" because of the shape of the symbol used for them—which provide a more general facility for communication between APL programs and the less abstract parts of the system. The I-beam functions were first introduced by the system programmers to allow them to execute System/360 instructions from within APL programs, and thus use APL as a direct aid in their programming activity. The obvious convenience of functions of this kind, which appeared to be part of the language, led to the introduction of the monadic I-beam function for direct use by anyone. Various arguments to this function yielded information about the environment such as available space and time of day."
25. Minker, Jack (January 2004). "Beginning of Computing and Computer Sciences at the University of Maryland" (https://web.archive.org/web/20110610064807/http://www.cs.umd.edu/department/dept-history/minker-report.pdf) (PDF). Section 2.3.4: University of Maryland. p. 38. Archived from the original (http://www.cs.umd.edu/department/dept-history/minker-report.pdf) (PDF) on June 10, 2011. Retrieved May 23, 2011.
26. Stebbens, Alan. "How it all began" (https://web.archive.org/web/20160304000314/http://lathwellproductions.ca/wordpress/film). Archived from the original (http://lathwellproductions.ca/wordpress/film) on 2016-03-04. Retrieved 2011-05-22.
27. "Xerox APL Language and Operations Reference Manual" (http://www.softwarepreservation.org/projects/apl/Books/197506_Xerox%20APL%20Language%20and%20Operations%20Reference%20Manual_90131C.pdf) (PDF). Archived (https://ghostarchive.org/archive/20221009/http://www.softwarepreservation.org/projects/apl/Books/197506_Xerox%20APL%20Language%20and%20Operations%20Reference%20Manual_90131C.pdf) (PDF) from the original on 2022-10-09.
28. "SIGAPL" (http://www.sigapl.org/). Sigapl.org. Retrieved June 17, 2013.
29. "Fifty Years of BASIC, the Programming Language That Made Computers Personal" (http://time.com/69316/basic/). *Time*. April 29, 2014. Retrieved April 29, 2018.

30. "MCM Computers M70/M700" (https://web.archive.org/web/20180403063223/http://www.old-computers.com/museum/computer.asp?c=346). *old-computers.com*. Archived from the original (http://www.old-computers.com/museum/computer.asp?c=346) on April 3, 2018. Retrieved April 8, 2018.
31. Stachniak, Stachniak (2011). *Inventing the PC: The MCM/70 Story* (https://books.google.com/books?id=cyWOA2FED7EC). McGill Queens's University Press. ISBN 978-0-7735-3852-8.
32. Miller, Michael (December 17, 2014). "PCs That Paved the Way for the Altair" (http://uk.pcmag.com/opinion/38348/opinion/pcs-that-paved-the-way-for-the-altair). *PC Magazine*. Ziff Davis. Retrieved April 29, 2018.
33. "VideoBrain Family Computer" (https://books.google.com/books?id=OQEAAAAAMBAJ&q=videobrain+family+computer+apl%2Fs&pg=PA133/s), *Popular Science*, November 1978, advertisement.
34. "A Look at SuperPet" (https://archive.org/stream/1981-12-compute-magazine/Compute_Issue_019_1981_Dec#page/n131/mode/2up). *Compute!*. Small System Services Inc. December 1981. Retrieved April 29, 2018.
35. Gates, Bill (January 31, 1976). "An Open Letter to Hobbyists" (http://www.digibarn.com/collections/newsletters/homebrew/V2_01/index.html). *Homebrew Computer Club Newsletter*. Retrieved April 29, 2018.
36. Hui, Roger. "Remembering Ken Iverson" (http://keiapl.org/rhui/remember.htm). *keiapl.org*. KEIAPL. Retrieved January 10, 2015.
37. ACM A.M. Turing Award. "Kenneth E. Iverson – Citation" (http://amturing.acm.org/award_winners/iverson_9147499.cfm). *amturing.acm.org*. ACM. Retrieved January 10, 2015.
38. ACM SIGPLAN. "APL2: The Early Years" (http://www.sigapl.org/Articles/JimBrown-TechCompromise.php). *www.sigapl.org*. ACM. Retrieved January 10, 2015.
39. Falkoff, Adin D. (1991). "The IBM family of APL systems". *IBM Systems Journal*. **30** (4): 416–432. doi:10.1147/sj.304.0416 (https://doi.org/10.1147%2Fsj.304.0416). S2CID 19030940 (https://api.semanticscholar.org/CorpusID:19030940).
40. "IBM APL2" (http://www.edm2.com/index.php/IBM_APL2). *EDM2*. 2019-10-09. Retrieved 2021-11-17.
41. "APL2: What's New" (http://www-01.ibm.com/support/docview.wss?uid=swg22012321). *ibm.com*. ibm. Retrieved April 22, 2018.
42. Micro APL. "Overview of the APL System" (http://www.microapl.co.uk/apl/apl_concepts_chapter1.html). *www.microapl.co.uk*. Micro APL. Retrieved January 10, 2015.
43. Robertson, Graeme. "A Personal View of APL2010" (https://web.archive.org/web/20150402093211/http://archive.vector.org.uk/art10500450). *archive.vector.org.uk*. Vector – Journal of the British APL Association. Archived from the original (http://archive.vector.org.uk/art10500450) on April 2, 2015. Retrieved January 10, 2015.
44. Rodriguez, P.; Rojas, J.; Alfonseca, M.; Burgos, J. I. (1989). "An Expert System in Chemical Synthesis written in APL2/PC". *ACM SIGAPL APL Quote Quad*. **19** (4): 299–303. doi:10.1145/75144.75185 (https://doi.org/10.1145%2F75144.75185). S2CID 16876053 (https://api.semanticscholar.org/CorpusID:16876053).
45. IBM. "APL2: A Programming Language for Problem Solving, Visualization and Database Access" (http://www-03.ibm.com/software/products/en/apl2). *www-03.ibm.com*. IBM. Retrieved January 10, 2015.
46. Pike, Rob (2018-03-25). "Ivy" (https://web.archive.org/web/20190813210651/https://godoc.org/robpike.io/ivy). *GoDoc*. Archived from the original (https://godoc.org/robpike.io/ivy) on 2019-08-13.

47. "Wolfram Language FAQ" (https://www.wolfram.com/language/faq/). Wolfram. Retrieved February 20, 2020. "LISP and APL were two early influences"
48. Texas Instruments (1977). "TI 745 full page ad: Introducing a New Set of Characters" (https://books.google.com/books?id=wMe6erbb5V4C&q=apl%20terminal%20%22texas%20instruments%22&pg=PA32). *Computerworld*. **11** (27): 32. Retrieved January 20, 2015.
49. Dyalog. "APL Fonts and Keyboards" (http://www.dyalog.com/apl-font-keyboard.htm). *www.dyalog.com*. Dyalog. Retrieved January 19, 2015.
50. Smith, Bob. "NARS2000 Keyboard" (http://www.sudleyplace.com/APL/Keyboard.ahtml). *www.sudleyplace.com*. Bob Smith / NARS2000. Retrieved January 19, 2015.
51. MicroAPL Ltd. "Introduction to APL – APL Symbols" (http://www.microapl.co.uk/apl/introduction_chapter1.html). *www.microapl.co.uk*. MicroAPL Ltd. Retrieved January 8, 2015.
52. Brown, James A.; Hawks, Brent; Trimble, Ray (1993). "Extending the APL character set". *ACM SIGAPL APL Quote Quad*. **24** (1): 41–46. doi:10.1145/166198.166203 (https://doi.org/10.1145%2F166198.166203).
53. Kromberg, Morten. "Unicode Support for APL" (https://web.archive.org/web/20150120194338/http://archive.vector.org.uk/art10500090). *archive.vector.org.uk*. Vector, Journal of the British APL Association. Archived from the original (http://archive.vector.org.uk/art10500090) on January 20, 2015. Retrieved January 8, 2015.
54. Hsu, Aaron. "Computer Science Outreach and Education with APL" (http://video.dyalog.com/Dyalog13/?v=kIItfQJEVdM). Dyalog, Ltd. Retrieved July 15, 2016.
55. Dyalog, Inc. APL fonts and keyboards. http://www.dyalog.com/apl-font-keyboard.htm
56. MicroAPL. "Operators" (http://www.microapl.co.uk/apl/apl_concepts_chapter5.html). *www.microapl.co.uk*. MicroAPL. Retrieved January 12, 2015.
57. Primitive Functions. "Primitive Functions" (http://www.microapl.co.uk/apl_help/ch_020_010_140.htm). *www.microapl.co.uk/*. Retrieved January 1, 2015.
58. Workspace. "The Workspace" (http://www.microapl.co.uk/apl/apl_concepts_chapter2.html). *www.microapl.co.uk*. Retrieved January 1, 2015.
59. "example" (https://web.archive.org/web/20130708114840/http://catpad.net/michael/apl/). Catpad.net. Archived from the original (http://catpad.net/michael/apl) on July 8, 2013. Retrieved June 17, 2013.
60. APL Symbols. "Entering APL Symbols" (http://www.microapl.co.uk/apl/introduction_chapter2.html). *www.microapl.co.uk*. Retrieved January 1, 2015.
61. Dickey, Lee, A list of APL Transliteration Schemes (http://www.math.uwaterloo.ca/apl_archives/apl/translit.schemes) Archived (https://web.archive.org/web/20060929174125/http://www.math.uwaterloo.ca/apl_archives/apl/translit.schemes) 2006-09-29 at the Wayback Machine, 1993
62. Iverson K.E., "Notation as a Tool of Thought (http://www.jsoftware.com/papers/tot.htm) Archived (https://web.archive.org/web/20130920071911/http://www.jsoftware.com/papers/tot.htm) 2013-09-20 at the Wayback Machine", *Communications of the ACM*, 23: 444-465 (August 1980).
63. Batenburg. "APL Efficiency" (http://www.ekevanbatenburg.nl/PRVAPL.HTML). *www.ekevanbatenburg.nl*. Retrieved January 1, 2015.
64. Vaxman. "APL Great Programming" (http://www.vaxman.de/publications/apl_slides.pdf) (PDF). *www.vaxman.de*. Archived (https://ghostarchive.org/archive/20221009/http://www.vaxman.de/publications/apl_slides.pdf) (PDF) from the original on 2022-10-09. Retrieved January 1, 2015.
65. Janko, Wolfgang (May 1987). "Investigation into the efficiency of using APL for the programming of an inference machine". *ACM SIGAPL APL Quote Quad*. **17** (4): 450–456. doi:10.1145/384282.28372 (https://doi.org/10.1145%2F384282.28372).

66. Borealis. "Why APL?" (http://www.aplborealis.com/whyapl.html). *www.aplborealis.com*. Retrieved January 1, 2015.
67. Iverson, Kenneth E. "A Dictionary of APL" (http://www.jsoftware.com/papers/APLDictionary.htm). *www.jsoftware.com*. JSoftware; Iverson Estate. Retrieved January 20, 2015.
68. "APL concepts" (http://www.microapl.co.uk/APL/apl_concepts_chapter6.html). Microapl.co.uk. Retrieved February 3, 2010.
69. "Nested array theory" (https://web.archive.org/web/20110709072354/http://www.nial.com/ArrayTheory.html). Nial.com. Archived from the original (http://www.nial.com/ArrayTheory.html) on 2011-07-09. Retrieved February 3, 2010.
70. "Programmera i APL", Bohman, Fröberg, Studentlitteratur, ISBN 91-44-13162-3
71. Iverson, Kenneth E. "APL Syntax and Semantics" (http://www.jsoftware.com/papers/APLSyntaxSemantics.htm). *www.jsoftware.com*. I. P. Sharp Associates. Retrieved January 11, 2015.
72. MicroAPL. "APL Primitives" (http://www.microapl.co.uk/apl_help/ch_020_020.htm). *www.microapl.co.uk*. MicroAPL. Retrieved January 11, 2015.
73. NARS2000. "APL Font – Extra APL Glyphs" (http://wiki.nars2000.org/index.php/APL_Font). *wiki.nars2000.org*. NARS2000. Retrieved January 11, 2015.
74. Fox, Ralph L. "Systematically Random Numbers" (http://www.sigapl.org/article1.php). *www.sigapl.org*. SIGAPL. Retrieved January 11, 2015.
75. Scholes, John (January 26, 2009). *Conway's Game of Life in APL* (https://www.youtube.com/watch?v=a9xAKttWgP4) (video). YouTube. Retrieved November 20, 2021.
76. Further technical details in APL Wiki's article "Conway's Game of Life" (https://aplwiki.com/wiki/Conway%27s_Game_of_Life). Retrieved November 20, 2021.
77. McDonnell, E.E. The introduction to *A Source Book in APL* (https://code.jsoftware.com/wiki/Doc/A_Source_Book_in_APL#origins), APL Press, 1981. (full book scan (http://www.softwarepreservation.org/projects/apl/Papers/ASourceBookInAPL/view))
78. Acharya, R; Pereira, (904567457) N.E. APL Programming Language (https://courses.cs.vt.edu/~cs5314/Lang-Paper-Presentation/Papers/HoldPapers/APL.pdf#page=3) Archived (https://web.archive.org/web/20211103210435/https://courses.cs.vt.edu/~cs5314/Lang-Paper-Presentation/Papers/HoldPapers/APL.pdf#page=3) 2021-11-03 at the Wayback Machine. Paper for CS5314 (Concepts of Programming Languages) at Virginia Tech.
79. APL Wiki. APL Conference (https://aplwiki.com/wiki/APL_conference). Retrieved 13 Oct 2021.
80. Jacob Williams. Degenerate Conic: New Blood (http://degenerateconic.com/new-blood/). Retrieved 13 Oct 2021.
81. APL Wiki. APL logo (https://aplwiki.com/wiki/APL_logo). Retrieved 20 May 2022.
82. "2017 Annual Report" (https://www.simcorp.com/-/media/files/investor/annual-reports/simcorp-annual-report-2017.pdf) (PDF). SimCorp. February 1, 2018. Archived (https://ghostarchive.org/archive/20221009/https://www.simcorp.com/-/media/files/investor/annual-reports/simcorp-annual-report-2017.pdf) (PDF) from the original on 2022-10-09. Retrieved April 3, 2018. "Sofia is a front-to-back investment management platform like SimCorp Dimension. ... Sofia is based on the APL coding language just like some parts of SimCorp Dimension."
83. Lee, Georges; Lelouche, Ruddy; Meissonnier, Vincent; Zarri, Gian Piero (September 1, 1982). "Using APL in an Artificial Intelligence environment" (https://www.researchgate.net/publication/234789115). *ACM SIGAPL APL Quote Quad*. **13** (1): 183–191. doi:10.1145/390006.802242 (https://doi.org/10.1145%2F390006.802242). Retrieved April 3, 2018.
84. Fordyce, K.; Sullivan, G. (1985). "Artificial Intelligence Development Aids" (https://doi.org/10.1145%2F255315.255347). *APL Quote Quad*. APL 85 Conf. Proc. (15): 106–113. doi:10.1145/255315.255347 (https://doi.org/10.1145%2F255315.255347).

85. Alfonseca, Manuel (July 1990). "Neural networks in APL" (https://www.researchgate.net/publication/220731305). *ACM SIGAPL APL Quote Quad*. **20** (4): 2–6. doi:10.1145/97811.97816 (https://doi.org/10.1145%2F97811.97816). Retrieved April 3, 2018.
86. Kromberg, Morten. "Robot Programming in APL" (http://begriffs.com/posts/2014-11-26-robots-in-apl.html). *www.dyalog.com/*. Retrieved January 6, 2015.
87. Holmes, W N (May 1978). "Is APL a Programming Language?" (https://doi.org/10.1093%2Fcomjnl%2F21.2.128). *The Computer Journal*. **21** (2): 128–131. doi:10.1093/comjnl/21.2.128 (https://doi.org/10.1093%2Fcomjnl%2F21.2.128).
88. Hsu, Aaron (November 18, 2017). "Design Patterns vs. Anti-pattern in APL" (https://web.archive.org/web/20180323152845/https://confengine.com/functional-conf-2017/proposal/4620/design-patterns-vs-anti-pattern-in-apl). *functionalconf.com*. Archived from the original (https://confengine.com/functional-conf-2017/proposal/4620/design-patterns-vs-anti-pattern-in-apl) on March 23, 2018. Retrieved 2018-04-07.
89. Magnenat-Thalmann, Nadia; Thalmann, Daniel (1985). *Computer Animation Theory and Practice* (https://books.google.com/books?id=neGoCAAAQBAJ&pg=PA38). Springer-Verlag. p. 38. ISBN 9784431684336. Retrieved April 3, 2018. "Digital Effects is another production house that worked on Tron. They used a laser-scanning system to digitize, store and reproduce images. Judson Rosebush, president of Digital Effects, is the primary designer of APL VISION and FORTRAN VISION, two computer animation packages that are currently used."
90. Gutsell, Sam (October 17, 2017). "Stormwind Simulator at Dyalog '16" (https://www.optima-systems.co.uk/stormwind-simulator-dyalog-16). *www.optima-systems.co.uk*. Optima Systems. Retrieved April 3, 2018. "Stormwind is a [3D boating simulator] that has gained a huge amount of interest in the APL community."
91. "OP-Pohjola ja Tieto hoitivat sovelluksen muutostyöt sujuvalla yhteistyöllä" (https://www.tieto.com/sites/default/files/migrated/documents/Case_OP-Pohjola_fi2806.pdf) [Smooth cooperation between OP-Pohjola and Tieto enabled app modification] (PDF). *www.tieto.com* (in Finnish). Tieto. Archived (https://ghostarchive.org/archive/20221009/https://www.tieto.com/sites/default/files/migrated/documents/Case_OP-Pohjola_fi2806.pdf) (PDF) from the original on 2022-10-09. Retrieved April 3, 2018.
92. "Vi idag" (http://profdoccare.se/var-ide/om-oss/vi-idag/) [We today]. *profdoccare.se* (in Swedish). Retrieved April 3, 2018. "Through the choice of APL as a technical platform, it is relatively easy to quickly build a solution that can be called a executable prototype (translated from the original)"
93. Brenner, Charles. "DNA Identification Technology and APL" (http://dna-view.com/DNAtechID.htm). *dna-view.com*. Presentation at the 2005 APL User Conference. Retrieved January 9, 2015.
94. Brenner, Charles. "There's DNA Everywhere – an Opportunity for APL" (https://www.youtube.com/watch?v=oXlP3r6PzeE). *www.youtube.com*. YouTube. Archived (https://ghostarchive.org/varchive/youtube/20211114/oXlP3r6PzeE) from the original on 2021-11-14. Retrieved January 9, 2015.
95. "Awards – 1973 – Lawrence Breed" (https://web.archive.org/web/20120402212031/http://awards.acm.org/citation.cfm?id=0694605&srt=all&aw=145&ao=GMHOPPER&yr=1973). Association for Computing Machinery. Archived from the original (http://awards.acm.org/citation.cfm?id=0694605&srt=all&aw=145&ao=GMHOPPER&yr=1973) on April 2, 2012.
96. "Awards – 1973 – Richard Lathwell" (https://web.archive.org/web/20120402212035/http://awards.acm.org/citation.cfm?id=3412588&srt=all&aw=145&ao=GMHOPPER&yr=1973). Association for Computing Machinery. Archived from the original (http://awards.acm.org/citation.cfm?id=3412588&srt=all&aw=145&ao=GMHOPPER&yr=1973) on April 2, 2012.

97. "Awards – 1973 – Roger Moore" (https://web.archive.org/web/20120402212037/http://awards.acm.org/citation.cfm?id=4987585&srt=all&aw=145&ao=GMHOPPER&yr=1973). Association for Computing Machinery. Archived from the original (http://awards.acm.org/citation.cfm?id=4987585&srt=all&aw=145&ao=GMHOPPER&yr=1973) on April 2, 2012.
98. "IBM 5100" (https://web.archive.org/web/20180430050157/http://www.old-computers.com/museum/computer.asp?c=795). *old-computers.com*. Archived from the original (http://www.old-computers.com/museum/computer.asp?c=795) on April 30, 2018. Retrieved April 8, 2018.
99. "Welcome, IBM, to personal computing" (https://archive.org/stream/byte-magazine-1975-12/1975_12_BYTE_00-04_Assembling_an_Altair#page/n91/mode/2up). *Byte*. December 1975. p. 90. Retrieved April 29, 2018.
100. "Chronology of APL and its Influences on Computer Language Development" (http://www.sigapl.org/APLChronology.php). *www.sigapl.org*. ACM. Retrieved April 29, 2018.
101. Larry Breed (August 2006). "How We Got To APL\1130" (https://web.archive.org/web/20080512031437/http://www.vector.org.uk/archive/v223/APL_1130.htm). *Vector (British APL Association)*. **22** (3). ISSN 0955-1433 (https://www.worldcat.org/issn/0955-1433). Archived from the original (http://www.vector.org.uk/archive/v223/APL_1130.htm) on May 12, 2008. Retrieved April 29, 2018.
102. Roger Moore (2005). "History of I. P. Sharp Associates Timesharing and Network" (https://web.archive.org/web/20190404053142/http://www.rogerdmoore.ca/INF/ERInstallationHistory.htm). *Rogerdmoore.ca*. Roger Moore. Archived from the original (http://rogerdmoore.ca/INF/ERInstallationHistory.htm) on April 4, 2019. Retrieved March 7, 2018.
103. Blumenthal, Marcia (May 18, 1981). "VAX-11s Acquire APL Processor" (https://books.google.com/books?id=MCFtKT_NaYIC&q=vax%20apl&pg=PA2). *Computerworld*. Retrieved April 22, 2018.
104. "Kenneth E. Iverson Award for Outstanding Contribution to APL" (https://web.archive.org/web/20120226063703/http://www.sigapl.org/award.htm). SIGPLAN Chapter on Array Programming Languages (SIGAPL). Archived from the original (http://www.sigapl.org/award.htm) on February 26, 2012.
105. Mark Schora (2021-01-26). "Log-On Software announces Log-On APL2" (https://log-on.com/2021/01/26/log-on-software-announces-log-on-apl2/). *Log-On Software*. Retrieved 2021-11-17.
106. Kelley, R.A. (1973). "APLGOL, an Experimental Structured Programming Language" (https://ieeexplore.ieee.org/document/5391426/authors#authors). *IBM Journal of Research and Development*. **17**: 69–73. doi:10.1147/rd.171.0069 (https://doi.org/10.1147%2Frd.171.0069).
107. Johnston, Ronald L. (July 1977). "APLGOL: Structured Programming Facilities for APL" (https://archive.org/details/Hewlett-Packard_Journal_Vol._28_No._11_1977-07_Hewlett-Packard/page/n9/mode/2up). Hewlett-Packard Journal.
108. "Dyalog Ltd website" (https://www.dyalog.com/). Retrieved 6 June 2018.
109. "Dyalog at 25" (http://www.vector.org.uk/archive/v234b/d25.pdf) (PDF). *Vector Magazine*. British APL Association. September 2008. Retrieved April 14, 2018.
110. Kromberg, Morten (22 October 2007). "Arrays of objects" (https://www.dyalog.com/uploads/documents/Papers/Arrays%20of%20Objects.pdf) (PDF). *Proceedings of the 2007 symposium on Dynamic languages*. p. 20. doi:10.1145/1297081.1297087 (https://doi.org/10.1145%2F1297081.1297087). ISBN 9781595938688. S2CID 18484472 (https://api.semanticscholar.org/CorpusID:18484472). Archived (https://ghostarchive.org/archive/20221009/https://www.dyalog.com/uploads/documents/Papers/Arrays%20of%20Objects.pdf) (PDF) from the original on 2022-10-09. Retrieved 27 August 2018.
111. Scholes, John. "D: A functional subset of Dyalog APL" (http://archive.vector.org.uk/art10007770). British APL Association.

112. Scholes, John. "Translation of D-functions into tacit form" (https://dfns.dyalog.com/n_tacit.htm). Dyalog Ltd.
113. "Dyalog – Prices and Licences" (https://www.dyalog.com/prices-and-licences.htm#basiclic).
114. "2016 Iverson Award Recognises Dyalog's CEO and CXO" (http://www.dyalog.com/dyalogue-newsletters.htm?nl=27&a=158). Retrieved 6 June 2018.
115. "Nested Arrays Research System – NARS2000: An Experimental APL Interpreter" (http://www.nars2000.org/). *NARS2000*. Sudley Place Software. Retrieved July 10, 2015.
116. "APLX has been withdrawn from commercial sale but can be downloaded free of charge" (http://microapl.com/apl/). Microapl.com. Retrieved April 14, 2018.
117. "York APL" (https://www.softwarepreservation.org/projects/apl/Books/YorkAPL).
118. "GNU APL" (http://directory.fsf.org/wiki/GNU_APL). *directory.fsf.org*. Free Software Directory. Retrieved September 28, 2013.
119. Stallman, Richard M. "RMS Berättar" (https://web.archive.org/web/20181126072301/http://www.lysator.liu.se/history/garb/txt/87-2-rms.txt). Archived from the original (http://www.lysator.liu.se/history/garb/txt/87-2-rms.txt) on November 26, 2018. Retrieved April 22, 2018.
120. Budd, Timothy (1988). *An APL Compiler* (https://books.google.com/books?id=rTb2BwAAQBAJ&pg=PA1). Springer-Verlag. ISBN 978-0-387-96643-4.
121. SIGAPL. "What is APL?" (http://www.sigapl.org/about.php). *www.sigapl.org*. SIGAPL. Retrieved January 20, 2015.
122. Ju, Dz-Ching; Ching, Wai-Mee (1991). "Exploitation of APL data parallelism on a shared-memory MIMD machine". *Newsletter ACM SIGPLAN Notices*. **26** (7): 61–72. doi:10.1145/109625.109633 (https://doi.org/10.1145%2F109625.109633). S2CID 8584353 (https://api.semanticscholar.org/CorpusID:8584353).
123. Hsu, Aaron W.; Bowman, William J. "Revisiting APL in the Modern Era" (http://www.cs.princeton.edu/~dpw/obt/abstracts/obt12_submission_11.pdf) (PDF). *www.cs.princeton.edu*. Indiana University / Princeton. Archived (https://ghostarchive.org/archive/20221009/http://www.cs.princeton.edu/~dpw/obt/abstracts/obt12_submission_11.pdf) (PDF) from the original on 2022-10-09. Retrieved January 20, 2015.
124. Ching, W.-M.; Ju, D. (1991). "Execution of automatically parallelized APL programs on RP3" (http://domino.research.ibm.com/tchjr/journalindex.nsf/c469af92ea9eceac85256bd50048567c/f892e104dfc4d0fd85256bfa0067fb42!OpenDocument). *IBM Journal of Research & Development*. **35** (5/6): 767–777. doi:10.1147/rd.355.0767 (https://doi.org/10.1147%2Frd.355.0767). Retrieved January 20, 2015.
125. Blelloch, Guy E.; Sabot, Gary W. (1990). "Compiling Collection-Oriented Languages onto Massively Parallel Computers". *Journal of Parallel and Distributed Computing*. **8** (2): 119–134. CiteSeerX 10.1.1.51.5088 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5088). doi:10.1016/0743-7315(90)90087-6 (https://doi.org/10.1016%2F0743-7315%2890%2990087-6). "Collection oriented languages include APL, APL2"
126. Jendrsczok, Johannes; Hoffmann, Rolf; Ediger, Patrick; Keller, Jörg. "Implementing APL-like data parallel functions on a GCA machine" (https://web.archive.org/web/20150122211834/https://www.fernuni-hagen.de/imperia/md/content/fakultaetfuermathematikundinformatik/pv/97-08/papergca_09_1_.pdf) (PDF). *www.fernuni-hagen.de*. Fernuni-Hagen.De. pp. 1–6. Archived from the original (https://www.fernuni-hagen.de/imperia/md/content/fakultaetfuermathematikundinformatik/pv/97-08/papergca_09_1_.pdf) (PDF) on January 22, 2015. Retrieved January 22, 2015. "GCA – Global Cellular Automation. Inherently massively parallel. 'APL has been chosen because of the ability to express matrix and vector' structures."

127. Brenner of IBM T.J.Watson Research Center, Norman (1984). "VLSI circuit design using APL with fortran subroutines". *Proceedings of the international conference on APL – APL '84*. Vol. 14. ACM SIGAPL. pp. 77–79. doi:10.1145/800058.801079 (https://doi.org/10.1145%2F800058.801079). ISBN 978-0897911375. S2CID 30863491 (https://api.semanticscholar.org/CorpusID:30863491). "APL for interactiveness and ease of coding" {{cite book}}: |journal= ignored (help)

128. Gamble, D.J.; Hobson, R.F. (1989). "Towards a graphics/Procedural environment for constructing VLSI module generators". *Conference Proceeding IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*. pp. 606–611. doi:10.1109/PACRIM.1989.48437 (https://doi.org/10.1109%2FPACRIM.1989.48437). S2CID 7921438 (https://api.semanticscholar.org/CorpusID:7921438). "VLSI module generators are described. APL and C, as examples of interpreted and compiled languages, can be interfaced to an advanced graphics display"

129. Lee, Robert S. (1983). "Two Implementations of APL" (https://books.google.com/books?id=qURs4j9vKn4C&q=%22IBM%20APL%27s%20fast%20execution%22&pg=PA379). *PC Magazine*. **2** (5): 379. Retrieved January 20, 2015.

130. MARTHA and LLAMA. "The APL Computer Language" (https://web.archive.org/web/20150213004709/http://marthallama.org/apl/). *marthallama.org*. MarthaLlama. Archived from the original (http://marthallama.org/apl/) on February 13, 2015. Retrieved January 20, 2015.

131. Metzger, Robert; Wen, Zhaofang (2000). *Automatic Algorithm Recognition and Replacement: A New Approach to Program Optimization* (https://books.google.com/books?id=u38h_fV3UqgC&pg=PA12). The MIT press. ISBN 9780262133685. Retrieved May 6, 2018.

132. Snyder, Lawrence (1982). "Recognition and Selection of Idioms for Code Optimization". *Acta Informatica*. **17** (3). doi:10.1007/BF00264357 (https://doi.org/10.1007%2FBF00264357). S2CID 8369972 (https://api.semanticscholar.org/CorpusID:8369972).

133. Cheng, Feng Sheng (1981). "Idiom matching: an optimization technique for an APL compiler" (https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=7896&context=rtd). Iowa State University. Retrieved May 6, 2018. {{cite journal}}: Cite journal requires |journal= (help)

134. "Idiom Recognition" (http://help.dyalog.com/16.0/Content/Language/Defined%20Functions%20and%20Operators/Idiom%20Recognition/Idiom%20Recognition.htm). dyalog.com. Retrieved May 6, 2018.

135. Strawn, George O. (March 1977). "Does APL really need run-time parsing?". *Software: Practice and Experience*. **7** (2): 193–200. doi:10.1002/spe.4380070207 (https://doi.org/10.1002%2Fspe.4380070207). S2CID 1463012 (https://api.semanticscholar.org/CorpusID:1463012).

136. "Compiler User Guide" (http://docs.dyalog.com/16.0/Compiler%20User%20Guide.pdf) (PDF). *www.dyalog.com*. Dyalog Ltd. Archived (https://ghostarchive.org/archive/20221009/http://docs.dyalog.com/16.0/Compiler%20User%20Guide.pdf) (PDF) from the original on 2022-10-09. Retrieved May 7, 2018.

137. Driscoll, Graham C. Jr.; Orth, Donald L. (November 1986). "Compiling APL: The Yorktown APL Translator". *IBM Journal of Research and Development*. **30** (6): 583–593. doi:10.1147/rd.306.0583 (https://doi.org/10.1147%2Frd.306.0583). S2CID 2299699 (https://api.semanticscholar.org/CorpusID:2299699).

138. "Chronology of APL" (http://www.sigapl.org/APLChronology.php). *www.sigapl.org*. ACM. Retrieved May 7, 2018.

139. Wai-Mee, Ching (November 1986). "Program Analysis and Code Generation in an APL/370 Compiler". *IBM Journal of Research and Development*. **30** (6): 594–602. doi:10.1147/rd.306.0594 (https://doi.org/10.1147%2Frd.306.0594). S2CID 17306407 (https://api.semanticscholar.org/CorpusID:17306407).

140. "The APEX Project" (http://www.snakeisland.com/apexup.htm).
141. "APL Compiler (message from Jim Weigang to the comp.lang.apl Newsgroup)" (http://www.chilton.com/~jimw/aplcomp.html). Apr 5, 1994.

Further reading

- *An APL Machine* (http://www-public.slac.stanford.edu/sciDoc/docMeta.aspx?slacPubNumber=slac-r-114) (1970 Stanford doctoral dissertation by Philip Abrams)
- *A Personal History Of APL* (http://sigapl.org/Articles/MichaelMontalbanoPersonalViewOfAPL.php) (1982 article by Michael S. Montalbano)
- McIntyre, Donald B. (1991). "Language as an intellectual tool: From hieroglyphics to APL" (https://web.archive.org/web/20060504050437/http://www.research.ibm.com/journal/sj/304/ibmsj3004N.pdf) (PDF). *IBM Systems Journal*. **30** (4): 554–581. doi:10.1147/sj.304.0554 (https://doi.org/10.1147%2Fsj.304.0554). Archived from the original (http://www.research.ibm.com/journal/sj/304/ibmsj3004N.pdf) (PDF) on May 4, 2006.
- Iverson, Kenneth E. (1991). "A Personal view of APL" (https://web.archive.org/web/20080227012149/http://www.research.ibm.com/journal/sj/304/ibmsj3004O.pdf) (PDF). *IBM Systems Journal*. **30** (4): 582–593. doi:10.1147/sj.304.0582 (https://doi.org/10.1147%2Fsj.304.0582). Archived from the original (http://www.research.ibm.com/journal/sj/304/ibmsj3004O.pdf) (PDF) on February 27, 2008.
- *A Programming Language* (https://web.archive.org/web/20141027152546/http://www.softwarepreservation.org/projects/apl/Books/APROGRAMMING%20LANGUAGE/view) by Kenneth E. Iverson
- *APL in Exposition* (http://www.softwarepreservation.org/projects/apl/Papers/197201_APL%20In%20Exposition_320-3010.pdf/view) by Kenneth E. Iverson
- Brooks, Frederick P.; Kenneth Iverson (1965). *Automatic Data Processing, System/360 Edition*. ISBN 0-471-10605-4.
- Askoolum, Ajay (August 2006). *System Building with APL + Win*. Wiley. ISBN 978-0-470-03020-2.
- Falkoff, Adin D.; Iverson, Kenneth E.; Sussenguth, Edward H. (1964). "A Formal Description of System/360" (https://web.archive.org/web/20080227012111/http://www.research.ibm.com/journal/sj/032/falkoff.pdf) (PDF). *IBM Systems Journal*. **3** (2): 198–261. doi:10.1147/sj.32.0198 (https://doi.org/10.1147%2Fsj.32.0198). Archived from the original (http://www.research.ibm.com/journal/sj/032/falkoff.pdf) (PDF) on February 27, 2008.
- Wexelblat, Richard L, ed. (1981). "XIV". *History of Programming Languages: Proceedings of the History of Programming Languages Conference, Los Angeles, Calif., June 1-3, 1978*. ISBN 978-0127450407.
- Banon, Gerald Jean Francis (1989). *Bases da Computacao Grafica*. Rio de Janeiro: Campus. p. 141.
- LePage, Wilbur R. (1978). *Applied A.P.L. Programming*. Prentice Hall.
- Mougin, Philippe; Ducasse, Stephane (November 2003). "OOPAL: Integrating array programming in object-oriented programming" (https://web.archive.org/web/20061114145417/http://www.fscript.org/documentation/OOPAL.pdf) (PDF). *ACM SIGPLAN Notices*. **38** (11): 65–77. doi:10.1145/949343.949312 (https://doi.org/10.1145%2F949343.949312). Archived from the original (http://www.fscript.org/documentation/OOPAL.pdf) (PDF) on November 14, 2006.
- Dyalog Limited (September 2006). *An Introduction to Object Oriented Programming For APL Programmers* (https://web.archive.org/web/20071004214341/http://www.dyalog.dk/whatsnew/OO4APLERS.pdf) (PDF). Dyalog Limited. Archived from the original (http://www.dyalog.dk/whatsnew/OO4APLERS.pdf) (PDF) on October 4, 2007.

- Shustek, Len (October 10, 2012). "The APL Programming Language Source Code" (http://www.computerhistory.org/atchm/the-apl-programming-language-source-code/#A-Taste-of-APL). Computer History Museum (CHM). Archived (https://web.archive.org/web/20170906205616/http://www.computerhistory.org/atchm/the-apl-programming-language-source-code/) from the original on September 6, 2017. Retrieved September 6, 2017.
- Svoboda, Antonín; White, Donnamaie E. (2016) [2012, 1985, 1979-08-01]. *Advanced Logical Circuit Design Techniques* (http://www.donnamaie.com/Advanced_logic_ckt/Advanced_Logical_Circuit_Design_Techniques%20sm.pdf) (PDF) (retyped electronic reissue ed.). Garland STPM Press (original issue) / WhitePubs Enterprises, Inc. (reissue). ISBN 978-0-8240-7014-4. LCCN 78-31384 (https://lccn.loc.gov/78-31384). Archived (https://web.archive.org/web/20170414163013/http://www.donnamaie.com/Advanced_logic_ckt/Advanced_Logical_Circuit_Design_Techniques%20sm.pdf) (PDF) from the original on 2017-04-14. Retrieved 2017-04-15. [1] (http://www.donnamaie.com/) [2] (https://books.google.com/books?id=g3uzAAAAIAAJ)

Video

- The Origins of APL (https://www.youtube.com/watch?v=8kUQWuK1L4w) on YouTube – a 1974 talk show style interview with the original developers of APL.
- APL demonstration (https://www.youtube.com/watch?v=_DTpQ4Kk2wA) on YouTube – a 1975 live demonstration of APL by Professor Bob Spence, Imperial College London.
- Conway's Game Of Life in APL (https://www.youtube.com/watch?v=a9xAKttWgP4) on YouTube – a 2009 tutorial by John Scholes of Dyalog Ltd. which implements Conway's Game of Life in a single line of APL.
- 50 Years of APL (https://www.youtube.com/watch?v=ra_JyBCI4Xg) on YouTube – a 2009 introduction to APL by Graeme Robertson.

External links

Online resources

- TryAPL.org (http://tryapl.org/), an online APL primer
- APL (https://curlie.org/Computers/Programming/Languages/APL) at Curlie
- APL2C (http://www.apl2c.de/home/Links/links.html), a source of links to APL compilers

-